SISTEMI EMBEDDED
AA 2013/2014

Building a Nios II Computer
from scratch

Federico Baronti

Introduction

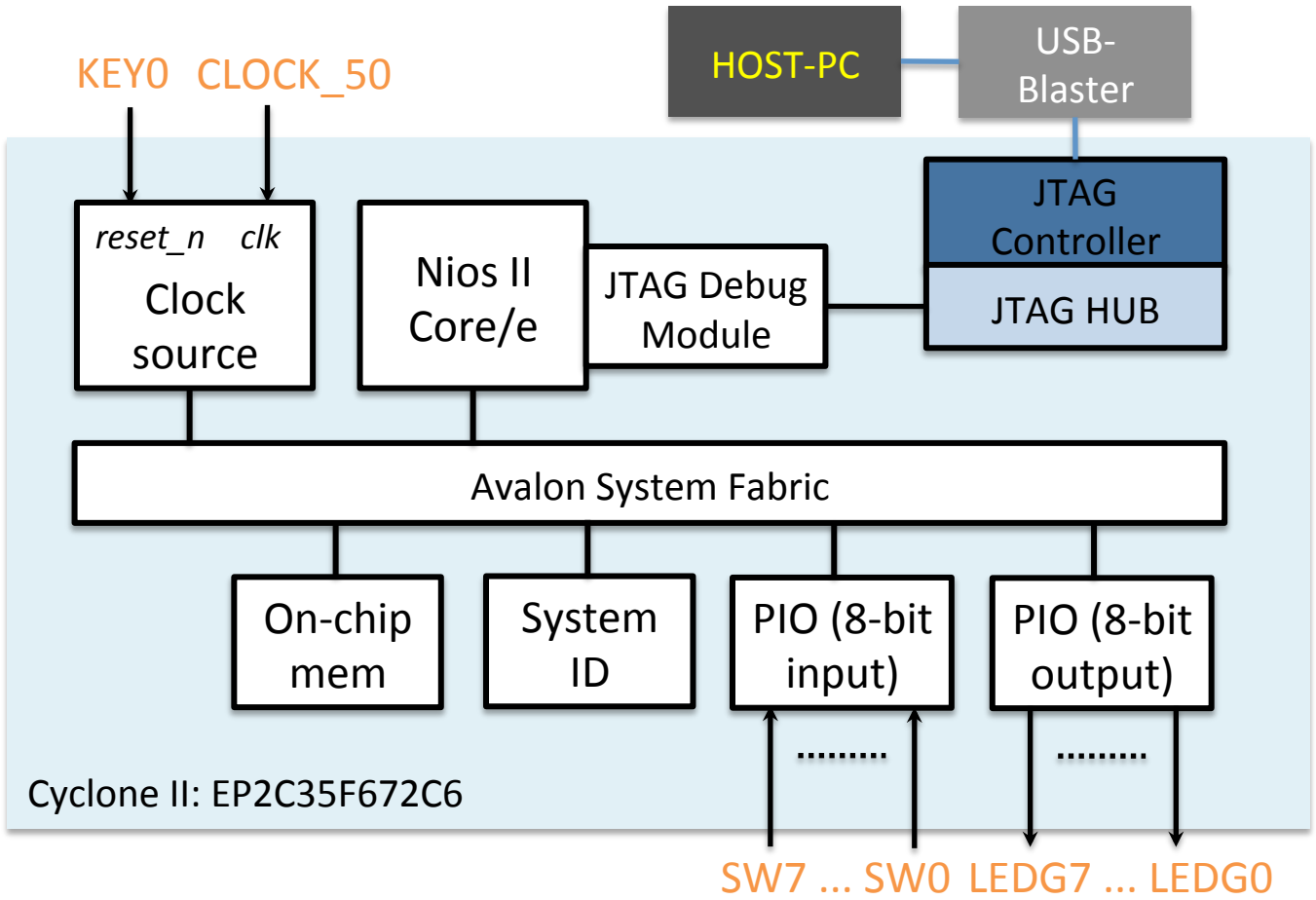
- Problem:
  - Build a (NIOS II) SoPC tailored to application needs
- Solutions:
  - Use library cores and custom HDL code
  - **Use specific design tools (SOPC Builder/Qsys) to help assemble the system**
    - Components (CPUs, memory (controllers), peripherals,…) selected from Altera, other vendors or custom libraries
    - Connections(Avalon System Fabric) are generated automatically by the tool
      - **Need for standard interfaces**
- DE2_basic_ and DE2_media_ computers are pre-built Nios II systems with different choices for the proc. economy and fast) and the peripherals

Avalon System Fabric

- Overview of Avalon standard interfaces:
  - Clock
  - Reset
  - Interrupt
  - Memory-Mapped (master and slave)
  - Streaming (source and sink)
  - Conduit

Example: First Nios System

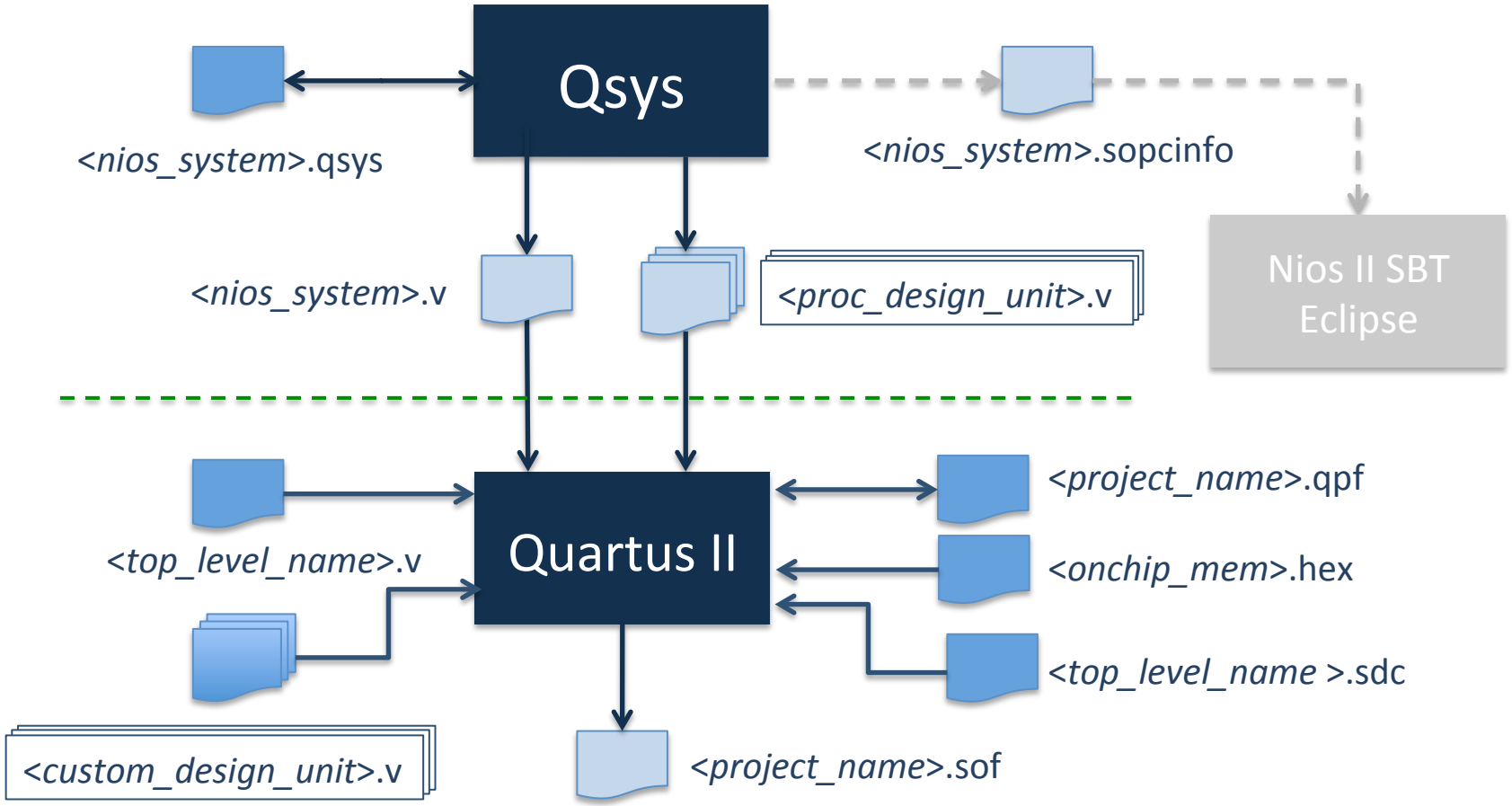
- Handles **switches** and **LEDs** through PIO peripherals



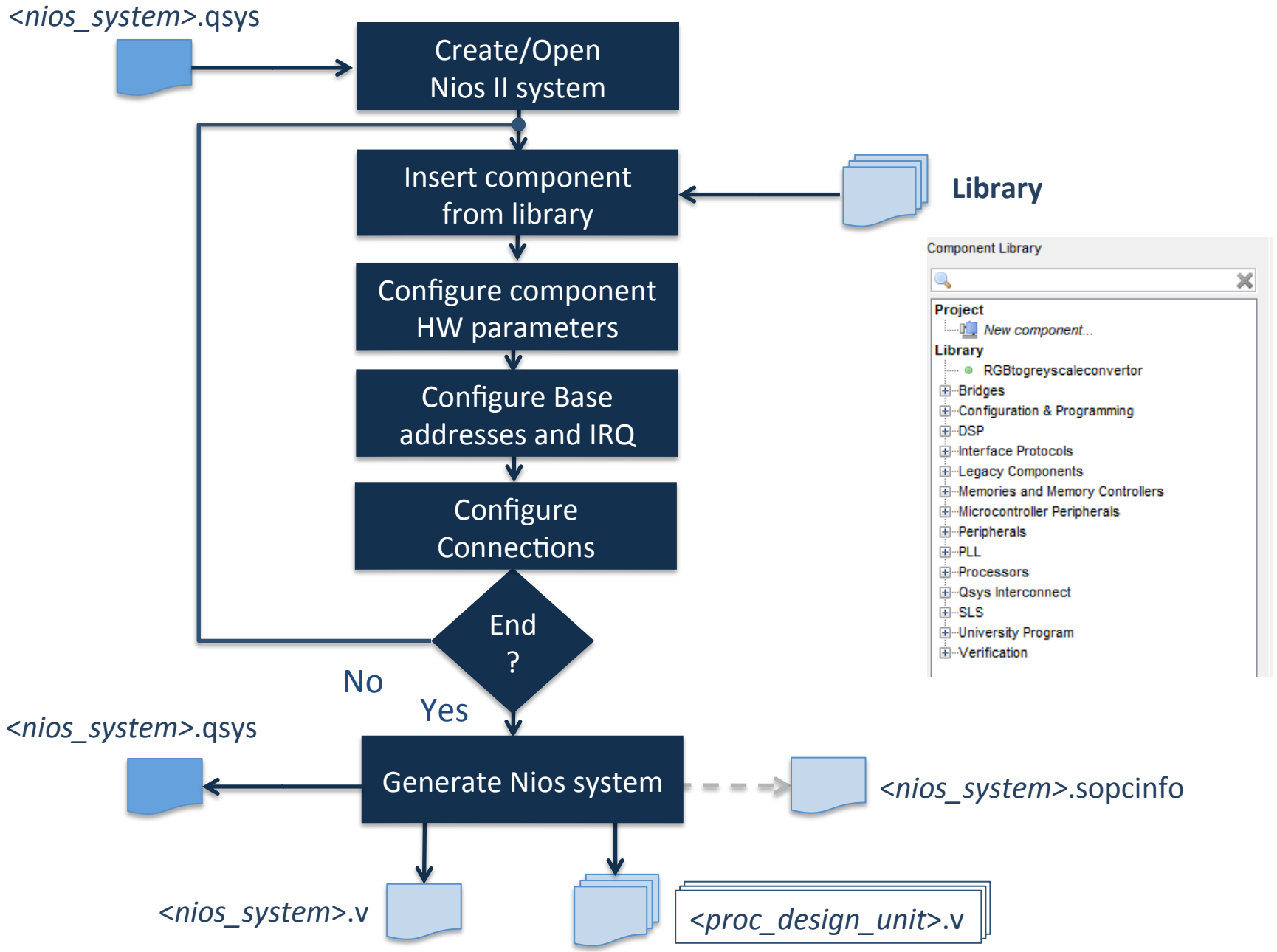
First Nios System units

- CPU (simplest, *i.e.*, economy variant) with JTAG Debug Module
- On-chip memory for program and data (8 KB)
- 2 PIOs
  - Input for reading slider switches (8 bit)
  - Output for driving green LEDs (8 bit)
- System ID Peripheral for computer identification

Nios II Hardware Flow



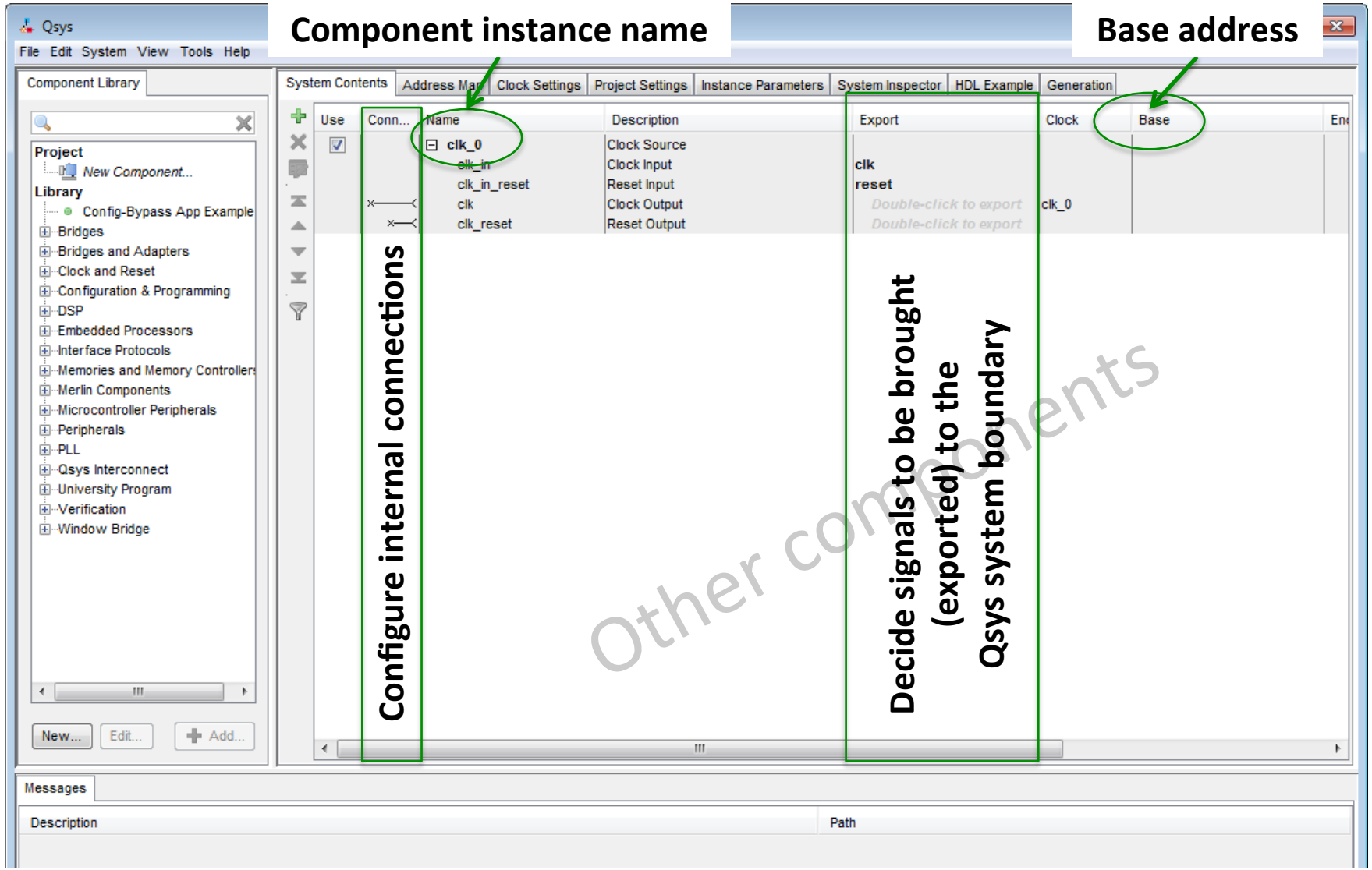
Qsys Flow


<nios_system>.qsys
Create/Open Nios II system
Insert component from library
Library
Component Library
Project
New component...
Library
RGBtogreyscaleconvertor
Bridges
Configuration & Programming
DSP
Interface Protocols
Legacy Components
Memories and Memory Controllers
Microcontroller Peripherals
Peripherals
PLL
Processors
Qsys Interconnect
SLS
University Program
Verification
Configure component HW parameters
Configure Base addresses and IRQ
Configure Connections
End ?
No
Yes
<nios_system>.qsys
Generate Nios system
<nios_system>.sopcinfo
<nios_system>.v
<proc_design_unit>.v

Guided example (1)

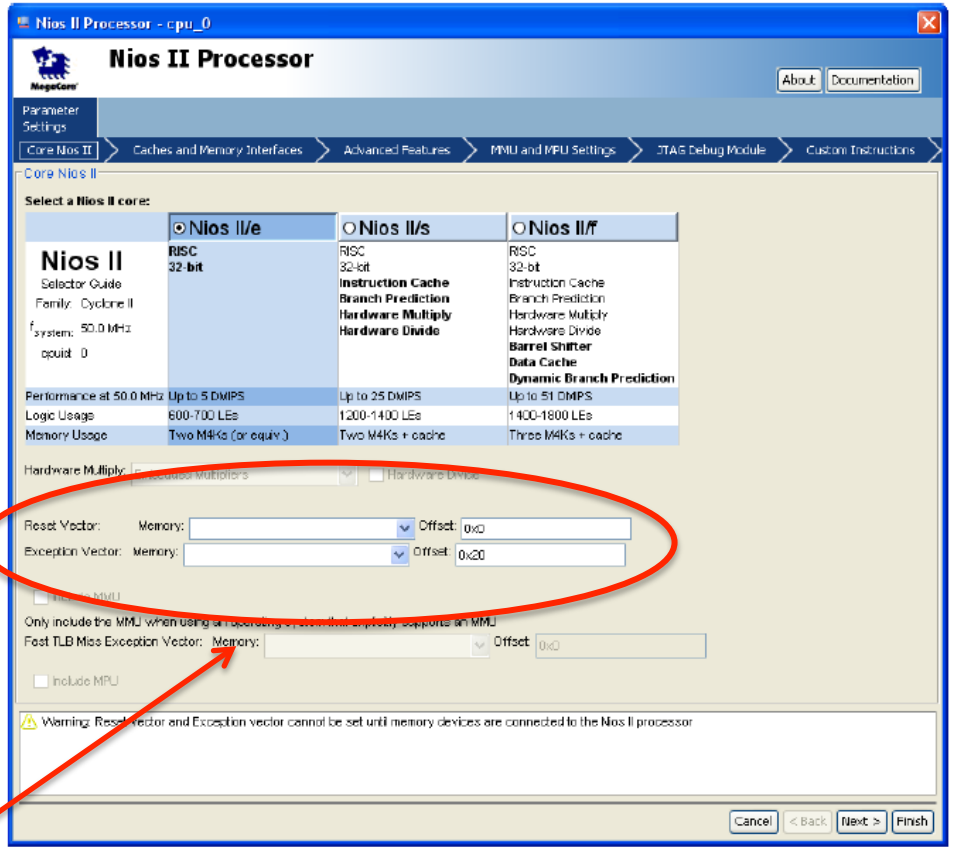
- Create a new project in Quartus II
- Launch Qsys tool
- Define the Nios_system components
  - Clock source: *clk* (it is added automatically)
  - Nios II Proc.: *nios2_proc*
    - Choose the economy version of the NiosII proc. (NiosII/e) and the Level 1 for the JTAG Debug Module
  - On-chip Memory: *onchip_memory*
  - PIO: *green_leds*
    - Output for driving LEDS
  - PIO: *sliders*
    - Input for reading slider switches status
  - System ID Peripheral: *sysid* (ID = 1!)

Qsys main window



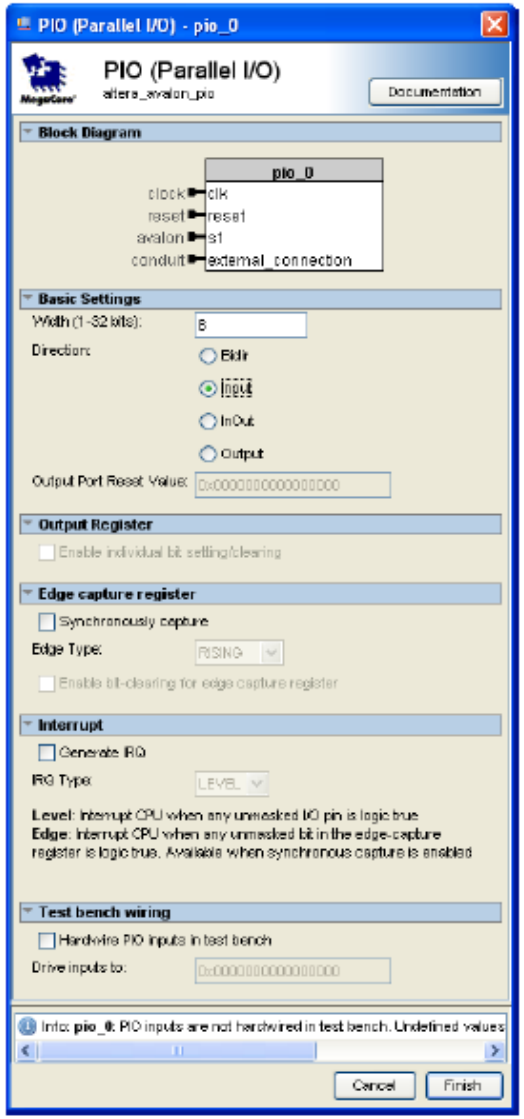
CPU choice

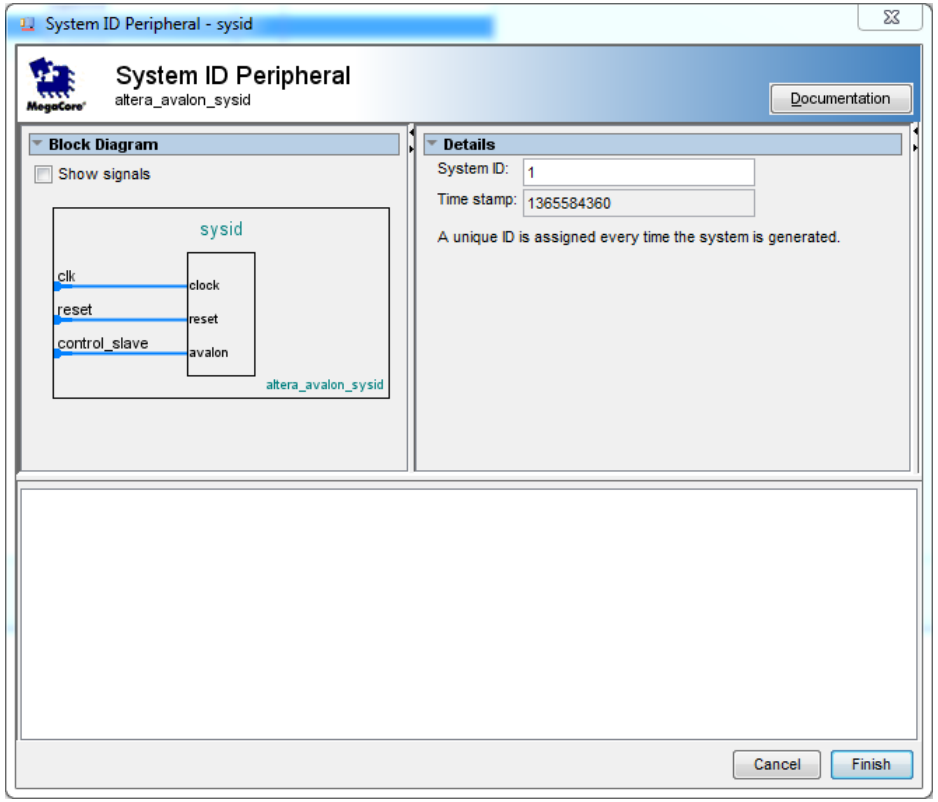
- Choose the most suited processor core
- 3 variants:
  - Economy
  - Standard
  - Fast
- Different features
  - Trade-off performance-cost



At least one memory must be present in the Qsys system in order to configure the **Reset** and **Exception** addresses

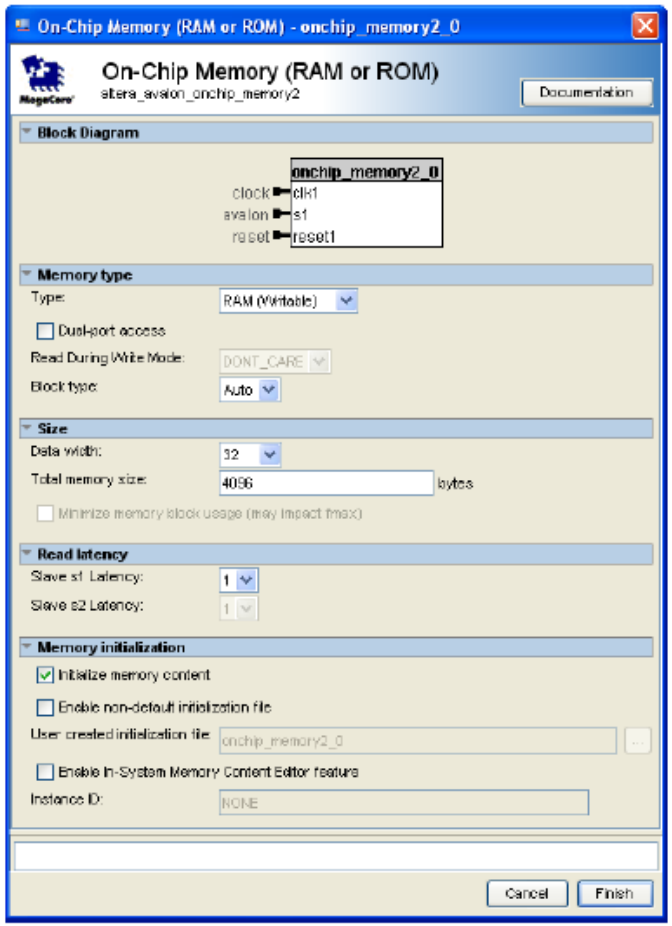
Additional peripherals





On-chip memory

- Define the organization of the on chip-memory
  - Type (ROM, RAM)
  - Size
  - Word length
- Initialization file: onchip_mem.hex



Guided example (2)

- **Configure internal connections**
  - Route *clk* from Clock Source component to the other components
  - Create *reset* network
    - "Route" reset signals from Clock Source and JTAG Debug Module (within the Nios II proc.) components to the other components
    - Can be done automatically using Create Global Reset Network command (System menu)
  - Link the Avalon Memory-Mapped Interfaces:
    - *data_master* (Nios II proc.), *jtag_debug_module* (Nios II proc.), *s1* (*onchip_memory*), *s1* (PIO: *sliders*, *green_leds*), control_slave (*sysid*)
    - *instruction_master* (Nios II proc.), *jtag_debug_module* (Nios II proc.), s1 (*onchip_memory*)

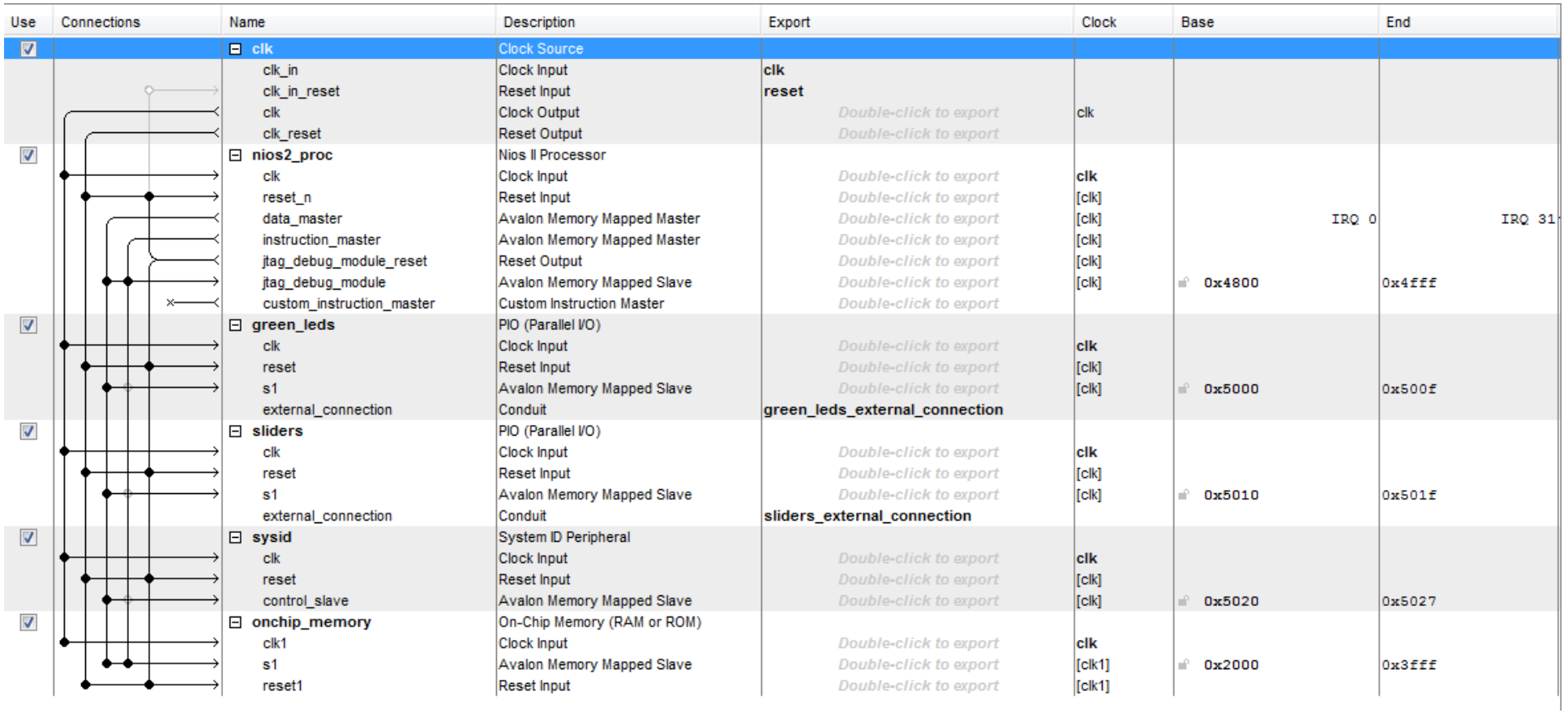
Guided example (3)

- **Export external connections**
  - *Sliders* and *green_leds* PIOs have conduit interfaces, the related signals (external_connection) must be routed to the Qsys system boundary

- **Assign base addresses**
  - Manually to each component with slave Memory-Mapped Interfaces (pay attention to avoid overlaps!)
  - Assign Base Addresses (from the System menu)

Guided example (4)

We are now ready to generate the Qsys system and go back to Quartus II



Use	Connections	Name	Description	Export	Clock	Base	End
☑		**clk**	Clock Source				
		clk_in	Clock Input	**clk**			
		clk_in_reset	Reset Input	**reset**			
		clk	Clock Output	*Double-click to export*	clk		
		clk_reset	Reset Output	*Double-click to export*			
☑		**nios2_proc**	Nios II Processor				
		clk	Clock Input	*Double-click to export*	**clk**		
		reset_n	Reset Input	*Double-click to export*	[clk]		
		data_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]	IRQ 0	IRQ 31
		instruction_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]		
		jtag_debug_module_reset	Reset Output	*Double-click to export*	[clk]		
		jtag_debug_module	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x4800	0x4fff
		custom_instruction_master	Custom Instruction Master	*Double-click to export*			
☑		**green_leds**	PIO (Parallel I/O)				
		clk	Clock Input	*Double-click to export*	**clk**		
		reset	Reset Input	*Double-click to export*	[clk]		
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x5000	0x500f
		external_connection	Conduit	**green_leds_external_connection**			
☑		**sliders**	PIO (Parallel I/O)				
		clk	Clock Input	*Double-click to export*	**clk**		
		reset	Reset Input	*Double-click to export*	[clk]		
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x5010	0x501f
		external_connection	Conduit	**sliders_external_connection**			
☑		**sysid**	System ID Peripheral				
		clk	Clock Input	*Double-click to export*	**clk**		
		reset	Reset Input	*Double-click to export*	[clk]		
		control_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x5020	0x5027
☑		**onchip_memory**	On-Chip Memory (RAM or ROM)				
		clk1	Clock Input	*Double-click to export*	**clk**		
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]	0x2000	0x3fff
		reset1	Reset Input	*Double-click to export*	[clk1]		

Guided example (6)

- Back to Quartus II
  - Import Qsys system into Quartus project. Do **one** of the followings:
    - Method I: Add the .qip file stored in *nios_system*>/synthesis to the project
    - Method II: Add the .qsys file to the project
  - Create the root module of the project
  - Include the Nios_system module as hierarchical block (Verilog)
  - Import pin assignment from de2.qsf
  - Compile the project to make the hardware ready

Guided example (7)

- **Integrating Qsys system into Quartus II project**
  - Method I: Add the (Quartus II file) .qip file stored in <*nios_system*>/synthesis to the project
    - .qip file is created when generating the Qsys system together with the .sopcinfo and the HDL files
    - It lists all the files necessary for compilation in Quartus II, including the references to the HDL files generated by Qsys

Guided example (8)

- **Integrating Qsys system into Quartus II project**
  - Method II: Add the .qsys file to project
    - The Qsys system is **now** (re)generated by Quartus II at each compilation
    - The generated HDL files are stored at a different path than those generated directly by Qsys
      - db/ip/<*nios_system*>
    - Note that the *sysid* timestamp changes at each compilation in Quartus II
    - The BSP must be regenerated using the new sopcinfo file after each compilation, even if we have not made any change to the Qsys system!

Guided example (7)

- Project root module

```
// my_DE2_first_computer.v

module my_DE2_first_computer(
        //input
        CLOCK_50,
        KEY,
        SW,
        //output
        LEDG
);
        input CLOCK_50;
        input [0:0] KEY;
        input [7:0] SW;

        output [7:0] LEDG;

        // Add the nios_system instance
        //  The instance template can be copied from Qsys HDL example tab

endmodule
```

Testing First Nios System (1)

- Write a program that makes the GREEN LEDS to be controlled by the SLIDERS SWITCHES
- If successful, generate the hex file to initialize the on-chip memory. Recompile the project and reprogram the FPGA. Your program should run!
- To generate the hex file from elf. Open the Nios 2 Command Shell and navigate to the Eclipse project folder. Customize the following command:

elf2hex --record=4 --width=32 --base=<*onchip_memory* ***base*** *address*>
--end=<*onchip_memory* ***end*** *address*> --input=<*eclipse_project_name*.elf>
--output=../../Hardware/onchip_mem.hex

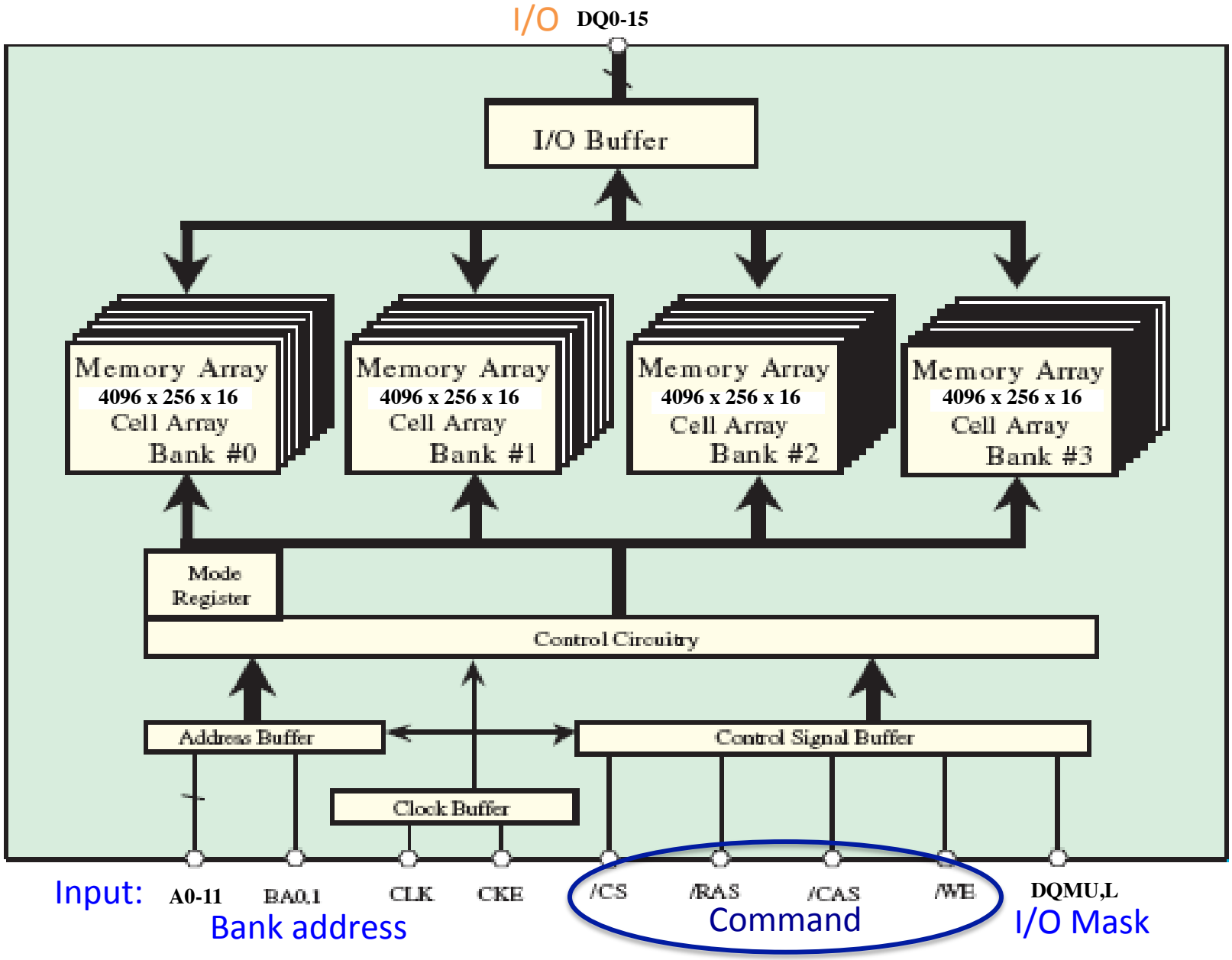
Testing First Nios System (2)

- Go back to Qsys, add the JTAG-UART peripheral, regenerate the Nios system and compile the design again (top level entry does not need to be changed)
- Write a program that say Hello to the host telling him the sistem ID and timestamp
  - The new file .sopcinfo needs to be used as the Nios system has been regenerated

Testing First Nios System (3)

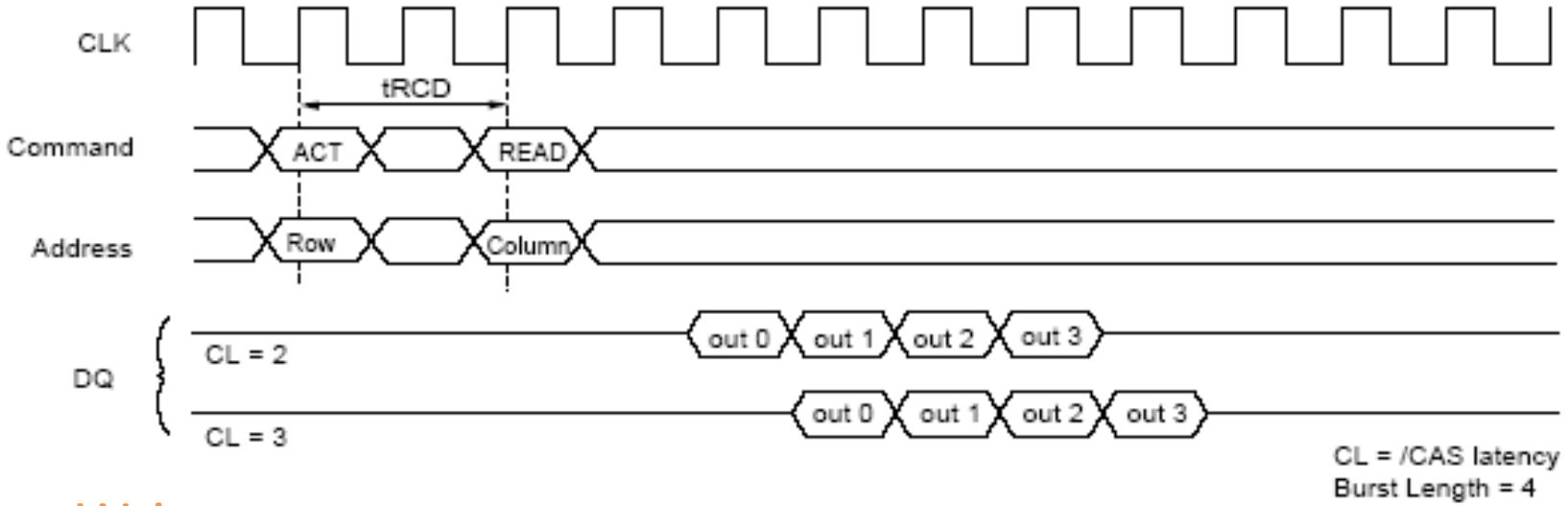
- Allocated on chip memory is not enough!
  - JTAG-UART device driver requires more memory than the one available
  - In a future lesson, we will learn some techniques to reduce the memory footprint of our software
  - Now, we can:
    - enlarge the on-chip memory. Note that EP2C35 FPGA has 105 x M4Kb=52.5 KB; some M4K blocks are used to implement the proc. and the JTAG Debug Module
    - add the SDRAM Controller to our Qsys system to use the 8 MB SDRAM memory (Zentel A3V64S40ETP -G6) present on the DE2 board

SDRAM memory (1)

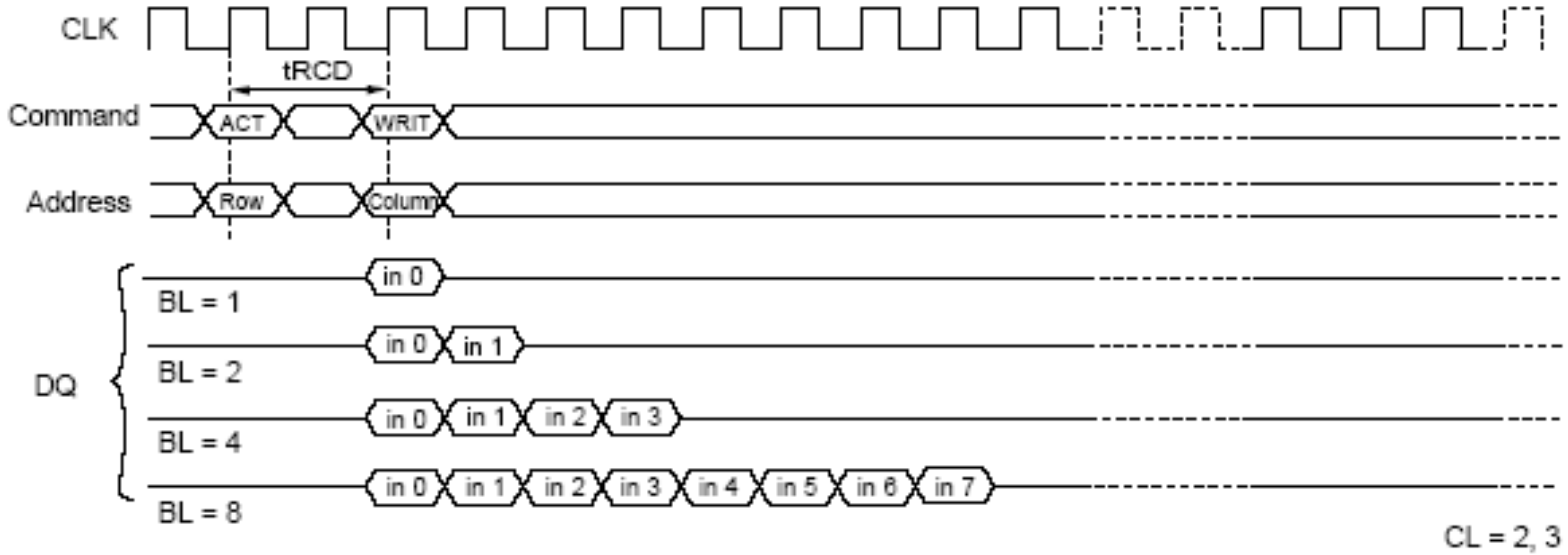


SDRAM memory (2)

Read



Write



SDRAM memory (2)

Parameter	Symbol	Version		Unit	Note
		-6	-7		
Row active to row active delay	t_{RRD}(min)	12	14	ns	1
RAS to CAS delay	t_{RCD}(min)	18	21	ns	1
Row precharge time	t_{RP}(min)	18	21	ns	1
Row active time	t_{RAS}(min)	40	42	ns	1
	t_{RAS}(max)	100	100	us	
Row cycle time	t_{RC}(min)	58	63	ns	1
Last data in to row precharge	t_{RDL}(min)	2	2	CLK	2
Col. address to col. address delay	t_{CCD}(min)	1	1	CLK-	
Last data in to new col. address delay	t_{CDL}(min)	1	1	CLK	2
Last data in to burst stop	t_{BDL}(min)	1	1	CLK	2
Mode register set cycle time	t_{MRD}(min)	2	2	CLK	
Refresh interval time	t_{REF}(max)	64	64	ms	
Auto refresh cycle time	t_{ARFC}(min)	60	70	ns	

SDRAM memory (3)

Parameter		Symbol	-6		-7		Unit	Note
			Min	Max	Min	Max		
CLK cycle time	CAS latency=3	t_{CC} (3)	6	1000	7	1000	ns	1
	CAS latency=2	t_{CC} (2)	10		10			
CLK to valid output delay	CAS latency=3	t_{SAC} (3)		5.5		6	ns	1,2
	CAS latency=2	t_{SAC} (2)		6		6		
Output data hold time	CAS latency=3	t_{OH} (3)	2.5		2.5		ns	2
	CAS latency=2	t_{OH} (2)	2.5		2.5			
CLK high pulse width		t_{CH}	2.5		2.5		ns	3
CLK low pulse width		t_{CL}	2.5		2.5		ns	3
Input setup time		t_{SI}	1.5		1.5		ns	3
Input hold time		t_{HI}	1		1		ns	3
Transition time of CLK		t_{SLZ}	0		0		ns	2
CLK to output in Hi-Z	CAS latency=3	t_{SHZ}		5.5		6	ns	
	CAS latency=2			6		6		

SDRAM memory (4)

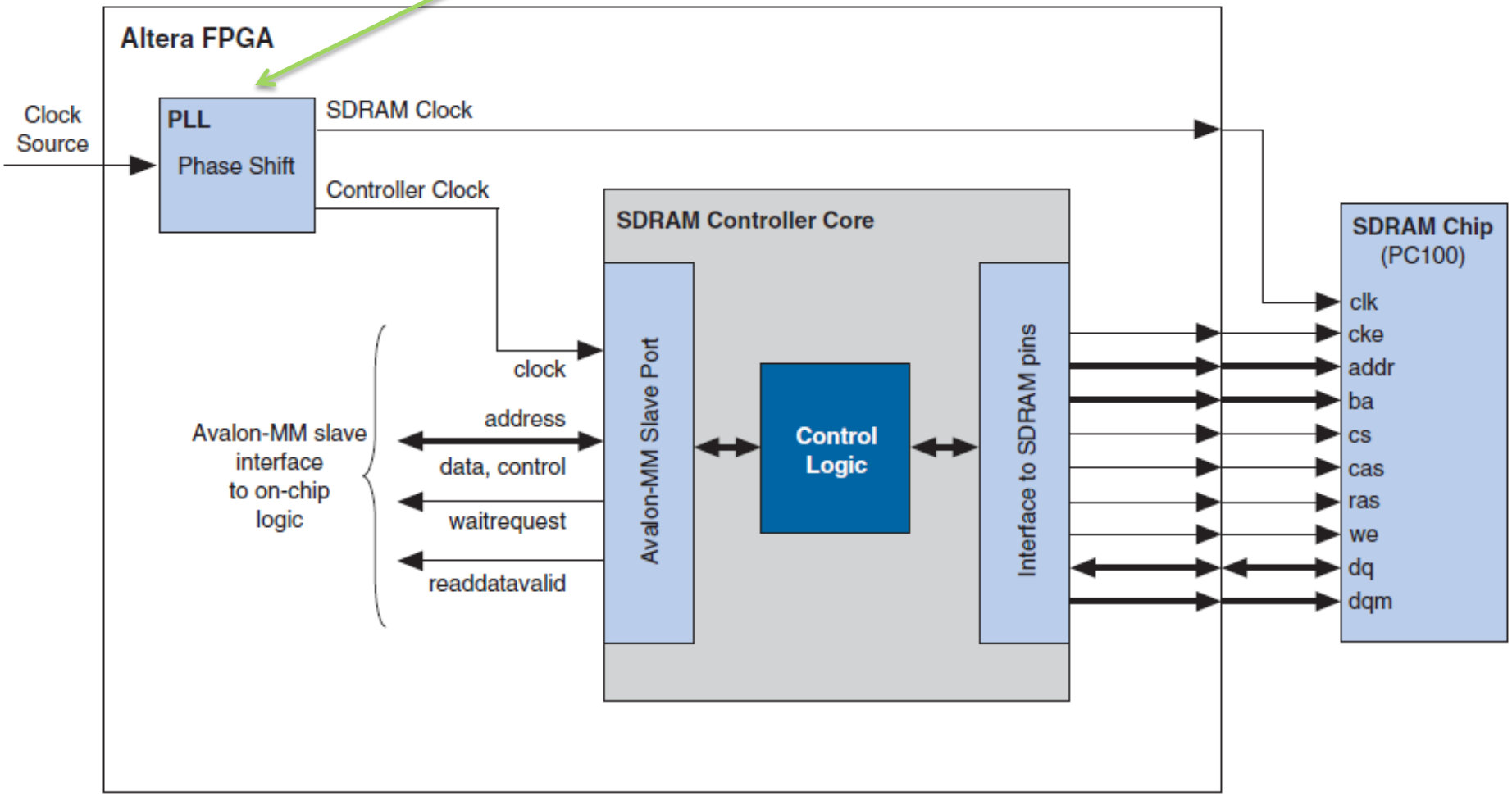
Initialization sequence

4. After stable power and stable clock, wait 200us.
5. Issue precharge all command (PALL).
6. After tRP delay, set 2 or more auto refresh commands (REF).
7. Set the mode register set command (MRS) to initialize the mode register.

SDRAM controller (1)

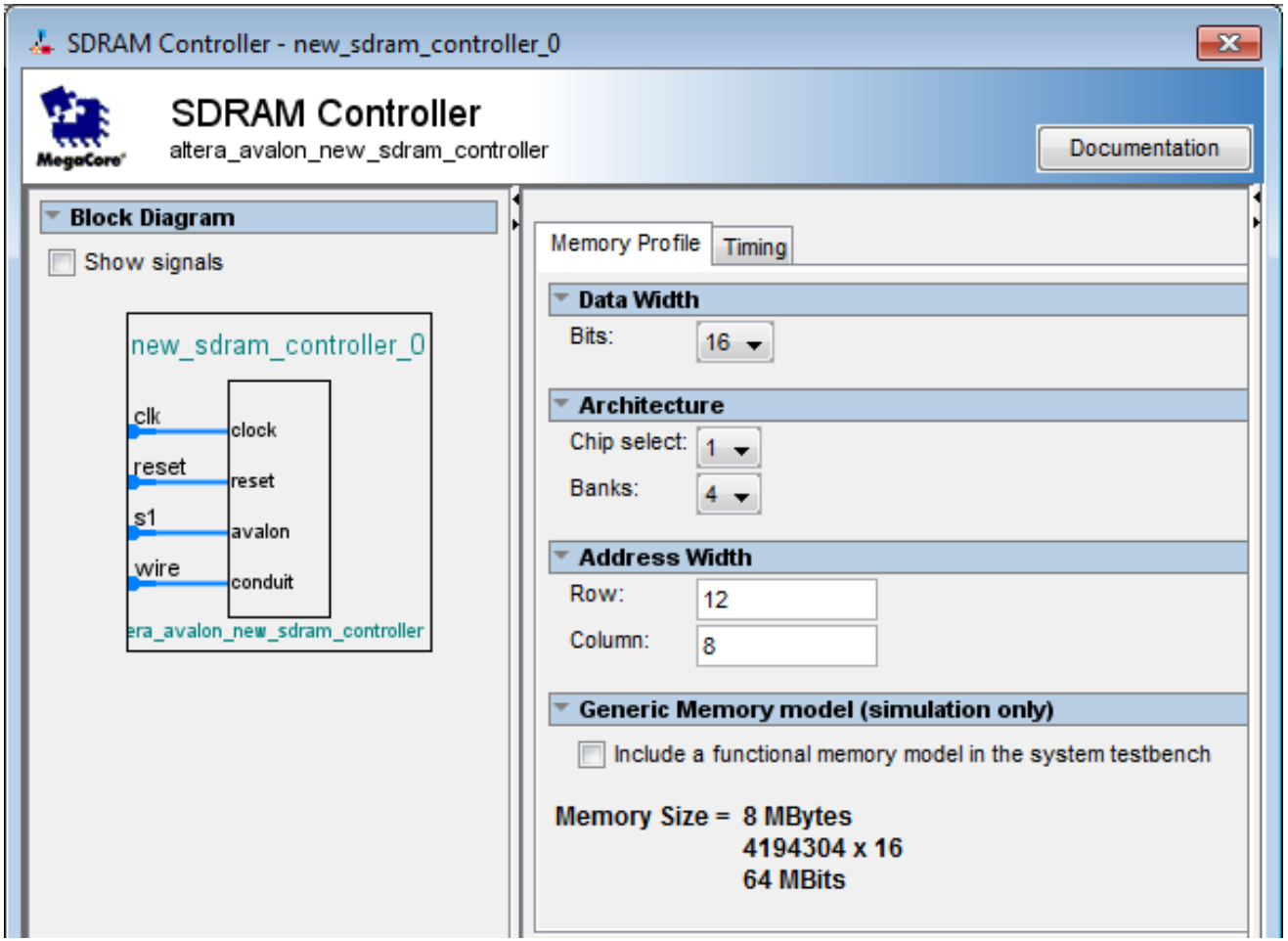
A PLL can be used to compensate for the clock skew

DE2 board *SDRAM Clock* must lead the *Controller Clock* by 3 ns (Phase shift)



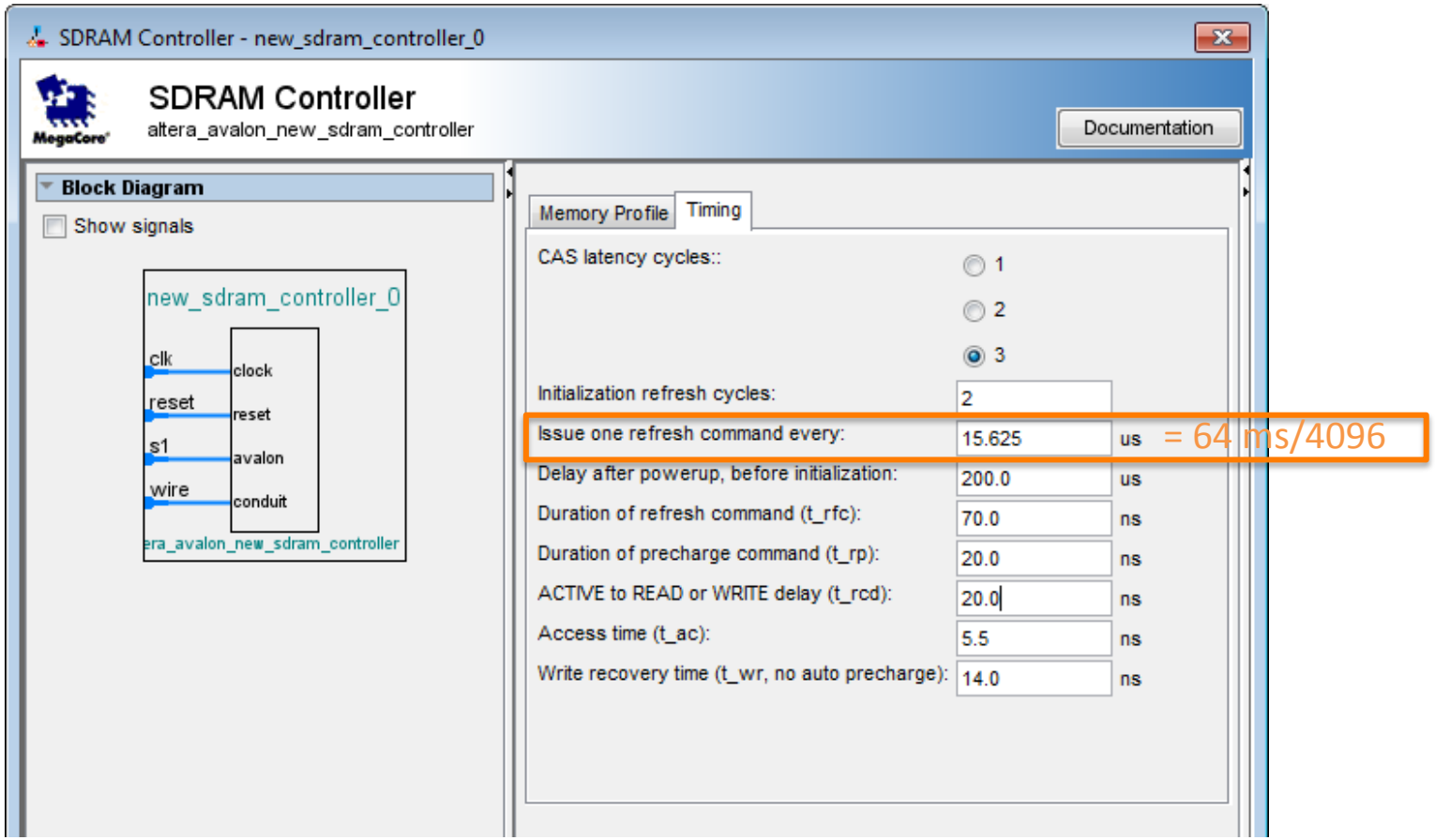
SDRAM controller (2)

- *Library/Memory and Memory Controllers/SDRAM Interfaces*



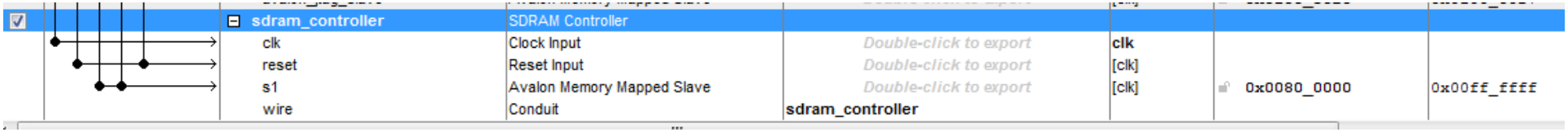
SDRAM controller (3)

- *Library/Memory and Memory Controllers/SDRAM Interfaces*



SDRAM controller (4)

- Instantiate and configure the component for SDRAM memory
- Set Qsys internal connection: clock, reset and Avalon MM slave
- Export signals towards the memory chip (Conduit interface)
- Assign Base Address



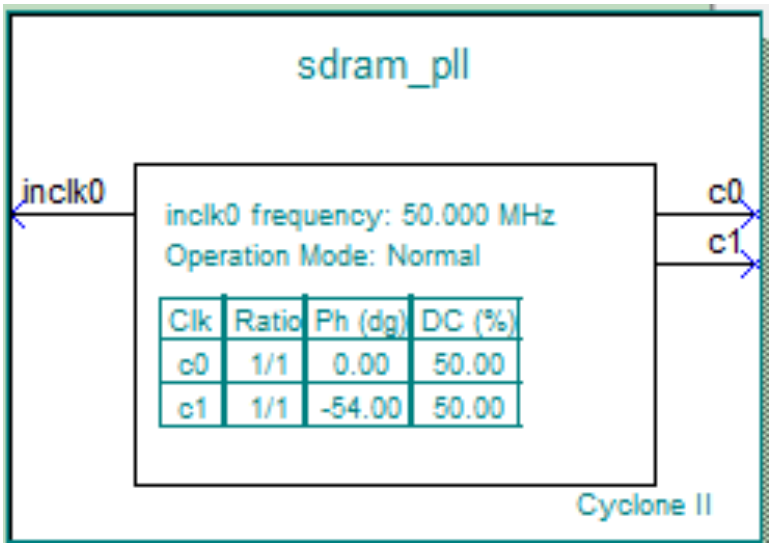
- Move Reset and Exception addresses to freshly created SDRAM controller

SDRAM controller (5)

- Generate the Qsys system (mandatory if using the .qip file) and go back to Quartus II
- Update the Qsys system instance (you can use the template in the HDL Example tab of Qsys)
- Update the module interface to include the external SDRAM controller signals
  - Connect them to new Qsys system instance
  - Create the PLL to generate the SDRAM clock

SDRAM Clock

- Require instantiating and configuring a PLL
  - Can be done using the MegaWizard Plug-in Manager [I/O Library]
  - c0 and c1 have the same frequency as inclok0, i.e., 50 MHz but are shifted eachother by 3 ns
- Integrate the PLL into the top module
- Compile the design

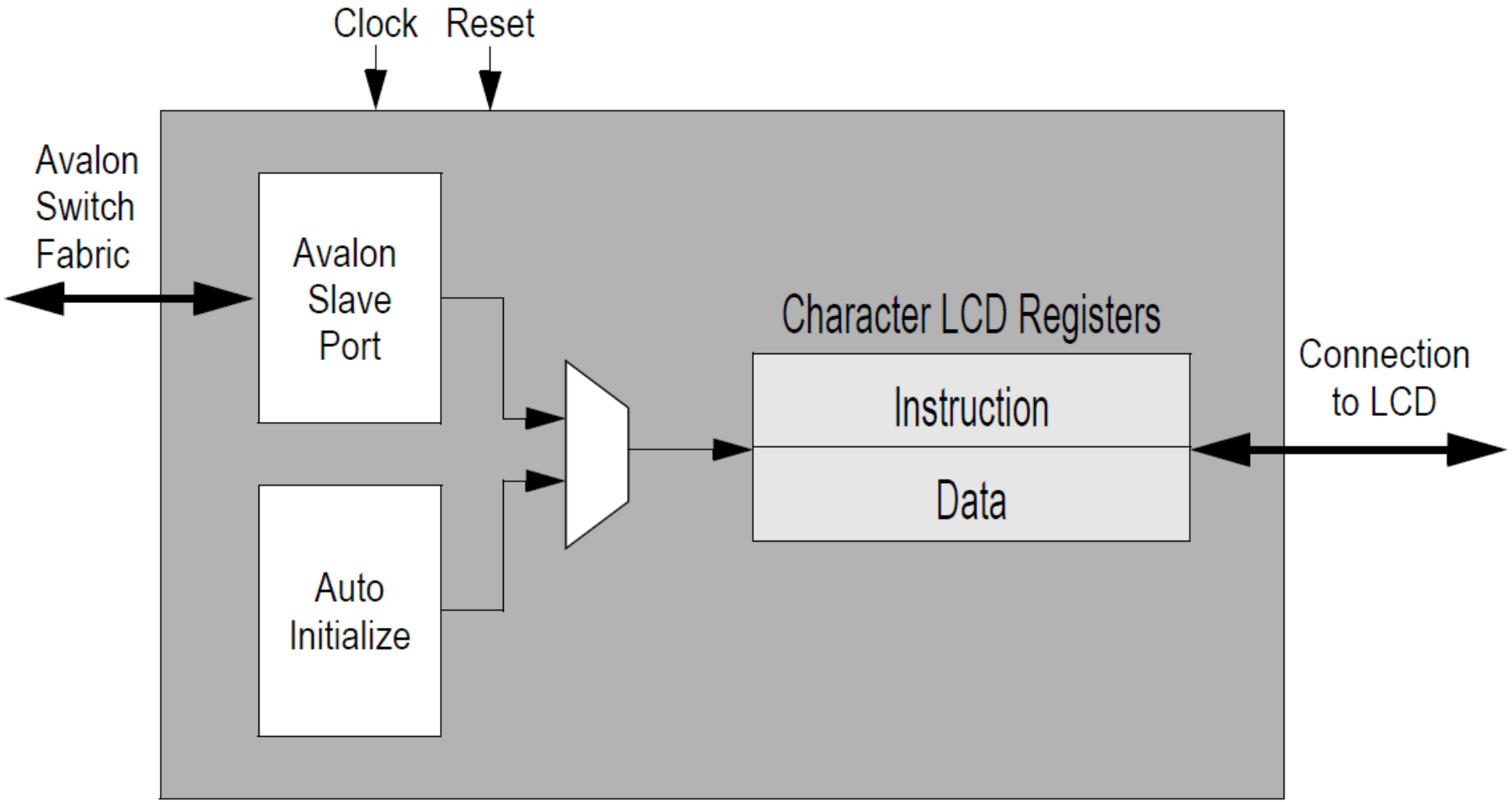


Putting into practice

- Create a new project in Eclipse and see if the new computer works with the SDRAM memory
- If ok, re-enable the stdio functions and write a simple program that use them
  - Work on the Blocking/Non blocking I/O operations
- When done, go ahead to integrate the LCD into your computer

16x2 Character Display (1)

Architecture of the 16x2 character display peripheral



16x2 Character Display (2)

External signals of the FPGA connected to the 16x2 character display

Signal Name	FPGA Pin No.	Description
LCD_DATA[0]	PIN_J1	LCD Data[0]
LCD_DATA[1]	PIN_J2	LCD Data[1]
LCD_DATA[2]	PIN_H1	LCD Data[2]
LCD_DATA[3]	PIN_H2	LCD Data[3]
LCD_DATA[4]	PIN_J4	LCD Data[4]
LCD_DATA[5]	PIN_J3	LCD Data[5]
LCD_DATA[6]	PIN_H4	LCD Data[6]
LCD_DATA[7]	PIN_H3	LCD Data[7]
LCD_RW	PIN_K4	LCD Read/Write Select, 0 = Write, 1 = Read
LCD_EN	PIN_K3	LCD Enable
LCD_RS	PIN_K1	LCD Command/Data Select, 0 = Command, 1 = Data
LCD_ON	PIN_L4	LCD Power ON/OFF
LCD_BLON	PIN_K2	LCD Back Light ON/OFF

Port declaration of the 16x2 character LCD module

```
module character_lcd_0 (
    // Inputs
    clk,
    reset,
    address,
    chipselect,
    read,
    write,
    writedata,

    // Bidirectionals
    LCD_DATA,

    // Outputs
    LCD_ON,
    LCD_BLON,
    LCD_EN,
    LCD_RS,
    LCD_RW,
    readdata,
    waitrequest
);
```

Character LCD API

- Header file: altera_up_character_lcd.h
- Device type: alt_up_character_lcd_dev
- Function prototypes:
  - alt_up_character_lcd_dev* alt_up_character_lcd_open_dev(const char* name);
  - void alt_up_character_lcd_init(alt_up_character_lcd_dev *lcd);
  - int alt_up_character_lcd_set_cursor_pos (alt_up_character_lcd_dev *lcd, unsigned x_pos, unsigned y_pos);
  - void alt_up_character_lcd_string(alt_up_character_lcd_dev *lcd, const char *ptr);
  - ...

Test the new Nios II system

- Write a simple program that wtites a string on the 16x2 character display

References

- Altera “Embedded Peripherals User Guide,” ***ug_embedded_ip.pdf***
  - Section I - Chapter 2. SDRAM controller
- Zentel, “A3V64S40FTP datasheet”
- Altera, “Using the SDRAM Memory on Altera’s DE2 Board,” *tut_DE2_sdram_verilog.pdf* with Verilog Design
- Altera, “16x2 Character Display for Altera DE2-Series Boards,” *Character_LCD.pdf*