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

Contributed by David H. *Rev 1 – January 3, 2017*

Introduction

The ADSP-CM41x processors feature peripherals such as the Oscillator Watchdog (OSCWD), Oscillator Comparator Unit (OCU), and Voltage Monitoring Unit (VMU), which can be used to help determine faults in the clocks, identify power supply issues for the processor, and 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.

Oscillator Watchdog (OSCWD)

The Oscillator Watchdog is part of the Clock Generation Unit (CGU) and 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, which is used as a reference point for 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 Oscillator Comparator Unit (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 errors occur in the SYSCLK0 domain, it is possible to generate both a fault and an interrupt from the OCU. Combined with the OSCWD, this 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 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); and, in the case of a dead input clock, a fault will occur.

The OCU has two operational modes:

- **Manual** - a single detection cycle is run (stops upon completion whether a fault occurs or not and no longer monitors clocks).
- **Automatic** - a continuous detection cycle is run (remains enabled and continues monitoring the clocks until manually stopped, even if a fault occurred).

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

Configuring the OCU

The OCU has several registers that need to 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 a LFO frequency of 16 MHz, the above equation yields a Clock Reference Count of 1 MHz. The input and LFO clock sources populated on the EZ-KIT evaluation system both have a PPM of 50.

Voltage Monitoring Unit (VMU)

The Voltage Monitoring Unit (VMU) provides over-voltage and under-voltage detection by monitoring the V_{DD_EXT} and V_{DD_INT} power domains and generates asynchronous signals if 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 itself 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 flash memory power-down signal during power event
- Glitch rejection
- Programmable (1 – 17 μs) Fault Delay signal
- GPIO Pin Safe States

Operation of the VMU

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 will trigger an interrupt for a power event in either the V_{DD_INT} or V_{DD_EXT} power domain. The VMU will also, regardless if enable or disabled, provide a control signal to the embedded flash memory to facilitate its power-down requirements in case an over-voltage or under-voltage power event occurs.

The VMU also features glitch rejection, which provides the ability to ignore small temporary glitches in the power supplies. The VMU has both slow and fast comparators for V_{DD_INT}, and only a slow comparator for V_{DD_EXT}. These comparators have limited programmability, located in the `REF_PRG[1:0]` field located in the `TEPADS1_VMU_CTL[5:4]` register, with programmable values as follows:

REF_PRG[0] = 0 → UVLO_I1 = 1.124 V (typ.)
REF_PRG[0] = 1 → UVLO_I2 = 1.11 V (typ.)
REF_PRG[1] = 0 → UVLO_E1 = 2.97 V (typ.)
REF_PRG[1] = 1 → UVLO_E2 = 2.80 V (typ)

Configuring the VMU

The VMU has several registers associated with it, which can be found in the TEPADS1 group. The control register for the VMU is `TEPADS1_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 tri-state) in the event of a power fault or a clock fault.

The VMU has an input trigger (FAULT_TRIG_IN) that is connected to the OCU module, which allows a clocking event to trigger the VMU to activate the GPIO Safe States, as shown in Figure 1.

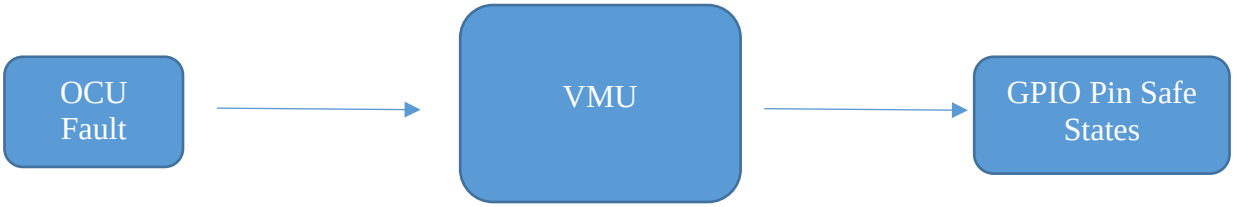

Figure 1. Event Sequence for GPIO Pin Safe States

Default Protection from Boot/Reset

Several features of the VMU and OSCWD are active from Boot/Reset of the processor.

At system reset, the OSCWD is only enabled to check for dead clock and for bad oscillation, as controlled by the Bad Oscillator Upper Frequency (BOUF). With its default settings, it will asynchronously assert its fault signal if these conditions are detected. This is adequate to protect 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 will assert the Power Down signal to the embedded flash controller.

When to Enable VMU, OCSWD and OCU

Once the boot ROM has completed, the VMU can be configured and enabled at any point in the application code. However, it is recommended that the OSCWD and the OCU be enabled after the CGU has been configured and enabled. If the OSCWD and OCU are enabled before the CGU has been configured, false errors may occur.

Best Practices to Utilize the VMU when Failure Events Occur

The following sections describe some best practice techniques 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. The only code that can execute at that point is code contained within the VMU interrupt handlers, which is called at the time of a valid VMU fault.

If the application is going to call any functions within the VMU interrupt handlers, those functions must execute in SRAM. The easiest way to force sections of code into SRAM is to use the `__ramfunc` type when declaring functions that will be called in the VMU interrupt handler. This instructs the linker to place that 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, which 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 if 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 is 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.

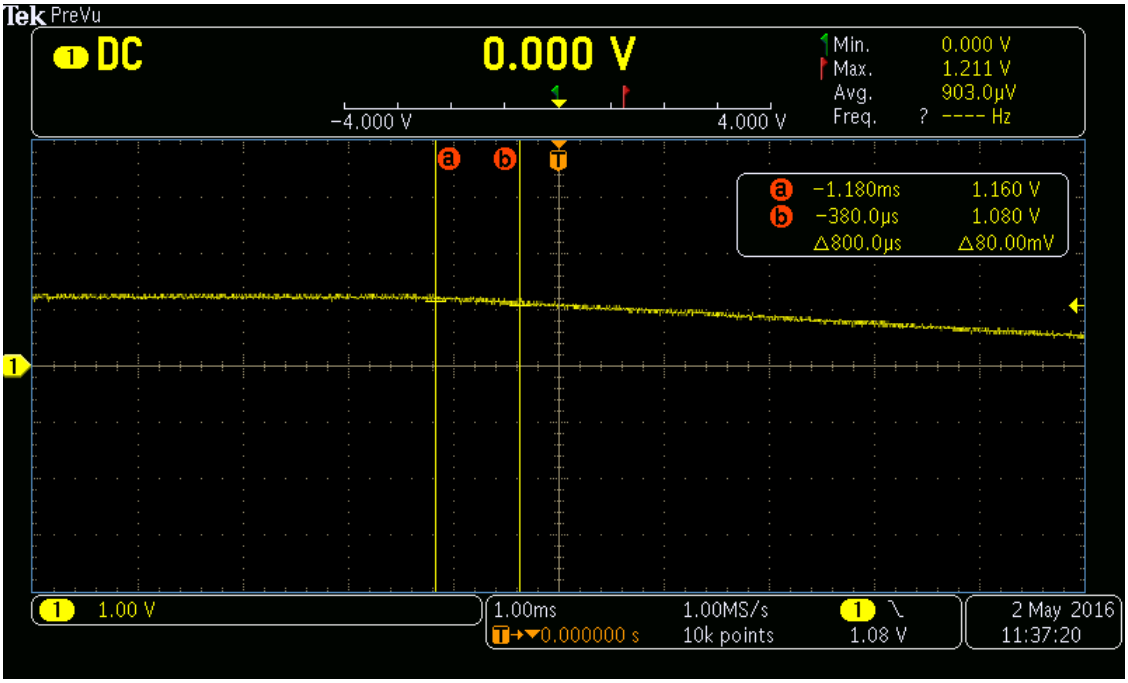

Figure 2. Waveform of V_{DD_INT} Power Failure

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 will no longer function properly. A custom PCB design might require adding capacitance to the V_{DD_INT} or V_{DD_EXT} supplies, depending on 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.

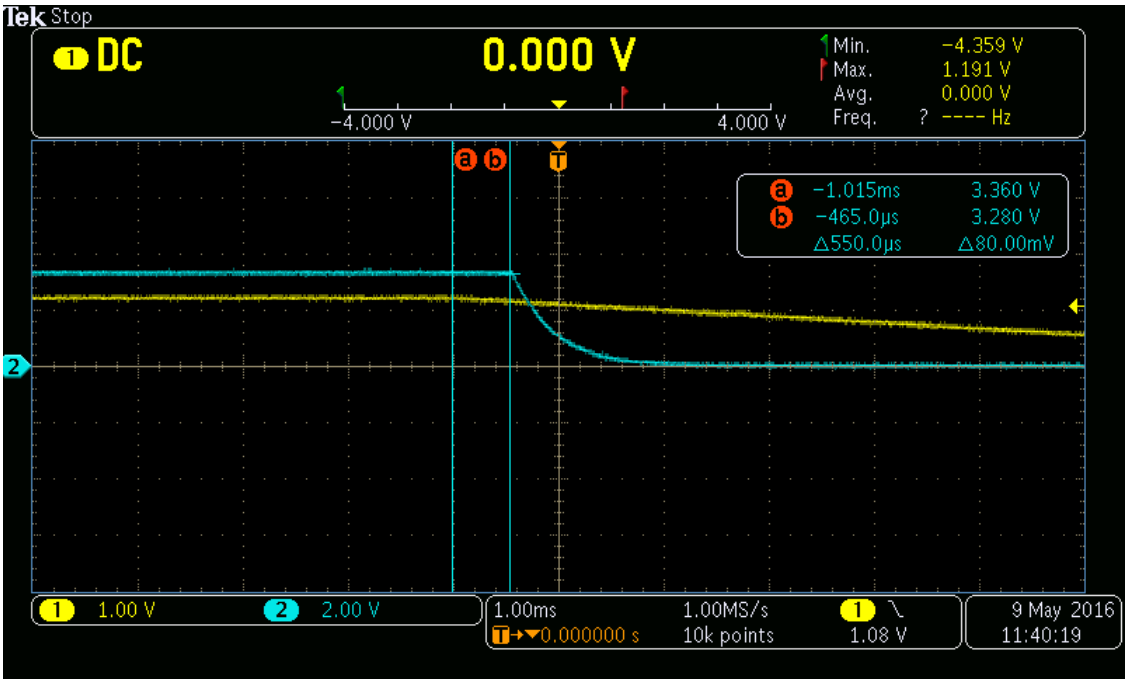

Figure 3. Waveform of V_{DD_INT} and GPIO Pin Safe State on TP50

In the *example application project*[3] associated with this EE-note, 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 will trigger the GPIO Pin Safe States via 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 is an oscilloscope plot obtained after replacing the 30 MHz oscillator on the ADSP-CM419F EZ-Kit with a 30 MHz waveform generated by a Keysight 33600A Series waveform generator, connected to the ADSP-CM419F EZ-Kit using a SMC-to-BNC shielded cable.

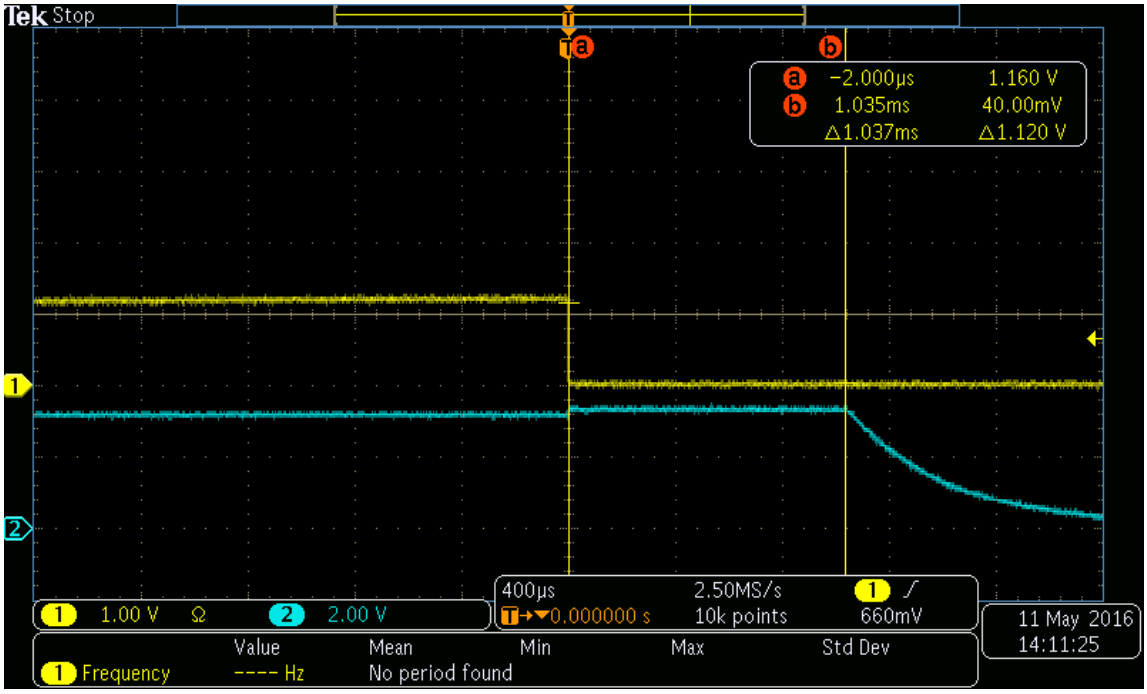

Figure 4. Waveform of GPIO Pin Safe State Relative to Dead Clock

As can be seen, the GPIO Pin Safe States is engaged ~1.037 ms after the clock signal to the processor was 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.

Typical Use Case Example

A typical use case for the ADSP-CM41x processor in a PV Solar application uses PWM output from the ADSP-CM41x processor to control MOSFETs or IGBTs in a PV inverter, as shown in Figure 5. GPIOs are then used to control relays that connect the inverter from the grid or power sources. In this use case, the safety features of the VMU can be extremely useful in setting PWMs to specific states when a power or clock event occurs, along with shutting down the relays.

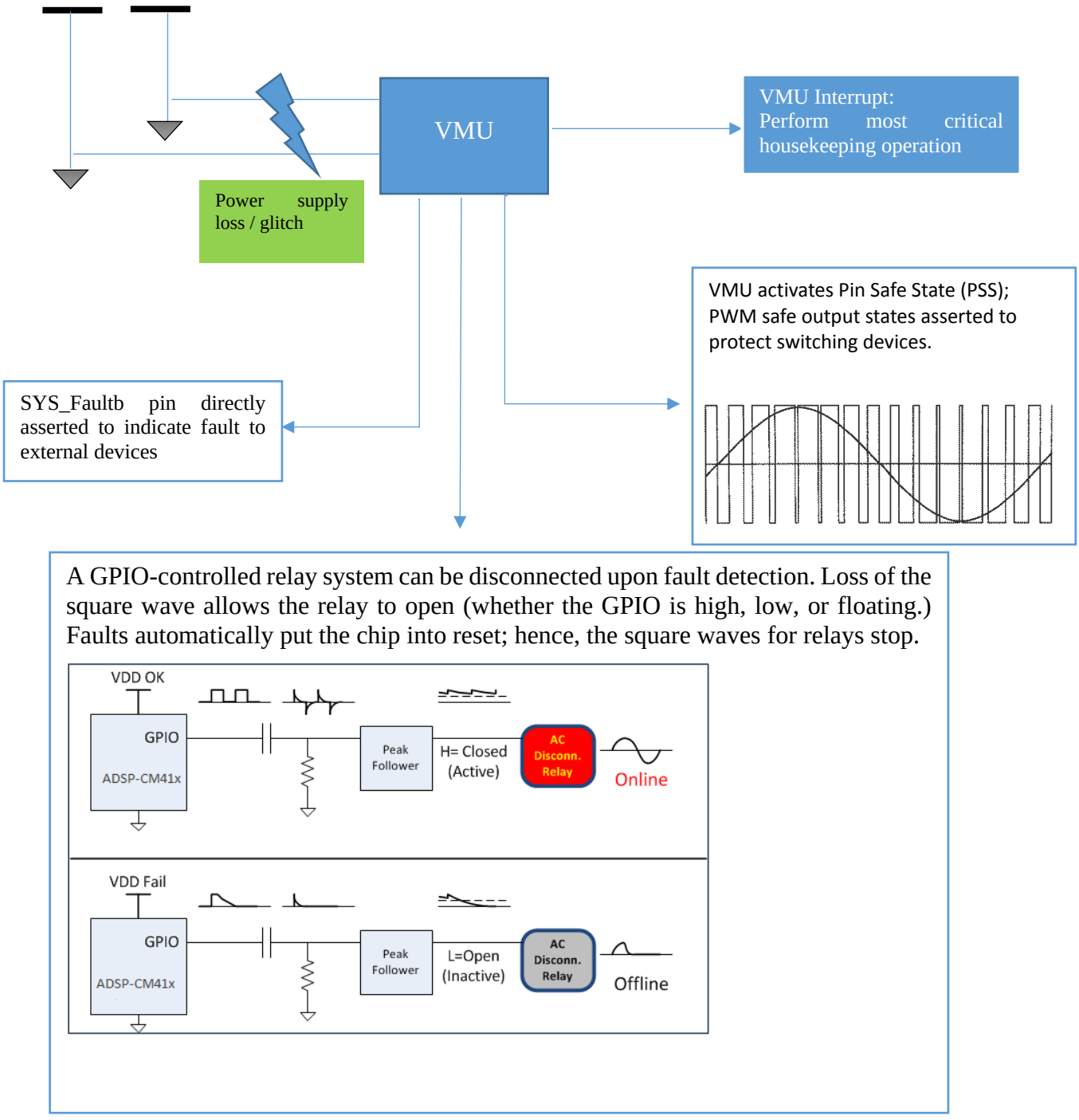

Figure 5. Typical Use Case Example

In this typical use case, the VMU can be used to do a controlled shutdown of the PV inverter. In Figure 5, 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 disconnect relays, and trigger an interrupt to handle fast and basic essential code. The flash memory is also powered down to avoid corruption of data.

Example Code

The ZIP file associated with this EE-note contains an example project used to properly configure the device to utilize 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 6. As described in Oscillator Watchdog (OSCWD), the OSCWD block is part of the CGU, which must first be initialized properly to define the internal clock tree.

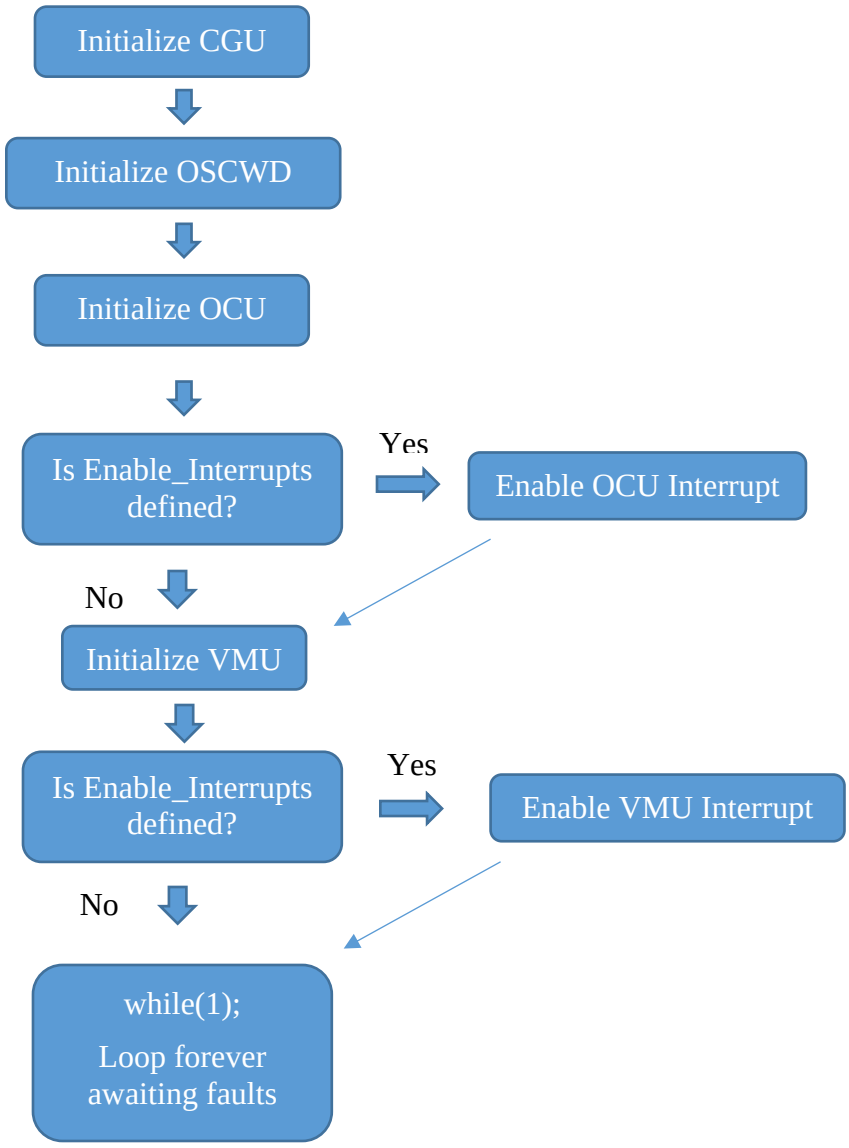

Figure 6. Flow Diagram of Example Application

Initializing the OSCWD

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

```
//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 (HODF) setting is determined by the input clock to the ASDP-CM41x device. Table 1 shows a list of common input clocks.

HODF[5:0]	Sub-Harmonic Frequency	Nominal Lower Fail Limit	Input Clock Frequency	Nominal Upper Fail Limit	2nd Harmonic Frequency
14	10 MHz	14 MHz	20 MHz	28 MHz	40 MHz
21	15 MHz	21 MHz	30 MHz	42 MHz	60 MHz
37	25 MHz	37 MHz	50 MHz	74 MHz	100 MHz

Table 1: HODF Frequency Selection Examples for Fundamental Crystals

The BOUF setting is determined by the equation:

Target Upper Frequency Limit = BOUF[4:0] * 2 MHz + 32 MHz

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

The function to enable the OSCWD is called osc_wd_Enable(), which 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

More information on the OSCWD can be found in the CGU Chapter of the ***ADSP-CM41x Hardware Reference Manual***[4].

Initializing the OCU

The OCU_Init.c file contains the OCU initialization function, OCU_Init(), shown in Listing 3.

```
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_Init() function reads from the CGU registers to obtain the relevant SYSCLK0 clock information, after which the OCU_REFCNT, OCU_MINCNT and OCU_MAXCNT can be calculated and set based on the

values of the Clock Reference Count and clock PPMs. In addition to these registers, the OCU_CMONCNT and the OCU_LMONCNT registers must also be programmed before the OCU is fully configured. The OCU_CMONCNT is the core clock frequency (96 MHz), and the OCU_LMONCNT 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

The next step is to enable the OCU interrupt and enable 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

After the OCU is configured for error detection, it can be enabled for Manual or Automatic mode. In the example application, Automatic mode is used to continuously monitor the clocks. The code for enabling the interrupts/faults and the OCU itself is shown in the code snippet from the `main()` function in Listing 5.

```
#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, as shown in Listing 6.

```
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 `Freq_Fault` status bit is cleared.

More information on the OCU can be found in the Oscillator Comparator Unit chapter in the ***hardware reference manual***.

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. Before enabling the VMU, PORTB is configured to connect to the VMU to provide outputs that can be tested on the bench, as shown in Listing 7.

```
*pREG_PORTB_FER|=0x0000f;        // When mux[2i+1:2i] is written then fer[i] must be 0
*pREG_PORTB_MUX|=0x00003C00;                  // mux PB 5,6 to VMU_TOUT
*pREG_PORTB_FER|=0x0060;                      // setup PB 6,5 to be in periphial mode
*pREG_TEPADS1_PORTB_TRIPST = 0x00C3C000;      // set portb 7,8 to drive high in trip
state, 11 to high as well for tin toggle
*pREG_TEPADS1_PORTB_TRIPSEL = 0xff7f;         // set 7 to force_early, 8 and rest to
force_late
*pREG_PORTB_DIR_SET = 0x0180;                 // set 7/8 to outputs
*pREG_PORTB_DIR_SET = 0x0800;                 // set 11 to input
*pREG_PORTB_DATA = 0x0000;                    // write 0 to 7/8
```

Listing 7. Initializing PORTB GPIO

In the `VMU_init.c` source file, the function `VMU_Init()` is used to configure the `TEPADS_VMU_CTL` register, as shown in Listing 8.

```
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_TEPADS1_VMU_CTL = 0x00;  // Reset VMU

  if(vdd_int_en)
    *pREG_TEPADS1_VMU_CTL |= BITM_TEPADS_VMU_CTL_IVDDCOMPEN;   // Enable VMU VDDINT
Monitoring
    *pREG_TEPADS1_VMU_CTL   |=   ((vdd_int_ref   <<   BITP_TEPADS_VMU_CTL_REFPRG)   &
BITM_TEPADS_VMU_CTL_REFPRG);  // VDDINT reference value
  if(vdd_ext_en)
    *pREG_TEPADS1_VMU_CTL |= BITM_TEPADS_VMU_CTL_EVDDCOMPEN;   // Enable VMU VDDEXT
Monitoring
    *pREG_TEPADS1_VMU_CTL   |=   (((vdd_ext_ref)   <<   BITP_TEPADS_VMU_CTL_REFPRG)   &
BITM_TEPADS_VMU_CTL_REFPRG);   // VDDEXT reference value
  if(fast_comp_en)
    *pREG_TEPADS1_VMU_CTL  |=  BITM_TEPADS_VMU_CTL_FCEN;    //  Enable  VDDINT  fast
comparator

  *pREG_TEPADS1_VMU_CTL   |=   ((fault_delay   <<   BITP_TEPADS_VMU_CTL_FAULTDLY)   &
BITM_TEPADS_VMU_CTL_FAULTDLY);

  *pREG_TEPADS1_VMU_CTL |= BITM_TEPADS_VMU_CTL_EN;  // Enable VMU
  for(int i=0; i<1000; ++i);    //Wait for VMU to activate
}
```

Listing 8. 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 will run in SRAM space and not from flash memory. Listing 9 shows the `VMU0_VDDINT_EVT_Handler()` function.

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

Listing 9. 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, thus 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.

More information on the VMU can be found in the Voltage Monitoring Unit chapter in the ***hardware reference manual***.

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 10. The `TEPADS0_PORTx_TRIPST` register is responsible for setting the Safe State of the pins, whereas the `TEPADS0_PORTx_TRIPSEL` 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_TEPADS0_PORTA_TRIPST = temp_reg;
    }
    else if(safe_state == TRISTATE)
    {
      for(int i=0; i<32; i+=2)
        temp_reg |= (TRISTATE << i) & (3 << i);
      *pREG_TEPADS0_PORTA_TRIPST = temp_reg;
    }
    else if(safe_state == LOW)
    {
      for(int i=0; i<32; i+=2)
        temp_reg |= (LOW << i) & (3 << i);
      *pREG_TEPADS0_PORTA_TRIPST = temp_reg;
    }
    else if(safe_state == HIGH)
    {
      for(int i=0; i<32; i+=2)
        temp_reg |= (HIGH << i) & (3 << i);
      *pREG_TEPADS0_PORTA_TRIPST = temp_reg;
    }
    // Response
    if(response == EARLY)
      *pREG_TEPADS0_PORTA_TRIPSEL = 0;
    else if(response == LATE)
      *pREG_TEPADS0_PORTA_TRIPSEL = BITM_TEPADS_PORT_TRIPSEL_SEL;
    break;
….}
```

Listing 10. 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 1, December 2016. Analog Devices, Inc.

[4] *ADSP-CM41x Mixed-Signal Control Processor with ARM Cortex-M4/M0 and 16-bit ADCs Hardware Reference.* Rev 0.2, May 2016. Analog Devices, Inc.

Document History

Revision	Description
Rev 1 – January 3rd, 2017 *by David H.*	Initial release.