Computer Aided Design
Assignment Nº2

Alireza Moradi, Pooya Kabiri, Ali M. Movahedian

Department of Computer Science

Iran University of Science and Technology

December 2020

0 Team

Alireza Moradi	96521479
Pooya Kabiri	96521434
AliMohammad Movahedian	96521497

1 Lecture

1.1 The Presenters

David Andrew Patterson (born November 16, 1947) is an American computer pioneer and academic who has held the position of professor of computer science at the University of California, Berkeley since 1976. He announced retirement in 2016 after serving nearly forty years, becoming a distinguished engineer at Google. He currently is vice chair of the board of directors of the RISC-V Foundation, and the Pardee Professor of Computer Science, Emeritus at UC Berkeley.

John Leroy Hennessy (born September 22, 1952) is an American computer scientist, academician, businessman, and Chair of Alphabet Inc. Hennessy is one of the founders of MIPS Computer Systems Inc. as well as Atheros and served as the tenth President of Stanford University. Hennessy announced that he would step down in the summer of 2016. He was succeeded as President by Marc Tessier-Lavigne. Marc Andreessen called him "the godfather of Silicon Valley."

1.2 The Event

The ACM A. M. Turing Award, often referred to as the "Nobel Prize of Computing," carries a 1 million dollar prize, with financial support provided by Google, Inc.

It is named for Alan M. Turing, the British mathematician who articulated the mathematical foundation and limits of computing.

It's an annual prize given by the Association for Computing Machinery (ACM) for contributions "of lasting and major technical importance to the computer field".

1.3 Summary

When computers began using integrated circuits, Moore's Law meant control stores could become much larger. Larger memories in turn allowed much more complicated ISAs. Some manufacturers chose to make micro-programming available by letting select customers add custom features they called "writable control store" (WCS). Gordon Moore believed Intel's next ISA would last the lifetime of Intel, so he hired many clever computer science Ph.D.'s and sent them to a new facility in Portland to invent the next great ISA. From complex to reduced instruction set computers.John Cocke and his colleagues developed simpler ISAs and compilers for minicomputers. As an experiment, they retargeted their research compilers to use only the simple register-register operations and load-store data transfers of the IBM 360 ISA, avoiding the more complicated instructions. They found that programs ran up to three times faster using the simple subset. These observations and the shift to high-level languages led to the opportunity to switch from CISC to RISC.

First, the RISC instructions were simplified so there was no need for a microcoded interpreter. The RISC instructions were typically as simple as micro-instructions and could be executed directly by the hardware.

Second, the fast memory, formerly used for the microcode interpreter of a CISC ISA, was repurposed to be a cache of RISC instructions. (A cache is a small, fast memory that buffers recently executed instructions, as such instructions are likely to be reused soon.)

Third, register allocators based on Gregory Chaitin's graph-coloring scheme made it much easier for compilers to efficiently use registers, which benefited these register-register ISAs.

Finally, Moore's Law meant there were enough transistors in the 1980s to include a full 32-bit datapath, along with instruction and data caches, in a single chip.

A more hardware-centric approach is to design architectures tailored to a specific problem domain and offer significant performance (and efficiency) gains for that domain, hence, the name "domain-specific architectures" (DSAs), a class of processors tailored for a specific domain programmable and often Turing-complete but tailored to a specific class of applications. In this sense, they differ from application-specific integrated circuits (ASICs) that are often used for a single function with code that rarely changes. DSAs are often called accelerators, since they accelerate some of an application when compared to executing the entire application on a general-purpose CPU. Moreover, DSAs can achieve better performance because they are more closely tailored to the needs of the application; examples of DSAs include graphics processing units (GPUs), neural network processors used for deep learning, and processors for software-defined networks (SDNs). We have considered two different approaches to improve program performance by improving efficiency in the use of hardware technology: First, by improving the performance of modern high-level languages that are typically interpreted; and second, by building domain-specific architectures that greatly improve performance and efficiency compared to general-purpose CPUs. DSLs are another example of how to improve the hardware/software interface that enables architecture innovations like DSAs. Achieving significant gains through such approaches will require a vertically integrated design team that understands applications, domain-specific languages and related compiler technology, computer architecture and organization, and the underlying implementation technology. The need to vertically integrate and make design decisions across levels of abstraction was characteristic of much of the early work in computing before the industry became horizontally structured. In this new era, vertical integration has become more important, and teams that can examine and make complex trade-offs and optimizations will be advantaged.

1.4 Conclusion

High-level, domain-specific languages and architectures, freeing architects from the chains of proprietary instruction sets, along with demand from the public for improved security, will usher in a new golden age for computer architects. Aided by open source ecosystems, agilely developed chips will convincingly demonstrate advances and thereby accelerate commercial adoption.

1.5 Idea

In my opinion, computer and computer-related technologies are becoming more and more important in every aspect of our professional, educational, personal and daily life. Sometimes modern approaches should be discovered so that a huge difference can be made, like a giant leap.

Here the Moore's Law acts as the old method and this new approach maybe can make a huge difference with violation of the old theories. For example like classical and quantum physics.

2 Sequence Detector in Verilog

2.1 Mealy FSM

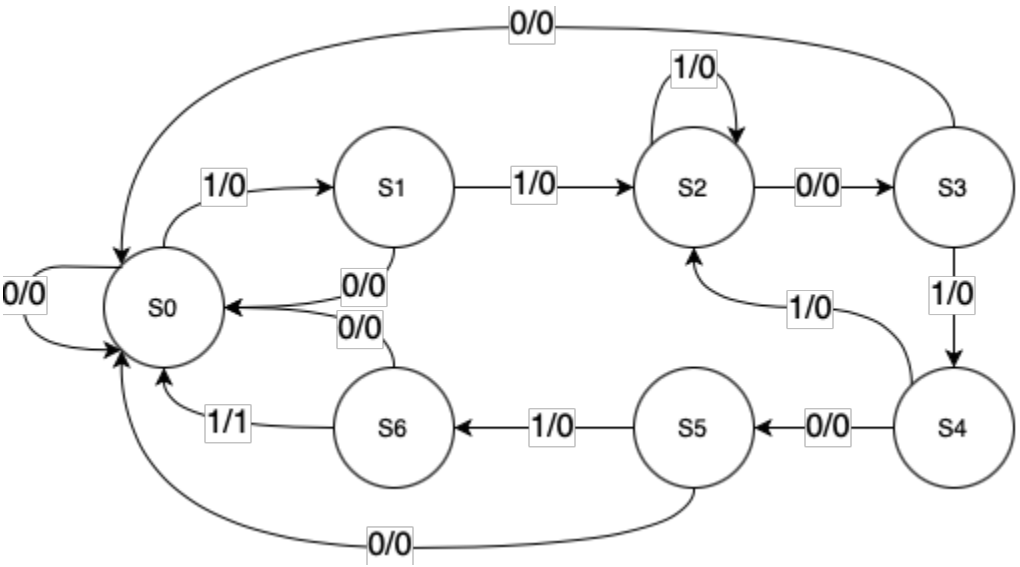

Figure 1: Mealy FSM

2.2 Moore FSM

Current State	X = 0	X = 1
S0	(S0,0)	(S1,0)
S1 = S1	(S0,0)	(S2,0)
S2 = S11	(S3,0)	(S2,0)
S3 = S110	(S0,0)	(S4,0)
S4 = S1101	(S5,0)	(S2,0)
S5 = S11010	(S0,0)	(S6,0)
S6 = S110101	(S0,0)	(S0,1)

Table 1: Mealy States

So the new states will be as follows:

- (S0,0) = SA
- (S1,0) = SB
- (S2,0) = SC
- (S3,0) = SD
- (S4,0) = SE
- (S5,0) = SF
- (S6,0) = SG
- (S0,1) = SH

Current State	X = 0	X = 1	Output
SA	SA	SB	0
SB	SA	SC	0
SC	SD	SC	0
SD	SA	SE	0
SE	SF	SC	0
SF	SA	SG	0
SG	SA	SH	0
SH	SA	SA	1

Table 2: Moore States

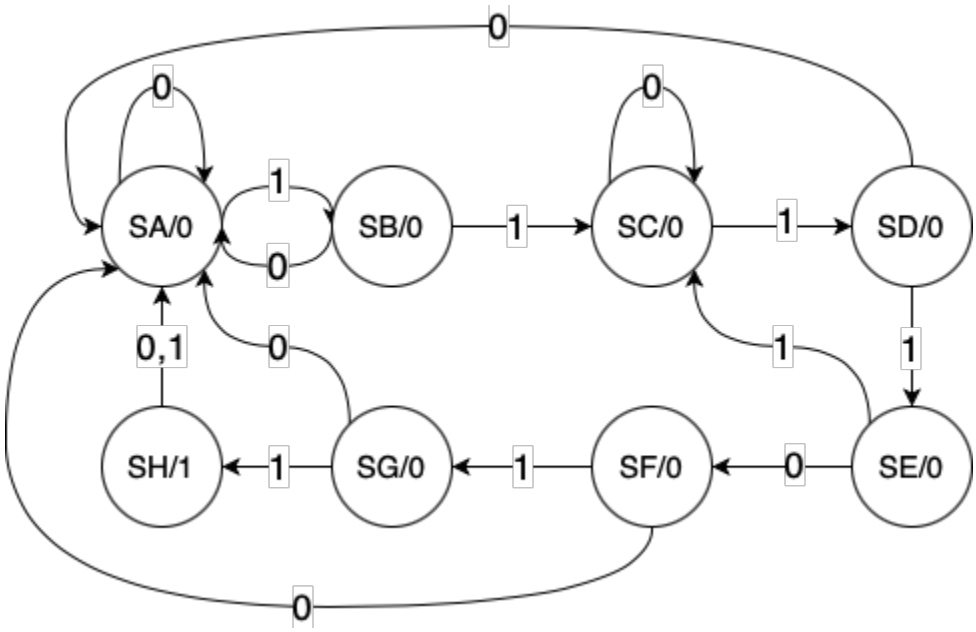

Figure 2: Moore FSM

2.3 Mealy FSM in Verilog

This code is also attached to the zip file along with the testbench and waveform. Note the you should have `Active-HDL 9.2` installed to be able to open the project file.(Download)

```
module mealy1101011 ( input clk,
                  input reset,
                  input in,
                  output reg out );
          parameter S0 = 0,
                    S1 = 1,
                    S11 = 2,
                    S110 = 3,
                    S1101 = 4,
                    S11010 = 5,
                    S110101 = 6;
        reg [2:0] cur_state, next_state;

function reg[2:0] detect_next_state_and_output(in);
    begin
      case (cur_state)
      S0 : begin
        if (in == 1) begin
                        next_state = S1;
                        detect_next_state_and_output = 0;
                        end
        else begin
                        next_state = S0;
```

```
                    detect_next_state_and_output = 0;
                    end
    end
        S1 : begin
      if (in) begin
                    next_state = S11;
                    detect_next_state_and_output = 0;
                    end
      else begin
                    next_state = S0;
                    detect_next_state_and_output = 0;
                    end
    end
        S11 : begin
      if (in) begin
                    next_state = S11;
                    detect_next_state_and_output = 0;
                    end
      else begin
                    next_state = S110;
                    detect_next_state_and_output = 0;
                    end
    end
        S110 : begin
      if (in) begin
                    next_state = S1101;
                    detect_next_state_and_output = 0;
                    end
      else begin
                    next_state = S0;
                    detect_next_state_and_output = 0;
                    end
    end
        S1101 : begin
      if (in) begin
                    next_state = S11;
                    detect_next_state_and_output = 0;
                    end
      else begin
                    next_state = S11010;
                    detect_next_state_and_output = 0;
                    end
    end
        S11010 : begin
      if (in) begin
                    next_state = S110101;
                    detect_next_state_and_output = 0;
                    end
      else begin
```

```
                        next_state = S0;
                        detect_next_state_and_output = 0;
                        end
      end
          S110101 : begin
        if (in) begin
                        next_state = S0;
                        detect_next_state_and_output = 1;
                        end
        else begin
                        next_state = S0;
                        detect_next_state_and_output = 0;
                        end
      end
          default: detect_next_state_and_output=0;
      endcase
    end
endfunction

  always @ (posedge clk) begin
    if (reset) begin
              cur_state = S0;
                out = 0;
        end
        else cur_state <= next_state;
  end

  always @ (cur_state or in) begin
    assign out = detect_next_state_and_output(in) == 1 ? 1:0;
  end
endmodule
```

2.4 Function to detect the next state in Mealy FSM

The `detect_next_state_and_output` function in the above code detects the next state and output of the FSM.

2.5 TestBench and Waveform

```
`timescale 1ps / 1ps
module mealy1101011_tb;
//Parameters declaration:
defparam UUT.S0 = 0;
parameter S0 = 0;
defparam UUT.S1 = 1;
parameter S1 = 1;
defparam UUT.S11 = 2;
```

```
parameter S11 = 2;
defparam UUT.S110 = 3;
parameter S110 = 3;
defparam UUT.S1101 = 4;
parameter S1101 = 4;
defparam UUT.S11010 = 5;
parameter S11010 = 5;
defparam UUT.S110101 = 6;
parameter S110101 = 6;

//Internal signals declarations:
reg clk;
reg reset;
reg in;
wire out;

always #10 clk = ~clk;

// Unit Under Test port map
        mealy1101011 UUT (
                .clk(clk),
                .reset(reset),
                .in(in),
                .out(out));

initial begin
    $monitor($realtime,,"ps %h %h %h %h ",clk,reset,in,out);
    clk <= 0;
    reset <= 1;
    in <= 0;
    repeat (2) @ (posedge clk);
    reset <= 0;
    // Generate a directed pattern
    @(posedge clk) in <= 1;
    @(posedge clk) in <= 1;
    @(posedge clk) in <= 0;
    @(posedge clk) in <= 1;
    @(posedge clk) in <= 0;
    @(posedge clk) in <= 1;
    @(posedge clk) in <= 1;
    @(posedge clk) reset <= 1;
    @(posedge clk) reset <= 0;
    @(posedge clk) in <= 1;
    @(posedge clk) in <= 0;
    @(posedge clk) in <= 1;
    @(posedge clk) in <= 1;
    @(posedge clk) in <= 0;
    @(posedge clk) in <= 0;
    @(posedge clk) in <= 1;
```

```
    @(posedge clk) in <= 1;
    @(posedge clk) in <= 0;
    @(posedge clk) in <= 1;
    @(posedge clk) in <= 0;
    @(posedge clk) in <= 1;
    @(posedge clk) in <= 1;
    @(posedge clk) in <= 1;
    @(posedge clk) in <= 1;
    @(posedge clk) in <= 0;
    @(posedge clk) in <= 0;
    @(posedge clk) in <= 1;
    @(posedge clk) in <= 1;
    @(posedge clk) in <= 0;
    @(posedge clk) in <= 1;
    @(posedge clk) in <= 1;
    #20 $finish;
    end
endmodule
```

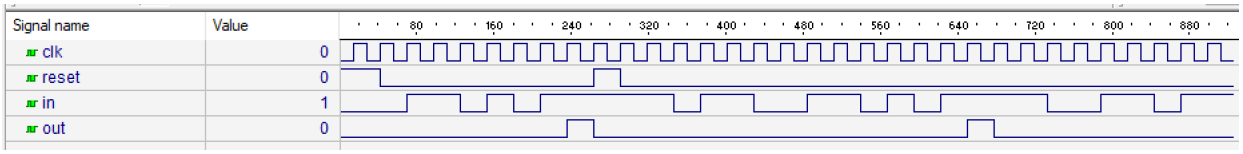

Figure 3: Waveform

3 Intelligent House

3.1 Music & Light Systems

```
module TimeBased (input clk,
                  input reset,
// in 1 means moving to next state, in 0 means staying in current state
                  input in,
                  output reg [1:0] out ); // First bit for Music, Second bit for Lights
          parameter sunrise         = 0,
                    morning         = 1,
                    evening         = 2,
                    night           = 3;
        reg [1:0] cur_state, next_state;

function reg[2:0] detect_next_state_and_output(in);
    begin
      case (cur_state)
      sunrise : begin
        if (in == 1) begin
```

```
                        next_state = morning;
                        detect_next_state_and_output = 2'b00;          // Lights off, Music off
                        end
        else begin
                        next_state = sunrise;
                        detect_next_state_and_output = 2'b01;        // Lights off, Music on
                        end
      end
          morning : begin
        if (in) begin
                        next_state = evening;
                        detect_next_state_and_output = 2'b10;        // Lights on, Music off
                        end
        else begin
                        next_state = morning;
                        detect_next_state_and_output = 2'b00;        // Lights off, Music off
                        end
      end
          evening : begin
        if (in) begin
                        next_state = night;
                        detect_next_state_and_output = 2'b00;        // Lights off, Music off
                        end
        else begin
                        next_state = evening;
                        detect_next_state_and_output = 2'b10;        // Lights on, Music off
                        end
      end
          night : begin
        if (in) begin
                        next_state = sunrise;
                        detect_next_state_and_output = 2'b01;        // Lights off, Music on
                        end
        else begin
                        next_state = night;
                        detect_next_state_and_output = 2'b00;        // Lights off, Music off
                        end
      end
          default: detect_next_state_and_output=2'b00;        // Lights off, Music off
    endcase
        end
  endfunction

  always @ (posedge clk) begin
    if (reset) begin
              cur_state = night;
                out = 0;
        end
        else cur_state <= next_state;
```

```
  end

  always @ (cur_state or in) begin
    assign out = detect_next_state_and_output(in);
        end
endmodule
```

3.2 Air Conditioner Management System

```
module AC ( input clk,
                  input reset,
                  input [6:0] in,                          // 7 bit temprature. from 0 to 127
                 // First bit for heating system, Second bit for cooling system
                  output reg [1:0] out );
          parameter ideal = 0,
                            hot         = 1,
                            cold         = 2;
        reg [1:0] cur_state, next_state;


function reg[2:0] detect_next_state_and_output(in);
    begin
      case (cur_state)
      ideal : begin
        if (in > 30) begin
                        next_state = hot;
                        // Heating system on, Cooling system off
                        detect_next_state_and_output = 2'b01;
                        end
        else if(in < 12) begin
                        next_state = cold;
                        // Heating system off, Cooling system on
                        detect_next_state_and_output = 2'b10;
                        end
                  else begin
                          next_state = ideal;
                          // Heating system off, Cooling system off
                        detect_next_state_and_output = 2'b00;
                        end
      end
          hot : begin
        if (in > 24) begin
                        next_state = hot;
                        // Heating system on, Cooling system off
                        detect_next_state_and_output = 2'b01;
                        end
        else if(in < 12) begin
```

```
                        next_state = cold;
                        // Heating system off, Cooling system on
                        detect_next_state_and_output = 2'b10;
                        end
                    else begin
                            next_state = ideal;
                            // Heating system off, Cooling system off
                        detect_next_state_and_output = 2'b00;
                        end
      end
          cold : begin
        if (in > 30) begin
                        next_state = hot;
                        // Heating system on, Cooling system off
                        detect_next_state_and_output = 2'b01;
                        end
        else if(in < 18) begin
                        next_state = cold;
                        // Heating system off, Cooling system on
                        detect_next_state_and_output = 2'b10;
                        end
                    else begin
                            next_state = ideal;
                            // Heating system off, Cooling system off
                        detect_next_state_and_output = 2'b00;
                        end
      end
      // Heating system off, Cooling system off
          default: detect_next_state_and_output=2'b00;
    endcase
        end
  endfunction


  always @ (posedge clk) begin
    if (reset) begin
              cur_state = ideal;
                out = 0;
        end
        else cur_state <= next_state;
  end

  always @ (cur_state or in) begin
    assign out = detect_next_state_and_output(in);
        end
endmodule
```

3.3 Open Window Control System

```
module PhraseDetector ( input clk,
                  input reset,
                  input [7:0] in,                 // Input character
                  output reg out);                // Phrase "open window" detected or not
          parameter silence   = 0,
                  o           = 1,
                  op          = 2,
                  ope         = 3,
                  open        = 4,
                  open_       = 5,
                  open_w      = 6,
                  open_wi     = 7,
                  open_win     = 8,
                  open_wind    = 9,
                  open_windo   = 10,
                  open_window  = 11;

        reg [1:0] cur_state, next_state;

function reg[2:0] detect_next_state_and_output(in);
    begin
      case (cur_state)
      silence : begin                               // silence
        if (in == "o") begin
                        next_state = o;
                        detect_next_state_and_output = 0;
                        end
        else begin
                        next_state = silence;
                        detect_next_state_and_output = 0;
                        end
      end
          o : begin                                       // o
        if (in == "p") begin
                        next_state = op;
                        detect_next_state_and_output = 0;
                        end
        else begin
                        next_state = silence;
                        detect_next_state_and_output = 0;
                        end
      end
          op : begin                                       // op
        if (in == "e") begin
                        next_state = ope;
                        detect_next_state_and_output = 0;
```

```
                    end
      else begin
                    next_state = silence;
                    detect_next_state_and_output = 0;
                    end
    end
        ope : begin                                          // ope
      if (in == "n") begin
                    next_state = open;
                    detect_next_state_and_output = 0;
                    end
      else begin
                    next_state = silence;
                    detect_next_state_and_output = 0;
                    end
    end
        open : begin                                          // open
      if (in == " ") begin
                    next_state = open_;
                    detect_next_state_and_output = 0;
                    end
      else begin
                    next_state = silence;
                    detect_next_state_and_output = 0;
                    end
    end
        open_ : begin                                         // open_
      if (in == "w") begin
                    next_state = open_w;
                    detect_next_state_and_output = 0;
                    end
      else begin
                    next_state = silence;
                    detect_next_state_and_output = 0;
                    end
    end
        open_w : begin                                        // open_w
      if (in == "i") begin
                    next_state = open_wi;
                    detect_next_state_and_output = 0;
                    end
      else begin
                    next_state = silence;
                    detect_next_state_and_output = 0;
                    end
    end
        open_wi : begin                                       // open_wi
      if (in == "n") begin
                    next_state = open_win;
```

```
                        detect_next_state_and_output = 0;
                        end
        else begin
                        next_state = silence;
                        detect_next_state_and_output = 0;
                        end
    end
        open_win : begin                                        // open_win
        if (in == "d") begin
                        next_state = open_wind;
                        detect_next_state_and_output = 0;
                        end
        else begin
                        next_state = silence;
                        detect_next_state_and_output = 0;
                        end
    end
        open_wind : begin                                        // open_wind
        if (in == "o") begin
                        next_state = open_windo;
                        detect_next_state_and_output = 0;
                        end
        else begin
                        next_state = silence;
                        detect_next_state_and_output = 0;
                        end
    end
        open_windo : begin                                       // open_windo
        if (in == "w") begin
                        next_state = open_window;
                        detect_next_state_and_output = 0;
                        end
        else begin
                        next_state = silence;
                        detect_next_state_and_output = 1;        // "open window" detected!
                        end
    end
        open_window : begin                                      // open_window
              next_state = silence;
              detect_next_state_and_output = 0;
    end

        default: detect_next_state_and_output=0;
  endcase
      end
endfunction
```

```
  always @ (posedge clk) begin
    if (reset) begin
              cur_state = silence;
                out = 0;
        end
        else cur_state <= next_state;
  end

  always @ (cur_state or in) begin
    assign out = detect_next_state_and_output(in) == 1 ? 1:0;
        end
endmodule
```