Recap from last lecture

- Programmable logic
  - Register Transfer Level (RTL) design
  - Algorithmic State Machine ASM

- Today’s lecture
  - Algorithmic State Machine (ASM)
    - Datapath design
    - Control design
  - Example: sequential multiplier design
Verilog Hints

- **Compilation errors**
  - **READ** error messages !!
  - Watch for the first error message, other errors often depend on the 1st one

- **Logical errors, eg. Lab5 (TLC) code:**
  - You intended to have:
    » HG = 1 if state is (S0 OR S1 OR S2)
  - You wrote:
    » assign HG = (state == S0 | S1 | S2); // compiles OK, but is this correct?
    » NO: check the value of ( S0 | S1 | S2 )
    » Eg. If S0 = 000, S1 = 001, S2 = 010, then S0 | S1 | S2 = ........
    » Is this what you wanted? When is HG=1 ?
  - Correct code (do you see the difference?):
    » assign HG = (state == S0) | (state == S1) | (state == S2);

Separation of Control and Data

- **Data processing path (Datapath)**
  - Processes data, discrete elements of information
  - Part of the system, which operates on data
    » Registers, adders, comparators, ALU, etc.
    » Well structured, design well understood
    » Common to many systems, implemented with standard components

- **Control logic**
  - Part of the system which controls datapath
    » Provides command signals
    » Much smaller than datapath
    » Unique to every ckt
    » Requires custom design

- **Control and datapath:**
Algorithmic State Machines (ASM)

- **Control logic**
  - Controls sequence of operations in the datapath
    - E.g., triggers transfer of register value followed by addition
  - Sequential circuit with different control states

- **Representation of control logic:**
  - “Algorithmic State Machine” (ASM)

- **ASM flow chart describes sequence of events**
  - Contains “state box,” “decision box,” and “conditional box”

![ASM Flow Chart]

---

ASM Block

- **State boxes form state diagram:**

![State Diagram]

- All operations between states happen in synch and in one clock cycle
Design Example – binary multiplier

Example:
- $10111 \times 10011$
  - $B = 10111 = 23_{10}$ multiplicand
  - $Q = 10011 = 19_{10}$ multiplier

\[
\begin{array}{c}
10111 \\
10011 \\
10111 \\
10111 \\
00000 \\
00000 \\
10111 \\
\hline
0110110101 = 437_{10}
\end{array}
\]

Multiplier Operation

- Design of a sequential multiplier
- Block diagram:
  - How does it work?
Multiplier Operation

- **Initialization:**
  - Load Multiplicand into \( B \)
  - Load Multiplier into \( Q \)
  - Load number of bits \( n \) into \( P \)

- **Start:**
  - Multiplication begins when \( S=1 \)

- **Each step:**
  - If \( Q_0 \) (LSB in \( Q \)) is 1 then \( A \leftarrow A + B \)
  - Shift right \( CAQ, C \leftarrow 0 \)

- **Control logic activates all functions at the right time**

Multiplier ASM

- **ASM representation of multiplier?**
Multiplier Control Operation

- **Example:**
  - $10111 \times 10011$
  - $B = 10111$
  - $Q = 10011$

- **Control steps:**
  - **Start**
    - $S = 1$
  - **Initialization**
    - $A \leftarrow 0$
    - $C \leftarrow 0$
    - $P \leftarrow 5$
  - **Multiplication**

### Table: Multiplier Control Operation

<table>
<thead>
<tr>
<th>Control Steps</th>
<th>C</th>
<th>A</th>
<th>Q</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>After initialization</td>
<td>0</td>
<td>00001</td>
<td>10011</td>
<td>5</td>
</tr>
<tr>
<td>dec P: $Q = 0$; add B</td>
<td>10111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1st partial product</td>
<td>0</td>
<td>10111</td>
<td>10111</td>
<td>4</td>
</tr>
<tr>
<td>shift right CAQ</td>
<td>0</td>
<td>01011</td>
<td>10101</td>
<td></td>
</tr>
<tr>
<td>dec P: $Q = 0$; add B</td>
<td>10111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd partial product</td>
<td>1</td>
<td>00010</td>
<td>10101</td>
<td>3</td>
</tr>
<tr>
<td>shift right CAQ</td>
<td>0</td>
<td>10001</td>
<td>01100</td>
<td></td>
</tr>
<tr>
<td>dec P: $Q = 0$; no add</td>
<td>no add</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3rd partial product</td>
<td>0</td>
<td>10001</td>
<td>01100</td>
<td>2</td>
</tr>
<tr>
<td>shift right CAQ</td>
<td>0</td>
<td>01000</td>
<td>10110</td>
<td></td>
</tr>
<tr>
<td>dec P: $Q = 0$; no add</td>
<td>no add</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4th partial product</td>
<td>0</td>
<td>01000</td>
<td>10110</td>
<td>1</td>
</tr>
<tr>
<td>shift right CAQ</td>
<td>0</td>
<td>00100</td>
<td>01011</td>
<td></td>
</tr>
<tr>
<td>dec P: $Q = 0$; add B</td>
<td>10111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5th partial product</td>
<td>0</td>
<td>11011</td>
<td>10110</td>
<td>0</td>
</tr>
<tr>
<td>shift right CAQ</td>
<td>0</td>
<td>01101</td>
<td>10101</td>
<td></td>
</tr>
</tbody>
</table>

$P = 0$ stops multiplier

Result = 0110110101
Multiplier State Diagram

- State diagram representation of ASM

- RTL operations:
  - $T_0$: initial state
  - $T_1$: $A \leftarrow 0, C \leftarrow 0, P \leftarrow 0$
  - $T_2$: $P \leftarrow P - 1$
    if $(Q_0=1)$ then
    $(A \leftarrow A+B, C \leftarrow Cout)$
  - $T_3$: shift right CAO, $C \leftarrow 0$

Control Logic Design

- Control logic for multiplier:

- $L$ denotes if addition is performed
- $T_0 - T_3$ control the operations according to RTL specification

- Control logic design
  - State assignment: binary
    - Conventional combinatorial circuit design
  - State assignment: one-hot (only one bit is high)
    - Direct translation from state diagram
Multiplier Control Logic

- Binary coded state assignment

<table>
<thead>
<tr>
<th>Current state</th>
<th>Input</th>
<th>Next state</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G_1 )</td>
<td>( G_2 )</td>
<td>( S )</td>
<td>( Z )</td>
</tr>
<tr>
<td>0 0</td>
<td>0</td>
<td>X</td>
<td>0 0</td>
</tr>
<tr>
<td>0 0</td>
<td>1</td>
<td>X</td>
<td>0 1</td>
</tr>
<tr>
<td>0 1</td>
<td>X</td>
<td>X</td>
<td>1 0</td>
</tr>
<tr>
<td>1 0</td>
<td>X</td>
<td>X</td>
<td>1 1</td>
</tr>
<tr>
<td>1 1</td>
<td>X</td>
<td>0</td>
<td>1 0</td>
</tr>
<tr>
<td>1 1</td>
<td>X</td>
<td>1</td>
<td>0 0</td>
</tr>
</tbody>
</table>

- State transition 01 → 10 and 01 → 11 independent of input
- Output is basically decoding of binary state coding

- D flip-flop input equations:
  - \( D_{G_1} = T_1 + T_2 + T_3Z' \)
  - \( D_{G_0} = T_0S + T_2 \)
Multiplier Control Logic

- Binary coding of states can yield complex circuits
  - Multiplier example works nicely because of in-order transition of state

- Alternate approach: one-hot coding
  - One flip-flop per state
  - Current state indicated by single 1
  - State transitions achieved by “passing the 1”

- Direct translation of state diagram!

- Incoming arrows on state diagram determine input logic
  - $D_{T0} = T_0S' + T_3Z$;  $D_{T1} = T_0S$;  $D_{T2} = T_1 + T_3Z'$;  $D_{T3}=T_2$

Multiplier Control Logic

- One-hot coded states
  - One FF per state

![Multiplier Control Logic Diagram](image-url)
Automated Design

- RTL-based design is automated (synthesis & verification tools)

Homework

- Read Mano
  - 8-8