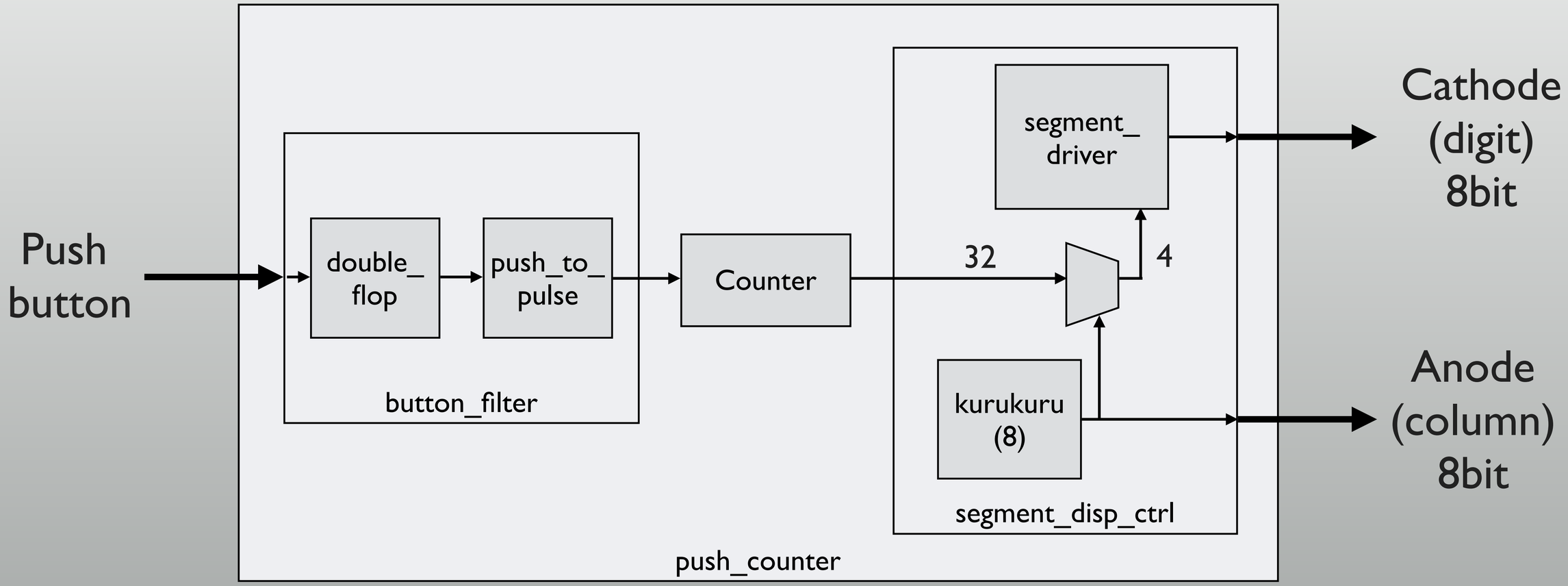
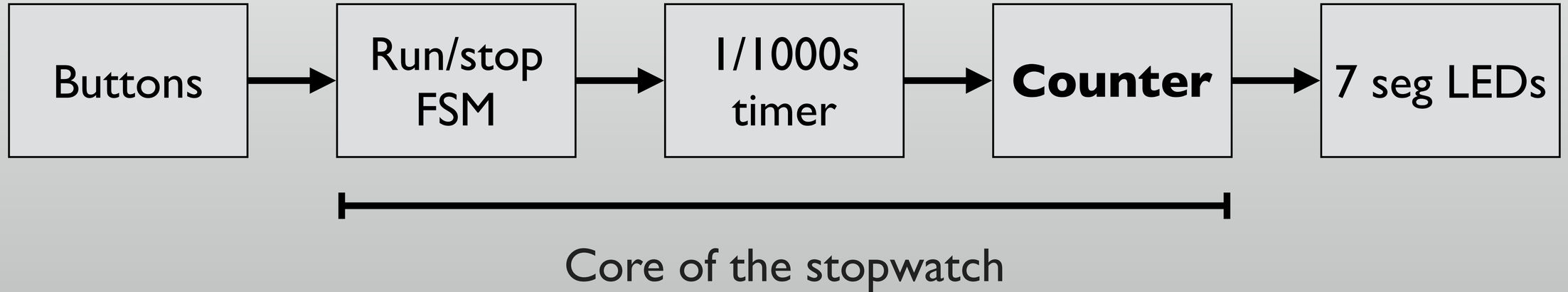
Reconfigurable Architecture (13)

Code Review

Making a stopwatch: the base design



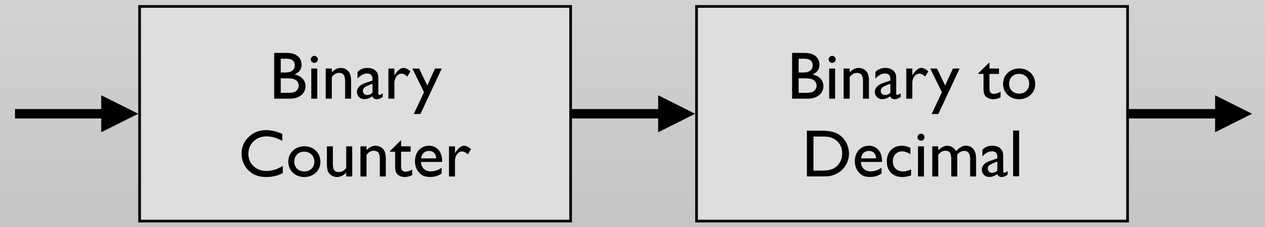
Basic organization



2 Strategies

Original: binary counter

Strategy 1:
Binary counter + Bin-Dec converter



Strategy 2:
Making a decimal counter

Binary (Hexadecimal) to Decimal

- Requires dividers by 10
  - Quotient and remainder
  - Results in very slow or very large circuit

```
1   begin
2       case(LED_activating_counter)
3         2'b00: begin
4             Anode_Activate = 4'b0111;
5             LED_BCD = displayed_number/1000;
6         end
7         2'b01: begin
8             Anode_Activate = 4'b1011;
9             LED_BCD = (displayed_number % 1000)/100;
10        end
11        2'b10: begin
12            Anode_Activate = 4'b1101;
13            LED_BCD = ((displayed_number % 1000)%100)/10;
14        end
15        2'b11: begin
16            Anode_Activate = 4'b1110;
17            LED_BCD = ((displayed_number % 1000)%100)%10;
18        end
19      endcase
20  end
```

Implementation Result for 4 digits

- Working at 100MHz is possible
  - Delay ≒ 8ns
  - Requires 13% of BlockRAM!
    (for only bin-dec conversion)

Timing

Worst Negative Slack (WNS):	2.043 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	7258

Implemented Timing Report

Resource	Utilization	Available	Utilization %
LUT	2411	63400	3.80
LUTRAM	199	19000	1.05
FF	2223	126800	1.75
BRAM	18	135	13.33
IO	4	210	1.90
BUFG	2	32	6.25

Making a decimal counter

- else if else if else if else if…
  - Maintenance is hard

```
reg [ 3:0]    cnt1;
reg [ 3:0]    cnt2;
reg [ 3:0]    cnt3;
reg [ 3:0]    cnt4;
always @ (posedge CLK) begin
   if (RST) begin
     cnt1 <= 0;
     cnt2 <= 0;
     cnt3 <= 0;
     cnt4 <= 0;
   end else if (PLUS) begin
     if (cnt4 == 10) begin
         cnt1 <= 0;
         cnt2 <= 0;
         cnt3 <= 0;
     end else if (cnt3 == 10) begin
         cnt4 <= cnt4 + 1;
         cnt3 <= 0;
     end else if (cnt2 == 10) begin
         cnt3 <= cnt3 + 1;
         cnt2 <= 0;
     end else if (cnt1 == 10) begin
         cnt2 <= cnt2 + 1;
         cnt1 <= 0;
     end else begin
         cnt1 <= cnt1 + 1;
     end
   end
end
```

Oh, no…

* With a single register, but still not good for maintenance

```
1     reg [15:0]    COUNTER;
2     always @ (posedge CLK) begin
3         if (RST)begin
4             COUNTER <= 0;
5         end
6         else begin
7             if (PLUS)begin
8                 COUNTER<=COUNTER+1;
9             end
10            else if (COUNTER[3:0]==10)begin
11                COUNTER[7:4]<=COUNTER[7:4]+1;
12                COUNTER[3:0]<=0;
13            end
14            else if(COUNTER[7:4]==10)begin
15                COUNTER[11:8]<=COUNTER[11:8]+1;
16                COUNTER[7:0]<=0;
17            end
18            else if(COUNTER[11:8]==10)begin
19                COUNTER[15:12]<=COUNTER[15:12]+1;
20                COUNTER[11:0]<=0;
21            end
22            else if(COUNTER[15:12]==10)begin
23                COUNTER[19:16]<=COUNTER[19:16]+1;
24                COUNTER[15:12]<=0;
25            end
26        end
27    end
```

A little bit more uniform way

- Separate description for each column
  - “Never write same thing more than once” rule in programming

```
1   // 0.001
2   always @( posedge CLK ) begin
3       if( CLR )
4         QA <= 4'd0;
5       else if( EN==1'b1) begin
6           if( QA==4'd9 )
7             QA <= 4'd0;
8           else
9             QA <= QA + 1'b1;
10      end
11  end
12
13  // 0.001
14  always @( posedge CLK ) begin
15      if(CLR )
16        QB <= 4'd0;
17      else if( EN==1'b1 && QA==4'd9 ) begin
18          if( QB==4'd9 )
19            QB <= 4'd0;
20          else
21            QB <= QB + 1'b1;
22      end
23  end
```

Using modularity

- Encapsulate the counter in dec_cnt
  - Reuse for every columns
  - Can be extended to any # of columns easily

```
1  module dec_cnt
2    ( input wire CLK, RST,
3      input wire        CIN,
4      output wire       COUT,
5      output reg [3:0] VAL);
6
7     always @ (posedge CLK) begin
8        if (RST) begin
9           VAL <= 0;
10       end else begin
11          if (CIN) begin
12             VAL <= (VAL==9) ? 0 : VAL+1;
13          end
14       end
15    end
16
17    assign COUT = (VAL==9) & CIN;
18 endmodule
19
20 module dec_cnt4
21   ( input wire CLK, RST,
22     input wire          TIME,
23     output wire [15:0] VAL);
24
25    wire [3:0]           C;
26
27    dec_cnt c1 (.CLK(CLK), .RST(RST), .CIN(TIME), .COUT(C[0]), .VAL(VAL[ 3: 0]));
28    dec_cnt c2 (.CLK(CLK), .RST(RST), .CIN(C[0]), .COUT(C[1]), .VAL(VAL[ 7: 4]));
29    dec_cnt c3 (.CLK(CLK), .RST(RST), .CIN(C[1]), .COUT(C[2]), .VAL(VAL[11: 8]));
30    dec_cnt c4 (.CLK(CLK), .RST(RST), .CIN(C[2]), .COUT(C[3]), .VAL(VAL[15:12]));
31 endmodule
```

Implementation Results

- Very small and fast
  - delay < 4ns
  - 1/200 LUT, 1/100 FF compared to bin-dec converter

Timing	Setup
Worst Negative Slack (WNS):	6.185 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	28

Resource	Utilization	Available	Utilization %
LUT	11	63400	0.02
FF	16	126800	0.01
IO	19	210	9.05
BUFG	1	32	3.13

Basic design principle: three-Y's

* Hierarchy: dividing a system into modules, further sub-dividing of modules until the prices are easy to understand
* Modularity: modules have well-defined functions and interfaces, can be connected together easily without unanticipated side-effects
* Regularity: seek uniformity among the modules, common module are reused