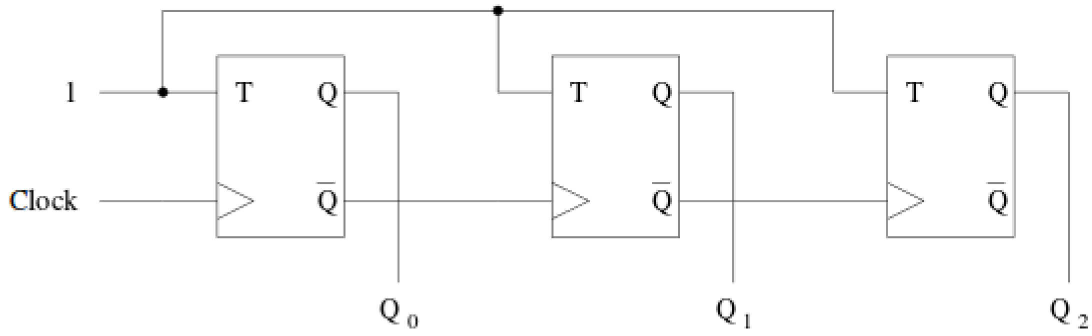
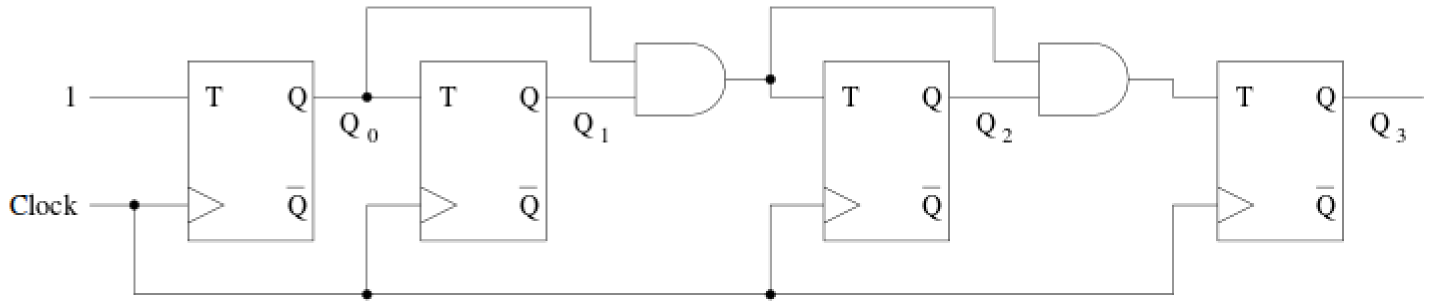
UCF

Stands For Opportunity

Example: 4-Bit Ripple Adder





Example: 4-Bit Ripple Adder

```
// module D_FF with synchronous reset
module D_FF(q, d, clk, reset);

output q;
input d, clk, reset;
reg q;

// Lots of new constructs. Ignore the functionality of the
// constructs.
// Concentrate on how the design block is built in a top-down fashion.
always @(posedge reset or negedge clk)
if (reset)
     q <= 1'b0;
else
     q <= d;

endmodule
```

Example: 4-Bit Ripple Adder

```
module T_FF(q, clk, reset);

output q;
input clk, reset;
wire d;

D_FF dff0(q, d, clk, reset);
not n1(d, q); // not is a Verilog-provided primitive. case sensitive
endmodule
```

```
module stimulus;

reg clk;
reg reset;
wire[3:0] q;

// instantiate the design block
ripple_carry_counter r1(q, clk, reset);

// Control the clk signal that drives the design block. Cycle time = 10
initial
    clk = 1'b0; //set clk to 0
always
    #5 clk = ~clk; //toggle clk every 5 time units

// Control the reset signal that drives the design block
// reset is asserted from 0 to 20 and from 200 to 220.
initial
begin
    reset = 1'b1;
    #15 reset = 1'b0;
    #180 reset = 1'b1;
```

Example: 4-Bit Ripple Adder

```
    #10 reset = 1'b0;
    #20 $finish; //terminate the simulation
end

// Monitor the outputs
initial
    $monitor($time, " Output q = %d",  q);

endmodule
```

Example: 4-Bit Ripple Adder

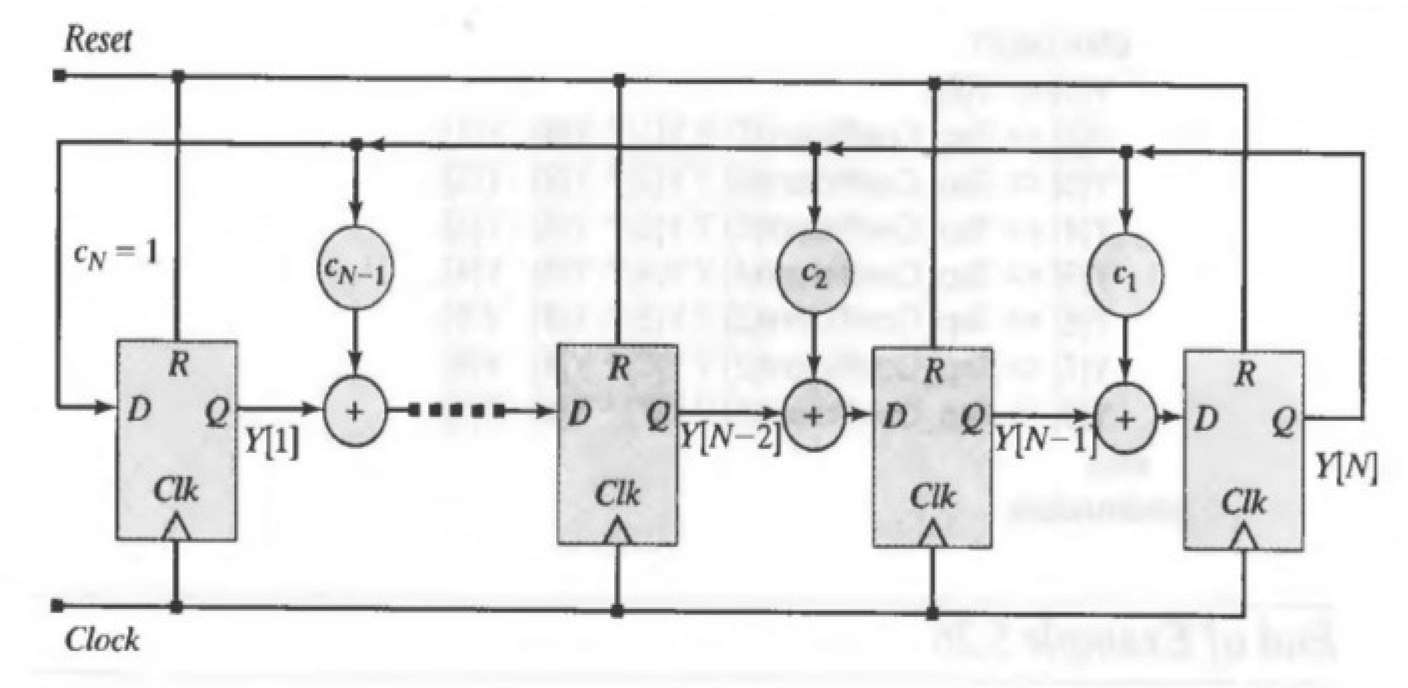
```
module ripple_carry_counter(q, clk, reset);

output [3:0] q;
input clk, reset;

//4 instances of the module T_FF are created.
T_FF tff0(q[0],clk, reset);
T_FF tff1(q[1],q[0], reset);
T_FF tff2(q[2],q[1], reset);
T_FF tff3(q[3],q[2], reset);

endmodule
```

LFSR-Linear-Feedback Shift Register



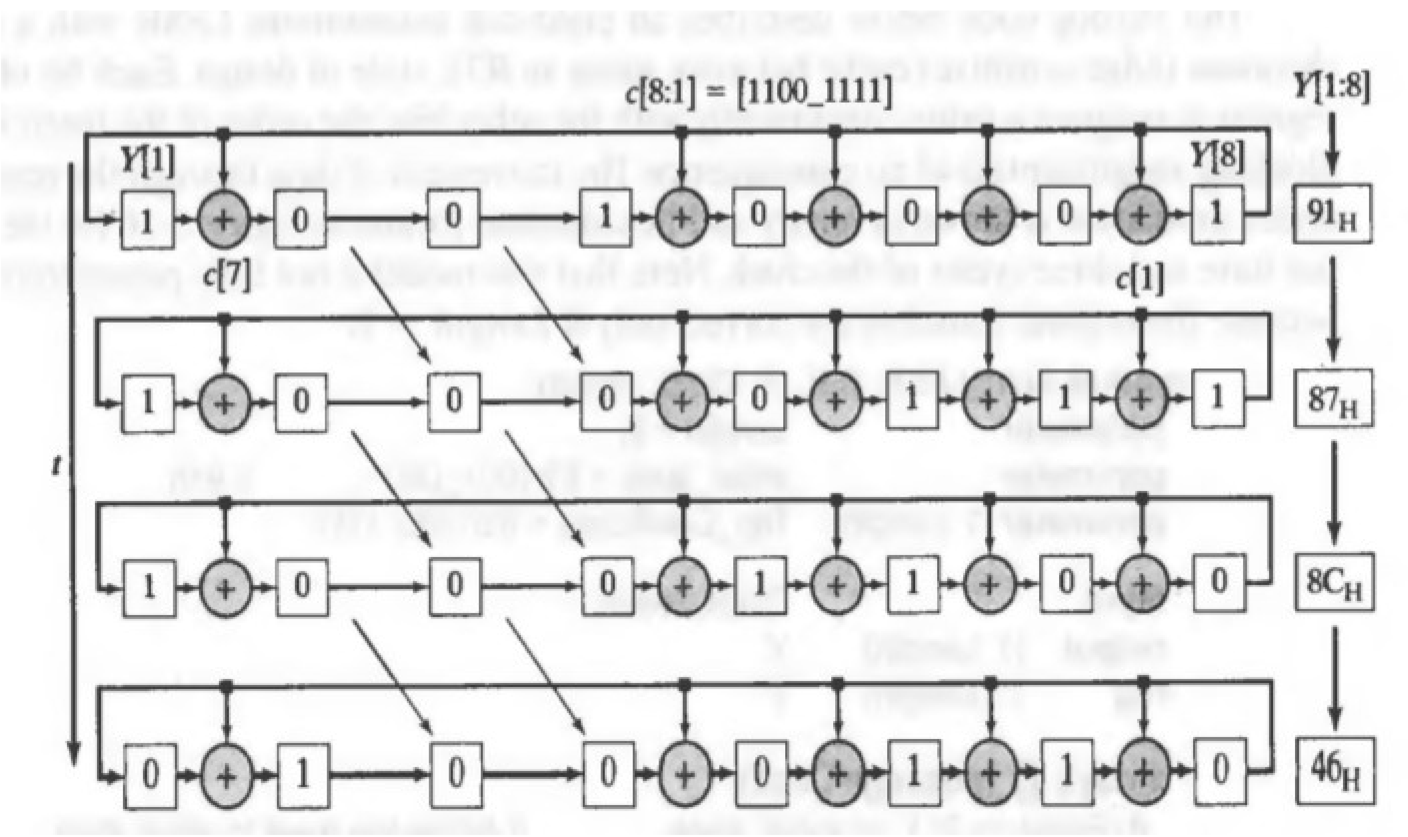
Data-Flow Model of LFSR

```
module Auto_LFSR_RTL (Y, Clock, Reset);
  parameter                Length = 8;
  parameter                initial_state = 8'b1001_0001;     // 91h
  parameter [1: Length]    Tap_Coefficient = 8'b1100_1111;

  input                    Clock, Reset;
  output    [1: Length]    Y;
  reg       [1: Length]    Y;

  always @ (posedge Clock)
   if (Reset == 0) Y <= initial_state;        // Active-low reset to initial state
     else begin
      Y[1] <= Y[8];
      Y[2] <= Tap_Coefficient[7] ? Y[1] ^ Y[8] : Y[1];
      Y[3] <= Tap_Coefficient[6] ? Y[2] ^ Y[8] : Y[2];
      Y[4] <= Tap_Coefficient[5] ? Y[3] ^ Y[8] : Y[3];
      Y[5] <= Tap_Coefficient[4] ? Y[4] ^ Y[8] : Y[4];
      Y[6] <= Tap_Coefficient[3] ? Y[5] ^ Y[8] : Y[5];
      Y[7] <= Tap_Coefficient[2] ? Y[6] ^ Y[8] : Y[6];
      Y[8] <= Tap_Coefficient[1] ? Y[7] ^ Y[8] : Y[7];
    end
endmodule
```

Repetitive Logic Modeling



Repetitive Logic Modeling

```
module Auto_LFSR_ALGO (Y, Clock, Reset);
 parameter     Length = 8;
 parameter     initial_state = 8'b1001_0001;
 parameter     [1: Length] Tap_Coefficient = 8'b1100_1111;
 input         Clock, Reset;
 output        [1: Length] Y;
 integer       Cell_ptr;
 reg           Y;

 always @ (posedge Clock)
  begin
   if (Reset == 0) Y <= initial_state;                 // Arbitrary initial state, 91h
   else begin  for (Cell_ptr = 2; Cell_ptr <=Length; Cell_ptr = Cell_ptr +1)
    if (Tap_Coefficient [Length - Cell_ptr + 1] == 1)
     Y[Cell_ptr] <= Y[Cell_ptr - 1]^ Y [Length];
     else
     Y[Cell_ptr] <= Y[Cell_ptr - 1];
     Y[1] <= Y[Length];
   end
  end
endmodule
```

A for loop has the form:

```
for(initial_statement;   control_expression;   index_statement)
statement_for_execution;
```

For loop, repeat loop, while loop, ...

```
for(initial_statement;   control_expression;   index_statement)
statement_for_execution;
```

```
...
word_address = 0;
repeat (memory_size)
 begin
  memory [ word_address] = 0;
  word_address = word_address + 1;
 end
...
```

```
while (expression) statement;
```

Majority Module

```
module Majority_4b (Y, A, B, C, D);
  input   A, B, C, D;
  output Y;
  reg      Y;
  always @ (A or B or C or D) begin
    case ({A, B,C, D})
      7, 11, 13, 14, 15:        Y = 1;
      default                   Y = 0;
    endcase
  end
endmodule

module Majority (Y, Data);
  parameter      size = 8;
  parameter      max = 3;
  parameter      majority = 5;
  input          [size-1: 0]      Data;
  output                          Y;
  reg                             Y;
  reg            [max-1: 0]       count;
  integer                         k;

  always @ (Data) begin
    count = 0;
    for (k = 0; k < size; k = k + 1) begin
      if (Data[k] == 1) count = count + 1;
    end
    Y = (count >= majority);
  end
endmodule
```

Parametrized Models

```
module Auto_LFSR_Param (Y, Clock, Reset);
  parameter             Length = 8;
  parameter             initial_state = 8'b1001_0001; //   Arbitrary initial state
  parameter [1: Length] Tap_Coefficient = 8'b1100_1111;

  input                 Clock, Reset;
  output [1: Length]    Y;
  reg [1: Length]       Y;
  integer               k;

  always @ (posedge Clock)
    if (Reset==0) Y <= initial_state;
     else begin
       for (k = 2; k <= Length; k = k + 1)
         Y[k] <= Tap_Coefficient[Length-k+1] ? Y[k-1] ^ Y[Length] : Y[k-1];
         Y[1] <= Y[Length];
    end
endmodule
```

Clock Signal Generation

```
parameter half_cycle = 50;
parameter stop_time = 350;
initial
 begin: clock_loop        // Note: clock_loop is a named block of statements
  clock = 0;
  forever
   begin
    #half_cycle clock = 1;
    #half_cycle clock = 0;
    end
 end

initial
 #350 disable clock_loop;
```

Finding First 1 Circuit

```
module find_first_one (index_value, A_word, trigger);
  output [3: 0]    index_value;
  input [15: 0]    A_word;
  input            trigger;
  reg    [3: 0]    index_value;
  always @  (trigger)
    begin: search_for_1
     index_value = 0;
     for (index_value = 0; index_value <= 15; index_value = index_value + 1)
       if (A_word[index_value] == 1) disable search_for_1;
   end
endmodule
```

Final issues

- Please fill out the student info sheet before leaving
- Come by my office hours (right after class)
- Any questions or concerns?