

# **EEET2162 / 2035 – Advanced Digital Design 1 / Design with Hardware Description Languages – Lecture 9**

**Introduction to QSYS**

**Dr. Glenn Matthews**

# EEET2162 / 2035 - Course Update

- A reminder that 'Major Project' has been released and is due Friday, 25/05/2018 by 5:00pm.
  - The projects are quite complex and students are requested to ensure that they devote sufficient time developing both the technical outcomes and the supporting documentation.
  - The projects have been designed that they will nominally required 72 hours of work (36 hours from each group member).
  - As mentioned above, students can work in groups of two (same course code).
  - The marking guide will be released shortly.
- The additional hardware (HDMI monitors, function generators) are being sourced and will be available in the laboratories shortly.
  - In the meantime, the other interfaces and basic HDL communication blocks with the external ICs can be established.
  - Each major design block should be simulated in relative isolation (using ModelSim) and then verified on the actual Cyclone V SoC.
  - The DE-10 Nano Development Boards are available for day loan **ONLY** from 10.9.15.
    - Please ensure that the boards are returned before 4:30pm daily so your colleagues can gain access.

# Introduction to QSYS

- As discussed in previous lectures the Cyclone V SoC (System-on-Chip) consists of both a Hard Processor (HPS) and an Field Programmable Gate Array (FPGA).
  - The HPS is a dual core ARM Cortex-A9 (Application Processor) capable of operating at 925MHz.
  - The FPGA component (in the DE-10 Nano Development Board) contains up to 110k logic elements.
- Although this course has focused primarily focused on the FPGA aspects, the real advantage of the SoC is when both the HPS and FPGA are used together.
  - When utilising the SoC, the HPS can run an operating system (such as Linux or QNX) and then communicate with the FPGA component.
  - Specialised peripherals can be developed in the FPGA (such as an FFT Algorithm) and accessed via the HPS.
  - The basic read / write commands from the HPS can be emulated via ModelSim and hence the FPGA component can be tested in relative isolation.
  - To perform the mapping between the HPS and the FPGA another tool within Quartus is required named QSYS (System Integrator).
    - Note that in recent revisions of Quartus QSYS has been renamed to 'System Integrator'.

# Cyclone V SoC – General Architecture

- Figure 1 depicts the overall structure of the Cyclone V SoC.
  - The system consists of the ARM Cortex-A9 HPS which is interconnected via various interfaces and bridges.
  - Access to the FPGA is via three specific bridges that need to be configured before the system can be accessed.
  - Failure to configure the bridges correctly will cause the HPS to 'crash' and generate a segmentation fault.
  - The HPS includes a DDR SDRAM controller that can be accessed via the HPS as well as the FPGA.
    - This peripheral can be utilised to allow the FPGA to access the DDR3 RAM for data storage and manipulation.
  - The 'Boot-ROM' is used to perform basic initialisation and allow access to devices such as the SD / MMC Controller.

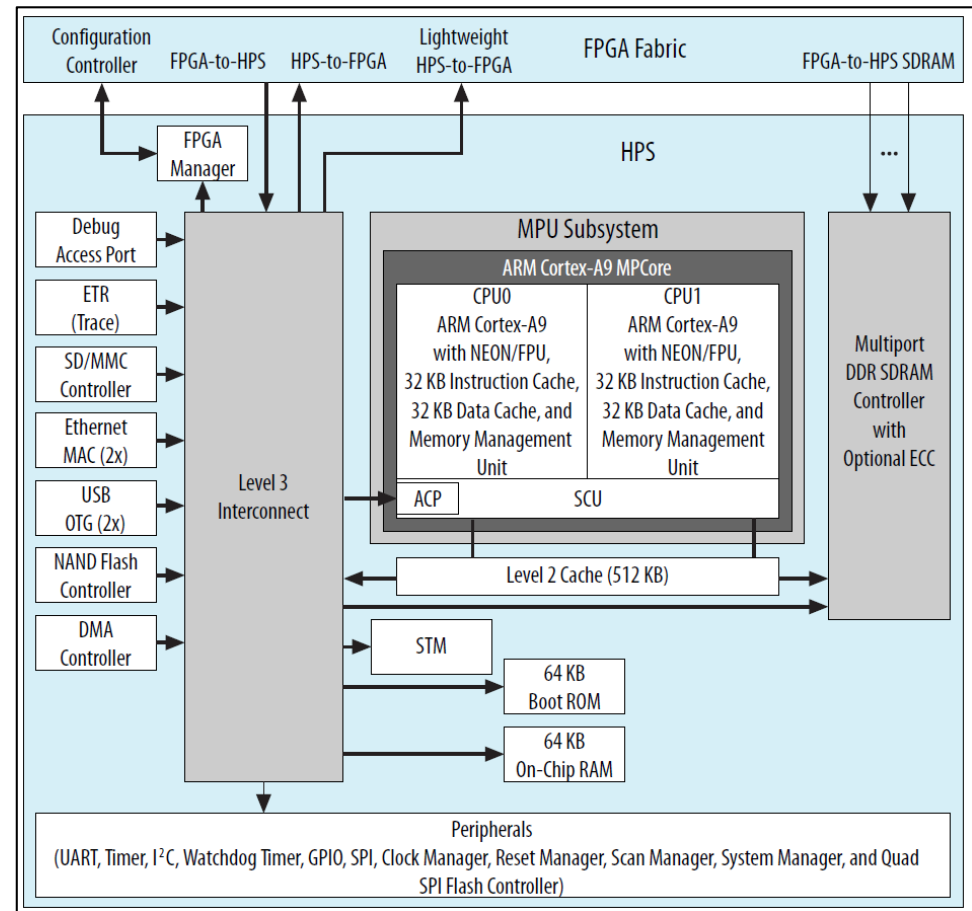
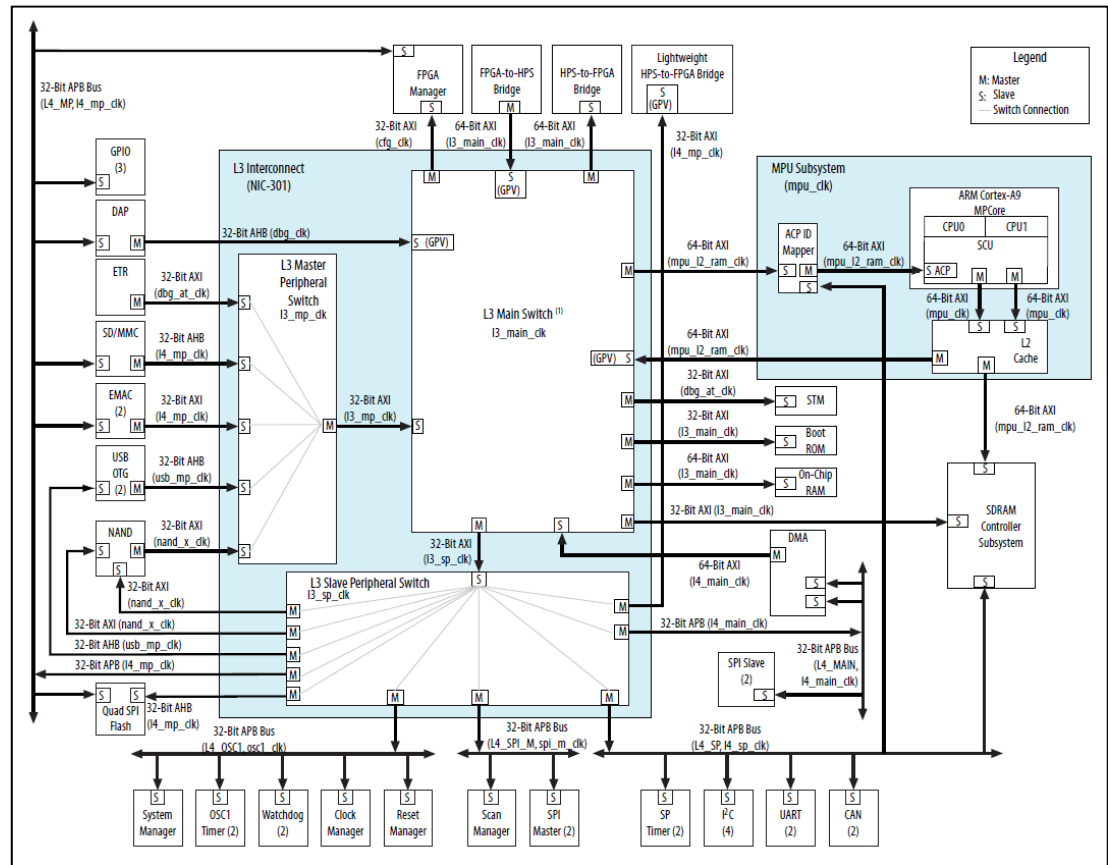


Figure 1 – Cyclone V SoC General Structure [1]

# Cyclone V SoC – System Interconnects

- To connect the various subsystems together various levels of interconnection are required.
    - Figure 2 illustrates the various buses and associated clocks from the HPS perspective.
    - The actual interconnect is based upon the ARM CoreLink Network Interconnect which provides several communication protocols such as the AMBA, AHB and APB systems.
    - The system is developed around the concepts of 'master' and 'slave' devices.
    - A master device controls the bus transaction, whilst the slave is designed to respond.
    - Note that the core CPU peripherals (System Manager, Watchdog, etc) are connected over the slower L4 bus.
- 



**Figure 2 – Cyclone V SoC System Interconnects [2]**



# Cyclone V SoC – Initialisation Process - 1

- Similar to most ARM processors, the various systems need to be configured before they can be accessed by the MPU (Microprocessor Unit).
  - Many of the system clocks are enabled by default which is different to microprocessors such as the Cortex-M4.
  - However, the FPGA components are not initialised and if an access is attempted, then the HPS will likely cause a segmentation fault and crash the system.
- The typical boot process for the HPS is described in Figure 3.

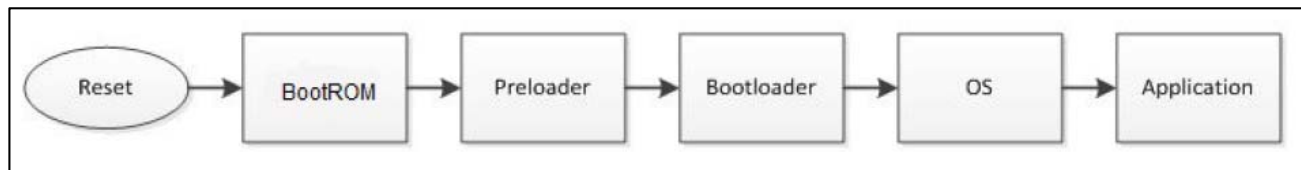


Figure 3 – Cyclone V SoC Boot Process[3]

- The BootROM is responsible for minimal HPS initialisation.
  - The SD Card interface is enabled which allows for the Preloader to be launched.
- In the case of the systems in the laboratory a custom port of U-boot has been created to support the Cyclone V SoC.
  - The first part of the U-Boot SPL (Secondary Program Loader) configures the SDRAM and performs basic HPS initialisation.
  - In the case of the Cyclone V SoC it **ONLY** enables one of the MPU cores.
  - The second core is configured and initialised via the operating system (QNX 7.0).

# Cyclone V SoC – Initialisation Process - 2

- Once the SDRAM interface is configured, the SPL is responsible for loading the Bootloader (U-Boot) from the SD Card to the DDR3 RAM.
  - The actual U-Boot binary is 360KB, and will not fit into the 64KB of onboard RAM.
  - A jump is then performed to U-Boot and the console can be accessed.
- U-Boot provides a minimal terminal interface which can then be used to initialise other core features (such as the FPGA) and then launch the main operating system kernel.
  - One of the benefits of utilising the U-boot terminal interface is that memory locations can be read / written without being concerned that a HPS driver is required to access the device.
  - This process is useful when debugging hardware at the register level.
  - Note that that access to the HPS JTAG port can be achieved via the 'USB Blaster' port with a tool known as OpenOCD.
    - OpenOCD can be downloaded from the following location: <http://openocd.org/>
    - This tool is not required, however can be useful for debugging kernel issues relating to the HPS.

```
U-Boot SPL 2017.05-rc1 (May 09 2017 - 15:30:09)
drivers/ddr/altera/sequencer.c: Preparing to start memory calibration
drivers/ddr/altera/sequencer.c: CALIBRATION PASSED
drivers/ddr/altera/sequencer.c: Calibration complete
Trying to boot from MMC1

U-Boot 2017.05-rc1 (May 09 2017 - 15:30:09 +1000)

CPU:      Altera SoCFPGA Platform
FPGA:     Altera Cyclone V, SE/A6 or SX/C6 or ST/D6, version 0x0
BOOT:     SD/MMC Internal Transceiver (3.0V)
Watchdog  enabled
I2C:      ready
DRAM:     1 GiB
MMC:      dwmmc0@ff704000: 0
In:        serial
Out:       serial
Err:       serial
Model:     Terasic DE0-Nano(Atlas)
Net:       eth0: ethernet@ff702000
Hit any key to stop autoboot:  0
=> 
```

Figure 4 – Cyclone V SoC U-Boot Console

# Cyclone V SoC – Initialisation Process - 3

- The final step in the initialisation process is to boot the main operating system kernel.
  - In EEET2166 (Real-time Operating Systems) the operating system utilised is QNX.
  - QNX is a lightweight industrial operating system that is highly configurable.
  - The driver base is quite minimal and hence the developer is required to custom develop the 'resource manager' code to support additional hardware.
  - QNX is centred around distributed processing and hence other nodes can be added in with relative ease.
  - The same operating system can be utilised for different boards such as the Beaglebone Black (TI AM335x) and the iMX.6 (NXP).
- Although the kernel has been launched the FPGA itself has not been configured.
  - In a 'production' environment the developer does not have the ability to load the FPGA configuration via the Quartus Programming tool.
  - Several options exist to load the configuration, including an serial ECPS device .
    - After the SoC is reset, the FPGA looks for a serial configuration device and if found loads the configuration file.
    - Alternatively the configuration can be uploaded to the FPGA via the U-boot bootloader.



# Cyclone V SoC – Initialisation Process - 4

- There are two different methods for uploading the configuration file to the FPGA.
  - Both methods utilise the HPS to communicate with the 'FPGA Manager' subsystem.
    - In both approaches the configuration file is loaded serially in 32-bit chunks.
  - The first approach (and preferred method) is to instruct U-boot to upload the .rbf (RAW Binary file) before the main operating system is launched.
    - In this approach the FPGA configuration is set before the Operating System (OS) is launched and hence the device would appear during the actual OS initialisation process.
  - The second approach is to allow the OS kernel to send the configuration (via a .rbf file) to the FPGA post boot.
    - The advantage of this approach is that it permits the FPGA to undergo a reconfiguration process should the task change.
    - A potential use for this may allow for dynamic 'firmware' upgrades.
    - The QNX build developed by the Course Coordinator does currently not allow this form of configuration, however it is under development.
- Although the FPGA configuration can be uploaded there is still a series of registers that need to be set within the HPS before the system can be accessed.

# Cyclone V SoC – FPGA Bridges - 1

- From Figures 2 and 5, there are three separate bridges that can be utilised to enable communication between the HPS and the FPGA.
  - FPGA-to-HPS Bridge
  - HPS-to-FPGA Bridge (Base address: 0xC0000000)
  - Lightweight HPS-to-FPGA Bridge (Base address: 0xFF200000).
- As their name suggests, each different bridge can be used for alternative functionality, however they are all configured via the Lightweight bridge.
  - The HPS2FPGA and FPGA2HPS bridges have developer configurable widths, while the LWHPS2FPGA is set at 32-bits.
  - Figure 4 depicts the various clock sources and master / slave interfaces.
  - The address range decoding is not fixed within the FPGA fabric and is the responsibility of the developer.

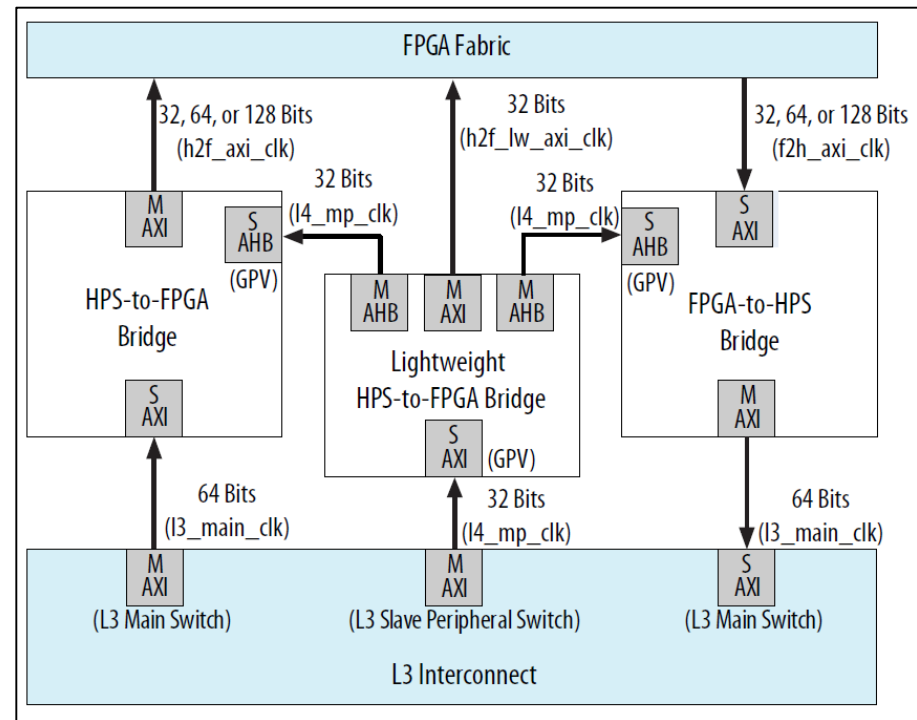


Figure 5 – Cyclone V FPGA Bridges [4]

## Cyclone V SoC – FPGA Bridges - 2

- In the following example, the FPGA fabric is accessed via the LWHPS2FPGA bus.
  - The bus width is set to 32-bits.
  - To enable access to the bridge (and prevent the HPS segmentation fault) the values in Table 1 must be written to the given memory addresses after the .rbf file has been loaded.
    - The loading process is automatically provided by U-boot assuming that the .rbf file is called **soc\_system.rbf**
  - To manually write the prescribed values via U-boot, the memory modify (mm) command can be utilised.

Memory Address	Alias	Value
0xFFD08028	SYSMGR->FPGA->MODULE	0x00000000
0xFFD0501C	RSTMGR->BRGMODRST	0x00000000
0xFF800000	L3REGS->REMAP	0x00000019

Table 1 – HPS / FPGA Bridge Configuration 'Unlock' Registers

# Cyclone V SoC – QSYS Example

- Many of the standard communication interfaces (SPI, I2C and UART) are not directly accessible via the exposed GPIO on the DE-10 Nano Development Kit.
  - It is up to the developer to incorporate these into the overall FPGA architecture should the developed system require that particular interface.
  - In many cases an 'IP Block' can be purchased which contains the required functionality.
  - However, as one of the primary purposes of the Cyclone V SoC is to develop new peripherals it is useful to understand how these systems can be created.
- The example considered is a reduced version of an I2C Controller that would be suitable to control the ADV7513 HDMI Transmitter.
  - This particular device is located on the DE-10 Nano Development Board.
  - A series of configuration words (up to 25) need to be written to the ADV7513 to enable the HDMI output to function correctly.
  - In the laboratories it is suggested to utilise a ROM and a state machine to send the values, however the HPS can also be applied.
  - The developed pseudo-I2C controller is only capable of transmitting data in 3-byte clusters and will not read from the device.
    - The approach utilised is very inefficient, however demonstrates the use of QSYS.

# Cyclone V SoC – ADV7513 I2C Communication - 1

- The first step in developing a custom peripheral within QSYS is to create the actual peripheral as a 'stand-alone' FPGA project.
  - This process is identical to previous projects.
  - In the case of the ADV7513 three bytes are required for each configuration option.
    - Byte 0 is the device address, (0x7A), byte 1 is the memory address of the ADV7513 and finally byte 2 is the actual payload.
- The I2C system is used extensively in multi-device systems where a wide variety of peripherals are required to be connected as in Figure 6.
  - I2C is a multi-master system, which allows multiple devices to communicate independently.
  - The physical I/O is open collector (open drain) and the bus is 'pulled-up' to a common voltage.
    - When the bus is idle it will be pulled to the voltage rail.
    - When a device wishes to communicate the bus is actively pulled low.
    - The resistors on the bus also limit the number of connected devices by effectively creating an 'RC' circuit with the other units.

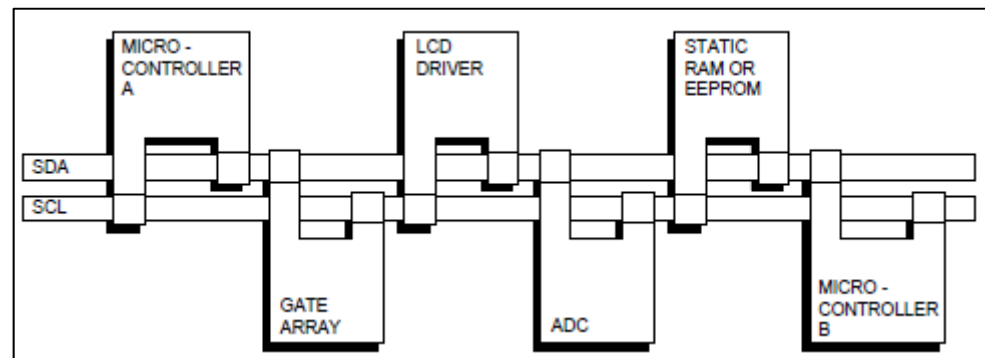


Figure 6 – I2C Bus Configuration [5]

# Cyclone V SoC – ADV7513 I2C Communication - 2

- At a minimum an I2C transaction requires a start-bit, the payload, an acknowledgment and the stop-bit to be transmitted.
  - Figure 7 depicts a standard transaction over an I2C bus.
  - Two signal lines are used for form the bus (SDA and SCL).
  - SDA is the bidirectional serial line, whilst SCL is the serial clock which is generated by the 'master' device.
  - When IDLE, the bus should float to the rail voltage.
  - The serial clock for I2C is generally fixed at 100kHz (Standard), 400kHz (Full), 1MHz (Fast) or 3.2MHz (High Speed).
- To signify a transaction the start condition needs to be generated.
  - This is achieved by a high to low transition on SDA whilst SCL is high.

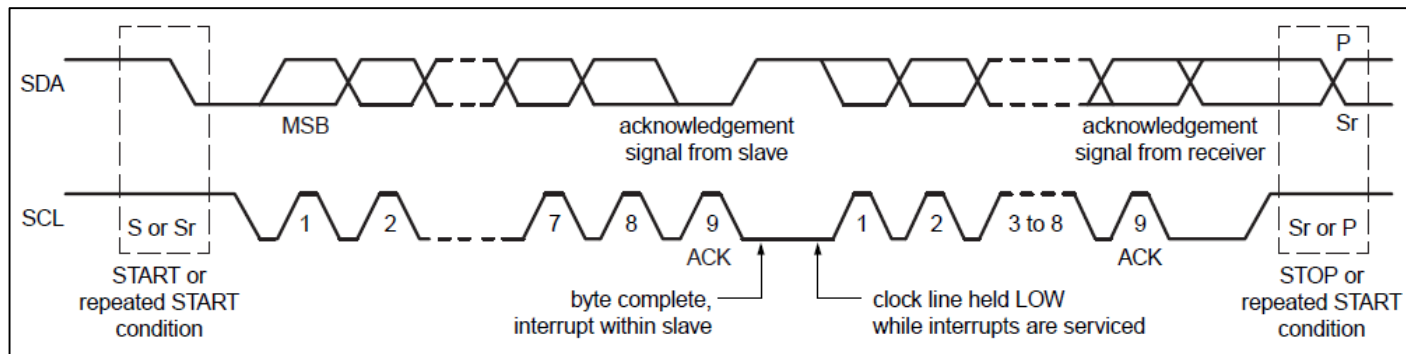


Figure 7 – I2C Bus Communication [6]



# Cyclone V SoC – ADV7513 I2C Communication - 3

- Once the start-bit has been transmitted, the data is shifted out via the SDA pin.
  - Data is shifted out MSB first and then the serial clock line toggled.
  - After all eight bits have been shifted out over the SDA line, an additional clock is generated to allow the slave device to acknowledge the transmission.
  - As the bus is open-collector, to signify the acknowledgement the SDA line is pulled low by the slave.
  - If the acknowledgement is received, then the next data byte can be clocked out.
    - A 'start' condition is not required for the next byte.
  - If the acknowledgement is not received, then a stop condition is sent and the process fails.
- Under normal circumstances the required number of bytes are shifted out and then the stop-bit issued.
  - Before another transaction can be sent, a new-start condition is required.
  - To simplify the design it is also possible to send a repeat-start condition if the stop has not been transmitted.
  - The stop-bit is signified by a low to high transition on SDA while SCL is high.

# Cyclone V SoC – Bidirectional Communication - 1

- The issue now is how to implement the actual peripheral in the FPGA and then instantiate it via QSYS.

- To create a bidirectional pin, an alternative port type known must be utilised:

```
inout SDA;    // Create a bidirectional port called SDA.
```

- The syntax to utilise the port is the same for the dedicated **input** / **output** ports.
  - However, care needs to be taken when using the port to communicate beyond the module in which it was created.
  - The key to resolving any potential bus contentions is to utilise a tri-state buffer as illustrated in Figure 8.
- Tri-state buffer are often used in bus systems where multiple devices need to transmit over the same data lines.
  - In this instance, if the OE (Output Enable) pin is low then the value at DATAIN is replicated at OUT0.
  - Conversely, in this example, if OE is high, then the output (OUT0) is in a high-impedance state.
    - Even though the buffer is considered to be 'off', there is still a slight loading on the bus.
  - Tri-state buffers should be used sparingly and where possible a multiplexer should be used instead.

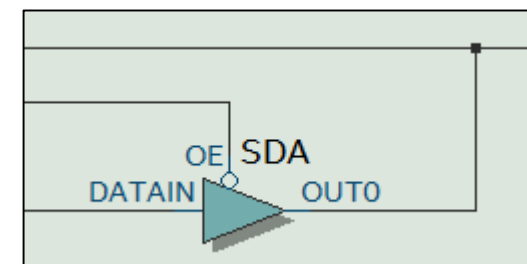


Figure 8 – Tri-state buffer

# Cyclone V SoC – Bidirectional Communication - 2

- To set an I/O to a tri-state value consider the following piece of code:
  - An 'if' statement is utilised to set the output enable of the buffer.

```
inout SDA;           // Create a bidirectional port called SDA.
reg i2cSDAOut;       // Tri-state control for SDA Pin.(output = 1).
reg SDAOut;          // Internal module output bit for SDA.

assign SDA = i2cSDAOut ? SDAOut : 1'bz; // Tristate control for the I2C SDA pin.
```

- In this instance, if i2cSDAOut is logic 1, the SDA I/O will be enabled and the value of SDAOut will be present.
  - Alternatively to read from the same I/O, the following code can be utilised:

```
reg slaveAckBit;     // Create a register variable called slaveAckBit.

i2cSDAOut = 1'b0;    // Set the tri-state buffer. (no output)
slaveAckBit = SDA;    // Read the value on SDA and store it in slaveAckBit.
```

- Equally, in ModelSim, the **noforce** command can be used in Tcl script to release the pin and place it in a high-impedance condition.
  - The timing requirements of when the I/O should be released need to be taken into consideration when using the SDA bit under tri-state conditions.

# Cyclone V SoC – I2C Communication - 1

- Now that the basics of the I2C protocol has been discussed, the next step in the process is to implement the ‘clocking’ behaviour.
  - The actual implementation is quite convoluted, however the basic steps are discussed here.
  - Although inefficient, one potential method to generate the clocking signal is to utilise a state machine.
- From Figure 9, it can be seen that data is placed on the bus and then the clock undergoes a complete transition from low – high and then back again.
  - This can be implemented by dividing each clock pulse into four separate states.
    - State 0 – Clock Low, State 1 – Clock High, State 2 – Clock High, State 3 – Clock Low.
  - Although the clock and data transitions could be achieved by utilising two states, the slave acknowledgement occurs during the high level clock transition.
  - If the clocking of the state machine is split into four sections this can be achieved quite readily.

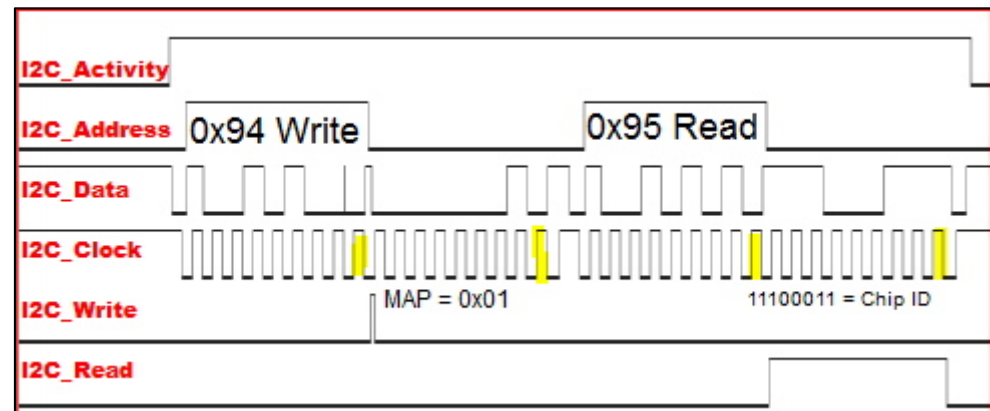


Figure 9 – Sample I2C Communication [7]

# Cyclone V SoC – I2C Communication - 2

- To assist with mapping the peripheral into a QSYS structure, a register map needs to be created.
  - A simple method to create the register layout is to utilise an address decoder (3 – 8 multiplexer) to control the individual output values from the D-type flip-flops.
  - Given that the I2C peripherals communicate on a byte level, the D-type flip-flops are also 8-bits wide.
  - Thus, the definition of the register is as follows:
  - The ‘preload’ values have been utilised to allow for registers to be set to default configurations.
  - For example, upon reset a microprocessor may set certain registers to known states.
  - The value ‘ld’ is utilised to latch data into the registers on the rising-edge of the clock.
  - Note that all outputs are considered (**q**) so there are no issues relating to inferred latches in this design.

```
module registerNbit (clk, clr_n, d, ld , q);
parameter WIDTH = 32;           // 32-bits wide.
parameter PRELOAD = 0;          // Preload = 0.
input [WIDTH-1:0] d;             // Data In
input clk, clr_n, ld;            // Clock, Reset, Load.
output reg [WIDTH-1:0] q;        // Output register

always @(posedge clk, negedge clr_n)
begin
    if(!clr_n) q <= PRELOAD;
    else
        begin
            if(ld) q <= d;
        end
end
endmodule
```

# Cyclone V SoC – I2C Communication - 3

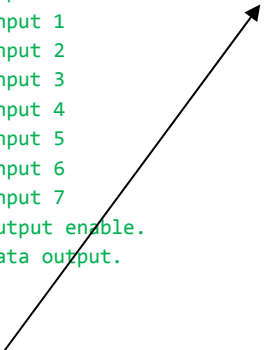
- The next aspect of the design is to create a technique that can be used to read from the various registers.
  - As mentioned previously writing to the individual registers can be achieved by controlling the 'ld' value.
  - A 3-to-8 multiplexer is utilised to select which one of the registers will be output back to the HPS.
  - A sample 3-8 decoding multiplexer appears below:

```
module mux8xNbit (sel, in0, in1, in2, in3, in4, in5, in6, in7, oe, q);

parameter WIDTH = 32; // Default 32-bits wide.
input [2:0] sel; // Multiplexer output select.
input [WIDTH-1:0] in0; // Input 0
input [WIDTH-1:0] in1; // Input 1
input [WIDTH-1:0] in2; // Input 2
input [WIDTH-1:0] in3; // Input 3
input [WIDTH-1:0] in4; // Input 4
input [WIDTH-1:0] in5; // Input 5
input [WIDTH-1:0] in6; // Input 6
input [WIDTH-1:0] in7; // Input 7
input oe; // Output enable.
reg [WIDTH-1:0] q_out; // Data output.
output [WIDTH-1:0] q;

assign q = oe ? WIDTH-1'b0 : q_out;

// Multiplexer that only depends on the output enable (oe) pin being set.
always @*
begin
    case (sel)
        3'h0: q_out = in0;
        3'h1: q_out = in1;
        3'h2: q_out = in2;
        3'h3: q_out = in3;
        3'h4: q_out = in4;
        3'h5: q_out = in5;
        3'h6: q_out = in6;
        3'h7: q_out = in7;
        default: q_out = 0;
    endcase
end
endmodule
```



- The actual mapping to addresses is achieved by the 'always' statement in the parent entity.



# Cyclone V SoC – QSYS Signals - 1

- Before proceeding to the actual address mapping it is important to consider the various signals that are presented to the designer via QSYS.
  - Irrespective of the FPGA bridge utilised, the same basic signals are available.
  - QSYS also creates the various higher-level address spaces and tri-state buffering for the various FPGA ‘peripherals’ to communicate back to the HPS.
  - Table 2 lists the essential signals, their width and corresponding direction from the HPS perspective.
  - The waveforms in Figure 10 demonstrate the required timing for both a read and write also from the HPS perspective.
  - These waveforms can be generated in ModelSim to ensure that the module(s) developed meet the timing requirements.
    - Note that the timing can be adjusted for various wait states, however it is suggested to leave it as default.

Signal	Direction	Width (bits)
write	HPS->FPGA	1
read	HPS->FPGA	1
writedata	HPS->FPGA	32
readdata	FPGA->HPS	32
clk	FPGA->HPS	1
address	HPS->FPGA	Dependent on Bridge

Table 2 – Exposed QSYS Signals

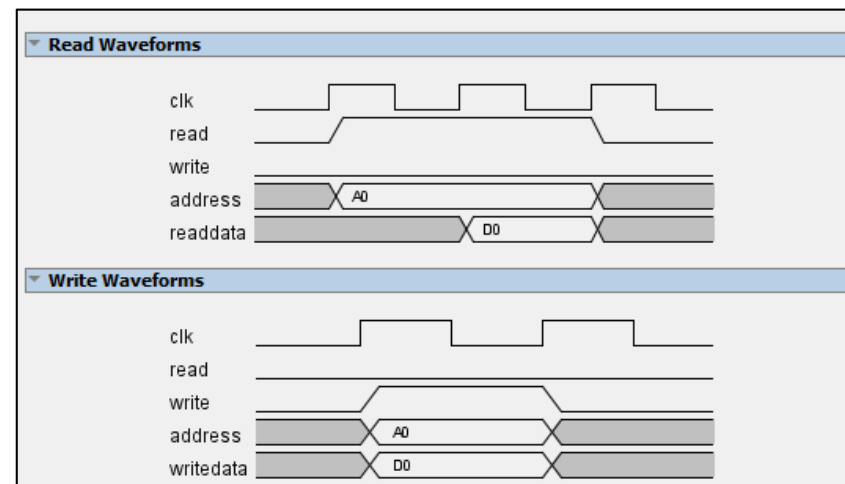


Figure 10 – Read / Writing Timing Waveforms

## Cyclone V SoC – QSYS Signals - 2

- Therefore, to create the registers and the address map the signals 'datawrite', 'write' 'dataread' and 'read' are required.
  - Given that the system to be designed is 8-bits, each required address is only 1-byte wide.
    - If larger registers (more bits) were required, then the register address offsets would also need to change.
    - Note that this register map does not take into consideration the HPS 'stride' and the ability to access non-aligned memory addresses.
  - Table 3 lists the required register map to be created and the corresponding descriptions.

Register Address	Alias	Description
0x00	rI2C_SETUP	Setup Register for I2C Operations (Clock Speed)
0x01	rI2C_STATUS	Status Register (Slave Ack)
0x02	rI2C_ADDRESS	I2C Address - Byte 0
0x03	rI2C_PAYLOAD0	I2C Payload 0 - Byte 1
0x04	rI2C_PAYLOAD1	I2C Payload 1 - Byte 2
0x05	rI2C_TRANSMIT	Transmit Register (Address Sensitive)

Table 3 – I2C Controller Register Map

- As only six addressable registers are required, the bottom three bits of the incoming address is required.
  - Using the vector notation the bits of interest can be extracted to form a 3-bit wire.

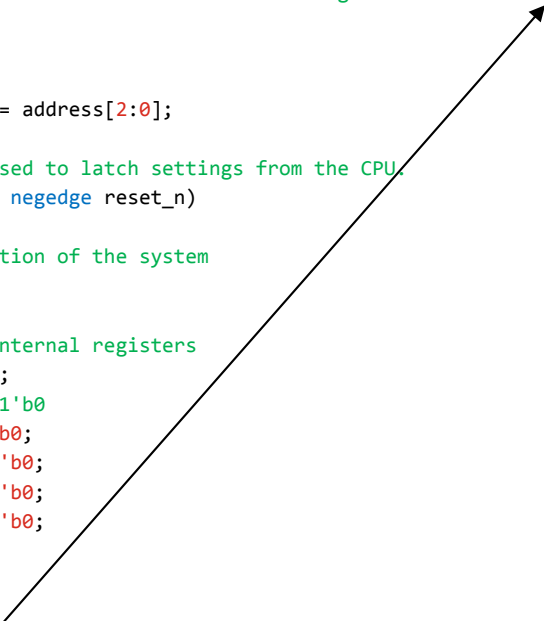
# Cyclone V SoC – QSYS Signals - 3

- Assuming that all of the registers have been instantiated, the address map (and the loading of the registers) is achieved by using an 'always' block in the register file entity.
  - Note that the reset behaviour is active low and has been propagated through via the top-level entity.
  - Furthermore, the status register (rI2C\_STATUS) is slightly more complicated as it operates in reverse (set by FPGA Peripheral).

```
// A 4-bit address can be used to communicate with the register bank
wire [2:0] internalAddress;

// Assign the internal logic
assign internalAddress[2:0] = address[2:0];

// Register file behaviour used to latch settings from the CPU.
always @(posedge busClock or negedge reset_n)
begin
    // Decide on the reset action of the system
    if(reset_n == 0)
    begin
        // Reset all of the internal registers
        rI2C_SETUP_ld <= 1'b0;
        // rI2C_STATUS_ld <= 1'b0
        rI2C_ADDRESS_ld <= 1'b0;
        rI2C_PAYLOAD0_ld <= 1'b0;
        rI2C_PAYLOAD1_ld <= 1'b0;
        rI2C_TRANSMIT_ld <= 1'b0;
    end
    else if(reset_n == 1)
    begin
        // The system is not in reset, so now provide the
        // functionality to load the individual registers.
        if((write == 1) && (read == 0))
        begin
            case (internalAddress)
                3'h0: rI2C_SETUP_ld <= 1'b1;
                3'h1: rI2C_STATUS_ld <= 1'b1;
                3'h2: rI2C_ADDRESS_ld <= 1'b1;
                3'h3: rI2C_PAYLOAD0_ld <= 1'b1;
                3'h4: rI2C_PAYLOAD1_ld <= 1'b1;
                3'h5: rI2C_TRANSMIT_ld <= 1'b1;
            endcase
        end
        else if((write == 0) && (read == 0))
        begin
            rI2C_SETUP_ld <= 1'b0;
            // rI2C_STATUS_ld <= 1'b0
            rI2C_ADDRESS_ld <= 1'b0;
            rI2C_PAYLOAD0_ld <= 1'b0;
            rI2C_PAYLOAD1_ld <= 1'b0;
            rI2C_TRANSMIT_ld <= 1'b0;
        end
    end
end
```



# Cyclone V SoC – QSYS Signals - 4

- Once the entire code for the peripheral has been tested it can now be simulated in ModelSim.
  - To simplify the process a set of functions (procedures) can be created to emulate both a bus read / write from the HPS.
  - It is also possible to simulate the actual HPS component, however that is not necessary for this example.
  - The two functions to 'read' and 'write' to the emulated HPS Bridge are demonstrate below.

```
# Create a function to write to the data bus at  
# a given address.
```

```
proc writeBus {addressValue busValue} {  
    force -freeze address 16#$addressValue  
    force -freeze writeData 16#$busValue  
    force -freeze write 1  
    run 20000  
    force -freeze write 0  
    run 20000  
}
```

```
# Release the data and address buses.
```

```
noforce writeData  
noforce address
```

```
# Create a function to read from an register to the data bus at  
# a given address.
```

```
proc readBus {addressValue} {  
    force -freeze address 16#$addressValue  
    force -freeze write 0  
    force -freeze read 1  
    run 20000  
    force -freeze read 0  
    run 20000  
}
```

```
# Release the address bus.
```

```
noforce readData  
noforce address
```

```
# Write the value 0x0000007A to 'memory location' 0x02.
```

```
writeBus 02 0000007A
```

```
# Read the 32-bit value from 'memory location' 0x02.
```

```
readBus 02
```

# I2C Controller – ModelSim HPS Interface - 1

- Before the example can be compiled to run under QSYS a full simulation should be performed to confirm it meets the required functionality.
  - Figure 11 depicts the ModelSim simulation as three data values are loaded into the address and payload registers.
  - Note: 0x02 (Address) = 0x7A, 0x03 (Payload0) = 0x98 and 0x00 (Payload1) = 0x31.
  - Address 0x05 is written to as a method to start the I2C Transfer.

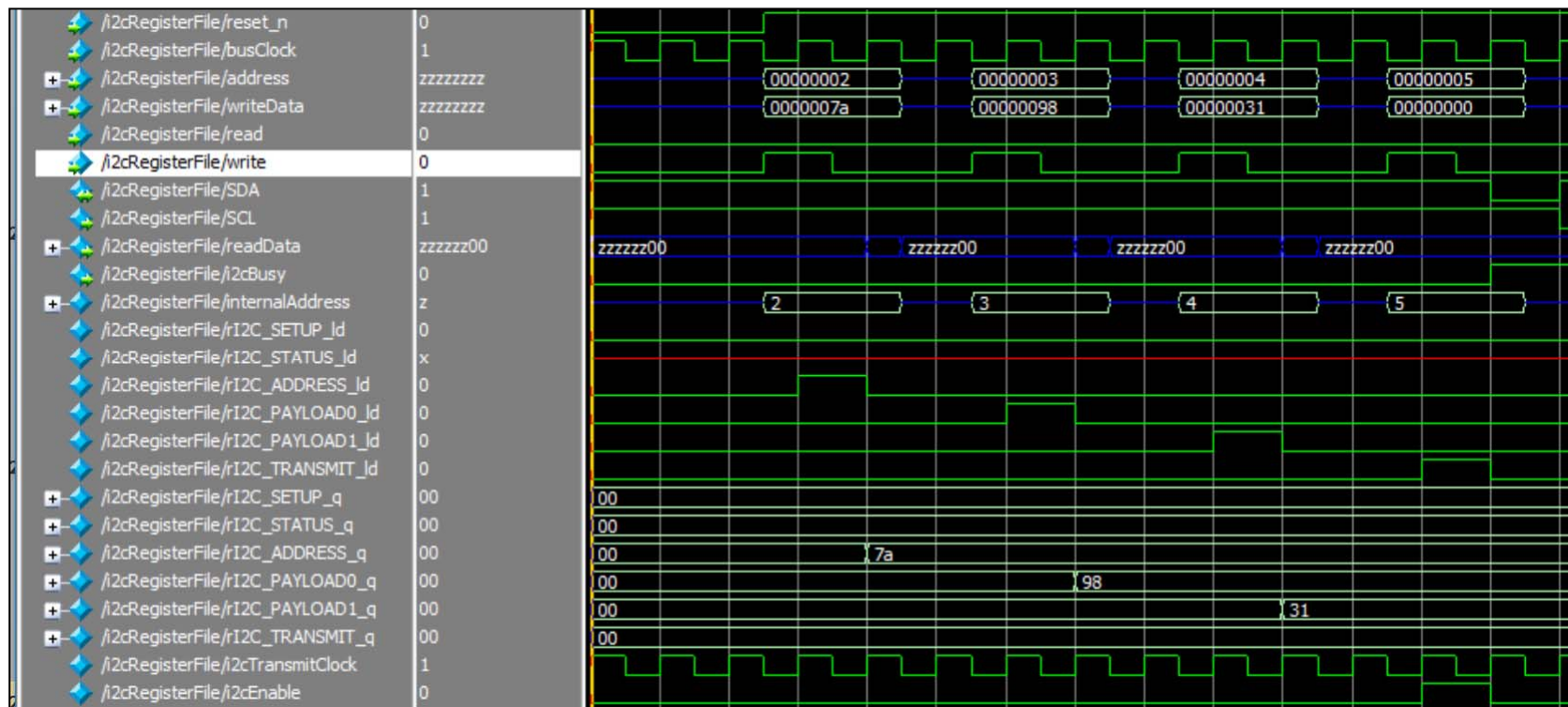


Figure 11 – I2C Controller ModelSim Data Load Verification

## I2C Controller – ModelSim HPS Interface - 2

- Although the data has been successfully latched into the I2C register file, the functionality of the actual system should also be confirmed.
  - The system clock has not been divided down to a suitable I2C frequency (currently 50MHz / 4).
    - A PLL / counter could be utilised to divide the clock down.
    - A separate clock for the HPS and the I2C SCL should be utilised.
  - Figure 12 depicts the first of the three data values previously loaded being transmitted over the bus.
  - The acknowledgement from the slave device was achieved by setting a signal at the corresponding time index via ModelSim.
  - An additional bit (i2cBusy) has been utilised to indicate that the I2C bus is currently transmitting data.
    - This bit could be utilised by a higher-level state machine as a control signal.
  - Note that in ModelSim, a red trace indicates an undefined signal, whilst a blue signifies that the value is high impedance (or non-forced).

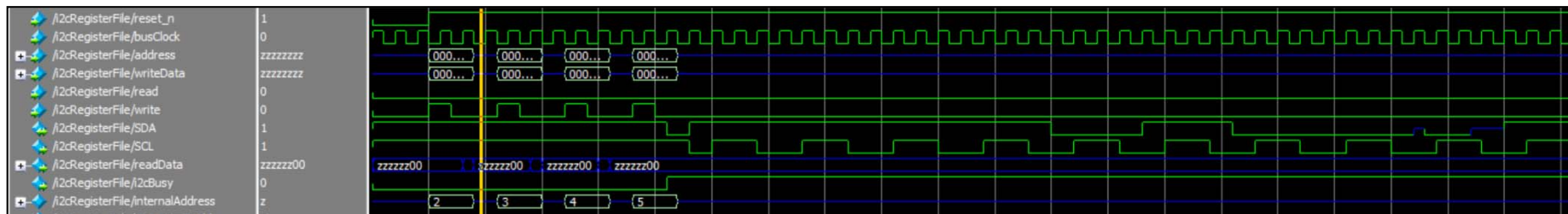
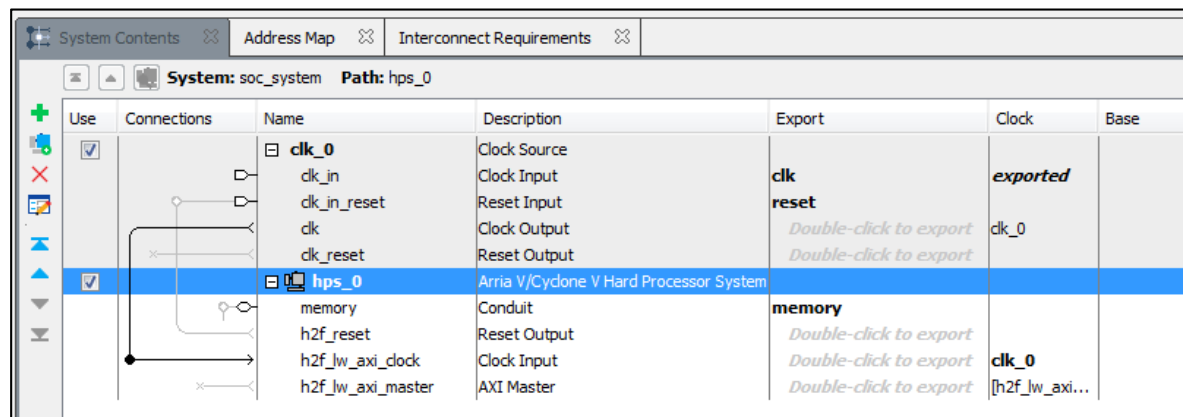


Figure 12 – I2C Controller ModelSim Data Transmit Verification



# QSYS Implementation - Template

- Now that the actual stand-alone design is verified the Verilog code can be incorporated into QSYS.
  - To assist with the creation of QSYS projects, a sample template will be uploaded to the Canvas Website.
  - It is suggested that a new project is created when porting the design across to QSYS.
- QSYS can be launched via the 'Tools->QSYS' menu option in Quartus.
  - Once QSYS has launched, the reference design 'soc\_system.qsys' template can be opened.
  - The default configuration options appear in Figure 13.
  - A clock from the FPGA to HPS (for the bridge access) is included as clk\_0 as well as the HPS component termed hps\_0.
  - Note that the default template requires additional configuration before it will compile.













Use	Connections	Name	Description	Export	Clock	Base
<input checked="" type="checkbox"/>		clk_0	Clock Source			
<input checked="" type="checkbox"/>		clk_in	Clock Input	clk	exported	
<input checked="" type="checkbox"/>		clk_in_reset	Reset Input	reset		
<input checked="" type="checkbox"/>		clk	Clock Output	Double-click to export	clk_0	
<input checked="" type="checkbox"/>		clk_reset	Reset Output	Double-click to export		
<input checked="" type="checkbox"/>		hps_0	Arria V/Cyclone V Hard Processor System			
<input checked="" type="checkbox"/>		memory	Conduit	memory		
<input checked="" type="checkbox"/>		h2f_reset	Reset Output	Double-click to export		
<input checked="" type="checkbox"/>		h2f_lw_axi_clock	Clock Input	Double-click to export	clk_0	
<input checked="" type="checkbox"/>		h2f_lw_axi_master	AXI Master	Double-click to export	[h2f_lw_axi...	

Figure 13 – QSYS Default HPS Configuration

# QSYS Implementation – HPS Settings - 1

- The default template includes a 'stub' for the LWHPS2FPGA bridge.
  - Recall that this bridge is set to operate using 32-bit bus transactions.
  - The stub needs to be configured before it can be utilised.
  - Double-clicking the HPS object will launch the dialog in Figure 14.
  - Whilst the system is highly configurable, the important option is to ensure that the correct bridges are enabled (under AXI Bridges).

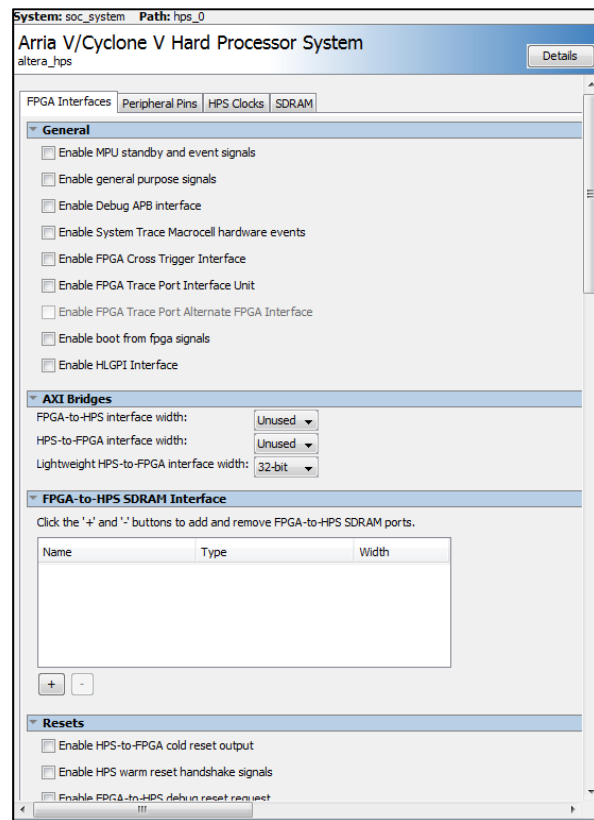


Figure 14 – QSYS Bridge Configuration

- If using the Lightweight bridge, then ensure that that dialog appears as in Figure 14.
- It is also possible to enable multiple bridges and adjust the widths as necessary.
- Once the HPS is configured, two additional connections are presented (Figure 15).
  - l2f\_lw\_axi\_clock – Clock used for the Lightweight bridge.
  - h2f\_lw\_axi\_master – Master interface used to communicate with the developed peripheral.
  - Note these names will change if any of the other bridges are utilised.

hps_0			
memory	Conduit	memory	
h2f_reset	Reset Output	Double-click to export	
h2f_lw_axi_clock	Clock Input	Double-click to export	clk_0
h2f_lw_axi_master	AXI Master	Double-click to export	[h2f_lw_axi_clock]

Figure 15 – QSYS Default HPS Configuration

# QSYS Implementation – Peripheral Configuration

- Once the bridge is configured, the actual HDL developed can be converted into a peripheral that the HPS can access.
  - The core files from the HDL only project should be copied into the new project.
  - A new component can be created from the 'IP Catalog->Project->New Component' window within QSYS.
  - The component editor appears as illustrated in Figure 16.
  - The core details of the component should be entered as this will also assist with version control.
  - As part of the development process QSYS generates a Tcl script to represent the new component.
  - If the project is moved to another computer, the entire directory structure is required to ensure that the component can be recreated.
  - After the details have been entered the actual model HDL needs to be imported via the 'Files' tab.

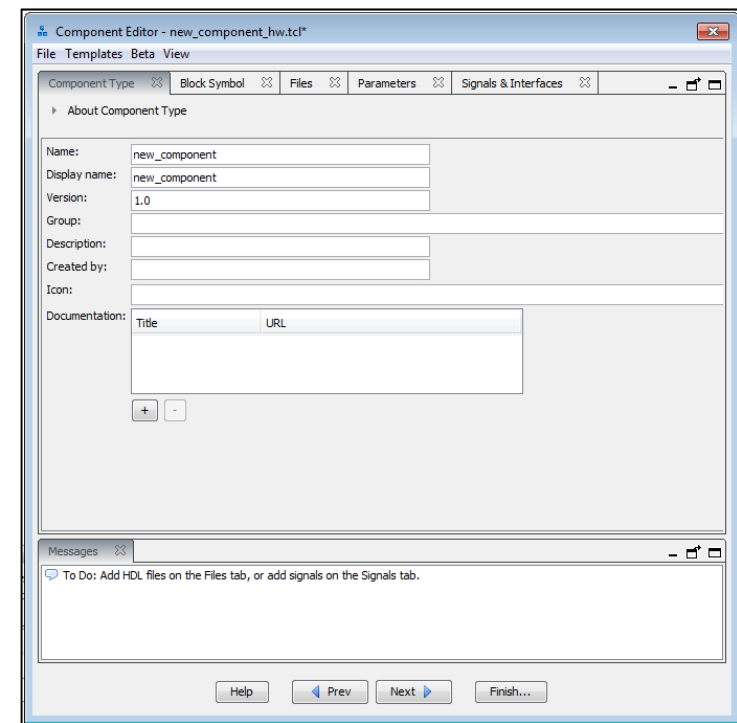


Figure 16 – QSYS Component Editor Dialog

# QSYS Implementation – HDL Signals - 1

- The files imported should include the TLE which includes the HPS bridge signals as per Figure 17.
  - Clicking on the 'Analyze Synthesis Files' button will allow QSYS to generate the block symbol as well as the data on the 'Signals & Interfaces' tab (Figure 18).
  - When the signals are imported the actual settings are incorrect and will not synthesise correctly.
    - Although the names are correct the actual types need to be configured correctly.

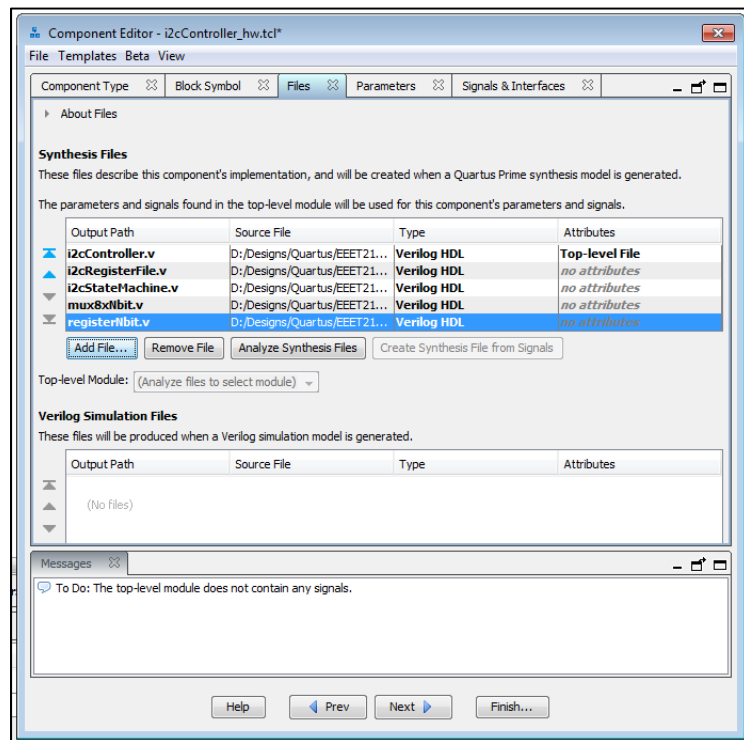


Figure 17 – QSYS HDL Import

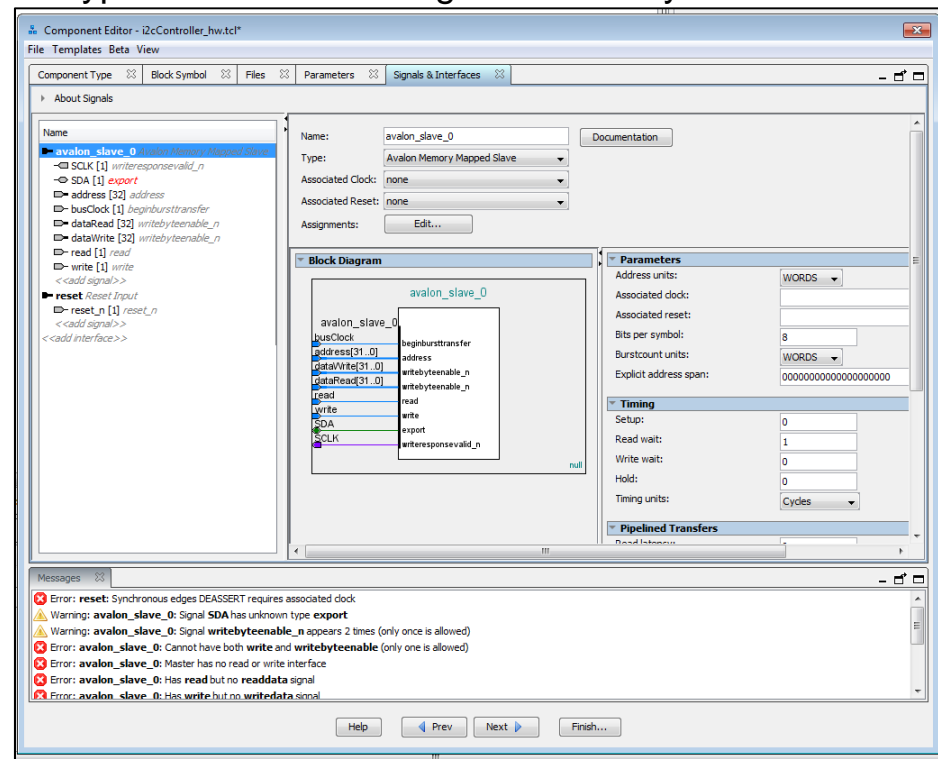


Figure 18 – QSYS Default Signal Mapping

## QSYS Implementation – HDL Signals - 2

- For each of the signals generated, the configuration options need to be changed.
  - A new interface (<< add interface >>) needs to be added for the clock of type **Clock Input**.
    - The signal **busClock[1]** is then 'dragged' under the new interface and the signal type set to clk.
  - The **dataRead[32]** signal type is set to **readdata** and the direction is an **output**.
  - The **dataWrite[32]** signal type is set to **writedata** and the direction is an **input**.
  - **address** is set to be four bits wide.
- The next issue is mapping the SDA and SCLK signals beyond the QSYS implementation.
  - A new interface (<< add interface >>) of type Conduit is added in.
    - The associated clock is '**clock\_sink**' and the associated reset is **reset**.
    - The SDA and SCLK signals are dragged to the conduit\_end interface.
    - For the SDA signal, the signal type is export\_sda with a 'bidir' direction.
    - Alternatively the SCLK signal type is export\_sclk with an 'output' direction.
- The final set is to set the clock and reset for the avalon\_slave\_0 interface.
  - The associated clock is '**clock\_sink**' and the reset is '**reset**'
  - Under the reset interface, also ensure that the clock is set to '**clock\_sink**'.

# QSYS Implementation – HDL Signals - 3

- At the end of the configuration, the 'Signals & Interfaces' tab should appear as in Figure 19.
  - Clicking 'Finish' will finalise the actual component.
- To instantiate the QSYS component, it needs to be added into the project 'System Contents' pane.
  - Double-clicking the component will create a new instance.
  - Although now as part of the QSYS implementation it now needs to be connected to the various interfaces.
  - The dark lines in the QSYS pane indicate the current connectivity.
  - A large black circle indicates that a node where the device is connected.
  - In this instances we need to connect the **'reset'**, **'avalon\_slave\_0'**, **clock\_sink** and **'conduit\_end'** nodes.

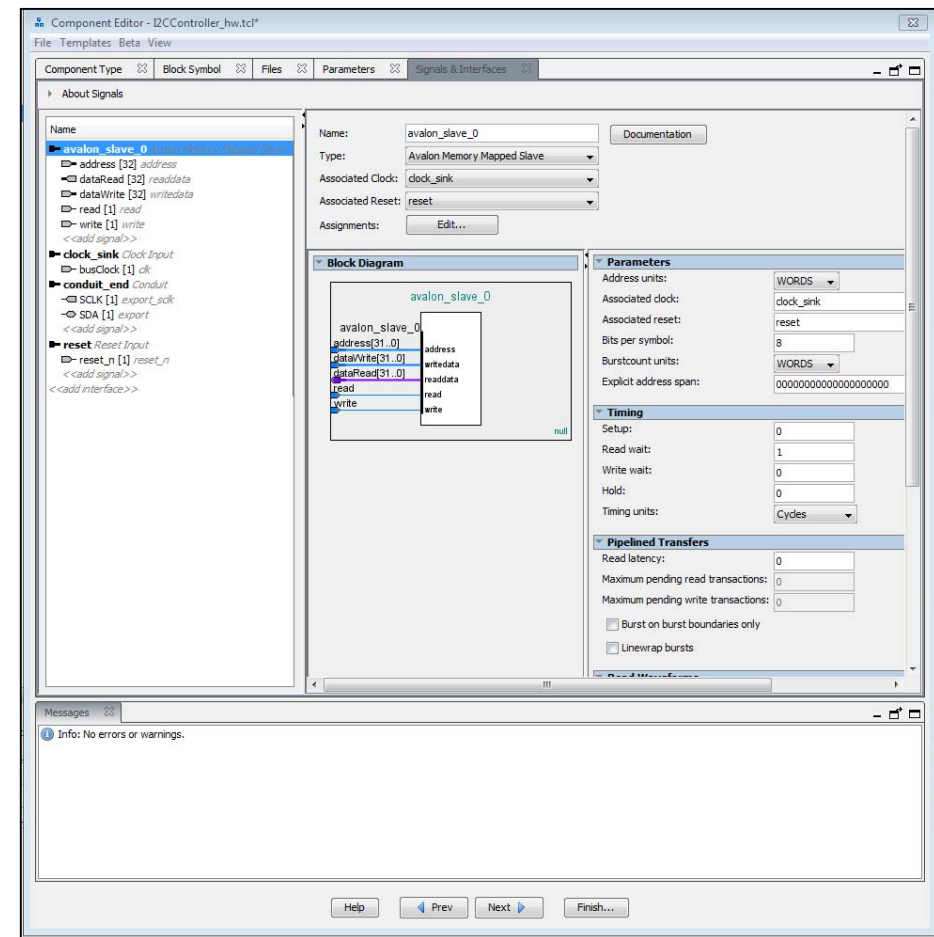


Figure 19 – QSYS Default HPS Configuration



# QSYS Implementation – HPS Mapping

- The connections for the developed component are as follows (Figure 20) :
  - I2CController\_0-reset -> clk\_0-lk\_reset
  - I2CController\_0-avalon\_slave\_0 -> hps\_0-h2f\_lw\_axi\_master
  - I2CController-clock\_sink -> clk\_0-clk
- Finally, the conduit\_end (which represents the SDA and SCLK) pins needs be routed.
  - The conduit needs to be exported via double-clicking in the 'export' column.
  - The can be exported as I2C\_conduit\_end.
  - The last set is to generate the corresponding HDL from the 'Generate->Generate HDL' menu item.

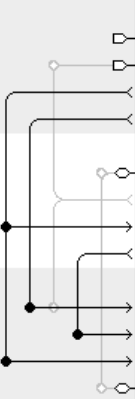
Use	Connections	Name	Description	Export	Clock	Base	End
<input checked="" type="checkbox"/>		<b>clk_0</b> clk_in clk_in_reset clk clk_reset	Clock Source Clock Input Reset Input Clock Output Reset Output	<b>clk</b> <b>reset</b> <i>Double-click to export</i> <i>Double-click to export</i>	<i>exported</i> clk_0		
<input checked="" type="checkbox"/>		<b>hps_0</b> memory h2f_reset h2f_lw_axi_clock h2f_lw_axi_master	Arria V/Cyclone V Hard Processor System Conduit Reset Output Clock Input AXI Master	<b>memory</b> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	<b>clk_0</b> [h2f_lw_axi_clock]		
<input checked="" type="checkbox"/>		<b>I2CController_0</b> reset avalon_slave_0 clock_sink conduit_end	I2C Controller Reset Input Avalon Memory Mapped Slave Clock Input Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <b>i2c_conduit_end</b>	[clock_sink] [clock_sink] <b>clk_0</b> [clock_sink]	0x0000_0000	0x0000_001f

Figure 20 – QSYS HPS Mapping

# QSYS Implementation – HPS Implementation

- The compilation for the QSYS process can take several minutes depending on the actual machine utilised.
  - If any errors are detected at this point, then the QSYS implementation needs to be corrected before proceeding.
  - At the end of the compilation the IP Variation (**soc\_system.qip**) should be added into the project.
  - A top-level entity can now be created to instantiate the HPS variation.
    - Note that a template appears on the Canvas website.

```
module i2cQSYS(  
    // Top Level Inputs into the system  
    input  CLOCK_50,    /// System Clock  
    output [14:0] hps_memory_mem_a,  
    output [2:0]  hps_memory_mem_ba,  
    output        hps_memory_mem_ck,  
    output        hps_memory_mem_ck_n,  
    output        hps_memory_mem_cke,  
    output        hps_memory_mem_cs_n,  
    output        hps_memory_mem_ras_n,  
    output        hps_memory_mem_cas_n,  
    output        hps_memory_mem_we_n,  
    output        hps_memory_mem_reset_n,  
    inout  [39:0] hps_memory_mem_dq,  
    inout  [4:0]  hps_memory_mem_dqs,  
    inout  [4:0]  hps_memory_mem_dqs_n,  
    output        hps_memory_mem_odt,  
    output [4:0]  hps_memory_mem_dm,  
    input        hps_memory_oct_rzqin,  
  
    // I2C Specific I/O  
    output        i2c_sclk,  
    inout         i2c_sda,  
    input  [3:0]  SW  
);  
  
    // Set a wire indicating the bridge clock.  
    wire main_clk = CLOCK_50;  
  
    // Create an instance of the SoC System.  
    soc_system soc(  
        .memory_mem_a      (hps_memory_mem_a),  
        .memory_mem_ba     (hps_memory_mem_ba),  
        .memory_mem_ck     (hps_memory_mem_ck),  
        .memory_mem_ck_n   (hps_memory_mem_ck_n),  
        .memory_mem_cke    (hps_memory_mem_cke),  
        .memory_mem_cs_n   (hps_memory_mem_cs_n),  
        .memory_mem_ras_n  (hps_memory_mem_ras_n),  
        .memory_mem_cas_n  (hps_memory_mem_cas_n),  
        .memory_mem_we_n   (hps_memory_mem_we_n),  
        .memory_mem_reset_n (hps_memory_mem_reset_n),  
        .memory_mem_dq     (hps_memory_mem_dq),  
        .memory_mem_dqs    (hps_memory_mem_dqs),  
        .memory_mem_dqs_n  (hps_memory_mem_dqs_n),  
        .memory_mem_odt    (hps_memory_mem_odt),  
        .memory_mem_dm     (hps_memory_mem_dm),  
        .memory_oct_rzqin  (hps_memory_oct_rzqin),  
        );  
  
    // Additional Signals required for I2C Controller.  
    .clk_clk (main_clk),  
    .reset_reset_n(SW[0]),  
    .i2c_conduit_end_export_sclk(i2c_sclk),  
    .i2c_conduit_end_export(i2c_sda),  
endmodule
```

# QSYS Implementation – HPS Settings

- Similar to all FPGA projects once the TLE has been created the 'Analysis and Synthesis' process should be invoked.
  - This will ensure that the design will at least compile and provide Quartus with a list of I/Os that need to be mapped.
  - Fortunately the mapping for the HPS is achieved via a Tcl script.
  - Once the 'Analysis and Synthesis' step is successful, the mapping script can be invoked.
- To enable the pin mapping for the HPS, from the Quartus menu select 'Tools->Tcl Scripts'.
  - Navigate to the:  
'soc\_system/synthesis/submodules/hps\_sdram\_p0\_pin\_assignments.tcl' script and execute it.
  - If the script runs successfully then the additional pins for SDA and SCLK can be added in via the pin planner.
- The last step in the process is to perform a full compilation and load the design via the Quartus programmer.
  - Next week we will go through the steps required to access the newly developed component via the HPS.

# Questions?

- Any questions?

# References

## Image References

- [1] - Altera Corporation, Cyclone V Device Overview, Figure 11, pp 31, 2016
- [2] - Altera Corporation, Cyclone V Hard Processor System Technical Reference Manual, Chapter 7, Figure 7.1, pp 2, 2016
- [3] - Altera Corporation, HPS SoC Boot Guide - Cyclone V SoC Development Kit, - Figure 1, pp 2, 2016
- [4] - Altera Corporation, Cyclone V Hard Processor System Technical Reference Manual, Chapter 8, Figure 8.1, pp 3, 2016
- [5] - NXP Semiconductors UM10204 I2C-bus specification and user manual, Figure 2, pp 7, 2014
- [6] - NXP Semiconductors, UM10204 I2C-bus specification and user manual, Figure 6, pp 10, 2014
- [7] - John Kneen, "John Kneen: Microcontrollers – I2C\_STM32F407, [https://sites.google.com/site/johnkneenmicrocontrollers/18b-i2c/i2c\\_stm32f407](https://sites.google.com/site/johnkneenmicrocontrollers/18b-i2c/i2c_stm32f407).