Application Note

Lockup Avoidance for UCD3138 Devices

Xuemei Lu

ABSTRACT

Lockup occurs when a device can no longer be reprogrammed. Lockup can be caused by either an incorrect download process or firmware with bugs. This application note describes the possible reasons that can cause lockup in UCD3138 devices and ways to avoid lockup.

Table of Contents

List of Figures

Trademarks

All trademarks are the property of their respective owners.

1 Why Lockup Can Occur

A lockup can occur when there is a valid checksum that cannot be cleared. Different UCD3138 devices have different numbers of checksum.

A UCD3138(A) device has one single block. There are two locations for checksum and each checksum is 4 bytes.

- 0x7fc – boot checksum, sum of the pflash bytes from address 0 to 0x7fB
- 0x7ffc – overall checksum for pflash, sum of the pflash bytes from address 0 to 0x7ffB

Figure 1-1 is a flowchart showing how UCD3138(A) ROM handles the checksum verification. Regardless of which two checksums is valid, a jump occurs to address 0 to execute code in pflash.

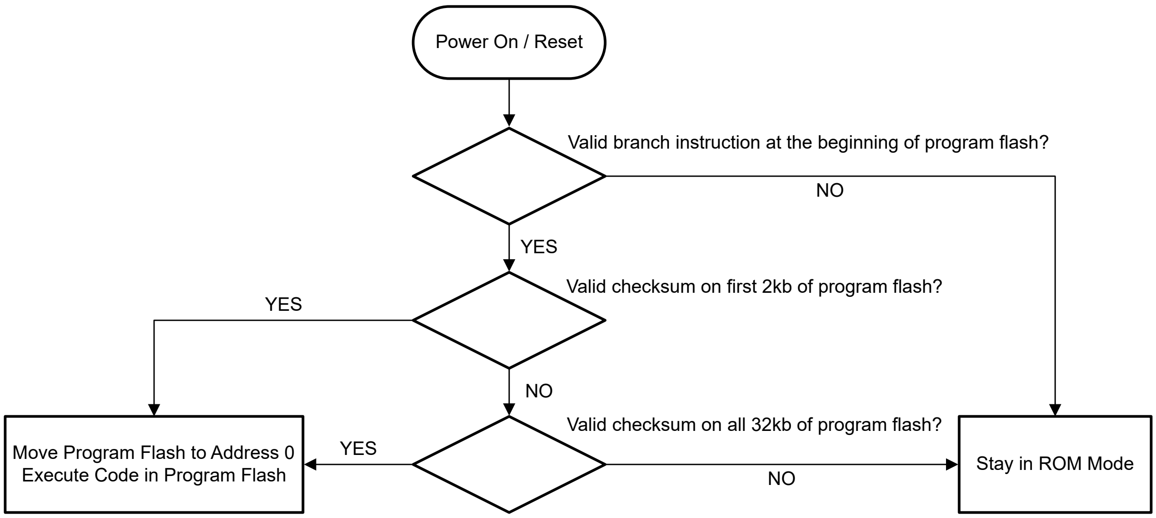


Figure 1-1. Checksum Verification Flow Chart in UCD3138(A) ROM

A UCD3138064(A) device has block 1 and block 2. There are four locations for checksum and each checksum is 4 bytes.

- 0x7fc – boot checksum for block 1, sum of the pflash bytes from address 0 to 0x7fB
- 0x7ffc – overall checksum for block 1, sum of the pflash bytes from address 0 to 0x7ffB
- 0x87fc – boot checksum for block 2, sum of the pflash bytes from address 0x8000 to 0x87fB
- 0xfffc – overall checksum for the 64K program combining block 1 and 2, or for block 2 alone

Figure 1-2 is a flowchart showing how UCD3138064(A) ROM handles the checksum verification.

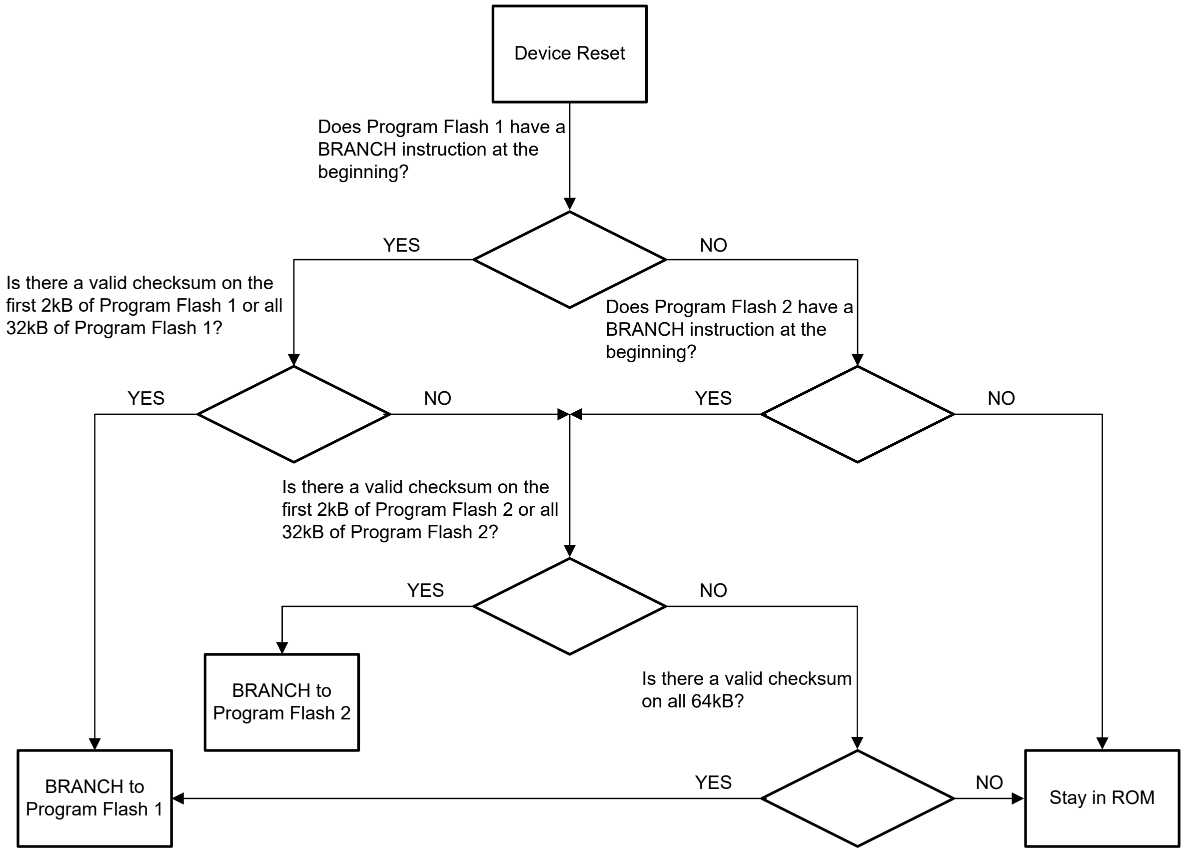


Figure 1-2. Checksum Verification Flow Chart in UCD3138064(A) ROM

A UCD3138128(A) has block0, block1, block2, and block3. There are four locations for checksum and each checksum is 8 bytes.

- 0x7f8 – boot checksum for block 0, sum of the pflash words from address 0 to 0x7f7
- 0x7ff8 – overall checksum for block 0, sum of the pflash words from address 0 to 0x7ff7
- 0xfff8 – overall checksum for the 64K program combining block 0 and 1, sum of the pflash words from address 0 to 0xfff7
- 0x1fff8 – overall checksum for the 128K program, or the 64K program combining block 2 and 3

Figure 1-3 is a flowchart showing how UCD3138128(A) ROM handles the checksum verification.

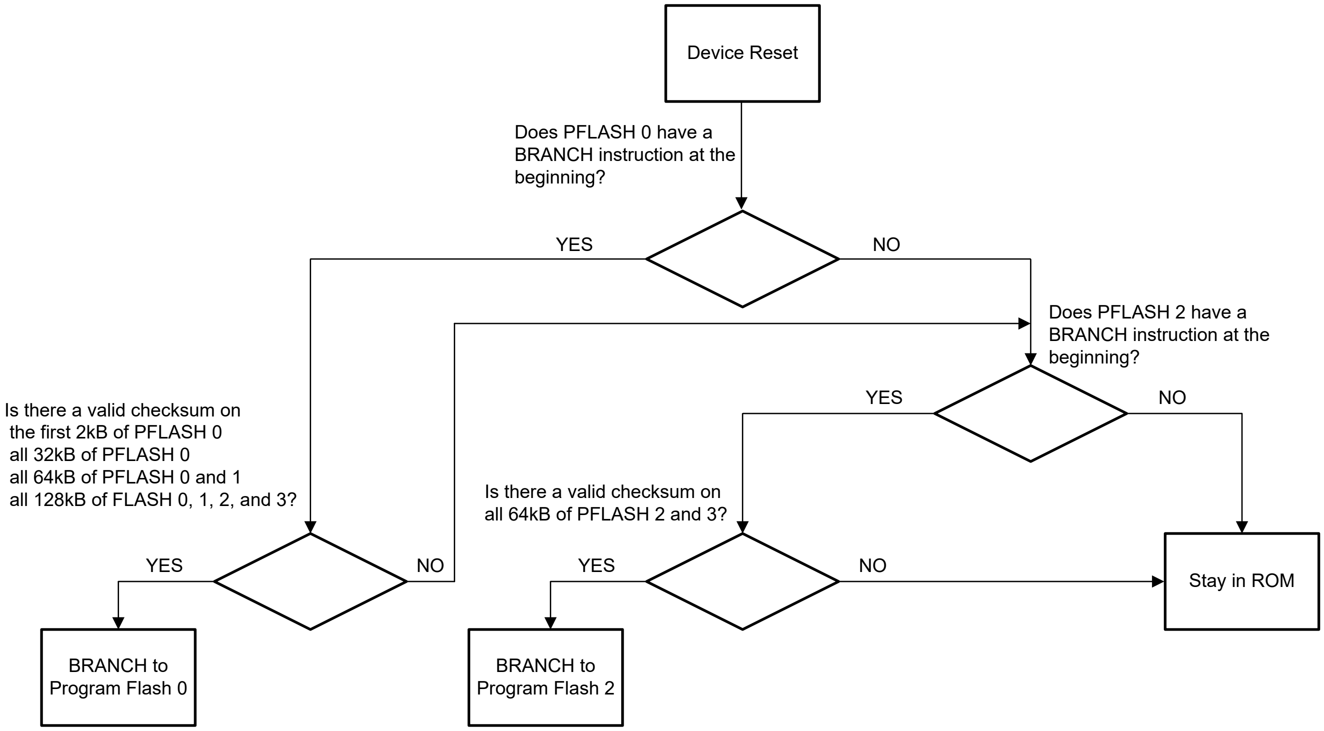


Figure 1-3. Checksum Verification Flow Chart in UCD3138128(A) ROM

2 Reasons for Lockup

2.1 Wrong Code in the load.asm

In load.asm, if the following red lines are removed, but the two blue lines remain, this can cause the device to reset continuously since r4 does not have a valid value. If this is the case, the device locks permanently and these 5 lines are removed entirely or remain.

```
B_fast_interrupt
         .align 4
         .sect ".text"
             .state32
c_int00
;            B      c_int00
             LDR          r13, c_sup_stack_top ; initialize supervisor stack pointer
;            LDR          r4,c_mfbalr1_half0 ;point r4 at program flash base address register
;            MOV          r0,#0x62 ;make block size 32K, address 0, read only
;            STRH   r0,[r4]; store it there
             LDR          r0,c_mfbalr2_half0_load ;set up data flash for write only
             STRH   r0,[r4,#8] ;put it into mfbalr2
```

2.2 Misoperation with TI GUI

If there is no bootloader in the application, but the *Boot support* is selected, a valid checksum at address 0x7f4 (or 0x7f8) can exist for the boot. Since there is no bootloader used, the checksum at 0x7f4 (or 0x7f8) cannot be cleared and can cause a lockup.

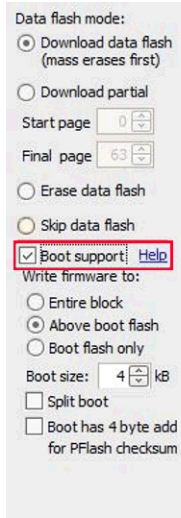


Figure 2-1. Boot Support Selection

2.3 zero_out_integrity_word Function Fails

zero_out_integrity_word function is used to clear the specified location of checksum. A lockup can occur when the zero_out_integrity_word function does not compile in 32-bit mode. This function is normally in an independent file named zero_out_integrity_word.c. The function name or file name can be slightly different from project to project, but generally looks like the following.

```
#define program_flash_integrity_word (*((volatile unsigned long *) 0x7ffc))
//last word in flash, when executing from Flash.  used to store integrity code

void zero_out_integrity_word(void)
{
    DecRegs.FLASHILOCK.all = 0x42DC157E;// Write key to Program Flash Interlock Register
    DecRegs.MFBALR1.all = MFBALRX_BYTE0_BLOCK_SIZE_32K; //enable program flash write
program_flash_integrity_word = 0;
   DecRegs.MFBALR1.all = MFBALRX_BYTE0_BLOCK_SIZE_32K + MFBALRX_BYTE0_RONLY;

    while(DecRegs.PFLASHCTRL.bit.BUSY != 0)
    {
        ; //do nothing while it programs
}

    return;
}
```

To check whether this function/file is compiled in 32-bit mode:

- Right click on zero_out_integrity_word.c > *Properties* > *CCS Build* > *ARM Compiler* > *Processor Option* > *Designate code state*.
- If currently 16, change to 32 (as shown in Figure 2-2.)
- Rebuild the project.

Note that this option is done with the configuration in CCS, so the zero_out_integrity_word function can possibly fail when the CCS version or compiler version is changed.

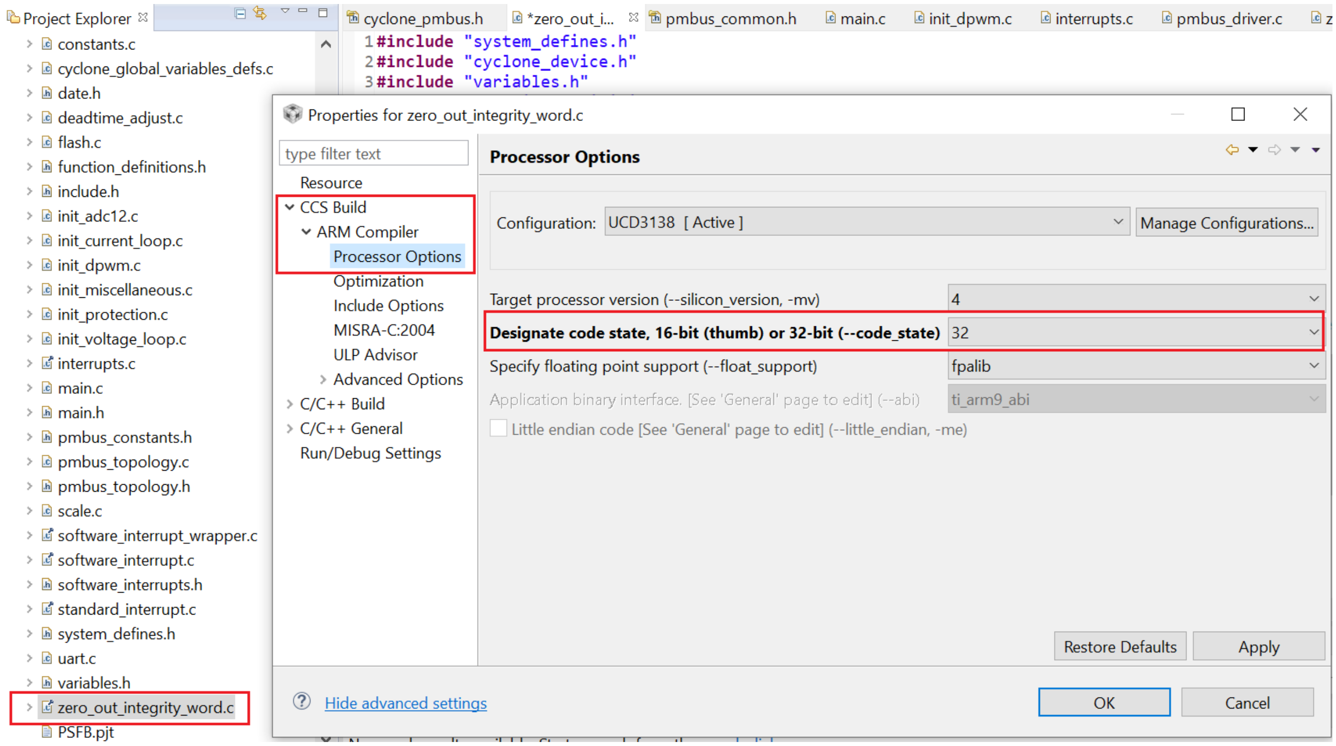


Figure 2-2. Designating Code State

Another option is to use a preprocessor directive #pragma telling the compiler to compile the specified function in 32-bit mode. The following is an example.

```
#define program_flash_integrity_word (*((volatile unsigned long *) 0x7ffc))
//last word in flash, when executing from Flash.  used to store integrity code

#pragma CODE_STATE(zero_out_integrity_word, 32) // 16 = thumb mode, 32 = ARM mode
void zero_out_integrity_word(void)
{
    DecRegs.FLASHILOCK.all = 0x42DC157E;// Write key to Program Flash Interlock Register
    DecRegs.MFBALR1.all = MFBALRX_BYTE0_BLOCK_SIZE_32K; //enable program flash write
    program_flash_integrity_word = 0;
   DecRegs.MFBALR1.all = MFBALRX_BYTE0_BLOCK_SIZE_32K + MFBALRX_BYTE0_RONLY;
    while(DecRegs.PFLASHCTRL.bit.BUSY != 0)
    {
        ; //do nothing while it programs
}

    return;
}
```

2.4 PMBus Communication Fails

A lockup can occur when the zero_out_integrity_word function works, but PMBus communication fails because the zero_out_integrity_word normally gets triggered by a PMBus command.

To debug, place a back-door at the beginning of the application code as shown in the following. This is to make sure the device is able to recover when the back-door condition is met.

```
void main()
{
    volatile unsigned int dummy;
    if(GioRegs.FAULTIN.bit.FLT3_IN == 0)// Re-Check pin assignment
    {
        clear_integrity_word();
}
}
```

2.5 Unexpected Occurrences

A lockup can occur if an unexpected occurrence happens during programming with old GUI (for example, the power shuts off or loose wires). This is especially true in the case that boot checksum is used. This is because old GUI programs go from low address to high address sequentially. If this occurs, use the latest GUI tool Fusion Digital Power Studio. This tool programs the checksums in the end, allowing the device the chance to recover on reset.

3 How to Avoid a Lockup

To avoid a lockup, follow the three steps:

Step 1: DO NOT write checksum when programming a UCD device if there is a possibility the firmware is not working. While checksum can still jump to pflash for execution after programming, a UCD device can always get back to ROM on reset. Follow the configurations in Figure 3-1.

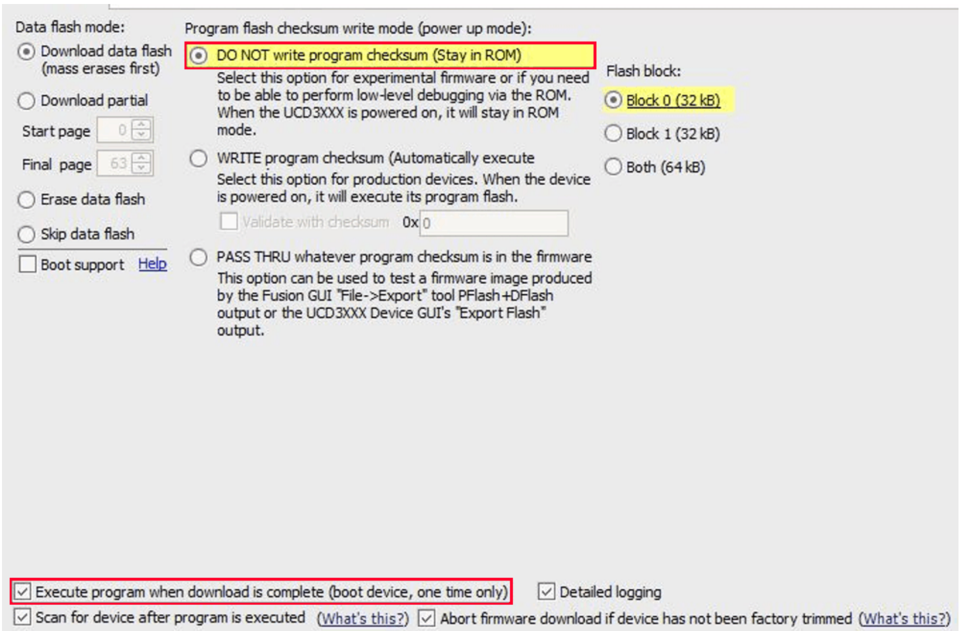


Figure 3-1. Programming Without Checksum

Step 2: Confirm the expected location of checksum is cleared correctly. Click on the command *Command Program to jump to ROM (Send Byte 0xD9 to address xx)* to send UCD back to ROM mode.

Figure 3-2. Jumping to ROM Mode

Step 3: Go to the *Checksums* tag. Select the *Dump* button to read each of the checksums by configuring the *Block configuration*. Check whether the expected location of checksum is cleared. If cleared correctly, the field shows all zeros.

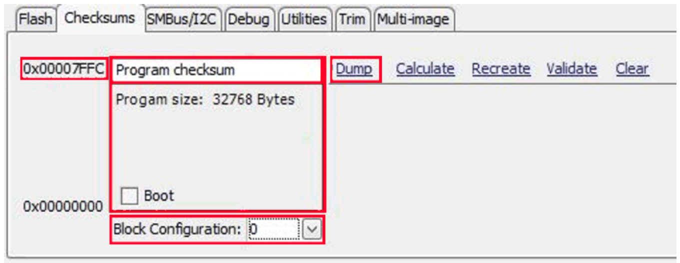


Figure 3-3. Verification on Checksum Value

Check code to confirm the correct checksum is cleared. Once cleared as expected, program UCD device with checksum.

4 Unlock with JTAG

If the device cannot be reprogrammed, but the firmware is executing normally and the memory debugger read or write functionality works, there is a way to unlock with JTAG.

4.1 Enable JTAG Functionality

JTAG port mainly includes 4 pins TCK/TDI/TDO/TMS. Those 4 pins can work in GPIO mode if the JTAG function is disabled, which is the normal case in application. To enable JTAG functionality, check the configuration of registers in the following, and reconfigure them if necessary. Set the IOMUX register to be 0 to enable JTAG, and make sure none of TCK/TDI/TDO/TMS pins works in GPIO mode. This can be done via the memory debugger in UCD3xxx Device GUI.

```
        MiscAnalogRegs.IOMUX.all = 0; //enable JTAG

                 MiscAnalogRegs.GLBIOEN.bit.TCK_IO_EN = 0;
                 MiscAnalogRegs.GLBIOEN.bit.TDI_IO_EN = 0;
                 MiscAnalogRegs.GLBIOEN.bit.TDO_IO_EN = 0;
                 MiscAnalogRegs.GLBIOEN.bit.TMS_IO_EN = 0;
```

4.2 New Target Configuration in CCS

On the menu on the top of CCS, click *View* > *Target Configurations*. In the target configurations pane, right click on User Defined, select *New target configuration*. Give the configuration a meaningful name, (for example, emulator type) and the UCD variant in the name, as shown in Figure 4-1. Click *Finish*.

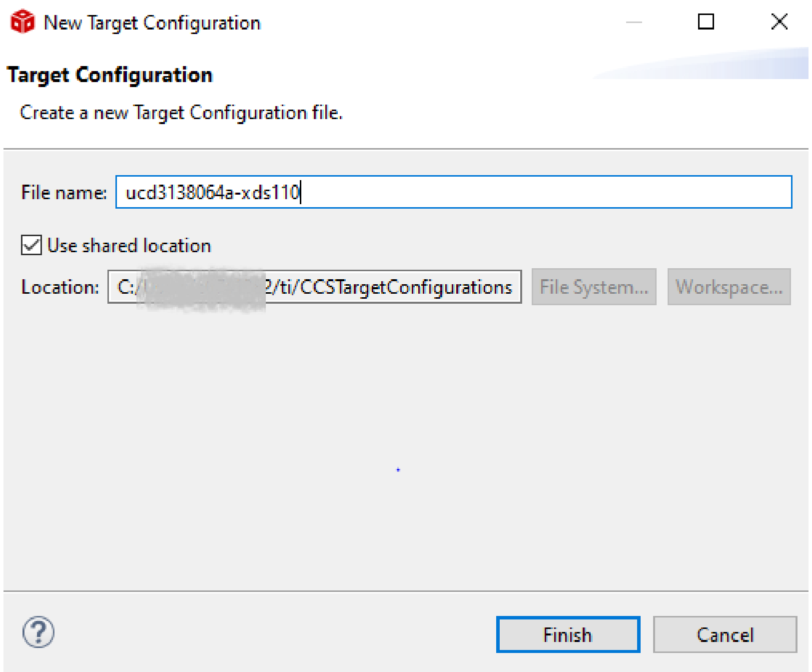


Figure 4-1. New Target Configuration

In the *Connection* and *Board or Device* item, select the proper emulator type and UCD variant respectively. Take the JTAG XDS110 and UCD3138064A as an example.

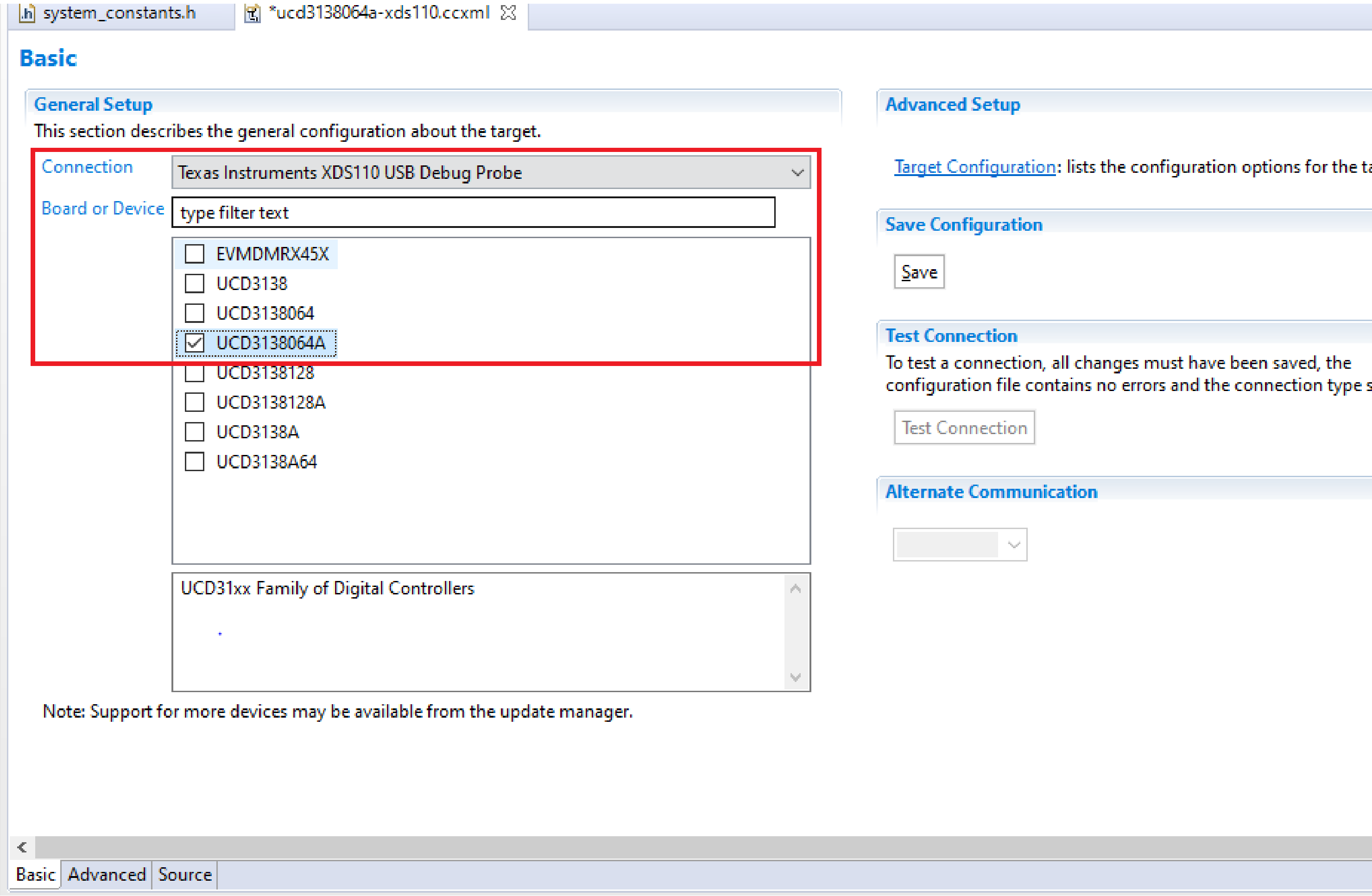


Figure 4-2. Emulator Type and UCD Variant Selection

From the Advanced tab, change the TCLK frequency from 5.5 MHz to 1.0 MHz. Experiment with values larger than 1 MHz, but less than 5.5 MHz. Click Save.

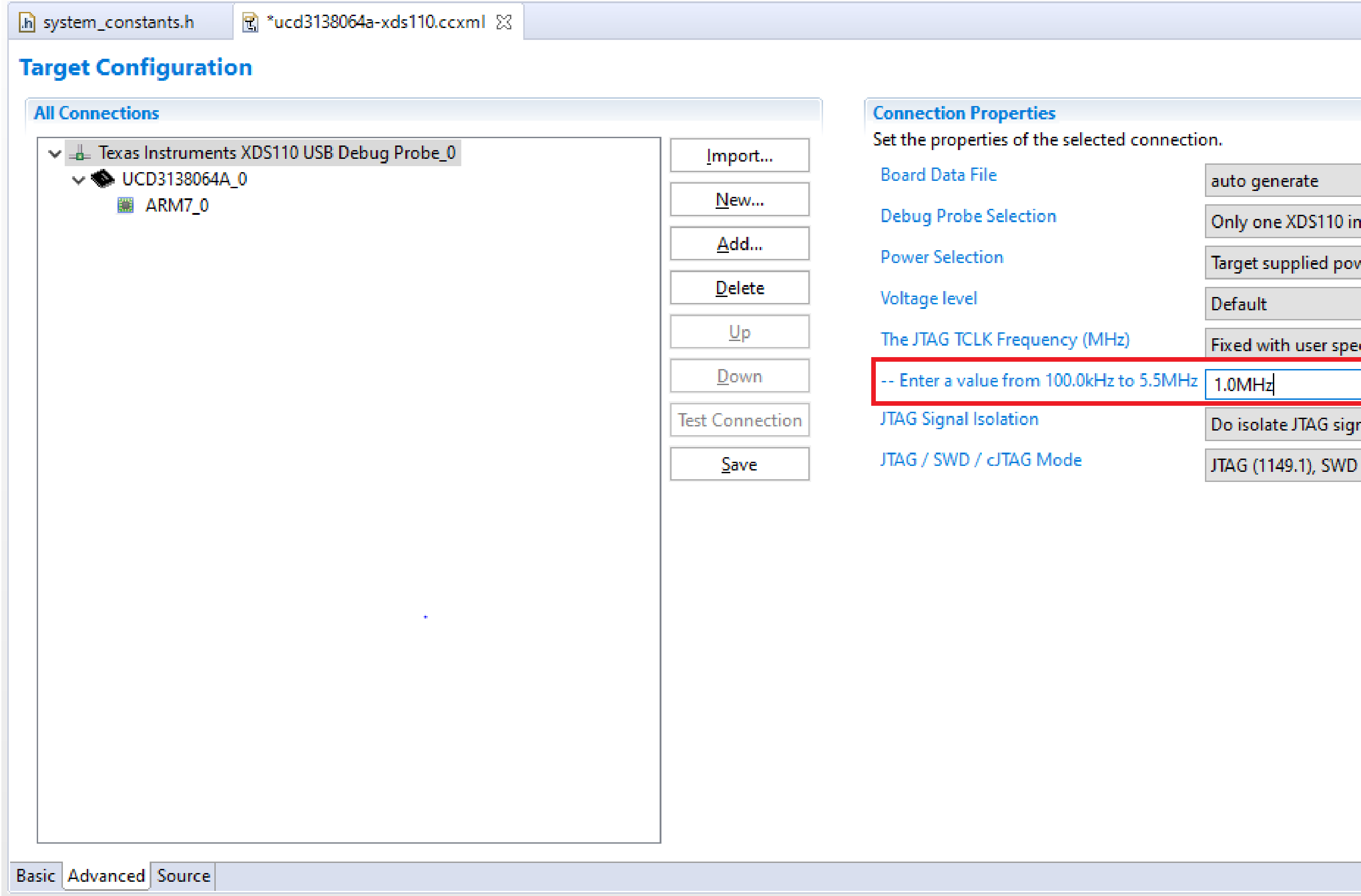
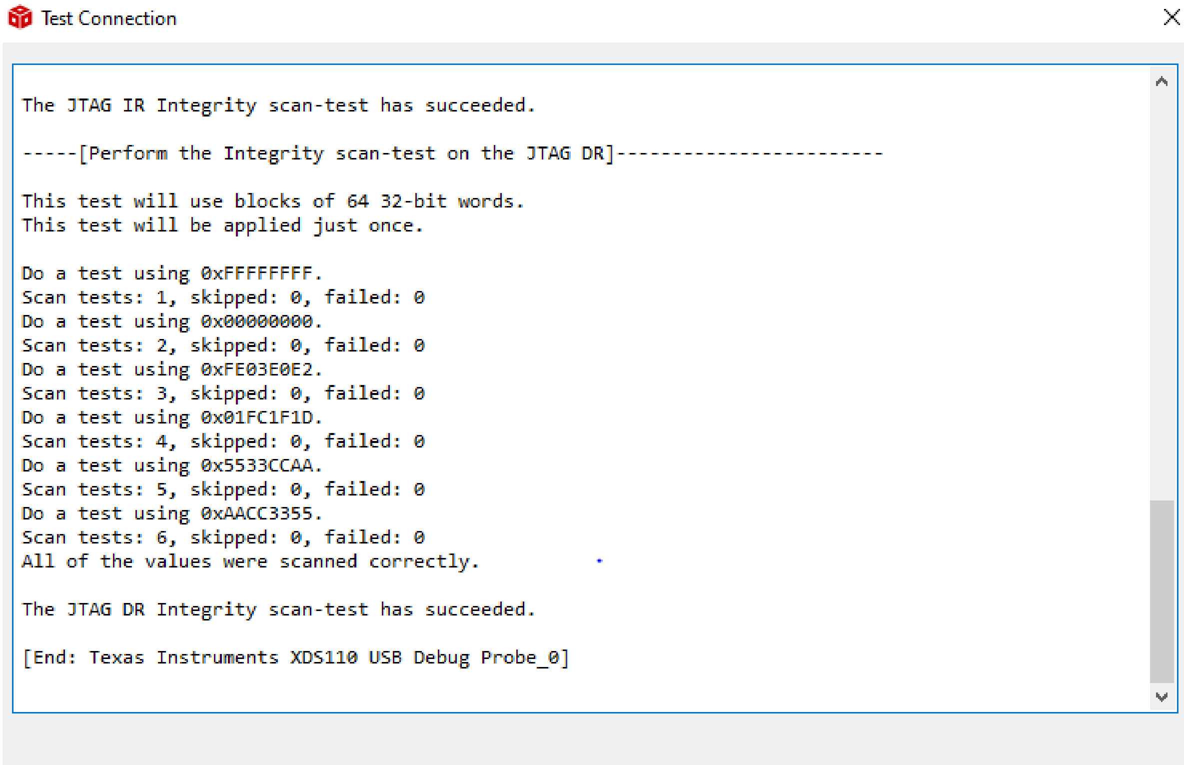


Figure 4-3. TCLK Frequency Configuration

Click *Test Connection*. When the test is complete, scroll down to the bottom of the pop-up window and observe the message as shown in Figure 4-4. If the message does not appear, there is an issue with either the physical connection or the setup.



Test Connection

```
The JTAG IR Integrity scan-test has succeeded.

-----[Perform the Integrity scan-test on the JTAG DR]------------------------

This test will use blocks of 64 32-bit words.
This test will be applied just once.

Do a test using 0xFFFFFFFF.
Scan tests: 1, skipped: 0, failed: 0
Do a test using 0x00000000.
Scan tests: 2, skipped: 0, failed: 0
Do a test using 0xFE03E0E2.
Scan tests: 3, skipped: 0, failed: 0
Do a test using 0x01FC1F1D.
Scan tests: 4, skipped: 0, failed: 0
Do a test using 0x5533CCAA.
Scan tests: 5, skipped: 0, failed: 0
Do a test using 0xAACC3355.
Scan tests: 6, skipped: 0, failed: 0
All of the values were scanned correctly.

The JTAG DR Integrity scan-test has succeeded.

[End: Texas Instruments XDS110 USB Debug Probe_0]
```

Figure 4-4. Success on JTAG Connection

4.3 Clear the Flash

Right click on the ucd3138064a-xds110.ccxml, select *Launch Selected Configuration*. The debug window appears as in Figure 4-5. On the top menu, click *Run* > *Connect Target*.

Use the following steps:

- Go to the *Registers* tab, clear the RONLY bit in MFBALR1 register.
- Set the value of the FLASHILOCK register as 0x42DC157E.
- Set the MASS_ERASE bit in PFLASHCTRL1. This erases the block 1 in UCD3138064A.
- Exit from debug mode and reset the device. The device is now in ROM mode.

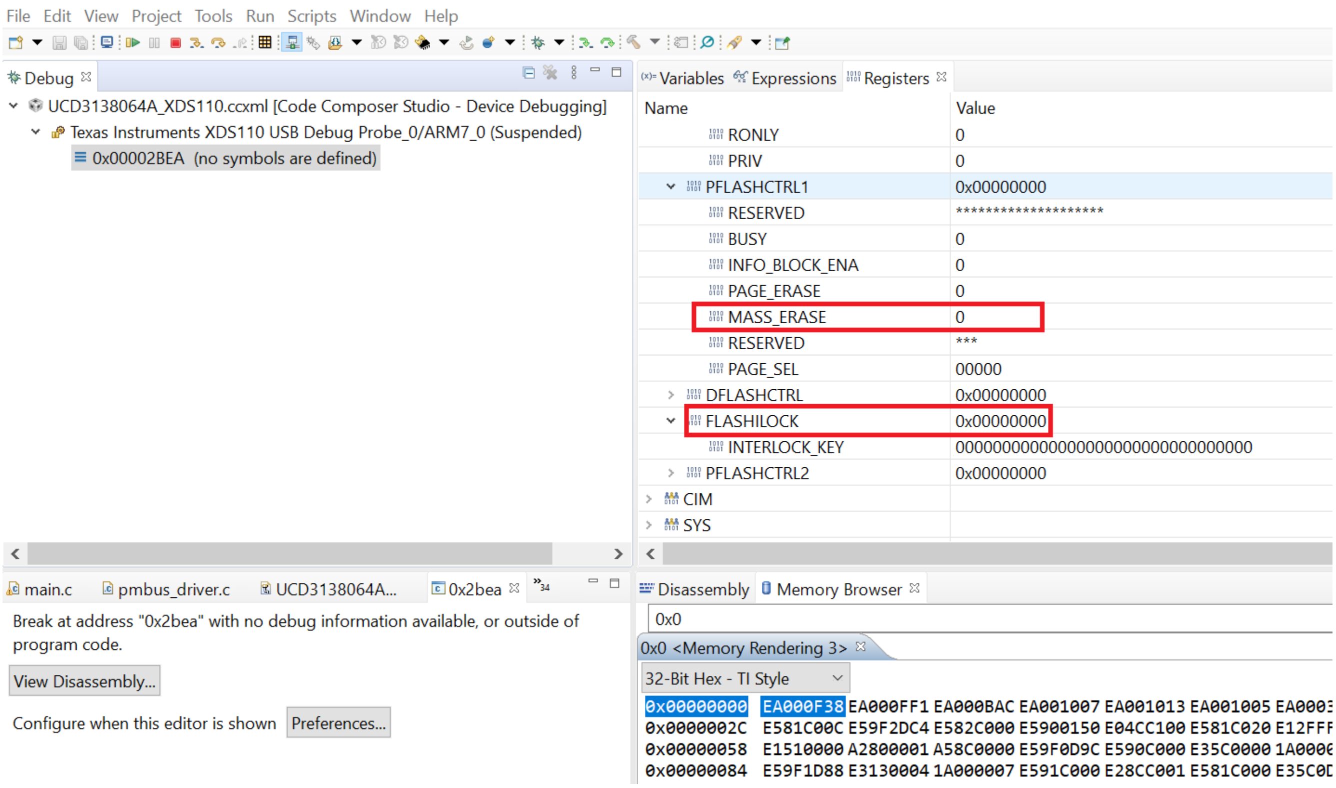


Figure 4-5. Debug Mode

5 Summary

A lockup can occur when there is a valid checksum that cannot be cleared, which can be caused by either an incorrect download process or firmware with bugs. To avoid a lockup, check the possible reasons listed and follow the instructions before programming with checksum. If the device is locked, but the firmware is executing normally and PMBus works, the device is able to recover with JTAG. Otherwise, there is no way to save.

6 References

- Texas Instruments, *Fusion Digital Power Studio*.
- Texas Instruments, *UCD3138064 Programmer's Manual*.
- Texas Instruments, *UCD3138A64/UCD3138128 Programmer's Manual*.