Notes - Unit 7

Introduction to Digital System Design

Digital System Model

- FSM + Datapath Circuit:

   ![Digital System Model Diagram]

Example: Car Lot Counter

If $A = 1 \rightarrow$ No light received (car obstructing LED A)
If $B = 1 \rightarrow$ No light received (car obstructing LED B)

If car enters the lot, the following sequence $(A|B)$ must be followed:
$00 \rightarrow 10 \rightarrow 11 \rightarrow 01 \rightarrow 00$

If car leaves the lot, the following sequence $(A|B)$ must be followed:
$00 \rightarrow 01 \rightarrow 11 \rightarrow 10 \rightarrow 00$

A car might stay in a state for many cycles since the car speed is very large compared to that of the clock frequency.

Digital System (FSM + Datapath circuit)

- Usually, when ‘resetn’ (asynchronous clear) and ‘clock’ are not drawn, they are implied.
- **Finite State Machine (FSM):**

  ![Finite State Machine Diagram]

- **Algorithmic State Machine (ASM) chart:**

  ![Algorithmic State Machine Chart]
EXAMPLE: ACCUMULATOR

DIGITAL SYSTEM (FSM + Datapath circuit)
- sclr: Synchronous clear. If \( E = '1' \) and \( sclr = '1' \), then the output bits of the registers are set to zero.

Finite State Machine (FSM):
\[
E | restart/Ei | sclr
\]

Algorithmic State Machine (ASM):
**EXAMPLE: 7-SEGMENT SERIALIZER**

**DIGITAL SYSTEM** (FSM + Datapath circuit)
- Most FPGA Development board have a number of 7-segment displays (e.g., 4, 8). However, only one can be used at a time.
- If we want to display four digits (inputs A, B, C, D), we can design a serializer that will only show one digit at a time on the 7-segment displays.
- Since only one 7-segment display can be used at a time, we need to serialize the four BCD outputs. In order for each digit to appear bright and continuously illuminated, each digit is illuminated for 1 ms every 4 ms (i.e., a digit is un-illuminated for 3 ms and illuminated for 1 ms). This is taken care of by feeding the output ‘z’ of the counter to 0.001 s to the enable input of the FSM. This way, state transitions only occur each 0.001 s.
- In the figure, the enable signals for the four 7-segment displays are active low (this is usually the case).

**Algorithmic State Machine (ASM) chart:** This is a Moore-type FSM.

---

### Diagram Description

- **Inputs:** A, B, C, D
- **Outputs:** 7-segment display
- **Counter (0.001s):**
- **2-to-4 decoder:**
- **Finite State Machine:**
- **State Transitions:**
  - **S1:** resetn = 0
  - **S2:** s = 01
  - **S3:** s = 10
  - **S4:** s = 11
- **Enable Signals:** active low

---

**Diagram:**

- A → B → C → D → 4 (BCD to 7 segments decoder)
- 7-segment display
- Counter
- 2-to-4 decoder
- Finite State Machine
- State transitions
- Enable signals active low

---
EXAMPLE: BIT-COUNTING CIRCUIT

SEQUENTIAL ALGORITHM

\[
C \leftarrow 0 \\
\text{while } A \neq 0 \\
\quad \text{if } a_0 = 1 \text{ then} \\
\quad \quad C \leftarrow C + 1 \\
\quad \text{end if} \\
\quad \text{right shift } A \\
\text{end while}
\]

DIGITAL SYSTEM (FSM + Datapath circuit)
- sclr: Synchronous clear. In this case, if sclr = '1', the count is initialized to zero (here, we do not need EC to be 1).

**Algorithmic State Machine (ASM) chart:**
**EXAMPLE: SIMPLE PROCESSOR**

**DIGITAL SYSTEM** (FSM + Datapath circuit)

- **Operation**: Every time w = ‘1’, we grab the instruction from `fun` and execute it:
  - `funq = [f2][f1][f0][Ry][Rx][Rx0]`

<table>
<thead>
<tr>
<th><code>f</code></th>
<th>Operation</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Load Rx, Data</td>
<td>Rx ← Data</td>
</tr>
<tr>
<td>001</td>
<td>Move Rx, Ry</td>
<td>Rx ← Ry</td>
</tr>
<tr>
<td>010</td>
<td>Add Rx, Ry</td>
<td>Rx ← Rx + Ry</td>
</tr>
<tr>
<td>011</td>
<td>Sub Rx, Ry</td>
<td>Rx ← Rx - Ry</td>
</tr>
<tr>
<td>100</td>
<td>Not Rx</td>
<td>Rx ← NOT (Rx)</td>
</tr>
<tr>
<td>101</td>
<td>And Rx, Ry</td>
<td>Rx ← Rx AND Ry</td>
</tr>
<tr>
<td>110</td>
<td>Or Rx, Ry</td>
<td>Rx ← Rx OR Ry</td>
</tr>
<tr>
<td>111</td>
<td>Xor Rx, Ry</td>
<td>Rx ← Rx XOR Ry</td>
</tr>
</tbody>
</table>
• Control Circuit:

• Arithmetic-Logic Unit (ALU):

<table>
<thead>
<tr>
<th>op</th>
<th>Operation</th>
<th>Function</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>$y := A$</td>
<td>Transfer ‘A’</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>$y := A + 1$</td>
<td>Increment ‘A’</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>$y := A - 1$</td>
<td>Decrement ‘A’</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>$y := B$</td>
<td>Transfer ‘B’</td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>$y := B + 1$</td>
<td>Increment ‘B’</td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>$y := B - 1$</td>
<td>Decrement ‘B’</td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>$y := A + B$</td>
<td>Add ‘A’ and ‘B’</td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td>$y := A - B$</td>
<td>Subtract ‘B’ from ‘A’</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>$y := \text{not } A$</td>
<td>Complement ‘A’</td>
<td>Logic</td>
</tr>
<tr>
<td>1001</td>
<td>$y := \text{not } B$</td>
<td>Complement ‘B’</td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>$y := A \text{ AND } B$</td>
<td>AND</td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>$y := A \text{ OR } B$</td>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>$y := A \text{ NAND } B$</td>
<td>NAND</td>
<td></td>
</tr>
<tr>
<td>1101</td>
<td>$y := A \text{ NOR } B$</td>
<td>NOR</td>
<td></td>
</tr>
<tr>
<td>1110</td>
<td>$y := A \text{ XOR } B$</td>
<td>XOR</td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>$y := A \text{ XNOR } B$</td>
<td>XNOR</td>
<td></td>
</tr>
</tbody>
</table>
- Algorithmic State Machine (ASM):

```
resetn=0

S1

S2

Algorithmic State Machine (ASM):

S1
-----------------
\( \text{resetn} = 0 \)

S2
-----------------
\( E_{\text{fun}} \leftarrow 1 \)

S3a
-----------------
\( E_\text{ext}, \text{Ex} \leftarrow 1 \)
\( \text{done} \leftarrow 1 \)

S3b
-----------------
\( \text{O}_\text{G}, \text{Ex} \leftarrow 1 \)
\( \text{done} \leftarrow 1 \)

S4a
-----------------
\( E_{\text{so}} \leftarrow 1 \)
\( E_\text{A} \leftarrow 1 \)

S4b
-----------------
\( E_{\text{so}} \leftarrow 1 \)
\( \text{op} \leftarrow 011 \)
\( \text{done} \leftarrow 1 \)

S5a
-----------------
\( E_{\text{so}} \leftarrow 1 \)
\( E_\text{A} \leftarrow 1 \)

S5b
-----------------
\( E_{\text{so}} \leftarrow 1 \)
\( \text{op} \leftarrow 1000 \)
\( \text{done} \leftarrow 1 \)

S6a
-----------------
\( E_{\text{so}} \leftarrow 1 \)
\( E_\text{A} \leftarrow 1 \)

S6b
-----------------
\( E_{\text{so}} \leftarrow 1 \)
\( \text{op} \leftarrow 101 \)
\( \text{done} \leftarrow 1 \)

S7a
-----------------
\( E_{\text{so}} \leftarrow 1 \)
\( E_\text{A} \leftarrow 1 \)

S7b
-----------------
\( E_{\text{so}} \leftarrow 1 \)
\( \text{op} \leftarrow 110 \)
\( \text{done} \leftarrow 1 \)

S8a
-----------------
\( E_{\text{so}} \leftarrow 1 \)
\( E_\text{A} \leftarrow 1 \)

S8b
-----------------
\( E_{\text{so}} \leftarrow 1 \)
\( \text{op} \leftarrow 111 \)
\( \text{done} \leftarrow 1 \)
```
**EXAMPLE: ARBITER CIRCUIT**

**DIGITAL SYSTEM** (FSM + Datapath circuit)
- Three devices can request access to a certain resource at any time (example: access to a bus made of tri-state buffers, only one tri-state buffer can be enabled at a time). The FSM can only grant access to one device at a time. There should be a priority level among devices.
- If the FSM grants access to one device, one must wait until the request signal to that device is deasserted (i.e. set to zero) before granting access to a different device.

- Algorithmic State Machine (ASM) chart:
**Example: Displaying Patterns on 7-Segment Displays.**

- Different patterns are shown based on the selector 'sel' signal. Two 7-segment displays are used.
- 'stop' input: If it is asserted (stop = 1), the lights' pattern freezes.
- The input 'x' selects the rate of change (every 1.5, 1.0, 0.5, or 0.25 seconds).

\[ \text{segs}[7..0] : \quad \text{sel}[1..0] \]

\[ \begin{array}{c}
00 \\
01 \\
10 \\
11 \\
\end{array} \]

- \( \text{segs} \) : 7-segment display patterns.
- \( \text{sel} \) : Selector signal.
- \( \text{stop} \) : Stop input.
- \( \text{resetn} \) : Reset input.
- \( \text{clock} \) : Clock signal.

**Digital System (FSM + Datapath Circuit)**

- Finite State Machines (FSM) for different time intervals:
  - Counter (1.5s)
  - Counter (1.0s)
  - Counter (0.5s)
  - Counter (0.25s)

- XOR gates for selecting the appropriate pattern based on the 'sel' signal.
- Decoder for converting the FSM output to 7-segment display patterns.
- Buffers for driving the displays.

- On the NEXYS4, only one 7-segment display can be used at a time.

\[ x = 00 \rightarrow \text{Lights change every 1.5 s} \]
\[ x = 01 \rightarrow \text{Lights change every 1.0 s} \]
\[ x = 10 \rightarrow \text{Lights change every 0.5 s} \]
\[ x = 11 \rightarrow \text{Lights change every 0.25 s} \]
- Algorithmic State Machine (ASM) chart:

![Algorithmic State Machine (ASM) chart]

- Algorithmic State Machine (ASM) chart: This is the FSM that controls the output MUX

![Algorithmic State Machine (ASM) chart]
**Example: Serial Multiplier**

**Sequential Algorithm**

\[ P \leftarrow 0, \text{Load } A,B \]

while \( B \neq 0 \)

if \( b_0 = 1 \) then

\[ P \leftarrow P + A \]

end if

left shift \( A \)

right shift \( B \)

end while

**Digital System** (FSM + Datapath circuit)

- Note that this algorithm can also be run on a simple processor. Here, we use dedicated circuitry.
  
  sclr: Synchronous clear. In this case, if sclr = ‘1’ and E = ‘1’, the register contents are initialized to 0.

- Algorithmic State Machine (ASM) chart:

  ![Algorithmic State Machine (ASM) chart](image-url)

- Parallel Access

  \( s_1 = 1 \rightarrow \text{Load} \)

  \( s_1 = 0 \rightarrow \text{Shift} \)