431 - Introduction to VLSI CAD
Video acquisition and display: part 2 (NIOS)
Department of Electrical and Computer Engineering, McGill University
Montréal, Québec, Fall 2013

# 1 Introduction

Quartus II Qsys[1] tool is used to generate a Qsys component interconnect system. The Qsys system is then included as a sub-system entity inside the FPGA top-level module. Interconnected components are taken from the Qsys component library (embedded processor, on-chip memories, etc.) and used-defined custom components (ex: register file). Those components use standard protocol to communicate between each other through the interconnect, which is composed of automatically generated logic taking care of arbitration, priority and bus sizing among other things. The Qsys interconnect supports a few open-standard AMBA buses like AXI-3 and AXI-4, but this project should use the Altera Avalon standard[2] with memory-mapped Avalon master and slave interface types. Memory-mapped Avalon interface types are composed of a few dedicated signals (address, data, read request, write request, clock, etc.) which behavior must follow the Avalon standard defined for those interface types.

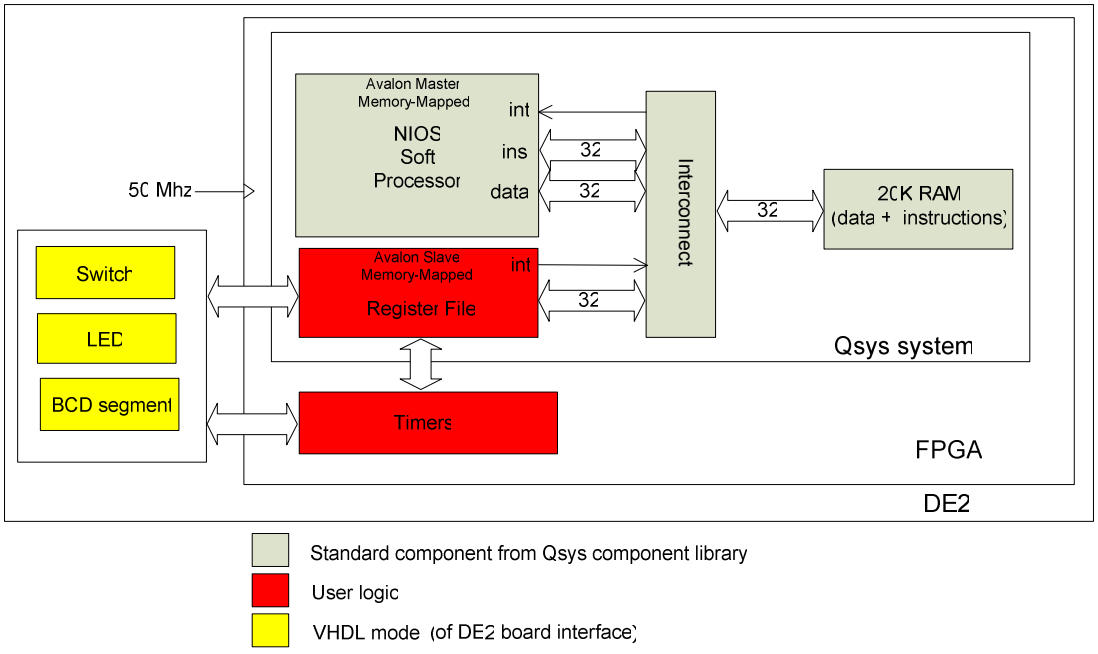


**Figure 1. FPGA with Qsys NIOS subsystem.**

In the block diagram shown in Figure 1, the NIOS and the block RAM are standard components instantiated from the Qsys component library. The NIOS has accessed to the RAM for instructions and data from which it can operate. It must also be able to communicate to the register file through the Qsys interconnect. You have to created a custom register file component and configure it through the component editor in Qsys, which actually creates a wrapper around "your" logic such as it can be connected using the Qsys GUI. Note that the Qsys system that is going to be generated is **not** the top level of your system. You would need to **manually** instantiate the Qsys system into your FPGA top level. Further to the Qsys system, your FPGA top level VHDL file includes two programmable timers that **you have to design**. Those timers are responsible for generating events that can trig interrupts to be sent toward the processor. The NIOS code is explicitly written around these events that are generated by the timers (refer to section 3). In order to exercise the NIOS, you must first build your Qsys system.

[1] Quartus II Handbook v13.0, Volume I, Section II, "System Design with Qsys"
[2] Altera "Avalon Interface Specifications".

# 2 Qsys

In Quartus, start building your system with the Qsys tool by adding the NIOS and its associated ROM and RAM blocks from the component library and interconnect them. Once your register file is designed (at least the entity), create a new component (File -> New Component) and add it to your system. The ports of the register are associated to five interfaces: 1- An Avalon slave memory-mapped interface, 2- a clock sync interface, 3- a reset sink interface, 4 - an interrupt interface and 5- a conduit_end interface. The conduit_end interface maps all non-Avalon signals (i.e. all signals that are connected to the timer logic block, the LEDs, switches and BCD segments of the DE2 board). Once your register file component has been created and added to your system, you can generate your system (it is highly recommended to go through the "NIOS II Hardware Development Tutorial" before proceeding with the actual system requires for this project). In order to perform simulation of the system later, you need to check the option indicating to Qsys that it has to generate files for the testbench. During generation, all the files pertaining to the testbench and synthesis will be created, including your Qsys top level (to be distinguished from your FPGA top level file). At this point, the memory initialization files required for the NIOS to operate are missing. in order to simulate the Qsys system with Modelsim. Initially, read and write accesses to the register file could be simulated, and later, the entire system including the timers and DE2 board model (if desired). The Qsys system will later be used as a based for the software driver controlling a video acquisition application

## 2.1 Notes

- For simulation purposes, you must specify that your simulation model must be taken "from synthesis" in the component editor.

# 3 Timers

Two timers must be instantiated into the design, Timer 0 and Timer 1. Timer 0 serves as a time base for the NIOS processor and generates an event once every second. The routine executed by the processor on the occurrence of timer 0 event displays on the BCD segments HEX(4:0) the amount of timer 1 events that occurred since the last timer 0 event. In other words, it displays how many ISRs associated with timer 1 were served during the last second. The interconnectivity between a timer and the register file in the Qsys system resembles to what is shown in Figure 1.

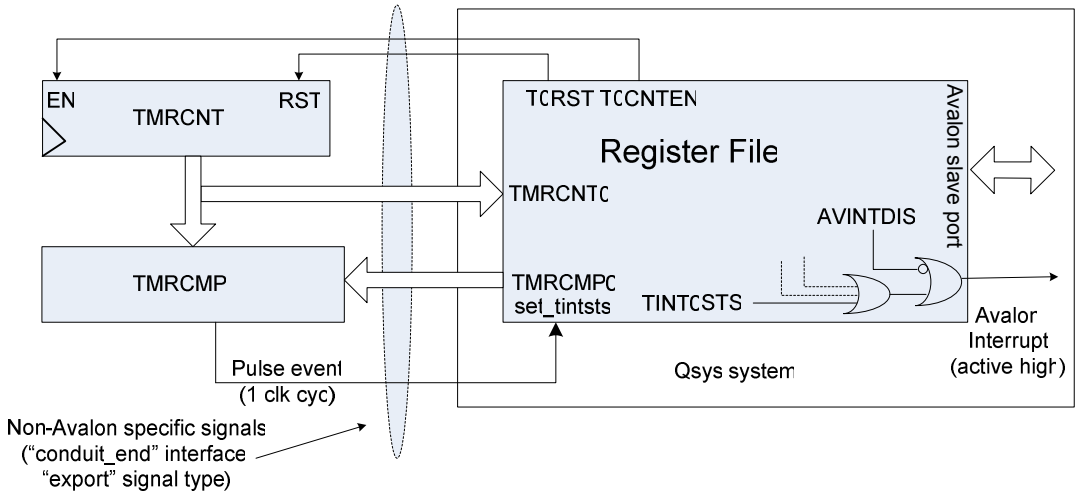


**Figure 2. Timer block diagram and connectivity with register file.**

# 4 NIOS II System Build Tool (SBT)

Once your Qsys system has been generated, start the ”NIOS II Software Build Tools for Eclipse. This is in there that you add your BSP and C/C++ application. The BSP (Board Support Package) is created by specifying your .sopcinfo file generated by. The SBT is aware of the hardware characteristic of your system (address mapping, Avalon components, etc.). Once the project is created, you can add your source main file, code your application and build your project. You can then use the “run as hardware” tool to download the code directly into the NIOS. In order to initialize the RAM upon FPGA configuration, right-click on you C application, and choose Make Target -> Build, and select “generate”). The .qip file that is created under the mem_init directory needs to be added to you Quartus project, and on the next compilation, it will be used to initialize the RAM.

# 5 NIOS

The highest rate achievable to serve the interruot routine executed on the NIOS II processor has to be determined. On every occurrence of timer 1 events, the NIOS enters an ISR that executes some “register manipulations”. Those registers are part of the register file, which means they are not “internal” to the processor. The interrupt service routine of timer 1 event consists of the following:

- GP0 = GP0 +1
- GP1 = GP1 * GP0

By reading the interrupt control register (refer to section 7, INTCTL register), the processor can identify the source of the interruption, and execute the code associated with it. Besides timer 0 and timer 1 interrupt sources, there is also an overrun interrupt source which is used to inform the application that it did not have enough time to execute the ISR before another interrupt (of timer 1) occurred. The interrupt overrun source is a flag that is set whenever the timer 1 event occurs while its interrupt status bit (T1INTSTS) is still active. Whenever an interrupt overrun occurs, led LEDR(0) should be activated. There are no requirements to provide interrupt overrun on timer 0 as this one only serves as a time-base. Note that the entire system operates on a 50 MHz external clock.

The application must also be written to poll (read) the INTCTL register instead relying on the interrupts. This makes the NIOS continuously read the INTCTL register until it detects an event. The highest “polling status bits for routine execution rate” also need to be determined.

## 5.1 Implementation notes

CHECK following two points:

- For the NIOS to access the register file, you can use dedicated IO read and write functions to do so. Use IORD_32DIRECT() and IOWR_32DIRECT(), documented in chapter 9 of the NIOS II Software Developer’s Handbook. You can also use a pointer to an integer (volatile int *ptr) to access your custom Avalon component.
- Pre-defined addresses (#DEFINE) of your Qsys component (which were mapped into Qsys system) can be found in the system.h file that is created during the BSP generation (under the NIOS_II_BSP_project_name/software directory).

# 6 Functional Simulation

A successful FPGA implementation involves taking the design through a phase of validation using a functional simulator. This step is followed by an implementation phase using a synthesis and place and route tool. The last phase would be to exercise the design on the actual hardware with the generated bitstream. We must avoid spending too much time debugging on the hardware because it is more time consuming than debugging using a functional simulator (by one order of magnitude…). The time being spent at the beginning of a project to develop a validation methodology by creating accurate VHDL models and predicting the outputs of a simulation is a very good investment to save time later debugging your hardware. This is going to be especially true for the final project. This is the reason why a **mandatory** simulation of the complete system is required.

Before simulating with the timers, Qsys can create a testbench for you, which can be invoked by ModelSim through the NIOS II SBT. This could initially serve to test if your register file can respond to read and write accesses. Once this basic simulation works, you can go a step further by adding timers, FPGA and top level system VHDL files run the simulation including your timer, and even DE2 interfaces model if you desire. Initially, you may want to decrease the time base of timer 0 to 1 or 10 ms in order to decrease simulation time. Note that you might need to run the simulation a few hundreds of us in order for the NIOS to complete its boot process and get into the routine.

# 7 Register file

The register file is made of a few 32-bit DWORD aligned registers that permits the configuration of the timers and allow the control of the interrupt flow. Register bit fields may have different attribute settings, some bits being readable only (RO), readable and writable by the processor (R/W), write only (WO) or read-write-to-clear (readable and writable to 1 to clear the bit). The register file is custom Qsys component.

## 7.1 Register description

**INTCTL** address 0x00 INTerrupt ConTroL

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Reserved | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| Reserved | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Reserved | | AVINTDIS | T1INTOVR | T1INTSTS | T0INTSTS | T1INTEN | T0INTEN |

| TxINTEN<br>RW | Timer INTerrupt ENable<br>This bit enables or disables detection of the timer x event. |
|---|---|

| Value | Description |
|---|---|
| 0b | Timer x interrupt is disabled. |
| 1b | Timer x interrupt is enabled. |

| TxINTSTS<br>RW2C | Timer x INTerrupt StaTuS<br>This bit indicates the status of the timer x interrupt. Writing 1 clears this bit if it is set. Writing 0 has not effect. |
|---|---|

| Value | Description |
|---|---|
| 0b | Timer x interrupt is inactive |
| 1b | Timer x interrupt is active |

| T1INTOVR<br>RW2C | Timer1 InTerrupt OVeRrun<br>This bit is set to one to indicate that an interrupt overrun has occurred for timer 1. Writing this bit to one clears the bit to zero. Writing zero has no effect. The value of this register field can be monitored on a red LED. |
|---|---|

| Value | Description |
|---|---|
| 0b | No interrupt overrun |
| 1b | An interrupt overrun has been detected. |

| AVINTDIS<br>RW | Avalon INTerrupt DISable<br>This bit enables or disables the interrupt send on the Avalon bus. |
|---|---|

| Value | Description |
|---|---|
| 0b | Avalon interrupt is enabled. |
| 1b | Avalon interrupt is disabled. |

**TCTL** address 0x04 Timer ConTroL

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| Reserved | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Reserved | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| Reserved | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Reserved | | | | T1CNTEN | T0CNTEN | T1RST | T0RST |

| TxRST | Timer x ReSeT |
|---|---|
| RW | This bit field can reset the timer x counter to 0. |

| Value | Description |
|---|---|
| 0b | Do nothing |
| 1b | Force timer counter x to 0. |

| TxCNTEN | Timer 0 CouNTer ENable |
|---|---|
| R/W | This bit enables/disables the 32-bit counter associated with timer x. |

| Value | Description |
|---|---|
| 0b | Timer x counter is disabled. |
| 1b | Timer x counter is enabled. |

| **T0CNT** | address 0x08 | Timer 0 CouNTer |
|---|---|---|
| **T1CNT** | address 0x0C | Timer 1 CouNTer |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| TxCNT(31:24) | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| TxCNT(23:16) | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TxCNT(15:8) | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TxCNT (7:0) | | | | | | | |

| TxCNT<br>R0(31:0) | Timer x CouNTer<br>This register represents the value of a 32-bit counter associated with timer x. The counter increments once every 16 clock cycles of a clock running at 50 MHz. The counter is reset to zero whenever TMRCTL.TxRST is set, when the counter reaches the value programmed in TMRCMPx, or when the timer counter wraps-around. |
|---|---|

| Value | Any 32-bit value. |
|---|---|

| **T0CMP0** | address 0x10 | Timer 0 CoMParator |
|---|---|---|
| **T1CMP1** | address 0x14 | Timer 1 CoMParator |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| TxCMP(31:24) | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| TxCMP(23:16) | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| TxCMP(15:8) | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TxCMP (7:0) | | | | | | | |

| TxCMP<br>RW(31:0) | Timer x CoMParator<br>Whenever the value programmed in this register is reached by the timer counter, the timer counter is reset to zero. The event associated with timer x is generated when the value of TxCNT reaches TxCMP. |
|---|---|

| Value | Any 32-bit value. |
|---|---|

| **GP0** | address 0x18 | General Purpose 0 |
|---|---|---|
| **GP1** | address 0x1C | General Purpose 1 |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|
| GPx(31:24) | | | | | | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| GPx(23:16) | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| GPx(15:8) | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| GPx(7:0) | | | | | | | |

| GPx<br>RW | General Purpose x<br>This is a general purpose register. |
|---|---|
| Value | Any 32-bit value |

# 8 Evaluation Criteria (listed by degree of importance)

- Functionality of the system on the hardware (polling + ISR)
- Functional simulation (polling or ISR)
- Synthesis and place and route reports
- Improvement proposal, explanation of problems encountered, resolution (or not) and action taken.

# 9 Report, simulation and synthesis

The functionality of the project must be demonstrated on the DE2 platform. Do not hesitate to make use of the push button, LEDs, HEX and switches to improve flexibility of your design and enhanced the demonstration of your C program. These initiatives are taken into considerations during the demonstration. The functionality of the system also be demonstrated through a functional simulation with Modelsim. Key points in the simulation should be pointed out during your demo. Quality of the waveforms that you would select to show the functionality is essential. The constraint file along with the synthesis report summary must be given in the appendix, as well as the main files related to the program the NIOS is executing. In particular, the amount of warnings from the synthesis and place and route tools must be minimized, and ideally all paths must be constrained. The report is to be limited to 5 pages (single sided). Appendices are considered extra. Dual column format is recommended but not necessary. It should be clear and easy to read. A detailed block diagram must accompany all substantial piece of code. The report should include a short description of the design and any interesting or important elements encountered in the design process (clock frequency, processor core size, instruction and data memory sizes, interrupt routine size versus interrupt rate, etc.). Also, the interesting features of the design should be pointed out. You could include any state diagrams that indicate how your system operates as well as describe its overall operation.

Due date: Demo has to be scheduled between September 30th and October 9$^{th}$. The report should be handed during the demo.

# 10 History

- 1.0 Initial revision

Appendix A

- You might encounter a problem simulating your NIOS testbench with modelsim in VHDL. There is a limitations of 256 character in the pathnames that can cause the Modesim testbench to crash during compile. If that happens, do the following (excerpt from a resolution found on the web):

1. Backup the msim_setup.tcl file which is located
<project_directory>\software\<software_project\obj\default\runtime\sim\mentor directory.
2. Open niosII 13.0 command shell in Qsys and change to the <project directory>, where the *.spd file is located
3. Type the following command and hit enter to compile the BFM source files into single library.
ip-make-simscript --spd=<your_niosii_system>_tb.spd --outputdirectory=./software/<software_project>/obj/default/runtime/sim --compile-to-work
4. Next, copy lines involving your RAM memory from backup msim_setup.tcl and paste it into the new generated msim_setup.tcl file.
For example:
file copy -force H:/<project_directory>/software/hello_world_an351/mem_init/hdl_sim/niosii_system_ram.dat ./
file copy -force H:/<project_directory>/software/hello_world_an351/mem_init/niosii_system_ram.hex ./