

FPGA Synthesis

1. Introduction

In order to design digital systems using programmable logic devices such as FPGAs, computer aided design (CAD) software products are used. These software solutions offer the whole package, from the early designing stage to the final FPGA implementation stage. Thus, 5 steps can be distinguished:

- Description: VHDL source code
- Synthesis: compile VHDL code, transform description into a netlist (basic logic gates and interconnections)
- Functional simulation: simulation based on test benches and waveforms (usually from the IDE, such as Xilinx ISE Simulator, ModelSimetc.)
- Implementation: adapting the netlist to the available resources of the device (technology mapping, place and route)
- Configuration: programming the device

Each stage mentioned above is important in order to design a fully working digital system, on a FPGA development board. This laboratory work will focus on the last 2 steps because the first 3 steps are already known from the previous lab. The purpose of this laboratory work is to perform the programming of a FPGA device.

2. FPGA synthesis

This section presents a couple of design examples that are based on the VHDL language and uses the Xilinx ISE environment for implementation.

Applications

Real-time clock

This is a design that implements a real-time clock maintaining time in hours, minutes and seconds, with the capability to set the time. The design is targeted for the Nexys 2 development board (Xilinx Spartan-3E FPGA, 500k gates) or for the Nexys 3 development board (Xilinx Spartan-6 FPGA). For this example, the system clock is generated by the 50 MHz on-board oscillator (100 MHz for the Nexys 3 development board).

The input signals are the following:

- *clk*: system clock with a frequency of 50 or 100 MHz
- *reset*: reset signal, used for initializing and resetting the time to 00:00
- *hour*: displays how many hours have passed.

Output signals:

- *segments(7:0)*: data bus for the 7 segments display
- *anode(3:0)*: anode 4-bit bus for the 7 segments display

The design consists of a single functional block.

Below are the steps necessary to implement and run the real-time clock on the FPGA board.

1. In the *Project Navigator* window, create a *New Project*. Choose HDL source type and enter any desired name.
2. In the following window, choose Spartan-3E as the development board, with the device as **XC3S500E**, package **FG320** and speed **-4**. For the Spartan-6 development board, use Xilinx ISE 14.4, with the device set as **XC6SLX16**, package **CS324**, speed **-2**. Leave the other options unchanged and finish the creation of the project.
3. Extract the file from *clock2.zip*. In the *Hierarchy* pane, *Project->Add Copy of Source* and select the .vhd file from the unzipped folder.
4. Select the top module **counter (clock.vhd)** and run *Synthesize – XST* from the *Process/design* pane.
5. (optional) Modify the maximum *fanout*. Open the *Synthesis Report* from the *Synthesize - XST* tab and find the maximum fanout (and the corresponding resource). Modify the default fanout value of **500** to **50** in the *Process Properties* of the *Synthesize* tab (right-click the tab to access it). The fanout value can be changed in the *Xilinx Specific Options* tab. Re-run *Synthesize – XST* and check again the maximum fanout value from the *Synthesis Report*.

Explanation: large fanouts can cause difficulties during the routing phase. The Xilinx environment allows limiting the fanout by duplicating gates with large fanouts or by inserting buffers. Note that by reducing the maximum fanout value, the time needed to do a complete *Synthesize* operation will be lower.

6. (optional) In the *Process/design* pane, double-click the *View RTL Schematic* process. This will open a new window which illustrates the schematic representation of the synthesis result.
7. Select the top-level **clock** module from the *Hierarchy* pane. Expand *User Constraint* from the *Process/design* pane and double-click the *Floorplan Area / IO / Logic – Post Synthesis (I/O Pin Planning (PlanAhead) – Post-Synthesis* in Xilinx 14.4).
8. In the *I/O Ports* panel, expand the **segments** entry and select **segments[0]**:

- a. In the *Site* column of this port, select **M13**
 - b. *I/O Std* column, select **LVC MOS33**
 - c. Keep default values for the other options
9. Repeat the previous step to assign constraints to the remaining I/O ports, according to the table below (only for Nexys 3 – Spartan 6 development board). Note that the values in the **Site** column are different for the Nexys 2 development boards.

Port	Site	I/O Standard
clk	V10	LVC MOS33
reset	D9	LVC MOS33
hour	C4	LVC MOS33
anode[0]	N16	LVC MOS33
anode[1]	N15	LVC MOS33
anode[2]	P18	LVC MOS33
anode[3]	P17	LVC MOS33
segments[7]	T17	LVC MOS33
segments[6]	T18	LVC MOS33
segments[5]	U17	LVC MOS33
segments[4]	U18	LVC MOS33
segments[3]	M14	LVC MOS33
segments[2]	N14	LVC MOS33
segments[1]	L14	LVC MOS33
segments[0]	M13	LVC MOS33

10. Save the I/O port constraints and exit the PlanAhead software.
11. (optional) The new .ucf file is added to the project. Select this file in the *Hierarchy* pane and in the *Process/design* pane, double-click the *Edit Constraints (Text)* line. Review the file to check if the constraints have been set correctly.
12. In the *Process/design* pane, right-click the *Generate Programming File* process, select the *Startup Options* category and change the *CLK* option to **JTAG Clock**. Click OK to finish.
13. Run the *Generate Programming File* process.
14. Connect the development board (Spartan-3E or Spartan-6) to a USB port of the computer, through the provided USB cable. Make sure the **SELECT** pin is set on USB and move the power switch near the power connector to the ON position.
15. In the *Process/design* pane, expand *Configure Target Device* and double-click the *Manage Configuration Project (iMPACT)*.
16. In the *ISE iMPACT* window, create a new project. In the new dialog box, the *Configure devices using Boundary-Scan (JTAG)* option should be already selected. Ensure that the *Automatically connect to a cable and identify Boundary-Scan chain* option is selected.

17. If the *Auto Assign Configuration Files Query Dialog* window appears, select the **Yes** button to continue assigning the configuration file. The *Assign New Configuration File* dialog window opens. Browse to the project folder, select the **.bit** file (counter.bit), and click the *Open* button.
18. In the *Attach SPI or BPI PROM* dialog box, select the **NO** button.
19. The *Device Programming Properties – Device 1 Programming Properties* dialog window will open, with the *FPGA Device Specific Programming Properties* property automatically selected. Select the **OK** button.
20. In the device chain area, right-click on the device (xc3s500e or xc6slx16) and select *Program*. If programming succeeds, the clock should start counting from 00:00.
21. There are 2 buttons below the display which are programmed as follows:
 - a. Left button (BTNL) shows how many hours have passed
 - b. Right button (BTNR) resets the clock to 00:00.

FIFO Memory

The goal of this application is to implement a FIFO memory on a Nexys 2 or Nexys 3 development board.

Figure 1 below illustrates the block diagram of the FIFO memory. The capacity is 8 words of 8 bits each.

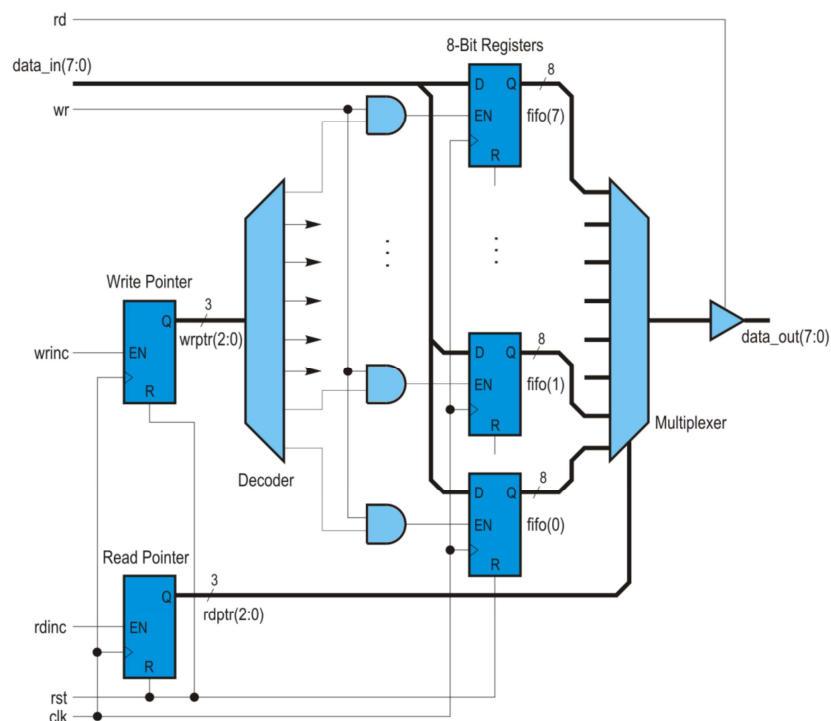


Figure 1. Block design of the FIFO memory

When the read signal *rd* is asserted, the output *data_out* of the memory must be enabled. When the read signal is not asserted, the output must be placed in the high-impedance state. When the write signal *wr* is asserted, the value from the *data_in* input of the memory must be written into one of the 8 registers. The read and write pointers indicate which register to read and which register to write. To increment the read pointer, the *rdinc* signal is used, and to increment the write pointer, the *wrinc* signal is used.

Create a new project in Xilinx 14.4, input a name and select the Spartan-3E or Spartan-6 board. Set the following settings:

Nexys 2 development board:

- Device **XC3S500E**
- Package **FG320**
- Speed **-4**

Nexys 3 development board:

- Device **XC6SLX16**
- Package **CS324**
- Speed **-2**

Create a new VHDL module for the FIFO memory. Specify the inputs and outputs according to Figure 1. Write a process for each of the following elements of the memory:

- Read pointer
- Write pointer
- Decoder
- Register set
- Multiplexer
- Tri-state buffer

The processes will communicate via internal signals, which should be defined in the declarative part of the architecture.

The design also needs a control module/unit. The block design of such a unit is visible in Figure 2.

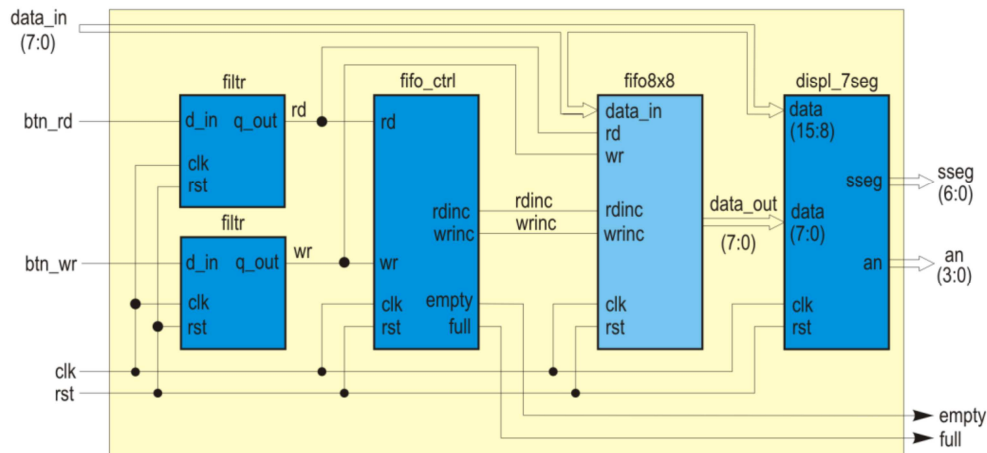


Figure 2. Block design of the control unit for a 8x8 FIFO memory

The control unit has as main inputs the read and write signals of the memory (*rd* and *wr*), which generate the *rdinc* and *wrinc* signals for incrementing the read and write pointers from the FIFO memory module. Also, the control unit will generate the *empty* and *full* status signals (optional):

- *Empty* signal is set to logical 1 when no words have been written in the FIFO memory or all words have been read
- *Full* signal is set to logical 1 when the FIFO memory contains 8 words that have not been read.

The *filtr* block from Figure 2 is a debouncing module for the buttons used to generate the *rd* and *wr* signals. When one of these buttons is pressed, the corresponding pin of the FPGA device will be connected to logical 1. This module will be instantiated 2 times in the top-level module and will generate a single pulse when the corresponding button is pressed.

The debouncing module is created as follows:

- Select *Edit-> Language Templates*, expand *Synthesis Constructs->Coding Examples*, then *Misc* and select the template called *Debounce circuit*.
- Insert the signal declaration from the template before the **begin** keyword of the *filtr* module, and the other lines of the template after the **begin** keyword of the same module.
- Replace the strings <clock> and <reset> with *clk* and *rst*, then save the file.

The block *displ_7seg* from Figure 2 is the 7 segments display module, which you can find in the archive *fifo.zip*.

Create a new VHDL module for a decoder for the 7 segments display and name it *hex2sseg*. This module has as input a 4-bit hex code and outputs the 7-bit vector to drive the segments of

the display (*sseg*). In this file, after the **begin** keyword, select *Edit->Language Templates->Synthesis Constructs-> Coding Examples-> Misc*, right-click on the template *7-Segment Display Hex Conversion* and select *Use in File*. Save the *hex2sseg.vhd* file.

In order to implement the FIFO memory on a development board, we must first define the peripherals of the board which will be used:

- Input data will be entered from the SW7-SW0 slide switches
- The status of the slide switches will be displayed on the 2 left-hand side digits of the 7 segments display
- The byte read from the memory will be displayed on the 2 right-hand side digits of the 7 segments display
- *rst* signal assigned to button left (BTNL)
- *rd* signal assigned to button up (BTNU)
- *wr* signal assigned to button down (BTND)
- *full* status signal will be displayed on LD0 Led
- *empty* status signal will be displayed on LD1 Led

Perform the following steps to implement and run the design on the Spartan-3E or Spartan-6 development board:

1. Select the top-module from the *Hierarchy* pane and the run the *Synthesize* process from the *Process/design* pane for the whole design. Correct any errors that may appear.
2. Expand *User Constraints* and run *I/O Pin Planning (PlanAhead) – Post-Synthesis*. Assign to the ports the corresponding sites. Use the table below as a reference for a Spartan-6 development board.

Port	Site	I/O Standard
clk	V10	LVC MOS33
rst	C4	LVC MOS33
btn_rd	A8	LVC MOS33
btn_wr	C9	LVC MOS33
empty	U16	LVC MOS33
full	V16	LVC MOS33
an[0]	N16	LVC MOS33
an[1]	N15	LVC MOS33
an[2]	P18	LVC MOS33
an[3]	P17	LVC MOS33
sseg [6]	T17	LVC MOS33
sseg [5]	T18	LVC MOS33
sseg [4]	U17	LVC MOS33
sseg [3]	U18	LVC MOS33

sseg [2]	M14	LVC MOS33
sseg [1]	N14	LVC MOS33
sseg [0]	L14	LVC MOS33
data_in[7]	T5	LVC MOS33
data_in[6]	V8	LVC MOS33
data_in[5]	U8	LVC MOS33
data_in[4]	N8	LVC MOS33
data_in[3]	M8	LVC MOS33
data_in[2]	V9	LVC MOS33
data_in[1]	T9	LVC MOS33
data_in[0]	T10	LVC MOS33

3. In the *Process/design pane*, right-click the *Generate Programming File* process, select the *Startup Options* category and change the *CLK* option to **JTAG Clock**. Click **OK** to finish.
4. Run the *Generate Programming File* process.
5. Connect the development board (Spartan-3E or Spartan-6) to a USB port of the computer, through the provided USB cable. Make sure the **SELECT** pin is set on USB and move the power switch near the power connector to the **ON** position.
6. In the *Process/design pane*, expand *Configure Target Device* and double-click the *Manage Configuration Project (iMPACT)*.
7. In the *ISE iMPACT* window, create a new project. In the new dialog box, the *Configure devices using Boundary-Scan (JTAG)* option should be already selected. Ensure that the *Automatically connect to a cable and identify Boundary-Scan chain* option is selected.
8. If the *Auto Assign Configuration Files Query Dialog* window appears, select the **Yes** button to continue assigning the configuration file. The *Assign New Configuration File* dialog window opens. Browse to the project folder, select the **.bit** file, and click the *Open* button.
9. In the *Attach SPI or BPI PROM* dialog box, select the **NO** button.
10. The *Device Programming Properties – Device 1 Programming Properties* dialog window will open, with the *FPGA Device Specific Programming Properties* property automatically selected. Select the **OK** button.
11. In the device chain area, right-click on the device (xc3s500e or xc6slx16) and select *Program*.
12. Verify the operation of the FIFO memory on the development board.