## SISTEMI EMBEDDED

Building a Nios II <u>Computer</u> from scratch

Federico Baronti

Last version: 20180326

## 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 (<u>Avalon System Interconnect Fabric</u>) are generated automatically by the tool
      - Need for standard interfaces

## Avalon System Interconect Fabric

- Overview of Avalon standard interfaces:
  - 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 (64 KB)
- 2 PIOs
  - Input for reading slider switches (10 bit)
  - Output for driving red LEDs (10 bit)
- System ID Peripheral for computer identification

### Nios II Hardware Flow



#### **Qsys Flow**



# Guided example (1)

- Open the template Quartus project: DE10\_Lite\_First\_Computer
- 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* (dual-port configuration, single clock)
  - PIO: *leds* 
    - Output for driving LEDS
  - PIO: sliders
    - Input for reading slider switches status
  - System ID Peripheral: sysid (ID = 1!)

### Qsys main window

🗼 Qsys File Edit System View Tools Help	Component instance name	Base address	×
Component Library	System Contents       Address Mar       Clock Settings       Project Settings       Instance Parameters       System Inspector       HDL Example       Gener         Image: System Contents       Address Mar       Clock Settings       Instance Parameters       System Inspector       HDL Example       Gener         Image: System Contents       Clock Settings       Description       Export       Clock         Image: System Contents       Clock Source       Clock Source       Clock Input       Clock Input         Image: System Contents       Clock Input       Clock Output       Clock Clock Input       Clock Input         Image: System Contents       Clock Clock Output       Clock Output       Clock Input       Clock Input         Image: System Contents       Clock Clock Output       Clock Output       Clock Input       Clock Input         Image: System Contents       Clock Clock Output       Clock Output       Clock Input       Clock Input         Image: System Contents       Clock Clock Output       Clock Output       Clock Input       Clock Input	Base	En
<ul> <li>Bridges</li> <li>Bridges and Adapters</li> <li>Clock and Reset</li> <li>Configuration &amp; Programming</li> <li>DSP</li> <li>Embedded Processors</li> <li>Interface Protocols</li> <li>Meroires and Memory Controllers</li> <li>Microcontroller Peripherals</li> <li>Peripherals</li> <li>PLL</li> <li>Osys Interconnect</li> <li>University Program</li> <li>Verification</li> <li>Window Bridge</li> </ul>	Configure internal connections Configure internal connections Decide signals to be routed (exported) to the Osys system boundary	ts	4
Messages Description	Path		

## **CPU** choice

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

negoCore Nio	s II Processor			About Document
Parameter Settlerer				
Core Nos II Ca	aches and Memory Interfaces	Advanced Reatures	MMU and MPU Settings > JT	AG Debug Module 🔷 Custom Instru
Core Nics II				
Select a Nios II core	#:			
	Nios II/e	ONios II/s	ONios II/f	
Nios II Selector Cuide Family: Cyclone II f <sub>system</sub> : 50.0 MHz cpuid: 0	RISC 32-bit	RISC 32-kit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide	RSC 32-bt Instructor Cache Branch Prediction Hordware Multidy Hordware Divide Barrel Shifter Data Cache Dynamic Branch Predictio	n
Performance at 50.0 M	MHz Up to 5 DMIPS	Lip to 25 DMIPS	Up to 51 DMPS	
Logic Usage	600-700 LEs	1200-1400 LEs	1400-1800 LEs	
Memory Usage	Two M4Ks (priequivi)	Two M4Ks + cache	Three M4Ks + cache	
Hardware Multiply:	Connor Multiplic's	Hardware Dwice		
Read Vector: A	Janong	Dffset: pv		
Evention Ventor: M			·	
Ехсерцы уецы. ".	emory.	V 011301. UX2	1	
T THE R MINU				
Only include the MMU	when using a reserving	Here and the scale one all W	MU	
Fast TLB Miss Excepti	ion Vector: Memory:	1	0ffset 0x0	
include MPU				
🕂 Warning Reselve	ctor and Exception vector canno	i be set until memory devices	are connected to the Nos II proces	ssor

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 (Pa atters_avalor	rallel I/O)			
Block Diagram				
clock reset avaion conduit	pio_U cik reset si external_connection			
Basic Settings				
Width (1-32 bits):	в			
Direction	🔿 Bidir			
	⊙ liqui			
	◯ InOut			
	Output			
Output Port Resst Value	Dx000000000000000000000000000000000000			
Output Register				
Enable individual bit	setting/clearing			
Edge capture register	ir 🗌			
Synchronously copt	ure			
Edge Type:	RISING 🖌			
Enable bit-clearing f	or edge capture register			
<ul> <li>Interrupt</li> </ul>				
🔄 Generate RQ				
IRG Type:	LEVBL V			
Level: Interrupt CPU when any unwested IO pin is logic brue Edge: Interrupt CPU when any unwested bit in the edge-capture register is logic brue. Available when synchronous capture is enabled				
Test bench wiring				
Hardwire PIO inputs	in test bench			
Drive inputs to:	Dx00000000000000			
🕕 Inte: pio_0: PIC inputs	are not hardwired in test bench. Undefined values			
< U	>			
	Cancel Finish			

MegaCore altera_avalon_sysid	K	
Show signals  Sysid  clk  reset reset control_slave avalon atte	ra_avalon_sysid	me the system is generated.

## **On-chip memory**

- Define the organization of the on chip-memory
  - Type = RAM
  - Dual-port access
  - Single clock operation
  - Size = 65536 bytes
  - Word length = 32
- Initialization file: <*nios\_system*>\_onchip\_ memory.hex

On-Chip Memory (RAM)	or ROM) - onchip_memory2_0			
MageCare On-Chip M aters_svsion_onc	emory (RAM or ROM) htp_metrory2			
Block Diagram				
	onchip_memory2_0 clock == dk1 avaion == s1 reset == reset1			
Memory type				
Туре:	RAM (Writable)			
Duskport access				
Read During Write Mode:	DONT_CARE M			
Block type:	Auto 👻			
▼ Size				
Data width:	32 🗸			
Total memory size:	4096 bytes			
Minimize memory black us	sage (may Impact fmax)			
Read latency				
Slave st Latency:	1 😒			
Slave s2 Latency:				
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 D:	NONE			
	Carcel			

# 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, leds), control\_slave (sysid)
  - instruction\_master (Nios II proc.), jtag\_debug\_module (Nios II proc.), s2 (onchip\_memory)

## Guided example (3)

#### Export external connections

 Sliders and 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	
<b>V</b>		🗖 cik	Clock Source					
		clk_in	Clock Input	clk				
	$\sim \rightarrow$	clk_in_reset	Reset Input	reset				
		clk	Clock Output	Double-click to export	clk			
		clk_reset	Reset Output	Double-click to export				
1		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]		0x4fff	
	×	custom_instruction_master	Custom Instruction Master	Double-click to export				
<b>v</b>		green_leds	PIO (Parallel I/O)					
	$  + + + + \rightarrow  $	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]		0x500f	
		external_connection	Conduit	green_leds_external_connection				
1		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]		0x501f	
		external_connection	Conduit	sliders_external_connection				
<b>V</b>		⊡ sysid	System ID Peripheral					
	$  + + + + \rightarrow  $	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]		0x5027	
1		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]		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/Edit the root module of the project
  - Include the Nios\_system module as hierarchical block (Verilog)
  - Compile the project to make the hardware ready

## Guided example (7)

- Integrating Qsys system into Quartus II project
  - <u>Method I</u>: 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

Max 10: 10M50DAF484C7G



## Guided example (7b)

#### Project root module

```
// DE10_Lite_First_Computer.v
```

```
module DE10_Lite_first_computer(
    //input
    MAX10_CLOCK1_50,
    KEY,
    SW,
    //output
    LEDR
);
    input MAX10_CLOCK1_50;
```

```
input [1:0] KEY;
input [9:0] SW;
```

```
output [9:0] LEDR;
```

#### // Add the nios\_system instance

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

## Guided example (7c)

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

// DE10\_Lite\_First\_Computer.v

```
module DE10_Lite_first_computer(
input MAX10_CLOCK1_50,
input [1:0] KEY,
input [9:0] SW,
```

output [9:0] LEDR

);

#### // Add the nios\_system instance

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

# Testing First Nios System (1)

- Write a program that makes the RED 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 DE10-Lite 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

# 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