Lab 5

Part 1: FSM Synthesis
   – Re-design the super-controller and the controller of Labs 2/3 using different state encoding algorithms and various flip-flop types.

Part 2: Sequential Subtractor Design
   – Implementation of a Subtractor as a sequential circuit
   – Mealy & Moore Implementation
Lab 5

Part 3: Partially Specified FSM
  – Reconstruct FSM from a partially specified automata and schematic

Part 4: Input Encoding
  – Implementation of an unspecified FSM, but using an input encoding that will optimize the circuit by adding don’t-care conditions.
FSM Design Flow

1. Obtain the *state diagram* of the sequential circuit;
2. Minimize states if possible;
3. Encode states according to the encoding algorithm;
4. Generate *next state & output* tables based on state diagram and flip-flop’s excitation table;
5. Derive the logic for next-states and outputs;
6. Implement the logic with flip-flops and combinational gates.
State Encoding

- **Minimum-Bit-Change Algorithm**
  - Assign state encodings such that the total number of bit transitions between all states is minimized.
  - Particularly suitable for SR flip-flops.

- **Prioritized Adjacency Algorithm**
  - Assign adjacent encodings (codes that are 1 bit different from each other) to pairs with high priority
  - Not specific to a particular flip-flop type
Super-Controller Design

Behavior:

• The super-controller starts working when input \textit{Begin} changes from 1 to 0:
  – FSM enters an ‘Init’ state when \textit{Begin} = 1;
  – FSM leaves the ‘Init’ state when \textit{Begin} = 0.

• At each phase, the super-controller needs to
  – Signal the controller,
  – Await the controller’s \textbf{End} response.

Implementation requirements:

• T flip flops
Controller Design

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Init</th>
<th>LoadA</th>
<th>Shift</th>
<th>A/S</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1,3,5,7,9,11,13</td>
<td>0</td>
<td>1/0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2,4,6,8,10,12,14</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>1/0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- A great deal of repetition from cycle 1 to cycle 16! Can we “collapse” them into two states?
  - Need a counter to track how far the FSM has proceeded on its way to the final state.

- Implementation requirements:
  - T flip-flops
Sequential Subtractor

- An $n$-bit subtractor is basically a sequence of $n$ single bit subtractors.
- Sequential Subtractor
  - Reusing the same single bit full subtractor $n$ times with different inputs and borrow.
  - At $i^{th}$ location, subtraction requires the bits $x_i$ and $y_i$ at that bit location and borrow produced by previous subtraction at the location on the right.
Sequential Subtractor

- Implementation guidelines:
  - Make state minimized Moore and Mealy Subtractors
Partially Specified FSM

- You will be given the following state diagram:

- You will also be given an incomplete schematic that performs the actions of the state labeled with a “?”
Partially Specified FSM

Implementation Guidelines

- Fill out the automata
- Implement the missing next-state logic with JK flip flops
Input Encoding

Take advantage of equivalent inputs to increase the number of don’t-cares in next-state logic

Utilize external gates on your inputs to optimize your next-state logic

We will provide an FSM in the handout that you will implement using the above ideas
Input Encoding

Implementation guidelines:

1. Implement the FSM without any input encoding heuristics

2. Identify the equivalent inputs and find a way to encode them to induce don’t-cares

3. Implement the FSM, taking the don’t-cares into account.