Synchronous System Halt and Run on the ADSP-CM41x Processor

Contributed by Manjunath, Manasa, Kiranmai Pernapati and Prasanth Rajagopal *Rev 1 – March 21, 2019*

Introduction

The ADSP-CM41xF is a dual core mixed-signal control processor containing an Arm® Cortex®-M4 processor core and an Arm Cortex-M0 processor core. It includes several peripherals, such as the pulse-width modulators (PWM), that support motor control and inverter applications. Development on the processor is supported through the IAR Embedded Workbench and Keil™ MDK-Lite tool chains. The synchronous halt operation is used when:

- The peripherals must stop running due to the debugger entering emulation mode (halt execution)
- The peripherals must start running due to the debugger exiting from emulation mode (resume execution)
- External components (such as motors) that are connected and PWMs must synchronously halt or run due to the execution or halt of the control code logic.

Synchronous halt operation is not supported as received by debuggers like IAR and Keil. However, the ADSP-CM41xF processor is designed to support the operation in hardware. The processor uses the Embedded Cross Trigger (ECT) which consists of the Cross Trigger Interface (CTI) and the Cross Trigger Matrix (CTM). Both the IAR Embedded Workbench C-SPY debugger and Keil debugger interface and drive the J-Link Lite emulator that supports the ADSP-CM41xF processor family. The C-SPY and µVision® support macros that can be executed when specific events occur.

This application note provides an IAR EWARM and Keil debug scripts to set up the CTI registers to synchronously halt and run various peripherals in the system. The zip file[1] associated with this note includes some examples that can be executed on an ADSP-CM41xF processor evaluation board.

Figure 1 and Figure 2 provide examples of synchronous halt and run in a system.

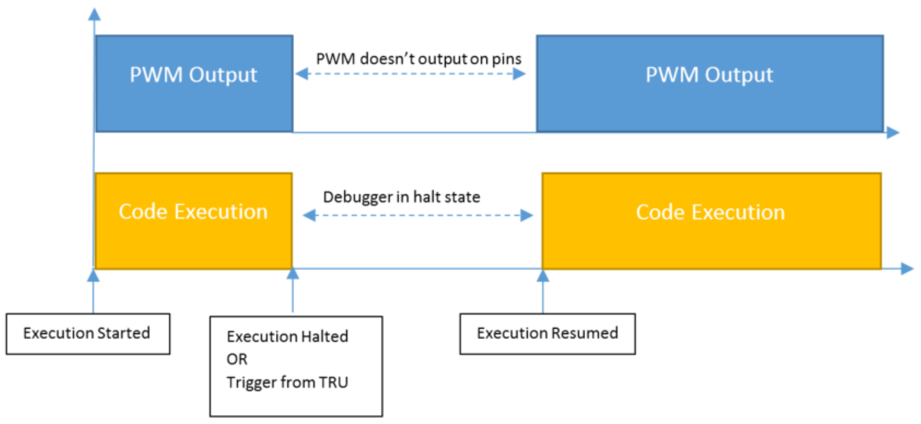


Figure 1. Synchronous Halt and Run – PWM

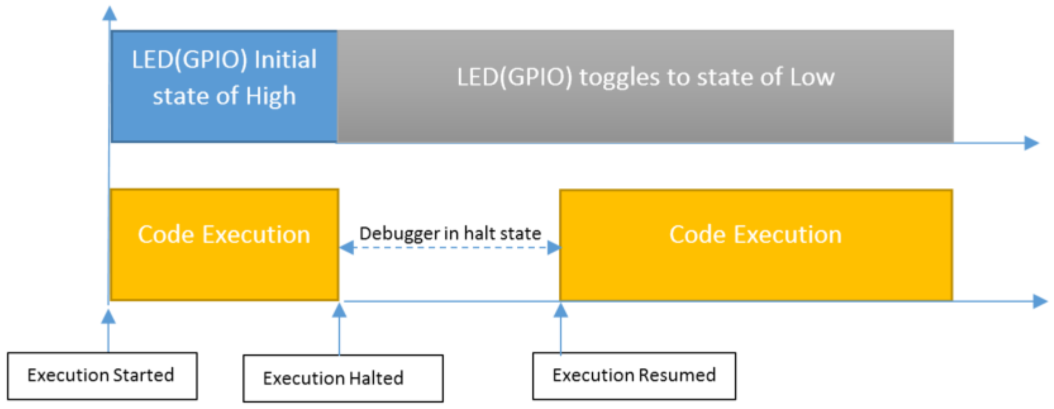


Figure 2. Synchronous Halt and Run – GPIO

Embedded Cross Trigger (ECT)

ECT provides an interface to the CoreSight debug system enabling the subsystems to interact (cross trigger) with each other. The main function of the ECT (CTI and CTM) is to pass debug events from one connected subsystem to another. Program execution on both of the subsystems can be stopped at the same time. The different subsystems connected to the ECT depend on the processor design. For example, in a multiprocessor system, the interface can connect to each of the cores and to the trace subsystem. For a uniprocessor system, the interface can connect to the core and trace subsystem.

- CTI – A CoreSight component for enabling the cross triggering of events across a system
- CTM – A CoreSight component for connecting multiple cross trigger interfaces

Figure 3 shows the debug trigger flow sequence. On each CTI, there are four channels, eight inputs, and eight output debug triggers. All of the eight inputs and outputs can be mapped to a single channel or different channels based on the debug trigger-to-channel mapping. When a trigger input occurs, it creates a channel event. The channel event causes all the output debug triggers to be triggered. The embedded cross trigger depends on the debug trigger it connects to.

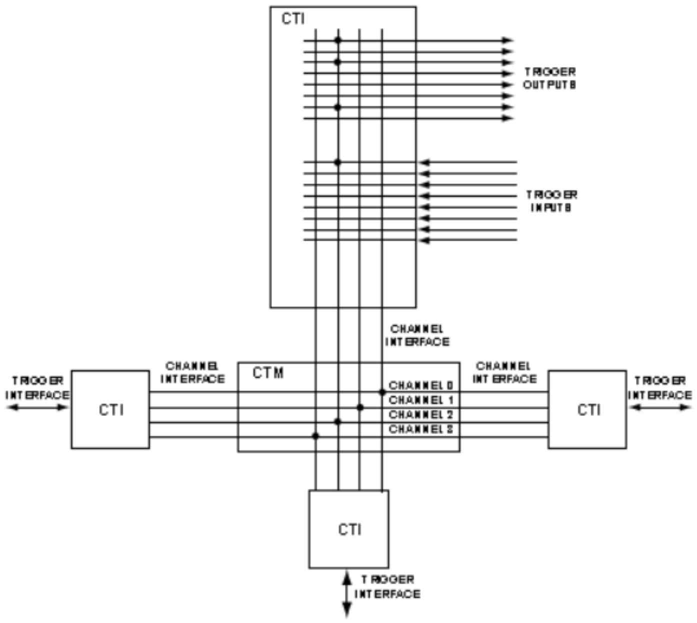


Figure 3. Embedded Cross Trigger Flow

Cross Trigger Interface

The Cortex-M4 and Cortex-M0 subsystems on the ADSP-CM41xF processors have dedicated cross trigger interfaces. Figure 4 shows the cross trigger system interface. The ECT provides an interface to the debug system. The CTM combines the channel requests generated by the CTI blocks and broadcasts them to all other CTI blocks as channel triggers. This configuration enables the subsystems to cross trigger with each other.

Dedicated cross trigger interfaces include:

- CTI0 for Cortex M4
- CTI1 for Cortex M0
- CTI2 for TPIU in debug system
- CTI3 for connections to and from both the M4 and M0 system trigger routing units (TRU0, TRU1)

The primary focus of this EE-Note is on halting system components such as the PWM when a debugger halt operation occurs. Additional examples include the use of cross triggers to control the halt operation through a system trigger event such as GPIO trigger or control additional system components when a debugger halt operation occurs.

Table 1 provides the cross trigger debug connection and trigger input/output port information for each CTI block.

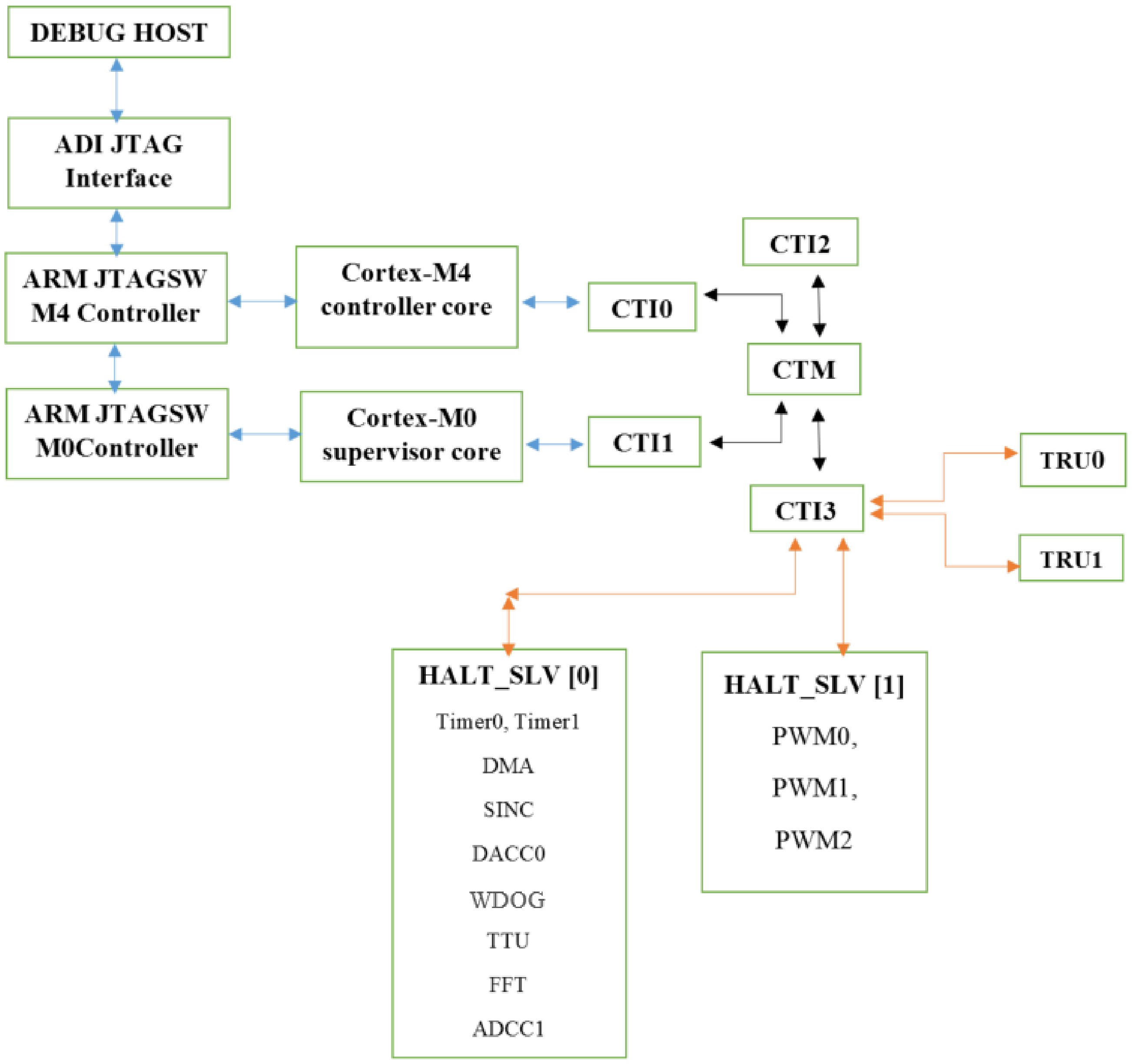


Figure 4. Cross Trigger System Interface

CTI0 (M4 Debug Connections)					
Trigger Input Port	**Source**	**ACK Used?**	**Trigger Output Port**	**Destination**	**ACK Used?**
CTITRIGIN[7]	M4 ETM TRIG_OUT	N	CTITRIGOUT[7]	M4 DBGSTART	N
CTITRIGIN[6]	M4 ETM TRIG[2]	N	CTITRIGOUT[6]	M4 DBGEND	N
CTITRIGIN[5]	M4 ETM TRIG[1]	N	CTITRIGOUT[5]	M4 ETM EXTIN[1]	N
CTITRIGIN[4]	M4 ETM TRIG[0]	N	CTITRIGOUT[4]	M4 ETM EXTIN[0]	N
CTITRIGIN[3]	Unused	N	CTITRIGOUT[3]	SEC_M4_CTI0_EVT2	Y
CTITRIGIN[2]	Unused	N	CTITRIGOUT[2]	SEC_M4_CTI0_EVT1	Y
CTITRIGIN[1]	Unused	N	CTITRIGOUT[1]	SEC_M4_CTI0_EVT0	Y
CTITRIGIN[0]	M4 HALTED	Y	CTITRIGOUT[0]	M4 EDBGRQ	Y
CTI1 (M0 Debug Connections)					
Trigger Input Port	**Source**	**ACK Used?**	**Trigger Output Port**	**Destination**	**ACK Used?**
CTITRIGIN[7]	Unused	N	CTITRIGOUT[7]	M0 DBGSTART	N
CTITRIGIN[6]	Unused	N	CTITRIGOUT[6]	M0_DBGEND	N
CTITRIGIN[5]	Unused	N	CTITRIGOUT[5]	Unused	N
CTITRIGIN[4]	Unused	N	CTITRIGOUT[4]	Unused	N
CTITRIGIN[3]	Unused	N	CTITRIGOUT[3]	Unused	N
CTITRIGIN[2]	Unused	N	CTITRIGOUT[2]	Unused	N
CTITRIGIN[1]	Unused	N	CTITRIGOUT[1]	SEC_M0_CTI1_EVT0	N
CTITRIGIN[0]	M0 HALTED	Y	CTITRIGOUT[0]	M0 EDBGRQ	Y
CTI2 (ETF/TPIU Connections)					
Trigger Input Port	**Source**	**ACK Used?**	**Trigger Output Port**	**Destination**	**ACK Used?**
CTITRIGIN[7]	Unused	N	CTITRIGOUT[7]	Unused	N
CTITRIGIN[6]	Unused	N	CTITRIGOUT[6]	Unused	N
CTITRIGIN[5]	Unused	N	CTITRIGOUT[5]	Unused	N
CTITRIGIN[4]	Unused	N	CTITRIGOUT[4]	Unused	N
CTITRIGIN[3]	Unused	N	CTITRIGOUT[3]	Unused	Y
CTITRIGIN[2]	Unused	N	CTITRIGOUT[2]	Unused	Y
CTITRIGIN[1]	ETF_FULL	N	CTITRIGOUT[1]	ETF FLUSHIN	N
CTITRIGIN[0]	ACQCOMP FULL	N	CTITRIGOUT[0]	ETF TRIGIN	N
CTI3 (System Trigger Unit Connections)					

Trigger Input Port	**Source**	**ACK Used?**	**Trigger Output Port**	**Destination**	**ACK Used?**
CTITRIGIN[7]	TRGS_M0_CTI3_SLV7	N	CTITRIGOUT[7]	ctitrigoutack for halt_slv[1:0]	N
CTITRIGIN[6]	TRGS_M0_CTI3_SLV6	N	CTITRIGOUT[6]	TRGS_M0_CTI3_MST6	N
CTITRIGIN[5]	TRGS_M4_CTI3_SLV5	N	CTITRIGOUT[5]	TRGS_M0_CTI3_MST5	N
CTITRIGIN[4]	TRGS_M4_CTI3_SLV4	N	CTITRIGOUT[4]	TRGS_M4_CTI3_MST4	N
CTITRIGIN[3]	TRGS_M4_CTI3_SLV3	N	CTITRIGOUT[3]	TRGS_M4_CTI3_MST3	N
CTITRIGIN[2]	TRGS_M4_CTI3_SLV2	N	CTITRIGOUT[2]	TRGS_M4_CTI3_MST2	N
CTITRIGIN[1]	TRGS_M4_CTI3_SLV1	N	CTITRIGOUT[1]	halt_slv[1]	Y
CTITRIGIN[0]	TRGS_M4_CTI3_SLV0	N	CTITRIGOUT[0]	halt_slv[0]	Y

Table 1.Cross Trigger Connections

CTI registers cannot be accessed from the Cortex-M0 core.

As shown in Table 1, there are two system cross trigger halts in CTI3: HALT_SLV_0 and HALT_SLV_1. These two systems include different sets of peripheral components. Executing the CTI3 trigger halt sequence over these sections halts the operations of the corresponding components. Additional programming is not required to control the halting of components under HALT_SLV_0 and HALT_SLV_1; the design is hard-wired to automatically issue a halt command.

Table 2 lists the components included in HALT_SLV_0 and HALT_SLV_1 for the ADSP-CM41xF processors.

- HALT_SLV_1 halts the PWM units
- HALT_SLV_0 halts the fabric masters, timers, watchdog (WDOG) timers and trigger timing units (TTU). The PWM units are separated from the HALT_SLV_1 trigger because stopping an external component such as a motor or inverter at an indeterminate state for debug can result in damage to the component.

HALT_SLV_0	M4P TIMER 0 and TIMER 1 M4P TTU0 and TTU1 M4P WDOG0 and WDOG1 DMA 4-13 SINC DACC0 M0P DMA 0-3 FFT ADCC1
HALT_SLV_1	PWM0 PWM 1 PWM 2

Table 2. HALT_SLV_0 and HALT_SLV_1 Components

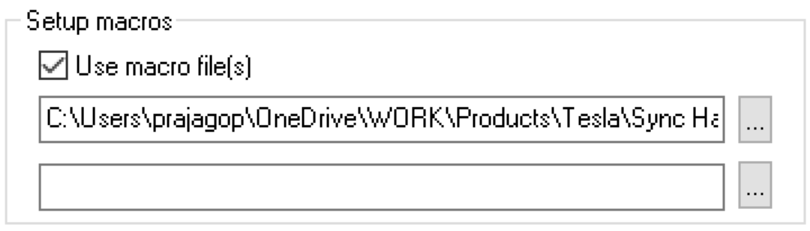
Implementation of Synchronous Halt and Run with IAR

The macro script files contain instructions for CTI3 to perform a debug halt and run. These macros are leveraged to configure the CTI ports for synchronous halt and run during the program load sequence. Synchronous restart is executed upon any user restart (for example, run or step) and through reserved macro support.

Use the following sequence to set up and implement synchronous system halt and run for HALT_SLV_0 and HALT_SLV_1.

1. Add macro scripts to the project.

The zip file[1] associated with this EE-Note includes the macro file `CM41xF_M4JLinkScript_CTISyncHaltbeta.mac`. To enable synchronous halt and run, add this macro script file to the project under *IAR Embedded Workbench Project>Options>Debugger>Setup Macros*. Check the *Use macro file* in IAR project options as shown in Figure 5.


Setup macros

☑ Use macro file(s)

C:\Users\prajagop\OneDrive\WORK\Products\Tesla\Sync Ha

Figure 5. Set Up Macro

The macros have been set up so that, by default, all the ports in both HALT_SLV_0 and HALT_SLV_1 are enabled for synchronous halt and run. The macro script contains the macro parameters shown in Table 3 which must be included in the project options:

Macro Parameter	**Description**
`--macro_param excludeSLV0=1`	Removes HALT_SLV_0 group

`--macro_param excludeSLV1=1`	Removes HALT_SLV_1 group
`--macro_param excludeSLV2=1`	Removes trigger to CTITRIGOUT[2]
`--macro_param excludeSLV3=1`	Removes trigger to CTITRIGOUT[3]
`--macro_param excludeSLV4=1`	Removes trigger to CTITRIGOUT[4]
`--macro_param excludeSLV5=1`	Removes trigger to CTITRIGOUT[5]
`--macro_param excludeSLV6=1`	Removes trigger to CTITRIGOUT[6]

Table 3. Macro Script Parameters

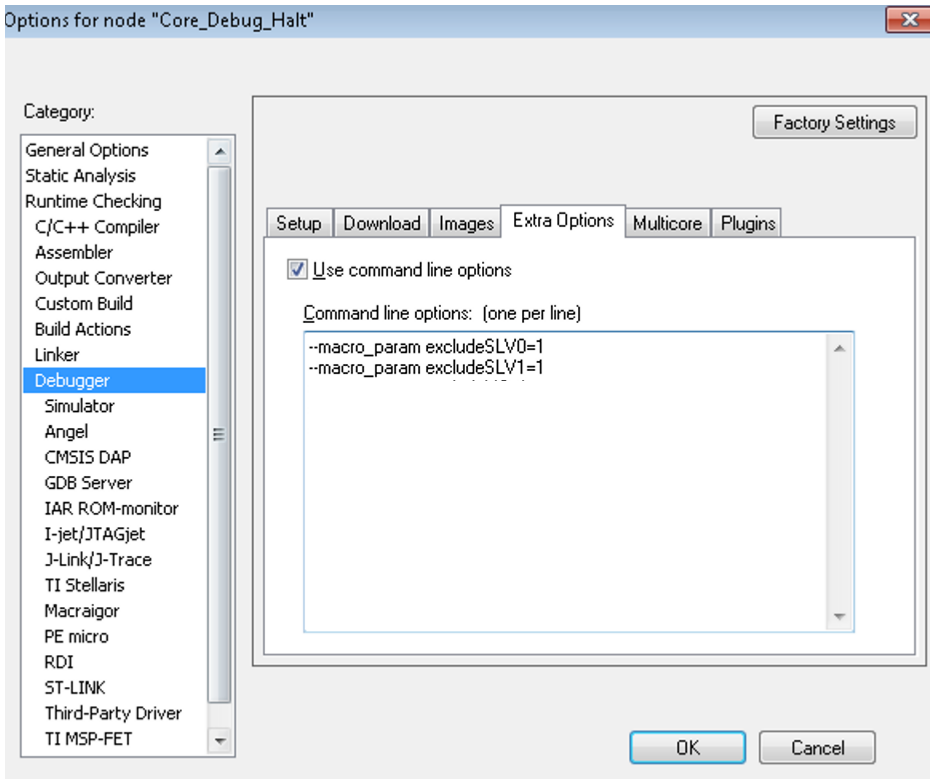
If a macro_param is 0, the system is in synchronous halt and run mode. In this mode, when program execution is halted, the components under it also *halt* the operation.

If a macro_param is 1, the system is in synchronous halt and run mode. In this mode, when program execution is halted, the components under it *do not halt* the operation.

By default, these macro parameters are defined as 0 in the script file.

2. Debug and load the executable to the ADSP-CM419F EZ-Board.

Use command line parameters to exclude halt slave groups. Add command line parameters under the *Project>Options>Debugger>Extra Options* tab as shown in Figure 6.



☑ Use command line options

Command line options: (one per line)

```
--macro_param excludeSLV0=1
--macro_param excludeSLV1=1
```

Figure 6. Link Macro Script File to Debugger

3. Halt the execution. Click **Break** on the toolbar to stop the program execution. Upon halting, peripheral halts.

4. Restart the execution. Restart is executed on any restart (for example, run or step).

Implementation of Synchronous Halt and Run with Keil µVision

The initialization file contains instructions for CTI3 to perform halt and run. These instructions execute line by line during debug. After the completion of execution, program execution begins. To halt the running program, click the **Stop** button in Debug Session toolbar. The restart executed upon user either click on Resume Toolbox button or enter the `run()`command in the command window.

Use the following sequence to set up and implement synchronous system halt and run for HALT_SLV_0 and HALT_SLV_1.

Associated zip file[1] includes the initialization file `CM41x_M4JLinkScript_CTISyncHaltRun.ini`.

1. To enable synchronous halt and run, add this initialization script file to the project under "Keil µVision Project->Options for Target->Debug->Initialization File".

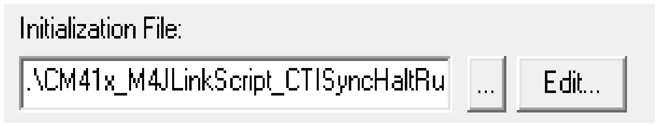


Figure 7 Initialization File to Debugger

The macros have been set up so that, by default, all the ports in both HALT_SLV_0 and HALT_SLV_1 are enabled for synchronous halt and run. The initialization file contains the symbols shown in Table 4 which must be included in the project.

Symbol	Description
`excludeSLV0=1`	Removes HALT_SLV_0 group
`excludeSLV1=1`	Removes HALT_SLV_1 group
`excludeSLV2=1`	Removes trigger to CTITRIGOUT[2]
`excludeSLV3=1`	Removes trigger to CTITRIGOUT[3]
`excludeSLV4=1`	Removes trigger to CTITRIGOUT[4]
`excludeSLV5=1`	Removes trigger to CTITRIGOUT[5]
`excludeSLV6=1`	Removes trigger to CTITRIGOUT[6]

Table 4. Symbols

If a symbol is 0, the system is in synchronous halt and run mode. In this mode, when program execution is halted, the components under it also *halt* the operation.

If a symbol is 1, the system is in synchronous halt and run mode. In this mode, when program execution is halted, the components under it *do not halt* the operation.

By default, these symbols are defined as 0 in the script file.

2. Debug and load the executable to ADSP-CM419F EZ-Board.
3. Halt the execution. Click **Stop** on the toolbar to stop the program execution. Upon halting, peripheral will halt.

4. Restart the execution. Restart by user, either using a Toolbox button or a Command Window.

   Figure 8 shows the restart using the toolbox button:

   Keil µVision View->Toolbox

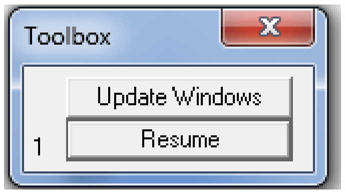


Figure 8 Toolbox Window

Click **Resume** on Toolbox, to run peripheral from halt stage.

   Figure 9 shows the restart using a command window:

   Keil µVision View->Command Window

   Enter the command `run()` in the command window to run peripheral from halt stage.

```
>run()
```

Figure 9 Command Window

Synchronous Halt and Run Examples - IAR Workbench and Keil µVision.

The zip file[1] associated with this EE-Note includes examples that demonstrate system halt and restart using the CTI interface to the IAR C-SPY debugger macro and Keil debugger initialization file. The following sections explain the examples.

HALT_SLV_0

HALT_SLV_0 controls Timer1 and WDOG0 and WDOG1. Timer1 is configured and run in continuous PWM mode. TM1_TMR5 is probed using an oscilloscope.

When the execution is halted in system halt and run mode, the timer is also halted. Figure 10 shows Timer1 during system halt and run mode using the IAR workbench. Figure 11 shows Timer1 during system halt and run mode using Keil µVision.

When a system is *excluded* from halt, the timers continue to run even when halted in debug window. Figure 12 shows Timer1 when exempted from system halt and run using the IAR workbench. Figure 13 shows Timer1 when exempted from system halt and run using Keil µVision.

A watch dog timer is initiated with a definite and large count value. When halted in system halt and run mode and executed in steps, the current count value in the `WDOG_STAT` register stops decrementing. In exemption mode, a single step execution results in higher decrement in value.

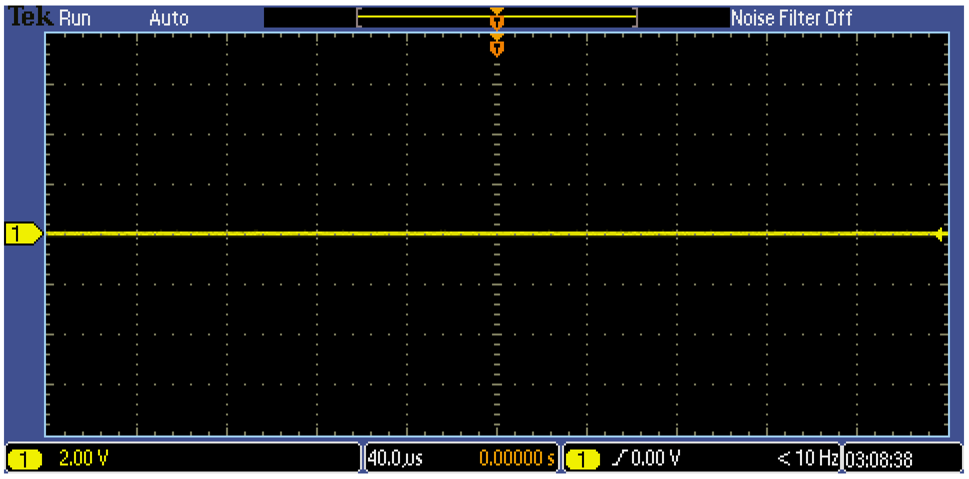


Figure 10. Timer1 During System Halt and Run - IAR

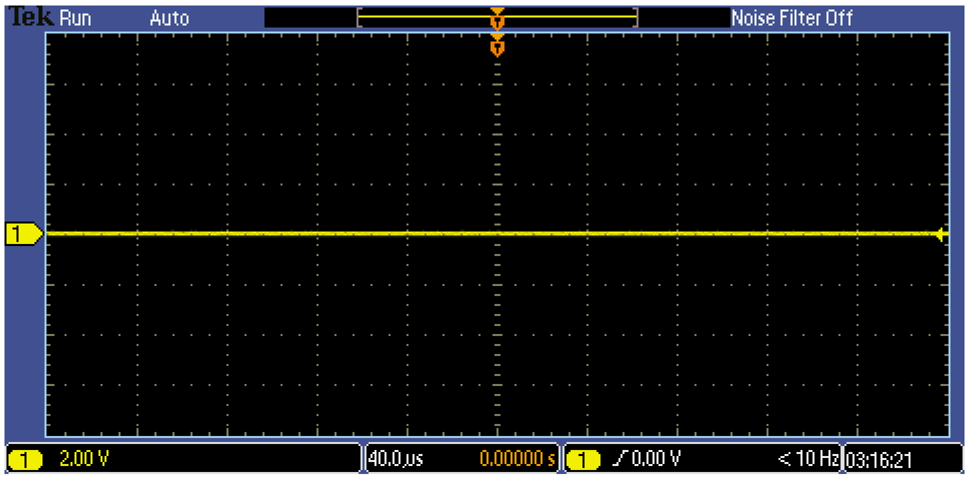


Figure 11. Timer1 Exempted from System Halt and Run - Keil

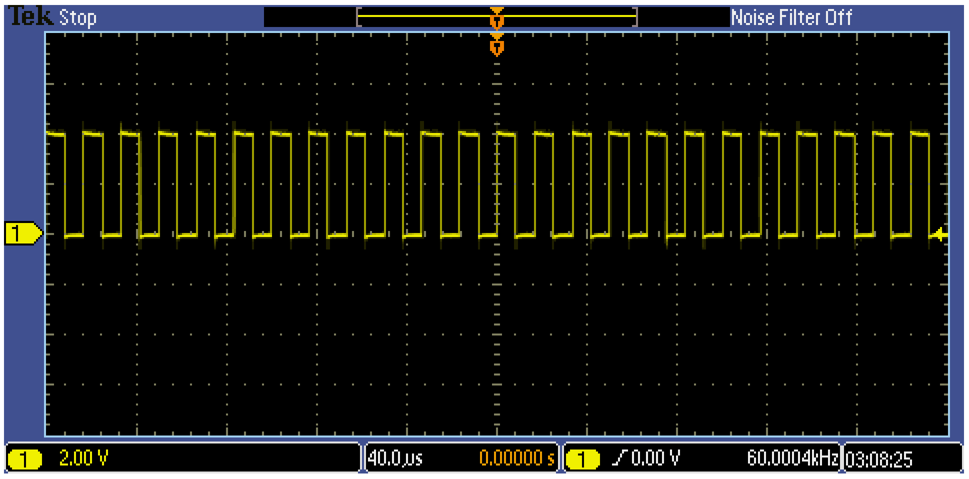


Figure 12. Timer1 During System Halt and Run - IAR

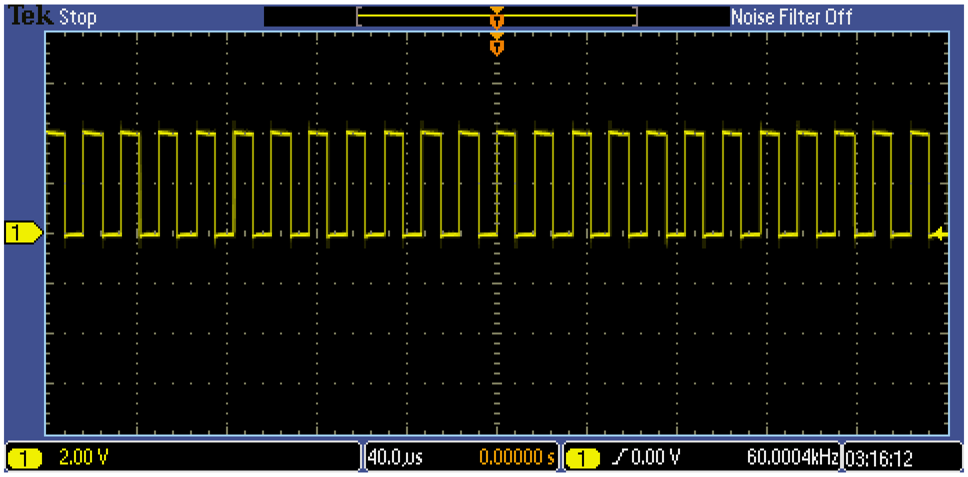


Figure 13. Timer1 Exempted from System Halt and Run - Keil

HALT_SLV_1

HALT_SLV_1 controls PWM0. Channel C of PWM0 is enabled and probed. The signal is probed using an oscilloscope. Figure 14 shows PWM0 during system halt and run mode using the IAR workbench. Figure 15 shows PWM0 during halt and run mode using Keil µVision. Figure 16 shows PWM0 when exempted from system halt and run mode using the IAR workbench. Figure 17 shows PWM0 when exempted from system halt and run mode using Keil µVision.

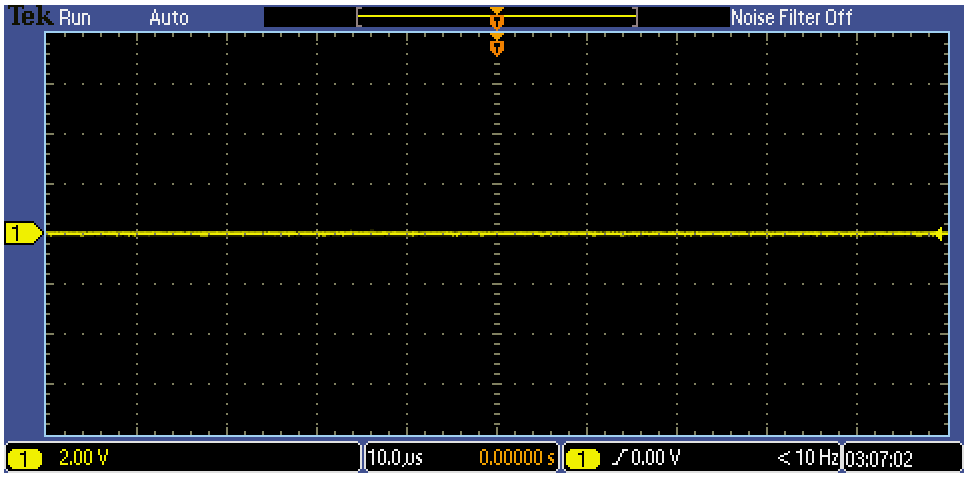


Figure 14. PWM0 During System Halt and Run - IAR

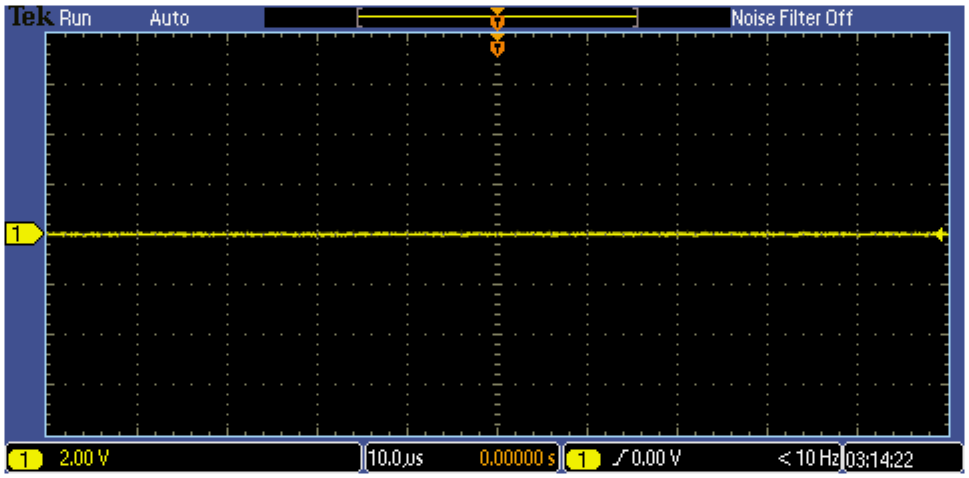


Figure 15. During System Halt and Run - Keil

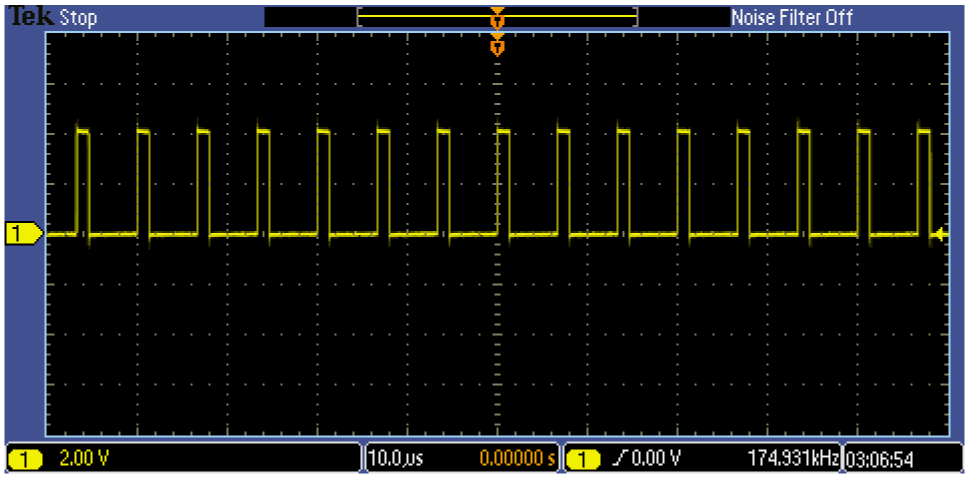


Figure 16. PWM0 Exempted from System Halt and Run - IAR

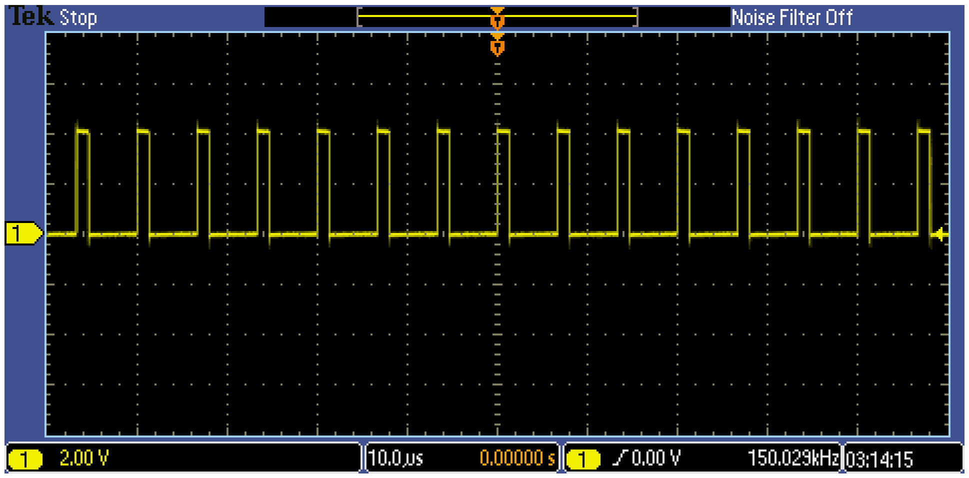


Figure 17. PWM0 Exempted from System Halt and Run - Keil

Other CTI3 Triggers

Other CTI3 triggers include halting PWM using a push button and halting GPIO using a master trigger.

Halt PWM using GPIO Push Button

Using the TRU module, the CTI3_SLV2 slave is triggered by the TRGM_PINT5_BLOCK master trigger. When the push button PB1 is pressed, it triggers CTI3_INEN2. The CTI subsequently trigger CTI3_OUTEN1 which halts the PWMs through HALT_SLV. Note that the debugger is also halted in this case.

HALT GPIO using TRGM_CTI3_MST2

Using the TRU module, the TRGM_CTI3_MST2 master trigger is routed to the GPIO PORT-F (LED3) slave. Initially, the LED is configured as set. When the debugger is halted, it triggers CTI3_INEN2. The CTI unit routes and triggers CTI3_OUTEN2. Since the TRGM_CTI3_MST2 (master) is assigned to PORT-F (slave), this toggles the PORT-F LED. When the debugger resumes, the trigger is acknowledged and cleared, while the state of GPIO/LED remains same. The next halt trigger toggles the LED again.

Halting M0

Halting the M0 processor core when the M4 core is halted is not directly supported by the hardware or the debugger. Using CTI3, it is possible to develop a technique by which the halt operation can be mimicked. For example, once TRGS_M4_CTI3_MST2 is triggered, it can be further propagated to trigger TRU0 of M0 core through general-purpose inter-core triggering. Once the trigger reaches M0, it can trigger a slave trigger interrupt which can wait for a mailbox (MBOX) variable to be set. This variable can be set inside the M4 processor core script as soon as the M4 processor core resumes operation. The trigger interrupt in the M0 processor core can be made a high priority to ensure that no other interrupts preempt it. Figure 18 shows a halt operation sequence when triggered from the M4 processor core. Figure 19 shows a resume operation sequence when triggered from the M4 processor core.

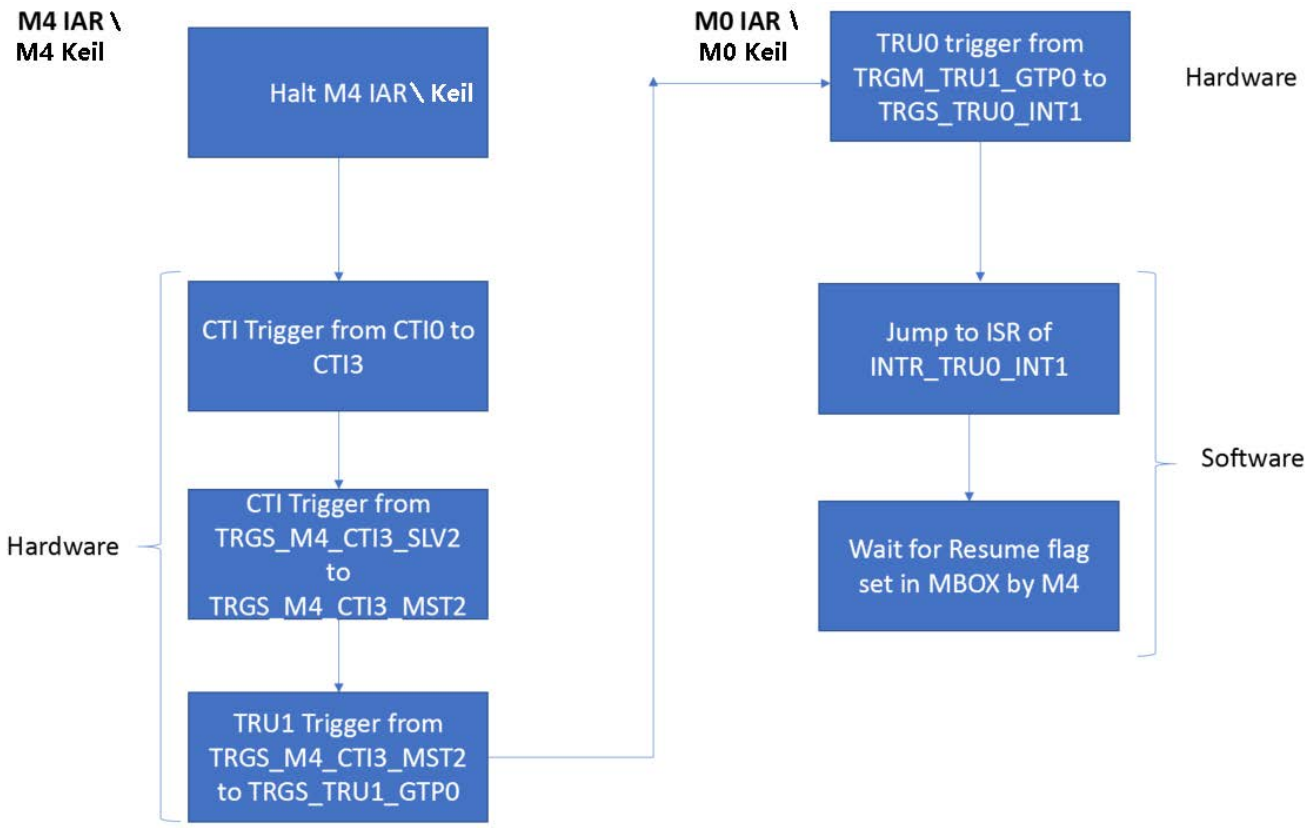

M4 IAR \ M4 Keil
Halt M4 IAR\ Keil
CTI Trigger from CTI0 to CTI3
CTI Trigger from TRGS_M4_CTI3_SLV2 to TRGS_M4_CTI3_MST2
TRU1 Trigger from TRGS_M4_CTI3_MST2 to TRGS_TRU1_GTP0
Hardware
M0 IAR \ M0 Keil
TRU0 trigger from TRGM_TRU1_GTP0 to TRGS_TRU0_INT1
Hardware
Jump to ISR of INTR_TRU0_INT1
Wait for Resume flag set in MBOX by M4
Software

Figure 18. M0 Halt Operation

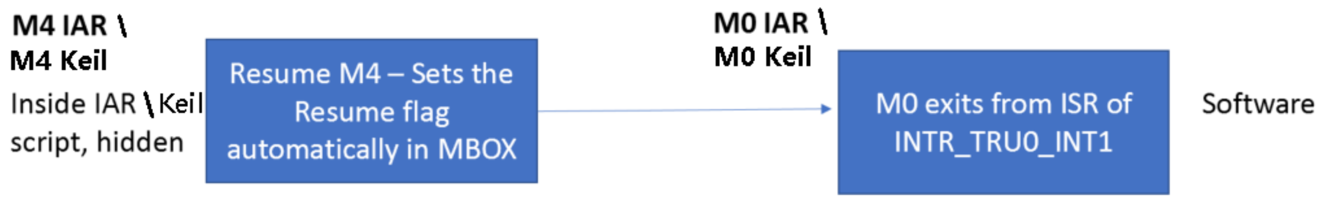

M4 IAR \ M4 Keil
Inside IAR \Keil script, hidden
Resume M4 – Sets the Resume flag automatically in MBOX
M0 IAR \ M0 Keil
M0 exits from ISR of INTR_TRU0_INT1
Software

Figure 19. M0 Resume Operation

Using the CTI3 for Additional Cases

The examples provided in the associated zip file[1] can be used for the following triggers in the CTI3 output matrix.

- HALT_SLV_0
- HALT_SLV_1
- TRGS_M4_CTI3_MST2

The example provided in the associated zip file for TRGS_M4_CTI3_MST2 can be modified for the following cases.

- TRGS_M4_CTI3_MST3
- TRGS_M4_CTI3_MST4
- TRGS_M0_CTI3_MST5
- TRGS_M0_CTI3_MST6

Summary

The C-SPY debug macro script file and µVision debug initialization file are used to trigger synchronous system halt and run by configuring CTI3. The macro script file also can be used to exclude peripherals from system halt and run. Units which cannot be halted from the generic debugger can be halted using the Coresight debug system.

References

[1] *Associated ZIP File for EE-402: Synchronous System Halt and Run on the ADSP-CM41xF Processor*, March 2019. Analog Devices, Inc.

[2] *ADSP-CM41x Mixed-Signal Control Processor with ARM Cortex-M4/ARM Cortex-M0 and 16-bit ADCs Hardware Reference*. Revision 0.3, October 2017, Analog Devices, Inc.

[3] *ARM® Cortex®-M4 Processor Technical Reference Manual*. Revision: r0p1, February 2015. ARM Limited.

[4] *ARM®v7-M Architecture Reference Manual*. Revision C, September 2008. ARM Limited.

Document History

Revision	Description
Rev 1 – March 21, 2019 by Prasanth Rajagopal	Initial Release