KCPSM6 Reference Design for the KC705 Evaluation Board

I2C Communication

including…

PCA9548 Bus Switch and M24C08 EEPROM

Ken Chapman

18th March 2013

© Copyright 2012-2013 Xilinx

Disclaimer

Notice of Disclaimer

Xilinx is disclosing this Application Note to you "**AS-IS**" with no warranty of any kind. This Application Note is one possible implementation of this feature, application, or standard, and is subject to change without further notice from Xilinx. You are responsible for obtaining any rights you may require in connection with your use or implementation of this Application Note. XILINX MAKES NO REPRESENTATIONS OR WARRANTIES, WHETHER EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL XILINX BE LIABLE FOR ANY LOSS OF DATA, LOST PROFITS, OR FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR INDIRECT DAMAGES ARISING FROM YOUR USE OF THIS APPLICATION NOTE.

Copyright © 2012-2013, Xilinx, Inc.
This file contains proprietary information of Xilinx, Inc. and is protected under U.S. and international copyright and other intellectual property laws.

© Copyright 2012-2013 Xilinx

XILINX

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 I2C communication and the ability to use this interface to control PCA9548 and M24C08 devices.

The reference design is presented on the Kintex-7 KC705 Evaluation Kit. The board employs an I2C Bus Switch (PCA9548) which must be controlled in order to communicate with any of the other I2C devices on the board. In this design the M24C08 EEPROM is the chosen target. Although this makes the design somewhat specific the KC705 board, the reference design itself should provide a valid starting point for any 7-Series based design and it is hoped that the I 2C related source can be reused.

The M24C08 EEPROM provided on the KC705 Evaluation Kit is an 8kbit memory organised as 1024 bytes. The PCA9548 Bus Switch supports the connection and selection of 8 busses. The source code provided has been written to implement the I2C transactions required by these specific devices. Hopefully these examples will enable you to implement I2C transactions for devices you need to communicate with in the future. The code that implements the I2C signaling can normally be reused as provided.

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

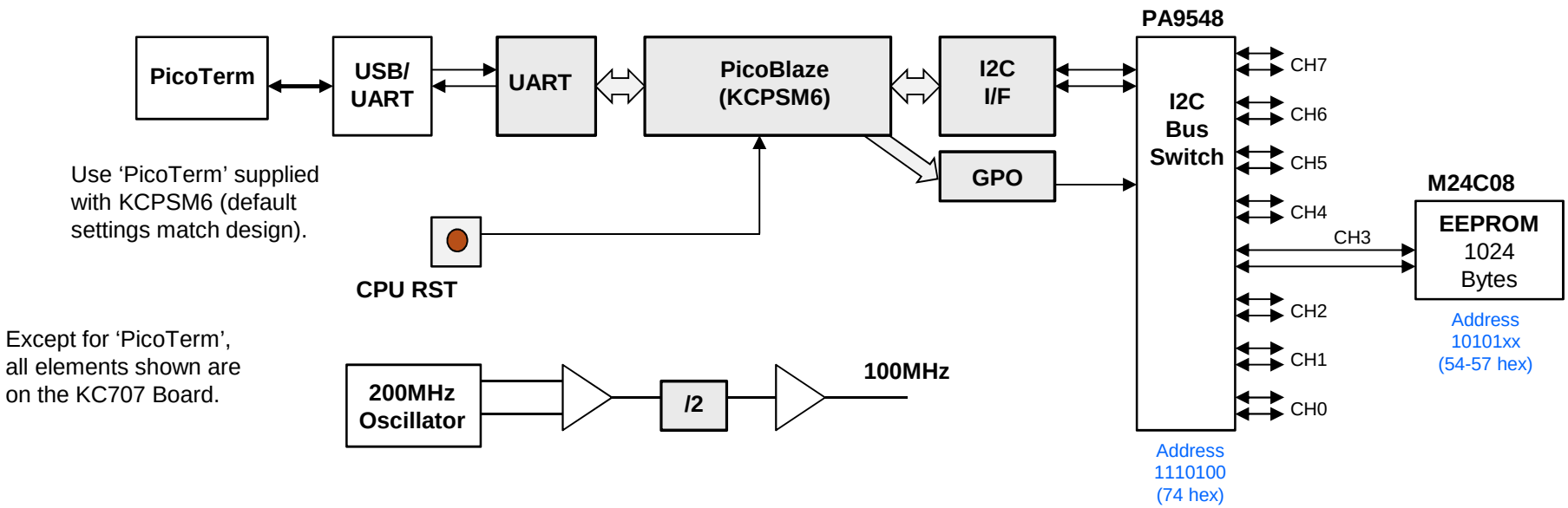
© Copyright 2012-2013 Xilinx

XILINX

Overview of Reference Design

The design implements a bridge between the user of a terminal (PicoTerm) and the M24C08 EEPROM on the KC705 board. Whilst the primary focus of this reference design is the implementation of I2C communication with KCPSM6 (PicoBlaze) and its ability to control and communicate two I2C devices, the inclusion of the USB/UART link in the design makes it possible for direct user interaction including reading and writing EEPROM data.

Please be aware that the PSM code provided consists in total of 1635 instructions but the vast majority of these are related to user interaction. In fact, over 1000 instructions are directly associated with the generation of text massages. For this reason it is useful to know immediately that all the fundamental I2C communication, Bus Switch control and EEPROM read/write operations are actually implemented by just 151 instructions. These I2C specific routines are contained in their own 'i2c_routines.psm' and 'kc705_i2c_devices.psm' files ready to be reused in your own designs.



I2C Data Rate verses System Clock Frequency

The design provided operates at 100MHz but KCPSM6 can be used with a clock frequencies up to ~240MHz (depending on device type and speed grade). 'Classic' I2C communication is implemented with a data rate not exceeding 100k-bit/s. This data rate is rather slow relative to the available performance of KCPSM6 or 7-Series devices but this is actually a good reason to use KCPSM6 which can efficiently implement delays using a few instructions. A constant (I2C_time_reference) is defined within the 'i2c_routines.psm' file which can be modified when using a different system clock frequency. A full description is provided in the PSM file.

© Copyright 2012-2013 Xilinx

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_i2c_eeprom.vhd

- kc705_kcpsm6_uart_i2c_eeprom.ucf
- kcpsm6.vhd
- m24c08_i2c_uart_bridge.vhd
- uart_tx6.vhd
- uart_rx6.vhd

Software Definition

KCPSM6 Assembler

m24c08_i2c_uart_bridge.psm

- i2c_routines.psm — Primary definition of I2C signalling.
- kc705_i2c_devices.psm — I2C Transactions for PCA9548 and M24C08 devices.
- PicoTerm_routines.psm
- soft_delays_100mhz.psm

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

Hint – The ‘m24c08_i2c_uart_bridge.vhd’ file is not provided. Assemble the PSM code in the normal way to generate this file.

© Copyright 2012-2013 Xilinx

XILINX

Confirming I2C Communication With Devices

It is generally a good idea to verify that communication with devices is possible before you attempt to use them. This reference design includes such a built-in test as part of its initialisation sequence, and because it is a reference design, it displays information as it does so.

When an I2C bus master (KCPSM6 in this case) transmits a byte of information, the receiving slave device is expected to respond with an acknowledgement bit (ACK). Therefore, observing this acknowledgement is a good indication that communication is possible with that slave.

```
PicoTerm.exe

Reference Design: M24C08 EEPROM Controller for KC705 Board
                  KCPSM6 implements an I2C 'Master'

Assembly Date: 12 Oct 2012   Time: 15:29:19
Assembler Version: v2.35
Hardware Design: A

Testing I2C communication
  Bus Switch (PCA9548)... Pass
  1KB EEPROM (M24C08).... Pass

Menu
 H - Display this menu
 R - Read (all 1K Bytes)
 W - Write (Byte)

> _
```

Hint – There are more descriptions provided with the related code in 'm24c08_i2c_uart_bridge.psm'

At any time that KCPSM6 does not receive an 'ACK' when it is expecting to do so then it will display the message shown below and stop.

```
PicoTerm.exe

Reference Design: M24C08 EEPROM Controller for KC705 Board
                  KCPSM6 implements an I2C 'Master'

Assembly Date: 12 Oct 2012   Time: 15:43:56
Assembler Version: v2.35
Hardware Design: A

Testing I2C communication
  Bus Switch (PCA9548)...

ERROR - I2C unable to communicate!
        Please try a complete power cycle of the KC705 board.
```


© Copyright 2012-2013 Xilinx

Reading M24C08 EEPROM Contents

The M24C08 is an 8kbit memory organised as 1024 bytes (10-bit address range 000 to 3FF hex) . The memory can be randomly accessed and the reference design can read and display the contents of all bytes as shown in the example below.

In this image the value 19 hex is located as address 042 hex.

> R

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F		+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
000	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	200	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
010	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	210	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
020	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	220	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
030	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	230	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
040	FF	FF	19	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	240	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
050	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	250	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
060	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	260	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
070	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	270	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
080	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	280	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
090	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	290	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0A0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	2A0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0B0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	2B0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	2C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	2D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0E0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	2E0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0F0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	2F0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
100	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	300	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
110	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	310	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
120	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	320	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
130	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	330	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
140	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	340	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
150	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	350	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
160	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	360	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
170	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	370	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
180	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	380	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
190	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	390	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
1A0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	3A0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
1B0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	3B0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
1C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	3C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
1D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	3D0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
1E0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	3E0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
1F0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	3F0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

Hint - See the 'M24C08_read' routine in 'kc705_i2c_devices.psm'.

© Copyright 2012-2013 Xilinx

Writing Data to M24C08 EEPROM

Any of the 1024 bytes can be written to (modified) and reference design enables you to perform this in a simple and obvious way.

```
Menu
 H - Display this menu
 R - Read (all 1K Bytes)
 W - Write (Byte)


> W

Please enter a 10-bit (3-digit hexadecimal) address  > 02e

Please enter an 8-bit data (2-digit hexadecimal) value > c7
Ok


> _
```

Once the address and data has been transmitted, it does take a few milliseconds for the M24C08 to modify the memory contents. This design waits 20ms before reporting 'Ok' in order to cover the worse case delay indicated in the data sheet.

Hint - See the 'M24C08_write' routine in 'kc705_i2c_devices.psm'.

WARNING - Whilst the M24C08 is fully random access and any location can be written with any value it should be appreciated that there are a limited number of 'Erase/Write' cycles. The M24C08 data sheet indicates that over 1 million cycles are possible; this sound like a large number but if an application wrote one byte per second then this number would be exceed in just 12 days.

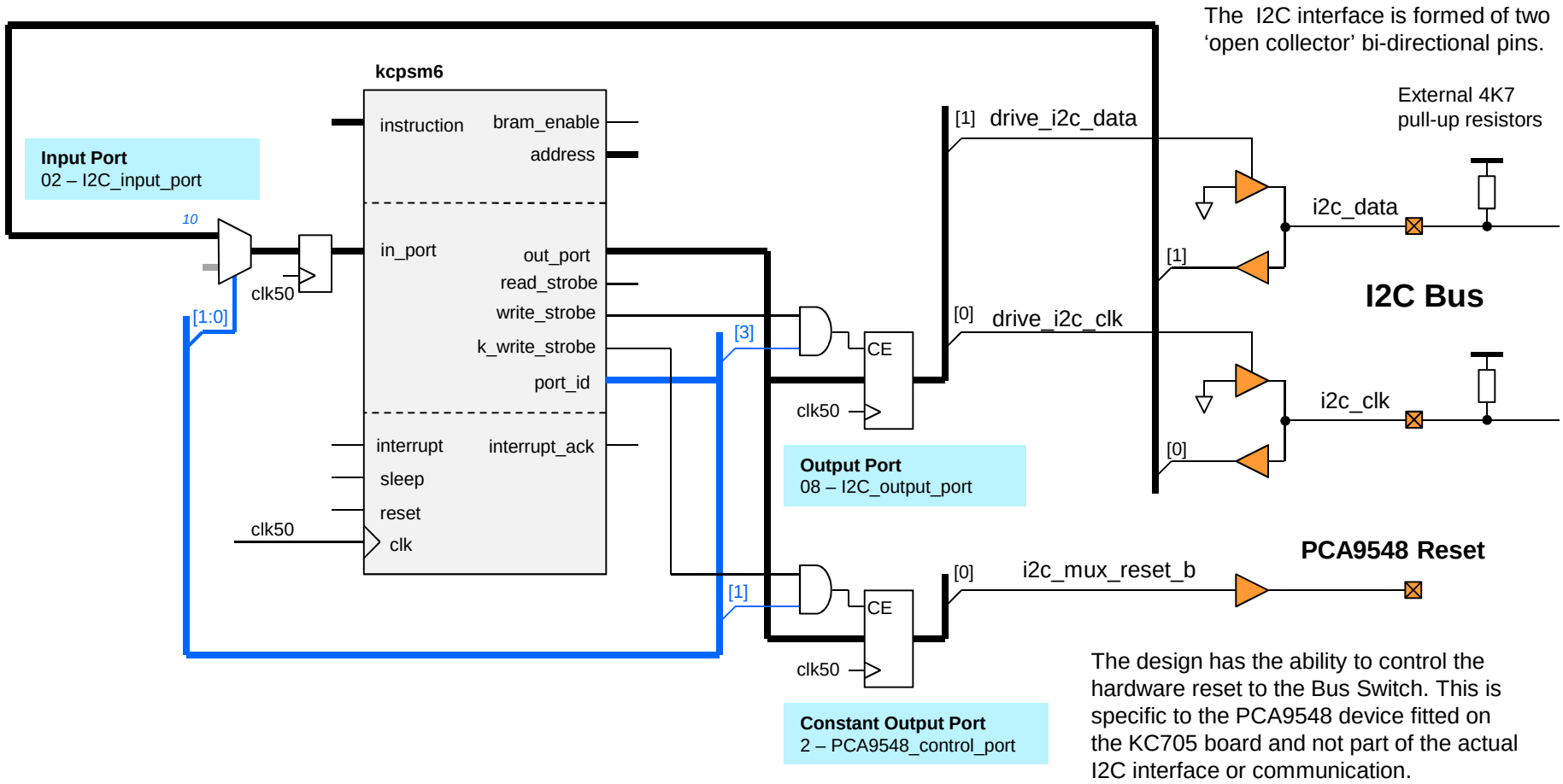
```
> R
        +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F        +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F

    000 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    200 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    010 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    210 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    020 FF FF FF FF FF FF FF FF FF FF FF FF FF FF C7 FF    220 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    030 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    230 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    040 FF FF 19 FF FF FF FF FF FF FF FF FF FF FF FF FF    240 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    050 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    250 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    060 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    260 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
    070 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    270 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```


© Copyright 2012-2013 Xilinx

Connecting KCPSM6 to I2C Pins

For clarity this diagram only shows the ports assigned to drive and monitor the I2C bus in the reference design.

Please see 'kc705_kcpsm6_i2c_eeprom.vhd' for further description and the definition of this hardware.



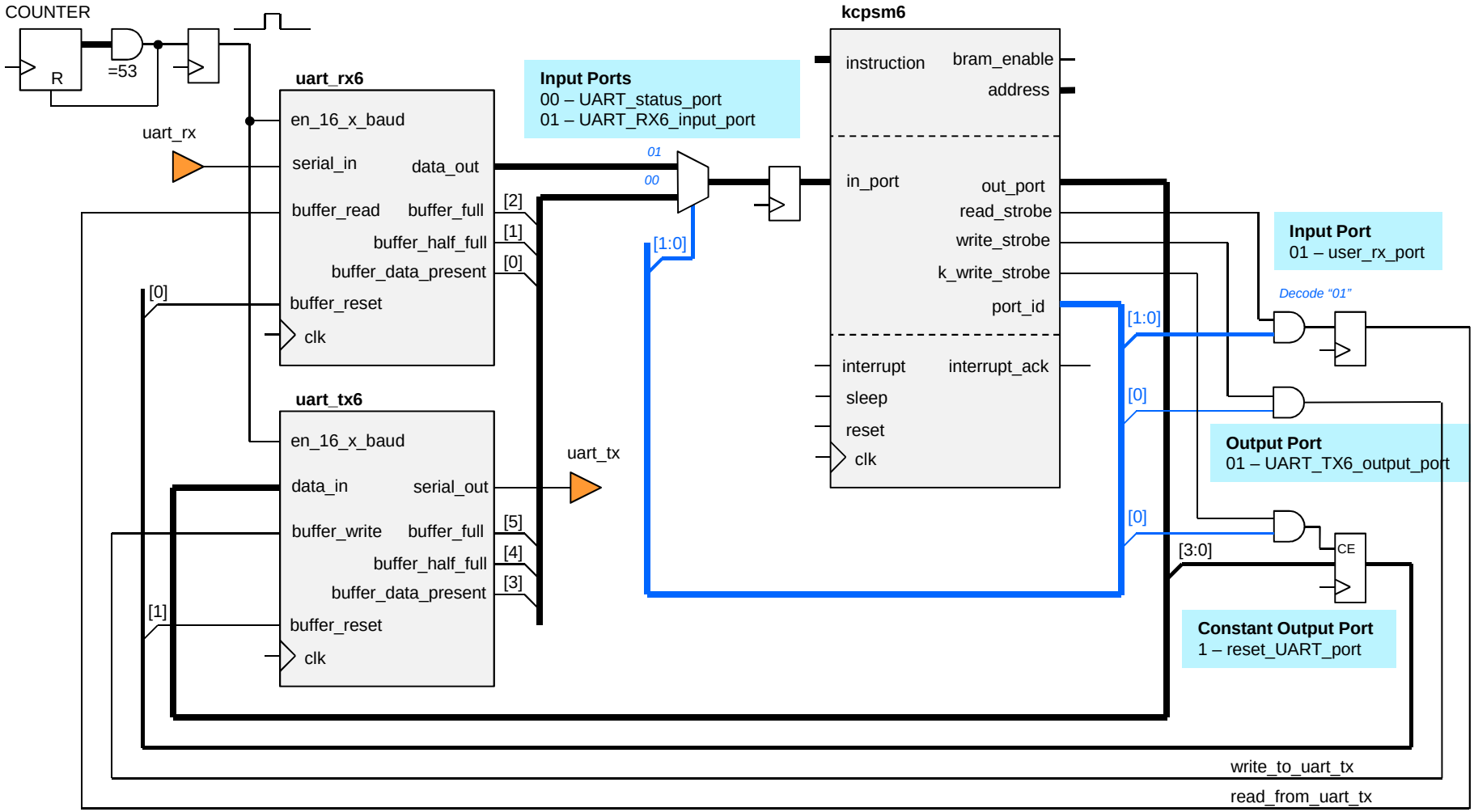
The design has the ability to control the hardware reset to the Bus Switch. This is specific to the PCA9548 device fitted on the KC705 board and not part of the actual I2C interface or communication.

© Copyright 2012-2013 Xilinx

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.

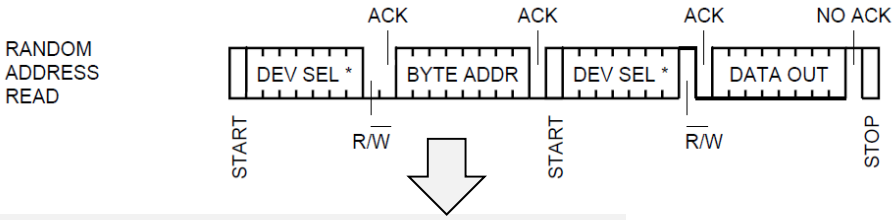


© Copyright 2012-2013 Xilinx

I2C Transactions ('kc705_i2c_devices.psm')

The 'kc705_i2c_devices.psm' provides routines that implement the I2C transactions to read from and write to the I2C Bus Switch (PCA9548) and EEPROM (M24C08) on the KC705 board. All I2C transactions are formed from the same fundamental elements but must be crafted to the requirements of each target. As such the routines provided are also reference examples for when you need to implement transactions suitable for different devices.

Read transaction from the M24C08 data sheet




```
M24C08_read: CALL I2C_initialise
             CALL I2C_start
             LOAD s5, M24C08_base_address
             OR s5, s8
             SL0 s5
             CALL I2C_Tx_byte
             CALL I2C_Rx_ACK
             RETURN C
             LOAD s5, s7
             CALL I2C_Tx_byte
             CALL I2C_Rx_ACK
             RETURN C
             CALL I2C_start
             LOAD s5, M24C08_base_address
             OR s5, s8
             SL1 s5
             CALL I2C_Tx_byte
             CALL I2C_Rx_ACK
             RETURN C
             CALL I2C_Rx_byte
             LOAD sD, s5
             CALL I2C_Tx_NACK
             CALL I2C_stop
             RETURN
```

Hint – Carry flag will be set if slave did not respond.

Each I2C transaction consists of the appropriate sequence of elements. The data sheet for a slave device will illustrate the format of the transactions that it requires. Shown to the left is the illustration of the transaction required to read a byte of data from the M24C08 EEPROM. Below is the illustration of the transaction required to read the control register from the PCA9548 Bus Switch. Not only is it clear that transactions do have different formats depending on the device and operation being performed, but it is also clear that there is no consistency to the way in which transactions are represented. However, it doesn't take long to interpret what each vendor means and everything starts to look familiar!

Read transaction from the PCA9548 data sheet

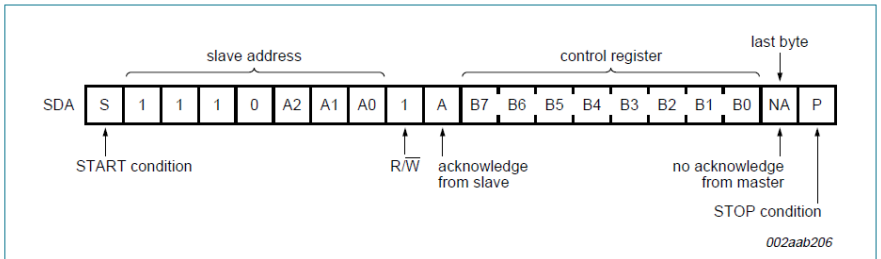


Fig 13. Read control register

Shown on the left is the implementation of the EEPOM read transaction provided in the 'kc705_i2c_devices.psm' file. Highlighted in **red** are the series of calls made to the routines provided in the 'i2c_routines.psm' file to implement the M24C08 read transaction shown above it. The majority of the task has been implemented by the standard routines and only a few instructions are required to customise the transaction with device address, EEPROM address and data.

Hint - The 'kc705_i2c_devices.psm' file includes comprehensive descriptions and comments.

© Copyright 2012-2013 Xilinx

I2C Signalling ('i2c_routines.psm')

The fundamental signalling is implemented by KCPSM6 using the routines provided in 'i2c_routines.psm'. In most cases it should be possible to reuse this code in any situation that KCPSM6 acts as the only 'master' on a traditional I2C bus. In this case, the term 'traditional' implies that slave designs have 7-bit addresses and all communication is byte aligned. As provided, the code also ensures that that data rate is less that 100k-bit/s; it would be very easy for KCPSM6 and 7-Series devices to operate at higher speeds but it is vital to consider the specifications of the slave devices and the dynamics of the signals on the board, i.e., Potentially long traces with high capacitance combined with 'open collector' drivers and pull-up resistors will never be super fast!

The 'i2c_routines.psm' file contains descriptions that should be treated as being the principle documentation. Below, are some points that you must understand to use the code successfully in your own designs. The following pages contain some details about the implementation of I2C signalling which you probably don't need to understand in order to implement your own I2C transactions; hopefully it is still interesting and a useful design example.

Timing KCPSM6 can be provides with a system clock of any reasonable frequency and the code provided will implement software delays which will ensure that I2C communication does not exceed 100k-bit/s. However, this does require that you set one constant in the way described in the PSM file. In the provided design, the clock rate is 100MHz so the 'I2C_time_reference' has been set to 24 decimal.

```
CONSTANT I2C_time_reference, 24'd
;
;  I2C_time_reference =  ( fclk - 6 ) / 4
```

Hint - You can see this constant being used in the 'I2C_delay_1us' sub-routine which is then used to implement a variety of different delays .

I/O Ports As shown on page 5, KCPSM6 uses one output port to control the drive of the two I2C outputs and uses one input port to read the states of the two I2C signals. The constants shown below must be set to define the ports allocated in your design.

```
CONSTANT I2C_input_port, 02       ;port address of I2C input port
CONSTANT I2C_output_port, 08      ;port address of I2C output port
```

Register and Routines

```
s0, s1, s5 and sF
```

Whilst I2C transactions have a common look and feel, they must be constructed (typically by you) to meet the requirements defined for the slave device you are working with. For example, the format of transactions used to communicate with the PCA9548 and M24C08 devices in this design are different but constructed from the same elements (routines in this PSM file).

When implementing a transaction you will call the various routines in the 'i2c_routines.psm' file in the sequence required. These routine will use the registers shown above so you will need to be careful which ones you use yourself. 's5' is used to transfer bytes of information to or from some routines. 's0' and 's1' are temporarily used by some of the routines. However, it is vital that you do not use 'sF' register at any time during a transaction as it is used to control the I2C signals from the start through to the end of a transaction.

© Copyright 2012-2013 Xilinx

I2C Signalling ('i2c_routines.psm')

Lowest Level I2C Signalling

You would not need to invoke (CALL) the routines shown on this page; they are the very lowest level control of the I2C signals which are invoked by the next level of routines up contained in the same file. However, they do show how KCPSM6 controls the I2C bus using the output port.

```
CONSTANT I2C_clk, 00000001'b      ;Bit to which CLK is assigned on both ports
CONSTANT I2C_data, 00000010'b     ;Bit to which DATA is assigned on both ports
```

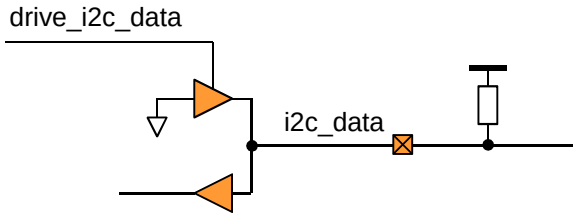
These two constants identify that bit0 is assigned to CLK and bit1 is assigned to DATA (see page 5)

Control of 'Data' Signal

```
I2C_data_Low: AND sF, ~I2C_data
              OUTPUT sF, I2C_output_port
              RETURN
```

When KCPSM6 sets the 'drive_i2c_data' signal Low ('0') it enables the output buffer which in turn drives the external I2C data signal Low ('0').

Hint - ~I2C_data = 11111101'b




```
I2C_data_Z: OR sF, I2C_data
            OUTPUT sF, I2C_output_port
            RETURN
```

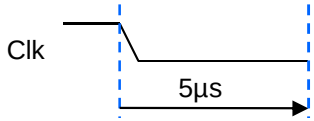
When KCPSM6 sets the 'drive_i2c_data' signal High ('1') it disables the output buffer so that it is high impedance ('Z'). The actual state of the external I2C data signal depends on the external components; the pull-up will result in a High level ('1') providing none of the slaves are driving the signal Low.

Control of 'Clk' Signal

KCPSM6 drives the I2C clock is the same way that it drives the I2C data. Likewise, the actual logic level of the external I2C clock signal will only be High ('1'0 when neither KCPSM6 or any slave is forcing the signal Low ('0'). However, the following routines do a bit more that just controlling the clock pin.

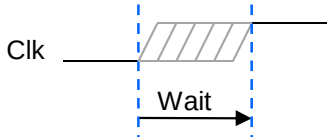
```
I2C_clk_Low: AND sF, ~I2C_clk
             OUTPUT sF, I2C_output_port
             CALL I2C_delay_5us
             RETURN
```

Driving the I2C clock Low ('0') is straightforward. The subsequent delay of 5µs ensures that the set-up time before any other operation related with the I2C signals is consistent with a data rate not exceeding 100k-bit/s.




```
        I2C_clk_Z: OR sF, I2C_clk
                   OUTPUT sF, I2C_output_port
I2C_wait_clk_High: INPUT s0, I2C_input_port
                   TEST s0, I2C_clk
                   JUMP Z, I2C_wait_clk_High
                   RETURN
```

KCPSM6 can only place the I2C clock output into high impedance ('Z') and must then wait for the external resistor to pull the signal High ('1'). This routine reads and tests the external clock signal until it observes a High level. Note that an I2C slave may deliberately hold the clock Low as a way to slow down communication (***clock stretching***).



© Copyright 2012-2013 Xilinx

I2C Signalling ('i2c_routines.psm')

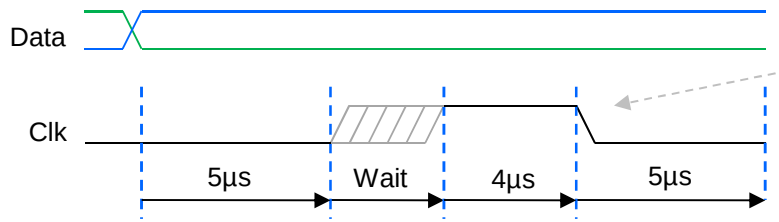
Transmitting ACK, NACK and bytes

There are three routines that you will invoke (CALL) associated with transmitting information from the KCPSM6 master to a slave. At the heart of any I2C transaction is the requirement to transmit bytes of information (i.e. data, address, control or combinations of these). Unsurprisingly, the transmission of a byte is simply the transmission of 8-bits with the only critical detail being that the information is transmitted most significant bit (MSB) first. When a master receives a byte of information from slave it is expected to acknowledge receipt of that byte by transmitting an 'ACK' bit to the slave. This is simply the transmission of one bit but with the fixed value of '0'. A master may also transmit a 'NACK' bit (value '1'). This 'no acknowledge' is most commonly used by the master to inform a slave that it should stop sending information (rather than an indication that the last byte was in any way corrupted).

Whether a bit to be transmitted from a master to a slave represents ACK, NACK or data, the requirement is for the master to present the bit on the 'data' signal and then, after allowing an adequate set-up time, to generate a Low to High transition on the 'clk' signal (the slave captures the data on the rising edge of the clock). The ACK and NACK routines shown below illustrate in detail how the data is presented 5µs before the rising edge of the clock (remembering that the slave has the ability to stretch the clock and the master must wait). Also, after meeting minimum timing requirements, the clock is returned Low in readiness for the next operation.

```
I2C_Tx_ACK: CALL I2C_data_Low
            JUMP I2C_clk_pulse
```

```
I2C_Tx_NACK: CALL I2C_data_Z
             JUMP I2C_clk_pulse
```

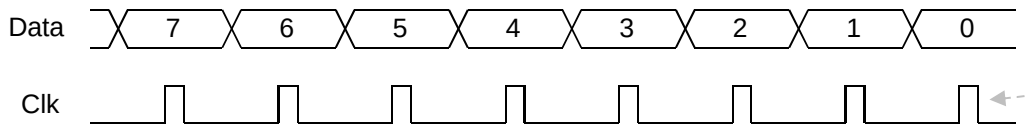



```
I2C_clk_pulse: CALL I2C_delay_5us
               CALL I2C_clk_Z
               CALL I2C_delay_4us
               CALL I2C_clk_Low
               RETURN
```

Transmission of a byte is simply a case of transmitting each bit ('0' or '1') MSB first.

```
    I2C_Tx_byte: LOAD s1, 10000000'b
I2C_Tx_next_bit: TEST s5, s1
                 JUMP NZ, I2C_Tx1
                 CALL I2C_data_Low
                 JUMP I2C_Tx_tsu
        I2C_Tx1: CALL I2C_data_Z
     I2C_Tx_tsu: CALL I2C_clk_pulse
                 SR0 s1
                 RETURN C
                 JUMP I2C_Tx_next_bit
```

The 'I2C_Tx_byte' routine transmits the byte value provided in register 's5'.



Note that each clock pulse effectively defines the timing (which may be stretched by the slave)

© Copyright 2012-2013 Xilinx

I2C Signalling ('i2c_routines.psm')

Receiving bits

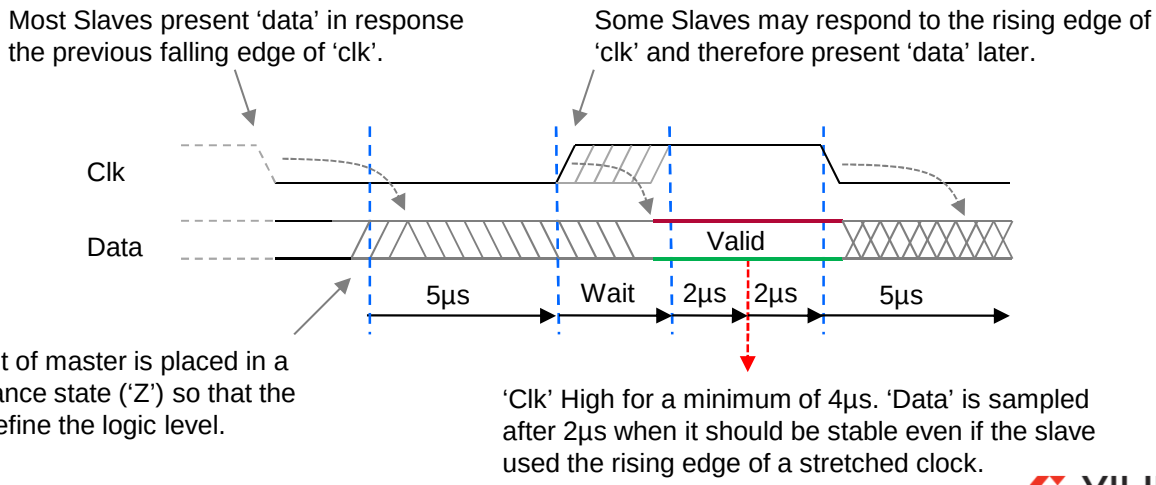
There are two routines that you will invoke (CALL) associated with receiving information from a slave device (see next page). Bytes of information (e.g. Data or control register values) consist of 8-bits that are read most significant bit (MSB) first. Each time the master transmits a byte of information to a slave then the slave is expected to respond with an 'ACK' (value of '0') which the master must receive. In this case the received 'ACK' bit normally has a logical meaning in that the reception of a 'NACK' ('no acknowledge' with value of '1') would typically indicate that the slave is not responding for some reason. Hence, it is common practice to test the state of the 'ACK' bit (or bits) during a transaction to confirm that the slave devices is responding.

To receive any bit of information from a slave device the KCPSM6 master must ensure that the 'data' output is placed into the high impedance state so that the logic level of the 'data' signal can be defined by the slave device and then read by the master. In a classic I2C arrangement the slave device would be expected to present a bit of information to the 'data' signal (i.e. a data bit or ACK) in response to a High to Low transition on the 'clk' signal generated by the master, and the master would be expected to read that bit at the next rising edge of the 'clk'. However, there are some slave devices that present data in response to the rising edge so always check data product sheets carefully.

Receiving a Bit (lower level routine)

The 'I2C_Rx_bit' routine shown below will receive one bit of data and is intended to be compatible with all slave devices regardless of which clock edge they respond to. The key observation being that the 'data' signal is sampled a suitable time after both a previous falling edge and rising edge of the 'clk' but definitely before the next falling edge is generated. However, it should be noted that this does increase the overall delays resulting a in a lower communication rate (i.e. this routine could be optimised for higher speed).

```
I2C_Rx_bit: CALL I2C_data_Z
            CALL I2C_delay_5us
            CALL I2C_clk_Z
            CALL I2C_delay_2us
            INPUT s0, I2C_input_port
            TEST s0, I2C_data
            SLA s5
            CALL I2C_delay_2us
            CALL I2C_clk_Low
            RETURN
```



© Copyright 2012-2013 Xilinx

I2C Signalling ('i2c_routines.psm')

Receiving ACK, NACK and bytes

You will probably need to invoke (CALL) the 'I2C_Rx_ACK' routine shown below in every I2C transaction that you implement. The 'I2C_Rx_byte' routine will be required in any transaction that involves reading information from a slave.

```
I2C_Rx_ACK: CALL I2C_Rx_bit
            TEST s5, 00000001'b
            RETURN
```

Your KCPSM6 master must read the 'ACK' bits produced by a slave as these bits are part of the fundamental I2C transaction format. Whilst you could choose to otherwise ignore these bits it is good practice to verify that the slave device is responding as expected (i.e. 'ACK' bits are '0' indicating a positive acknowledgement). For this reason the 'I2C_Rx_ACK' routine includes a 'TEST' instruction so that the carry flag is ready to use in a 'JUMP' instruction at the higher level.

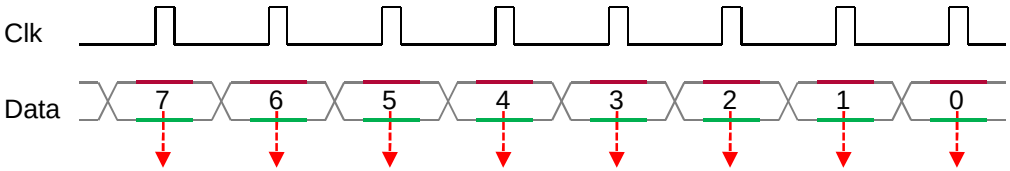
Received bit	Meaning	Carry Flag
'0'	ACK	NC
'1'	NACK	C

Hint – It is common practice to only check the first 'ACK' bit received in a transaction. If this confirms that communication with a slave has been established it is generally reasonable to assume that the remainder of the transaction to complete successfully.

Hint – When developing your code to implement transactions required for your slave devices then do initially include checks all ACK bits as this will help reveal any mistakes. Once your code is stable and working you may choose to streamline your code.

Receiving a byte is simply a case of receiving 8-bits MSB first.

```
    I2C_Rx_byte: LOAD s1, 8'd
I2C_Rx_next_bit: CALL I2C_Rx_bit
                 SUB s1, 1'd
                 JUMP NZ, I2C_Rx_next_bit
                 RETURN
```



The 'I2C_Rx_byte' routine returns the byte value in register 's5'.

© Copyright 2012-2013 Xilinx

I2C Signalling ('i2c_routines.psm')

Initialise, Start or Repeated Start (S or Sr) and Stop (P) All of these routines will be used in every transaction.

Initialise KCPSM6 register and the I2C Bus Signals

```
I2C_initialise: LOAD sF, I2C_clk
                OR sF, I2C_data
                OUTPUT sF, I2C_output_port
                RETURN
```

Prior to starting any I2C transaction it is vital that the states of both the 'clk' and 'data' signals are known state. Of equal importance, the contents of 'sF' must also be defined as this register is used to remember and define the drive states of the outputs states throughout a transaction.

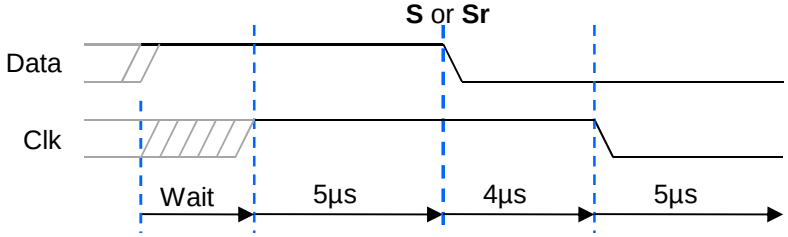
IMPORTANT - Do not use 'sF' in your own code whilst a transaction is in progress.

Start (S) or Repeated Start (Sr)

The master will initiate a bus transaction with a 'Start' (S) which will be recognised by all slaves connected to the bus. All slaves will then observe the device address and (all being well and correct) only the specified slave will be selected for communication with the remainder effectively going to sleep. Some transactions require the 'start' to be issued again and is known as a 'Repeated Start' (Sr). The routine provided and shown below can be used for both cases.

```
I2C_start: CALL I2C_data_Z
           CALL I2C_clk_Z
           CALL I2C_delay_5us
           CALL I2C_data_Low
           CALL I2C_delay_4us
           CALL I2C_clk_Low
           RETURN
```

'Start' (S) or Repeated Start (Sr) is a unique situation in which the 'Data' line is made to perform a High to Low transition whilst the 'Clk' is High.

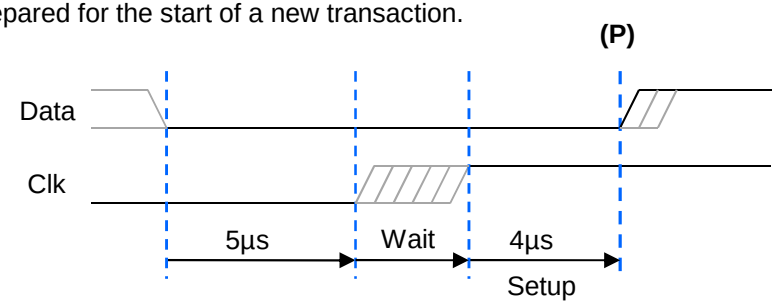


Stop (P)

The master will complete a transaction with a 'Stop' (P) which will be recognised by *all* slaves; not just the one that has been communicating with the master. In this way, all slaves are aware that the bus is returning to the idle state and should be prepared for the start of a new transaction.

```
I2C_stop: CALL I2C_data_Low
          CALL I2C_delay_5us
          CALL I2C_clk_Z
          CALL I2C_delay_4us
          CALL I2C_data_Z
          RETURN
```

'Stop' (P) which is another unique situation in which the 'Data' line is made to perform a Low to High transition whilst the 'Clk' is High.



© Copyright 2012-2013 Xilinx