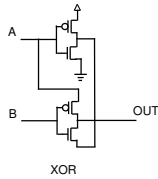


## Additions...

Taken from:  
<http://www.ap-asic.org/2000/proceedings/2-2.pdf>  
 Which has 16 XNOR or XOR implementations.



There are lots of ways to implement XOR in transistors, but the most popular way is to just make one out of NAND's and NOR's.  
 For more information, check out:  
 For transistor stuff:  
<http://www.ece.mtu.edu/ee/faculty/jitan/ece2171/notes/lec06.pdf>  
 for semiconductor physics:  
<http://britneyspears.ac/physics/basics/basics.htm>  
 For XOR and XNOR using NAND's and NOR's:  
<http://www.ecs.umass.edu/ece/tessier/courses/221/lecture/lect11-engin112.pdf>

## Example Of State Machine Design

Design a FSM (state diagram) for the following problem definition:

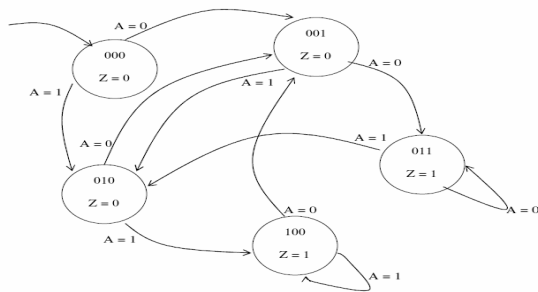
input: A

output: Z

If A has the same input for 2 or more consecutive clock cycles Z = 1.

Otherwise, Z = 0.

## Example Of State Machine Design



## Example Of State Machine Design

| Current state  |                |                |   | Next state      |                 |                 |   |  |
|----------------|----------------|----------------|---|-----------------|-----------------|-----------------|---|--|
| Q <sub>2</sub> | Q <sub>1</sub> | Q <sub>0</sub> | A | Q <sub>2n</sub> | Q <sub>1n</sub> | Q <sub>0n</sub> | Z |  |
| 0              | 0              | 0              | 0 | 0               | 0               | 1               | 0 |  |
| 0              | 0              | 0              | 1 | 0               | 1               | 0               | 0 |  |
| 0              | 0              | 1              | 0 | 0               | 1               | 1               | 0 |  |
| 0              | 0              | 1              | 1 | 0               | 1               | 0               | 0 |  |
| 0              | 1              | 0              | 0 | 0               | 0               | 1               | 0 |  |
| 0              | 1              | 0              | 1 | 1               | 0               | 0               | 0 |  |
| 0              | 1              | 1              | 0 | 0               | 1               | 1               | 1 |  |
| 0              | 1              | 1              | 1 | 0               | 1               | 0               | 0 |  |
| 1              | 0              | 0              | 0 | 0               | 0               | 1               | 1 |  |
| 1              | 0              | 0              | 1 | 1               | 0               | 0               | 0 |  |
| 1              | 0              | 1              | 0 | d               | d               | d               | d |  |
| 1              | 0              | 1              | 1 | d               | d               | d               | d |  |
| 1              | 1              | 0              | 0 | d               | d               | d               | d |  |
| 1              | 1              | 0              | 1 | d               | d               | d               | d |  |
| 1              | 1              | 1              | 0 | d               | d               | d               | d |  |
| 1              | 1              | 1              | 1 | d               | d               | d               | d |  |

## Example Of State Machine Design

| Q <sub>2</sub> Q <sub>1</sub> \ Q <sub>0</sub> A | 00 | 01 | 11 | 10 |
|--|----|----|----|----|
| 00   | 0  | 0  | 0  | 0  |
| 01   | 0  | 1  | 0  | 0  |
| 11   | d  | d  | d  | d  |
| 10   | 0  | 1  | d  | d  |

$$Q_{2n} = Q_2A + Q_1Q_0'A$$

| Q <sub>2</sub> Q <sub>1</sub> \ Q <sub>0</sub> A | 00 | 01 | 11 | 10 |
|--|----|----|----|----|
| 00   | 0  | 1  | 1  | 1  |
| 01   | 0  | 0  | 1  | 1  |
| 11   | d  | d  | d  | d  |
| 10   | 0  | 0  | d  | d  |

$$Q_{1n} = Q_0 + Q_2'Q_1'A$$

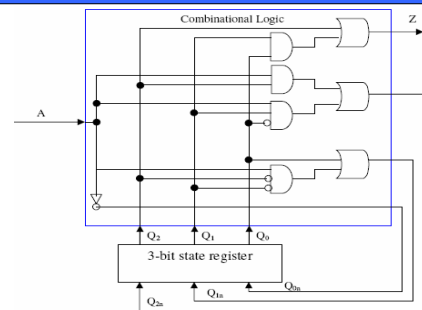
| Q <sub>2</sub> Q <sub>1</sub> \ Q <sub>0</sub> A | 00 | 01 | 11 | 10 |
|--|----|----|----|----|
| 00   | 1  | 0  | 0  | 1  |
| 01   | 1  | 0  | 0  | 1  |
| 11   | d  | d  | d  | d  |
| 10   | 1  | 0  | d  | d  |

$$Q_{0n} = A'$$

| Q <sub>2</sub> \ Q <sub>1</sub> Q <sub>0</sub> | 00 | 01 | 11 | 10 |
|--|----|----|----|----|
| 0  | 0  | 0  | 1  | 0  |
| 1  | 1  | d  | d  | d  |

$$Z = Q_2 + Q_1Q_0$$

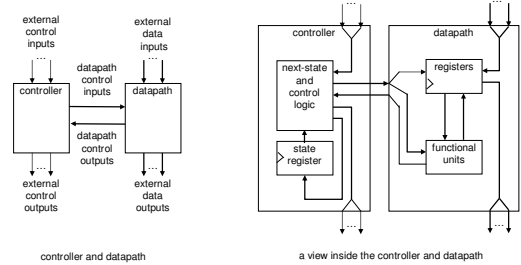
## Example Of State Machine Design



# The Next Step...

So far, our state machines have just had outputs of either one or zero depending on the state.  
 How do you make a state machine do useful and meaningful calculations?  
**Add a datapath!**

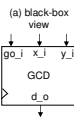
# Custom single-purpose processor basic model



# Example: greatest common divisor

- First create algorithm

*The Greatest common divisor algorithm takes two numbers and finds the largest integer that will divide into both of them without a remainder.*



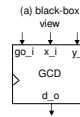
# Example: greatest common divisor

- First create algorithm

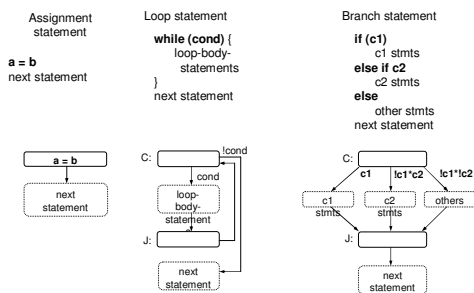
(b) desired functionality

```

0: int x, y;
1: while (1) {
2:   while (!go_i);
3:   x = x_i;
4:   y = y_i;
5:   while (x != y) {
6:     if (x < y)
7:       y = y - x;
8:     else
9:       x = x - y;
9:   d_o = x;
}
    
```



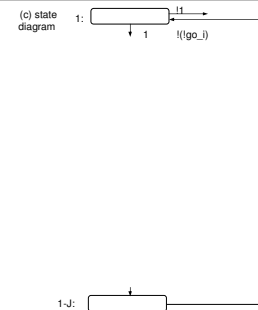
# State diagram templates



Use these templates to convert your algorithm into a Finite State Machine with Data

# Example: greatest common divisor

- First create algorithm
- Convert algorithm to "complex" state machine



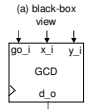
(a) black-box view

```

0: int x, y;
1: while (1) {
2:   while (!go_i);
3:   x = x_i;
4:   y = y_i;
5:   while (x != y) {
6:     if (x < y)
7:       y = y - x;
8:     else
9:       x = x - y;
9:   d_o = x;
}
    
```

## Example: greatest common divisor

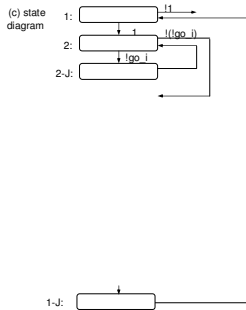
- First create algorithm
- Convert algorithm to “complex” state machine



(b) desired functionality

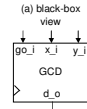
```

0: int x, y;
1: while (1) {
2:   while (!go);
3:   x = x;
4:   y = y;
5:   while (x != y) {
6:     if (x < y)
7:       y = y - x;
8:     else
9:       x = x - y;
10:  }
11:  d_o = x;
}
    
```



## Example: greatest common divisor

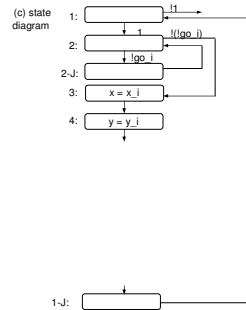
- First create algorithm
- Convert algorithm to “complex” state machine



(b) desired functionality

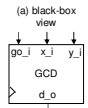
```

0: int x, y;
1: while (1) {
2:   while (!go);
3:   x = x;
4:   y = y;
5:   while (x != y) {
6:     if (x < y)
7:       y = y - x;
8:     else
9:       x = x - y;
10:  }
11:  d_o = x;
}
    
```



## Example: greatest common divisor

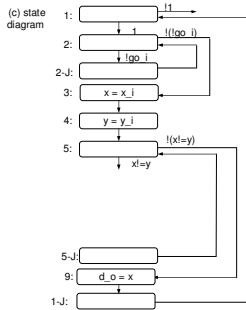
- First create algorithm
- Convert algorithm to “complex” state machine



(b) desired functionality

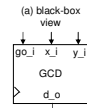
```

0: int x, y;
1: while (1) {
2:   while (!go);
3:   x = x;
4:   y = y;
5:   while (x != y) {
6:     if (x < y)
7:       y = y - x;
8:     else
9:       x = x - y;
10:  }
11:  d_o = x;
}
    
```



## Example: greatest common divisor

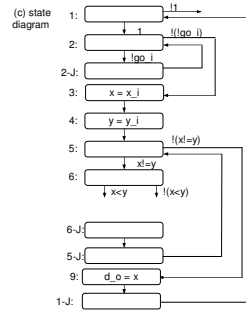
- First create algorithm
- Convert algorithm to “complex” state machine



(b) desired functionality

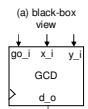
```

0: int x, y;
1: while (1) {
2:   while (!go);
3:   x = x;
4:   y = y;
5:   while (x != y) {
6:     if (x < y)
7:       y = y - x;
8:     else
9:       x = x - y;
10:  }
11:  d_o = x;
}
    
```



## Example: greatest common divisor

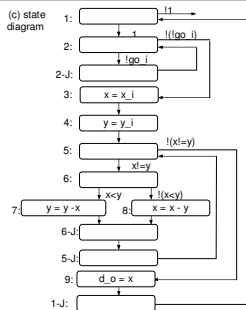
- First create algorithm
- Convert algorithm to “complex” state machine



(b) desired functionality

```

0: int x, y;
1: while (1) {
2:   while (!go);
3:   x = x;
4:   y = y;
5:   while (x != y) {
6:     if (x < y)
7:       y = y - x;
8:     else
9:       x = x - y;
10:  }
11:  d_o = x;
}
    
```



## Creating the datapath

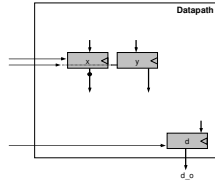
1. Create a register for any variable
2. Create a functional unit for each arithmetic operation
3. Connect the ports, registers and functional units
  1. Based on reads and writes
  2. Use multiplexors for multiple sources
4. Create unique identifier
  - for each datapath component control input and output

## Creating the datapath

1. Create a register for any variable

```

0: int x, y;
1: while (!go_0) {
2:   x = x;
3:   y = y;
4:   while (x != y) {
5:     if (x < y)
6:       y = y - x;
7:     else
8:       x = x - y;
9:   }
10:  d_o = x;
}
    
```

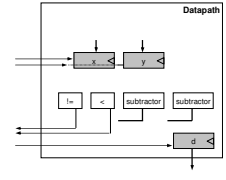


## Creating the datapath

1. Create a register for any variable
2. Create a functional unit for each arithmetic operation

```

0: int x, y;
1: while (!go_0) {
2:   x = x;
3:   y = y;
4:   while (x != y) {
5:     if (x < y)
6:       y = y - x;
7:     else
8:       x = x - y;
9:   }
10:  d_o = x;
}
    
```



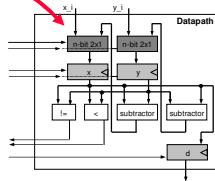
## Creating the datapath

3. Connect the ports, registers and functional units

- Based on reads and writes
- Use multiplexers for multiple sources

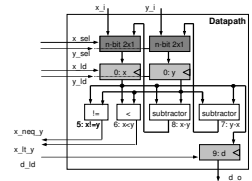
```

0: int x, y;
1: while (!go_0) {
2:   x = x;
3:   y = y;
4:   while (x != y) {
5:     if (x < y)
6:       y = y - x;
7:     else
8:       x = x - y;
9:   }
10:  d_o = x;
}
    
```

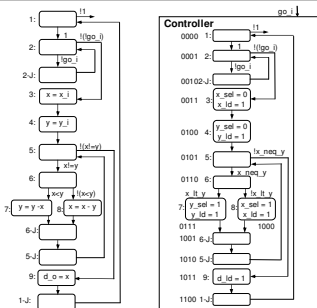


## Creating the datapath

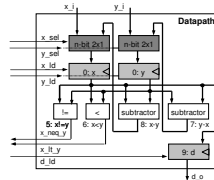
- Create a register for any declared variable
- Create a functional unit for each arithmetic operation
- Connect the ports, registers and functional units
  - Based on reads and writes
  - Use multiplexers for multiple sources
- Create unique identifier
  - for each datapath component control input and output



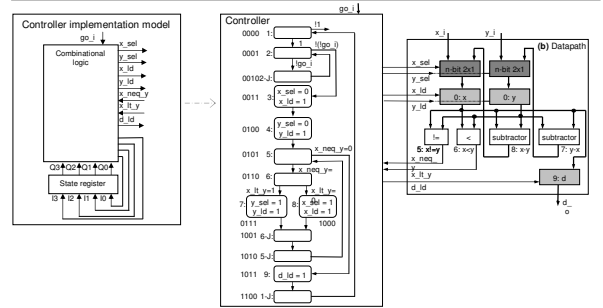
## Creating the controller's FSM



- Same structure as FSMD
- Replace complex actions/conditions with datapath configurations



## Splitting into a controller and datapath



## Controller state table for the GCD example

| Inputs |    |    |    |       |     |    | Outputs |    |    |    |       |       |      |      |      |
|--------|----|----|----|-------|-----|----|---------|----|----|----|-------|-------|------|------|------|
| Q3     | Q2 | Q1 | Q0 | x_req | x_R | go | I3      | I2 | I1 | I0 | x_sel | y_sel | x_id | y_id | d_id |
| 0      | 0  | 0  | 0  | 0     | 0   | 0  | 0       | 0  | 0  | 1  | X     | X     | 0    | 0    | 0    |
| 0      | 0  | 0  | 1  | 0     | 0   | 0  | 0       | 0  | 1  | 0  | X     | X     | 0    | 0    | 0    |
| 0      | 0  | 0  | 1  | 0     | 1   | 0  | 0       | 0  | 1  | 1  | X     | X     | 0    | 0    | 0    |
| 0      | 0  | 1  | 0  | 0     | 0   | 0  | 0       | 0  | 1  | 0  | X     | X     | 0    | 0    | 0    |
| 0      | 0  | 1  | 1  | 0     | 0   | 0  | 0       | 1  | 0  | 0  | 0     | X     | 1    | 0    | 0    |
| 0      | 1  | 0  | 0  | 0     | 0   | 0  | 0       | 1  | 0  | 1  | X     | 0     | 0    | 1    | 0    |
| 0      | 1  | 0  | 1  | 0     | 0   | 0  | 1       | 0  | 1  | 1  | X     | X     | 0    | 0    | 0    |
| 0      | 1  | 0  | 1  | 1     | 0   | 0  | 0       | 1  | 1  | 0  | X     | X     | 0    | 0    | 0    |
| 0      | 1  | 1  | 0  | 0     | 0   | 0  | 1       | 0  | 0  | 0  | X     | X     | 0    | 0    | 0    |
| 0      | 1  | 1  | 0  | 0     | 1   | 0  | 0       | 1  | 1  | 1  | X     | X     | 0    | 0    | 0    |
| 0      | 1  | 1  | 1  | 0     | 0   | 0  | 1       | 0  | 0  | 1  | X     | 1     | 0    | 1    | 0    |
| 1      | 0  | 0  | 0  | 0     | 0   | 0  | 1       | 0  | 0  | 1  | 1     | X     | 1    | 0    | 0    |
| 1      | 0  | 0  | 1  | 0     | 0   | 0  | 1       | 0  | 1  | 0  | X     | X     | 0    | 0    | 0    |
| 1      | 0  | 1  | 0  | 0     | 0   | 0  | 0       | 1  | 0  | 1  | X     | X     | 0    | 0    | 0    |
| 1      | 0  | 1  | 1  | 0     | 0   | 0  | 1       | 1  | 0  | 0  | X     | X     | 0    | 0    | 1    |
| 1      | 1  | 0  | 0  | 0     | 0   | 0  | 0       | 0  | 0  | 0  | X     | X     | 0    | 0    | 0    |
| 1      | 1  | 0  | 1  | 0     | 0   | 0  | 0       | 0  | 0  | 0  | X     | X     | 0    | 0    | 0    |
| 1      | 1  | 1  | 0  | 0     | 0   | 0  | 0       | 0  | 0  | 0  | X     | X     | 0    | 0    | 0    |
| 1      | 1  | 1  | 1  | 0     | 0   | 0  | 0       | 0  | 0  | 0  | X     | X     | 0    | 0    | 0    |

## To Do before next Thursday

- Look for Homework 2 on the web-page.
- Finish reading through chapter 2 if you haven't already.