The **Microelectronics Training Center**

The MTC is an initiative within the INVOMEC division

Industrialization & Training in Microelectronics

Lab-exercise

Lab 4:

Build and simulate a complete system including a Uart

Cluster: Cluster1
Module: Module6a

Target group: Students

Version: 1.1
Date: 13/02/07
Author: Geert Vanwijnsberghe

This material was developed with support of the European Social Fund.
ESF: Prevent and combat unemployment by promoting employability, entrepreneurship, adaptability and equal opportunities between women and men, and by investment in people.
http://www.esf-agentschap.be

Introduction

Before implementing our system in the FPGA on the XUP board we will add an interface to a uart. This will allow us to set up a serial communication between the PC and our microprocessor. To demonstrate the correct behavior of the complete system we will write a small assembler program that stores the characters being sent by the PC until a character 0 is received. After this the characters are sent back by the microprocessor to the PC in a reverse order.

We will not reinvent the uart but we will use a miniUart core from www.OpenCores.Org.
This core was however slightly modified in order to support a baud rate of 115200. To make the assembly code very simple an extra interface unit between the cpu and the miniUart has been developed allowing us to send bytes and receive bytes from the miniUart without checking its status flags first.
The VHDL memory model we used up to now is completely technology independent. In this module we will generate a new memory by the core generator (COREGEN) of Xilinx that maps our memory on the available block ram in the VirtexII-pro FPGA.
Before implementing this complete system we will simulate and verify this.

Objectives

After completing this module, you should be able to:

- include existing IPs in a system
- use the CoreGen of Xilinx
- write a small assembly program
- simulate a complete system

Knowledge background

- Basic VHDL knowledge
- VAS assembler

Classification

Level:
Duration:

Input

Folders

- assembler : VAS vhdl files
- system_vhdl : all vhdl files of the system
- simulation : folder where you have to startup the simulation
  - o compile.do :
- testbench_vhdl :
  - o uart_fake.vhd
  - o TB_uart_fake.vhd
  - o TB_system.vhd

The lab

The next figure shows the complete system that is built in this module.

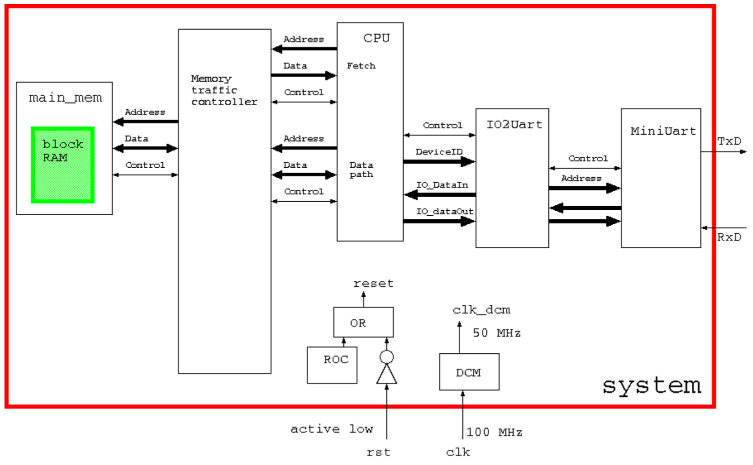


Figure 1

MiniUart

Have a look at the vhdl code of the MiniUart. You see that it consists of a baud rate generator, an Rxunit and a Txunit. The baud rate generator has been modified in order to have a baud rate of 115200 when the clock is driven with 50 MHz. The specification of the MiniUart is:

- 2 bit address for reading :
    - 00 : put received byte on DataOut
    - 01 : put status info on DataOut
- status info
    - bit 7-4 : unused
    - bit 3 : Transmit buffer empty
    - bit 2 : Data available from the receiver
    - bit 1 : Frame error in received data
    - bit 0 : Output error
- synchronous wrt rising edge of SysClk
- asserting CS_N and RD_N will capture the received data and reset the Data available flag.
- asserting CS_N and WR_N will start the transmission of the byte on DataIn if the transmit buffer is empty.

IO2Uart

This is an extra module in hardware that could also have been implemented in software. The main functionality of this unit is that the microprocessor can read or write a byte to it without checking the internal flags of the miniUart. This unit will wait until the Data available flag is asserted before reading the received byte and it will wait until the Transmit buffer empty flag is set before writing a new byte to the miniUart. You should have a look at the code and you will see that is consists of a simple FSM. Since only 1 device is attached to the CPU the DeviceID is not used.

Glue logic

As you can see in figure 1 also two small units were added to the system. The ROC (reset on configuration) is a module that generates a reset pulse after the FPGA has been configured. This means that the system will automatically be reset and the external reset is actually only needed to reset the system during operation.
The DCM (Digital clock manager) is a unit that can divide and/or multiply its clock input. This instantiation generates a 50MHz clock out of the 100 MHz clock.

Assembler code

The next code does the string inversion. Characters (bytes) are read from device 0 (D0 = IO2Uart) and pushed on the stack until character "0" is received. After this the characters are popped from the stack again and send to device 0.

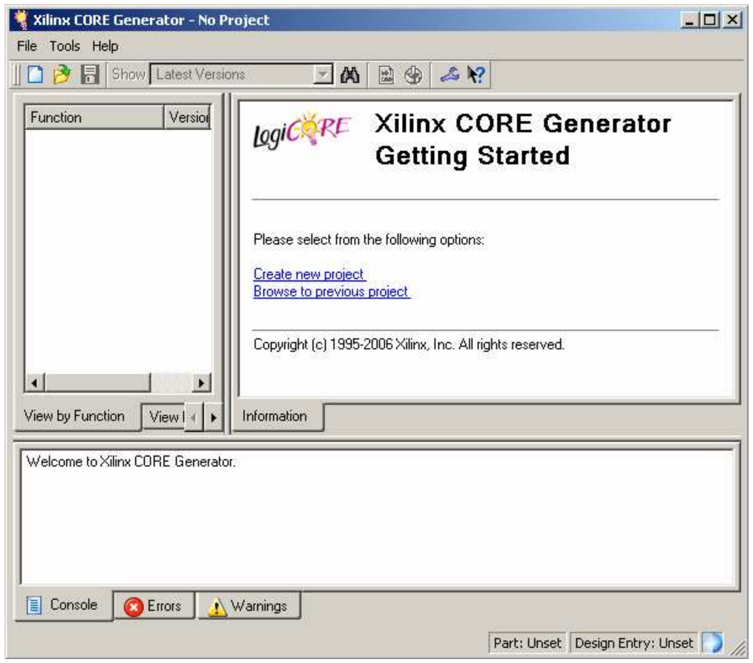
```
.CHAR_0 #48;

LDM CHAR_0 R1;

$READCHAR:
INB D0 R0;
CMP R0 R1;
BEQ WRITEBACK;
PSH R0;
INC R2;
BRA READCHAR;

$WRITEBACK:
POP R0;
OUB D0 R0;
DEC R2;
BGT WRITEBACK;
END;
```

Use module_5c to translate this assembler code into a ram.coe file (Memory Coefficients file) This file is used by the Xilinx CoreGen to generate the necessary Block RAM modules and initialize their contents.

ⓘ Note: This program does not use any data (data.asm). Therefore you should run the simulation with the generic UseTestData set to false
Modelsim> vsim -Gusetestdata=false work.assembler

Generate the block ram

Start the Xilinx CORE Generator.



Xilinx CORE Generator Getting Started

Please select from the following options:

Create new project
Browse to previous project

Copyright (c) 1995-2006 Xilinx, Inc. All rights reserved.

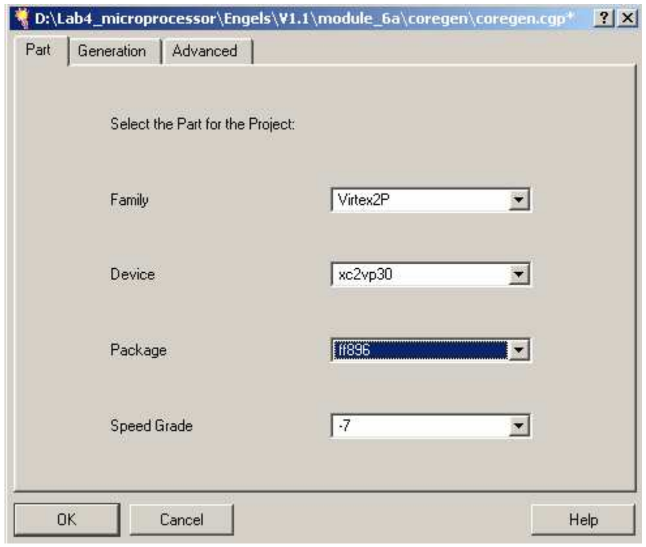
Welcome to Xilinx CORE Generator.

Part: Unset Design Entry: Unset

1. Select : Create new project
   Specify a new non existing folder and a project name (eg. coregen)

ⓘ note : specify a folder "coregen" in your module_6a folder next to the folders assembler,system_vhdl,testbench_vhd and simulation.

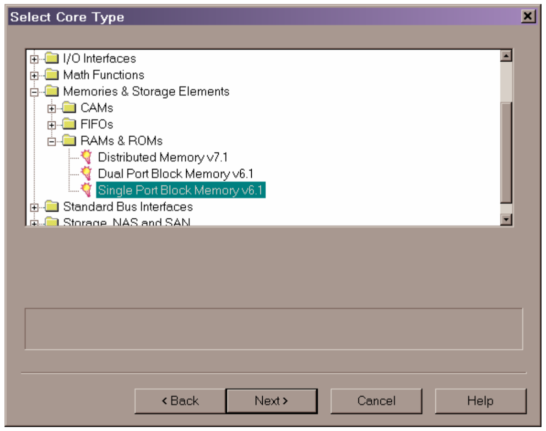
2. Specify Family, Device and Package


D:\Lab4_microprocessor\Engels\V1.1\module_6a\coregen\coregen.cgp*
Part Generation Advanced
OK Cancel Help

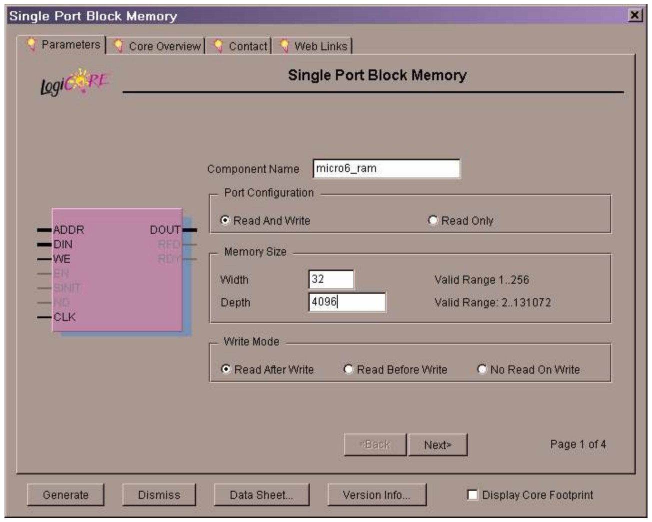
Select the Part for the Project:

Family Virtex2P
Device xc2vp30
Package ff896
Speed Grade -7

3. In the Select Core Type window, select **Memories & Storage Elements → RAMs & ROMs → Single Port Block Memory v6.2** and double click.



4. The **Single Port Block Memory** window is displayed. In the **Memory Size** field, change **Width** to 32 and **Depth** to 4096. Click **Next** after doing this.



Single Port Block Memory

Component Name: micro6_ram

Port Configuration: [✓ Read And Write] Read Only

Memory Size

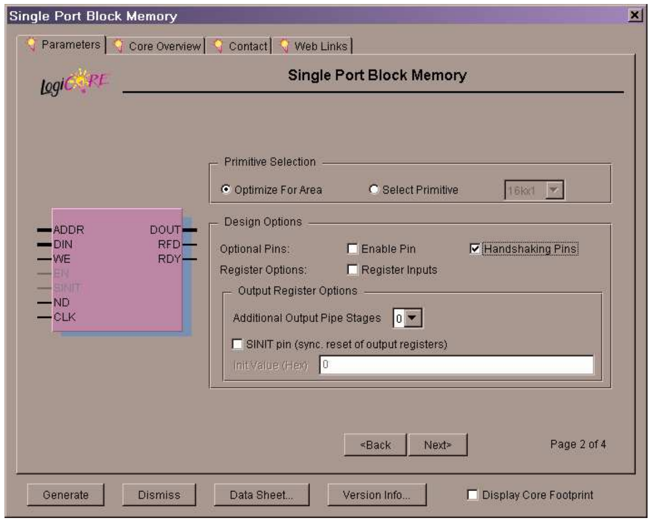
Width	32	Valid Range 1..256
Depth	4096	Valid Range: 2..131072

Write Mode: [✓ Read After Write] Read Before Write No Read On Write

Page 1 of 4

☐ Display Core Footprint

5. In the next window, check **Handshaking Pins** in the **Design Options** field. Click **Next**.



Single Port Block Memory

Primitive Selection: [✓ Optimize For Area] Select Primitive 16kx1

Design Options

Optional Pins: ☐ Enable Pin ☑ Handshaking Pins

Register Options: ☐ Register Inputs

Output Register Options

Additional Output Pipe Stages: 0

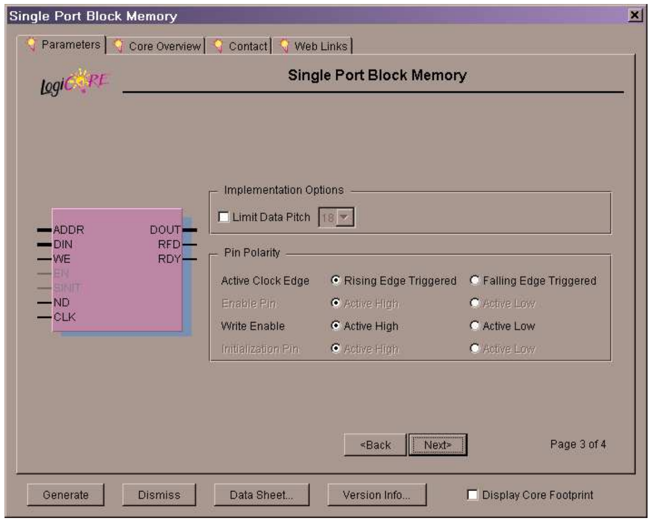
☐ SINIT pin (sync. reset of output registers)

Init Value (Hex): 0

Page 2 of 4

☐ Display Core Footprint

6. Leave the options in the next window unchanged. Proceed to the next window by clicking **Next**.



Single Port Block Memory

Implementation Options: ☐ Limit Data Pitch 18

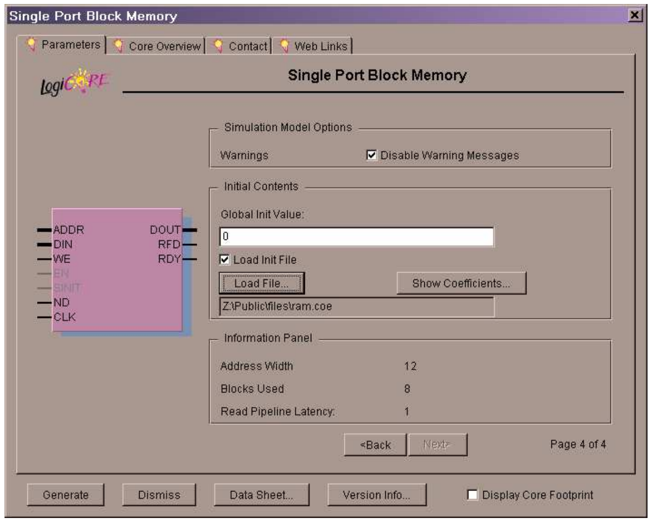
Pin Polarity

Active Clock Edge	[✓ Rising Edge Triggered]	Falling Edge Triggered
Enable Pin	[✓ Active High]	Active Low
Write Enable	[✓ Active High]	Active Low
Initialization Pin	[✓ Active High]	Active Low

Page 3 of 4

☐ Display Core Footprint

7. In this window, check **Load Init File** in the **Initial Contents** field. Cick the **Load File** button and browse to the lab directory. Choose `ram.coe(=output of VAS assembler)` and click **Open**. The file name and path are displayed in black font. In case of invalid coefficient files, the name and path are red.



Single Port Block Memory

Simulation Model Options

Warnings: ☑ Disable Warning Messages

Initial Contents

Global Init Value: 0

☑ Load Init File

Z:\Public\files\ram.coe

Information Panel

Address Width	12
Blocks Used	8
Read Pipeline Latency:	1

Page 4 of 4

☐ Display Core Footprint

8. Click **Generate**. *CoreGen* starts generating the Block RAM. When finished, an info message like the one shown below is displayed.

```
Successfully generated micro6_ram.
```

9. Close the CORE Generator

The following files were created:

micro6_ram.edn:
Electronic Data Netlist (EDN) file containing the information required to implement the module in a Xilinx FPGA.

micro6_ram.mif:
Memory Initialization File which is automatically generated by the CORE Generator System for some modules when a simulation flow is specified. A MIF data file is used to support HDL functional simulation of modules which use arrays of values.

micro6_ram.vhd:
VHDL wrapper file provided to support functional simulation. This file contains simulation model customization data that is passed to a parameterized simulation model for the core.

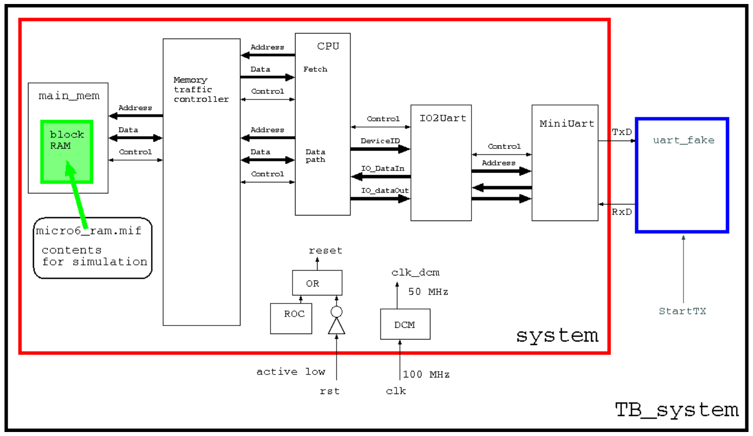
micro6_ram.vho:
VHO template file containing code that can be used as a model for instantiating a CORE Generator module in a VHDL design.

micro6_ram.xco:
CORE Generator input file containing the parameters used to regenerate a core.

Simulate the complete system

In order to simulate the compete system in an easy way a “uart_fake” was developed. The VHDL entity of this model has 3 generics : Baudrate, IdleBits and TXstring. It also has 3 ports RX input, TX output and a StartTX input. When StartTX changes from 0 to 1 the TXstring will be sent out of the TX port with the specified Baudrate. Between the characters a number of extra stopbits (IdleBits) will be sent. The RX input is continuously monitored and the received characters are sent to the log window of the simulator.

Have a look at the VHDL description of the testbench (TB_system.vhd).



Before starting the simulation you should copy micro6_ram.mif from the coregen folder to your simulation folder. An alternative is to modify micro6_ram.vhd in the coregen folder.

Start modelsim in your simulation folder.

```
Modelsim> do compile.do
Modelsim> vsim work.tb_system
Modelsim> run 6 ms
```

You should get the following output in your transcript window:

```
………..
# Char received in hex = 20 --
# ** Note:  transmit of fake uart done
#    Time: 3864400 ns  Iteration: 0  Instance: /tb_system/u2
# Char received in hex = 6B -- k
# Char received in hex = 6A -- j
# Char received in hex = 69 -- i
# Char received in hex = 6C -- l
# Char received in hex = 65 -- e
# Char received in hex = 64 -- d
# Char received in hex = 6E -- n
# Char received in hex = 69 -- i
# Char received in hex = 45 -- E
```

Now you are ready for the next module where you will put this system into the FPGA on the XUP board.