CSE140L: Components and Design Techniques for Digital Systems Lab

FSMs

Tajana Simunic Rosing
Hardware Description Languages and Sequential Logic

- **Flip-flops**
  - representation of clocks - timing of state changes
  - asynchronous vs. synchronous

- **FSMs**
  - structural view (FFs separate from combinational logic)
  - behavioral view (synthesis of sequencers – not in this course)

- **Datapath** = data computation (e.g., ALUs, comparators) + registers
  - use of arithmetic/logical operators
  - control of storage elements
FSM design example – Moore vs. Mealy

- Remove one 1 from every string of 1s on the input
Verilog FSM - Reduce 1s example

- Moore machine

```verilog
module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;

  parameter zero  = 2'b00;
  parameter one1  = 2'b01;
  parameter two1s = 2'b10;

  reg out;
  reg [2:1] state; // state variables
  reg [2:1] next_state;

  always @(posedge clk)
    if (reset) state = zero;
    else       state = next_state;
```

State assignment (easy to change, if in one place)
always @(in or state)

    case (state)
        zero:
            // last input was a zero
            begin
                if (in) next_state = one1;
                else    next_state = zero;
            end
        one1:
            // we've seen one 1
            begin
                if (in) next_state = twols;
                else    next_state = zero;
            end
        twols:
            // we've seen at least 2 ones
            begin
                if (in) next_state = twols;
                else    next_state = zero;
            end
    endcase

endmodule
module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;
  reg out;
  reg state; // state variables
  reg next_state;

  always @(posedge clk)
    if (reset) state = zero;
    else state = next_state;

  always @(in or state)
    case (state)
      zero: // last input was a zero
        begin
          out = 0;
          if (in) next_state = one;
          else next_state = zero;
        end
      one: // we've seen one 1
        if (in) begin
          next_state = one; out = 1;
        end else begin
          next_state = zero; out = 0;
        end
    endcase
endmodule
Synchronous Mealy Machine

module reduce (clk, reset, in, out);
    input clk, reset, in;
    output out;
    reg out;
    reg state; // state variables

    always @(posedge clk)
        if (reset) state = zero;
        else
            case (state)
                zero: // last input was a zero
                    begin
                        out = 0;
                        if (in) state = one;
                        else state = zero;
                    end
                one: // we've seen one 1
                    if (in) begin
                        state = one; out = 1;
                    end else begin
                        state = zero; out = 0;
                    end
            endcase
endmodule
Example: Traffic light controller

- Highway/farm road intersection
Traffic light controller (cont.)

- Detectors C sense the presence of cars waiting on the farm road
  - with no car on farm road, light remain green in highway direction
  - if vehicle on farm road, highway lights go from Green to Yellow to Red, allowing the farm road lights to become green
  - these stay green only as long as a farm road car is detected but never longer than a set interval; after the interval expires, farm lights transition from Green to Yellow to Red, allowing highway to return to green
  - even if farm road vehicles are waiting, highway gets at least a set interval of green
- Assume you have an interval timer that generates:
  - a short time pulse (TS) and
  - a long time pulse (TL),
  - in response to a set (ST) signal.
  - TS is to be used for timing yellow lights and TL for green lights
Traffic light controller (cont.)

- **inputs**
  - reset: place FSM in initial state
  - C: detect vehicle on the farm road
  - TS: short time interval expired
  - TL: long time interval expired

- **outputs**
  - HG, HY, HR: assert green/yellow/red highway lights
  - FG, FY, FR: assert green/yellow/red highway lights
  - ST: start timing a short or long interval

- **state**
  - HG: highway green (farm road red)
  - HY: highway yellow (farm road red)
  - FG: farm road green (highway red)
  - FY: farm road yellow (highway red)

- **transitions**
  - Reset
  - (TL•C)'
  - TL•C / ST
  - TS / ST
  - TS'
  - (TL+C')'
  - TL+C' / ST
  - FG
  - HY
  - FY
  - HG
### Traffic light controller (cont.)

- Generate state table with symbolic states
- Consider state assignments

#### Output Encoding - Similar Problem to State Assignment
(Green = 00, Yellow = 01, Red = 10)

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Present State</th>
<th>Next State</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>C TL TS</td>
<td>HG</td>
<td>HG</td>
<td>ST H F</td>
</tr>
<tr>
<td>0 - -</td>
<td>HG</td>
<td>HG</td>
<td>0 Green Red</td>
</tr>
<tr>
<td>- 0 -</td>
<td>HG</td>
<td>HG</td>
<td>0 Green Red</td>
</tr>
<tr>
<td>1 1 -</td>
<td>HG</td>
<td>HY</td>
<td>1 Green Red</td>
</tr>
<tr>
<td>- - 0</td>
<td>HY</td>
<td>HY</td>
<td>0 Yellow Red</td>
</tr>
<tr>
<td>- - 1</td>
<td>HY</td>
<td>FG</td>
<td>1 Yellow Red</td>
</tr>
<tr>
<td>1 0 -</td>
<td>FG</td>
<td>FG</td>
<td>0 Red Green</td>
</tr>
<tr>
<td>0 - -</td>
<td>FG</td>
<td>FY</td>
<td>1 Red Green</td>
</tr>
<tr>
<td>- 1 -</td>
<td>FG</td>
<td>FY</td>
<td>1 Red Green</td>
</tr>
<tr>
<td>- - 0</td>
<td>FY</td>
<td>FY</td>
<td>0 Red Yellow</td>
</tr>
<tr>
<td>- - 1</td>
<td>FY</td>
<td>HG</td>
<td>1 Red Yellow</td>
</tr>
</tbody>
</table>

**SA1:** HG = 00, HY = 01, FG = 11, FY = 10
**SA2:** HG = 00, HY = 10, FG = 01, FY = 11
**SA3:** HG = 0001, HY = 0010, FG = 0100, FY = 1000 (one-hot)
module FSM(HR, HY, HG, FR, FY, FG, ST, TS, TL, C, reset, Clk);
output HR;
output HY;
output HG;
output FR;
output FY;
output FG;
output ST;
input TS;
input TL;
input C;
input reset;
input Clk;
reg [6:1] state;
reg ST;

parameter highwaygreen = 6'b001100;
parameter highwayyellow = 6'b010100;
parameter farmroadgreen = 6'b100001;
parameter farmroadyellow = 6'b100010;

assign HR = state[6];
assign HY = state[5];
assign HG = state[4];
assign FR = state[3];
assign FY = state[2];
assign FG = state[1];
Traffic light controller FSM

initial begin state = highwaygreen; ST = 0; end

always @(posedge Clk)
begin
  if (reset)
    begin state = highwaygreen; ST = 1; end
  else
    begin
      ST = 0;
      case (state)
        highwaygreen:
          if (TL & C) begin state = highwayyellow; ST = 1; end
        highwayyellow:
          if (TS) begin state = farmroadgreen; ST = 1; end
        farmroadgreen:
          if (TL | !C) begin state = farmroadyellow; ST = 1; end
        farmroadyellow:
          if (TS) begin state = highwaygreen; ST = 1; end
      endcase
    end
end
endmodule
module Timer(TS, TL, ST, Clk);
  output TS;
  output TL;
  input   ST;
  input   Clk;
  integer value;

  assign TS = (value >= 4); // 5 cycles after reset
  assign TL = (value >= 14); // 15 cycles after reset

  always @(posedge ST) value = 0; // async reset

  always @(posedge Clk) value = value + 1;
endmodule
Tying it all together (FSM + timer) with structural Verilog (same as a schematic drawing)
Finite state machines summary

• Models for representing sequential circuits
  – abstraction of sequential elements
  – finite state machines and their state diagrams
  – inputs/outputs
  – Mealy, Moore, and synchronous Mealy machines

• Finite state machine design procedure
  – deriving state diagram
  – deriving state transition table
  – determining next state and output functions
  – implementing combinational logic

• Hardware description languages
  – Use good coding style
  – Communicating FSMs
CSE140L: Components and Design Techniques for Digital Systems Lab

Programmable Logic Devices

Tajana Simunic Rosing

Source: Vahid, Katz, Culler
Evolution of Programmable Technologies

- Discrete devices: relays, transistors (1940s-50s)
- Discrete logic gates (1950s-60s)
- Integrated circuits (1960s-70s)
  - Map circuit to Data Book parts
  - e.g. TTL packages: Data Book for 100’s of different parts
- Gate Arrays (IBM 1970s)
  - “Custom” integrated circuit chips
  - Transistors are already on the chip
  - Place and route software puts the chip together automatically
  - + Large circuits on a chip
  - + Automatic design tools (no tedious custom layout)
  - - Only good if you want 1000’s of parts

trend toward higher levels of integration
Programmable Logic Technologies

• Fuse and anti-fuse
  – Fuse makes or breaks link between two wires
  – Typical connections are 50-300 ohm
  – One-time programmable (testing before programming?)
  – Very high density

• EPROM and EEPROM
  – High power consumption
  – Typical connections are 2K-4K ohm
  – Fairly high density

• RAM-based
  – Memory bit controls a switch that connects/disconnects two wires
  – Typical connections are .5K-1K ohm
  – Can be programmed and re-programmed in the circuit
  – Low density
Comparing RAM

- **Register file**
  - Fastest
  - But biggest size

- **SRAM**
  - Fast (e.g. 10ns)
  - More compact than register file

- **DRAM**
  - Slowest (e.g. 20ns)
    - And refreshing takes time
  - But very compact
  - Different technology for large caps.
ROM Types

- **Mask-programmed ROM**
  - Programmed at manufacturing time

- **Fuse-Based Programmable ROM**
  - Programming blows fuses
  - **One-Time Programmable ROM**

- **EPROM**
  - Erase with ultraviolet light

- **EEPROM**
  - Erasing one word at a time *electronically*

- **Flash**
  - Erase large blocks of words *simultaneously*
Memory in Verilog

- Modeled as an array of registers

```verilog
reg[15:0] memword[0:1023]; // 1,024 registers of 16 bits each

// Example Memory Block Specification
// Uses enable to control both write and read
// ------------------------------
// Read and write operations of memory.
// Memory size is 64 words of 4 bits each.
module memory (Enable, ReadWrite, Address, DataIn, DataOut);
    input Enable, ReadWrite;
    input [3:0] DataIn;
    input [5:0] Address;
    output [3:0] DataOut;
    reg [3:0] DataOut;
    reg [3:0] Mem [0:63]; // 64 x 4 memory
always @ (Enable or ReadWrite)
    if (Enable)
        if (ReadWrite)
            DataOut = Mem[Address]; // Read
        else
            Mem[Address] = DataIn; // Write
    else DataOut = 4'bz; // High impedance state
endmodule
```

Source: John Wawrzynek
Programmable Logic Devices (PLD)

- PLDs combine PLA/PAL with memory and other advanced structures
  - Similar to PLA/PAL, hence Field-Programmable Gate Arrays

- Types:
  - Antifuse PLDs
  - EPLD & EEPLD
  - FPGAs with RAMs
  - FPGA with processing
    - Digital Signal Processing
    - General purpose CPU

<table>
<thead>
<tr>
<th>Name</th>
<th>Re-programmable</th>
<th>Volatile</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuse</td>
<td>no</td>
<td>no</td>
<td>Bipolar</td>
</tr>
<tr>
<td>EPROM</td>
<td>yes</td>
<td>no</td>
<td>UVCMOS</td>
</tr>
<tr>
<td>EEPROM</td>
<td>yes in circuit</td>
<td>no</td>
<td>EECMOS</td>
</tr>
<tr>
<td>SRAM</td>
<td>yes in circuit</td>
<td>yes</td>
<td>CMOS</td>
</tr>
<tr>
<td>Antifuse</td>
<td>no</td>
<td>no</td>
<td>CMOS+</td>
</tr>
</tbody>
</table>
Field-Programmable Gate Arrays

- Logic blocks
  - To implement combinational and sequential logic
- Interconnect
  - Wires to connect inputs and outputs to logic blocks
- I/O blocks
  - Special logic blocks at periphery of device for external connections
- Key questions:
  - How to make logic blocks programmable?
  - How to connect the wires?
  - After the chip has been manufactured
Antifuse PLDs

- Actel’s Axcelerator Family

- Antifuse:
  - open when not programmed
  - Low resistance when programmed
Actel’s Axcelerator C-Cell

• C-Cell
  – Basic multiplexer logic plus more inputs and support for fast carry calculation
  – Carry connections are “direct” and do not require propagation through the programmable interconnect
Actel’s Accelerator R-Cell

- **R-Cell**
  - Core is D flip-flop
  - Muxes for altering the clock and selecting an input
  - Feed back path for current value of the flip-flop for simple hold
  - Direct connection from one C-cell output of logic module to an R-cell input; Eliminates need to use the programmable interconnect

- **Interconnection Fabric**
  - Partitioned wires
  - Special long wires
Altera’s Erasable Programmable Logic Devices (EPLDs)

- Historical Perspective
  - PALs: same technology as programmed once bipolar PROM
  - EPLDs: CMOS erasable programmable ROM (EPROM) erased by UV light
- Altera building block = MACROCELL
Altera EPLDs contain 10s-100s of independently programmed macrocells

Personalized by EPROM bits:

**Synchronous Mode**
- Flipflop controlled by global clock signal
- Local signal computes output enable

**Asynchronous Mode**
- Flipflop controlled by locally generated clock signal

+ Seq Logic: could be D, T positive or negative edge triggered
+ Product term to implement clear function
AND-OR structures are relatively limited
Cannot share signals/product terms among macrocells

EPM5128:
- 8 Fixed Inputs
- 52 I/O Pins
- 8 LABs
- 16 Macrocells/LAB
- 32 Expanders/LAB
Altera’s EEPLD

- Altera’s MAX 7k Block Diagram
EEPLD

- Altera’s MAX 7k Logic Block
SRAM based PLD

- Altera’s Flex 10k Block Diagram
SRAM based PLD

- Altera’s Flex 10k Logic Array Block (LAB)
SRAM based PLD

- Altera’s Flex 10k Logic Element (LE)
FPGA with DSP

• Altera’s Stratix II: Block Diagram
FPGA with DSP

- Altera’s Stratix II:
  - DSP Detail
FPGA with General Purpose CPU & Analog

- Actel’s Fusion Family Diagram
  - FPGA with ARM 7 CPU and Analog Components

```
<table>
<thead>
<tr>
<th>Flash Memory</th>
<th>Optional ARM or 8051 Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>User Applications</td>
</tr>
<tr>
<td></td>
<td>Fusion Applets</td>
</tr>
<tr>
<td></td>
<td>Fusion Smart Backbone</td>
</tr>
<tr>
<td>Analog Smart Peripheral 1</td>
<td>Analog Smart Peripheral 2</td>
</tr>
</tbody>
</table>
```

Level 3
Level 2
Level 1
Level 0
Programmable Logic Summary

• Discrete Gates
• Packaged Logic
• PLAs
• Ever more general architectures of programmable combinational + sequential logic and interconnect
  – Altera
  – Actel
  – Xilinx