

Computer Aided Design

Assignment N^o2

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

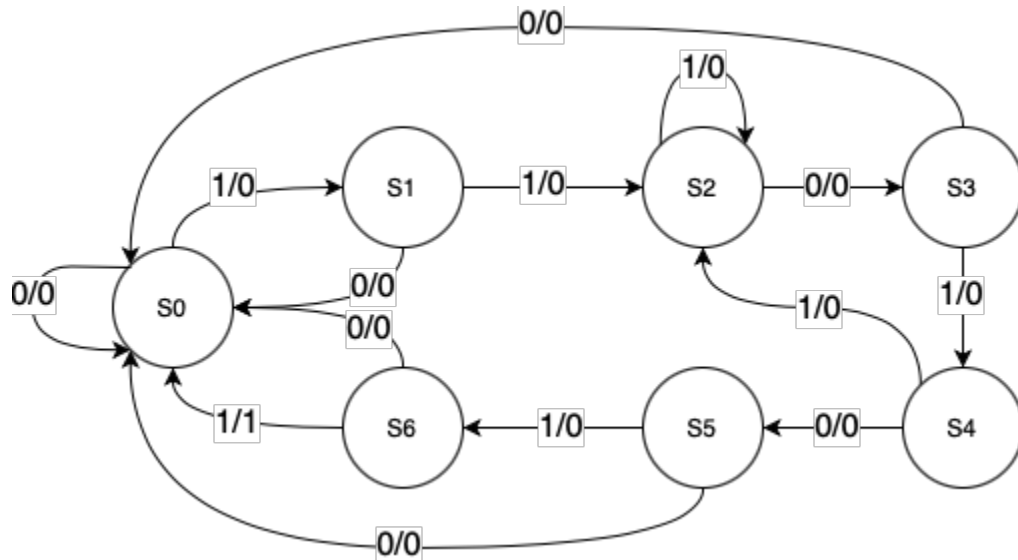


Figure 1: Mealy FSM

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



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](#))

5

```

24         detect_next_state_and_output = 0;
25     end
26 end
27     S1 : begin
28     if (in) begin
29         next_state = S11;
30         detect_next_state_and_output = 0;
31     end
32     else begin
33         next_state = S0;
34         detect_next_state_and_output = 0;
35     end
36 end
37     S11 : begin
38     if (in) begin
39         next_state = S11;
40         detect_next_state_and_output = 0;
41     end
42     else begin
43         next_state = S110;
44         detect_next_state_and_output = 0;
45     end
46 end
47     S110 : begin
48     if (in) begin
49         next_state = S1101;
50         detect_next_state_and_output = 0;
51     end
52     else begin
53         next_state = S0;
54         detect_next_state_and_output = 0;
55     end
56 end
57     S1101 : begin
58     if (in) begin
59         next_state = S11;
60         detect_next_state_and_output = 0;
61     end
62     else begin
63         next_state = S11010;
64         detect_next_state_and_output = 0;
65     end
66 end
67     S11010 : begin
68     if (in) begin
69         next_state = S110101;
70         detect_next_state_and_output = 0;
71     end
72     else begin

```

```

73         next_state = S0;
74         detect_next_state_and_output = 0;
75     end
76 end
77 S110101 : begin
78     if (in) begin
79         next_state = S0;
80         detect_next_state_and_output = 1;
81     end
82     else begin
83         next_state = S0;
84         detect_next_state_and_output = 0;
85     end
86 end
87     default: detect_next_state_and_output=0;
88 endcase
89 end
90 endfunction
91
92 always @ (posedge clk) begin
93     if (reset) begin
94         cur_state = S0;
95         out = 0;
96     end
97     else cur_state <= next_state;
98 end
99
100 always @ (cur_state or in) begin
101     assign out = detect_next_state_and_output(in) == 1 ? 1:0;
102 end
103 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

```

1  `timescale 1ps / 1ps
2  module mealy1101011_tb;
3  //Parameters declaration:
4  defparam UUT.S0 = 0;
5  parameter S0 = 0;
6  defparam UUT.S1 = 1;
7  parameter S1 = 1;
8  defparam UUT.S11 = 2;

```

```

9  parameter S11 = 2;
10 defparam UUT.S110 = 3;
11 parameter S110 = 3;
12 defparam UUT.S1101 = 4;
13 parameter S1101 = 4;
14 defparam UUT.S11010 = 5;
15 parameter S11010 = 5;
16 defparam UUT.S110101 = 6;
17 parameter S110101 = 6;
18
19 //Internal signals declarations:
20 reg clk;
21 reg reset;
22 reg in;
23 wire out;
24
25 always #10 clk = ~clk;
26
27 // Unit Under Test port map
28     mealy1101011 UUT (
29         .clk(clk),
30         .reset(reset),
31         .in(in),
32         .out(out));
33
34 initial begin
35     $monitor($realtime,,"ps %h %h %h %h ",clk,reset,in,out);
36     clk <= 0;
37     reset <= 1;
38     in <= 0;
39     repeat (2) @ (posedge clk);
40     reset <= 0;
41     // Generate a directed pattern
42     @(posedge clk) in <= 1;
43     @(posedge clk) in <= 1;
44     @(posedge clk) in <= 0;
45     @(posedge clk) in <= 1;
46     @(posedge clk) in <= 0;
47     @(posedge clk) in <= 1;
48     @(posedge clk) in <= 1;
49     @(posedge clk) reset <= 1;
50     @(posedge clk) reset <= 0;
51     @(posedge clk) in <= 1;
52     @(posedge clk) in <= 0;
53     @(posedge clk) in <= 1;
54     @(posedge clk) in <= 1;
55     @(posedge clk) in <= 0;
56     @(posedge clk) in <= 0;
57     @(posedge clk) in <= 1;

```



```

58     @(posedge clk) in <= 1;
59     @(posedge clk) in <= 0;
60     @(posedge clk) in <= 1;
61     @(posedge clk) in <= 0;
62     @(posedge clk) in <= 1;
63     @(posedge clk) in <= 1;
64     @(posedge clk) in <= 1;
65     @(posedge clk) in <= 1;
66     @(posedge clk) in <= 0;
67     @(posedge clk) in <= 0;
68     @(posedge clk) in <= 1;
69     @(posedge clk) in <= 1;
70     @(posedge clk) in <= 0;
71     @(posedge clk) in <= 1;
72     @(posedge clk) in <= 1;
73     #20 $finish;
74     end
75 endmodule

```

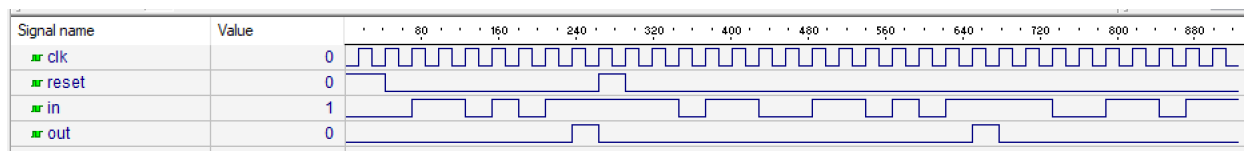


Figure 3: Waveform

3 Intelligent House

3.1 Music & Light Systems

```

1  module TimeBased (input clk,
2      input reset,
3      // in 1 means moving to next state, in 0 means staying in current state
4      input in,
5      output reg [1:0] out ); // First bit for Music, Second bit for Lights
6      parameter sunrise = 0,
7          morning = 1,
8          evening = 2,
9          night = 3;
10     reg [1:0] cur_state, next_state;
11
12     function reg[2:0] detect_next_state_and_output(in);
13     begin
14         case (cur_state)
15         sunrise : begin
16             if (in == 1) begin

```

```

17         next_state = morning;
18         detect_next_state_and_output = 2'b00;           // Lights off, Music off
19         end
20     else begin
21         next_state = sunrise;
22         detect_next_state_and_output = 2'b01;           // Lights off, Music on
23         end
24     end
25     morning : begin
26     if (in) begin
27         next_state = evening;
28         detect_next_state_and_output = 2'b10;           // Lights on, Music off
29         end
30     else begin
31         next_state = morning;
32         detect_next_state_and_output = 2'b00;           // Lights off, Music off
33         end
34     end
35     evening : begin
36     if (in) begin
37         next_state = night;
38         detect_next_state_and_output = 2'b00;           // Lights off, Music off
39         end
40     else begin
41         next_state = evening;
42         detect_next_state_and_output = 2'b10;           // Lights on, Music off
43         end
44     end
45     night : begin
46     if (in) begin
47         next_state = sunrise;
48         detect_next_state_and_output = 2'b01;           // Lights off, Music on
49         end
50     else begin
51         next_state = night;
52         detect_next_state_and_output = 2'b00;           // Lights off, Music off
53         end
54     end
55     default: detect_next_state_and_output=2'b00;         // Lights off, Music off
56 endcase
57 end
58 endfunction
59
60 always @ (posedge clk) begin
61     if (reset) begin
62         cur_state = night;
63         out = 0;
64     end
65     else cur_state <= next_state;

```

```

66     end
67
68     always @ (cur_state or in) begin
69         assign out = detect_next_state_and_output(in);
70     end
71 endmodule

```

3.2 Air Conditioner Management System

```

1  module AC ( input clk,
2              input reset,
3              input [6:0] in,                // 7 bit temprature. from 0 to 127
4              // First bit for heating system, Second bit for cooling system
5              output reg [1:0] out );
6      parameter ideal = 0,
7                  hot      = 1,
8                  cold     = 2;
9      reg [1:0] cur_state, next_state;
10
11
12  function reg[2:0] detect_next_state_and_output(in);
13  begin
14      case (cur_state)
15      ideal : begin
16          if (in > 30) begin
17              next_state = hot;
18              // Heating system on, Cooling system off
19              detect_next_state_and_output = 2'b01;
20          end
21          else if(in < 12) begin
22              next_state = cold;
23              // Heating system off, Cooling system on
24              detect_next_state_and_output = 2'b10;
25          end
26          else begin
27              next_state = ideal;
28              // Heating system off, Cooling system off
29              detect_next_state_and_output = 2'b00;
30          end
31      end
32      hot : begin
33          if (in > 24) begin
34              next_state = hot;
35              // Heating system on, Cooling system off
36              detect_next_state_and_output = 2'b01;
37          end
38          else if(in < 12) begin

```

```

39         next_state = cold;
40         // Heating system off, Cooling system on
41         detect_next_state_and_output = 2'b10;
42     end
43     else begin
44         next_state = ideal;
45         // Heating system off, Cooling system off
46         detect_next_state_and_output = 2'b00;
47     end
48 end
49 cold : begin
50     if (in > 30) begin
51         next_state = hot;
52         // Heating system on, Cooling system off
53         detect_next_state_and_output = 2'b01;
54     end
55     else if(in < 18) begin
56         next_state = cold;
57         // Heating system off, Cooling system on
58         detect_next_state_and_output = 2'b10;
59     end
60     else begin
61         next_state = ideal;
62         // Heating system off, Cooling system off
63         detect_next_state_and_output = 2'b00;
64     end
65 end
66 // Heating system off, Cooling system off
67 default: detect_next_state_and_output=2'b00;
68 endcase
69 end
70 endfunction
71
72
73 always @ (posedge clk) begin
74     if (reset) begin
75         cur_state = ideal;
76         out = 0;
77     end
78     else cur_state <= next_state;
79 end
80
81 always @ (cur_state or in) begin
82     assign out = detect_next_state_and_output(in);
83 end
84 endmodule

```

3.3 Open Window Control System

```
1 module PhraseDetector ( input clk,
2                         input reset,
3                         input [7:0] in,           // Input character
4                         output reg out);          // Phrase "open window" detected or not
5     parameter silence  = 0,
6                   o      = 1,
7                   op     = 2,
8                   ope    = 3,
9                   open   = 4,
10                  open_  = 5,
11                  open_w = 6,
12                  open_wi = 7,
13                  open_win = 8,
14                  open_wind = 9,
15                  open_windo = 10,
16                  open_window = 11;
17
18     reg [1:0] cur_state, next_state;
19
20 function reg[2:0] detect_next_state_and_output(in);
21     begin
22         case (cur_state)
23         silence : begin // silence
24             if (in == "o") begin
25                 next_state = o;
26                 detect_next_state_and_output = 0;
27             end
28
29             else begin
30                 next_state = silence;
31                 detect_next_state_and_output = 0;
32             end
33         end
34         o : begin // o
35             if (in == "p") begin
36                 next_state = op;
37                 detect_next_state_and_output = 0;
38             end
39
40             else begin
41                 next_state = silence;
42                 detect_next_state_and_output = 0;
43             end
44         end
45         op : begin // op
46             if (in == "e") begin
47                 next_state = ope;
48                 detect_next_state_and_output = 0;
49             end
50
51             else begin
52                 next_state = open;
53                 detect_next_state_and_output = 0;
54             end
55         end
56         ope : begin // ope
57             if (in == "n") begin
58                 next_state = open;
59                 detect_next_state_and_output = 0;
60             end
61
62             else begin
63                 next_state = silence;
64                 detect_next_state_and_output = 0;
65             end
66         end
67         open : begin // open
68             if (in == "_") begin
69                 next_state = open_;
70                 detect_next_state_and_output = 0;
71             end
72
73             else begin
74                 next_state = silence;
75                 detect_next_state_and_output = 0;
76             end
77         end
78         open_ : begin // open_
79             if (in == "w") begin
80                 next_state = open_w;
81                 detect_next_state_and_output = 0;
82             end
83
84             else begin
85                 next_state = silence;
86                 detect_next_state_and_output = 0;
87             end
88         end
89         open_w : begin // open_w
90             if (in == "i") begin
91                 next_state = open_wi;
92                 detect_next_state_and_output = 0;
93             end
94
95             else begin
96                 next_state = silence;
97                 detect_next_state_and_output = 0;
98             end
99         end
100        open_wi : begin // open_wi
101            if (in == "n") begin
102                next_state = open_win;
103                detect_next_state_and_output = 0;
104            end
105
106            else begin
107                next_state = silence;
108                detect_next_state_and_output = 0;
109            end
110        end
111        open_win : begin // open_win
112            if (in == "d") begin
113                next_state = open_wind;
114                detect_next_state_and_output = 0;
115            end
116
117            else begin
118                next_state = silence;
119                detect_next_state_and_output = 0;
120            end
121        end
122        open_wind : begin // open_wind
123            if (in == "o") begin
124                next_state = open_windo;
125                detect_next_state_and_output = 0;
126            end
127
128            else begin
129                next_state = silence;
130                detect_next_state_and_output = 0;
131            end
132        end
133        open_windo : begin // open_windo
134            if (in == "w") begin
135                next_state = open_window;
136                detect_next_state_and_output = 0;
137            end
138
139            else begin
140                next_state = silence;
141                detect_next_state_and_output = 0;
142            end
143        end
144        open_window : begin // open_window
145            if (in == " ") begin
146                next_state = silence;
147                detect_next_state_and_output = 1;
148            end
149
150            else begin
151                next_state = silence;
152                detect_next_state_and_output = 0;
153            end
154        end
155    endcase
156 end
```

```

47         end
48     else begin
49         next_state = silence;
50         detect_next_state_and_output = 0;
51     end
52 end
53 ope : begin                                // ope
54     if (in == "n") begin
55         next_state = open;
56         detect_next_state_and_output = 0;
57     end
58     else begin
59         next_state = silence;
60         detect_next_state_and_output = 0;
61     end
62 end
63 open : begin                                // open
64     if (in == " ") begin
65         next_state = open_;
66         detect_next_state_and_output = 0;
67     end
68     else begin
69         next_state = silence;
70         detect_next_state_and_output = 0;
71     end
72 end
73 open_ : begin                                // open_
74     if (in == "w") begin
75         next_state = open_w;
76         detect_next_state_and_output = 0;
77     end
78     else begin
79         next_state = silence;
80         detect_next_state_and_output = 0;
81     end
82 end
83 open_w : begin                                // open_w
84     if (in == "i") begin
85         next_state = open_wi;
86         detect_next_state_and_output = 0;
87     end
88     else begin
89         next_state = silence;
90         detect_next_state_and_output = 0;
91     end
92 end
93 open_wi : begin                                // open_wi
94     if (in == "n") begin
95         next_state = open_win;

```

```

96         detect_next_state_and_output = 0;
97     end
98     else begin
99         next_state = silence;
100        detect_next_state_and_output = 0;
101    end
102 end
103     open_win : begin // open_win
104     if (in == "d") begin
105         next_state = open_wind;
106         detect_next_state_and_output = 0;
107     end
108     else begin
109         next_state = silence;
110         detect_next_state_and_output = 0;
111     end
112 end
113     open_wind : begin // open_wind
114     if (in == "o") begin
115         next_state = open_windo;
116         detect_next_state_and_output = 0;
117     end
118     else begin
119         next_state = silence;
120         detect_next_state_and_output = 0;
121     end
122 end
123     open_windo : begin // open_windo
124     if (in == "w") begin
125         next_state = open_window;
126         detect_next_state_and_output = 0;
127     end
128     else begin
129         next_state = silence;
130         detect_next_state_and_output = 1; // "open window" detected!
131     end
132 end
133     open_window : begin // open_window
134         next_state = silence;
135         detect_next_state_and_output = 0;
136     end
137
138     default: detect_next_state_and_output=0;
139 endcase
140 end
141 endfunction
142
143
144

```

```
145 always @ (posedge clk) begin
146     if (reset) begin
147         cur_state = silence;
148         out = 0;
149     end
150     else cur_state <= next_state;
151 end
152
153 always @ (cur_state or in) begin
154     assign out = detect_next_state_and_output(in) == 1 ? 1:0;
155 end
156 endmodule
```
