Chapter 8 – Sequencing and Control

Overview

- Datapath and Control
  - Algorithmic State Machines (ASM)
    - ASM chart
  - ASM chart examples
    - Binary multiplier
- Hardwired Control
  - Control design methods
  - Sequence register and decoder
  - One flip-flop per state
- Microprogrammed control

Datapath and Control

- Datapath - performs data transfer and processing operations
- Control Unit - Determines the enabling and sequencing of the operations

Control Unit Types

- Two distinct classes:
  - Programmable
  - Non-programmable
- A programmable control unit has:
  - A program counter (PC) or other sequencing register with contents that points to the next instruction to be executed
  - An external ROM or RAM array for storing instructions and control information
  - Decision logic for determining the sequence of operations and logic to interpret the instructions
- A non-programmable control units does not fetch instructions from a memory and is not responsible for sequencing instructions
  - This type of control unit is our focus in this chapter
Algorithmic State Machines

- The function of a state machine (or sequential circuit) can be represented by a state table or a state diagram.
- A flowchart is a way of showing actions and control flow in an algorithm.
- An Algorithmic State Machine (ASM) is simply a flowchart-like way to specify state diagrams for sequential logic and, optionally, actions performed in a datapath.
- While flowcharts typically do not specify “time”, an ASM explicitly specifies a sequence of actions and their timing relationships.

Algorithmic State Machines (continued)

Processing task defined by
Sequencing mechanism

RTL Micro-operations

Controlled by

ASM Flowchart/ Hardware algorithm

ASM Primitives

1. State Box (a rectangle)
   - The State Box is a rectangle, marked with the symbolic state name, containing register transfers and output signals activated when the control unit is in the state.

2. Scalar Decision Box (a diamond)
   - The Scalar Decision Box is a diamond that describes the effects of a specific input condition on the control. It has one input path and two exit paths, one for TRUE (1) and one for FALSE (0).

3. Vector Decision Box (a hexagon)
   - The Vector Decision Box is a hexagon that describes the effects of a specific n-bit (n > 2) vector of input conditions on the control. It has one input path and up to $2^n$ exit paths, each corresponding to a binary vector value.

4. Conditional Output Box (oval)
   - The Conditional Output Box is an oval with entry from a decision block and outputs activated for the decision conditions being satisfied.

State Box

- A rectangle with:
  - The symbolic name for the state marked outside the upper left top
  - Containing register transfer operations and outputs activated within or while leaving the state
  - An optional state code, if assigned, outside the upper right top

(IDLE) IDLE

(Register transfers or outputs) R 0

(Optional state code) 0000
Scalar Decision Box

- A diamond with:
  - One input path (entry point).
  - One input condition, placed in the center of the box, that is tested.
  - A TRUE exit path taken if the condition is true (logic 1).
  - A FALSE exit path taken if the condition is false (logic 0).

Vector Decision Box

- A hexagon with:
  - One Input Path (entry point).
  - A vector of input conditions, placed in the center of the box, that is tested.
  - Up to $2^n$ output paths. The path taken has a binary vector value that matches the vector input condition.

Conditional Output Box

- An oval with:
  - One input path from a decision box or decision boxes.
  - One output path
  - Register transfers or outputs that occur only if the conditional path to the box is taken.
  - Transfers and outputs in a state box are Moore type - dependent only on state
  - Transfers and outputs in a conditional output box are Mealy type - dependent on both state and inputs

Connecting Boxes Together

- By connecting boxes together, we begin to see the power of expression.

- What are the:
  - Inputs?
  - Outputs?
  - Conditional Outputs?
  - Transfers?
  - Conditional Transfers?
ASM Blocks

- One state box along with all decision and conditional output boxes connected to it is called an ASM Block.
- The ASM Block includes all items on the path from the current state to the same or other states.

ASM chart - Multiplier Example

- Example: (101 × 011) Base 2
- Note that the partial product summation for \( n \) digits, base 2 numbers requires adding up to \( n \) digits (with carries) in a column.
- Note also \( n \times m \) digit multiply generates up to an \( m + n \) digit result (same as decimal).

Multiplexer Example: Operation

1. The multiplicand (top operand) is loaded into register B.
2. The multiplier (bottom operand) is loaded into register Q.
3. Register C||Q is initialized to 0 when G becomes 1.
4. The partial products are formed in register C||A||Q.
5. Each multiplier bit, beginning with the LSB, is processed (if bit is 1, use adder to add B to partial product; if bit is 0, do nothing)
6. C||A||Q is shifted right using the shift register
   - Partial product bits fill vacant locations in Q as multiplier is shifted out
   - If overflow during addition, the outgoing carry is recovered from C during the right shift
7. Steps 5 and 6 are repeated until Counter P = 0 as detected by Zero detect.
   - Counter P is initialized in step 4 to \( n - 1 \), \( n \) = number of bits in multiplier
Initially, the multiplicand is in B and the multiplier in Q.

The control unit decides whether to add based on the value of Q. It also checks the value of Z to determine whether the multiplication finished.

Multiplier: Control Signal Table

<table>
<thead>
<tr>
<th>Module</th>
<th>Microoperation</th>
<th>Control Signal Name</th>
<th>Control Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register A:</td>
<td>A ? 0</td>
<td>Initialize</td>
<td>IDLE, G</td>
</tr>
<tr>
<td></td>
<td>A ? A + B</td>
<td>Load</td>
<td>MUL0</td>
</tr>
<tr>
<td></td>
<td>C = A</td>
<td></td>
<td>Q</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register B:</td>
<td>B ? IN</td>
<td>Load_B</td>
<td>LOADB</td>
</tr>
<tr>
<td></td>
<td>C ? 0</td>
<td>Clear_C</td>
<td>IDLE, G + MUL1</td>
</tr>
<tr>
<td></td>
<td>C ? Cout</td>
<td>Load</td>
<td></td>
</tr>
<tr>
<td>Register Q:</td>
<td>Q ? IN</td>
<td>Load_Q</td>
<td>LOADQ</td>
</tr>
<tr>
<td></td>
<td>C ? A</td>
<td></td>
<td>Q</td>
</tr>
<tr>
<td>Counter P:</td>
<td>P ? n - 1</td>
<td>Initialize</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P ? P - 1</td>
<td>Shift_dec</td>
<td></td>
</tr>
</tbody>
</table>

Hardwired Control

- **Control Design Methods**
- Procedure specializations that use a single signal to represent each state
  - Sequence Register and Decoder
    - Sequence register with encoded states, e.g., 00, 01, 10, 11.
    - Decoder outputs produce "state" signals, e.g., 0001, 0010, 0100, 1000.
  - One Flip-flop per State
    - Flip flop outputs as "state" signals, e.g., 0001, 0010, 0100, 1000.
Finding the equations for M1 and M0 is easier due to the decoded states:

\[ M1 = MUL0 \]
\[ M0 = IDLE \cdot G + MUL1 \cdot Z \]

Note that since there are five variables, a K-map is harder to use, so we have directly written reduced equations.

The output equations using the decoded states:

\[ \text{Initialize} = IDLE \cdot G \]
\[ \text{Load} = MUL0 \cdot Q_0 \]
\[ \text{Clear}_C = IDLE \cdot G + MUL1 \]
\[ \text{Shift}_{\text{dec}} = MUL1 \]

Doing multiple level optimization, extract IDLE \cdot G:

\[ \text{START} = \text{IDLE} \cdot G \]
\[ M1 = MUL0 \]
\[ M0 = \text{START} + MUL1 \cdot \overline{Z} \]
\[ \text{Initialize} = \text{START} \]
\[ \text{Load} = MUL0 \cdot Q_0 \]
\[ \text{Clear}_C = \text{START} + MUL1 \]
\[ \text{Shift}_{\text{dec}} = MUL1 \]

The resulting circuit using flip-flops, a decoder, and the above equations is given on the next slide.
One Flip-Flop per State (easier method)

- This method uses one flip-flop per state and a simple set of transformation rules to implement the circuit.
- The design starts with the ASM chart, and replaces
  1. State Boxes with flip-flops,
  2. Scalar Decision Boxes with a demultiplexer with 2 outputs,
  3. Vector Decision Boxes with a (partial) demultiplexer
  4. Junctions with an OR gate, and
  5. Conditional Outputs with AND gates.
- Each is discussed in detail next.

State Box Transformation Rules

- Each state box transforms to a D Flip-Flop
- Entry point is connected to the input.
- Exit point is connected to the Q output.

Scalar Decision Box Transformation Rules

- Each Decision box transforms to a Demultiplexer
- Entry points are “Enable” inputs.
- The Condition is the “Select” input.
- Decoded Outputs are the Exit points.

Vector Decision Box Transformation Rules

- Each Decision box transforms to a Demultiplexer
- Entry point is Enable inputs.
- The Conditions are the Select inputs.
- Demultiplexer Outputs are the Exit points.
Junction Transformation Rules

- Where two or more entry points join, connect the entry variables to an OR gate
- The Exit is the output of the OR gate

Conditional Output Box Rules

- Entry point is Enable input.
- The Condition is the “Select” input.
- Demultiplexer Outputs are the Exit points.
- The Control OUTPUT is the same signal as the exit value.

Multiplier Example: Flip-flop per State
Design Logic Diagram

Microprogrammed Control

- *Microprogrammed Control* — a control unit with binary control values stored as words in memory.
- *Microinstructions* — words in the control memory.
- *Microprogram* — a sequence of microinstructions.
- *Control Memory* — RAM or ROM memory holding the microinstructions.
- *Writeable Control Memory* — RAM Memory into which microinstructions may be written
Microprogrammed Control Unit