Amirhosein Yavarikhoo

810199514

FIR Filter

1. Hardware Implementation

Datapath and controller are designed as below:

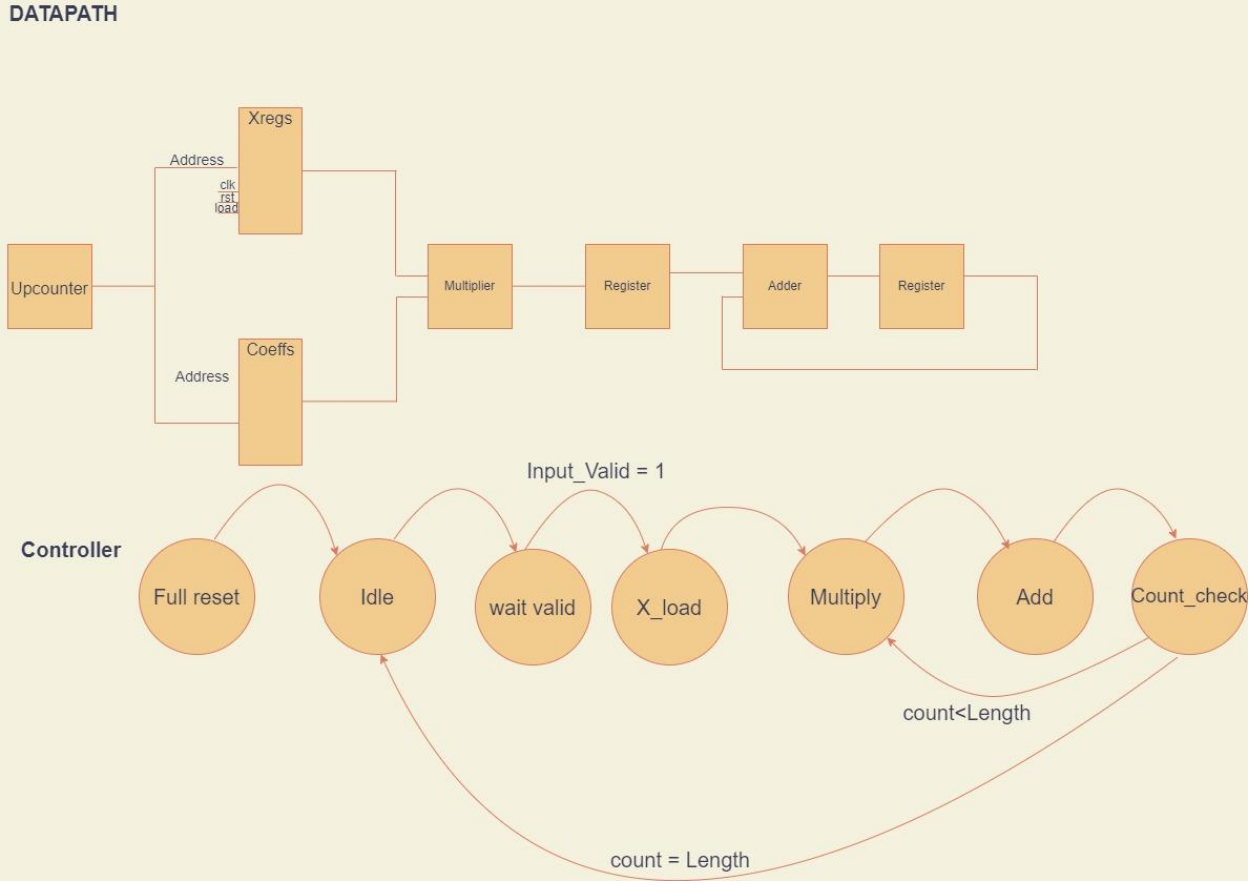

Figure 1 Controller and datapath design

در هر وضعیت کنترلر ، سیگنال ها به صورت زیر هستند:

Controller state	Asserted Signals
Full Reset	Count_rst, Xreg_rst, Pipeline registers reset
Idle	Output_valid
Wait_valid	Counter_rst
X_load	Xreg_load
Multiply	Multiplier_register_load

Add	Result_register_load, count_en
Count_chek	none

2.

Datapath code:

```
module Datapath(clk,Xreg_rst,Xreg_load,count_rst,count_en,count_out,mult_reg_load,result_reg_load,data_in,data_out);
  parameter LENGTH = 64;
  parameter WIDTH = 16;
  input clk;
  input Xreg_rst;
  input Xreg_load;
  input count_rst;
  input count_en;
  input mult_reg_load;
  input result_reg_load;
  input [WIDTH-1:0] data_in;
  output [2*WIDTH + 5:0] data_out;
  output[7:0] count_out;
  wire [WIDTH-1:0] LUT_out;
  wire [WIDTH-1:0] X_out;
  wire [2*WIDTH+5:0] Mult_out;
  wire [2*WIDTH+5:0] Mult_reg_out;
  wire [2*WIDTH+5:0] adder_out;
  wire [2*WIDTH+5:0] result_reg_out;
  UpCounter counter (.clk(clk),.rst(count_rst),.en(count_en),.out(count_out));
  Coeff_Regs #(.LENGTH(LENGTH),.WIDTH(WIDTH)) LUT (.address(count_out),.out(LUT_out));
  Xregs #(.LENGTH(LENGTH),.WIDTH(WIDTH)) X_Table (.clk(clk),.rst(Xreg_rst),.load(Xreg_load),.address(count_out),.data_in(data_in),.data_out(X_out));
  Multiplier #(.WIDTH(WIDTH)) Mult (.a(X_out),.b(LUT_out),.out(Mult_out));
  Register #(.WIDTH(2*WIDTH+6))Mult_Reg(.clk(clk),.rst(count_rst),.load(mult_reg_load),.d_in(Mult_out),.d_out(Mult_reg_out));
  Register #(.WIDTH(2*WIDTH+6))Result_Reg(.clk(clk),.rst(count_rst),.load(result_reg_load),.d_in(adder_out),.d_out(result_reg_out));
  Adder #(.WIDTH(2*WIDTH+6)) adder (.a(Mult_reg_out),.b(result_reg_out),.out(adder_out));
  assign data_out = result_reg_out;

endmodule

```

Figure 2 Datapath code

```
module Controller (clk,input_valid,rst,Xreg_rst,Xreg_load,count_rst,count_en,mult_reg_load,result_reg_load,count_out,output_valid);
  parameter LENGTH = 64;
  parameter WIDTH = 16;
  input clk;
  input input_valid;
  input rst;
  output logic Xreg_rst;
  output logic Xreg_load;
  output logic count_rst;
  output logic count_en;
  output logic mult_reg_load;
  output logic result_reg_load;
  output logic output_valid;
  input[7:0] count_out;
  parameter [2:0] full_rst=0,Idle = 1,wait_valid = 2,x_load=3,mult=4,add=5,count_check=6;
  logic [2:0] ps,ns;
  always @ (posedge clk,ps) begin //state assignments
    Xreg_rst=1'b0;
    Xreg_load= 1'b0;
    count_rst = 1'b0;
    count_en = 1'b0;
    mult_reg_load = 1'b0;
    result_reg_load = 1'b0;
    output_valid = 1'b0;
    ns = full_rst;
    case (ps)
      full_rst:begin
        ns = Idle;
        Xreg_rst = 1'b1;
        count_rst = 1'b1;
      end
      Idle:begin
        ns = wait_valid;
        output_valid = 1'b1;
      end
      wait_valid:begin
        ns = input_valid?x_load:wait_valid;
        count_rst = 1'b1;
      end
      x_load:begin
        ns = mult;
        Xreg_load = 1'b1;
```

Figure 3 Controller Code

3. Verification based on simulation:

For the first scenario, first five inputs are given to the circuit. The outputs are identical to those found in theory.

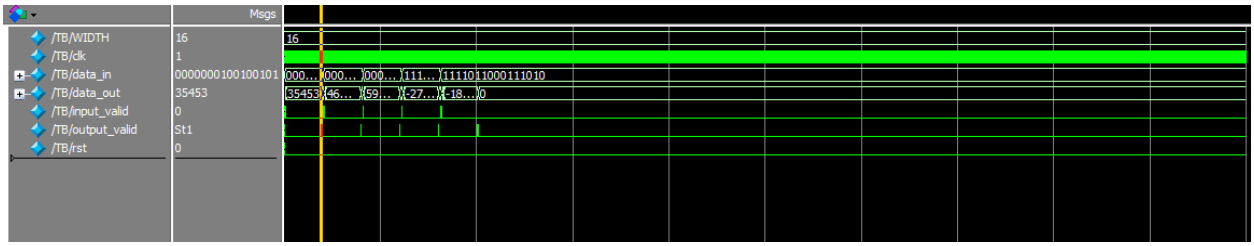

Figure 4 first output

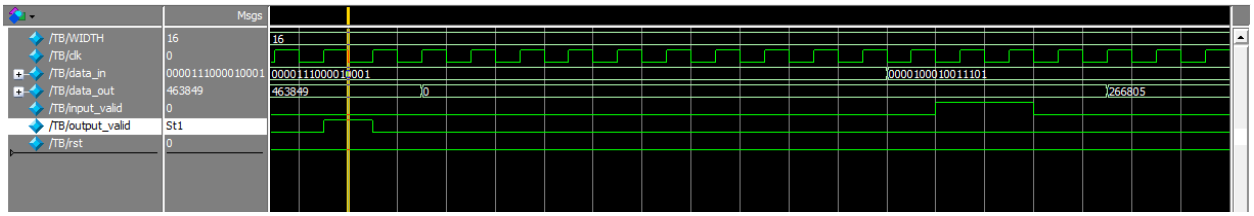

Figure 5 second output

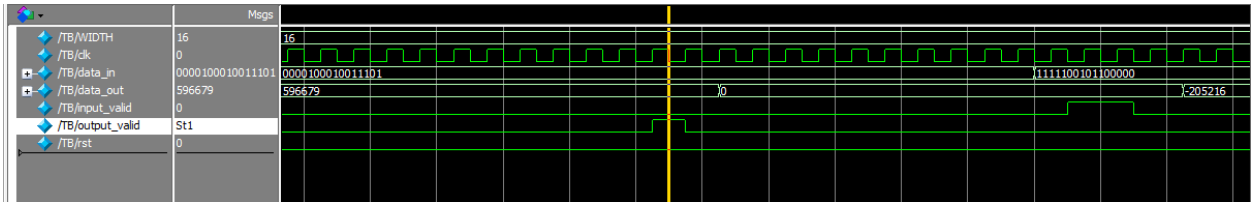

Figure 6 third output

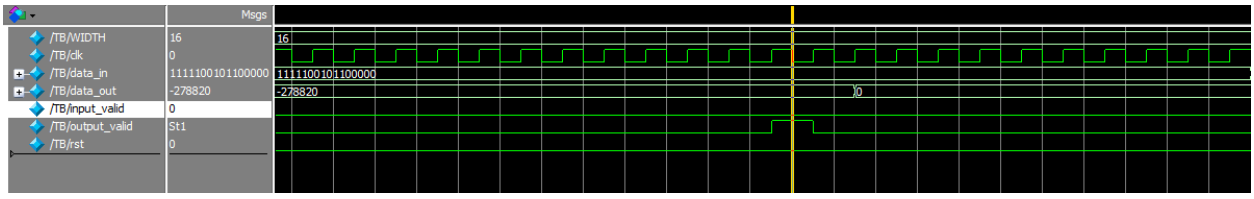

Figure 7 fourth output

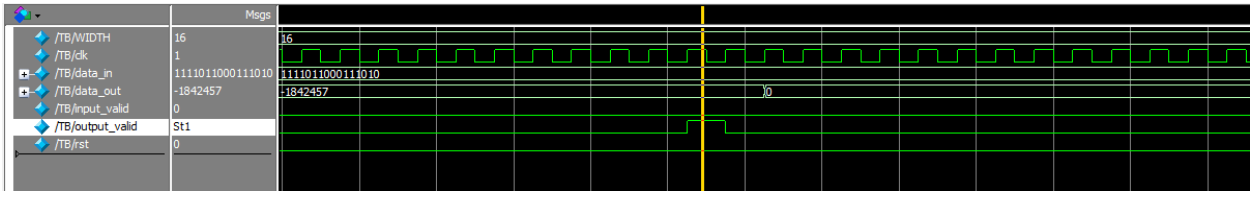

Figure 8 Fifth output

Verification based on assertion:

For the first assertion, we focus on the counter. Using this assertion in the testbench, we can ensure that the multiplication and addition operations have been performed the required number of times (in this case, 64).

```
42  property count_end;
43    @(posedge clk) (UUT.dp.counter.out==8'd64)|->##3(UUT.dp.counter.out==8'd0);
44  endproperty
45  counter_assertion:
46          assert property (count_end) $display($stime,,,"Counter Successfully Reset");
47  else
48    $display ($stime,,,"Counter didn't reset");
```

Figure 9 first assertion

Output will be as below:

```
#       10135  Counter Successfully Reset
#       10145  Counter Successfully Reset
#       10155  Counter Successfully Reset
```

Figure 10 counter verification assertion output

For the second and third assertions, we check the multiplication and addition operations.

```
50  property add_function;
51  @ (posedge clk) (UUT.Cu.ps==3'd5) |-> (UUT.dp.adder.out == UUT.dp.adder.a + UUT.dp.adder.b);
52  endproperty
53  adder_assertion:
54          assert property (add_function) $display ($stime,,,"Add Successfull");
55                  else  $display ($stime,,,"Add Failed");
56  //assertion 3
57  property mult_function;
58  @ (posedge clk) (UUT.Cu.ps==3'd4) |-> (UUT.dp.Mult.out == UUT.dp.Mult.a * UUT.dp.Mult.b);
59  endproperty
60  Mult_assertion:
61          assert property (mult_function) $display ($stime,,,"Mult Successfull");
62                  else  $display ($stime,,,"Mult Failed");
```

Figure 11 multiply and add assertion

Output will be as below:

```
#        9735  Add Successfull
#        9755  Mult Successfull
#        9765  Add Successfull
#        9785  Mult Successfull
#        9795  Add Successfull
#        9815  Mult Successfull
#        9825  Add Successfull
#        9845  Mult Successfull
#        9855  Add Successfull
#        9875  Mult Successfull
#        9885  Add Successfull
#        9905  Mult Successfull
#        9915  Add Successfull
#        9935  Mult Successfull
#        9945  Add Successfull
#        9965  Mult Successfull
#        9975  Add Successfull
#        9995  Mult Successfull
#       10005  Add Successfull
#       10025  Mult Successfull
#       10035  Add Successfull
#       10055  Mult Successfull
#       10065  Add Successfull
#       10085  Mult Successfull
#       10095  Add Successfull
```

Figure 12 assertion results

We check the outputs with the file given:

```
//assertion 4
property check_output;
@ (posedge clk) $rose (output_valid)|-> (data_out == result[k]);
endproperty
result_assert:
assert property (check_output) $display($stime,,,"Operation Successfull");
else begin
                $display ($stime,,,"Operation not successfull");
                $display ($stime,,, UUT.data_out);
                $display ($stime,,, result[k]);
        end
```

Figure 13 output assertion

For the last assertion

```
107  property first_data;
108  @ (posedge clk) $rose (UUT.dp.Xreg_load) |-> ##1(UUT.dp.X_Table.data_wires[1]==data_input[k]);
109  endproperty
110  assert property (first_data) $display ($stime,,,"Load Successfull");
111         else $display ($stime,,,"Load Not Successfull");
112
```

Figure 14 X load assertion

The output will be as below:

```
#        1995  Operation Successfull
#        2115  Load Successfull
#        4035  Operation Successfull
#        4155  Load Successfull
#        6075  Operation Successfull
#        6195  Load Successfull
#        8115  Operation Successfull
#        8235  Load Successfull
#       10155  Operation Successfull
#       10275  Load Successfull
#       12195  Operation Successfull
#       12315  Load Successfull
#       14235  Operation Successfull
#       14355  Load Successfull
#       16275  Operation Successfull
#       16395  Load Successfull
#       18315  Operation Successfull
```

Figure 15 assertion results

5. Synthesis:

Synthesis with L=50 and W = 16.

Flow Status	Successful - Wed Oct 18 19:11:43 2023
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	FIR
Top-level Entity Name	FIR
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	1,884 / 33,216 (6 %)
Total combinational functions	1,856 / 33,216 (6 %)
Dedicated logic registers	1,119 / 33,216 (3 %)
Total registers	1119
Total pins	58 / 475 (12 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	2 / 70 (3 %)
Total PLLs	0 / 4 (0 %)

Figure 16 synthesis report

Synthesis with L=100 and Width = 16.

Flow Status	Successful - Wed Oct 18 19:14:48 2023
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	FIR
Top-level Entity Name	FIR
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	1,896 / 33,216 (6 %)
Total combinational functions	1,866 / 33,216 (6 %)
Dedicated logic registers	1,120 / 33,216 (3 %)
Total registers	1120
Total pins	58 / 475 (12 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	2 / 70 (3 %)
Total PLLs	0 / 4 (0 %)

Figure17 synthesis report

Synthesis with L = 50 and Width = 8.

Flow Status	Successful - Wed Oct 18 19:17:17 2023
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	FIR
Top-level Entity Name	FIR
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	966 / 33,216 (3 %)
Total combinational functions	947 / 33,216 (3 %)
Dedicated logic registers	578 / 33,216 (2 %)
Total registers	578
Total pins	34 / 475 (7 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	1 / 70 (1 %)
Total PLLs	0 / 4 (0 %)

Figure18 synthesis report

Synthesis with L=100 and W=8.

Flow Status	Successful - Wed Oct 18 19:19:08 2023
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	FIR
Top-level Entity Name	FIR
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	968 / 33,216 (3 %)
Total combinational functions	956 / 33,216 (3 %)
Dedicated logic registers	578 / 33,216 (2 %)
Total registers	578
Total pins	34 / 475 (7 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	1 / 70 (1 %)
Total PLLs	0 / 4 (0 %)

Figure19 synthesis report

Registers	Logic Elements	حالت
1119	1884	L=50, W=16
1120	1896	L=100, W=16
578	966	L=50, W=8
578	968	L=100, W=8

Maximum frequency with L=64 and Width = 16.

	Fmax	Restricted Fmax	Clock Name	Note
1	99.37 MHz	99.37 MHz	clk	