Protecting ADSP-CM41x Devices from Input Clock/Power Supply Faults

Contributed by Prasant Rajagopal and David Harrington *Rev 2 – April 25, 2018*

Introduction

The ADSP-CM41x processors feature peripherals such as the Oscillator Watchdog (OSCWD), the Oscillator Comparator Unit (OCU), and the Voltage Monitoring Unit (VMU). These peripherals can be used to determine faults in the clocks, identify power supply issues for the processor, allow for a controlled shutdown of the flash memory and set the general-purpose I/O pins (GPIOs) to safe states. This EE-Note describes how to use these architectural blocks effectively in systems designed around ADSP-CM41x devices.

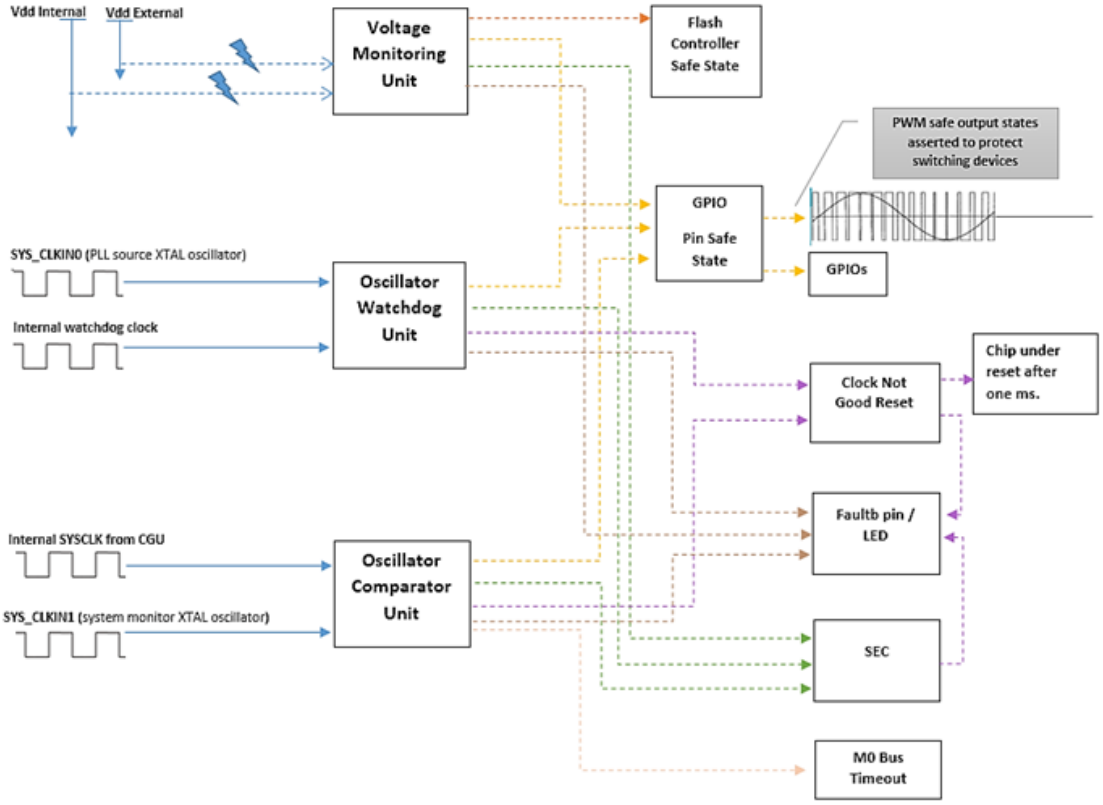

Figure 1: Clock and Power Supply Safety Blocks Block Diagram

Figure 1 shows the clock and supply inputs and the blocks used to detect faults. It also shows the outcomes when a fault is detected. See the *ADSP-CM41x Mixed-Signal Control Processor with ARM Cortex-M4/M0 and 16-bit ADCs Hardware Reference* [4] for details on the functional blocks. This Application Note must be used as reference while consulting the Hardware Reference Manual.

Although the Pin Safe State unit resides inside the VMU, it can be programmed to respond to fault detection from the OCU and OSCWD as well. Similarly, the Clock Not Good Reset unit resides inside the OSCWD but can be programmed to respond to fault detection from the OCU. These units are shown as separate blocks in the block diagram to assist in programming. Additionally, the fault pin assertion of the OSCWD and OCU can also be controlled through the VMU.

Oscillator Watchdog (OSCWD)

The OSCWD is part of the Clock Generation Unit (CGU). It is used to monitor the system clock (CLKIN0) for issues such as bad upper and lower frequency limits and harmonic oscillation problems.

OSCWD Operation

The OSCWD uses an internal auxiliary clock with a frequency of 1 MHz. This frequency is used as a reference point for the detection of frequency oscillation and limit issues.

Configuring the OSCWD

The OSCWD has a single control register `OSCWDCTL` located in the CGU register group. This register provides access to several configurable features:

- Fault Pin Disable
- `OSCWDCTL` Register Lock
- Watchdog Lower Frequency Limit
- Harmonic Oscillation Detection Enable
- Clock Not Good Enable
- Bad Oscillator Frequency Limit
- Bad Oscillator Frequency Limit Detection Enable
- Fault Enable
- OCU Monitor Disable

Oscillator Comparator Unit (OCU)

While the OSCWD monitors the CLKIN input to the device, the OCU monitors the frequency of SYSCLK0, which is an output clock from the on-chip PLL on ADSP-CM41x devices. If a fault, frequency drift, or other clock error occurs in the SYSCLK0 domain, the OCU can generate both a fault and an interrupt. Combined with the OSCWD, this configuration provides the ability to monitor the entire clock path from the crystal/oscillator input to the clock supplied to the peripherals.

OCU Operation

The OCU monitors the frequency of the input clock using an additional Low Frequency Oscillator (LFO) input. If the frequency drift of the input clock exceeds the specification, a fault is issued. The OCU can also

detect gross frequency errors on either clock. Errors are reported to the System Event Controller (SEC); a fault occurs when there is a dead input clock.

The OCU has two operational modes:

- **Manual** - a single detection cycle runs (the OCU stops upon completion whether a fault occurs or not and stops monitoring the clocks).
- **Automatic** - a continuous detection cycle runs (the OCU remains enabled and continues monitoring the clocks until manually stopped, even if a fault occurs).

In both modes, an interrupt is generated and a fault occurs when the OCU detects a clock issue.

Configuring the OCU

The OCU has several registers that must be configured before it can be enabled. To configure the OCU, the clock reference count, the PPM of the clock source, and the PPM of the crystal LFO clock must be known. To determine the clock reference count, use the following equation:

$$\text{Clock Reference Count} = \text{LFO\_Frequency} / 16$$

Since the *ADSP-CM419F EZ-Kit® Evaluation System*[1] uses an LFO frequency of 16 MHz, the equation yields a clock reference count of 1MHz. The input and LFO clock sources populated on the EZ-KIT evaluation system both have a PPM of 50.

Voltage Monitoring Unit (VMU)

The VMU provides over-voltage and under-voltage detection. It monitors the V_{DD_EXT} and V_{DD_INT} power domains and generates asynchronous signals when the voltages exceed or drop below the programmable limits. The VMU also generates a control signal for the power sequencing requirements of the embedded flash memory, even if the VMU is disabled.

The VMU module supports the following safety features:

- Over-voltage and under-voltage detection on V_{DD_INT} and V_{DD_EXT}
- Programmable under-voltage thresholds
- Generates a flash memory power-down signal during a power event
- Glitch rejection
- Programmable (1 - 17 µs) fault delay signal
- GPIO pin safe states

Using the VMU to Handle Failure Events

The following sections describe the methods for handling failure events.

Executing in SRAM

When the VMU detects a fault, it immediately starts to shut down the onboard flash memory. Since application code on the ADSP-CM41x device executes from flash, the code immediately stops executing. Then, the only code that can execute is contained within the VMU interrupt handlers. The code is called when a valid VMU fault happens.

If the application is going to call any functions within the VMU interrupt handlers, those functions must execute in SRAM. Use the `__ramfunc` type in IAR EWARM when declaring functions to be called in the VMU interrupt handler to force sections of code into SRAM. This method instructs the linker to place the section of code into SRAM as part of the initialization process.

Board Design Considerations

According to the *ADSP-CM411F/412F/413F/416F/417F/418F/419F Mixed-Signal Dual-Core Control Processor with ARM Cortex-M4/M0 and 16-bit ADCs Preliminary Datasheet*[2], the VMU specifications call for a maximum ramp rate of 1 V/1.3 ms for V_{DD_INT} and 1 V/250 µS for V_{DD_EXT}. A minimum capacitance of 32 µF is required on the V_{DD_INT} power supply, and a minimum capacitance of 90 µF is required on the V_{DD_EXT} power supply.

The capacitors on the V_{DD_INT} and V_{DD_EXT} power supplies provide a short duration of power to the processor when a power failure occurs. This power, in turn, allows the processor to attempt to safely shut down the flash memory and execute code from the VMU interrupt handler.

In theory, adding more capacitance to the V_{DD_INT} and V_{DD_EXT} power supplies can allow the processor to remain active longer during a power event when more time in the VMU interrupt handler is required.

ADSP-CM419 EZ-Kit Power Loss Performance Example for V_{DD_INT}

The ADSP-CM419 EZ-Kit was designed with ~43 µF of capacitance on V_{DD_INT}. Figure 2 shows an oscilloscope waveform of V_{DD_INT} dropping from the nominal 1.21 V to under the 1.08 V threshold, where the power is too low for the processor to continue to function properly.

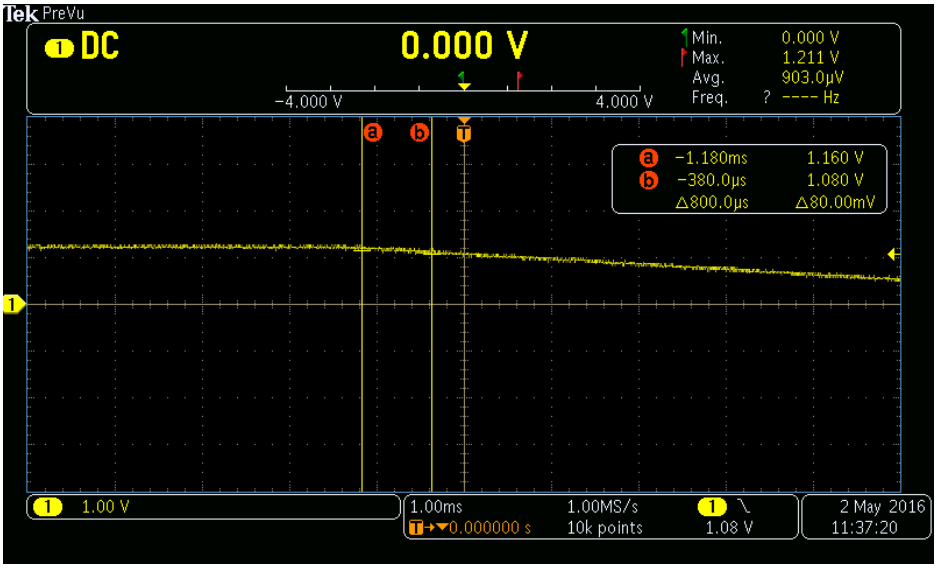

Figure 2: Waveform of V_{DD_INT} Power Failure

The PCB capacitance allows the V_{DD_INT} domain to remain powered for 800 µs before reaching the 1.08 V low point where the ADSP-CM41x device no longer functions properly. A custom PCB design could require adding capacitance to the V_{DD_INT} or V_{DD_EXT} supplies depending on the processor load.

Figure 3 shows the response time of the GPIO Pin Safe States relative to the V_{DD_INT} power failure. The GPIO tested is located at test point TP50 on the ADSP-CM419F EZ-Kit.

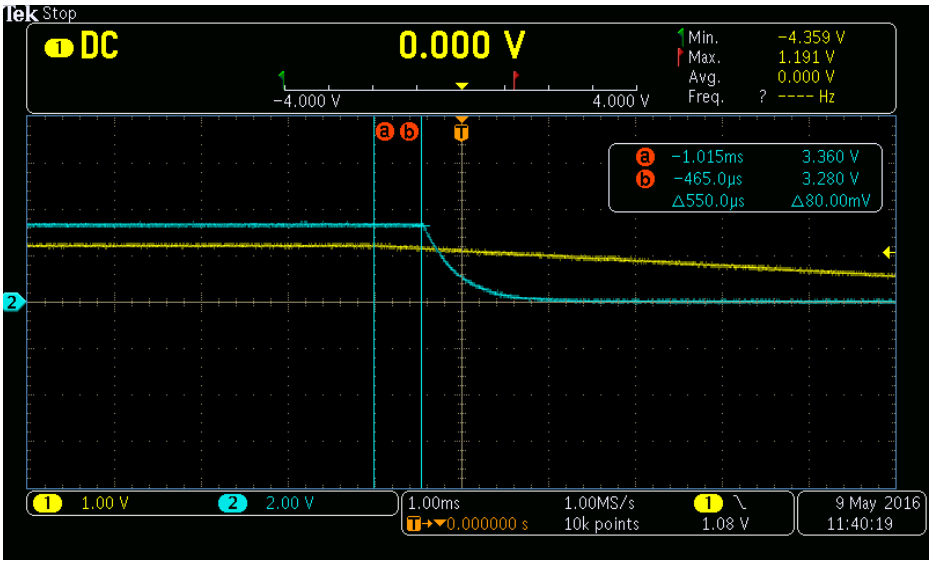

Figure 3: Waveform of V_{DD_INT} and GPIO Pin Safe State on TP50

In the *Example Application Project* in the ZIP file [3] associated with this EE-Note, the PORTB pins are used for GPIO Pin Safe States. During normal operation, the PORTB pins are high, whereas they are driven to the low Pin Safe State when an event triggers the VMU. The GPIO Pin Safe States engage ~550 µS after the start of the V_{DD_INT} power failure.

ADSP-CM419F EZ-Kit Clock Loss Performance Example for GPIO Pin Safe States

When properly configured, a dead or bad clock signal triggers the GPIO Pin Safe States through the VMU. To test this feature, the 30 MHz oscillator on the ADSP-CM419F EZ-Kit is disconnected from the ADSP-CM419 processor by removing resistor R5 from the PCB.

Figure 4 shows an oscilloscope plot obtained after replacing the 30 MHz oscillator on the ADSP-CM419F EZ-Kit with a 30 MHz waveform. The waveform was generated by a Keysight 33600A Series waveform generator connected to the ADSP-CM419F EZ-Kit using a SMC-to-BNC shielded cable.

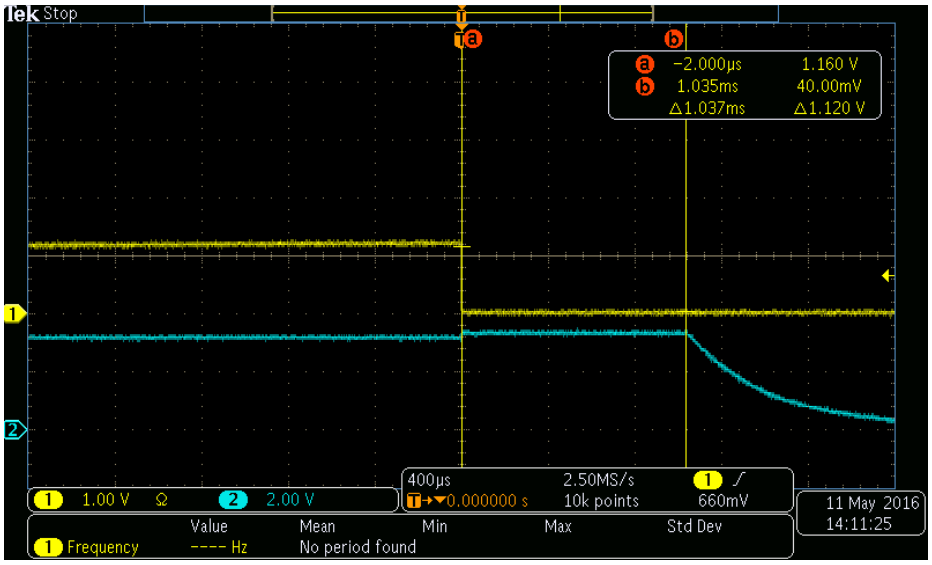

Figure 4: Waveform of GPIO Pin Safe State Relative to Dead Clock

As shown, the GPIO Pin Safe State engages ~1.037 ms after the clock signal to the processor is disconnected. Since the x-axis is set to 400 µS per division, the 30 MHz clock signal appears as a solid line on the oscilloscope.

VMU Operation

The VMU module provides voltage monitoring on the V_{DD_INT} and V_{DD_EXT} power rails upon power-up of the processor. When enabled, the VMU triggers an interrupt for a power event in either the V_{DD_INT} or V_{DD_EXT} power domain. The VMU provides a control signal to the embedded flash memory to facilitate its power-down requirements when an over-voltage or under-voltage power event occurs (regardless of whether it is enabled or disabled).

The VMU also features glitch rejection, which provides the ability to ignore small temporary glitches in the power supplies. These VMU comparators have limited programmability through the `REF_PRG[1:0]` field of the `PADS1_VMU_CTL[5:4]` register. While comparator high thresholds (Over Voltage) are limited by process constraints, low thresholds (Under Voltage) are programmable via `PADS1_VMU_CTL .REF_PRG[1:0]` bits.

`REF_PRG[0] = 0` → UVLO_I1 //Under Voltage threshold detection for (VDD_INT)
`REF_PRG[0] = 1` → UVLO_I2 //Under Voltage threshold detection for (VDD_INT)
`REF_PRG[1] = 0` → UVLO_E1//Under Voltage threshold detection for (VDD_EXT)
`REF_PRG[1] = 1` → UVLO_E2//Under Voltage threshold detection for (VDD_EXT)

Refer to the device datasheet for the exact under voltage and over voltage thresholds.

Configuring the VMU

The VMU has several associated registers in the PADS register group. The control register for the VMU is `PADS_VMU_CTL`.

GPIO Safe States

One of the important functional safety features included on ADSP-CM41x devices is the concept of *GPIO Safe States*. The GPIO Safe States allows the user to set a predetermined value to the GPIOs (high, low, or tristate) in the event of a power fault or a clock fault. The Pin-Safe State block resides in the VMU but can

be directly asserted when faults are detected from the OCU or OSCWD or when using the Trigger Routing Unit (TRU).

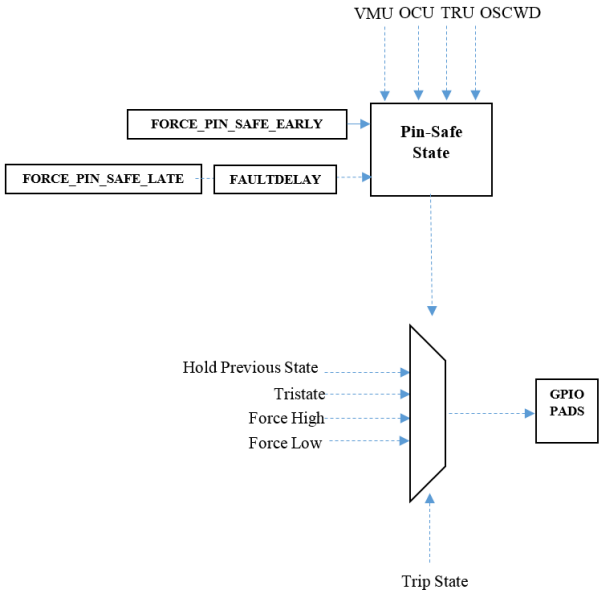

Figure 5: Pin Safe States

The VMU has an input trigger (FAULT_TRIG_IN) that is connected to the OCU module and the OSCWD. It allows a clocking event to trigger the VMU to activate the GPIO Safe States as shown in Figure 6.

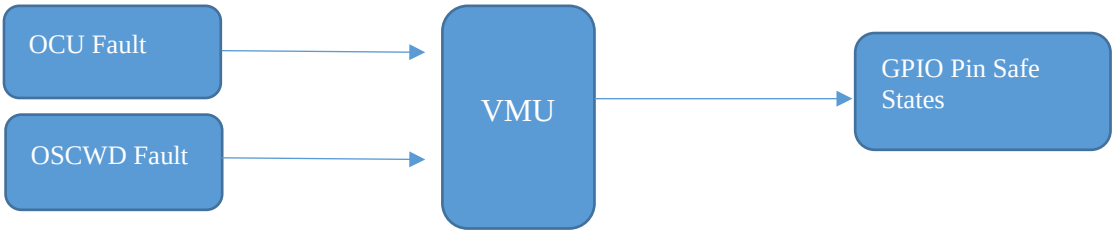

Figure 6: Event Sequence for GPIO Pin Safe States

VMU Use Case Example

Figure 7 shows a typical use case for the ADSP-CM41x processor in a PV solar application. The application uses a PWM output from the ADSP-CM41x processor to control the MOSFETs or IGBTs in a PV inverter. The GPIOs control the relays that connect the inverter from the grid or power sources. The safety features of the VMU set the PWMs to specific states when a power or clock event occurs and shut down the relays.

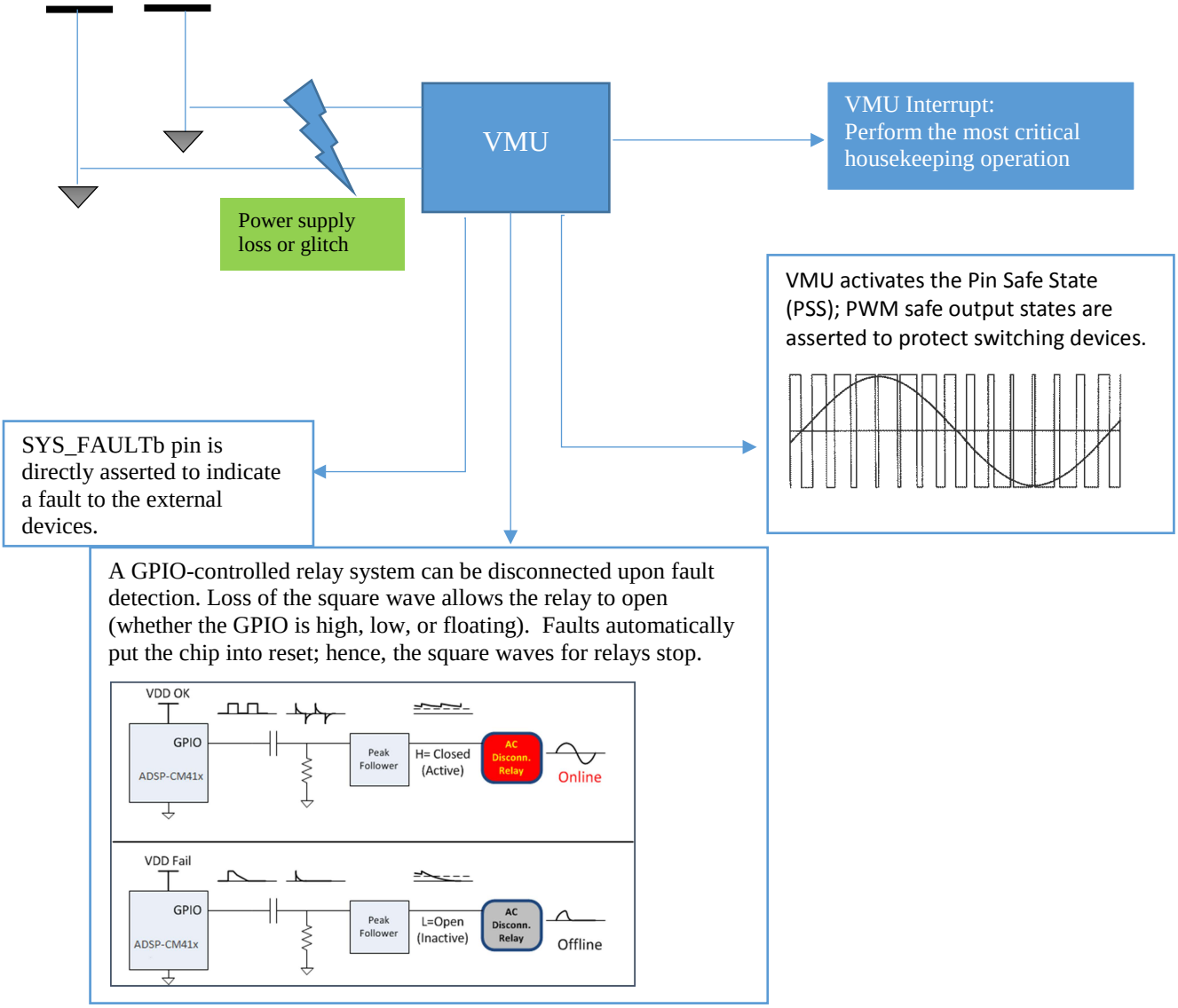

Figure 7: Typical Use Case Example

The VMU performs a controlled shutdown of the PV inverter. In Figure 7, when a power failure or glitch in the V_{DD_INT} or V_{DD_EXT} power domains occurs, the properly configured VMU can trigger the GPIO Pin Safe States to: set the PWM signals to a desired state, shut down disconnected relays, and trigger an interrupt to handle fast and basic essential code. The flash memory is also powered down to avoid corruption of data.

Default Protection from Boot or Reset

Several features of the VMU and OSCWD are active from booting or a system reset of the processor.

- At system reset, the OSCWD is only enabled to check for a dead clock and for bad oscillation, as controlled by the bad oscillator upper frequency limit (`CGU_OSCWDCTL.BOUF`) bit field. With its default settings, the OSCWD module asynchronously asserts its fault signal when it detects these

conditions. This feature protects the system prior to the boot ROM applying the factory trim values to the AUX_CLK oscillator for the OSCWD.

- At system reset, the VMU supports emergency power down. If the VMU detects a sudden loss of V_{DD_EXT} and V_{DD_INT}, it asserts the power down signal to the embedded flash controller.

Enabling the VMU, OCSWD and OCU

Once the boot ROM has completed, configure and enable the VMU at any point in the application code. Enable the OSCWD and the OCU modules after configuring and enabling the CGU module. If the OSCWD and OCU are enabled before the CGU has been configured, false errors can occur.

Control of Common Fault Responses

Table 1 shows the functional block, register bit or signal that controls typical fault events.

Diagnostic Signal/Event	Response	Enabling Control in Software
VMU Fault signal	Pin Safe State assertion	Directly controlled from VMU
	SYS_FAULTb pin assertion	Directly controlled from VMU
	Flash shutdown	Directly controlled from VMU
OSCWD dead clock fault signal	Pin Safe State assertion	Through CKNG reset trip assertion
	SYS_FAULTb pin assertion	CGU_OSCWDCTL.FAULTEN | CGU_OSCWDCTL.FAULTPINDIS Or Through CKNG reset trip assertion
	CLKNG reset trip assertion	CGU_OSCWDCTL.CNGEN
	Flash shutdown	Through CKNG reset trip assertion
CKNG reset trip assertion	Pin Safe State assertion	PADS_VMU_TRIPEN.OSC0CLK
	Flash shutdown	PADS_VMU_TRIPEN.OSC0CLK
	SYS_FAULTb pin assertion	PADS_VMU_TRIPEN.OSC0CLK
OSCWD asynchronous fault signals	SYS_FAULTb pin assertion	SYSBLK_FAULT_TRIPEN.OSC0FAULT | PADS_VMU_TRIPEN.OSC0FAULT
	Pin Safe State assertion	PADS_VMU_TRIPEN.OSC0FAULT
	CKNG reset trip assertion	CGU_OSCWDCTL.CNGEN
	Flash shutdown	PADS_VMU_TRIPEN.OSC0FAULT
OCU dead clock fault signal	SYS_FAULTb pin assertion	SYSBLK_FAULT_TRIPEN.OCU0CLK | PADS_VMU_TRIPEN.OCU0CLK
	Pin Safe State assertion	PADS_VMU_TRIPEN.OCU0CLK
	CKNG reset trip	SYSBLK_CLKNG_TRIPEN.OCU0CLK & CGU_OSCWDCTL.CNGEN
	Flash shutdown	PADS_VMU_TRIPEN.OCU0FAULT
OCU asynchronous fault signals	SYS_FAULTb pin assertion	SYSBLK_FAULT_TRIPEN.OCU0FAULT | PADS_VMU_TRIPEN.OCU0CLK
	Pin Safe State assertion	PADS_VMU_TRIPEN.OCU0CLK
	CKNG reset trip	SYSBLK_CLKNG_TRIPEN.OCU0FAULT & CGU_OSCWDCTL.CNGEN
	Flash shutdown	PADS_VMU_TRIPEN.OCU0CLK

Table 1: Event Responses

Example Code

The ZIP file [3] associated with this EE-Note contains an example project that can be used to properly configure the device using the features discussed throughout this document. It was developed and tested using the ADSP-CM419F EZ-Kit evaluation system.

Example Code Flow

The code flow for the example is shown in Figure 8. As described in Oscillator Watchdog (OSCWD), the OSCWD block is part of the CGU. It must first be initialized properly to define the internal clock tree.

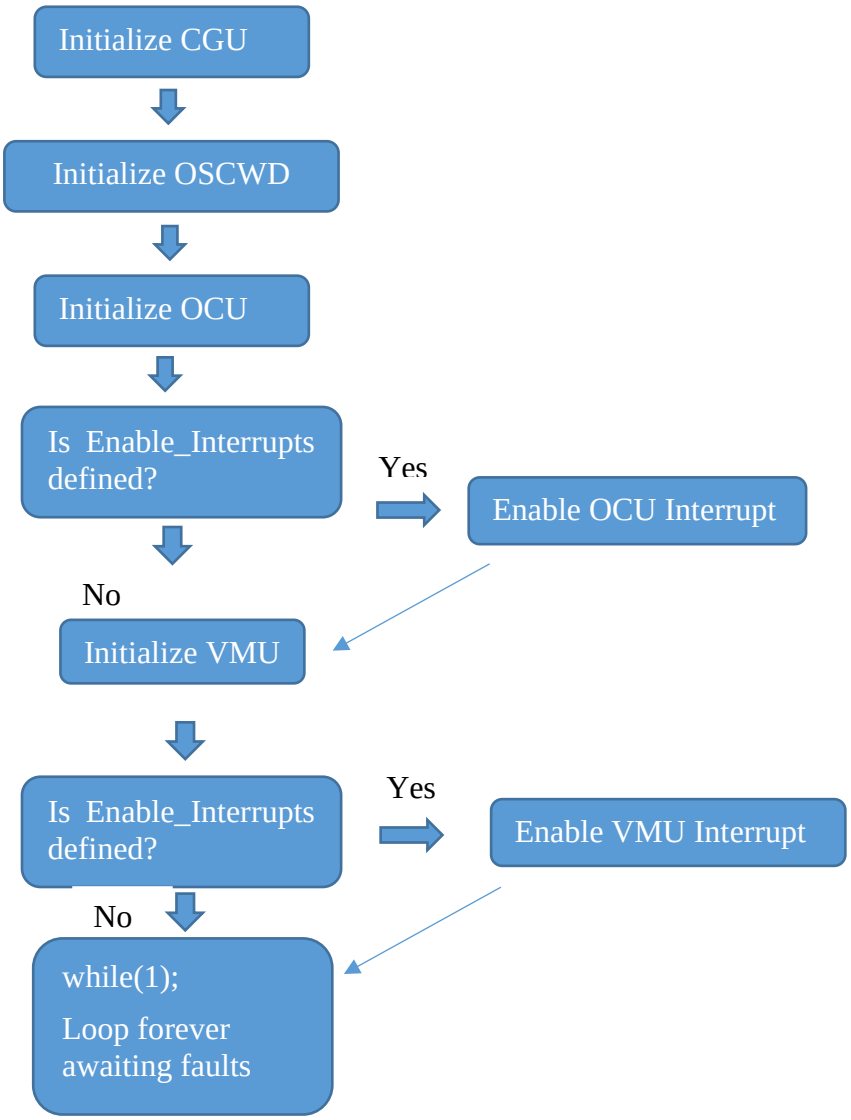

Figure 8: Flow Diagram of Example Application

Initializing the OSCWD

In the provided example application, several `#define` directives in `main.c` are used to configure the OSCWD. Listing 1 shows the `#define` directives for the OSCWD.

```
//Defines for the Oscillator Watchdog Unit.
#define FAULTPINDIS 0      //Fault Pin Disable
#define LOCK 0             //Lock OSCWDCTL register
#define HODF 21            //Watchdog Harmonic Lower Frequency Limit
#define HODEN 1            //Harmonic Oscillation Detection Enable
#define CNGEN 1            //Clock Not Good Enable
#define BOUF 0             //Bad Oscillator Frequency Limit
#define BOUEN 1            //Bad Oscillator Frequency Limit Detection Enable
#define FAULTEN 1          //Fault Enable
#define MONDIS 0           //OCU Monitor Disable
```

Listing 1: OSCWD Configuration Definitions

The Harmonic Oscillator Detection Frequency (`CGU_OSCWDCTL.HODF`) setting is determined by the input clock to the ASDP-CM41x device. Table 2 shows a list of common input clocks.

CGU_OSCWDCTL. HODF[5:0]	Subharmonic Frequency (MHz)	Nominal Lower Fail Limit (MHz)	Input Clock Frequency (MHz)	Nominal Upper Fail Limit (MHz)	Second Harmonic Frequency (MHz)
14	10	14	20	28	40
21	15	21	30	42	60
37	25	37	50	74	100

Table 2: HODF Frequency Selection Examples for Fundamental Crystals

The `CGU_OSCWDCTL.BOUF` setting is determined by the equation:

Upper Frequency Limit = `CGU_OSCWDCTL.BOUF`[4:0] * 2 MHz + 32 MHz

The example application assumes a 30 MHz input clock because that is the input clock present on the ADSP-CM419F EZ-Kit evaluation platform.

The function `osc_wd_Enable()` enables the OSCWD. It is located in the `OSCWD_Init.c` file and shown in Listing 2. This function takes the values from the above `#define` directives and loads them into the `OSCWDCTL` register.

```
void osc_wd_Enable(int FAULTPINDIS, int LOCK, int HODF, int HODEN, int CNGEN, int BOUF, int
BOUEN, int FAULTEN, int MONDIS){

  *pREG_CGU0_OSCWDCTL|=((FAULTPINDIS<<BITP_CGU_OSCWDCTL_FAULTPINDIS)&
BITM_CGU_OSCWDCTL_FAULTPINDIS);
  *pREG_CGU0_OSCWDCTL |= ((LOCK << BITP_CGU_OSCWDCTL_LOCK) & BITM_CGU_OSCWDCTL_LOCK);
  *pREG_CGU0_OSCWDCTL |= ((HODF << BITP_CGU_OSCWDCTL_HODF) & BITM_CGU_OSCWDCTL_HODF);
  *pREG_CGU0_OSCWDCTL|=((HODEN << BITP_CGU_OSCWDCTL_HODEN) & BITM_CGU_OSCWDCTL_HODEN);
  *pREG_CGU0_OSCWDCTL|=((CNGEN << BITP_CGU_OSCWDCTL_CNGEN) & BITM_CGU_OSCWDCTL_CNGEN);
  *pREG_CGU0_OSCWDCTL|=((BOUF << BITP_CGU_OSCWDCTL_BOUF) & BITM_CGU_OSCWDCTL_BOUF);
  *pREG_CGU0_OSCWDCTL|=((BOUEN << BITP_CGU_OSCWDCTL_BOUEN) & BITM_CGU_OSCWDCTL_BOUEN);
  *pREG_CGU0_OSCWDCTL|=((FAULTEN<<BITP_CGU_OSCWDCTL_FAULTEN)& BITM_CGU_OSCWDCTL_FAULTEN);
  *pREG_CGU0_OSCWDCTL|=((MONDIS<<BITP_CGU_OSCWDCTL_MONDIS)& BITM_CGU_OSCWDCTL_MONDIS);
}
```

Listing 2: OSCWD Enable Function

Refer to the CGU chapter of the *ADSP-CM41x Mixed-Signal Control Processor with ARM Cortex-M4/M0 and 16-bit ADCs Hardware Reference* [4] for more information about the OSCWD module.

Initializing the OCU

The `OCU_Init.c` file contains the OCU initialization function `OCU_Init()`. Listing 3 shows the initialization function.

```
void OCU_Init(unsigned int ref_cnt, unsigned short int clk_ppm, unsigned short int lfo_ppm)
{
  unsigned int sysclk = 0;
  unsigned int sysclk_cnt = 0;
  unsigned int ocu_clk = LFO_CM41x / 16;

  while((*pREG_OCU0_STAT & BITM_OCU_STAT_BUSY));
  *pREG_OCU0_CTL &= 0xFFFFFFFC;  // Clear Measure and Auto bit

  //Get SYS_CLK from CGU
   unsigned short int msel = (*pREG_CGU0_CTL & BITM_CGU_CTL_MSEL) >> BITP_CGU_CTL_MSEL;
   unsigned short int syssel = (*pREG_CGU0_DIV & BITM_CGU_DIV_SYSSEL) >> BITP_CGU_DIV_SYSSEL;
   unsigned short int df = (*pREG_CGU0_CTL & BITM_CGU_CTL_DF) >> BITP_CGU_CTL_DF;
   sysclk = ((CLKIN_CM41x / (df + 1)) * msel) / syssel;
  *pREG_OCU0_REFCNT = ref_cnt;

  // Calculate expected SYSCLK count in OCU_CLKCNT
  sysclk_cnt = (ref_cnt / ocu_clk) * sysclk;
  *pREG_OCU0_MINCNT = (sysclk_cnt * (1 - (clk_ppm*0.000001)) * (1 - (lfo_ppm*0.000001)) - 2);
  *pREG_OCU0_MAXCNT = (sysclk_cnt * (1 + (clk_ppm*0.000001)) * (1 + (lfo_ppm*0.000001)) + 2);
}
```

Listing 3: OCU Initialization Function

The OCU module initialization occurs as follows:

1. The `OCU_Init()` function reads from the CGU registers to obtain the relevant SYSCLK0 clock information.

2. Calculate and set the `OCU_REFCNT`, `OCU_MINCNT` and `OCU_MAXCNT` values based on the values of the clock reference count and clock PPMs.
3. Program the `OCU_CMONCNT` and the `OCU_LMONCNT` registers. The `OCU_CMONCNT` is the core clock frequency (96 MHz); the `OCU_LMONCNT` register is the LFO frequency divided by 16 (1 MHz). The code to set these registers is also in the `main()` function (Listing 4).

```
*pREG_OCU0_CMONCNT = 96000000;
*pREG_OCU0_LMONCNT = 1000000;
```

Listing 4: Initializing OCU_CMONCNT and OCU_LMONCNT Registers

4. Enable the OCU interrupt and the types of faults being monitored. The OCU can detect four fault conditions, and any combination up to and including all of them can be enabled at once:
   - Frequency Fault
   - Core Clock Fault
   - LFO Clock Fault
   - Dead Clock
5. Enable the OCU for manual or automatic mode. In the example application, automatic mode is used to continuously monitor the clocks. Listing 5 shows the code snippet from the `main()` function for enabling the interrupts, faults, and the OCU.

```
#ifdef Enable_Interrupts
  adi_nvic_EnableInt(INTR_OCU0_ERR, 1);
#endif
  OCU_Enable_Fault(FREQ_FAULT);
  OCU_Start_Auto();
```

Listing 5: Enabling OCU Interrupts and Faults

OCU Interrupt

The OCU has a weak interrupt handler called `OCU0_ERR_Handler()` that can be used in the event of a clock signal fault. Listing 6 shows the interrupt handler function.

```
void OCU0_ERR_Handler(void)
{
  if((*pREG_OCU0_STAT & BITM_OCU_STAT_FREQ_FAULT) >> BITP_OCU_STAT_FREQ_FAULT )
  {
    *pREG_PORTA_DATA = 0x2000;
    *pREG_OCU0_STAT &= 0xFFFFFFFE;
  }
}
```

Listing 6: OCU Interrupt Handler

In the example application, LED2 is enabled when the OCU interrupt is triggered, giving a visual indication of a detected clock signal error event. Since the OCU has several different faults that it can detect, such as bad upper and lower frequency limits and harmonic issues, the status of each fault bit can be interrogated to execute different code for each fault. In this example, the OCU is configured to monitor for frequency fault. Once a fault is detected, LED2 is set and the `OCU_STAT.FREQ_FAULT` status bit is cleared.

Refer to the OCU (Oscillator Comparator Unit) chapter of the *ADSP-CM41x Mixed-Signal Control Processor with ARM Cortex-M4/M0 and 16-bit ADCs Hardware Reference* [4] for more information about the OCU.

Initializing the VMU

The VMU has a programmable delay for the fault signal "FAULT_3V". This signal can be multiplexed to an external GPIO that can be used in the application.

In the `VMU_init.c` source file, the function `VMU_Init()` is used to configure the `PADS_VMU_CTL` register, as shown in Listing 7.

```
void VMU_Init(bool vdd_int_en,
              bool vdd_ext_en,
              bool fast_comp_en,
              unsigned short int vdd_int_ref,
              unsigned short int vdd_ext_ref,
              unsigned short int fault_delay)
{
  *pREG_PADS1_VMU_CTL = 0x00;  // Reset VMU

  if(vdd_int_en)
    *pREG_PADS1_VMU_CTL |= BITM_PADS_VMU_CTL_IVDDCOMPEN;  // Enable VMU VDDINT Monitoring
    *pREG_PADS1_VMU_CTL |= ((vdd_int_ref << BITP_PADS_VMU_CTL_REFPRG) &
BITM_PADS_VMU_CTL_REFPRG);  // VDDINT reference value
  if(vdd_ext_en)
    *pREG_PADS1_VMU_CTL |= BITM_PADS_VMU_CTL_EVDDCOMPEN;  // Enable VMU VDDEXT Monitoring
    *pREG_PADS1_VMU_CTL |= (((vdd_ext_ref) << BITP_PADS_VMU_CTL_REFPRG) &
BITM_PADS_VMU_CTL_REFPRG);   // VDDEXT reference value
  if(fast_comp_en)
    *pREG_PADS1_VMU_CTL |= BITM_PADS_VMU_CTL_FCEN;  // Enable VDDINT fast comparator

  *pREG_PADS1_VMU_CTL |= ((fault_delay << BITP_PADS_VMU_CTL_FAULTDLY) &
BITM_PADS_VMU_CTL_FAULTDLY);

  *pREG_PADS1_VMU_CTL |= BITM_PADS_VMU_CTL_EN;  // Enable VMU
  for(int i=0; i<1000; ++i);    //Wait for VMU to activate
}
```

Listing 7:Initializing the VMU

VMU Interrupt

The VMU has two interrupt functions, one for V_{DD_INT} detection and one for V_{DD_EXT} detection. In the example application, the VMU is set to monitor the voltage on V_{DD_INT}. The VMU interrupts are declared using `__ramfunc`, so they run in SRAM space and not from flash memory. Listing 8 shows the `VMU0_VDDINT_EVT_Handler()` function.

```
void VMU0_VDDINT_EVT_Handler(void){
  *pREG_PORTF_DATA = 0x0004;
}
```

Listing 8: VMU V_{DD_INT} Handler Function

For this example, LED3 on the EZ-Kit is used to provide a visual indication that the `VMU0_VDDINT_EVT_Handler` has been triggered.

The VMU interrupts execute in SRAM. Therefore, anything in the interrupt function will execute even after the flash memory has been powered down and until power dips below the low-level threshold.

Refer to the VMU (Voltage Monitoring Unit) chapter of the *ADSP-CM41x Mixed-Signal Control Processor with ARM Cortex-M4/M0 and 16-bit ADCs Hardware Reference* [4] for more information about the VMU.

Initializing GPIO Safe States

In the `GPIO_Safe_State_Init.c` source file is the function for initializing the safe states for each PORT, as shown in Listing 9. The `PADS0_PORTx_TRIPST` register is responsible for setting the Safe State of the pins, whereas the `PADS0_PORTx_TRIPSEL` register is responsible for setting the early or late response of the GPIO Safe States.

The included example application was designed as an example of setting the GPIO Safe Sates for all of the port pins at once. Each GPIO can be individually set to a different safe state depending on the application requirements.

```
void GPIO_Safe_State_Init(type_PORT_NAME port_name,
                          type_SAFE_STATE safe_state,
                          type_RESPONSE response
                          )
{
  unsigned int temp_reg=0;
  switch (port_name)
  {
  case PORTA:
    *pREG_PORTA_FER_CLR  = 0xFFFF;
    *pREG_PORTA_DIR_SET  = 0xFFFF;
    *pREG_PORTA_DATA_SET  = 0xFFFF;
    // Configure safe state
    if(safe_state == HOLD)
    {
      for(int i=0; i<32; i+=2)
      {
        temp_reg |= (HOLD << i) & (3 << i);
      }
      *pREG_PADS0_PORTA_TRIPST = temp_reg;
    }
    else if(safe_state == TRISTATE)
    {
      for(int i=0; i<32; i+=2)
        temp_reg |= (TRISTATE << i) & (3 << i);
      *pREG_PADS0_PORTA_TRIPST = temp_reg;
```

```
    }
    else if(safe_state == LOW)
    {
      for(int i=0; i<32; i+=2)
        temp_reg |= (LOW << i) & (3 << i);
      *pREG_PADS0_PORTA_TRIPST = temp_reg;
    }
    else if(safe_state == HIGH)
    {
      for(int i=0; i<32; i+=2)
        temp_reg |= (HIGH << i) & (3 << i);
      *pREG_PADS0_PORTA_TRIPST = temp_reg;
    }
    // Response
    if(response == EARLY)
      *pREG_PADS0_PORTA_TRIPSEL = 0;
    else if(response == LATE)
      *pREG_PADS0_PORTA_TRIPSEL = BITM_PADS_PORT_TRIPSEL_SEL;
    break;
….}
```

Listing 9: GPIO Safe State Initialization

References

[1] *ADSP-CM419F EZ-KIT® Manual.* Rev 1.0, April 2016. Analog Devices, Inc.

[2] *ADSP-CM411F/412F/413F/416F/417F/418F/419F Mixed-Signal Dual-Core Control Processor with ARM Cortex-M4/M0 and 16-bit ADCs Preliminary Datasheet.* Rev PrB, February 2016. Analog Devices, Inc.

[3] *Associated ZIP File for Protecting ADSP-CM41x Devices from Input Clock/Power Supply Faults (EE-393).* Rev 2, December 2017. Analog Devices, Inc.

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

Document History

Revision	Description
Rev 1 – December 15th, 2016 *by David H.*	Initial release.
Rev 2 – April 25th, 2018 *by Prasanth R.*	Added Control of Common Fault Responses figure. Added Clock and Power Supply Safety Blocks Block Diagram.