UCD3138064
Programmer's Manual

SLUUAD8B – May 2014

Table of Contents

Scope of this Document

The following topics are covered in the UCD3138064 Enhancements Programmer's Manual

- **Memory Map Changes in UCD3138064 compared to UCD3138**
  - **Added FLASH BLOCK**
  - **Relocation of Fast Peripherals**
  - **Relocation of ROM**
- **Added Interfaces for External EEPROM**
  - **I2C Interface for External EEPROM**
  - **SPI Interface for External EEPROM**
- **Other Changes in UCD3138064 compared to UCD3138**
  - **Changes to IOMUX to support SPI and I2C pin multiplexing**
  - **BLANK_PCM_ENABLE bit to optimize Peak Current Mode response time**
- **How to migrate firmware programs from UCD3138 to UCD3138064**

Topics covered in the following UCD3138 programmer's manuals are also relevant to UCD3138064:

UCD3138 ARM and Digital System Programmer's Manual (Literature #: SLUU994)

- Boot ROM & Boot Flash
  - o BootROM Function
  - o Memory Read/Write Functions
  - o Checksum Functions
  - o Flash Functions
  - o Avoiding Program Flash Lock-Up
- ARM7 Architecture
  - o Modes of Operation
  - o Hardware/Software Interrupts
  - o Instruction Set
  - o Dual State Inter-working (Thumb 16-bit Mode/ARM 32-bit Mode)
- Memory & System Module
  - o Address Decoder, DEC (Memory Mapping)
  - o Memory Controller (MMC)
  - o Central Interrupt Module
- Register Map for all of the above peripherals in UCD3138

UCD3138 Monitoring and Communications Programmer's Manual (Literature #: SLUU996)

- ADC12
  - Control, Conversion, Sequencing & Averaging
  - Digital Comparators
  - Temperature Sensor
  - PMBUS Addressing
  - Dual Sample & Hold
- Miscellaneous Analog Controls (Current Sharing, Brown-Out, Clock-Gating)
- PMBUS Interface
- General Purpose Input Output (GPIO)
- Timer Modules
- PMBus
- Register Map for all of the above peripherals in UCD3138

UCD3138 Digital Power Peripheral Programmer's Manual(Literature #: SLUU995)

- Digital Pulse Width Modulator (DPWM)
  - Modes of Operation (Normal/Multi/Phase-shift/Resonant etc)

  - Automatic Mode Switching
  - DPWMC, Edge Generation & Intra-Mux
- Front End
  - Analog Front End
  - Error ADC or EADC
  - Front End DAC
  - Ramp Module
  - Successive Approximation Register Module
- Filter
  - Filter Math
- Loop Mux
  - Analog Peak Current Mode
  - Constant Current/Constant Power (CCCP)
  - Automatic Cycle Adjustment
- Fault Mux
  - Analog Comparators
  - Digital Comparators
  - Fault Pin functions
  - DPWM Fault Action
  - Ideal Diode Emulation (IDE), DCM Detection
  - Oscillator Failure Detection
- Register Map for all of the above peripherals in UCD3138

FUSION_DIGITAL_POWER_DESIGNER GUI for Isolated Power Applications (Literature #: SLUA676)

For the most up to date product specifications please consult the UCD3138064 Device datasheet (Literature # SLUSB72) available at http://www.ti.com/product/ucd3138064.

1 Introduction

The UCD3138064 device is a 64kB program flash memory derivative of Texas Instruments' UCD3138 with additional options to interface with external memory. These are provided to support field updates of firmware and support larger size firmware programs. A summary of the key differences between UCD3138064 and UCD3138 is provided in Table 2-1 in UCD3138064 device datasheet (Literature #: SLUSB72).

This manual highlights the differences between the UCD3138 and the UCD3138064. It describes the newly added features in UCD3138064 and changes to the features in UCD3138. It also gives guidance on converting firmware programs from the UCD3138 to the UCD3138064. The UCD3138 Programmer's Manuals mentioned earlier should be used for information on all elements that are common to both devices.

The UCD3138064 adds:

- An additional completely independent 32 Kbyte FLASH memory block for a total of 64 Kbytes (vs. 32kbytes available in UCD3138)
- An I^2C Master interface for interface to external EEPROM (unavailable in UCD3138)
- An SPI interface for external EEPROM (unavailable in UCD3138)

The two FLASH blocks are completely independent. It is possible to execute from one of the blocks while simultaneously writing to or erasing the other block. This permits live switching from one program version to another without interrupting power supply operation.

The Boot ROM program is changed slightly to support the new memory capabilities.

The additional memory and peripherals require a significant change in the memory map of the UCD3138064. This is mostly transparent to the programmer because almost all of the existing register names stay the same. New device files are supplied which change the addresses used by the development tools.

Three additional registers are added to the DEC-Address-Manager-area to support the new FLASH block.

Most of the I^2C and SPI interface additions are in new sets of peripheral registers, but the pin multiplexing is in the IOMUX register in the Miscellaneous Analog Control peripheral. IOMUX is also changed to permit optimal multiplexing of the 48 pin version of UCD3138064.

The BLANK_PCM_EN bit is added to the DPWMCTRL2 registers. This enhances PCM (Peak Current Mode).

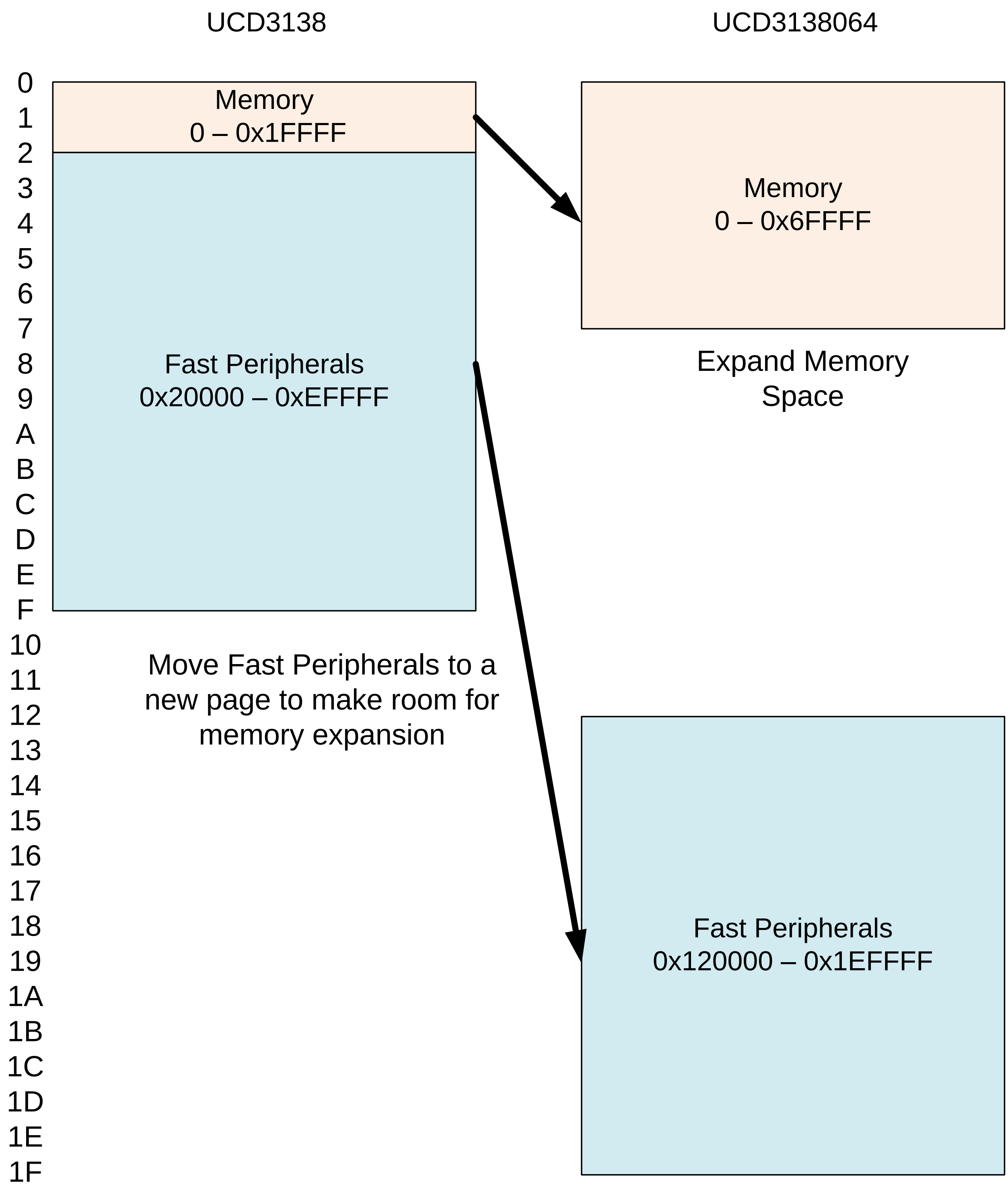
Finally, this manual describes how to take TI-provided UCD3138 reference code (eg. UCD3138 EVM firmware code) and convert it to apply to UCD3138064.

2 Memory Map Changes for UCD3138064

The addition of extra Program FLASH to the UCD3138064 requires that the other memories and many of the peripherals move to higher addresses.

2.1 Overall memory map changes

The figure below shows the memory map changes for the UCD3138064.

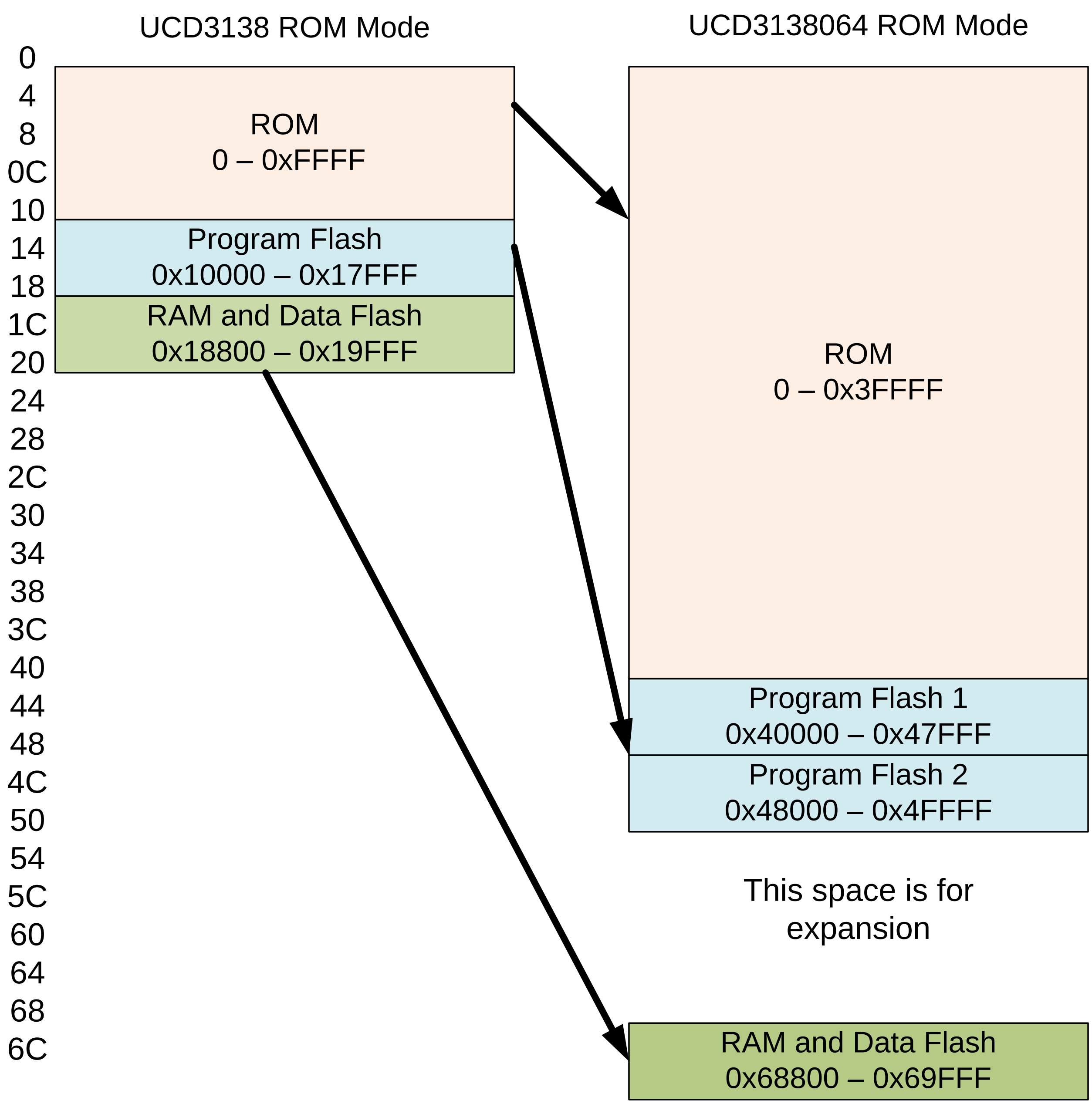


Overall Memory Map Changes (UCD3138 vs. UCD3138064)

The fast peripherals are moved up by 0x100000 bytes. This makes it very simple to calculate their new addresses. The slow peripherals, those mapped to 0xFFF7F600 and above, are not moved at all, so they are not shown on the figure. Fast peripherals use the same address decode scheme as memories. They have address mapping registers and run at the processor clock speed. Slow peripherals are on a separate I/O bus with no address mapping registers. The I/O bus and the slow peripherals run at half the processor clock speed.

2.2 Changes to ROM, RAM, and FLASH addresses

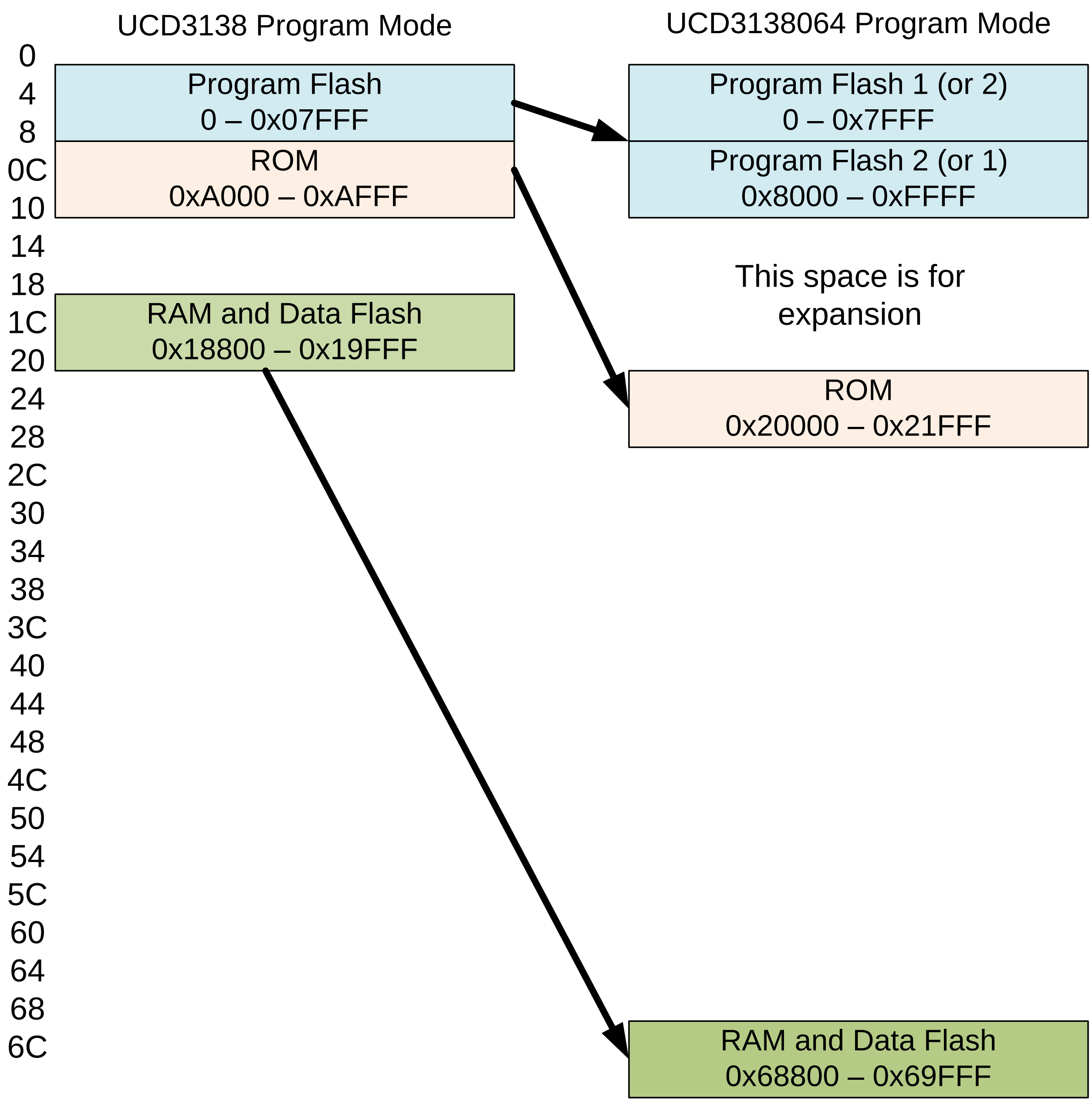
The UCD3138064 has different memory maps, depending on whether it is in ROM mode or in FLASH mode. In ROM mode, the ROM is located at location 0 so that it provides the reset and interrupt vector table. Here are the maps for UCD3138 and UCD3138064 in ROM mode.



Memory Map changes to ROM, RAM, and FLASH in ROM mode (UCD3138 vs. UCD3138064)

In the ARM core, the reset and interrupt vectors start at location 0 in memory. At power up reset, the ROM must be at location 0 to provide the vectors. In flash mode, the flash must be at location 0 to control the interrupt vectors. This is accomplished by changing the addresses of the ROM and FLASH. In ROM mode, the ROM must extend from location 0 to the location where it will be in FLASH mode. The ROM reset vectors make it jump to the locations it will execute from when the program FLASH is moved down to the zero location. This way, when flash mode is entered, the ROM simply remaps itself to the higher address. The program is already executing there. Then it remaps the flash to location 0 and jumps the vector at 0. The ROM on the UCD3138064 is 8K. That same 8K image is repeated throughout the entire memory space from 0 to 0x3FFFF in ROM mode.

The figure below shows the changes for FLASH mode for the UCD3138064:



Memory Map changes to ROM, RAM, and FLASH in FLASH mode (UCD3138 vs. UCD3138064)

Note that the program FLASH blocks, 1 and 2, are interchangeable. Either one can be moved to location 0. This means that two independent programs with different interrupt vectors can be stored in chip at the same time. The programs themselves can switch control between the two programs, or it can be done through a reset using the ROM program. The entire 64K space can also be used to hold a single program. A 2K boot flash/62K program space configuration is also possible.

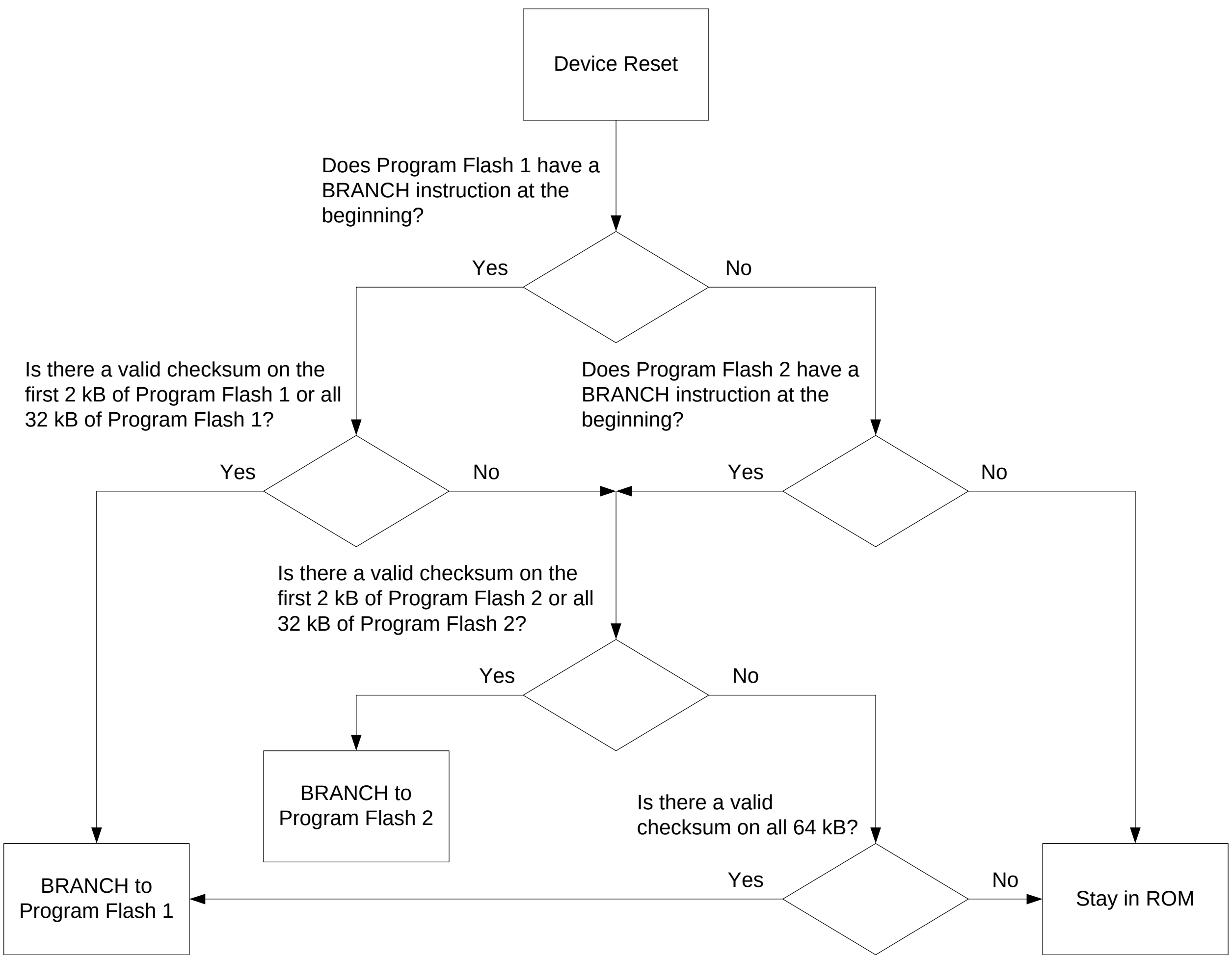
3 Changes to ROM Boot Program

The ROM program changes how it handles program startup. Instead of 2 checksums on the UCD3138, there are now 4 locations for checksums on the UCD3138064, and one of those does double duty as a checksum for two different blocks. The checksums and their locations are:

0x7fc – Boot block for Block 1
0x7ffc – Overall checksum for Block 1
0x87fc – Boot block for Block 2
0xfffc – Overall checksum for a 64K program combining Blocks 1 and 2 or for Block 2 alone

The locations above assume that Block1 is mapped to location 0, and Block 2 is mapped to 0x8000.

Here is a flowchart showing the order in which the ROM verifies the checksums:



The branch instruction check prevents the checksum program from looking at an empty block of memory.

When the flowchart says "Branch to Program Flash 1", this means that it puts Block 1 at location 0 and branches to location 0. For ""Branch to Program Flash 2", it puts Block 2 at 0 and branches there. The other block is always put at 0x8000.

The UCD3138 has a PMBus command code 0xF0 which causes the program to execute. On the UCD3138064, the same command causes Block 1 to be placed in control. A 0xF7 command has been added which puts Block 2 at location 0 and starts executing there.

In addition, the ROM has been expanded to 8K bytes.

4 I^{2}C Interface to EEPROM

The I^{2}C interface is a modified PMBus interface. When it is used for I^{2}C mode, many of the bit fields are not used. For a complete list of the registers and bit fields, see 7, I^{2}C Interface Reference.

4.1 I2C Initialization

Very few of the I^{2}C interface register values need to be changed from their default states. Here is the complete initialization required to write to an I^{2}C EEPROM device.

Configure FAULT0 and FAULT 1 pins for I2C.

MiscAnalogRegs.IOMUX.bit.FAULT_01_MUX_SEL = 2;

Fault 0 is used for I^{2}C clock, and Fault 1 is used for ^{I2}C data

Enable master mode and I^{2}C mode.

I2CRegs.I2CCTRL3.bit.MASTER_EN = 1;
I2CRegs.I2CCTRL3.bit.I2C_MODE_EN = 1;

Since the I^{2}C interface is derived from a PMBus interface, the Clock Low Timeout function should be disabled for full I^{2}C emulation:

I2CRegs.I2CCTRL3.bit.CLK_LO_DIS = 1;

In practice, the Clock Low Timeout is very unlikely to occur with an EEPROM. It can be enabled safely to help detect firmware issues with the I^{2}C interface.

4.2 I2C speed setting

The default speed for the I^{2}C hardware is 100 KHz. Setting the FAST_MODE_PLUS bit will give approximately 1 MHz:

I2CRegs.I2CCTRL3.bit.FAST_MODE_PLUS = 1;

It is also possible to set the FAST_MODE bit for 400 KHz.

4.3 Enabling the I2C EOM Interrupt

Some of the program logic will be simpler if the I^2C EOM interrupt is enabled in the I^2C peripheral. It is not necessary to enable it in the CIM (Central Interrupt Module). The firmware will just poll the interrupt request flag in the CIM to determine if the EOM bit is set. For details see Section 4.55 Waiting Until the EEPROM is Ready for the Next Write.

4.4 Writing to the EEPROM with I^2C

The I^2C interface is very simple to use for writing to an external EEPROM. Many I^2C EEPROMs require 2 bytes of address followed by 1 to 128 bytes of data.

The I^2C write is started by a firmware write to the I2CRegs.I2CCTRL1 register.

This means 2 things:

1. The I2CRegs.I2CTXBUF register should be initialized first
2. Only one write should be performed to the I2CRegs.I2CCTRL1 register

To make a single write to the I2CRegs.I2CCTRL1 register convenient, create a variable with the same type:
union I2CCTRL1_REG i2cctrl1;

Then the variable can be initialized easily with bitwise structure statements before being written once to the actual register.

The first step, which can be done at initialization, is to put the address into the variable:

```
i2cctrl1.bit.SLAVE_ADDR = 0x50; //default address of eeprom
```

The byte count and read/write bit need to be configured as well:

```
i2cctrl1.bit.BYTE_COUNT = 130 ;//send address and 128 bytes
i2cctrl1.bit.RW = 0; //write
```

Before writing to the control register, the transmit buffer must be written. It is a good idea to use a variable for the transmit buffer as well. It is not necessary for the first write, but subsequent writes in the same I^2C message are started by the write to the transmit buffer.

There are no bitfields in the transmit buffer, so it can be represented by a simple Uint32:

```
Uint32 i,txbuf;
```

It can be loaded with the data to write:

```
        txbuf  = ((eeprom_address & 0x80) << 8);
//low byte of page address in second byte
        txbuf = txbuf +  ((eeprom_address & 0xff00) >> 8);
//high byte of page address in first byte.
        txbuf = txbuf +  ((write_data[0]) << 16);
//first byte of data to write in third byte.
        txbuf = txbuf +  ((write_data[1]) << 24);
//second byte of data to write in fourth byte.
```

The page size for this EEPROM is 128 bytes, so only the most significant bit of the low address byte is used. This ensures that the address starts at the beginning of the page. The address

needs to go out high byte first, low byte second, so the low byte of the address is put into the second byte of txbuf. The high byte is put in the first byte of txbuf.

Then both registers can be written:

```
I2CRegs.I2CTXBUF.all = txbuf ;
I2CRegs.I2CCTRL1 = i2cctrl1; //start write
```

This will write the two bytes of the address and the first two bytes of data to the EEPROM.

Next, the firmware needs to wait for the write to complete. When the write is complete, the DATA_REQUEST bit will be set. The firmware needs to wait until the bit is set:

```
while (I2CRegs.I2CST.bit.DATA_REQUEST == 0)
{
        ; //wait for data to go out
}
```

Note that the I2CST register is cleared on read.

Then the firmware can loop writing to I2CTXBUF and waiting for data request until all the data is sent. The I^{2}C hardware will automatically issue an I^{2}C stop sequence when the number of bytes in BYTE_COUNT has been sent out. This will cause the EEPROM to start programming the page. Smaller numbers of bytes can be written out if desired, simply by changing the BYTE_COUNT value.

4.5 Waiting Until the EEPROM is Ready for the Next Write with I^{2}C

It is possible to use a firmware delay function programmed for the maximum EEPROM write time. But many EEPROMs are self clocked, and will normally write more quickly than the specified maximum. These EEPROMs generally will NACK I^{2}C messages until programming is complete. Here is the sequence used to support this feedback from the EEPROM.

4.5.1 Step 1 Detect end of final write to EEPROM

Instead of a DATA_REQUEST, the final write command will return an EOM bit. So the firmware can first wait for the EOM bit to be set:

```
if (I2CRegs.I2CST.bit.EOM == 1) //when i2c is done
```

4.5.2 Step 2. Try a simple read from EEPROM

Once the EOM is received, it is necessary poll the EEPROM to see when it is done.
A one byte read is a legitimate command for many EEPROMs. It will just read the next location in the EEPROM, assuming the EEPROM is done writing. Since the slave address is already initialized, only 2 bitfields in I2CCTRL2 must be written to:

```
i2cctrl1.bit.RW = 1; //read
i2cctrl1.bit.BYTE_COUNT =  1;
I2CRegs.I2CCTRL1 = i2cctrl1;
```

This will cause the I^{2}C interface to send out one byte with the slave address, and receive one byte back from the EEPROM.

4.5.3 Step 3. Determine if message is ACKed or NACKed

This step is complicated by the fact that the NACK bit will be set before the EOM bit on messages which are NACKed. One method for dealing with this is to read the EOM interrupt bit from the Central Interrupt Module. The interrupt is enabled at the I^2C:

```
I2CRegs.I2CINTM.bit.EOM = 0; ,
```

but it is not enabled in the Central Interrupt Module. This way the EOM state can be polled without reading from the I2CST register. This makes the program logic simpler.

```
if(CimRegs.INTREQ.bit.INTREQ_DIGI_COMP == 1)
//if i2c eom interrupt is being requested
{
        if(I2CRegs.I2CST.bit.NACK == 0) //if we were acked, it's done
```

The digital comparators share the interrupt bit with the I2C. If the digital comparator interrupt is also being used, this scheme will not work.

If the NACK bit is not set, then the write of the next page can begin. If it is set, then steps 2 and 3 must be repeated until the NACK bit is not set in the if-statement above.

4.6 Reading From the EEPROM at a Specific Address with I2C (Random Read)

The EEPROM random read sequence involves sending:

1. An I^2C start sequence
2. The EEPROM I^2C address (1 byte) with the R/W bit set to zero. This signals a write.
3. 2 bytes of address within the EEPROM
4. An I^2C repeated start sequence
5. The EEPROM I^2C address (1 byte) with the R/W bit set to one. This signals a read
6. As many reads as are desired.
7. A NACK from the UCD3138064 followed by a stop sequence.

This sequence is the same as the extended command sequence in the PMBus, so the EXT_CMD bit can be used.

The I2CTXBUF register is loaded with the address at which the read will start:

```
I2CRegs.I2CTXBUF.all = (((eeprom_address & 0xff) << 8) +
                        ((eeprom_address & 0xff00) >> 8));
```

Then the control register is loaded using the RAM variable:

```
i2cctrl1.bit.EXT_CMD = 1 ;
//set it as extended command to get 2 bytes out
i2cctrl1.bit.RW = 1; //read
i2cctrl1.bit.CMD_ENA = 1; //otherwise we'll just get a read.
i2cctrl1.bit.BYTE_COUNT =  16;
I2CRegs.I2CCTRL1 = i2cctrl1; //need single write to ctrl1.
```

The EXT_CMD and CMD_ENA bits are only set for this function, so it is efficient to clear them in the i2cctrl1 variable as soon as possible. This example will read 16 bytes.

```
i2cctrl1.bit.EXT_CMD = 0 ; //clear it for other uses.
i2cctrl1.bit.CMD_ENA = 0;
```

If the CMD_ENA bit is not set, the I^2C logic will ignore the contents of TXBUF and the EXT_CMD bit and only send out a read address.

The code sequence above will handle steps 1 through 5 above, and will accept the first 4 bytes of data from the EEPROM.

The I^2C peripheral will signal that the bytes are ready by setting the DATA_READY flag in I2CST:

```
while(I2CRegs.I2CST.bit.DATA_READY == 0)
{
        //wait for DATA_READY
}
eeprom_buffer[0] = I2CRegs.I2CRXBUF.all;
 //this read starts an i2c read.
```

When the firmware reads from the RXBUF, the I^2C peripheral will start clocking in the next 4 bytes. After the desired number of bytes are read in, the I^2C peripheral will automatically send a NACK and a stop sequence to finish the transaction.

4.7 I^2C Current Address Read

Only one address is needed to read a long string of data from the EEPROM. Because of the size of BYTE_COUNT, the I^2C interface can only support reading 255 bytes per I^2C message. With the current address read, additional blocks can be read easily and efficiently. A random read is used first to set the address. After this current address reads are executed until all the data is read.

To execute a current address read, do exactly the same thing as the random read above, but omit the sending of the address. Omit the loading of the TXBUF register and the setting of the CMD_ENA and EXT_CMD bits. This will cause the I^2C interface to just issue the device address with a read state in the LSB. Then it will read data from the EEPROM just as the random read does.

```
i2cctrl1.bit.RW = 1; //read
i2cctrl1.bit.BYTE_COUNT =  16;
I2CRegs.I2CCTRL1 = i2cctrl1; //need single write to ctrl1.
```

5 SPI Interface to EEPROM

The SPI interface uses 4 wires to interface to the EEPROM. The peripheral logic and registers are completely different from I^2C, even though 2 of the pins (Fault 0 and Fault 1) are still the same. The pins are:

- TDI – SPI_MISO
- TDO- DPI_MOSI
- FAULT1 SPI_CLK
- FAULT0 SPI_CS

Most of the control for SPI is in SPIRegs. Like I^2C, the IOMUX must also be configured.

5.1 SPI Initialization

The initialization for SPI is very simple. Two IOMUX bitfields need to be configured to enable the 4 SPI pins:

```
MiscAnalogRegs.IOMUX.bit.FAULT_01_MUX_SEL  = 1;
// SPI CS/CLK  on pins Fault-0 / Fault-1

MiscAnalogRegs.IOMUX.bit.JTAG_DATA_MUX_SEL = 4;
// MOSI / MISO on pins  TDO / TDI
```

Next the SPI interface needs to be enabled:

```
SPIRegs.SPICTRL.bit.SPIEN   = 1;
// enable SPI - only non-default statement needed.
```

This is all that is required for initialization.

5.2 SPI Key Registers and bitfields

There are a few key registers and bitfields for the SPI interface. They are:

- SPICTRL.bit.TXCNT – sets number of bytes to transmit
- SPICTRL.bit.RXCNT – sets number of bytes to receive
- SPICTRL.bit.FRLMLEN – sets number of messages for which the hardware will automatically hold CS (Chip Select) low
- SPISTAT.bit.SPIF – shows that the SPI operation is complete
- SPITX0 – 4 byte transmit buffer – writing to this register starts the next SPI operation
- SPITX1 – another 4 byte transmit buffer – sent out after SPITX0
- SPIRX0-3 – 4 receive buffers of 4 bytes each

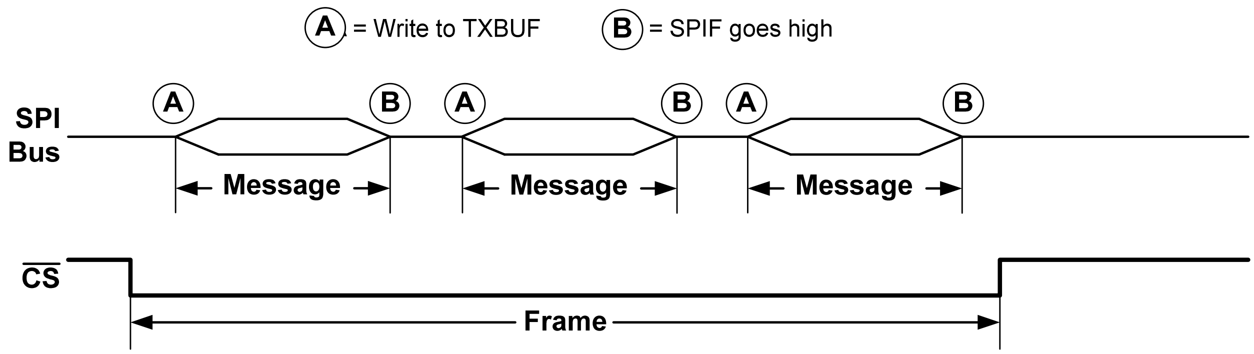
5.3 Frames and Messages

A single SPI transaction with an EEPROM can be much longer than will fit in the SPI buffers. Generally an SPI transaction is defined by the Chip Select pin going low at the beginning and high at the end. This is called a frame.

For the purposes of this document, a message is one emptying of the TXBUF and/or one loading of the RXBUF. Messages are always started by writing to one or more TX Buffer registers. Sometimes it may be necessary to change the values in TXCNT and RXCNT before writing to the TX Buffer. The SPI interface signals that the message is done by setting the SPIF bit.

A frame contains one or more messages. The first message in a frame needs to set up the FRMLEN bitfield. This field dictates how many messages there are in that frame. The SPI hardware will keep the CS (Chip Select) line active for all the messages in the frame. For some long frames, the CS pin must be controlled by firmware.

Here is an example of a frame with 3 messages:



There are several different length limitations on SPI messages on the UCD3138064.

- TX Buffer size – 8 bytes
- TXCNT size – 15 bytes
- RX Buffer size – 16 bytes
- RXCNT size – 31 bytes

So it is possible to transmit up to 15 bytes per message. This will require two writes to the TX Buffer Registers. Similarly, up to 31 bytes can be read at a time, with two reads from the RX Buffer Registers.

In a typical application, it is simpler to use the buffer sizes as the maximum message size. In this case, each message has up to 8 bytes of transmitted data and 16 bytes of received data.

If the message is small, the frame will have only one message. The maximum number of messages per frame is 31 messages. In practical applications, generally the maximum number of messages per frame is 16.

This means that the longest write frame is 8 bytes times 16, or 128 bytes. However some SPI EEPROMS have pages of 256 bytes or more. And the page write is started by the CS pin going inactive. In this case, instead of using the frame control of CS, it is simpler to just use direct firmware control of the pin.

5.4 SPI Programming Overview

Every SPI frame has very similar starting and ending C statements.

At the start of every frame, write to TXCNT, RXCNT, and FRMLEN (or activate CS pin)

Then write to the TX buffers with SPITX0 last to start the message.

The next step is to poll the SPIF. When it goes high, the message is done.

It is necessary to write a 1 to the SPIF to clear it.

When the message is done, read any data which needs to be read from the RX buffers. If more needs to be read or written in the frame, write any new values to TXCNT and RXCNT, and then write to at least SPITX0. Even if TXCNT is 0, a write to SPITX0 is what starts the next message.

This sequence continues until all of the messages have gone out and the frame is done. If FRMLEN is being used, then CS will automatically go inactive. If CS is being controlled directly by firmware, then it should be made inactive.

5.5 SPI Simple Example – Read EEPROM Status

Reading EEPROM status just requires sending 1 byte and receiving 1 byte – the status byte.

The first step is to write to TXCNT, RXCNT, and FRMLEN

```
SPIRegs.SPICTRL.bit.TXCNT   = 1;
SPIRegs.SPICTRL.bit.RXCNT   = 1;
SPIRegs.SPICTRL.bit.FRMLEN  = 1;
```

5.5.1 SPI sequence start – write to SPITX0

The sequence is always started by a write to SPITX0. For instance, in the status read, the read status command byte for the EEPROM is put into SPI_TX0:

```
SPIRegs.SPITX0.all = SPI_CMD_READ_STATUS_REGISTER;
        //starts SPI engine
```

With the write to the SPITX0 register, the SPI_CS line will go low, the SPI_CLK will start, and the data will be transferred.

Just like I^2C, the SPITX0 register should be written only once. If individual bytes must be assembled into the 4 bytes of the register, assemble them in a 4 byte variable in RAM and then write that variable into the register.

5.5.2 SPIF tells when sequence is complete or when new data is needed

The SPIF needs to be cleared by writing a 1 to it, so that the next end of message can be detected. Here is the code:

```
while(SPIRegs.SPISTAT.bit.SPIF == 0)
{
        //waiting for SPI to finish its message
}
SPIRegs.SPISTAT.bit.SPIF = 1; //1 to clear spif
```

5.5.3 SPI Read Status From SPIRX0

The final step is to read from the SPIRX0 and get the status byte. It will appear in the least significant bit of SPIRX0. Normally the status byte is read to see if a write is complete – if the EEPROM is no longer busy.

```
#define SPI_STATUS_BUSY_MASK (0x80)
        if((SPIRegs.SPIRX0.all & SPI_STATUS_BUSY_MASK) !=0)
//busy is zero, ready is 1.
```

5.6 SPI Read from Memory

Memory read on a typical SPI EEPROM is very flexible. The method described below reads 256 bytes with each frame.

TXCNT is loaded with a 5 because there are 3 bytes of meaningful data and 2 bytes of dummy data for a delay.

RXCNT is loaded with 16 to read 16 bytes each message.

FRMLEN is loaded with 16 to permit 16 messages.

Then the command and address bytes are combined in the RAM variable and written to the SPITX0 register to start the first command.

Once SPIF is set by the hardware and cleared by the firmware, the first 16 bytes are read from the SPIRXx registers. TXCNT should be written with a 0. Then something random can be written to the SPITX0 register to start the read, followed again by polling SPIF.

This same sequence is followed until 16 bytes have been read a total of 16 times. Then the SPI hardware will automatically make the CS pin inactive.

5.6.1 SPI Write Page to Memory

Using powers of 2 for TXCNT and FRMLEN, the SPI hardware does not support a full 256 byte write. So it is necessary to use the GPIO control of the CS pin.

The default value of SPIDIR.bit.SCS already supports an output, so it doesn't need to be written. But the pin function bit needs to be written:

```
SPIRegs.SPIFUNC.bit.SCS = 1;
//take firmware control of chip select.
```

Next it is necessary to send out a 1 byte command and a 3 byte page address, so TXCNT gets 4 bytes:

```
SPIRegs.SPICTRL.bit.TXCNT   = 4;
SPIRegs.SPICTRL.bit.RXCNT   = 0;
```

FRMLEN just needs to be short to avoid having CS active when the frame is done and CS control is returned to the SPI hardware.

```
SPIRegs.SPICTRL.bit.FRMLEN  = 1;
```

Then the write data needs to be assembled in a variable before the write to SPITX0:

```
temp = SPI_CMD_MAIN_MEMORY_PAGE_PROGRAM_THROUGH_BUFFER +
        ((eeprom_address & 0x3ff00)<< 1) + (eeprom_address & 0xff);

SPIRegs.SPITX0.all = temp; //starts SPI engine
```

Then there is a wait for the SPI message to go out, which just polls and then clears SPIF:

```
spi_engine_busy_wait();
        //wait for command and address to finish
```

Now WRSTART is set to a 1. WRSTART selects which SPITXx register starts the SPI message. When sending sequential data, using a write to SPITX1 is more convenient than a write to SPITX0:

```
SPIRegs.SPICTRL.bit.WRSTART = 1;
```

Now 8 bytes will go out with every message

```
SPIRegs.SPICTRL.bit.TXCNT   = 8; // now send out 8 bytes
```

To send out 256 bytes, 32 messages are required:

```
for(i = 0;i < 32;i++)
{
        SPIRegs.SPITX0.all = *source ++;
        SPIRegs.SPITX1.all = *source ++; //write to start
        spi_engine_busy_wait();
}
```

Source is a pointer to a 32 bit word.

Then it is necessary to return control of the CS to the SPI hardware:

```
SPIRegs.SPIFUNC.bit.SCS = 0;
        //release firmware control of chip select.
```

It is also good to clear WRSTART for the normal short messages which only write to SPITX0:

```
SPIRegs.SPICTRL.bit.WRSTART = 0; //most commands use 0, set it back to that.
```

This must be followed by a delay to permit the EEPROM to write the page. The delay can be accomplished using the timer interrupt. If maximum speed is desired, the firmware can continuously read the status register and wait for the EEPROM to signal that it is not busy.

6 Register Changes for Program Flash Block 2

Three registers are added to the UCD3138064 to control the second block of program flash. They are:

- DecRegs.PFLASHCTRL2
- DecRegs. MFBAHR17
- DecRegs.MFBALR17
-

In addition, PFLASHCTRL, which controls block 1, has its name changed to PFLASHCTRL1.

The new registers work the same as the registers for block 1. PFLASHCTRL2 is used for page and mass erase, and for status monitoring of block 2. The MFB… registers are used to change the address mapping of Block 2. They can be used to map block 2 to the 0 location in memory so that a code image in block 2 can provide the interrupt vectors to the processor.

Please refer to UCD3138 ARM and Digital System programmer's manual (literature #: SLUU994) for more information on these types of registers.

There is still only one DecRegs.FLASHILOCK register, but a different value must be written to it to unlock program flash block 2. Data flash and block 1 still have the same number as before:

```
#define PROGRAM_FLASH1_INTERLOCK_KEY 0x42DC157E

#define PROGRAM_FLASH2_INTERLOCK_KEY 0x6C97D0C5

#define DATA_FLASH_INTERLOCK_KEY 0x42DC157E
```

See the Dec – Address Manager Reference section of this document for information on the detailed memory and bit maps for these registers.

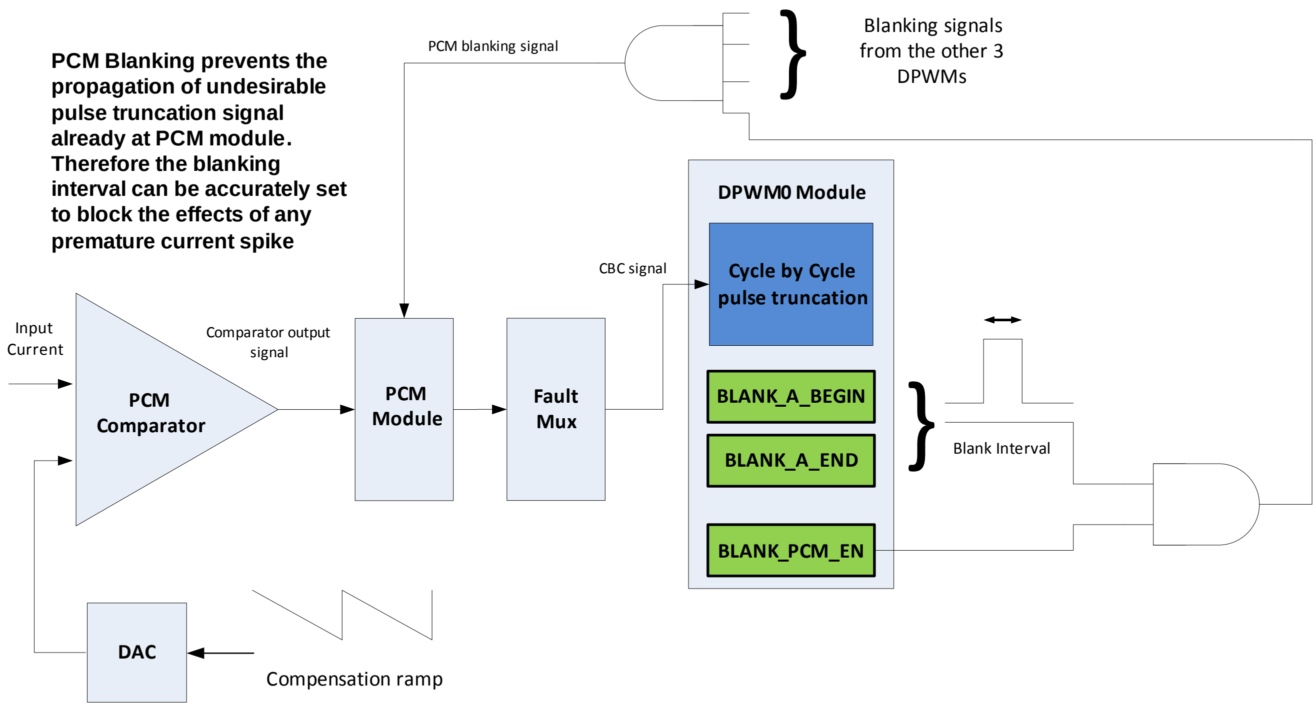
7 Enhanced Peak Current Mode Blanking

The BLANK_PCM_EN bit in DPWMCTRL2 provides improved blanking of noise spikes in the Peak Current Mode comparator signal path. PCM blanking provides an earlier blanking than the blanking using the Blank A and Blank B enable bits on DPWM modules when used for PCM. It also makes it possible to use blanking signals from multiple DPWMs for all DPWMs.

When Blank A enable bit in a DPWM module is used, the blanking signal from the DPWM is 'and'ed with the CBC signal in the DPWM logic. This means that the CBC signal must first propagate though the PCM module and then through the Fault Mux before it arrives at the DPWM for blanking. It also means that the only blanking signals are available are Blank A and Blank B from that DPWM. With PSFB (Phase Shifted Full Bridge), only Blank A is available, because Blank B is being used to generate a waveform.

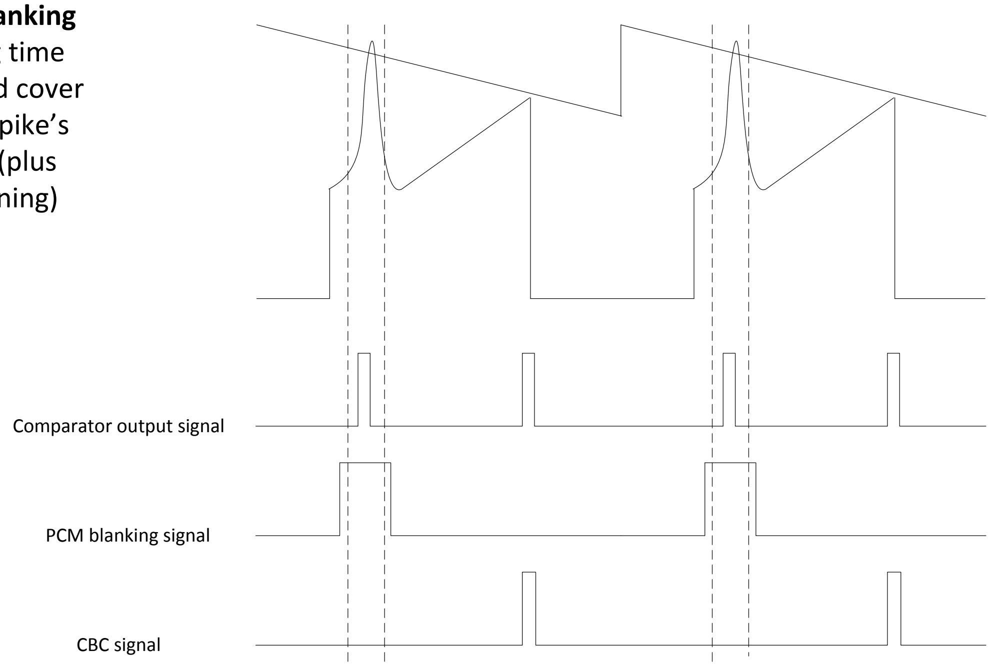
With the BLANK_PCM_EN bit set, the blanking signal is sent to the PCM module, so it can be 'and'ed much sooner in the signal path. The latency from input to DPWM shut off is not changed, but the signal blanking interval is much closer to the actual interval from the blanking registers. This makes the blanking calculation easier. In addition, all the blanking signals are combined in the PCM module, so up to 8 blanking intervals can be applied to the CBC signal to all the DPWMs.

Here is the block diagram:

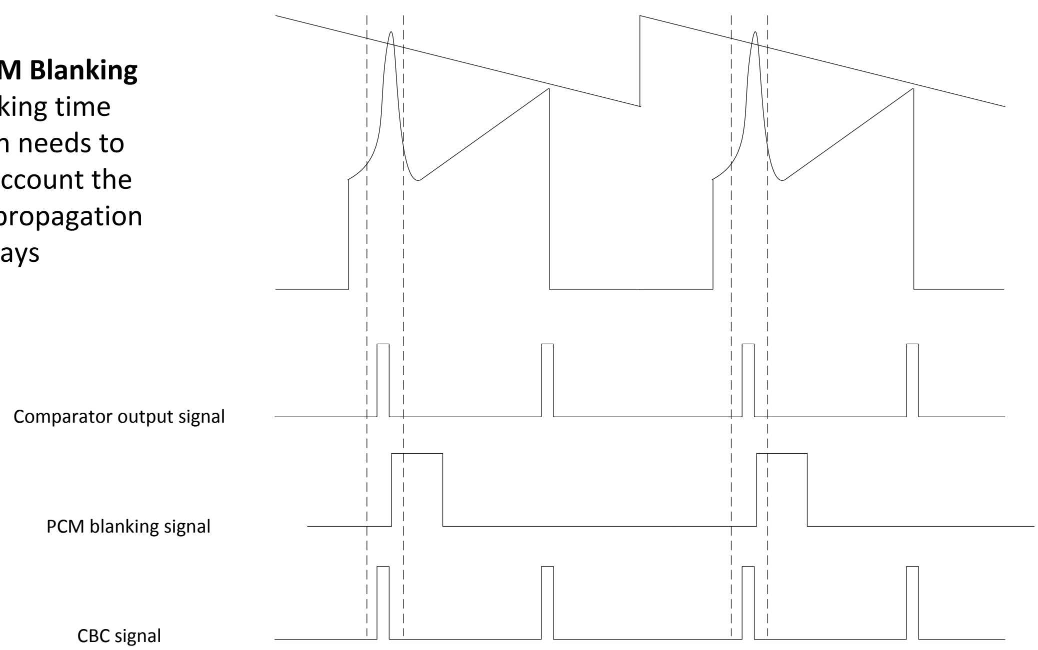


Here is the relative timing with and without BLANK_PCM_EN:

With PCM Blanking
The blanking time interval should cover the current spike's occurrence (plus some margining)



With DPWM Blanking
The blanking time calculation needs to take into account the CBC signal propagation delays



8 Changes to HFO_LN_FILTER_EN bit

On the UCD3138, it is recommended to clear the HFO_LN_FILTER_EN bit in the CLKTRIM register. On the UCD3138064, the HFO_LN_FILTER_EN register is controlled by the test program, it has become a trim bit. This way TI can control the bit setting at test time for optimal results. It is recommended that writes to HFO_LN_FILTER_EN be removed when moving programs from UCD3138 to UCD3138064.

9 I2C Interface Reference

This section describes the new registers added for the I2C interface

9.1 I2C Control Register 1 (I2CCTRL1)

Address FFF7E400

Bit Number	**20**	**19**	**18**	**17**
Bit Name	PRC_CALL	GRP_CMD	PEC_ENA	EXT_CMD
Access	R/W	R/W	R/W	R/W
Default	0	0	0	0

Bit Number	**16**	**15:8**	**7:1**	**0**
Bit Name	CMD_ENA	BYTE_COUNT	SLAVE_ADDR	RW
Access	R/W	R/W	R/W	R/W
Default	0	0000_0000	000_0000	0

Bit 20: PRC_CALL – Master Process Call Message Enable
0 = Default state for all messages besides Process Call message (Default)
1 = Enables transmission of Process Call message
Bit 19: GRP_CMD – Master Group Command Message Enable
0 = Default state for all messages besides Group Command message (Default)
1 = Enables transmission of Group Command message
Bit 18: PEC_ENA – Master PEC Processing Enable
0 = Disables PEC processing (Default)
1 = Enables PEC byte transmission/reception
Bit 17: EXT_CMD – Master Extended Command Code Enable
0 = Use 1 byte for Command Code (Default)
1 = Use 2 bytes for Command Code
Bit 16: CMD_ENA – Master Command Code Enable
0 = Disables use of command code on Master initiated messages (Default)
1 = Enables use of command code on Master initiated messages
Bits 15-8: BYTE_COUNT – Indicates number of data bytes transmitted in current message. Byte count does not include any device addresses, command words or block lengths in block messages. In block messages, the PMBus Interface automatically inserts the block length into the message based on the byte count setting. The firmware only needs to load the address, command words and data to be transmitted. PMBus Interface supports byte writes up to 255 bytes.
Bits 7-1: SLAVE_ADDR – Specifies the address of the slave to which the current message is directed towards.
Bit 0: RW – Indicates if current Master initiated message is read operation or write operation.
0 = Message is a write transaction (data from Master to Slave) (Default)
1 = Message is a read transaction (data from Slave to Master)

9.2 I2C Transmit Data Buffer (I2CTXBUF)

Address FFF7E404

Bit Number	31:24	23:16	15:8	7:0
Bit Name	BYTE3	BYTE2	BYTE1	BYTE0
Access	R/W	R/W	R/W	R/W
Default	0000_0000	0000_0000	0000_0000	0000_0000

Bits 31-24: BYTE3 – Last data byte transmitted from Transmit Data Buffer
Bits 23-16: BYTE2 – Third data byte transmitted from Transmit Data Buffer
Bits 15-8: BYTE1 – Second data byte transmitted from Transmit Data Buffer
Bits 7-0: BYTE0 – First data byte transmitted from Transmit Data Buffer

9.3 I2C Receive Data Register (I2CRXBUF)

Address FFF7E408

Bit Number	31:24	23:16	15:8	7:0
Bit Name	BYTE3	BYTE2	BYTE1	BYTE0
Access	R	R	R	R
Default	-	-	-	-

Bits 31-24: BYTE3 – Last data byte received in Receive Data Buffer
Bits 23-16: BYTE2 – Third data byte received in Receive Data Buffer
Bits 15-8: BYTE1 – Second data byte received in Receive Data Buffer
Bits 7-0: BYTE0 – First data byte received in Receive Data Buffer

9.4 I2C Acknowledge Register (I2CACK)

Address FFF7E40C

Bit Number	0
Bit Name	ACK
Access	R/W
Default	0

Bit 0: ACK – Allows firmware to acknowledge or not acknowledge received data
0 = NACK received data (Default)
1 = Acknowledge received data, bit clears upon issue of ACK on PMBus

9.5 I2C Status Register (I2CST)

Address FFF7E410

Bit Number	**21**	**20**	**19**	**18**
Bit Name	SCL_RAW	SDA_RAW	CONTROL_RAW	ALERT_RAW
Access	R	R	R	R
Default	-	-	-	-

Bit Number	**17**	**16**	**15**
Bit Name	CONTROL_EDGE	ALERT_EDGE	MASTER
Access	R	R	R
Default	-	-	-

Bit Number	**14**	**13**	**12**	**11**	**10**
Bit Name	LOST_ARB	BUS_FREE	UNIT_BUSY	RPT_START	SLAVE_ADDR_READY
Access	R	R	R	R	R
Default	-	-	-	-	-

Bit Number	**9**	**8**	**7**	**6**
Bit Name	CLK_HIGH_DETECTED	CLK_LOW_TIMEOUT	PEC_VALID	NACK
Access	R	R	R	R
Default	-	-	-	-

Bit Number	**5**	**4**	**3**	**2:0**
Bit Name	EOM	DATA_REQUEST	DATA_READY	RD_BYTE_COUNT
Access	R	R	R	R
Default	-	-	-	-

Bit 21: SCL_RAW – PMBus Clock Pin Real Time Status
0 = PMBus clock pin observed at logic level low
1 = PMBus clock pin observed at logic level high
Bit 20: SDA_RAW – PMBus Data Pin Real Time Status
0 = PMBus data pin observed at logic level low
1 = PMBus data pin observed at logic level high
Bit 19: CONTROL_RAW – Control Pin Real Time Status
0 = Control pin observed at logic level low
1 = Control pin observed at logic level high
Bit 18: ALERT_RAW – Alert Pin Real Time Status
0 = Alert pin observed at logic level low
1 = Alert pin observed at logic level high
Bit 17: CONTROL_EDGE – Control Edge Detection Status
0 = Control pin has not transitioned
1 = Control pin has been asserted by another device on PMBus
Bit 16: ALERT_EDGE – Alert Edge Detection Status
0 = Alert pin has not transitioned
1 = Alert pin has been asserted by another device on PMBus
Bit 15: MASTER – Master Indicator
0 = PMBus Interface in Slave Mode or Idle Mode
1 = PMBus Interface in Master Mode
Bit 14: LOST_ARB – Lost Arbitration Flag

0 = Master has attained control of PMBus
1 = Master has lost arbitration and control of PMBus

Bit 13: BUS_FREE – PMBus Free Indicator
0 = PMBus processing current message
1 = PMBus available for new message

Bit 12: UNIT_BUSY – PMBus Busy Indicator
0 = PMBus Interface is idle, ready to transmit/receive message
1 = PMBus Interface is busy, processing current message

Bit 11: RPT_START – Repeated Start Flag
0 = No Repeated Start received by interface
1 = Repeated Start condition received by interface

Bit 10: SLAVE_ADDR_READY – Slave Address Ready
0 = Indicates no slave address is available for reading
1 = Slave address ready to be read from Receive Data Register (Bits 6:0)

Bit 9: CLK_HIGH_DETECTED – Clock High Detection Status
0 = No Clock High condition detected
1 = Clock High exceeded 50us during message

Bit 8: CLK_LOW_TIMEOUT – Clock Low Timeout Status
0 = No clock low timeout detected
1 = Clock low timeout detected, clock held low for greater than 35ms

Bit 7: PEC_VALID – PEC Valid Indicator
0 = Received PEC not valid (if EOM is asserted)
1 = Received PEC is valid

Bit 6: NACK – Not Acknowledge Flag Status
0 = Data transmitted has been accepted by receiver
1 = Receiver has not accepted transmitted data

Bit 5: EOM – End of Message Indicator
0 = Message still in progress or PMBus in idle state.
1 = End of current message detected

Bit 4: DATA_REQUEST – Data Request Flag
0 = No data needed by PMBus Interface
1 = PMBus Interface request additional data. PMBus clock stretching enabled to stall bus until firmware provides transmit data.

Bit 3: DATA_READY – Data Ready Flag
0 = No data available for reading by processor
1 = PMBus Interface read buffer full, firmware required to read data prior to further bus activity. PMBus clock stretching enabled to stall bus until data is read by firmware.

Bits 2-0: RD_BYTE_COUNT – Number of Data Bytes available in Receive Data Register
0 = No received data
1 = 1 byte received. Data located in Receive Data Register, Bits 7-0
2 = 2 bytes received. Data located in Receive Data Register, Bits 15-0
3 = 3 bytes received. Data located in Receive Data Register, Bits 23-0
4 = 4 bytes received. Data located in Receive Data Register, Bits 31-0

9.6 I2C Interrupt Mask Register (I2CINTM)

Address FFF7E414

Bit Number	9	8	7	6
Bit Name	CLK_HIGH_DETECT	LOST_ARB	CONTROL	ALERT
Access	R/W	R/W	R/W	R/W
Default	1	1	1	1

Bit Number	5	4	3
Bit Name	EOM	SLAVE_ADDR_READY	DATA_REQUEST

Access	R/W	R/W	R/W
Default	1	1	1

Bit Number	**2**	**1**	**0**
Bit Name	DATA_READY	BUS_LOW_TIMEOUT	BUS_FREE
Access	R/W	R/W	R/W
Default	1	1	1

Bit 9: CLK_HIGH_DETECT – Clock High Detection Interrupt Mask
0 = Generates interrupt if clock high exceeds 50us during message
1 = Disables interrupt generation for Clock High detection (Default)
Bit 8: LOST_ARB – Lost Arbitration Interrupt Mask
0 = Generates interrupt upon assertion of Lost Arbitration flag
1 = Disables interrupt generation upon assertion of Lost Arbitration flag (Default)
Bit 7: CONTROL – Control Detection Interrupt Mask
0 = Generates interrupt upon assertion of Control flag
1 = Disables interrupt generation upon assertion of Control flag (Default)
Bit 6: ALERT – Alert Detection Interrupt Mask
0 = Generates interrupt upon assertion of Alert flag
1 = Disables interrupt generation upon assertion of Alert flag (Default)
Bit 5: EOM – End of Message Interrupt Mask
0 = Generates interrupt upon assertion of End of Message flag
1 = Disables interrupt generation upon assertion of End of Message flag (Default)
Bit 4: SLAVE_ADDR_READY – Slave Address Ready Interrupt Mask
0 = Generates interrupt upon assertion of Slave Address Ready flag
1 = Disables interrupt generation upon assertion of Slave Address Ready flag (Default)
Bit 3: DATA_REQUEST – Data Request Interrupt Mask
0 = Generates interrupt upon assertion of Data Request flag
1 = Disables interrupt generation upon assertion of Data Request flag (Default)
Bit 2: DATA_READY – Data Ready Interrupt Mask
0 = Generates interrupt upon assertion of Data Ready flag
1 = Disables interrupt generation upon assertion of Data Ready flag (Default)
Bit 1: BUS_LOW_TIMEOUT – Clock Low Timeout Interrupt Mask
0 = Generates interrupt upon assertion of Clock Low Timeout flag
1 = Disables interrupt generation upon assertion of Clock Low Timeout flag (Default)
Bit 0: BUS_FREE – Bus Free Interrupt Mask
0 = Generates interrupt upon assertion of Bus Free flag
1 = Disables interrupt generation upon assertion of Bus Free flag (Default)

9.7 I2C Control Register 2 (I2CCTRL2)

Address FFF7E418

Bit Number	**22:21**	**20**	**19**	**18:16**
Bit Name	RX_BYTE_ACK_CNT	MAN_CMD	TX_PEC	TX_COUNT
Access	R/W	R/W	R/W	R/W
Default	11	0	0	000

Bit Number	**15**	**14:8**	**7**	**6:0**
Bit Name	PEC_ENA	SLAVE_MASK	MAN_SLAVE_ACK	SLAVE_ADDR
Access	R/W	R/W	R/W	R/W
Default	0	111_1111	0	111_1100

Bit 22-21: RX_BYTE_ACK_CNT – Configures number of data bytes to automatically acknowledge when receiving data in slave mode.

00 = 1 byte received by slave. Firmware is required to manually acknowledge every received byte.
01 = 2 bytes received by slave. Hardware automatically acknowledges the first received byte. Firmware is required to manually acknowledge after the second received byte.
10 = 3 bytes received by slave. Hardware automatically acknowledges the first 2 received bytes. Firmware is required to manually acknowledge after the third received byte.
11 = 4 bytes received by slave. Hardware automatically acknowledges the first 3 received bytes. Firmware is required to manually acknowledge after the fourth received byte (Default)

Bit 20: MAN_CMD – Manual Command Acknowledgement Mode

0 = Slave automatically acknowledges received command code (Default)
1 = Data Request flag generated after receipt of command code, firmware required to issue ACK to continue message

Bit 19: TX_PEC – Asserted when the slave needs to send a PEC byte at end of message. PMBus Interface will transmit the calculated PEC byte after transmitting the number of data bytes indicated by TX Byte Cnt(Bits 19:17).

0 = No PEC byte transmitted (Default)
1 = PEC byte transmitted at end of current message

Bit 18-16: TX_COUNT– Number of valid bytes in Transmit Data Register

0 = No bytes valid (Default)
1 = One byte valid, Byte #0 (Bits 7:0 of Receive Data Register)
2 = Two bytes valid, Bytes #0 and #1 (Bits 15:0 of Receive Data Register)
3 = Three bytes valid, Bytes #0-2 (Bits 23:0 of Receive Data Register)
4 = Four bytes valid, Bytes #0-3 (Bits 31:0 of Receive Data Register)

Bit 15: PEC_ENA – PEC Processing Enable

0 = PEC processing disabled (Default)
1 = PEC processing enabled

Bit 14-8: SLAVE_MASK – Used in address detection, the slave mask enables acknowledgement of multiple device addresses by the slave. Writing a '0' to a bit within the slave mask enables the corresponding bit in the slave address to be either '1' or '0' and still allow for a match. Writing a '0' to all bits in the mask enables the PMBus Interface to acknowledge any device address. Upon power-up, the slave mask defaults to 7Fh, indicating the slave will only acknowledge the address programmed into the Slave Address (Bits 6-0).

Bit 7: MAN_SLAVE_ACK– Manual Slave Address Acknowledgement Mode

0 = Slave automatically acknowledges device address specified in SLAVE_ADDR, Bits 6-0 (Default)
1 = Enables the Manual Slave Address Acknowledgement Mode. Firmware is required to read received address and acknowledge on every message

Bits 6-0: SLAVE_ADDR – Configures the current device address of the slave. Used in automatic slave address acknowledge mode (default mode). The PMBus Interface will compare the received device address with the value stored in the Slave Address bits and the mask configured in the Slave Mask bits. If matching, the slave will acknowledge the device address.

9.8 I2C Hold Slave Address Register (I2CHSA)

Address FFF7E41C

Bit Number	7:1	0
Bit Name	SLAVE_ADDR	SLAVE_RW
Access	R	R
Default	-	-

Bits 7-1: SLAVE_ADDR – Stored device address acknowledged by the slave
Bit 0: SLAVE_RW – Stored R/W bit from address acknowledged by the slave
0 = Write Access
1 = Read Access

9.9 I2C Control Register 3 (I2CCTRL3)

Address FFF7E420

Bit Number					**23**
Bit Name					I2C_MODE_EN
Access					R/W
Default					0

Bit Number	**22**	**21**	**20**	**19**	**18**
Bit Name	MASTER_EN	SLAVE_EN	CLK_LO_DIS	IBIAS_B_EN	IBIAS_A_EN
Access	R/W	R/W	R/W	R/W	R/W
Default	0	1	0	0	0

Bit Number	**17**	**16**	**15**	**14**	**13**
Bit Name	SCL_DIR	SCL_VALUE	SCL_MODE	SDA_DIR	SDA_VALUE
Access	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0

Bit Number	**12**	**11**	**10**	**9**
Bit Name	SDA_MODE	CNTL_DIR	CNTL_VALUE	CNTL_MODE
Access	R/W	R/W	R/W	R/W
Default	0	0	0	0

Bit Number	**8**	**7**	**6**	**5**
Bit Name	ALERT_DIR	ALERT _VALUE	ALERT_MODE	CNTL_INT_EDGE
Access	R/W	R/W	R/W	R/W
Default	0	0	0	0

Bit Number	**4**	**3**	**2**	**1**	**0**
Bit Name	FAST_MODE_PLUS	FAST_MODE	BUS_LO_INT_EDGE	ALERT_EN	RESET
Access	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0

Bit 23: I2C_MODE_EN – I2C Mode Enable
0 = Enables PMBus interface capability (Default)
1 = Enables I2C interface capability

Bit 22: MASTER_EN – PMBus Master Enable
0 = Disables PMBus Master capability (Default)
1 = Enables PMBus Master capability

Bit 21: SLAVE_EN – PMBus Slave Enable
0 = Disables PMBus Slave capability
1 = Enables PMBus Slave capability (Default)

Bit 20: CLK_LO_DIS – Clock Low Timeout Disable
0 = Clock Low Timeout Enabled (Default)
1 = Clock Low Timeout Disabled

Bit 19: IBIAS_B_EN – PMBus Current Source B Control
0 = Disables Current Source for PMBUS address detection thru ADC (Default)
1 = Enables Current Source for PMBUS address detection thru ADC

Bit 18: IBIAS_A_EN – PMBus Current Source A Control
0 = Disables Current Source for PMBUS address detection thru ADC (Default)
1 = Enables Current Source for PMBUS address detection thru ADC
Bit 17: SCL_DIR – Configures direction of PMBus clock pin in GPIO mode
0 = PMBus clock pin configured as output (Default)
1 = PMBus clock pin configured as input
Bit 16: SCL_VALUE – Configures output value of PMBus clock pin in GPIO Mode
0 = PMBus clock pin driven low in GPIO Mode (Default)
1 = PMBus clock pin driven high in GPIO Mode
Bit 15: SCL_MODE – Configures mode of PMBus Clock pin
0 = PMBus clock pin configured in functional mode (Default)
1 = PMBus clock pin configured as GPIO
Bit 14: SDA_DIR – Configures direction of PMBus data pin in GPIO mode
0 = PMBus data pin configured as output (Default)
1 = PMBus data pin configured as input
Bit 13: SDA_VALUE – Configures output value of PMBus data pin in GPIO Mode
0 = PMBus data pin driven low in GPIO Mode (Default)
1 = PMBus data pin driven high in GPIO Mode
Bit 12: SDA_MODE – Configures mode of PMBus Data pin
0 = PMBus data pin configured in functional mode (Default)
1 = PMBus data pin configured as GPIO
Bit 11: CNTL_DIR – Configures direction of Control pin in GPIO mode
0 = Control pin configured as output (Default)
1 = Control pin configured as input
Bit 10: CNTL_VALUE – Configures output value of Control pin in GPIO Mode
0 = Control pin driven low in GPIO Mode (Default)
1 = Control pin driven high in GPIO Mode
Bit 9: CNTL_MODE – Configures mode of Control pin
0 = Control pin configured in functional mode (Default)
1 = Control pin configured as GPIO
Bit 8: ALERT_DIR – Configures direction of Alert pin in GPIO mode
0 = Control pin configured as output (Default)
1 = Control pin configured as input
Bit 7: ALERT_VALUE – Configures output value of Alert pin in GPIO Mode
0 = Alert pin driven low in GPIO Mode (Default)
1 = Alert pin driven high in GPIO Mode
Bit 6: ALERT_MODE – Configures mode of Alert pin
0 = Alert pin configured in functional mode (Default)
1 = Aler3 pin configured as GPIO
Bit 5: CNTL_INT_EDGE – Control Interrupt Edge Select
0 = Interrupt generated on falling edge of Control (Default)
1 = Interrupt generated on rising edge of Control

Bit 4: FAST_MODE_PLUS – Fast Mode Plus Enable
0 = Standard 100 KHz mode enabled (Default)
1 = Fast Mode Plus enabled (1MHz operation on PMBus)
Bit 3: FAST_MODE – Fast Mode Enable
0 = Standard 100 KHz mode enabled (Default)
1 = Fast Mode enabled (400KHz operation on PMBus)
Bit 2: BUS_LO_INT_EDGE – Clock Low Timeout Interrupt Edge Select
0 = Interrupt generated on rising edge of clock low timeout (Default)
1 = Interrupt generated on falling edge of clock low timeout
Bit 1: ALERT_EN – Slave Alert Enable
0 = PMBus Alert is not driven by slave, pulled up high on PMBus (Default)
1 = PMBus Alert driven low by slave

Bit 0: RESET – PMBus Interface Synchronous Reset
0 = No reset of internal state machines (Default)
1 = Control state machines are reset to initial states

10 SPI Reference

10.1 SPI Control Register (SPICTRL)

Address FFF7E800

Bit Number					**23:21**
Bit Name					CLKRATE
Access					R/W
Default					000

Bit Number	**20:16**	**15:11**	**10:7**	**6**	**5**
Bit Name	FRMLEN	RXCNT	TXCNT	WRSTORE	WRSTART
Access	R/W	R/W	R/W	R/W	R/W
Default	00000	00000	0000	0	0

Bit Number	**4**	**3**	**2**	**1**	**0**
Bit Name	POL	PHA	INTEN	MODE	SPIEN
Access	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0

Bit 23:21: CLKRATE – Master clock rate relative to ICLK
0 = SCK is ICLK/2 (Default)
1 = SCK is ICLK/4 (Default)
2 = SCK is ICLK/8 (Default)
3 = SCK is ICLK/16 (Default)
Bit 20:16: FRMLEN – Sets the number of messages (TXCNT + RXCNT) to hold CS low.
Bit 15:11: RXCNT – Sets the number bytes to receive after TXCNT bytes have been transmitted
Bit 10:7: TXCNT – Sets the bytes to transmit from the SPITX registers
Bit 6: WRSTORE – Places or discards data received during TXCNT
0 = Data received during TXCNT discarded (Default)
1 = Data received during TXCNT placed in RXBUF
Bit 5: WRSTART- Sets which WRREG initiates transfer
0 = Write to SPITX-0 starts message transfer (Default)
1 = Write to SPITX-1 starts message transfer
Bit 4: POL – The polarity bit, together with the phase bit, determines the transfer-mode.
Bit 3: PHA – The phase bit, together with the polarity bit, determines the transfer-mode.
Bit 2: INTEN – Enable interrupt generation to the CPU
0 = Disabled (Default)
1 = Enabled
Bit 1: MODE – Configures SPI mode

0 = Master Mode (Default)
1 = Slave mode
Bit 0: SPIEN – Enable for SPI Module
0 = Disabled (Default)
1 = Enabled

10.2 SPI Status Register (SPISTAT)

Address FFF7E804

Bit Number		**7:3**	**2**	**1**	**0**
Bit Name		FRMCNT	WRCOL	BUSY	SPIF
Access		R	R	R	R
jDefault		0	0	0	0

Bit 12:4: FRMCNT – Indicates the number of messages remaining in the FRMLEN before SCS will go inactive.
Bit 2: WCOL – SPI interface is busy – only accurate in very simple cases
Bit 1: BUSY – SPI interface is busy – only accurate in very simple cases
Bit 0: SPIF – SPI Flag, set when current message is complete. Write a 1 to this bit to clear it.

10.3 SPI Pin Function Register (SPIFUNC)

Address FFF7E808

Bit Number		**3**	**2**	**1**	**0**
Bit Name		MISO	MOSI	SCS	SCK
Access		R/W	R/W	R/W	R/W
Default		0	0	0	0

Bit 3: MISO – Selects SPI or GPIO function for SPI-MISO pin
0 = SPI function (Default)
1 = GPIO function
Bit 2: MOSI – Selects SPI or GPIO function for SPI-MOSI pin
0 = SPI function (Default)
1 = GPIO function
Bit 1: SCS – Selects SPI or GPIO function for SPI-CS pin
0 = SPI function (Default)
1 = GPIO function
Bit 0: SCK – Selects SPI or GPIO function for SPI-CK pin
0 = SPI function (Default)
1 = GPIO function

10.4 SPI Pin Direction Register (SPIDIR)

Address FFF7E80C

Bit Number		**3**	**2**	**1**	**0**
Bit Name		MISO	MOSI	SCS	SCK
Access		R/W	R/W	R/W	R/W
Default		0	0	0	0

Bit 3: MISO – Selects direction for SPI-MISO pin in GPIO mode
0 = Output (Default)
1 = Input
Bit 2: MOSI – Selects direction for SPI-MOSI pin in GPIO mode

0 = Output (Default)
1 = Input
Bit 1: SCS – Selects direction for SPI-CS pin in GPIO mode
0 = Output (Default)
1 = Input
Bit 0: SCK – Selects direction for SPI-CS pin in GPIO mode
0 = Output (Default)
1 = Input

10.5 SPI Pin GP Out Register (SPIGPOUT)

Address FFF7E810

Bit Number		**3**	**2**	**1**	**0**
Bit Name		MISO	MOSI	SCS	SCK
Access		R/W	R/W	R/W	R/W
Default		0	0	0	0

Bit 3: MISO – Selects value for SPI-MISO pin in GPIO output mode
0 = Pin driven low (Default)
1 = Pin driven high
Bit 2: MOSI – Selects value for SPI-MOSI pin in GPIO output mode
0 = Pin driven low (Default)
1 = Pin driven high
Bit 1: SCS – Selects value for SPI-CS pin in GPIO output mode
0 = Pin driven low (Default)
1 = Pin driven high
Bit 0: SCK – Selects value for SPI-CS pin in GPIO output mode
0 = Pin driven low (Default)
1 = Pin driven high

10.6 SPI Pin GP In Register (SPIGPIN)

Address FFF7E814

Bit Number		**3**	**2**	**1**	**0**
Bit Name		MISO	MOSI	SCS	SCK
Access		R	R	R	R
Default		0	0	0	0

Bit 3: MISO – Selects value for SPI-MISO pin in GPIO output mode
0 = Pin driven low (Default)
1 = Pin driven high
Bit 2: MOSI – Selects value for SPI-MOSI pin in GPIO output mode
0 = Pin driven low (Default)
1 = Pin driven high
Bit 1: SCS – Selects value for SPI-CS pin in GPIO output mode
0 = Pin driven low (Default)
1 = Pin driven high
Bit 0: SCK – Selects value for SPI-CS pin in GPIO output mode
0 = Pin driven low (Default)
1 = Pin driven high

10.7 SPI TX Buffer Register (SPITX0)

Address FFF7E818

Bit Number	**31:0**

Bit Name	DATA
Access	R/W
Default	0

Bits 31:0: Data to be transmitted by SPI interface

10.8 SPI TX Buffer Register (SPITX1)

Address FFF7E81C

Bit Number	**31:0**
Bit Name	DATA
Access	R/W
Default	0

Bits 31:0: Data to be transmitted by SPI interface

10.9 SPI Read Buffer Register (SPIRX0)

Address FFF7E820

Bit Number	**31:0**
Bit Name	DATA
Access	R
Default	0

Bits 31:0: Data received by SPI interface

10.10 SPI Read Buffer Register (SPIRX1)

Address FFF7E824

Bit Number	**31:0**
Bit Name	DATA
Access	R
Default	0

Bits 31:0: Data received by SPI interface

10.11 SPI Read Buffer Register (SPIRX2)

Address FFF7E828

Bit Number	**31:0**
Bit Name	DATA
Access	R
Default	0

Bits 31:0: Data received by SPI interface

11 IOMUX Reference

The IOMUX register in the Misc Analog Control peripheral is one of the very few registers to be changed between the UCD3138 and UCD3138064.

The FAULT_01_MUX_SEL bitfield is added to permit the use of Fault 0 and Fault 1 as SPI or I2C pins.

The JTAG_DATA_MUX_SEL field is enhanced to permit the use of TDO and TDI pins for SPI.

11.1 I/O Mux Control Register (IOMUX)

Address FFF7F030

Bit Number			**12:11**
Bit Name			FAULT_01 _MUX_SEL
Access			R/W
Default			00

Bit Number	**10:9**	**8:7**	**6:4**
Bit Name	EXT_TRIG_MUX_SEL	JTAG_CLK_MUX_SEL	JTAG_DATA_MUX_SEL
Access	R/W	R/W	R/W
Default	00	10	000

Bit Number	**3:2**	**1**	**0**
Bit Name	SYNC_MUX_SEL	UART_MUX_SEL	PMBUS_MUX_SEL
Access	R/W	R/W	R/W
Default	00	0	0

Bits 12-11: FAULT_01_MUX_SEL – Fault 0 and 1 Pin Mux Select

I/O Pin	**0**	**1**	**2**	**3**
FAULT-0	FAULT-0	SPI-CS	I2C-DATA	
FAULT-1	FAULT-1	SPI-CLK	I2C-CLK	

Bits 10-9: EXT_TRIG_MUX_SEL – EXT_TRIG Pin Mux Select

I/O Pin	**0**	**1**	**2**	**3**
EXT_TRIG	EXT_TRIG	TCAP	SYNC	PWM-0

Bits 8-7: JTAG_CLK_MUX_SEL – TCK Pin Mux Select

I/O Pin	**0**	**1**	**2**	**3**
TCK	TCK	TCAP	SYNC	PWM-0

Bits 6-4: JTAG_DATA_MUX_SEL – TDO/TDI Pin Mux Select

I/O Pin	**0**	**1**	**2**	**3**	**4**
TDO	TDO	SCI_TX-0	ALERT	FAULT-0	SPI-MOSI
TDI	TDI	SCI_RX-0	CONTROL	FAULT-1	SPI-MISO

Bits 3-2: SYNC_MUX_SEL – SYNC Pin Mux Select :w

I/O Pin	**0**	**1**	**2**	**3**
SYNC	SYNC	TCAP	EXT_TRIG	PWM-0

Bit 1: UART_MUX_SEL – SCL/SDA Pins Mux Select

I/O Pin	**0**	**1**
SCI_TX-1	SCI_TX-1	ALERT
SCI_RX-1	SCI_RX-1	CONTROL

Bit 0: PMBUS_MUX_SEL – SCL/SDA Pins Mux Select

I/O Pin	**0**	**1**

SCL	SCL	SCI_TX-0
SDA	SDA	SCI_RX-0

12 DEC-Address Manager Reference

The DEC generates the memory selects and SAR peripheral select signals by decoding the address and control signals from the ARM processor. In addition, the DEC provides the control signals for the Program and Data Flash.

The assigned memory selects for Cyclone are as follows:

Memory Select 0 => Boot ROM (1Kx32)
Memory Select 1 => Program Flash 1(8Kx32)
Memory Select 2 => Data Flash (512x32)
Memory Select 3 => Data RAM (1Kx32)
Memory Select 4 => Loop Mux (1Kx32)
Memory Select 5 => Fault Mux (1Kx32)
Memory Select 6 => ADC12 Control (1Kx32)
Memory Select 7 => DPWM3 (1Kx32)
Memory Select 8 => Filter 2 (1Kx32)
Memory Select 9 => DPWM 2 (1Kx32)
Memory Select 10 => Front End Control 2 (1Kx32)
Memory Select 11 => Filter 1 (1Kx32)
Memory Select 12 => DPWM 1 (1Kx32)
Memory Select 13 => Front End Control 1 (1Kx32)
Memory Select 14 => Filter 0 (1Kx32)
Memory Select 15 => DPWM 0 (1Kx32)
Memory Select 16 => Front End Control 0 (1Kx32)
Memory Select 17 => Program Flash 2(8Kx32)

12.1 Memory Fine Base Address High Register 0 (MFBAHR0)

Address FFFFFE00

Bit Number	15:0
Bit Name	ADDRESS[31:16]
Access	R/W
Default	0000_0000_0000_0000

Bits 15-0: ADDRESS[31:16] – 16 Most Significant Bits of the Base Address. The Base Address sets the 22 most significant bits of the memory address.

12.2 Memory Fine Base Address Low Register 0 (MFBALR0)

Address FFFFFE04

Bit Number	15:10	8	7:4	1	0
Bit Name	ADDRESS[15:10]	MS	BLOCK_SIZE	RONLY	PRIV
Access	R/W	R/W	R/W	R/W	R/W
Default	000000	0	0000	0	0

Bits15-10: ADDRESS[15:10] – 6 Least Significant Bits of the Base Address. The Base Address sets the 22 most significant bits of the memory address.
Bit 8: MS – Memory Map Select
0 = Memory Map configuration not updated (Default)
1 = Enables the fine and coarse memory selects and activates the memory map
Bits 7-4: BLOCK_SIZE – Configures the size of the memory
0000 = Memory select is disabled (Default)
0001 = 1K Bytes
0010 = 2K Bytes
0011 = 4K Bytes
0100 = 8K Bytes
0101 = 16K Bytes
0110 = 32K Bytes
0111 = 64K Bytes
1000 = 128K Bytes
1001 = 256K Bytes
1010 = 512K Bytes
1011 = 1M Bytes
1100 = 2M Bytes
1101 = 4M Bytes
1110 = 8M Bytes
1111 = 16M Bytes
Bit 1: RONLY – Read-only protection. This bit sets read-only protection for the memory selected by the memory select. An illegal access exception is generated when a write is attempted to the memory.
0 = Read/write access to memory (Default)
1 = Read accesses to memory only
Bit 0: PRIV – Privilege mode protection. This bit sets privilege mode protection for the memory Registration selected by the memory select. An illegal access exception is generated on any access to memory protected by privilege mode.
0 = User/privilege mode accesses to memory (Default)
1 = Privilege mode accesses to memory only

12.3 Memory Fine Base Address High Register 1-3,17 (MFBAHRx)

Address FFFFFE08 – Memory Fine Base Address High Register 1
Address FFFFFE10– Memory Fine Base Address High Register 2
Address FFFFFE18 – Memory Fine Base Address High Register 3
Address FFFFFE88 – Memory Fine Base Address High Register 17

Bit Number	15:0
Bit Name	ADDRESS[31:16]
Access	R/W
Default	0000_0000_0000_0000

Bits 15-0: ADDRESS[31:16] – 16 Most Significant Bits of the Base Address. The Base Address sets the 22 most significant bits of the memory address.

12.4 Memory Fine Base Address Low Register 1-3,17(MFBALRx)

Address FFFFFE0C – Memory Fine Base Address Low Register 1
Address FFFFFE14 – Memory Fine Base Address Low Register 2
Address FFFFFE1C – Memory Fine Base Address Low Register 3
Address FFFFFE8C – Memory Fine Base Address Low Register 17

Bit Number	15:10	9	7:4	1	0
Bit Name	ADDRESS[15:10]	AW	BLOCK_SIZE	RONLY	PRIV
Access	R/W	R/W	R/W	R/W	R/W
Default	000000	0	0000	0	0

Bits 15-10: ADDRESS[15:10] – 6 Least Significant Bits of the Base Address. The Base Address sets the 22 most significant bits of the memory address.
Bit 9: AW – Auto-wait-on-write. When this bit is set, any write operation on this memory select takes two system cycles.

0 = Write operation is not supplemented with an additional cycle (Default)
1 = Write operation takes an additional cycle

Bits 7-4: BLOCK_SIZE – Configures the size of the memory

0000 = Memory select is disabled (Default)
0001 = 1K Bytes
0010 = 2K Bytes
0011 = 4K Bytes
0100 = 8K Bytes
0101 = 16K Bytes
0110 = 32K Bytes
0111 = 64K Bytes
1000 = 128K Bytes
1001 = 256K Bytes
1010 = 512K Bytes
1011 = 1M Bytes
1100 = 2M Bytes
1101 = 4M Bytes
1110 = 8M Bytes
1111 = 16M Bytes

Bit 1: RONLY – Read-only protection. This bit sets read-only protection for the memory selected by the memory select. An illegal access exception is generated when a write is attempted to the memory.

0 = Read/write access to memory (Default)
1 = Read accesses to memory only

Bit 0: PRIV – Privilege mode protection. This bit sets privilege mode protection for the memory Registration selected by the memory select. An illegal access exception is generated on any access to memory protected by privilege mode.

0 = User/privilege mode accesses to memory (Default)
1 = Privilege mode accesses to memory only

12.5 Memory Fine Base Address High Register 4 (MFBAHR4)

Address FFFFFE20 – Memory Fine Base Address High Register 4

Bit Number	**15:0**
Bit Name	ADDRESS[31:16]
Access	R/W
Default	0000_0000_0001_0010

Bits 15-0: ADDRESS[31:16] – 16 Most Significant Bits of the Base Address. The Base Address sets the 22 most significant bits of the memory address.

12.6 Memory Fine Base Address Low Register 4-16 (MFBALRx)

Address FFFFFE24 – Memory Fine Base Address Low Register 4
Address FFFFFE2C – Memory Fine Base Address Low Register 5
Address FFFFFE34 – Memory Fine Base Address Low Register 6
Address FFFFFE3C – Memory Fine Base Address Low Register 7
Address FFFFFE44 – Memory Fine Base Address Low Register 8
Address FFFFFE4C – Memory Fine Base Address Low Register 9
Address FFFFFE54 – Memory Fine Base Address Low Register 10
Address FFFFFE5C – Memory Fine Base Address Low Register 11
Address FFFFFE64 – Memory Fine Base Address Low Register 12
Address FFFFFE6C – Memory Fine Base Address Low Register 13
Address FFFFFE74 – Memory Fine Base Address Low Register 14
Address FFFFFE7C – Memory Fine Base Address Low Register 15
Address FFFFFE84 – Memory Fine Base Address Low Register 16

Bit Number	**15:10**	**9**	**8:2**	**1**	**0**
Bit Name	ADDRESS[15:10]	AW	RESERVED	RONLY	PRIV
Access	R/W	R/W	-	R/W	R/W
Default	000000	0	0_0000_00	0	0

Bits 15-10: ADDRESS[15:10] – 6 Least Significant Bits of the Base Address. The Base Address sets the 22 most significant bits of the memory address.
Bit 9: AW – Auto-wait-on-write. When this bit is set, any write operation on this memory select takes two system cycles.
0 = Write operation is not supplemented with an additional cycle (Default)
1 = Write operation takes an additional cycle
Bits 8-2: RESERVED – Unused bits
Bit 1: RONLY – Read-only protection. This bit sets read-only protection for the memory selected by the memory select. An illegal access exception is generated when a write is attempted to the memory.
0 = Read/write access to memory (Default)
1 = Read accesses to memory only
Bit 0: PRIV – Privilege mode protection. This bit sets privilege mode protection for the memory Registration selected by the memory select. An illegal access exception is generated on any access to memory protected by privilege mode.
0 = User/privilege mode accesses to memory (Default)
1 = Privilege mode accesses to memory only

12.7 Memory Fine Base Address High Register 5 (MFBAHR5)

Address FFFFFE28 – Memory Fine Base Address High Register 5

Bit Number	15:0
Bit Name	ADDRESS[31:16]
Access	R/W
Default	0000_0000_0001_0011

Bits 15-0: ADDRESS[31:16] – 16 Most Significant Bits of the Base Address. The Base Address sets the 22 most significant bits of the memory address.

12.8 Memory Fine Base Address High Register 6 (MFBAHR6)

Address FFFFFE30 – Memory Fine Base Address High Register 6

Bit Number	15:0
Bit Name	ADDRESS[31:16]
Access	R/W
Default	0000_0000_0001_0100

Bits 15-0: ADDRESS[31:16] – 16 Most Significant Bits of the Base Address. The Base Address sets the 22 most significant bits of the memory address.

12.9 Memory Fine Base Address High Register 7 (MFBAHR7)

Address FFFFFE38 – Memory Fine Base Address High Register 7

Bit Number	15:0
Bit Name	ADDRESS[31:16]
Access	R/W
Default	0000_0000_0001_0101

Bits 15-0: ADDRESS[31:16] – 16 Most Significant Bits of the Base Address. The Base Address sets the 22 most significant bits of the memory address.

12.10 Memory Fine Base Address High Register 8 (MFBAHR8)

Address FFFFFE40 – Memory Fine Base Address High Register 8

Bit Number	15:0
Bit Name	ADDRESS[31:16]
Access	R/W
Default	0000_0000_0001_0110

Bits 15-0: ADDRESS[31:16] – 16 Most Significant Bits of the Base Address. The Base Address sets the 22 most significant bits of the memory address.

12.11 Memory Fine Base Address High Register 9 (MFBAHR9)

Address FFFFFE48 – Memory Fine Base Address High Register 9

Bit Number	**15:0**
Bit Name	ADDRESS[31:16]
Access	R/W
Default	0000_0000_0001_0111

Bits 15-0: ADDRESS[31:16] – 16 Most Significant Bits of the Base Address. The Base Address sets the 22 most significant bits of the memory address.

12.12 Memory Fine Base Address High Register 10 (MFBAHR10)

Address FFFFFE50 – Memory Fine Base Address High Register 10

Bit Number	**15:0**
Bit Name	ADDRESS[31:16]
Access	R/W
Default	0000_0000_0001_1000

Bits 15-0: ADDRESS[31:16] – 16 Most Significant Bits of the Base Address. The Base Address sets the 22 most significant bits of the memory address.

12.13 Memory Fine Base Address High Register 11 (MFBAHR11)

Address FFFFFE58 – Memory Fine Base Address High Register 11

Bit Number	**15:0**
Bit Name	ADDRESS[31:16]
Access	R/W
Default	0000_0000_0001_1001

Bits 15-0: ADDRESS[31:16] – 16 Most Significant Bits of the Base Address. The Base Address sets the 22 most significant bits of the memory address.

12.14 Memory Fine Base Address High Register 12 (MFBAHR12)

Address FFFFFE60 – Memory Fine Base Address High Register 12

Bit Number	**15:0**
Bit Name	ADDRESS[31:16]
Access	R/W
Default	0000_0000_0001_1010

Bits 15-0: ADDRESS[31:16] – 16 Most Significant Bits of the Base Address. The Base Address sets the 22 most significant bits of the memory address.

12.15 Memory Fine Base Address High Register 13 (MFBAHR13)

Address FFFFFE68 – Memory Fine Base Address High Register 13

Bit Number	15:0
Bit Name	ADDRESS[31:16]
Access	R/W
Default	0000_0000_0001_1011

Bits 15-0: ADDRESS[31:16] – 16 Most Significant Bits of the Base Address. The Base Address sets the 22 most significant bits of the memory address.

12.16 Memory Fine Base Address High Register 14 (MFBAHR14)

Address FFFFFE70 – Memory Fine Base Address High Register 14

Bit Number	15:0
Bit Name	ADDRESS[31:16]
Access	R/W
Default	0000_0000_0001_1100

Bits 15-0: ADDRESS[31:16] – 16 Most Significant Bits of the Base Address. The Base Address sets the 22 most significant bits of the memory address.

12.17 Memory Fine Base Address High Register 15 (MFBAHR15)

Address FFFFFE78 – Memory Fine Base Address High Register 15

Bit Number	15:0
Bit Name	ADDRESS[31:16]
Access	R/W
Default	0000_0000_0001_1101

Bits 15-0: ADDRESS[31:16] – 16 Most Significant Bits of the Base Address. The Base Address sets the 22 most significant bits of the memory address.

12.18 Memory Fine Base Address High Register 16 (MFBAHR16)

Address FFFFFE80 – Memory Fine Base Address High Register 16

Bit Number	15:0
Bit Name	ADDRESS[31:16]
Access	R/W
Default	0000_0000_0001_1110

Bits 15-0: ADDRESS[31:16] – 16 Most Significant Bits of the Base Address. The Base Address sets the 22 most significant bits of the memory address.

12.19 Program Flash Control Register 1(PFLASHCTRL1)

Address FFFFFE90

Bit Number	11	10	9	8	7:5	4:0
Bit Name	BUSY	Reserved	PAGE_ERASE	MASS_ERASE	RESERVED	PAGE_SEL
Access	R	R/W	R/W	R/W	R/W	R/W

Default	-	Leave as 0	0	0	000	00000

Bit 11: BUSY – Program Flash Busy Indicator
0 = Program Flash available for read/write/erase access
1 = Program Flash unavailable for read/write/erase access
Bit 10: Reserved – leave as a 0
Bit 9: PAGE_ERASE – Program Flash Page Erase Enable
0 = No Page Erase initiated on Program Flash (Default)
1 = Page Erase on Program Flash enabled. Page erased is based on PAGE_SEL (Bits 4-0). Interlock Key must be set in Program Flash Interlock Register (Section 11.21) to initiate Page Erase cycle. This bit is cleared upon completion of Page Erase cycle.
Bit 8: MASS_ERASE – Program Flash Mass Erase Enable
0 = No Mass Erase initiated on Program Flash (Default)
1 = Mass Erase of Program Flash enabled. Interlock Key must be set in Program Flash Interlock Register (Section 11.21) to initiate Mass Erase cycle. This bit is cleared upon completion of Mass Erase cycle.
Bits 4-0: PAGE_SEL – Selects page to be erased during Page Erase Cycle

12.20 Data Flash Control Register (DFLASHCTRL)

Address FFFFFE94

Bit Number	11	10	9	8	7:6	5:0
Bit Name	BUSY	Reserved	PAGE_ERASE	MASS_ERASE	RESERVED	PAGE_SEL
Access	R	R/W	R/W	R/W	R/W	R/W
Default	-	Leave as 0	0	0	00	000000

Bit 11: BUSY – Data Flash Busy Indicator
0 = Data Flash available for read/write/erase access
1 = Data Flash unavailable for read/write/erase access
Bit 10: Reserved – leave as a 0
Bit 9: PAGE_ERASE – Data Flash Page Erase Enable
0 = No Page Erase initiated on Data Flash (Default)
1 = Page Erase Cycle on Data Flash enabled. Page erased is based on PAGE_SEL (Bits 4-0). This bit is cleared upon completion of Page Erase cycle.
Bit 8: MASS_ERASE – Data Flash Mass Erase Enable
0 = No Mass Erase initiated on Data Flash (Default)
1 = Mass Erase of Data Flash enabled. Bit is cleared upon completion of mass erase.
Bits 5-0: PAGE_SEL – Selects page to be erased during Page Erase Cycle

12.21 Flash Interlock Register (FLASHILOCK)

Address FFFFFE98

Bit Number	31:0
Bit Name	INTERLOCK_KEY
Access	R/W
Default	0000_0000_0000_0000_0000_0000_0000_0000

Bit 31-0: INTERLOCK_KEY – Flash Interlock Key. Register must be set to 0x42DC157E prior to every Program Flash#1 or Data Flash write, Mass Erase or Page Erase or 0x6C97D0C5 prior to every Program Flash#2 write, Mass Erase or Page Erase. If the Interlock Key is not set, the write/erase cycle to the Flash will not initiate. This register will clear upon the completion of a write or erase cycle to the Flash modules.

12.22 Program Flash #2 Control Register (PFLASHCTRL2)

Address FFFFFE9C

Bit Number	11	10	9	8	7:5	4:0
Bit Name	BUSY	Reserved	PAGE_ERASE	MASS_ERASE	RESERVED	PAGE_SEL
Access	R	R/W	R/W	R/W	R/W	R/W
Default	-	Leave as 0	0	0	000	00000

Bit 11: BUSY – Program Flash Busy Indicator
0 = Program Flash available for read/write/erase access
1 = Program Flash unavailable for read/write/erase access
Bit 10: Reserved – leave as a 0
Bit 9: PAGE_ERASE – Program Flash Page Erase Enable
0 = No Page Erase initiated on Program Flash (Default)
1 = Page Erase on Program Flash enabled. Page erased is based on PAGE_SEL (Bits 4-0). Interlock Key must be set in Program Flash Interlock Register (Section 11.21) to initiate Page Erase cycle. This bit is cleared upon completion of Page Erase cycle.
Bit 8: MASS_ERASE – Program Flash Mass Erase Enable
0 = No Mass Erase initiated on Program Flash (Default)
1 = Mass Erase of Program Flash enabled. Interlock Key must be set in Program Flash Interlock Register (Section 11.21) to initiate Mass Erase cycle. This bit is cleared upon completion of Mass Erase cycle.
Bits 4-0: PAGE_SEL – Selects page to be erased during Page Erase Cycle

13 DPWMCTRL2 Reference

The BLANK_PCM_EN bit is added to DPWMCTRL2 for Cyclone 64. For a description, see Section 7 Enhanced Peak Current Mode Blanking.

13.1 DPWM Control Register 2 (DPWMCTRL2)

Address 00150008 – DPWM 3 Control Register 2
Address 00170008 – DPWM 2 Control Register 2
Address 001A0008 – DPWM 1 Control Register 2
Address 001D0008 – DPWM 0 Control Register 2

Bit Number			16
Bit Name			BLANK_PCM_EN
Access			R/W
Default			0

Bit Number	15:12	11	10
Bit Name	SYNC_IN_DIV_RATIO	PEAK_CUR_HR_CTRL	RESON_DEADTIME_COMP_EN
Access	R/W	R/W	R/W
Default	0000	0	0

Bit Number	9:8	7	6
Bit Name	FILTER_DUTY_SEL	IDE_DUTY_B_EN	IDE_DETECT_EN
Access	R/W	R/W	R/W
Default	00	0	0

Bit Number	5:4	3:2
Bit Name	SAMPLE_TRIG1_OVERSAMPLE	SAMPLE_TRIG1_MODE
Access	R/W	R/W
Default	00	00

Bit Number	1	0
Bit Name	SAMPLE_TRIG_2_EN	SAMPLE_TRIG_1_EN
Access	R/W	R/W
Default	0	1

Bit 16: BLANK_PCM_EN – Comparator Blanking Window B Enable for PCM
0 = Comparator Blanking A Window Disabled (Default)
1 = Comparator Blanking A Window for PWM-B Enabled

Bits 15-12: SLAVE_SYNC_IN_DIV_RATIO – Sets the number of syncs to be masked before a resync

Bit 11: PEAK_CUR_HR_CTRL – Peak Current High Resolution Control
0 = Disables high resolution functionality during Peak Current Mode (Default)
1 = Enables high resolution functionality during Peak Current Mode

Bit 10: RESON_DEADTIME_COMP_EN – Sets the method at which High Side CLA-Duty is used in calculations
0 = CLA Duty from Filter (Default)
1 = CLA Duty from Filter minus deadtime adjustment

Bits 9-8: FILTER_DUTY_SEL – Sets which register is used for the max duty calculation at the Filter in RESON and MESH modes.
0 = PWM Period Register (Default)
1 = Event 2
2 = PWM Period Adjust Register (Bits 13:0)

Bit 7: IDE_DUTY_B_EN – IDE Duty Cycle Side B Enable
0 = Disabled (Default)

1 = Enabled

Bit 6: IDE_DETECT_EN – IDE Detect Enable

0 = Disabled (Default)

1 = Enabled

Bits 5-4: SAMPLE_TRIG1_OVERSAMPLE – Oversample Select for Sample Trigger 1

00 = Trigger an EADC Sample at PWM Sample Trig Register value (Default)

01 = Trigger an EADC Sample at PWM Sample Trig Register value and at PWM Sample Trig Register value divided by 2

10 = Trigger a EADC Sample at PWM Sample Trig Register value, at PWM Sample Trig Register value divided by 2 and at PWM Sample Trig Register value divided by 4

11 = Trigger a EADC Sample at PWM Sample Trig Register value, at PWM Sample Trig Register value divided by 2, at PWM Sample Trig Register value divided by 4 and at PWM Sample Trig Register value divided by 8

Bits 3-2: SAMPLE_TRIG1_MODE – Mode select for Sample Trigger 1

00 = Trigger value is set using PWM Sample Trig Register value (Default)

01 = Trigger value is adaptive midpoint (EV1+CLA_DUTY/2 + Adaptive Offset) and uses current CLA value at update event

10 = Trigger value is adaptive midpoint (EV1+CLA_DUTY/2 + Adaptive Offset) and uses previous CLA value at update event

11 = Trigger value is adaptive midpoint (EV1+CLA_DUTY + Fixed offset + Adaptive Offset) and uses current CLA value at update event

Bit 1: SAMPLE_TRIG_2_EN – Sample Trigger 2 Enable

0 = Disable Sample Trigger 2 (Default)

1 = Enable Sample Trigger 2

Bit 0: SAMPLE_TRIG_1_EN – Sample Trigger 1 Enable

0 = Disable Sample Trigger 1 (Default)

1 = Enable Sample Trigger 1

14 Converting UCD3138 programs to UCD3138064

The main consideration in converting programs from UCD3138 to UCD3138064 is related to the changes in the memory map. There are also a few changes necessary because of the addition of an additional FLASH bank, and the related changes to register names for FLASH control.

14.1 Change linker addresses

To change the linker addresses, replace the following files in the program:

Old file	New File
Cyclone.cmd	Cyclone_64.cmd
Cyclone_headers.cmd	Cyclone_64_headers.cmd
cyclone_global_variables_defs.c	cyclone_64_global_variables_defs.c

14.2 Change header files which define peripherals

There are three header files which change. They are:

- Cyclone_dec.h
- Cyclone_dpwm.h
- Cyclone_misc_analog.h

The new versions are called Cyclone_64_...h.

The TI EVM codes generally have a:

#include "cyclone_device.h"

statement in each .c file.

Cyclone_device.h has #include statements to include all of the header files for the peripherals. There are two different ways to change the header files to match Cyclone 64.

1. Replace all #include "cyclone_device.h" with #include "cyclone_64_device.h" This involves editing most of the .c files or
2. Make a new cyclone_device.h file which includes the new cyclone 64 files. This may be a bit more confusing, but it removes the need to edit the #include statements in all the .c files. Either .h file should also include cyclone_64_constants.h

14.3 Changes to the Flash Control Registers

The changes to the flash control registers are described in Section 6, 21Register Changes for Program Flash Block 2. For programs which use the full features of the UCD3138064, read Section 6, and study reference codes from TI.

For a quick test of a 32K byte or less code, replace DecRegs.PFLASHCTRL with DecRegs.PFLASHCTRL1 wherever it occurs in the code.

14.4 Set Blank_PCM_EN for Peak Current Mode

For Peak Current Mode, the BLANK_PCM_EN bit should be set in the appropriate DPWMCTRL2 registers that provide blanking for PCM.

PCM blanking provides an earlier blanking than the blanking using the Blank A enable bit on DPWM modules when used for PCM. For more information, see Section 7, Enhanced Peak Current Mode Blanking.

14.5 Update Parm Info/Parm Value Pointers

The parm info/parm value PMBus commands are used by the GUI. They support memory accesses for the memory debugger and some of the filter design functions. The GUI is designed for a standard set of pointers. There is a list of memory areas. Each memory area has a start pointer and a length. The GUI accesses each memory area by sending the number for the area, and the offset within that area.

There are areas for RAM, DFLASH, PFLASH, one for each fast peripheral, and one for the block of slow peripherals at the end of the memory space. For conversion to the UCD3138064, it is necessary to move some blocks to new addresses. In addition, the second flash block gets its own pointer value.

14.5.1 Changes in pmbus.h

It is necessary to change the parm info/parm value pointers in pmbus.h. The changes are marked in bold, italic, underline, and red:

```
//  Memory limits used by the PARM_INFO and PARM_VALUE commands.
#define RAM_START_ADDRESS  0x00069000      // Beginning of RAM
#define RAM_END_ADDRESS             0x00069FFF      // End of RAM
#define RAM_LENGTH                  (RAM_END_ADDRESS - RAM_START_ADDRESS + 1)


// Allow access to peripherals, but not core ARM regs.
#define REG_START_ADDRESS   0xFFF7E400      // Beginning of Register space
#define REG_END_ADDRESS             0xFFF7fdff      // End of Register space
#define REG_LENGTH                  (REG_END_ADDRESS - REG_START_ADDRESS + 1)

// Allow read-only access to Data Flash
#define DFLASH_START_ADDRESS        0x00068800      // Beginning of DFLASH
#define DFLASH_END_ADDRESS          0x00068FFF      // End of DFLASH
#define DFLASH_LENGTH                       (DFLASH_END_ADDRESS - DFLASH_START_ADDRESS +
1)

// Allow read-only access to Constants in Program Flash
#define PFLASH_1_CONST_START_ADDRESS       0x00000000      // Beginning of PFLASH_1 Constants
#define PFLASH_1_CONST_END_ADDRESS         0x00007FFF      // End of PFLASH Constants
#define PFLASH_1_CONST_LENGTH                       (((unsigned
short)PFLASH_1_CONST_END_ADDRESS) - PFLASH_1_CONST_START_ADDRESS + 1)

// Allow read-only access to Program in Program Flash
#define PFLASH_1_PROG_START_ADDRESS        0x00000000      // Beginning of PFLASH Program
#define PFLASH_1_PROG_END_ADDRESS          0x00007FFF      // End of PFLASH Program
#define PFLASH_1_PROG_LENGTH               (((unsigned short)PFLASH_1_PROG_END_ADDRESS) -
PFLASH_1_PROG_START_ADDRESS + 1)

 //fast peripherals
#define LOOP_MUX_START_ADDRESS  0x00120000
#define LOOP_MUX_LENGTH       0x00000078

#define FAULT_MUX_START_ADDRESS 0x00130000
#define FAULT_MUX_LENGTH       0x00000080

#define ADC_START_ADDRESS      0x00140000
#define ADC_LENGTH           0x00000098

#define DPWM3_START_ADDRESS     0x00150000
#define DPWM3_LENGTH          0x0000008c

#define FILTER2_START_ADDRESS   0x00160000
#define FILTER2_LENGTH        0x00000064
```

```
#define DPWM2_START_ADDRESS    0x00170000
#define DPWM2_LENGTH           0x0000008c

#define FE_CTRL2_START_ADDRESS  0x00180000
#define FE_CTRL2_LENGTH         0x00000044

#define FILTER1_START_ADDRESS   0x00190000
#define FILTER1_LENGTH          0x00000064

#define DPWM1_START_ADDRESS    0x001a0000
#define DPWM1_LENGTH           0x0000008c

#define FE_CTRL1_START_ADDRESS  0x001b0000
#define FE_CTRL1_LENGTH         0x00000044

#define FILTER0_START_ADDRESS   0x001c0000
#define FILTER0_LENGTH          0x00000064

#define DPWM0_START_ADDRESS    0x001d0000
#define DPWM0_LENGTH           0x0000008c

#define FE_CTRL0_START_ADDRESS  0x001e0000
#define FE_CTRL0_LENGTH         0x00000044

#define SYSTEM_REGS_START_ADDRESS 0xfffffd00
#define SYSTEM_REGS_LENGTH      0x2d0

// Allow read-only access to Program in Program Flash2
#define PFLASH_2_PROG_START_ADDRESS      0x00008000     // Beginning of PFLASH Program
#define PFLASH_2_PROG_END_ADDRESS        0x0000FFFF     // End of PFLASH Program
#define PFLASH_2_PROG_LENGTH             (((unsigned short)PFLASH_2_PROG_END_ADDRESS) -
PFLASH_2_PROG_START_ADDRESS + 1)


#define NUM_MEMORY_SEGMENTS     20      // 20 memory segments for Cyclone 64
```

14.5.2 Changes in Parm Info/Parm Value File

It is also necessary to add start and length for pflash 2 to the lists in the c file with parm info/parm value in it. These may be in different files in different codes.

```
SYSTEM_REGS_START_ADDRESS,
PFLASH_2_PROG_START_ADDRESS};
SYSTEM_REGS_LENGTH,
PFLASH_2_PROG_LENGTH    };
```

In the same list, change the PFLASH names to PFLASH_1:

```
PFLASH_1_CONST_START_ADDRESS,
```

PFLASH_1_PROG_START_ADDRESS,
PFLASH_1_CONST_LENGTH,
PFLASH_1_PROG_LENGTH,

Generally the table holding the above, will have some conditional compilation based on NUM_MEMORY_SEGMENTS. The if-statements can be removed, and the code which was compiled with NUM_MEMORY_SEGMENTS = 19 should be retained with the PFLASH2 segment added as above. The array sizes for these tables need to be changed from 19, either to 20 or to NUM_MEMORY_SEGMENTS.

```
const Uint32        parm_mem_start[NUM_MEMORY_SEGMENTS] = {

const Uint16        parm_mem_length[NUM_MEMORY_SEGMENTS] = {     RAM_LENGTH,
```

14.6 Changes to load.asm

It is also necessary to change load.asm. The stack addresses and the ram address for ram clearing need to be changed to match the new RAM location. For clarity, and if dual processor compilation is desired, it might be good to rename load.asm to load_64.asm.

```
SUP_STACK_TOP       .equ    0x69ffc ;Supervisor mode (SWI stack) starts at top of memory
FIQ_STACK_TOP       .equ    0x69e00 ;allocate 256 bytes to supervisor stack, then do FIQ stack
IRQ_STACK_TOP       .equ    0x69d00 ;allocate 256 bytes to fiq stack, then start irq stack
USER_STACK_TOP      .equ    0x69b00 ;Allocate 512 bytes to irq stack, regular stack gets rest, down
to variables
```

The ARM assembly language can only load immediate constants with 8 bits of information. They can be shifted, so a stack top like 0x1a000 can be loaded immediately, like this:

```
MOV     R13, #USER_STACK_TOP ; initialize stack pointer
```

This will not work with the new stack addresses, however. For all stack addresses with the UCD3138064, it will necessary to put values into flash to be copied into the stack registers. Generally some of the stack addresses will already be in this format. Here is the instruction format:

```
LDR             R13, c_user_stack_top ; initialize stack pointer
```

This instruction loads the constant from flash. The load is actually PC relative. The constant is placed in flash using a .long assembler pseudo-op. This should be placed at the end of the load.asm file, outside of the assembly instruction area.

```
c_user_stack_top .long USER_STACK_TOP
```

There is another change necessary for the new RAM addresses – this is for the RAM clearing loop.

```
MOV     A2, #105   ;point at 0x69000 - start of RAM
```

Also in load.asm, it is probably necessary to comment out or change these lines:

```
¿        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
```

Commenting them out will work. Their main function is to make the program flash read only. This isn't as important as it was on UCD30xx series controllers because the FLASH key now also protects the program flash from writes. Executing the statements above is problematic because it is possible that the device is set up to have block2 in the address 0 position instead of block 1. This would occur in a multi-image system.

14.7 Changes to system_defines.h

Change data flash addresses in system_defines.h.

```
#define DATA_FLASH_START_ADDRESS        (0x68800)
#define DATA_FLASH_END_ADDRESS          (0x68fff)
```

14.8 Changes to software interrupt addresses

Change RAM addresses in the software interrupt for checksum clearing/flash erasure.

```
case 12: // clear integrity word.
        {
                {
                        register Uint32 * program_index = (Uint32 *) 0x69000;
                        //store destination address for program
                        register Uint32 * source_index = (Uint32 *)zero_out_integrity_word;
                        //Set source address of PFLASH; …………
                {
                        register FUNC_PTR func_ptr;
                        func_ptr=(FUNC_PTR)0x69000;    //Set function to 0x69000
                        func_ptr();
                }    //execute erase checksum
```

14.9 Changes to Device ID

Change ISO to 64V in device ID:

```
#define DEVICE          UCD310064V1                             //Device Name
```

And/or

```
#define DEVICE_ID       "UCD310064V1|0.0.35.0068|120227"
```

This tells the GUI that this is a cyclone 64 instead of a 32K cyclone.

14.10 Delete Write to HFO_LN_FILTER_EN

In the UCD3138, the following statement is recommended for initialization:

MiscAnalogRegs.CLKTRIM.bit.HFO_LN_FILTER_EN = 0;

In the UCD3138064, this statement is not recommended. For more information, see Section 8, Changes to HFO_LN_FILTER_EN bit.

15 References

1. UCD3138064 Device Datasheet (Literature Number: SLUSB72)
2. UCD3138 Device Datasheet (Literature Number: SLUSAP2)
2. UCD3138 Digital Power Peripherals Programmer's Manual (Literature Number:SLUU995)
3. UCD3138 Monitoring & Communications Programmer's Manual (Literature Number:SLUU996)
4. UCD3138 ARM and Digital System Programmer's Manual (Literature Number:SLUU994)
5. FUSION_DIGITAL_POWER_DESIGNER for Isolated Power Applications (Literature Number: SLUA676)

Document Revision History

Version	Release Date	List of Changes
Revision 1.0	May 2013	Initial Document
Revision A	May 2014	Edits for clarity and formatting Update references with names and literature numbers Update information on HFO_LN_FILTER_EN

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice.

Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment. TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed. TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards. TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI. Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions. Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements. TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications. TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use. TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements. Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products

Audio www.ti.com/audio
Amplifiers amplifier.ti.com
Data Converters dataconverter.ti.com
DLP® Products www.dlp.com
DSP dsp.ti.com
Clocks and Timers www.ti.com/clocks
Interface interface.ti.com
Logic logic.ti.com
Power Mgmt power.ti.com
Microcontrollers microcontroller.ti.com
RFID www.ti-rfid.com
Wireless Connectivity www.ti.com/wirelessconnectivity

Applications

Automotive and Transportation www.ti.com/automotive
Communications and Telecom www.ti.com/communications
Computers and Peripherals www.ti.com/computers
Consumer Electronics www.ti.com/consumer-apps
Energy and Lighting www.ti.com/energy
Industrial www.ti.com/industrial
Medical www.ti.com/medical
Security www.ti.com/security
Space, Avionics and Defense www.ti.com/space-avionics-defense
Video and Imaging www.ti.com/video
OMAP Mobile Processors www.ti.com/omap

TI E2E Community Home Page e2e.ti.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265