Embedded Systems

"System On Programmable Chip"

NIOS II – Avalon Bus

René Beuchat

Laboratoire d'Architecture des Processeurs

rene.beuchat@epfl.ch
Embedded system on Altera FPGA

Goal:

• To understand the architecture of an embedded system on FPGA
• To be able to design a specific interface
• To be able to construct a full system based on a standard softcore bus in a FPGA and using blocs modules
• To understand, use and program a softcore processor
Contents

• NIOS II a softcore processor
• System On FPGA
• Avalon Bus
• Design of a specific slave programmable interface on Avalon

• Reference:
  http://www.altera.com/literature/lit-nio2.jsp
NIOS II

• Softcore Processor from Altera
  ➢ A processor implemented with Logic Elements (LUT+DFF) in a FPGA
  ➢ A processor synthesized by a compiler and placed & routed on the FPGA
  ➢ A processor described by a HDL language (VHDL/Verilog/…)

• 32 bits Architecture
• 3 versions
• 256 instructions available for user implementation
NIOS II –
Embedded system NIOSII/Avalon Architecture

Note: The same principles are available for Altera, Xilinx, Actel or others FPGA
AVALON Switch Fabric

Some Avalon specifications:

- Multi-Master
- Arbitrage « slave-side »
- Concurrent Master-Slave Access
- Synchronous transfers
## NIOS II Processor

### 3 processors architectures

<table>
<thead>
<tr>
<th></th>
<th>Nios II /f Fast</th>
<th>Nios II /s Standard</th>
<th>Nios II /e Economy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>6 Stage</td>
<td>5 Stage</td>
<td>None</td>
</tr>
<tr>
<td>Multiplier *</td>
<td>1 Cycle</td>
<td>3 Cycle</td>
<td>None</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>Dynamic</td>
<td>Static</td>
<td>None</td>
</tr>
<tr>
<td>Instruction Cache</td>
<td>Configurable</td>
<td>Configurable</td>
<td>None</td>
</tr>
<tr>
<td>Data Cache</td>
<td>Configurable</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
The ALU can be extended by user own instructions, until 256.
The instructions can be:

- Combinatorial, single clock cycle
- Multi-cycles, synchronized by clk and stall
- Parameterized

They can have access to all the FPGA resources.

They can use their own internal registers.
• For cycles consuming operations, a **hardware accelerator** can be included/developed

• A **Master unit** which has access to Memory and Programmable Interfaces for accelerated operations or with hard real time constrains
NIOS II Processor, hardware accelerator
Computer architecture

• Classical architecture
  ➢ Processor
  ➢ Memories
  ➢ Input/Output (programmable) interface
  ➢ Address bus
  ➢ Data Bus (tri-state)
  ➢ General decoder
Computer architecture on FPGA (Altera)

• SOPC architecture (Altera)
  ➢ Processor
  ➢ Memories
  ➢ Input/Output (programmable) interface
  ➢ Address bus
  ➢ Separated Data Bus In/Out $\rightarrow$ multiplexers
  ➢ Local decoder on the Avalon bus
  ➢ Bus transfers size adaptation is done at Avalon bus level
System on FPGA example
To interconnect all the masters and slaves inside the FPGA, an generated internal bus:

- Master/Slave modules
- Synchronous bus on clock rising edge
- Separate data in and data out
- Wait state by configuration or dynamic
- Hold / Set up available
- Actual version (>1.0) allows data path until **1024 bits** (8, 16, 32, 64, 128, 256, 512, 1024)
### Avalon « slave » main signals

<table>
<thead>
<tr>
<th>Signal Type</th>
<th>Width</th>
<th>Direction</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>1</td>
<td>In</td>
<td>(No)</td>
<td>Global clk for system module and Avalon bus modules. <strong>All transactions synchronous to clk rising edge</strong></td>
</tr>
<tr>
<td>nReset</td>
<td>1</td>
<td>In</td>
<td>No</td>
<td>Global Reset of the system</td>
</tr>
<tr>
<td>address</td>
<td>1..32</td>
<td>In</td>
<td>No</td>
<td>Address for Avalon bus modules</td>
</tr>
<tr>
<td>ChipSelect</td>
<td>1</td>
<td>In</td>
<td>Old</td>
<td>Selection of the Avalon bus module</td>
</tr>
<tr>
<td>read/read_n</td>
<td>1</td>
<td>In</td>
<td>No</td>
<td>Read request to the slave</td>
</tr>
<tr>
<td>ReadData</td>
<td>8, 16, 32, .. (1024)</td>
<td>Out</td>
<td>No</td>
<td>Read data from the slave module</td>
</tr>
<tr>
<td>write/write_n</td>
<td>1</td>
<td>In</td>
<td>No</td>
<td>Write request to the slave</td>
</tr>
<tr>
<td>WriteData</td>
<td>8, 16, 32, .. (1024)</td>
<td>In</td>
<td>No</td>
<td>Data from Master to Slave module</td>
</tr>
<tr>
<td>Irq</td>
<td>1</td>
<td>Out</td>
<td>No</td>
<td>Interrupt request to the master</td>
</tr>
</tbody>
</table>
Avalon
« slave » signals

• The **Address**[\(n .. 0\)] is used to access a specific register/memory position in the selected module.

• An address is a **word address** view from the slaves. A word has the width of the slave interface: 8, 16, 32, 64, 128, 256, 512 or 1024 bits

• Only the minimum number of addresses is necessary. *Ex: a module with 6 internal registers needs 3 bits of addresses (6< 2**3)*
The **ChipSelect** is generated by the Avalon bus and selects the module, actually is included in read/write signals. *Thus it is deprecated*

The **Read** and **Write** signals specifies the direction of the transfers and validate the cycle. They are provided by a Master and received by the slave modules.

The direction is the view of the Master unit.

**ReadData(..)** and **WriteData(..)** bus transfers the data from (read)/ to (write) the Slaves.
Avalon
« slave » signals

• **BE (Byte Enable)** signals specify the bytes to transfers.
  ➢ The number of BE activated are a power of 2
  ➢ They start at a multiple of the size to transfer

• A **master address** is a **byte** address

• A **slave address** is a **word** address

• The Avalon make the addresses translation and the multiple accesses if necessary
## Avalon byte enable (BE)

<table>
<thead>
<tr>
<th>ByteEnable_n[3..0]</th>
<th>Transfer action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>Full 32 bits access</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>Lower 2 Bytes access</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>Upper 2 Bytes access</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>Lower Byte (0) access</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>Mid Low Byte (1) access</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>Mid Upper Byte (2) access</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>Upper Byte (3) access</td>
</tr>
</tbody>
</table>

Specify bytes to be transferred
Active low signals in this representation:
- byteenable_n
Avalon
Master to slave addresses: Master 32 bits, Slave 8 bits

**Master Add**
- **BE 3**: 0x..0
- **BE 2**: 0x..4
- **BE 1**: 0x..8
- **BE 0**: Byte Address

**Slave Add**
- **BE 0**: Word = Byte (8 bits)

- **BE 1**: 0
- **BE 2**: 1
- **BE 3**: 2
- **BE 4**: 3
- **BE 5**: 4
- **BE 6**: 5
- **BE 7**: 6
- **BE 8**: 7
- **BE 9**: 8
- **BE A**: 9
- **BE B**: A

**Byte Address**
- **RB-P2012**
Avalon
Master to slave addresses: Master 32 bits, Slave 16 bits

Master Address | BE | BE | BE | BE | Slave Address
---------------|----|----|----|----|-----------------
0x..0          | 3  | 2  | 1  | 0  | 0x..0

0x..4          | 3  | 2  | 1  | 0  | 1x..0

0x..8          | 3  | 2  | 1  | 0  | 2x..0

Byte Address   Word = Doublet (16 bits)
4               | 5  | 0  |
Avalon
Master to slave addresses: Master 32 bits, Slave 32 bits

Master Add | BE | BE | BE | BE |
---|---|---|---|---|
0x..0 | 3 | 2 | 1 | 0 |

Slave Add | BE | BE | BE | BE |
---|---|---|---|---|
0 | 3 | 2 | 1 | 0 |

Word = Quadlet (32 bits)
Avalon
Master to slave addresses: Master 32 bits, Slave 64 bits

<table>
<thead>
<tr>
<th>Master Add</th>
<th>Slave Add</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x..0</td>
<td>0</td>
</tr>
<tr>
<td>0x..4</td>
<td>1</td>
</tr>
<tr>
<td>0x..8</td>
<td>1</td>
</tr>
</tbody>
</table>

Word = Octlet (64 bits)
<table>
<thead>
<tr>
<th>Signal Type</th>
<th>Width</th>
<th>Direction</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WaitRequest/WaitRequest_n</td>
<td>1</td>
<td>Out</td>
<td>No</td>
<td>Assert by the slave when it is not able to answer in this clock cycle to read or write access</td>
</tr>
<tr>
<td>ByteEnable/ByteEnable_n</td>
<td>1, 2, 4, 8, .., 128</td>
<td>In</td>
<td>No</td>
<td>The bytes to transfer</td>
</tr>
<tr>
<td>BeginTransfer</td>
<td>1</td>
<td>In</td>
<td>No</td>
<td>Inserted by Avalon fabric at and only at first clock of each transfer</td>
</tr>
<tr>
<td>ReadDataValid/ReadDataValid_n</td>
<td>1</td>
<td>Out</td>
<td>No</td>
<td>For read transfer with variable latency, means data are valid to master</td>
</tr>
<tr>
<td>BurstCount</td>
<td>1..11</td>
<td>In</td>
<td>No</td>
<td>Number of burst transfers</td>
</tr>
<tr>
<td>BeginBurstTransfer</td>
<td>1</td>
<td>In</td>
<td>No</td>
<td>First cycle of a burst transfer, valid for 1 clock cycle</td>
</tr>
</tbody>
</table>
### Avalon « slave » signals

<table>
<thead>
<tr>
<th>Signal Type</th>
<th>Width</th>
<th>Direction</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadyForData</td>
<td>1</td>
<td>Out</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>DataAvailable</td>
<td>1</td>
<td>Out</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>ResetRequest/ResetRequest_n</td>
<td>1</td>
<td>Out</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>ArbiterLock/ArbiterLock_n</td>
<td>1</td>
<td>In</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>
Slave view of transfers

- Transfers are synchronous on the rising edge of the Clk
- Between Clk, the timing relation between signals are NOT relevant
Avalon (slave view)
Read transfer, 0 wait, asynchronous peripheral

This Example Demonstrates
Read transfer from an asynchronous peripheral
Zero wait states
Zero setup

Relevant PTF Parameters
Read_Wait_States = “0”
Setup_Time = “0”

ReadData available at next rising edge of clk (E)
Avalon (slave view)
Read transfer, 1 wait

Wait cycle specified by design
Avalon (slave view)
Read transfer, 2 wait
Avalon (slave view)
Read transfer, wait request generated by slave device
Avalon (slave view)
Read transfer, 1 set up and 1 wait
Avalon (slave view)
Read transfer, burst of 4 from Master A, 2 from master B

Pipeline of master access

ReadDataValid activated by slave for each data
Avalon (slave view)
Write transfer, 0 wait

Diagram showing the Avalon bus interface with signal flow and timing diagram.
Avalon (slave view)
Write transfer, 1 wait
Avalon (slave view)
Write transfer, wait request generated by slave
Avalon (slave view)
Write transfer, 1 set up, 1 hold, 0 wait
Avalon (slave view)
Write transfer, burst transfer of 4, wait request generated by slave
Avalon (slave view)
Read transfers with latency (ex. 2 cycles)

Wait request here means:
- delay address cycle
- Fixed latency (here 2)
Avalon (slave view)

Read transfers with latency, and **readdatavalid** generated by slave

**readdatavalid** specify when data are ready
Bus avalon

Master view

• The master start a transfer (read or write)
• It provide the Addresses (32 bits on NIOSII)
• It waits on **WaitRequest** signal to resume the transfer
### Avalon master signals (1)

<table>
<thead>
<tr>
<th>Signal Type</th>
<th>Width</th>
<th>Direction</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>1</td>
<td>in</td>
<td>yes</td>
<td>Global clock signal for the system module and Avalon bus module. All bus transactions are synchronous to clk.</td>
</tr>
<tr>
<td>reset</td>
<td>1</td>
<td>in</td>
<td>no</td>
<td>Global reset signal. Implementation is peripheral-specific.</td>
</tr>
<tr>
<td>address</td>
<td>1 - 32</td>
<td>out</td>
<td>yes</td>
<td>Address lines from the Avalon bus module. All Avalon masters are required to drive a byte address on their address output port.</td>
</tr>
<tr>
<td>byteenable</td>
<td>0, 2, 4</td>
<td>out</td>
<td>no</td>
<td>Byte-enable signals to enable specific byte lane(s) during transfers to memories of width greater than 8 bits. Implementation is peripheral-specific.</td>
</tr>
<tr>
<td>read</td>
<td>1</td>
<td>out</td>
<td>no</td>
<td>Read request signal from master port. Not required if master never performs read transfers. If used, readdata must also be used.</td>
</tr>
<tr>
<td>readdata</td>
<td>8, 16, 32</td>
<td>in</td>
<td>no</td>
<td>Data lines from the Avalon bus module for read transfers. Not required if the master never performs read transfers. If used, read must also be used.</td>
</tr>
<tr>
<td>write</td>
<td>1</td>
<td>out</td>
<td>no</td>
<td>Write request signal from master port. Not required if the master never performs write transfers. If used, writedata must also be used.</td>
</tr>
</tbody>
</table>
## Avalon master signals (2)

<table>
<thead>
<tr>
<th>Signal Type</th>
<th>Width</th>
<th>Direction</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>writtenata</td>
<td>8, 16, 32</td>
<td>out</td>
<td>no</td>
<td>Data lines to the Avalon bus module for write transfers. Not required if the master never performs write transfers. If used, write must also be used.</td>
</tr>
<tr>
<td>waitrequest</td>
<td>1</td>
<td>in</td>
<td>yes</td>
<td>Forces the master port to wait until the Avalon bus module is ready to proceed with the transfer.</td>
</tr>
<tr>
<td>irq</td>
<td>1</td>
<td>in</td>
<td>no</td>
<td>Interrupt request has been flagged by one or more slave ports.</td>
</tr>
<tr>
<td>irqnumber</td>
<td>6</td>
<td>in</td>
<td>no</td>
<td>The interrupt priority of the interrupting slave port. Lower value has higher priority.</td>
</tr>
<tr>
<td>endofpacket</td>
<td>1</td>
<td>in</td>
<td>no</td>
<td>Signal for streaming transfers. May be used to indicate an end of packet condition from the slave to the master port. Implementation is peripheral-specific.</td>
</tr>
<tr>
<td>readdatavalid</td>
<td>1</td>
<td>in</td>
<td>no</td>
<td>Signal for read transfers with latency and is for a master only. Indicates that valid data from a slave port is present on the readdata lines. Required if the master is latency-aware.</td>
</tr>
<tr>
<td>flush</td>
<td>1</td>
<td>out</td>
<td>no</td>
<td>Signal for read transfers with latency. Master can clear any pending latent read transfers by asserting flush.</td>
</tr>
</tbody>
</table>
Avalon (Master view)
Basic fundamental transfers
Avalon (Master view)
Read transfer, 0 wait
Avalon (Master view)
Read transfer, wait generated by slave/Avalon bus

Wait cycles
Avalon (Master view)
Write transfer, 0 wait
Avalon (Master view)
Write transfer, wait generated by slave

Wait cycles
Avalon (Master view)

**Read** transfers with latency, and **readdatavalid** generated by slave

![Diagram of Avalon bus protocol]

Flush : Kill previous Read data
Avalon (Master view)
Burst Write transfers

Address and BurstCount available for the whole transfer
Write can be deactivated by the master
The number of burstcount needs to be generated
Address and BurstCount available for the first cycle only
Read signal only for the first cycle
The number of burstcount ReadDataValid needs to be generated
The master could start a new transfer in 2
Bus avalon transfers resume

- Separate:
  - address, data in, data out
- Synchronous on clock’s rising edge
- Bus Internal or external wait request
- Transfers with latency available
- Multi-masters
- Arbitration at slave side
Avalon Address view

- 2 different views of addresses from master and slave, mode of decoding:
  - **Memory** (dynamic bus sizing)
  - **Register** (native transfers) ← deprecated

- Example:
  - Master 32 bits data
  - Slave 8 bits data

<table>
<thead>
<tr>
<th>Master addresses</th>
<th>31..24</th>
<th>23..16</th>
<th>15..8</th>
<th>7..0</th>
<th>Slave addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x….00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x….04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x….08</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x….0C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x….10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x….14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Bus seen on the slave side</th>
<th>7..0</th>
<th>Slave addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x..00</td>
<td></td>
<td>0x..00</td>
</tr>
<tr>
<td>0x..01</td>
<td></td>
<td>0x..01</td>
</tr>
<tr>
<td>0x..02</td>
<td></td>
<td>0x..02</td>
</tr>
<tr>
<td>0x..03</td>
<td></td>
<td>0x..03</td>
</tr>
<tr>
<td>0x..04</td>
<td></td>
<td>0x..04</td>
</tr>
<tr>
<td>0x..05</td>
<td></td>
<td>0x..05</td>
</tr>
</tbody>
</table>
Address view, Memory model

- **Memory model, dynamic bus sizing:**
  - No hole in the master address space
  - Need multiplexers on the data path
  - Master byte address = Slave byte address
  - 1 x 32 bits master transfer $\rightarrow$ 4 x 8 bits slave access by Avalon switch
- **BEx : ByteEnable x**

<table>
<thead>
<tr>
<th>Master addresses</th>
<th>BE3</th>
<th>BE2</th>
<th>BE1</th>
<th>BE0</th>
<th>Slave addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x....00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x....00</td>
</tr>
<tr>
<td>0x....04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x....04</td>
</tr>
<tr>
<td>0x....08</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x....08</td>
</tr>
<tr>
<td>0x....0C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x....0C</td>
</tr>
<tr>
<td>0x....10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x....10</td>
</tr>
<tr>
<td>0x....14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x....14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Bus seen on the slave side</th>
</tr>
</thead>
<tbody>
<tr>
<td>7..0</td>
</tr>
<tr>
<td>0x...00</td>
</tr>
<tr>
<td>0x...01</td>
</tr>
<tr>
<td>0x...03</td>
</tr>
<tr>
<td>0x...05</td>
</tr>
</tbody>
</table>
Memory model for Avalon memory slave

Avalon Master

Avalon memory Slave Interface

Clk

Address[31..0]

Write

WriteData[31..0]

Read

ReadData[31..0]

WaitRequest

Av_Add[5..0]

Av_CS

Av_Write

Av_WriteData[7..0]

Av_Read

Av_ReadData[7..0]

Avalon Bus Switch

RB-P2012
Address view, Register model (deprecated)

- **Register model, native transfer:**
  - Holes the master address space
  - NO multiplexers needed on the data path to align data
  - Master byte address ≠ Slave byte address
- Access by size of master bus (i.e. 32 bits), 8 bits available, highest bits undefined
- 1 master transfer = 1 slave transfer

![Data bus seen on the Avalon Master side and Data Bus seen on the slave side diagram]
Memory model for Avalon register slave

Avalon Master

Address[31..0]
WriteEnable[3..0]
Write[31..0]
Read[31..0]
WaitRequest

Avalon register Slave Interface

Av_Add[5..0]
Av_CS
Av_Write[7..0]
Av_Read[7..0]
Av_WriteData[7..0]
Av_ReadData[31..0]

A[7..2]
8 → 32 undefined extension

dec

Avalon Bus Switch
Architecture of EP1C12

- 12,000 logic Elements (LE)
- 52 x 4 Kbits RAM
- 2 x PLLs
- 180 IOs on 4 bancs
- Proprietary Configuration Bus
- JTAG Port

Quelques limites de fonctionnement

- multiplexor 16→1 : fmax LE = 275 MHz
- counter 64 bits : fmax LE = 160 MHz
- memory : fmax M4K = 220 MHz
- PLL : fmax PLL = 275 MHz
Logics Elements (LE)

Function Generator

Data[3..0] → Look-Up Table (LUT) → Carry Chain → Preset

Register Chain In

Clock
Enable

D Q
ENA

LUT Chain Row, Col, Local Routing
Register Chain Out Row, Col, Local Routing
Clear

Register (T,D,JK,SR)
Developments Tools from ALTERA

Quartus II → Hardware Description
- Schematic Editor, VHDL, ...
- Synthesis + placement routing
- Simulation (graphical éditor)
- Signal TAP

SOPC Builder → SOC NIOS II
- Configuration + SOC generation
- Programmable Interface library
- Own Programmable Interfaces
- Generation SDK

NIOS II IDE → NIOS II Code
- Project management
- Compiler + Link Editor
- Debugger
- SOC Programmer

Upgrade Your Web Edition Software to an Altera® Software Subscription
- Checking for informational updates regarding the Quartus® II Design Software. This may take about 1 minute.

Upgrade Benefits
- Full Stratix®, HardCopy® & APEX® 20KC Device Support
- LogicLock® Block-Based Design Flow
- SignalTap® II Logic Analyzer
- Chip Editor for incremental design changes

SOPC Builder
From Concept to System in Minutes

NIOS II IDE
INTEGRATED DEVELOPMENT ENVIRONMENT
Developments Tools from ALTERA

Quartus //

Working processus

Edition

Vérification

Synthèse

Téléchargement

Compilation

Contraintes

Autres

Status

Project Navigator

Edition

Messages

Console Script
Developments Tools from ALyER

SOPC Builder

- Processor: Nios II
- Bus Arbitration
- Components Library
- Memory Map
- Interrupts
- SOC

The diagram shows the SOPC Builder interface with various components and settings related to Nios II processor, bus arbitration, components library, memory map, and interrupts.
Developments Tools from ALTERA

NIOS II IDE (development)
Developments Tools from ALTERA

NIOS II IDE (debugger)
Conclusion

Some positives points of a softcore architecture

- Fast implementation
- Modular Architecture
- Simplicity
- Good documentation
- Nice for teaching complex integrated embedded systems
- Ease of development of our own programmable interface on internal bus (i.e. Avalon in VHDL, Verilog)
- Full system on FPGA, easily adaptable
- Operating System included (uC/OS II)

Some negate points

- Quite big tools to develop a system
- Thus tools to learn