UCD3138064 – Using D1D4 and PMBus for Program Upgrades in the Field

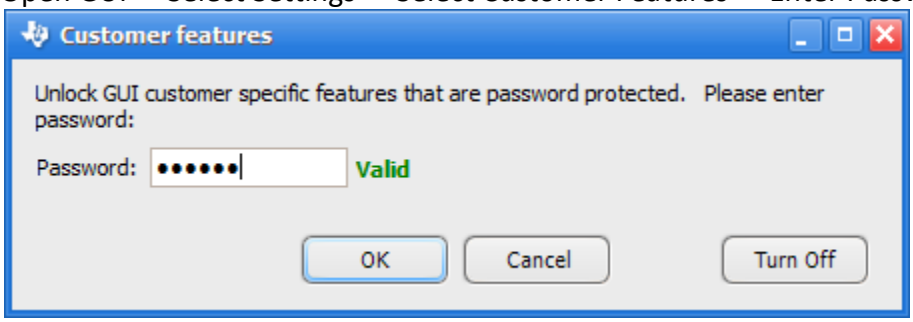
When the PSU firmware needs to be update the system will send the new firmware image through the PMBus. Describes the mechanism implemented in power management firmware which allows the PSUs to be re-programmed. This document use TI Fusion Digital Power Designer v1.9.45 version.

In this document, we will use below two programs.

- d1d4 boot flash 001
- example main with d1 command 001

1) Enable specific mode for ISP:

Open GUI -> Select Settings -> Select Customer Features -> Enter Password: **12d1d4**



2) **Download the BootFlash code**

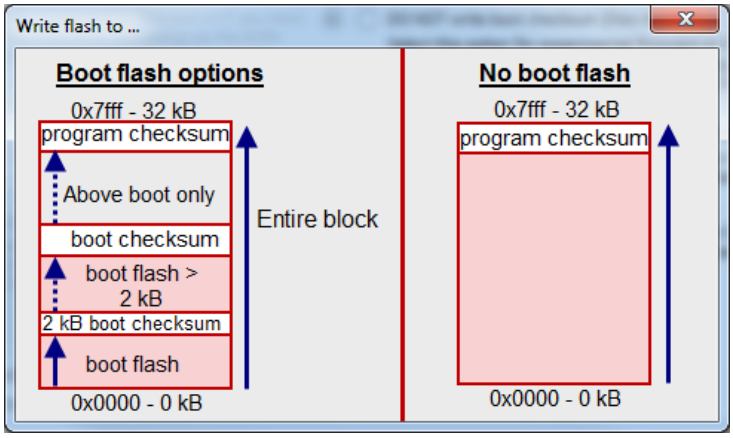
Must to select “Boot support” and Boot Flash Only, depends on Boot size.

Boot support:

Entire Block ➔ 0x0000~0Xffff (64kb)

Above Boot Flash ➔PFlash checksum excludes/ includes boot.

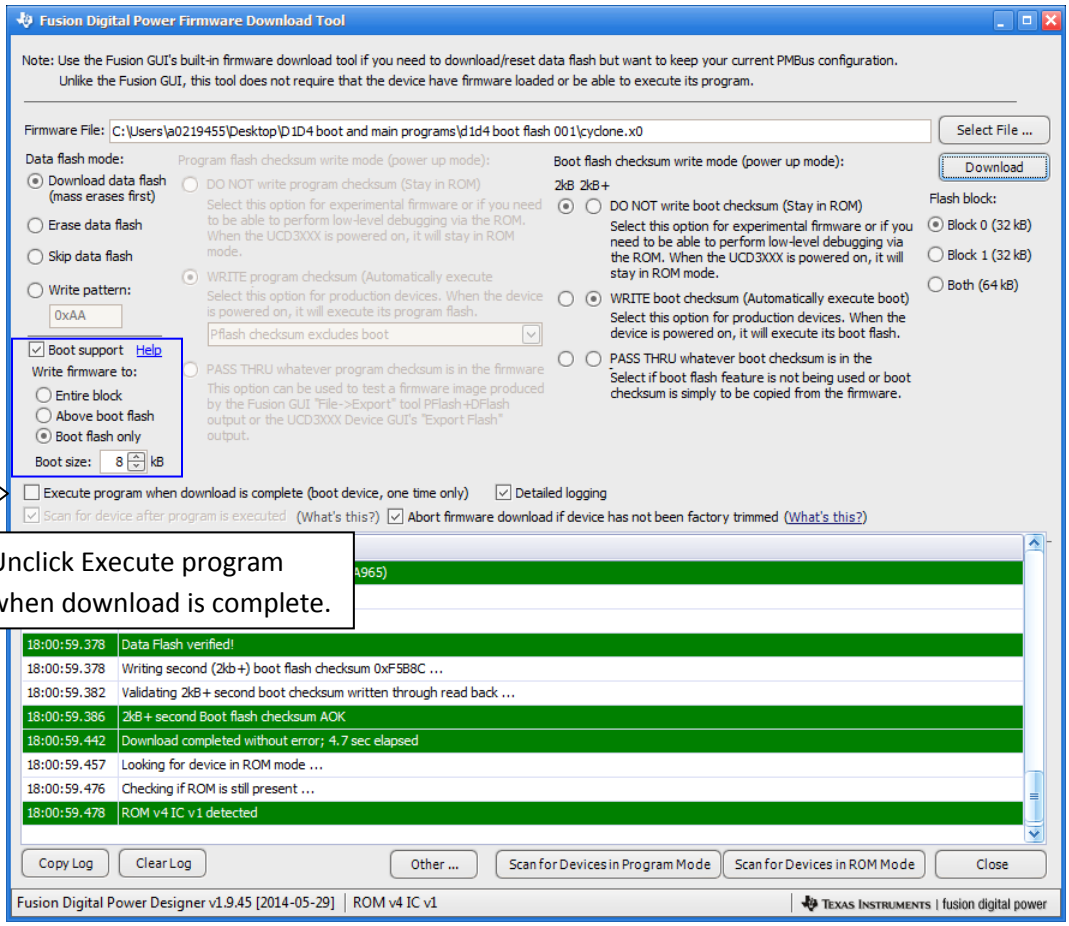
Boot flash only ➔depends on boot size. with 8KB boot size in this sample code.



Boot Flash checksum write mode:

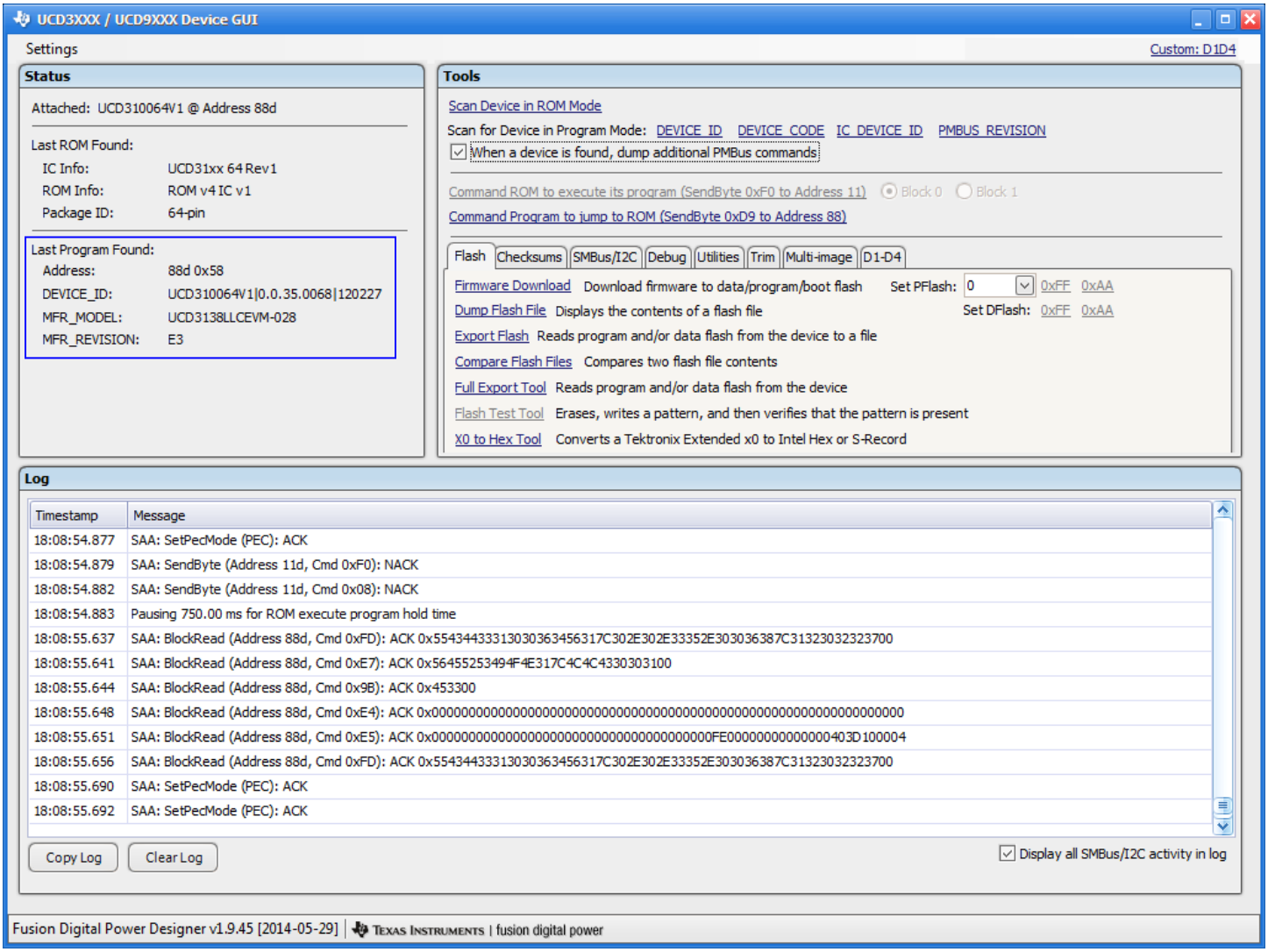
2KB ➔ Do not write boot checksum

2KB+ ➔Write boot checksum.



Unclick Execute program when download is complete.

3) Click <u>DEVICE ID</u> and we can read the program information.



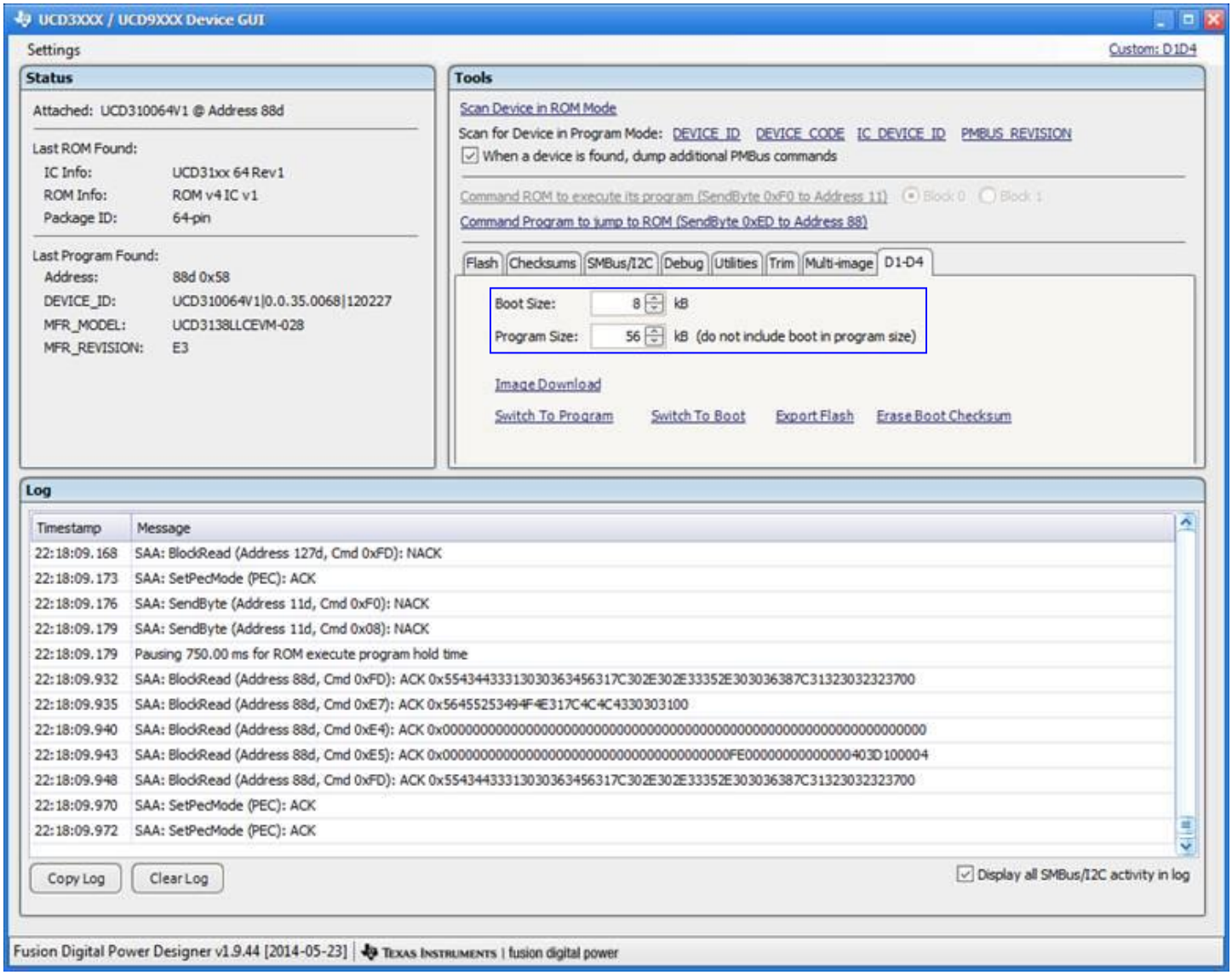
4) Click D1-D4 Tab:

Enter Boot Size and Program Size (do not include boot in program size).

Image Download ➔Download program image

Switch to Program ➔Execute its program

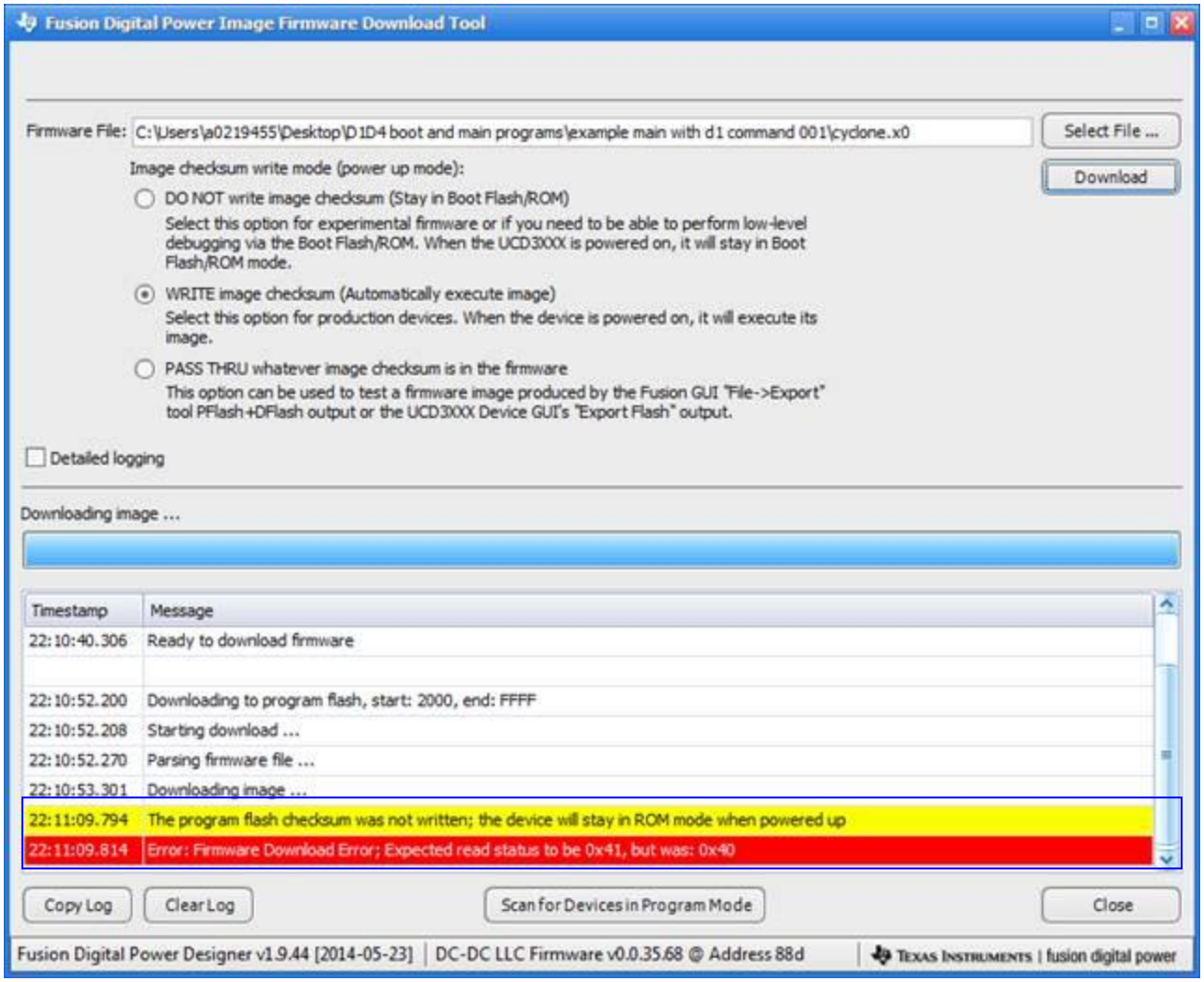
Switch To Boot ➔ Back to boot flash program



5) Image download (for older version): Select main program and click download:

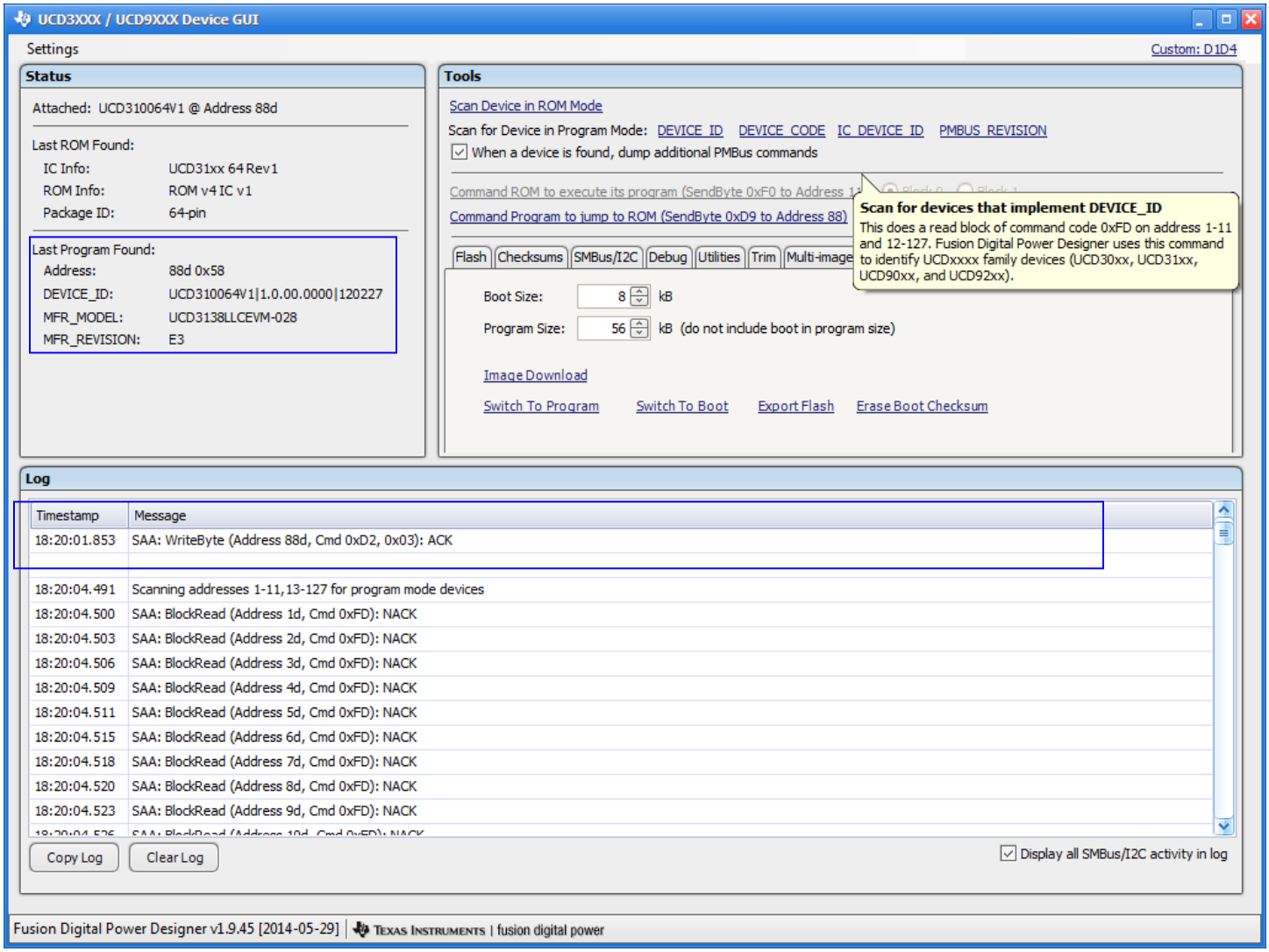
Select WRITE image checksum (Automatically execute image).

If select “Detailed logging”, and you can see “Downloading image…”



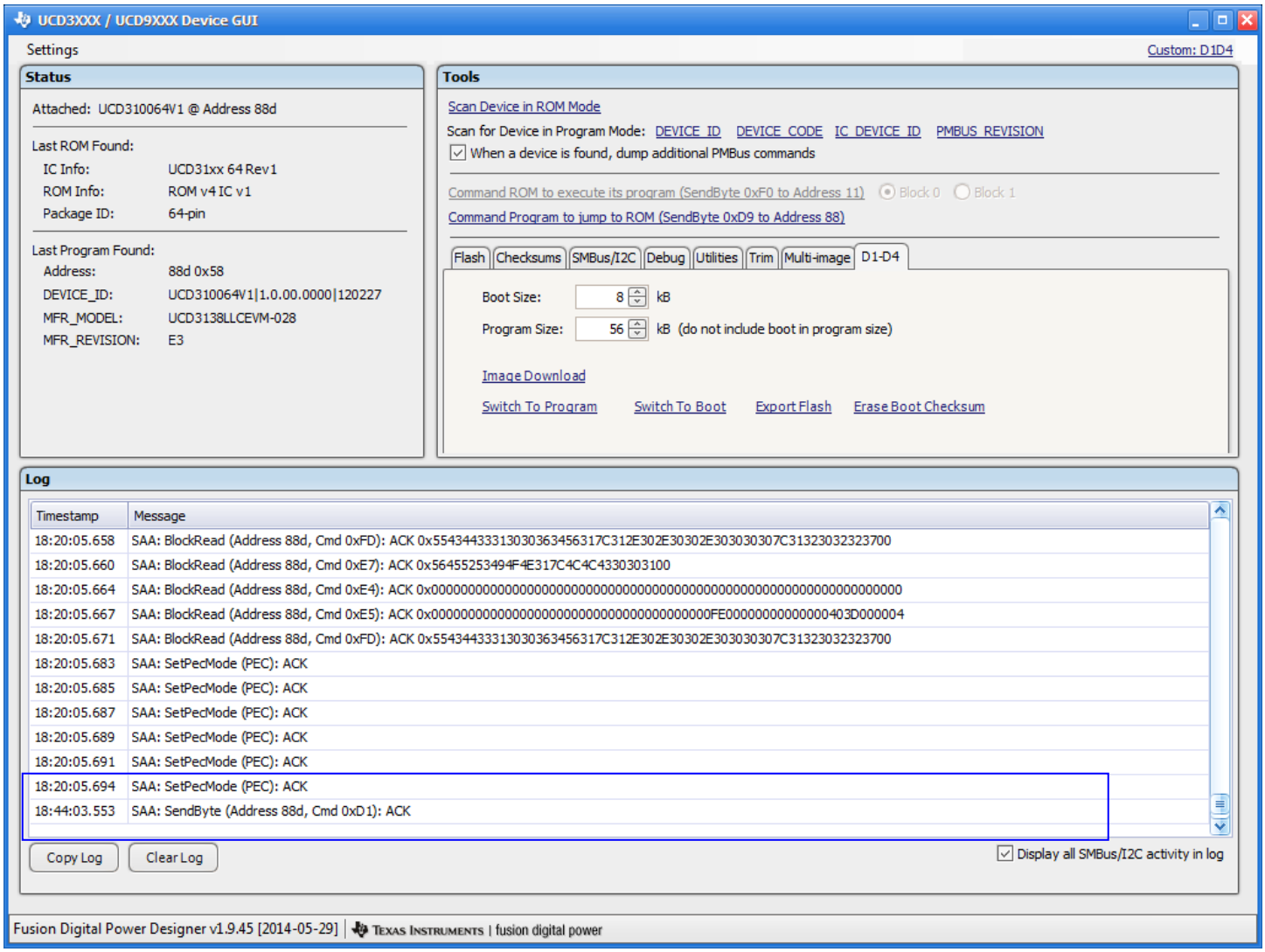
6) "<u>Switch to Program</u>":

GUI will WriteByte with 0xD2, and Bootloader status is 0x40, and then enter Bootloader, if download is complete and the checksum is valid, GUI will send back 0x41. Then GUI can read the new image information.

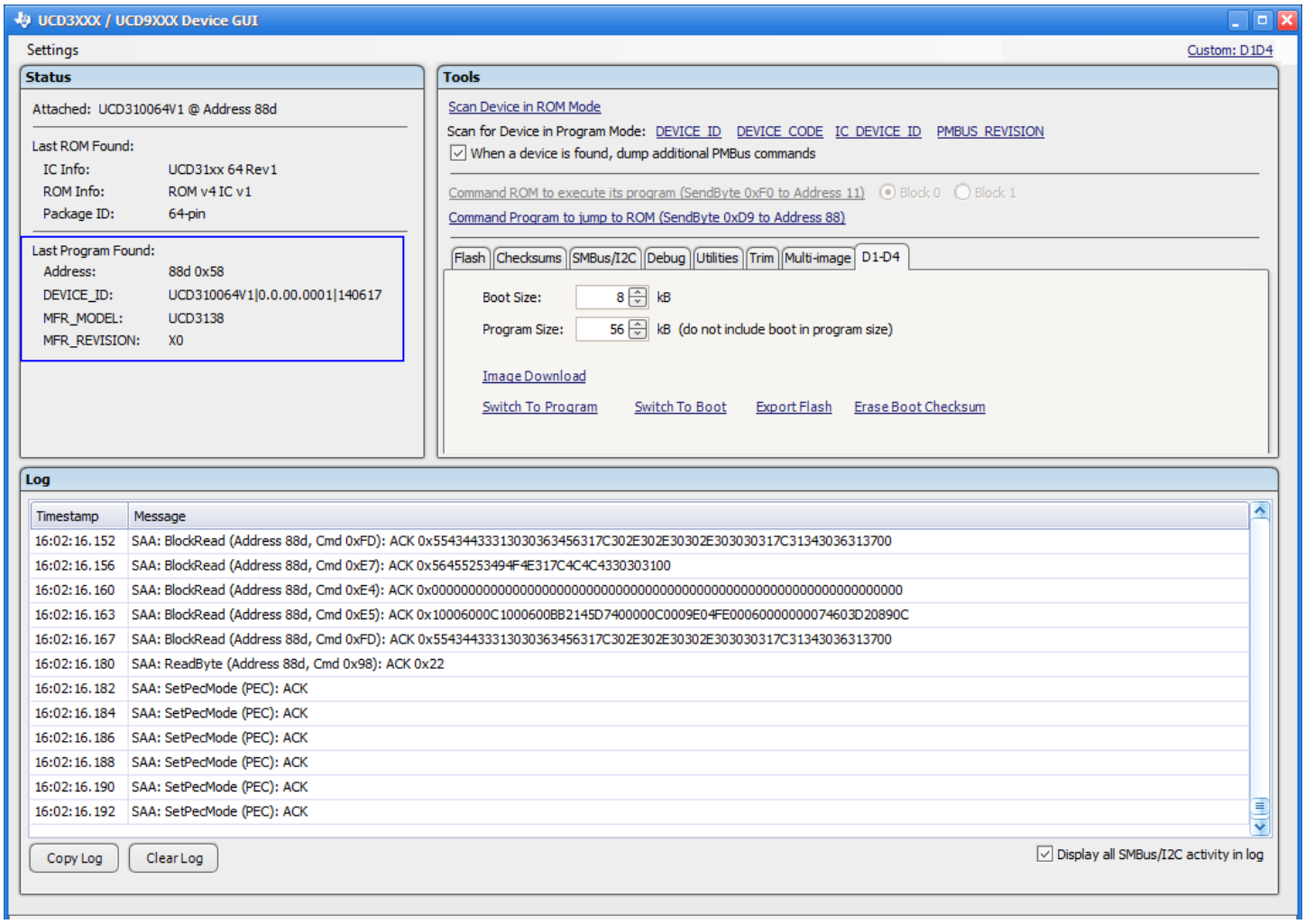


7) "Switch To Boot":

If user wants to upgrade your firmware, please click "Switch To Boot" then GUI will send 0xD1. GUI will send out A 32-bit Key is used to access the Bootloader.



8) Image Download (for upgrade version):



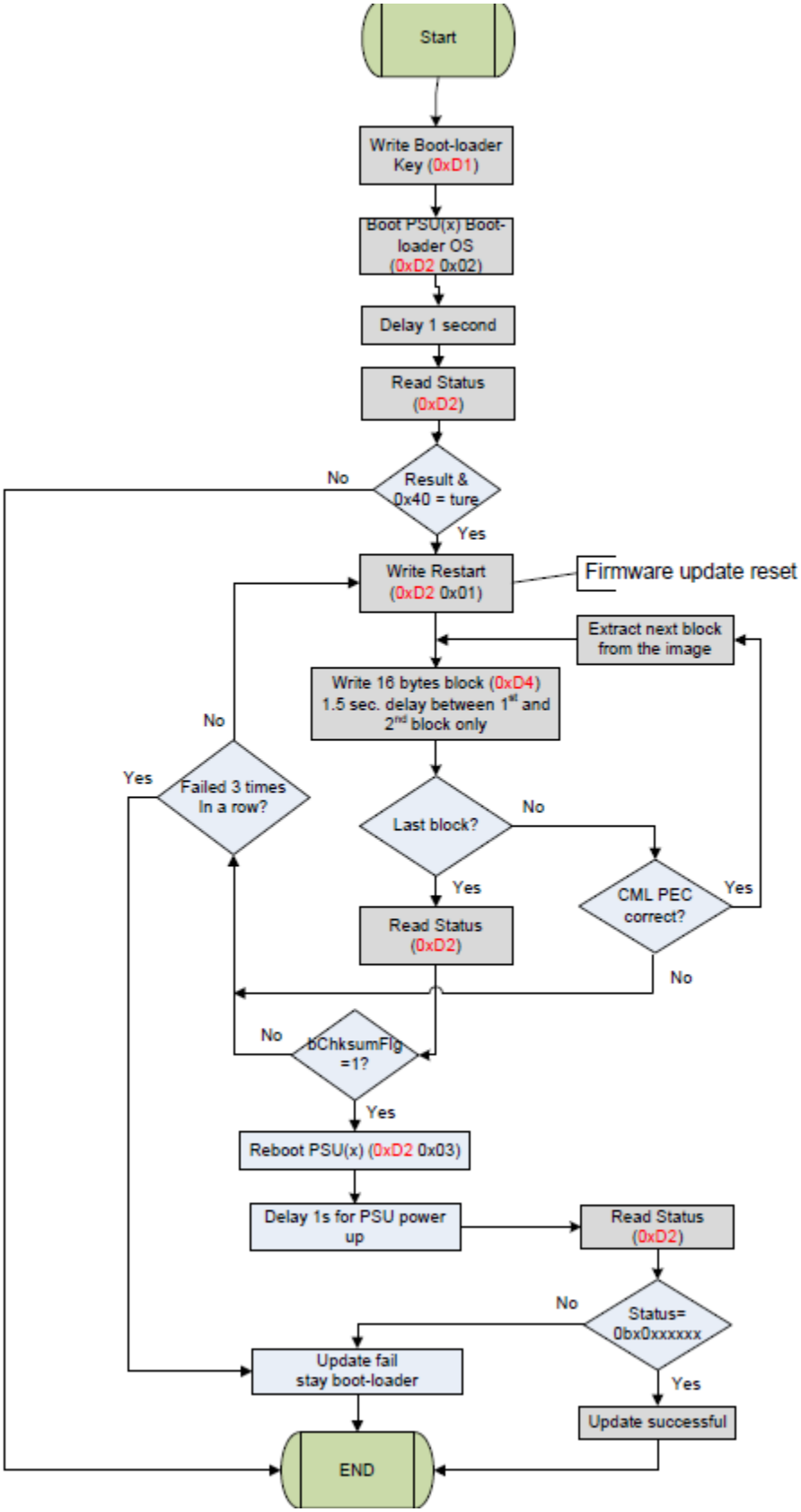
9) Update complete:

Please see below is related commands for firmware update (ISP, In System Programming).

Firmware Update Related Commands

Command Code	PM-Bus Cmd. Name	Data Length	Command Name	Description
0xD1	MFR_SPEC_01	4	Bootloader Key	A 32-bit key is used to access the Bootloader. This mechanism is provided to reduce the probability of accidental entry with the final result of stopping the power supply to the system.
0xD2	MFR_SPEC_02	1	Bootloader Status/Cmd	This command reflects status information during programming. When written this is a command register to the Bootloader.
0xD4	MFR_SPEC_04	16	Bootloader Memory Block	When written, 16 bytes of data are queued to write to the specified memory address. When read, 16 bytes are read starting from the specified memory location. Note: this command does not have the "data count" byte that is part of a block write command (i.e. the data length is 16 bytes)

Firmware Flowchart



For main program

Interrupts.c

boot_entry()

In standard interrupt, add the case and then Jump to boot flash.

case 12: // clear integrity words, depending on arg1

```
{
        register Uint32 * program_index = (Uint32 *) program_area; //store destination address for program
        register Uint32 * source_index; //Used for source address of PFLASH;
        register Uint32 counter;

        if(arg1 == 1) //1 means first word;
        {
                if(DecRegs.MFBALR1.bit.ADDRESS == 0) //here if flash block 1 is at 0
                {
                        DecRegs.FLASHILOCK.all = PROGRAM_FLASH1_INTERLOCK_KEY;
                }
                else //if it's program flash 2;
                {
                        DecRegs.FLASHILOCK.all = PROGRAM_FLASH2_INTERLOCK_KEY;
                }
                source_index = (Uint32 *)((void*)zero_out_integrity_word_1);      // 20140516
        }
        else if(arg1 == 2)//2 means second word;
        {
                if(DecRegs.MFBALR1.bit.ADDRESS == 0) //here if flash block 1 is at 0
                {
                        DecRegs.FLASHILOCK.all = PROGRAM_FLASH2_INTERLOCK_KEY;
                }
                else //if it's program flash 2;
                {
                        DecRegs.FLASHILOCK.all = PROGRAM_FLASH1_INTERLOCK_KEY;
                }
                source_index = (Uint32 *)((void*)zero_out_integrity_word_2);      // 20140516
        }
        else
        {
                return; //reject other arg1 values
        }
        for(counter=0; counter < 32; counter++) //Copy program from PFLASH to RAM
        {
                *(program_index++)=*(source_index++);
        }
        DecRegs.MFBALR1.bit.RONLY = 0; //enable program flash 1 write
        DecRegs.MFBALR17.bit.RONLY = 0; //enable program flash 2 write
        {
                register FUNC_PTR func_ptr;
                func_ptr=(FUNC_PTR)((void*)program_area);     //Set function to program area
```

```
                func_ptr();
        }       //execute erase checksum
        DecRegs.MFBALR1.bit.RONLY = 1; //restore it to read only
        DecRegs.MFBALR17.bit.RONLY = 1; //restore it to read only
        SysRegs.SYSECR.bit.RESET = 2;                   // 20140516
}
```

pmbus.c

int32 d1_command_check(void)

Bootloader Key, this PMBus mechanism is provided to reduce the probability of accidental entry with the final result of stopping the power supply to the system. And Jump to “jump_to_boot()”.

int32 pmbus_write_message(void)

```
        case PMBUS_CMD_MFR_MULTIIMAGE:
                PMBusRegs.PMBACK.byte.BYTE0 = 1;  //ack this because it will never return;
                jump_to_boot();
        case 0xd1: //d1 command to go to boot flash
                return d1_command_check();
```

Only Multiimage command is jump to boot - jump with any multi image command - may want to use password in real system, and d1 command to go to boot flash.

pmbus_commands.h

#define PMBUS_CMD_MFR_MULTIIMAGE 0xEB

Multiimage command definition.

Pmbus_handler.c

Add **PMBST_BYTE0_PEC_VALID** into
void pmbus_write_block_handler(void)
void pmbus_idle_handler()

Before end of message, good data ready, pec valid. Copy 4 buffer into pmbus_buffer and then execute pmbus_write_message().

software_interrupts.h

```
#pragma SWI_ALIAS (jump_to_boot, 50)
void jump_to_boot();
```

Provide the entry point so that program can jump to boot flash.

cyclone_64.cmd

```
BOOT_ENTRY
        .boot_entry_vector: :{} > BOOT_ENTRY
```

Define the boot entry vector in ROM section.

load.asm

```
                .global _boot_entry
                .sect ".boot_entry_vector"
_boot_entry  ;
```

Label for boot entry vector - jump here to enter boot program.

Pmbus_manuf_specific_commands.c

```
void rom_back_door(void)
{
        // Call a SWI to clear the integrity words.
        clear_integrity_word(1);                // 20140623
}
//=============================================================================
// pmbus_write_rom_mode()
//      Erases the program integrity word in FLASH, then waits for watchdog timer to reset the
// CPU.  There is no return code or return from this function.
//=============================================================================
int pmbus_write_rom_mode(void)
{
        rom_back_door();
        return PMBUS_SUCCESS;                   // Note: This line is never reached.
}
```

For Boot Flash:

New Add:

- **Copy_pflahs_erase_to_ram.c**
  Store destination address for program, call “erase_to_pflash_1_kernel” and then copy program from PFLASH to RAM.

- **Copy_pflash_write_to_ram.c**
  Store destination address for program, call “write_to_pflash_1_kernel” and then copy program from PFLASH to RAM.

- **Erase_pflash_1_pages_kernel.c**
  Selects page to be erased during Page Erase Cycle. And then enable program flash page erase.

- **Multiimage.c**
  Define the Multiimage interface.

- **Write_or_erase_pflahs_block_1.c**
  Assume that calling routine will check busy - just write or erase merrily away depends on which function is loaded into flash.

- **Write_to_pflash_1_kernel.c**
  Writes to pflash block 1.

cyclone_64.cmd

```
/*-------------------------------------------------------------------------*/
/* boot p-Flash   8K   0x0  - 0x1FFF                                       */
/*-------------------------------------------------------------------------*/
BFLASH    (RX) : org = 0x00000024, len = 0x000007D8    /* boot flash 2K verfiied by ROM*/
```

```
   EBFLASH    (RX) : org = 0x00000800, len = 0x00001738    /* extended boot flash       */

   /*-------------------------------------------------------------------------- */
   /* P-Flash   vectors for main                                                 */
   /*-------------------------------------------------------------------------- */
   PVECS1       : org = 0x00002000, len = 0x00000004   /* Vector table            */
   PVECS2       : org = 0x00002004, len = 0x00000004   /* Vector table            */
   PVECS3       : org = 0x00002008, len = 0x00000004   /* Vector table            */
   PVECS4       : org = 0x0000200C, len = 0x00000004   /* Vector table            */
   PVECS5       : org = 0x00002010, len = 0x00000004   /* Vector table            */
   PVECS6       : org = 0x00002014, len = 0x00000004   /* Vector table            */
   PVECS7       : org = 0x00002018, len = 0x00000004   /* Vector table            */
   PVECS8       : org = 0x0000201C, len = 0x00000004   /* Vector table            */
```

In SECTIONS:

```
        .texte                  : {./debug/init_pmbus.obj} >  (EBFLASH)/*    put main into 2K boot flash      */
        .texte1                 : {./debug/pmbus.obj} >       (EBFLASH)/*                 */
        .texte2                 : {./debug/pmbus_handler.obj} >      (EBFLASH)/*       */
        .texte3                 : {./debug/multiimage.obj} >  (EBFLASH)/*       */
        .texte4                 : {./debug/pmbus_manuf_info_commands.obj} >    (EBFLASH)/*       */
        .texte5                 : {./debug/pmbus_manuf_specific_commands.obj} >         (EBFLASH)/*       */

        .text                   : {} > (BFLASH align(16))      /* Code                    */
        .const                  : {} > (BFLASH align(16))      /* Constant data                */
        .cinit                  : {} > (BFLASH align(16))      /* Initialization tables          */

         .vectors               : {} > VECS
        .pvectors1              : {} > PVECS1
        .pvectors2              : {} > PVECS2
        .pvectors3              : {} > PVECS3
        .pvectors4              : {} > PVECS4
        .pvectors5              : {} > PVECS5
        .pvectors6              : {} > PVECS6
        .pvectors7              : {} > PVECS7
        .pvectors8              : {} > PVECS8
        .sine                   : {} > SINE
```

Function_definitions.h

Add the function prototype definition.

```
void write_to_pflash_1_kernel(void);
void erase_pflash_1_pages_kernel(void);
void copy_pflash_write_to_ram(void);
void write_or_erase_pflash_block_1(void);
void copy_pflash_erase_to_ram(void);
void start_main_program(void);
void init_multiimage_interface(void);
Uint8 pmbus_read_multiimage(void);
Uint8 pmbus_write_multiimage(void);
```

load_64.asm

```
.sect ".vectors"
.state32
B       c_int00
B       vec_2
B       vec_3
B       vec_4
B       vec_5
B       vec_6
B       vec_7
B       vec_8
B       entry_from_main
        ; Extra vector to go to special entry point from main program - doesn't branch back to main program at
        beginning

.sect ".pvectors1"
        ;these are just place holders to provide destinations for the branches above and elsewhere in the code.
        ;The actual code is in the main program, not this boot.
.state32
```

main.c

If checksum at 0Xfffc(64KB) this function will send back 1 otherwise 0.

```
int32 verify_pflash_checksum(void)
{
        register int32 checksum = 0;
        register int32 address;

        for(address = 0x2000; address < 0xfffc; address++)
        {
                checksum = checksum + *(Uint8 *) address;
        }
        if(checksum == (*(int32 *) 0xfffc))
        {
                return 1;
        }
        else
        {
                return 0;
        }
}
```

This function for extended boot flash checksum, if memory address of 8KB checksum is true, this function will send back 1 otherwise 0.

```
int32 verify_extended_boot_flash_checksum(void)
{
        register int32 checksum = 0;
        register int32 address;

        for(address = 0x800; address < 0x1ffc; address++)
```

```
        {
                checksum = checksum + *(Uint8 *) address;
        }
        if(checksum == (*(int32 *) 0x1ffc))
        {
                return 1;
        }
        else
        {
                return 0;
        }
}
```

This function only execute at Multiimage to write block2.

```
void write_to_pflash_block_2(void)
{
        DecRegs.FLASHILOCK.all = PROGRAM_FLASH2_INTERLOCK_KEY;
        *(Uint32 *)multiimage_address = multiimage_data;
}
```

This function only execute at Multiimage to erase page.

```
void erase_page_in_pflash_block_2(void)
{
        //assume that calling routine will check busy - just write merrily away
        DecRegs.FLASHILOCK.all = PROGRAM_FLASH2_INTERLOCK_KEY;
        DecRegs.PFLASHCTRL2.bit.PAGE_SEL = multiimage_page_to_erase;
        DecRegs.PFLASHCTRL2.bit.PAGE_ERASE = 1;          //set bit to start erase of page.
        //outside firmware can wait for busy to be done.
}
```

This function only execute at Multiimage to erase block2.

```
void erase_pflash_block_2(void)
{
        //assume that calling routine will check busy - just write merrily away
        DecRegs.FLASHILOCK.all = PROGRAM_FLASH2_INTERLOCK_KEY;
        DecRegs.PFLASHCTRL2.bit.MASS_ERASE = 1; //set bit to start erase of page.
        //outside firmware can wait for busy to be done.
}
```

In this common_code. If Bootloader_status send back 0x40, it's mean Bootloader function is active.

```
void common_code()
{
        if(verify_extended_boot_flash_checksum() == 0)
        {
                // clear the boot flash checksum
                copy_pflash_write_to_ram();
                multiimage_data = 0;
                multiimage_address = 0x7fc;
                write_or_erase_pflash_block_1();
                for(;;)
                {
                 //infinite loop so we don't execute anything
                 }
```

```
        }
        bootloader_status = 0x40; //flag that bootloader is active.
        init_pmbus();
#ifdef DEBUG_ENABLED
        init_multiimage_interface();
#endif
        for(;;)
        {
                pmbus_handler();
        }
}
```

To check pflash checksum, if true go to start_main_program.

```
void main()
{
        if(verify_pflash_checksum())
        {
                start_main_program();
        }
        common_code();
}
```

pmbus.c

point to start of memory for download, otherwise send ack since we're not coming back.

```
int32 pmbus_write_d2_handler(void)
{
        if(pmbus_buffer[1] == 1)
        {
                multiimage_address = 0x2000;
                bootloader_status = 0x40; //flag that bootloader is active, clear other faults from bootloader.

        }
        if(pmbus_buffer[1] == 3)
        {
                PMBusRegs.PMBACK.byte.BYTE0 = 1;
                start_main_program(); //response to switch command - ignores checksum.
        }
        return 0;
}
```

Support D4 command to write 16byte bootloaded memory.

```
int32 pmbus_write_d4_handler(void)
{
        int i = 0; //index into eeprom buffer
        int j;

        if(multiimage_address == 0x2000) //if this is the first record
        {
                program_flash_erase();
        }
```

```
	for(j = 0; j < 16; j = j + 4) //16 bytes into 4 words
	{
		multiimage_buffer[i++] = (pmbus_buffer[1 + j] << 24) +
							(pmbus_buffer[2 + j] << 16) +
							(pmbus_buffer[3 + j] << 8) +
							pmbus_buffer[4 + j];
	}
	if(multiimage_address > 0x7fff) //if it's in the other block
	{
		for(i = 0;i < 4;i++)
		{
			multiimage_data = multiimage_buffer[i];
			write_to_pflash_block_2();

			while(DecRegs.PFLASHCTRL2.bit.BUSY != 0)
			{
				; //do nothing while it writes
			}

			multiimage_address = multiimage_address + 4; //point at next word.
		}
	}
	else
	{
		copy_pflash_write_to_ram();
		for(i = 0;i < 4;i++)
		{
			multiimage_data = multiimage_buffer[i];
			write_or_erase_pflash_block_1();

			multiimage_address = multiimage_address + 4; //point at next word.
		}
	}
	if(multiimage_address == 0x10000) //if we have finished programming the flash, check the checksum
	{
		bootloader_status = bootloader_status + verify_pflash_checksum(); //put checksum status into
lsb of bootloader status.
	}

	return 0;
}
```

…

Look at command byte from a write perspective. If we didn't enable the "DEBUG_ENABLE", our PMBus only support PMBus read one byte (0XD2) and PMBus 16byte write Bootloader memory (0XD4).

```
int32 pmbus_write_message(void)
{
	switch (pmbus_buffer[0])
	{
```

```
		case PMBUS_CMD_ROM_MODE: //can be taken out, but will make it difficult to reprogram boot flash - can always download a main program which clears boot flash
			return pmbus_write_rom_mode();

#ifdef DEBUG_ENABLED
		case PMBUS_CMD_MFR_MULTIIMAGE:
			return pmbus_write_multiimage();
		case PMBUS_CMD_MFR_PARM_INFO:
			return pmbus_write_parm_info();
		case PMBUS_CMD_MFR_PARM_VALUE:
			return pmbus_write_parm_value();
#endif
		case 0xd2:
			return pmbus_write_d2_handler();
		case 0xd4:
			return pmbus_write_d4_handler();
```

Look at command byte from a read perspective. If we didn't enable the "DEBUG_ENABLE", our PMBus only support PMBus read one byte (0XD2) and PMBus status (0x7E).

```
int32 pmbus_read_message(void)
{
	switch (pmbus_buffer[0])
	{
#ifdef DEBUG_ENABLED
	case PMBUS_CMD_MFR_MULTIIMAGE:
		return pmbus_read_multiimage();
#endif
	case 0xd2:
		return pmbus_read_one_byte_handler(0x40);
	case PMBUS_CMD_STATUS_CML:
		return pmbus_read_one_byte_handler(0);
		default:
			pmbus_number_of_bytes = 16;
			pmbus_buffer[0] = 0xff;
			pmbus_buffer[1] = 0xff;
			pmbus_buffer[2] = 0xff;
			pmbus_buffer[3] = 0xff;
			pmbus_buffer[4] = 0xff;
			pmbus_buffer[5] = 0xff;
			pmbus_buffer[6] = 0xff;
			pmbus_buffer[7] = 0xff;
			pmbus_buffer[8] = 0xff;
			pmbus_buffer[9] = 0xff;
			pmbus_buffer[10] = 0xff;
			pmbus_buffer[11] = 0xff;
			pmbus_buffer[12] = 0xff;
			pmbus_buffer[13] = 0xff;
			pmbus_buffer[14] = 0xff;
```

```
                    pmbus_buffer[15] = 0xff;
                    return 0;
        }
}
```

pmbus.h

For new added PMBus command , use must to revise the Pmbus table to support D1, D2 & D4.

```
#define CMD_DCDC_NONPAGED  \
    {0x00, 0x00, \
     0x00, 0x00, \
     0x00, 0x00, \
     0x00, 0x00, \
     0x00, 0x00, \
     0x00, 0x00, \
     0x00, 0x00, \
     0x00, 0x00, \
     0x00, 0x00, \
     0x00, 0xFE, \
     0x00, 0x00, \
     0x00, 0x00, \
     0x00, 0x00, \
     0x00, 0x40, \
     0x3D, 0x10, \
     0x00, 0x04 \
     }
```

variables.h

```
// For support Multiimage, we create a new Multiimage_parameters structure for multiimage variables.
struct MULTIIMAGE_PARAMETERS_STRUCT
{
        Uint8 write_delay;
        Uint16 page_write_delay;
        Uint16 block_erase_delay;
        Uint16 checksum_erase_delay;
        Uint16 image_switch_delay;
        Uint8 memory_read_delay;
        Uint8 boot_block_size;
        Uint8 block_erase_needed;
        Uint8 page_size;
        Uint8 block_size;
        Uint8 memory_size;
        Uint8 erase_page_size;
};
Global variable definenition.
EXTERN struct MULTIIMAGE_PARAMETERS_STRUCT multiimage_parameters;
EXTERN int multiimage_state; //state of multiimage handler
EXTERN int multiimage_address, multiimage_data, multiimage_page_to_erase;
EXTERN Uint32 multiimage_buffer[32];
```

```
EXTERN Uint32 *multiimage_copy_start;
EXTERN int multiimage_read_state; //tells what will be read with the multiimage
EXTERN int multiimage_error_code;
EXTERN int multiimage_calculated_checksum;
EXTERN int multiimage_read_checksum;
EXTERN int multiimage_checksum_byte_count;
EXTERN int multiimage_block_number;
EXTERN Uint8 bootloader_status;
```

pmbus_manuf_specific_commands.c

rom_back_door only provide a method to clear the boot flash checksum. Erases the program integrity word in FLASH.

```
void rom_back_door(void)
{
        copy_pflash_write_to_ram();
        multiimage_data = 0;
        multiimage_address = 0x7fc;
        write_or_erase_pflash_block_1();
}
```