Lecture 10 Streaming and Iterative Design on Numerical Hardware

Outline

10.1 Streaming and Iterative Design and Integration

10.2 Streaming Design on DDOT

10.3 Iterative Design with Streaming Width Four

10.4 Iterative Design with Streaming Width Two

Outline

10.1 Streaming and Iterative Design and Integration

- 10.1.1 Streaming and Iterative Designs
  - Streaming Design Structure
    - Pros: 1) maximize the parallelization of floating-point (FP) operations. 2) pipeline the input/output as streaming data
    - Cons: requires a large number of I/Os and resources, including RAMs, which may not be feasible for FPGAs due to limited hardware resources or for ASICs due to power constraints
  - Iterative Design Structure
    - Limits the streaming width ``STW'' less than the number of input and output ``N''.
    - ``STW'' data can be fed in/out over ``ceil(N/STW)'' clock cycles, where the `` ceil(X)'' means rounds fractions ``X'' up to the nearest integer.
      - E.g.: N=96 DWs, STW=2, It takes 48 clock cycles

- 10.1.2 Multi-Controller Design with Pipelined and Parallel Framework.
  - A. High-Performance Design Analysis
    - A data frame containing eight bytes, with a design streaming width of four bytes
    - Assume that the longest data path spans four clock cycles

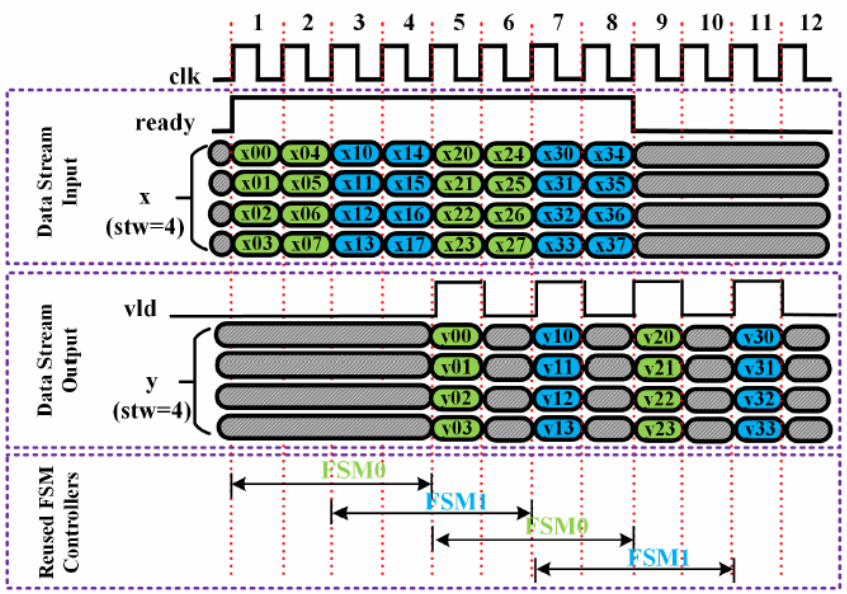


FIGURE 10.1
Timing Analysis for Iterative Design

10.1 Streaming and Iterative Design and Integration

- 10.1.2 Multi-Controller Design with Pipelined and Parallel Framework.
  - B. Multi-Controller Structural Assessment
    - The number of clock cycles needed to feed each data frame into the design engine can be calculated as

      $NO\_CY = \lceil N/STW \rceil$ the [] function rounds a number up.
    - The number of timing controllers needed:

      $NO\_TCs = \lceil Longest\_Path/NO\_CY \rceil.$ (10.1)
    - For this case, it takes two $\lceil 4/2 \rceil = 2$ controllers, the priority: FSM0>FSM1

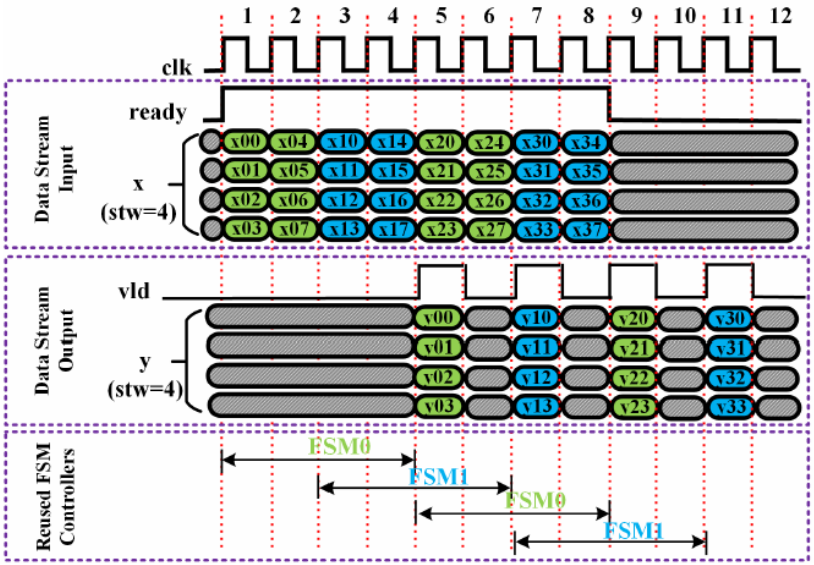


FIGURE 10.1
Timing Analysis for Iterative Design

Outline

- Recall the Basic Design on DDOT in Ch9:
  - Non-pipeline design
  - Hardware cost: 15 FPUs and 1 timing controller
  - Latency: 4n clock cycles needed for processing n data frames.

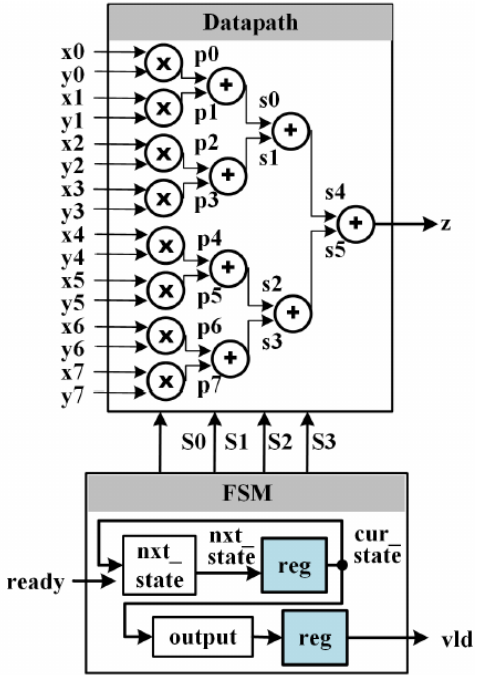


FIGURE 9.14
Design Structure of Basic Design on FP DDOT

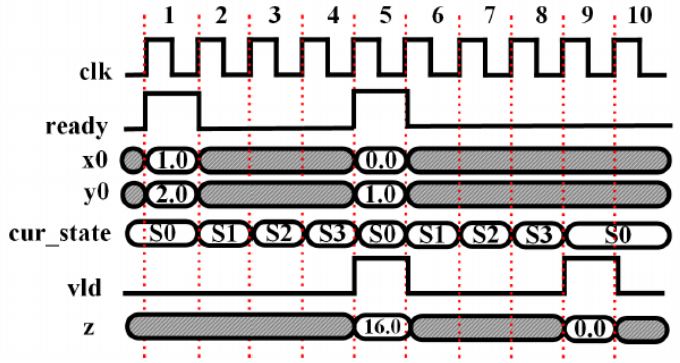


FIGURE 9.15
Timing Diagram of Basic Design on FP DDOT

- 10.2.1 Design Structure of Streaming Design on DDOT
  - The streaming width is configured at eight, enabling the input of eight FP data sets ``x0-x7'' and eight FP data sets ``y0-y7'' into the design engine during each clock cycle.
  - The number of clock cycles needed to feed each data frame into the design engine:

$$NO\_CY = \lceil N/STW \rceil = \lceil 8/8 \rceil = 1.$$

  - The number of timing controllers needed:

$$NO\_TCs = \lceil Longest\_Path/NO\_CY \rceil = \lceil 4/1 \rceil = 4.$$

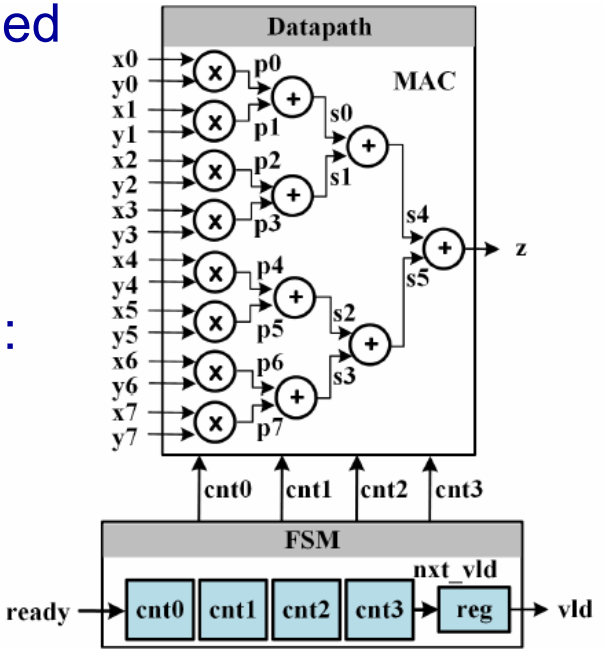
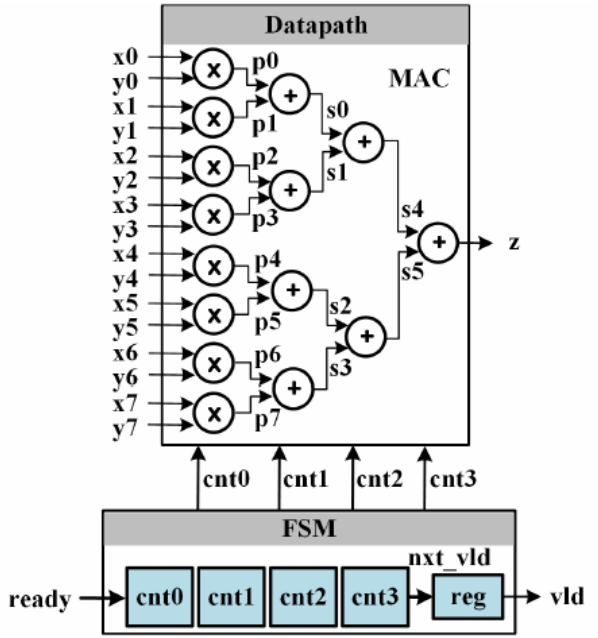
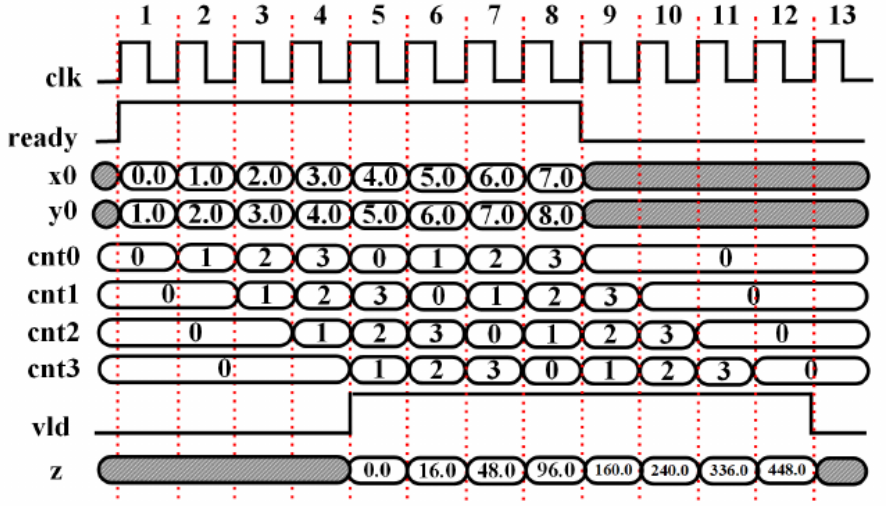


FIGURE 10.2
Streaming Design on DDOT

- 10.2.2 Timing Diagram of Streaming Design on DDOT
  - Counter ``cnt0'' monitors the processing of the first data frame across clock cycles 1-4.
  - The other three counters, namely ``cnt1'', ``cnt2'', and ``cnt3'', oversee the ddot processing of the subsequent three data frames.
  - 1st package: 8x0.0 (x0-x7) and 8x1.0 (y0-y7),
  - 2nd package: 8x1.0 and 8x2.0, until the 8th package: 8x7.0 and 8x8.0





GURE 10.3
ning Diagram of Streaming Design on DDOT

- 10.2.3 Timing Controller of Streaming Design on DDOT
  - The Overall Architecture of Each Timing Controller
    - A sequential counter
    - A counter enable circuit
      - The signals labeled with ``x" in the names, such as ``stx_rdy‘’, ``cntx_en‘’, ``nxt_cntx‘’, and ``cntx‘’, correspond to the control signals for all the counters: ``cnt0" to ``cnt3‘’.
      - The ``cntx_en" output is activated by the ``stx_rdy" pulse signal and stays active as long as the counter is actively monitoring data. This condition is determined through the use of a Reduction OR gate.
      - The ``stx_rdy" trigger signal is controlled by the arbiter, a critical component responsible for ensuring the desired rotation sequence of counters within the counter enable circuit.

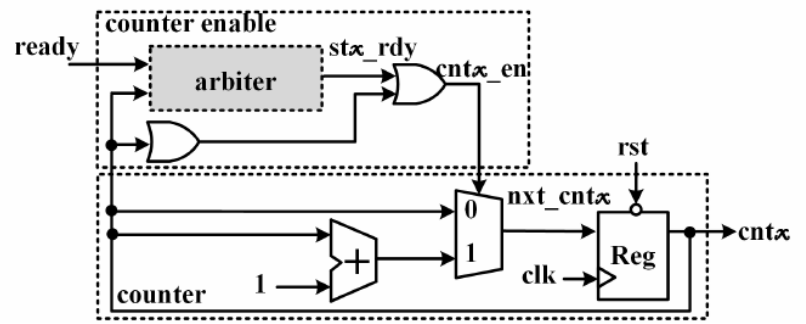
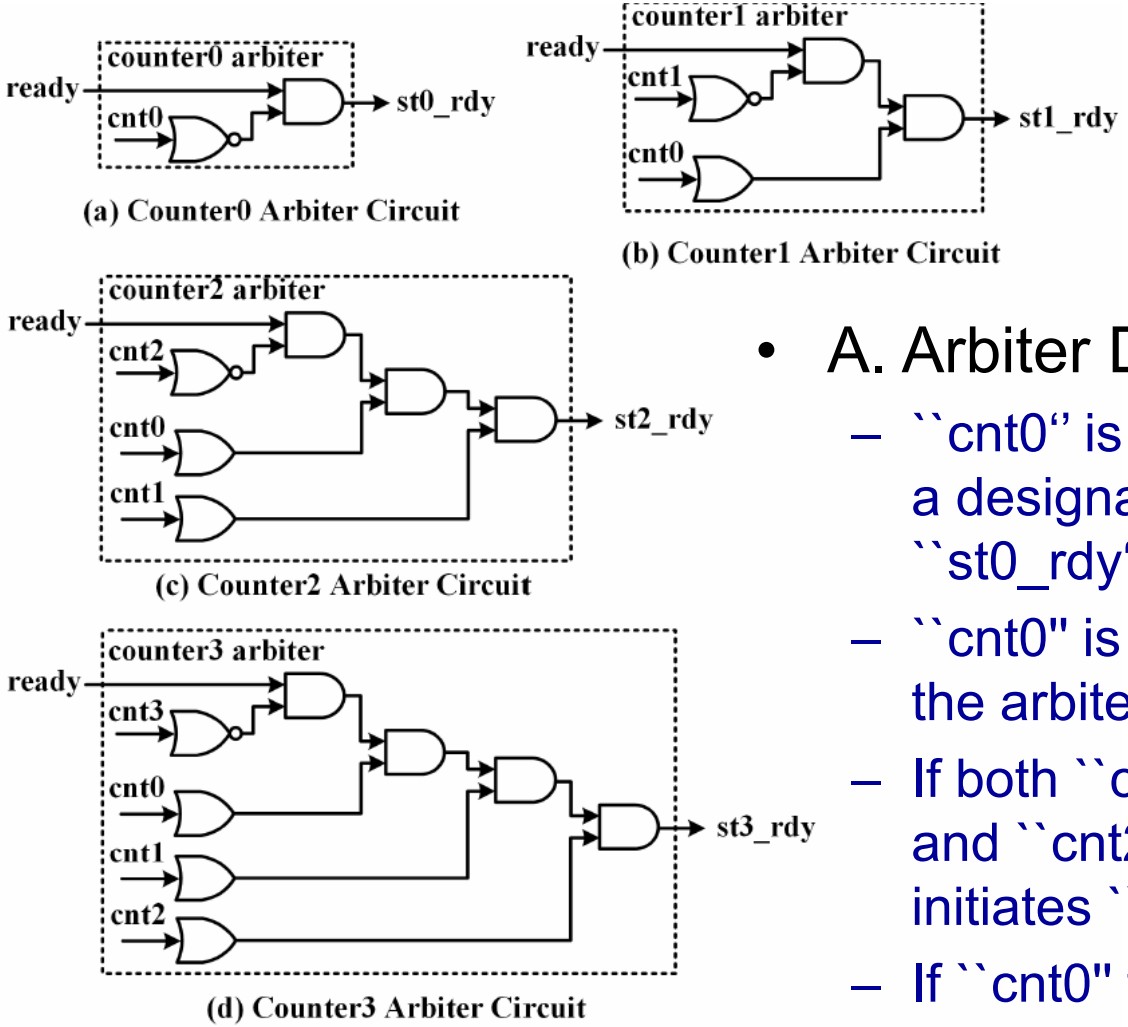


FIGURE 10.4
Timing Controller of Streaming Design on DDOT

- 10.2.3 Timing Controller of Streaming Design on DDOT
  - Arbiter Design
    - The primary challenge lies in coordinating the rotation of these four controllers in a predetermined order, based on dynamic statuses.
    - The pre-defined priorities of these four counters, from highest to lowest, are labeled as ``cnt0", ``cnt1", ``cnt2", and ``cnt3‘’.
  - Arbiter Design Methodology
    - The ``cnt0‘’ will be employed as long as it is available;
    - If ``cnt0‘’ is busy and the ``cnt1‘’ is available, then ``cnt1‘’ will be selected;
    - If both ``cnt0‘’ and ``cnt1‘’ are busy, and ``cnt2‘’ is ``cnt1‘’ , then ``cnt2‘’ will be selected;
    - If all ``cnt0‘’, ``cnt1‘’, and ``cnt2‘’ are busy, and ``cnt3‘’ is available, then ``cnt3‘’ will be selected.
    - *Busy (non-zero): Reduction OR;*
    - *Available (zero): NOT reduction OR*



- A. Arbiter Design
  - ``cnt0'' is available, it's activated using a designated pulse signal labeled as ``st0_rdy''.
  - ``cnt0" is busy and ``cnt1" is available, the arbiter triggers ``st1_rdy''.
  - If both ``cnt0" and ``cnt1" are busy, and ``cnt2" is available, the arbiter initiates ``st2_rdy''.
  - If ``cnt0" to ``cnt2" are busy, but ``cnt3" is available, the arbiter commences ``st3_rdy".

FIGURE 10.5
Arbiter Design of Timing Controller (Streaming Design on DDOT)

- B. Timing Controllers
  - Commencing with the initial data frame, control is assumed by the ``cnt0'' spanning clock cycles 1 to 4.
  - The second data frame regulated by ``cnt1'', operating from clock cycles 2 to 5, then the third data frame guided by ``cnt2'' spanning clock cycles 3 to 6. Subsequently, the fourth data frame is overseen by ``cnt3'' over clock cycles 4 to 7.
  - It follows with another four data frames over clock cycles 5-8, being processed by ``cnt0-cnt3'' sequentially.

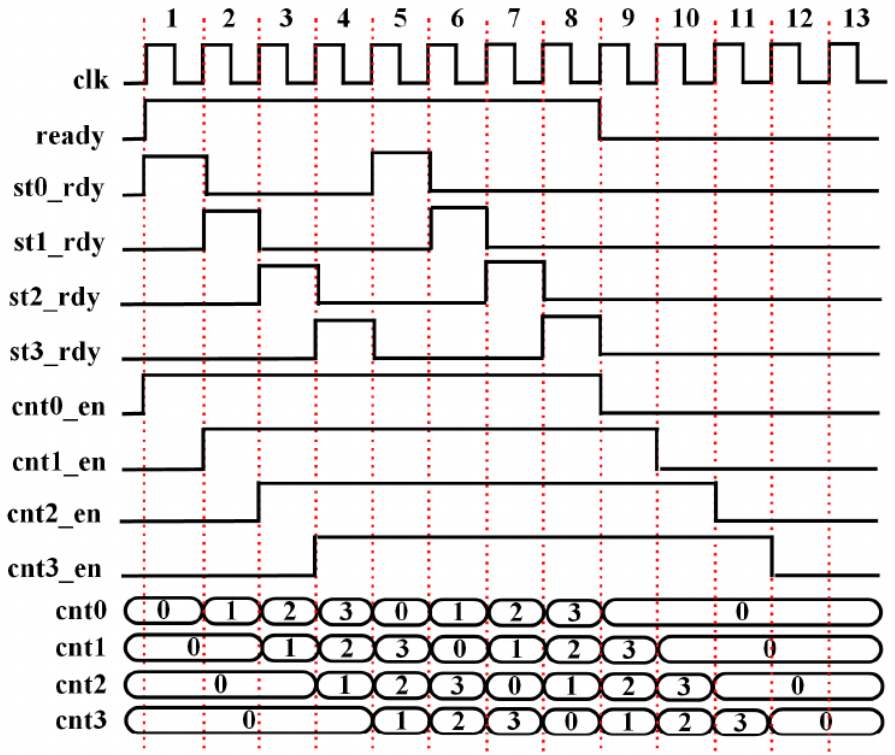
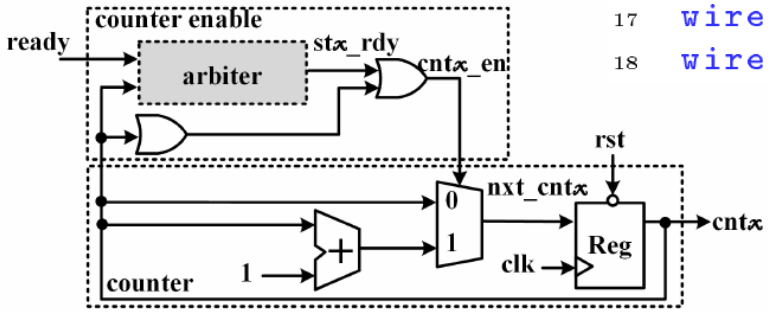


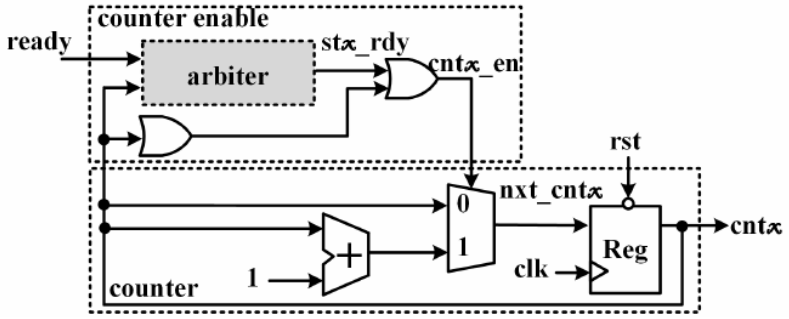
FIGURE 10.6
Timing Diagram of Timing Controller (Streaming Design on DDOT)

- 10.2.4 Verilog Code for Streaming Design on DDOT
  - Timing controller design

```
1  //****************************************
2  //**** Timing Controller Design *********
3  //****************************************
4  reg  [1:0] cnt0, cnt1, cnt2, cnt3;
5  wire st0_rdy = ready & ~|cnt0 ;
6  wire st1_rdy = ready & ~|cnt1 & |cnt0 ;
7  wire st2_rdy = ready & ~|cnt2 & |cnt0 & |cnt1 ;
8  wire st3_rdy = ready & ~|cnt3 & |cnt0 & |cnt1 & |cnt2 ;
9
10 wire cnt0_en = st0_rdy | |cnt0 ;
11 wire cnt1_en = st1_rdy | |cnt1 ;
12 wire cnt2_en = st2_rdy | |cnt2 ;
13 wire cnt3_en = st3_rdy | |cnt3 ;
14
15 wire [1:0] nxt_cnt0 = cnt0_en ? (cnt0+2'd1) : cnt0;
16 wire [1:0] nxt_cnt1 = cnt1_en ? (cnt1+2'd1) : cnt1;
17 wire [1:0] nxt_cnt2 = cnt2_en ? (cnt2+2'd1) : cnt2;
18 wire [1:0] nxt_cnt3 = cnt3_en ? (cnt3+2'd1) : cnt3;
```



- 10.2.4 Verilog Code for Streaming Design on DDOT
  - Timing controller design




```
20  always @(posedge clk) begin
21    if(rst) begin
22      cnt0<=2'd0    ;
23      cnt1<=2'd0    ;
24      cnt2<=2'd0    ;
25      cnt3<=2'd0    ;
26    end else begin
27      cnt0<=nxt_cnt0;
28      cnt1<=nxt_cnt1;
29      cnt2<=nxt_cnt2;
30      cnt3<=nxt_cnt3;
31    end
32  end
33
34  wire nxt_vld = &cnt0 | &cnt1 | &cnt2 | &cnt3;
35  always @(posedge clk) begin
36    if(rst) begin
37      vld <=1'b0    ;
38    end else begin
39      vld <=nxt_vld ;
40    end
41  end
```

Outline

10.3 Iterative Design with Streaming Width Four

- Iterative Design vs. streaming design
  - Streaming design
    - Features eight sets of single-precision FP inputs
    - Hardware cost:
      - 512 single-bit IO connections (resulting from 16x32 connections), along with the incorporation of eight FP multipliers and seven FP adders.
  - Iterative Design
    - Strike a balance between resource utilization and meeting the specified requirements
    - Enabled a reduction in the streaming width within the design engine, thereby decreasing the number of necessary IO connections and overall resource costs.

- 10.3.1 Iterative Design Structure with Streaming Width Four
  - Employs a streaming width of four, enabling the input of a complete data frame, comprising eight FP data, within two clock cycles.
  - The number of clock cycles needed to feed

each data frame into the design engine:

$$\bar{NO\_CY} = \lceil N/\bar{STW} \rceil = \lceil 8/4 \rceil = 2.$$

  - The number of timing controllers needed:

$$\bar{NO\ \ TC} = \lceil Longest\ \ Path/\bar{NO\ \ CY} \rceil = \lceil 5/2 \rceil = 3.$$

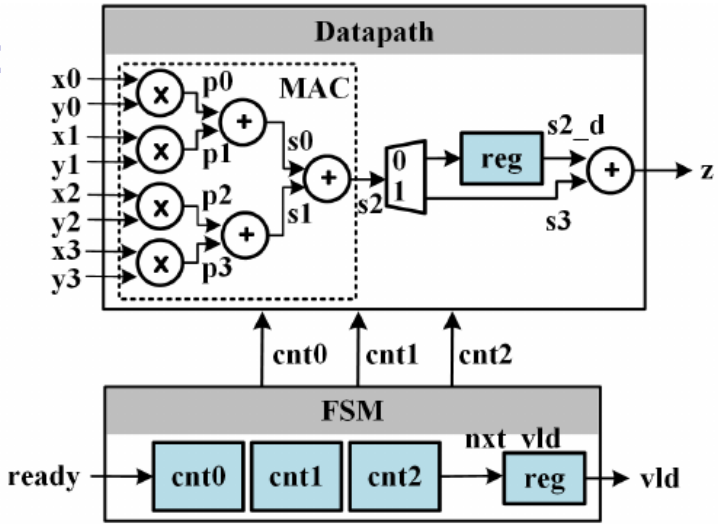


FIGURE 10.7
Iterative Design Structure with Streaming Width Four

- 10.3.1 Iterative Design Structure with Streaming Width Four
  - Data path design
    - Multiplication-Addition Circuit (MAC)
      - Four FP multipliers and three consecutive cascading FP adders
    - DeMux
      - The first MAC output is routed to a register output
      - The second MAC output is routed to

      The input of the following FP adder

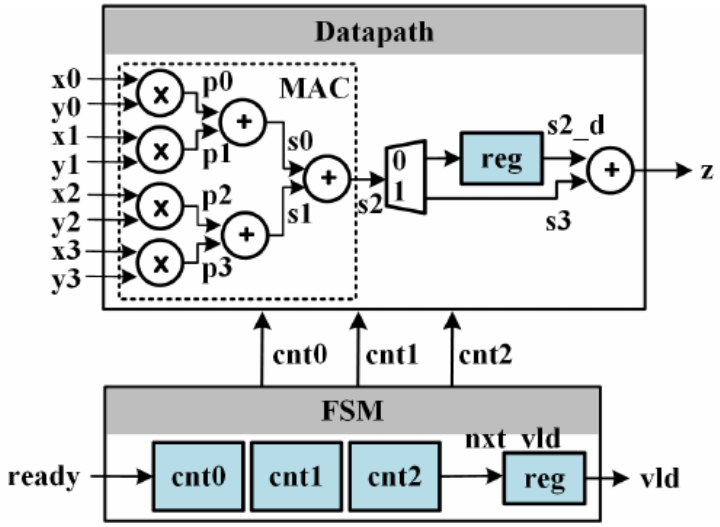
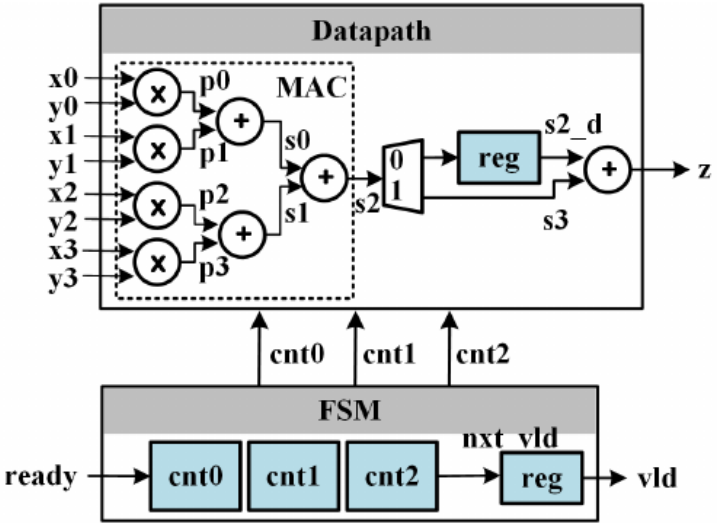


FIGURE 10.7
Iterative Design Structure with Streaming Width Four

- 10.3.2 Timing Diagram of Iterative Design with Streaming Width Four
  - Four Data Frames Example
    - The products ``p0-p3'' can be obtained in the second clock cycle, the summations ``s0-s1'' can be reached by the third clock cycle, and the intermediate MAC result ``s2'' for the first half data frame can be received by the fourth clock cycle.
    - Subsequently, this intermediate outcome is registered as ``s2_d'' within the fifth clock cycle.



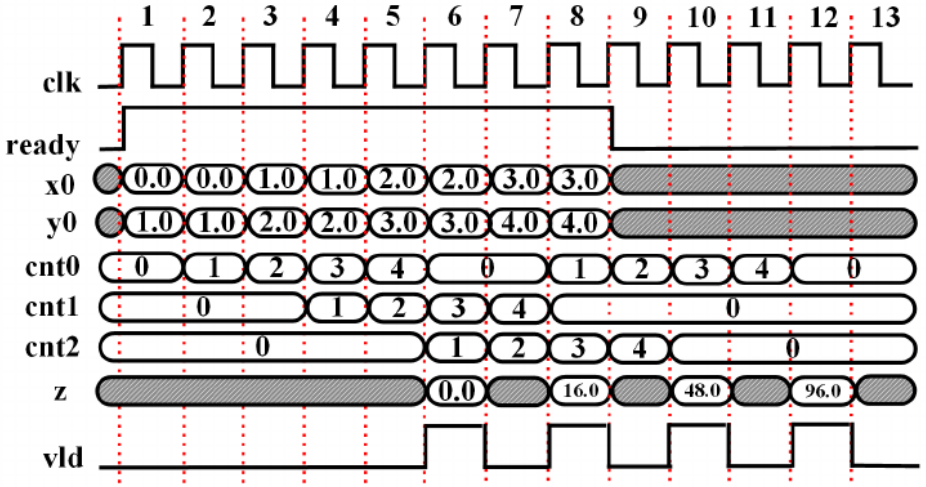
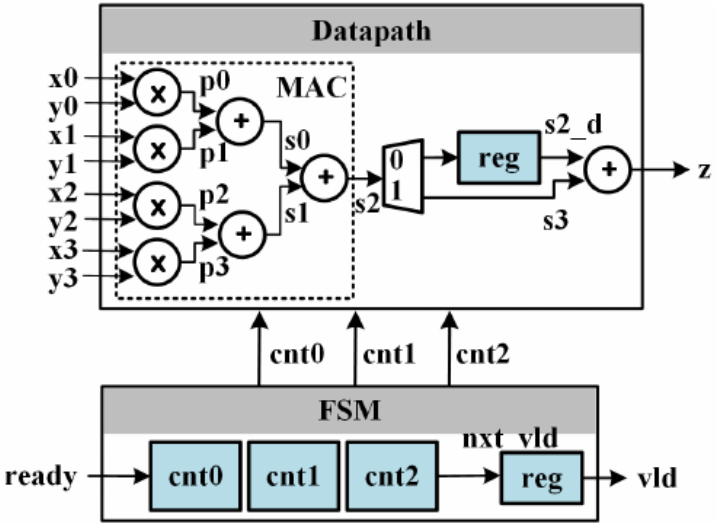


FIGURE 10.8
Timing Diagram of Iterative Design with Streaming Width Four

- 10.3.2 Timing Diagram of Iterative Design with Streaming Width Four
  - By the fifth clock cycle, the MAC outcome ``s2'' for the second half of the same data frame also comes to the fore, subsequently being channeled to the ``s3'' bus. Significantly, the ultimate outcome can be achieved by accumulating ``s2_d'' and ``s3'', effectuating the final summation ``z = FP 0.0'' within the sixth clock cycle.
  - Simultaneously, the counters ``cnt1'' and ``cnt2'' contribute to the concurrent management of the second and third data frames.



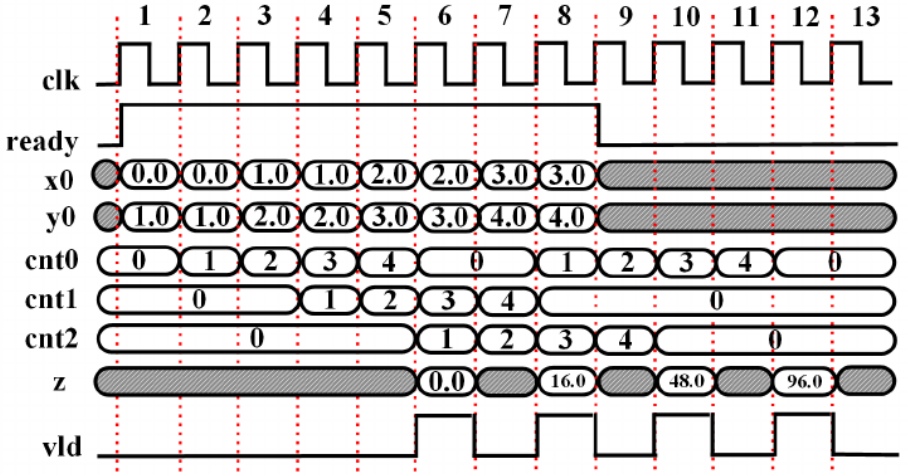
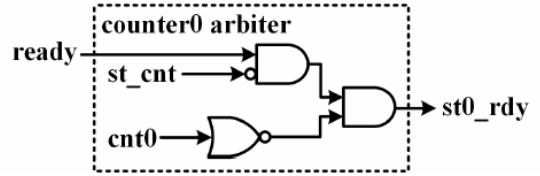


FIGURE 10.8
Timing Diagram of Iterative Design with Streaming Width Four

- 10.3. 3 Timing Controller of Iterative Design with Streaming Width Four
  - ``st_cnt'' counter
    - It is instrumental in distinguishing these two clock cycles: the first clock cycle is marked when ``st_cnt'' is zero, leading to the use of an inverter output as the input of the AND gate.
  - ``st0_rdy'' start indicator
    - is exclusively allowed during the first clock cycle of each data frame input period, and only when ``cnt0'' is in an idle state.



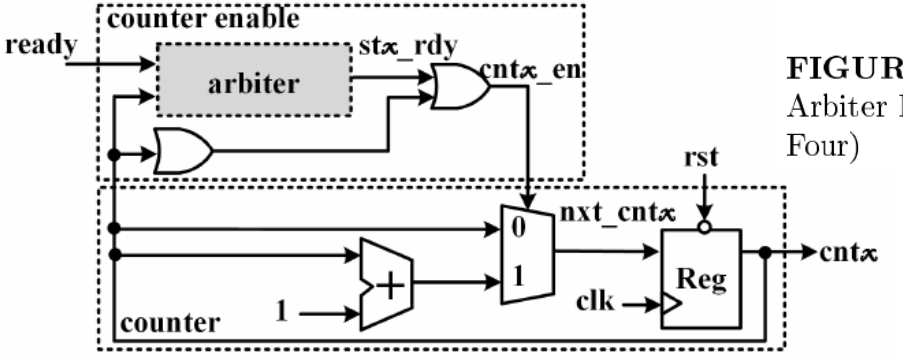


FIGURE 10.9
Arbiter Design of Timing Controller (Iterative Design with Streaming Width Four)

- 10.3. 3 Timing Controller of Iterative Design with Streaming Width Four
  - The ``st_cnt'' counter is utilized to keep track of the 2-clock cycle duration of each data frame input.
  - The counter enable signals ``cnt0_en'', ``cnt1_en'', and ``cnt2_en'' are triggered the moment the corresponding start indicator is active.

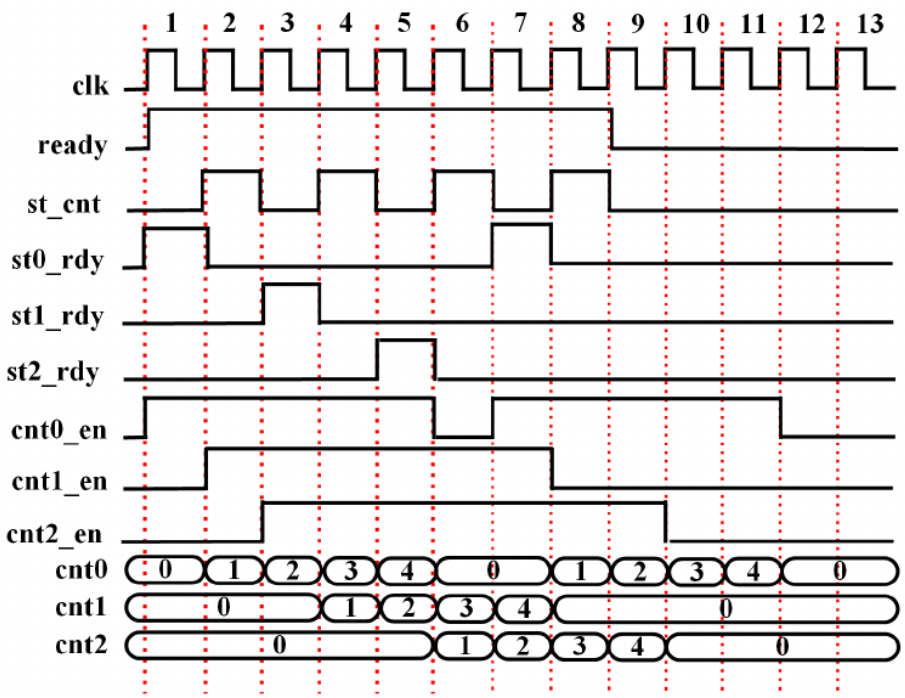
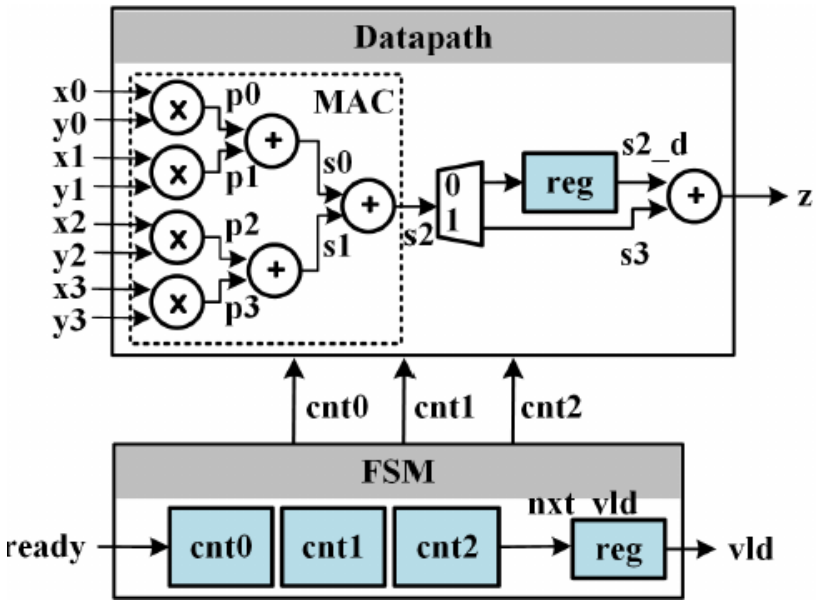


FIGURE 10.10
Timing Diagram of Timing Controller (Iterative Design with Streaming Width Four)

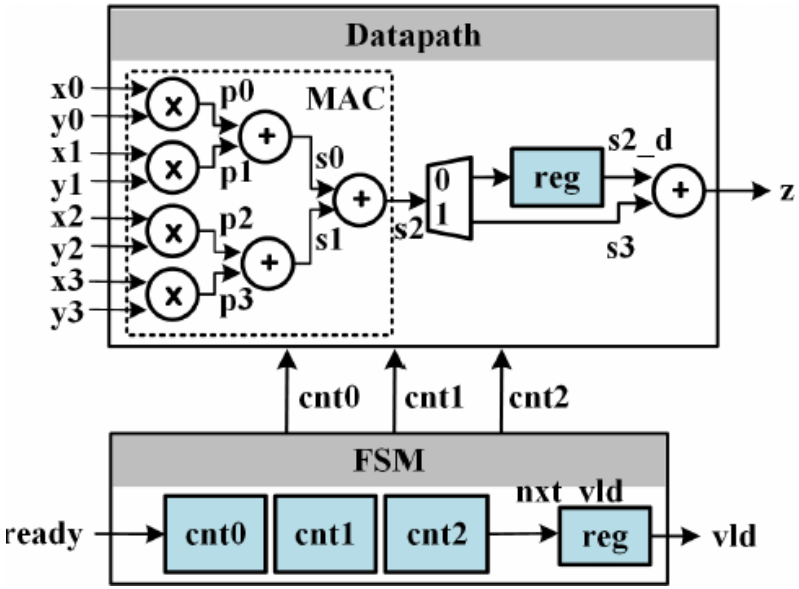
- 10.3.4 Verilog Code for Iterative Design with Streaming Width Four

```
1  module iterative_stw4_ddot (input          clk   ,
2                              input          rst   ,
3                              input          ready,
4                              input  [31:0]  x0,x1,x2,x3,
5                              input  [31:0]  y0,y1,y2,y3,
6                              output reg     vld   ,
7                              output [31:0]  z     );
8  //*********************************
9  //****     Datapath Design *********
10 //*********************************
11 reg  [2:0]  cnt0, cnt1, cnt2;
12 wire [31:0] p0, p1, p2, p3;
13 wire [31:0] s0, s1, s2, s3;
14 wire        nxt_vld;
```



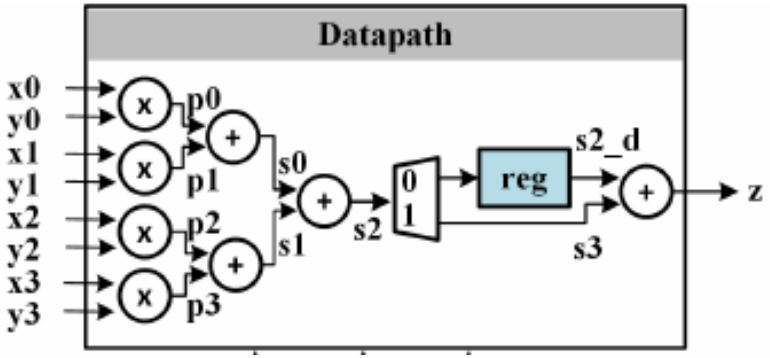
- 10.3.4 Verilog Code for Iterative Design with Streaming Width Four

```
16  // The first stage of multipliers
17  FP_multiplier u0_fp_mul(.clock    (clk  ),
18                          .reset    (rst  ),
19                          .io_in_a  (x0   ),
20                          .io_in_b  (y0   ),
21                          .io_out_s(p0    ));
22
23  FP_multiplier u1_fp_mul(.clock    (clk  ),
24                          .reset    (rst  ),
25                          .io_in_a  (x1   ),
26                          .io_in_b  (y1   ),
27                          .io_out_s(p1    ));
28
29  FP_multiplier u2_fp_mul(.clock    (clk  ),
30                          .reset    (rst  ),
31                          .io_in_a  (x2   ),
32                          .io_in_b  (y2   ),
33                          .io_out_s(p2    ));
34
35  FP_multiplier u3_fp_mul(.clock    (clk  ),
36                          .reset    (rst  ),
37                          .io_in_a  (x3   ),
38                          .io_in_b  (y3   ),
39                          .io_out_s(p3    ));
```



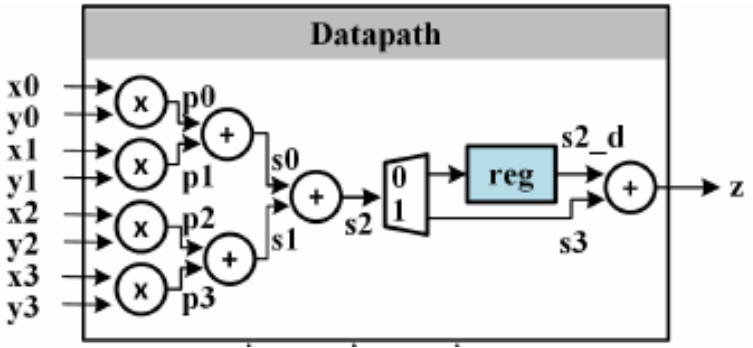
10.3 Iterative Design with Streaming Width Four

```
41  // The second stage of adders
42  FP_adder u0_fp_add(.clock    (clk  ),
43                     .reset    (rst  ),
44                     .io_in_a  (p0   ),
45                     .io_in_b  (p1   ),
46                     .io_out_s(s0   ));
47
48  FP_adder u1_fp_add(.clock    (clk  ),
49                     .reset    (rst  ),
50                     .io_in_a  (p2   ),
51                     .io_in_b  (p3   ),
52                     .io_out_s(s1   ));
53
54  // The third stage of adders
55  FP_adder u2_fp_add(.clock    (clk  ),
56                     .reset    (rst  ),
57                     .io_in_a  (s0   ),
58                     .io_in_b  (s1   ),
59                     .io_out_s(s2   ));
```



10.3 Iterative Design with Streaming Width Four

```
61  wire s2_d_flag = &cnt0[1:0] | &cnt1[1:0] | &cnt2[1:0];
62  wire [31:0] nxt_s2_d = s2_d_flag ? s2 : 32'h0;
63  reg  [31:0] s2_d;
64  always @(posedge clk) begin
65    if(rst) begin
66      s2_d <= 32'h0   ;
67    end else begin
68      s2_d <= nxt_s2_d;
69    end
70  end
71
72  assign s3 = cnt0[2]|cnt1[2]|cnt2[2] ? s2 : 32'h0;
73
74  // The fourth stage of adders
75  FP_adder u3_fp_add(.clock    (clk  ),
76                     .reset    (rst  ),
77                     .io_in_a  (s2_d ),
78                     .io_in_b  (s3   ),
79                     .io_out_s(z    ));
```



10.3 Iterative Design with Streaming Width Four

```
81  //*****************************************
82  //****     FSM Controller Design *********
83  //*****************************************
84  reg   st_cnt;
85  wire nxt_st_cnt = ready ? st_cnt+1'b1 : st_cnt;
86  always @(posedge clk) begin
87    if(rst) begin
88      st_cnt<=1'd0       ;
89    end else begin
90      st_cnt<=nxt_st_cnt;
91    end
92  end
93
```

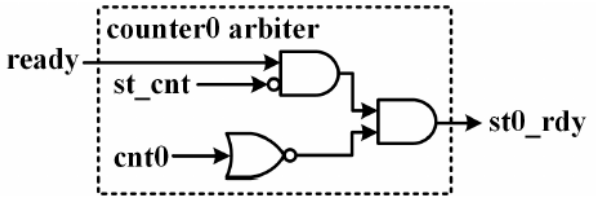
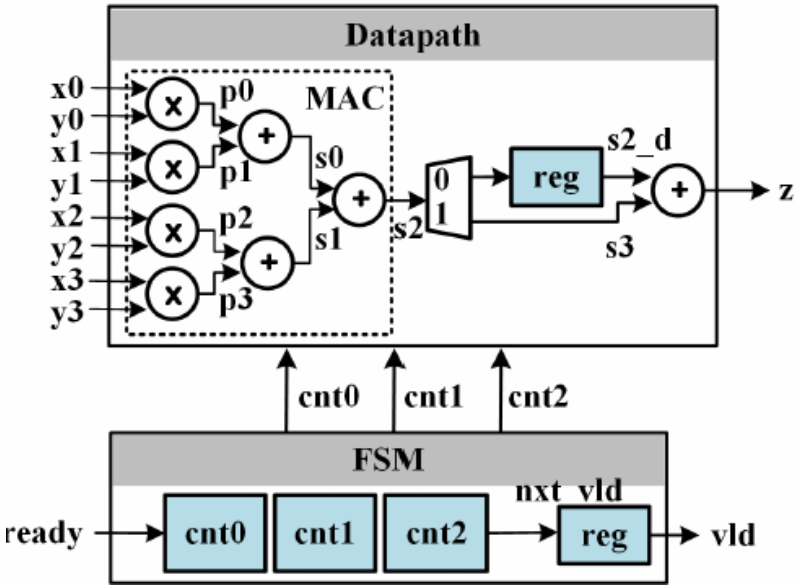


FIGURE 10.9
Arbiter Design of Timing Controller (Iterative Design with Streaming Width Four)

```
94  wire st0_rdy = ready & ~st_cnt & ~|cnt0 ;
95  wire st1_rdy = ready & ~st_cnt & ~|cnt1 & |cnt0;
96  wire st2_rdy = ready & ~st_cnt & ~|cnt2 & |cnt0 & |cnt1;
97
98  wire cnt0_en = st0_rdy | |cnt0 ;
99  wire cnt1_en = st1_rdy | |cnt1 ;
100 wire cnt2_en = st2_rdy | |cnt2 ;
101
102 wire [2:0] nxt_cnt0 = cnt0[2] ? 3'd0 :
103                                 cnt0_en ? (cnt0+3'd1) : cnt0;
104 wire [2:0] nxt_cnt1 = cnt1[2] ? 3'd0 :
105                                 cnt1_en ? (cnt1+3'd1) : cnt1;
106 wire [2:0] nxt_cnt2 = cnt2[2] ? 3'd0 :
107                                 cnt2_en ? (cnt2+3'd1) : cnt2;
```

10.3 Iterative Design with Streaming Width Four

```
108  always @(posedge clk) begin
109    if(rst) begin
110      cnt0<=3'd0     ;
111      cnt1<=3'd0     ;
112      cnt2<=3'd0     ;
113    end else begin
114      cnt0<=nxt_cnt0;
115      cnt1<=nxt_cnt1;
116      cnt2<=nxt_cnt2;
117    end
118  end
119
120  assign nxt_vld = cnt0[2] | cnt1[2] | cnt2[2];
121  always @(posedge clk) begin
122    if(rst) begin
123      vld <=1'b0     ;
124    end else begin
125      vld <=nxt_vld ;
126    end
127  end
128  endmodule
```



Outline

- 10.4.1 Iterative Design Structure with Streaming Width Two
  - Employs a streaming width of two, enabling the input of a complete data frame, comprising eight FP data, within four clock cycles.
  - The number of clock cycles needed to feed each data frame into the design engine:

$$NO\_CY = \lceil N/STW \rceil = 4.$$

  - The number of timing controllers needed:

$$NO\_TC = \lfloor Longest\_Path/NO\_CY \rfloor = \lceil 6/4 \rceil = 2.$$

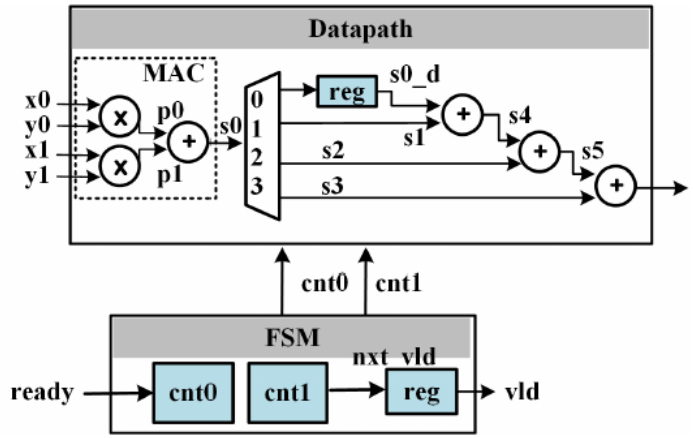


FIGURE 10.11
Iterative Design Structure with Streaming Width Two

- 10.4.1 Iterative Design Structure with Streaming Width Two
  - Data path design
    - Multiplication-Addition Circuit (MAC)
      - Two FP multipliers and one FP adders
    - DeMux
      - The first MAC output is routed to a register output
      - The second, third, and fourth MAC outputs are routed to the inputs of the following FP adders

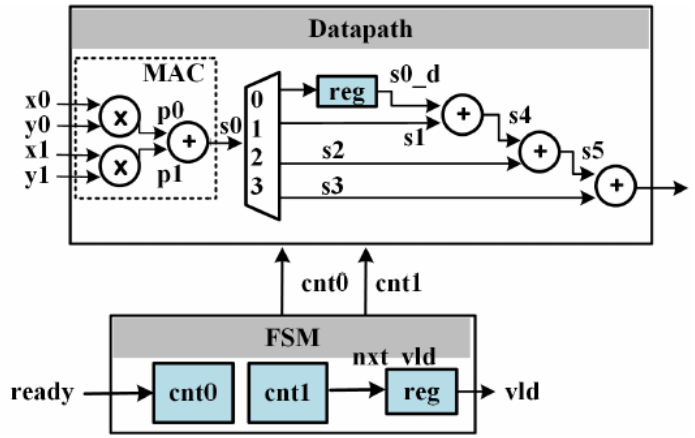


FIGURE 10.11
Iterative Design Structure with Streaming Width Two

- 10.4.2 Timing Diagram of Iterative Design with Streaming Width Two
  - ``cnt0" takes charge of overseeing the computation for the first data frame, spanning clock cycles 1-6.
  - The second data frame falls under the control of ``cnt1".

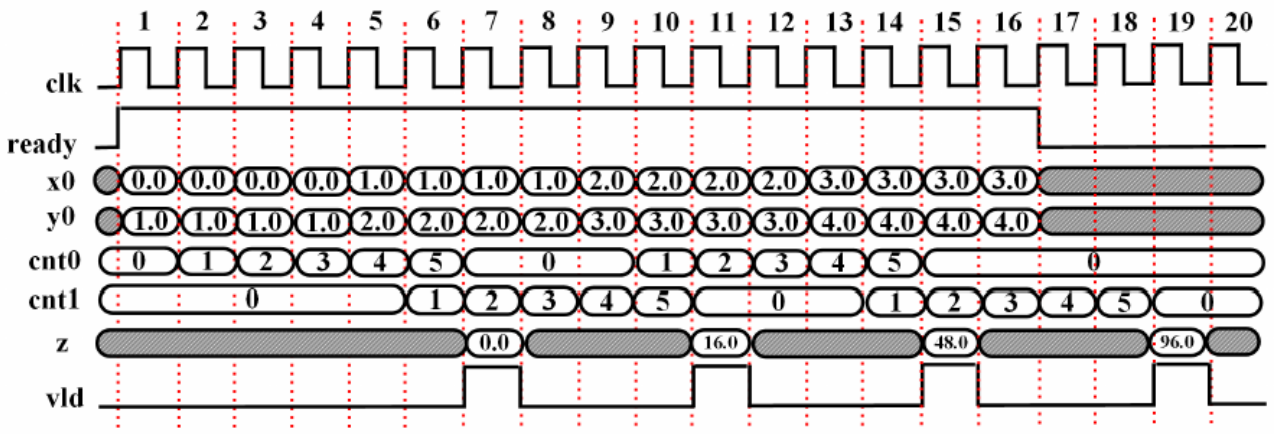
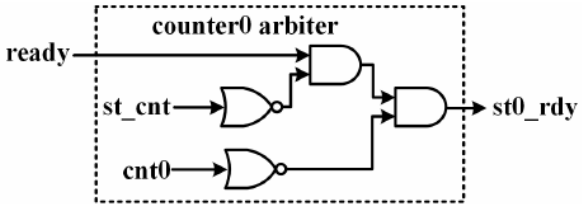


FIGURE 10.12
Timing Diagram of Iterative Design with Streaming Width Two

- 10.4.3 Timing Controller of Iterative Design with Streaming Width Two
  - ``st_cnt'' data stream counter
    - Track the progress of the four clock cycles required for each data frame input.
  - ``st0_rdy" start indicator
    - A NOT Reduction OR gate is utilized to distinguish the instances when the ``st_cnt" counter is decimal 2'd0.



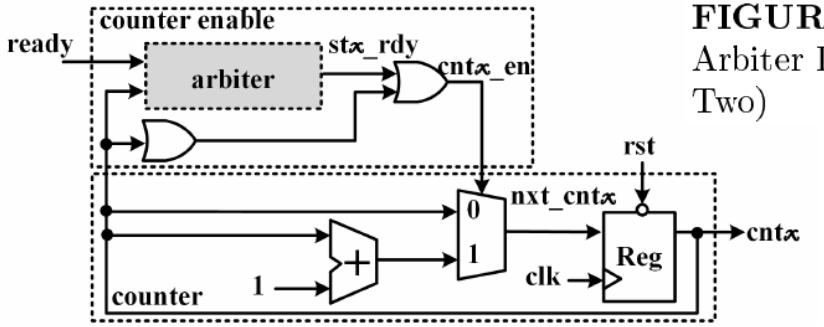


FIGURE 10.13
Arbiter Design of Timing Controller (Iterative Design with Streaming Width Two)

- 10.4.3 Timing Controller of Iterative Design with Streaming Width Two
  - The ``st_cnt'' counter is utilized to keep track of the 4-clock cycle duration of each data frame input.
  - The counter enable signals ``cnt0_en'', ``cnt1_en'', and ``cnt2_en'' are triggered the moment the corresponding start indicator is active.

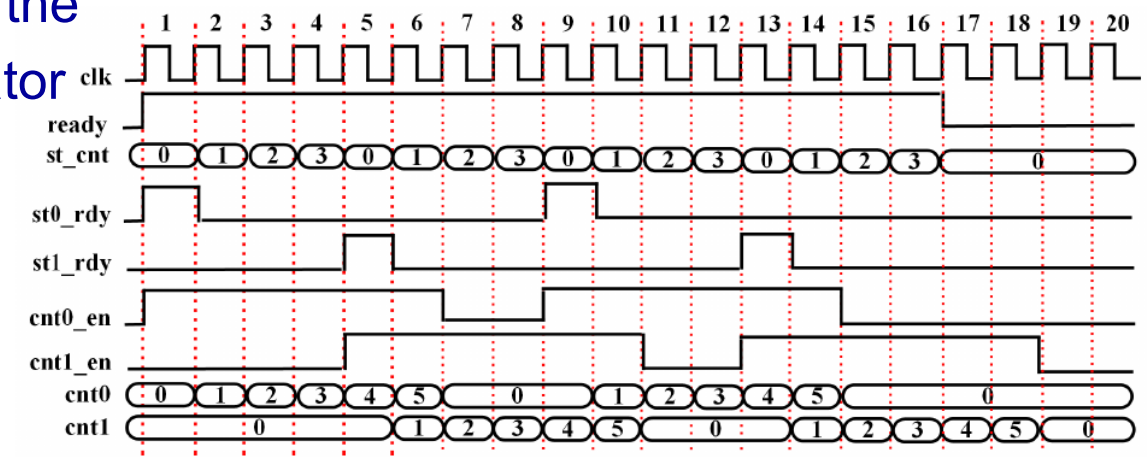


FIGURE 10.14
Timing Diagram of Timing Controller (Iterative Design with Streaming Width Two)

10.4 Iterative Design with Streaming Width Two

```
1  module iterative_st2_ddot (input          clk   ,
2                             input          rst   ,
3                             input          ready ,
4                             input  [31:0]  x0,x1 ,
5                             input  [31:0]  y0,y1 ,
6                             output reg     vld   ,
7                             output [31:0]  z     );
8  //**********************************
9  //****     Datapath Design *********
10 //**********************************
11 reg  [2:0]  cnt0, cnt1;
12 wire [31:0] p0, p1;
13 wire [31:0] s0, s1, s2, s3, s4, s5;
14 wire        nxt_vld;
15
16 // The first stage of multipliers
17 FP_multiplier u0_fp_mul(.clock    (clk  ),
18                         .reset    (rst  ),
19                         .io_in_a  (x0   ),
20                         .io_in_b  (y0   ),
21                         .io_out_s(p0    ));
22
23 FP_multiplier u1_fp_mul(.clock    (clk  ),
24                         .reset    (rst  ),
25                         .io_in_a  (x1   ),
26                         .io_in_b  (y1   ),
27                         .io_out_s(p1    ));
28
```

- 10.4.4 Verilog Code for Iterative Design with Streaming Width Two

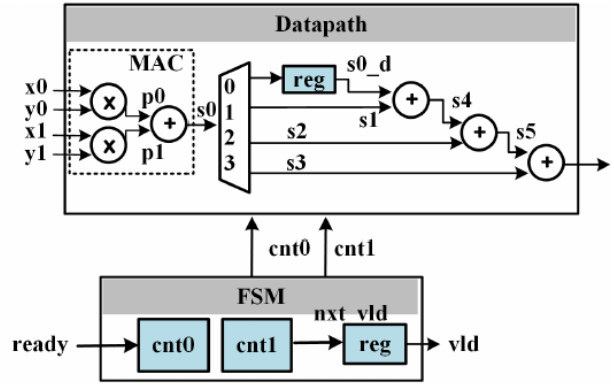


FIGURE 10.11
Iterative Design Structure with Streaming Width Two

10.4 Iterative Design with Streaming Width Two

```
29  // The second stage of adders
30  FP_adder u0_fp_add(.clock    (clk  ),
31                     .reset    (rst  ),
32                     .io_in_a  (p0   ),
33                     .io_in_b  (p1   ),
34                     .io_out_s(s0   ));
35
36  wire s0_d_flag = cnt0==3'd2|cnt1==3'd2;
37  wire [31:0] nxt_s0_d = s0_d_flag ? s0 : 32'h0;
38  reg  [31:0] s0_d;
39  always @(posedge clk) begin
40    if(rst) begin
41      s0_d<=32'h0    ;
42    end else begin
43      s0_d<=nxt_s0_d ;
44    end
45  end
46
47  assign s1 = (cnt0==3'd3|cnt1==3'd3) ? s0 : 31'h0;
48  assign s2 = (cnt0==3'd4|cnt1==3'd4) ? s0 : 31'h0;
49  assign s3 = (cnt0==3'd5|cnt1==3'd5) ? s0 : 31'h0;
```

```
51  // The third stage of adders
52  FP_adder u1_fp_add(.clock    (clk  ),
53                     .reset    (rst  ),
54                     .io_in_a  (s0_d ),
55                     .io_in_b  (s1   ),
56                     .io_out_s(s4   ));
57
58  // The fourth stage of adders
59  FP_adder u2_fp_add(.clock    (clk  ),
60                     .reset    (rst  ),
61                     .io_in_a  (s4   ),
62                     .io_in_b  (s2   ),
63                     .io_out_s(s5   ));
64
65  // The fifth stage of adders
66  FP_adder u3_fp_add(.clock    (clk  ),
67                     .reset    (rst  ),
68                     .io_in_a  (s5   ),
69                     .io_in_b  (s3   ),
70                     .io_out_s(z    ));
```

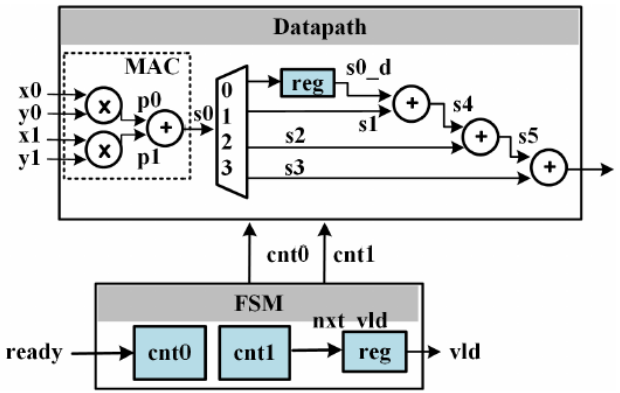


FIGURE 10.11
Iterative Design Structure with Streaming Width Two

10.4 Iterative Design with Streaming Width Two

```
72  //*****************************************
73  //****     FSM Controller Design *********
74  //*****************************************
75  reg  [1:0] st_cnt;
76  wire [1:0] nxt_st_cnt = ready ? st_cnt+2'd1 : st_cnt;
77  always @(posedge clk) begin
78    if(rst) begin
79      st_cnt<=2'd0       ;
80    end else begin
81      st_cnt<=nxt_st_cnt;
82    end
83  end
84
85  wire st0_rdy = ready & ~|st_cnt & ~|cnt0;
86  wire st1_rdy = ready & ~|st_cnt & ~|cnt1 & |cnt0 ;
87
88  wire cnt0_en = st0_rdy | |cnt0 ;
89  wire cnt1_en = st1_rdy | |cnt1 ;
90
91  wire [2:0] nxt_cnt0 = cnt0==3'd5 ? 3'd0 :
92                                cnt0_en ? cnt0+3'd1 : cnt0;
93  wire [2:0] nxt_cnt1 = cnt1==3'd5 ? 3'd0 :
94                                cnt1_en ? cnt1+3'd1 : cnt1;
```

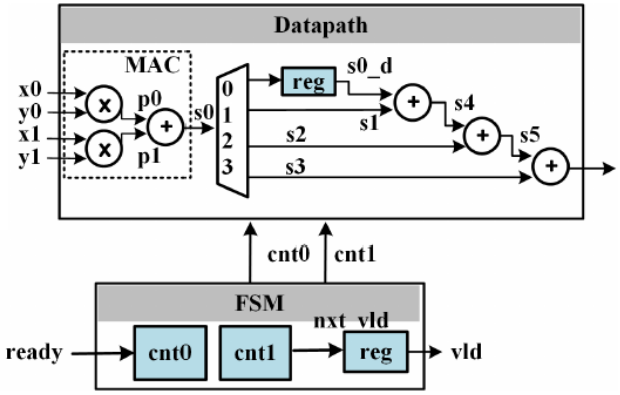


FIGURE 10.11
Iterative Design Structure with Streaming Width Two

10.4 Iterative Design with Streaming Width Two

```
95  always @(posedge clk) begin
96    if(rst) begin
97      cnt0<=3'd0    ;
98      cnt1<=3'd0    ;
99    end else begin
100     cnt0<=nxt_cnt0;
101     cnt1<=nxt_cnt1;
102   end
103 end
104
105 assign nxt_vld = cnt0==3'd5|cnt1==3'd5;
106 always @(posedge clk) begin
107   if(rst) begin
108     vld <=1'b0    ;
109   end else begin
110     vld <=nxt_vld ;
111   end
112 end
113 endmodule
```

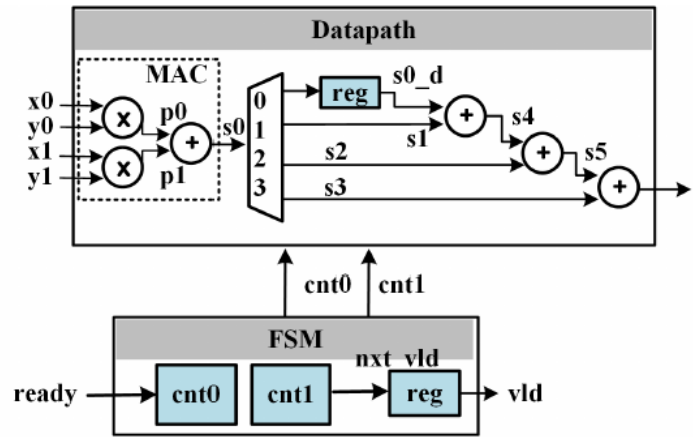


FIGURE 10.11
Iterative Design Structure with Streaming Width Two