More Finite State Machines: Multiplication as an Example
Learning Objectives

- Design a Finite State Machine that controls a datapath
- Use Register Transfer Language to describe how state is modified each clock cycle
Step 1: Understand your problem
We can perform binary multiplication using the same algorithm as decimal multiplication

\[ 1101 \ (13) \]
\[ \times \ 1010 \ (10) \]
Multiplying two unsigned $N$-bit integers produces at most a $2N$-bit integer

\[
\begin{array}{c}
1111 \ (15) \\
\times \ 1111 \ (15) \\
\hline
1111 \\
1111 \\
1111 \\
1111 \\
\hline
11100001 \ (225)
\end{array}
\]
Multiplication can be described as a series of shift and add operations

\[
\begin{align*}
00001101 \ (13) \ & \ A \\
\times \ 1010 \ (10) & \\
\hline
0000 & \\
00011010 & \ A \ll 1 \\
0000 & \\
01101000 & \ A \ll 3 \\
\hline
10000010 \ (130) & \ P = (A \ll 1) + (A \ll 3)
\end{align*}
\]
We determine whether to add each iteration by one-bit from the multiplicand

\[
\begin{array}{c}
1101 \ (13) \\
\times 1010 \ (10) \\
\hline
0000 \\
1101 \\
0000 \\
0000 \\
1101 \\
\hline
10000010 \ (130)
\end{array}
\]
We can use bitshifting and bitmasks to extract one bit from the multiplicand for control

\[
\begin{array}{c}
1101 \ (13) \\
\times 1010 \ (10)
\end{array}
\]
You try another example

\[ 1001 \times 0011 \]
Write some C code that can perform the shift-add multiply algorithm

```
1101 (13) A
× 1010 (10) B
0000
1101
0000
1101
---
10000010 (130) P
```

```c
int shift_add_multiply(unsigned short X, unsigned short Y) {
    unsigned int A = (unsigned int) X;
    unsigned short B = Y;
    unsigned int P = 0;
    return P;
}
```
Step 2: Understand the datapath
Identify needed state components from variables (what state is stored?)

```c
int shift_add_multiply(...) {
    unsigned int A = (unsigned int) X;
    unsigned short B = Y;
    unsigned int P = 0;

    return P;
}
```
Identify how we modify that state

A (2N-bit reg)  EN

B (N-bit reg)  EN

P (2N-bit reg)  EN
Define specification for control FSM

A (2N-bit reg)  EN
D
Q

B (N-bit reg)  EN
D
Q

<<1

ALU

Y

A
B
OP

ALU

P (2N-bit reg)  EN
D
Q

X

0 1

>>1
Control FSM has 3 inputs and 7 outputs

- B is zero (Z)
- B_Q[0] is 1 (B[0])
- START

- Load A (LA)
- Shift A (SA)
- Load B (LB)
- Shift B (SB)
- Add when 1/Zero when 0 (OP)
- Load P (LP)
- DONE
Control and Datapath

ALU

A (2N-bit reg)

B (N-bit reg)

P (2N-bit reg)

Control FSM

START

DONE

=0?

D Q

EN

<<1

>>1

X

Y

A (2N-bit reg)

B (N-bit reg)

P (2N-bit reg)

EN

EN

Q

Q

OP

A

B
Step 3: Design your FSM (don’t forget timing!)
Register transfers show source state elements, operations, and destination state elements

\[
\begin{align*}
A &= X \\
B &= Y \\
P &= 0 \\
A &= A \ll 1 \\
B &= B \gg 1 \\
P &= P + A
\end{align*}
\]
Translate register transfers into control signals

### Control FSM

- **X**
- **Y**
- **ALU**
- **OP**
- **A (2N-bit reg)**
  - **D**
  - **Q**
  - **EN**
- **B (N-bit reg)**
  - **D**
  - **Q**
  - **EN**
- **P (2N-bit reg)**
  - **D**
  - **Q**
  - **EN**

- **START**
- **DONE**

- **A**
- **B**
- **OP**

- **D**
- **Q**

- **<<1**

- **>>1**

- **B_Q[0]**

- **=0?**