CPSC 321 Computer Architecture
Fall Semester 2004
Lab # 6
Introduction to Sequential Circuit Modeling Verilog
Due Date: One week after your lab session
Complete by yourself

1 Structural Modeling of Systems using Decoders and Multiplexers

1.1 Ripple-Carry Adder/Subtractor with Decoders [10 pts]

Design and structurally define in Verilog a 32-bit adder/subtractor using *decoders* as a basic building block. Use 32 decoders to replace each of the 32 one-bit adders. Figure 1 shows how one can define a full-adder using a decoder and a multiplexer. Add a behavioral model to test-bench your design.

1.2 Ripple-Carry Adder/Subtractor with Multiplexers [10 pts]

Design and structurally define in Verilog a 32-bit adder/subtractor using *multiplexer* as a basic building block. Use 32 multiplexers to replace each of the 32 one-bit adders. Figure 1 shows how one can define a full-adder using a decoder and a multiplexer. Add a behavioral model to test-bench your design.

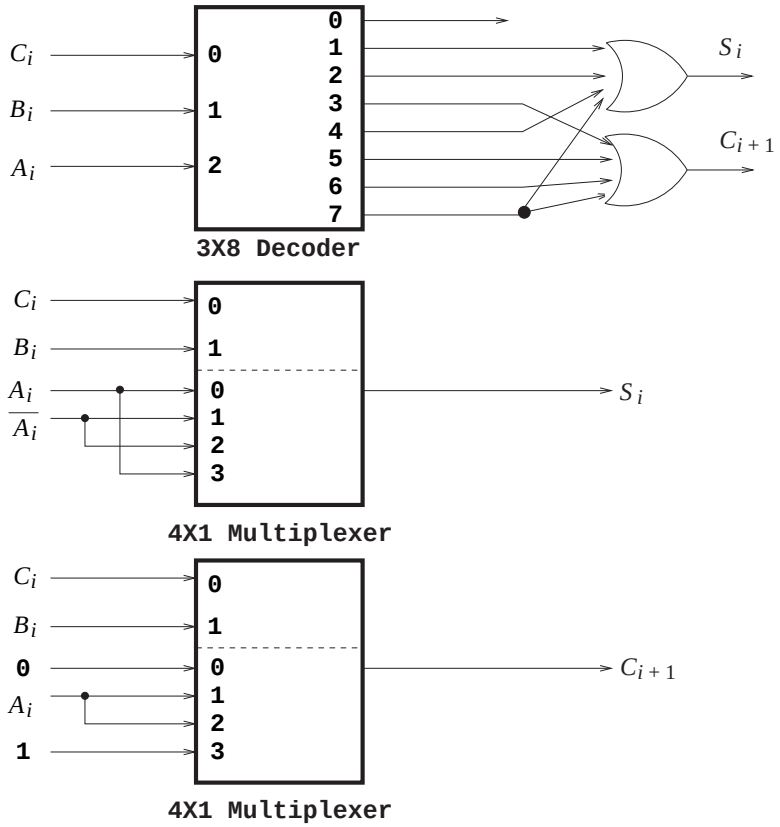

Figure 1: Full-Adders based on Decoders and Multiplexers

2 Structural Modeling of 1-bit Memory Elements

Sequential Circuits (SC) differ from combinational circuits in that they contain *feedback* paths which connect the output of gates to the inputs of "upstream" gates. SCs are said to possess *local state* and to have "writable" (modifiable) memory.

In SCs the *next state* $S(t_{n+1})$ of the system at time t_{n+1}, is determined by the *current state* $S(t_n)$ and the inputs $X(t)$ currently present at the next-state decision time $t < t_{n+1}$. The outputs of SCs are functions of the current state $S(t)$ at time t and optionally of the input at the same time t, where $t_j \leq t < t_{j+1}$, for some j. The formal model of a SC is the Finite-State Machine (FSM) model. SCs are used for two main purposes: (a) as storage devices (eg, registers or main memory), and (b) as building blocks in control units. In this Lab we will investigate the design of SCs as basic building blocks for register storage.

The MIPS processor that we will build in class contains many memory elements. In this lab you will model *structurally* three classic 1-bit memory elements in Verilog: the SR latch, the D latch, and the D-flip flop (D-FF), using logic gates. We will augment the capabilities of a D-FF with *asynchronous* PRESET and CLEAR.

2.1 Structural Modeling of a SR-Latch [10 Points]

The SR latch is the simplest memory element that can be constructed from standard logic gates. It consists of a pair of NOR, or NAND gates, connected as shown in Figure 2 and 3, respectively. The cross-coupling of the gates creates a feedback loop that allows the output of the system to settle to a 1-bit value, ie, 1-bit values are stored in the system as the current state. Notice the convention for labeling outputs in the figures. Within a block, outputs are both marked as Q, and outside the block the presence of the inversion "bubble" allows as to label the outputs as Q and Q', respectively. In the following, we assume that $Q' \equiv \overline{Q}$.

The stored value is output on the line Q, and its inverse on $\overline{Q}$. We can store a new value in the latch simply by "pulsing" one of the two input lines. Pulsing an input line means to momentarily assert (ie, give a logically high or "1" value to) the input. Thus, an input line is pulsed if it is asserted and then de-asserted. The two inputs to the latch are called Set and Reset. When neither Set nor Reset is asserted, the latch simply outputs whatever value was previously stored in it.

If we now pulse the Set input, the latch will store the value "1", and will output "1" on output line Q (and "0" on $\overline{Q}$). After the pulse, the latch continues to remember and output the value "1". Similarly, pulsing Reset will cause the latch to remember and output "0". Thus, the latch remembers whichever input line was most recently pulsed. Set and Reset should never be pulsed at the same time; this is an illegal combination of inputs for the SR latch. The following Verilog module implements an SR latch:

```
module SRlatch (q, qbar, set, reset);
  output q, qbar;
  input  set, reset;
  nor    #1 (q, qbar, reset);
  nor    #1 (qbar, q, set);
endmodule
```

Note that the module is defined structurally (ie, with gates only), and that both NOR gates have been given a non-zero *propagation delay* (the '`#1`' does this).

Propagation Delays The *propagation delay* T_{pr} of a circuit is the time it elapses between a change of an input to propagate and change the output of this circuit. In Verilog and other HDLs, propagation delays **must** be modeled in a gate-level sequential circuit to make it behave properly. Propagation delay

is optional for combinational circuits but it may have to be considered for the proper simulation of the circuit.
Task: Create a new file for this lab and type the SR latch into this file. Add the following test module:

```
module testSR(q, qbar, set, reset);
  input  q, qbar;
  output set, reset;
  reg    set, reset;

  initial begin
     $monitor ($time," q= %d, qbar= %d, set= %d, reset= %d",
               q, qbar, set, reset);
     set = 0; reset = 0;
     #100 set = 1; #100 set = 0;  /* Set pulse at $time==100 $*/
     #100 $finish;                /* $finish simulation       */
  end
endmodule

module testBench;
  wire    q, qbar, set, reset;
  SRlatch l(q, qbar, set, reset);
  testSR  t(q, qbar, set, reset);
endmodule
```

In the test module both Set and Reset are de-asserted at simulation time zero. After a delay of 100 time steps Set is asserted and after another delay of 100, Set is De-asserted. Thus, Set has been pulsed for 100 time steps.

> **More about the value** "x" : In Verilog, when the value of a variable is "x", means that the value is undetermined. ie, the value could be either "0" or "1". When two variables containing "x" and "0" are the inputs to a NOR gate, the gate output must be "x", because we don't know what the correct output of the gate should be. On the other hand, if the two inputs are "x" and "1", the NOR gate output will be "0", not "x", because in this case the input "1" is a controlling input.

2.2 Structural Modeling of a D-Latch [10 Points]

Although the SR latch is simple, it has certain drawbacks:

- the stored value is updated as soon as the inputs, Set or Reset, change, and
- there is one illegal input combination (Set and Reset asserted at the same time).

2.2.1 Design of a D-Latch

To implement and simulate the MIPS processor, we will need a memory element that changes its state only when a change is permitted, not just any time its inputs change. One such device is the D-latch. As you can see in Figure 5, the D-latch is built from an SR latch and a pair of AND gates. A D-latch overcomes both drawbacks of the SR latch: it has a single data input, `D`, rather than the pair of inputs, Set and Reset. This prevents the illegal input combination of both Set and Reset asserted. And, it has a *clock*

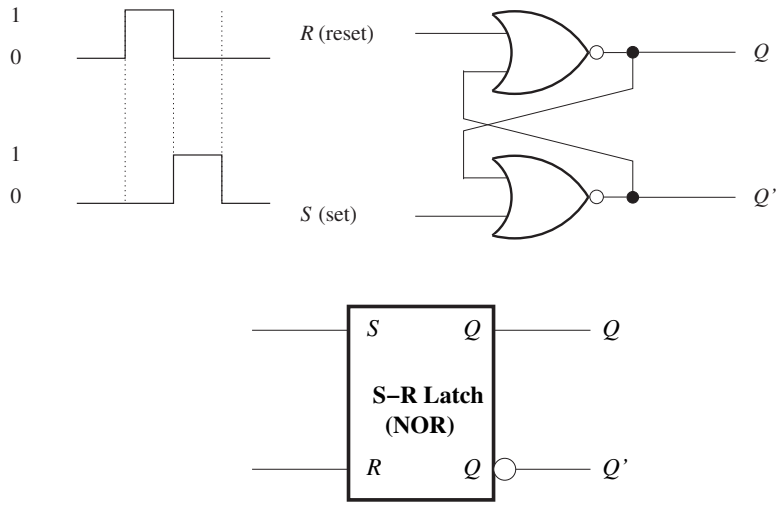

Figure 2: Set-Reset (SR) Latch with NOR Gates and its Symbol

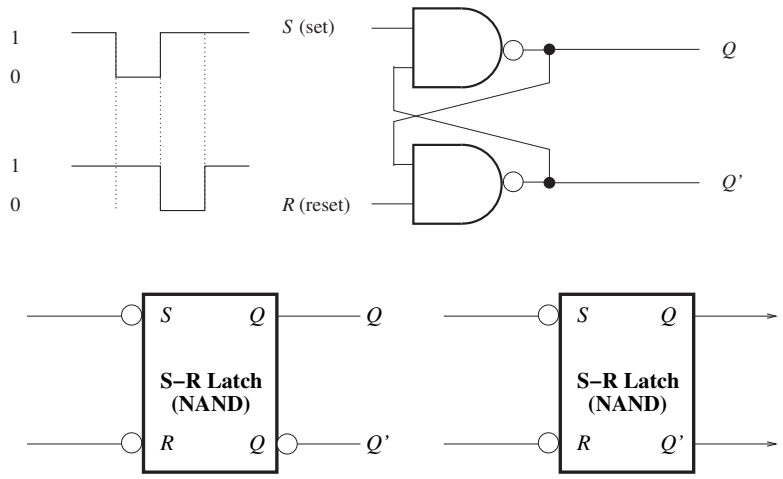

Figure 3: Set-Reset (SR) Latch with NAND Gates and its Symbol

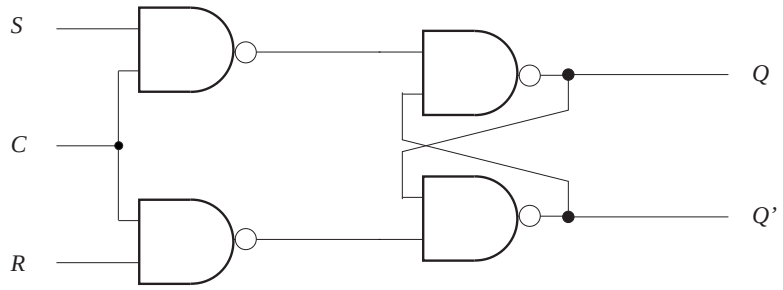

Figure 4: Set-Reset (SR) Latch with NAND Gates and Control Input

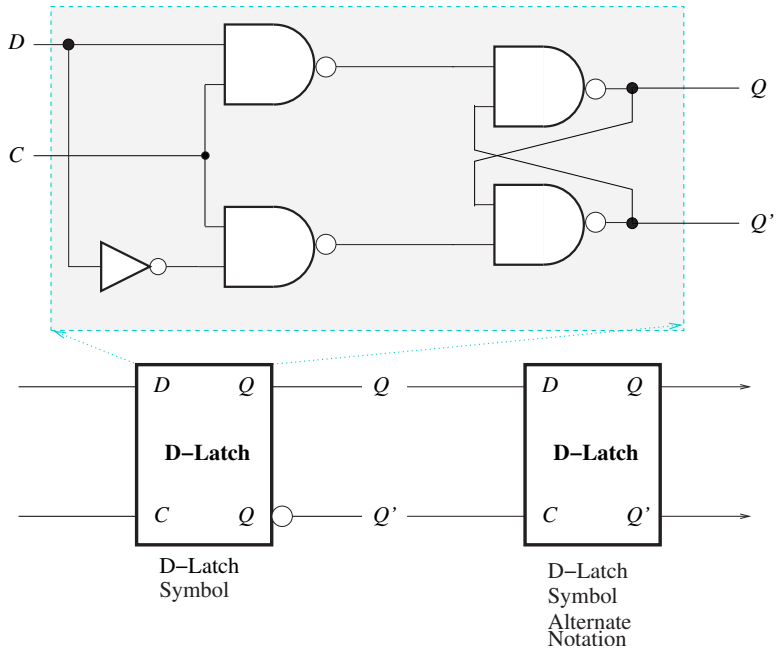

Figure 5: A D-latch based on an SR-Latch with NAND Gates and Control Input and its Symbol

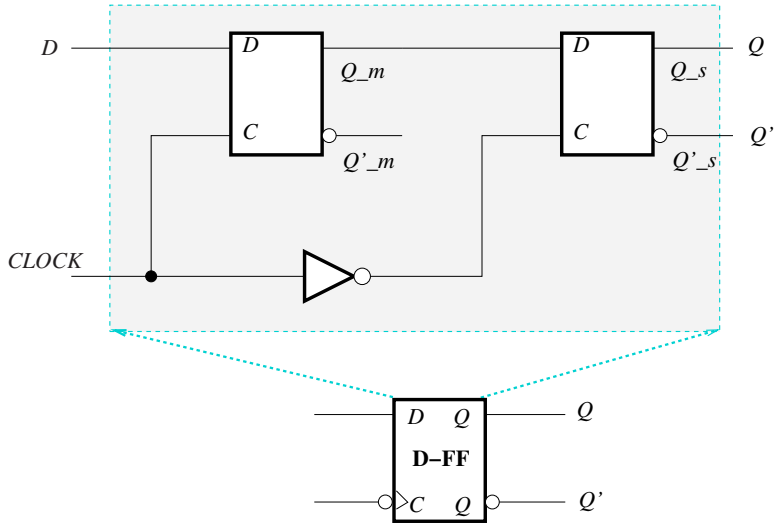

Figure 6: A Master-Slave ("*edge-triggered*") based on D-Latches and its Symbol

input, which determines *when* the memory element is permitted to change state. A clock is a free-running periodic signal that alternates between high ("1") and low ("0") signal levels. The time needed for the clock to complete one full cycle (eg, change from low to high, high to low and then back to high) is called a clock cycle denoted by T_{cc}. Clock cycles are also known as clock periods, or sometimes just "clocks" for short. The exact instant when the clock jumps from a "1" to a "0" is called a **negative clock edge** (or falling or trailing). The instant the clock jumps from a "0" to a "1" is called a **positive clock edge** (or rising or leading).

In the D-latch the pair of AND gates are used to "guard" the embedded SR latch by prohibiting the SR latch's inputs from being pulsed unless the clock signal is "1." When the clock signal is "0", the latch's output cannot be changed, regardless of the value of latch's D input. Thus, a D-latch can be updated only when the clock signal permits.

2.2.2 Clock Signal Generator

In Verilog a clock signal is easily provided using behavioral modeling. The following module represents a `m555` "timer" (clock) chip, which we will use whenever we need a clock signal. Here is the module:

```
module m555 (clock);
  parameter InitDelay = 5, Ton = 50, Toff = 50;

  output clock;
  reg    clock;

  initial begin
     #InitDelay clock = 1;
  end
  always begin
    #Ton  clock = ~clock;
    #Toff clock = ~clock;
  end
endmodule
```

The `m555` module declares a single output, clock, which is defined to be a 1-bit Verilog register. Recall that the `always` keyword defines a Verilog thread of control, which loops forever updating the clock once every time through the loop. The clock cycle time is defined to be 100 time steps. The `initial` keyword defines another thread that executes only once at simulation `$time` = 0, to initialize the clock register. (We will discuss behavioral modeling more fully in the next lab.)

Below is the test module for the D-latch.

```
module testD(q, qbar, clock, data);
  input  q, qbar, clock;
  output data;
  reg    data;
  initial begin
     $monitor ($time, " q= %d, clock= %d, data= %d", q, clock, data);
     data = 0;
      #75 data = 1;
     #100 data = 0;
     #100 $finish;   /* $finish simulation after 100 time simulation units */
  end
endmodule
module testBenchD;
  wire   clock, q, qbar, data;
```

```
  m555   clk(clock);
  Dlatch dl(q, qbar, clock, data);
  testD  td(q, qbar, clock, data);
endmodule
```

Check-Off Requirements Implement the D-latch and test it.

3 Modeling a D Flip-Flop [20 Points]

A D-latch is an example of a *level-triggered* memory device. As long as the clock input is at "1" the latch is "open" and will read and store the data input. When the clock goes to "0", the latch is "closed", and the output is not affected by any changes in the data input. We will need a memory element that can only be changed at particular specified moments. Instead of a level-triggered device that is open for half the clock cycle, we will use a device that can be updated only at those brief instants when the clock jumps from one level to another. A clock transition from "1" to "0" is called a negative clock edge, and we want our memory element to updated only when a negative clock edge occurs. ie, we want our memory element to be edge-triggered, not level-triggered. We can construct a negative edge-triggered D Flip-Flop from a pair of level-triggered D-latches, as shown in Figure 6. These two D-latches are in the *Master-Slave* configuration. An inverter negates the clock input to the second latch, so only one latch can be open at any time. When the clock signal is "1", the first latch is open and accepts a new value, but the second latch is closed. When a negative edge of the clock occurs, the first latch closes and the second one opens, reading the value stored in the first latch.

3.1 Set-Up and Hold Times

All flip-flops have the Set-Up and Hold times requirements. The input must be stable for a minimum amount of time before the clock edge, called the Set-Up time. Additionally, the input must remain stable after the clock edge for an amount of time called the Hold Time. Hold times are usually zero, but minimum Set-up times need to be maintained. In actual circuits, a change of the input within its Set-up period may even lead to *meta-stable* flip-flop state where the output is unpredictable.
Check-Off Requirements Implement the D flip-flop and test it.

4 Structural Modeling D-Flip Flops with Preset and Clear Capabilities

This part requires you to extend the baseline D-Flip Flop with additional capabilities.

4.1 D-Flip Flops w/Preset and Clear [20 Points]

Extend the capabilities of the base D-FF you implemented in previous sections with the *Load*, *Preset* and *Clear* capability, as shown in Fig. 7. Let's call the extended D-FF-PC. The `Write_en` line is set (to 1) only when we want to allow the D-FF-PC to sample and store its current input D in its storage (Q, and Q'). The actual storing of the current input takes place when the clock signal `CLK` makes the HIGH to LOW transition. The `Clear_b` signal is set to "active low" (ie, 0) value when we want to set the contents of the D-FF-PC to 0 ($Q = 0$, and $Q' = 1$). In a similar fashion, the `Preset_b` is set to "active low" (0) signal value when we want to preset the D-FF-PC to 1 ($Q = 1$, and $Q' = 0$). Both `Clear_b` and `Preset_b` are *asynchronous* to and have precedence over the current clock signal `CLK`.

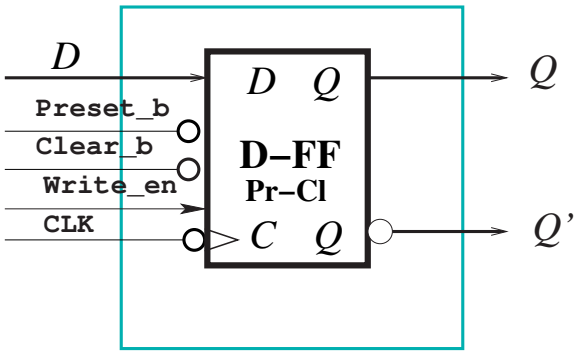

Figure 7: D-Flip Flop with Load, Preset and Clear Capabilities

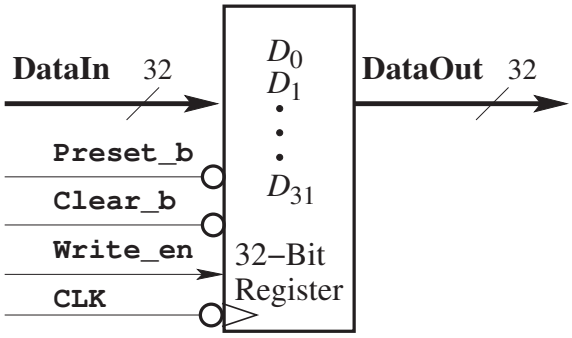

Figure 8: n–bit Register with Parallel Load, Preset and Clear

4.2 Registers from D Flip-Flops w/Preset and Clear [20 Points]

A n-bit register consists of n, 1-bit storage elements. Each storage element is a 1-bit D-FF with Load, asynchronous Preset and Clear signals, as defined previously in Subsection 4.1. All flip flops are fed with a common clock signal `CLK`, a `Write_en`, and asynchronous `Clear_b` and `Preset_b` signals, as explained in Subsection 4.1. Fig. 8 shows a register block of this type.

Implement a 32-bit register with 32, 1-bit D-FF with Preset, Clear and Parallel Load.