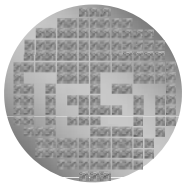


VLSI Design

Tsung-Chu Huang

Department of Electronic Engineering
National Changhua University of Education
Email: tch@cc.ncue.edu.tw

2016/05/10

Sequential Circuits

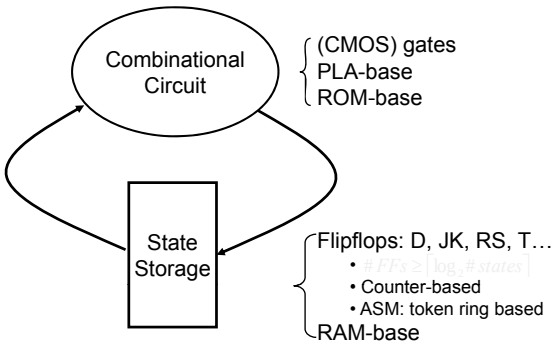
A. Asynchronous
1. Extremely large set! Even synchronous circuits are usually considered as asynchronous circuits in general sense.
2. For some applications, the delay of signals can be looked as a small D-FF and they are usually designed according to Huffman model.
3. Some specials: self-time logics, hand-shaking, bi-directional, etc.

B. Synchronous

1. Single Clock
1. Single Phase
   - 1C1P DFF Array-based Finite State Machine
   - The rest can be transferred to the DFF-based
   - Most popular and easy to design
2. Complemented Phase, Double-edge-triggered, Double-data-rate
3. Multiple phase (exclusive pulses)

2. Multiple Clocks
1. Clock domain interfacing, complicated!

Implementations of 1C1PD-FSM

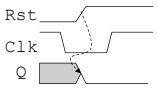


Example: Flipflops

- Positive-Edge Triggered D-Flipflop:

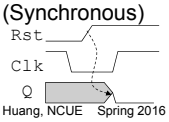
```
module DFF(Clk, D, Q);
input  Clk, D;
output Q;
reg    Q;
    always@(posedge Clk)
       Q=D;
endmodule
```

- Resettable DFF: (Asynchronous)




```
module DFF(Rst,Clk, D, Q);
input  Rst, Clk, D;
output Q;
reg    Q;
    always@(posedge Clk or posedge Rst)
       if(Rst) Q=1'b0;
       else    Q=D;
endmodule
```

- Resettable DFF: (Synchronous)




```
module DFF(Rst,Clk, D, Q);
input  Rst, Clk, D;
output Q;
reg    Q;
    always@(posedge Clk)
       if(Rst) Q=1'b0;
       else    Q=D;
endmodule
```


Typical Design Styles of 1C1PD-FSM

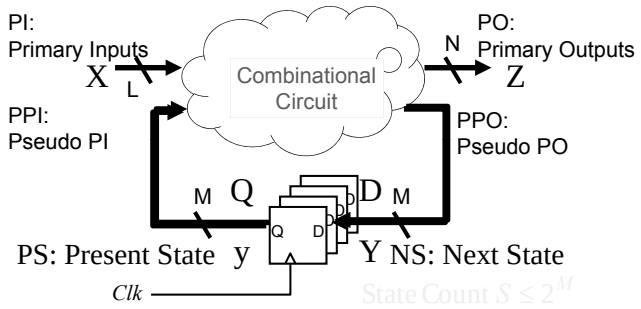
A. Typical Finite State Machine
1. Manual Design: Reviewed and illustrated by a simple example.
2. Automatically Synthesis: Using Verilog HDL.

B. Algorithmic State Machine (ASM)

C. Counter-Based Finite State Machine

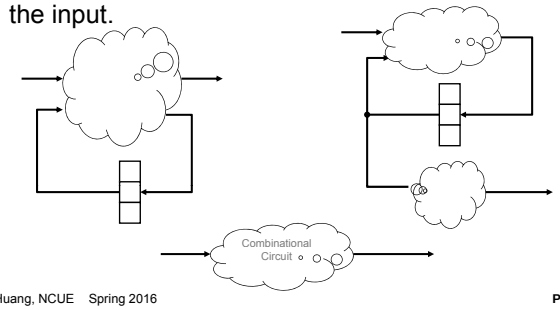
Huffman Model for a Finite State Machine

Single-Clock Single-Phase DFF Array-based



Classification of FSM's

- Mealy Machine: The outputs are dependent on both the state and the input.
- Moore Machine: The outputs are dependent on only the present state.

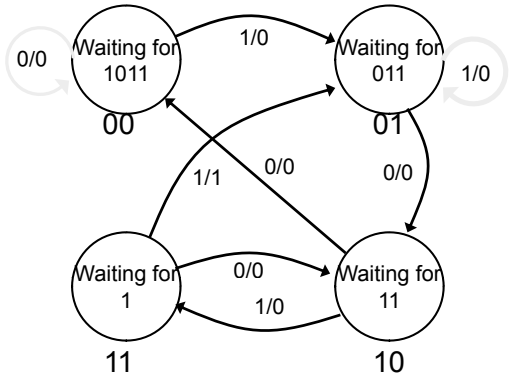


Optimal FSM Design

1. **Problem Formulation**
   - The most difficult for automation
2. **Symbolic State Diagram**
   - Problem entry for GUI base synthesizers
3. **State Naming**
   - Assign a binary state to a state symbol.
4. **DXQZ: Transition/Output Table**
   - = Binary code state diagram
5. **Combinational Logic Compaction**
   - 2-level Optimization
     - Manually Karnaugh-Map
     - Quintus-McKlusky Algorithm
   - Multiple (several) Level Logic Optimization
     - Espresso (Several-level PLA optimalization)

State Diagram

Example of Mealy Machine: A Lock with Password 1011



State Table & Flow Table

Transition Table

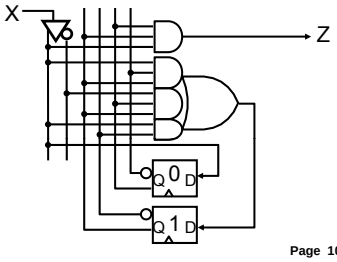
Q		X	D		Z
Q_1	Q_0	X_0	D_1	D_0	Z_0
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	1	0	0
1	1	1	0	1	1

K-map or
McKlusky Method:

$$D_1 = \overline{Q}_1 X + Q_1 Q_0 \overline{X} + Q_1 \overline{Q}_0 X$$

$$D_0 = X$$

$$Z = Q_1 Q_0 X$$



Typical Description of FSMs in HDL

```
always@(Present_State or Inputs)
     case(ALL CASES)
          Next_State = fs(Present_State, Inputs);
          Outputs    = fo(Present_State, Inputs);
     endcase
```

```
always@(posedge Rst or posedge Clk)
     if(Rst)  Present_State = Initial_State;
     else     Present_State = Next_State;
```


Blocking and Non-blocking Assignments

- **Blocking Assignments**

```
always@(posedge Clk)
      begin
            X = Y + 1;
            Y = X << 1;
      end
```

Assume X==3 and Y==6 at first, the simulator evaluates Y+1==7 and gives 7 to X; then it evaluates X<<1 == 14 and gives 14 to Y.

```
always@(posedge Clk)
      begin
            Y = X << 1;
            X = Y + 1;
      end
```

Assume X==3 and Y==6 at first, the simulator evaluates X<<1==6 and gives 6 to Y; then it evaluates Y+1 == 7 and gives 7 to Y.

- **Non-blocking Assignments**

```
always@(posedge Clk)
      begin
            X <= Y + 1;
            Y <= X << 1;
      end
```

Assume X==3 and Y==6 at first, the simulator evaluates X'==Y+1==7 and Y'==X<<1==6, then it gives values: X←X'==7 and Y←Y'==6;

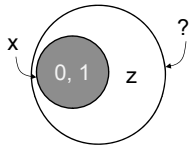
```
always@(posedge Clk)
      begin
            Y <= X << 1;
            X <= Y + 1;
      end
```

The simulation results will be the same as the above and it can be synthesized.

Switch-Case

case, casex, casez

```
case (expression)
      constant1:  statement1;
      :
      constantj:  statementj;
      :
      default:    statementd;
endcase
```




```
casex(c)
      4'b00xx: statement1;
      :
      4'b0xxx: statementj;
      :
      default: statementd;
endcase
```

```
casez(c)
      8'b01z?11??: statement1;
      :
      8'b1100????: statementj;
      :
      default:    statementd;
endcase
```

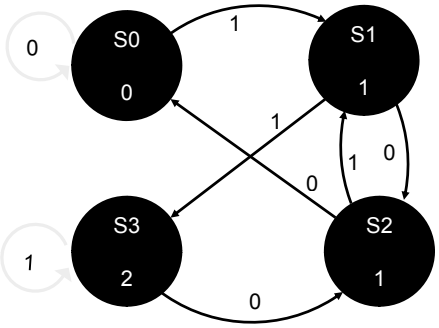

Typical Verilog Description

Example of Mealy Machine: A Lock with Password 1101

```
module Lock(Clk, X, Z)
input     Clk, X;
output    Z;
reg       Z;
parameter W1011=2'b00, W011=2'b01, W11=2'b10, W1=2'b11;
reg [1:0] Present_State, Next_State;
      always@(posedge Clk)
            Present_State = Next_State;
      always@(X or Present_State) begin
            Next_State = Present_State;
            case(Present_State)
                  W1011  : Next_State = X ? W011 : W1011;
                  W011   : Next_State = X ? W011 : W11;
                  W11    : Next_State = X ? W1   : W1011;
                  W1     : Next_State = X ? W011 : W11;
                  default: Next_State = W011;
            endcase
            case(Present_State)
                  W1     : Z = X ? 1 : 0;
                  default: Z = 0;
            endcase
      end // always
endmodule
```

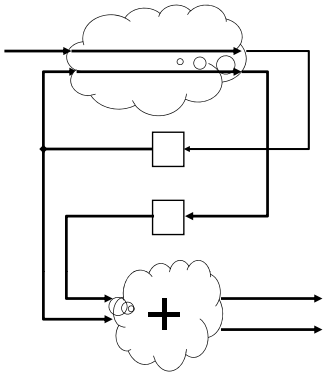

State Diagram

Example of Moore Machine: Recent Ones



State Diagram

Example of Moore Machine: Recent Ones



Typical Verilog Description

Example of Mealy Machine: A Lock with Password 1101

```
module Recent1s(Clk, X, Z)
input     Clk, X;
output    [1:0] Z;
reg       [1:0] Z;
reg [1:0] Present_State, Next_State;
      always@(posedge Clk)
            Present_State = Next_State;
      always@(X or Present_State) begin
            Next_State = Present_State;
            case(Present_State)
                  0      : Next_State = X ? 1 : 0;
                  1      : Next_State = X ? 2 : 1;
                  2      : Next_State = X ? 1 : 0;
                  3      : Next_State = X ? 2 : 1;
                  default: Next_State = 0;
            endcase
      end // always
      assign Z = Present_State[1] + Present_State[0];
endmodule
```

Moore decoder from present state to output

Algorithmic State Machine

- A special design style of general finite state machines.
- It can be designed by mapping from the flow-chart.
- It can be synthesized manually.
- High area overhead and low performance due multiplexer stack.
  1. Describe your system in pseudo code;
  2. Draw the flow-chart;
  3. Map the flow-chart to ASM chart;
  4. Map the ASM chart to the HDL or circuit.

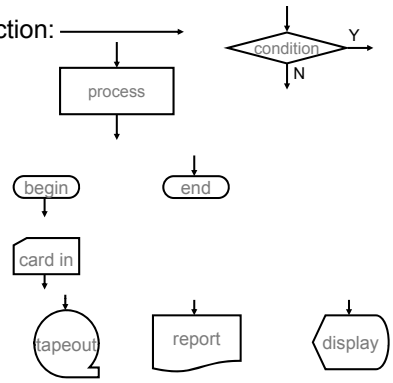
Pseudo Code

- A high-level, almost-executable description
- Used for leader's instruction, explanation of algorithms, etc.
- Pascal-like or C-like codes with some comprehensive English is preferable.
- Example:

```
1  C-sim()
2  {
3    C-frontier ← ∅;
4    For each FF, f, propagate-C-frontier(f);
5    While(C-frontier ≠ ∅){
6      Get a gate g from C-frontier that has the minimum fanout;
7      A-frontier ← g;
8      c←controlling value of g;
9      Set control(g) = FAILED;
10     Repeat
11       Select an input of gate g, i, with a don't-care X;
12       If Justify(i, c) then control(g) = SUCCESS;
13     Until control(g)=SUCCESS or no input has an X value;
14     If control(g)=FAILED then propagate-C-frontier(g);
15   }
16   Return values at FF's as the control pattern;
17 }

18 propagate-C-frontier(g)
19 {
20   For each non-FF fanout gate, h, of g
21     if h ∈{NAND, NOR, XNOR} or h does not have a don't care
22       then propagate C-frontier(h).
23     else put h into C-frontier;
24 }
```

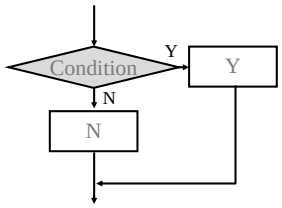

Basic Flowchart Elements

- Flow and Direction:
- Decision:
- Process:
- Terminals:
- (Card) Input:
- Specific Outputs:

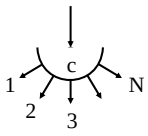


Sequence Control
if-then-else

if (C) { Y } else { N }



Sequence Control
switch

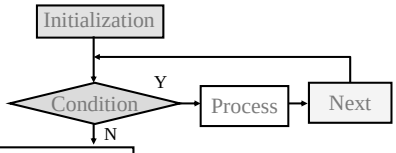



```
if (expr==const1) {P1}
else if (expr==const2) {P2}
     else if (expr==const3) {P3}
          else if (expr==const4) {P4}
               else {Pdefault}
```

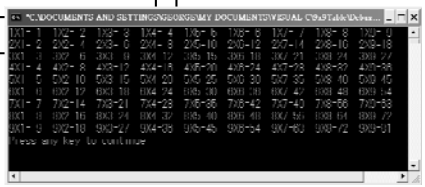
```
switch (expr) {
       case const1: {P1} break;
       case const2: {P2} break;
       case const3: {P3} break;
       case cosnt4: {P4} break;
       ...
       default:     {Pdefault}
}
```


Sequence Control
For-loop

for (I ; C ; N) { P }

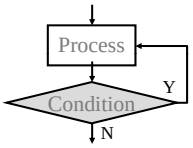



```
for(i=1;i<=9;i++) {
   for(j=1;j<=9;j++)
      printf("%dX%d=%2d\t",i,j,i*j);
   printf("\n");
}
```

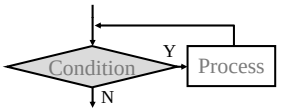


Sequence Control
Do-While-Loop and While-Do-Loop

do { P } while(C) ;

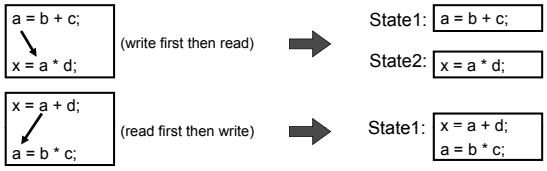


while (C) { P } ;



Map Flowchart to ASM Chart

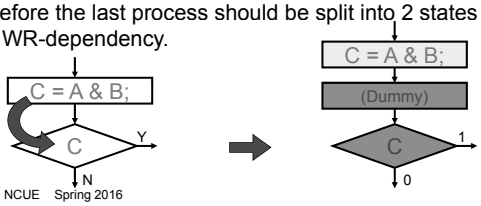
- Split Processes in flowchart into States that each state can be executed in 1 clock cycle.
- Note that a WR-dependent process should be split into two states.



Map Decision to Demultiplexer

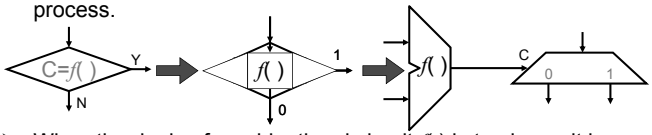
- If the condition is a 1-bit variable, directly map the decision to a demultiplexer that is a combinational gate (i.e., it takes a delay less than the clock cycle).

- The clock state of a decision is belong to the last process, therefore the last process should be split into 2 states if there is a WR-dependency.

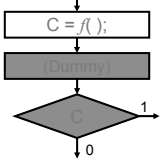


Map Decision with Expression to Diamond

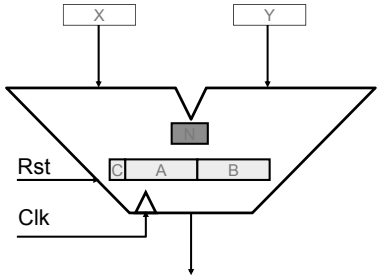
- If the condition is an expression instead of a direct input, map the decision to a a diamond.
- A diamond is composed of a demultiplexer and an extra process.



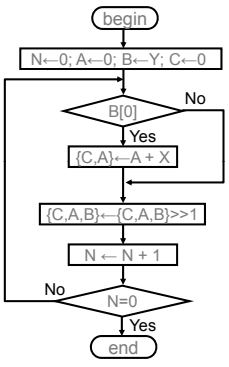
- When the dealy of combinational circuit *f*( ) is too large, it is better to split it to two states with a demultiplexer.

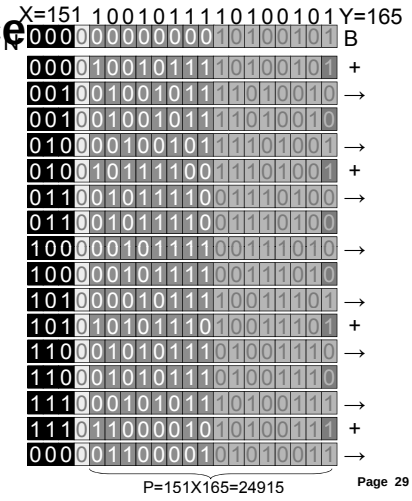


Example
8-bit Multiplier



For Instance





Example of Verification using C

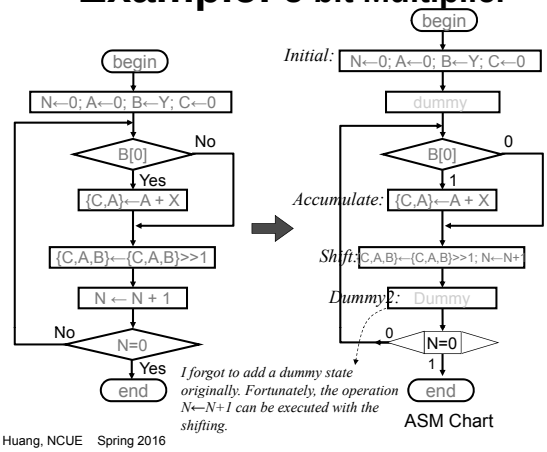
```
#include <stdio.h>
main()
{
  unsigned char X, Y, C, A, B, N, Verified;
  int      i, P, Q, Acc, S;

  Verified = 1;
  for(i=0;i<(1<<16);i++){
    X=i/(1<<8);
    Y=i%(1<<8);
    P=X*Y;        // Golden Circuit

    // Algorithm Started Here
    N=0; A=0; B=Y; C=0; // Initialization
    do {
      // Accumulate
      if(B%2) {Acc = A + C*256 + X; A=Acc%256; C=(Acc>>8)%2;}
      // Shift CAB
      S = C*65536 + A*256 + B; S = S >> 1; B=S%256; A=(S>>8)%256; C=(S>>16)%2;
      N++;
    } while(N!=8);
    Q=(A<<8)+B;
    if(Q!=P) {
      Verified=0;
      printf("Error when X=%d, Y=%d, Q=%d\n", X, Y, Q);
      break;
    }
  }
  if(Verified) printf("The circuit is exhaustively functionally verified.\n");
}
```

Example: Verification using Assembly Language

```
BEGIN:      MOV   N, #States
            CLR   A
            CLR   C
            MOV   B, X
LOOP:       JE    SHIFT
            ADC   Y
SHIFT:      SHR   CAB
            DEC   N
            JNZ   LOOP
            END
```

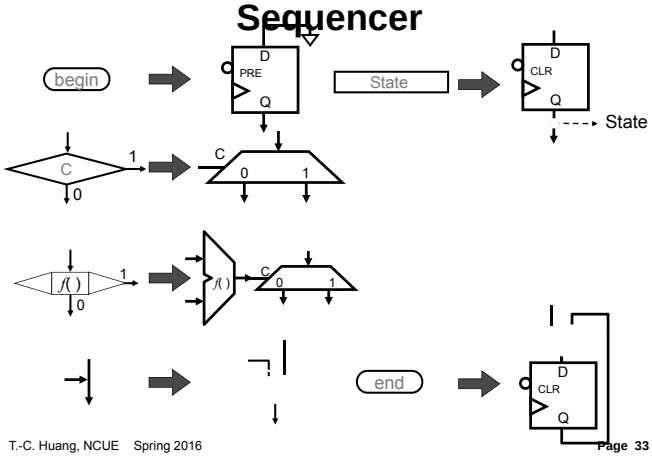
C and Assembly codes can be the fast logic simulator.

Example: 8-bit Multiplier

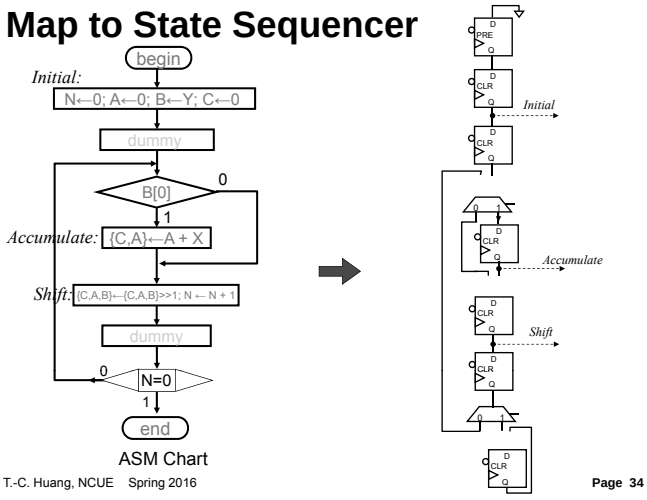


I forgot to add a dummy state originally. Fortunately, the operation N←N+1 can be executed with the shifting.

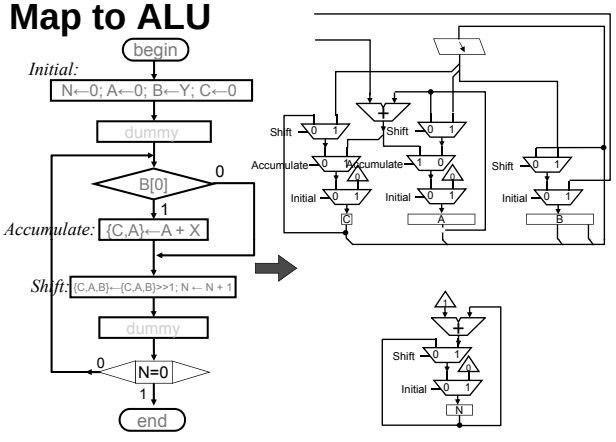
Map ASM Elements to Sequencer

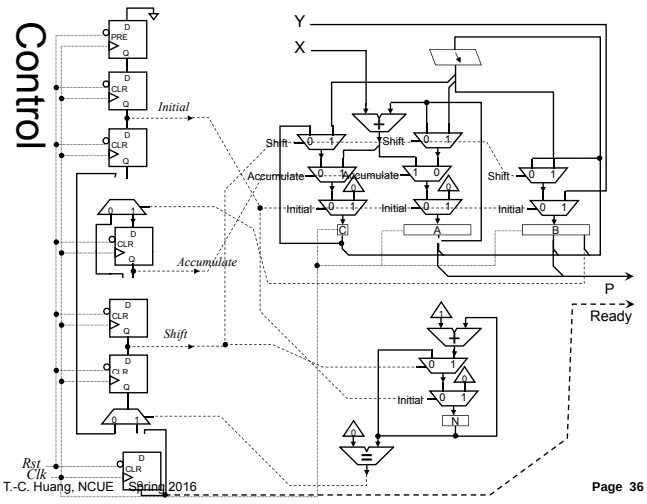


Map to State Sequencer



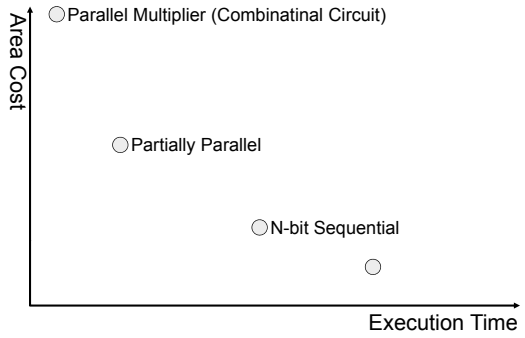
Map to ALU





N-Bit Multiplier

assign P = A * B;



Counter/Timer Based State Machine

Counters in General Sense

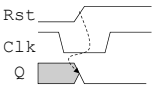
- **A general counter is an FSM that NS depends only on PS.**
- **Binary Counter**
  - Ripple Counter: Toggle FFs triggered by Input.
  - Parallel counter takes much space while FSM design takes much time.
- **Gray Counter**
  - Minimum transition count for low power, low noise application
- **Shifting Counter**
  - Incomplete coding.
  - Almost zero propagation time in Huffman model of sequential circuits → Fastest
  - Examples:
    - **Ring Counter**
      - 1-hot code for some direct or fast application
    - **Johnson Counter**
      - Fastest and with 2N states
    - **LFSR (Random) Counter**
      - Random number generation, random test and compression
      - Decoder required for use as counter.

Example: Counters

- Basic Up-Counting Counter:

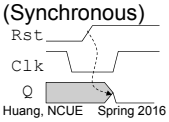
```
module DFF(Clk, Q);
input  Clk;
output [7:0] Q;
reg    [7:0] Q;
    always@(posedge Clk)
      Q=Q+1;
endmodule
```

- Resettable Counter: (Asynchronous)




```
module DFF(Rst,Clk, Q);
input  Rst, Clk;
output [15:0] Q;
reg    [15:0] Q;
    always@(posedge Clk or posedge Rst)
      if(Rst) Q=15'b0;
      else    Q=Q+1;
endmodule
```

- Resettable Counter: (Synchronous)




```
module DFF(Rst,Clk, Q);
input  Rst, Clk;
output [31:0] Q;
reg    [31:0] Q;
    always@(posedge Clk)
      if(Rst) Q=31'b0;
      else    Q=Q+1;
endmodule
```


Example: Counters

- Basic Down-Counting Counter:

```
module DFF(Clk, Q);
input  Clk;
output [7:0] Q;
reg    [7:0] Q;
    always@(posedge Clk)
      Q=Q-1;
endmodule
```

- Loadable Counter:

```
module DFF(Load, Clk, D, Q);
input  Load, Clk, D;
output [15:0] Q;
reg    [15:0] Q;
    always@(posedge Clk)
      if(Load) Q=D;
      else    Q=Q+1;
endmodule
```

- Modulo Counter:

```
module DFF(Rst,Clk, Q);
input  Rst, Clk;
output [7:0] Q;
reg    [7:0] Q;
    always@(posedge Clk)
      if(Rst) Q=7'b0;
      else   if(Q==99) Q=7'b0; //%100
             else Q=Q+1;
endmodule
```


Counter/Timer Based State Machine

Combination of Counter and FSM