# 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**

1 Why Lockup Can Occur	2
2 Reasons for Lockup	4
2.1 Wrong Code in the load.asm	4
2.2 Misoperation with TI GUI	4
2.3 zero out integrity word Function Fails	5
2.4 PMBus Communication Fails	6
2.5 Unexpected Occurrences	7
3 How to Avoid a Lockup	7
4 Unlock with JTAG	8
4.1 Enable JTAG Functionality	8
4.2 New Target Configuration in CCS	8
4.3 Clear the Flash	
5 Summary	
6 References	

#### List of Figures

Figure 1-1. Checksum Verification Flow Chart in UCD3138(A) ROM	2
-igure 1-2. Checksum Verification Flow Chart in UCD3138064(A) ROM	3
-igure 1-3. Checksum Verification Flow Chart in UCD3138128(A) ROM	4
-igure 2-1. Boot Support Selection	5
Figure 2-2. Designating Code State	6
Figure 3-1. Programming Without Checksum	7
Figure 3-2. Jumping to ROM Mode	7
-igure 3-3. Verification on Checksum Value	8
-igure 4-1. New Target Configuration	9
Figure 4-2. Emulator Type and UCD Variant Selection	10
Figure 4-3. TCLK Frequency Configuration	11
Figure 4-4. Success on JTAG Connection	12
-igure 4-5. Debug Mode	13

#### Trademarks

All trademarks are the property of their respective owners.

1



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

#### 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.



#### Reasons for Lockup

Project Explorer ≅ □ \$ > i constants.c > i cyclone_global_variables_defs.c	Cyclone_pmbus. Cyclone_pmbus. 1#include " 2#include " 3#include "	h	it_dpwm.c 🛛 🖻 interrupts.c	le pmbus_driver.c
<ul> <li>&gt; M date.h</li> <li>&gt; deadtime_adjust.c</li> <li>&gt; flash.c</li> <li>&gt; function_definitions.h</li> <li>&gt; include.h</li> <li>&gt; init_adc12.c</li> <li>&gt; init_current_loop.c</li> </ul>	<ul> <li>Properties for zero_out_</li> <li>type filter text</li> <li>Resource</li> <li>CCS Build</li> <li>ARM Compiler</li> <li>Processor Options</li> </ul>	Processor Options Configuration: UCD3138 [Active]	~	→ → × ↔ • ↔ • •
<ul> <li>Init_dpwm.c</li> <li>Init_miscellaneous.c</li> <li>Init_protection.c</li> <li>Init_protection.c</li> <li>Init_voltage_loop.c</li> <li>Advanced Options</li> <li>C/C++ Build</li> <li>C/C++ General Run/Debug Settings</li> <li>Init_voltage_interrupt_wrapper.c</li> </ul>	Target processor version (silicon_version, -mv) Designate code state, 16-bit (thumb) or 32-bit (code_state) Specify floating point support (float_support) Application binary interface. [See 'General' page to edit] (abi) Little endian code [See 'General' page to edit] (little_endian, -n	4 32 fpalib ti_arm9_abi me)	× × ×	
<ul> <li>if software_interrupt.c</li> <li>is software_interrupts.h</li> <li>if standard_interrupt.c</li> <li>is system_defines.h</li> <li>if uart.c</li> <li>in variables.h</li> <li>if zero_out_integrity_word.c</li> <li>in PSFB.pjt</li> </ul>	② Hide advanced setting	<u>15</u>	Restore De OK	faults Apply Cancel

#### 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.

Data flash mode:	Program flash checksum write mode (power up mode):
Download data flash	DO NOT write program checksum (Stay in ROM)
(mass erases first) O Download partial Start page	Select this option for experimental firmware or if you need to be able to perform low-level debugging via the ROM. When the UCD3XXX is powered on, it will stay in ROM mode.       Flash block:
Final page 63 💭 O Erase data flash O Skip data flash	WRITE program checksum (Automatically execute Select this option for production devices. When the device is powered on, it will execute its program flash.     Both (64 kB)       Validate with checksum     0x 0
Boot support Help	PASS THRU whatever program checksum is in the firmware This option can be used to test a firmware image produced by the Fusion GUI "File->Export" tool PFlash +OFlash output or the UCD3XXX Device GUI's "Export Flash" output.
Execute program when	n download is complete (boot device, one time only)

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.

Scan for Device in Program Mode: DEVICE	DEVICE CODE PMBUS RE	VISION
Command ROM to execute its program (Sen	Byte 0xF0 to Address 11)	lock 0 🚫 Block 1

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.

7



0x00007FFC	Program checksum	Dump	Calculate	Recreate	<u>Validate</u>	Clear
	Progam size: 32768 Bytes					
0_00000000	Boot					

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*.

WWW.ti.com	S RUMENTS					
🎁 New Targe	t Configuratior	1		_		×
Target Confi	guration					
Create a new	Target Configu	ration file.				
File name:	ucd3138064a-xo	ds110				
Location: C	:/	2/ti/CCSTargetConfiguratio	ns File Syste	m V	/orkspace	e
?			Finish		Cancel	

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.

Unlock with JTAG

system_constan	ts.h 😰 *ucd3138064a-xds110.ccxml 🛛	
<b>asic</b> General Setup This section descr	ribes the general configuration about the target.	Advanced Setup
Connection	Texas Instruments XDS110 USB Debug Probe 🗸 🗸	Target Configuration: lists the configuration options for the
Board or Device	type filter text         EVMDMRX45X         UCD3138         UCD3138064         UCD3138064A         UCD3138128         UCD3138128A         UCD3138A         UCD3138A	Save Configuration Save Save Test Connection To test a connection, all changes must have been saved, th configuration file contains no errors and the connection ty Test Connection Alternate Communication
Note: Support fo	UCD31xx Family of Digital Controllers	

#### 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.



All Connections		Connection Properties	
L Texas Instruments YDS110 LISE Debug Probe 0	Immed	Set the properties of the selected connection	on.
✓ ▲ IEXas institutients xb3110 050 Debug Probe_0 ✓ ◆ UCD3138064A_0	<u>I</u> mport	Board Data File	auto generate
ARM7_0	<u>N</u> ew	Debug Probe Selection	Only one XDS110 ir
	<u>A</u> dd	Power Selection	Target supplied pov
	<u>D</u> elete	Voltage level	Default
	Up	The JTAG TCLK Frequency (MHz)	Fixed with user spe
	Down	Enter a value from 100.0kHz to 5.5MHz	1.0MHz
	Test Connection	JTAG Signal Isolation	Do isolate JTAG sigr
	Save	JTAG / SWD / cJTAG Mode	JTAG (1149.1), SWD

#### 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



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

The JTAG DR Integrity scan-test has succeeded.

All of the values were scanned correctly.

#### 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.

Texas

NSTRUMENTS

www.ti.com

V INSTRUMENTS

**FEXAS** 

File Edit View Project Tools Run Scripts Window Help			
Image: Comparison of the state of the s	≪ ▼ I ≪ I Ø I Ø ▼ I ■		
<ul> <li>V UCD3138064A XDS110.ccxml [Code Composer Studio - Device Debuaging]</li> </ul>	Name	Value	
Probe 0/ARM7 0 (Suspended)		0	
■ 0x00002BEA (no symbols are defined)		0	
	✓ IIII PELASHCTRI 1	0x0000000	
	## RESERVED	****	
	BUSY	0	
	INFO BLOCK ENA	0	
	Bill PAGE ERASE	0	
	MASS_ERASE	0	
	HIN RESERVED	***	
	PAGE_SEL	00000	
	> DFLASHCTRL	0x0000000	
	✓ <sup>III</sup> FLASHILOCK	0x0000000	
	INTERLOCK_KEY	000000000000000000000000000000000000000	
	> IN PFLASHCTRL2	0x0000000	
	> ## CIM		
	> ## SYS		
< >>	<		
le main.c le pmbus_driver.c	■ Disassembly <sup>1</sup> Memory Browser <sup>13</sup>		
Break at address "0x2bea" with no debug information available, or outside of	0x0		
program code.	0x0 <memory 3="" rendering=""> 🛱</memory>		
View Disassembly	32-Bit Hex - TI Style		
view Disdsserioly		1000BAC FA001007 FA001013 FA001005 FA0003	
Configure when this editor is shown Preferences	0x0000002C E581C00C E59F2DC4 E5	582C000 E5900150 E04CC100 E581C020 E12FFF	
	0x00000058 E1510000 A2800001 A	58C0000 E59F0D9C E590C000 E35C0000 1A0000	
	0X00000084 E59F1D88E313000414	4000001 E23TC000 E58CC001 E281C000 E32C0F	

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.

#### IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to TI's Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265 Copyright © 2023, Texas Instruments Incorporated