

Lab 5: Registers

In this lab we are going to investigate and build several sequential circuits. The behavior of sequential systems depends not only on the current values of the input variables, but also on the sequence of input values that occurred in the past. Such systems have some kind of storage of memory elements. In this lab we are going to design a couple types of registers.

1 Prelab

- 1.1. Design an eight-bit synchronous latch in Verilog.
- 1.2. Design an eight-bit PC register in Verilog.
- 1.3. Write a program which calls that above two designs to test that they function properly.

2 Lab

You will implement five different 8-bit registers: PC (Program Counter), MAR (Memory Addressing Register), OUT (Output), ACCA (Accumulator A), and IRX (Instruction Register X). In addition, you will design two single one-bit registers: C and Z. **What simple circuit elements are C and Z?**

0.1.Information

- 2.1. MAR, OUT, ACCA, and IRX are all 8-bit registers with synchronous parallel load. These registers all have a clock input, an 8-bit data input, and an active low load/enable input, as well as an 8-bit output.
- 2.2. The PC is an 8-bit register with synchronous parallel load capability, synchronous count, and an asynchronous reset. The PC has a clock input, an 8-bit data input, and 3 additional (active low) inputs:
 - PC_Load load the program counter.
 - PC_Increment increments the program counter by 1.
 - Reset resets the program counter to 0.
- 2.3. Implement these registers (synchronous load and synchronous load/count) in Verilog. Include each on in a higher-level design file. Use a DIP switch for the input data, and switches on the evaluation board for PC_Load, PC_Increment, and PC_Clock. **Examine the RTL Schematic to verify proper implementation.**
- 2.4. Verify that the load function work correctly for the parallel load register and that both the load and increment functions work correctly for the load/increment register.

3 Supplement: Registers

A register is a collection of flip-flops which are operated as a set, rather than as individuals (*The Electrical Engineering Handbook, CRC Press*). Two specific examples of registers are discussed in the following sub-sections.

3.1 Simple Latch

Shown in Figure 1, the device takes D_In, Clock, and Load as inputs; it returns D_Out as an output. On an edge of the clock values in D_In will be read, and if load is active used to set the value of D_Out, otherwise the value of D_Out will remain, unchanged.

- Indicated by the absence of a bubble, the device is sensitive to rising edges.
- Indicated by the bubble, Load is active-low. Only when Load is active (low) can a new value be read into the register, otherwise the previous value is held (kept)

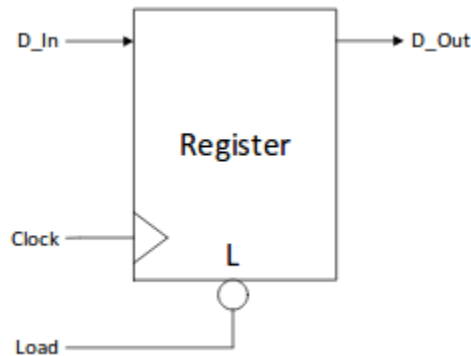


Figure 1: Simple Register/Latch

3.2 Program Counter

Another type of register is called a program counter (PC). It keeps track of which instruction in memory to execute, and must provide the functionality discussed below.

- The program counter needs to know where to start from. It must initialize itself to a specific value when it gets reset. In this case, the program counter should be reset to zero to start execution at the first instruction of the program. This behavior is controlled by the Reset line.
- Programs are, **usually**, executed sequentially. After executing the instruction at address 0x0123, the program will execute the instruction at address 0x0124. As such, the PC needs to support incrementation. This behavior is controlled by the Increment line.
- Other times, the program needs to execute code in a different area of memory (jump

around). Flow control statements such as *for* and *while* do this. In these cases, the PC needs to support being loaded with a new (specified) address. This behavior is controlled by the Load line.

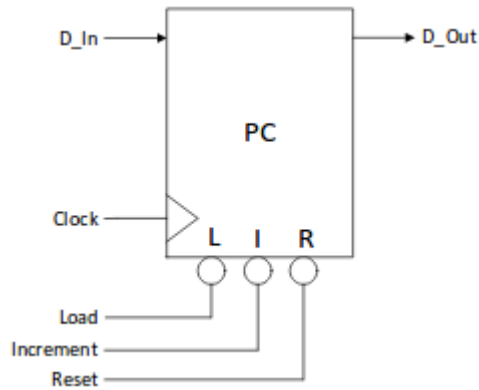


Figure 2: Program Counter (PC) Register.

Specifically, the Program Counter used in this lab (Figure 2) should behave as follows:

- 3.2.1. The PC should increment D_Out to $D_Out + 1$ on the edge of Clock.
- 3.2.2. When Load is low, the input data D_In should be latched into the register on the rising edge of Clock.

The system, which controls PC, will ensure that Load and Increment are never low at the same time. In your program, you should have PC do something sensible, like latch D_In , if both happen to be low simultaneously.

- 3.2.3. When Reset is low, PC should immediately reset to $0x00$ on the next clock edge.

This is normally called a synchronous counter with synchronous load and asynchronous reset. Note: From the bubbles we know that control lines are active low, they should be held high until needed.