Embedded Systems

"System On Programmable Chip"

Design Methodology using QuartusII and SOPC Builder tools

René Beuchat

LAP - EPFL

rene.beuchat@epfl.ch
Goals:

• to be able to design a programmable interface for an embedded system on a FPGA with Altera tools suite

  (note: a similar way is available for other FPGA manufacturer)

• to integrate it on an FPGA based embedded system

• finally to program the system in C
Altera Tools Suite

Quartus II

→ hardware description

Schematic Edition, VHDL, ...
Synthesis + place & route
Signal TAP
ModelSim

SOPC Builder

→ SOC NIOS II

Configuration + SOC generation
Peripherals Libraries (IP)
Own modules import
SDK Generation (software)

NIOS II IDE or SBT

→ Code NIOS II

Edition + projects management
Compiler + link editor
Debugger
SOC Programmer
Quartus II

Rules:
• for each programmable interface to design, we create a project in its own directory
• For the system design including the software, we create an other project

• NEVER use space and special characters in all the names (directory, files, project)
• Don't use "My Documents" (space)
Programmable Interfaces development

A separate project for each interface (recommended)

Project_1
Int.Prog. 1

Project...
Int.Prog. ...

Project_n
Int.Prog. n

Embedded system development

A separate project for the main design (recommended)

Project_NIOS_System
Full System

Quartus II:
Implementation VHDL/Schematic...
Compilation
Simulation

SOPC Builder:
New component creation

Quartus II:
Implementation Schematic...

SOPC Builder:
Design creation

Generate System

Quartus II:
End of system
Pins assignment (.tcl)
Quartus II/SOPC/NIOS IDE Full system Development

**Hardware System Compilation**

Project_NIOS_System
Full System

**Quartus II:**
Compilation

*Hardware debug → Signal Tap*

Compilation (again)

**SOPC Builder:**
Call NIOS IDE

**Embedded system Software development**

A separate Working Space for each System (recommended)

**NIOS IDE:**
Create a NIOSII Application
C Library compilation
Software Project design

**Project_NIOS_System (software)**
Quartus II/SOPC/NIOS IDE Full system Development

**Hardware/Software debug**

- **QuartusII: Signal Tap:** logic analyzer
- **NIOS IDE: Debugger:** software debug

**Hardware platform**
(ex: Cyclone robot)

- Connected through JTAG interface

**Download FPGA**
"hardware" file (.sof)

- **Compile and Debug**
  (Download code)
  Through JTAG interface

**Error:**

- Hardware: Modify design, simulate compile/generate, download
- Software: Modify C, compile/debug

Design the software application
Tools utilization

Design your Programmable Interface
Quartus II New project

Run QuartusII,

- **File → New project Wizard…**

- **Choice a directory name and project name (they could be the same)**

- **Family**: Cyclone

- **Device**: EP1C12Q240C8

→ for Cyclone Robot
Quartus II New project

Run QuartusII,

• File → New project Wizard…

• Choice a directory name and project name (they could be the same)

• **Family**: CycloneII

• **Device**: EP2C20F484C8

→ for FPGA4U.epfl.ch
Run **Quartus II**,

- **File → New project Wizard…**

- *Choice a directory name and project name (they could be the same)*

- **Family:** Cyclone IV E

- **Device:** EP4CE22F17C6

→ *for FPGA DE0 board*
Quartus II New file

Design of an entry file:

• **File → New ...**

• **Select an entry method**
  - VHDL
  - Block Diagram/Schematic File
  - .. Another
The file name is the name of the entity/architecture!!

Use the template to help in the VHDL language and structure

Don't forget the Library as:

- LIBRARY ieee;
- USE ieee.std_logic_1164.all;
- USE ieee.numeric_std.all;
  - Or (menthor or synopsys libraries) NO MORE!!
  - USE ieee.std_logic_arith.all;
  - USE ieee.std_logic_unsigned.all;
ENTITY Avalon_pwm IS
PORT
( 
   Clk : IN STD_LOGIC;
   nReset : IN STD_LOGIC;
   avs_Address : IN STD_LOGIC_VECTOR(2 downto 0);
   avs_CS : IN STD_LOGIC;
   avs_Read : IN STD_LOGIC;
   avs_Write : IN STD_LOGIC;
   avs_WriteData : IN STD_LOGIC_VECTOR(15 downto 0);
   avs_ReadData : OUT STD_LOGIC_VECTOR(15 downto 0);
   PWMa : OUT STD_LOGIC;
   PWMb : OUT STD_LOGIC
);
END Avalon_pwm;

Filename: Avalon_pwm.vhd
architecture comp of Avalon_pwm is

SIGNAL RegPeriod : unsigned (15 downto 0); -- Reg. Periode PWM
SIGNAL RegNewDuty : unsigned (15 downto 0); -- Register Duty
SIGNAL RegCommand : std_logic_vector (15 downto 0); -- Comm. Register
SIGNAL RegStatus : std_logic_vector (15 downto 0); -- Status Register
SIGNAL RegPreScaler : unsigned (15 downto 0); -- PreScaler value

SIGNAL CntPWM : unsigned (15 downto 0); -- Counter for PWM
SIGNAL CntPreScaler : unsigned (15 downto 0); -- Counter prescaler
SIGNAL PreClkEn : std_logic; -- Prescaler Clk En
SIGNAL PWMEn : std_logic; -- PWM enable

Begin
....

End comp;
VHDL a process example for a Prescaler

-- Prescaler process
-- PreClkEn generation: divide Clk by RegPreScaler value
-- PreClkEn = '1' for 1 clk cycle every RegPreScaler time

PrPreScaler:
process(Clk, Reset_n)
begin
  if Reset_n = '0' then
    CntPreScaler <= (others => '0');  -- Initialize @ 0
    PreClkEn <= '0';
  elsif rising_edge(clk) then
    if RegPreScaler = to_unsigned(0, 15) then  -- if ...=0 then ...
      PreClkEn <= '0';
    elsif (PWMEn = '1') then
      if CntPreScaler < RegPreScaler - 1 then
        CntPreScaler <= CntPreScaler + 1;
        PreClkEn <= '0';
      else
        CntPreScaler <= (others => '0');  -- Reset PreScaler Counter
        PreClkEn <= '1';  -- Activate for 1 clk cycle
      end if;
    end if;
  end if;
end process PrPreScaler;

| Clk          | _|''|_''|''|''|''|''|''|''|''|''|''|''|''|''|''|
|--------------|--------------------------
| PreClkEn     | ______/'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Quartus II Symbol creation / add

• Once the entity is define a symbol can be created to be used with Schematic design

• **File → Create/Update**
  ➢ **Create Symbol Files for Current File**

• **It can now be use in a Schematic**

• **File → New → Bloc Diagram/Schematic**

• **File → Save as… with the Schematic name**
Quartus II Schematic edition

- To add external access pin in a schematic:
  
  Select Input (Output or Bidir)

  For automatic insertion from a symbol:
  Right click on the symbol and:
  Generate pins for Symbol Ports
• The name for a bus in a schematic is:  
• *Bus_Name*[15..0]  
• With [ ] around bit field, MSb at left (15) LSb at right (0)
Quartus II Simulation with ModelSim

• Once the Programmable interface is designed, it has to be compiled:
  • *Processing ➔ Compiler tool* or
  • If no error are found ➔ go to simulation

• Call External Simulator ➔ ModelSim-Altera
Programmable Interface

• If the simulation is correct:
  ➢ The design of this programmable is finish for the Hardware part.
  ➢ Otherwise correct your VHDL and.. Compile/Simulate again!

• Good work!

• Now: The element can be added to a Library of components
Programmable Interface → Library

• In Windows, Create a directory (for example) "MyLibrary"
• Inside, create a directory (for example) "Avalon_PWM_MSE"
• Copy inside just the VHDL of the interface (for example) "Avalon_PWM.vhd"

Now we have to add this component in the Library of SOPC system
SOPC Builder Create System with NIOSII

Adding New component in SOPC Builder

And creating a NIOS II System
In this example, we want to create new components and implement a full system with the following elements:

- NIOS II, standard version (middle)
- (Serial)- JTAG
- Memory Flash- EPCS16
- SRAM, internal 16kBytes
- PIO, Input Output separated
- PWM (2x) → *need to create them in library before* !!
- ODO (2x) → *need to create them in library before* !!
QuartusII/SOPC Builder Embedded System creation

• Close the previous project and start a **New project**.
  i.e: "**RobotCyclone**" in a new directory

• Create New Bloc schematic ➔
  ➢ Add component (*left-click-click* in the schematic window or ⚙):
  ➢ **MegaWizard** ➔ Select: SOPC Builder

  ➢ VHDL to generate
  ➢ Path/Name: NIOSII_Cyclone
• Select "Create **New** component" or **New**

*Depend on the software version*
SOPC Builder Create component

- This operation is to tell to the Library manager the link between your programmable interface design and the Avalon maker.
- It has to know the function of all the defined signals.
The Component Editor is called

Select tab: HDL

and add

- MyLibrary\Avalon_PWM_MSE\Avalon_PWM.vhd
- Select Top Level Module
SOPC Builder Create component

- Go to the **Signals** tab
- In the column, for:
  - Name: **Clk, Reset_n**
  - Interface $\rightarrow$ **clock**
  - Signal Type $\rightarrow$ **clk, reset_n**

  - Name: **avs_...**
  - Interface $\rightarrow$ **avalon_slave_0**
  - Signal Type $\rightarrow$ **address, chipselect, ...**

  - Name: **PWMa, PWMb**
  - Interface $\rightarrow$ **export_0** (or new conduit Out)
  - Signal Type $\rightarrow$ **export**
SOPC Builder Create component

- Go to the **Interfaces** tab
- And make parameters for the interface:
  - Clock Name: clock
  - Conduit Output Name: export_0 (rename)
  - Avalon Slave Name: Avalon_slave_0
  - Slave addressing:
    - DYNAMIC (!! NATIVE no more supported !!)
  - Slave Timing:
    - Setup: 0 Hold: 0
    - Read Wait: 1, Write Wait: 0
SOPC Builder Create component

- Go to the **Component Wizard** tab
- And enter your parameters
- Change the Component version for each new version
- Select your own component Group

**Finish**

- Your module can be now included on a design in SOPC
IP to be added to available interfaces

- In SOPC Builder:
  - If the new component is not available, it is necessary to add the path to the directory where the component file is located.

- **Tools → Options**

- **File → Refresh Component List…**
Add the following elements:

- **Memories and…** → On Chip Memory
  → RAM: select **32 bits** and **16 kBytes**
- **Memories and…** → Flash → EPCS Flash...
- **Interface Protocol** → Serial → JTAG UART
- **PIO, separate Input/Output, 8 bits**
  - NIOSII processor → /s,
  - Reset: in EPCS,
  - exception vector: in on-chip memory
  - version Debug level2
- Add **2x your PWM**, rename them ..._L / ..._R
SOPC Builder system integration
SOPC Builder system integration

- If you have problems (error) with address or IRQ → System → Auto assign Base Address / IRQ

- **Generate** → and wait few minutes
- The full Avalon system is automatically build and will generate a VHDL file
- A copy of the library components used will be added to your project
- **Exit** when terminate
Quartus II System

- System with the NIOS
- Add Input/Output connectors
Quartus II System

- System with the NIOS + PLL

  ➢ To change the input Clk frequency of 24 MHz to higher, the PLL component is used.
Quartus II System → PLL

• Instantiate add a component (left-click-click) and select the "MegaWizard Plug-In Manager"

• Give a Name (ie: MyPLL)

• Select I/O → ALTPPLL
Quartus II System → PLL

- Input Clock: 24 MHz
- Output c0: 50 MHz
- No Lock

- Finish
- Add it to your design
Full design

We have to add the pin number: a script will do that!
QuartusII Pinning assignment (fpga4u)

- Copie the .tcl file from fpga4u.epfl.ch in your project directory:
  - [http://fpga4u.epfl.ch/images/b/be/Pin_assign_FPGA4U.tcl](http://fpga4u.epfl.ch/images/b/be/Pin_assign_FPGA4U.tcl)
  - → project directory
- Run the script Tools -> Tcl Scripts…
QuartusII Embedded System creation

Select the Pinassign_robot... file and → Run
Full design

Design with PLL, INPUT/OUTPUT and pin number
Name the element from the .tcl file name:

- PWM_RA, PWM_RB, PWM_LA, PWM_LB
- PortA[7..0]
- Clk, Reset_n
Compilation

• Compile your design:

• Processing → Compiler Tools → Start

• No error ??

• Congratulations the hardware is finish !!
• Call of NIOSII IDE:
  ➢ Start again SOPC Builder and left-click-click on the NIOSII system
  ➢ Select the NIOSII IDE and go to the software part design.
  ➢ Since version 10 → SBT (Software Build Tools)
NIOS IDE software design/debug

NIOS II IDE is the « old » tools
SBT (Software Build Tools) is the new version
The functions are basically the same.
Working space

• Specify a Working Space directory:
• Suggestion **Create a directory: WS**
  In your project directory
• Wait a short time and …
Select Workbench at the right top to start.
NIOSII IDE

- Ready for a software design
- *File → NIOSII C/C++ Application*
• Select "Hello World Small" for a template
• Change the name, ie: "Robot"
• A template project is created: Robot
• A library prepared (Robot_syslib(…))
  $\rightarrow$ right click on the library folder
  Select Build Project $\rightarrow$ the library is built
• Specifically the "system.h" file
• It contains hardware description parameters from SOPC
• In the "altera.components" the "io.h" file defines macro to access the hardware.
• In the "Robot" folder, the "hello_world_small.c" is a good starting point.

• Add those lines:
  - `#include "system.h"
  - `#include "io.h"`
The addresses found in the "system.h" are generated from the SOPC description:

- `#define AVALON_PWM_R_TYPE "Avalon_pwm_MSE"
- `#define AVALON_PWM_R_BASE 0x00000010`
- `#define AVALON_PWM_R_SPAN 16`
From io.h:

```c
#define IOWR_16DIRECT(BASE, OFFSET, DATA) \ 
  __builtin_sthio (__IO_CALC_ADDRESS_DYNAMIC ((BASE), (OFFSET)), (DATA))
```

To initialize the PWM_R:

IOWR_16DIRECT(AVALON_PWM_R_BASE, 0*2, 100); // Period
IOWR_16DIRECT(AVALON_PWM_R_BASE, 1*2, 40);   // Duty
IOWR_16DIRECT(AVALON_PWM_R_BASE, 4*2, 4);    // prescaler
IOWR_16DIRECT(AVALON_PWM_R_BASE, 3*2, 3);    // Pol=1, Enable

Base Address of PWM_R

Data to write

Reg Num * 2 bytes/16 bits
From io.h:

/* Dynamic bus access functions */

#define __IO_CALC_ADDRESS_DYNAMIC(BASE, OFFSET) \\
    ((void*)((alt_u8*)BASE) + (OFFSET))

#define IORD_32DIRECT(BASE, OFFSET) \\
    __builtin_ldwio (__IO_CALC_ADDRESS_DYNAMIC ((BASE), (OFFSET)))
#define IORD_16DIRECT(BASE, OFFSET) \\
    __builtin_ldhuio (__IO_CALC_ADDRESS_DYNAMIC ((BASE), (OFFSET)))
#define IORD_8DIRECT(BASE, OFFSET) \\
    __builtin_ldbuio (__IO_CALC_ADDRESS_DYNAMIC ((BASE), (OFFSET)))

#define IOWR_32DIRECT(BASE, OFFSET, DATA) \\
    __builtin_stwio (__IO_CALC_ADDRESS_DYNAMIC ((BASE), (OFFSET)), (DATA))
#define IOWR_16DIRECT(BASE, OFFSET, DATA) \\
    __builtin_sthio (__IO_CALC_ADDRESS_DYNAMIC ((BASE), (OFFSET)), (DATA))
#define IOWR_8DIRECT(BASE, OFFSET, DATA) \\
    __builtin_stbio (__IO_CALC_ADDRESS_DYNAMIC ((BASE), (OFFSET)), (DATA))
From io.h:

/* Dynamic bus access functions */
#define __IO_CALC_ADDRESS_DYNAMIC(BASE, OFFSET) \
    (((void *)(((alt_u8*)BASE) + (OFFSET)))
  ➢ Calculate byte address from Base (peripheral/memory) address and byte offset in this peripheral/memory
#define IORD_32DIRECT(BASE, OFFSET) \ 
    __builtin_ldwio (__IO_CALC_ADDRESS_DYNAMIC ((BASE), (OFFSET)))
  ➢ ldwio → load word (32 bits) i/o transfer
  ➢ ldhuio → load half-word (16 bits) unsigned i/o transfer
  ➢ ldbuio → load byte (8 bits) unsigned i/o transfer
  ➢ ld : load from per./mem. to processor
  ➢ st : store from processor to per./mem.
NIOSII IDE, io.h → Native access

From io.h:

/* Native bus access functions */
#define __IO_CALC_ADDRESS_NATIVE(BASE, REGNUM) \ 
    ((void *)((alt_u8*)BASE) + ((REGNUM) * (SYSTEM_BUS_WIDTH/8))))

#define IORD(BASE, REGNUM) \ 
    __builtin_ldwio (__IO_CALC_ADDRESS_NATIVE ((BASE), (REGNUM)))
#define IOWR(BASE, REGNUM, DATA) \ 
    __builtin_stwio (__IO_CALC_ADDRESS_NATIVE ((BASE), (REGNUM)), (DATA))

The accesses are only on 32 bits (SYSTEM_BUS_WIDTH)
- BASE is the (Byte) address of the selected device
- REGNUM is the offset address inside the selected device
- DATA is the value to transfer
Now you need to connect the robot through the JTAG interface

**Tools → QuartusII Programmer**

Select xxx.sof file to program

Install the Hardware for JTAG interface (ByteBlaster)

Start

The hardware part is downloaded
In the NIOS IDE:
Specify the hardware to use through the JTAG interface

- Select the library
- Run → Debug
- NIOSII hardware
In the NIOS IDE:
Specify the hardware to use through the JTAG interface

Now we are ready to debug
Memory Mapping on Avalon Bus

Memory Mapping Dynamic
- Memory Mapping Native
Memory Mapping on Avalon Bus

- What is the problem?
  - Master can be of different data bus sizes: ex: 16, 32, 64 bits
  - Slave can have different bus size: 8, 16, 32, 64, 128, 256, 512 or 1024 bits!!

- What represent a Master address
- What represent a Slave address in:
  - Dynamic model
  - Native model
Memory Mapping on Avalon Bus

• Rules:
  • Master provide Addresses in the range of 1..32 bits (i.e. 32 bits): A[31..0]
  • The **Master** address is a **BYTE** address
    ➢➔ if the address is incremented by 1, the next BYTE is selected by the master
    ➢➔ mode little-endian with NIOSII
  • The **BE[ ]**: Byte Enable signals specify the bytes to transfer on a word address
Master view of memory addresses (little-endian)

- Example of a 16 bits data in memories of different sizes, with the value 0x5678:
  - 0x78 at byte address 0x1000, and
  - 0x56 at byte address 0x1001

<table>
<thead>
<tr>
<th>BE[..]</th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Little-Endian</td>
<td></td>
<td>[0]</td>
<td>[1]</td>
<td>[2]</td>
</tr>
<tr>
<td>1000</td>
<td>78</td>
<td>56</td>
<td>1000</td>
<td>56</td>
</tr>
<tr>
<td>1001</td>
<td>56</td>
<td></td>
<td>1002</td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td></td>
<td></td>
<td>1004</td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td></td>
<td></td>
<td>1006</td>
<td></td>
</tr>
</tbody>
</table>

8 bits master | 16 bits master | 32 bits master
### Slave view of memory addresses (little-endian)

- The address provided by the Avalon bus to a slave is a **slave word address**
- Ex. Slave with 16 bytes space
- 16 bytes → 8 doublets → 4 quadlets → 2 octlets

<table>
<thead>
<tr>
<th>Master</th>
<th>Slave 8 bits 16 Bytes space</th>
<th>Slave 16 bits 8 doublet space</th>
<th>Slave 32 bits 4 quadlet space</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>BE[..]</th>
<th>Little-Endian</th>
<th>8 bits slave</th>
<th>16 bits slave</th>
<th>32 bits slave</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
</tr>
<tr>
<td>.00</td>
<td>.00</td>
<td>.00</td>
<td>.00</td>
<td>.00</td>
</tr>
<tr>
<td>.01</td>
<td>.01</td>
<td>.02</td>
<td>.02</td>
<td>.02</td>
</tr>
<tr>
<td>.02</td>
<td></td>
<td>.03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.03</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **BE[..]**: Big-Endian (BE) vs. Little-Endian (LE)
Avalon change of memory addresses (little-endian)

- The master view is independent of the slave view, the Avalon bus adapt the different cases
- For **processor** view (C/assembly programming) the addresses are **Bytes addresses**
- For **hardware** programmable interface the view is a **word address with selected Bytes Enable**
- Needed Multiplexers are provided by the Avalon bus and automatically generated by SOPC Builder

<table>
<thead>
<tr>
<th>Master</th>
<th>Slave 8 bits 16 Bytes space</th>
<th>Slave 16 bits 8 doublet space</th>
<th>Slave 32 bits 4 quadlet space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slave receive</td>
<td>→</td>
<td>Avs_A[3..0]</td>
<td>Avs_A[2..0]</td>
</tr>
</tbody>
</table>
Avalon change of memory addresses (little-endian)

- Ex: Master 32 bits, slave 16 bits:
  - Avm_A[0] → not connected

<table>
<thead>
<tr>
<th>Master</th>
<th>Slave 16 bits 8 doublet space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slave receive</td>
<td>Avs_A[2..0]</td>
</tr>
</tbody>
</table>
Avalon change of memory addresses (little-endian)

DYNAMIC Model

<table>
<thead>
<tr>
<th>Master 32 bits</th>
<th>Slave 16 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avm_ A[..0]</td>
<td>Avm_ A[..0]</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>..0000</td>
<td>..0000</td>
</tr>
<tr>
<td>..0100</td>
<td>..0010</td>
</tr>
<tr>
<td>..1000</td>
<td>..0100</td>
</tr>
<tr>
<td>..1100</td>
<td>..0110</td>
</tr>
<tr>
<td>..1000</td>
<td>..1000</td>
</tr>
<tr>
<td>..1010</td>
<td>..1010</td>
</tr>
<tr>
<td>..1100</td>
<td>..1100</td>
</tr>
<tr>
<td>..1110</td>
<td>..1110</td>
</tr>
</tbody>
</table>
# Avalon change of memory addresses (little-endian)

**NATIVE Model**

Bytes master offset 3 and 2 not available to 16 bits slaves!

<table>
<thead>
<tr>
<th>Master 32 bits</th>
<th>Slave 16 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avm_ A[..0]</td>
<td>Avm_ A[..0]</td>
</tr>
<tr>
<td></td>
<td>Avs_ A[..0]</td>
</tr>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>1100</td>
<td>1100</td>
</tr>
</tbody>
</table>
From io.h:

/* Dynamic bus access functions */

#define __IO_CALC_ADDRESS_DYNAMIC(BASE, OFFSET) \\
((void*)((alt_u8*)BASE) + (OFFSET))

#define IORD_32DIRECT(BASE, OFFSET) \\
  __builtin_ldwio (__IO_CALC_ADDRESS_DYNAMIC ((BASE), (OFFSET)))
#define IORD_16DIRECT(BASE, OFFSET) \\
  __builtin_ldhuio (__IO_CALC_ADDRESS_DYNAMIC ((BASE), (OFFSET)))
#define IORD_8DIRECT(BASE, OFFSET) \\
  __builtin_ldbuio (__IO_CALC_ADDRESS_DYNAMIC ((BASE), (OFFSET)))

#define IOWR_32DIRECT(BASE, OFFSET, DATA) \\
  __builtin_stwio (__IO_CALC_ADDRESS_DYNAMIC ((BASE), (OFFSET)), (DATA))
#define IOWR_16DIRECT(BASE, OFFSET, DATA) \\
  __builtin_sthio (__IO_CALC_ADDRESS_DYNAMIC ((BASE), (OFFSET)), (DATA))
#define IOWR_8DIRECT(BASE, OFFSET, DATA) \\
  __builtin_stblio (__IO_CALC_ADDRESS_DYNAMIC ((BASE), (OFFSET)), (DATA))
NIOSII IDE, io.h ➔ Dynamic access

From io.h:

/* Dynamic bus access functions */
#define __IO_CALC_ADDRESS_DYNAMIC(BASE, OFFSET) 
  ((void *)(((alt_u8*)BASE) + (OFFSET)))

  ➢ Calculate byte address from Base (peripheral/memory) address and byte offset in this peripheral/memory
#define IORD_32DIRECT(BASE, OFFSET) 
  __builtin_ldwio (__IO_CALC_ADDRESS_DYNAMIC ((BASE), (OFFSET)))

➢ ldwio ➔ load word (32 bits) i/o transfer
➢ ldhuio ➔ load half-word (16 bits) unsigned i/o transfer
➢ ldbuio ➔ load byte (8 bits) unsigned i/o transfer

➢ ld : load from per./mem. to processor
➢ st : store from processor to per./mem.
From io.h:

/* Native bus access functions */

#define __IO_CALC_ADDRESS_NATIVE(BASE, REGNUM) \ 
  ((void *)(((alt_u8*)BASE) + ((REGNUM) * (SYSTEM_BUS_WIDTH/8))))

#define IORD(BASE, REGNUM) \ 
  __builtin_ldwio (__IO_CALC_ADDRESS_NATIVE ((BASE), (REGNUM)))
#define IOWR(BASE, REGNUM, DATA) \ 
  __builtin_stwio (__IO_CALC_ADDRESS_NATIVE ((BASE), (REGNUM)), (DATA))

The accesses are only on 32 bits (SYSTEM_BUS_WIDTH) from master
- BASE is the (Byte) address of the selected device
- REGNUM is the offset address inside the selected device
- DATA is the value to transfer

- The slaves are mapped to SYSTEM_BUS_WIDTH