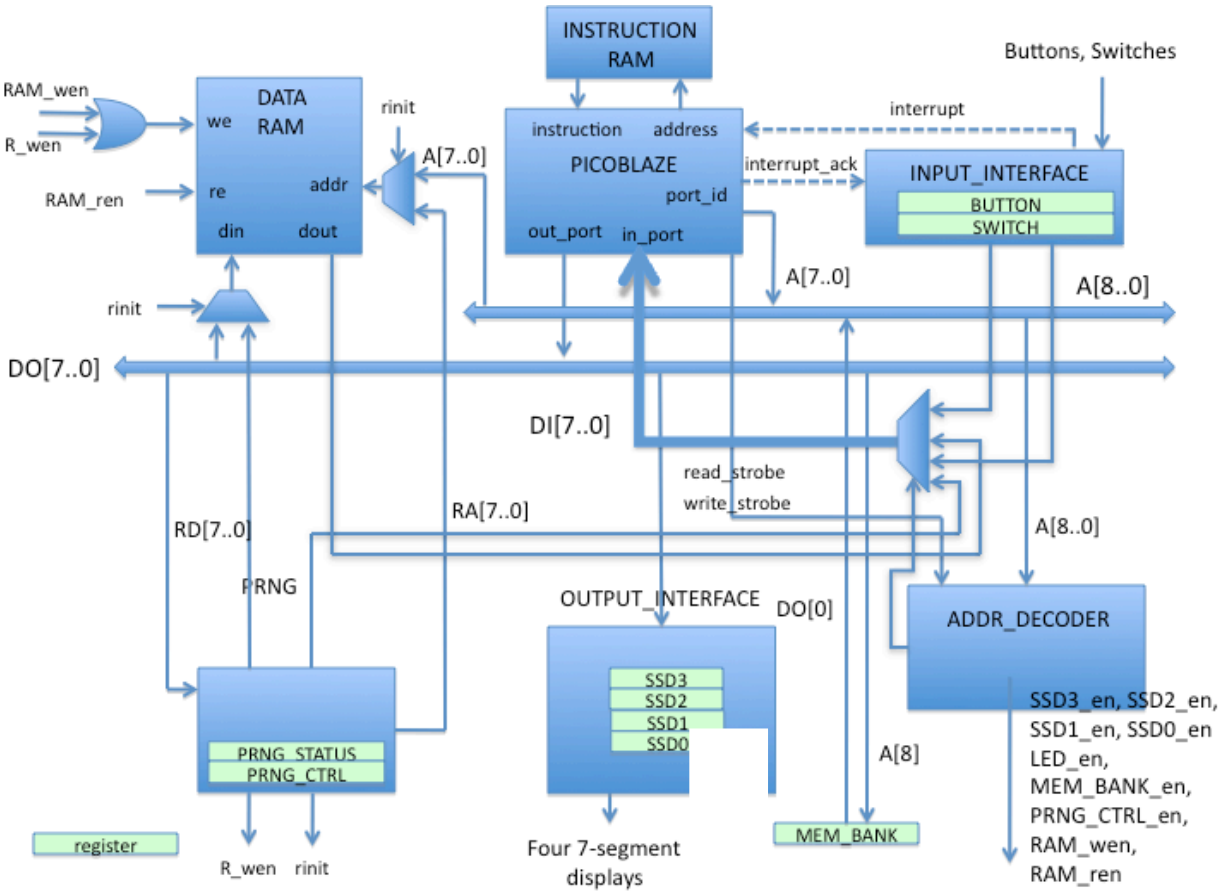
Experiment 7

Design, implement, and verify experimentally a circuit, shown in the block diagram below, composed of the following major components:

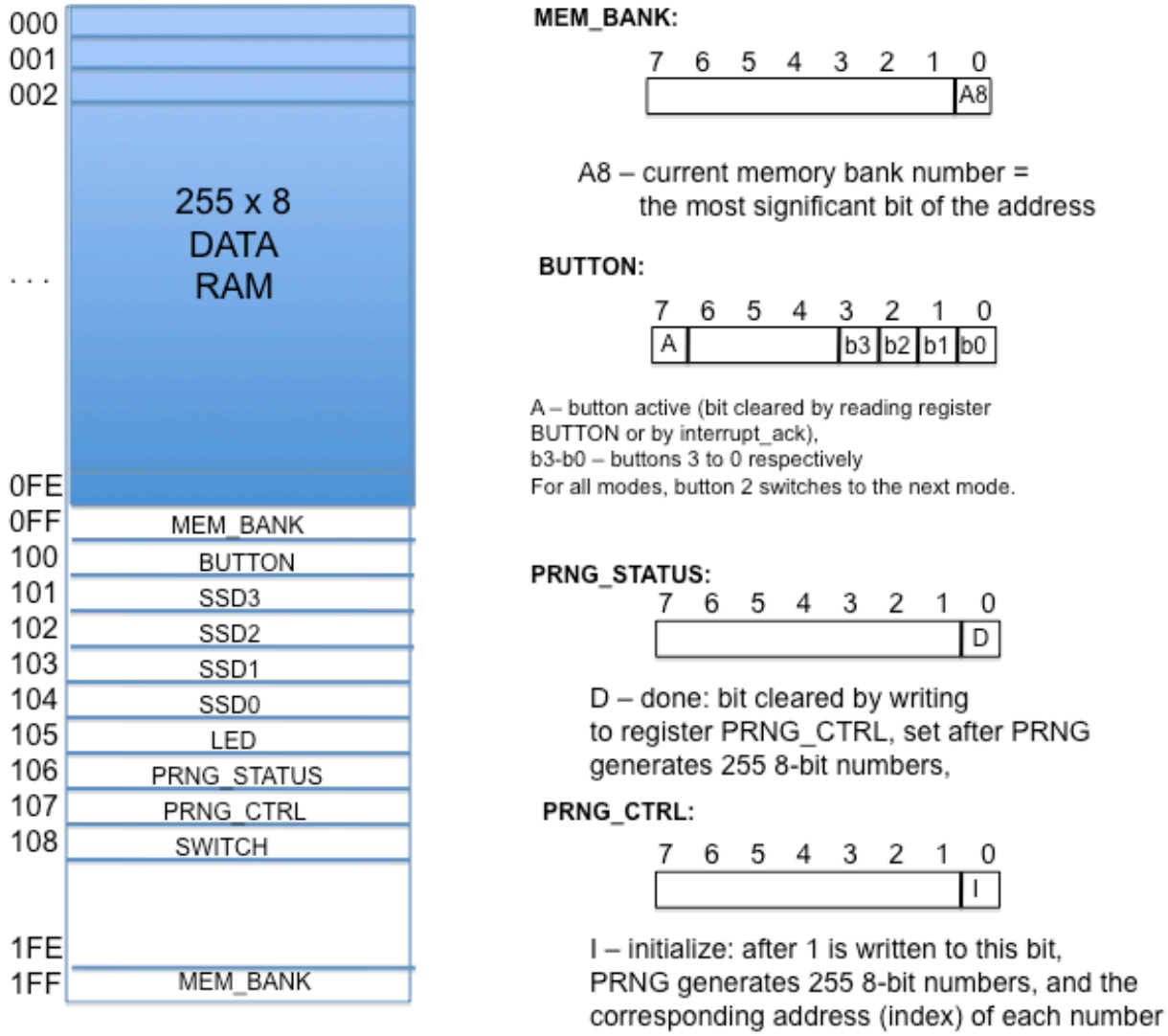


- Picoblaze microcontroller with dedicated Instruction RAM
- Single-port 256x8 Data RAM (out of which we use only 255 locations)
- 8-bit Programmable Pseudorandom Number Generator (PRNG) with hardwired control unit and internal registers PRNG_STATUS and PRNG_CTRL
- Input Interface with the internal registers BUTTON and SWITCH
- Output Interface with the internal registers SSD3-SSD0
- Address Decoder.

During configuration, 256x8 RAM should be initialized to values {"00", "01", ..., "FE", FF"}, i.e., the value at each location should be equal to the address of that location.

The clk signal is not shown in the diagram, but is assumed to be connected to each synchronous component.

The Picoblaze should be able to access Data RAM and internal registers of Input Interface, Output Interface, and PRNG using Memory Map shown in the diagram below:



The Memory Map consists of two banks of memory:

Bank 0, covering addresses 000 to 0FF, includes 255 locations of Data RAM and the MEM_BANK register.
Bank 1, for memory

mapped I/O, covers addresses 100 to 1FF, includes 10 registers: BUTTON, SSD3-SSD0, LED, PRNG_STATUS, PRNG_CTRL, SWITCH, and MEM_BANK.

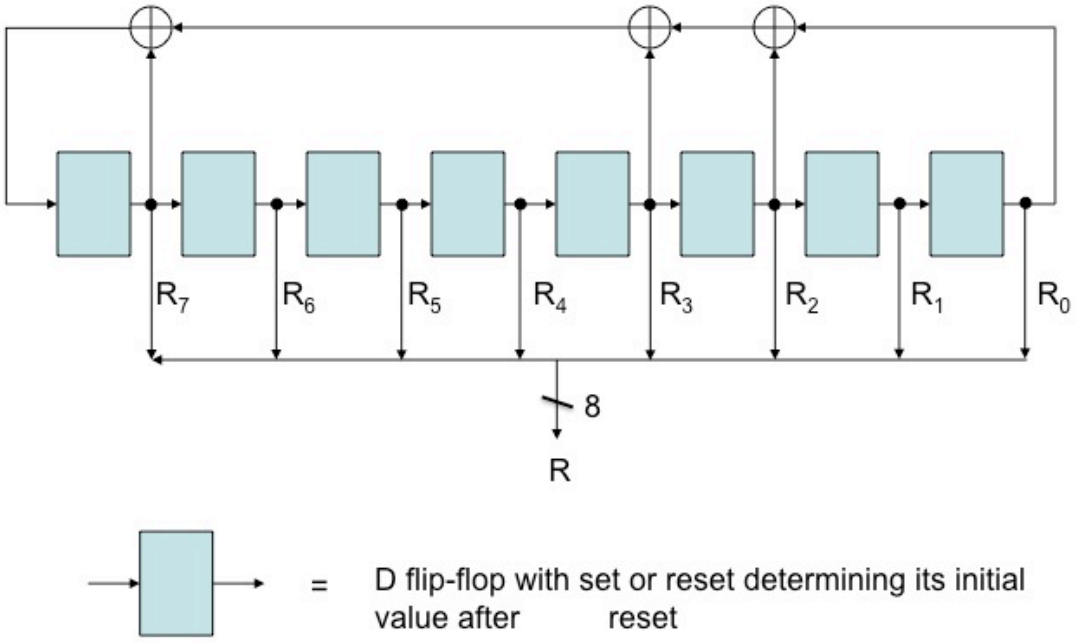
The register MEM_BANK is visible under two memories addressed 0FF and 1FF (writing to both of these addresses changes the same register).

Writing 0 to the least significant bit of MEM_BANK, changes the bank to Bank 0, and sets the most significant bit of the address A[8] to 0, until the next write to MEM_BANK.
Writing 1 to the least significant bit of MEM_BANK, changes the bank to Bank 1, and sets the most significant bit of the address A[8] to 1, until the next write to MEM_BANK.

Input Interface contains the register BUTTON. The four least significant bits of this register correspond to active positions of buttons b3..b0, respectively. The most significant bit of BUTTON, A, is set as the value of an OR function on BUTTON (3-0). Bit A is cleared by reading the register BUTTON or by an active value of interrupt_ack from Picoblaze. All remaining bits are cleared by reading the register BUTTON. Picoblaze becomes aware of the BUTTON being pressed using polling or interrupts (**1 bonus point for using interrupts**). The input interface also contains the "register" SWITCH, which contains the value of the 8 switches of the board.

PRNG has two modes of operation: Idle and Active.
In the default Idle mode, rinit =0, R_wen = 0, RA = 00, and RD = the current state of PRNG. Writing 1 to the bit I of PRNG_CTRL changes the mode of PRNG to Active, and clears the bit D of PRNG_STATUS.
In the Active mode, rinit =1, R_wen = 1 every 8 clock cycles, RA changes between 00 and FE every 8 clock cycles, and RD is equal to a new 8-bit random number every 8 clock cycles. After 255x8 clock cycles, the bit D (Done) of PRNG_STATUS is set to 1, and bit I of PRNG_CTRL is cleared. Reading PRNG_STATUS clears bit D.

The Pseudorandom Number Generator should be based on the Linear Feedback Shift Register, as shown in the diagram below.



In our circuit, PRNG is used to initialize Data RAM. Each time an initialization is performed, the PRNG should start from a different state, which is a last state reached during the previous initialization (please note that the "soft" reset does not change the memory contents).

The program of Picoblaze should support the following four major modes of operation:

- Browsing mode (code "Br")
- Edit mode (code "Ed")
- Initialization mode (code "In")
- Sorting mode (code "So").

In all these modes,
- current memory address should be displayed using another pair of 7 segment displays available on the board
-Button 2= next mode from the list (Browsing, Edit, Initialization, Sorting), changing in the wrap-around fashion

- In general, button 3 is select, button 2 is next, buttons 1 and 0 are up and down respectively.
-Whenever a new mode is selected, a code of this mode (Br, Ed, In, or So) should be displayed using two seven segment displays until some button other than button 2 is pressed. When this happens, the mode is entered.
- Switch 0 should be a soft reset and should trigger every time switch 0 is ‘1’.

"Soft" reset should have the following effect:

- Current mode should be set to the Browsing mode, and the code of this mode, Br, should be displayed using two seven segment displays
- current value of the memory address should be set to 0, and this address should be displayed using two seven segment displays.
- Pseudorandom Number Generator should be initialized to "25" in the hexadecimal representation.

The modes are defined below, and their implementation constitutes subsequent tasks. The exact scope and score for each task may depend on whether you work alone or in a group of two.

Task 1 Browsing Mode (for individuals: required, 1 point; for teams: required, 0.75 point)

After pressing button 3 the circuit should always display a value of a memory location at the current memory address using seven segments displays in the hexadecimal notation.

Button 1 should increment the current memory address in the wrap-around fashion ("FE" followed by "00").

Button 0 should decrement the current memory address in the wrap-around fashion ("00" followed by "FE").

Task 2 Edit Mode (for individuals: required, 1 point; for teams: required, 0.75 point)

The basic functionality should be the same as in the Browsing mode.

However, this time, whenever a data at a certain memory location is displayed, pressing Button 3, should allow you to edit (increment or decrement) data at a given memory location.

Button 1 should increment the current memory location in the wrap-around fashion ("FE" followed by "00").

Button 0 should decrement the current memory location in the wrap-around fashion ("00" followed by "FE").

The current value should be displayed using seven segment displays.

Pressing Button 3 again should fix the value of the current memory location, and allow you to change current memory address using Button 1 and Button 0.

Task 3 Initialization Mode (for individuals: required, 1 point; for teams: required, 0.75 points)

In this mode, each time Button 3 is pressed, the entire memory is initialized with 255 pseudorandom values generated by the 8-bit Pseudorandom Number Generator.

The initialization should not effect the value of the current memory address.

Task 4 Sorting Mode (for individuals: required, 2 points; for teams: required, 2 points)

In this mode, a user first chooses a sorting type using Button 1.

The following sorting types should be available (**for teams:** all modes listed below, **1 point**, **for individuals:** a default type only, other types = **1 bonus point**)

- sorting signed numbers in the descending order (code: Sd, default type), switches: "00"
- sorting signed numbers in the ascending order (code: SA), switches: "01"
- sorting unsigned numbers in the descending order (code: Ud), switches: "10"
- sorting unsigned numbers in the ascending order (code: UA), switches: "11".

Changing the switches changes the sorting type. The code of the currently selected type should be shown on the seven segment displays.

Pressing Button 3 should initiate sorting. The end of sorting should be indicated by displaying the code of Done (Do) using seven segment displays.

Task 5 Advanced Testbench for Sorting (for individuals: 1; for teams: required, 0.75 point)

Develop an advanced testbench based on files capable of automatically testing a subset of your circuit responsible for sorting composed of:

- Single-port 256x8 Data RAM (out of which we use only 255 locations)
- Picoblaze microcontroller with dedicated Instruction RAM
- Address Decoder.

The testbench should be able to initialize Data RAM directly, without using PRNG.
The input file for this testbench should have the following format:

- number of entries to be sorted (in decimal)
- empty line
- numbers to be sorted in the initial order (in the hexadecimal notation, one number per line)
- empty line
- numbers after sorting (in the hexadecimal notation, one number per line)

Lines starting from "#" should be ignored as containing comments.
The testbench should display the binary answer: "Circuit operating correctly" or "Circuit operating incorrectly".
In case an error is detected, the testbench should stop as soon as the first error is detected, and display:

- address=memory address, where the error was detected
- expected = expected value at this memory location,
- actual = actual value at this memory location.

Task 6 Keyboard Input (for individuals: bonus, 2 points; for teams: bonus, 1.5 points)

Replace the buttons on the board with keyboard input via the PS/2 port, along with the input_interface block shown in the block diagram. “Down” should replace button 0, “Up” should replace button 1, “Enter” should replace button 3, and “Right” should replace button 2.

You are permitted to use sample code for the PS/2 keyboard interface. This code can be found online or in the course textbook.

It is recommended that you use the Basys 2 manual as a reference
http://www.digilentinc.com/Data/Products/BASYS/BASYS_E_RM.pdf for the scancodes.

Task 7 Mouse Input (for individuals: bonus 3 points; for teams: bonus 2.25 points)

Replace the input_interface block with an entity that tracks a mouse pointer on a screen. The position of the mouse should be displayed via the VGA connector. If the mouse is clicked when the cursor is in a particular quadrant of the screen, a "button" press should be issued. See below for the mapping of quadrants to button presses.

Button 3	Button 1
Button 0	Button 2

For each of the above Tasks (except Task 5, which requires only functional simulation):

Perform the following tasks to verify the correctness of your designs:

1. Debug your assembly language program using programming environment introduced during the lab.
2. Perform functional simulation.
3. Synthesize your code and perform post-synthesis simulation using Active-HDL.
4. Prepare the UCF (User Constraints File) specifying pin allocations.
5. Implement your code using an appropriate UCF file and chip specification.
6. Check thoroughly implementation reports. Pay attention to pin allocations.
7. Perform timing simulation. Find out what is the maximum clock frequency you can run your circuit at.
8. Download the bitstream to the FPGA board.
9. Verify experimentally the correct operation of your circuit.

Include in the lab report:

1. A detailed block diagram of the Datapath of your circuit.
2. Assembly language source code of the program run on Picoblaze.
3. VHDL code for your circuit and all testbenches used to verify this circuit.
4. Your UCF file for FPGA.

5. Simulation waveforms from the functional, post-synthesis, and timing simulations, proving the correct operation of your circuit, and demonstrating the delay of its critical path.
6. A short report listing all tasks completed successfully and any problems encountered.
   For Tasks 1-4, determine the following parameters of the entire circuit
   - maximum clock frequency
   - critical path
   - resource utilization

Important Dates

	Monday section	Tuesday section	Wednesday section	Thursday section
Hands-on Session and Introduction to the Experiment	04/12/2010	04/13/2010	04/14/2010	04/15/2010
Deliverables Due using Blackboard	**05/03/2010 5:45 PM**	**05/04/2010 7:10 PM**	**05/05/2010 5:45 PM**	**05/06/2010 7:10 PM**
Demonstrations and Exit Quiz	05/03/2009 7:20-10:00 PM	05/04/2009 7:20-10:00 PM	05/05/2009 7:20-10:00 PM	05/05/2009 7:20-10:00 PM

Please email your suggestions to Kris Gaj