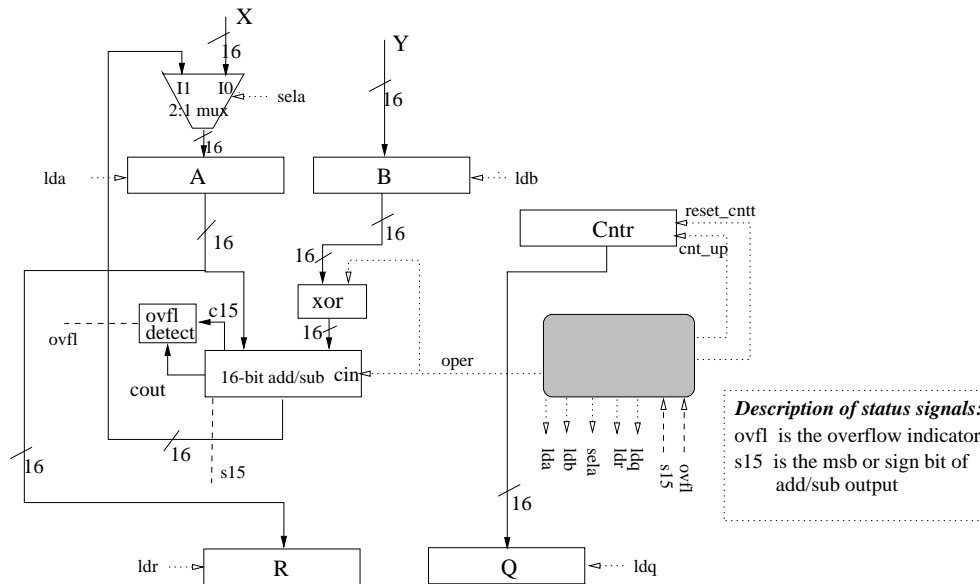


ECE 366, Fall 2001, Instructor: Prof. Shantanu Dutt

Midterm Exam Solutions

1. Datapath Control

A datapath is given below.



Description of control signals:
lda = 1 means load/write A
ldb = 1 means load/write B
sela = 0 means X is connected to input of reg. A
sela = 1 means add/sub output is connected to input of reg. A

oper = 0/1 means ADD/SUB resp.
cnt_up = 1 means Cntr <- Cntr + 1
reset_cntr = 1 means Cntr <- 0
ldr = 1 means load/write R
ldq = 1 means load/write Q

Description of combinational logic units:
ovfl detect: $c15 \text{ xor } cout$
 xor: xor's each bit of M w/ "oper"
 $\Rightarrow \text{xor o/p is } M \text{ if } oper=1 \text{ else it is } M$

NOTE: All write/cnt up operations complete only at the +ve edge of the next cc (since they all load new values into regs)
 NOTE: Signal lines without any specified width are 1-bit wide

(a) Give the Moore state diagram of the FSM for the control unit that performs the following function described below using regular programming language statements:

```

begin
  cnt=0; A=X; B=Y;
  while (A ≥ B) do begin
    A = A - B; cnt=cnt+1;
  endwhile
  R =A; Q=cnt;
end

```

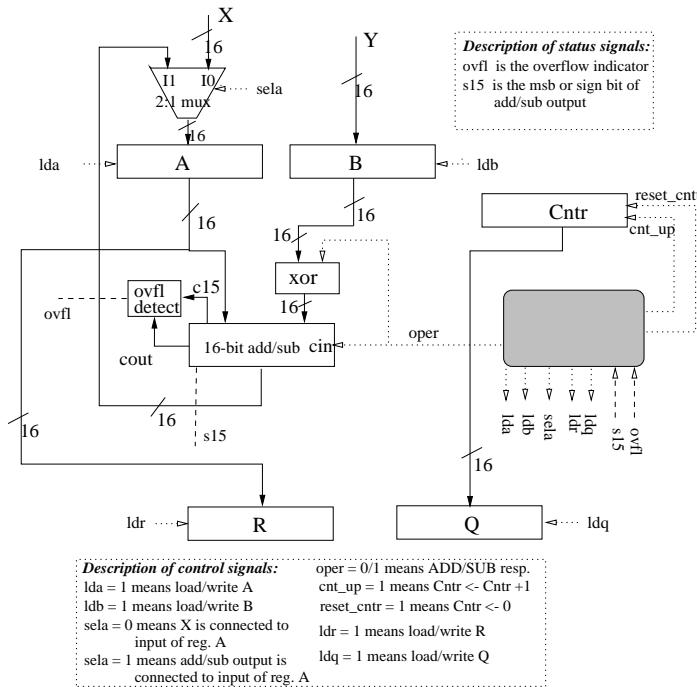
IMPORTANT: State beside each state of your FSM the RTL description of what is being accomplished in that state.

You will be graded most importantly on correctness of your FSM and then on the speed and number of states of your design. **50**

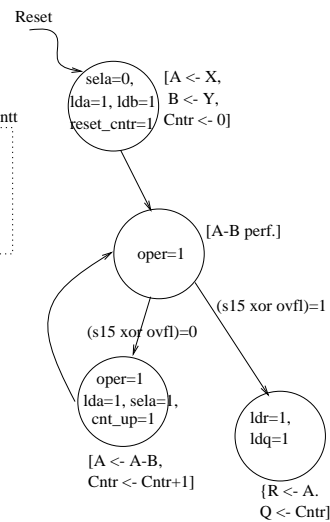
(b) What function of X and Y is stored at the end in R and Q? **10**

Solution:

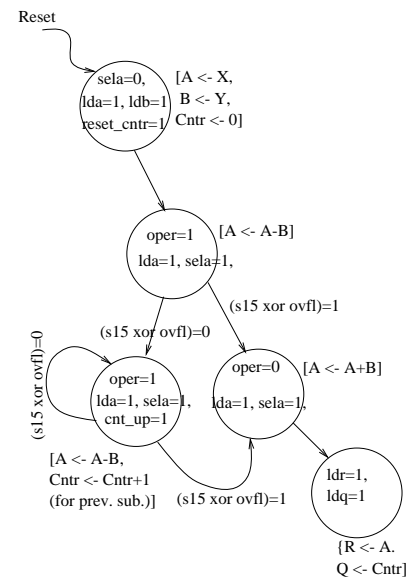
(a)



(a) Datapath for iterative-subtraction based division



(b) FSM for iter-sub division
 Analysis: $2Q+3$ cc's (Q is quotient)



(c) Faster FSM for iter-sub division
 Analysis: $Q+4$ cc's

(b) Finally R has the remainder and Q the integer quotient of the division X/Y.

2. VHDL Code Modification

Consider the following VHDL behavioral description of a D-FF:

```
entity dff is
port (d, clk, reset:in bit; q: out bit);
end entity fsm1;

architecture behav of dff is

transition: process is

wait until (clk'event and clk='1') or (reset='1')
if reset='1' then q <= '0' after 8 ns;
else q <= d after 8 ns;
end process transition;
end architecture behav;
```

In the above description, note that the “reset” signal is *asynchronous*, i.e., irrespective of the state of the clock, if the reset signal is asserted (made = 1) then the output of the D-FF goes to 0 (with a delay of 8 ns).

As far as the simulation of the above description is concerned, at every positive edge of the clock the if-then-else statement in process “transition” will be executed irrespective of whether the “d” input is changed or if “reset” is asserted. This results in a waste of simulation effort if neither “d” changes nor is “reset” asserted. Change the above code so that the next round of simulation of the process is performed only if either “d” changes or “reset” is asserted. Note that the D-FF still has to be positive edge triggered.

Solution: Multiple solutions are possible. Two are given below:

(i) Process code is changed to:

transition: process is

```
wait until (clk'event and clk='1') or (reset='1')
```

```
if reset = '1' then q <= '0' after 8 ns;
```

```
else q <= d after 8 ns;
```

```
if reset = '1' then wait on reset;
```

```
else wait on d, reset;
```

```
end process transition;
```

(ii) Simpler solution (less effective as process will loop back even when “reset” remains '1'). Process code is changed to:

transition: process is

```
wait until (clk'event and clk='1') or (reset='1')
```

```
if reset = '1' then q <= '0' after 8 ns;
```

```
else q <= d after 8 ns;
```

```
wait on d, reset;
```

```
end process transition;
```

3. Timing Diagram

Consider the following VHDL code in which two processes communicate using signalling (hand-shaking).

```
entity comm_modules is
end entity comm_modules;
```

```
architecture behav of comm_modules is
```

```
signal read, dataready : bit := '0';
```

```
signal data: integer;
```

```
begin
```

```
consumer: process is
```

```
variable reg1, reg2: integer;
```

```
begin
```

```
wait until dataready = '0';
```

```
read <= 1 after 20 ns; – signalling for next data item
```

– 20 ns is the delay of the “read” signal to reach the “producer”

– process, i.e., 20 ns is the “read” signal’s propagation delay.

– Note that the above signal assignment statement with a delay of 20 ns

– does NOT advance simulation time; it only causes the “read” signal

– to become ‘1’ at time $t + 20$ ns where t is the current simulation time.

– Since there is no advance of simulation time, the next statement below

– is executed without any further time elapsing since the execution of the

– above signal assignment statement. i.e., it is executed at time t .

```
wait until dataready = '1';
```

– note that “wait” statement (of all types) causes an advance of simulation time.

– Thus in the above statement, if t is the current simulation time

– and if “dataready” becomes ‘1’ after x ns ($x > 0$) from the

– current time, then the next statement is executed at time $t + x$ ns.

```
reg1 := data; – store sent data in register reg1
```

```
read <= 0 after 20 ns;
```

```
wait for 10 ns; – simulating time to process data in reg1
```

```
reg2 := reg1; – simulating storing processed data in reg2
```

– can now loop back to obtain next data item

```
end process consumer;
```

```

producer: process is
variable count: integer := 0;
begin
wait until read = '1';
count := count+1;
wait for 5 ns; – simulating time to produce new data
data <= new_data(count) after 20 ns;

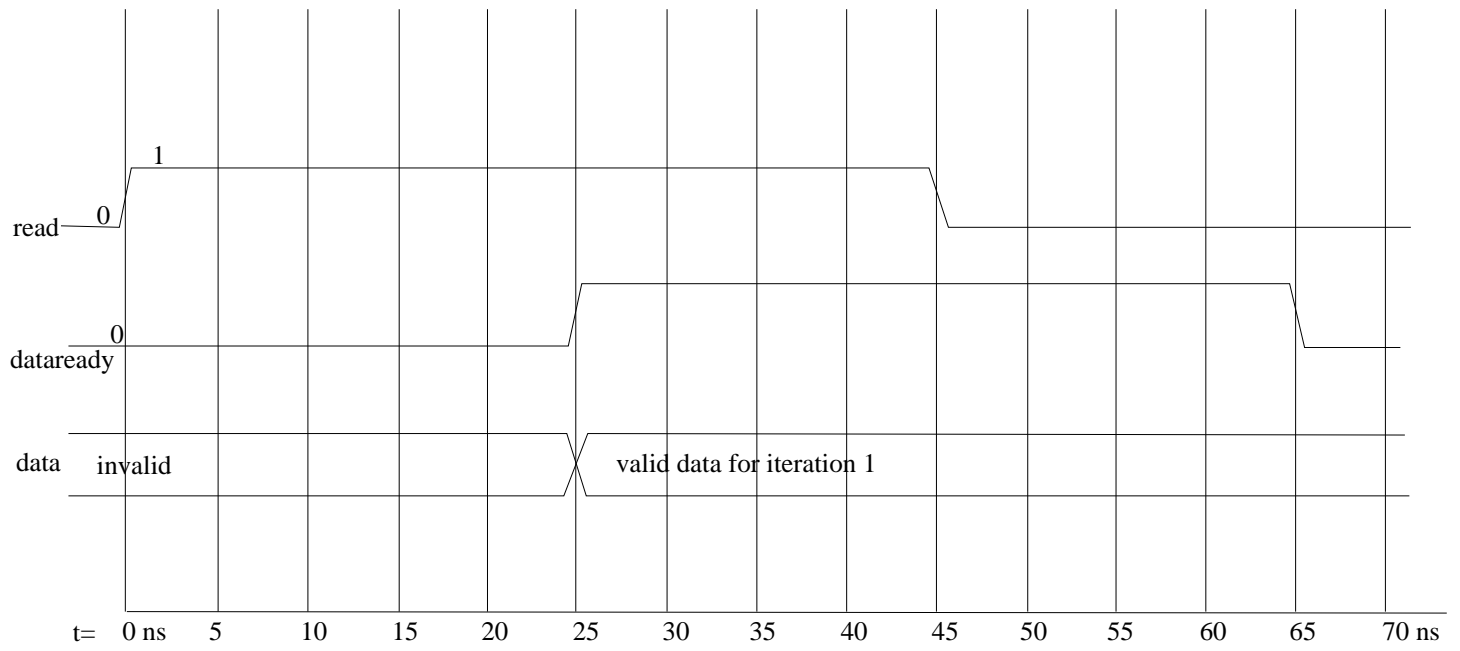
– new_data is a function that produces the next data item;
– the function is not given here for simplicity and it is not relevant to our problem
– 20 ns is the propagation delay on the “data” line only after
– which time the new data is seen by the “consumer” process.

dataready <= '1' after 20 ns; – again, 20 ns is the propagation delay
wait until read = '0'; – wait until signal that data has been stored by the “consumer” process
dataready <= '0' after 20 ns; – signal invalid data for the next iteration
end process producer;
end architecture behav;

```

In the timing chart given on the next page, draw the various transitions of the 3 signal lines “read”, “dataready” and “data” at the correct times as determined from the above code for one iteration of signaling, i.e., starting with “read” going to '1' at time $t = 0$ (this transition is already shown in the chart), show exactly one $0 \rightarrow 1$ and $1 \rightarrow 0$ transition on the “read” and “dataready” lines. The availability of valid new data on the “data” line should be shown by a *crossover* as illustrated in the figure.

Solution:



(a) Timing Chart