TMS320C64x+ DSP Megamodule

Reference Guide

Literature Number: SPRU871K
August 2010

Contents

Preface **15**

1 Overview **17**

1.1 Introduction 18

1.2 C64x+ Megamodule Overview 19

1.2.1 C64x+ CPU 19

1.2.2 Level 1 Program (L1P) Memory Controller 19

1.2.3 Level 1 Data (L1D) Memory Controller 19

1.2.4 Level 2 (L2) Memory Controller 19

1.2.5 Internal DMA (IDMA) 20

1.2.6 Bandwidth Management (BWM) 20

1.2.7 Interrupt Controller (INTC) 20

1.2.8 Memory Protection Architecture (MPA) 21

1.2.9 Power-Down Controller (PDC) 21

1.2.10 Extended Memory Controller (EMC) 21

2 Level 1 Program Memory and Cache **23**

2.1 Introduction 24

2.1.1 Purpose of the Level 1 Program (L1P) Memory and Cache 24

2.1.2 Features 24

2.2 Terms and Definitions 24

2.3 L1 Program Memory Architecture 24

2.3.1 L1P Memory 24

2.4 L1P Cache 25

2.4.1 L1P Cache Architecture 25

2.4.2 Replacement and Allocation Strategy 26

2.4.3 L1P Mode Change Operations 26

2.4.4 L1P Freeze Mode 27

2.5 Program Initiated Coherence Operations 29

2.5.1 Global Coherence Operation 29

2.5.2 Block Coherence Operation 29

2.6 L1P Cache Control Registers 30

2.6.1 Memory Mapped Cache Control Register Overview 30

2.6.2 CPU Cache Control Registers 30

2.6.3 L1P Cache Configuration Registers 31

2.6.4 Privilege and Cache Control Operations 34

2.7 L1P Performance 34

2.7.1 L1P Miss Penalty 34

2.7.2 L1P Miss Pipelining 34

2.8 Power-Down Support 36

2.8.1 Static Power-Down 36

2.8.2 Dynamic Power-Down 36

2.8.3 Feature-Oriented Power-Down 36

2.9 L1P Memory Protection 37

2.9.1 Protection Checks on L1P Accesses 37

2.9.2 Memory Protection Registers 38

3 Level 1 Data Memory and Cache **49**

3.1 Introduction 50
3.1.1 Purpose of the Level 1 Data (L1D) Memory and Cache 50
3.1.2 Features 50
3.1.3 Terms and Definitions 50
3.2 L1D Memory Architecture 50
3.2.1 L1D Memory 50
3.3 L1D Cache 51
3.3.1 L1D Cache Architecture 51
3.3.2 Replacement and Allocation Strategy 52
3.3.3 L1D Mode Change Operations 53
3.3.4 L1D Freeze Mode 54
3.3.5 Program-Initiated Cache Coherence Operations 56
3.3.6 Cache Coherence Protocol 58
3.4 L1D Cache Control Registers 59
3.4.1 Memory Mapped L1D Cache Control Register Overview 59
3.4.2 CPU L1D Cache Control Registers 59
3.4.3 L1D Cache Configuration Registers 60
3.4.4 L1D Cache Coherence Operation Registers 62
3.4.5 Privilege and Cache Control Operations 66
3.5 L1D Memory Performance 66
3.5.1 L1D Memory Banking 66
3.5.2 L1D Miss Penalty 68
3.5.3 L1D Write Buffer 68
3.5.4 L1D Miss Pipelining 69
3.6 L1D Power-Down Support 69
3.7 L1D Memory Protection 70
3.7.1 Protection Checks on L1D Accesses 70
3.7.2 L1D Memory Protection Registers 70
3.7.3 Protection Checks on Accesses to Memory Protection Registers 79

4 Level 2 Memory and Cache 81

4.1 Introduction 82
4.1.1 Purpose of the Level 2 (L2) Memory and Cache 82
4.1.2 Features 82
4.1.3 Terms and Definitions 82
4.2 Level 2 Memory Architecture 82
4.2.1 L2 Memory 82
4.3 L2 Cache 84
4.3.1 L2 Cache Architecture 84
4.3.2 Replacement and Allocation Strategy 85
4.3.3 Reset Behavior 85
4.3.4 L2 Mode Change Operations 86
4.3.5 L2 Freeze Mode 87
4.3.6 Program Initiated Cache Coherence Operations 88
4.3.7 Cacheability Controls 90
4.3.8 L1-L2 Coherence Support 91
4.4 L2 Cache Control Registers 93
4.4.1 Memory Mapped L2 Cache Control Registers Overview 93
4.4.2 L2 Configuration Register (L2CFG) 94
4.4.3 L2 Cache Coherence Operation Registers 95
4.4.4 Memory Attribute Registers (MARn) 100
4.4.5 Memory Attribute Registers (MARn) 106
4.4.6 Privilege and Cache Control Registers 106
4.5 L2 Power-Down 107

4.5.1 L2 Memory Dynamic Power-Down 107
4.5.2 L2 Memory Static Power-Down 108
4.5.3 L2 Power-Down Control Registers 109
4.6 L2 Memory Protection 112
4.6.1 Protection Checks on CPU, IDMA and Other System Master Accesses 112
4.6.2 L2 Memory Protection Registers 113
4.6.3 Protection Checks on Accesses to Memory Protection Registers 124
5 Internal Direct Memory Access (IDMA) Controller 125
5.1 Introduction 126
5.1.1 Purpose of the Internal Direct Memory Access (IDMA) Controller 126
5.1.2 Features 126
5.2 Terms and Definitions 126
5.3 IDMA Architecture 127
5.3.1 IDMA Channel 0 127
5.3.2 IDMA Channel 1 129
5.4 Registers 131
5.4.1 IDMA Channel 0 Status Register (IDMA0_STAT) 132
5.4.2 IDMA Channel 0 Mask Register (IDMA0_MASK) 133
5.4.3 IDMA Channel 0 Source Address Register (IDMA0_SOURCE) 134
5.4.4 IDMA Channel 0 Destination Address Register (IDMA0_DEST) 135
5.4.5 IDMA Channel 0 Count Register (IDMA0_COUNT) 136
5.4.6 IDMA Channel 1 Status Register (IDMA1_STAT) 137
5.4.7 IDMA Channel 1 Source Address Register (IDMA1_SOURCE) 138
5.4.8 IDMA Channel 1 Destination Address Register (IDMA1_DEST) 139
5.4.9 IDMA Channel 1 Count Register (IDMA1_COUNT) 140
5.5 Privilege Levels and IDMA Operation 141
6 Bandwidth Management Architecture 143
6.1 Introduction 144
6.1.1 Purpose of the Bandwidth Management 144
6.1.2 Resource Bandwidth Protected by Bandwidth Management 144
6.1.3 Requestors Managed by Bandwidth Management 144
6.1.4 Terms and Definitions 144
6.2 Architecture 145
6.2.1 Bandwidth Arbitration via Priority Levels 145
6.2.2 Priority Level: -1 145
6.2.3 Priority Declaration 145
6.3 Registers 146
6.3.1 CPU Arbitration Control Register (CPUARBD, CPUARBU, CPUARBE) 147
6.3.2 User Coherence Arbitration Control Register (UCARBD, UCARBU) 149
6.3.3 IDMA Arbitration Control Register (IDMAARBD, IDMAARBU, IDMAARBE) 150
6.3.4 Slave DMA Arbitration Control Register (SDMAARBD, SDMAARBU, SDMAARBE) 151
6.3.5 Master DMA Arbitration Control Register (MDMAARBE) 152
6.4 Privilege and Bandwidth Management Registers 153
7 Interrupt Controller 155
7.1 Introduction 156
7.1.1 Purpose of the C64x+ Megamodule Interrupt Controller (INTC) 156
7.1.2 Features 156
7.1.3 Functional Block Diagram 157
7.1.4 Terms and Definitions 157
7.2 Interrupt Controller Architecture 158
7.2.1 Event Registers 158
7.2.2 Event Combiner 160
7.2.3 Interrupt Selector 162

7.2.4 Exception Combiner 164
7.3 C64x+ Megamodule Events 166
7.4 Interrupt Controller - CPU Interaction 167
7.4.1 CPU – Interrupt Controller Interface 167
7.4.2 CPU Servicing of Interrupt Events 168
7.5 Registers 169
7.5.1 Event Registers 170
7.5.2 Event Combiner Registers 174
7.5.3 CPU Interrupt Selector Registers 178
7.5.4 CPU Exception Registers 181
7.5.5 Advanced Event Generator Mux Registers (AEGMUX*n*) 184
7.5.6 Privilege and Interrupt Controller Registers 186
8 Memory Protection 187
8.1 Introduction 188
8.1.1 Purpose of the Memory Protection 188
8.1.2 Privilege Levels 188
8.2 Terms and Definitions 188
8.3 Memory Protection Architecture 188
8.3.1 Memory Protection Pages 188
8.3.2 Permission Structure 189
8.3.3 Invalid Accesses and Exceptions 190
8.4 Memory Protection Architecture Registers 191
8.4.1 Memory Protection Page Attribute (MPPA) Registers 192
8.4.2 Memory Protection Fault Registers (MPFAR, MPFSR, MPFCR) 192
8.4.3 Memory Protection Lock Registers (MPLK*n*) 196
8.4.4 Keys Shorter than 128 Bits 199
8.5 Permission Checks on Accesses to Memory Protection Registers 199
9 Power-Down Controller 201
9.1 Introduction 202
9.1.1 C64x+ Megamodule Power Down-Management 202
9.1.2 Power-Down Capabilities Overview 202
9.2 Power-Down Features 202
9.2.1 L1P Memory 202
9.2.2 L2 Memory 203
9.2.3 Cache Power-Down Modes 203
9.2.4 CPU Power-Down 203
9.2.5 C64x+ Megamodule Power-Down 203
9.2.6 Miscellaneous Power-Down 204
9.3 Power-Down Controller Command Register (PDCCMD) 204
10 Miscellaneous 207
10.1 Introduction 208
10.2 Megamodule Revision ID Register (MM_REVID) 208
10.3 Bus Error Register (BUSERR) 209
10.4 Bus Error Details Register (BUSERRCLR) 210
A General Terms and Definitions 211
B Cache Terms and Definitions 213
C Revision History 217

List of Figures

1-1. TMS320C64x+ Megamodule Block Diagram 18
2-1. Data Access Address Organization 25
2-2. L1P Configuration Register (L1PCFG) 31
2-3. L1P Cache Control Register (L1PCC) 32
2-4. L1P Invalidate Base Address Register (L1PIBAR) 32
2-5. L1P Invalidate Word Count Register (L1PIWC) 33
2-6. L1P Invalidate Register (L1PINV) 33
2-7. Memory Page Protection Attribute Registers (L1PMPPA*x*) 40
2-8. Memory Protection Lock Register 0 (L1PMPLK0) 42
2-9. Memory Protection Lock Register 1 (L1PMPLK1) 42
2-10. Memory Protection Lock Register 2 (L1PMPLK2) 42
2-11. Memory Protection Lock Register 3 (L1PMPLK3) 43
2-12. Memory Protection Lock Command Register (L1PMPLKCMD) 43
2-13. Memory Protection Lock Status Register (L1PMPLKSTAT) 44
2-14. L1P Memory Protection Fault Address Register (L1PMPFAR) 45
2-15. L1P Memory Protection Fault Set Register (L1PMPFSR) 46
2-16. L1P Memory Protection Fault Clear Register (L1PMPFCLR) 47
3-1. Data Access Address Organization 51
3-2. L1D Cache Configuration Register (L1DCFG) 60
3-3. L1D Cache Control Register (L1DCC) 61
3-4. L1D Invalidate Register (L1DINV) 62
3-5. L1P Writeback Register (L1DWB) 63
3-6. L1D Writeback-Invalidate Register (L1DWBINV) 63
3-7. L1D Invalidate Base Address Register (L1DIBAR) 64
3-8. L1D Invalidate Word Count Register (L1DIWC) 64
3-9. L1D Writeback Base Address Register (L1DWBAR) 65
3-10. L1D Writeback-Invalidate Word Count Register (L1DWIWC) 65
3-11. Address to Bank Number Mapping 66
3-12. Potentially Conflicting Memory Accesses 67
3-13. Memory Protection Register (MPPA*xx*) 72
3-14. Level 1 Data Memory Protection Lock Register 0 (L1DMPLK0) 74
3-15. Level 1 Data Memory Protection Lock Register 1 (L1DMPLK1) 74
3-16. Level 1 Data Memory Protection Lock Register 2 (L1DMPLK2) 74
3-17. Level 1 Data Memory Protection Lock Register 3 (L1DMPLK3) 74
3-18. Level 1 Data Memory Protection Lock Command Register (L1DMPLKCMD) 75
3-19. Level 1 Data Memory Protection Lock Status Register (L1DMPLKSTAT) 76
3-20. Memory Protection Fault Address Register (L1DMPFAR) 76
3-21. Memory Protection Fault Set Register (L1DMPFSR) 77
3-22. Memory Protection Fault Clear Register (L1DMPFCR) 78
4-1. L2 Cache Address Organization 84
4-2. L2 Configuration Register (L2CFG) 94
4-3. L2 Writeback Base Address Register (L2WBAR) 95
4-4. L2 Writeback Word Count Register (L2WWC) 95
4-5. L2 Writeback-Invalidate Base Address Register (L2WIBAR) 96
4-6. L2 Writeback-Invalidate Word Count Register (L2WIWC) 96
4-7. L2 Invalidate Base Address Register (L2IBAR) 97
4-8. L2 Invalidate Word Count Register (L2IWC) 97

4-9. L2 Writeback Register (L2WB) 98
4-10. L2 Writeback-Invalidate Register (L2WBINV) 98
4-11. L2 Invalidate Register (L2INV) 99
4-12. Memory Attribute Register (MAR*n*) 106
4-13. Level 2 Power-Down Wake Register (L2PDWAKE*n*) 109
4-14. Level 2 Power-Down Sleep Register (L2PDSLEEP*n*) 110
4-15. Level 2 Power-Down Status Register (L2PDSTAT*n*) 111
4-16. L2 Memory Protection Page Attribute Registers (L2MPPA*n*) 116
4-17. Level 2 Memory Protection Lock 0 Register (L2MPLK0) 117
4-18. Level 2 Memory Protection Lock 1 Register (L2MPLK1) 118
4-19. Level 2 Memory Protection Lock 2 Register (L2MPLK2) 118
4-20. Level 2 Memory Protection Lock 3 Register (L2MPLK3) 118
4-21. Level 2 Memory Protection Lock Command Register (L2MPLKCMD) 119
4-22. Level 2 Memory Protection Lock Status Register (L2MPLKSTAT) 120
4-23. Level 2 Memory Protection Fault Address Register (L2MPFAR) 121
4-24. Level 2 Memory Protection Fault Set Register (L2MPFSR) 122
4-25. Level 2 Memory Protection Fault Clear Register (L2MPFCLR) 123
5-1. IDMA Channel 0 Transaction 128
5-2. IDMA Channel 1 Transaction 129
5-3. Example of IDMA Channel 1 130
5-4. IDMA Channel 0 Status Register (IDMA0_STAT) 132
5-5. IDMA Channel 0 Mask Register (IDMA0_MASK) 133
5-6. IDMA Channel 0 Source Address Register (IDMA0_SOURCE) 134
5-7. IDMA Channel 0 Destination Address Register (IDMA0_DEST) 135
5-8. IDMA Channel 0 Count Register (IDMA0_COUNT) 136
5-9. IDMA Channel 1 Status Register (IDMA1_STAT) 137
5-10. IDMA Channel 1 Source Address Register (IDMA1_SOURCE) 138
5-11. IDMA Channel 1 Destination Address Register (IDMA1_DEST) 139
5-12. IDMA Channel 1 Count Register (IDMA1_COUNT) 140
6-1. CPU Arbitration Control Register (CPUARBD, CPUARBU, CPUARBE) 148
6-2. User Coherence Arbitration Control Register (UCARBD, UCARBU) 149
6-3. IDMA Arbitration Control Register (IDMAARBD, IDMAARBU, IDMAARBE) 150
6-4. Slave DMA Arbitration Control Register (SDMAARBD, SDMAARBU, SDMAARBE) 151
6-5. Master DMA Arbitration Control Register (MDMAARBE) 152
7-1. C64x+ Megamodule Interrupt Controller Block Diagram 157
7-2. Event Flag Register Structure 158
7-3. Event Clear Register Structure 159
7-4. Event Set Register Structure 159
7-5. Event Combiner 160
7-6. Event Mask Register Structure 160
7-7. 32-Masked Event Flag Register Structure 161
7-8. Interrupt Selector Block Diagram 162
7-9. CPU Interrupt Routing Diagram 162
7-10. Interrupt Exception Event Block Diagram 163
7-11. System Exception Routing Diagram 164
7-12. Exception Mask Register Structure 164
7-13. Masked Exception Flag Register Structure 165
7-14. CPU Event Routing Diagram 167
7-15. Event Flag Register 0 (EVTFLAG0) 170

7-16. Event Flag Register 1 (EVTFLAG1) 170
7-17. Event Flag Register 2 (EVTFLAG2) 170
7-18. Event Flag Register 3 (EVTFLAG3) 171
7-19. Event Set Register 0 (EVTSET0) 172
7-20. Event Set Register 1 (EVTSET1) 172
7-21. Event Set Register 2 (EVTSET2) 172
7-22. Event Set Register 3 (EVTSET3) 172
7-23. Event Clear Register 0 (EVTCLR0) 173
7-24. Event Clear Register 1 (EVTCLR1) 173
7-25. Event Clear Register 2 (EVTCLR2) 173
7-26. Event Clear Register 3 (EVTCLR3) 173
7-27. Event Mask Register 0 (EVTMASK0) 174
7-28. Event Mask Register 1 (EVTMASK1) 174
7-29. Event Mask Register 2 (EVTMASK2) 174
7-30. Event Mask Register 3 (EVTMASK3) 175
7-31. Masked Event Flag Register 0 (MEVTFLAG0) 176
7-32. Masked Event Flag Register 1 (MEVTFLAG1) 176
7-33. Masked Event Flag Register 2 (MEVTFLAG2) 176
7-34. Masked Event Flag Register 3 (MEVTFLAG3) 176
7-35. Interrupt Mux Register 1 (INTMUX1) 178
7-36. Interrupt Mux Register 2 (INTMUX2) 178
7-37. Interrupt Mux Register 3 (INTMUX3) 178
7-38. Interrupt Exception Status Register (INTXSTAT) 179
7-39. Interrupt Exception Clear Register (INTXCLR) 180
7-40. Dropped Interrupt Mask Register (INTDMASK) 180
7-41. Exception Combiner Mask Register 0 (EXPMASK0) 181
7-42. Exception Mask Register 1 (EXPMASK1) 181
7-43. Exception Mask Register 2 (EXPMASK2) 181
7-44. Exception Mask Register 3 (EXPMASK3) 181
7-45. Masked Exception Flag Register 0 (MEXPFLAG0) 183
7-46. Masked Exception Flag Register 1 (MEXPFLAG1) 183
7-47. Masked Exception Flag Register 2 (MEXPFLAG2) 184
7-48. Masked Exception Flag Register 3 (MEXPFLAG3) 184
7-49. Advanced Event Generator Mux Register 0 (AEGMUX0) 185
7-50. Advanced Event Generator Mux Register 1 (AEGMUX1) 185
8-1. Permission Fields 189
8-2. Allowed IDs Bit Fields 189
8-3. Memory Protection Fault Address Register (MPFAR) 192
8-4. Memory Protection Fault Status Register (MPFSR) 193
8-5. Memory Protection Fault Command Register (MPFCR) 194
8-6. Memory Protection Lock Register (MPLK0) 196
8-7. Memory Protection Lock Register (MPLK1) 196
8-8. Memory Protection Lock Register (MPLK2) 196
8-9. Memory Protection Lock Register (MPLK3) 196
8-10. Memory Protection Lock Command Register (MPLKCMD) 197
8-11. Memory Protection Lock Status Register (MPLKSTAT) 198
9-1. Power-Down Controller Command Register (PDCCMD) 204
10-1. Megamodule Revision ID Register (MM_REVID) 208
10-2. Bus Error Register (BUSERR) 209

10-3. Bus Error Details Register (BUSERRCLR) 210

List of Tables

2-1. L1P Cache Registers Summary 25
2-2. Cache Size Specified by the L1PMODE bit in the L1PCFG Register 26
2-3. Switching L1P Modes 27
2-4. L1P Global Coherence Operations 29
2-5. L1P Block Cache Operations 29
2-6. L1P Specific Cache Control Operations Registers 30
2-7. L1P Configuration Register (L1PCFG) Field Descriptions 31
2-8. L1P Cache Control Register (L1PCC) Field Descriptions 32
2-9. L1P Invalidate Base Address Register (L1PIBAR) Field Descriptions 32
2-10. L1P Invalidate Word Count Register (L1PIWC) Field Descriptions 33
2-11. L1P Invalidate Register (L1PINV) Field Descriptions 33
2-12. Permissions for L1P Cache Control Registers 34
2-13. L1P Miss Pipelining Performance (Average Number of Stalls per Execute Packet) 35
2-14. Permission Bits Examined With Each Fetch 37
2-15. Memory Protection Registers 38
2-16. Memory Page Protection Attribute Registers 39
2-17. Memory Page Protection Attribute Register (L1PMPPA*x*) Field Descriptions 40
2-18. Memory Protection Lock Registers 42
2-19. Memory Protection Lock Command Register (L1PMPLKCMD) Field Descriptions 43
2-20. Memory Protection Lock Status Register (L1PMPLKSTAT) Field Descriptions 44
2-21. Memory Protection Fault Registers 45
2-22. L1P Memory Protection Fault Address Register (L1PMPFAR) Field Descriptions 45
2-23. L1P Memory Protection Fault Set Register (L1PMPFSR) Field Descriptions 46
2-24. L1P Memory Protection Fault Clear Register (L1PMPFCLR) Field Descriptions 47
2-25. Permissions for L1P Memory Protection Registers 48
3-1. Data Access Address Set Field Width 52
3-2. Cache Size Specified by the L1DMODE in the L1DCFG 53
3-3. Switching L1D Modes 53
3-4. Global Coherence Operations 56
3-5. Block Cache Operations 57
3-6. L1D Specific Cache Control Operations 59
3-7. L1D Cache Configuration Register (L1DCFG) Field Descriptions 60
3-8. L1D Cache Control Register (L1DCC) Field Descriptions 61
3-9. L1D Invalidate Register (L1DINV) Field Descriptions 62
3-10. L1D Writeback Register (L1DWB) Field Descriptions 63
3-11. L1D Writeback-Invalidate Register (L1DWBINV) Field Descriptions 63
3-12. L1D Invalidate Base Address Register (L1DIBAR) Field Descriptions 64
3-13. L1D Invalidate Word Count Register (L1DIWC) Field Descriptions 64
3-14. L1D Writeback Base Address Register (L1DWBAR) Field Descriptions 65
3-15. L1D Writeback-Invalidate Word Count Register (L1DWIWC) Field Descriptions 65
3-16. Permissions for L1D Cache Control Registers 66
3-17. L1D Performance Summary 69
3-18. Memory Protection Lock Registers 70
3-19. L1D Memory Protection Attribute Register Addresses 71
3-20. Memory Protection Register (MPPA*xx)* Field Descriptions 72
3-21. Memory Protection Defaults 73
3-22. Level 1 Data Memory Protection Command Register (L1DMPLKCMD) Field Descriptions 75

3-23. Level 1 Data Memory Protection Status Register (L1DMPLKSTAT) Field Descriptions 76
3-24. Memory Protection Fault Address Register (L1DMPFAR) Field Descriptions 76
3-25. Memory Protection Fault Set Register (L1DMPFSR) Field Descriptions 77
3-26. Memory Protection Fault Clear Register (L1DMPFCR) Field Descriptions 78
3-27. Permissions for L1D Memory Protection Registers 79
4-1. C64x+ Megamodule 2 × 128-bit Banking Scheme 83
4-2. Cache Registers Summary 84
4-3. L2MODE Description 85
4-4. Cache Size Specified by L2CFG.L2MODE 86
4-5. Switching L2 Modes 86
4-6. Freeze Mode Summary 87
4-7. Global Coherence Operations 88
4-8. Block Cache Operations 89
4-9. L2 to L1D Coherence Commands 92
4-10. Cache Control Registers 93
4-11. L2 Configuration Register (L2CFG) Field Descriptions 94
4-12. L2 Writeback Base Address Register (L2WBAR) Field Descriptions 95
4-13. L2 Writeback Word Count Register (L2WWC) Field Descriptions 95
4-14. L2 Writeback-Invalidate Base Address Register (L2WIBAR) Field Descriptions 96
4-15. L2 Writeback-Invalidate Word Count Register (L2WIWC) Field Descriptions 96
4-16. L2 Invalidate Base Address Register (L2IBAR) Field Descriptions 97
4-17. Invalidate Word Count Register (L2IWC) Field Descriptions 97
4-18. L2 Writeback Register (L2WB) Field Descriptions 98
4-19. L2 Writeback-Invalidate Register (L2WBINV) Field Descriptions 98
4-20. L2 Invalidate Register (L2INV) Field Descriptions 99
4-21. Memory Attribute Registers 100
4-22. Memory Attribute Register (MAR*n*) Field Descriptions 106
4-23. Permissions for L2 Cache Control Registers 106
4-24. L2 Memory Power-Down Register Summary 107
4-25. L2 Memory Power-Down Register Summary 109
4-26. Level 2 Power-Down Wake Register (L2PDWAKE*n*) Field Descriptions 109
4-27. Level 2 Power-Down Sleep Register (L2PDSLEEP*n*) Field Descriptions 110
4-28. Level 2 Power-Down Status Register (L2PDSTAT*n*) Field Descriptions 111
4-29. Permissions for L2 Power-Down Control Registers 111
4-30. L2 Memory Protection Registers 113
4-31. Level 2 Memory Protection Page Attribute Registers 114
4-32. Memory Protection Page Attribute Registers (MPPA*n*) Field Descriptions 116
4-33. Default Page Attribute Fields 117
4-34. Memory Protection Lock Registers 117
4-35. Level 2 Memory Protection Lock Command Register (L2MPLKCMD) Field Descriptions 119
4-36. Level 2 Memory Protection Lock Status Register (L2MPLKSTAT) Field Descriptions 120
4-37. Memory Protection Fault Registers 121
4-38. Level 2 Memory Protection Fault Address Register (L2MPFAR) Field Descriptions 121
4-39. Level 2 Memory Protection Fault Set Register (L2MPFSR) Field Descriptions 122
4-40. Level 2 Memory Protection Fault Clear Register (L2MPFCLR) Field Descriptions 123
4-41. Permissions for L2 Memory Protection Registers 124
5-1. IDMA Register Description 127
5-2. Internal Direct Memory Access (IDMA) Registers 131
5-3. IDMA Channel 0 Status Register (IDMA0_STAT) Field Descriptions 132

5-4. IDMA Channel 0 Mask Register (IDMA0_MASK) Field Descriptions 133
5-5. IDMA Channel 0 Source Address Register (IDMA0_SOURCE) Field Descriptions 134
5-6. IDMA Channel 0 Destination Address Register (IDMA0_DEST) Field Descriptions 135
5-7. IDMA Channel 0 Count Register (IDMA0_COUNT) Field Descriptions 136
5-8. IDMA Channel 1 Status Register (IDMA1_STAT) Field Descriptions 137
5-9. IDMA Channel 1 Source Address Register (IDMA1_SOURCE) Field Descriptions 138
5-10. IDMA Channel 1 Destination Address Register (IDMA1_DEST) Field Descriptions 139
5-11. IDMA Channel 1 Count Register (IDMA1_COUNT) Field Descriptions 140
5-12. Permissions for IDMA Registers 141
6-1. Priority Declaration Methods 145
6-2. Arbitration Registers 146
6-3. Arbitration Register Default Values 146
6-4. CPU Arbitration Control Register (CPUARBD, CPUARBU, CPUARBE) Field Descriptions 148
6-5. User Coherence Arbitration Control Register (UCARBD, UCARBU) Field Descriptions 149
6-6. IDMA Arbitration Control Register (IDMAARBD, IDMAARBU, IDMAARBE) Field Descriptions 150
6-7. Slave DMA Arbitration Control Register ((SDMAARBD, SDMAARBU, SDMARBE) Field Descriptions 151
6-8. Master DMA Arbitration Control Register (MDMAARBE) Field Descriptions 152
6-9. Permissions for Bandwidth Management Registers 153
7-1. Interrupt Controller Registers 158
7-2. System Event Mapping 166
7-3. Interrupt Controller Registers 169
7-4. Event Flag Registers (EVTFLAG*n*) Field Descriptions 171
7-5. Event Set Registers (EVTSET*n*) Field Descriptions 172
7-6. Event Clear Registers (EVTCLR*n*) Field Descriptions 173
7-7. Event Mask Registers (EVTMASK*n*) Field Descriptions 175
7-8. Masked Event Flag Registers (MEVTFLAG*n*) Field Descriptions 177
7-9. Interrupt Mux Registers (INTMUX*n*) Field Descriptions 178
7-10. Interrupt Exception Status Register (INTXSTAT) Field Descriptions 179
7-11. Interrupt Exception Clear Register (INTXCLR) Field Descriptions 180
7-12. Dropped Interrupt Mask Register (INTDMASK) Field Descriptions 180
7-13. Exception Mask Registers (EXPMASK*n*) Field Descriptions 182
7-14. Masked Exception Flag Registers (MEXPFLAG*n*) Field Descriptions 184
7-15. Advanced Event Generator Mux Registers (AEGMUX*n*) Field Descriptions 185
7-16. Permissions for Interrupt Controller Registers 186
8-1. Allowed IDs Bit Field Descriptions 189
8-2. Request Type Access Controls 190
8-3. Memory Protection Architecture Registers 191
8-4. Memory Protection Fault Address Register (MPFAR) Field Descriptions 192
8-5. Memory Protection Fault Status Register (MPFSR) Field Descriptions 193
8-6. Memory Protection Fault Command Register (MPFCR) Field Descriptions 194
8-7. Interpretation of MPFSR Access Type Field 194
8-8. Memory Protection Lock Command Register (MPLKCMD) Field Descriptions 197
8-9. Memory Protection Lock Status Register (MPLKSTAT) Field Descriptions 198
8-10. Allowed Accesses to Memory Protection Registers 199
9-1. C64x+ Megamodule Power-Down Features 202
9-2. Power-Down Controller Command Register (PDCCMD) Field Descriptions 204
9-3. Permissions for PDC Command Register 205
10-1. Miscellaneous Registers 208
10-2. Megamodule Revision ID Register (MM_REVID) Field Descriptions 208

10-3. Bus Error Register (BUSERR) Field Descriptions 209
10-4. Bus Error Details Register (BUSERRCLR) Field Descriptions 210
A-1. List of General Terms and Definitions 211
B-1. List of Cache-Related Terms and Definitions 213
C-1. Document Revision History 217

Read This First

About This Manual

This document describes the TMS320C64x+™ megamodule peripherals.

Notational Conventions

This document uses the following conventions.

- Hexadecimal numbers are shown with the suffix h. For example, the following number is 40 hexadecimal (decimal 64): 40h.
- Registers in this document are shown in figures and described in tables.
  - Each register figure shows a rectangle divided into fields that represent the fields of the register. Each field is labeled with its bit name, its beginning and ending bit numbers above, and its read/write properties below. A legend explains the notation used for the properties.
  - Reserved bits in a register figure designate a bit that is used for future device expansion.

Related Documentation From Texas Instruments

The following documents describe the C6000 devices and related support tools. Copies of these documents are available on the Internet at www.ti.com. *Tip:* Enter the literature number in the search box provided at www.ti.com.

The current documentation that describes the C6000 devices, related peripherals, and other technical collateral, is available in the C6000 DSP product folder at: www.ti.com/c6000.

SPRU732 — *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide.* Describes the CPU architecture, pipeline, instruction set, and interrupts for the TMS320C64x and TMS320C64x+ digital signal processors (DSPs) of the TMS320C6000 DSP family. The C64x/C64x+ DSP generation comprises fixed-point devices in the C6000 DSP platform. The C64x+ DSP is an enhancement of the C64x DSP with added functionality and an expanded instruction set.

SPRAA84 — *TMS320C64x to TMS320C64x+ CPU Migration Guide.* Describes migrating from the Texas Instruments TMS320C64x digital signal processor (DSP) to the TMS320C64x+ DSP. The objective of this document is to indicate differences between the two cores. Functionality in the devices that is identical is not included.

Overview

The TMS320C64x+™ DSP is the new generation of the TMS320C64x™ DSP architecture. It presents some new features that did not exist in the C64x™ DSP architecture as well as some existing features that have been enhanced.

The C64x+™ megamodule is the name used to designate the CPU together with the hardware providing memory, bandwidth management, interrupt, memory protection, and power-down support. This chapter provides an overview of the main components and features of the C64x+ megamodule. The C64x+ CPU is not described in this document since it is fully covered in the *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide* (SPRU732).

Topic **Page**

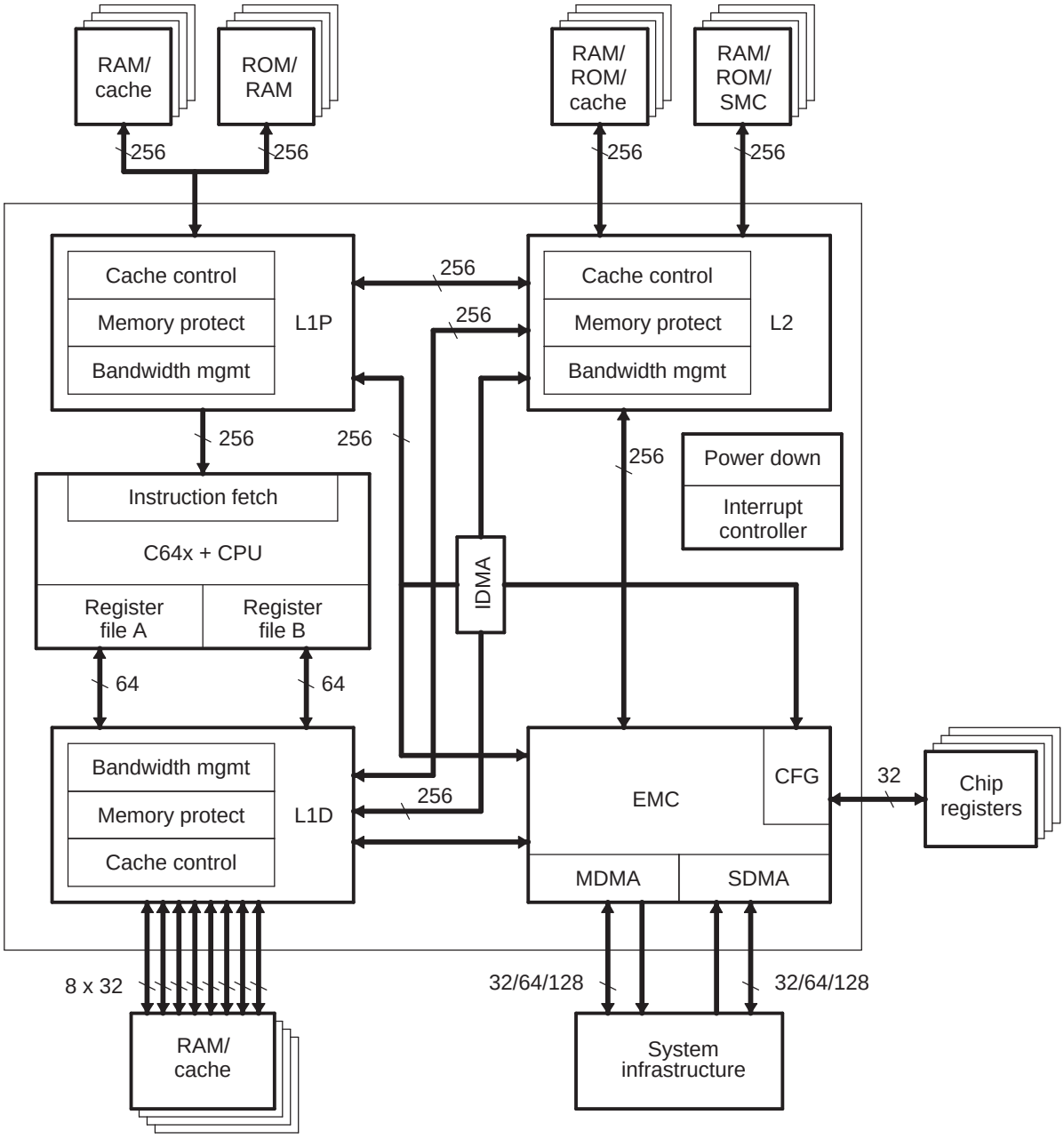
1.1 Introduction **18**
1.2 C64x+ Megamodule Overview **19**

1.1 Introduction

The C64x+™ megamodule includes the following components: C64x+ CPU, Level 1 program (L1P) memory controller, Level 1 data (L1D) memory controller, Level 2 (L2) memory controller, Internal DMA (IDMA), bandwidth management (BWM), interrupt controller (INTC), power-down controller (PDC), and an extended memory controller (EMC).

A block diagram of the megamodule is shown in Figure 1-1.

Figure 1-1. TMS320C64x+ Megamodule Block Diagram



1.2 C64x+ Megamodule Overview

The following sections provide an overview of the main components and features of the C64x+ megamodule.

1.2.1 C64x+ CPU

The C64x+ CPU is an extension of the C64x CPU. The C64x+ CPU provides new features such as:

- New instructions that increase execution performance.
- Increased code compactness.
- Software and hardware exception for better debugging capabilities.

The C64x+ devices are object code compatible with the C64x devices.

The C64x+ CPU is not described further in this document. For more information on the C64x+ CPU, refer to the *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide* (SPRU732).

1.2.2 Level 1 Program (L1P) Memory Controller

The L1P memory controller interfaces the CPU fetch pipeline to L1P memory. You can configure part of the L1P memory as a one-way set-associative cache. Cache sizes of 4KB, 8KB, 16KB, or 32KB are supported.

The L1P provides bandwidth management, memory protection, and power-down support. The L1P memory is always initiated to either all SRAM or maximum cache after reset. The behavior is specific to each C64x+ device.

For more information on the L1P cache/memory, refer to Chapter 2.

1.2.3 Level 1 Data (L1D) Memory Controller

The L1D memory controller interfaces the CPU data path to L1D memory. Part of the L1D memory can be configured as a two way set-associative cache. Cache sizes of 4KB, 8KB, 16KB, or 32KB are supported.

The L1D provides bandwidth management, memory protection, and power-down support. The L1D memory is always initiated to either all SRAM or maximum cache after reset. The behavior is specific to each device.

For more information on the L1D cache/memory, refer to Chapter 3.

1.2.4 Level 2 (L2) Memory Controller

The L2 memory controller interfaces level 1 memories to higher-level memories. Part of the L2 memory can be configured as a four way set-associative cache. Cache sizes of 32KB, 64KB, 128KB, or 256KB are supported.

The L2 provides bandwidth management, memory protection, and power-down support. The L2 memory is always initiated to all SRAM after reset. If you want to initiate cache modes, you must do so during device run time.

If you configure part of internal memory as cache, the L2 controller provides a means of writing back changes made to its contents, or invalidating the cache's contents altogether. This can be performed on a block or global basis. These actions constitute coherence operations that you specify; that is, they are intended to make the cached information coherent with the original memory location's content. Writebacks and invalidations also occur automatically by virtue of how cache architectures operate. These activities are generally called coherence operations throughout this document. Coherence operations are described in more detail in Chapter 2, Chapter 3, and Chapter 4.

Refer to Chapter 4 for more information on the L2 cache/memory.

1.2.5 Internal DMA (IDMA)

The internal DMA (IDMA), is a DMA local to the megamodule- that is, it provides data move services only within the megamodule (L1P, L1D, L2, and CFG).

There are two IDMA channels (0 and 1).

- Channel 0 allows data to transfer between the peripheral configuration space (CFG) and any local memories (L1P, L1D, and L2).
- Channel 1 enables data to transfer between the local memories (L1P, L1D, and L2).

The IDMA data transfers occur in the background of CPU operation. That is, once a channel transfer is programmed, it happens concurrent with other CPU activity, and without additional CPU intervention.

For more information on the IDMA, refer to Chapter 5.

1.2.6 Bandwidth Management (BWM)

The C64x+ megamodule includes a set of resources (L1P, L1D, L2, and configuration bus) and a set of requestors (CPU, SDMA, IDMA, and coherence operations) that need to use these resources. In order to avoid blocking a requestor from accessing a resource for a long period of time, the C64x+ megamodule implements a bandwidth management scheme in order to assure some bandwidth to all of the requestors.

Refer to Chapter 6 for more information on the BWM.

1.2.7 Interrupt Controller (INTC)

The C64x+ CPU provides two types of asynchronous signaling services:

- Interrupts
- Exceptions

Interrupts provide the means to redirect normal program flow due to the presence of an external or internal hardware signal. Exceptions are similar in that they also redirect program flow, but they are normally associated with error conditions in the system.

The C64x+ CPU can receive 12 maskable/configurable interrupts, 1 maskable exception, and 1 unmaskable interrupt/exception. The CPU can also respond to a variety of internal exception conditions, though these are documented in the *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide* (SPRU732) , since they are wholly contained within the CPU .

The megamodule includes an interrupt controller that allows up to 124 system events to be routed to the CPU interrupt/exception inputs. These 124 events can either be directly connected to the maskable interrupts, or grouped together as interrupts or exceptions. These various routing choices allow a great deal of flexibility in event handling.

An error event is signaled when an interrupt is signaled to the CPU and there is already a flag pending for this interrupt. In addition to routing events, the interrupt controller detects when the CPU misses an interrupt. You can use this error event to notify the CPU when it misses a real time event. The INTC hardware saves the missed interrupt number in a register so that corrective action can be taken.

Refer to Chapter 7 for more information on the INTC.

1.2.8 Memory Protection Architecture (MPA)

The C64x+ megamodule offers memory protection support for its local memories (L1P, L1D, and L2). System level memory protection is device-specific and is not available on all devices. Refer to the device-specific data manual for more information.

Memory protection is defined globally, but implemented locally. Thus, the overall protection scheme is defined for the entire C64x+ megamodule, but each resource implements its own protection hardware. This distributed method of memory protection means you only need to learn one memory protection interface, while the C64x+ remains flexible enough to support future peripherals and memories.

To implement the memory protection scheme, the memory map is divided into "pages" and each page has an associated set of permissions. Invalid accesses are signaled with an exception and reported to the system in memory fault registers. Additionally, MPA supports privilege modes (supervisor and user) and memory locks.

Due to the distributed implementation of the memory protection scheme, the overall definitions are described in Chapter 8. Refer to each resource's chapter for specific details for MPA implementation.

1.2.9 Power-Down Controller (PDC)

The power-down controller allows software-driven power-down management for all of the C64x+ megamodule components. The CPU can power-down all or part of the C64x+ megamodule through the power-down controller based on its own execution thread or in response to an external stimulus from a host or global controller.

Refer to Chapter 9 for more information on the PDC.

1.2.10 Extended Memory Controller (EMC)

The extended memory controller (EMC) is a bridge from the megamodule to the rest of the device. It includes three ports:

- Configuration registers (CFG) - This port provides access to the memory-mapped registers which control various peripherals and resources on C64x+ devices.

  NOTE: This port does not provide access to those control registers found within the CPU or the megamodule.

- Master DMA (MDMA) - The master DMA provides access to resources outside of the megamodule for transactions initiated within the megamodule (i.e., those accesses where the megamodule is the master for the transaction). The master DMA is typically used for CPU/cache accesses to memory beyond the L2 level. These accesses can be in the form of cache line allocates, writebacks, and non-cacheable loads and stores to system memory.
- Slave DMA (SDMA) - The slave DMA provides access to resources inside the megamodule to system masters found outside the megamodule such as DMA controllers, HPI, etc. That is, transfers initiated outside the megamodule where the megamodule is the slave in the transaction.

The CFG bus is always 32 bits wide, and should always be accessed as 32-bit values using 32-bit load / store instructions or the IDMA. The MDMA and SDMA ports can be 32, 64, or 128 bits wide; their actual width is specific to each device, so refer to the device-specific data manual.

Level 1 Program Memory and Cache

Topic **Page**

2.1 Introduction 24
2.2 Terms and Definitions 24
2.3 L1 Program Memory Architecture 24
2.4 L1P Cache 25
2.5 Program Initiated Coherence Operations 29
2.6 L1P Cache Control Registers 30
2.7 L1P Performance 34
2.8 Power-Down Support 36
2.9 L1P Memory Protection 37

2.1 Introduction

2.1.1 Purpose of the Level 1 Program (L1P) Memory and Cache

The purpose of the Level 1 program (L1P) memory and cache is to maximize performance of the code execution. The configurability of the L1P cache offers the flexibility required in many systems.

2.1.2 Features

The L1P memory and cache provide the memory flexibility that is required in devices that use the C64x+ megamodule.

- Configurable L1P cache size: 0K, 4K, 8K, 16K, and 32K.
- Memory protection.
- Cache block and global coherence operations.

2.2 Terms and Definitions

Refer to Appendix A and Appendix B of this document for detailed definitions of the terms used in this chapter. Appendix A describes general terms used throughout the this reference guide and Appendix B defines terms related to the memory and cache architecture.

2.3 L1 Program Memory Architecture

2.3.1 L1P Memory

L1P memory supports up to 1 MB of RAM and ROM. This memory is divided up into two regions. Each region may be no larger than 512KB. L1P memory can not be cached within Level 1 data (L1D) cache, Level 1program (L1P) cache, or Level 2 (L2) cache within the same megamodule.

The L1P memory's base address is constrained to 1MB boundaries. The total size of L1P memory must be a multiple of 16K bytes.

2.3.1.1 L1P Regions

L1P memory is divided into two regions (denoted L1P region 0 and L1P region 1).

The regions have two primary impacts on L1P:

1. Each region may have a different number of wait-states.
2. Each region has separate memory protection entries.

The two regions appear consecutively in memory. Region 0 may be 0K bytes (disabled) or any power-of-2 size in the 16K to 512K range. Region 1 starts at the end of region 0. Its size may be any multiple of 16K, from 16K to 512K bytes. The size of region 1 must be less than or equal to the size of region 0, when region 0 is enabled.

The two L1P regions divide the memory protection entries into two sets. L1P provides 32 memory protection pages, as described in Section 2.9.2. The first 16 pages cover region 0, and the second 16 pages cover region 1. When region 0 is 0K bytes, its memory protection pages go unused.

The actual memory configuration is device-specific. Refer to the device-specific data manual for more information.

2.3.1.2 L1P Access

The L1P regions can only be written to using EDMA or IDMA accesses; the L1P regions cannot be written to using CPU stores. The L1P regions can be read from using EDMA or IDMA accesses. CPU access is limited to instruction fetch. The CPU cannot read from L1P, even if L1P is memory-mapped. See Section 2.9.1.2.

2.3.1.3 L1P Wait States

The two regions may have different wait states. The maximum number of wait states is 3. The number of wait states is *not* configurable in software, it is defined when the chip is created. L1P SRAM typically has 0 wait states. ROM at the L1 level may have greater than 0 wait states. Refer to the device-specific data manual for more information.

2.4 L1P Cache

L1P cache is necessary to facilitate fetching program code at a fast clock rate in order to maintain a large system memory. The cache is responsible for hiding the latency associated with reading from and writing to the slower system memory.

It is possible to convert part or all of L1P into cache. L1P supports cache sizes of 4K, 8K, 16K, and 32K.

L1P cache converts memory from RAM to cache by starting at the top of the L1P memory map and working downwards. To explain, the highest addresses of L1P region 1 are the first to become cache. L1P cache may only occupy region 1.

The cache controller initializes after resetting to either "all RAM" or "maximal cache." Refer to the device-specific data manual for specific behavior.

The operation of the L1P cache is controlled through several registers. Table 2-1 provides a summary of these registers. These registers are mentioned throughout this section and will be described in more detail in Section 2.6.

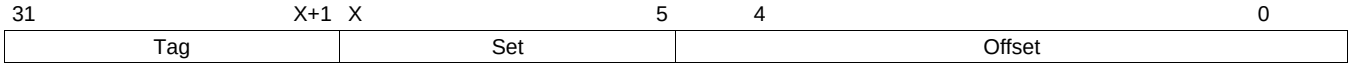
Table 2-1. L1P Cache Registers Summary

Address	Acronym	Register Description	Section
0184 0020h	L1PCFG	Level 1 Program Configuration Register	Section 2.6.3.1
0184 0024h	L1PCC	Level 1 Program Cache Control Register	Section 2.6.3.2
0184 4020h	L1PIBAR	Level 1 Program Invalidate Base Address Register	Section 2.6.3.3
0184 4024h	L1PIWC	Level 1 Program Invalidate Word Count Register	Section 2.6.3.4
0184 5028h	L1PINV	Level 1 Program Invalidate Register	Section 2.6.3.5

2.4.1 L1P Cache Architecture

The L1P cache is a direct-mapped cache, meaning that every physical memory location in the system has one possible location in the cache where it may reside. When the CPU attempts to fetch a piece of code, L1P must check whether the requested address resides in the L1P cache. To do so, the 32-bit address provided by the CPU is partitioned into three fields (tag, set, and offset), as shown in Figure 2-1.

Figure 2-1. Data Access Address Organization



The offset of 5 bits accounts for the fact that an L1P line size is 32 bytes. The cache control logic ignores bits 0 through 4 of the address. The set field indicates the L1P cache line address where the data would reside, if it were cached. The width of the set field depends on the amount of L1P configured as cache. L1P uses the set field to look up and check the tag for any already-cached data from that address, as well as the valid bit, which indicates whether the address in the tag actually represents a valid address held in cache.

The tag field is the upper portion of the address that identifies the true physical location of the data element. On a program fetch, if the tag matches and the corresponding valid bit is set, then it is a "hit," and the data is read directly from the L1P cache location and returned to the CPU. Otherwise, it is a "miss" and the request is sent on to the L2 controller for the data to be fetched from its location in the system. Misses may or may not directly result in CPU stalls.

The CPU cannot write data to L1P under normal circumstances.

The L1P cache configuration dictates the size of the set and tag fields.

2.4.2 Replacement and Allocation Strategy

The L1P cache operates as a direct-mapped cache in all cache configurations. This means that each location in system memory can reside in exactly one location in the L1P cache. Because L1P is direct-mapped, its replacement strategy is simple: each newly cached line replaces the previously cached line.

The L1P controller implements a read-allocate cache. This means that the L1P will fetch a complete line of 32 bytes on a read miss.

2.4.3 L1P Mode Change Operations

The C64x+ L1P architecture allows the size of L1P cache to be selected at run time. Programs select the size of L1P cache by writing the requested mode to the L1PMODE field in the L1PCFG register.

Table 2-2. Cache Size Specified by the L1PMODE bit in the L1PCFG Register

L1PMODE Setting of the L1PCFG Register	Amount of L1P Cache
000b	0K
001b	4K
010b	8K
011b	16K
100b	32K
101b	"Maximal Cache." Maps to 32K.
110b	
111b	"Maximal Cache." Maps to 32K.

NOTE: In general, a larger value of L1PMODE specifies a larger cache size (up to the size of the implemented L1P memory). The maximum L1P cache size is the smaller of "largest power-of-2 that fits in L1P RAM size" and 32K.

The actual range of L1P cache modes is constrained by the size of L1P region 1. The L1P cache can be no larger than 16K when L1P region 1 is only 16K in size. Thus, the encoding 011b through 111b are mapped to 16K cache on devices whose L1P region 1 is only 16K.

On these devices, the L1PMODE settings 100b through 111b select the 16K cache mode, as opposed to the 32K cache mode. Thus, modes 000b through 011b always select the requested size, 0K through 16K. Modes 100b through 111b select the maximum size implied by the size of L1P memory (16K or 32K).

As a result of this policy, programs wanting no more than a certain amount of cache should program the value corresponding to this upper bound. Programs desiring "as much cache as possible" should program 111b into L1PMODE.

When programs initiate a cache mode change, the L1P cache itself invalidates its current contents. This ensures that no false hits occur due to changing interpretation of cache tags.

While the invalidation is necessary to ensure correct cache behavior, it is not sufficient to prevent data loss due to portions of L1P RAM becoming cache. Thus, to safely change L1P cache modes, applications must adhere to the procedure in Table 2-3.

Table 2-3. Switching L1P Modes

To switch from. . .	To. . .	The program must perform the following steps. . .
A mode with *no* or *some* L1P cache	A mode with *more* L1P cache	1. DMA, IDMA or copy any needed data out of the affected range of L1P RAM. (If none requires saving, no DMA is necessary). 2. Write the desired cache mode to the L1PMODE field in the L1PCFG register. 3. Read back L1PCFG. This stalls the CPU until the mode change completes.
A mode with *some* L1P cache	A mode with *less* or *no* L1P cache	1. Write the desired cache mode to the L1PMODE field in the L1PCFG register. 2. Read back L1PCFG. This stalls the CPU until the mode change completes.

2.4.4 L1P Freeze Mode

The L1P cache directly supports a freeze mode of operation for applications. This mode allows applications to prevent CPU data accesses from evicting program code from the cache. This feature is useful in an interrupt context. L1P freeze mode only affects L1P cache. L1P RAM is not affected by freeze mode.

While in freeze mode, the L1P cache will service read hits normally. Read hits return data from the cache. In freeze mode, the L1P cache will not allocate new cache lines on read misses, nor will it cause any existing cache contents to be marked invalid.

The OPER field in the L1PCC register controls whether L1P is frozen or it is operating normally, as shown in Example 2-1.

The CPU places L1P into freeze mode by writing 001b to the OPER field in the L1PCC register. The CPU returns L1P to normal operation by writing 0b to the OPER field in the L1PCC register.

The POPER field in the L1PCC register holds the previous value of the OPER field. The value of the OPER field is copied to the POPER field in the L1PCC register on writes to the L1PCC register. Copying the value of the OPER field to the POPER field alleviates the cycle cost of reading the L1PCC register (in order to save the previous value of the OPER field) before it is written. If the POPER field is not in the L1PCC register, the program must read, write, and then read again to fully freeze the cache while recording its previous operating mode. If the POPER field is in the L1PCC register, this operation reduces to a single write followed by a read.

When you write to the L1PCC register, the following operations occur:

1. The OPER field copies to the POPER field in the L1PCC register.
2. The POPER field loses its previous value.
3. The OPER field updates according to the value that the CPU writes to bit 0 of the L1PCC register. Thus, writing to the L1PCC register only modifies the OPER field in this register.

Programs cannot directly modify the POPER field with a single write. This is not problematic since the value held in the POPER field does not change the behavior of L1P cache and only interests programs that have recently written to the OPER field.

The software must perform a write to the L1PCC register followed by a read of the L1PCC to ensure that the L1PCC updates. Performing a write to followed by a read of the L1PCC register guarantees that the requested mode is in effect.

The goal of the OPER field in the L1PCC register is to avoid the substantial CPU cycle penalty and code size involved in a read-write-reread sequence that would otherwise be necessary. Thus, applications may quickly freeze L1P and record the previous "freeze" state of L1P with the short sequence of code in Example 2-1.

Example 2-1. L1P Quick Freeze Example Code Sequence

```
   MVKL  L1PCC, A0   ; Point to L1PCC
   MVKH  L1PCC, A0   ;
|| MVK   1b, B0      ; OPER encoding for 'freeze'
   STW   B0, *A0[0]  ; Write 1b to L1PCC.OPER
   LDW   *A0[0], A1  ; Read L1PCC to get L1PCC.POPER
   NOP   4
  ; At this point, L1P is frozen, and the CPU has the old OPER value ; in bit 16 of A1.
```

The L1PCC can be used for unfreezing the cache in a manner similar to how it was frozen. Example 2-2 illustrates how this is performed:

Example 2-2. Restore Example Code Sequence for the OPER bit in the L1PCC

```
; Assume A1 holds value read in at the end of the L1P Quick Freeze Example Code Sequence above.
   MVKL L1PCC, A0   ; Point to L1PCC
   MVKH L1PCC, A0   ;
|| SHRU A1, 16, A1  ; Shift POPER field into OPER's position
   STW A1, *A0[0]   ; Write to L1PCC, restoring old value of OPER
   LDW *A0[0], A1   ; Read back L1PCC to ensure change is complete
   NOP 4
  ; At this point, L1P is in its previous state (frozen or unfrozen)
```

The L1D cache also supports a freeze mode. Refer to Chapter 3 for more details on the L1D cache freeze mode.

It is often desirable to freeze both caches together. Therefore, consecutive writes to L1DCC and L1PCC followed by reading both L1DCC and L1PCC are sufficient to ensure that both L1D and L1P are frozen. Example 2-3 illustrates a sequence that freezes both L1D and L1P.

Example 2-3. Example Code Sequence for Freezing L1D and L1P Simultaneously

```
   MVKL L1DCC, A0    ; \
|| MVKL L1PCC, B0    ;  |__ Generate L1DCC pointer in A0
   MVKH L1DCC, A0    ;  |   and L1PCC pointer in B0
|| MVKH L1PCC, B0    ; /
|| MVK 1b, A1        ; \___ OPER encoding for 'freeze'
|| MVK 1b, B1        ; /    in both A1 and B1.
   STW A1, *A0       ; Write to L1DCC.OPER
|| STW B1, *B0       ; Write to L1PCC.OPER
   LDW *A0, A1       ; Get old freeze state into A1 from L1DCC
|| LDW *B0, B1       ; Get old freeze state into B1 from L1PCC
   NOP 4
  ; At this point, L1D and L1P are frozen.
  ; The old value of L1DCC.OPER is in bit 16 of A1.
  ; The old value of L1PCC.OPER is in bit 16 of B1.
```

2.5 Program Initiated Coherence Operations

The C64x+ L1P architecture supports program-initiated cache coherence operations. These operations operate either on a block of addresses or on the entire L1P cache.

2.5.1 Global Coherence Operation

Global cache operations synchronize L1P with "the system" between major events, such as a task switch, L1P mode change, or change to memory protection settings. Thus, global cache operations are viewed as “synchronous” with respect to other program activity.

You can globally invalidate the L1P cache under software control. The program must write a 1 to the I bit of the L1PINV register in order to initiate a global invalidation operation.

During the global invalidation of the L1P cache, no writeback operation is performed because code is not modified.

The I bit of the L1PINV register resets to 0 upon completion of the global coherence operation. The program can poll this field to detect the completion of the operation. The polling code must be located outside the L1P cache.

Table 2-4 provides a summary of the L1P global coherence operations.

Table 2-4. L1P Global Coherence Operations

Cache Operation	Register Used	L1P Effect
L1P Invalidate	L1PINV	All lines invalidated in L1P.

You can also globally invalidate the L1P cache by setting the IP bit in the L2CFG register to 1. The IP bit provides backward compatibility with C64x and C64x+ devices. You should not use the IP bit in new applications; use the L1PINV register for new applications.

2.5.2 Block Coherence Operation

Block coherence operations have similar functionality as the global coherence operation; however, they apply only to a defined block of code. This block is defined by the base address and the word size (32-bit) in the associated L1PIBAR and L1PIWC, respectively.

The block coherence operations are designed to be as efficient as possible, while minimizing the impact to tasks running concurrently on the CPU. Block cache operations are running in the “background” of CPU activity.

The L1P invalidate word count field of the L1PIWC sets to 0 upon completion of the block coherence operation. The program can poll this field to detect the completion of the operation. The polling code must be located outside of the affected block of the L1P cache.

Table 2-5 provides a summary of the L1P block cache operations.

Table 2-5. L1P Block Cache Operations

Cache Operation	Register Used	L1P Effect
L1P Invalidate	L1PIBAR L1PIWC	All lines in range invalidated in L1P.

2.6 L1P Cache Control Registers

2.6.1 Memory Mapped Cache Control Register Overview

The C64x+ megamodule memory system provides a set of registers to govern the operation of L1P cache. These registers allow for changing cache modes and manually initiating cache coherence operations.

Table 2-6 lists the registers for the L1P specific cache control operations registers. See the device-specific data manual for the memory address of these registers.

Refer to Chapter 4 for a detailed list of the available cache control operations provided.

Table 2-6. L1P Specific Cache Control Operations Registers

Address	Acronym	Register Description	Section
0184 0020h	L1PCFG	Configures the size of L1P cache	Section 2.6.3.1
0184 0024h	L1PCC	Controls L1P operating mode (freeze / normal)	Section 2.6.3.2
0184 4020h	L1PIBAR	Specified range is invalidated in L1P without being written back	Section 2.6.3.3
0184 4024h	L1PIWC		Section 2.6.3.4
0184 5028h	L1PINV	Entire contents of L1P is invalidated without being written back.	Section 2.6.3.5

In addition to the L1P-specific registers listed above, the L1P cache is directly affected by writes to L2-specific controls as well. Refer to Chapter 4 for the complete list of cache control operations and their affect on the L1P cache.

2.6.2 CPU Cache Control Registers

The CPU has a single internal control register, the control state register (CSR) that dedicates a field to cache control operations (the PCC field in the CSR register). The PCC field is provided for backward compatibility with C64x+/C64x/C62x/C67x devices. You should not use this field in new applications. Use the L1PCFG and L1PCC registers described in Section 2.6.3 instead.

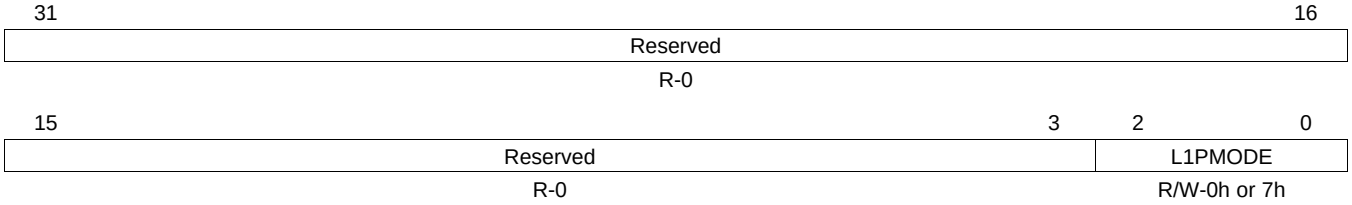
2.6.3 L1P Cache Configuration Registers

The L1P configuration register (L1PCFG) and the L1P cache control register (L1PCC) control the operation of L1P.

2.6.3.1 L1P Configuration Register (L1PCFG)

The L1P configuration register (L1PCFG) controls the size of L1P cache and is shown in Figure 2-2 and described in Table 2-7.

Figure 2-2. L1P Configuration Register (L1PCFG)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-7. L1P Configuration Register (L1PCFG) Field Descriptions

Bit	Field	Value	Description
31-3	Reserved	0	Reserved
2-0	L1PMODE	0-7h	Defines the size of the L1P cache. The L1PMODE field powers-up as either 0h or 7h. Refer to the device-specific data manual for more information.
		0	L1P cache disabled
		1h	4K
		2h	8K
		3h	16K
		4h	32K
		5h	Maximal cache
		6h	Maximal cache
		7h	Maximal cache

2.6.3.2 L1P Cache Control Register (L1PCC)

The L1PCC cache control register (L1PCC) controls whether L1P is frozen or unfrozen.

The L1P cache control register (L1PCC) is shown in Figure 2-3 and described in Table 2-8.

Figure 2-3. L1P Cache Control Register (L1PCC)

31	17	16
Reserved		POPER
R-0		R-0

15	1	0
Reserved		OPER
R-0		R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-8. L1P Cache Control Register (L1PCC) Field Descriptions

Bit	Field	Value	Description
31-17	Reserved	0	Reserved
16	POPER	0-1	Holds the previous value of the OPER bit.
15-1	Reserved	0	Reserved
0	OPER		Controls the L1P freeze mode.
		0	Freeze mode disabled
		1	Freeze mode enabled

2.6.3.3 L1P Invalidate Base Address Register (L1PIBAR)

The L1P invalidate base address register (L1PIBAR) defines the base address of the block invalidation that the coherence operation will act upon.

The L1P invalidate base address register (L1PIBAR) is shown in Figure 2-4 and described in Table 2-9.

Figure 2-4. L1P Invalidate Base Address Register (L1PIBAR)

31	0
L1P Invalidate Base Address	
W-x	

LEGEND: W = Write only; -x, value is indeterminate, see your device-specific data manual

Table 2-9. L1P Invalidate Base Address Register (L1PIBAR) Field Descriptions

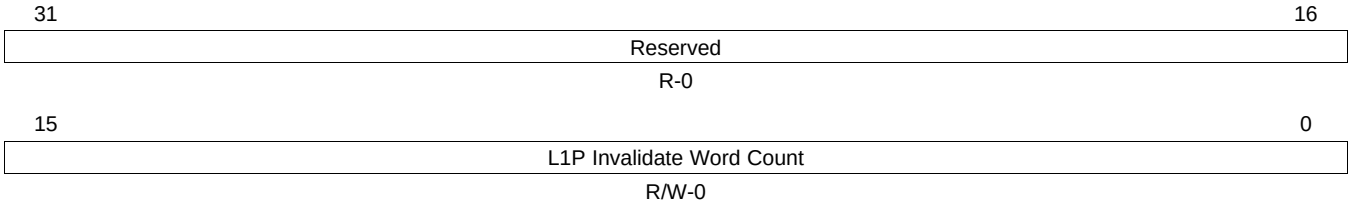
Bit	Field	Value	Description
31-0	L1PIBAR	0-FFFF FFFFh	32-bit base address for block invalidation.

2.6.3.4 L1P Invalidate Word Count (L1PIWC)

The L1P invalidate word count register (L1PIWC) defines the size of the block invalidation that the coherence operation will act upon. The size is defined in 32-bit words.

The L1P invalidate word count register (L1PIWC) is shown in Figure 2-5 and described in Table 2-10.

Figure 2-5. L1P Invalidate Word Count Register (L1PIWC)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

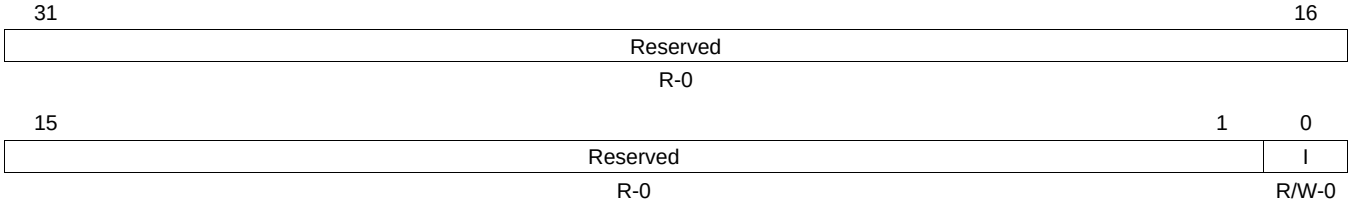
Table 2-10. L1P Invalidate Word Count Register (L1PIWC) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	L1PIWC	0-FFFFh	Word count for block invalidation.

2.6.3.5 L1P Invalidate Register (L1PINV)

The L1P invalidate register (L1PINV) controls the global invalidation of the L1P cache and is shown in Figure 2-6 and described in Table 2-11

Figure 2-6. L1P Invalidate Register (L1PINV)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-11. L1P Invalidate Register (L1PINV) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	I		Controls the global invalidation of L1P cache.
		0	Normal operation.
		1	All L1P cache lines are invalidated.

2.6.4 Privilege and Cache Control Operations

The impact of privilege on cache control operations can be summarized as follows:

- Supervisor code may change L1P cache size.
- User-mode code may not change L1P cache size.
- Both supervisor and user mode code may issue invalidates to L1P.
- Both supervisor and user modes may freeze or unfreeze L1P at any time.

Table 2-12 summarizes who may access which L1P cache control registers.

Table 2-12. Permissions for L1P Cache Control Registers

Register	Supervisor	User
L1PCFG	R/W	R
L1PCC	R/W	R/W
L1PINV	R/W	R/W
L1PIBAR	W	W
L1PIWC	R/W	R/W

2.7 L1P Performance

2.7.1 L1P Miss Penalty

A program fetch which hits L1P completes in a single cycle without stalling the CPU. An L1P miss that hits in L2 may stall the CPU for up to X cycles, depending on the parallelism of the execute packets in the vicinity of the miss. Section 2.7.2 describes this in more detail.

An L1P miss that misses in L2 cache stalls the CPU until the L2 retrieves the data from external memory and transfers the data to the L1P, which then returns the data to the CPU. This delay depends upon the type of external memory used to hold the program, as well as other aspects of system loading.

The C64x+ DSP allows an execute packet to span two fetch packets. This spanning does not change the penalty for a single miss. However, if both fetch packets are not present in L1P, two cache misses occur.

2.7.2 L1P Miss Pipelining

Miss pipelining can hide much of this overhead by overlapping the processing for several cache misses. Additionally, some amount of the cache miss overhead can be overlapped with dispatch stalls that occur in the fetch pipeline.

For L1P miss pipelining to be effective, there must be multiple outstanding cache misses. The C64x+ DSP fetch pipeline accomplishes this by attempting to fetch one new fetch packet every cycle, as long as there is room in the fetch pipeline. To understand how this works, it is necessary to understand the nature of the fetch pipeline itself.

The fetch and decode pipeline is divided into 6 stages leading up to, but not including the first execution stage, E1. The stages are:

- PG - Program Generate
- PS - Program Send
- PW - Program Wait
- PR - Program Read
- DP - Dispatch
- DC - Decode

The C6000 DSP instructions are grouped into two groupings: fetch packets and execute packets. The CPU fetches instructions from memory in fixed bundles of 8 instructions, known as fetch packets. The instructions are decoded and separated into bundles of parallel-issue instructions known as execute packets. A single execute packet may contain between 1 and 8 instructions. Thus, a single fetch packet may contain multiple execute packets. An execute packet may also span two fetch packets on the C64x+ DSP. The program read (PR) stage of the pipeline is responsible for identifying a sequence of execute packets within a sequence of fetch packets. The dispatch (DP) stage is responsible for extracting and dispatching them to functional units.

As a result of the disparity between fetch packets and execute packets, the entire fetch pipeline need not advance every cycle. Rather, the PR pipeline stage only allows the program wait (PW) stage to advance its contents into the PR stage when the DP stage has consumed the complete fetch packet held in PR. The stages before PR advance as needed to fill in gaps. Thus, when there are no cache misses, the early stages of the fetch pipeline are stalled while the DP stage pulls the individual execute packets from the current fetch packet. These stalls are referred to as dispatch stalls.

The C64x+ DSP takes advantage of these dispatch stalls by allowing the earlier stages of the pipeline to advance toward DP while cache misses for those stages are still pending. Cache misses may be pending for the PR, PW, and PS pipeline stages. It is not necessary to expose these cache stalls to the CPU because the DP stage stalls the PR stage with a dispatch stall while it consumes the fetch packets in the PR stage of the pipeline. When a fetch packet is consumed completely; however, the contents of the PW stage must advance into the PR stage. At this point, the CPU stalls if DP requests an execute packet from PR for which there is still an outstanding cache miss.

When a branch is taken, the fetch packet containing the branch target advances through the fetch pipeline every cycle until the branch target reaches the E1 pipeline stage. Branch targets override the dispatch stall described above. As a result, they do not gain as much benefit from miss pipelining as other instructions. However, the fetch packets that immediately follow a branch target do benefit. Although the code in the fetch packets that follows the branch target may not execute immediately, the branch triggers several consecutive fetches for this code. Thus, it pipelines any misses for that code. In addition, no stalls are registered for fetch packets that were requested prior to the branch being taken, but that never made it to the DP pipeline stage.

The miss performance is measured with sustained back-to-back misses in straight-line (non-branching) code incurs an average miss penalty based on the average parallelism of the code. The average miss penalty for a long sequence of sustained misses in straight-line code is summarized in Table 2-13. The code is fetched from L2SRAM. Two different configurations are presented. The first configuration features 0 wait state for L2SRAM, 2 × 128 bit banks. This configuration is available in the DM644x devices. The second configuration features 1 wait state L2SRAM, 4 × 128 bit banks. This configuration is available in the C645x devices.

Table 2-13. L1P Miss Pipelining Performance (Average Number of Stalls per Execute Packet)

L2 Type	0 wait state, 2 × 128 bit banks		1 wait state, 4 × 128 bit banks	
Instructions per Execute Packet	L2 SRAM	L2 Cache	L2 SRAM	L2 Cache
1	0.000	0.000	0.000	0.000
2	0.001	0.497	0.167	0.499
3	0.501	1.247	0.751	1.249
4	0.997	1.997	1.329	1.999
5	1.499	2.747	1.915	2.749
6	2.001	3.497	2.501	3.499
7	2.497	4.247	3.079	4.249
8	2.999	4.997	3.665	4.999

2.8 Power-Down Support

The L1P memory can be powered-down in several ways in order to save power.

2.8.1 Static Power-Down

The L1P memory is powered-down when the entire megamodule is powered-down. The following software sequence is required to power-down the C64x+ megamodule:

1. Enable power-down by setting the MEGPD field in the PDCCMD register to 1.
2. Enable the CPU interrupt(s) that you want to wake the megamodule up; disable all others.
3. Execute an IDLE instruction.

The megamodule stays in powered-down mode until the interrupts enabled in step 2 above are awakened.

The power-down controller command register (PDCCMD) is described in Chapter 9. If a DMA access occurs to the L1D, L1P, or L2 memory while the megamodule is powered-down, the PDC wakes up all three memory controllers. When the DMA access has been serviced, the PDC will power-down the memory controllers again.

NOTE: Powering-down the megamodule as described here is often called static power-down. This term is used to describe this mode since it is often used for longer periods of time. The use of the term dynamic power-down elsewhere in this chapter implies that they are used for limited periods of time.

2.8.2 Dynamic Power-Down

The L1P memory is automatically powered-down while the CPU executes code from the SPLOOP buffer. For more information about the SPLOOP buffer, refer to the *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide* (SPRU732) .

2.8.3 Feature-Oriented Power-Down

When the L1P cache is disabled (000b is written to the MODE field of the L1PCFG register) it is in a power-down state to conserve energy further.

2.9 L1P Memory Protection

L1P memory supports memory protection to offer the robustness required in many systems. Several levels of memory protection are available. Not all of the levels are available on all of the devices. Refer to the device-specific data manual for more information. Refer to Chapter 8 for the details of C64x+ protection.

2.9.1 Protection Checks on L1P Accesses

Unlike the L1D, L1P implements different memory protection rules for CPU program fetches from L1P memory versus CPU DMA and IDMA accesses to L1P memory. The following sections detail those differences.

All three memory controllers feature two exception outputs which are routed to the megamodule interrupt controller. One of these exception outputs indicates that a CPU-triggered (“local”) memory exception occurred. The other indicates that a DMA-triggered (“remote”) exception occurred. Most programs will likely route the CPU-triggered exception input to the CPU’s exception input and the DMA-triggered input to an interrupt input.

2.9.1.1 Protection Checks on CPU Program Fetches

L1P performs memory protection checks on all fetches. Each fetch packet has two permission bits associated with it (shown in Table 2-14) that determine the execution privileges associated with the code contained within the cache line or corresponding region of L1P RAM/L1P ROM. L1P provides the results of the permission checks for all fetches, regardless of where the fetched data eventually arrives from.

Table 2-14. Permission Bits Examined With Each Fetch

Bit	Description
UX	User mode may eXecute
SX	Supervisor mode may eXecute

2.9.1.2 Protection Checks on CPU Data Accesses

The CPU cannot directly access L1P RAM and ROM via load and store instructions. However, it can attempt to access L1P’s control registers with load and store instructions.

The permissions associated with the registers are checked for reads of its registers. L1P checks writes off of its registers. It will signal a CPU-triggered memory exception in response to disallowed writes. The details of the exception are recorded in L1PMPFAR/L1PMPFSR, and L1P signals a CPU memory protection exception event to the interrupt controller.

2.9.1.3 Protection Checks on DMA/IDMA Accesses

DMA and IDMA access to L1P memory are constrained to L1P RAM and ROM. DMA/IDMA cannot access L1P cache. Accesses to the L1P RAM under the L1P cache are governed by protection entries associated with the L1P RAM.

Each DMA/IDMA access is checked against the SR/SW/UR/UW and access ID fields for the corresponding memory protection page. DMA/IDMA accesses to region 0 index into the first 16 memory protection pages. DMA/IDMA accesses to region 1 index into the second 16 memory protection pages.

Upon an invalid access to L1P memory via a DMA or IDMA, the L1P signals an exception. The details of this exception are recorded in L1PMPFAR/L1PMPFSR. L1P signals a DMA memory protection exception event to the interrupt controller.

2.9.2 Memory Protection Registers

The following registers govern the operation of memory protection within L1P. The MMRs fall into three main categories:

- Memory Page Protection Attribute (MPPA) registers: Page attribute registers store the permissions associated with each protected page.
- Memory Protection Lock (MPLK) registers: Peripherals may choose to implement a hardware memory protection lock. When engaged, the lock disables all updates to the memory protection entries for that peripheral.
- Memory Protection Fault (MPFxR) registers: Each peripheral that generates memory protection faults provides MPFAR, MPFSR, and MPFCR registers for recording the details of the fault.

Table 2-15. Memory Protection Registers

Address	Acronym	Register Description	Section
0184 A6xxh	L1PMPPA*xx*	Level 1 Memory Page Protection Attribute Registers	Section 2.9.2.1
0184 A500h	L1PMPLK0	Level 1 Memory Protection Lock Register 0 (L1PMPLK0)	Section 2.9.2.2.1
0184 A504h	L1PMPLK1	Level 1 Memory Protection Lock Register 1 (L1PMPLK1)	Section 2.9.2.2.2
0184 A508h	L1PMPLK2	Level 1 Memory Protection Lock Register 2 (L1PMPLK2)	Section 2.9.2.2.3
0184 A50Ch	L1PMPLK3	Level 1 Memory Protection Lock Register 3 (L1PMPLK3)	Section 2.9.2.2.4
0184 A510h	L1PMPLKCMD	Level 1 Memory Protection Lock Command Register (L1PMPLKCMD)	Section 2.9.2.2.5
0184 A514h	L1PMPLKSTAT	Level 1 Memory Protection Lock Status Register (L1PMPLKSTAT)	Section 2.9.2.2.6
0184 A400h	L1PMPFAR	Level 1 Memory Protection Fault Address Register (L1PMPFAR)	Section 2.9.2.3.1
0184 A404h	L1PMPFSR	Level 1 Memory Protection Fault Set Register (L1PMPFSR)	Section 2.9.2.3.2
0184 A408h	L1PMPFCLR	Level 1 Memory Protection Fault Clear Register (L1PMPFCLR)	Section 2.9.2.3.3

2.9.2.1 Memory Page Protection Attribute Registers

Table 2-16 lists the registers for the memory page protection.

L1P implements 32 memory protection pages. L1PMPPA0 through L1PMPPA15 correspond to region 0 and L1PMPPA16 through L1PMPPA31 correspond to region 1.

Refer to the device-specific data manual to determine the page size and number of pages used on a particular device.

Table 2-16. Memory Page Protection Attribute Registers

Address	Acronym	Register Description	Section
0184 A600h	L1PMPPA0	Level 1 Memory Page Protection Attribute Register 0	Section 2.9.2.1.1
0184 A604h	L1PMPPA1	Level 1 Memory Page Protection Attribute Register 1	Section 2.9.2.1.1
0184 A608h	L1PMPPA2	Level 1 Memory Page Protection Attribute Register 2	Section 2.9.2.1.1
0184 A60Ch	L1PMPPA3	Level 1 Memory Page Protection Attribute Register 3	Section 2.9.2.1.1
0184 A610h	L1PMPPA4	Level 1 Memory Page Protection Attribute Register 4	Section 2.9.2.1.1
0184 A614h	L1PMPPA5	Level 1 Memory Page Protection Attribute Register 5	Section 2.9.2.1.1
0184 A618h	L1PMPPA6	Level 1 Memory Page Protection Attribute Register 6	Section 2.9.2.1.1
0184 A61Ch	L1PMPPA7	Level 1 Memory Page Protection Attribute Register 7	Section 2.9.2.1.1
0184 A620h	L1PMPPA8	Level 1 Memory Page Protection Attribute Register 8	Section 2.9.2.1.1
0184 A624h	L1PMPPA9	Level 1 Memory Page Protection Attribute Register 9	Section 2.9.2.1.1
0184 A628h	L1PMPPA10	Level 1 Memory Page Protection Attribute Register 10	Section 2.9.2.1.1
0184 A62Ch	L1PMPPA11	Level 1 Memory Page Protection Attribute Register 11	Section 2.9.2.1.1
0184 A630h	L1PMPPA12	Level 1 Memory Page Protection Attribute Register 12	Section 2.9.2.1.1
0184 A634h	L1PMPPA13	Level 1 Memory Page Protection Attribute Register 13	Section 2.9.2.1.1
0184 A638h	L1PMPPA14	Level 1 Memory Page Protection Attribute Register 14	Section 2.9.2.1.1
0184 A63Ch	L1PMPPA15	Level 1 Memory Page Protection Attribute Register 15	Section 2.9.2.1.1
0184 A640h	L1PMPPA16	Level 1 Memory Page Protection Attribute Register 16	Section 2.9.2.1.1
0184 A644h	L1PMPPA17	Level 1 Memory Page Protection Attribute Register 17	Section 2.9.2.1.1
0184 A648h	L1PMPPA18	Level 1 Memory Page Protection Attribute Register 18	Section 2.9.2.1.1
0184 A64Ch	L1PMPPA19	Level 1 Memory Page Protection Attribute Register 19	Section 2.9.2.1.1
0184 A650h	L1PMPPA20	Level 1 Memory Page Protection Attribute Register 20	Section 2.9.2.1.1
0184 A654h	L1PMPPA21	Level 1 Memory Page Protection Attribute Register 21	Section 2.9.2.1.1
0184 A658h	L1PMPPA22	Level 1 Memory Page Protection Attribute Register 22	Section 2.9.2.1.1
0184 A65Ch	L1PMPPA23	Level 1 Memory Page Protection Attribute Register 23	Section 2.9.2.1.1
0184 A660h	L1PMPPA24	Level 1 Memory Page Protection Attribute Register 24	Section 2.9.2.1.1
0184 A664h	L1PMPPA25	Level 1 Memory Page Protection Attribute Register 25	Section 2.9.2.1.1
0184 A668h	L1PMPPA26	Level 1 Memory Page Protection Attribute Register 26	Section 2.9.2.1.1
0184 A66Ch	L1PMPPA27	Level 1 Memory Page Protection Attribute Register 27	Section 2.9.2.1.1
0184 A670h	L1PMPPA28	Level 1 Memory Page Protection Attribute Register 28	Section 2.9.2.1.1
0184 A674h	L1PMPPA29	Level 1 Memory Page Protection Attribute Register 29	Section 2.9.2.1.1
0184 A678h	L1PMPPA30	Level 1 Memory Page Protection Attribute Register 30	Section 2.9.2.1.1
0184 A67Ch	L1PMPPA31	Level 1 Memory Page Protection Attribute Register 31	Section 2.9.2.1.1

2.9.2.1.1 Memory Page Protection Attribute Registers

The size of each page differs from region to region and from one device to another. Some pages may not be used on a particular device. Program unused pages to a value of all zeroes for debug purposes.

Refer to the device-specific data manual to determine the page size and number of pages used on a particular device.

The general structure of the memory page protection attribute register (L1PMPPA*xx*) is shown in Figure 2-7 and described in Table 2-17.

Figure 2-7. Memory Page Protection Attribute Registers (L1PMPPA*x*)

31															16
Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AID5	AID4	AID3	AID2	AID1	AID0	AIDX	LOCAL	Reserved		SR	SW	SX	UR	UW	UX
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 2-17. Memory Page Protection Attribute Register (L1PMPPA*x*) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15	AID5		Controls access from ID = 5.
		0	Access denied.
		1	Access granted.
14	AID4		Controls access from ID = 4.
		0	Access denied.
		1	Access granted.
13	AID3		Controls access from ID = 3.
		0	Access denied.
		1	Access granted.
12	AID2		Controls access from ID = 2.
		0	Access denied.
		1	Access granted.
11	AID1		Controls access from ID = 1.
		0	Access denied.
		1	Access granted.
10	AID0		Controls access from ID = 0.
		0	Access denied.
		1	Access granted.
9	AIDX		Controls access from ID>=6
		0	Access denied.
		1	Access granted.
8	LOCAL		Controls access from CPU to local memories (L1/L2)
		0	Access denied.
		1	Access granted.
7-6	Reserved	0	Reserved.
5	SR		Supervisor read access type.
		0	Normal operation.
		1	Indicates a Supervisor read request.

Table 2-17. Memory Page Protection Attribute Register (L1PMPPA*x*) Field Descriptions (continued)

Bit	Field	Value	Description
4	SW		Supervisor write access type.
		0	Normal operation.
		1	Indicates a Supervisor write request.
3	SX		Supervisor execute access type.
		0	Normal operation.
		1	Indicates a Supervisor execute request.
2	UR		User read access type.
		0	Normal operation.
		1	Indicates a User read request.
1	UW		User write access type.
		0	Normal operation.
		1	Indicates a User write request.
0	UX		User execute access type.
		0	Normal operation.
		1	Indicates a User execute request.

2.9.2.2 Memory Protection Lock Registers

The L1P implements a 64-bit lock register for controlling write access to the memory protection registers.

Table 2-18 lists the registers for the memory protection lock. See the device-specific data manual for the memory address of these registers.

Table 2-18. Memory Protection Lock Registers

Address	Acronym	Register Description	Section
0184 A500h	L1PMPLK0	Memory Protection Lock Register 0 (L1PMPLK0)	Section 2.9.2.2.1
0184 A504h	L1PMPLK1	Memory Protection Lock Register 1 (L1PMPLK1)	Section 2.9.2.2.2
0184 A508h	L1PMPLK2	Memory Protection Lock Register 2 (L1PMPLK2)	Section 2.9.2.2.3
0184 A50Ch	L1PMPLK3	Memory Protection Lock Register 3 (L1PMPLK3)	Section 2.9.2.2.4
0184 A510h	L1PMPLKCMD	Memory Protection Lock Command Register (L1PMPLKCMD)	Section 2.9.2.2.5
0184 A514h	L1PMPLKSTAT	Memory Protection Lock Status Register (L1PMPLKSTAT)	Section 2.9.2.2.6

2.9.2.2.1 Memory Protection Lock Register 0 (L1PMPLK0)

The memory protection lock register 0 (L1PMPLK0) is shown in Figure 2-8.

Figure 2-8. Memory Protection Lock Register 0 (L1PMPLK0)

LEGEND: W = Write only; -x, value is indeterminate, see your device-specific data manual

2.9.2.2.2 Memory Protection Lock Register 1 (L1PMPLK1)

The memory protection lock register 1 (L1PMPLK1) is shown in Figure 2-9.

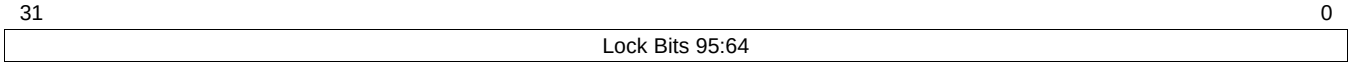
Figure 2-9. Memory Protection Lock Register 1 (L1PMPLK1)

LEGEND: W = Write only; -x, value is indeterminate, see your device-specific data manual

2.9.2.2.3 Memory Protection Lock Register 2 (L1PMPLK2)

The memory protection lock register 2 (L1PMPLK2) is shown in Figure 2-10.

Figure 2-10. Memory Protection Lock Register 2 (L1PMPLK2)



LEGEND: W = Write only; -x, value is indeterminate, see your device-specific data manual

2.9.2.2.4 Memory Protection Lock Register 3 (L1PMPLK3)

The memory protection lock register 3 (L1PMPLK3) is shown in Figure 2-11.

Figure 2-11. Memory Protection Lock Register 3 (L1PMPLK3)

31	0
Lock Bits 127:96	
W-x	

LEGEND: W = Write only; -x, value is indeterminate, see your device-specific data manual

2.9.2.2.5 Memory Protection Lock Command Register (L1PMPLKCMD)

The memory protection lock command register (L1PMPLKCMD) is shown in Figure 2-12 and described in Table 2-19.

Figure 2-12. Memory Protection Lock Command Register (L1PMPLKCMD)

31	16
Reserved	
R-0	

15	3	2	1	0
Reserved		KEYR	LOCK	UNLOCK
R-0		W-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = value after reset

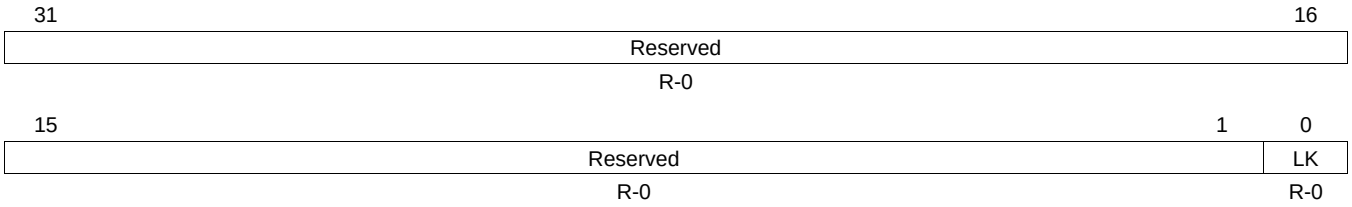
Table 2-19. Memory Protection Lock Command Register (L1PMPLKCMD) Field Descriptions

Bit	Field	Value	Description
31-3	Reserved	0	Reserved
2	KEYR		Reset status
		0	No effect.
		1	Reset status
1	LOCK		Interface to complete a lock sequence.
		0	No effect.
		1	Locks the lock provided that the software executed the sequence correctly.
0	UNLOCK		Interface to complete an unlock sequence.
		0	No effect.
		1	Unlocks the lock provided that the software executed the sequence correctly.

2.9.2.2.6 Memory Protection Lock Status Register (L1PMPLKSTAT)

The memory protection lock status register (L1PMPLKSTAT) is shown in Figure 2-13 and described in Table 2-20.

Figure 2-13. Memory Protection Lock Status Register (L1PMPLKSTAT)



LEGEND: R = Read only; -*n* = value after reset

Table 2-20. Memory Protection Lock Status Register (L1PMPLKSTAT) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	LK		Indicates the lock's current status.
		0	Lock is disengaged.
		1	Lock is engaged.

As illustrated above, the memory protection architecture allows for lock sizes up to 128 bits. The L1P implements only a 64-bit lock behind the lock interface. Thus, the values written to L1PMPLCK1, L1PMPLCK2, and L1PMPLCK3 are ignored. The behavior of the lock mechanism with respect to keys that are shorter than 128 bits is defined in Chapter 8.

2.9.2.3 Memory Protection Fault Registers

Table 2-21 lists the registers for the memory protection fault. See the device-specific data manual for the memory address of these registers.

In order to allow programs to diagnose a memory protection fault after an exception occurs, the L1P implements two registers dedicated to storing information about the fault.

Table 2-21. Memory Protection Fault Registers

Address	Acronym	Register Description	Section
0184 A400h	L1PMPFAR	Level 1 Program Memory Fault Address Register	Section 2.9.2.3.1
0184 A404h	L1PMPFSR	Level 1 Program Memory Fault Set Register	Section 2.9.2.3.2
0184 A408h	L1PMPFCLR	Level 1 Program Memory Fault Clear Register	Section 2.9.2.3.3

2.9.2.3.1 L1P Memory Protection Fault Address Register (L1PMPFAR)

The memory protection fault address register (L1PMPFAR) is shown in Figure 2-14 and described in Table 2-22.

Figure 2-14. L1P Memory Protection Fault Address Register (L1PMPFAR)

31	0
Fault Address	
R-x	

LEGEND: R = Read only; -x, value is indeterminate, see your device-specific data manual

Table 2-22. L1P Memory Protection Fault Address Register (L1PMPFAR) Field Descriptions

Bit	Field	Value	Description
31-0	Fault Address	0-FFFF FFFFh	Reserved

2.9.2.3.2 L1P Memory Protection Fault Set Register (L1PMPFSR)

The memory protection fault set register (L1PMPFSR) is shown in Figure 2-15 and described in Table 2-23.

Figure 2-15. L1P Memory Protection Fault Set Register (L1PMPFSR)

31	16
Reserved	
R-0	

15	9	8	7	6	5	4	3	2	1	0
FID		LOCAL	Reserved		SR	SW	Reserved	UR	UW	Reserved
R-0		R-0	R-0		R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 2-23. L1P Memory Protection Fault Set Register (L1PMPFSR) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-9	FID	0-7Fh	Bit 6:0 of faulting requestor. If ID is narrower than 7 bits, the remaining bits return 0. If ID is wider than 7 bits, the additional bits get truncated. FID =0 if LOCAL =1.
8	LOCAL	0	Normal operation.
		1	Access was a "LOCAL" access.
7-6	Reserved	0	Reserved
5	SR		Supervisor read access type.
		0	Normal operation.
		1	Indicates a Supervisor read request.
4	SW		Supervisor write access type.
		0	Normal operation.
		1	Indicates a Supervisor write request.
3	Reserved	0	Reserved
2	UR		User read access type.
		0	Normal operation.
		1	Indicates a User read request.
1	UW		User write access type.
		0	Normal operation.
		1	Indicates a User write request.
0	Reserved	0	Reserved

2.9.2.3.3 L1P Memory Protection Fault Clear Register (L1PMPFCLR)

The memory protection fault clear register (L1PMPFCLR) is shown in Figure 2-16 and described in Table 2-24.

Figure 2-16. L1P Memory Protection Fault Clear Register (L1PMPFCLR)

31	16
Reserved	
R-0	

15	1	0
Reserved		MPFCLR
R-0		W-0

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 2-24. L1P Memory Protection Fault Clear Register (L1PMPFCLR) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	MPFCLR		Command to clear the L1DMPFAR.
		0	No effect.
		1	Clear L1DMPFAR and L1DMPFCR.

The L1PMPFAR and L1PMPFSR registers only store enough information for one fault. Generally, the hardware records the information about the first fault and generates an exception only for that fault. L1P has a notion of “local” (CPU triggered) and “remote” (DMA/IDMA triggered) faults. The L1P allows a “local” fault to replace a “remote” fault and generate a new exception.

The fault information is held until the software clears it by writing a 1 to the MPFCLR bit in the L1PMPFCR register. Writing a 0 to the MPFCLR bit in the L1PMPFCR register has no effect.

2.9.2.3.4 Protection Checks on Accesses to Memory Protection Registers

L1P implements permission checks on the memory protection registers themselves. The rules are as follows:

- All requestors can *read* any memory protection (MP) register at any time in all circumstances, except L1PMPLK0 through L1PMPLK3.
- Supervisor can *write* the register.

Table 2-25 summarizes which L1P memory protection registers are accessible by role and what protection checks are performed in the megamodule.

Table 2-25. Permissions for L1P Memory Protection Registers

Register	Supervisor	User
L1PMPFAR	R	R
L1PMPFSR	R	R
L1PMPFCR	W	/
L1PMPLK0	W	/
L1PMPLK1	W	/
L1PMPLK2	W	/
L1PMPLK3	W	/
L1PMPLKCMD	W	/
L1PMPLKSTAT	R	R
L1PMPPAxx	R/W	R

Level 1 Data Memory and Cache

Topic **Page**

3.1 Introduction 50
3.2 L1D Memory Architecture 50
3.3 L1D Cache 51
3.4 L1D Cache Control Registers 59
3.5 L1D Memory Performance 66
3.6 L1D Power-Down Support 69
3.7 L1D Memory Protection 70

3.1 Introduction

3.1.1 Purpose of the Level 1 Data (L1D) Memory and Cache

The purpose of the L1D memory and cache is to maximize performance of the data processing. The configurability of the L1D memory and cache offers the flexibility to use L1D cache or L1D memory in a system.

3.1.2 Features

The L1D memory and cache provide the following features:

- Configurable L1D cache size: 0K, 4K, 8K, 16K, 32K.
- Memory protection
- Cache block and global coherence operations

3.1.3 Terms and Definitions

Refer to Appendix A and Appendix B of this document for detailed definitions of the terms used in this chapter. Appendix A describes general terms used throughout this reference guide and Appendix B defines terms related to the memory and cache architecture.

3.2 L1D Memory Architecture

3.2.1 L1D Memory

The L1D memory supports up to 1MB of memory-mapped RAM and ROM. L1D memory may not be cached within L1D cache, L1P cache, or L2 cache within the same megamodule.

The L1D memory's base address is constrained to 1MB boundaries. The total size of L1D memory must be a multiple of 16K bytes.

L1D memory is not cached within L1D cache, L1P cache, or L2 cache within the same megamodule.

3.2.1.1 L1D Regions

L1D memory is divided into two regions, denoted L1D region 0 and L1D region 1. The regions have the following different features:

- Each region has separate memory protection entries.
- Part of L1D region 1 can be converted into data cache.

The two regions appear consecutively in memory. Region 0 may be 0K bytes (thus disabled), or any power-of-2 size in the range 16K to 512K. Region 1 starts at the end of region 0. Its size may be any multiple of 16K from 16K to 512K bytes. When region 0 is enabled, the size of region 1 must be less than or equal to the size of region 0, when region 0 is enabled.

The two L1D regions divide the memory protection entries into two sets. There are 32 data memory protection pages as described in Section 3.7.2. The first 16 pages cover region 0, and the second 16 pages cover region 1. When region 0 is 0K bytes, its memory protection pages go unused.

The actual memory configuration is device-specific. Refer to the device-specific data manual for more information.

3.3 L1D Cache

The C64x+ L1D memory and cache architecture allow converting part or all of L1D region 1 into a read-allocate, writeback, two-way set-associative cache. The cache is necessary to facilitate reading and writing data at the full CPU clock rate, while still having a large system memory. It is the cache's responsibility to hide much of the latency associated with reading from and writing to the slower system memory.

The cache controller design supports a range of cache sizes, from 4K through 32K. However, a given device may implement less than 32K of L1D RAM in region 1.

The L1D cache converts L1D memory to cache starting at the highest L1D memory address in L1D region 1 and working downwards.

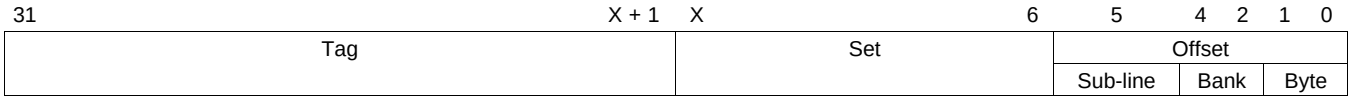
The L1D memory is initialized as either "All RAM" or "maximal cache" at reset. Refer to the device-specific data manual for specific behavior.

The operation of the L1D Cache is controlled through several registers. These registers are described in more detail in Section 3.4.

3.3.1 L1D Cache Architecture

L1D cache is a two-way set associative cache, meaning that every physical memory location in the system has two possible locations in the cache where it can reside. When the CPU attempts to access a piece of data, the L1D cache must check whether the requested address resides in either way of the L1D cache. To do so, the 32-bit address provided by the CPU is partitioned into six fields, as shown in Figure 3-1.

Figure 3-1. Data Access Address Organization



The offset of six bits accounts for the fact that an L1D line size is 64 bytes. The cache control logic ignores bits 0 through 5 of the address (the byte, bank, and sub-line fields). Bits 0 through 5 only determine which bank and which bytes within a bank to access; thus, they are irrelevant to the cache's tag compare logic. The set field indicates that the L1D cache line address where the data would reside, if it were cached. The width of the set field depends on the amount of L1D that you configure as cache, as defined in Table 3-1 below. Use the set field to look up and check the tags in each way for any already-cached data from that address as well as the valid bit, which indicates whether the address in the tag actually represents a valid address held in cache.

The tag field is the upper portion of the address that identifies the true physical location of the data element. The cache compares the tag to the stored tags for both ways of the L1D cache.

If one of the tags matches and you set the corresponding valid bit is set on reads, then it is a "hit," and the data cache returns data to the CPU directly from the L1D cache. Otherwise, the read is a "miss", and the CPU stalls while the request is sent on to the Level 2 (L2) memory to fetch the data from its location elsewhere in the system.

The CPU can also write data to L1D. When the CPU performs a store, the L1D performs the same tag comparison as it does for reads. If a valid matching tag is found, then the write is a "hit", and the data is written directly into the L1D cache location. Otherwise, the write is a "miss" and the data is queued in the L1D write buffer. This buffer is used to prevent CPU stalls on write misses. Since the CPU does not wait for data to return on writes, there is no reason to stall during the L2 access.

The L1D cache configuration determines the size of the set and tag fields, as shown in Table 3-1.

Table 3-1. Data Access Address Set Field Width

L1DMODE Setting in the L1DCFG Register	Amount of L1D Cache	'X' Bit Position	Description
000b	0K	N/A	L1D is all RAM
001b	4K	10	32 L1D cache lines
010b	8K	11	64 L1D cache lines
011b	16K	12	128 L1D cache lines
100b	32K	13	256 L1D cache lines
101b	Reserved. Maps to 32K		
110b			
111b	"Maximal Cache." Maps to 32K		

Another characteristic of the data cache is the ability to evict data from the L1D cache to L2. Since the CPU is able to modify the contents of the L1D cache, it must be capable of updating the data in its true physical location. This occurs when a new L1D line replaces one that has been modified, or when the CPU tells the L1D cache to write back modified data through software control.

3.3.2 Replacement and Allocation Strategy

The L1D cache operates with a fixed two-way set associativity in all cache configurations. This means that each location in system memory can reside in either of two possible locations in the L1D cache.

The L1D cache is a read-allocate-only cache. This means that the L1D cache will fetch a complete line of 64 bytes only on a read miss. Write misses are sent directly to L2 through the L1D write buffer. The replacement strategy calls for the least-recently-used (LRU) L1D line to be replaced with the new line. This keeps the most recently-accessed data in the L1D cache at all times.

L1D is writeback cache. Write hits are processed directly within L1D. The update is not passed to L2 or the rest of the memory system immediately. When a cache line is modified, that line's associated "dirty bit" is set to 1. L1D writes back only dirty lines when evicting them to make room for newly-cached data, when the program initiates a manual coherence operation to force its writeback, or when the CPU initiates a long-distance read to a non-cacheable memory having a set match.

3.3.3 L1D Mode Change Operations

You can configure the size of L1D cache at run time. Programs select the size of L1D cache by writing the requested mode to the L1DMODE field in the L1DCFG register.

The L1DMODE field in the L1DCFG register selects the L1D cache mode according to Table 3-2.

Table 3-2. Cache Size Specified by the L1DMODE in the L1DCFG

L1DMODE Setting in L1DCFG Register	Amount of L1D Cache
000b	0K
001b	4K
010b	8K
011b	16K
100b	32K
101b	Reserved. Maps to 32K
110b	
111b	"Maximal Cache." Maps to 32K

The actual range of L1D cache modes is constrained by the size of L1D region 1.

In general, a larger value of L1DMODE specifies a larger cache size, up to the size of the implemented L1D memory. The maximum L1D cache size is the smaller of "largest power-of-2 that fits in L1D region 1 RAM size" and 32K.

For example, when L1D region 1 is only 16K in size, the L1D cache can be no larger than 16K. In this case, the encoding 011b through 111b maps to 16K. On these devices, L1DMODE settings 100b through 111b will select the 16K cache mode as opposed to the 32K cache mode. That is, modes 000b through 011b always select the requested size, 0K through 16K. Modes 100b through 111b selects the maximum size implied by the size of L1D memory: 16K or 32K.

As a result of this policy, programs wanting no more than a certain amount of cache should program the value corresponding to this upper bound. Programs desiring "as much cache as possible" should program 111b into L1DMODE.

When programs initiate a cache mode change, the L1D cache itself writes back and invalidates its current contents without loss of data.

A writeback-invalidate is necessary to ensure correct cache behavior and to ensure no cached data is lost; however, it is not sufficient to prevent data loss in addressable L1D memory locations becoming cache. To safely change L1D cache modes, applications must adhere to the procedure in Table 3-3.

Table 3-3. Switching L1D Modes

To switch from. . .	To. . .	The program must perform the following steps. . .
A mode with *no* or *some* L1D cache	A mode with *more* L1D cache	1. DMA, IDMA, or copy any needed data out of the affected range of L1D RAM. 2. Write the desired cache mode to the L1DMODE field in the L1DCFG register. 3. Read back L1DCFG. This stalls the CPU until the mode change completes.
A mode with *some* L1D cache	A mode with *less* or *no* L1D cache	1. Write the desired cache mode to the L1DMODE field in the L1DCFG register. 2. Read back L1DCFG. This stalls the CPU until the mode change completes.

3.3.4 L1D Freeze Mode

The L1D cache directly supports a freeze mode of operation for applications. This mode allows real-time applications to limit the amount of data evicted from L1D during various sections of code, such as interrupt handlers. L1D freeze mode only affects L1D cache. L1D RAM is not affected by freeze mode.

The L1D cache services read hits and write hits normally while in freeze mode, with the small exception that the LRU bit is not modified. Read hits return data from the cache. Write hits update the cached data for the cache line and mark it dirty, as necessary. The LRU bit is not updated. The LRU bit is the bit which indicates the least recently used way for the affected cache line). In freeze mode, the L1D cache does not allocate new cache lines on read misses, nor will it evict any existing cache contents in freeze mode. Write misses in the L1D write buffer are queued normally.

In freeze mode, the L1D cache still responds normally to cache-coherence commands issued from L2 (snoop-read, snoop-write), as well as any program-initiated cache controls (writeback, invalidate, writeback-invalidate, and mode change). L1D's freeze mode has no impact on whether L2 allocates cache lines. Likewise, L2's freeze mode has no impact on whether L1D allocates cache lines.

The OPER field in the L1DCC register controls the L1D freeze mode. The CPU places L1D into freeze mode by writing 1 to the OPER field. The CPU returns L1D to normal operation by writing 0 to the OPER field in the L1DCC register.

The POPER field in the L1DCC register holds the previous value of the OPER field. The value of OPER in the L1DCC register is copied to the POPER field in the L1DCC register on writes to the L1DCC register. This alleviates the cycle cost of reading the L1DCC register (in order to save OPER's previous value) before it is written. If the POPER field is not in the L1DCC register, the program must read, write, and then read again to fully freeze the cache while recording its previous operating mode. If the POPER field is in the L1DCC register, this operation only requires a single write followed by a read.

The following operations occur when you perform a write to the L1DCC register:

1. The content of the OPER field copies to the POPER field in the L1DCC register.
2. The POPER field loses its previous value.
3. The OPER field updates according to the value that the CPU writes to bit 0 of the L1DCC register. Thus, the write to the L1DCC register only modifies the OPER field in the L1DCC register.

In order to ensure that the L1PCC register updates, the software must perform a write to the L1PCC register followed by a read of the L1PCC register. This guarantees that the requested mode is in effect.

Programs cannot directly modify the POPER field with a single write.

The goal of the OPER and the POPER fields in the L1DCC register is to avoid the CPU cycle penalty and code size involved in a read-write-reread sequence that would otherwise be necessary. Thus, applications may quickly freeze L1D and record the previous "freeze" state of L1D with the short sequence of code in Example 3-1.

Example 3-1. L1D Quick Freeze Example Code Sequence

```
   MVKL L1DCC, A0    ; Point to L1DCC
   MVKH L1DCC, A0    ;
|| MVK 1, B0         ; OPER encoding for 'freeze'
   STW B0, *A0[0]    ; Write 1 to L1DCC.OPER
   LDW *A0[0], A1    ; Read L1DCC to get L1DCC.POPER
   NOP 4
  ; At this point, L1D is frozen, and the CPU has the old OPER value ; in bit 16 of A1.
```

You can use the L1DCC register for unfreezing the cache in a manner similar to how it was frozen. Example 3-2 illustrates.

Example 3-2. L1DCC.OPER Restore Example Code Sequence

```
   MVKL L1DCC, A0    ; Point to L1DCC
   MVKH L1DCC, A0    ;
|| SHRU A1, 16, A1   ; Shift POPER field into OPER's position
   STW A1, *A0[0]    ; Write to L1DCC, restoring old value of OPER
   LDW *A0[0], A1    ; Read back L1DCC to ensure change is complete
   NOP 4
  ; At this point, L1D is in its previous state (frozen or unfrozen).
```

NOTE: Both L1D and L1P offer freeze modes via this sort of mechanism. (Refer to Chapter 2 for more information on implementing L1P). It is often desirable to freeze both caches together. Therefore, consecutive writes to the L1DCC register and to the L1PCC register followed by reading both the L1DCC register and the L1PCC register is sufficient to ensure that both L1P and L1D are frozen. Example 3-3 illustrates a sequence that freezes both L1P and L1D.

Example 3-3. Example Code Sequence for Freezing L1P and L1D Simultaneously

```
   MVKL L1DCC, A0    ; \
|| MVKL L1PCC, B0    ;  |__ Generate L1DCC pointer in A0
   MVKH L1DCC, A0    ;  |   and L1PCC pointer in B0
|| MVKH L1PCC, B0    ; /
|| MVK 1, A1         ; \___ OPER encoding for 'freeze'
|| MVK 1, B1         ; /    in both A1 and B1.
   STW A1, *A0       ; Write to L1DCC.OPER
|| STW B1, *B0       ; Write to L1PCC.OPER
   LDW *A0, A1       ; Get old freeze state into A1 from L1DCC
|| LDW *B0, B1       ; Get old freeze state into B1 from L1PCC
   NOP 4
  ; At this point, L1P and L1D are frozen.
  ; The old value of L1DCC.OPER is in bit 16 of A1.
  ; The old value of L1PCC.OPER is in bit 16 of B1.
```

3.3.5 Program-Initiated Cache Coherence Operations

The C64x+ L1D cache supports program-initiated cache coherence operations. These operations operate either on a block of addresses, or on the entire L1D cache.

The following cache coherence operations are supported:

- Invalidation: Valid cache lines are made invalid. Content of the affected cache lines is discarded.
- Writeback: The content of all dirty cache lines is written to a lower-level memory.
- Writeback-invalidation: Writeback operation followed by invalidation. Only the content of the dirty cache lines is written to lower-level memory, but all of the lines are invalidated.

3.3.5.1 Global Coherence Operations

Global cache operations execute on the entire L1D cache. The global coherence operations supported are invalidation, writeback, and writeback-invalidation.

In order to initiate a global invalidation operation, the program must write a 1 to the I bit of the L1DINV register.

Upon completion of the global invalidate operation, the I bit of the L1DINV register resets to 0. The program can poll this bit to detect the completion of the operation.

In order to initiate a global writeback operation, the program needs to write a 1 to the C bit of the L1DWB register.

The C bit of the L1DWB register resets to 0 upon completion. The program can poll this bit to detect the completion of the operation.

The writeback-invalidation operation is controlled in a similar way. In order to initiate a global writeback-invalidation operation, the program must write a 1 to the C bit of the L1DWBINV register.

Table 3-4 provides a summary of the L1D global cache coherence operations.

Table 3-4. Global Coherence Operations

Cache Operation	Register Used	L1D Effect
L1D Writeback	L1DWB	All updated data written back to L2 / external, but left valid on L1D.
L1D Writeback with Invalidate	L1DWBINV	All updated data written back to L2 / external. All lines invalidated within L1D.
L1D Invalidate	L1DINV	All lines invalidated in L1D. Updated data is dropped.

CAUTION

The L1D global-invalidate causes all updated data in L1D to be discarded, rather than written back to the lower levels of memory. This can cause incorrect operation in programs that expect the updates to be written to the lower levels of memory. Therefore, most programs use either the L1D block writeback-invalidate (described in Section 3.3.5.2) or the global L2 operations, rather than the L1D global-invalidate.

You can also globally invalidate the L1D cache by setting the ID bit in the L2CFG register to 1 for legacy reasons. The ID field is provided for backward compatibility with C64x and C64x+ devices, but it should not be used in new applications. New applications should use the L1DINV register.

3.3.5.2 Block Coherence Operations

Block coherence operations have similar functionality as the global coherence operations; however, they apply only to a defined block of data. This block is defined by the base address and by the word (32-bit) size in the associated memory-mapped registers.

The block coherence operations are designed to be as efficient as possible, while minimizing impact to tasks running concurrently on the CPU. Block coherence operations are usually running in the "background" of CPU activity.

The block coherence operations supported are invalidation, writeback, and writeback-invalidation. Each operation has two registers associated with it. The L1DXXBAR register defines the base address of the block and the L1DXXWC register defines the word size of the block

Writing a non-zero value to the word count field in the L1DXXWC register initiates a block coherence operation.

The word count field decrements upon completion of the block coherence during the operation, the operation sets to 0. The program can poll this bit to detect the completion of the operation.

Table 3-5 provides a summary of the L1D block cache coherence operations.

Table 3-5. Block Cache Operations

Cache Operation	Register Used	L1D Effect
L1D Writeback	L1DWBAR L1DWWC	Updated data written back to L2 / external, but left valid in L1D.
L1D Writeback with Invalidate	L1DWIBAR L1DWIWC	Updated data written back to L2 / external. All lines in range invalidated within L1D.
L1D Invalidate	L1DIBAR L1DIWC	All lines in range invalidated in L1D. Updated data is dropped.

NOTE: Reads or writes to the addresses within the block being operated on while a block cache operation is in progress may cause those addresses to not be written back or invalidated as requested. To avoid this, programs should not access addresses within the range of cache lines affected by a block cache operation while the operation is in progress. Programs may poll the appropriate word count field to determine when the block operation is complete.

Two simultaneous accesses to the same bank incur a one-cycle stall penalty, except under the following special cases:

- The memory accesses are both writes to non-overlapping bytes within the same word. Therefore, bits 31-2 of the address are the same.
- The memory accesses are both reads that access all or part of the same word. Thus, bits 31-2 of the address are the same. In this case, the two accesses may overlap.
- One or both of the memory accesses is a write that misses L1D and is serviced by the write buffer instead. (See section Section 3.5.3 for information on the write buffer).
- The memory accesses form a single nonaligned access. Nonaligned accesses do not cause bank-conflict stalls, even though the memory system may subdivide them into multiple accesses.

Notice that a read access and a write access in parallel to the same bank always causes a stall. Two reads or two writes to the same bank may not stall as long as the above conditions are met.

Simultaneous CPU and DMA/IDMA accesses to distinct L1D memory banks do not stall. Accesses to the same bank result in a conflict between CPU and DMA/IDMA. One or the other stall based on the rules described in Chapter 6.

3.3.6 Cache Coherence Protocol

The C64x+ L1D cache remains coherent with respect to DMA activity in L2 RAM. To support this paradigm, the L1D cache accepts cache coherence commands arriving from L2.

3.3.6.1 L2 to L1D Cache Coherence Protocol

To support L1D cache coherence with respect to DMA/IDMA traffic in L2 RAM, the L1D controller supports two cache coherence commands arriving from L2: snoop-read (SNPR) and snoop-write (SNPW). The L2 only sends these snoop commands, when necessary, in response to DMA and IDMA activity in L2 RAM.

Snoop-read is sent to L1D when L2 detects that the L1D cache holds the requested line, and that the line is dirty. L1D responds by returning the requested data.

Snoop-write is sent to L1D when L2 detects that the L1D holds the requested line. It does not matter if the line is modified within L1D. The L1D updates its contents accordingly.

3.3.6.2 L1D to L2 Cache Coherence Protocol

In order to reduce excessive snoop traffic to L1D, L2 filters the snoops so that unnecessary snoops are not sent to L1D.

L2 keeps a shadow copy of L1D's tag memory. L2 consults its local copy of the L1D tags to decide whether a snoop command to L1D is warranted.

L2 primarily updates its shadow tags in response to L1D read miss requests, and secondarily in response to L1D victim writebacks. When L1D issues a read request, it also indicates whether or not the line is allocated within L1D; and if so, what way within the set the line is allocated in. L2 can update the corresponding set in its shadow tags from this information.

In addition to tracking which addresses are present in L1D cache, L2 tracks also tracks whether or not those lines are dirty in the C64x+ DSP.

3.4 L1D Cache Control Registers

3.4.1 Memory Mapped L1D Cache Control Register Overview

The C64x+ memory system provides a set of registers to govern the operation of L1D cache. These registers allow for changing cache modes and manually initiating cache coherence operations.

Table 3-6 below lists the L1D cache control registers.

Table 3-6. L1D Specific Cache Control Operations

Type of Operation	Register Name	Address	Action	Section
Mode Select	L1DCFG	0184 0040h	Configures the size of L1D cache.	Section 3.4.3.1
	L1DCC	0184 0044h	Controls L1D operating mode (freeze/normal).	Section 3.4.3.2
Block Cache Operation	L1DWIBAR	0184 4030h	Specified range is written back and invalidated within L1D.	Section 3.4.4.2
	L1DWIWC	0184 4034h		
	L1DWBAR	0184 4040h	Specified range is written back from L1D and left valid.	
	L1DWWC	0184 4044h		
	L1DIBAR	0184 4048h	Specified range is invalidated in L1D without being written back.	
	L1DIWC	0184 404Ch		
Global Cache Operation	L1DWB	0184 5040h	Entire contents of L1D is written back, but left valid.	Section 3.4.4.1
	L1DWBINV	0184 5044h	Entire contents of L1D is written back and invalidated.	
	L1DINV	0184 5048h or ID bit in CCFG	Entire contents of L1D is invalidated without being written back.	

In addition to the L1D-specific registers listed above, the L1D cache is directly affected by writes to L2-specific controls as well. Refer to Chapter 4 for the complete list of cache control operations and their effect on the L1D cache.

3.4.2 CPU L1D Cache Control Registers

The CPU has a single internal control register that dedicates a field to cache control operations, CSR. The DCC field in the CSR control register controlled the operation of L1D in various ways in previous devices.

The CSR no longer controls the operation of L1D in the C64x+ DSP. L1D ignores the value held in the DCC field. The former action of the DCC field is now part of the L1DCC and L1DCFG registers, as described in Section 3.3.4.

3.4.3 L1D Cache Configuration Registers

The L1DCFG and L1DCC registers control the operation of L1D.

3.4.3.1 L1D Cache Configuration (L1DCFG) Register

The L1D cache configuration register (L1DCFG) controls the size of L1D cache and is shown in Figure 3-2 and described in Table 3-7.

Figure 3-2. L1D Cache Configuration Register (L1DCFG)

31	16
Reserved	
R-0	

15	3	2	0
Reserved		L1DMODE	
R-0		R/W-0h or 7h	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 3-7. L1D Cache Configuration Register (L1DCFG) Field Descriptions

Bit	Field	Value	Description
31-3	Reserved	0	Reserved
2-0	L1DMODE	0-7h	Defines the size of the L1D cache. The L1DMODE field powers-up as either 0h or 7h. Refer to the device-specific data manual for further information.
		0h	L1D cache disabled.
		1h	4K
		2h	8K
		3h	16K
		4h	32K
		5h	Maximal cache size.
		6h	Maximal cache size.
		7h	Maximal cache size.

3.4.3.2 L1D Cache Control (L1DCC) Register

The L1D cache control register (L1DCC) controls whether L1D is frozen or unfrozen and is shown in Figure 3-3 and described in Table 3-8.

Figure 3-3. L1D Cache Control Register (L1DCC)

31	17	16
Reserved		POPER
R-0		R-0

15	1	0
Reserved		OPER
R-0		R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 3-8. L1D Cache Control Register (L1DCC) Field Descriptions

Bit	Field	Value	Description
31-17	Reserved	0	Reserved
16	POPER	0-1	Holds the previous value of the OPER field.
15-1	Reserved	0	Reserved
0	OPER		Controls the L1D freeze mode.
		0	Freeze mode disabled.
		1	Freeze mode enabled.

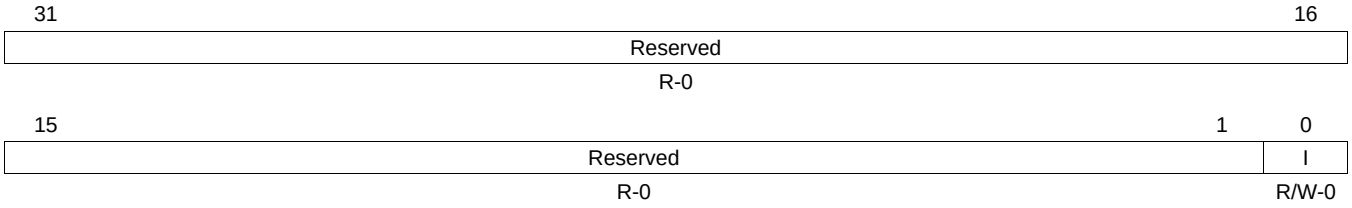
3.4.4 L1D Cache Coherence Operation Registers

3.4.4.1 Global Coherence Operation Registers

3.4.4.1.1 L1D Invalidate Register (L1DINV)

The L1D invalidate register (L1DINV) controls the global invalidation of the L1D cache and is shown in Figure 3-4 and described in Table 3-9.

Figure 3-4. L1D Invalidate Register (L1DINV)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 3-9. L1D Invalidate Register (L1DINV) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved.
0	I		Controls the global invalidation of L1D cache.
		0	Normal operation.
		1	All L1D cache lines are invalidated.

3.4.4.1.2 L1D Writeback Register (L1DWB)

The L1D writeback register (L1DWB) is shown in Figure 3-5 and described in Table 3-10.

Figure 3-5. L1P Writeback Register (L1DWB)

31	16
Reserved	
R-0	

15	1	0
Reserved		C
R-0		R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 3-10. L1D Writeback Register (L1DWB) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	C		Controls the global writeback operation of L1D cache.
		0	Normal L1D operation.
		1	Dirty L1D lines are written back.

3.4.4.1.3 L1D Writeback-Invalidate Register (L1DWBINV)

The L1D writeback-invalidate register (L1DWBINV) controls the writeback-invalidate operation of L1D cache and is shown in Figure 3-6 and described in Table 3-11.

Figure 3-6. L1D Writeback-Invalidate Register (L1DWBINV)

31	16
Reserved	
R-0	

15	1	0
Reserved		C
R-0		R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 3-11. L1D Writeback-Invalidate Register (L1DWBINV) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	C		Controls the global writeback-invalidate operation of L1D cache.
		0	Normal L1D operation.
		1	Dirty L1D lines written back, all L1D lines are invalidated.

3.4.4.2 Block Coherence Operation Registers

3.4.4.2.1 L1D Invalidate Base Address Register (L1DIBAR)

The L1D invalidate base address register (L1DIBAR) defines the base address of the block that will be invalidated and is shown in Figure 3-7 and described in Table 3-12.

Figure 3-7. L1D Invalidate Base Address Register (L1DIBAR)

31	0
L1D Invalidate Base Address (L1DIBAR)	
W-x	

LEGEND: W = Write only; *-x* = value is indeterminate, see your device-specific data manual

Table 3-12. L1D Invalidate Base Address Register (L1DIBAR) Field Descriptions

Bit	Field	Value	Description
31-0	L1DIBAR	0-FFFF FFFFh	Defines the base address for the L1D block invalidate operation.

3.4.4.2.2 L1D Invalidate Word Count Register (L1DIWC)

The L1D invalidate word count register (L1DIWC) defines the size of the block that will be invalidated. The size is defined in 32-bit words and is shown in Figure 3-8 and described in Table 3-13.

Figure 3-8. L1D Invalidate Word Count Register (L1DIWC)

31	16
Reserved	
R-0	

15	0
L1D Invalidate Word Count (L1DIWC)	
R/W-0	

LEGEND: R/W = Read/Write; R = Read only; *-n* = value after reset

Table 3-13. L1D Invalidate Word Count Register (L1DIWC) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	L1DIWC	0-FFFFh	Word count for block invalidation

3.4.4.2.3 L1D Writeback Base Address Register (L1DWBAR)

The L1D writeback base address register (L1DWBAR) defines the base address of the block that will be written back and is shown in Figure 3-9 and described in Table 3-14.

Figure 3-9. L1D Writeback Base Address Register (L1DWBAR)

31	0
L1D Writeback Base Address (L1DWBAR)	
W-x	

LEGEND: W = Write only; *-x* = value is indeterminate, see your device-specific data manual.

Table 3-14. L1D Writeback Base Address Register (L1DWBAR) Field Descriptions

Bit	Field	Value	Description
31-0	L1DWBAR	0-FFFF FFFFh	Defines the base address for the L1D block writeback operation

3.4.4.2.4 L1D Writeback-Invalidate Word Count Register (L1DWIWC)

The L1D writeback-invalidate word count register (L1DWIWC) defines the size of the block that will be invalidated. The size is defined in 32-bit words and is shown in Figure 3-10 and described in Table 3-15.

Figure 3-10. L1D Writeback-Invalidate Word Count Register (L1DWIWC)

31	16
Reserved	
R-0	

15	0
L1D Writeback-Invalidate Word Count (L1DWIWC)	
R/W-0	

LEGEND: R/W = Read/Write; R = Read only; *-n* = value after reset

Table 3-15. L1D Writeback-Invalidate Word Count Register (L1DWIWC) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	L1DWIWC	0-FFFFh	Word count for block invalidation

3.4.5 Privilege and Cache Control Operations

The impact of privilege on cache control operations can be summarized as follows:

- Supervisor code may change L1D cache size.
- User-mode code may not change L1D cache size.
- Only supervisor code may issue global invalidates to L1D.
- Both supervisor and user modes may freeze or unfreeze L1D at any time.

Table 3-16 summarizes which L1D cache control registers are accessible and what protection checks are performed in the megamodule according to role.

Table 3-16. Permissions for L1D Cache Control Registers

Register	Supervisor	User
L1DCFG	R/W	R
L1DCC	R/W	R/W
L1DWIBAR	W	W
L1DWIWC	R/W	R/W
L1DWBAR	W	W
L1DWWC	R/W	R/W
L1DIBAR	W	W
L1DIWC	R/W	R/W
L1DWB	R/W	R/W
L1DWBINV	R/W	R/W
L1DINV	R/W	R

3.5 L1D Memory Performance

The performance of the L1D memory depends on several factors. This section describes the impact of the banking architecture, the write buffer, and the miss pipelining on the performance of the L1D memory.

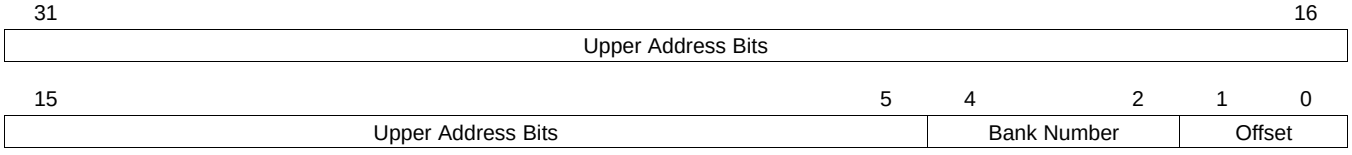
3.5.1 L1D Memory Banking

The L1D has a least-significant bit (LSB) based memory banking structure that divides memory into eight 32-bit-wide banks. These banks are single-ported, allowing only one access per cycle. L1D RAM and L1D cache both share the same bank structure.

The banks are interleaved based on the low-order bits of the address. Specifically, for aligned memory accesses, address bits [4:2] determine the bank number. The mapping of bits to bank number varies with the device endian mode.

In Figure 3-11, bits 4-2 of the address select the bank and bits 1-0 select the byte within the bank.

Figure 3-11. Address to Bank Number Mapping



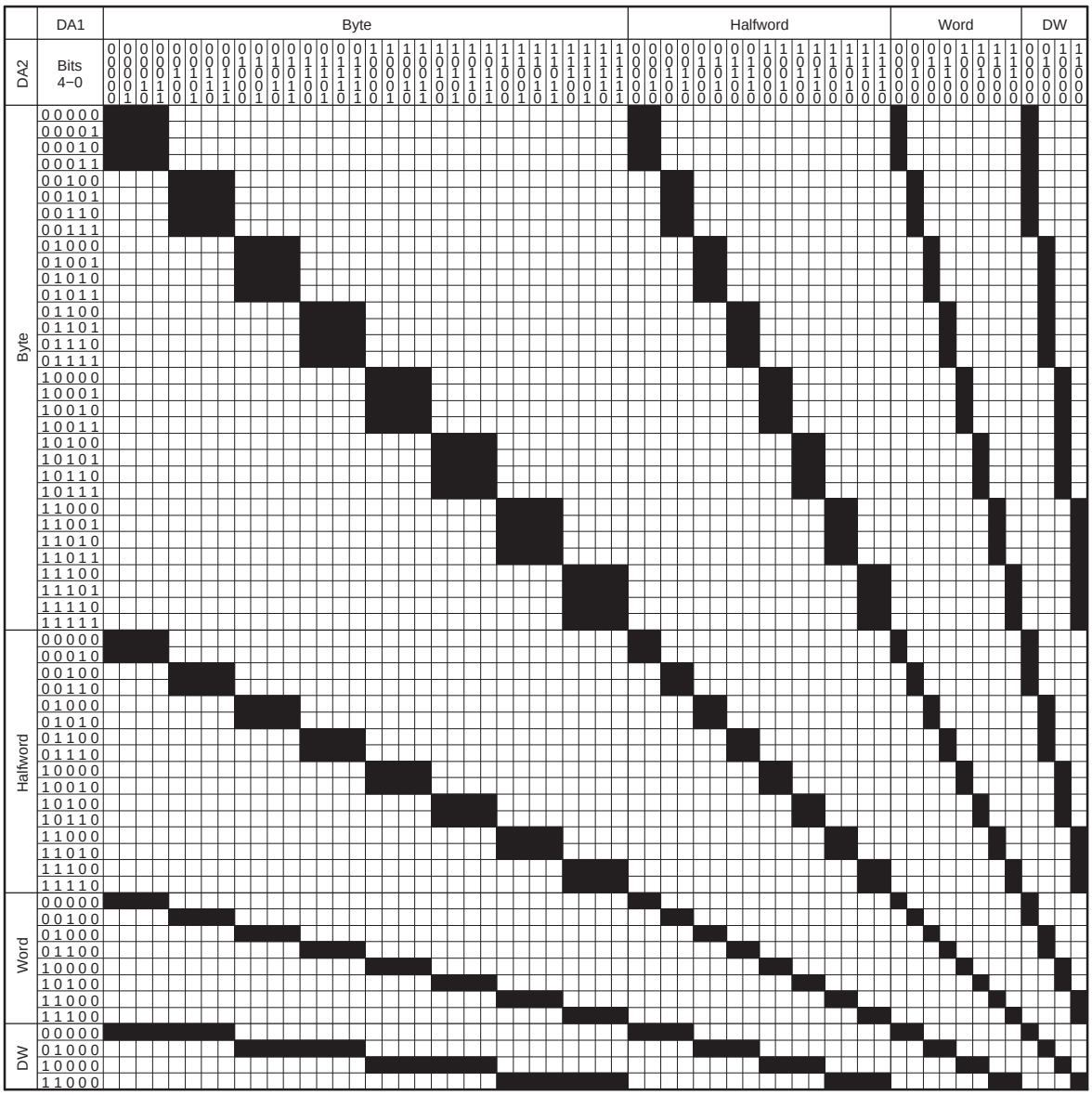
The shaded areas in Figure 3-12 show combinations of parallel accesses that may result in bank-conflict stalls according to the LSBs of addresses for the two accesses. Two simultaneous accesses to the same bank incur a one-cycle stall penalty, except under the following special cases:

- The memory accesses are both writes to non-overlapping bytes within the same word. Thus, bits 2 through 31 of the address are the same.
- The memory accesses are both reads that access all or part of the same word. Thus, bits 2 through 31 of the address are the same. In this case, the two accesses may overlap.
- The memory accesses form a single nonaligned access. Nonaligned accesses do not cause bank-conflict stalls, even though the memory system may subdivide them into multiple accesses.

Notice that a read access and a write access in parallel to the same bank always causes a stall. Two reads or two writes to the same bank may not stall as long as the above conditions are met.

Simultaneous CPU and DMA/IDMA accesses to distinct L1D memory banks do not stall. Accesses to the same bank results in a conflict between CPU and DMA/IDMA. One or the other stalls based on the rules described in Chapter 6.

Figure 3-12. Potentially Conflicting Memory Accesses



3.5.2 L1D Miss Penalty

The L1D can service up to two data accesses from the CPU every cycle. Accesses that hit L1D complete without stalls, unless a bank conflict occurs as described in Section 3.3.6.1.

Reads that miss L1D stall the CPU while the requested data is fetched. The L1D is a read-allocate cache, and so it will allocate a new line for the requested data.

An L1D read miss that also misses L2 stalls the CPU while the L2 retrieves the data from external memory. Once the data is retrieved, it is stored in L2 and transferred to the L1D. The external miss penalty varies depending on the type and width of external memory used to hold external data, as well as other aspects of system loading.

If there are two read misses to the same line in the same cycle, only one miss penalty is incurred. Similarly, if there are two accesses in succession to the same line and the first one is a miss, the second access does not incur any additional miss penalty.

The process of allocating a line in L1D can result in a victim writeback. Victim writebacks move updated data out of L1D to the lower levels of memory. When updated data is evicted from L1D, the cache moves the data to the victim buffer. Once the data is moved to the victim buffer, the L1D resumes processing of the current read miss. Further processing of the victim writeback occurs in the background. Subsequent read and write misses, however, must wait for the victim writeback to process. If the read misses do not conflict with existing victims, the read misses are pipelined with the victim writebacks in order to reduce the performance penalty.

The L1D pipelines read misses. Consecutive read misses to different lines may overlap, reducing the overall stall penalty.

Write misses do not stall the CPU directly. Write misses are queued in the write buffer that is between L1D and L2. Although the CPU does not always stall for write misses, the write buffer can stall the CPU under various circumstances. Section 3.5.3 describes the effects of the write buffer.

3.5.3 L1D Write Buffer

The L1D does not write allocate. Rather, write misses are passed directly to L2 without allocating a line in L1D. A 128-bit wide by 4-entry write buffer exists between the L1D cache and the L2 memory to capture these write misses. The write buffer provides a 128-bit path for writes from L1D to L2 with room for four outstanding write requests.

Writes that miss L1D do not stall the CPU unless the write buffer is full. If the write buffer is full, a write miss can indirectly stall the CPU until there is room in the buffer for the write. The write buffer can also stall the CPU by extending the time for a read miss. Reads that miss L1D are not processed as long as the write buffer is not empty. Once the write buffer empties, the read miss processes. This is necessary as a read miss may overlap an address for which a write is pending in the write buffer.

The L2 can process a new request from the write buffer every L2 cycle (L2 cycle = 2 × CPU cycles), provided that the requested L2 bank is not busy. You can merge multiple elements within a buffer together for a single memory access if they are contiguous in memory to reduce the potential for buffer stalls and DMA contention.

The write buffer allows write requests to merge and merges two write misses into a single transaction, provided that the write request obeys the following rules:

- The new write miss resides within the same 128-bit quad-word as the immediately preceding write miss.
- The two writes are to locations in L2 SRAM (not to locations that may be held in L2 cache).
- The first write has just been placed in the write buffer queue.
- The second write is presently being placed in the buffer queue.
- The first write has not yet been presented to the L2 controller.
- Both writes have the same privilege level.

The previous conditions occur in a number of situations, such as when a program makes a large series of sequential writes or when it makes a burst of small writes to a structure in memory. Write merging increases the effective capacity of the write buffer in these cases by reducing the number of independent stores that are present in the write buffer. This reduces the stall penalty for programs with a large number of write misses.

As a secondary benefit, write merging reduces the number of memory operations executed in L2. This improves the overall performance of the L2 memory by reducing the total number of individual write operations L2 must process. Adjacent accesses are combined into a single access to an L2 bank, rather than multiple accesses to that bank. This allows other requestors to access that bank more quickly, and it allows the CPU to move on to the next bank immediately in the next cycle.

3.5.4 L1D Miss Pipelining

The L1D cache pipelines read misses. Miss pipelining can hide much of the miss overhead by overlapping the processing of several cache misses.

Table 3-16 presents a summary of the L1D performance. Two different configurations are presented. The first configuration features 0 wait state for L2SRAM, 2 × 128 bit banks. This configuration is available in the DM644x devices. The second configuration features 1 wait state L2SRAM, 4 × 128 bit banks. This configuration is available in the C645x devices.

Table 3-17. L1D Performance Summary

L2 Type	0 wait state, 2x128 bit banks		1 wait state, 4x128 bit banks	
Parameter	**L2 SRAM**	**L2 Cache**	**L2 SRAM**	**L2 Cache**
Single Read Miss	10.5	12.5	12.5	14.5
2 Parallel Read Misses (pipelined)	10.5 + 4	12.5 + 8	12.5 + 4	14.5 + 8
M Consecutive Read Misses (pipelined)	10.5 + 3 × (M – 1)	12.5 + 7 × (M – 1)	12.5 + 3 × (M – 1)	14.5 + 7 × (M – 1)
M Consecutive Parallel Read Misses (pipelined)	10.5 + 4 × (M/2 – 1) + 3 × M/2	12.5 + 8 × (M/2 – 1) + 7 × M/2	12.5 + 4 × (M – 1)	14.5 + 8 × (M/2 – 1) + 7 × M/2

3.6 L1D Power-Down Support

You can set the L1D memory to powered-down mode when the CPU is in idle mode. The following software sequence is required to power-down the C64x+ megamodule:

1. Enable power-down by setting the MEGPD field in the PDCCMD register to 1.
2. Enable the CPU interrupt(s) that you want to wake-up the megamodule; disable all others.
3. Execute an IDLE instruction.

The megamodule stays in powered-down mode until the interrupt(s) that you enabled in step 2, above wake them up

If a DMA access occurs to the L1D, L1P, or L2 memory while the megamodule is powered-down, the power down controller (PDC) wakes up all three memory controllers. When the DMA access has been serviced, the PDC will power-down the memory controllers again.

Refer to Chapter 9 for more information about the PDCCMD register and the power-down capabilities of the C64x+ megamodule.

NOTE: Powering-down the megamodule as described here is often called static power-down. This term is used to describe this mode since it is often used for longer periods of time.

3.7 L1D Memory Protection

L1D memory supports memory protection to offer the robustness required in many systems. Several levels of memory protection are available. Not all the levels are available on all the devices. Refer to the device-specific data manual for more information. Familiarize yourself with Chapter 8 before you read this section.

3.7.1 Protection Checks on L1D Accesses

3.7.1.1 Protection Checks on CPU, IDMA and Other System Master Accesses

Protection checks are performed for all accesses that are serviced directly by the L1D on devices that include memory protection support. This includes accesses from the CPU, IDMA other system master accesses.

The L2 controller determines whether a given CPU request is allowed or disallowed based on the privilege level associated with the request and the permission settings on the address range that the request accesses. The exact rules for these checks are set forth in Chapter 8.

The L1D memory controllers feature two exception outputs that are routed to the C64x+ interrupt controller. One of these exception outputs indicates that a CPU-triggered (“local”) memory exception occurred. The other indicates that a system master-triggered (“remote”) exception occurred.

3.7.1.2 Additional Protection Checks on Program Initiated Cache Coherence Operations

Protection checks are performed on program initiated cache coherence operations to ensure the integrity of the memory protection. Both user and supervisor code may issue manual cache coherence operations.

However, user code cannot globally invalidate L1D cache or change the size of L1D cache. Only supervisor code may initiate a global invalidation or change the amount of memory allocated to cache.

3.7.2 L1D Memory Protection Registers

The following registers govern the operation of the L1D memory protection. They fall into three main categories:

- Memory Protection Page Attribute Registers (MPPA): These registers store the permissions associated with each protected page.
- Memory Protection Lock Registers (MPLK): Peripherals may choose to implement a hardware memory protection lock. When engaged, the lock disables all updates to the memory protection entries for that peripheral.
- Memory Protection Fault Registers (MPFxR): Each peripheral that generates memory protection faults provides the MPFAR, MPFSR, and MPFCR registers with recording the details of the fault.

Table 3-18 lists the registers for the memory protection lock. See the device-specific data manual for the memory address of these registers.

Table 3-18. Memory Protection Lock Registers

Address	Acronym	Register Description	Section
0184 AExxh	L1DMPPAxx	Memory Protection Page Attribute Register	Section 3.7.2.1
0184 AD00h	L1DMPLK0	Memory Protection Lock Register 0	Section 3.7.2.2.1
0184 AD04h	L1DMPLK1	Memory Protection Lock Register 1	Section 3.7.2.2.2
0184 AD08h	L1DMPLK2	Memory Protection Lock Register 2	Section 3.7.2.2.3
0184 AD0Ch	L1DMPLK3	Memory Protection Lock Register 3	Section 3.7.2.2.4
0184 AD10h	L1DMPLKCMD	Memory Protection Lock Command Register	Section 3.7.2.2.5
0184 AD14h	L1DMPLKSTAT	Memory Protection Lock Status Register	Section 3.7.2.2.6
0184 AC00h	L1DMPFAR	Memory Protection Fault Address Register	Section 3.7.2.3.1
0184 AC04h	L1DMPFSR	Memory Protection Fault Set Register	Section 3.7.2.3.2
0184 AC08h	L1DMPFCR	Memory Protection Fault Clear Register	Section 3.7.2.3.3

3.7.2.1 Memory Protection Attribute Registers

L1D implements 32 memory protection pages. L1DMPPA0 through L1DMPPA15 correspond to region 0. L1DMPPA 16 through L1DMPPA31 correspond to region 1.

Table 3-19. L1D Memory Protection Attribute Register Addresses

L1D Region 0		L1D Region 1	
Register	**Address**	**Register**	**Address**
L1DMPPA0	0184 AE00h	L1DMPPA16	0184 AE40h
L1DMPPA1	0184 AE04h	L1DMPPA17	0184 AE44h
L1DMPPA2	0184 AE08h	L1DMPPA18	0184 AE48h
L1DMPPA3	0184 AE0Ch	L1DMPPA19	0184 AE4Ch
L1DMPPA4	0184 AE10h	L1DMPPA20	0184 AE50h
L1DMPPA5	0184 AE14h	L1DMPPA21	0184 AE54h
L1DMPPA6	0184 AE18h	L1DMPPA22	0184 AE58h
L1DMPPA7	0184 AE1Ch	L1DMPPA23	0184 AE5Ch
L1DMPPA8	0184 AE20h	L1DMPPA24	0184 AE60h
L1DMPPA9	0184 AE24h	L1DMPPA25	0184 AE64h
L1DMPPA10	0184 AE28h	L1DMPPA26	0184 AE68h
L1DMPPA11	0184 AE2Ch	L1DMPPA27	0184 AE6Ch
L1DMPPA12	0184 AE30h	L1DMPPA28	0184 AE70h
L1DMPPA13	0184 AE34h	L1DMPPA29	0184 AE74h
L1DMPPA14	0184 AE38h	L1DMPPA30	0184 AE78h
L1DMPPA15	0184 AE3Ch	L1DMPPA31	0184 AE7Ch

3.7.2.1.1 Memory Protection Register (MPPA*xx*)

The size of each page differs from region to region and from device to device. Some pages cannot be used on a particular device. Program unused pages to a value of all zeroes for debug purposes.

Refer to the device-specific data manual to determine the page size and number of pages used on a particular device.

The memory protection (MPPAxx) register is shown in Figure 3-13 and described in Table 3-20.

Figure 3-13. Memory Protection Register (MPPA*xx*)

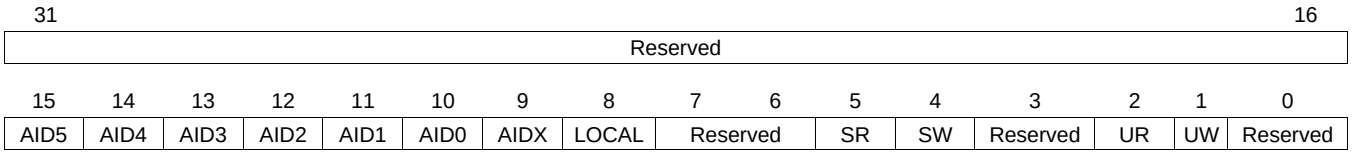


Table 3-20. Memory Protection Register (MPPA*xx*) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15	AID5		Controls access from ID = 5.
		0	Access denied.
		1	Access granted.
14	AID4		Controls access from ID = 4.
		0	Access denied.
		1	Access granted.
13	AID3		Controls access from ID = 3.
		0	Access denied.
		1	Access granted.
12	AID2		Controls access from ID = 2.
		0	Access denied.
		1	Access granted.
11	AID1		Controls access from ID = 1.
		0	Access denied.
		1	Access granted.
10	AID0		Controls access from ID = 0.
		0	Access denied.
		1	Access granted.
9	AIDX		Controls ID >=6.
		0	Access denied.
		1	Access granted.
8	LOCAL		Controls access from CPU to local memories (L1/L2).
		0	Access denied.
		1	Access granted.
7-6	Reserved	0	Reserved
5	SR		Supervisor read access type.
		0	Normal operation.
		1	Indicates a Supervisor read request.
4	SW		Supervisor write access type.
		0	Normal operation.
		1	Indicates a Supervisor write request.

Table 3-20. Memory Protection Register (MPPA*xx*) Field Descriptions (continued)

Bit	Field	Value	Description
3	Reserved	0	Reserved
2	UR		User read access type.
		0	Normal operation.
		1	Indicates a User read request.
1	UW		User write access type.
		0	Normal operation.
		1	Indicates a User write request.
0	Reserved	0	Reserved

In contrast to L2 and L1P, L1D does not implement the SX (supervisor execute) and UX (user execute) bits. The SX and UX fields in the L1DMPPA register always read as zero and do not respond to writes.

Table 3-21 illustrates the two memory protection default configurations.

Table 3-21. Memory Protection Defaults

Allowed IDs (Bits 15:8)	Reserved Bits (Bits 7:6)	Access Types (Bits 5:0)	Notes
1111 1111	11	110 110	All devices may access, both from User and Supervisor modes.

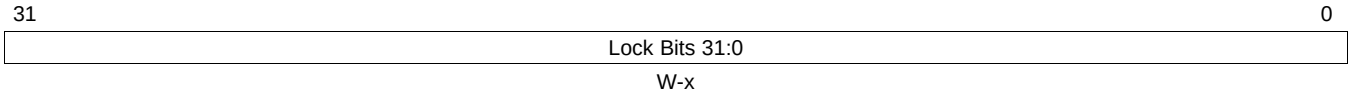
3.7.2.2 Memory Protection Lock Registers

The L1D controller implements a 64-bit lock register for controlling write access to the memory protection registers. The behavior of these lock registers is defined in Chapter 8.

3.7.2.2.1 Level 1 Data Memory Protection Lock Register 0 (L1DMPLK0)

The level 1 data memory protection lock register 0 (L1DMPLK0) is shown in Figure 3-14.

Figure 3-14. Level 1 Data Memory Protection Lock Register 0 (L1DMPLK0)

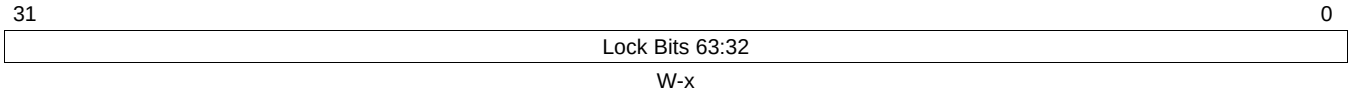


LEGEND: W = Write only; -*x* = value is indeterminate, see your device-specific data manual

3.7.2.2.2 Level 1 Data Memory Protection Lock Register 1 (L1DMPLK1)

The level 1 data memory protection lock register 1 (L1DMPLK1) is shown in Figure 3-15.

Figure 3-15. Level 1 Data Memory Protection Lock Register 1 (L1DMPLK1)

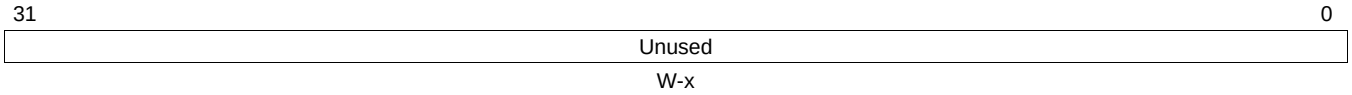


LEGEND: W = Write only; -*x* = value is indeterminate, see your device-specific data manual

3.7.2.2.3 Level 1 Data Memory Protection Lock Register 2 (L1DMPLK2)

The level 1 data memory protection lock register 2 (L1DMPLK2) is shown in Figure 3-16.

Figure 3-16. Level 1 Data Memory Protection Lock Register 2 (L1DMPLK2)

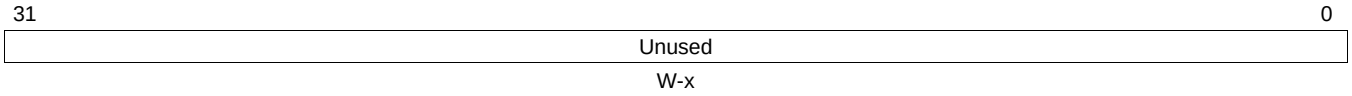


LEGEND: W = Write only; -*x* = value is indeterminate, see your device-specific data manual

3.7.2.2.4 Level 1 Data Memory Protection Lock Register 3 (L1DMPLK3)

The level 1 data memory protection lock register 3 (L1DMPLK3) is shown in Figure 3-17.

Figure 3-17. Level 1 Data Memory Protection Lock Register 3 (L1DMPLK3)



LEGEND: W = Write only; -*x* = value is indeterminate, see your device-specific data manual

3.7.2.2.5 Level 1 Data Memory Protection Lock Command Register (L1DMPLKCMD)

The level 1 data memory protection lock command register (L1DMPLKCMD) is shown in Figure 3-18 and described in Table 3-22.

Figure 3-18. Level 1 Data Memory Protection Lock Command Register (L1DMPLKCMD)

31	16
Reserved	
R-0	

15	3	2	1	0
Reserved		KEYR	LOCK	UNLOCK
R-0		W-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 3-22. Level 1 Data Memory Protection Command Register (L1DMPLKCMD) Field Descriptions

Bit	Field	Value	Description
31-3	Reserved	0	Reserved
2	KEYR		Reset status.
		0	No effect.
		1	Reset status.
1	LOCK		Interface to complete a lock sequence.
		0	No effect.
		1	Locks the lock provided that the software executed the sequence correctly.
0	UNLOCK		Interface to complete an unlock sequence.
		0	No effect.
		1	Unlocks the lock provided that the software executed the sequence correctly.

3.7.2.2.6 Level 1 Data Memory Protection Lock Status Register (L1DMPLKSTAT)

The level 1 data memory protection lock status register (L1DMPLKSTAT) is shown in Figure 3-19 and described in Table 3-23.

Figure 3-19. Level 1 Data Memory Protection Lock Status Register (L1DMPLKSTAT)

31	16
Reserved	
R-0	

15	1	0
Reserved		LK
R-0		R-0

LEGEND: R = Read only; -*n* = value after reset

Table 3-23. Level 1 Data Memory Protection Status Register (L1DMPLKSTAT) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved.
0	LK		Indicates the lock's current status.
		0	Lock is disengaged.
		1	Lock is engaged.

As illustrated above, the memory protection architecture allows for lock sizes up to 128 bits. The L1D controller only implements a 64-bit lock behind the lock interface. Thus, the values written to L1DMPLCK2 and L1DMPLCK3 are ignored. The behavior of the lock mechanism with respect to this shorter key is defined in Chapter 8.

3.7.2.3 Memory Protection Fault Registers

In order to allow programs to diagnose a memory protection fault after an exception occurs, the three L1D registers are dedicated to storing information about the fault. These registers are illustrated in Figure 3-20 through Figure 3-22 below.

3.7.2.3.1 Memory Protection Fault Address Register (L1DMPFAR)

The memory protection fault address register (L1DMPFAR) is shown in Figure 3-20 and described in Table 3-24.

Figure 3-20. Memory Protection Fault Address Register (L1DMPFAR)

31	0
Fault Address	
R-x	

LEGEND: R = Read only; -*x* = value is indeterminate, see your device-specific data manual

Table 3-24. Memory Protection Fault Address Register (L1DMPFAR) Field Descriptions

Bit	Field	Value	Description
31-0	Fault Address	0-FFFF FFFFh	Address of the fault.

3.7.2.3.2 Memory Protection Fault Set Register (L1DMPFSR)

The memory protection fault set register (L1DMPFSR) is shown in Figure 3-21 and described in Table 3-25.

Figure 3-21. Memory Protection Fault Set Register (L1DMPFSR)

31	16
Reserved	
R-0	

15	9	8	7	6	5	4	3	2	1	0
FID		LOCAL	Reserved		SR	SW	Reserved	UR	UW	Reserved
R-0		R-0	R-0				R-0	R/W-config		R-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 3-25. Memory Protection Fault Set Register (L1DMPFSR) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved.
15-9	FID	0-7Fh	Bits 6:0 of ID of faulting requestor. If ID is narrower than 7 bits, the remaining bits return 0. If ID is wider than 7 bits, the additional bits get truncated. FID = 0. If LOCAL = 1.
8	LOCAL		LOCAL access.
		0	Normal operation.
		1	Access was a LOCAL access.
7-6	Reserved	0	Reserved
5	SR		Supervisor read access type.
		0	Normal operation.
		1	Indicates a supervisor read request.
4	SW		Supervisor write access type.
		0	Normal operation.
		1	Indicates a supervisor write request.
3	Reserved	0	Reserved
2	UR		User read access type.
		0	Normal operation.
		1	Indicates a user read request.
1	UW		User write access type.
		0	Normal operation.
		1	Indicates a user write request.
0	Reserved	0	Reserved

3.7.2.3.3 Memory Protection Fault Clear Register (L1DMPFCR)

The memory protection fault clear register (L1DMPFCR) is shown in Figure 3-22 and described in Table 3-26.

Figure 3-22. Memory Protection Fault Clear Register (L1DMPFCR)

31	16
Reserved	
R-0	
15	0
MPFCLR	
W-0	

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 3-26. Memory Protection Fault Clear Register (L1DMPFCR) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved.
0	MPFCLR		Command to clear the L1DMPFAR register.
		0	No effect.
		1	Clear the L1DMPFAR and the L1DMPFCR registers.

Chapter 8 provides the definition and meanings for these registers.

The L1DMPFAR and L1DMPFSR registers only store enough information for one fault. The hardware records the information about the first fault and generates an exception only for that fault.

The fault information is preserved until software clears it by writing a 1 to the MPFCLR field in the L1DMPFCR register. Writing a 0 to the MPFCLR field in the L1DMPFCR has no effect. L1D ignores the value written to bits 31:1 of the L1DMPFCR register.

3.7.3 Protection Checks on Accesses to Memory Protection Registers

L1D implements permission checks on the memory protection registers themselves. The rules are as follows:

- All requestors may read any memory protection (MP) register at any time in all circumstances, except the lock key registers (L1DMPLK0 through L1DMPLK3).
- Supervisor may write the registers.

Table 3-27 summarizes which L1D memory protection registers are accessible and what protection checks are performed in the megamodule according to role.

Table 3-27. Permissions for L1D Memory Protection Registers

Register	Supervisor	User
L1DMPFAR	R	R
L1DMPFSR	R	R
L1DMPFCR	W	/
L1DMPLK0	W	/
L1DMPLK1	W	/
L1DMPLK2	W	/
L1DMPLK3	W	/
L1DMPLKCMD	W	/
L1DMPLKSTAT	R	R
L1DMPPAxx	R/W	R

Level 2 Memory and Cache

Topic **Page**

4.1 Introduction 82
4.2 Level 2 Memory Architecture 82
4.3 L2 Cache 84
4.4 L2 Cache Control Registers 93
4.5 L2 Power-Down 107
4.6 L2 Memory Protection 112

4.1 Introduction

4.1.1 Purpose of the Level 2 (L2) Memory and Cache

The L2 memory controller provides an on-chip memory solution between the faster level 1 memories (L1D, L1P) and slower external memories. It is advantageous in that it supports larger memory sizes than the L1 memories, while providing faster access than external memories.

Similar to the L1 memories, you can configure L2 to provide both cached and non-cached (i.e., addressable) memories.

4.1.2 Features

The L2 memory and cache provides the memory flexibility required in a device using the C64x+ megamodule:

- Two memory ports, port 0 and port 1.
- Configurable L2 cache size: 32KB, 64KB, 128KB, and 256KB
- Memory protection
- Supports cache block and global coherence operations
- Four configurable power-down pages

4.1.3 Terms and Definitions

Refer to Appendix A and Appendix B of this document for detailed definitions of the terms used in this chapter. Appendix A describes general terms used throughout this reference guide. Appendix B defines terms related to the memory and cache architecture.

4.2 Level 2 Memory Architecture

4.2.1 L2 Memory

4.2.1.1 L2 Memory Ports

The L2 memory provides two 256-bit wide memory ports, they are referred to as port 0 and port 1. The usage of the two ports is device-dependant. In most devices, the two memory ports are used as follows:

- Port 0
  - L2 RAM
  - L2 cache
- Port 1
  - L2 ROM
  - L2 RAM
  - Shared memory interface

4.2.1.2 L2 Memory Sizes

The L2 controller supports memory sizes in the 64K to 819K range for each port.

4.2.1.3 L2 Memory Banking

The L2 memory implements two separate memory ports. Each memory port can control one of:

- 4 × 128-bit banks
- 2 × 128-bit banks
- 1 × 256-bit bank

Refer to the device-specific data manual for more information about the banking scheme implemented on a particular device.

The two memory ports may address memory sections which may or may not be contiguous.

Table 4-1 illustrates how port 0 and port 1 banking looks in the 2 × 128-bit case for the little endian mode.

Table 4-1. C64x+ Megamodule 2 × 128-bit Banking Scheme

Port 1							
Bank 1				Bank 0			
xx14	xx16	xx15	xx14	xx13	xx12	xx11	xx10
xx0F	xx0E	xx0D	xx0C	xx0B	xx0A	xx09	xx08
xx07	xx06	xx05	xx04	xx03	xx02	xx01	xx00
Port 0							
Bank 1				Bank 0			
yy17	yy16	yy15	yy14	yy13	yy12	yy11	yy10
yy0F	yy0E	yy0D	yy0C	yy0B	yy0A	yy09	yy08
yy07	yy06	yy05	yy04	yy03	yy02	yy01	yy00

NOTE: The two memory ports may or may not be contiguous.

An L1P read miss (32 bytes) requires all memory banks of a single port. No other access can be made on that port during the same cycle, or until completion of the access when the L2 memory is high-latency (the latency of the memory is determined at chip design - refer to the device-specific data manual for more information). An L1D read miss (64 bytes) and victim writebacks require all memory banks on a single port for two consecutive accesses.

4.2.1.4 Simultaneous Accesses to L2 Memory

When various requestors such as L1P, L1D, IDMA, etc. try to access the L2 memory simultaneously, their accesses are arbitrated by the rules defined in Chapter 6.

4.3 L2 Cache

The C64x+ CPU default configuration maps all L2 memory as RAM/ROM. The L2 controller's port 0 supports 32KB, 64KB, 128KB, or 256KB of 4-way set-associative cache. Any remaining memory beyond 256KB on port 0 and all memory attached to port 1 is always RAM or ROM.

The operation of the L2 cache is controlled through several registers. Table 4-2 provides a summary of these registers. These registers are mentioned throughout this section and are described in more detail in Section 4.4.

Table 4-2. Cache Registers Summary

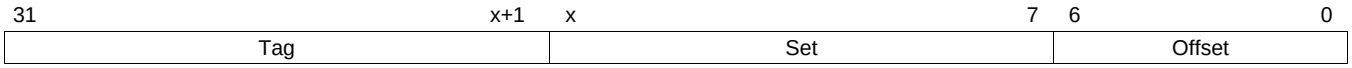
Acronym	Register Description	Section
L2CFG	Level 2 Configuration Register	Section 4.4.2
L2WBAR	Level 2 Writeback Base Address Register	Section 4.4.3.1.1
L2WWC	Level 2 Writeback Word Count Register	Section 4.4.3.1.2
L2WIBAR	Level 2 Writeback-Invalidate Base Address Register	Section 4.4.3.1.3
L2WIWC	Level 2 Writeback-Invalidate Word Count Register	Section 4.4.3.1.4
L2IBAR	Level 2 Invalidate Base Address Register	Section 4.4.3.1.5
L2IWC	Level 2 Invalidate Word Count Register	Section 4.4.3.1.6
L2WB	Level 2 Writeback Register	Section 4.4.3.2.1
L2WBINV	Level 2 Writeback-Invalidate Register	Section 4.4.3.2.2
L2INV	Level 2 Invalidate Register	Section 4.4.3.2.3
MARn	Memory Attribute Registers	Section 4.4.4

4.3.1 L2 Cache Architecture

The L2 cache is a read-and-write allocate, four-way set associative cache. In order to track the line state of the L2 cache, a four-way tag RAM is included. The address organization within the L2 tags is a function of the partitioning performed between cache and RAM, controlled via the L2MODE field in the L2CFG register control register bits.

Figure 4-1 outlines this function for the various sizes of cache supported.

Figure 4-1. L2 Cache Address Organization



The offset of 7 bits accounts for the fact that an L2 line size is 128 bytes. The cache control logic ignores this portion of the address. The set field indicates the L2 cache line address where the data would reside within each way, if it were cached. The width of the set field depends on the amount of L2 configured as cache, as defined in Table 4-3. The L2 controller uses the set field to look up and check the tags in each way for any already-cached data. It also looks up the valid bit, which indicates whether the contents of the line are considered valid for purposes of a tag compare.

The L2 cache configuration dictates the size of the set and tag fields, as described in Table 4-3.

Table 4-3. L2MODE Description

L2MODE Setting of the L2CFG Register	Amount of L2 Cache	X Bit Position	Description
000b	0K	N/A	L2 is all RAM
001b	32K	12	64 L2 cache lines
010b	64K	13	128 L2 cache lines
011b	128K	14	256 L2 cache lines
100b	256K	15	512 L2 cache lines
101b	Reserved. Maps to 256K.		
110b			
111b	Maximal cache. Maps to 256K.		

NOTE: In general, a larger value of L2MODE specifies a larger cache size, up to the size of the implemented L2 memory in port 0. The maximum actual L2 cache size is the smaller of the largest power-of-2 that fits in L2 RAM size and 256K.

The tag field is the upper portion of the address that identifies the true physical location of the cache line. The cache compares the tag field for a given address to the stored tag in all four ways of the L2 cache.

If any of the tags match and the cached data is valid, then the access is a "hit", and the element is read directly from or written directly to the L2 cache location. Otherwise, it is a "miss", and the requestor remains stalled while the L2 fetches a complete line from its system memory location. On read misses, the data is passed directly to the appropriate L1 cache as part of the fetch. On write misses, the L2 merges the write with the fetched line.

Since the contents of the L2 can be modified, the L2 cache is able to update the data in its true physical location. The L2 cache is a writeback cache, meaning that it writes out updates only when it needs to. Data is evicted from the L2 cache, written back to its proper location in system memory. This occurs when a new L2 line replaces one that has been modified, or when the L2 controller is told by the CPU (via software) to write back modified data. In the event of an eviction or writeback, the data is sent to its location in system memory via the EMC.

4.3.2 Replacement and Allocation Strategy

The L2 cache operates with a fixed four-way set associativity in all cache modes. This means that each location in system memory can reside in any one of four possible locations in the L2 cache.

The L2 controller implements a read and write-allocate cache. This means that the L2 will fetch a complete line of 128 bytes on any miss for a cacheable location, regardless of whether it is a read or a write. The replacement strategy is identical to that of the L1D, in that the least-recently-used (LRU) L2 line is replaced with the new line.

4.3.3 Reset Behavior

In response to a global reset, the L2 cache is switched to “All-RAM mode.”

In response to a local reset, the L2 cache is left in its current operating mode. However, the entire contents of the cache are invalidated. All requestors are stalled while this invalidation takes place.

If Level 1 cache support is enabled within the megamodule, then the L2 controller takes the necessary steps to ensure that the Level 1 caches respond in the same manner as the L2 to hard and soft resets.

4.3.4 L2 Mode Change Operations

The size of the L2 cache can be configured at run time. Programs select the size of L2 cache by writing the requested mode to the L2MODE field in the L2CFG register. Table 4-4 illustrates the valid settings for L2MODE.

Table 4-4. Cache Size Specified by L2CFG.L2MODE

L2MODE setting of the L2CFG Register	Amount of L2 Cache
000b	0K
001b	32K
010b	64K
011b	128K
100b	256K
101b	Reserved. Maps to 256K on C64x+ Megamodule.
110b	
111b	Maximal Cache. Maps to 256K on C64x+ Megamodule.

Typically, programs set the L2 mode shortly after reset and leave it unchanged. However, some programs change the L2 cache mode on the fly, particularly around OS task switches in a complex system. Be careful to maintain memory system coherence and correct cache operation by ensuring that you follow this procedure.

Table 4-5 outlines the required steps that you must perform:

Table 4-5. Switching L2 Modes

To Switch From	To	The Program Must Perform the Following Steps:
A mode with *no* or *some* L2 cache	A mode with more L2 cache	1. DMA, IDMA or copy any needed data out of the affected range of L2 RAM (If none requires saving, no DMA is necessary). 2. Wait for completion of any DMAs/IDMAs issued in the previous step. 3. Write the desired cache mode to the L2MODE field in the L2CFG register. 4. Read back the L2CFG register. This stalls the CPU until the mode change completes.
A mode with *some* L2 cache	A mode with *less* or *no* L2 cache	1. Write the desired cache mode to the L2MODE field in the L2CFG register. 2. Read back the L2CFG register. This stalls the CPU until the mode change completes.

When a program writes a new cache mode to the L2CFG register, the L2 performs the following steps:

- L2 cache is written back and invalidated if it is enabled.
- The L2 cache sets to the requested mode.

Note: Changing L2's mode does not affect the contents of either L1 cache.

4.3.5 L2 Freeze Mode

The L2 cache offers a freeze mode. The content of the L2 cache is frozen in this mode (i.e., it will not update as during normal operation). L2 freeze mode allows real-time applications to limit the amount of data evicted from L2 during various sections of code, such as interrupt handlers. Use the L2CC field in the L2CFG register to set this mode.

The freeze mode affects the operation of L2 cache only. L2 RAM is not affected by this mode. L2's freeze mode has no impact on L1D or L1P caches. Likewise, the L1's freeze modes have no impact on L2 cache.

The L2 cache responds to read and write hits normally when in freeze mode. L2 sends read and write misses directly to external memory, as if L2 cache were not present. The L2 never allocates a new cache line while frozen. Lines may only be evicted from L2 during freeze mode by program-initiated cache coherence operations, as defined in Section 4.3.6.

Table 4-6 provides a summary of the L2 freeze mode, set through the L2CC field in the L2CFG register.

Table 4-6. Freeze Mode Summary

L2 Mode	L2MODE	L2 Cache Enabled L2CC = 0	L2 Cache Enabled L2CC = 0	L2 Cache Freeze L2CC = 1
All RAM	000	No effect, because L2 is all RAM.		
Mixed cache and RAM, or all cache.	1000	Cache operates normally.	Cache frozen. Hits proceed normally. L1D misses are serviced as long-distance accesses for requested bytes only. L1P misses serviced as long-distance fetch for 1 fetch packet. No LRU updates in this mode.	

4.3.6 Program Initiated Cache Coherence Operations

The L2 memory architecture supports a variety of coherence operations that fall into two primary categories: block operations that operate on a specific range of addresses, and global operations which operate on the entire contents of one or more caches.

The following cache coherence operations are supported:

- Invalidation: Valid cache lines are made invalid. Content of the affected cache lines is discarded
- Writeback: The content of a valid and dirty cache line is written to a lower-level memory.
- Writeback-invalidation: Writeback operation followed by invalidation. Only the content of the affected cache lines is written to lower-level memory, but all the lines are invalidated.

4.3.6.1 Global Coherence Operations

Global coherence operations execute on the entire L2 cache. Some global coherence operations also affect L1 caches.

Table 4-7 lists all of the L2 global cache commands and the operations they perform on each of the three caches.

Table 4-7. Global Coherence Operations

Cache Operation	Register Used	L1P Effect	L1D Effect	L2 Effect
L2 Writeback	L2WB	No effect.	All updated data written back to L2/external, but left valid in L1D.	All updated data written back externally, but left valid in L2 cache.
L2 Writeback with Invalidate	L2WBINV	All lines invalidated in L1P.	All updated data written back to L2/external. All lines invalidated within L1D.	All updated data written back externally. All lines invalidated in L2.
L2 Invalidate	L2INV	All lines invalidated in L1P.	All lines invalidated in L1D. Updated data is dropped.	All lines invalidated in L2. Updated data is dropped.

Programs initiate global cache operations by writing a 1 to the appropriate register bit for each of the L2WB, L2WBINV, and L2INV registers.

Programs can write a 1 to the control register to initiate the coherence operation for the L2WB, L2WBINV, and L2INV registers. The control register sets to 0 upon completion of the operation. Programs can poll this bit to determine when the command completes.

Example 4-1 gives an example of how to use the L2WBINV register.

Example 4-1. Global Coherence Operation Example

```
/* ------------------------------------------------------------ */
/* Write back and Invalidate anything held in cache.            */
/* ------------------------------------------------------------ */
L2WBINV = 1;

/* ------------------------------------------------------------ */
/* OPTIONAL: Spin waiting for operation to complete.            */
/* ------------------------------------------------------------ */
while ((L2WBINV & 1) != 0) ;
```

The hardware does not require programs to poll for completion of these commands. The hardware may, however, stall programs while the global commands proceed.

Global cache operations work correctly regardless of the L2 freeze state. Further, global cache operations do not change the frozen state of the L2 cache.

4.3.6.2 Block Coherence Operations

Block coherence operations have similar functionality as the global coherence operations; however, they only apply to a defined block of data. This block is defined by the base address and by the word (32-bit) size in the associated registers.

Table 4-8 lists all of the block cache commands and the operation they perform on each of the three caches.

Table 4-8. Block Cache Operations

Cache Operation	Register Used	L1P Effect	L1D Effect	L2 Effect
L2 Writeback	L2WBAR L2WWC	No effect.	Updated data written back to L2/external, but left valid in L1D.	Updated data written back externally, but left valid in L2 cache.
L2 Writeback with Invalidate	L2WIBAR L2WIWC	All lines in range invalidated in L1P.	Updated data written back to L2/external. All lines in range invalidated within L1D.	Updated data written back externally. All lines in range invalidated in L2.
L2 Invalidate	L2IBAR L2IWC	All lines in range invalidated in L1P.	All lines in range invalidated in L1D. Updated data is dropped.	All lines in range invalidated in L2. Updated data is dropped.

Programs initiate block cache operations by writing a word address to the base address register first, and then writing a word count to the word count register. (Writing 1 to WC indicates a length of 4 bytes). If necessary, C64x+ megamodule enforces one or both of the following:

- Only one program-initiated coherence operation may be in progress at a time.
- Writes to either xxBAR or xxWC stall while another block or global cache coherence operation is in progress.

The xxBAR/xxWC mechanism for setting up block cache operations allows you to specify ranges down to word granularity. However, the memory system operates at cache-line granularity. Thus, all cache lines that overlap the range specified are acted upon.

Example 4-2 gives an example of how to use the block cache control registers.

Example 4-2. Block Coherence Operation Example

```
/* ------------------------------------------------------------ */
/* Write base address of array to Base Address Register.        */
/* Then write the length of the array, in words, to the Word    */
/* Count register.                                              */
/* ------------------------------------------------------------ */
L2WBAR = &array[0];
L2WWC = size of(array) / size of(int);

                              /* . . . */

/* ------------------------------------------------------------ */
/* The CPU can execute other code here. Block cache operations  */
/* proceed in parallel with CPU execution, stalling the CPU     */
/* minimally.                                                   */
/* ------------------------------------------------------------ */

                              /* . . . */

/* ------------------------------------------------------------ */
/* OPTIONAL: Spin waiting for operation to complete.            */
/* ------------------------------------------------------------ */
while (L2WWC != 0) ;
```

Writing to the xxBAR register sets up the base address for the next cache coherency operation. Writing a non-zero value to xxWC initiates the operation. The block cache logic starts the coherence command based on the specific xxWC register written.

Programs should not rely on the contents of xxBAR after or during a cache control operation; rather, programs should always write a new value to xxBAR prior to writing xxWC. Reading xxWC returns a non-zero value while a block cache operation is in progress, and zero when it is complete.

Block cache operations work correctly regardless of the L2 freeze state.

4.3.7 Cacheability Controls

In some applications, some specific addresses may need to be read from their physical locations each time they are accessed (e.g., a status register within FPGA).

The L2 controller offers registers that control whether certain ranges of memory are cacheable, and whether one or more requestors are actually permitted to access these ranges. The registers are referred to as MARs (memory attribute registers). A complete list of MAR registers is provided in section Section 4.4.4.

Note: Using the volatile keyword in the C language does not protect a variable from being cached. If an application uses a memory location periodically updated by external hardware, in order to protect this operation in C code follow these two steps:

- Use the volatile keyword to prevent the code generation tools from incorrectly optimizing the variable.
- You must program the MAR register of the range containing the variable to prevent caching.

4.3.7.1 MAR Functions

Each MAR register implements a single bit. The permit copies (PC) bit in each MAR register controls whether the cache may hold a copy of the affected address range. If PC = 1, the affected address range is cacheable. If PC = 0, the affected address range is not cacheable.

MAR registers are run-time programmable, except as noted in Section 4.3.7.2. All MAR register bits reset to a value of 0, thereby making the entire address space non-cacheable by default (except as noted in Section 4.3.7.2).

4.3.7.2 Special MAR Registers

MAR0 through MAR15 represent reserved address ranges in the C64x+ megamodule, and therefore are treated as follows:

1. MAR0 is implemented as a read-only value. The PC of the MAR0 is always read as 1.
2. MAR1 through 15 correspond to internal and external configuration address spaces. Therefore, these registers are read-only, and their PC field reads as 0.

Because MAR0 through MAR15 are read-only, the software does not need to manipulate these registers.

4.3.7.3 L1 Interaction

When L1P or L1D makes a request to L2 for an address that is not held in L2 RAM or L2 cache, the L2 controller queries the corresponding MAR register for that address. If the permit copies (PC) bit in the MAR register is 0, the L2 cache controller treats this as a non-cacheable access and initiates a long-distance access. If the access is a long distance read, the CPU stalls until the read data returns and the L1D will write-back dirty data if present in the LRU cache set that matches the non-cacheable memory address.

Concerning L1D long distance requests, the net result of the PC bit in the MAR is to prevent non-cacheable data from being stored in the L2 and L1D caches. Thus, when PC = 0 in a given MAR register, neither the L1D nor the L2 cache retains a copy of data accessed within the address range covered by that MAR.

The MAR registers have no effect on L1P. If L1P is enabled, it will always cache program fetches regardless of MAR configuration.

4.3.8 L1-L2 Coherence Support

This section describes the interaction imposed by the coherence rules between the L2 cache and the L1D and L1P caches.

The C64x+ DSP maintains the following coherence model:

1. Coherence between the C64x+ megamodule's L2 RAM segments and L1D cache is maintained.
2. Coherence between the C64x+ megamodule's L2 RAM segments and L1P cache is not maintained.
3. Coherence between the external memory and cached copies in L1 or L2 caches is not maintained.

Items 1 and 3 above are identical to C64x behavior. Item 2 above is different from C64x. Coherence is provided between DMA writes to L2 RAM and program fetches from L2 RAM on the C64x. C64x+ megamodule-based devices require you to manually issue block invalidates (as described in Section 4.3.6.2) when transferring in a block of code.

The following sections outline the functions that provide the L2-RAM-to-L1cache coherence.

4.3.8.1 Cache Coherence Protocol

In order to support coherence between the L1 and L2 caches, snoop-read and snoop-write commands are used.

The cache coherence protocol implements some features that are different from the ones implemented in the C64x devices. In the C64x+ protocol, coherence is supported between DMAs and L1D in L2, but not between DMAs and L1P. Also, in the C64x memory architecture, L1D cache is kept inclusive within L2, and thus requires snoops in response to L2 cache activity. The C64x+ megamodule removes this inclusiveness, thus limiting snoops to those triggered by DMA activity. L1 and L2 are still coherent with respect to each other, even though L1 is not inclusive within L2.

Cache "A" is inclusive in cache "B", if A's contents are always a subset of B's. If a line is held in A, but not in B, then A is not inclusive in B. A non-cacheable write may hit in L2 if the address was previously cacheable. This can happen if applications dynamically change the settings of the MAR registers.

Table 4-9 lists the coherence commands L2 can issue to L1D on a per-cache-line basis.

Table 4-9. L2 to L1D Coherence Commands

Snoop Command	Name	L1D Action	Triggered by
SNPR	Snoop Read	L1D sends L2 the contents of the requested half-line in L1D. Does not modify the dirty/valid/LRU state for the line.	DMA read from L2 RAM when L1D shadow tags say line is present and modified in L1D.
SNPW	Snoop Write	Up to 256 bits of new data is sent from L2 to L1D. L1D and L2 both update their respective copies of the data. The dirty and valid bits for the line in L1D do not change.	DMA write to L2 RAM when L1D shadow tags say line is present in L1D. Whether the line is modified in L1D does not matter.

Note: These snoop commands represent hardware activity that is transparent. They are included to help you understand the cache operation better.

4.3.8.2 L2 Cache Evictions

In the C64x memory architecture, when L2 evicts a line, it snoop-invalidates L1D, thereby keeping L1D inclusive in L2. In the C64x+ architecture, when L2 evicts a line, it writes the victim out if it is dirty, without consulting L1D. It does not invalidate the line in L1D. L2 also does not invalidate lines in L1P when evicting a line. As a result, neither L1D nor L1P is inclusive in L2.

4.3.8.3 Policy Relative to L1D Victims

L1D victim writebacks do not trigger line allocations in L2. L1D victims are written directly to external memory if they miss L2.

L1D victim writebacks also do not update L2’s LRU if they hit in L2. They do update L2’s dirty status as needed.

4.3.8.4 DMA/IDMA Write Interaction

When a DMA or IDMA write occurs to L2 RAM, the behavior of L2 depends on whether the data is cached in L1D. The behavior in the C64x+ architecture is different from the behavior in the C64x architecture. In the C64x architecture, snoop-invalidate commands are sent to L1P and L1D. In the C64x+ architecture, DMA/IDMA writes never invalidate lines in L1P. DMA/IDMA writes send snoop-writes to L1D if the address range is present in L1D and otherwise nothing.

4.3.8.5 DMA/IDMA Read Interaction

The L2 memory keeps a shadow copy of L1D tags. This shadow includes both 'dirty' and 'valid' status.

When DMA/IDMA read L2 RAM, the L2 consults the shadow tag. If the given address is marked as 'valid' and 'dirty' in L1D, the L2 sends a snoop-read request for the address to L1D. L1D responds with the requested data.

The snoop-read leaves the data valid in L1D, and does not evict or write back the data to L2. As a consequence, a buffer allocated in L1D is to left allocated in L1D so that algorithms running on the CPU can subsequently refill the buffer without incurring cache miss penalties.

4.4 L2 Cache Control Registers

The C64x+ memory system provides a set of registers to govern the operation of L2 cache. These registers fall into several categories covered in the following sections:

- **Cache Size and Operating Mode Controls**. These registers control the size of the cache, and whether the cache is frozen or operating normally. They are described in Section 4.4.2.
- **Block-oriented and Global Coherence Operations.** These operations allow programs to manually move data out of the cache.
- **Cacheability Controls.** These registers control whether the cache is permitted to store copies of certain ranges of memory. They are described in Section 4.4.4.

4.4.1 Memory Mapped L2 Cache Control Registers Overview

Table 4-10 below lists the L2 cache registers.

Table 4-10. Cache Control Registers

Address	Acronym	Register Description	Section
0184 0000h	L2CFG	L2 Configuration Register	Section 4.4.2
0184 4000h	L2WBAR	L2 Writeback Base Address Register	Section 4.4.3.1.1
0184 4004h	L2WWC	L2 Writeback Word Count Register	Section 4.4.3.1.2
0184 4010h	L2WIBAR	L2 Writeback-Invalidate Base Address Register	Section 4.4.3.1.3
0184 4014h	L2WIWC	L2 Writeback-Invalidate Word Count Register	Section 4.4.3.1.4
0184 4018h	L2IBAR	L2 Invalidate Base Address Register	Section 4.4.3.1.5
0184 401Ch	L2IWC	L2 Invalidate Word Count Register	Section 4.4.3.1.6
0184 5000h	L2WB	L2 Writeback Register	Section 4.4.3.2.1
0184 5004h	L2WBINV	L2 Writeback-Invalidate Register	Section 4.4.3.2.2
0184 5008h	L2INV	L2 Invalidate Register	Section 4.4.3.2.3

4.4.2 L2 Configuration Register (L2CFG)

The L2CFG register controls operating the L2 cache. The L2CFG sets the amount of L2 memory that acts as cache, controls L2 freeze modes, and holds L1D/L1P invalidate bits.

The L2 configuration register (L2CFG) is shown in Figure 4-2 and described in Table 4-11.

Figure 4-2. L2 Configuration Register (L2CFG)

31 28	27 24	23 20	19 16
Reserved	NUM MM	Reserved	MMID
R-0	R-config	R-0	R-config

15 10	9	8	7 4	3	2 0
Reserved	IP	ID	Reserved	L2CC	L2MODE
R-0	W-0	W-0	R-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; W = Write only; -*n* = value after reset

Table 4-11. L2 Configuration Register (L2CFG) Field Descriptions

Bit	Field	Value	Description
31-28	Reserved	0	Reserved
27-24	NUM MM	0-Fh	Number of megamodules minus 1. Used in multi-processing environment.
23-20	Reserved	0	Reserved
19-16	MMID	0-Fh	Contains the megamodule ID number. Used in a multiprocessing environment where several megamodules are present.
15-10	Reserved	0	Reserved
9	IP		L1P global invalidate bit. Provided for backward compatibility, new applications should use the L1PINV register described in the Chapter 2.
		0	Normal L1P operation.
		1	All L1P lines are invalidated.
8	ID		L1D global invalidate bit. Provided for backward compatibility, new applications should use the L1DINV register described in the Chapter 3.
		0	Normal L1D operation.
		1	All L1D lines are invalidated.
7-4	Reserved	0	Reserved
3	L2CC		Controls the freeze mode
		0	Normal operation
		1	L2 cache frozen
2-0	L2MODE	0-7h	Defines the size of L2 cache.
		0h	L2 cache disabled.
		1h	32K
		2h	64K
		3h	128K
		4h	256K
		5h	Maximum cache
		6h	Maximum cache
		7h	Maximum cache

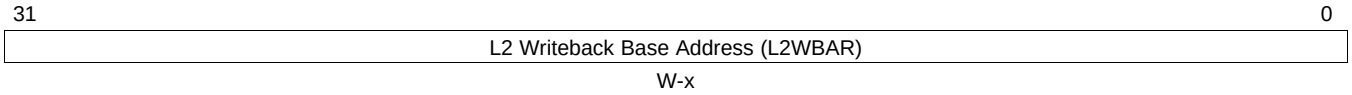
4.4.3 L2 Cache Coherence Operation Registers

4.4.3.1 Block Coherence Operation Registers

4.4.3.1.1 L2 Writeback Base Address Register (L2WBAR)

The L2 writeback base address register (L2WBAR) is shown in Figure 4-3 and described in Table 4-12.

Figure 4-3. L2 Writeback Base Address Register (L2WBAR)



LEGEND: W = Write only; -*x* = value is indeterminate, see your device-specific data manual

Table 4-12. L2 Writeback Base Address Register (L2WBAR) Field Descriptions

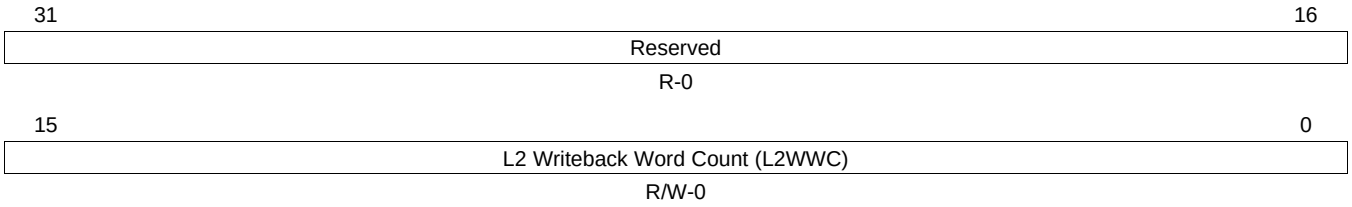
Bit	Field	Value	Description
31-0	L2WBAR	0-FFFF FFFFh	Defines the base address for the L2 block writeback operation.

4.4.3.1.2 L2 Writeback Word Count Register (L2WWC)

The L2 writeback word count register (L2WWC) defines the size of the block that will be invalidated. The size is defined in 32-bit words. Writing a number greater than FFE0h results in nothing being modified.

The L2 writeback word count register (L2WWC) is shown in Figure 4-4 and described in Table 4-13.

Figure 4-4. L2 Writeback Word Count Register (L2WWC)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-13. L2 Writeback Word Count Register (L2WWC) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	L2WWC	0-FFE0h	Word count for block invalidation. Writing FFE1h-FFFFh results in zero words being affected.

4.4.3.1.3 L2 Writeback-Invalidate Base Address (L2WIBAR)

The L2 writeback-invalidate base address register (L2WIBAR) is shown in Figure 4-5 and described in Table 4-14.

Figure 4-5. L2 Writeback-Invalidate Base Address Register (L2WIBAR)

31	0
L2 Writeback-Invalidate Base Address(L2WIBAR)	
W-x	

LEGEND: W = Write only; -*x* = value is indeterminate, see your device-specific data manual

Table 4-14. L2 Writeback-Invalidate Base Address Register (L2WIBAR) Field Descriptions

Bit	Field	Value	Description
31-0	L2WIBAR	0-FFFF FFFFh	Defines the base address for the L2 block writeback-invalidate operation

4.4.3.1.4 L2 Writeback-Invalidate Word Count Register (L2WIWC)

The L2 writeback-invalidate word count register (L2WIWC) defines the size of the block that will be invalidated. The size is defined in 32-bit words. Writing a number greater than FFE0h results in nothing being modified.

The L2 writeback-invalidate word count register (L2WIWC) is shown in Figure 4-6 and described in Table 4-15.

Figure 4-6. L2 Writeback-Invalidate Word Count Register (L2WIWC)

31	16
Reserved	
R-0	

15	0
L2 Writeback-Invalidate Word Count (L2WIWC)	
R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-15. L2 Writeback-Invalidate Word Count Register (L2WIWC) Field Descriptions

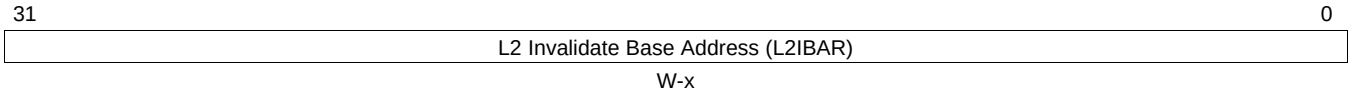
Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	L2WIWC	0-FFE0h	Word count for block invalidation. Writing FFE1h-FFFFh results in zero words being affected.

4.4.3.1.5 L2 Invalidate Base Address Register (L2IBAR)

The L2 invalidate base address register (L2IBAR) defines the base address of the block that will be invalidated.

The L2 invalidate base address register (L2IBAR) is shown in Figure 4-7 and described in Table 4-16.

Figure 4-7. L2 Invalidate Base Address Register (L2IBAR)



LEGEND: W = Write only; -*x* = value is indeterminate, see your device-specific data manual

Table 4-16. L2 Invalidate Base Address Register (L2IBAR) Field Descriptions

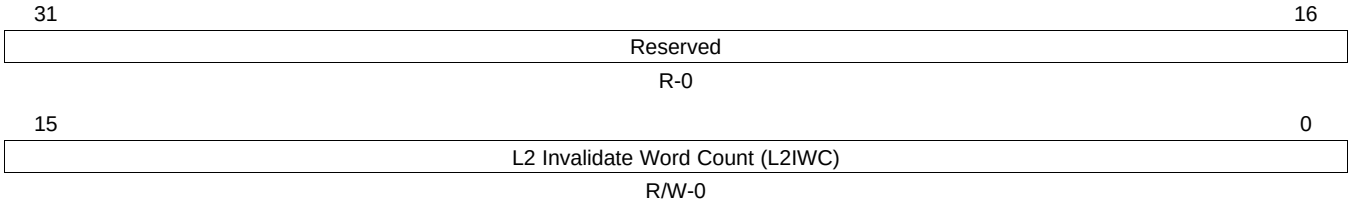
Bit	Field	Value	Description
31-0	L2IBAR	0-FFFF FFFFh	Defines the base address for the L2 block invalidate operation

4.4.3.1.6 L2 Invalidate Word Count Register (L2IWC)

The L2 invalidate word count register (L2IWC) defines the size of the block that will be invalidated. The size is defined in 32-bit words. Writing a number greater than FFE0h results in nothing being modified.

The L2 invalidate word count register (L2IWC) is shown in Figure 4-8 and described in Table 4-17.

Figure 4-8. L2 Invalidate Word Count Register (L2IWC)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-17. Invalidate Word Count Register (L2IWC) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	L2IWC	0-FFE0h	Word count for block invalidation. Writing FFE1h-FFFFh results in zero words being affected.

4.4.3.2 Global Coherence Operation Registers

4.4.3.2.1 L2 Writeback Register (L2WB)

The L2 writeback register (L2WB) controls the global writeback operation of the L2 cache.

The L2 writeback register (L2WB) is shown in Figure 4-9 and described in Table 4-18.

Figure 4-9. L2 Writeback Register (L2WB)

31	16
Reserved	
R-0	

15	1	0
Reserved		C
R/W-0		R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-18. L2 Writeback Register (L2WB) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	C		Controls the global writeback operation of L2 cache as described in Section 4.3.6.1.
		0	Normal operation
		1	Dirty L2 cache lines are written back

4.4.3.2.2 L2 Writeback-Invalidate Register (L2WBINV)

The L2 writeback-invalidate register (L2WBINV) controls the writeback-invalidate operation of L2 cache.

The L2 writeback-invalidate register (L2WBINV) is shown in Figure 4-10 and described in Table 4-19.

Figure 4-10. L2 Writeback-Invalidate Register (L2WBINV)

31	16
Reserved	
R-0	

15	1	0
Reserved		C
R-0		R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-19. L2 Writeback-Invalidate Register (L2WBINV) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	C		Controls the global writeback-invalidate operation of L2 cache as described in Section 4.3.6.1.
		0	Normal L2 operation
		1	Dirty L2 cache lines are written back. All L2 cache lines invalidated.

4.4.3.2.3 L2 Invalidate Register (L2INV)

The L2 invalidate register (L2INV) controls the global invalidation of the L2 cache and is shown in Figure 4-11 and described in Table 4-20.

Figure 4-11. L2 Invalidate Register (L2INV)

31	16
Reserved	
R-0	

15	1	0
Reserved		I
R-0		R/SW-0

LEGEND: R/W = Read/Write; R = Read only; *-n* = value after reset; R/SW = Read/writeable by the supervisor only.

Table 4-20. L2 Invalidate Register (L2INV) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	I		Controls the global invalidation of L2 cache.
		0	Normal operation
		1	All L2 cache lines are invalidated.

4.4.4 Memory Attribute Registers (MARn)

The L2 memory includes a set of registers to define the cacheability of external (to the megamodule) memory space(s). The registers, referred to as MARs (Memory Attribute Registers), are defined as shown in Table 4-33. The MAR registers are only writeable by Supervisor code.

Table 4-21 below lists the memory attribute memory mapped control registers.

Table 4-21. Memory Attribute Registers

Address	Acronym	Register Description	Defines Attributes for. . .
0184 8000h	MAR0	Memory Attribute Register 0	Local L2 RAM (fixed)
0184 8004h	MAR1	Memory Attribute Register 1	0100 0000h - 01FF FFFFh
0184 8008h	MAR2	Memory Attribute Register 2	0200 0000h - 02FF FFFFh
0184 800Ch	MAR3	Memory Attribute Register 3	0300 0000h - 03FF FFFFh
0184 8010h	MAR4	Memory Attribute Register 4	0400 0000h - 04FF FFFFh
0184 8014h	MAR5	Memory Attribute Register 5	0500 0000h - 05FF FFFFh
0184 8018h	MAR6	Memory Attribute Register 6	0600 0000h - 06FF FFFFh
0184 801Ch	MAR7	Memory Attribute Register 7	0700 0000h - 07FF FFFFh
0184 8020h	MAR8	Memory Attribute Register 8	0800 0000h - 08FF FFFFh
0184 8024h	MAR9	Memory Attribute Register 9	0900 0000h - 09FF FFFFh
0184 8028h	MAR10	Memory Attribute Register 10	0A00 0000h - 0AFF FFFFh
0184 802Ch	MAR11	Memory Attribute Register 11	0B00 0000h - 0BFF FFFFh
0184 8030h	MAR12	Memory Attribute Register 12	0C00 0000h - 0CFF FFFFh
0184 8034h	MAR13	Memory Attribute Register 13	0D00 0000h - 0DFF FFFFh
0184 8038h	MAR14	Memory Attribute Register 14	0E00 0000h - 0EFF FFFFh
0184 803Ch	MAR15	Memory Attribute Register 15	0F00 0000h - 0FFF FFFFh
0184 8040h	MAR16	Memory Attribute Register 16	1000 0000h - 10FF FFFFh
0184 8044h	MAR17	Memory Attribute Register 17	1100 0000h - 11FF FFFFh
0184 8048h	MAR18	Memory Attribute Register 18	1200 0000h - 12FF FFFFh
0184 804Ch	MAR19	Memory Attribute Register 19	1300 0000h - 13FF FFFFh
0184 8050h	MAR20	Memory Attribute Register 20	1400 0000h - 14FF FFFFh
0184 8054h	MAR21	Memory Attribute Register 21	1500 0000h - 15FF FFFFh
0184 8058h	MAR22	Memory Attribute Register 22	1600 0000h - 16FF FFFFh
0184 805Ch	MAR23	Memory Attribute Register 23	1700 0000h - 17FF FFFFh
0184 8060h	MAR24	Memory Attribute Register 24	1800 0000h - 18FF FFFFh
0184 8064h	MAR25	Memory Attribute Register 25	1900 0000h - 19FF FFFFh
0184 8068h	MAR26	Memory Attribute Register 26	1A00 0000h - 1AFF FFFFh
0184 806Ch	MAR27	Memory Attribute Register 27	1B00 0000h - 1BFF FFFFh
0184 8070h	MAR28	Memory Attribute Register 28	1C00 0000h - 1CFF FFFFh
0184 8074h	MAR29	Memory Attribute Register 29	1D00 0000h - 1DFF FFFFh
0184 8078h	MAR30	Memory Attribute Register 30	1E00 0000h - 1EFF FFFFh
0184 807Ch	MAR31	Memory Attribute Register 31	1F00 0000h - 1FFF FFFFh
0184 8080h	MAR32	Memory Attribute Register 32	2000 0000h - 20FF FFFFh
0184 8084h	MAR33	Memory Attribute Register 33	2100 0000h - 21FF FFFFh
0184 8088h	MAR34	Memory Attribute Register 34	2200 0000h - 22FF FFFFh
0184 808Ch	MAR35	Memory Attribute Register 35	2300 0000h - 23FF FFFFh
0184 8090h	MAR36	Memory Attribute Register 36	2400 0000h - 24FF FFFFh
0184 8094h	MAR37	Memory Attribute Register 37	2500 0000h - 25FF FFFFh
0184 8098h	MAR38	Memory Attribute Register 38	2600 0000h - 26FF FFFFh
0184 809Ch	MAR39	Memory Attribute Register 39	2700 0000h - 27FF FFFFh
0184 80A0h	MAR40	Memory Attribute Register 40	2800 0000h - 28FF FFFFh

Table 4-21. Memory Attribute Registers (continued)

Address	Acronym	Register Description	Defines Attributes for. . .
0184 80A4h	MAR41	Memory Attribute Register 41	2900 0000h - 29FF FFFFh
0184 80A8h	MAR42	Memory Attribute Register 42	2A00 0000h - 2AFF FFFFh
0184 80ACh	MAR43	Memory Attribute Register 43	2B00 0000h - 2BFF FFFFh
0184 80B0h	MAR44	Memory Attribute Register 44	2C00 0000h - 2CFF FFFFh
0184 80B4h	MAR45	Memory Attribute Register 45	2D00 0000h - 2DFF FFFFh
0184 80B8h	MAR46	Memory Attribute Register 46	2E00 0000h - 2EFF FFFFh
0184 80BCh	MAR47	Memory Attribute Register 47	2F00 0000h - 2FFF FFFFh
0184 80C0h	MAR48	Memory Attribute Register 48	3000 0000h - 30FF FFFFh
0184 80C4h	MAR49	Memory Attribute Register 49	3100 0000h - 31FF FFFFh
0184 80C8h	MAR50	Memory Attribute Register 50	3200 0000h - 32FF FFFFh
0184 80CCh	MAR51	Memory Attribute Register 51	3300 0000h - 33FF FFFFh
0184 80D0h	MAR52	Memory Attribute Register 52	3400 0000h - 34FF FFFFh
0184 80D4h	MAR53	Memory Attribute Register 53	3500 0000h - 35FF FFFFh
0184 80D8h	MAR54	Memory Attribute Register 54	3600 0000h - 36FF FFFFh
0184 80DCh	MAR55	Memory Attribute Register 55	3700 0000h - 37FF FFFFh
0184 80E0h	MAR56	Memory Attribute Register 56	3800 0000h - 38FF FFFFh
0184 80E4h	MAR57	Memory Attribute Register 57	3900 0000h - 39FF FFFFh
0184 80E8h	MAR58	Memory Attribute Register 58	3A00 0000h - 3AFF FFFFh
0184 80ECh	MAR59	Memory Attribute Register 59	3B00 0000h - 3BFF FFFFh
0184 80F0h	MAR60	Memory Attribute Register 60	3C00 0000h - 3CFF FFFFh
0184 80F4h	MAR61	Memory Attribute Register 61	3D00 0000h - 3DFF FFFFh
0184 80F8h	MAR62	Memory Attribute Register 62	3E00 0000h - 3EFF FFFFh
0184 80FCh	MAR63	Memory Attribute Register 63	3F00 0000h - 3FFF FFFFh
0184 8100h	MAR64	Memory Attribute Register 64	4000 0000h - 40FF FFFFh
0184 8104h	MAR65	Memory Attribute Register 65	4100 0000h - 41FF FFFFh
0184 8108h	MAR66	Memory Attribute Register 66	4200 0000h - 42FF FFFFh
0184 810Ch	MAR67	Memory Attribute Register 67	4300 0000h - 43FF FFFFh
0184 8110h	MAR68	Memory Attribute Register 68	4400 0000h - 44FF FFFFh
0184 8114h	MAR69	Memory Attribute Register 69	4500 0000h - 45FF FFFFh
0184 8118h	MAR70	Memory Attribute Register 70	4600 0000h - 46FF FFFFh
0184 811Ch	MAR71	Memory Attribute Register 71	4700 0000h - 47FF FFFFh
0184 8120h	MAR72	Memory Attribute Register 72	4800 0000h - 48FF FFFFh
0184 8124h	MAR73	Memory Attribute Register 73	4900 0000h - 49FF FFFFh
0184 8128h	MAR74	Memory Attribute Register 74	4A00 0000h - 4AFF FFFFh
0184 812Ch	MAR75	Memory Attribute Register 75	4B00 0000h - 4BFF FFFFh
0184 8130h	MAR76	Memory Attribute Register 76	4C00 0000h - 4CFF FFFFh
0184 8134h	MAR77	Memory Attribute Register 77	4D00 0000h - 4DFF FFFFh
0184 8138h	MAR78	Memory Attribute Register 78	4E00 0000h - 4EFF FFFFh
0184 813Ch	MAR79	Memory Attribute Register 79	4F00 0000h - 4FFF FFFFh
0184 8140h	MAR80	Memory Attribute Register 80	5000 0000h - 50FF FFFFh
0184 8144h	MAR81	Memory Attribute Register 81	5100 0000h - 51FF FFFFh
0184 8148h	MAR82	Memory Attribute Register 82	5200 0000h - 52FF FFFFh
0184 814Ch	MAR83	Memory Attribute Register 83	5300 0000h - 53FF FFFFh
0184 8150h	MAR84	Memory Attribute Register 84	5400 0000h - 54FF FFFFh
0184 8154h	MAR85	Memory Attribute Register 85	5500 0000h - 55FF FFFFh
0184 8158h	MAR86	Memory Attribute Register 86	5600 0000h - 56FF FFFFh
0184 815Ch	MAR87	Memory Attribute Register 87	5700 0000h - 57FF FFFFh

Table 4-21. Memory Attribute Registers (continued)

Address	Acronym	Register Description	Defines Attributes for. . .
0184 8160h	MAR88	Memory Attribute Register 88	5800 0000h - 58FF FFFFh
0184 8164h	MAR89	Memory Attribute Register 89	5900 0000h - 59FF FFFFh
0184 8168h	MAR90	Memory Attribute Register 90	5A00 0000h - 5AFF FFFFh
0184 816Ch	MAR91	Memory Attribute Register 91	5B00 0000h - 5BFF FFFFh
0184 8170h	MAR92	Memory Attribute Register 92	5C00 0000h - 5CFF FFFFh
0184 8174h	MAR93	Memory Attribute Register 93	5D00 0000h - 5DFF FFFFh
0184 8178h	MAR94	Memory Attribute Register 94	5E00 0000h - 5EFF FFFFh
0184 817Ch	MAR95	Memory Attribute Register 95	5F00 0000h - 5FFF FFFFh
0184 8180h	MAR96	Memory Attribute Register 96	6000 0000h - 60FF FFFFh
0184 8184h	MAR97	Memory Attribute Register 97	6100 0000h - 61FF FFFFh
0184 8188h	MAR98	Memory Attribute Register 98	6200 0000h - 62FF FFFFh
0184 818Ch	MAR99	Memory Attribute Register 99	6300 0000h - 63FF FFFFh
0184 8190h	MAR100	Memory Attribute Register 100	6400 0000h - 64FF FFFFh
0184 8194h	MAR101	Memory Attribute Register 101	6500 0000h - 65FF FFFFh
0184 8198h	MAR102	Memory Attribute Register 102	6600 0000h - 66FF FFFFh
0184 819Ch	MAR103	Memory Attribute Register 103	6700 0000h - 67FF FFFFh
0184 81A0h	MAR104	Memory Attribute Register 104	6800 0000h - 68FF FFFFh
0184 81A4h	MAR105	Memory Attribute Register 105	6900 0000h - 69FF FFFFh
0184 81A8h	MAR106	Memory Attribute Register 106	6A00 0000h - 6AFF FFFFh
0184 81ACh	MAR107	Memory Attribute Register 107	6B00 0000h - 6BFF FFFFh
0184 81B0h	MAR108	Memory Attribute Register 108	6C00 0000h - 6CFF FFFFh
0184 81B4h	MAR109	Memory Attribute Register 109	6D00 0000h - 6DFF FFFFh
0184 81B8h	MAR110	Memory Attribute Register 110	6E00 0000h - 6EFF FFFFh
0184 81BCh	MAR111	Memory Attribute Register 111	6F00 0000h - 6FFF FFFFh
0184 81C0h	MAR112	Memory Attribute Register 112	7000 0000h - 70FF FFFFh
0184 81C4h	MAR113	Memory Attribute Register 113	7100 0000h - 71FF FFFFh
0184 81C8h	MAR114	Memory Attribute Register 114	7200 0000h - 72FF FFFFh
0184 81CCh	MAR115	Memory Attribute Register 115	7300 0000h - 73FF FFFFh
0184 81D0h	MAR116	Memory Attribute Register 116	7400 0000h - 74FF FFFFh
0184 81D4h	MAR117	Memory Attribute Register 117	7500 0000h - 75FF FFFFh
0184 81D8h	MAR118	Memory Attribute Register 118	76000000h - 76FFFFFFh
0184 81DCh	MAR119	Memory Attribute Register 119	7700 0000h - 77FF FFFFh
0184 81E0h	MAR120	Memory Attribute Register 120	7800 0000h - 78FF FFFFh
0184 81E4h	MAR121	Memory Attribute Register 121	7900 0000h - 79FF FFFFh
0184 81E8h	MAR122	Memory Attribute Register 122	7A00 0000h - 7AFF FFFFh
0184 81ECh	MAR123	Memory Attribute Register 123	7B00 0000h - 7BFF FFFFh
0184 81F0h	MAR124	Memory Attribute Register 124	7C00 0000h - 7CFF FFFFh
0184 81F4h	MAR125	Memory Attribute Register 125	7D00 0000h - 7DFF FFFFh
0184 81F8h	MAR126	Memory Attribute Register 126	7E00 0000h - 7EFF FFFFh
0184 81FCh	MAR127	Memory Attribute Register 127	7F00 0000h - 7FFF FFFFh
0184 8200h	MAR128	Memory Attribute Register 128	8000 0000h - 80FF FFFFh
0184 8204h	MAR129	Memory Attribute Register 129	8100 0000h - 81FF FFFFh
0184 8208h	MAR130	Memory Attribute Register 130	8200 0000h - 82FF FFFFh
0184 820Ch	MAR131	Memory Attribute Register 131	8300 0000h - 83FF FFFFh
0184 8210h	MAR132	Memory Attribute Register 132	8400 0000h - 84FF FFFFh
0184 8214h	MAR133	Memory Attribute Register 133	8500 0000h - 85FF FFFFh
0184 8218h	MAR134	Memory Attribute Register 134	8600 0000h - 86FF FFFFh

Table 4-21. Memory Attribute Registers (continued)

Address	Acronym	Register Description	Defines Attributes for. . .
0184 821Ch	MAR135	Memory Attribute Register 135	8700 0000h - 87FF FFFFh
0184 8220h	MAR136	Memory Attribute Register 136	8800 0000h - 88FF FFFFh
0184 8224h	MAR137	Memory Attribute Register 137	8900 0000h - 89FF FFFFh
0184 8228h	MAR138	Memory Attribute Register 138	8A00 0000h - 8AFF FFFFh
0184 822Ch	MAR139	Memory Attribute Register 139	8B00 0000h - 8BFF FFFFh
0184 8230h	MAR140	Memory Attribute Register 140	8C00 0000h - 8CFF FFFFh
0184 8234h	MAR141	Memory Attribute Register 141	8D00 0000h - 8DFF FFFFh
0184 8238h	MAR142	Memory Attribute Register 142	8E00 0000h - 8EFF FFFFh
0184 823Ch	MAR143	Memory Attribute Register 143	8F00 0000h - 8FFF FFFFh
0184 8240h	MAR144	Memory Attribute Register 144	9000 0000h - 90FF FFFFh
0184 8244h	MAR145	Memory Attribute Register 145	9100 0000h - 91FF FFFFh
0184 8248h	MAR146	Memory Attribute Register 146	9200 0000h - 92FF FFFFh
0184 824Ch	MAR147	Memory Attribute Register 147	9300 0000h - 93FF FFFFh
0184 8250h	MAR148	Memory Attribute Register 148	9400 0000h - 94FF FFFFh
0184 8254h	MAR149	Memory Attribute Register 149	9500 0000h - 95FF FFFFh
0184 8258h	MAR150	Memory Attribute Register 150	9600 0000h - 96FF FFFFh
0184 825Ch	MAR151	Memory Attribute Register 151	9700 0000h - 97FF FFFFh
0184 8260h	MAR152	Memory Attribute Register 152	9800 0000h - 98FF FFFFh
0184 8264h	MAR153	Memory Attribute Register 153	990 00000h - 99FF FFFFh
0184 8268h	MAR154	Memory Attribute Register 154	9A00 0000h - 9AFF FFFFh
0184 826Ch	MAR155	Memory Attribute Register 155	9B00 0000h - 9BFF FFFFh
0184 8270h	MAR156	Memory Attribute Register 156	9C00 0000h - 9CFF FFFFh
0184 8274h	MAR157	Memory Attribute Register 157	9D00 0000h - 9DFF FFFFh
0184 8278h	MAR158	Memory Attribute Register 158	9E00 0000h - 9EFF FFFFh
0184 827Ch	MAR159	Memory Attribute Register 159	9F00 0000h - 9FFF FFFFh
0184 8280h	MAR160	Memory Attribute Register 160	A000 0000h - A0FF FFFFh
0184 8284h	MAR161	Memory Attribute Register 161	A100 0000h - A1FF FFFFh
0184 8288h	MAR162	Memory Attribute Register 162	A200 0000h - A2FF FFFFh
0184 828Ch	MAR163	Memory Attribute Register 163	A300 0000h - A3FF FFFFh
0184 8290h	MAR164	Memory Attribute Register 164	A400 0000h - A4FF FFFFh
0184 8294h	MAR165	Memory Attribute Register 165	A500 0000h - A5FF FFFFh
0184 8298h	MAR166	Memory Attribute Register 166	A600 0000h - A6FF FFFFh
0184 829Ch	MAR167	Memory Attribute Register 167	A700 0000h - A7FF FFFFh
0184 82A0h	MAR168	Memory Attribute Register 168	A800 0000h - A8FF FFFFh
0184 82A4h	MAR169	Memory Attribute Register 169	A900 0000h - A9FF FFFFh
0184 82A8h	MAR170	Memory Attribute Register 170	AA00 0000h - AAFF FFFFh
0184 82ACh	MAR171	Memory Attribute Register 171	AB00 0000h - ABFF FFFFh
0184 82B0h	MAR172	Memory Attribute Register 172	AC00 0000h - ACFF FFFFh
0184 82B4h	MAR173	Memory Attribute Register 173	AD00 0000h - ADFF FFFFh
0184 82B8h	MAR174	Memory Attribute Register 174	AE00 0000h - AEFF FFFFh
0184 82BCh	MAR175	Memory Attribute Register 175	AF00 0000h - AFFF FFFFh
0184 82C0h	MAR176	Memory Attribute Register 176	B000 0000h - B0FF FFFFh
0184 82C4h	MAR177	Memory Attribute Register 177	B100 0000h - B1FF FFFFh
0184 82C8h	MAR178	Memory Attribute Register 178	B20 00000h - B2FF FFFFh
0184 82CCh	MAR179	Memory Attribute Register 179	B300 0000h - B3FF FFFFh
0184 82D0h	MAR180	Memory Attribute Register 180	B400 0000h - B4FF FFFFh
0184 82D4h	MAR181	Memory Attribute Register 181	B500 0000h - B5FF FFFFh

Table 4-21. Memory Attribute Registers (continued)

Address	Acronym	Register Description	Defines Attributes for. . .
0184 82D8h	MAR182	Memory Attribute Register 182	B600 0000h - B6FF FFFFh
0184 82DCh	MAR183	Memory Attribute Register 183	B700 0000h - B7FF FFFFh
0184 82E0h	MAR184	Memory Attribute Register 184	B800 0000h - B8FF FFFFh
0184 82E4h	MAR185	Memory Attribute Register 185	B900 0000h - B9FF FFFFh
0184 82E8h	MAR186	Memory Attribute Register 186	BA00 0000h - BAFF FFFFh
0184 82ECh	MAR187	Memory Attribute Register 187	BB00 0000h - BBFF FFFFh
0184 82F0h	MAR188	Memory Attribute Register 188	BC00 0000h - BCFF FFFFh
0184 82F4h	MAR189	Memory Attribute Register 189	BD00 0000h - BDFF FFFFh
0184 82F8h	MAR190	Memory Attribute Register 190	BE00 0000h - BEFF FFFFh
0184 82FCh	MAR191	Memory Attribute Register 191	BF00 0000h - BFFF FFFFh
0184 8300h	MAR192	Memory Attribute Register 192	C000 0000h - C0FF FFFFh
0184 8304h	MAR193	Memory Attribute Register 193	C100 0000h - C1FF FFFFh
0184 8308h	MAR194	Memory Attribute Register 194	C200 0000h - C2FF FFFFh
0184 830Ch	MAR195	Memory Attribute Register 195	C300 0000h - C3FF FFFFh
0184 8310h	MAR196	Memory Attribute Register 196	C400 0000h - C4FF FFFFh
0184 8314h	MAR197	Memory Attribute Register 197	C500 0000h - C5FF FFFFh
0184 8318h	MAR198	Memory Attribute Register 198	C600 0000h - C6FF FFFFh
0184 831Ch	MAR199	Memory Attribute Register 199	C700 0000h - C7FF FFFFh
0184 8320h	MAR200	Memory Attribute Register 200	C800 0000h - C8FF FFFFh
0184 8324h	MAR201	Memory Attribute Register 201	C900 0000h - C9FF FFFFh
0184 8328h	MAR202	Memory Attribute Register 202	CA00 0000h - CAFF FFFFh
0184 832Ch	MAR203	Memory Attribute Register 203	CB00 0000h - CBFF FFFFh
0184 8330h	MAR204	Memory Attribute Register 204	CC00 0000h - CCFF FFFFh
0184 8334h	MAR205	Memory Attribute Register 205	CD00 0000h - CDFF FFFFh
0184 8338h	MAR206	Memory Attribute Register 206	CE00 0000h - CEFF FFFFh
0184 833Ch	MAR207	Memory Attribute Register 207	CF00 0000h - CFFF FFFFh
0184 8340h	MAR208	Memory Attribute Register 208	D000 0000h - D0FF FFFFh
0184 8344h	MAR209	Memory Attribute Register 209	D100 0000h - D1FF FFFFh
0184 8348h	MAR210	Memory Attribute Register 210	D200 0000h - D2FF FFFFh
0184 834Ch	MAR211	Memory Attribute Register 211	D300 0000h - D3FF FFFFh
0184 8350h	MAR212	Memory Attribute Register 212	D400 0000h - D4FF FFFFh
0184 8354h	MAR213	Memory Attribute Register 213	D500 0000h - D5FF FFFFh
0184 8358h	MAR214	Memory Attribute Register 214	D600 0000h - D6FF FFFFh
0184 835Ch	MAR215	Memory Attribute Register 215	D700 0000h - D7FF FFFFh
0184 8360h	MAR216	Memory Attribute Register 216	D800 0000h - D8FF FFFFh
0184 8364h	MAR217	Memory Attribute Register 217	D900 0000h - D9FF FFFFh
0184 8368h	MAR218	Memory Attribute Register 218	DA00 0000h - DAFF FFFFh
0184 836Ch	MAR219	Memory Attribute Register 219	DB00 0000h - DBFF FFFFh
0184 8370h	MAR220	Memory Attribute Register 220	DC00 0000h - DCFF FFFFh
0184 8374h	MAR221	Memory Attribute Register 221	DD00 0000h - DDFF FFFFh
0184 8378h	MAR222	Memory Attribute Register 222	DE00 0000h - DEFF FFFFh
0184 837Ch	MAR223	Memory Attribute Register 223	DF00 0000h - DFFF FFFFh
0184 8380h	MAR224	Memory Attribute Register 224	E000 0000h - E0FF FFFFh
0184 8384h	MAR225	Memory Attribute Register 225	E10 00000h - E1FF FFFFh
0184 8388h	MAR226	Memory Attribute Register 226	E200 0000h - E2FF FFFFh
0184 838Ch	MAR227	Memory Attribute Register 227	E300 0000h - E3FFF FFFh
0184 8390h	MAR228	Memory Attribute Register 228	E400 0000h - E4FF FFFFh

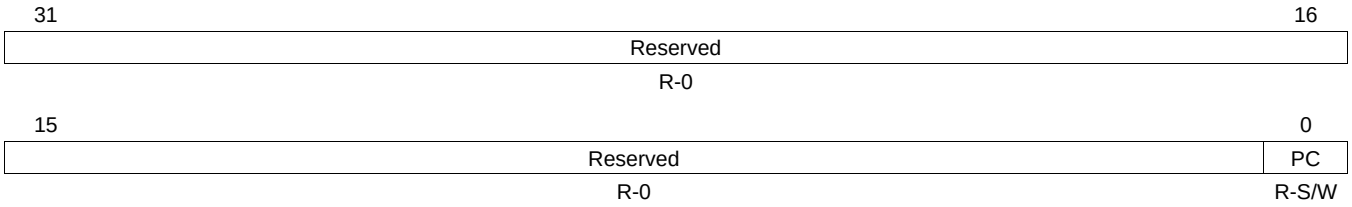
Table 4-21. Memory Attribute Registers (continued)

Address	Acronym	Register Description	Defines Attributes for. . .
0184 8394h	MAR229	Memory Attribute Register 229	E500 0000h - E5FF FFFFh
0184 8398h	MAR230	Memory Attribute Register 230	E600 0000h - E6FF FFFFh
0184 839Ch	MAR231	Memory Attribute Register 231	E700 0000h - E7FF FFFFh
0184 83A0h	MAR232	Memory Attribute Register 232	E800 0000h - E8FF FFFFh
0184 83A4h	MAR233	Memory Attribute Register 233	E900 0000h - E9FF FFFFh
0184 83A8h	MAR234	Memory Attribute Register 234	EA00 0000h - EAFF FFFFh
0184 83ACh	MAR235	Memory Attribute Register 235	EB00 0000h - EBFF FFFFh
0184 83B0h	MAR236	Memory Attribute Register 236	EC00 0000h - ECFF FFFFh
0184 83B4h	MAR237	Memory Attribute Register 237	ED00 0000h - EDFF FFFFh
0184 83B8h	MAR238	Memory Attribute Register 238	EE00 0000h - EEFF FFFFh
0184 83BCh	MAR239	Memory Attribute Register 239	EF00 0000h - EFFF FFFFh
0184 83C0h	MAR240	Memory Attribute Register 240	F000 0000h - F0FF FFFFh
0184 83C4h	MAR241	Memory Attribute Register 241	F100 0000h - F1FF FFFFh
0184 83C8h	MAR242	Memory Attribute Register 242	F200 0000h - F2FF FFFFh
0184 83CCh	MAR243	Memory Attribute Register 243	F300 0000h - F3FF FFFFh
0184 83D0h	MAR244	Memory Attribute Register 244	F400 0000h - F4FF FFFFh
0184 83D4h	MAR245	Memory Attribute Register 245	F500 0000h - F5FF FFFFh
0184 83D8h	MAR246	Memory Attribute Register 246	F600 0000h - F6FF FFFFh
0184 83DCh	MAR247	Memory Attribute Register 247	F700 0000h - F7FF FFFFh
0184 83E0h	MAR248	Memory Attribute Register 248	F800 0000h - F8FF FFFFh
0184 83E4h	MAR249	Memory Attribute Register 249	F900 0000h - F9FF FFFFh
0184 83E8h	MAR250	Memory Attribute Register 250	FA00 0000h - FAFF FFFFh
0184 83ECh	MAR251	Memory Attribute Register 251	FB00 0000h - FBFF FFFFh
0184 83F0h	MAR252	Memory Attribute Register 252	FC00 0000h - FCFF FFFFh
0184 83F4h	MAR253	Memory Attribute Register 253	FD00 0000h - FDFF FFFFh
0184 83F8h	MAR254	Memory Attribute Register 254	FE00 0000h - FEFF FFFFh
0184 83FCh	MAR255	Memory Attribute Register 255	FF00 0000h - FFFF FFFFh

4.4.5 Memory Attribute Registers (MARn)

The general structure of the L2 memory attribute register (MAR*n*) is shown in Figure 4-12 and described in Table 4-22.

Figure 4-12. Memory Attribute Register (MAR*n*)



LEGEND: R = Read only; -*n* = value after reset; R/SW = Read/Writeable by supervisor only

Table 4-22. Memory Attribute Register (MAR*n*) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	PC		Permit copies field enables/disables the cacheability of the affected address range.
		0	Memory range not cacheable.
		1	Memory range cacheable.

4.4.6 Privilege and Cache Control Registers

The L2 memory architecture provides memory protection support. The L2 memory protection architecture is described in more detail in Section 4.6.

Table 4-23 summarizes which L2 cache control registers are accessible according to role.

Table 4-23. Permissions for L2 Cache Control Registers

Register	Supervisor	User
L2CFG	R/W	R
L2INV	R/W	R
L2WB	R/W	R/W
L2WBINV	R/W	R/W
L2WBAR/WC	R/W	R/W
L2WIBAR/WC	R/W	R/W
L2IBAR/WC	R/W	R/W
MARxx	R/W	R

4.5 L2 Power-Down

The C64x+ megamodule architecture provides several power-down features. Some power-down features are transparent. Others are controlled through software. The user-controlled power-down features can be divided into two groups: dynamic and static. Dynamic power-down features are used at run-time for a limited period of time, whereas static power-down features are used for a longer period of time when the CPU is in idle mode. These power-down features are controlled through registers that are local to the specific module or part of the power-down controller (PDC). Read Chapter 9 prior to reading this section in order to understand this section better.

The L2 memory architecture provides support for dynamically powering-down portions of memory attached to Port 0 and Port 1 while a program is active. Programs can put pages of L2 memory to sleep and subsequently wake them manually. The C64x+ will also wake sleeping pages automatically when programs access them while sleeping.

4.5.1 L2 Memory Dynamic Power-Down

L2 memory is divided into four logical pages-two for each port-that can be powered independently. The power-of-2 size of the logical pages is device-specific. The power-of-2 size of the logical pages is half the power-of-2 size of the memory attached to each L2 port. Devices that implement a non-power-of-2 memory size on a given L2 port will have one page larger than the other. Refer to the device-specific data manual for more information.

The dynamic power-down features are programmable throughout a set of registers. Table 4-24 provides a summary. These registers are mentioned through the next sections and described in more detail in Section 4.5.3.

Table 4-24. L2 Memory Power-Down Register Summary

Address	Acronym	Register Description	Section
0184 C040h	L2PDWAKE0	Level 2 Power Down Wake Register 0	Section 4.5.3.1
0184 C044h	L2PDWAKE1	Level 2 Power Down Wake Register 1	Section 4.5.3.1
0184 C050h	L2PDSLEEP0	Level 2 Power Down Sleep Register 0	Section 4.5.3.2
0184 C054h	L2PDSLEEP1	Level 2 Power Down Sleep Register 1	Section 4.5.3.2
0184 C060h	L2PDSTAT0	Level 2 Power Down Status Register 0	Section 4.5.3.3
0184 C064h	L2PDSTAT1	Level 2 Power Down Status Register 1	Section 4.5.3.3

4.5.1.1 Manual Sleep Control

Programs can put certain logical memory pages to sleep, or wake-up certain logical pages. This allows the program to manually control what portions of L2 memory are powered-up or sleeping.

4.5.1.2 Wake and Sleep Commands

To put logical pages of L2 memory to sleep, programs write 1s to the page-sleep bits in the appropriate L2PDSLEEP register. For instance, to put 1 to sleep on port 1, the program writes the value 0000 0002 (bits 1 set to 1) to the L2PDSLEEP1 register.

Likewise, to wake logical pages in L2 memory, programs write 1s to the page-wake bits in the appropriate L2PDWAKE register. This procedure of waking up pages enables you to wake up multiple pages ahead of any accesses to those pages. Thus, part of the stall penalty associated with this wake-up process is hidden from the program.

Zero bits written to the L2PDWAKE and L2PDSLEEP registers have no effect on the corresponding logical pages.

A page put to sleep may be awakened by activity to addresses that are within that page. The page does not go back to sleep unless another L2PDSLEEP command is issued for it.

The L2 controller may drop sleep requests for a page if it is currently servicing accesses for that page, or if other requests for that page arrive within a short time window of receiving the sleep request. This prevents L2 from storing the sleep request until all outstanding requests are complete. Therefore, programs should check the L2PDSTATx register to determine whether or not a given sleep request was honored.

4.5.1.3 Power-Down Status Reporting

The L2PDSTAT0 and L2PDSTAT1 memory-mapped registers report the current sleep-status of each L2 memory page. These registers allow an application to determine which pages are placed to sleep. The registers report sleep status for both logical pages on each L2 port.

4.5.2 L2 Memory Static Power-Down

You can power-down the L2 memory at the same time that you power-down the entire megamodule. The L2 memory can also be powered-down when the entire megamodule is powered-down. The following software sequence is required to power-down the C64x+ megamodule:

1. Set the MEGPD field in the PDCCMD register to 1 to enable power-down mode.
2. Enable the CPU interrupt(s) that you want to wake the megamodule up; disable all others.
3. Execute an IDLE instruction.

The megamodule stays in powered-down until awakened by the interrupt(s) enabled in step 2, above.

If a DMA access occurs to the L1D, L1P, or L2 memory while the megamodule is powered-down, the PDC wakes all three memory controllers. When the DMA access has been serviced, the PDC will again power-down the memory controllers.

Note: Powering-down the megamodule as described here is often called static power-down. This term is used to describe this mode since it is often used for longer periods of time. The use of dynamic power-down elsewhere in this chapter implies that they are used for limited periods of time.

Refer to Chapter 9 for more information about the PDCCMD register and the power-down capabilities of the C64x+ megamodule.

4.5.3 L2 Power-Down Control Registers

Table 4-25 provides a summary of the L2 memory power-down registers.

Table 4-25. L2 Memory Power-Down Register Summary

Address	Acronym	Register Description	Section
0184 C040h	L2PDWAKE0	Level 2 Power Down Wake Register 0	Section 4.5.3.1
0184 C044h	L2PDWAKE1	Level 2 Power Down Wake Register 1	Section 4.5.3.1
0184 C050h	L2PDSLEEP0	Level 2 Power Down Sleep Register 0	Section 4.5.3.2
0184 C054h	L2PDSLEEP1	Level 2 Power Down Sleep Register 1	Section 4.5.3.2
0184 C060h	L2PDSTAT0	Level 2 Power Down Status Register 0	Section 4.5.3.3
0184 C064h	L2PDSTAT1	Level 2 Power Down Status Register 1	Section 4.5.3.3

4.5.3.1 Level 2 Power-Down Wake Register (L2PDWAKE*n*)

The four logical L2 pages can be awakened from a power-down state by programming the registers described in Figure 4-13.

The level 2 power-down wake register (L2PDWAKE*n*) is shown in Figure 4-13 and described in Table 4-26.

Figure 4-13. Level 2 Power-Down Wake Register (L2PDWAKE*n*)

31	16
Reserved	
R-0	

15	2	1	0
Reserved		P1	P0
R-0		W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 4-26. Level 2 Power-Down Wake Register (L2PDWAKE*n*) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1	P1		Wakes-up page 1, port 0 or port 1.
		0	Writing to this field has no effect on page 1.
		1	Writing a 1 to this field wakes-up the page from the power-down state.
0	PO		Wakes-up page 0, port 0 or port 1.
		0	Writing 0 to this field has no effect on page 0.
		1	Writing a 1 to this field wakes-up the page from power-down state.

4.5.3.2 Level 2 Power-Down Sleep Register (L2PDSLEEP*n*)

The four logical L2 pages can be powered-down by programming the two registers described in Figure 4-14 and Figure 4-13.

The level 2 power-down sleep register (L2PDSLEEP*n*) is shown in Figure 4-14 and described in Table 4-27.

Figure 4-14. Level 2 Power-Down Sleep Register (L2PDSLEEP*n*)

31	16
Reserved	
R-0	

15	2	1	0
Reserved		P1	P0
R-0		W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 4-27. Level 2 Power-Down Sleep Register (L2PDSLEEP*n*) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1	P1		Places page 1, port 0 or port 1 in a power-down state.
		0	Writing 0 to this field has no effect on page 1.
		1	Writing a 1 to this field places the page in power-down state.
0	P0		Places page 0, port 0 or port 1 in a power-down state.
		0	Writing 0 to this field has no effect on page 0.
		1	Writing a 1 to this field places the page in power-down state.

4.5.3.3 Level 2 Power-Down Status Register (L2PDSTAT*n*)

The power-down state of the four L2 pages is reported in two status registers.

The level 2 power-down status register (L2PDSTAT*n*) is shown in Figure 4-15 and described in Table 4-28.

Figure 4-15. Level 2 Power-Down Status Register (L2PDSTAT*n*)

31	16
Reserved	
R-0	

15	2	1	0
Reserved		P1	P0
R-0		R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 4-28. Level 2 Power-Down Status Register (L2PDSTAT*n*) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1	P1		Reports power-down status for page 1, port n.
		0	Page 1 is a normal state.
		1	Page 1 is in a power-down state.
0	P0		Reports power-down status for page 0, port n.
		0	Page 0 is a normal state.
		1	Page 0 is a power-down state.

4.5.3.4 Privilege and L2 Power-Down Control Registers

The impact of privilege on power-down control operations can be summarized as follows:

- Supervisor or user code may power-down and wake-up any of the L2 logical pages.

Table 4-29 summarizes who may access which power-down control registers:

Table 4-29. Permissions for L2 Power-Down Control Registers

Register	Supervisor	User
L2PDWAKE0	R/W	R/W
L2PDWAKE1	R/W	R/W
L2PDSLEEP0	R/W	R/W
L2PDSLEEP1	R/W	R/W
L2PDSTAT0	R	R
L2PDSTAT1	R	R

4.6 L2 Memory Protection

L2 memory supports memory protection to offer the robustness required in many systems. Several levels of memory protection are available. Not all the levels are available on all the devices. Refer to the device-specific data manual for more information. Familiarize yourself with Chapter 8 before reading this section.

4.6.1 Protection Checks on CPU, IDMA and Other System Master Accesses

Memory protection checks are performed for accesses that are serviced directly by the L2 from L1P, L1D, IDMA, and other system masters on devices that include memory protection support.

All three memory controllers feature two exception outputs which are routed to the C64x+ megamodule interrupt controller. One of these exception outputs indicates that a CPU-triggered (“local”) memory exception occurred. The other indicates that an exception triggered by a system master occurred. It is expected that most programs route the CPU-triggered exception input to the CPU’s exception input and the system master triggered input to an interrupt input.

L2 does not perform protection checks on CPU reads that arrive in L2, regardless of whether they hit or miss in L2. Reads ultimately return the access permissions to the requestor, thereby deferring the check to L1D or L1P. In contrast, L2 checks all CPU writes that hit L2, or that miss L2 and subsequently allocate a line in the L2 cache. L2 does not check permissions on non-cacheable writes that miss L2. Therefore, L2 checks all CPU accesses that end at L2, and defers checks for other access to the controller (L1P, L1D, or external peripheral) that ultimately services the access.

All system masters and IDMA accesses (reads and writes) to L2 memory are always checked. System masters and IDMA accesses to addresses held in L2 cache are not checked. L2 (or EMC) performs protection checks before issuing snoop-write commands to L1D for addresses held in L1D cache.

The L2 controller determines whether a given request is allowed or not allowed based on the privilege associated with the request, and the permission settings on the address range that the request accesses. Chapter 8 sets the exact rules for these checks forth.

L2 asserts an exception and denies the request if a given request has insufficient permission. Reads that are not allowed return garbage and writes that are not allowed are killed before the underlying memory is written. The L2 only permission-checks CPU writes that miss L2 if they are cacheable within L2 or later stages of the memory system if they are not cacheable.

4.6.2 L2 Memory Protection Registers

The following registers govern the operation of L2 memory protection. The registers fall into three main categories:

- Memory Protection Page Attribute (MPPA) registers. These registers store the permissions associated with each protected page.
- Memory Protection Lock (MPLK) registers. These registers implement a hardware memory protection lock. When engaged, the lock disables all updates to the memory protection entries for that peripheral.
- Memory Protection Fault (MPFxR) registers. Each peripheral that generates memory protection faults provides MPFAR, MPFSR, and MPFCLR registers for recording the details of the fault.

4.6.2.1 L2 Memory Protection Registers

Table 4-30 below lists the memory attribute registers.

Table 4-30. L2 Memory Protection Registers

Address	Acronym	Register Description	Section
0184 A2xxh	L2MPPAxx	Level 2 Memory Protection Page Attribute Registers	Section 4.6.2.2
0184 A100h	L2MPLK0	Level 2 Memory Protection Lock Register 0	Section 4.6.2.3.1.1
0184 A104h	L2MPLK1	Level 2 Memory Protection Lock Register 1	Section 4.6.2.3.1.2
0184 A108h	L2MPLK2	Level 2 Memory Protection Lock Register 2	Section 4.6.2.3.1.3
0184 A10Ch	L2MPLK3	Level 2 Memory Protection Lock Register 3	Section 4.6.2.3.1.4
0184 A110h	L2MPLKCMD	Level 2 Memory Protection Lock Command Register	Section 4.6.2.3.1.5
0184 A114h	L2MPLKSTAT	Level 2 Memory Protection Lock Status Register	Section 4.6.2.3.1.6
0184 A000h	L2MPFAR	Level 2 Memory Protection Fault Address Register	Section 4.6.2.4.1
0184 A004h	L2MPFSR	Level 2 Memory Protection Fault Set Register	Section 4.6.2.4.2
0184 A008h	L2MPFCR	Level 2 Memory Protection Fault Clear Register	Section 4.6.2.4.3

4.6.2.2 Memory Protection Page Attribute Registers (L2MPPAxx)

L2 implements 64 memory protection pages, with 32 pages per memory port. L2MPPA0 through L2MPPA31 correspond to port 0 and L2MPPA32 through L2MPPA63 correspond to port 1. The size of each page differs from port to port and from one device to another. Some pages may not be used on a particular device. Program unused pages to a value of all zeroes for debug purposes.

Refer to the device-specific data manual to determine the page size and number of pages used on a particular device.

Each page in L2 has 16 memory protection bits associated with it, as shown in Figure 4-16. The default value of the protection bits in these 64 memory protection pages is determined at reset. Table 4-33 illustrates the default configuration.

Table 4-31 below lists the memory attribute registers.

Table 4-31. Level 2 Memory Protection Page Attribute Registers

Address	Acronym	Register Description	Section
0184 A200h	L2MPPA0	Level 2 Memory Protection Page Attribute Register 0	Section 4.6.2.2.1
0184 A204h	L2MPPA1	Level 2 Memory Protection Page Attribute Register 1	Section 4.6.2.2.1
0184 A208h	L2MPPA2	Level 2 Memory Protection Page Attribute Register 2	Section 4.6.2.2.1
0184 A20Ch	L2MPPA3	Level 2 Memory Protection Page Attribute Register 3	Section 4.6.2.2.1
0184 A210h	L2MPPA4	Level 2 Memory Protection Page Attribute Register 4	Section 4.6.2.2.1
0184 A214h	L2MPPA5	Level 2 Memory Protection Page Attribute Register 5	Section 4.6.2.2.1
0184 A218h	L2MPPA6	Level 2 Memory Protection Page Attribute Register 6	Section 4.6.2.2.1
0184 A21Ch	L2MPPA7	Level 2 Memory Protection Page Attribute Register 7	Section 4.6.2.2.1
0184 A220h	L2MPPA8	Level 2 Memory Protection Page Attribute Register 8	Section 4.6.2.2.1
0184 A224h	L2MPPA9	Level 2 Memory Protection Page Attribute Register 9	Section 4.6.2.2.1
0184 A228h	L2MPPA10	Level 2 Memory Protection Page Attribute Register 10	Section 4.6.2.2.1
0184 A22Ch	L2MPPA11	Level 2 Memory Protection Page Attribute Register 11	Section 4.6.2.2.1
0184 A230h	L2MPPA12	Level 2 Memory Protection Page Attribute Register 12	Section 4.6.2.2.1
0184 A234h	L2MPPA13	Level 2 Memory Protection Page Attribute Register 13	Section 4.6.2.2.1
0184 A238h	L2MPPA14	Level 2 Memory Protection Page Attribute Register 14	Section 4.6.2.2.1
0184 A23Ch	L2MPPA15	Level 2 Memory Protection Page Attribute Register 15	Section 4.6.2.2.1
0184 A240h	L2MPPA16	Level 2 Memory Protection Page Attribute Register 16	Section 4.6.2.2.1
0184 A244h	L2MPPA17	Level 2 Memory Protection Page Attribute Register 17	Section 4.6.2.2.1
0184 A248h	L2MPPA18	Level 2 Memory Protection Page Attribute Register 18	Section 4.6.2.2.1
0184 A24Ch	L2MPPA19	Level 2 Memory Protection Page Attribute Register 19	Section 4.6.2.2.1
0184 A250h	L2MPPA20	Level 2 Memory Protection Page Attribute Register 20	Section 4.6.2.2.1
0184 A254h	L2MPPA21	Level 2 Memory Protection Page Attribute Register 21	Section 4.6.2.2.1
0184 A258h	L2MPPA22	Level 2 Memory Protection Page Attribute Register 22	Section 4.6.2.2.1
0184 A25Ch	L2MPPA23	Level 2 Memory Protection Page Attribute Register 23	Section 4.6.2.2.1
0184 A260h	L2MPPA24	Level 2 Memory Protection Page Attribute Register 24	Section 4.6.2.2.1
0184 A264h	L2MPPA25	Level 2 Memory Protection Page Attribute Register 25	Section 4.6.2.2.1
0184 A268h	L2MPPA26	Level 2 Memory Protection Page Attribute Register 26	Section 4.6.2.2.1
0184 A26Ch	L2MPPA27	Level 2 Memory Protection Page Attribute Register 27	Section 4.6.2.2.1
0184 A270h	L2MPPA28	Level 2 Memory Protection Page Attribute Register 28	Section 4.6.2.2.1
0184 A274h	L2MPPA29	Level 2 Memory Protection Page Attribute Register 29	Section 4.6.2.2.1
0184 A278h	L2MPPA30	Level 2 Memory Protection Page Attribute Register 30	Section 4.6.2.2.1
0184 A27Ch	L2MPPA31	Level 2 Memory Protection Page Attribute Register 31	Section 4.6.2.2.1
0184 A280h	L2MPPA32	Level 2 Memory Protection Page Attribute Register 32	Section 4.6.2.2.1
0184 A284h	L2MPPA33	Level 2 Memory Protection Page Attribute Register 33	Section 4.6.2.2.1
0184 A288h	L2MPPA34	Level 2 Memory Protection Page Attribute Register 34	Section 4.6.2.2.1

Table 4-31. Level 2 Memory Protection Page Attribute Registers (continued)

Address	Acronym	Register Description	Section
0184 A28Ch	L2MPPA35	Level 2 Memory Protection Page Attribute Register 35	Section 4.6.2.2.1
0184 A290h	L2MPPA36	Level 2 Memory Protection Page Attribute Register 36	Section 4.6.2.2.1
0184 A294h	L2MPPA37	Level 2 Memory Protection Page Attribute Register 37	Section 4.6.2.2.1
0184 A298h	L2MPPA38	Level 2 Memory Protection Page Attribute Register 38	Section 4.6.2.2.1
0184 A29Ch	L2MPPA39	Level 2 Memory Protection Page Attribute Register 39	Section 4.6.2.2.1
0184 A2A0h	L2MPPA40	Level 2 Memory Protection Page Attribute Register 40	Section 4.6.2.2.1
0184 A2A4h	L2MPPA41	Level 2 Memory Protection Page Attribute Register 41	Section 4.6.2.2.1
0184 A2A8h	L2MPPA42	Level 2 Memory Protection Page Attribute Register 42	Section 4.6.2.2.1
0184 A2ACh	L2MPPA43	Level 2 Memory Protection Page Attribute Register 43	Section 4.6.2.2.1
0184 A2B0h	L2MPPA44	Level 2 Memory Protection Page Attribute Register 44	Section 4.6.2.2.1
0184 A2B4h	L2MPPA45	Level 2 Memory Protection Page Attribute Register 45	Section 4.6.2.2.1
0184 A2B8h	L2MPPA46	Level 2 Memory Protection Page Attribute Register 46	Section 4.6.2.2.1
0184 A2BCh	L2MPPA47	Level 2 Memory Protection Page Attribute Register 47	Section 4.6.2.2.1
0184 A2C0h	L2MPPA48	Level 2 Memory Protection Page Attribute Register 48	Section 4.6.2.2.1
0184 A2C4h	L2MPPA49	Level 2 Memory Protection Page Attribute Register 49	Section 4.6.2.2.1
0184 A2C8h	L2MPPA50	Level 2 Memory Protection Page Attribute Register 50	Section 4.6.2.2.1
0184 A2CCh	L2MPPA51	Level 2 Memory Protection Page Attribute Register 51	Section 4.6.2.2.1
0184 A2D0h	L2MPPA52	Level 2 Memory Protection Page Attribute Register 52	Section 4.6.2.2.1
0184 A2D4h	L2MPPA53	Level 2 Memory Protection Page Attribute Register 53	Section 4.6.2.2.1
0184 A2D8h	L2MPPA54	Level 2 Memory Protection Page Attribute Register 54	Section 4.6.2.2.1
0184 A2DCh	L2MPPA55	Level 2 Memory Protection Page Attribute Register 55	Section 4.6.2.2.1
0184 A2E0h	L2MPPA56	Level 2 Memory Protection Page Attribute Register 56	Section 4.6.2.2.1
0184 A2E4h	L2MPPA57	Level 2 Memory Protection Page Attribute Register 57	Section 4.6.2.2.1
0184 A2E8h	L2MPPA58	Level 2 Memory Protection Page Attribute Register 58	Section 4.6.2.2.1
0184 A2ECh	L2MPPA59	Level 2 Memory Protection Page Attribute Register 59	Section 4.6.2.2.1
0184 A2F0h	L2MPPA60	Level 2 Memory Protection Page Attribute Register 60	Section 4.6.2.2.1
0184 A2F4h	L2MPPA61	Level 2 Memory Protection Page Attribute Register 61	Section 4.6.2.2.1
0184 A2F8h	L2MPPA62	Level 2 Memory Protection Page Attribute Register 62	Section 4.6.2.2.1
0184 A2FCh	L2MPPA63	Level 2 Memory Protection Page Attribute Register 63	Section 4.6.2.2.1

4.6.2.2.1 Memory Protection Page Attribute Registers (L2MPPA*n*)

The level 2 memory protection page attribute registers (L2MPPA*n*) are shown in Figure 4-16 and described in Table 4-32.

Figure 4-16. L2 Memory Protection Page Attribute Registers (L2MPPA*n*)

31															16
Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AID5	AID4	AID3	AID2	AID1	AID0	AIDX	LOCAL	Reserved		SR	SW	SX	UR	UW	UX
R/W-config input															

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 4-32. Memory Protection Page Attribute Registers (MPPA*n*) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15	AID5		Controls access from ID = 5.
		0	Access denied.
		1	Access granted.
14	AID4		Controls access from ID = 4.
		0	Access denied.
		1	Access granted.
13	AID3		Controls access from ID = 3.
		0	Access denied.
		1	Access granted.
12	AID2		Controls access from ID = 2.
		0	Access denied.
		1	Access granted.
11	AID1		Controls access from ID = 1.
		0	Access denied.
		1	Access granted.
10	AID0		Controls access from ID = 0.
		0	Access denied.
		1	Access granted.
9	AIDX		Controls access from ID >= 6.
		0	Access denied.
		1	Access granted.
8	LOCAL		Controls access from CPU to local memories (L1/L2)
		0	Access denied.
		1	Access granted.
7-6	Reserved	0	Reserved
5	SR		Supervisor read access type.
		0	Normal operation.
		1	Indicates a supervisor read request.
4	SW		Supervisor write access type.
		0	Normal operation.
		1	Indicates a supervisor write request.

Table 4-32. Memory Protection Page Attribute Registers (MPPA*n*) Field Descriptions (continued)

Bit	Field	Value	Description
3	SX		Supervisor execute access type.
		0	Normal operation.
		1	Indicates a supervisor execute request.
2	UR		User read access type.
		0	Normal operation.
		1	Indicates a user read request.
1	UW		User write access type.
		0	Normal operation.
		1	Indicates a user write request.
0	UX		User execute access type.
		0	Normal operation.
		1	Indicates a user execute request.

Table 4-33. Default Page Attribute Fields

Allowed IDs (Bits 15:8)	Reserved Bits (Bits 7:6)	Access Types (Bits 5:0)
1111 1111	11	111 111

4.6.2.3 Memory Protection Lock Registers

The L2 implements a 64-bit lock register for controlling write access to the memory protection registers. The behavior of these lock registers is defined in Chapter 8.

Table 4-34 below lists the memory protection lock registers.

Table 4-34. Memory Protection Lock Registers

Address	Acronym	Register Description	Section
0184 A100h	L2MPLK0	Level 2 Memory Protection Lock Register 0	Section 4.6.2.3.1.1
0184 A104h	L2MPLK1	Level 2 Memory Protection Lock Register 1	Section 4.6.2.3.1.2
0184 A108h	L2MPLK2	Level 2 Memory Protection Lock Register 2	Section 4.6.2.3.1.3
0184 A10Ch	L2MPLK3	Level 2 Memory Protection Lock Register 3	Section 4.6.2.3.1.4
0184 A110h	L2MPLKCMD	Level 2 Memory Protection Lock Command Register	Section 4.6.2.3.1.5
0184 A114h	L2MPLKSTAT	Level 2 Memory Protection Lock Status Register	Section 4.6.2.3.1.6

*4.6.2.3.1 Level 2 Memory Protection Lock Registers (L2MPLK*n*)*

The level 2 memory protection lock registers (L2MPLK*n*) are shown in Figure 4-17 through Figure 4-21 and described in Table 4-35.

4.6.2.3.1.1 Level 2 Memory Protection Lock 0 Register (L2MPLK0)

The level 2 memory protection lock 0 register (L2MPLK0) is shown in Figure 4-17.

Figure 4-17. Level 2 Memory Protection Lock 0 Register (L2MPLK0)

31	0
Lock Bits 31:0	
W-x	

LEGEND: W = Write only; -*x* = value is indeterminate, see your device-specific data manual

4.6.2.3.1.2 Level 2 Memory Protection Lock 1 Register (L2MPLK1)

The level 2 memory protection lock register 1 (L2MPLK1) is shown in Figure 4-18.

Figure 4-18. Level 2 Memory Protection Lock 1 Register (L2MPLK1)

31	0
Lock Bits 63:32	
W-x	

LEGEND: W = Write only; -*x* = value is indeterminate, see your device-specific data manual

4.6.2.3.1.3 Level 2 Memory Protection Lock 2 Register (L2MPLK2)

The level 2 memory protection lock 2 register (L2MPLK2) is shown in Figure 4-19.

Figure 4-19. Level 2 Memory Protection Lock 2 Register (L2MPLK2)

31	0
Lock Bits 95:64	
W-x	

LEGEND: W = Write only; -*x* = value is indeterminate, see your device-specific data manual

4.6.2.3.1.4 Level 2 Memory Protection Lock 3 Register (L2MPLK3)

The level 2 memory protection lock 3 register (L2MPLK3) is shown in Figure 4-20.

Figure 4-20. Level 2 Memory Protection Lock 3 Register (L2MPLK3)

31	0
Lock Bits 127:96	
W-x	

LEGEND: W = Write only; -*x* = value is indeterminate, see your device-specific data manual

4.6.2.3.1.5 Level 2 Memory Protection Lock Command Register (L2MPLKCMD)

The level 2 memory protection lock command register (L2MPLKCMD) is shown in Figure 4-21.

Figure 4-21. Level 2 Memory Protection Lock Command Register (L2MPLKCMD)

31	16
Reserved	
R-0	

15	3	2	1	0
Reserved		KEYR	LOCK	UNLOCK
R-0		W-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 4-35. Level 2 Memory Protection Lock Command Register (L2MPLKCMD) Field Descriptions

Bit	Field	Value	Description
31-3	Reserved	0	Reserved
2	KEYR		Reset status.
		0	No effect.
		1	Reset status.
1	LOCK		Interface to complete a lock sequence.
		0	No effect.
		1	Locks the lock provided that the software executed the sequence correctly.
0	UNLOCK		Interface to complete an unlock sequence.
		0	No effect.
		1	Unlock the lock provided that the software executed the sequence correctly.

4.6.2.3.1.6 Level 2 Memory Protection Lock Status Register (L2MPLKSTAT)

The level 2 memory protection lock status register (L2MPLKSTAT) is shown in Figure 4-22 and described in Table 4-36.

Figure 4-22. Level 2 Memory Protection Lock Status Register (L2MPLKSTAT)

31	16
Reserved	
R-0	

15	1	0
Reserved		LK
R-0		R-0

LEGEND: R = Read only; -*n* = value after reset

Table 4-36. Level 2 Memory Protection Lock Status Register (L2MPLKSTAT) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	LK		Indicates the lock's current status.
		0	Lock is disengaged.
		1	Lock is engaged.

As illustrated above, the memory protection architecture allows for lock sizes up to 128 bits. The L2 implements only a 64-bit lock behind the Lock interface. Thus, the values written to the L2MPLCK2 and the L2MPLCK3 registers are ignored. The behavior of the lock mechanism with respect to this shorter key is defined in Chapter 8.

4.6.2.4 Memory Protection Fault Registers

In order to allow programs to diagnose a memory protection fault after an exception occurs, the L2 implements two registers dedicated to storing information about the fault, and an additional register to allow clearing the fault information.

Table 4-37 below lists the memory attribute registers.

Table 4-37. Memory Protection Fault Registers

Address	Acronym	Register Description	Section
0184 A000h	L2MPFAR	Level 2 Memory Protection Fault Address Register	Section 4.6.2.4.1
0184 A004h	L2MPFSR	Level 2 Memory Protection Fault Set Register	Section 4.6.2.4.2
0184 A008h	L2MPFCR	Level 2 Memory Protection Fault Clear Register	Section 4.6.2.4.3

4.6.2.4.1 Level 2 Memory Protection Fault Address Register (L2MPFAR)

The level 2 memory protection fault address register (L2MPFAR) is shown in Figure 4-23 and described in Table 4-38.

Figure 4-23. Level 2 Memory Protection Fault Address Register (L2MPFAR)

31	0
Fault Address	
R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 4-38. Level 2 Memory Protection Fault Address Register (L2MPFAR) Field Descriptions

Bit	Field	Value	Description
31-0	Fault Address	0-FFFF FFFFh	Fault Address

4.6.2.4.2 Level 2 Memory Protection Fault Set Register (L2MPFSR)

The level 2 memory protection fault set register (L2MPFSR) is shown in Figure 4-24 and described in Table 4-39.

Figure 4-24. Level 2 Memory Protection Fault Set Register (L2MPFSR)

31	16
Reserved	
R-0	

15	9	8	7	6	5	4	3	2	1	0
FID		LOCAL	Reserved		SR	SW	Reserved	UR	UW	Reserved
R-0		R-0	R-0		R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 4-39. Level 2 Memory Protection Fault Set Register (L2MPFSR) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-9	FID	0-7Fh	Bit 6:0 of ID of faulting requestor. If ID is narrower than 7 bits, the remaining bits return 0. If ID is wider than 7 bits, the additional bits get truncated. FID = 0 if LOCAL = 1.
8	LOCAL	0	Normal operation.
		1	Access was a "LOCAL" access
7-6	Reserved	0	Reserved
5	SR		Supervisor read access type.
		0	Normal operation.
		1	Indicates a supervisor read request.
4	SW		Supervisor write access type.
		0	Normal operation.
		1	Indicates a supervisor write request.
3	Reserved	0	Reserved
2	UR		User read access type.
		0	Normal operation.
		1	Indicates a user read request.
1	UW		User write access type.
		0	Normal operation.
		1	Indicates a user write request.
0	Reserved	0	Reserved

4.6.2.4.3 Level 2 Memory Protection Fault Clear Register (L2MPFCLR)

The level 2 memory protection fault clear register (L2MPFCLR) is shown in Figure 4-25 and described in Table 4-40.

Figure 4-25. Level 2 Memory Protection Fault Clear Register (L2MPFCLR)

31	16
Reserved	
R-0	

15	1	0
Reserved		MPFCLR
R-0		W-0

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 4-40. Level 2 Memory Protection Fault Clear Register (L2MPFCLR) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	MPFCLR		Command to clear the L2MPFCR register.
		0	No effect.
		1	Clear the L2MPFAR and the L2MPFCR registers.

The memory access protection fault registers in Chapter 8 defines the definition and meanings of these registers.

The L2MPFAR and L2MPFSR registers only store enough information for one fault. Generally, the hardware records the information about the first fault and generates an exception only for that fault. L2 has a notion of “local” (CPU triggered) and “remote” (system masters/IDMA triggered) faults. A “local” fault is allowed to replace a “remote” fault and generate a new exception: this rule can be stated succinctly as: If the LOCAL field of the MPFSR register = 0, and the pending exception sets it to 1, the hardware records the new fault and signals the new exception.

The fault information is held until software clears it by writing a 1 to the MPFCLR field in the L2MPFCR register. There is no effect if software writes a 0 to the MPFCLR field in the L2MPFCR register. L2 ignores the value written to bits 1 through 31 L2MPFCR register.

4.6.3 Protection Checks on Accesses to Memory Protection Registers

L2 implements permission checks on the memory protection registers themselves. The rules are as follows:

- All requestors may read any memory protection (MP) register at any time in all circumstances, except L2MPLK0 through L2MPLK3 (these registers are not readable).
- Supervisor can write all the registers that are writable.

Table 4-41 summarizes which L2 memory protection registers are accessible by role and what protection checks are performed in the megamodule.

Table 4-41. Permissions for L2 Memory Protection Registers

Register	Supervisor	User
L2MPFAR	R	R
L2MPFSR	R	R
L2DMPFCR	W	/
L2DMPLK0	W	/
L2DMPLK1	W	/
L2DMPLK2	W	/
L2DMPLK3	W	/
L2DMPLKCMD	W	/
L2DMPLKSTAT	R	R
L2DMPPAxx	R/W	R

Internal Direct Memory Access (IDMA) Controller

Topic **Page**

5.1 Introduction 126
5.2 Terms and Definitions 126
5.3 IDMA Architecture 127
5.4 Registers 131
5.5 Privilege Levels and IDMA Operation 141

5.1 Introduction

This section provides the purpose and discusses the features of the IDMA controller.

5.1.1 Purpose of the Internal Direct Memory Access (IDMA) Controller

The purpose of the IDMA controller is to perform fast block transfers between any two memory locations local to the C64x+ megamodule. Local memory locations are defined as those in Level 1 program (L1P), Level 1 data (L1D), and Level 2 (L2) memories, or in the external peripheral configuration (CFG) memory. The IDMA cannot transfer data to or from the internal MMR space.

5.1.2 Features

The IDMA controller allows rapid data transfers between all local memories. It provides a fast way to page code and data sections into any memory-mapped RAM local to the megamodule. The key advantage of the IDMA controller is that it allows for transfers between slower (Level 2 - L2) and faster (Level 1 - L1D and L1P) memory. IDMA can provide lower latency than the cache controller since the transfers take place in the background of CPU operation, thereby removing stalls due to cache.

Additionally, the IDMA controller facilitates rapid programming of peripheral configuration registers accessed through the external configuration space (CFG) port of the megamodule. The IDMA controller view of the external configuration space that has a 32-word granularity and allows any register within a 32-word block to be individually accessed.

In summary:

- Optimized for burst transfers of memory blocks (contiguous data).
- Allows access to and from any local memory (L1P, L1D, L2 (pages 0 and 1), and external CFG (but, source and destination cannot both be in CFG). CFG is only accessible to channel 0. No CFG to CFG transfers.
- Indicates transfer completion through programmable interrupts to the CPU.

IDMA controller also provides the ability to do a block fill of memory, where the IDMA controller issues a block of writes using a fill value that you program.

5.2 Terms and Definitions

Refer to Appendix B of this document for a detailed definition of the terms used in this chapter. Appendix B describes general terms used throughout this reference guide.

5.3 IDMA Architecture

The IDMA controller allows both a means to rapidly transfer data between local memories and to rapidly program configuration registers. To fully support this, the IDMA controller consists of two channels, channel 0 and channel 1. The two channels are fully orthogonal to one another allowing concurrent operation.

The operation of the IDMA is controlled through several registers. Table 5-1 provides a summary of these registers. These registers are mentioned throughout this section and are described in more detail in Section 5.4.

Table 5-1. IDMA Register Description

Register	Description
IDMA0_STAT	IDMA0 Status Register
IDMA0_MASK	IDMA0 Mask Register
IDMA0_SOURCE	IDMA0 Source Address Register
IDMA0_DEST	IDMA0 Destination Address Register
IDMA0_COUNT	IDMA0 Block Count Register
IDMA1_STAT	IDMA1 Status Register
IDMA1_SOURCE	IDMA1 Source Address Register
IDMA1_DEST	IDMA1 Destination Address Register
IDMA1_COUNT	IDMA1 Block Count Register

5.3.1 IDMA Channel 0

IDMA channel 0 is intended for quick programming of configuration registers located in the external configuration space (CFG). It transfers data from a local memory (L1P, L1D, and L2) to the external configuration space.

The external configuration space includes the peripheral registers located outside of the megamodule whereas the internal configuration space includes the registers located inside of the megamodule. Any register described in this document belongs to the internal configuration space. For example, the registers that are used to control the level 1 data (L1D) cache are part of the internal configuration space. The internal configuration space is only accessible by the CPU using direct load/store instructions.

IDMA channel 0 can only access the external configuration space. It accesses blocks of 32 contiguous registers at a time. To implement this, IDMA channel 0 has five registers: status, mask, source address, destination address, and block count.

5.3.1.1 IDMA Channel 0 Operation

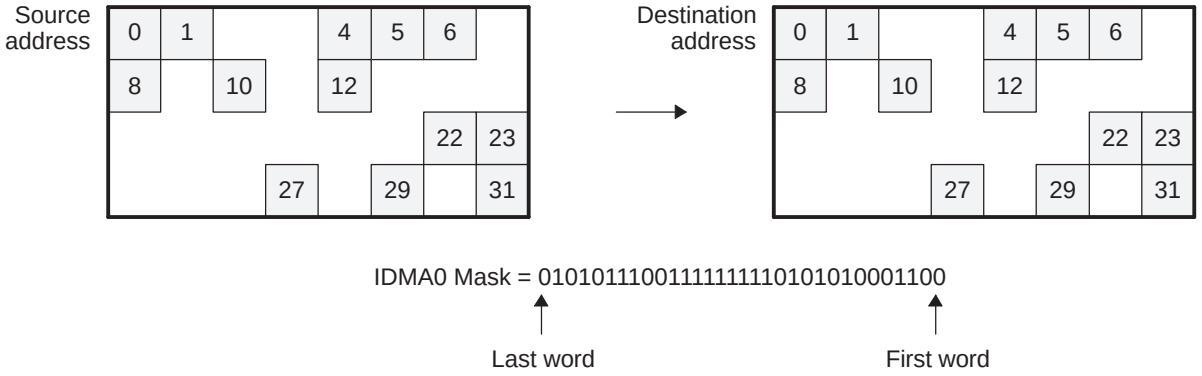
The source and destination addresses that are used for IDMA channel 0 must be 32-byte aligned for proper operation. Figure 5-1 shows one possible transfer using IDMA channel 0.

Define a block of 32 words that contain the values to initialize the CFG registers in a local memory (L1P, L1D, and L2) . Then, the IDMA channel 0 is programmed to transfer these values to the CFG registers.

A mask register is provided since it is not always desirable to program all of the 32 contiguous locations. That is, some locations may be reserved and may not represent actual registers; thus, you should not program them.

The mask register is a 32-bit register. Each bit in this register maps to one of the 32 words in the block that is going to be transferred. For example, bit 0 maps to word 0, bit 1 maps to word 1, etc. If you set the mask bit to 1, then the corresponding word in the block does not transfer.

Figure 5-1. IDMA Channel 0 Transaction



5.3.1.2 IDMA Channel 0 Exception

IDMA channel 0 generates an exception, routed to the C64x+ megamodule interrupt controller, when both the source and the destination addresses are to the CFG.

On the first cycle of operation that the IDMA controller operation is stopped, an exception is generated and any pending IDMA channel 0 requests are then processed. An exception on IDMA channel 0 does not in affect IDMA channel 1 in any way.

5.3.1.3 Programming IDMA Channel 0

IDMA transfers are automatically submitted when the CPU writes to the respective configuration registers. The CPU must write to all of the channel's registers, in sequential incrementing order, for an IDMA transfer to trigger. For channel 0, the CPU should write to the mask in this order: source address, destination address, and then count registers. The submission occurs following the write to the count register.

For each of the IDMA channels, one transfer can be active at any given time. The CPU can update the parameters to queue a subsequent transfer; but, the transfer is not initiated until the active transfer completes. This allows two transfers (active and pending) to be outstanding from the CPU at any given time.

Upon completion of the transfer, a CPU interrupt is optionally set.

5.3.1.3.1 IDMA Channel 0 Example 1

An example of making an update to configuration registers using IDMA channel 0 is shown in the following pseudo-code in Example 5-1.

Example 5-1. Update to Configuration Registers using IDMA Channel 0

```
IDMA0_MASK = 0x00000F0F          ; //Set mask for 8 regs -- 11:8, 3:0
IDMA0_SOURCE = MMR_ADDRESS       ; //Set source to config location
IDMA0_DEST = reg_ptr             ; //Set destination to data memory address
IDMA0_COUNT = 0                  ; //Set mask for 1 block

while (IDMA0_STATUS)             ; //Wait for transfer completion

 ... update register values ...

IDMA0_MASK = 0x00000F0F          ; //Set mask for 8 regs -- 11:8, 3:0
IDMA0_SOURCE = reg_ptr           ; //Set source to updated value pointer
IDMA0_DEST = MMR_ADDRESS         ; //Set destination to config location
IDMA0_COUNT = 0                  ; //Set mask for 1 block
```

5.3.1.3.2 IDMA Channel 0 Example 2

EDMA is a peripheral commonly available on C64x+ devices. An example of submitting multiple QDMA requests is illustrated in Example 5-2. There are sixteen 8-word locations in configuration space within the EDMA that can correspond to the QDMA. Each QDMA submits a transfer request, as defined by an 8-word parameter entry.

Example 5-2 shows how 32 QDMAs can be issued, modifying only the source address, destination address, and options for each QDMA, as is possible in a video application.

Refer to the EDMA documentation for more information about the QDMA.

Example 5-2. Update to 32 QDMAs using IDMA Channel 0

```
IDMA0_MASK = 0x4F4F4F4F            ; //Set mask for 0, 1, 2, 3, 6 for each QDMA
IDMA0_SOURCE = &qdma_list[0]       ; //Set source to transfer list
IDMA0_DEST = &QDMA[0]              ; //Set destination to base address of QDMAs
IDMA0_COUNT = 3                    ; //Set mask for 4 blocks (16 QDMAs)

IDMA0_MASK = 0x4F4F4F4F            ; //Set mask for 0, 1, 2, 3, 6 for each QDMA
IDMA0_SOURCE = &qdma_list[16]      ; //Set source to second half of transfer list
IDMA0_DEST = &QDMA[0]              ; //Set destination to base address of QDMAs
IDMA0_COUNT = 3                    ; //Set mask for 4 blocks (16 QDMAs)

while (IDMA0_STATUS)               ; //Wait for transfer completion
```

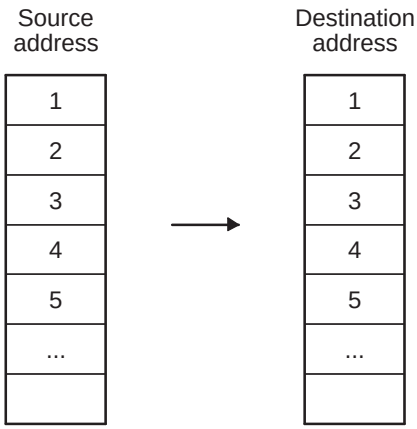
5.3.2 IDMA Channel 1

IDMA channel 1 is intended for transferring data between local memories. It moves data and program sections in the background without CPU operation to set up processing from fast memory. To allow this, IDMA channel 1 has four registers: status, source address, destination address, and count.

5.3.2.1 IDMA Channel 1 Operation

All source and destination addresses increment linearly throughout the transfer. The size (in bytes) of the transfer is set by the COUNT field in the IDMA channel 1 count register (IDMA1_COUNT). Following the transfer, a CPU interrupt is optionally set. Arbitration during any conflicts with the cache or EDMA is based on the priority set in the options field of the count register. Figure 5-2 shows a transfer using IDMA channel 1.

Figure 5-2. IDMA Channel 1 Transaction



5.3.2.2 Programming IDMA Channel 1

IDMA transfers are automatically submitted when the CPU writes to the respective configuration registers. The CPU must write to all of the channel's registers, in sequential incrementing order, for an IDMA transfer to trigger. For channel 1, the CPU should write to the source address, destination address, then to the count registers (in that order). The submission occurs following the write to the count register.

For each of the IDMA channels, one transfer can be active at any given time. The CPU can update the parameters to queue a subsequent transfer, but the transfer does not initiate until the active transfer completes. This allows two transfers per channel to be outstanding from the CPU at any given time.

5.3.2.2.1 IDMA Channel 1 Example

Example pseudo-code for paging in new data and paging out old data using IDMA channel 1 is shown in Example 5-3.

Example 5-3. Paging In New Data and Paging Out Old Data Using IDMA Channel 1

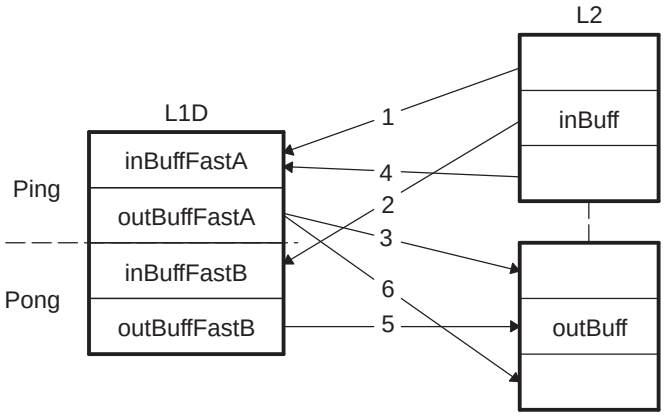
```
//Transfer ping buffers to/from L1D
//Return output buffer n - 1 to slow memory
IDMA1_SOURCE = outBuffFastA                  ; //Set source to fast memory output (L1D)
IDMA1_DEST = &outBuff[n-1]                   ; //Set destination to output buffer (L2)
IDMA1_COUNT = 7 << IDMA_PRI_SHIFT |            //Set priority to low
              0 << IDMA_INT_SHIFT |            //Do not interrupt CPU
              buffsize                       ; //Set count to buffer size

//Page in input buffer n + 1 to fast memory
IDMA1_SOURCE = inBuff[n+1]                   ; //Set source to buffer location (L2)
IDMA1_DEST = inBuffFastA                     ; //Set destination to fast memory (L1D)
IDMA1_COUNT = 7 << IDMA_PRI_SHIFT |            //Set priority to low
              1 << IDMA_INT_SHIFT |            //Interrupt CPU on completion
              buffsize                       ; //Set count to buffer size

... Process input buffer n in Pong -- inBuffFastB -> outBuffFastB ...
```

This example depicts using the IDMA return output data at its location in memory and to page in new data to fast memory for processing, as shown in Figure 5-3.

Figure 5-3. Example of IDMA Channel 1



5.3.2.2.2 Using IDMA Channel 1 to Perform Memory Fill

You can use the IDMA channel 1 to fill a section of a local memory with a specific value. Set the FILL field in the IDMA1_COUNT register to 1 to accomplish this. When the FILL field is set to 1, the value contained in the IDMA1 source address register is used as the fill value. This value is copied to the memory buffer that the IDMA1 destination address register points to. The COUNT field in the IDMA1_COUNT register defines the number of times that this value is copied.

5.4 Registers

The IDMA controller is programmed through a set of registers, listed in Table 5-2. See the device-specific data manual for the memory address of these registers.

Each of the registers is accessible for read/write access by the CPU. Access to each of the IDMA registers must be 32-bit aligned. Half word and byte writes to the IDMA registers write the *entire* register, and thus you should avoid them for proper operation. Nonaligned word and double word accesses result in unpredictable operation, and so you should avoid them as well.

Table 5-2 lists all of the registers in the IDMA.

Table 5-2. Internal Direct Memory Access (IDMA) Registers

Address	Acronym	Register Description	Section
0182 0000h	IDMA0_STAT	IDMA Channel 0 Status Register	Section 5.4.1
0182 0004h	IDMA0_MASK	IDMA Channel 0 Mask Register	Section 5.4.2
0182 0008h	IDMA0_SOURCE	IDMA Channel 0 Source Address Register	Section 5.4.3
0182 000Ch	IDMA0_DEST	IDMA Channel 0 Destination Address Register	Section 5.4.4
0182 0010h	IDMA0_COUNT	IDMA Channel 0 Block Count Register	Section 5.4.5
0182 0100h	IDMA1_STAT	IDMA Channel 1 Status Register	Section 5.4.6
0182 0108h	IDMA1_SOURCE	IDMA Channel 1 Source Address Register	Section 5.4.7
0182 010Ch	IDMA1_DEST	IDMA Channel 1 Destination Address Register	Section 5.4.8
0182 0110h	IDMA1_COUNT	IDMA Channel 1 Block Count Register	Section 5.4.9

5.4.1 IDMA Channel 0 Status Register (IDMA0_STAT)

The IDMA channel 0 status register (IDMA0_STAT) provides the activity state of the channel. There are two bits to denote whether a transfer is in progress (ACTV) and whether a transfer is pending (PEND).

The IDMA channel 0 status register (IDMA0_STAT) is shown in Figure 5-4 and described in Table 5-3.

Figure 5-4. IDMA Channel 0 Status Register (IDMA0_STAT)

31	16
Reserved	
R-0	

15	2	1	0
Reserved		PEND	ACTV
R-0		R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 5-3. IDMA Channel 0 Status Register (IDMA0_STAT) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	These reserved bit locations are always read as zeros. A value written to this field has no effect.
1	PEND		Pending transfer. The PEND bit sets when the CPU writes control registers and an active transfer is already in progress (ACTV = 1). The PEND bit clears when the transfer becomes active.
		0	No pending transfer
		1	Transfer is pending
0	ACTV		Active transfer. The ACTV bit sets when channel 0 begins reading data from the source address register (IDMA0_SOURCE) and clears following the last write to the destination address register (IDMA0_DEST).
		0	No active transfer
		1	Active transfer

5.4.2 IDMA Channel 0 Mask Register (IDMA0_MASK)

The IDMA channel 0 mask register (IDMA0_MASK) allows unwanted registers within the transfer block to be masked. There are 32 bits that allow individual control over the registers within the 32-word memory block identified by the source/destination address registers.

The IDMA channel 0 mask register (IDMA0_MASK) is shown in Figure 5-5 and described in Table 5-4.

Figure 5-5. IDMA Channel 0 Mask Register (IDMA0_MASK)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
M31	M30	M29	M28	M27	M26	M25	M24	M23	M22	M21	M20	M19	M18	M17	M16
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M15	M14	M13	M12	M11	M10	M9	M8	M7	M6	M5	M4	M3	M2	M1	M0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; *-n* = value after reset

Table 5-4. IDMA Channel 0 Mask Register (IDMA0_MASK) Field Descriptions

Bit	Field	Value	Description
31-0	M*n*		Register mask bit.
		0	Register access permitted (not masked).
		1	Register access blocked (masked).

5.4.3 IDMA Channel 0 Source Address Register (IDMA0_SOURCE)

The IDMA channel 0 source address register (IDMA0_SOURCE) identifies the source address for the IDMA transfer. The source of the transfer must be local to the C64x+ megamodule, either in L1P, L1D, L2, or CFG. The source address for the transfer must be a local RAM location, if the destination address is to CFG. Conversely, the source address must be to CFG, if the destination address is to a local RAM location. Additionally, the source address must be 32-byte aligned.

The IDMA channel 0 source address register (IDMA0_SOURCE) is shown in Figure 5-6 and described in Table 5-5.

Figure 5-6. IDMA Channel 0 Source Address Register (IDMA0_SOURCE)

31	16
SOURCEADDR	
R/W-0	

15	5	4	0
SOURCEADDR		Reserved	
R/W-0		R-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 5-5. IDMA Channel 0 Source Address Register (IDMA0_SOURCE) Field Descriptions

Bit	Field	Value	Description
31-5	SOURCEADDR	0-7FF FFFFh	Source address. Must point to a 32-byte aligned (for example, block-aligned) memory location local to the megamodule or to a valid configuration register space.
4-0	Reserved	0	Reserved

5.4.4 IDMA Channel 0 Destination Address Register (IDMA0_DEST)

The IDMA channel 0 destination address register (IDMA0_DEST) identifies the destination address for the IDMA transfer. The destination of the transfer must be local to the C64x+ megamodule, either in L1P, L1D, L2 or CFG. The destination address for the transfer must be a local RAM location, if the source address is to CFG. Conversely, the destination address must be to CFG, if the source address is to a local RAM location. Additionally, the source address must be 32-byte aligned.

The IDMA channel 0 destination address register (IDMA0_DEST) is shown in Figure 5-7 and described in Table 5-6.

Figure 5-7. IDMA Channel 0 Destination Address Register (IDMA0_DEST)

31	16
DESTADDR	
R/W-0	

15	5	4	0
DESTADDR		Reserved	
R/W-0		R-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 5-6. IDMA Channel 0 Destination Address Register (IDMA0_DEST) Field Descriptions

Bit	Field	Value	Description
31-5	DESTADDR	0-7FF FFFFh	Destination address. Must point to a 32-byte (window) aligned memory location local to the megamodule or to a valid configuration register space.
4-0	Reserved	0	Reserved

5.4.5 IDMA Channel 0 Count Register (IDMA0_COUNT)

The IDMA channel 0 count register (IDMA0_COUNT) identifies the number of 32-word blocks that are accessible during the data transfer. The 4-bit COUNT field allows up to 16 blocks to be accessed in succession. The mask field is applied to all blocks, allowing for repeated patterns to be accessed. All blocks are of contiguous 32-word regions and the source and destination addresses increment accordingly. Additionally, th eIDMA0_COUNT register allows a CPU interrupt (IDMA_INT0) to be enabled to notify the CPU that a transfer has completed.

The IDMA channel 0 count register (IDMA0_COUNT) is shown in Figure 5-8 and described in Table 5-7.

Figure 5-8. IDMA Channel 0 Count Register (IDMA0_COUNT)

31 29	28	27 16
Reserved	INT	Reserved
R-0	R/W-0	R-0

15 4	3 0
Reserved	COUNT
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 5-7. IDMA Channel 0 Count Register (IDMA0_COUNT) Field Descriptions

Bit	Field	Value	Description
31-29	Reserved	0	These reserved bit locations are always read as zeros. A value written to this field has no effect.
28	INT		CPU interrupt enable.
		0	Do not interrupt CPU on completion.
		1	Interrupt CPU (IDMA_INT0) on completion.
27-4	Reserved	0	These reserved bit locations are always read as zeros. A value written to this field has no effect.
3-0	COUNT	0-Fh	4-bit block count.
		0	Transfer to/from one 32-word blocks.
		1h-Fh	Transfer to/from n+1 32-word blocks.

5.4.6 IDMA Channel 1 Status Register (IDMA1_STAT)

The IDMA channel 1 status register (IDMA1_STAT) provides the activity state of the channel. There are two bits to denote whether a transfer is in progress (ACTV) and whether a transfer is pending (PEND).

The IDMA channel 1 status register (IDMA1_STAT) is shown in Figure 5-9 and described in Table 5-8.

Figure 5-9. IDMA Channel 1 Status Register (IDMA1_STAT)

31	16
Reserved	
R-0	

15	2	1	0
Reserved		PEND	ACTV
R-0		R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 5-8. IDMA Channel 1 Status Register (IDMA1_STAT) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	These reserved bit locations are always read as zeros. A value written to this field has no effect.
1	PEND		Pending transfer. Set when control registers are written to the CPU and there is already an active transfer in progress (ACTV = 1) and cleared when the transfer becomes active.
		0	No pending transfer.
		1	Transfer is pending.
0	ACTV		Active transfer. ACTV is set when channel 0 begins reading data from the source address register (IDMA1_SOURCE) and is cleared following the last write to the destination address register (IDMA1_DEST).
		0	No active transfer.
		1	Active transfer.

5.4.7 IDMA Channel 1 Source Address Register (IDMA1_SOURCE)

The IDMA channel 1 source address register (IDMA1_SOURCE) identifies the source address for the IDMA transfer. The source of the transfer must be local to the C64x+ megamodule, either in L1P, L1D, L2 or CFG. The source address must also be to a different port than the destination address (L2 port 0 and L2 port 1 are considered the same port) to obtain full throughput of 256 bits per EMC cycle.

If performing a block fill (FILL = 1 in IDMA1_COUNT) rather than a data transfer, IDMA1_SOURCE is used to program the fill value. Rather than reading from a source address, the IDMA transfers the programmed value to all locations in the destination buffer.

The IDMA channel 1 source address register (IDMA1_SOURCE) is shown in Figure 5-10 and described in Table 5-9.

Figure 5-10. IDMA Channel 1 Source Address Register (IDMA1_SOURCE)

31	16
SOURCEADDR	
R/W-0	

15	0
SOURCEADDR	
R/W-0	

LEGEND: R/W = Read/Write; *-n* = value after reset

Table 5-9. IDMA Channel 1 Source Address Register (IDMA1_SOURCE) Field Descriptions

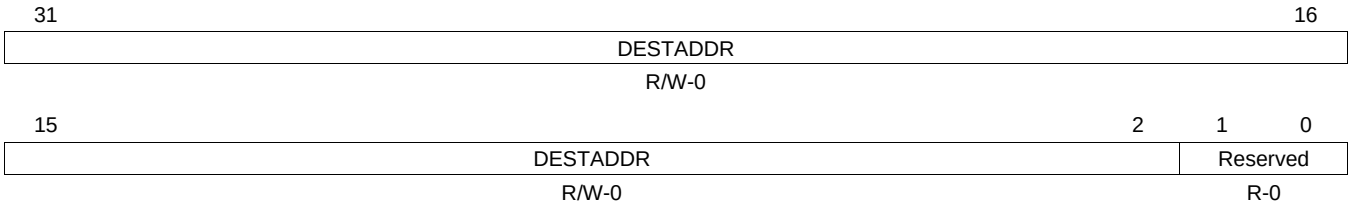
Bit	Field	Value	Description
31-0	SOURCEADDR	0-FFFF FFFFh	Source address. Must point to a word-aligned memory location local to the megamodule. When performing a block fill (FILL = 1 in IDMA1_COUNT), the source address is the fill value. Note that when performing a fill mode transfer, all 32-bits of the SOURCEADDR field are used when performing a memory transfer, the two LSBs are implemented as 00b.

5.4.8 IDMA Channel 1 Destination Address Register (IDMA1_DEST)

The IDMA channel 1 destination address register (IDMA1_DEST) identifies the destination address for the IDMA transfer. The destination of the transfer must be local to the C64x+ megamodule, either in L1P, L1D, L2 or CFG. The destination address must also be to a different port than the source address (L2 port 0 and L2 port 1 are considered the same port) to obtain full throughput of 256 bits per EMC cycle.

The IDMA channel 1 destination address register (IDMA1_DEST) is shown in Figure 5-11 and described in Table 5-10.

Figure 5-11. IDMA Channel 1 Destination Address Register (IDMA1_DEST)



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 5-10. IDMA Channel 1 Destination Address Register (IDMA1_DEST) Field Descriptions

Bit	Field	Value	Description
31-2	DESTADDR	0-3FFF FFFFh	Destination address. Must point to a word-aligned memory location local to the megamodule.
1-0	Reserved	0	Reserved

5.4.9 IDMA Channel 1 Count Register (IDMA1_COUNT)

The IDMA channel 1 count register (IDMA1_COUNT) identifies the transfer length in bytes. In addition, IDMA1_COUNT allows a CPU interrupt (IDMA_INT1) to be enabled and identifies the priority level (relative to CPU and other DMA accesses) to be specified.

The IDMA channel 1 count register (IDMA1_COUNT) is shown in Figure 5-12 and described in Table 5-11.

Figure 5-12. IDMA Channel 1 Count Register (IDMA1_COUNT)

31 29	28	27 17	16
PRI	INT	Reserved	FILL
R/W-0	R/W-0	R-0	R/W-0

15 0
COUNT
R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 5-11. IDMA Channel 1 Count Register (IDMA1_COUNT) Field Descriptions

Bit	Field	Value	Description
31-29	PRI	0-7h	Transfer priority. Used for arbitration between CPU and DMA accesses when there are conflicts. Note that priority can be any value between 0 (highest priority) and 7 (lowest priority).
28	INT		CPU interrupt enable.
		0	Do not interrupt CPU on completion.
		1	Interrupt CPU (IDMA_INT1) on completion.
27-17	Reserved	0	These reserved bit locations are always read as zeros. A value written to this field has no effect.
16	FILL		Block fill
		0	Block transfer from the source address register (IDMA1_SOURCE) to the destination address register (IDMA1_DEST).
		1	Perform a block fill using the source address register (IDMA1_SOURCE) as the fill value to the memory buffer pointed to by the destination address register (IDMA1_DEST).
15-0	COUNT	0-FFh	Byte count. A 16-bit count that defines the transfer length in bytes. Must be a multiple of 4 bytes. A transfer count of zero will not transfer any data, but generates an interrupt if requested by the INT bit. For correct operation, the two ISBs must always be 0.

A transfer count of zero is a possible programming option. The IDMA engine handles a count of zero by "completing" the transfer immediately (i.e., if the INT bit in the IDMA1_COUNT register asserts IDMA_INT1 to the CPU even though no data actually transfers). If a subsequent transfer is pending, it begins immediately.

5.5 Privilege Levels and IDMA Operation

Table 5-12 summarizes which who IDMA registers are accessible and what protection checks are performed in the megamodule based on role.

Table 5-12. Permissions for IDMA Registers

Register	Supervisor	User
IDMA0_STAT	R	R
IDMA0_MASK	R/W	R/W
IDMA0_SOURCE	R/W	R/W
IDMA0_DEST	R/W	R/W
IDMA0_COUNT	R/W	R/W
IDMA1_STAT	R	R
IDMA1_SOURCE	R/W	R/W
IDMA1_DEST	R/W	R/W
IDMA1_COUNT	R/W	R/W

Bandwidth Management Architecture

Topic **Page**

6.1 Introduction 144
6.2 Architecture 145
6.3 Registers 146
6.4 Privilege and Bandwidth Management Registers 153

6.1 Introduction

6.1.1 Purpose of the Bandwidth Management

The purpose of the bandwidth management is to assure that some of the requestors do not block the resources that are available in the C64x+ megamodule for extended periods of time.

Similar to the memory protection capability of the C64x+ CPU, bandwidth management (BWM) is defined globally (for the entire C64x+ megamodule), but implemented locally by each C64x+ megamodule resource. To this end, initializing bandwidth management consists of programming a common set of registers found in each of the C64x+ megamodule's resources.

6.1.2 Resource Bandwidth Protected by Bandwidth Management

The BWM control hardware manages the following four resources:

- Level 1 program (L1P) SRAM/cache
- Level 1 data (L1D) SRAM/cache
- Level 2 (L2) SRAM/cache
- Memory-mapped registers configuration bus

6.1.3 Requestors Managed by Bandwidth Management

Each of the following are potential requestors for the C64x+ megamodule resources listed in Section 6.1.2:

- CPU-initiated transfers:
  - Data access (for example, load/store)
  - Program access
- Programmable cache coherency operations (for example, writeback):
  - Block-based
  - Global
- Internal DMA (IDMA)-initiated transfers (and resulting coherency operations)
- Externally-initiated slave DMA (SDMA) transfers (and resulting coherency operations)

6.1.4 Terms and Definitions

See Appendix A for the terms and definitions used in this chapter.

6.2 Architecture

The bandwidth management scheme is viewable as weighted-priority-driven bandwidth allocation.

6.2.1 Bandwidth Arbitration via Priority Levels

Each requestor (DMA, IDMA, CPU, etc.) is assigned a priority level on a per-transfer basis. There are a total of 9 priority levels. They are:

Highest	Priority 0
	Priority 1
	Priority 2
	Priority 3
	Priority 4
	Priority 5
	Priority 6
	Priority 7
Lowest	Priority 8

When multiple requestors contend for a single resource, granting access to the highest priority requestor solves the conflict. When the contention occurs for multiple successive cycles, a contention counter guarantees that the lower priority requestor gets access to the resource every 1 out of *n* arbitration cycles, where *n* is programmable by the MAXWAIT bit (described in Section 6.3).

The BWM works by incrementing a contention counter every time a resource request is blocked. When a request is allowed to proceed, the stall count resets to 0. When the stall count reaches the MAXWAIT value, then the lower priority requestor's value sets to -1 and is allowed to perform at least one transfer. (The contention counter is not visible to you).

6.2.2 Priority Level: -1

In addition to the 9 priority levels described previously, the hardware uses a priority level of -1 to represent a transfer whose priority has been increased due to expiration of the contention counter (as explained below), or a transfer that is fixed as the highest priority transfer to a given resource. You cannot program a value of -1 into the BWM arbitration control registers.

6.2.3 Priority Declaration

Use various methods to declare priorities by the requestors as described in Table 6-1. For consistency, the priority values used in BWM arbitration registers are weighted equally with those defined in associated modules (for example, IDMA).

Table 6-1. Priority Declaration Methods

Requestor	Priority Declaration In …
CPU	BWM arbitration register (PRI field)
IDMA	IDMA transfer parameters
SDMA	Dictated by the external system master transfer parameters
User defined cache coherency	Fixed priorities

6.3 Registers

A set of registers called arbitration registers implement the bandwidth management architecture. The registers are implemented in the following blocks: L1D, L2, and extended memory controller (EMC). Table 6-2 lists the registers and their base address.

Table 6-2. Arbitration Registers

Block	Acronym	Register Name	Address	Section
L1P	None	NA	N/A	NA
L1D	CPUARBD	CPU Arbitration Control Register	0184 1040h	Section 6.3.1
	IDMAARBD	IDMA Arbitration Control Register	0184 1044h	Section 6.3.3
	SDMAARBD	Slave DMA Arbitration Control Register	0184 1048h	Section 6.3.4
	UCARBD	User Coherence Arbitration Control Register	0184 104Ch	Section 6.3.2
L2	CPUARBU	CPU Arbitration Control Register	0184 1000h	Section 6.3.1
	IDMAARBU	IDMA Arbitration Control Register	0184 1004h	Section 6.3.3
	SDMAARBU	Slave DMA Arbitration Control Register	0184 1008h	Section 6.3.4
	UCARBU	User Coherence Arbitration Control Register	0184 100Ch	Section 6.3.2
EMC	CPUARBE	CPU Arbitration Control Register	0182 0200h	Section 6.3.1
	IDMAARBE	IDMA Arbitration Control Register	0182 0204h	Section 6.3.3
	SDMAARBE	Slave DMA Arbitration Control Register	0182 0208h	Section 6.3.4
	MDMAARBE	Master DMA Arbitration Control Register	0182 020Ch	Section 6.3.5

Table 6-3 shows no arbitration registers for L1P. Indeed, there are no programmable-bandwidth management registers for the L1P; however, there are fixed-bandwidth management features in the L1P controller.

Notice that there are a set of arbitration registers for each resource. Each register corresponds to a different requestor.

The arbitration registers that belong to the same group (CPU, IDMA, SDMA, UC) have identical default values. They are generalized in Table 6-3 by calling the CPUARB, IDMAARB, SDMAARB, and UCARB registers.

Table 6-3. Arbitration Register Default Values

		Register Bit Default Value		Register Exists In ...			
Acronym	Register Name	PRI	MAXWAIT	L1P	L1D	L2	EMC
CPUARB	CPU Arbitration Control Register	1	16	No	Yes	Yes	Yes
IDMAARB	IDMA Arbitration Control Register	NA	16	No	Yes	Yes	Yes
SDMAARB	Slave DMA Arbitration Control Register	NA	1	No	Yes	Yes	Yes
UCARB	User Coherence Arbitration Control Register	NA	32	No	Yes	Yes	No
MDMAARB	Master DMA Arbitration Control Register	7	NA	No	No	No	Yes

The default values of CPUARB, IDMAARB, SDMAARB, and UCARB are sufficient for most applications. These registers define priorities that are internal to the C64x+ megamodule. The MDMAARBE register defines a priority for data transfers outside of the C64x+ megamodule. You may need to change its priority by programming the MDMAARBE register (as described in Section 6.3.5), depending on the system design. In most cases, MDMARBE should be programmed to a higher priority (lower value).

6.3.1 CPU Arbitration Control Register (CPUARBD, CPUARBU, CPUARBE)

The CPU arbitration control register (CPUARBD, CPUARBU, and CPUARBE) controls the bandwidth management of the CPU operations. The CPUARB register is shown in Figure 6-1 and described in Table 6-4. CPU-initiated transfers consist of two components:

1. The CPU issues program fetch transfers to the L1P controller, and the resulting L1P cache coherency operations (such as alloc/evict).
2. The CPU issues data/load store transfers to the L1D controller. The resulting L1D cache coherency operations (such as alloc/evict/long distance accesses) are in turn issued to the L2 controller.

Both program and data requests use CPUARB values to define the maximum wait time (MAXWAIT) and priority (PRI). CPUARB values do not only have an affect local to L1P or L1D. The priority/maximum wait time applied to L1D/L1P cache transactions is programmed at each block. These values are used to control arbitration at each relevant access within the C64x+ megamodule.

Similar to L1D/L1P (via CPUARBD), memory accesses made directly in L2 and EMC blocks (via the CPUARBU and CPUARBE registers, respectively) use the PRI and MAXWAIT field values locally for those blocks, and any further transactions resulting from these requests.

The default value of PRI is set so that the CPU transactions are the second to highest in the system. This should be a relatively typical value used in most systems, resulting in the CPU receiving highest priority most of the time, but, a short-real time deadline peripheral, such as a high speed serial port (that is typically programmed as the highest-priority transfer for SDMA requests) can interrupt the CPU transfers on a nearly immediate basis.

The CPU priority is run-time programmable, although you are expected to initialize the CPUARB registers at system initialization or accept the default values and leave them unchanged thereafter.

Figure 6-1. CPU Arbitration Control Register (CPUARBD, CPUARBU, CPUARBE)

31	19	18	16
Reserved		PRI	
R-0		R/W-1h	

15	6	5	0
Reserved		MAXWAIT	
R-0		R/W-10h	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-4. CPU Arbitration Control Register (CPUARBD, CPUARBU, CPUARBE) Field Descriptions

Bit	Field	Value	Description
31-19	Reserved	0	Reserved
18-16	PRI	0-7h	Priority field. Not all requestors support PRI = 8 (lowest). The PRI field used to make background transfers lower than all other real-time requests.
		0	Priority 0 (highest)
		1h	Priority 1
		2h	Priority 2
		3h	Priority 3
		4h	Priority 4
		5h	Priority 5
		6h	Priority 6
		7h	Priority 7 (lowest)
15-6	Reserved	0	Reserved
5-0	MAXWAIT	0-3Fh	Maximum wait time in EMC cycles. EMC cycle = 2 × CPU cycle.
		0	Always stalls due to higher priority requestor.
		1h	Maximum wait of 1 cycle (1/2 = 50% access)
		2h	Maximum wait of 2 cycles (1/3 = 33% access)
		3h	Reserved
		4h	Maximum wait of 4 cycles (1/5 = 20% access)
		5h-7h	Reserved
		8h	Maximum wait of 8 cycles (1/9 = 11% access)
		9h-Fh	Reserved
		10h	Maximum wait of 16 cycles (1/17 = 6% access)
		11h-1Fh	Reserved
		20h	Maximum wait of 32 cycles (1/33 = 3% access)
		21h-3Fh	Reserved

6.3.2 User Coherence Arbitration Control Register (UCARBD, UCARBU)

The user coherence arbitration control register (UCARBD and UCARBU) controls the bandwidth management of the user coherency operations. These operations consist of cache writeback and invalidate commands specified in a user's program. For more information about user coherency operations, see the Chapter 2, Chapter 3, and Chapter 4.

User coherency operations are broken into two types. They are listed with their fixed priorities relative to other requests in the system:

- Global user coherence is always the highest priority.
- Block-oriented coherence is always the lowest priority.

Since the user coherence priority is fixed the UCARB register does not include a priority (PRI) bit. Since the global user coherence operations are inherently highest priority, the MAXWAIT programmability does not apply to global cache operations and only applies to block-oriented user coherence operations. Block-oriented user coherency cache operations can affect both L1D and L2 memories; therefore, a version of the UCARB only exists only in the L2 (UCARBU) and L1D (UCARBD) registers.

The MAXWAIT bit (and the implied priorities) does not control the priority of coherency operations that result from DMA transactions or CPU transactions, which have their own registers.

The user coherence arbitration control register (UCARBD, UCARBU) is shown in Figure 6-2 and described in Table 6-5.

Figure 6-2. User Coherence Arbitration Control Register (UCARBD, UCARBU)

31	16
Reserved	
R-0	

15	6	5	0
Reserved		MAXWAIT	
R-0		R/W-20h	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-5. User Coherence Arbitration Control Register (UCARBD, UCARBU) Field Descriptions

Bit	Field	Value	Description
31-6	Reserved	0	Reserved
5-0	MAXWAIT	0-3Fh	Maximum wait time in EMC cycles. EMC cycle = 2 × CPU cycle.
		0	Always stalls due to a higher priority requestor
		1h	Maximum wait of 1 cycle (1/2 = 50% access)
		2h	Maximum wait of 2 cycles (1/3 = 33% access)
		3h	Reserved
		4h	Maximum wait of 4 cycles (1/5 = 20% access)
		5h-7h	Reserved
		8h	Maximum wait of 8 cycles (1/9 = 11% access)
		9h-Fh	Reserved
		10h	Maximum wait of 16 cycles (1/17 = 6% access)
		11h-1Fh	Reserved
		20h	Maximum wait of 32 cycles (1/33 = 3% access)
		21h-3Fh	Reserved

6.3.3 IDMA Arbitration Control Register (IDMAARBD, IDMAARBU, IDMAARBE)

The IDMA arbitration control register (IDMAARBD, IDMAARBU, and IDMAARBE) controls the bandwidth management of the IDMA operations. IDMA supports two active transfers at any point in time via IDMA channel 0 (used for memory to/from CFG space) and IDMA channel 1 (used for memory-to-memory transfers). For more information about the operation of the IDMA, see Chapter 5.

Use the MAXWAIT field to determine the maximum wait time for IDMA transactions. The priority level is not programmed using the IDMAARB register; therefore, the IDMAARB register does not include a PRI field. Instead, the priority level is programmed as part of the IDMA transfer parameters (that is, directly using the IDMA control registers, described in Chapter 5). In summary, the IDMA transfer priority is as follows:

- IDMA channel 0 is always the highest priority.
- IDMA channel 1 has a programmable priority using the PRI field in the IDMA channel 1 count register (IDMA1_COUNT).

IDMA transactions can affect L1D, L2, and EMC resources; therefore, the MAXWAIT field exists for each of these resources: L1D (IDMAARBD), L2 (IDMAARBU), and EMC (IDMAARBE).

The IDMA arbitration control register (IDMAARBD, IDMAARBU, IDMAARBE) is shown in Figure 6-3 and described in Table 6-6.

Figure 6-3. IDMA Arbitration Control Register (IDMAARBD, IDMAARBU, IDMAARBE)

31	16
Reserved	
R-0	

15	6	5	0
Reserved		MAXWAIT	
R-0		R/W-10h	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-6. IDMA Arbitration Control Register (IDMAARBD, IDMAARBU, IDMAARBE) Field Descriptions

Bit	Field	Value	Description
31-6	Reserved	0	Reserved
5-0	MAXWAIT	0-3Fh	Maximum wait time in EMC cycles. EMC cycle = 2 × CPU cycle.
		0	Always stalls due to higher priority requestor.
		1h	Maximum wait of 1 cycle (1/2 = 50% access)
		2h	Maximum wait of 2 cycles (1/3 = 33% access)
		3h	Reserved
		4h	Maximum wait of 4 cycles (1/5 = 20% access)
		5h-7h	Reserved
		8h	Maximum wait of 8 cycles (1/9 = 11% access)
		9h-Fh	Reserved
		10h	Maximum wait of 16 cycles (1/17 = 6% access)
		11h-1Fh	Reserved
		20h	Maximum wait of 32 cycles (1/33 = 3% access)
		21h-3Fh	Reserved

6.3.4 Slave DMA Arbitration Control Register (SDMAARBD, SDMAARBU, SDMAARBE)

The slave DMA arbitration control register (SDMAARBD, SDMAARBU, and SDMAARBE) controls the bandwidth management of the slave DMA (SDMA) operations.

The SDMA can support multiple active transfers at any point in time. The MAXWAIT field controls the maximum wait time for all slave DMA transaction. The priority level is not programmed using SDMAARB; therefore, SDMAARB does not include a PRI field. The system master dictates the priority level instead. Since priority settings outside the C64x+ megamodule are DMA/chip/peripheral specific, see the device-specific documentation for the priority allocation information.

NOTE: The SDMA priorities for all externally-generated DMA transactions (received from outside the C64x+ megamodule) are propagated through the C64x+ megamodule, including all resulting cache coherence operations (snoop, snoop-write).

SDMA transactions can affect L1D, L2, and EMC resources; therefore, the MAXWAIT field exists for L1D (SDMAARBD), L2 (SDMAARBU), and EMC (SDMAARBE).

The slave DMA arbitration control register (SDMAARBD, SDMAARBU, SDMARBE) is shown in Figure 6-4 and described in Table 6-7.

Figure 6-4. Slave DMA Arbitration Control Register (SDMAARBD, SDMAARBU, SDMAARBE)

31	16
Reserved	
R-0	

15	6	5	0
Reserved		MAXWAIT	
R-0		R/W-01h	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-7. Slave DMA Arbitration Control Register ((SDMAARBD, SDMAARBU, SDMARBE) Field Descriptions

Bit	Field	Value	Description
31-6	Reserved	0	Reserved
5-0	MAXWAIT	0-3Fh	Maximum wait time in EMC cycles. EMC cycle = 2 × CPU cycle.
		0	Always stalls due to higher priority requestor
		1h	Maximum wait of 1 cycle (1/2 = 50% access)
		2h	Maximum wait of 2 cycles (1/3 = 33% access)
		3h	Reserved
		4h	Maximum wait of 4 cycles (1/5 = 20% access)
		5h-7h	Reserved
		8h	Maximum wait of 8 cycles (1/9 = 11% access)
		9h-Fh	Reserved
		10h	Maximum wait of 16 cycles (1/17 = 6% access)
		11h-1Fh	Reserved
		20h	Maximum wait of 32 cycles (1/33 = 3% access)
		21h-3Fh	Reserved

6.3.5 Master DMA Arbitration Control Register (MDMAARBE)

The master DMA arbitration control register (MDMAARBE) controls the bandwidth management of the master DMA (MDMA) operations.

The priority (PRI) field controls the submission priority for the following:

- MDMA transactions: the result of cache misses or long distance accesses to non-CFG space
- Configuration bus transactions: long distance accesses to CFG space or IDMA transfers to CFG space

The MDMAARBE priority is different than the other programmable priorities in the system, in that there are two parts to the transaction. Each part is handled differently, as described below:

- Arbitration for the internal half of the transfer is dependent upon the initiator (that could be CPU (L1P or L1D), user coherence (L2), or IDMA, etc.) as defined in other sections.
- Arbitration for the external half of the transfer is DMA-dependent and the PRI field defines it. The PRI field value does not affect any of the internal arbitration for resources. This priority value is simply used for all transactions initiated via the DMA master interface or configuration interface.

NOTE: Since there is no internal arbitration resulting from the DMA arbitration control register (MDMAARBE), there is no MAXWAIT field.

The master DMA arbitration control register (MDMAARBE) is shown in Figure 6-5 and described in Table 6-8.

Figure 6-5. Master DMA Arbitration Control Register (MDMAARBE)

31	19	18	16
Reserved		PRI	
R-0		R/W-7h	

15	0
Reserved	
R-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 6-8. Master DMA Arbitration Control Register (MDMAARBE) Field Descriptions

Bit	Field	Value	Description
31-19	Reserved	0	Reserved
18-16	PRI	0-7h	Priority field: Not all requestors support PRI = 8 (lowest), this is used to make background transfers lower than all other real-time requests.
		0	Priority 0 (highest)
		1h	Priority 1
		2h	Priority 2
		3h	Priority 3
		4h	Priority 4
		5h	Priority 5
		6h	Priority 6
		7h	Priority 7 (lowest)
15-0	Reserved	0	Reserved

6.4 Privilege and Bandwidth Management Registers

Table 6-9 summarizes which bandwidth management registers are accessible according to a person's role (supervisor or user).

Table 6-9. Permissions for Bandwidth Management Registers

Register	Supervisor	User
CPUARBD	R/W	R
IDMAARBD	R/W	R
SDMAARBD	R/W	R
UCARBD	R/W	R
CPUARBU	R/W	R
IDMAARBU	R/W	R
SDMAARBU	R/W	R
UCARBU	R/W	R
CPUARBE	R/W	R
IDMAARBE	R/W	R
SDMAARBE	R/W	R
MDMAARBE	R/W	R

Interrupt Controller

Topic **Page**

7.1 Introduction 156
7.2 Interrupt Controller Architecture 158
7.3 C64x+ Megamodule Events 166
7.4 Interrupt Controller - CPU Interaction 167
7.5 Registers 169

7.1 Introduction

This section provides the purpose and discusses the features of the interrupt controller.

7.1.1 Purpose of the C64x+ Megamodule Interrupt Controller (INTC)

The C64x+ system provides a large assortment of system events. The interrupt controller provides a way to select the necessary events and route them to the appropriate CPU interrupt and exception inputs.

While you can use many of these same system events to drive other peripherals, such as the EDMA, the megamodule's interrupt controller is dedicated to managing the CPU.

7.1.2 Features

NOTE: The nonmaskable interrupt (NMI) is not supported on all C6000 devices, see the device-specific data manual for more information.

The interrupt controller interfaces the system events to the CPU's interrupt and exceptions inputs. The interrupt controller supports up to 128 system events.

There are 128 system events that act as inputs to the interrupt controller. They consist of both internally-generated events (within the megamodule) and chip-level events. The list of events are enumerated later in Section 7.3. In addition to these 128 events, the INTC register also receives the non-maskable and reset events and routes straight through to the CPU.

The interrupt controller outputs signals to the C64x+ CPU from these event inputs:

- One maskable, hardware exception (EXCEP)
- Twelve maskable hardware interrupts (INT4 through INT15)
- One non-maskable signal that you can use as either an interrupt or an exception (NMI)
- One reset signal (RESET)

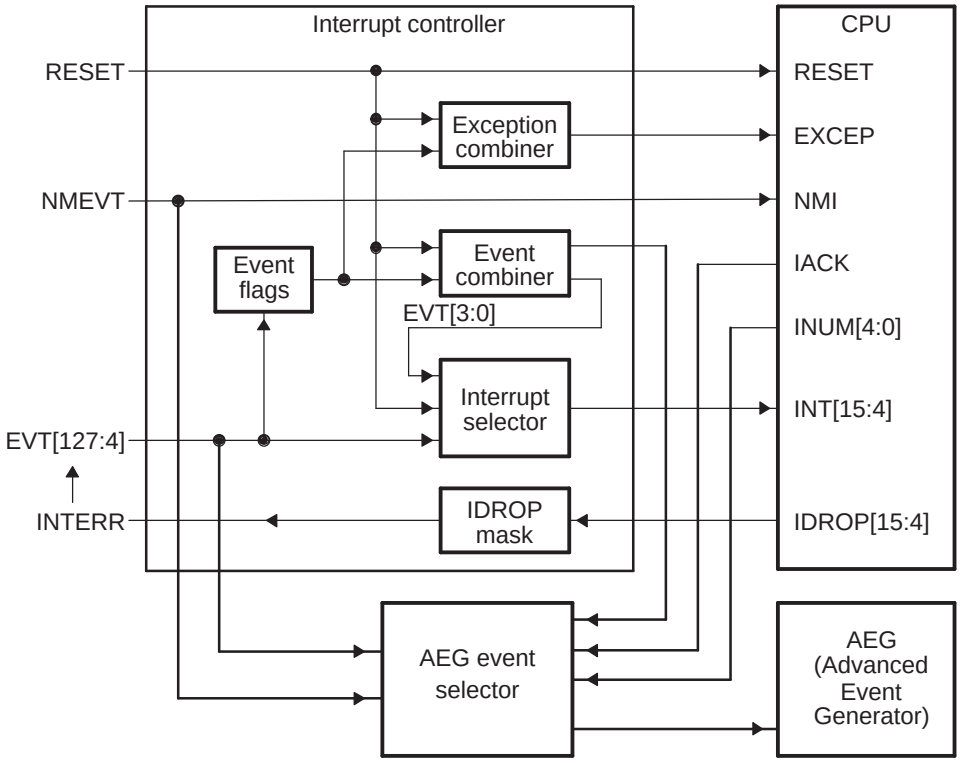
For more information on these CPU interrupt/exception signals, Refer to the *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide* (SPRU732) .

The interrupt controller includes the following modules to facilitate the routing of events to interrupts and exceptions:

- Interrupt Selector - routes any of the system events to the 12 maskable interrupts
- Event Combiner - reduces the large number of system events down to four
- Exception Combiner - lets any of the system events be grouped together for the single hardware exception input

7.1.3 Functional Block Diagram

Figure 7-1. C64x+ Megamodule Interrupt Controller Block Diagram



7.1.4 Terms and Definitions

Terms of specific importance in this chapter are:

System Event: any signal that generates internally or externally that is intended to notify the CPU that some activity has occurred and/or requires a response.

Interrupts: provide the means to redirect normal program flow due to the presence of an external or internal hardware signal (event).

Exceptions are similar to interrupts in that they also redirect program flow, but exceptions are normally associated with error conditions in the system.

Refer to Appendix A and Appendix B of this document for additional definitions of the terms used in this chapter. Appendix A describes general terms used throughout this reference guide and Appendix B defines terms related to the memory and cache architecture.

7.2 Interrupt Controller Architecture

The C64x+ megamodule interrupt controller is designed to provide flexible management of system events. This functionality is implemented using the set of registers listed in Table 7-1. These registers are mentioned throughout this chapter. Detailed descriptions for these registers are provided in Section 7.5.

Table 7-1. Interrupt Controller Registers

Register	Description	Type
EVTFLAG [3:0]	Event Flag Registers	Status
EVTCLR [3:0]	Event Clear Registers	Command
EVTSET [3:0]	Event Set Registers	Command
EVTMASK [3:0]	Event Mask Registers	Control
MEVTFLAG [3:0]	Masked Event Flag Registers	Status
EXPMASK [3:0]	Exception Mask Registers	Control
MEXPFLAG [3:0]	Masked Exception Flag Registers	Status
INTMUX [3:1]	Interrupt Mux Registers	Control
AEGMUX [1:0]	Advanced Event Generator Mux Registers	Control
INTXSTAT	Interrupt Exception Status Register	Status
INTXCLR	Interrupt Exception Clear Register	Command
INTDMASK	Dropped Interrupt Mask Register	Control

7.2.1 Event Registers

The interrupt controller contains a set of registers to manage the status of the system events received by the controller. The registers can be grouped as follows:

- Event flag registers (EVTFLAGx)
- Clear flag registers (EVTCLRx)
- Set flag registers (EVTSETx)

The event flag registers capture all system events that are received by the Interrupt Controller. There are four 32-bit registers to cover the 124 system event inputs. Each system event is mapped to a specific flag bit (EFxx) in one of the event flag registers.

The generic event flag register structure is shown in Figure 7-2.

Figure 7-2. Event Flag Register Structure

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF	EF
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

All 124 system events are individually mapped to a bit of the four 32-bit EVTFLAGx registers. This leaves the least significant four bits of EVTFLAG0 (EF03:EF00) not associated with a system event. These four bits are reserved and always zero. That is, there are no system event inputs that correspond to these fields. Instead, the system events associated with events 00 through 03 are generated internal (to the Interrupt Controller) by the Event Combiner, which are routed to the Interrupt Selector, as shown in Figure 7-1.

The event flags (EFxx) are latched register bits; that is, they retain the value of 1 for any event received. The EVTFLAGx registers are read-only and must be cleared through the write-only Event Clear registers EVTCLR[3:0].

Use the event clear registers to clear the event flag registers. There are four 32-bit event clear registers. The fields of these registers map one-to-one with the fields of the event flag registers. Writing a 1 to a specific field in an event clear register causes the corresponding event flag register field to clear.

The event clear register structure is shown in Figure 7-3.

Figure 7-3. Event Clear Register Structure

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC	EC
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: W = Write only; *-n* = value after reset

The event set registers are conceptually similar to the event clear registers. Use the event set registers to manually set any bit(s) within the event flag registers (e.g., it may be beneficial to use the event set registers to generate interrupts when testing interrupt service routines). There are four 32-bit event set registers whose fields map one-to-one to the fields of the event flag registers. Writing a 1 to a specific field in an event set register causes the corresponding event flag register to set to 1.

The event set register structure is shown in Figure 7-4.

Figure 7-4. Event Set Register Structure

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ES	ES	ES	ES	ES	ES	ES	ES	ES	ES	ES	ES	ES	ES	ES	ES
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ES	ES	ES	ES	ES	ES	ES	ES	ES	ES	ES	ES	ES	ES	ES	ES
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: W = Write only; *-n* = value after reset

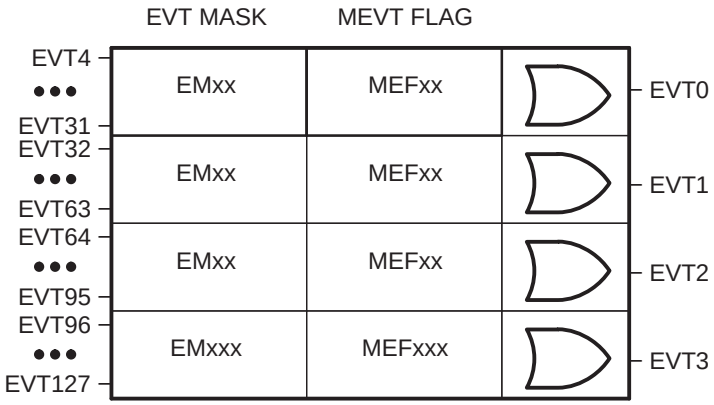
The interrupt controller uses the event clear and event set registers, rather than writing directly to the event flag registers to prevent potential race conditions. Without these additional registers, the CPU might have otherwise accidentally cleared event flags set during a read-modify-write operation of the flag bits.

If a new event is received during the same cycle, a clear is specified via an EVTCLRx register, the new event input takes precedence as an additional precaution against missing events.

7.2.2 Event Combiner

The event combiner (Figure 7-5) allows multiple system events to be combined into a single event. The combined events are routed to the interrupt selector. This allows the CPU to service all available system events even though the CPU only has twelve available interrupts.

Figure 7-5. Event Combiner



The basic concept of the event combiner is to perform an OR operation on a subset of the system event flags (described in Table 7-2). The results of the OR operation are provided as a new “combined” event).

The event combiner divides the 124 system events into four groups. The first group includes events 4 through 31, the second group includes events 32 through 63, the third group includes events 64 through 95, and the fourth group includes events 96 through 127. You can combine events within each group to provide a new “combined” event. These new events are designated EVT0, EVT1, EVT2, and EVT3. These events are routed to the interrupt selector along with the original 124 system events for a combined total of 128 events.

For each group there is an event mask register.

The general structure of the event mask register is shown in Figure 7-6.

Figure 7-6. Event Mask Register Structure

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EM	EM	EM	EM	EM	EM	EM	EM	EM	EM	EM	EM	EM	EM	EM	EM
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EM	EM	EM	EM	EM	EM	EM	EM	EM	EM	EM	EM	EM	EM	EM	EM
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

The event mask bits within the event mask registers act to enable/mask which received system events should be combined. The register is zero by default, thus all system events are unmasked and combined to form the associated EVTx. To mask out an event source (for example, disable an event from being combined), the corresponding mask bit must be set to 1. Note that the event mask bits for events 0 through 3 are reserved, and are always masked.

Example 7-1.

```
Assume an application requires the events 124-127 to be combined.
In order to accomplish this, EVTMASK3 will need to be programmed as follows:
 EVTMASK3 = 00001111111111111111111111111111
```

In addition to generating a combined output event based on programmable event combinations, the event combiner provides a masked view of the event flag registers.

The structure of the masked event flag register is shown in Figure 7-7.

Figure 7-7. 32-Masked Event Flag Register Structure

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

The content of the masked event flag registers is identical to the content of the event flag registers for the events that are enabled in the event mask registers. By reading the masked event flag registers, the CPU only sees the event flags pertaining to the corresponding combined event (EVT [3:0]), which can be useful in interrupt routines servicing combined events.

Example 7-2.

```
Assuming the following configuration:
  EVTFLAG3 = 01101010010011001110001110010101
  EVTMASK3 = 00001111111111111111111111111111

The Masked Event Flag register 3 will be:
  MEVTFLAG3 = 01100000000000000000000000000000
```

When servicing a combined interrupt, you must:

1. Read the MEVTFLAGx register corresponding to the combined event EVTx
2. Check for the first pending (i.e., flagged) events
3. Write this MEVTFLAGx value to the EVTCLRx register
4. Service the event indicated in step 2
5. Repeat steps 1 through 4 until the MEVTFLAGx register = 0

This procedure only evaluates and clears those events combined on EVTx. Further, any events that are masked in the EVTMASKx register are not be cleared (and they do not need to clear), even if they are set in the EVTFLAGx register (this allows you to use them to generate an exception).

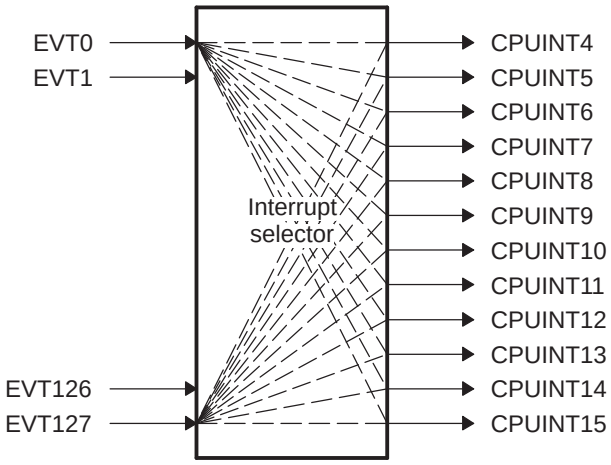
NOTE: The CPU should iterate steps 1 to 4 until no pending events are found before returning within the interrupt service routine. This ensures that any events that are received during the interrupt service routine are captured (also remember that if an event EVTx is received at the same time that its flag is cleared in the EVTCLRy [x] register, then it will not clear).

7.2.3 Interrupt Selector

7.2.3.1 Interrupt Selector Operation

The CPU has twelve maskable interrupts (CPUINT4 through 15) are available. The interrupt selector allows any of the 128 system events to route to any of the twelve CPU interrupt inputs, as shown in Figure 7-8.

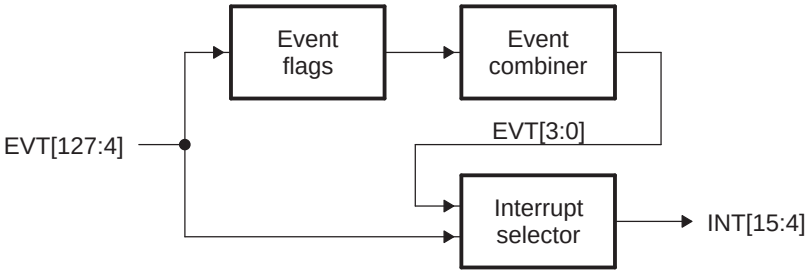
Figure 7-8. Interrupt Selector Block Diagram



The 128 system events are either event inputs or event combinations generated by the event combiner. The event combiner logic has the capability of grouping multiple event inputs to four possible event outputs. These outputs are then provided to the interrupt selector and treated as additional system events (EVT0 through EVT3).

The event combiner allows for a flexible interrupt routing scheme in addition to the interrupt selector. This flexibility the INTC module allows a large number of system interrupts to be serviced within the megamodule. It also allows a large number of interrupts to be simultaneously serviced within a CPU, thus increasing interrupt efficiency.

Figure 7-9. CPU Interrupt Routing Diagram



The interrupt selector contains interrupt multiplexing registers, INTMUX[3:1] that allow you to program the source for each of the 12 available CPU interrupts. Each of the events that are presented to the interrupt selector has an event number that is used to program these registers.

The order of the CPU interrupts (CPUINT4 through CPUINT15) determines the priority for pending interrupts. Since any interrupt service routine can be atomic (not nestable), the CPU interrupt priority only applies to pending interrupts. For more information regarding the CPU's interrupt features, refer to the *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide* (SPRU732) .

7.2.3.2 Interrupt Error Event

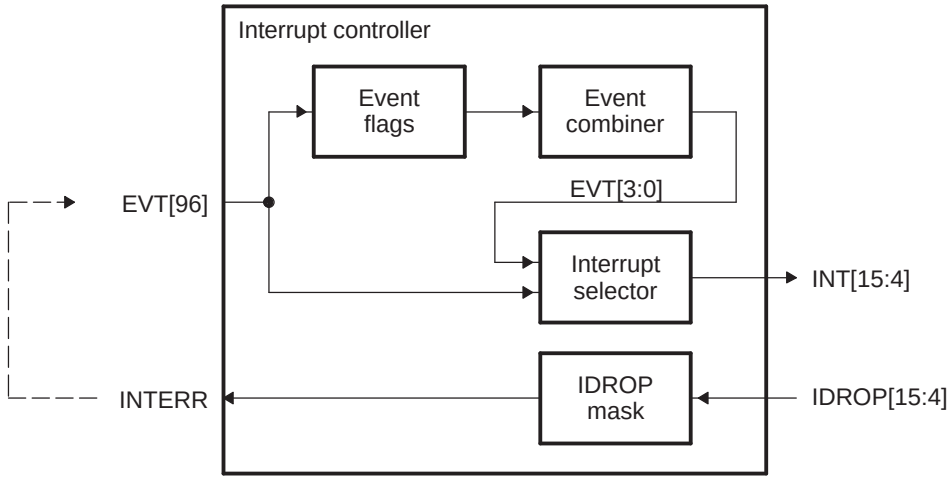
The C64x+ CPU along with the interrupt controller can generate a system event (EVT96) whenever the CPU detects that an interrupt has been dropped. This event is generated when a CPU interrupt is received while the associated CPU's interrupt flag bit is already set. This error event might indicate the programmer of possible problems in the code such as whether interrupts were disabled for an extended period of time or whether non-interruptible code sections were too long.

Since the interrupt drop detection logic is within the CPU, only interrupts that are sourced from a single system event can be detected. The dropping of interrupts based on combined events can only indicate that one or more of the interrupts in that group caused the error.

When the CPU detects the dropped error condition, it passes the information back to the interrupt controller's interrupt exception status register (INTXSTAT) which records the dropped interrupt's number and asserts a system event. This register is described in Section 7.5.3.2.

A block diagram including the signals related to exception generation is shown in Figure 7-10.

Figure 7-10. Interrupt Exception Event Block Diagram



The INTERR event is output from the interrupt controller and is internally routed back to the system event EVT96, as shown in Figure 7-10.

As INTXERR can only hold a single dropped CPU ID, only the first dropped interrupt detected is reported by INTERR (EVT96). The interrupt exception status is cleared through the exception clear register (INTXCLR), which is comprised of only a single clear bit. Writing a 1 to the CLEAR field in the INTXCLR register resets the INTXSTAT register to 0. A new IDROPx event can only be detected after the status is cleared by the hardware.

When servicing the dropped interrupt error event, the service routine should:

1. Read the INTXSTAT register.
2. Check the error condition.
3. Clear the error through the INTXCLR register.

To prevent one or more CPU interrupts from generating dropped interrupt errors, ignore them by programming the dropped interrupt mask register (INTDMASK).

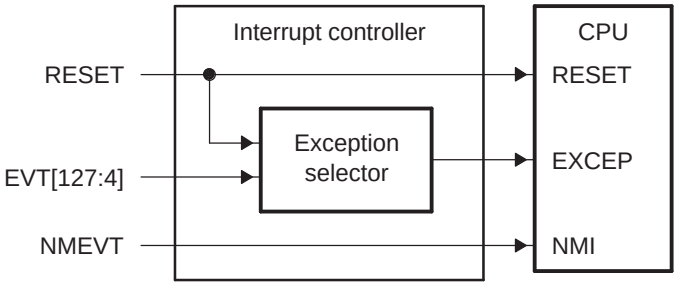
7.2.4 Exception Combiner

The C64x+ CPU has a single event input for system-level, maskable, exceptions. This input is denoted by EXCEP. The exception combiner allows multiple system events to be combined into the single exception event in Figure 7-12. This allows the CPU to service all available system events even though only one CPU exception input is available.

The exception combiner allows the system designer to select a subset of the system event flags in which to perform an OR operation to determine the EXCEP value.

A block diagram showing the routing of system exceptions through the exception combiner is shown in Figure 7-11.

Figure 7-11. System Exception Routing Diagram



NOTE: Reset and NMI are also shown in this diagram. In fact, when exceptions are enabled within the C64x+ CPU, the NMI signal is used as a non-maskable exception input. These two signals are combined within the CPU along with a variety of other CPU exceptions. For more information on CPU exceptions, refer to the *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide* (SPRU732) .

To allow only a subset of system events to generate an exception to the CPU, the exception combiner provides a set of four mask registers, EXPMASK[3:0] which are used to disable the events that are not desired. Since there is only one exception input to the CPU, all mask registers work in concert to combine up to 128 events to a single EXCEP output. This allows the CPU to service all available system exceptions.

The general structure of the exception mask register is provided in Figure 7-12:

Figure 7-12. Exception Mask Register Structure

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
XM	XM	XM	XM	XM	XM	XM	XM	XM	XM	XM	XM	XM	XM	XM	XM
R/W-FFFFh															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XM	XM	XM	XM	XM	XM	XM	XM	XM	XM	XM	XM	XM	XM	XM	XM
R/W-FFFFh															

LEGEND: R/W = Read/Write; *-n* = value after reset

The default value of the EXPMASKx registers are all 1s. This means that all events are masked; therefore, no system events generate an exception unless you program this register.

Similar to the event combiner discussed in Section 7.2.2, the exception combiner provides a set of masked exception flags (MEXPFLAGx) in combination with the exception mask registers. The masked exception flag registers provide a masked view of the event flag registers (from Section 7.2.1). By reading the masked exception flag registers, the only CPU sees the event flags pertaining to the CPU's EXCEP input.

The general structure of the masked exception flag registers is shown in Figure 7-13.

Figure 7-13. Masked Exception Flag Register Structure

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MXF	MXF	MXF	MXF	MXF	MXF	MXF	MXF	MXF	MXF	MXF	MXF	MXF	MXF	MXF	MXF
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MXF	MXF	MXF	MXF	MXF	MXF	MXF	MXF	MXF	MXF	MXF	MXF	MXF	MXF	MXF	MXF
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

The CPU should run an exception service routine to determine the cause of the exception and respond to the appropriate events upon receiving an exception. When servicing exceptions, the service routine must first determine whether the exception was generated internal to the CPU, by the non-maskable exception, or by the EXCEP signal.

If EXCEP was found to be the cause of the exception, the routine should read the masked exception flag registers (MEXPFLAG [3:0]) to determine which unmasked events triggered the exception.

When servicing a combined interrupt, you must:

1. Read the MEXPFLAG [3:0] registers.
2. Check the pending events to be serviced.
3. Write the MEXPFLAG [3:0] values to the EVTCLR [3:0] registers.

   Using the MEXPFLAGx values with the EVTCLRx registers only clears those events that were combined to generate EXCEP. Any events that are masked in EXPMASKx would not need to be cleared, even if set in the EVTFLAGx register; this allows them to be used to generate a combined interrupt event.
4. The CPU should iterate on steps 1 to 3 until no pending events are found before returning from the exception service routine. This ensures that any events received during the exception service routine are captured.

NOTE: Step 4 is critical if the CPU is required to respond to any new exceptions. Two facts indicate why this is the case:

- The output of the exception combiner is active when any unmasked event flag inputs are active.
- The CPU recognizes an exception request as a 0 to 1 transition.

Therefore, all unmasked event flags must clear before the CPU can recognize a new low to high transition on EXCEP.

7.3 C64x+ Megamodule Events

There are a number of events that the various components of the C64x+ megamodule generates. These events are routed to the interrupt controller so that when asserted, they can be serviced by the CPU. These events are listed in Table 7-2, along with their event mapping.

NOTE: The events that are shown as available events (4 through 8, 10, and 15 through 95) are to the megamodule for chip-level events. Therefore, each new C64x+ device can use these event inputs as necessary. Refer to the device-specific data manual for more information about how these available events are used.

Table 7-2. System Event Mapping

EVT Number	Event	From	Description
0	EVT0	INT controller	Output of event combiner 0, for events 1 through 31.
1	EVT1	INT controller	Output of event combiner 1, for events 32 through 63.
2	EVT2	INT controller	Output of event combiner 2, for events 64 through 95.
3	EVT3	INT controller	Output of event combiner 3, for events 96 through 127.
4-8	Available events.		
9	Reserved		
10	Available events.		
11-12	Reserved		
13	IDMAINT0	EMC	IDMA channel 0 interrupt
14	IDMAINT1	EMC	IDMA channel 1 interrupt
15-95	Available events.		
96	INTERR	INT controller	Dropped CPU interrupt event
97	EMC_IDMAERR	EMC	Invalid IDMA parameters
98-117	Reserved		
118	PDC_INT	PDC	PDC sleep interrupt
119	SYS_CMPA	SYS	CPU memory protection fault
120	L1P_CMPA	L1P	CPU memory protection fault
121	L1P_DMPA	L1P	DMA memory protection fault
122	L1D_CMPA	L1D	CPU memory protection fault
123	L1D_DMPA	L1D	DMA memory protection fault
124	L2_CMPA	L2	CPU memory protection fault
125	L2_DMPA	L2	DMA memory protection fault
126	EMC_CMPA	EMC	CPU memory protection fault
127	EMC_BUSERR	EMC	Bus error interrupt

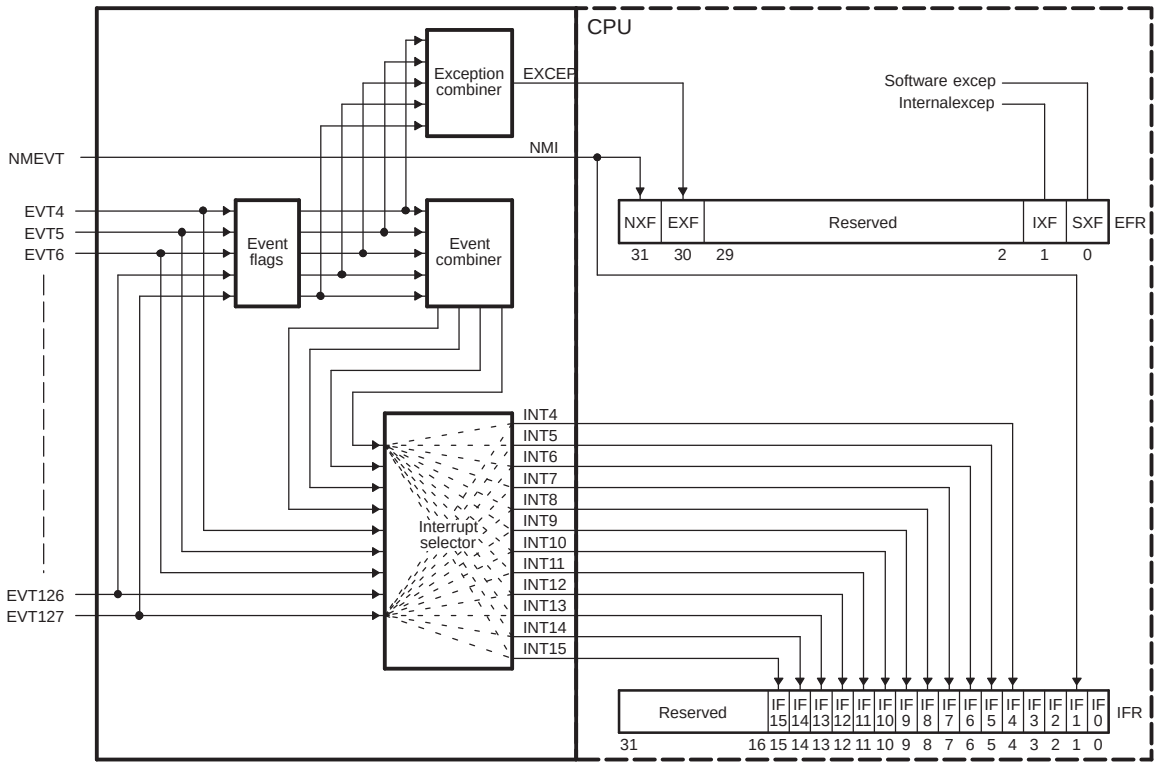
7.4 Interrupt Controller - CPU Interaction

7.4.1 CPU – Interrupt Controller Interface

The interrupt controller's outputs, as produced by the exception combiner and the interrupt selector, are provided to the C64x+ CPU.

The twelve interrupt signals are reflected in the CPU's interrupt flag register (IFR), as shown in Figure 7-14.

Figure 7-14. CPU Event Routing Diagram



You must enable interrupts in order for the CPU to recognize them. The CPU requires individual enables via the interrupt enable register (IER) and via the global interrupt enable field in the interrupt task register (ITSR.GIE).

Also note that the exception signal (EXCEP) is recorded in the CPU's exception flag register (EFR) in Figure 7-15. You must enable exception before the exception flag registers (EFR) shown can be recognized. Exception recognition is disabled after device reset for ease of system design and for backward compatibility. You can turn on exceptions by setting the global exceptions enable field (GEE) in the ITSR register (ITSR). You should enable exceptions prior to enabling any interrupts to ensure that an NMI is not received while its mode (exception vs. interrupt) is changing.

When system exceptions are not enabled in the CPU, the non-maskable interrupt (NMI) acts as an interrupt and when received will post a flag to the BIT1 field in the IFR register. When system exceptions are enabled in the CPU; however, this flag is not set. Rather, the exception source is identified in the exception flag register (EFR) to denote whether the source is NMI, EXCEP, an internal exception, or a software exception (SWE/SWENR).

All NMI processing shares the NMI interrupt vector, regardless of whether you are using it as an interrupt or it represents an exception. The CPU only uses its REP register as a vector as opposed to the NMI vector in the case where the SWENR generates an exception rather than SWE instruction.

For more detailed information, refer to the *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide* (SPRU732) .

7.4.2 CPU Servicing of Interrupt Events

For the case where the CPU services single-event interrupts (where system events are specified directly in the Interrupt Selector), there is no need to read or clear the Event Flag (EVTFLAGx) registers in the Interrupt Controller.

However, you must use event flags within an interrupt service routine or an exception service routine when servicing combined system events. These flags are used to determine the event (s) that initiated an interrupt or exception. In other words, the CPU's interrupt flag register (or exception flag register) tell the CPU a combined event has occurred, then the service routine must use the event flag register to determine the exact cause(s).

It is also important to note that within the service routine, the appropriate event flag register bits must be cleared by software in order to receive a subsequent event. If the event flag(s) does not clear, then a new system event will not be recognized. The new system event can not even be recognized as a dropped interrupt. This is because the CPU dropped interrupt logic applies to the CPU interrupt input (not the interrupt controller event input). Since the events are combined in the Interrupt Controller, the CPU has no visibility here.

In many systems, it may be tempting to have the service routine read, then clear the entire event flag register (EVTFLAGx). While this can work fine for some systems, you must take care that some of the event flags are not being polled by any of the system's code. If a particular event must be polled (read occasionally by some code within the system rather than allowing that event to interrupt the CPU), then indiscriminately clearing all of the event flag bits may cause unexpected results.

7.5 Registers

The C64x+ megamodule interrupt controller registers are listed in Table 7-3.

Table 7-3. Interrupt Controller Registers

Address	Acronym	Register Description	Section
0180 0000h to 0180 000Ch	EVTFLAG0	Event flag register 0	Section 7.5.1.1
	EVTFLAG1	Event flag register 1	Section 7.5.1.1
	EVTFLAG2	Event flag register 2	Section 7.5.1.1
	EVTFLAG3	Event flag register 3	Section 7.5.1.1
0180 0020h to 0180 002Ch	EVTSET0	Event set register 0	Section 7.5.1.2
	EVTSET1	Event set register 1	Section 7.5.1.2
	EVTSET2	Event set register 2	Section 7.5.1.2
	EVTSET3	Event set register 3	Section 7.5.1.2
0180 0040h to 0180 004Ch	EVTCLR0	Event clear register 0	Section 7.5.1.3
	EVTCLR1	Event clear register 1	Section 7.5.1.3
	EVTCLR2	Event clear register 2	Section 7.5.1.3
	EVTCLR3	Event clear register 3	Section 7.5.1.3
0180 0080h to 0180 008Ch	EVTMASK0	Event mask register 0	Section 7.5.2.1
	EVTMASK1	Event mask register 1	Section 7.5.2.1
	EVTMASK2	Event mask register 2	Section 7.5.2.1
	EVTMASK3	Event mask register 3	Section 7.5.2.1
0180 00A0h to 0180 00ACh	MEVTFLAG0	Masked event flag register 0	Section 7.5.2.2
	MEVTFLAG1	Masked event flag register 1	Section 7.5.2.2
	MEVTFLAG2	Masked event flag register 2	Section 7.5.2.2
	MEVTFLAG3	Masked event flag register 3	Section 7.5.2.2
0180 0104h to 0180 010Ch	INTMUX1	Interrupt mux register 1	Section 7.5.3.1
	INTMUX2	Interrupt mux register 2	Section 7.5.3.1
	INTMUX3	Interrupt mux register 3	Section 7.5.3.1
0181 0140h	AEGMUX0	Advanced event generator mux register 0	Section 7.5.5
0181 0144h	AEGMUX1	Advanced event generator mux register 1	Section 7.5.5
0180 0180h	INTXSTAT	Interrupt exception status register	Section 7.5.3.2
0180 0184h	INTXCLR	Interrupt exception clear register	Section 7.5.3.3
0180 0188h	INTDMASK	Dropped interrupt mask register	Section 7.5.3.4
0180 00C0h to 0180 00CCh	EXPMASK0	Exception Mask register 0	Section 7.5.4.1
	EXPMASK1	Exception Mask register 1	Section 7.5.4.1
	EXPMASK2	Exception Mask register 2	Section 7.5.4.1
	EXPMASK3	Exception Mask register 3	Section 7.5.4.1
0180 00E0h to 0180 00ECh	MEXPFLAG0	Masked Exception Flag register 0	Section 7.5.4.2
	MEXPFLAG1	Masked Exception Flag register 1	Section 7.5.4.2
	MEXPFLAG2	Masked Exception Flag register 2	Section 7.5.4.2
	MEXPFLAG3	Masked Exception Flag register 3	Section 7.5.4.2

7.5.1 Event Registers

The interrupt controller contains a set of status and control registers to manage the system events that are received by the controller. These include flag, set, and clear registers covering all 128 system events.

NOTE: Event flag bits 0 through 3 are reserved and are always 0. There are no events corresponding to these fields that get routed to the event flag register.

7.5.1.1 Event Flag Registers (EVTFLAG*n*)

The event flags in the event flag registers (EVTFLAG*n*) retain a value of 1 for any of the 128 system events received and are read-only registers. Use the write-only event clear registers (EVTCLR*n*) to clear the registers. Use the event set registers (EVTSET*n*) to manually set any bit(s) within EVTFLAG*n*, including masked bits. The event flag registers (EVTFLAG*n*) are shown in Figure 7-15 through Figure 7-18 and described in Table 7-4.

Figure 7-15. Event Flag Register 0 (EVTFLAG0)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EF31	EF30	EF29	EF28	EF27	EF26	EF25	EF24	EF23	EF22	EF21	EF20	EF19	EF18	EF17	EF16
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EF15	EF14	EF13	EF12	EF11	EF10	EF9	EF8	EF7	EF6	EF5	EF4	EF3	EF2	EF1	EF0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Figure 7-16. Event Flag Register 1 (EVTFLAG1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EF63	EF62	EF61	EF60	EF59	EF58	EF57	EF56	EF55	EF54	EF53	EF52	EF51	EF50	EF49	EF48
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EF47	EF46	EF45	EF44	EF43	EF42	EF41	EF40	EF39	EF38	EF37	EF36	EF35	EF34	EF33	EF32
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Figure 7-17. Event Flag Register 2 (EVTFLAG2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EF95	EF94	EF93	EF92	EF91	EF90	EF89	EF88	EF87	EF86	EF85	EF84	EF83	EF82	EF81	EF80
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EF79	EF78	EF77	EF76	EF75	EF74	EF73	EF72	EF71	EF70	EF69	EF68	EF67	EF66	EF65	EF64
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Figure 7-18. Event Flag Register 3 (EVTFLAG3)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EF127	EF126	EF125	EF124	EF123	EF122	EF121	EF120	EF119	EF118	EF117	EF116	EF115	EF114	EF113	EF112
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EF111	EF110	EF109	EF108	EF107	EF106	EF105	EF104	EF103	EF102	EF101	EF100	EF99	EF98	EF97	EF96
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 7-4. Event Flag Registers (EVTFLAG*n*) Field Descriptions

Bit	Field	Value	Description
31-0	EF*yyy*		Captures the state of event EVT*yyy*
		0	EVT*yyy* did not occur.
		1	EVT*yyy* occurred.

7.5.1.2 Event Set Registers (EVTSET*n*)

Use the event set registers (EVTSET*n*) to manually set any bit(s) within the event flag registers (EVTSET*n*).

The event set registers (EVTSET*n*) are shown in Figure 7-19 through Figure 7-22 and described in Table 7-5.

Figure 7-19. Event Set Register 0 (EVTSET0)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ES31	ES30	ES29	ES28	ES27	ES26	ES25	ES24	ES23	ES22	ES21	ES20	ES19	ES18	ES17	ES16
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ES15	ES14	ES13	ES12	ES11	ES10	ES9	ES8	ES7	ES6	ES5	ES4	ES3	ES2	ES1	ES0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: W = Write only; -*n* = value after reset

Figure 7-20. Event Set Register 1 (EVTSET1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ES63	ES62	ES61	ES60	ES59	ES58	ES57	ES56	ES55	ES54	ES53	ES52	ES51	ES50	ES49	ES48
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ES47	ES46	ES45	ES44	ES43	ES42	ES41	ES40	ES39	ES38	ES37	ES36	ES35	ES34	ES33	ES32
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: W = Write only; -*n* = value after reset

Figure 7-21. Event Set Register 2 (EVTSET2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ES95	ES94	ES93	ES92	ES91	ES90	ES89	ES88	ES87	ES86	ES85	ES84	ES83	ES82	ES81	ES80
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ES79	ES78	ES77	ES76	ES75	ES74	ES73	ES72	ES71	ES70	ES69	ES68	ES67	ES66	ES65	ES64
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: W = Write only; -*n* = value after reset

Figure 7-22. Event Set Register 3 (EVTSET3)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ES127	ES126	ES125	ES124	ES123	ES122	ES121	ES120	ES119	ES118	ES117	ES116	ES115	ES114	ES113	ES112
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ES111	ES110	ES109	ES108	ES107	ES106	ES105	ES104	ES103	ES102	ES101	ES100	ES99	ES98	ES97	ES96
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: W = Write only; -*n* = value after reset

Table 7-5. Event Set Registers (EVTSET*n*) Field Descriptions

Bit	Field	Value	Description
31-0	ES*yyy*		Sets EF*yyy* in the event flag registers (EVTFLAG*n*).
		0	No effect.
		1	Set EF*yyy* = 1

7.5.1.3 Event Clear Registers (EVTCLR*n*)

Use the event clear registers (EVTCLR*n*) to clear the event flags in the event flag registers (EVTCLR*n*).

The event clear registers (EVTCLR*n*) are shown in Figure 7-23 through Figure 7-26 and described in Table 7-6.

Figure 7-23. Event Clear Register 0 (EVTCLR0)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EC31	EC30	EC29	EC28	EC27	EC26	EC25	EC24	EC23	EC22	EC21	EC20	EC19	EC18	EC17	EC16
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EC15	EC14	EC13	EC12	EC11	EC10	EC9	EC8	EC7	EC6	EC5	EC4	EC3	EC2	EC1	EC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: W = Write only; -*n* = value after reset

Figure 7-24. Event Clear Register 1 (EVTCLR1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EC63	EC62	EC61	EC60	EC59	EC58	EC57	EC56	EC55	EC54	EC53	EC52	EC51	EC50	EC49	EC48
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EC47	EC46	EC45	EC44	EC43	EC42	EC41	EC40	EC39	EC38	EC37	EC36	EC35	EC34	EC33	EC32
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: W = Write only; -*n* = value after reset

Figure 7-25. Event Clear Register 2 (EVTCLR2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EC95	EC94	EC93	EC92	EC91	EC90	EC89	EC88	EC87	EC86	EC85	EC84	EC83	EC82	EC81	EC80
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EC79	EC78	EC77	EC76	EC75	EC74	EC73	EC72	EC71	EC70	EC69	EC68	EC67	EC66	EC65	EC64
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: W = Write only; -*n* = value after reset

Figure 7-26. Event Clear Register 3 (EVTCLR3)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EC127	EC126	EC125	EC124	EC123	EC122	EC121	EC120	EC119	EC118	EC117	EC116	EC115	EC114	EC113	EC112
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EC111	EC110	EC109	EC108	EC107	EC106	EC105	EC104	EC103	EC102	EC101	EC100	EC99	EC98	EC97	EC96
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: W = Write only; -*n* = value after reset

Table 7-6. Event Clear Registers (EVTCLR*n*) Field Descriptions

Bit	Field	Value	Description
31-0	EC*yyy*		Clears EF*yyy* in the event flag registers (EVTFLAG*n*).
		0	No effect.
		1	Set EF*yyy* = 0.

7.5.2 Event Combiner Registers

There are a set of event mask registers (EVTMASK [3:0]) to program the event combiner. These registers allow up to 32 events to be combined into a single combined event which can then be used by the interrupt selector. The event mask bits within the EVTMASK [3:0] registers act to mask (or enable) the received system events. There are four event signals presented to the interrupt selector (EVT [3:0]).

The event mask registers are shown below (Bits EM [3:0] are unused).

7.5.2.1 Event Mask Registers (EVTMASK*n*)

There are a set of event mask registers (EVTMASK0 through EVTMASK3) to program the event combiner. These registers allow up to 32 events to be combined into a single event output that is used as a single CPU interrupt or AET event. The event mask bits within the EVTMASK*n* register act as enablers for the received system events to be combined on the event outputs. There are four event outputs to the event and AET event selectors (EVT [3:0]).

The event mask registers (EVTMASK*n*) are shown in Figure 7-27 through Figure 7-30 and described in Table 7-7.

Figure 7-27. Event Mask Register 0 (EVTMASK0)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EM31	EM30	EM29	EM28	EM27	EM26	EM25	EM24	EM23	EM22	EM21	EM20	EM19	EM18	EM17	EM16
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EM15	EM14	EM13	EM12	EM11	EM10	EM9	EM8	EM7	EM6	EM5	EM4	EM3	EM2	EM1	EM0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R-1	R-1	R-1

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 7-28. Event Mask Register 1 (EVTMASK1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EM63	EM62	EM61	EM60	EM59	EM58	EM57	EM56	EM55	EM54	EM53	EM52	EM51	EM50	EM49	EM48
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EM47	EM46	EM45	EM44	EM43	EM42	EM41	EM40	EM39	EM38	EM37	EM36	EM35	EM34	EM33	EM32
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Figure 7-29. Event Mask Register 2 (EVTMASK2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EM95	EM94	EM93	EM92	EM91	EM90	EM89	EM88	EM87	EM86	EM85	EM84	EM83	EM82	EM81	EM80
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EM79	EM78	EM77	EM76	EM75	EM74	EM73	EM72	EM71	EM70	EM69	EM68	EM67	EM66	EM65	EM64
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Figure 7-30. Event Mask Register 3 (EVTMASK3)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EM127	EM126	EM125	EM124	EM123	EM122	EM121	EM120	EM119	EM118	EM117	EM116	EM115	EM114	EM113	EM112
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EM111	EM110	EM109	EM108	EM107	EM106	EM105	EM104	EM103	EM102	EM101	EM100	EM99	EM98	EM97	EM96
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 7-7. Event Mask Registers (EVTMASK*n*) Field Descriptions

Bit	Field	Value	Description
31-0	EM*yyy*		Disables event EVT*yyy* from being used as input to the event combiner.
		0	EVT*yyy* will be combined.
		1	EVT*yyy* is disabled from being combined.

7.5.2.2 Masked Event Flag Registers (MEVTFLAG*n*)

The event combiner provides a set of four masked event flag registers (a masked view of the event flag registers).

The masked event flag registers (MEVTFLAG*n*) are shown in Figure 7-31 through Figure 7-34 and described in Table 7-8.

Figure 7-31. Masked Event Flag Register 0 (MEVTFLAG0)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Figure 7-32. Masked Event Flag Register 1 (MEVTFLAG1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Figure 7-33. Masked Event Flag Register 2 (MEVTFLAG2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Figure 7-34. Masked Event Flag Register 3 (MEVTFLAG3)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF	MEF
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 7-8. Masked Event Flag Registers (MEVTFLAG*n*) Field Descriptions

Bit	Field	Value	Description

31-0	MEF*yyy*	0-FFFF FFFFh	Displays content of EF*yyy* when EM*yyy* = 0 in the event mask registers (EVTMASK*n*).
			If (EM*yyy* = 0)
			MEF*yyy* = EF*yyy*
			Else
			MEF*yyy* = 0

7.5.3 CPU Interrupt Selector Registers

7.5.3.1 Interrupt Mux Registers (INTMUX*n*)

The interrupt selector contains interrupt mux registers that allow you to program the source for each of the 12 available CPU interrupts.

The interrupt mux registers are shown in Figure 7-35 through Figure 7-37 and described in Table 7-9.

Figure 7-35. Interrupt Mux Register 1 (INTMUX1)

31	30	24	23	22	16
Reserved	INTSEL7		Reserved	INTSEL6	
R-0	R/W-7h		R-0	R/W-6h	

15	14	8	7	6	0
Reserved	INTSEL5		Reserved	INTSEL4	
R-0	R/W-5h		R-0	R/W-4h	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 7-36. Interrupt Mux Register 2 (INTMUX2)

31	30	24	23	22	16
Reserved	INTSEL11		Reserved	INTSEL10	
R-0	R/W-Bh		R-0	R/W-Ah	

15	14	8	7	6	0
Reserved	INTSEL9		Reserved	INTSEL8	
R-0	R/W-9h		R-0	R/W-8h	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 7-37. Interrupt Mux Register 3 (INTMUX3)

31	30	24	23	22	16
Reserved	INTSEL15		Reserved	INTSEL14	
R-0	R/W-Fh		R-0	R/W-Eh	

15	14	8	7	6	0
Reserved	INTSEL13		Reserved	INTSEL12	
R-0	R/W-Dh		R-0	R/W-Ch	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 7-9. Interrupt Mux Registers (INTMUX*n*) Field Descriptions

Field	Value	Description
INTSEL*nn*	0-7Fh	Contains the number of the event that maps to CPUINT*nn*.

7.5.3.2 Interrupt Exception Status Register (INTXSTAT)

The interrupt exception status register (INTXSTAT) provides information to determine what caused the exception that was generated. The INTXSTAT register holds the CPU interrupt and the system event number of the dropped event.

The interrupt exception status register (INTXSTAT) is shown in Figure 7-38 and described in Table 7-10.

Figure 7-38. Interrupt Exception Status Register (INTXSTAT)

31	24	23	16
SYSINT		CPUINT	
R-0		R-0	

15	1	0
Reserved		DROP
R-0		R-0

LEGEND: R = Read only; -*n* = value after reset

Table 7-10. Interrupt Exception Status Register (INTXSTAT) Field Descriptions

Bit	Field	Value	Description
31-24	SYSINT	0-FFh	System Event number
		0-7Fh	EVT0 to EVT128
		80h-FFh	Reserved
23-16	CPUINT	0-FFh	CPU interrupt number
		0-Fh	CPUINT0 to CPUINT15
		10h-FFh	Reserved
15-1	Reserved	0	Reserved
0	DROP		Dropped event flag
		0	No events dropped
		1	Event was dropped by the CPU

7.5.3.3 Interrupt Exception Clear Register (INTXCLR)

The interrupt exception status is cleared through the exception clear register, which is essentially a single clear bit, as shown below. A new IDROPx event can only be detected by the hardware after the status clears.

The interrupt exception clear register is shown in Figure 7-39 and described in Table 7-11.

Figure 7-39. Interrupt Exception Clear Register (INTXCLR)

31	1
Reserved	
R-0	
0	
CLEAR	
W-0	

LEGEND: R = Read only; W= Write only; -*n* = value after reset

Table 7-11. Interrupt Exception Clear Register (INTXCLR) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	CLEAR		Clears the interrupt exception status.
		0	No effect
		1	Clear interrupt exception status.

7.5.3.4 Dropped Interrupt Mask Register (INTDMASK)

The dropped interrupts that generate the INTERR event can be filtered by a mask register. Those CPU interrupts that are to be ignored by the drop detection hardware can be masked out in the dropped interrupt mask register (INTDMASK).

The dropped interrupt mask register (INTDMASK) is shown in Figure 7-40 and described in Table 7-12.

Figure 7-40. Dropped Interrupt Mask Register (INTDMASK)

31													16
Reserved													
R-0													
15	14	13	12	11	10	9	8	7	6	5	4	3	0
IDM15	IDM14	IDM13	IDM12	IDM11	IDM10	IDM9	IDM8	IDM7	IDM6	IDM5	IDM4	Reserved	
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	

LEGEND: R = Read only; -*n* = value after reset

Table 7-12. Dropped Interrupt Mask Register (INTDMASK) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-4	IDM*nn*		Disables CPUINT*nn* from being detected by the drop detection hardware.
		0	No effect.
		1	CPUINT*nn* ignored by the drop detection hardware.
3-0	Reserved	0	Reserved

7.5.4 CPU Exception Registers

7.5.4.1 CPU Exception Combiner Mask Registers (EXPMASK*n*)

Like the event combiner, the exception combiner has mask registers that are used to gate which events trigger EXCEP.

NOTE: The exception masks for events 0 through 3 are reserved and always masked.

The exception combiner mask register (EXPMASK*n*) is shown in Figure 7-41 through Figure 7-44 and described in Table 7-13.

Figure 7-41. Exception Combiner Mask Register 0 (EXPMASK0)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
XM31	XM30	XM29	XM28	XM27	XM26	XM25	XM24	XM23	XM22	XM21	XM20	XM19	XM18	XM17	XM16
R/W-FFFFh															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XM15	XM14	XM13	XM12	XM11	XM10	XM9	XM8	XM7	XM6	XM5	XM4	XM3	XM2	XM1	XM0
R/W-FFFFh															

LEGEND: R/W = Read/Write; -*n* = value after reset

Figure 7-42. Exception Mask Register 1 (EXPMASK1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
XM63	XM62	XM61	XM60	XM59	XM58	XM57	XM56	XM55	XM54	XM53	XM52	XM51	XM50	XM49	XM48
R/W-FFFFh															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XM47	XM46	XM45	XM44	XM43	XM42	XM41	XM40	XM39	XM38	XM37	XM36	XM35	XM34	XM33	XM32
R/W-FFFFh															

LEGEND: R/W = Read/Write; -*n* = value after reset

Figure 7-43. Exception Mask Register 2 (EXPMASK2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
XM95	XM94	XM93	XM92	XM91	XM90	XM89	XM88	XM87	XM86	XM85	XM84	XM83	XM82	XM81	XM80
R/W-FFFFh															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XM79	XM78	XM77	XM76	XM75	XM74	XM73	XM72	XM71	XM70	XM69	XM68	XM67	XM66	XM65	XM64
R/W-FFFFh															

LEGEND: R/W = Read/Write; -*n* = value after reset

Figure 7-44. Exception Mask Register 3 (EXPMASK3)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
XM127	XM126	XM125	XM124	XM123	XM122	XM121	XM120	XM119	XM118	XM117	XM116	XM115	XM114	XM113	XM112
R/W-FFFFh															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XM111	XM110	XM109	XM108	XM107	XM106	XM105	XM104	XM103	XM102	XM101	XM100	XM99	XM98	XM97	XM96
R/W-FFFFh															

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 7-13. Exception Mask Registers (EXPMASK*n*) Field Descriptions

Bit	Field	Value	Description
31-0	XM*yyy*		Enables event EVT*yyy* from being used in the exception combiner.
		0	EVT*yyy* will be combined.
		1	EVT*yyy* is disabled from being combined.

7.5.4.2 Masked Exception Flag Registers (MEXPFLAG*n*)

The exception combiner provides a set of four masked exception flag registers (a masked view of the exception flag registers).

The masked exception flag registers (MEXPFLAG*n*) are shown in Figure 7-45 through Figure 7-48 and described in Table 7-14.

Figure 7-45. Masked Exception Flag Register 0 (MEXPFLAG0)

31	30	29	28	27	26	25	24
MXF31	MXF30	MXF29	MXF28	MXF27	MXF26	MXF25	MXF24
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
23	22	21	20	19	18	17	16
MXF23	MXF22	MXF21	MXF20	MXF19	MXF18	MXF17	MXF16
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8
MXF15	MXF14	MXF13	MXF12	MXF11	MXF10	MXF9	MXF8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
7	6	5	4	3	2	1	0
MXF7	MXF6	MXF5	MXF4	MXF3	MXF2	MXF1	MXF0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Figure 7-46. Masked Exception Flag Register 1 (MEXPFLAG1)

31	30	29	28	27	26	25	24
MXF63	MXF62	MXF61	MXF60	MXF59	MXF58	MXF57	MXF56
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
23	22	21	20	19	18	17	16
MXF55	MXF54	MXF53	MXF52	MXF51	MXF50	MXF49	MXF48
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8
MXF47	MXF46	MXF45	MXF44	MXF43	MXF42	MXF41	MXF40
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
7	6	5	4	3	2	1	0
MXF39	MXF38	MXF37	MXF36	MXF35	MXF34	MXF33	MXF32
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Figure 7-47. Masked Exception Flag Register 2 (MEXPFLAG2)

31	30	29	28	27	26	25	24
MXF95	MXF94	MXF93	MXF92	MXF91	MXF90	MXF89	MXF88
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
23	22	21	20	19	18	17	16
MXF87	MXF86	MXF85	MXF84	MXF83	MXF82	MXF81	MXF80
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8
MXF79	MXF78	MXF77	MXF76	MXF75	MXF74	MXF73	MXF72
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
7	6	5	4	3	2	1	0
MXF71	MXF70	MXF69	MXF68	MXF67	MXF66	MXF65	MXF64
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Figure 7-48. Masked Exception Flag Register 3 (MEXPFLAG3)

31	30	29	28	27	26	25	24
MXF127	MXF126	MXF125	MXF124	MXF123	MXF122	MXF121	MXF120
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
23	22	21	20	19	18	17	16
MXF119	MXF118	MXF117	MXF116	MXF115	MXF114	MXF113	MXF112
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8
MXF111	MXF110	MXF109	MXF108	MXF107	MXF106	MXF105	MXF104
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
7	6	5	4	3	2	1	0
MXF103	MXF102	MXF101	MXF100	MXF99	MXF98	MXF97	MXF96
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

Table 7-14. Masked Exception Flag Registers (MEXPFLAG*n*) Field Descriptions

Bit	Field	Value	Description

31-0	MXF*yyy*	0-FFFF FFFFh	Displays content of EF*yyy* when XM*yyy* = 0 in the exception mask registers (EXPMASK*n*).
			If (XM*yyy* = 0)
			MXF*yyy* = EF*yyy*
			else
			MXF*yyy* = 0

*7.5.5 Advanced Event Generator Mux Registers (AEGMUX*n*)*

The Advanced Event Generator (AEG) allows any event to act as emulation triggers. The events that are sent to the AEG block are configured in the AEG mux registers (AEGMUX0 and AEGMUX1).

The AEGMUX registers are similar to the interrupt selector registers, in that the event to be passed on is simply encoded into a selector bitfield. The encoded value selects between the available system events (EVT[127:4], combined system events (EVT[3:0], the CPU interrupts (CPUINT[15:4], any interrupt acknowledge (IACK), and exception acknowledge (EACK). The combined events (EVT[3:0] are available and are set as the default events.

NOTE: The AEGMUX0 and AEGMUX1 registers are supported on AET enabled devices only. Refer to your device-specific datasheet to determine if your device supports AET.

The advanced event generator mux registers are shown in Figure 7-49 through Figure 7-50 and described in Table 7-15.

Figure 7-49. Advanced Event Generator Mux Register 0 (AEGMUX0)

31	24	23	16
AEGSEL3		AEGSEL2	
R/W-3h		R/W-2h	
15	**8**	**7**	**0**
AEGSEL1		AEGSEL0	
R/W-1h		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 7-50. Advanced Event Generator Mux Register 1 (AEGMUX1)

31	24	23	16
AEGSEL7		AEGSEL6	
R/W-7h		R/W-6h	
15	**8**	**7**	**0**
AEGSEL5		AEGSEL4	
R/W-5h		R/W-4h	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 7-15. Advanced Event Generator Mux Registers (AEGMUX*n*) Field Descriptions

Bit	Field	Value	Description
31-0	AEGSEL*n*	0-FFh	Advanced event generator (AEG) select.
		0-7Fh	EVT [127:0]: System events 0 to 127
		80-BFh	Reserved
		C0h	EXCEP: CPU Exception
		C1h	NMI: Non-maskable CPU interrupt
		C2-C3h	Reserved
		C4-CFh	CPUINT [15:4]: CPU interrupts
		D0-DFh	Reserved
		E0h	IACK: Interrupt acknowledge (for any interrupt)
		E1h	EACK: Exception acknowledge
		E2-E3h	Reserved
		E4-EFh	IACK [15:4]: Interrupt acknowledge for specific CPU interrupts
		F0-FFh	Reserved

7.5.6 Privilege and Interrupt Controller Registers

The C64x+ architecture provides memory protection support.

Table 7-16 summarizes which interrupt controller registers are accessible according to role.

Table 7-16. Permissions for Interrupt Controller Registers

Register	Supervisor	User
EVTFLAGx	R	R
EVTCLRx	W	R
EVTSETx	W	R
EVTMASKx	R/W	R
MEVTFLAGx	R	R
EXPMASKx	R/W	R
MEXPFLAGx	R	R
INTMUXx	R/W	R
AEGMUXx	R/W	R
INTSTAT	R	R
INTXCLR	W	R
INTDMASK	R/W	R

Memory Protection

Topic Page

8.1 Introduction 188
8.2 Terms and Definitions 188
8.3 Memory Protection Architecture 188
8.4 Memory Protection Architecture Registers 191
8.5 Permission Checks on Accesses to Memory Protection Registers 199

8.1 Introduction

8.1.1 Purpose of the Memory Protection

Memory protection provides many benefits to a system. Memory protection functionality can:

- Protect operating system data structures from poorly behaving code.
- Aid in debugging by providing greater information about illegal memory accesses.
- Allow the operating system to enforce clearly defined boundaries between supervisor and user mode accesses, leading to greater system robustness.

The C64x+ megamodule memory protection architecture provides these benefits through a combination of CPU privilege levels and a memory system permission structure.

8.1.2 Privilege Levels

The privilege of a thread determines what level of permissions that thread might have.

Code running on the CPU executes in one of two privilege modes: supervisor mode or user mode. Supervisor code is considered more trusted than user code. Examples of supervisor threads include operating system kernels and hardware device drivers. Examples of user threads include vocoders and end applications.

Supervisor mode is generally granted access to peripheral registers and the memory protection configuration. User mode is generally confined to the memory spaces that the OS specifically designates for its use.

CPU accesses as well as internal DMA and other accesses have a privilege level associated with them. The CPU privilege level is determined as described above. The Internal DMA accesses that are initiated by the CPU inherit the CPU's privilege level at the time they are initiated.

8.2 Terms and Definitions

Refer to Appendix A of this document for a detailed definition of the terms that are used in this chapter. Appendix A describes general terms that are used throughout the this reference guide.

8.3 Memory Protection Architecture

8.3.1 Memory Protection Pages

The C64x+ memory protection architecture divides the DSP internal memory (L1P, L1D, L2) into *pages*. Each page has an associated set of permissions. Section 8.3.2 and its subsections describe the permission sets.

Memories typically have power-of-2 page sizes. The sizes of the L1 and the L2 memory pages are specific to the device. Refer to the device-specific data manual for more information.

8.3.2 Permission Structure

The memory protection architecture defines a per-page permission structure with two permission fields in a 16-bit permission entry. Figure 8-1 shows the structure of a permission entry.

Figure 8-1. Permission Fields

31															16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Allowed IDs								Reserved		Access Types					
AID5	AID4	AID3	AID2	AID1	AID0	AIDX	LOCAL			SR	SW	SX	UR	UW	UX

8.3.2.1 Requestor-ID Based Access Controls

Each requestor on the device has an N-bit code associated with it that identifies it for privilege purposes. This ID accompanies all memory accesses and IDMAs made on behalf of that requestor. That is, when a requestor triggers an IDMA transfer directly by writing to IDMA registers, the IDMA engine will provide that ID alongside the transfer. Each CPU and every mastering peripheral (RapidIO, HPI, and EMAC) has an ID. Multiple system masters may share an ID in the same device. Each memory protection entry has an allowed ID field associated with it that indicates which requestors may access the given page. The memory protection hardware maps the IDs of all the possible requestors to bits in the allowed IDs field in the memory protection entries. The allowed IDs field discriminates between various CPUs, non-CPU requestors, and a given CPU's accesses to its own local memories.

- AID0 through AID5 map small-numbered IDs to allowed ID bits.
- An additional allowed ID bit, AIDX, captures access made by higher-numbered PrivIDs.
- The LOCAL bit treats CPU accesses to its local L1s and L2 specially.

Figure 8-2 illustrates and Table 8-1 describes the allowed IDs bit field.

Figure 8-2. Allowed IDs Bit Fields

15	14	13	12	11	10	9	8
AID5	AID4	AID3	AID2	AID1	AID0	AIDX	LOCAL

The allowed ID field is 8 bits.

When set to 1, the AID bit grants access to the corresponding ID. When cleared to 0, the AID bit denies access to the corresponding requestor.

Table 8-1. Allowed IDs Bit Field Descriptions

Bit	Field	Description
15	AID5	Allow accesses from ID = 5
14	AID4	Allow accesses from ID = 4
13	AID3	Allow accesses from ID = 3
12	AID2	Allow accesses from ID = 2
11	AID1	Allow accesses from ID = 1
10	AID0	Allow accesses from ID = 0
9	AIDX	Allow accesses from ID> = 6
8	LOCAL	Allow access from CPU to its local memories (L1/L2 only)

The above ID assignments for bits AID0 through AID5 apply to all IDMA and CPU memory accesses other than to the CPU's local L1 and L2 memories. The LOCAL bit governs CPU accesses to its own local L1 and L2 memories. The AIDX bit maps to IDs that do not have dedicated AID bits associated with them.

8.3.2.2 Request-Type Based Permissions

The memory protection model defines three fundamental functional access types: read, write, and execute. Read and write refer to data accesses -- accesses originating via the load/store units on the CPU or via the IDMA engine. Execute refers to accesses associated with program fetch.

The memory protection model allows controlling read, write, and execute permissions independently for both user and supervisor mode. This results in 6 permission bits, shown in Table 8-2.

Table 8-2. Request Type Access Controls

Bit	Field	Description
5	SR	Supervisor may read
4	SW	Supervisor may write
3	SX	Supervisor may execute
2	UR	User may read
1	UW	User may write
0	UX	User may execute

For each bit, a 1 permits the access type, and a 0 denies it. Thus UX = 1 means that User Mode may execute from the given page. The memory protection architecture allows you to specify all six of these bits separately. 64 different encodings are permitted altogether, although programs might not use all of them.

8.3.3 Invalid Accesses and Exceptions

When it encounters an invalid access, the memory protection hardware has two distinct duties:

- Prevent the access from occurring.
- Report the error to the operating environment.

Invalid accesses are those memory accesses which require greater permissions than those specified for the page or register involved. The following sections cover the behavior of the memory protection in the presence of invalid accesses.

8.3.3.1 Handling Invalid Accesses

When presented with an invalid access, the memory protection prevents the requestor from making the access and will make sure that the memory being protected does not change its state due to the invalid access.

8.3.3.2 Exception Generation

Upon detecting an invalid access, the memory protection hardware reports the error to the operating environment.

8.4 Memory Protection Architecture Registers

The memory protection architecture defines several sets of memory-mapped registers (MMRs). Each hardware block that implements memory protection architecture (MPA), implements these MMRs as part of its own register set. As a result, these MMRs reside within its configuration register address space.

The peripherals that implement the MMRs govern accesses to those MMRs.

The MMRs fall into three main categories:

- Memory Protection Page Attribute (MPPA) Registers: These registers store the permissions associated with each protected page. These are defined in Section 8.4.1.
- Memory Protection Fault (MPFxR) Registers: Each peripheral that generates memory protection faults provides MPFAR, MPFSR, and MPFCR registers for recording the details of the fault. These are defined in Section 8.4.2 and Section 8.4.2.1.
- Memory Protection Lock (MPLK) Registers: When engaged, the lock disables all updates to the memory protection entries for that peripheral. The MPLK register is defined in Section 8.4.3.

  Because each memory implements its own memory protection registers, refer to the device-specific data manual for more information about the memory map.

Table 8-3 lists the memory-mapped registers for the memory protection architecture. See the device-specific data manual for the memory address of these registers.

Table 8-3. Memory Protection Architecture Registers

Acronym	Register Description	Section
MPPA	Memory Protection Page Attribute	Section 8.4.1
MPFAR	Memory Protection Fault Address Register	Section 8.4.2
MPFSR	Memory Protection Fault Status Register	Section 8.4.2
MPFCR	Memory Protection Fault Command Register	Section 8.4.2
MPLK	Memory Protection Lock Registers	Section 8.4.3

8.4.1 Memory Protection Page Attribute (MPPA) Registers

Each memory that implements a notion of configurable memory protection pages provides a set of memory protection page attribute (MPPA) registers. One MPPA register covers each page that the peripheral implements. These registers typically appear in a contiguous block within the memory's MMR memory map.

Each MPPA register occupies 32 bits in the memory map, but only 16 of these bits are used. Section 8.3.2 describes the layout and definition of the MPPA register fields.

The reset value of the MPPA register is device-dependant.

8.4.2 Memory Protection Fault Registers (MPFAR, MPFSR, MPFCR)

All memories that implement the memory protection architecture and that generate exceptions provide a set of memory protection fault registers to report the details of a memory protection violation.

The C64x+ memory protection architecture (MPA) specifies three registers: memory protection fault address register (MPFAR), memory protection fault status register (MPFSR), and memory protection fault command register (MPFCR).

Memories that implement the memory protection architecture, but cannot generate exceptions, do not implement these registers.

8.4.2.1 Memory Access Protection Fault Registers

When a given piece of memory protection hardware detects a privilege violation, it captures some basic information about the violation as part of the exception-triggering process. Specifically, it captures the address of the fault, and the type of access that generated the fault.

The hardware records the address of the fault in the memory's memory protection fault address register (MPFAR). It records the rest of the information regarding the fault in the memory's memory protection fault status register (MPFSR). Software can write to the memory protection fault command register (MPFCR) to clear the fault.

8.4.2.1.1 Memory Protection Fault Address Register (MPFAR)

The memory protection fault address register (MPFAR) is shown in Figure 8-3 and described in Table 8-4.

Figure 8-3. Memory Protection Fault Address Register (MPFAR)

31	0
Faulting Address	
R-0	

LEGEND: R = Read only; -n = value after reset

Table 8-4. Memory Protection Fault Address Register (MPFAR) Field Descriptions

Bit	Field	Value	Description
31-0	Faulting Address	0-FFFF FFFFh	Address of the fault.

8.4.2.1.2 Memory Protection Fault Status Register (MPFSR)

The memory protection fault status register (MPFSR) is shown in Figure 8-4 and described in Table 8-5.

Figure 8-4. Memory Protection Fault Status Register (MPFSR)

31	16
Reserved	
R-0	

15	9	8	7	6	5	4	3	2	1	0
FID		LOCAL	Reserved		SR	SW	SX	UR	UW	UX
R-0		R-0	R-0		R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

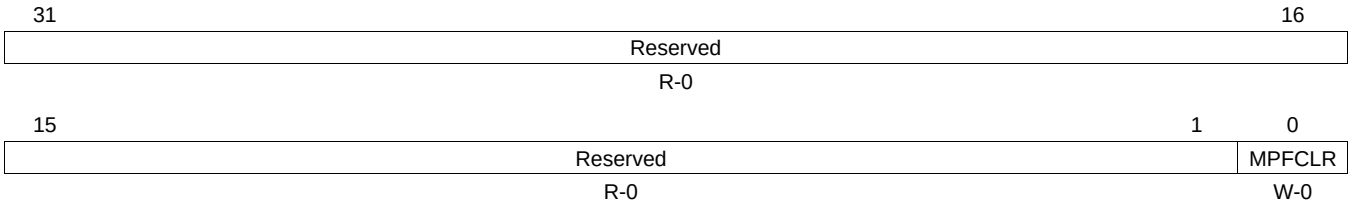
Table 8-5. Memory Protection Fault Status Register (MPFSR) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved.
15-9	FID	1Fh	Bits 6:0 ID of faulting requestor. If ID is narrower than 7 bits, the remaining bits return 0. If ID is wider than 7 bits, the additional bits are truncated. FID = 0 if LOCAL = 1.
8	LOCAL	0-1	Access was a “LOCAL” access.
7-6	Reserved	0	Reserved.
5	SR	0-1	When set, indicates a supervisor read request.
4	SW	0-1	When set, indicates a supervisor write request.
3	SX	0-1	When set, indicates a supervisor program fetch request.
2	UR	0-1	When set, indicates a user read request.
1	UW	0-1	When set, indicates a user write request.
0	UX	0-1	When set, indicates a user program fetch request.

8.4.2.1.3 Memory Protection Fault Command Register (MPFCR)

The memory protection fault command register (MPFCR) is shown in Figure 8-5 and described in Table 8-6.

Figure 8-5. Memory Protection Fault Command Register (MPFCR)



LEGEND: R = Read only; W = Write only; -n = value after reset; -*n* = value after reset

Table 8-6. Memory Protection Fault Command Register (MPFCR) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved.
0	MPFCLR		Command to clear the L1DMPFAR register.
		0	No effect.
		1	Clear the L1DMPFAR and the L1DMPFCR registers.

MPFAR records the address of the protection violation MPFSR records the access type, in a register formatted similarly to the memory protection page attribute register. The MPFCLR register includes a single command bit for clearing the MPFAR and the MPFCLR registers.

Caches generate two special access types (line fills and writebacks) that are distinct from normal functional accesses. The protection hardware indicates faults on cache transactions by encoding special patterns into the access type fields.

- Faulting line fill sets SR = SW = = UR = UW = UX = 1.
- Faulting victim writeback sets SW = UW = 1.

You can decode a memory protection fault as follows using this scheme in software:

- If the LOCAL field is set, the request was a local CPU request to its own memories. Otherwise, the ID of the faulting requestor is in bits 9 through 15 of the fault status register.
- The value of the access type field (SR, SW, SX, UR, UW, and UX) indicates the type of access that was at fault, as shown in Table 8-7.

Table 8-7. Interpretation of MPFSR Access Type Field

SR	SW	SX	UR	UW	UX	Meaning
1	0	0	0	0	0	Fault due to supervisor read
0	**1**	0	0	0	0	Fault due to supervisor write
0	0	**1**	0	0	0	Fault due to supervisor program fetch
0	0	0	**1**	0	0	Fault due to user read
0	0	0	0	**1**	0	Fault due to user write
0	0	0	0	0	**1**	Fault due to user program fetch
1	**1**	**1**	**1**	**1**	**1**	Fault due to cache line fill
0	**1**	0	0	**1**	0	Fault due to cache victim writeback
Others						Reserved -- may be defined by endpoint

The megamodule only generates the line-fill and victim writeback encodings if another master caches the content of the megamodule, and experiences a fault while doing so.

Each memory protection block captures its own memory protection fault information. Thus, each potential memory protection exception source has an associated MPFAR/MPFSR/MPFCR register set.

The MPFAR and MPFSR registers only store information for one fault. As a result of the fault, an exception generates. The fault information is held until software clears it by writing to MPFCR.

The supervisor clears the recorded fault by writing to a 1 to the MPFCLR (bit 0) in the MPFCR register. Writing a 1 to this bit clears both the MPFAR and the MPFSR registers. The MPFAR and MPFCR registers do not respond to writes. Once the supervisor clears the fault, the hardware records the next protection violation and signal an exception when it occurs. Writing a 1 to any other bit of the MPFCR register has no effect on the memory protection registers. Writing a 0 to the MPFCLR field in the MPFCR register also has no effect.

The various distinct memory protection blocks do not directly coordinate with each other. Some operations (such as cache line fills) can generate an exception at the endpoint and in the cache hierarchy. Therefore, a single invalid memory access may generate multiple exceptions in different blocks before a CPU acknowledges even the first exception. Nonetheless, each individual memory generates no more than one exception until the CPU clears that memory's MPFAR and MPFSR registers.

8.4.3 Memory Protection Lock Registers (MPLK*n*)

As an additional layer of security, the memory protection architecture defines a hardware "protection lock." Hardware locks provide an additional layer over all other access controls to a given memory's protection registers.

Devices that implement hardware locks on their protection entries implement the six registers shown in Figure 8-6 through Figure 8-11.

The memory protection lock registers are shown in Figure 8-6 through Figure 8-10 and described in Table 8-8.

Figure 8-6. Memory Protection Lock Register (MPLK0)

31	0
Lock Bits 31:0	
W-x	

LEGEND: W = Write only; -*x* = value is indeterminate, see your device-specific data manual

Figure 8-7. Memory Protection Lock Register (MPLK1)

31	0
Lock Bits 63:32	
W-x	

LEGEND: W = Write only; -*x* = value is indeterminate, see your device-specific data manual

Figure 8-8. Memory Protection Lock Register (MPLK2)

31	0
Lock Bits 95:64	
W-x	

LEGEND: W = Write only; -*x* = value is indeterminate, see your device-specific data manual

Figure 8-9. Memory Protection Lock Register (MPLK3)

31	0
Lock Bits 127:96	
W-x	

LEGEND: W = Write only; -*x* = value is indeterminate, see your device-specific data manual

8.4.3.1 Memory Protection Lock Command Register (MPLKCMD)

The memory protection lock command register (MPLKCMD) is shown in Figure 8-10 and described in Table 8-8.

Figure 8-10. Memory Protection Lock Command Register (MPLKCMD)

31	16
Reserved	
R-0	

15	3	2	1	0
Reserved		KEYR	LOCK	UNLOCK
R-0		W-0	W-0	W-0

LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 8-8. Memory Protection Lock Command Register (MPLKCMD) Field Descriptions

Bit	Field	Value	Description
31-3	Reserved	0	Reserved.
2	KEYR		Reset status.
		0	No effect.
		1	Reset status.
1	LOCK		Interface to complete a lock sequence.
		0	No effect.
		1	Locks the lock provided that the software executed the sequence correctly.
0	UNLOCK		Interface to complete an unlock sequence.
		0	No effect.
		1	Unlocks the lock provided that software executed the sequence correctly.

8.4.3.2 Memory Protection Lock Status Register (MPLKSTAT)

The memory protection lock status register (MPLKSTAT) is shown in Figure 8-11 and described in Table 8-9.

Figure 8-11. Memory Protection Lock Status Register (MPLKSTAT)

31	16
Reserved	
R-0	

15	1	0
Reserved		LK
R-0		R-0

LEGEND: R = Read only; -*n* = value after reset

Table 8-9. Memory Protection Lock Status Register (MPLKSTAT) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	LK		Indicates the lock's current status.
		0	Lock is disengaged.
		1	Lock is engaged.

The lock may exist in one of two states: locked or unlocked. Reset places the lock in the unlocked state via the LK field in the MPLKSTAT register.

Software may engage the lock as long as the lock is currently unlocked. To engage the lock, the application must perform the following steps exactly:

1. Write a 1 to the KEYR field of the MPLKCMD register. This resets some internal status for the MPLK0 through MPLK3 registers.
2. Write the key to MPLK0 through MPLK3. All four registers must be written exactly once. They may be written in any order.
3. Write a 1 to the LOCK field of the MPLKCMD register. This engages the lock.

If programs follow this sequence, the memory protection hardware engages the lock. The hardware performs the following actions when it engages the lock:

- Sets the LK field of the MPLKSTAT register to 1.
- Establishes the written key (or some subset) as the "unlock" key
- Blocks future writes to all MPPA and MPCFG registers for this memory

The hardware signals an exception if it detects an incorrect lock sequence. The hardware reports the address of the MPLK register written at the point of failure as the exception address in the MPFAR register.

Software executes a sequence similar to the locking sequence to unlock the peripheral's protection registers when they are currently locked:

1. Write a 1 to the KEYR field in the MPLKCMD register. This resets some internal status for the MPLK0 through the MPLK3 registers.
2. Write the unlock key to MPLK0 through the MPLK3 registers. The hardware compares the written value with the stored key value. Software must write to all four registers exactly once. The writes can arrive in any order.
3. Write a 1 to the UNLOCK field in the MPLKCMD register. If the key written in step 2 matches the stored key, the hardware disengages the lock. If the key written in step 2 does not match, the hardware signals an exception. The hardware reports the fault address as the address of the MPLKCMD register.

8.4.4 Keys Shorter than 128 Bits

In some devices, memories may implement keys shorter than 128 bits. In this case, applications that manipulate the lock should write the full 128 bit key when locking and unlocking the lock, even if the hardware does not take the full 128 bits into account.

8.5 Permission Checks on Accesses to Memory Protection Registers

Memories implementing the memory protection architecture implement permission checks on the memory protection registers. Table 8-10 summarizes these checks:

Table 8-10. Allowed Accesses to Memory Protection Registers

	Supervisor		User	
Register Set	**Read**	**Write**	**Read**	**Write**
MPPAx	Always	Unlocked	Always	Never
MPFAR, MPFSR	Always	Never	Always	Never
MPFCR	Never	Always	Never	Always
MPLK0-MPLK3	Never	During lock/unlock sequence	Never	Never
MPLKSTAT	Always	Never	Always	Never
MPLKCMD	Never	Start/end of lock/unlock	Never	Never

Power-Down Controller

Topic **Page**

9.1 Introduction **202**
9.2 Power-Down Features **202**
9.3 Power-Down Controller Command Register (PDCCMD) **204**

9.1 Introduction

This section provides the purpose and discusses the features of the power-down controller.

9.1.1 C64x+ Megamodule Power Down-Management

The C64x+ megamodule supports the ability to power-down various parts of the C64x+ megamodule. You can power-down the entire C64x+ megamodule using the C64x+ megamodule power-down controller. You can use these features to design systems for lower rate system power requirements.

NOTE: Peripherals located outside of the C64x+ megamodule may also provide their own power-down capabilities. These are not covered in this chapter, since they are outside the scope of this document.

9.1.2 Power-Down Capabilities Overview

Table 9-1 lists the power-down features available in the C64x+ megamodule and a brief description of how and when they are applied:

Table 9-1. C64x+ Megamodule Power-Down Features

Power-Down Feature	How/When Applied
L1P memory	During SPLOOP instruction execution
L2 memory	Software programmable via L2 control registers
Cache control hardware	When caches are disabled
CPU	Upon issuing an IDLE instruction
Entire C64x+ megamodule	Enabled by PDC and IDLE

9.2 Power-Down Features

9.2.1 L1P Memory

L1P memory is powered-down dynamically during the execution of instructions from the SPLOOP buffer. This feature is enabled automatically and is transparent to you. Upon completion of the SPLOOP instruction, the CPU resumes fetching from the L1P memory and the RAMs are awakened. In other words, the L1P is powered-down when it is not being accessed. For more information about the SPLOOP instruction, see the *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide* (SPRU732) .

NOTE: This L1P is also powered-down when the entire C64x+ megamodule is powered-down, as described in Section 9.2.5.

9.2.2 L2 Memory

L2 memory supports run-time power-down capability under software control. You can use this to temporarily power-down portions of the L2 that are not needed.

The L2 memory is divided into four logical pages (two per L2 port) from a power management perspective. You can power-down each page independently by programming the appropriate field in the level 2 power-down sleep register (L2PDSLEEP). Similarly, level 2 power-down wake register (L2PDWAKEn) allows for powering-up each page; L2 also wakes a page if it is accessed.

The power-down and wake-up procedures are further described in Section 4.5.

NOTE: This L2 is also powered-down when the entire C64x+ megamodule is powered-down (as described in Section 9.2.5).

9.2.3 Cache Power-Down Modes

When the L1D, L1P, or L2 caches are not enabled, they are kept in power-down mode.

NOTE: The three cache controllers are powered-down when the entire C64x+ megamodule is powered-down (as described in Section 9.2.5).

9.2.4 CPU Power-Down

While technically outside of the scope of this document, the CPU can be powered-down by issuing an IDLE instruction. The CPU is awakened via interrupt(s). For additional information on the IDLE instruction, see the *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide* (SPRU732) .

The IDLE instruction is also used as part of the procedure for powering-down the entire C64x+ megamodule, as described in Section 9.2.5.

9.2.5 C64x+ Megamodule Power-Down

NOTE: Powering-down the C64x+ megamodule as described in this section is often called *static* power-down. This term is used to describe this power-down mode since this mode is often used for longer periods of time. The term *dynamic* power-down used in this chapter implies that the power-down mode is used for limited periods of time.

The entire C64x+ megamodule can be powered-down using the following procedure. Other than the options previously specified, it is not possible to power-down only part of the C64x+ megamodule. Powering-down the C64x+ megamodule is completely under software control by programming the megamodule power-down (MEGPD) bit in the power-down controller command register (PDCCMD).

The following software sequence is required to power-down the C64x+ megamodule:

1. Enable power-down by setting the MEGPD field to 1 in the PDCCMD register.
2. Enable the CPU interrupt(s) that you want to wake-up the C64x+ megamodule. Disable all other interrupts.
3. Execute the IDLE instruction.

The C64x+ megamodule stays in a power-down state until awakened by the interrupt(s) that are enabled in step 2.

If a DMA access occurs to the L1D, L1P, or L2 memory while the C64x+ megamodule is powered-down, the PDC wakes all three memory controllers. When the DMA access has been serviced, the PDC will again power-down the memory controllers.

9.2.6 Miscellaneous Power-Down

9.2.6.1 Externally-Requested Power-Down

It may be desirable in some systems for the C64x+ megamodule to respond to an externally-driven power-down request. This can be accomplished, but only under CPU control, using the procedure described in Section 9.2.5.

External power-down requests are typically accomplished by using external hardware interrupts/exceptions. The interrupt service routine could follow the C64x+ megamodule power-down procedure to honor the external request.

9.2.6.2 C62x/C64x/C67x DSP Power-Down Modes

The power-down modes found in legacy devices (set through the control status register (CSR) in the CPU) are not supported within the C64x+ megamodule. Since these signals are exported to the C64x+ megamodule boundary, some C64x+ devices may provide additional power-down features that make use of the PWRD bits in the CSR register. See the device-specific data manual for more information about these features.

9.3 Power-Down Controller Command Register (PDCCMD)

Use the power-down command register (PDCCMD) to program the PDC (located at address 0181 0000h). By setting the MEGPD bit to 1 in the PDCCMD register, the C64x+ megamodule global static power-down mode is enabled; when the MEGPD register is set to 1, the C64x+ megamodule global static power-down mode is activated when the CPU enters the idle state. PDCCMD is only writeable when the CPU is in supervisor mode; PDCCMD is readable regardless of supervisor/user status.

The power-down controller command register (PDCCMD) is shown in Figure 9-1 and described in Table 9-2.

Figure 9-1. Power-Down Controller Command Register (PDCCMD)

31	17	16
Reserved		MEGPD
R-0		R/W-0

15	0
Reserved	
R-FFFFh	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 9-2. Power-Down Controller Command Register (PDCCMD) Field Descriptions

Bits	Field	Value	Description
31-17	Reserved	0	Reserved.
16	MEGPD		Power-down during IDLE
		0	Normal operation. Do not power-down the CPU or the C64x+ megamodule when the CPU is IDLE.
		1	Sleep mode. Power-down the CPU and the C64x+ megamodule when the CPU enters the IDLE state.
15-0	Reserved	1	Reserved. These bits are always read as 1.

Table 9-3 summarizes who may access the power-down controller command register.

Table 9-3. Permissions for PDC Command Register

Register	Supervisor	User
PDCCMD	R/W	R

Miscellaneous

Topic **Page**

10.1 Introduction 208
10.2 Megamodule Revision ID Register (MM_REVID) 208
10.3 Bus Error Register (BUSERR) 209
10.4 Bus Error Details Register (BUSERRCLR) 210

10.1 Introduction

Table 10-1 lists miscellaneous memory-mapped registers.

Table 10-1. Miscellaneous Registers

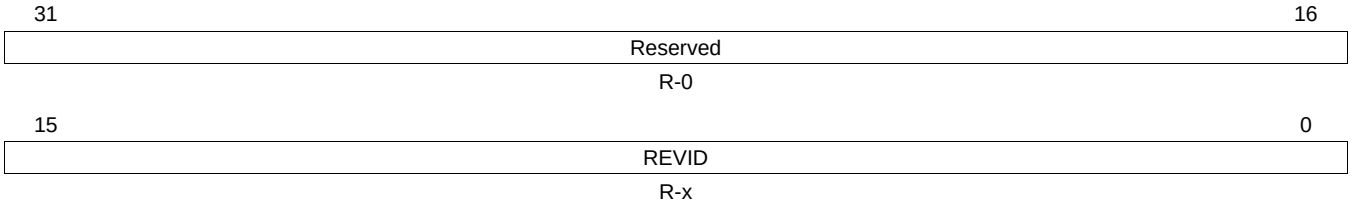
Address	Acronym	Register Description	Section
0181 2000h	MM_REVID	Megamodule Revision ID.	Section 10.2
0182 0400h	BUSERR	Bus Error Status	Section 10.3
0182 0404h	BUSERRCLR	Bus Error Clear	Section 10.4

10.2 Megamodule Revision ID Register (MM_REVID)

The C64x+ megamodule revision ID register (MM_REVID) provides information about the revision of the megamodule.

The megamodule revision ID register (MM_REVID) is shown in Figure 10-1 and described in Table 10-2.

Figure 10-1. Megamodule Revision ID Register (MM_REVID)



LEGEND: R = Read only; -n = value after reset; -x, value is indeterminate, see your device-specific data manual

Table 10-2. Megamodule Revision ID Register (MM_REVID) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved	0	Reserved bit locations.
15-0	REVID		Revision of the megamodule version implemented on the device. The megamodule revision is dependant on the silicon revision being that you are using. For more information, see your device-specific data manual.

10.3 Bus Error Register (BUSERR)

The bus error register (BUSERR) signals errors for external transactions on either the MDMA bus or the CFG bus.

The bus error register (BUSERR) is shown in Figure 10-2 and described in Table 10-3.

Figure 10-2. Bus Error Register (BUSERR)

31 29	28 16
ERR	Reserved
R-0	R-0

15 12	11 8	7 3	2 0
Reserved	XID	Reserved	STAT
R-0	R-0	R-0	R-0

LEGEND: R = Read only; -*n* = value after reset

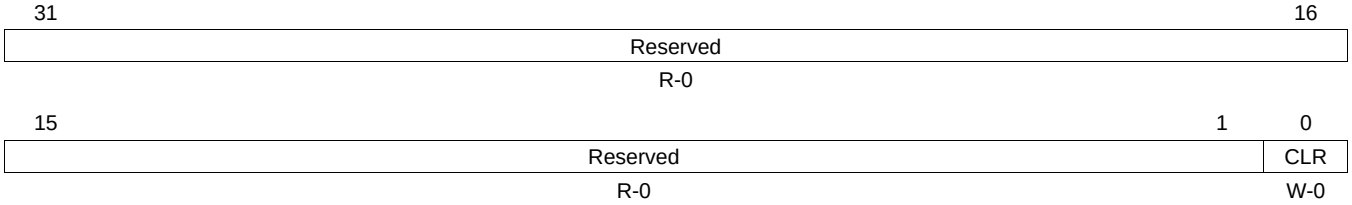
Table 10-3. Bus Error Register (BUSERR) Field Descriptions

Bit	Field	Value	Description
31-29	ERR	0-7h	Error detected
		0	No error
		1h	MDMA read status error detected
		2h	MDMA write status error detected
		3h	CFG read status error detected
		4h	CFG write status error detected
		5h-7h	Reserved
28-12	Reserved	0	Reserved
11-8	XID	0-Fh	Transaction ID
			Stores the transaction ID (RID or WID) when a read or write error is detected.
7-3	Reserved	0	Reserved
2-0	STAT	0-7h	Transaction status
		0	Success (should not cause error to be latched), or unrecognized RID/WID (should cause error to be latched)
		1h	Addressing error
		2h	Privilege error
		3h	Timeout error
		4h	Data error
		5h-6h	Reserved
		7h	Exclusive - operation failure

10.4 Bus Error Details Register (BUSERRCLR)

The bus error details register (BUSERRCLR) is shown in Figure 10-3 and described in Table 10-4.

Figure 10-3. Bus Error Details Register (BUSERRCLR)



LEGEND: R = Read only; W = Write only; -*n* = value after reset

Table 10-4. Bus Error Details Register (BUSERRCLR) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	CLR		Clear register
		0	Writes have no effect
		1	Writing a 1 to the CLR register clears all bits in the BUSERR register. Once an error is detected, you must clear the error register before additional errors can be detected and stored.

General Terms and Definitions

Table A-1 lists the general terms used throughout this document.

Table A-1. List of General Terms and Definitions

Term	Definition
C64x+	Generic name for the new C6000 DSP architecture.
C64x+ CPU	Designates the CPU hardware (functional units and registers).
C64x+ megamodule	Includes the C64x+ CPU plus all the supporting hardware for memory, bandwidth management, interrupt, memory protection, and power-down support.
CFG	External configuration space, includes the memory-mapped registers outside the C64x+ megamodule.
EMC	Extended memory controller. The EMC is a bridge from the C64x+ megamodule to the external world. EMC implements the ports that communicate with the external DMA. The EMC clock is half the CPU clock.
IDMA	Internal DMA. It is a DMA engine that is local to the C64x+ megamodule. It allows transfer of data between memories local to the C64x+ megamodule (L1P, L1D, L2) and the external configuration space.
L1D	Generic name for the level 1 data memory. This term may refer to the memory itself or the memory controller.
L1P	Generic name for the level 1 program memory. This term may refer to the memory itself or the memory controller.
L2	Generic name for the level 2 memory. This term may refer to the memory itself or the memory controller.

Cache Terms and Definitions

Table B-1 lists the cache-related terms used throughout this document that relate to the C64x+ memory architecture.

Table B-1. List of Cache-Related Terms and Definitions

Term	Definition
Allocation	The process of finding a location in the cache to store newly cached data. This process can include *evicting* data that is presently in the cache to make room for the new data.
Associativity	The number of *line frames* in each set. This is specified as the number of *ways* in the cache.
Capacity miss	A cache *miss* that occurs because the cache does not have sufficient room to hold the entire working set for a program. Compare with *compulsory miss* and *conflict miss*.
Clean	A cache *line* that is *valid* and that has not been written to by upper levels of memory or the CPU. The opposite state for a valid cache line is *dirty*.
Coherence	Informally, a memory system is coherent if any read of a data item returns the most recently written value of that data item. This includes accesses by the CPU and the EDMA.
Compulsory miss	Sometimes referred to as a *first-reference miss*. A compulsory miss is a cache *miss* that must occur because the data has had no prior opportunity to be allocated in the cache. Typically, compulsory misses for particular pieces of data occur on the first access of that data. However, some cases can be considered compulsory even if they are not the first reference to the data. Such cases include repeated write misses on the same location in a cache that does not *write allocate*, and cache misses to non-cacheable locations. Compare with *capacity miss* and *conflict miss*.
Conflict miss	A cache *miss* that occurs due to the limited *associativity* of a cache, rather than due to capacity constraints. A *fully-associative cache* is able to allocate a newly cached *line* of data anywhere in the cache. Most caches have much more limited associativity (see *set-associative cache*), and so are restricted in where they may place data. This results in additional cache misses that a more flexible cache would not experience.
Direct-mapped cache	A direct-mapped cache maps each address in the *lower-level memory* to a single location in the cache. Multiple locations may map to the same location in the cache. This is in contrast to a multi-way *set-associative cache*, which selects a place for the data from a *set* of locations in the cache. A direct-mapped cache can be considered a single-way set-associative cache.
Dirty	In a *writeback* cache, writes that reach a given level in the memory hierarchy may update that level, but not the levels below it. Thus, when a cache *line* is *valid* and contains updates that have not been sent to the next lower level, that line is said to be dirty. The opposite state for a valid cache line is *clean*.
DMA	Direct Memory Access. Typically, a DMA operation copies a block of memory from one range of addresses to another, or transfers data between a peripheral and memory. On the C64x+ DSP, DMA transfers are performed by the enhanced DMA (EDMA) engine. These DMA transfers occur in parallel to program execution. From a cache coherence standpoint, EDMA accesses can be considered accesses by a parallel processor.
Eviction	The process of removing a line from the cache to make room for newly cached data. Eviction can also occur under user control by requesting a *writeback-invalidate* for an address or range of addresses from the cache. The evicted line is referred to as the *victim*. When a victim line is *dirty* (that is, it contains updated data), the data must be written out to the next level memory to maintain *coherency*.
Execute packet	A block of instructions that begin execution in parallel in a single cycle. An execute packet may contain between 1 and 8 instructions.
Fetch packet	A block of 8 instructions that are fetched in a single cycle. One fetch packet may contain multiple *execute packets*, and thus may be consumed over multiple cycles.
First-reference miss	A cache *miss* that occurs on the first reference to a piece of data. First-reference misses are a form of *compulsory miss*.

Table B-1. List of Cache-Related Terms and Definitions (continued)

Term	Definition
Fully-associative cache	A cache that allows any memory address to be stored at any location within the cache. Such caches are very flexible, but usually not practical to build in hardware. They contrast sharply with *direct-mapped caches* and *set-associative caches*, both of which have much more restrictive *allocation* policies. Conceptually, fully-associative caches are useful for distinguishing between *conflict misses* and *capacity misses* when analyzing the performance of a direct-mapped or set-associative cache. In terms of set-associative caches, a fully-associative cache is equivalent to a set-associative cache that has as many *ways* as it does *line frames*, and that has only one *set*.
Higher-level memory	In a hierarchical memory system, higher-level memories are memories that are closer to the CPU. The highest level in the memory hierarchy is usually the Level 1 (L1) caches. The memories at this level exist directly next to the CPU. Higher-level memories typically act as caches for data from lower-level memory.
Hit	A cache hit occurs when the data for a requested memory location is present in the cache. The opposite of a hit is a *miss*. A cache hit minimizes stalling, since the data can be fetched from the cache much faster than from the source memory. The determination of hit versus miss is made on each level of the memory hierarchy separately-a miss in one level may hit in a lower level.
Invalidate	The process of marking *valid* cache *lines* as invalid in a particular cache. Alone, this action discards the contents of the affected cache lines, and does not write back any updated data. When combined with a *writeback*, this effectively updates the next lower level of memory that holds the data, while completely removing the cached data from the given level of memory. Invalidates combined with writebacks are referred to as *writeback-invalidates*, and are commonly used for retaining *coherence* between caches.
Least-Recently Used (LRU) allocation	For *set-associative caches* and *fully-associative caches*, the least-recently used allocation refers to the method used to choose among *line frames* in a *set* when allocating space in the cache. When all of the line frames in the set that the address maps to contain *valid* data, the line frame in the set that was read or written the least recently (furthest back in time) is selected to hold the newly cached data. The selected line frame is then *evicted* to make room for the new data.
Line	A cache line is the smallest block of data that the cache operates on. The cache line is typically much larger than the size of data accesses from the CPU or the next higher level of memory. For instance, although the CPU may request single bytes from memory, on a read *miss* the cache reads an entire line's worth of data to satisfy the request.
Line frame	A location in a cache that holds cached data (one *line*), an associated *tag* address, and status information for the line. The status information can include whether the line is *valid*, *dirty*, and the current state of that line's *least-recently used* (LRU).
Line size	The size of a single cache *line*, in bytes.
Load through	When a CPU request misses both the first-level and second-level caches, the data is fetched from the external memory and stored to both the first-level and second-level cache simultaneously. A cache that stores data and sends that data to the upper-level cache at the same time is a load-through cache. Using a load-through cache reduces the stall time compared to a cache that first stores the data in a lower level and then sends it to the higher-level cache as a second step.
Long-distance access	Accesses made by the CPU to a non-cacheable memory. Long-distance accesses are used when accessing external memory that is not marked as cacheable.
Lower-level memory	In a hierarchical memory system, lower-level memories are memories that are further from the CPU. In a C64x+ DSP system, the lowest level in the hierarchy includes the system memory below Level 2 (L2) and any memory-mapped peripherals.
LRU	Least Recently Used. See *least-recently used allocation* for a description of the LRU replacement policy. When used alone, LRU usually refers to the status information that the cache maintains for identifying the least-recently used *line* in a *set*. For example, consider the phrase "accessing a cache line updates the LRU for that line."
Memory ordering	Defines what order the effects of memory operations are made visible in memory. (This is sometimes referred to as consistency). Strong memory ordering at a given level in the memory hierarchy indicates it is not possible to observe the effects of memory accesses in that level of memory in an order different than program order. Relaxed memory ordering allows the memory hierarchy to make the effects of memory operations visible in a different order. Note that strong ordering does not require that the memory system execute memory operations in program order, only that it makes their effects visible to other requestors in an order consistent with program order. Section 8.3 covers the memory ordering assurances that the C64x+ DSP memory hierarchy provides.
Miss	A cache miss occurs when the data for a requested memory location is not in the cache. A miss may stall the requestor while the *line frame* is allocated and data is fetched from the next lower level of memory. In some cases, such as a CPU write miss from L1D, it is not strictly necessary to stall the CPU. Cache misses are often divided into three categories: *compulsory misses*, *conflict misses*, and *capacity misses*.
Miss pipelining	The process of servicing a single cache *miss* is pipelined over several cycles. By pipelining the miss, it is possible to overlap the processing of several misses, should many occur back-to-back. The net result is that much of the overhead for the subsequent misses is hidden, and the incremental stall penalty for the additional misses is much smaller than that for a single miss taken in isolation.

Table B-1. List of Cache-Related Terms and Definitions (continued)

Term	Definition
Read allocate	A read-allocate cache only allocates space in the cache on a read *miss*. A write miss does not cause an *allocation* to occur unless the cache is also a *write-allocate* cache. For caches that do not write allocate, the write data would be passed on to the next lower-level cache.
Set	A collection of *line frames* in a cache that a single address can potentially reside. A *direct-mapped cache* contains one line frame per set, and an N-way *set-associative cache* contains N line frames per set. A *fully-associative cache* has only one set that contains all of the line frames in the cache.
Set-associative cache	A set-associative cache contains multiple *line frames* that each *lower-level memory* location can be held in. When allocating room for a new *line* of data, the selection is made based on the *allocation* policy for the cache. The C64x+ devices employ a *least-recently used allocation* policy for its set-associative caches.
Snoop	A method by which a *lower-level memory* queries a *higher-level memory* to determine if the higher-level memory contains data for a given address. The primary purpose of snoops is to retain *coherency*, by allowing a lower-level memory to request updates from a higher-level memory. A snoop operation may trigger a *writeback*, or more commonly, a *writeback-invalidate*. Snoops that trigger writeback-invalidates are sometimes called snoop-invalidates.
Tag	A storage element containing the most-significant bits of the address stored in a particular line. Tag addresses are stored in special tag memories that are not directly visible to the CPU. The cache queries the tag memories on each access to determine if the access is a *hit* or a *miss*.
Thrash	An algorithm is said to thrash the cache when its access pattern causes the performance of the cache to suffer dramatically. Thrashing can occur for multiple reasons. One possible situation is that the algorithm is accessing too much data or program code in a short time frame with little or no reuse. That is, its working set is too large, and thus the algorithm is causing a significant number of capacity misses. Another situation is that the algorithm is repeatedly accessing a small group of different addresses that all map to the same set in the cache, thus causing an artificially high number of conflict misses.
Touch	A memory operation on a given address is said to touch that address. Touch can also refer to reading array elements or other ranges of memory addresses for the sole purpose of allocating them in a particular level of the cache. A CPU-centric loop used for touching a range of memory in order to allocate it into the cache is often referred to as a touch loop. Touching an array is a form of software-controlled pre-fetch for data.
Valid	When a cache line holds data that has been fetched from the next level memory, that line frame is valid. The invalid state occurs when the line frame holds no data, either because nothing has been cached yet, or because previously cached data has been invalidated for whatever reason (coherence protocol, program request, etc.). The valid state makes no implications as to whether the data has been modified since it was fetched from the lower-level memory; rather, this is indicated by the dirty or clean state of the line.
Victim	When space is allocated in a set for a new line, and all of the line frames in the set that the address maps to contain valid data, the cache controller must select one of the valid lines to evict in order to make room for the new data. Typically, the least-recently used (LRU) line is selected. The line that is evicted is known as the victim line. If the victim line is dirty, its contents are written to the next lower level of memory using a victim writeback.
Victim Buffer	A special buffer that holds victims until they are written back. Victim lines are moved to the victim buffer to make room in the cache for incoming data.
Victim Writeback	When a dirty line is evicted (that is, a line with updated data is evicted), the updated data is written to the lower levels of memory. This process is referred to as a victim writeback.
Way	In a set-associative cache, each set in the cache contains multiple line frames. The number of line frames in each set is referred to as the number of ways in the cache. The collection of corresponding line frames across all sets in the cache is called a way in the cache. For instance, a 4-way set-associative cache has 4 ways, and each set in the cache has 4 line frames associated with it, one associated with each of the 4 ways. As a result, any given cacheable address in the memory map has 4 possible locations it can map to in a 4-way set-associative cache.
Working set	The working set for a program or algorithm is the total set of data and program code that is referenced within a particular period of time. It is often useful to consider the working set on an algorithm-by-algorithm basis when analyzing upper levels of memory, and on a whole-program basis when analyzing lower levels of memory.
Write allocate	A write-allocate cache allocates space in the cache when a write miss occurs. Space is allocated according to the cache's allocation policy (LRU, for example), and the data for the line is read into the cache from the next lower level of memory. Once the data is present in the cache, the write is processed. For a writeback cache, only the current level of memory is updated—the write data is not immediately passed to the next level of memory.
Writeback	The process of writing updated data from a valid but dirty cache line to a lower-level memory. After the writeback occurs, the cache line is considered clean. Unless paired with an invalidate (as in writeback-invalidate), the line remains valid after a writeback.

Table B-1. List of Cache-Related Terms and Definitions (continued)

Term	Definition
Writeback cache	A writeback cache will only modify its own data on a write hit. It will not immediately send the update to the next lower-level of memory. The data will be written back at some future point, such as when the cache line is evicted, or when the lower-level memory snoops the address from the higher-level memory. It is also possible to directly initiate a writeback for a range of addresses using cache control registers. A write hit to a writeback cache causes the corresponding line to be marked as dirty-that is, the line contains updates that have yet to be sent to the lower levels of memory.
Writeback-invalidate	A writeback operation followed by an invalidation. See writeback and invalidate. On the C64x+ devices, a writeback-invalidate on a group of cache lines only writes out data for dirty cache lines, but invalidates the contents of all of the affected cache lines.
Write merging	Write merging combines multiple independent writes into a single, larger write. This improves the performance of the memory system by reducing the number of individual memory accesses it needs to process. For instance, on the C64x+ device, the L1D write buffer can merge multiple writes under some circumstances if they are to the same double-word address. In this example, the result is a larger effective write-buffer capacity and a lower bandwidth impact on L2.
Write-through cache	A write-through cache passes all writes to the lower-level memory. It never contains updated data that it has not passed on to the lower-level memory. As a result, cache lines can never be dirty in a write-through cache. The C64x+ devices do not utilize write–through caches.

Revision History

Table C-1 lists the changes made since the previous version of this document.

Table C-1. Document Revision History

Reference	Additions/Modifications/Deletions
Section 2.3.1.2	Changed paragraph.
Table 3-17	Changed table.
Section 4.4.3.1.2	Added last sentence to first paragraph.
Table 4-13	Changed Value range of L2WWC bit.
	Changed Description of L2WWC bit.
Section 4.4.3.1.4	Added last sentence to first paragraph.
Table 4-15	Changed Value range of L2WIWC bit.
	Changed Description of L2WIWC bit.
Section 4.4.3.1.6	Added last sentence to first paragraph.
Table 4-17	Changed Value range of L2IWC bit.
	Changed Description of L2IWC bit.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Communications and Telecom	www.ti.com/communications
DSP	dsp.ti.com	Computers and Peripherals	www.ti.com/computers
Clocks and Timers	www.ti.com/clocks	Consumer Electronics	www.ti.com/consumer-apps
Interface	interface.ti.com	Energy	www.ti.com/energy
Logic	logic.ti.com	Industrial	www.ti.com/industrial
Power Mgmt	power.ti.com	Medical	www.ti.com/medical
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Space, Avionics & Defense	www.ti.com/space-avionics-defense
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video and Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless-apps

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265