# SISTEMI EMBEDDED

## Building a Nios II Computer from scratch

Federico Baronti

Last version: 20170327

## Introduction

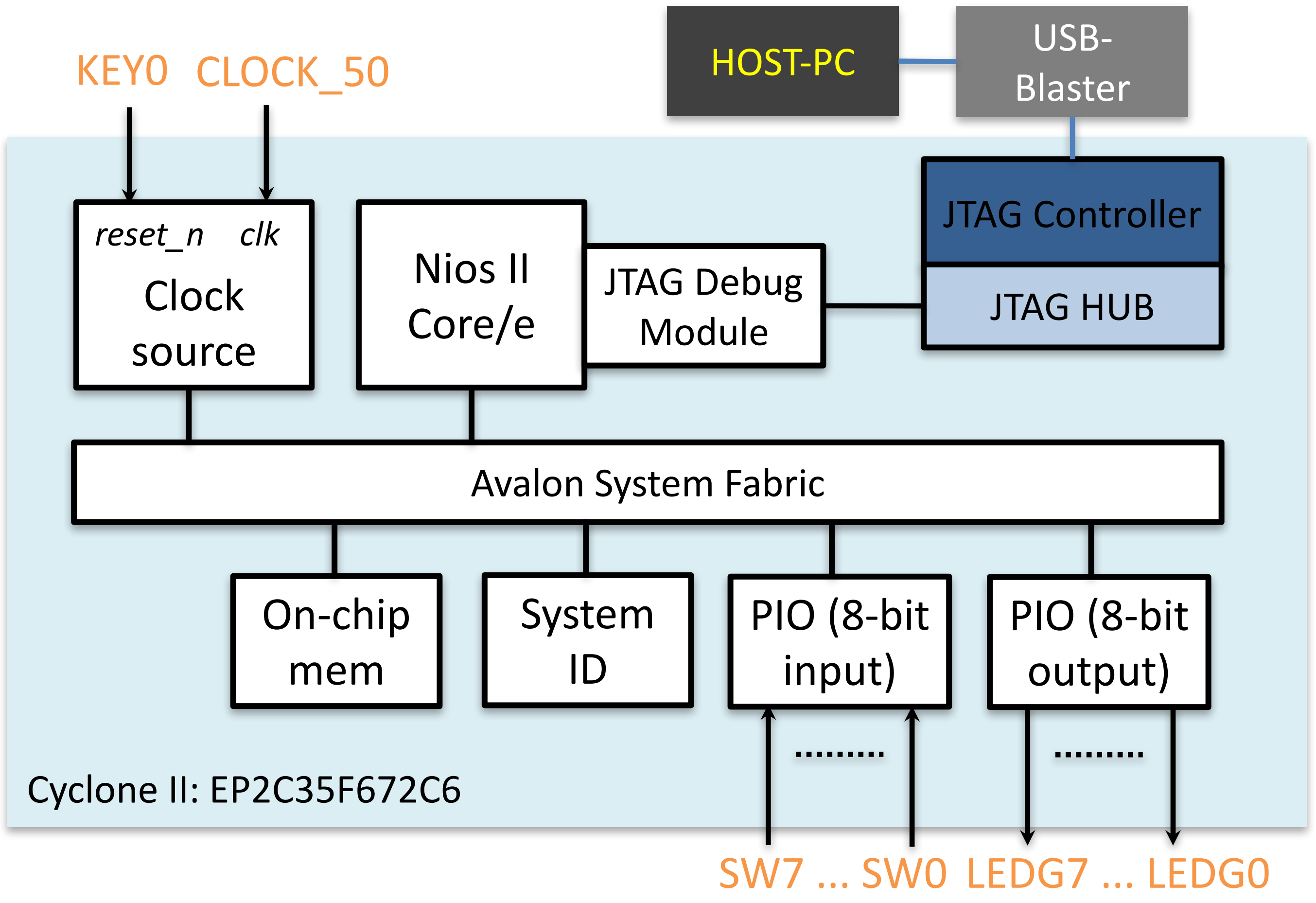
- Problem:
  - Build a (NIOS II) Computer tailored to application needs
- Solutions:
  - Use library cores and custom HDL code
  - **Use specific design tools (Qsys) to help assemble the system**
    - Components (CPUs, memory (controllers), peripherals,…) selected from Altera, other vendors or custom libraries
    - Connections (Avalon System Interconnect 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 available in the University Program package

# Avalon System Interconect Fabric

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

# Example: First Nios System

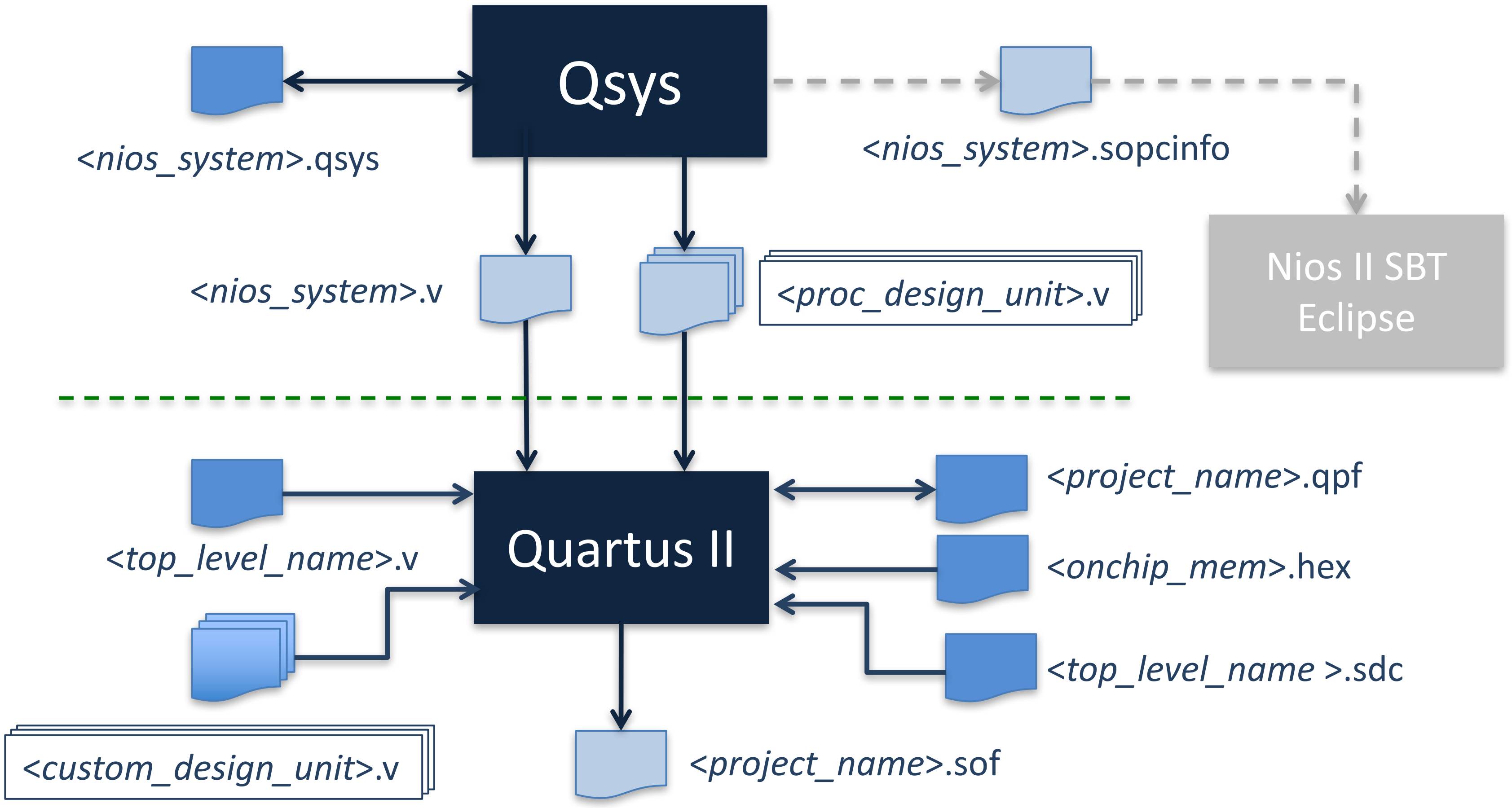
- Handles **slider switches** and **LEDs** through PIO peripherals



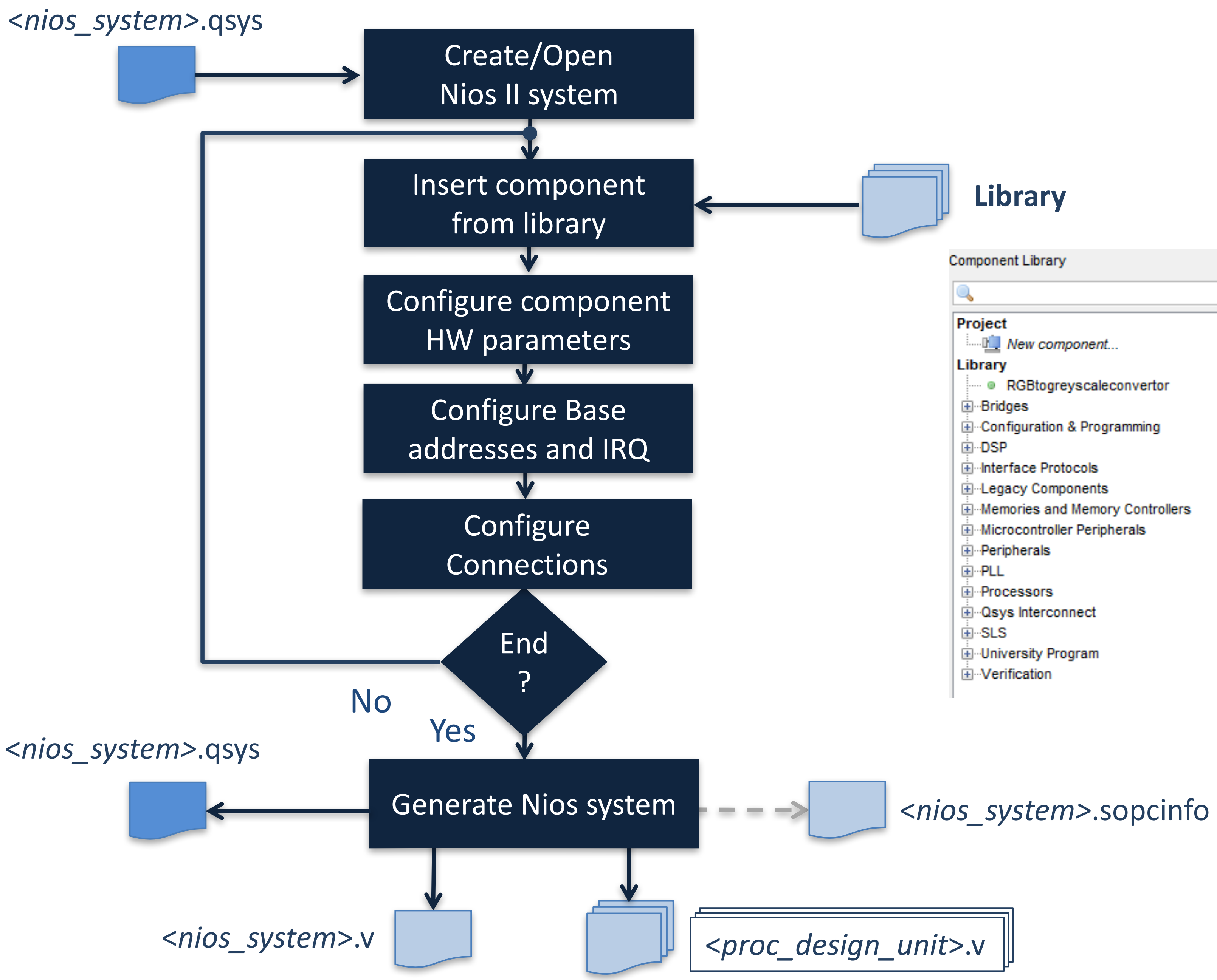
# First Nios Computer components

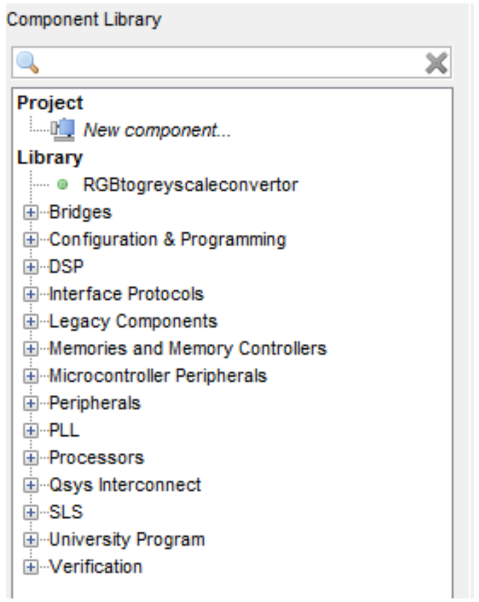
- CPU (simplest, *i.e.*, *economy* version) with JTAG Debug Module
- On-chip memory for program and data (8 KB)
- 2 PIOs
  - Input for reading slider switches (8 bit)
  - Output for driving green LEDs (8 bit)
- System ID Peripheral for computer identification

# Nios II Hardware Flow



# Qsys Flow




Component Library

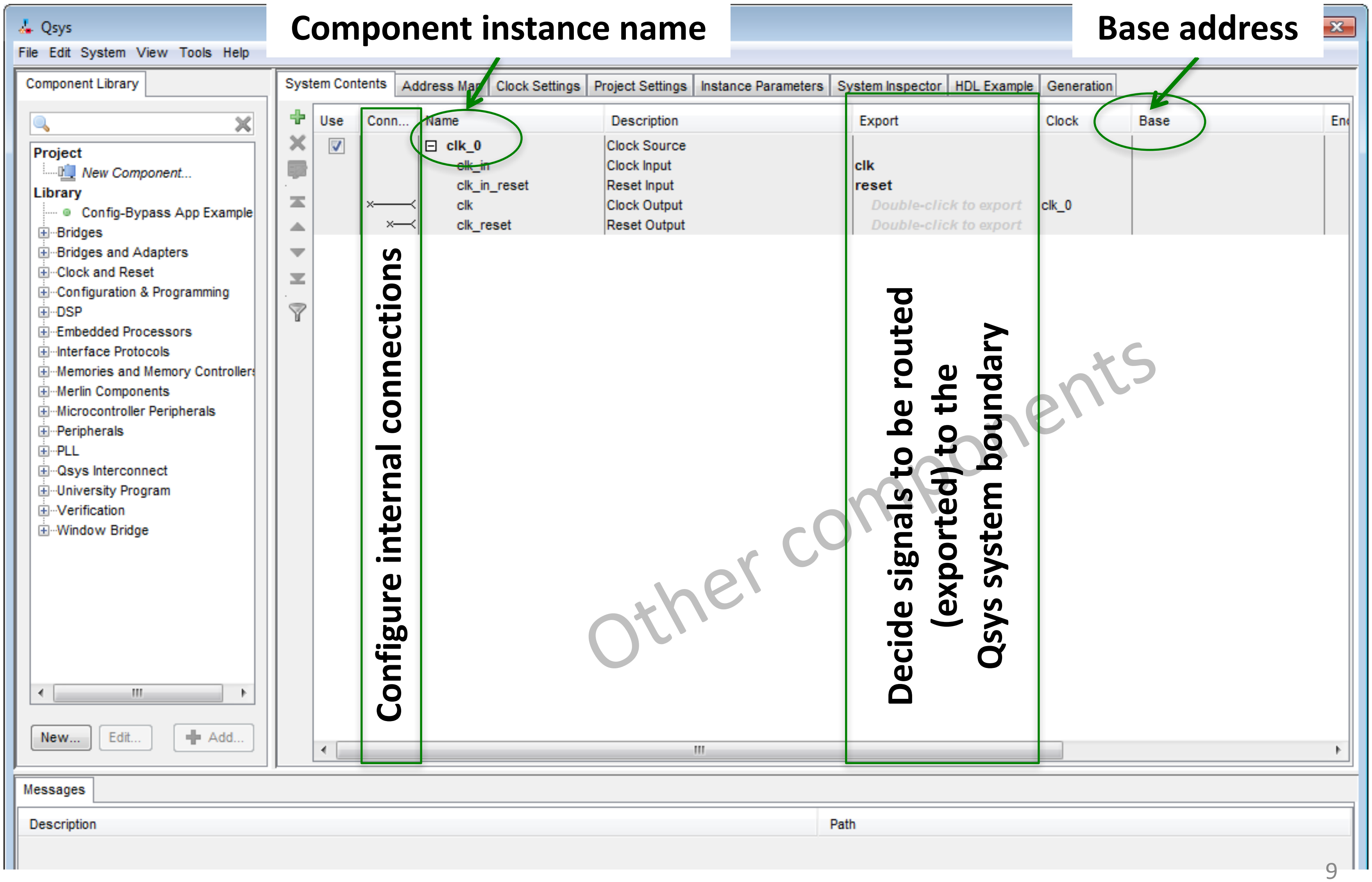
Project
- New component...

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

# Guided example (1)

- Create a new project in Quartus II
  - Select FPGA: Cyclone II EP2C35F672C6N
- 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!)

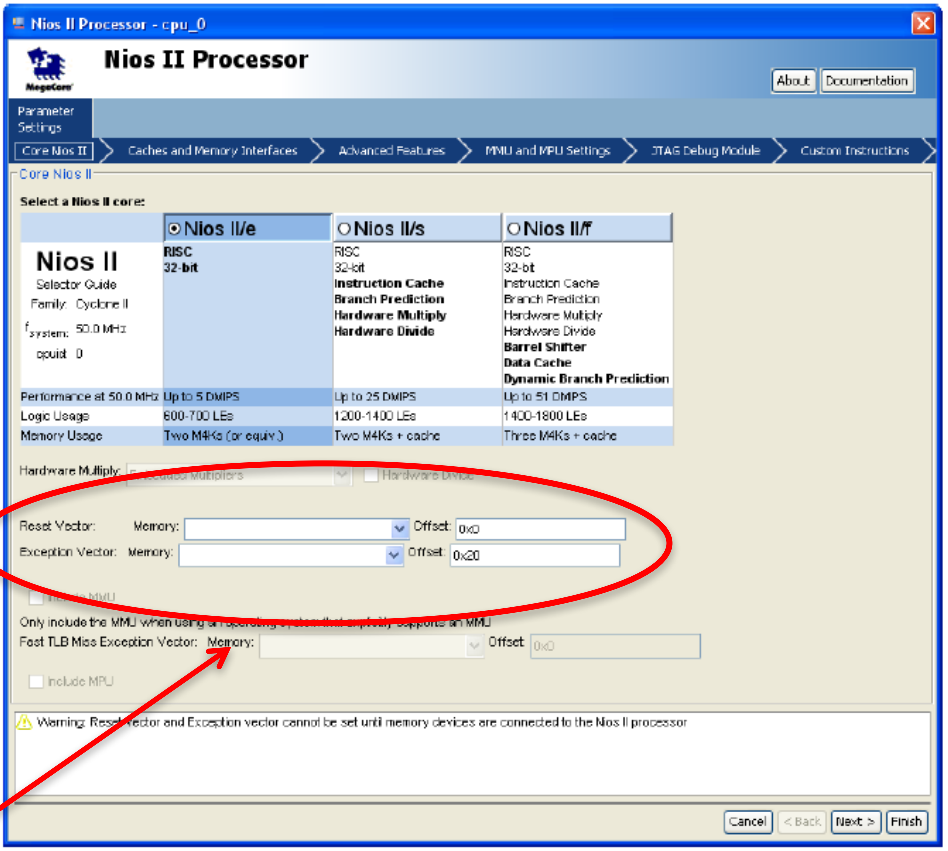
# Qsys main window



| Use | Conn... | Name | Description | Export | Clock | Base |
|---|---|---|---|---|---|---|
| ✓ | | clk_0 | Clock Source | | | |
| | | clk_in | Clock Input | clk | | |
| | | clk_in_reset | Reset Input | reset | | |
| | | clk | Clock Output | Double-click to export | clk_0 | |
| | | clk_reset | Reset Output | Double-click to export | | |

# CPU choice

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



Select a Nios II core:

| | [✓ Nios II/e] | Nios II/s | Nios II/f |
|---|---|---|---|
| Nios II Selector Guide<br>Family: Cyclone II<br>f system: 50.0 MHz<br>cpuid: 0 | **RISC**<br>**32-bit** | RISC<br>32-bit<br>**Instruction Cache**<br>**Branch Prediction**<br>**Hardware Multiply**<br>**Hardware Divide** | RISC<br>32-bit<br>Instruction Cache<br>Branch Prediction<br>Hardware Multiply<br>Hardware Divide<br>**Barrel Shifter**<br>**Data Cache**<br>**Dynamic Branch Prediction** |
| Performance at 50.0 MHz | Up to 5 DMIPS | Up to 25 DMIPS | Up to 51 DMIPS |
| Logic Usage | 600-700 LEs | 1200-1400 LEs | 1400-1800 LEs |
| Memory Usage | Two M4Ks (or equiv.) | Two M4Ks + cache | Three M4Ks + cache |

Hardware Multiply: Embedded Multipliers    Hardware Divide

Reset Vector: Memory: Offset: 0x0
Exception Vector: Memory: Offset: 0x20

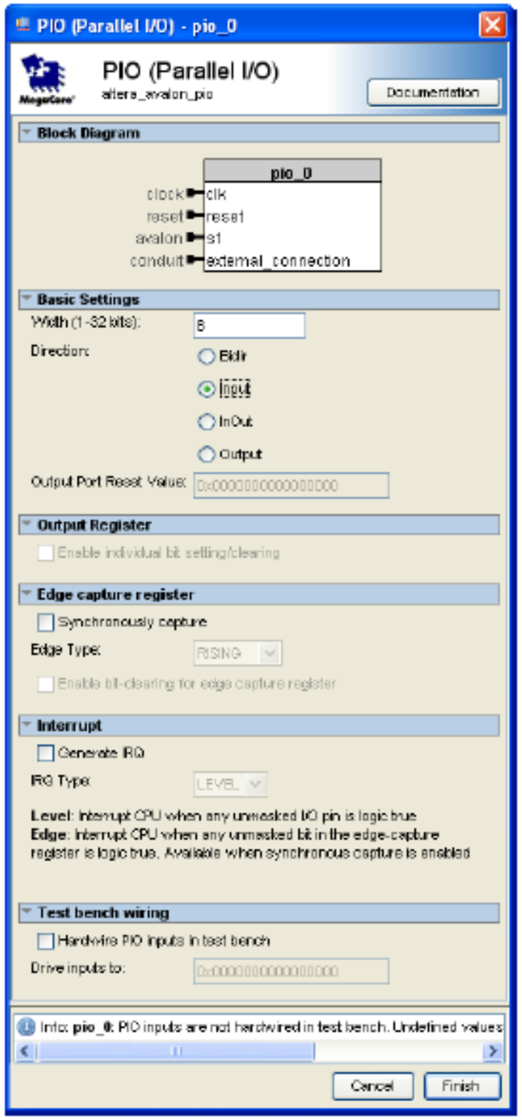
Include MMU
Only include the MMU when using an operating system that explicitly supports an MMU
Fast TLB Miss Exception Vector: Memory: Offset: 0x0

Include MPU

Warning: Reset Vector and Exception vector cannot be set until memory devices are connected to the Nios II processor

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

# Additional peripherals


PIO (Parallel I/O) - pio_0

PIO (Parallel I/O)
altera_avalon_pio

**Block Diagram**

**Basic Settings**

Width (1-32 bits): 8

Direction: ○ Bidir [✓ ◉ Input] ○ InOut ○ Output

Output Port Reset Value: 0x0000000000000000

**Output Register**

☐ Enable individual bit setting/clearing

**Edge capture register**

☐ Synchronously capture

Edge Type: RISING

☐ Enable bit-clearing for edge capture register

**Interrupt**

☐ Generate IRQ

IRQ Type: LEVEL

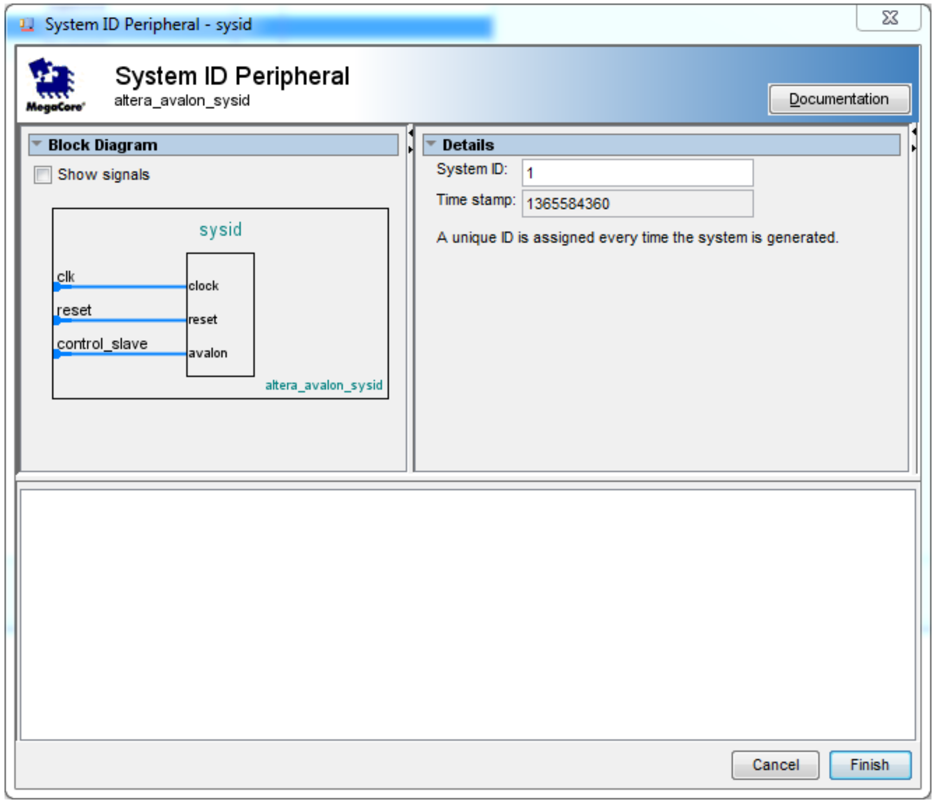
**Level**: Interrupt CPU when any unmasked I/O pin is logic true
**Edge**: Interrupt CPU when any unmasked bit in the edge-capture register is logic true. Available when synchronous capture is enabled

**Test bench wiring**

☐ Hardwire PIO inputs in test bench

Drive inputs to: 0x0000000000000000

Info: **pio_0**: PIO inputs are not hardwired in test bench. Undefined values


System ID Peripheral - sysid

System ID Peripheral
altera_avalon_sysid

**Block Diagram**

☐ Show signals

**Details**

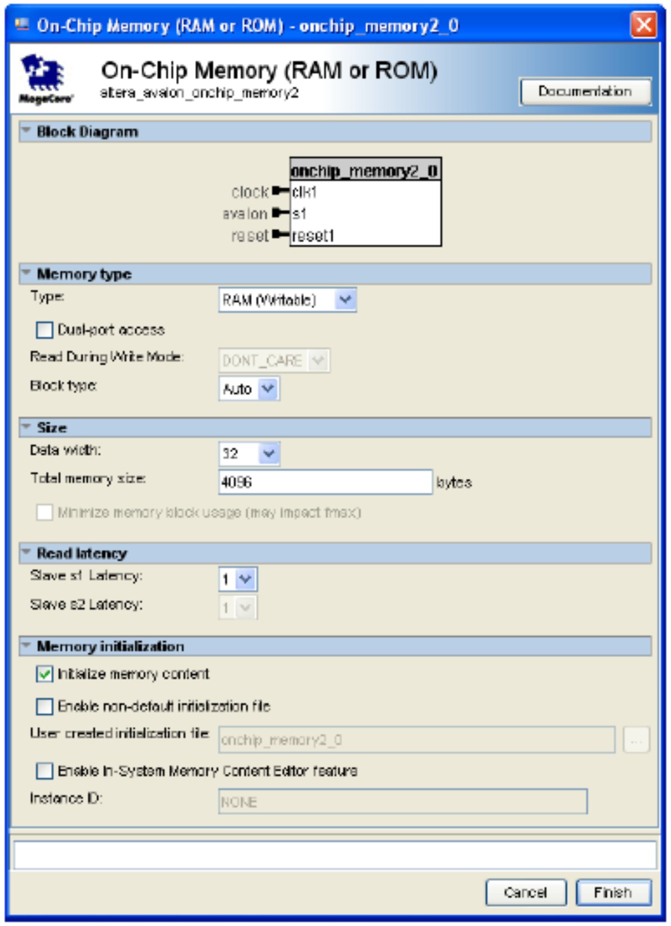
System ID: 1

Time stamp: 1365584360

A unique ID is assigned every time the system is generated.

# On-chip memory

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



On-Chip Memory (RAM or ROM) - onchip_memory2_0

On-Chip Memory (RAM or ROM)
altera_avalon_onchip_memory2

**Block Diagram**

**Memory type**

Type: RAM (Writable)

☐ Dual-port access

Read During Write Mode: DONT_CARE

Block type: Auto

**Size**

Data width: 32

Total memory size: 4096 bytes

☐ Minimize memory block usage (may impact fmax)

**Read latency**

Slave s1 Latency: 1

Slave s2 Latency: 1

**Memory initialization**

☑ Initialize memory content

☐ Enable non-default initialization file

User created initialization file: onchip_memory2_0

☐ Enable In-System Memory Content Editor feature

Instance ID: NONE

# 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 <u>Create Global Reset Network</u> 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*)

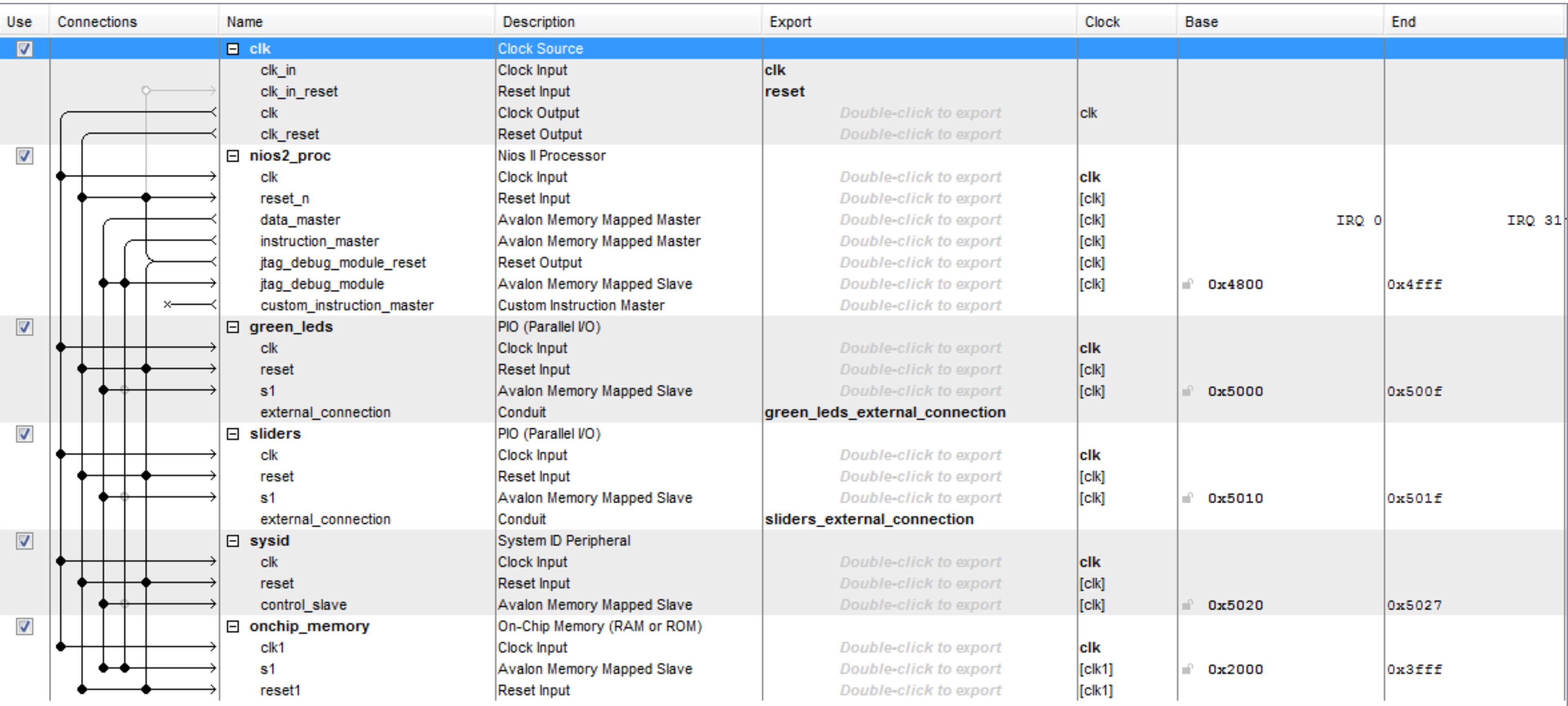
## Guided example (3)

- **Export external connections**
  - *Sliders* and *green_leds* PIOs have <u>conduit</u> 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:
    - **<u>Method I</u>**: Add the .qip file stored in *nios_system*>/synthesis to the project
    - <u>Method II</u>: 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)

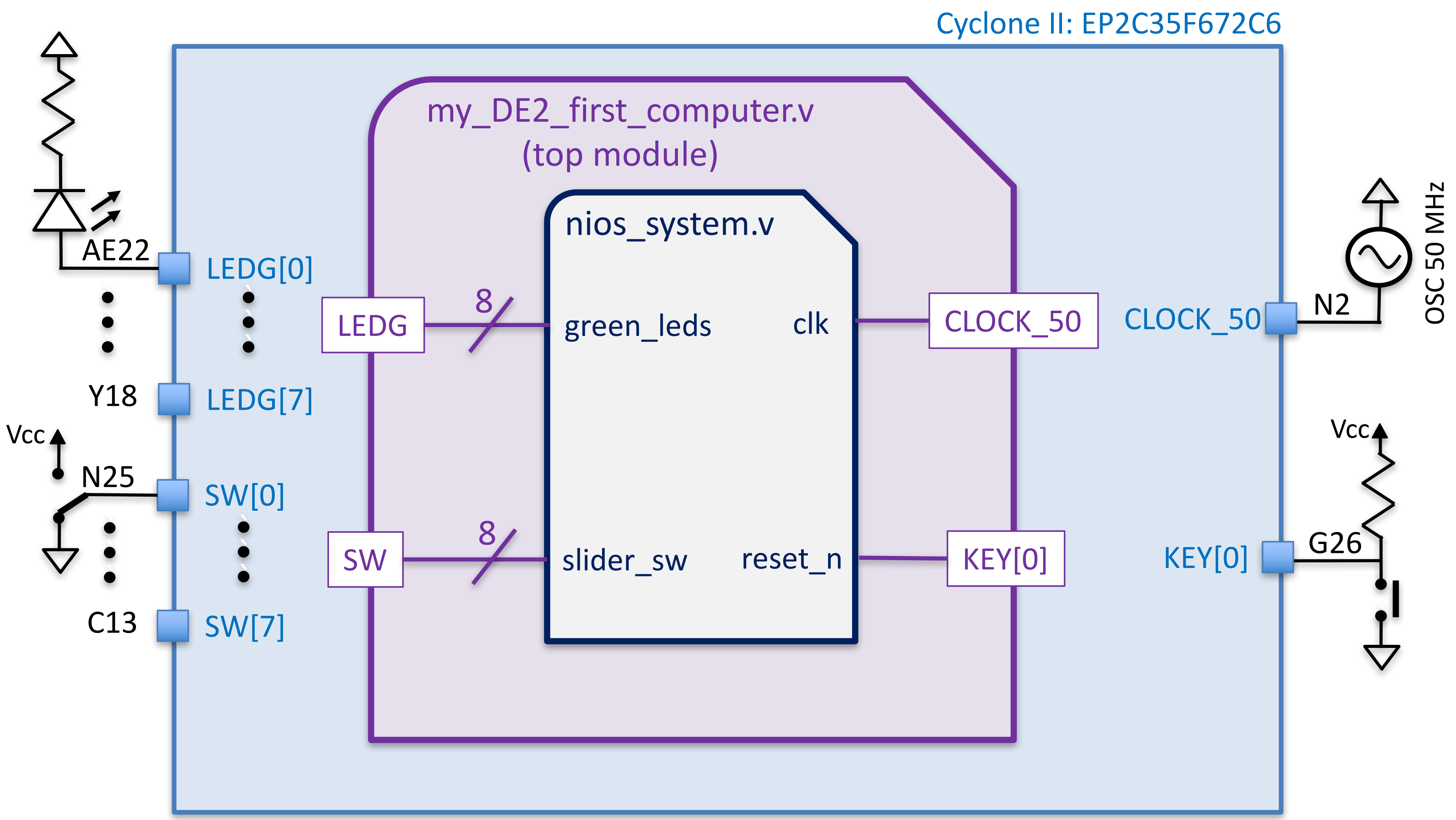
- **Integrating Qsys system into Quartus II project**
  - Method I: Add the (Quartus II file) .qip file stored in <*nios_system*>/synthesis to the project
    - .qip file is created when generating the Qsys system together with the .sopcinfo and the HDL files
    - It lists all the files necessary for 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 (7a)

- Project root module



# Guided example (7b)

- 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
```

## Guided example (7c)

- Project root module
  (using Verilog-2001 C-style port declaration)

```
// my_DE2_first_computer.v

module my_DE2_first_computer(
        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 Quartus project and reprogram the FPGA. Your program should run automatically!
- 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 (1a)

- Enrich the *First Nios System* w/ 2 additional PIOs properly configured to control the **push buttons** (w/ edge capture capability) and the **HEX3-HEX0 7-seg displays** available on the DE2 board.
  – Make the ID of this new computer equal to 2
  – **Test the computer running** the **LED rotation**, the **Fast Click** and the **Week day** programs
  – Recall that the push button signal is low when the switch is pressed and that a led of the 7-seg display is ON when driven low
    - Try to guess what's inside the parallel port peripheral connected to the HEX 7-seg displays used in the DE2 Basic Computer

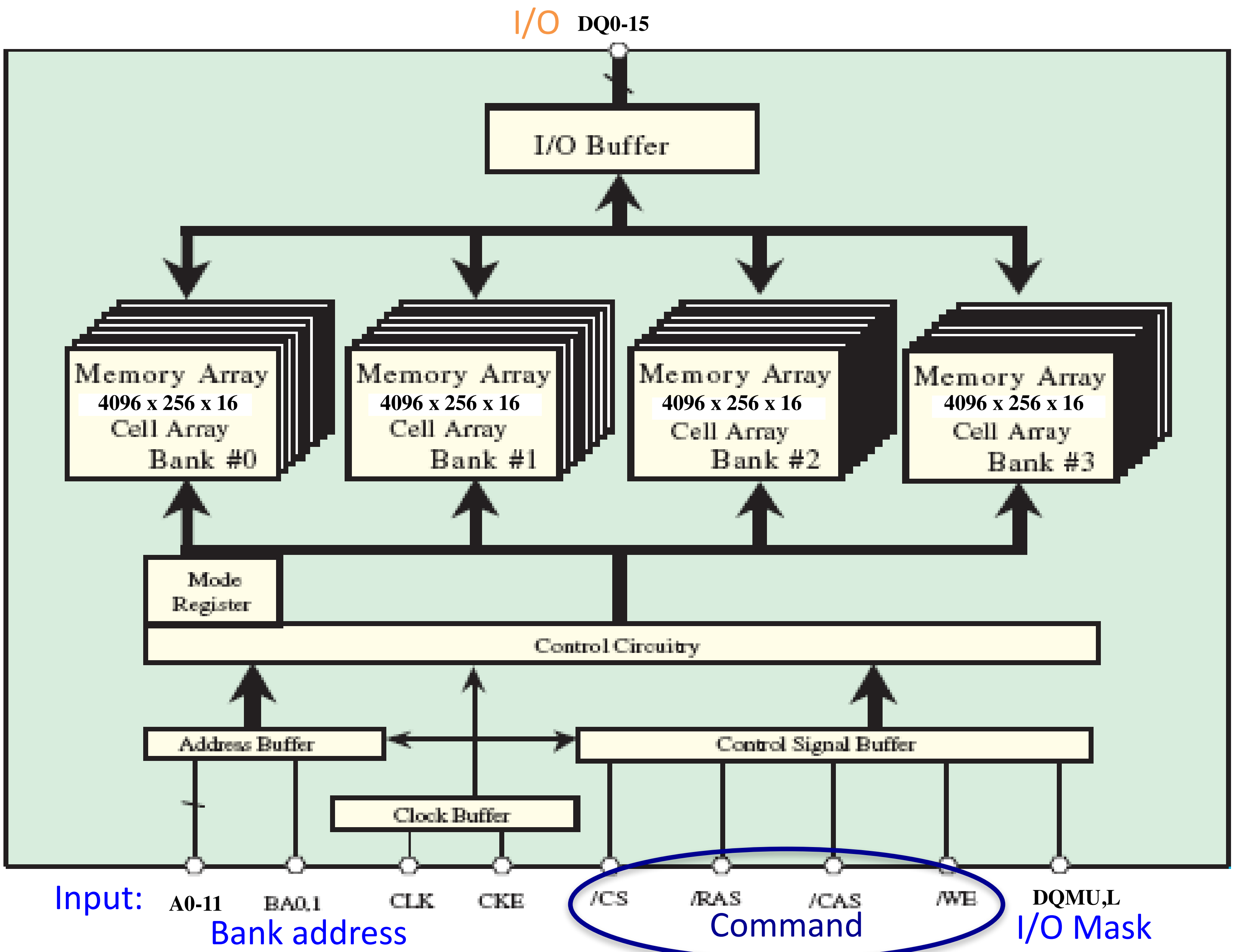
## Testing First Nios System (2)

- Go back to Qsys, add the JTAG-UART peripheral (Library/Interface Protocols/Serial), 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 together w/ the system ID and timestamp

# Testing First Nios System (3)

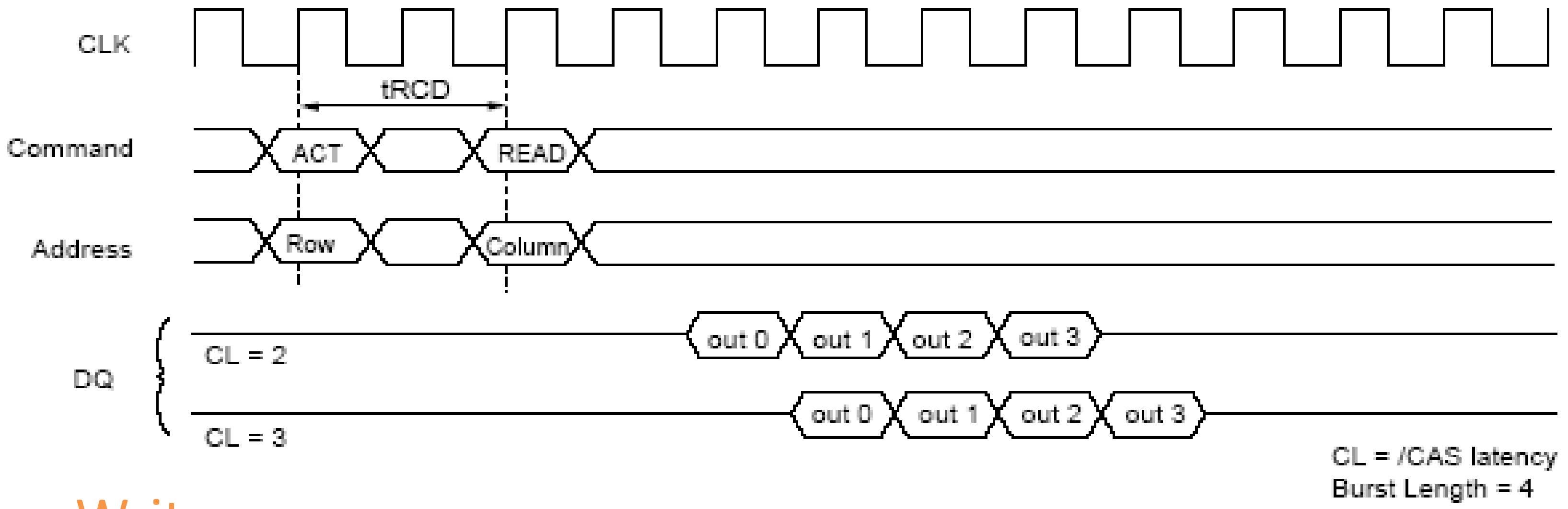
- Allocated on chip memory is not enough!
  - JTAG-UART device driver requires more memory than the one available
  - In a future lesson, we will learn some techniques to reduce the memory footprint of our software
  - Now, we can:
    - try to enlarge the on-chip memory. Note that EP2C35 FPGA has 105 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)

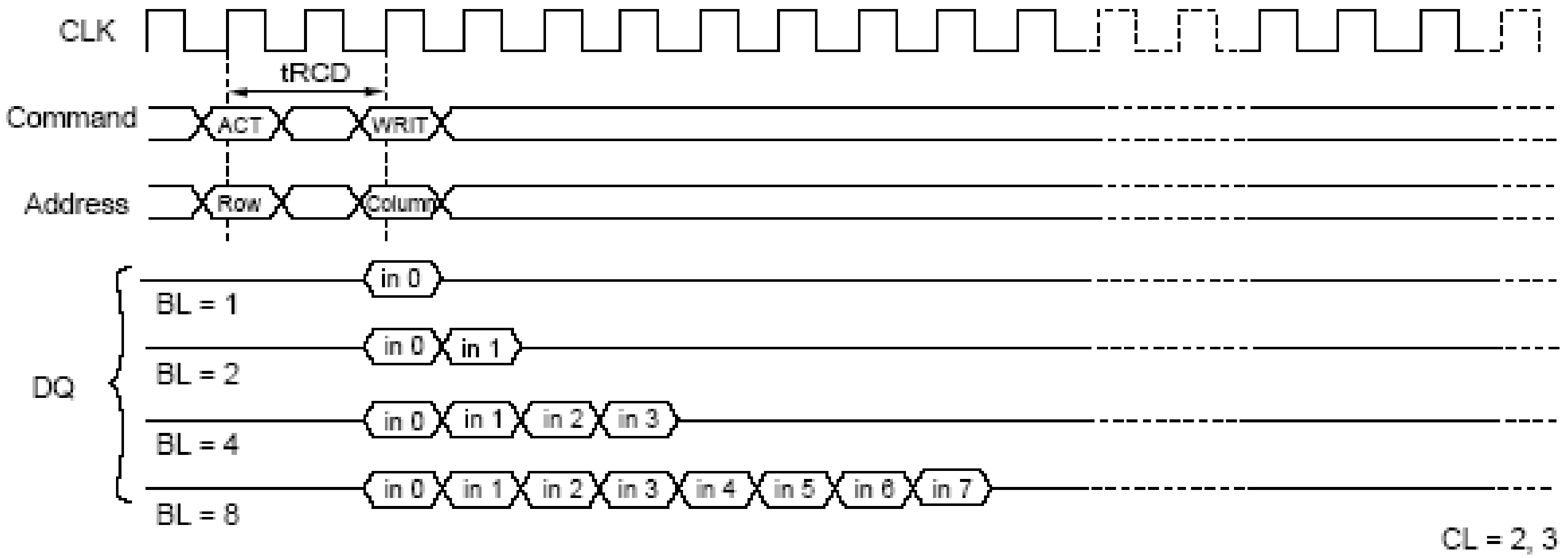


# SDRAM memory (2)

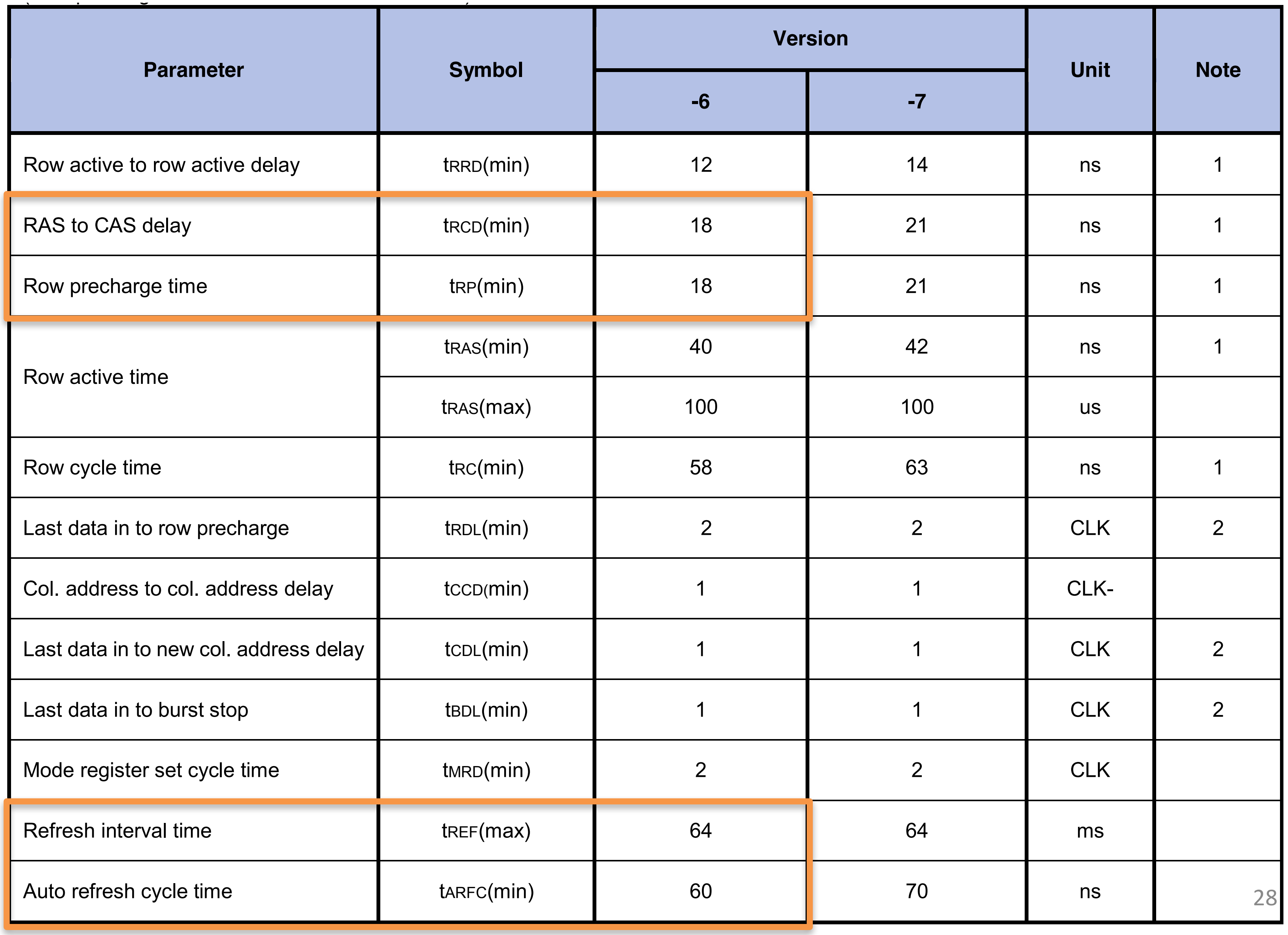
## Read



## Write

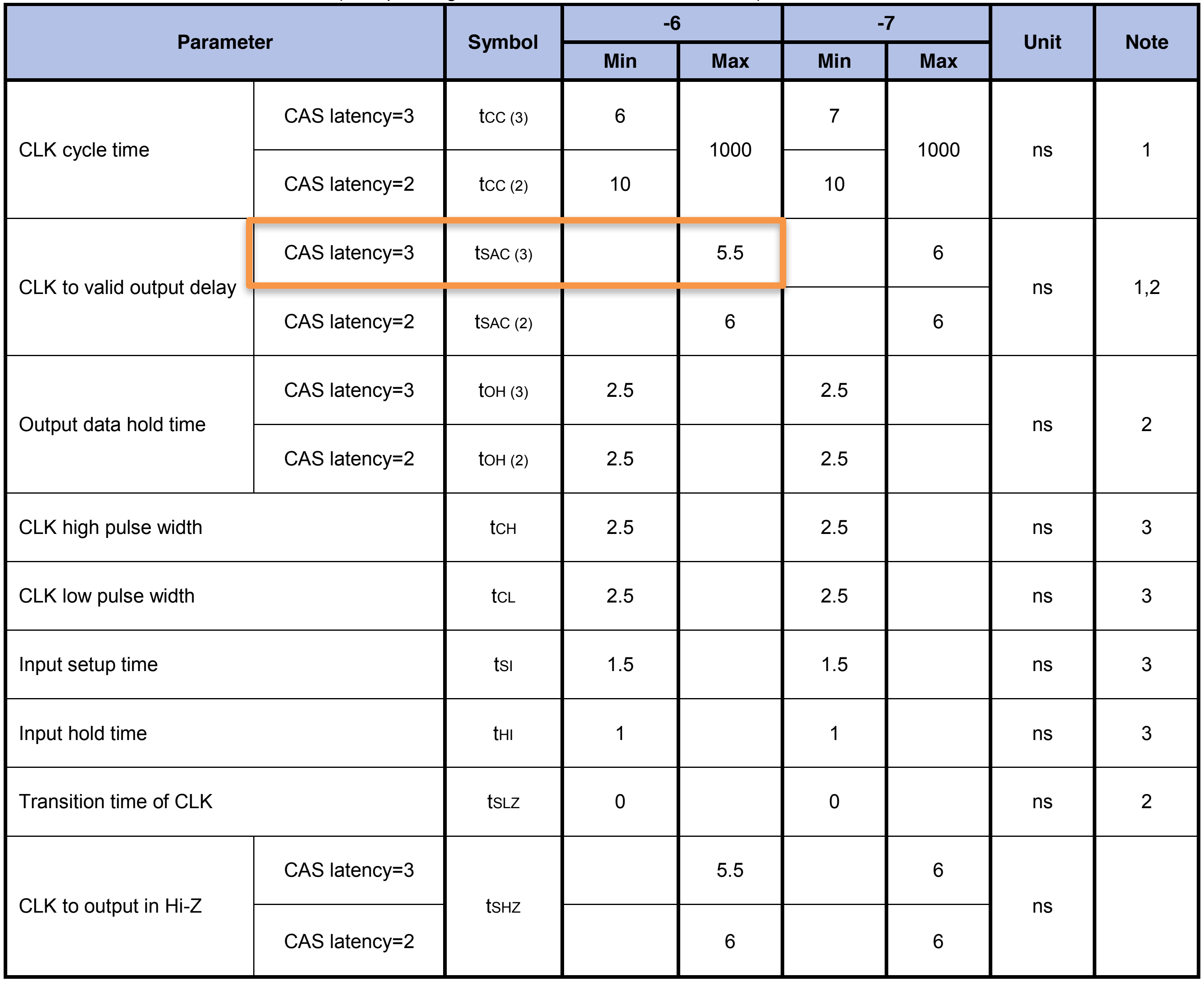


# SDRAM memory (2)



| Parameter | Symbol | Version | | Unit | Note |
|---|---|---|---|---|---|
| | | -6 | -7 | | |
| Row active to row active delay | tRRD(min) | 12 | 14 | ns | 1 |
| RAS to CAS delay | tRCD(min) | 18 | 21 | ns | 1 |
| Row precharge time | tRP(min) | 18 | 21 | ns | 1 |
| Row active time | tRAS(min) | 40 | 42 | ns | 1 |
| | tRAS(max) | 100 | 100 | us | |
| Row cycle time | tRC(min) | 58 | 63 | ns | 1 |
| Last data in to row precharge | tRDL(min) | 2 | 2 | CLK | 2 |
| Col. address to col. address delay | tCCD(min) | 1 | 1 | CLK- | |
| Last data in to new col. address delay | tCDL(min) | 1 | 1 | CLK | 2 |
| Last data in to burst stop | tBDL(min) | 1 | 1 | CLK | 2 |
| Mode register set cycle time | tMRD(min) | 2 | 2 | CLK | |
| Refresh interval time | tREF(max) | 64 | 64 | ms | |
| Auto refresh cycle time | tARFC(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)

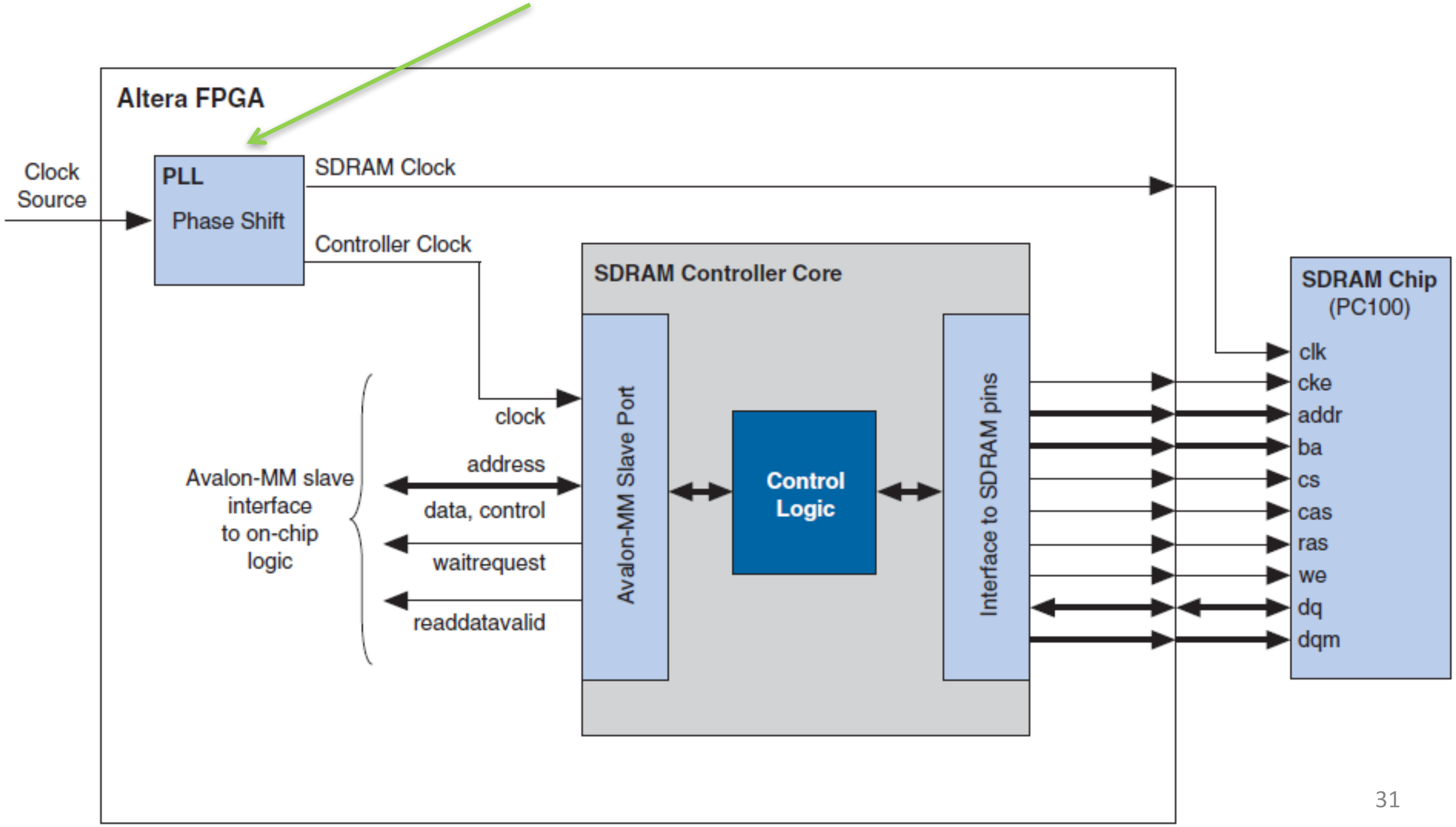
**Initialization sequence**

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)

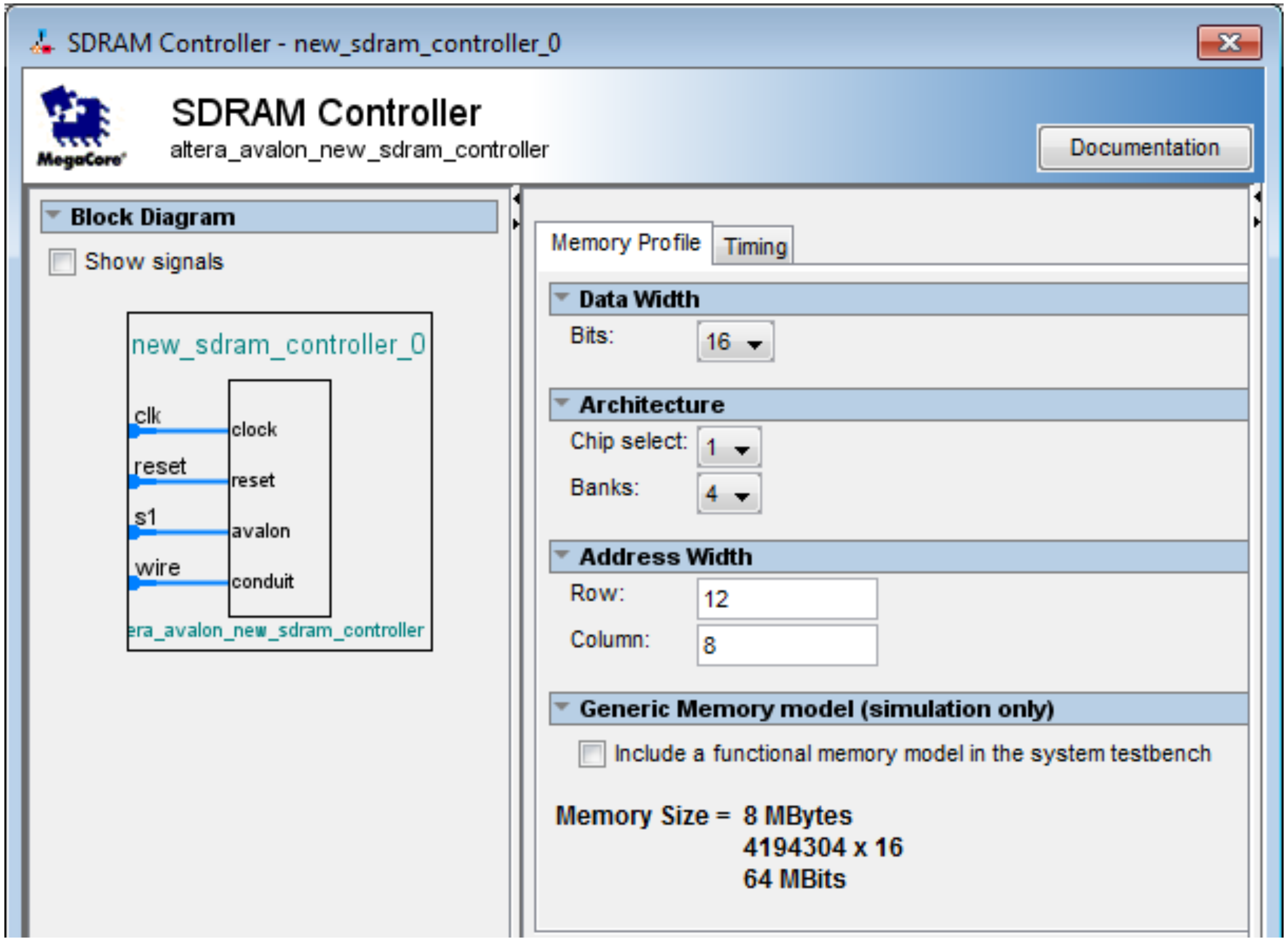
A PLL can be used to compensate for the clock skew

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



# SDRAM controller (2)

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



Show signals: ☐

**Data Width**

Bits: 16

**Architecture**

Chip select: 1

Banks: 4

**Address Width**

Row: 12

Column: 8

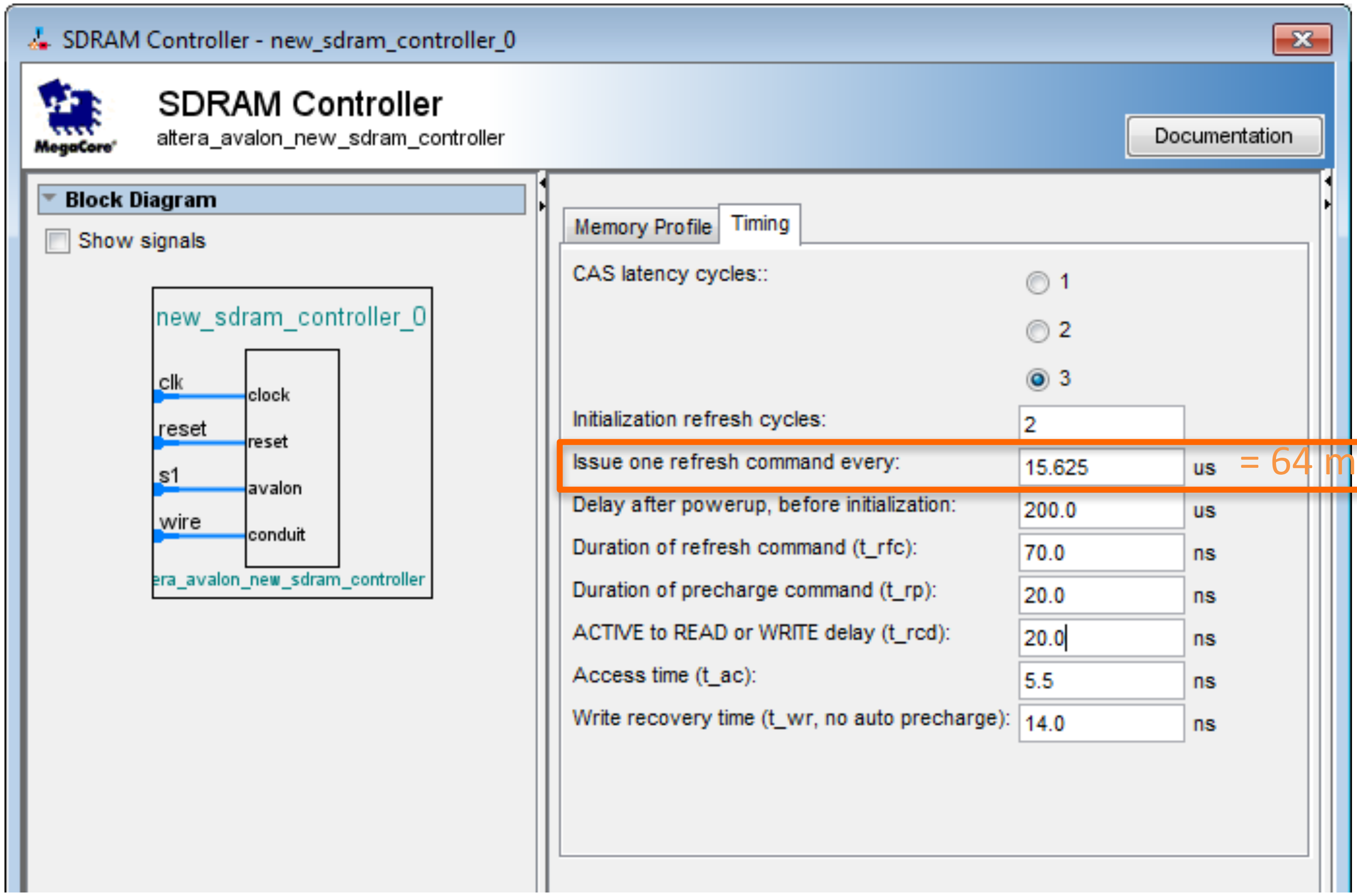
**Generic Memory model (simulation only)**

☐ Include a functional memory model in the system testbench

Memory Size = 8 MBytes
4194304 x 16
64 MBits

# SDRAM controller (3)

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



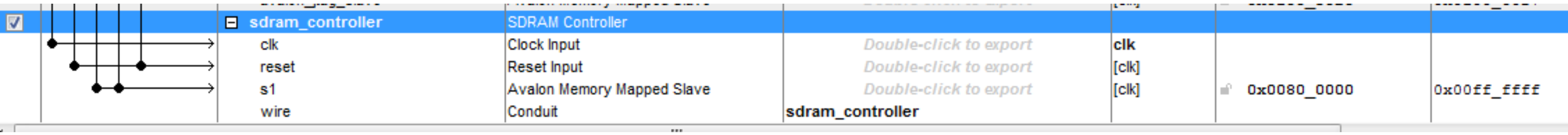
CAS latency cycles:: ( ) 1 ( ) 2 [✓ 3]

| | | |
|---|---|---|
| Initialization refresh cycles: | 2 | |
| Issue one refresh command every: | 15.625 | us |
| Delay after powerup, before initialization: | 200.0 | us |
| Duration of refresh command (t_rfc): | 70.0 | ns |
| Duration of precharge command (t_rp): | 20.0 | ns |
| ACTIVE to READ or WRITE delay (t_rcd): | 20.0 | ns |
| Access time (t_ac): | 5.5 | ns |
| Write recovery time (t_wr, no auto precharge): | 14.0 | ns |

= 64 ms/4096

# 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



| | | | | | |
|---|---|---|---|---|---|
| ⊟ sdram_controller | SDRAM Controller | | | | |
| 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] | 0x0080_0000 | 0x00ff_ffff |
| wire | Conduit | sdram_controller | | | |

- 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 controller (6)

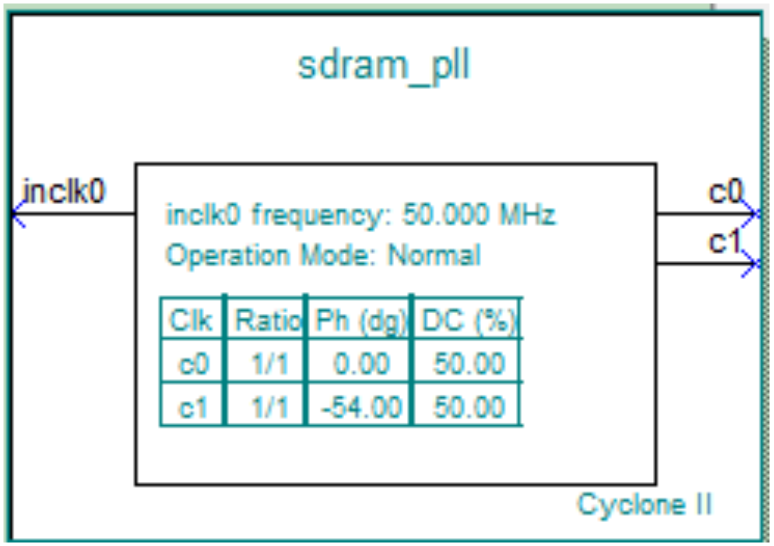
```
// my_DE2_first_computer.v

module my_DE2_first_computer(
        //input
        CLOCK_50,
        KEY,
        SW,
        //output
        LEDG
        // Memory (SDRAM)
        DRAM_DQ,
        DRAM_ADDR,
        DRAM_BA_1,
        DRAM_BA_0,
        DRAM_CAS_N,
        DRAM_RAS_N,
        DRAM_CLK,
        DRAM_CKE,
        DRAM_CS_N,
        DRAM_WE_N,
        DRAM_UDQM,
        DRAM_LDQM
);
```

# SDRAM controller (6)

```
    input                   CLOCK_50;
    input       [0:0]       KEY;
    input       [7:0]       SW;
    output      [7:0]       LEDG;
    //  Memory (SDRAM)
    inout       [15:0]      DRAM_DQ;
    output      [11:0]      DRAM_ADDR;
    output                  DRAM_BA_1;
    output                  DRAM_BA_0;
    output                  DRAM_CAS_N;
    output                  DRAM_RAS_N;
    output                  DRAM_CLK;
    output                  DRAM_CKE;
    output                  DRAM_CS_N;
    output                  DRAM_WE_N;
    output                  DRAM_UDQM;
    output                  DRAM_LDQM;
    // Add the nios_system instance
        //  The instance template can be copied from Qsys HDL example tab
    // (Generate) and Connect the SDRAM Clock (DRAM_CLK)
endmodule
```

# SDRAM Clock

- **DRAM_CLK must lead CLOCK_50 by 3 ns**
- 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



inclk0 frequency: 50.000 MHz
Operation Mode: Normal

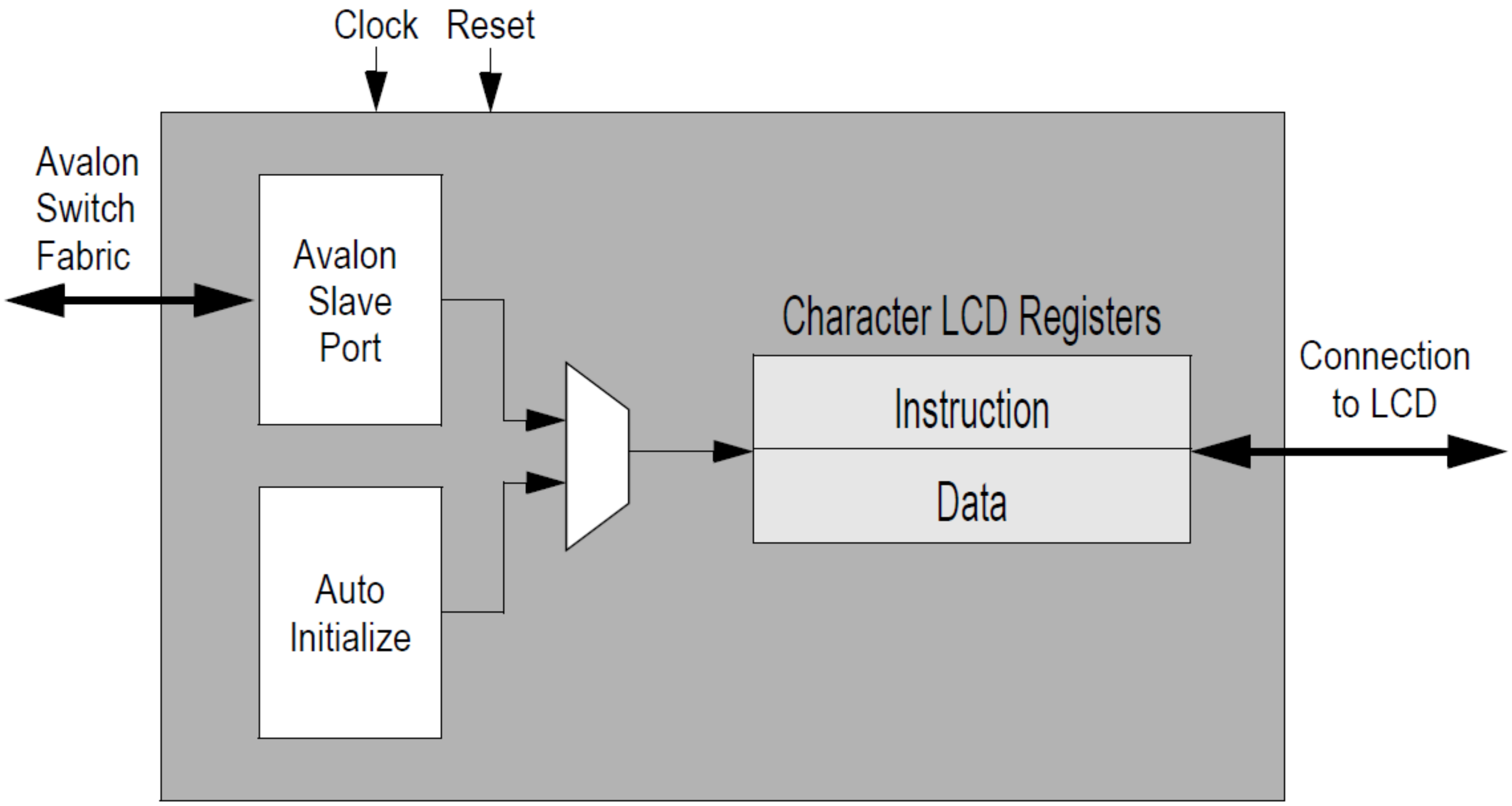
| Clk | Ratio | Ph (dg) | DC (%) |
|---|---|---|---|
| c0 | 1/1 | 0.00 | 50.00 |
| c1 | 1/1 | -54.00 | 50.00 |

## 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*