Post Configuration Access To SPI Flash

A KCPSM6 Reference Design for the KC705 Evaluation Board

Ken Chapman

18th March 2013

Disclaimer

This Document and Reference Design

The primary purpose of this document is to provide images to supplement the descriptions contained in the source VHDL and PSM code provided with this reference design.

It is assumed that you already have a copy of the KCPSM6 variant of PicoBlaze and are familiar with using it. In particular, this reference design builds on the UART based reference designs provided in the KCPSM6 package so this document focuses on the additions specific to SPI Flash memory.

The reference design is presented on the Kintex-7 KC705 Evaluation Kit. Except for the special requirements associated with the set up of the board, the reference design itself should provide a valid starting point for any 7-Series based design and it is hoped that the SPI related source can be reused.

The SPI Flash memory on the KC705 Evaluation Kit is a Micron/Numonyx N25Q128 device. The source code provided is therefore written to work with this device but it would also be expected to work with similar SPI Flash memory devices from other manufacturers. Most Flash memory devices appear to be the same with respect to general communication and read operations. The differences tend to relate to the internal organization of the flash memory (e.g. the size and number of sectors), the write and the erase operations. Even so, the source code provided should still provide a good starting point.

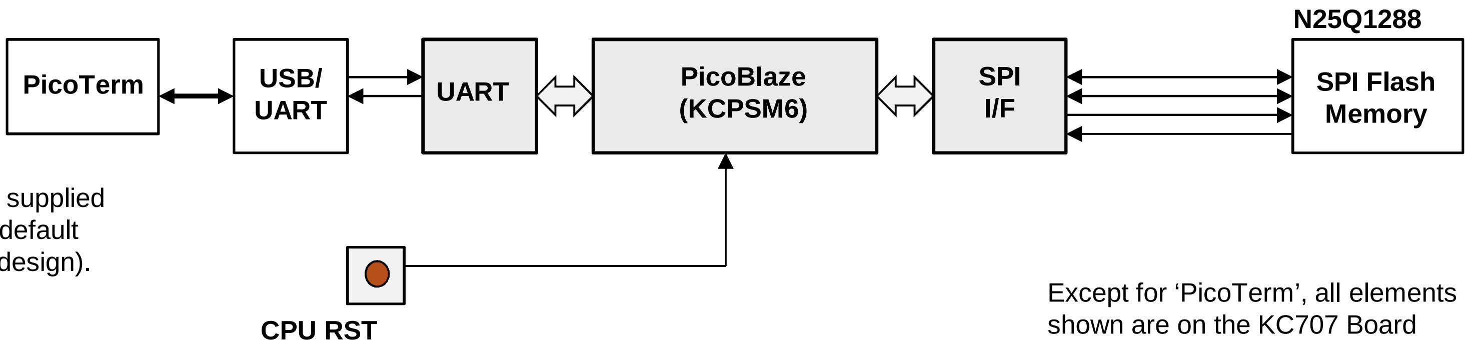
I do hope you find this reference design useful. Please provide any feedback related to this reference design (good or bad) to…

chapman@xilinx.com

Overview of Reference Design

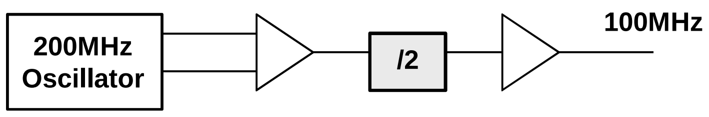
The design implements a bridge between the user of a terminal (PicoTerm) and the N25Q128 Flash memory on the KC705 board. Whilst the primary focus of this reference design is the ability to communicate with and control the N25Q128 using KCPSM6 (PicoBlaze), inclusion of the USB/UART link in the design makes it possible for direct user interaction including reading, writing and erasing operations to be invoked and observed.

Please be aware that the PSM code provided consists in total of 1604 instructions but the vast majority of these are related to user interaction. In fact, over 1150 instructions are directly associated with the generation of text massages. For this reason it is useful to know immediately that all the fundamental SPI communication and Flash memory operations are actually implemented by just 94 instructions and this is isolated in the ‘N25Q128_SPI_routines.psm’ file to make it easy to locate and for future reuse in your own designs.



Use ‘PicoTerm’ supplied with KCPSM6 (default settings match design).

Except for ‘PicoTerm’, all elements shown are on the KC707 Board



The design operates at 100MHz but there is nothing significant about the SPI communication that requires this frequency.
As provided, the SPI interface achieves a bit rate equal to the clock frequency divided by 24 (e.g. 3.57 Mbit/s with 100MHz clock).
Depending on device type and speed grade, KCPSM6 can be used with a clock up to ~240MHz.

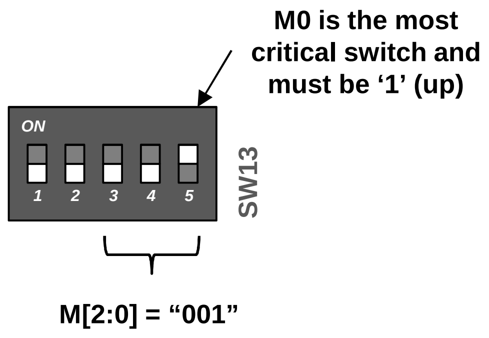
KC705 Setup & SPI Flash Connections

The Micron/Numonyx N25Q128 device is a 128M-bit (16M-Byte) SPI Flash memory which is connected to the Kintex-7 device on the KC705 board. Its primary purpose is to hold a configuration image that would be automatically loaded by the Kintex-7 device operating in Master SPI Mode. For this reason the Flash memory is connected to the pins specifically required for such configuration.

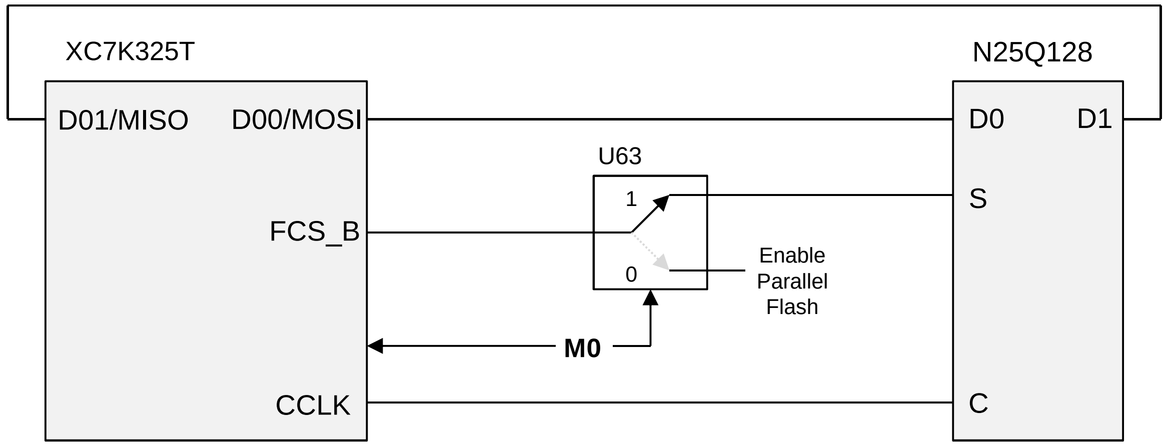
However, with the KC705 being an ‘evaluation kit’, it also provides a parallel Flash memory and the capability to use a Master BPI Mode for configuration. Unsurprisingly there are some DIP switches that you use to select the mode. Less obvious, is that when you select the mode you also steer a control signal on the board to either the SPI Flash or the Parallel Flash. In other words, only the intended type of Flash memory is connected to the Kintex-7 device. Therefore, the DIP switches must be set to Master SPI Mode otherwise the reference design will not be able to access the SPI Flash memory after configuration.

Hint – Failing to set the DIP switches to Master SPI mode is an easy mistake to make initially because you nearly always use JTAG configuration in conjunction with iMPACT during your first experiments and design development cycles. Note that you can set the Master SPI mode permanently because JTAG configuration will always be possible (i.e. switches do not need to be set to “101” to use JTAG).

DIP switches SW13 must be set to Master SPI Mode.



‘Classic’ 4-Wire SPI Communication (reusing configuration pins)

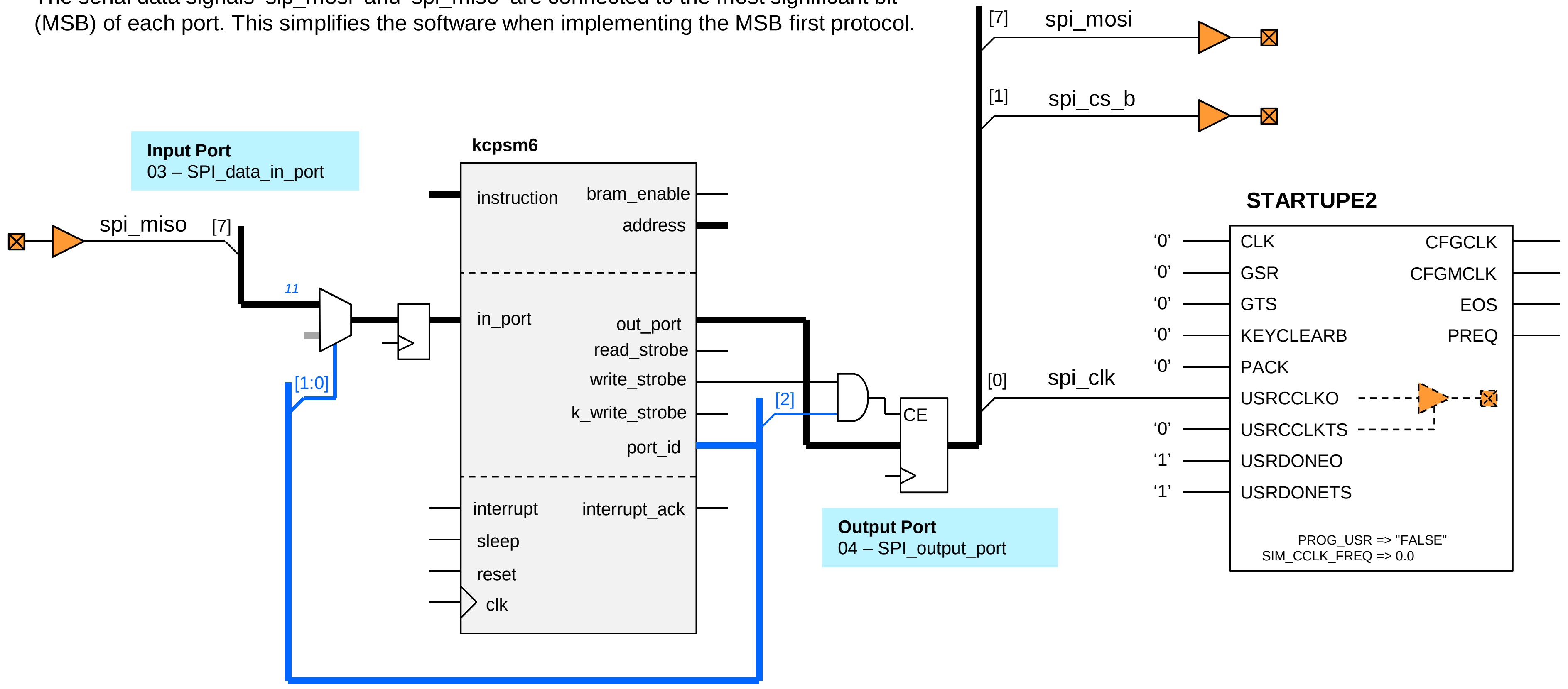


Connecting KCPSM6 to SPI Pins

For clarity this diagram only shows the ports assigned to drive and monitor the SPI signals in the reference design.

KCPSM6 drives ‘spi_clk’, ‘spi_cs_b’ and ‘spi_mosi’ with a single output port and reads ‘spi_miso’ with a single input port. The only special requirement relates to the fact that ‘CCLK’ is a dedicated configuration pin on the device and can only be accessed after configuration by using the STARTUPE2 primitive. Note that only the ‘USRCCLK0’ and ‘USRCCLKTS’ inputs to this primitive are used for this purpose and the other controls and signals are available for other purposes. As provided, the remaining controls are connected to ‘0’ or ‘1’ such that they have no affect on normal operation.

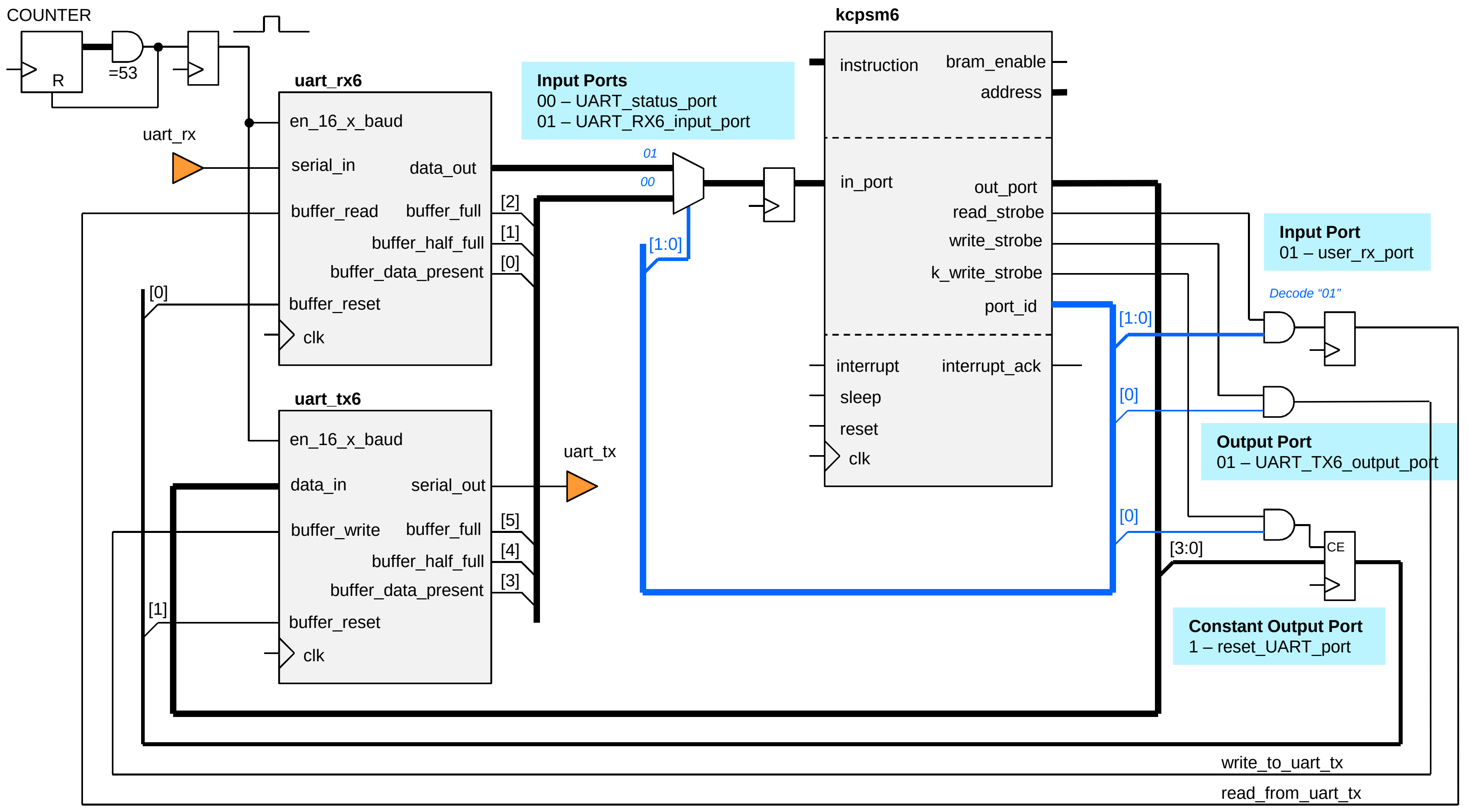
The serial data signals ‘sip_mosi’ and ‘spi_miso’ are connected to the most significant bit (MSB) of each port. This simplifies the software when implementing the MSB first protocol.



User Terminal UART Macros

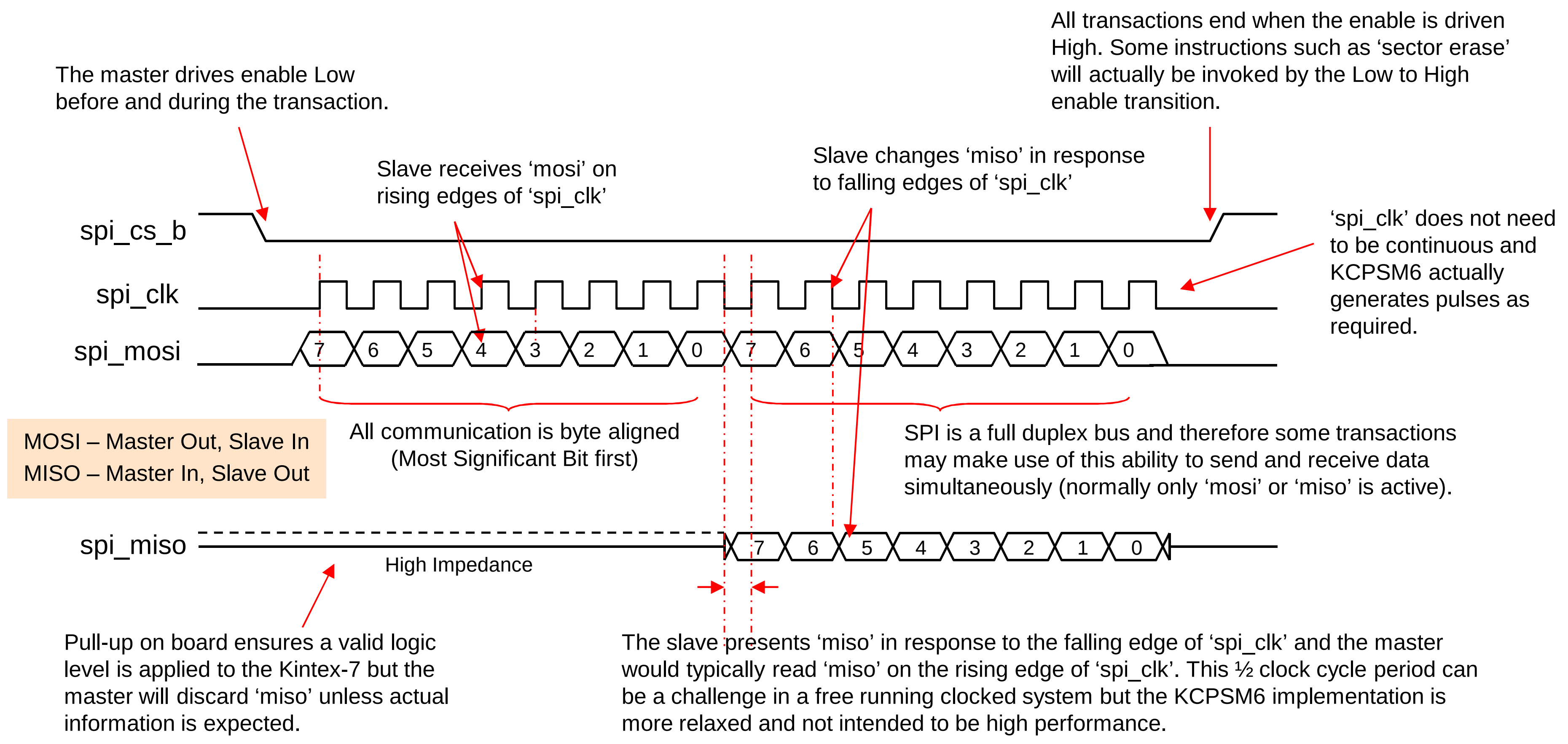
For clarity this diagram only shows the ports assigned to connect to the UART macros used to communicate with the user at 115,200 baud.

For more information about this part of the design please see the documentation provided with KCPSM6 and the UART6 macros.



SPI Fundamentals

In this situation KCPSM6 is the SPI bus master and the N25Q128 Flash Memory is the slave. The following diagram illustrates the key points of any SPI transaction and could actually be an 'RDSR' instruction reading the status byte from the Flash memory.



The master drives enable Low before and during the transaction.
Slave receives 'mosi' on rising edges of 'spi_clk'
Slave changes 'miso' in response to falling edges of 'spi_clk'
All transactions end when the enable is driven High. Some instructions such as 'sector erase' will actually be invoked by the Low to High enable transition.
spi_cs_b
spi_clk
spi_mosi
'spi_clk' does not need to be continuous and KCPSM6 actually generates pulses as required.
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
MOSI – Master Out, Slave In
MISO – Master In, Slave Out
All communication is byte aligned (Most Significant Bit first)
SPI is a full duplex bus and therefore some transactions may make use of this ability to send and receive data simultaneously (normally only 'mosi' or 'miso' is active).
spi_miso
7 6 5 4 3 2 1 0
High Impedance
Pull-up on board ensures a valid logic level is applied to the Kintex-7 but the master will discard 'miso' unless actual information is expected.
The slave presents 'miso' in response to the falling edge of 'spi_clk' and the master would typically read 'miso' on the rising edge of 'spi_clk'. This ½ clock cycle period can be a challenge in a free running clocked system but the KCPSM6 implementation is more relaxed and not intended to be high performance.

SPI Communication Code

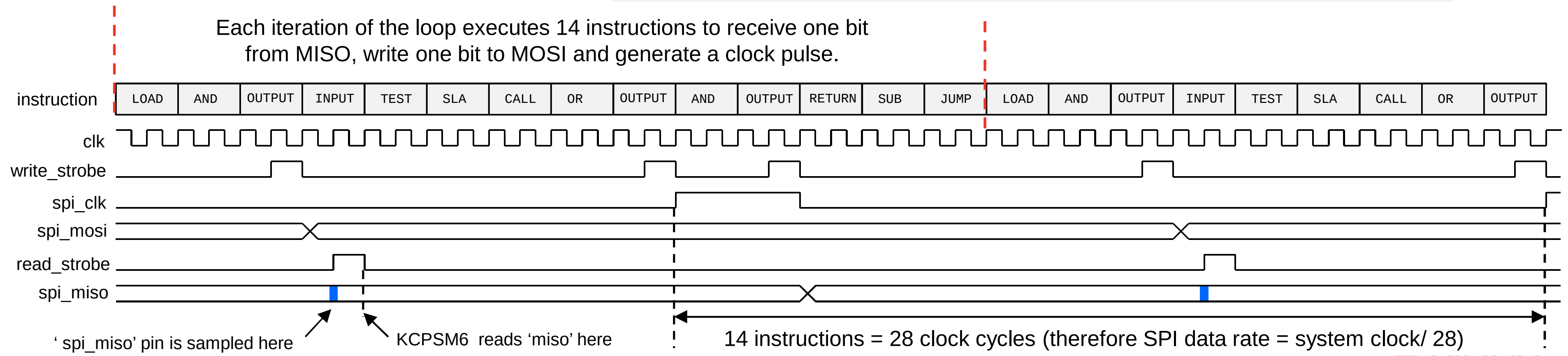
Hint - This page is provided mainly for educational purposes because SPI provides a good example of 'bit banging' of signals to implement an interface.

The 'N25Q128_SPI_routines.psm' file provides a set of routines that implement the fundamental SPI communication as well as complete N25Q128 transactions. In most cases you should be able to reuse this code as provided or only need to enhance the N25Q128 transactions. Shown below is the routine that implements the SPI communication to transmit and receive each byte. Whilst it is unlikely that you would need to adjust this low level code it is a nice example of 'bit banging' code, defines the SPI timing relative to the system clock and completes the description of the SPI signaling in this document.

```
    SPI_FLASH_tx_rx: LOAD s1, 08                  ;8-bits to transmit and receive
next_SPI_FLASH_bit: LOAD s0, s2                   ;prepare next bit to transmit
                    AND s0, spi_mosi              ;isolates data bit and spi_cs_b = 0
                    OUTPUT s0, SPI_output_port    ;output data bit ready to be used on rising clock edge
                    INPUT s3, SPI_data_in_port    ;read input bit
                    TEST s3, spi_miso             ;carry flag becomes value of received bit
                    SLA s2                        ;shift new data into result and move to next transmit bit
                    CALL SPI_clock_pulse          ;pulse spi_clk High
                    SUB s1, 01                    ;count bits
                    JUMP NZ, next_SPI_FLASH_bit   ;repeat until last bit
                    RETURN
```

```
SPI_clock_pulse: OR s0, spi_clk                ;clock High (bit0)
                 OUTPUT s0, SPI_output_port    ;drive clock High
                 AND s0, ~spi_clk              ;clock Low (bit0)
                 OUTPUT s0, SPI_output_port    ;drive clock Low
                 RETURN
```

Each iteration of the loop executes 14 instructions to receive one bit from MISO, write one bit to MOSI and generate a clock pulse.



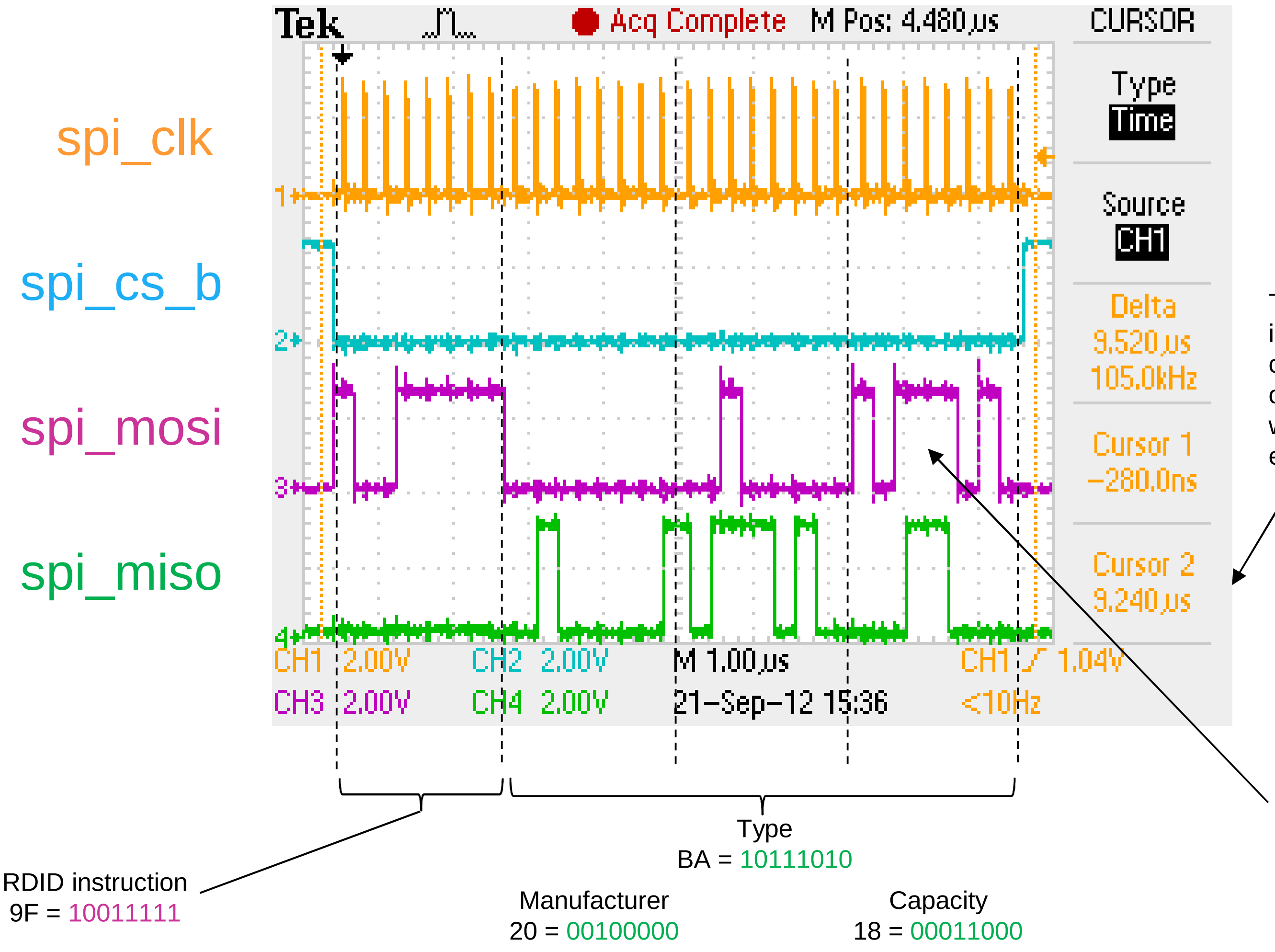
14 instructions = 28 clock cycles (therefore SPI data rate = system clock/ 28)

(Input port multiplexer has a pipeline register)

XILINX

SPI Transaction

The oscilloscope waveforms shown below were captured from the 'J7' header (SPI EXT) on the KC705 board and show an 'RDID' transaction (execution of the 'read_spi_flash_ID' routine in 'N25Q128_SPI_routines.psm').

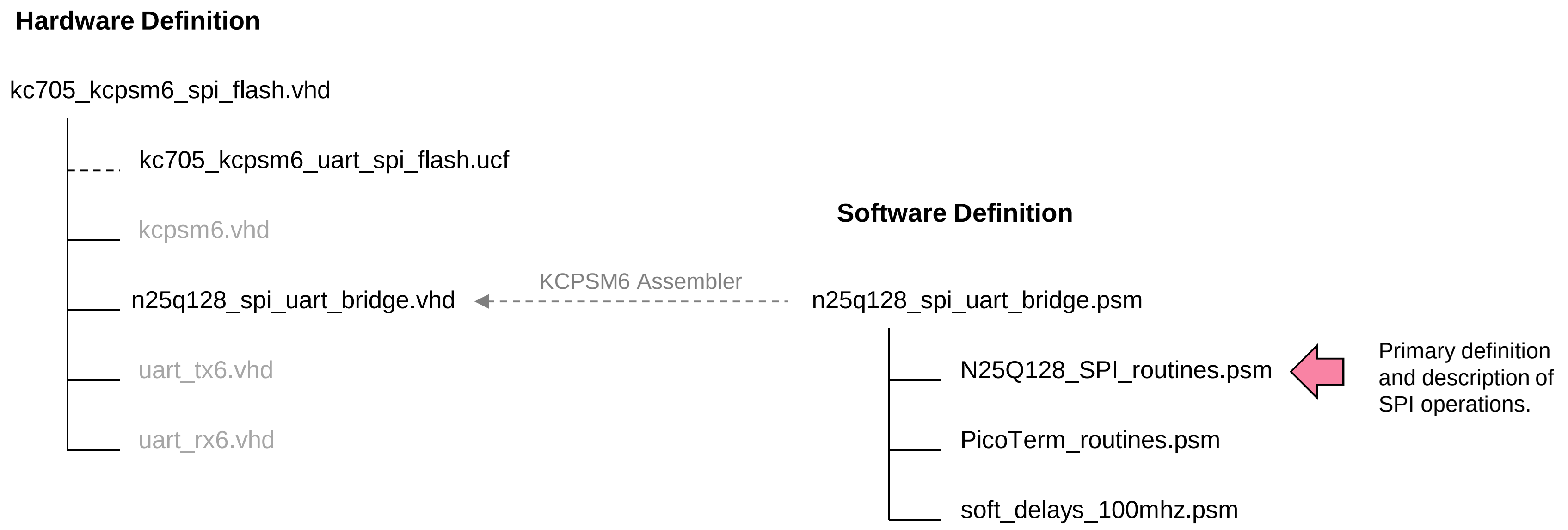


This 4-bytes transaction ('RDID' instruction and 3-byte response) consisted of 32 clock pulses and bits of information and completed in ~9µs which is approximately 3.56Mbps as expected from the PSM code.

Whilst the N25Q128 is responding with information on 'spi_miso' the 'spi_mosi' is ignored. Due to being full-duplex, KCPSM6 has to transmit something on 'spi_mosi' but anything can be present in these 'dummy bytes' (as they are commonly called). In this case the dummy bytes happen to be a delayed copy of data received.

Reference Design Files

All source files contain detailed descriptions and comments. In fact, the descriptions and comments in the source code should be considered the *main* documentation for this reference design with this PDF mainly used to provide an introduction and complementary graphics.

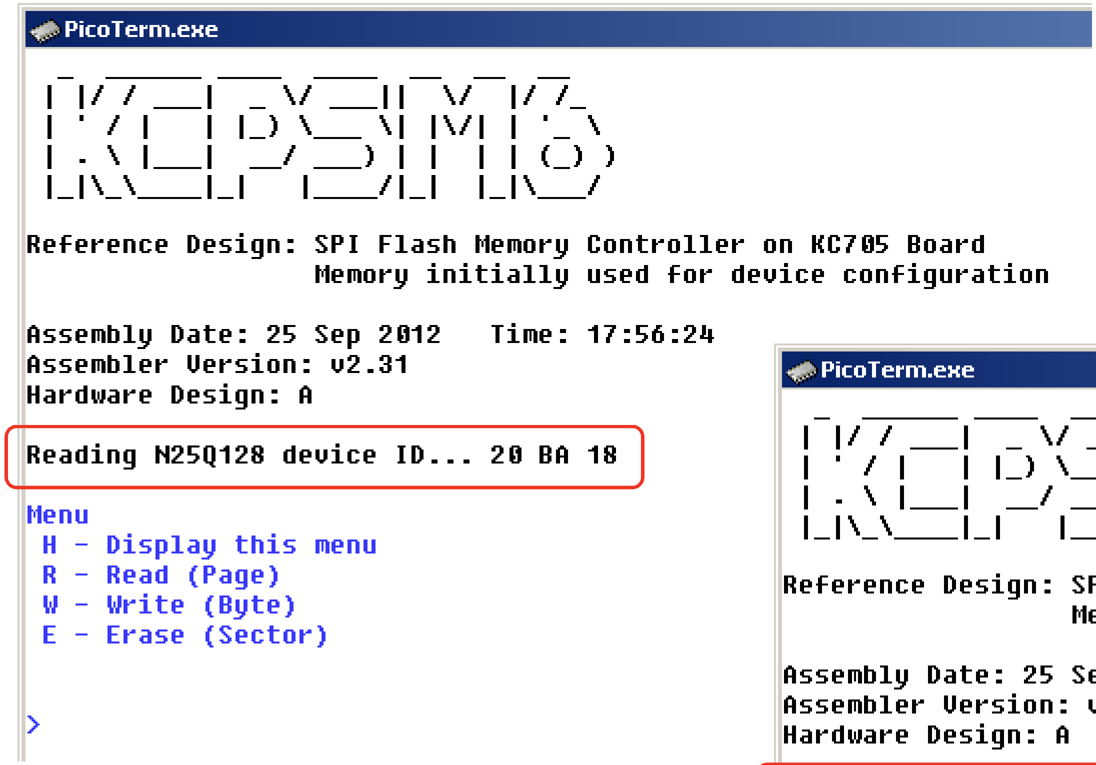

Hardware Definition
kc705_kcpsm6_spi_flash.vhd
kc705_kcpsm6_uart_spi_flash.ucf
kcpsm6.vhd
n25q128_spi_uart_bridge.vhd
uart_tx6.vhd
uart_rx6.vhd
KCPSM6 Assembler
Software Definition
n25q128_spi_uart_bridge.psm
N25Q128_SPI_routines.psm
PicoTerm_routines.psm
soft_delays_100mhz.psm
Primary definition and description of SPI operations.

Files shown in grey are provided in the KCPSM6 package and should be copied and added to your project directory

Hint – The ‘n25q128_spi_uart_bridge.vhd’ file is not provided. Assemble the PSM code in the normal way to generate this file.

N25Q128 Device ID

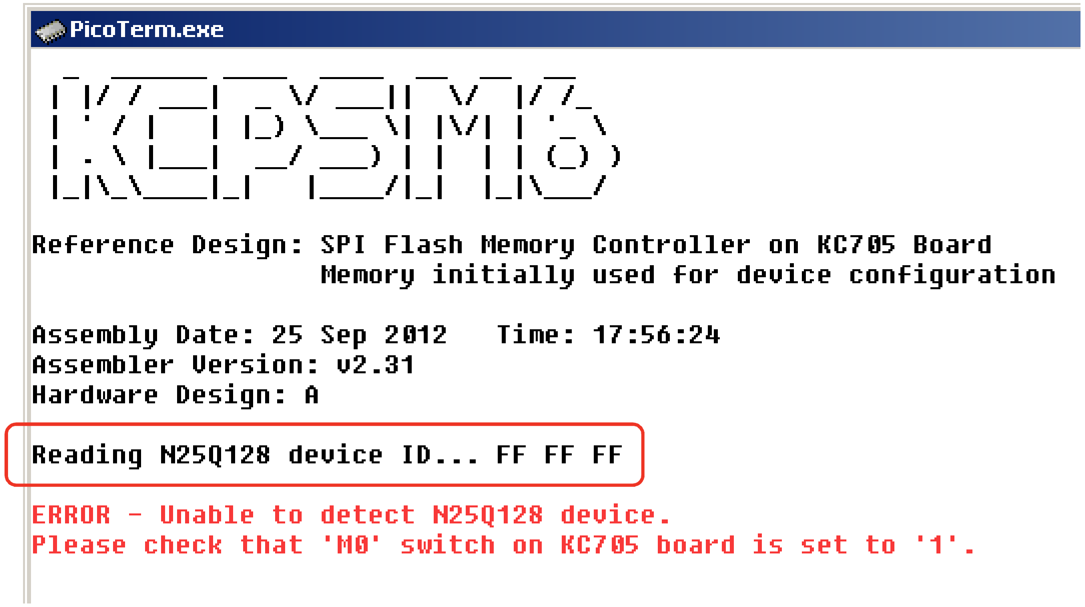
Probably the best thing to do first when communicating with any device is to attempt to read a known value. The N25Q128 Flash Memory can a identification code that can be read using the ‘RDID’ instruction. The reference design does attempt to read this value as part of its initialisation procedure and for the purposes of this reference design it even displays the values read.



PicoTerm.exe

```
  _  ______ ____  ____  __  __  __
 | |/ / ___|  _ \/ ___||  \/  |/ /_
 | ' / |   | |_) \___ \| |\/| | '_ \
 | . \ |___|  __/ ___) | |  | | (_) )
 |_|\_\____|_|   |____/|_|  |_|\___/

Reference Design: SPI Flash Memory Controller on KC705 Board
                  Memory initially used for device configuration

Assembly Date: 25 Sep 2012   Time: 17:56:24
Assembler Version: v2.31
Hardware Design: A

Reading N25Q128 device ID... 20 BA 18

Menu
 H - Display this menu
 R - Read (Page)
 W - Write (Byte)
 E - Erase (Sector)


>
```

If KCPSM6 does not read the expected known value then it will display a message and stop.



PicoTerm.exe

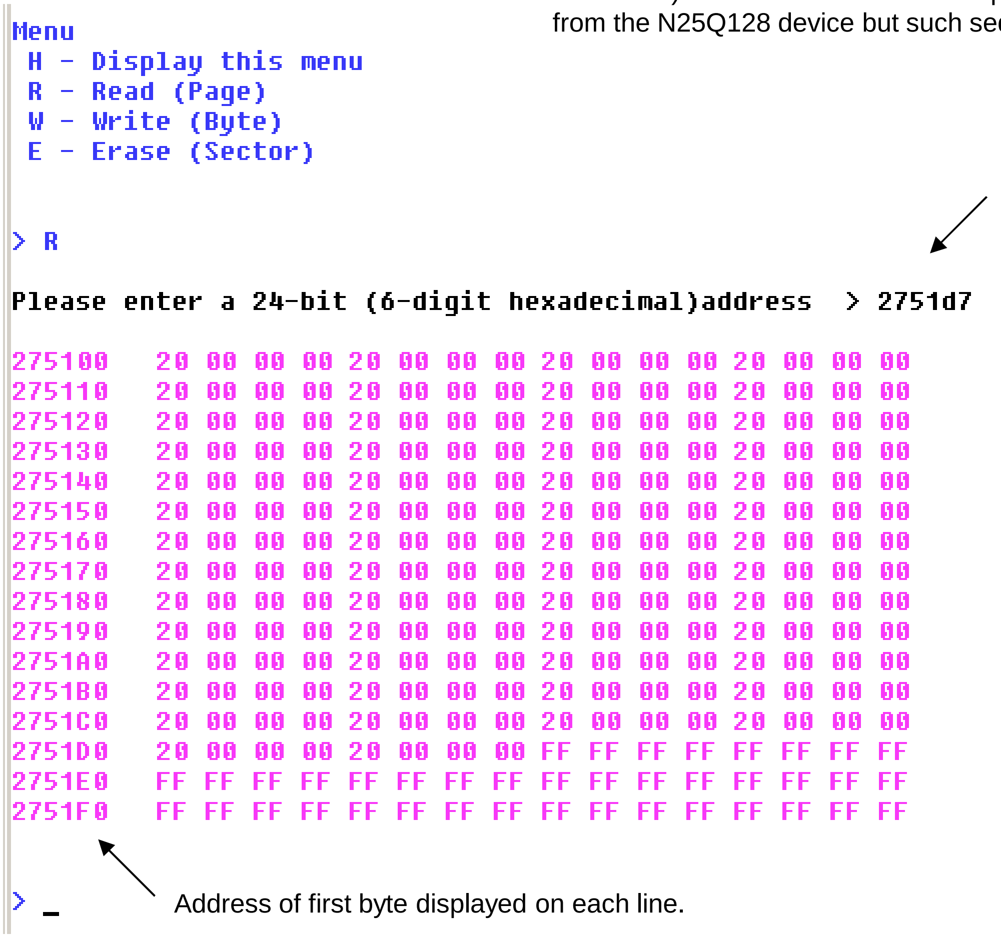
```
  _  ______ ____  ____  __  __  __
 | |/ / ___|  _ \/ ___||  \/  |/ /_
 | ' / |   | |_) \___ \| |\/| | '_ \
 | . \ |___|  __/ ___) | |  | | (_) )
 |_|\_\____|_|   |____/|_|  |_|\___/

Reference Design: SPI Flash Memory Controller on KC705 Board
                  Memory initially used for device configuration

Assembly Date: 25 Sep 2012   Time: 17:56:24
Assembler Version: v2.31
Hardware Design: A

Reading N25Q128 device ID... FF FF FF

ERROR - Unable to detect N25Q128 device.
Please check that 'M0' switch on KC705 board is set to '1'.
```

Hint - See the ‘read_spi_flash_ID’ routine in ‘N25Q128_SPI_routines.psm’.

Read Memory Contents

Data can be read sequentially from the N25Q128 starting at any location. The reference design actually reads one byte at a time and this effectively represents totally random access.

The reference design allows you to specify any 24-bit address. The design with then read the Page (see box below) in which that location is a part. In this case the design actually makes 256 separate reads from the N25Q128 device but such sequential reading could be optimised if required.



```
Menu
 H - Display this menu
 R - Read (Page)
 W - Write (Byte)
 E - Erase (Sector)


> R

Please enter a 24-bit (6-digit hexadecimal)address  > 2751d7

275100   20 00 00 00 20 00 00 00 20 00 00 00 20 00 00 00
275110   20 00 00 00 20 00 00 00 20 00 00 00 20 00 00 00
275120   20 00 00 00 20 00 00 00 20 00 00 00 20 00 00 00
275130   20 00 00 00 20 00 00 00 20 00 00 00 20 00 00 00
275140   20 00 00 00 20 00 00 00 20 00 00 00 20 00 00 00
275150   20 00 00 00 20 00 00 00 20 00 00 00 20 00 00 00
275160   20 00 00 00 20 00 00 00 20 00 00 00 20 00 00 00
275170   20 00 00 00 20 00 00 00 20 00 00 00 20 00 00 00
275180   20 00 00 00 20 00 00 00 20 00 00 00 20 00 00 00
275190   20 00 00 00 20 00 00 00 20 00 00 00 20 00 00 00
2751A0   20 00 00 00 20 00 00 00 20 00 00 00 20 00 00 00
2751B0   20 00 00 00 20 00 00 00 20 00 00 00 20 00 00 00
2751C0   20 00 00 00 20 00 00 00 20 00 00 00 20 00 00 00
2751D0   20 00 00 00 20 00 00 00 FF FF FF FF FF FF FF FF
2751E0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2751F0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

> _
```

Address of first byte displayed on each line.

Hint – 2751D7 hex appears to be the last location occupied by a configuration image for the 7K325T device on the KC705 board. Note that means that the first 28 hex (40) Sectors are used.

Hint - See the 'read_spi_byte' routine in 'N25Q128_SPI_routines.psm'.

The N25Q128 memory is 128Mbits accessed as 16M-Bytes.

Internally the memory is divided into 256 Sectors of 64K-Bytes.

Each Sector is formed of 256 Pages of 256 Bytes.

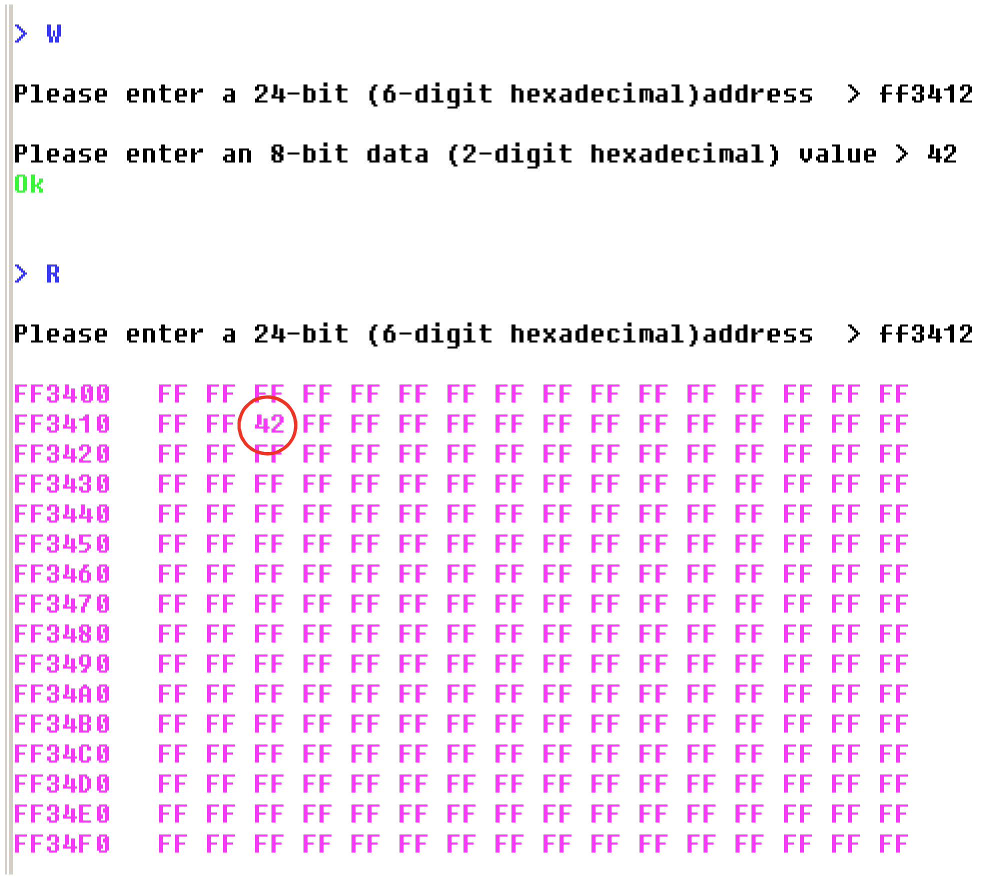
Hence the 24-bit address can be considered in three parts as follows…

address[23:16]	= Sector
address[15:9]	= Page
address[7:0]	= Byte

Writing Data

Data can be written to any location but it must be remembered that during a write operation bits can only be changed from ‘1’ to ‘0’. Therefore in most situations the memory will have been previously erased (all bytes in a sector to FF hex).

The reference design allows you to specify any 24-bit address and any 8-bit data value and it will then write that information into the N25Q128 device. In a similar way to reading data, it is also possible to write to the memory sequentially. However, writing is absolutely related to Page boundaries (see box below) so some additional consideration would be required.



```
> W

Please enter a 24-bit (6-digit hexadecimal)address  > ff3412

Please enter an 8-bit data (2-digit hexadecimal) value > 42
Ok


> R

Please enter a 24-bit (6-digit hexadecimal)address  > ff3412

FF3400   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF3410   FF FF 42 FF FF FF FF FF FF FF FF FF FF FF FF FF
FF3420   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF3430   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF3440   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF3450   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF3460   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF3470   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF3480   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF3490   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF34A0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF34B0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF34C0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF34D0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF34E0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF34F0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

<u>Hint</u> - See the ‘write_spi_byte’ routine
in ‘N25Q128_SPI_routines.psm’.

The N25Q128 memory is 128Mbits accessed as 16M-Bytes.

Internally the memory is divided into 256 Sectors of 64K-Bytes.

Each Sector is formed of 256 Pages of 256 Bytes.

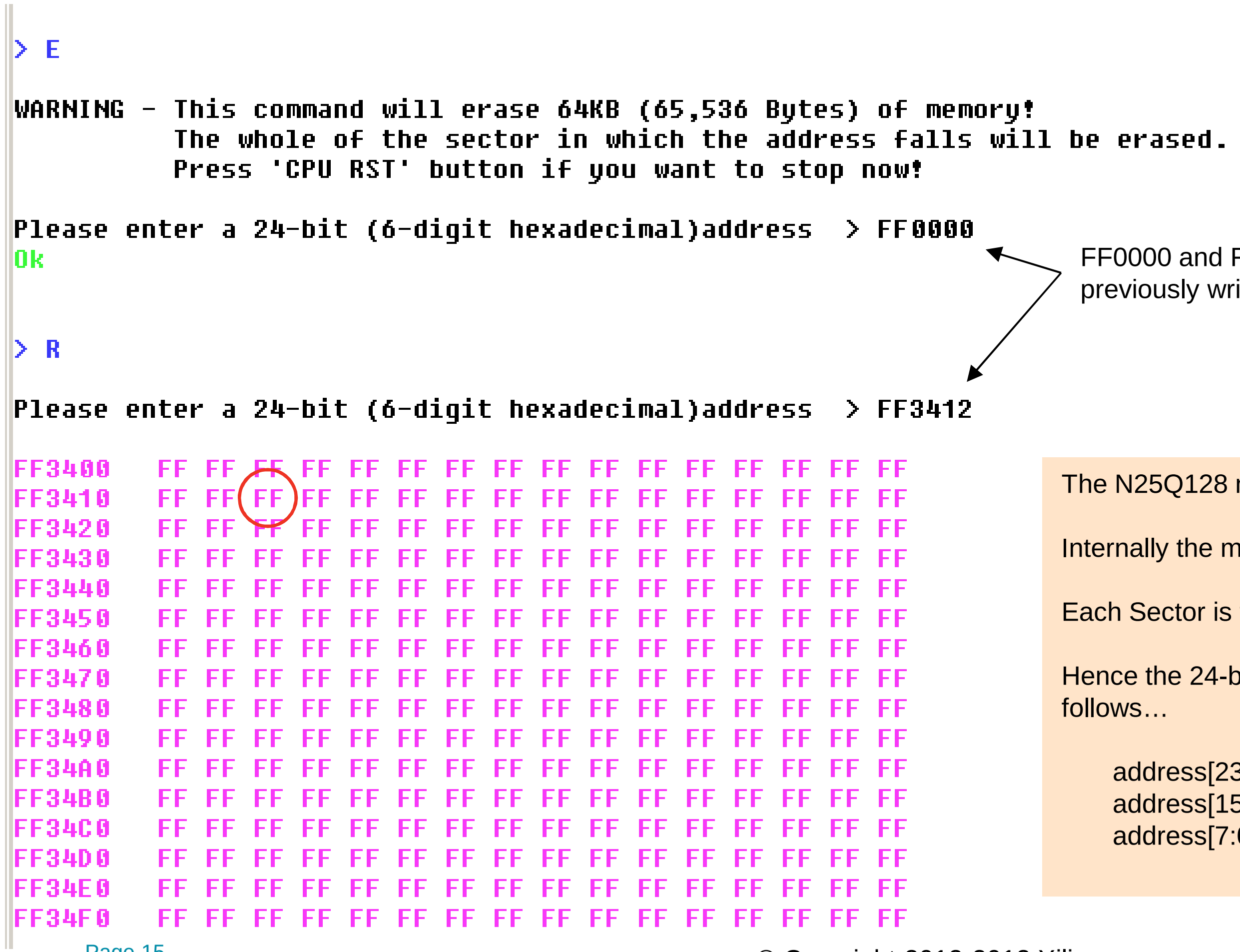
Hence the 24-bit address can be considered in three parts as follows…

address[23:16]	= Sector
address[15:9]	= Page
address[7:0]	= Byte

Erasing a Sector

In most cases (see N25Q128 data sheet for exceptions) the smallest range of memory that can be erased is a Sector (see box below). This means that 64K-Bytes within the specified sector will be erased (all bytes set to FF hex).

The reference design allows you to specify any 24-bit address. Then the Sector in which that address is located will be erased. Note that a typical sector erase time is ~0.7 seconds so you should be able to see the small delay between entering the last digit of the address and the ‘Ok’ being displayed.



```
> E

WARNING - This command will erase 64KB (65,536 Bytes) of memory!
          The whole of the sector in which the address falls will be erased.
          Press 'CPU RST' button if you want to stop now!

Please enter a 24-bit (6-digit hexadecimal)address  > FF0000
Ok


> R

Please enter a 24-bit (6-digit hexadecimal)address  > FF3412

FF3400   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF3410   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF3420   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF3430   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF3440   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF3450   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF3460   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF3470   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF3480   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF3490   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF34A0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF34B0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF34C0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF34D0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF34E0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF34F0   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

<u>YOU HAVE BEEN WARNED!</u>

FF0000 and FF3412 both fall within the FFxxxx Sector so the previously written data has been erased as have all 65,536 bytes.

<u>Hint</u> - See the ‘erase_spi_sector’ routine in ‘N25Q128_SPI_routines.psm’.

The N25Q128 memory is 128Mbits accessed as 16M-Bytes.

Internally the memory is divided into 256 Sectors of 64K-Bytes.

Each Sector is formed of 256 Pages of 256 Bytes.

Hence the 24-bit address can be considered in three parts as follows…

address[23:16]	= Sector
address[15:9]	= Page
address[7:0]	= Byte