[SoC Design]

HW Design for FFT (Lab2)

Chester Sungchung Park (박성정)

SoC Design Lab, Konkuk University

Webpage: http://soclab.konkuk.ac.kr

Outline

- Objectives
- Hardware architecture for FFT
  - Single-path delay feedback (SDF)
- Lab 2: H/W Design

Objectives

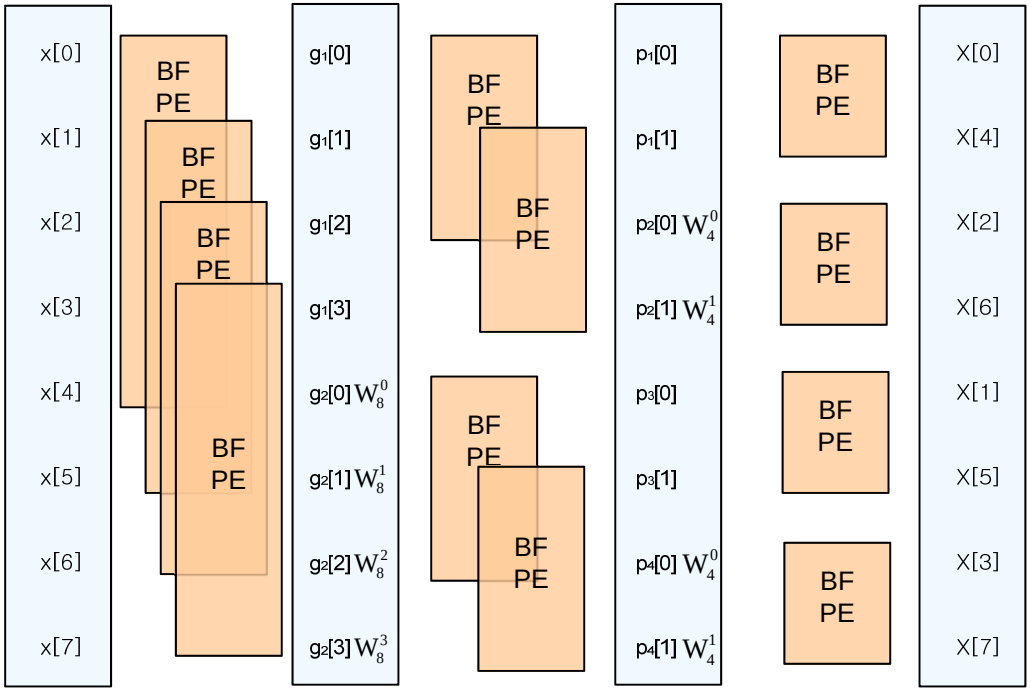
- After completing this lab, you will be able to:
  - Create a Vivado RTL project
  - Synthesize the RTL project
  - Implement the RTL project
  - Run Behavioral / Post-Synthesis simulations

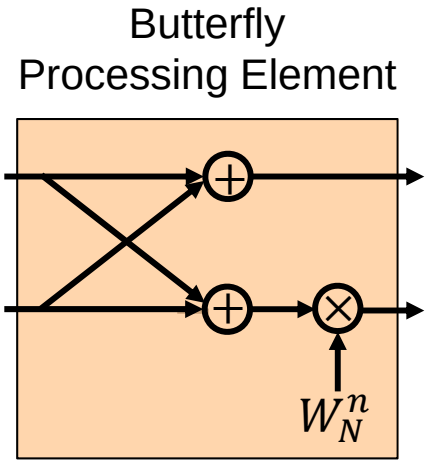
Hardware Architecture for FFT

- Parallel architectures
  - No-brainer implementation
- Pipeline architectures
  - Single-path Delay Feedback (SDF)

Parallel Architecture

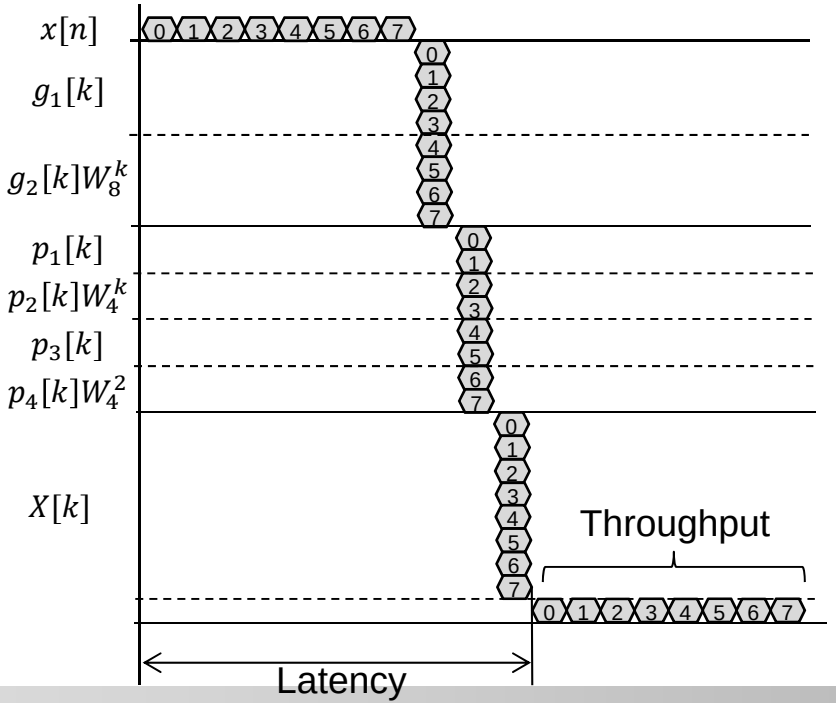
- Block diagram
  - Totally $N/2$ butterfly processing elements per stage
  - Example: 8-pt FFT ($N = 8$)





Parallel Architecture

- Performance metrics
  - Latency: $(N + \log_2 N)T_{clk}\ [sec]$
    - ✓ Reordering ignored
  - Throughput: $1/T_{clk}[samples/sec]$
  - Example: 8-pt FFT ($N = 8$)



Parallel Architecture

- Complexity metrics: S/W case
  - No. of instruction cycles

Input Loading

```
for (data=0;data<N;data++) {
    input[data].real_no=in_re[data];
    input[data].img_no=in_im[data];
}
```

Stage #1

```
for (n=0;n<32;n++) {
    temp[n]=add_cal2(input[n],input[n+32]);
    temp[n+32]=multiple2(sub_cal2(input[n],input[n+32]),W2[n])
    temp[n+32].real_no /= TW_factor;
    temp[n+32].img_no /= TW_factor;
}
```

.....

Stage #6

```
for(n=0;n<32;n++) {
    input[(2*n)]=add_cal2(temp[(2*n)],temp[(2*n)+1]);
    input[(2*n)+1]=sub_cal2(temp[(2*n)],temp[(2*n)+1]);
}
```

Reordering + Output Loading

```
for(n=0;n<N;n++) {
    X[n]=input[bit_revesal(n)];
}
for(n=0;n<N;n++) {
    out_re[n] = X[n].real_no;
    out_im[n] = X[n].img_no;
}
```

S/W Profiling (Cortex-A9)

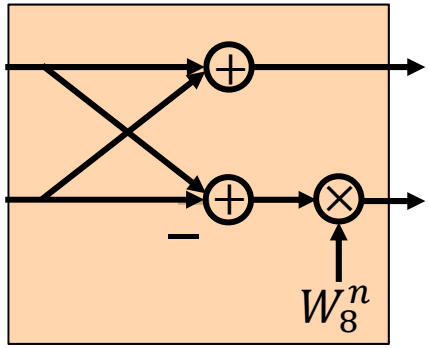
Functions	Execution Time	
	usec	%
Input	3.7	5.2
Stage #1	9.0	12.6
Stage #2	10.7	15.0
Stage #3	10.5	14.7
Stage #4	10.5	14.7
Stage #5	10.5	14.6
Stage #6	5.2	7.3
Reordering + Output	11.5	16.1
Total	71.7	100

Parallel Architecture

- Complexity metrics: H/W case
  - Area

	BF-PE	Register
4	4	8
8	12	24
16	32	64
64	192	384
N	$(N/2)\ log_2 N$	$N\ log_2 N$

Butterfly
Processing Element

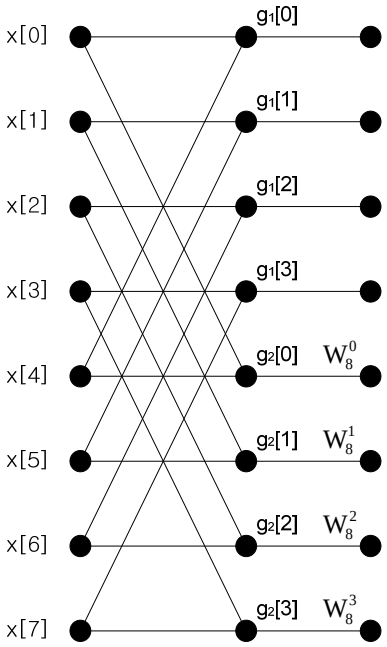


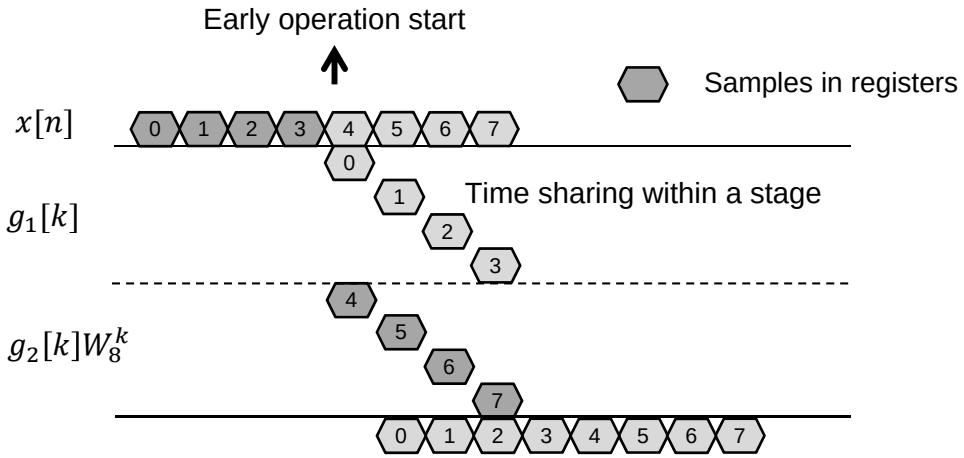
1 multipliers & 2 adders

Necessary to reduce No. of BF PEs
In order to reduce complexity (area)

Single-Path Delay Feedback

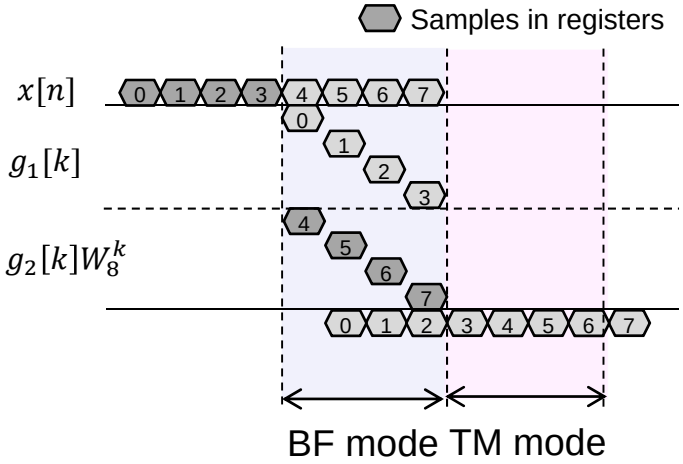
- Pipeline architecture
  - Efficient use of BF-PE
    - ✓ Time sharing within a stage
    - ✓ Early operation start
  - Example: 8-pt FFT ($N = 8$)



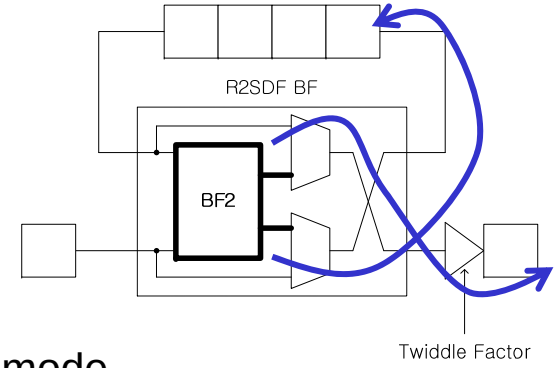


Single-Path Delay Feedback

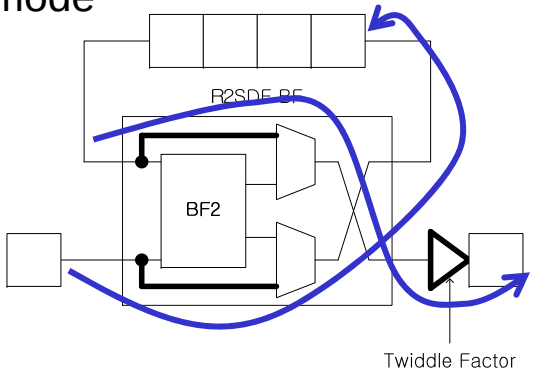
- Block diagram
  - One BF-PE + fewer registers per stage
  - Switching btw. BF/TM modes
  - Example: 8-pt FFT ($N = 8$)



BF mode

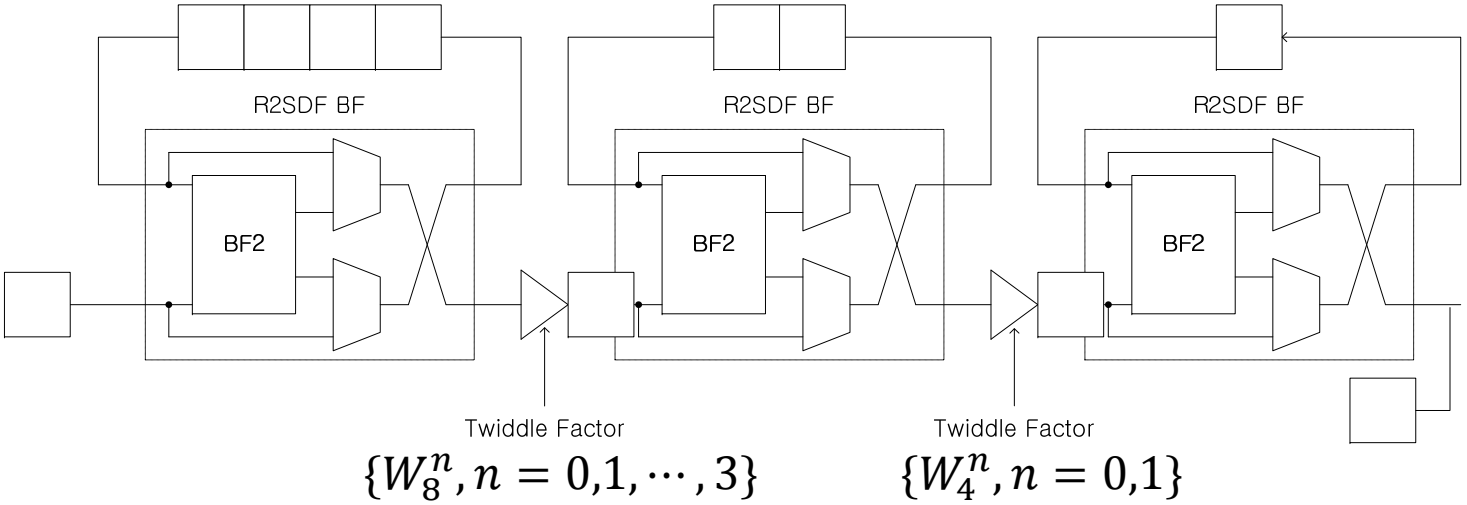


TM mode



Single-Path Delay Feedback

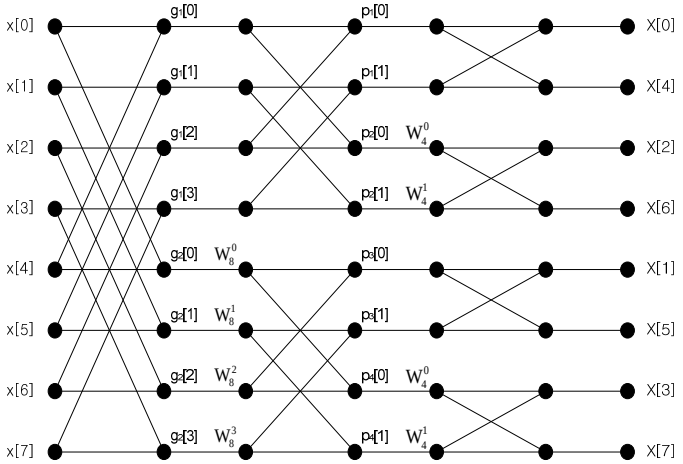
- Block diagram
  - Example: 8-pt FFT ($N = 8$)

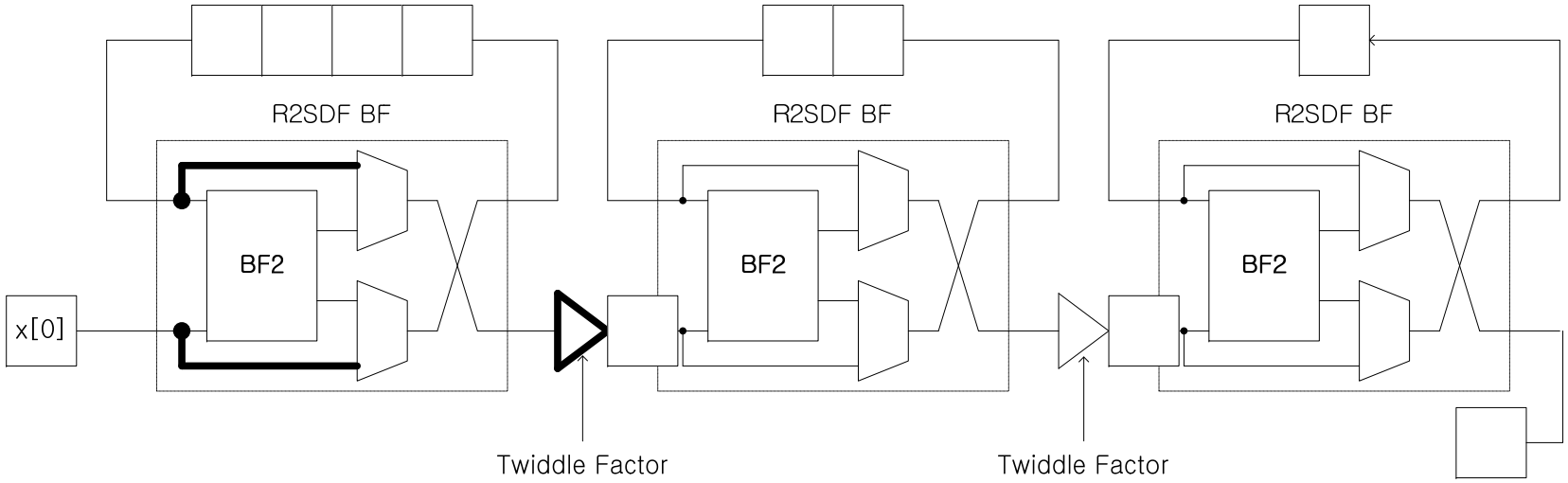


[E. H.Wold et al., IEEE T-Computer, 1984]

Single-Path Delay Feedback

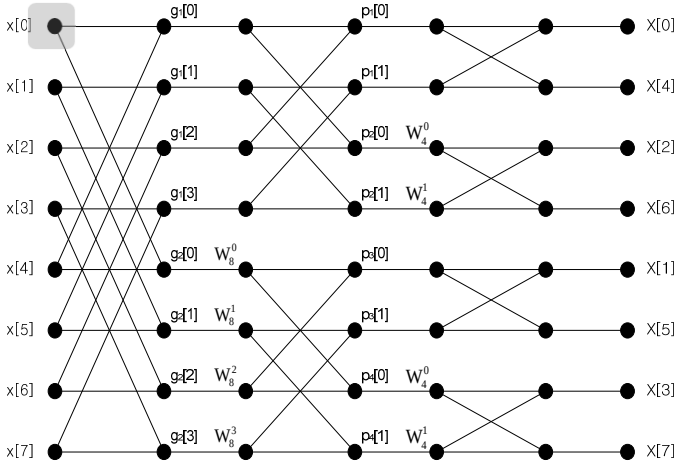
- Pipeline operation
  - Clock cycle 0

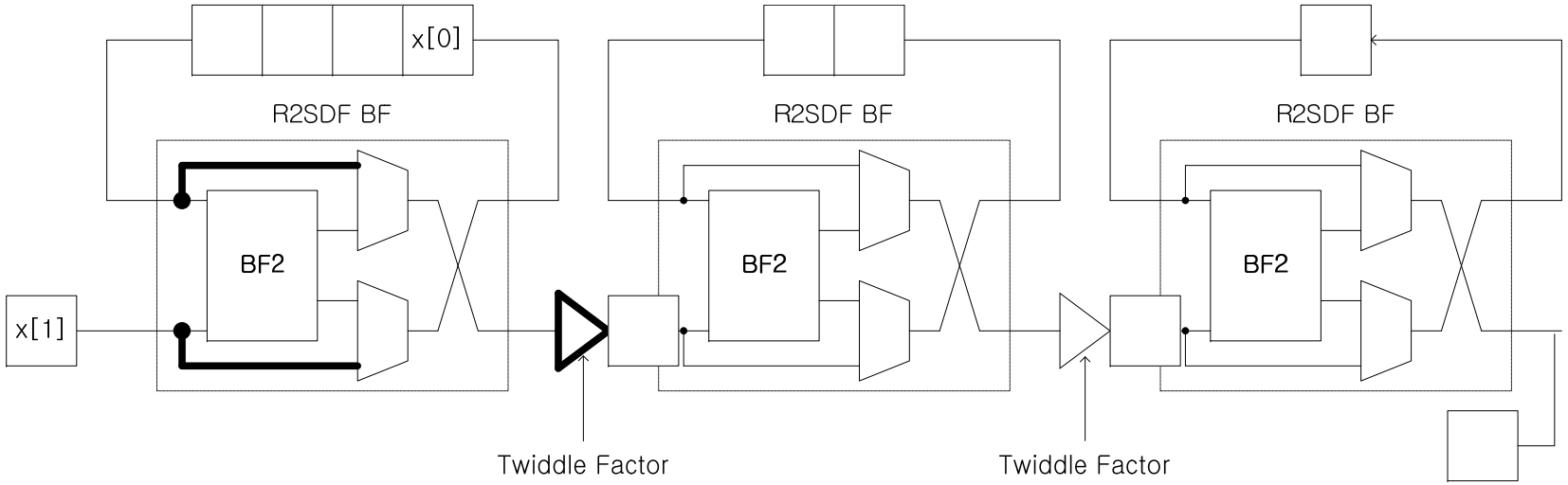




Single-Path Delay Feedback

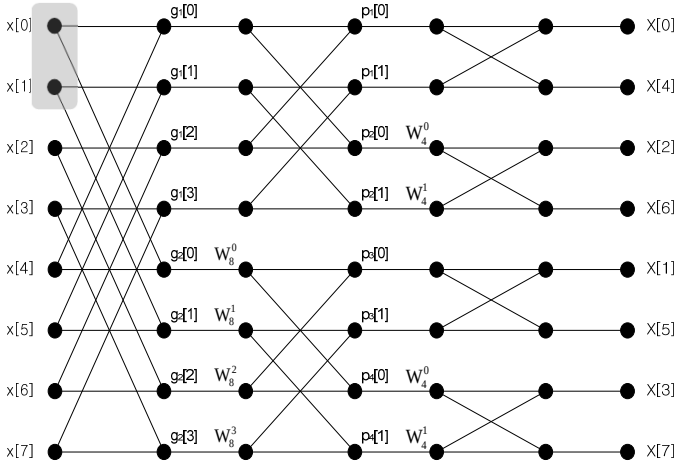
- Pipeline operation
  - Clock cycle 1

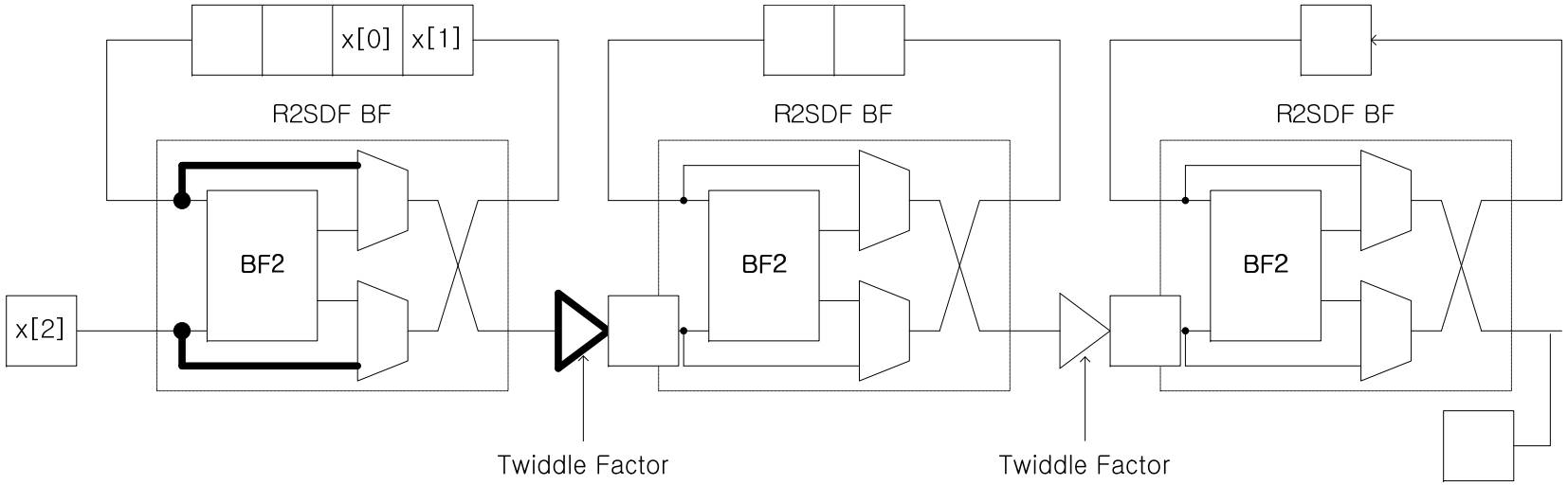




Single-Path Delay Feedback

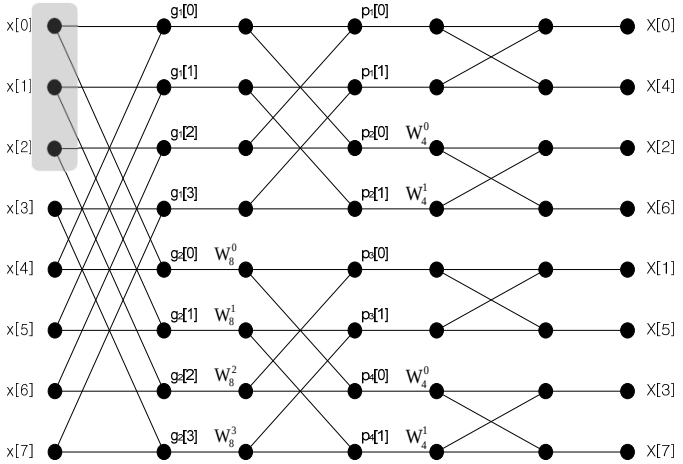
- Pipeline operation
  - Clock cycle 2

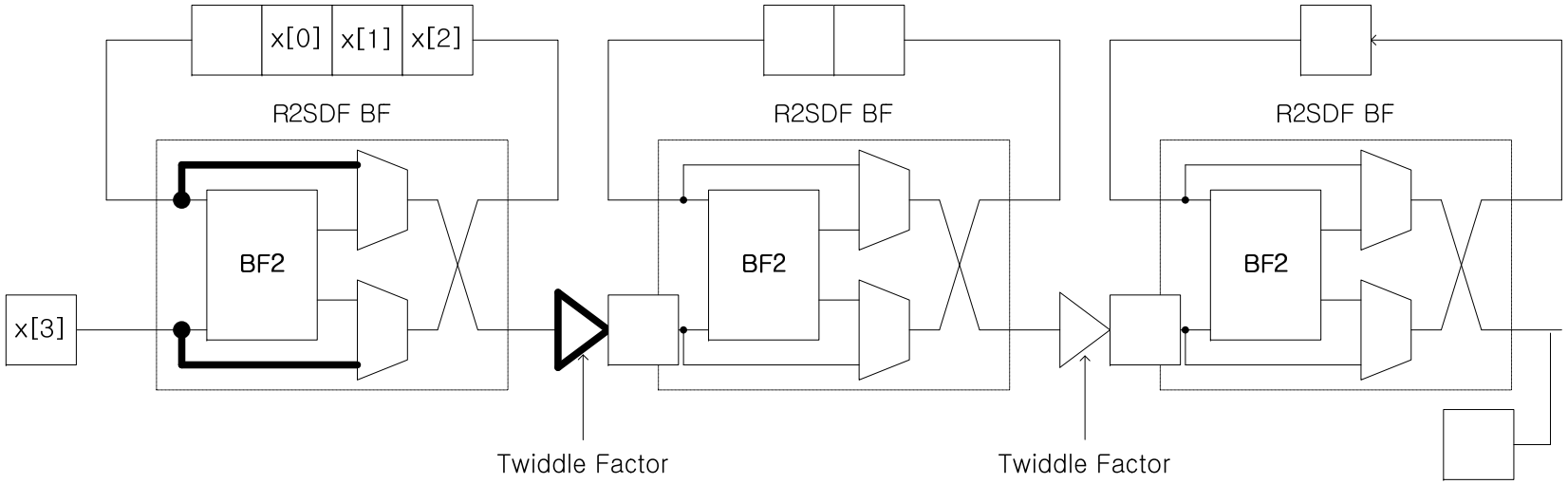




Single-Path Delay Feedback

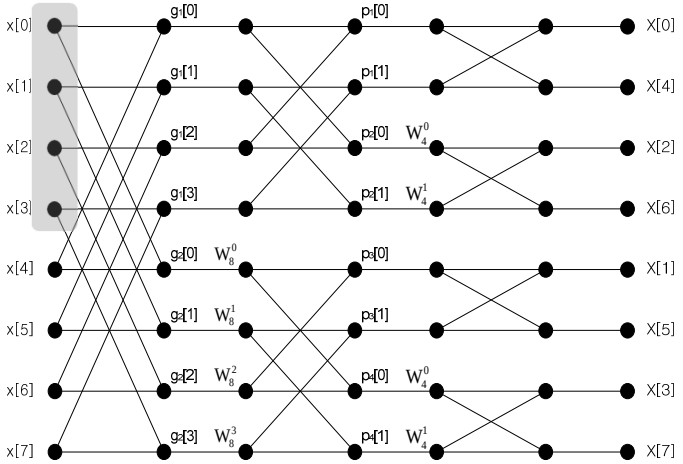
- Pipeline operation
  - Clock cycle 3

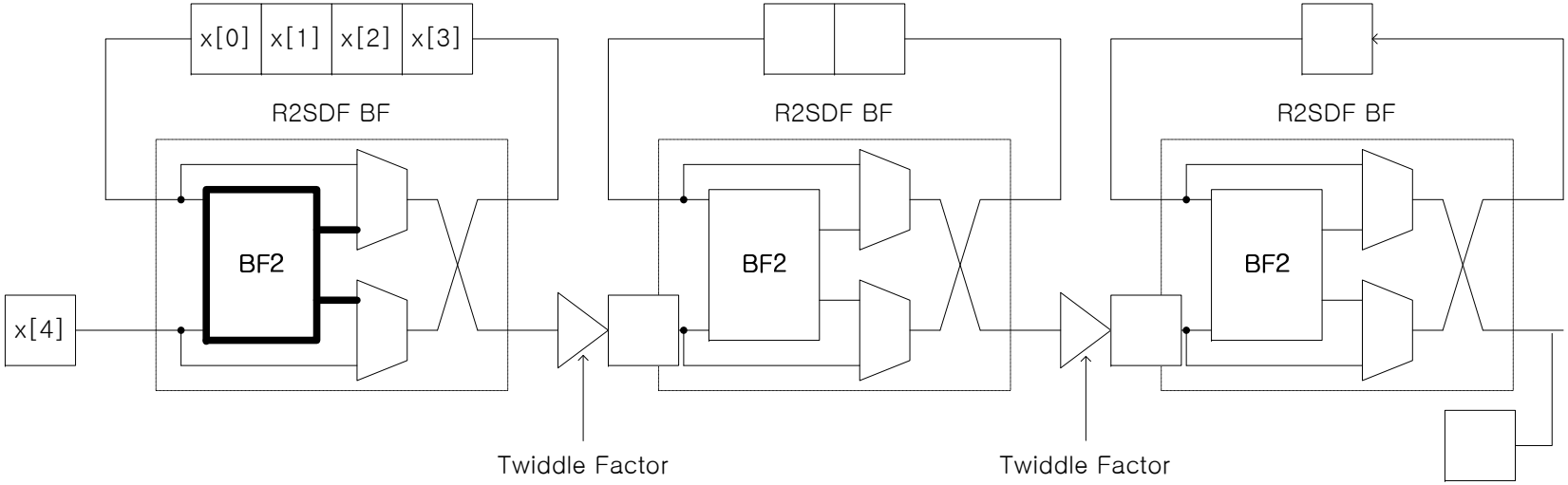




Single-Path Delay Feedback

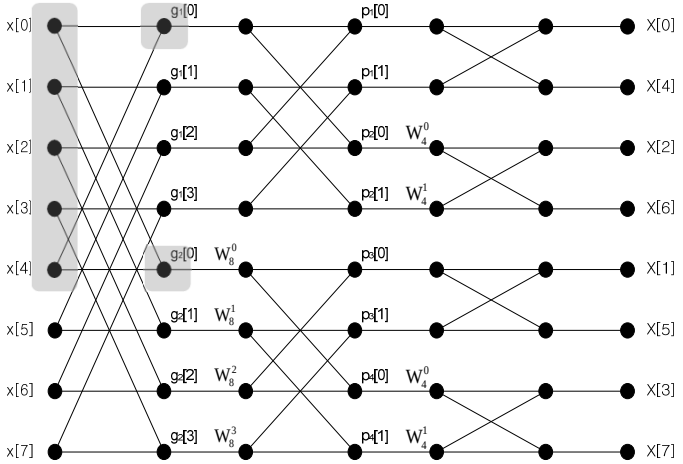
- Pipeline operation
  - Clock cycle 4

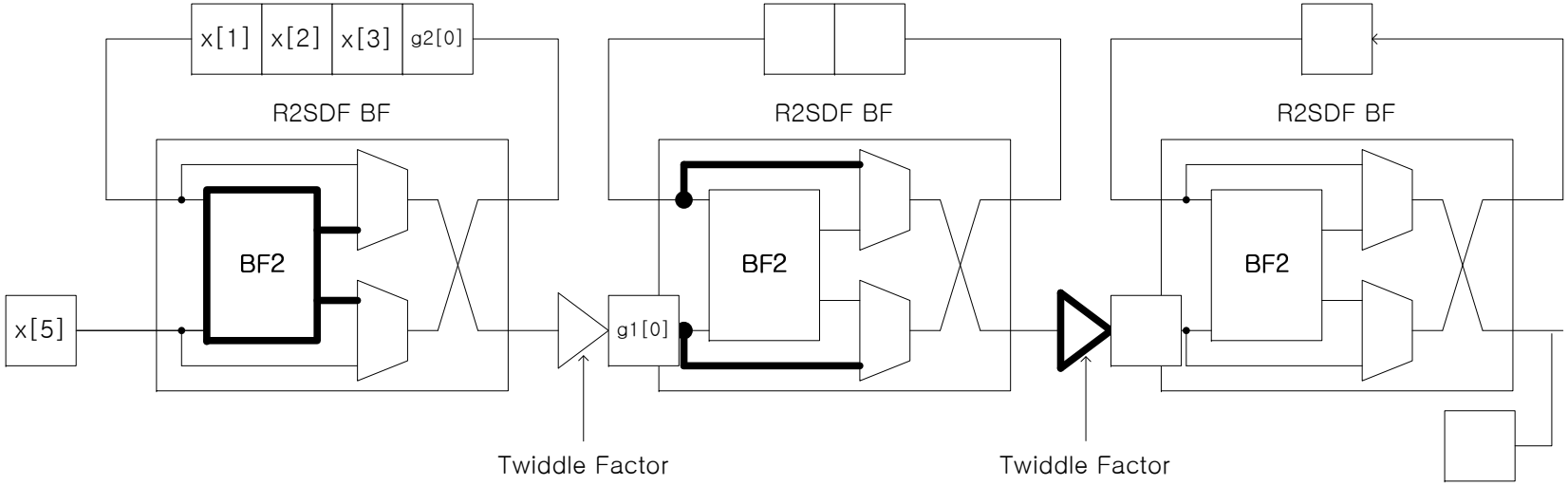




Single-Path Delay Feedback

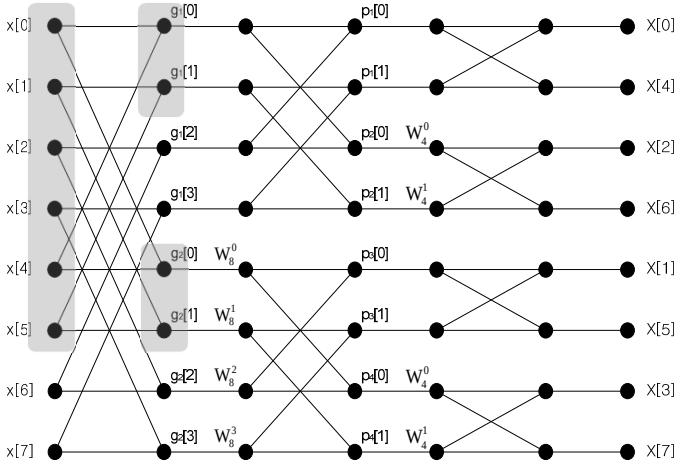
- Pipeline operation
  - Clock cycle 5

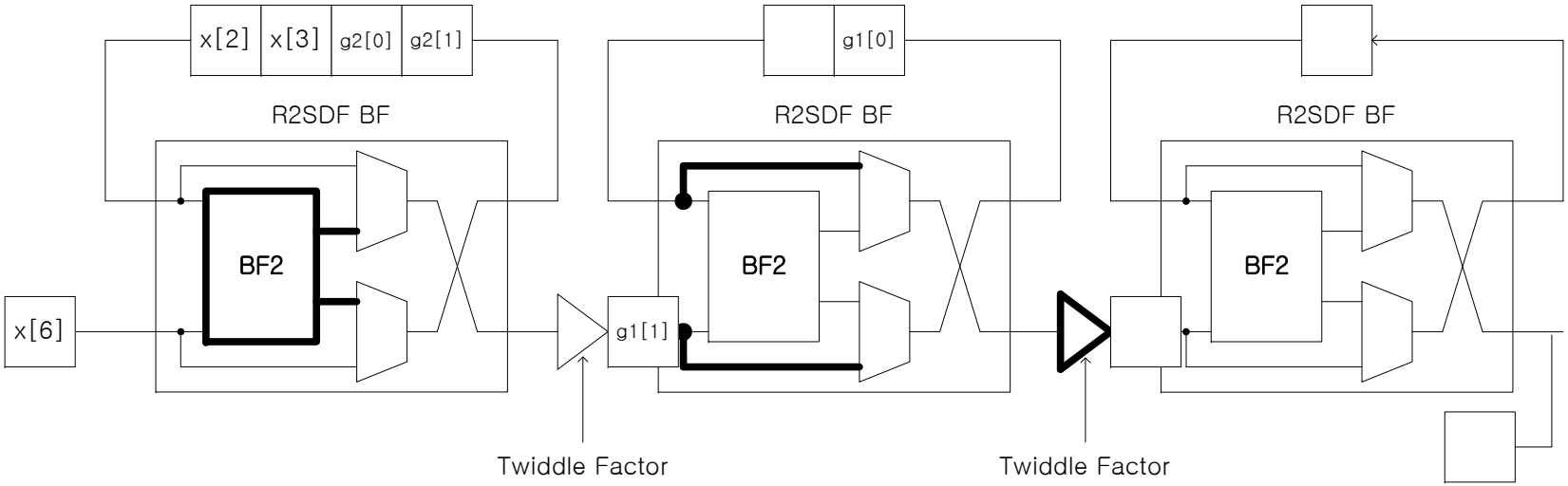




Single-Path Delay Feedback

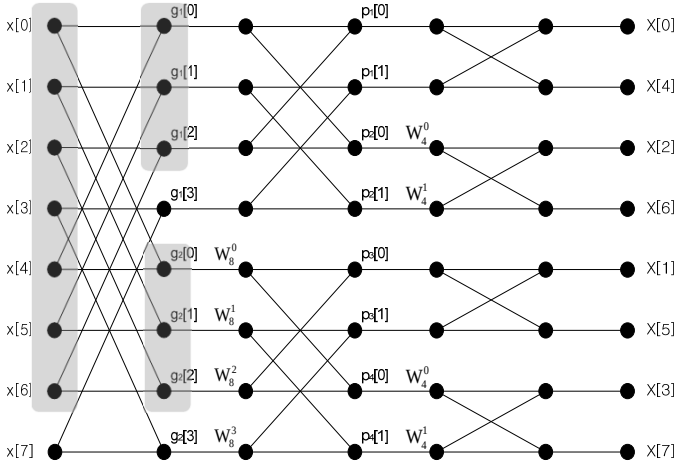
- Pipeline operation
  - Clock cycle 6

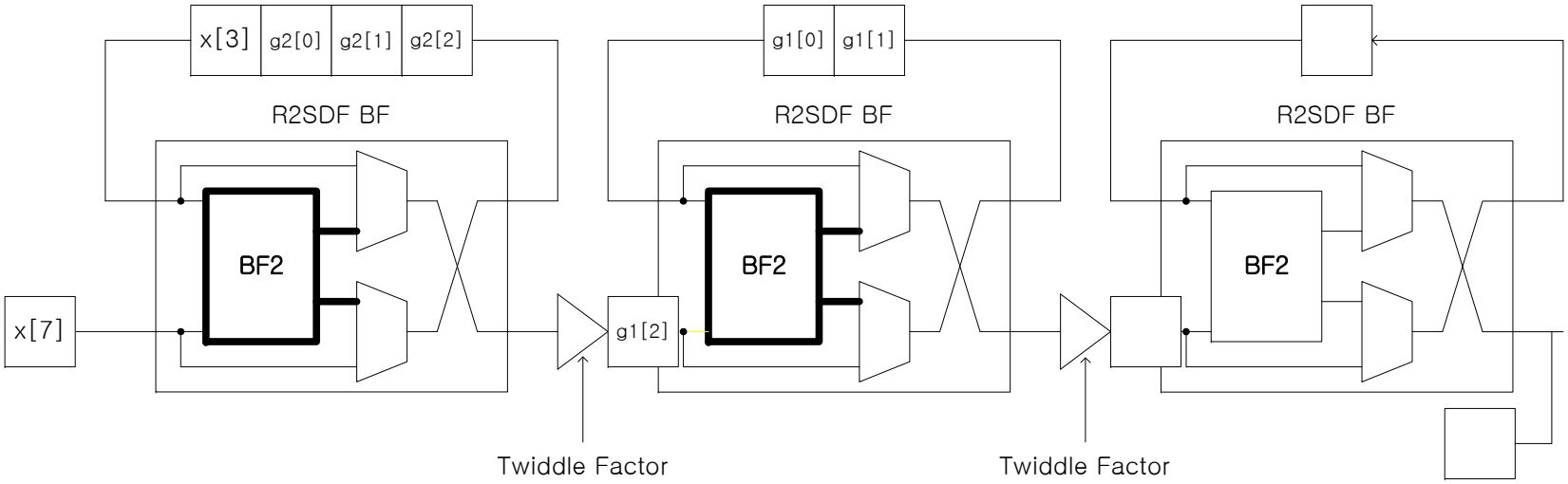




Single-Path Delay Feedback

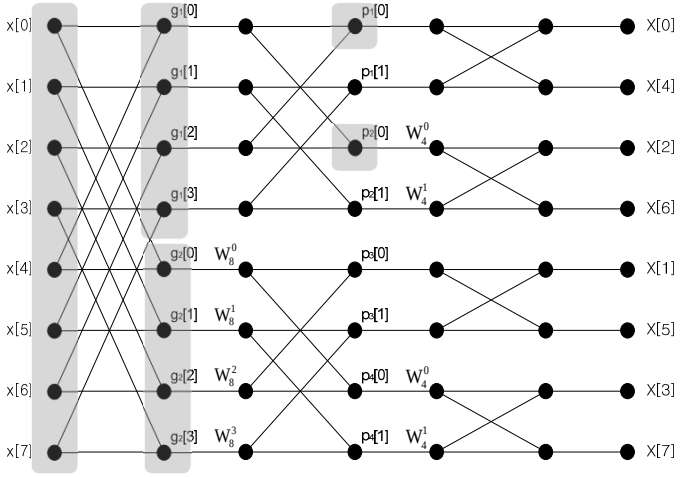
- Pipeline operation
  - Clock cycle 7

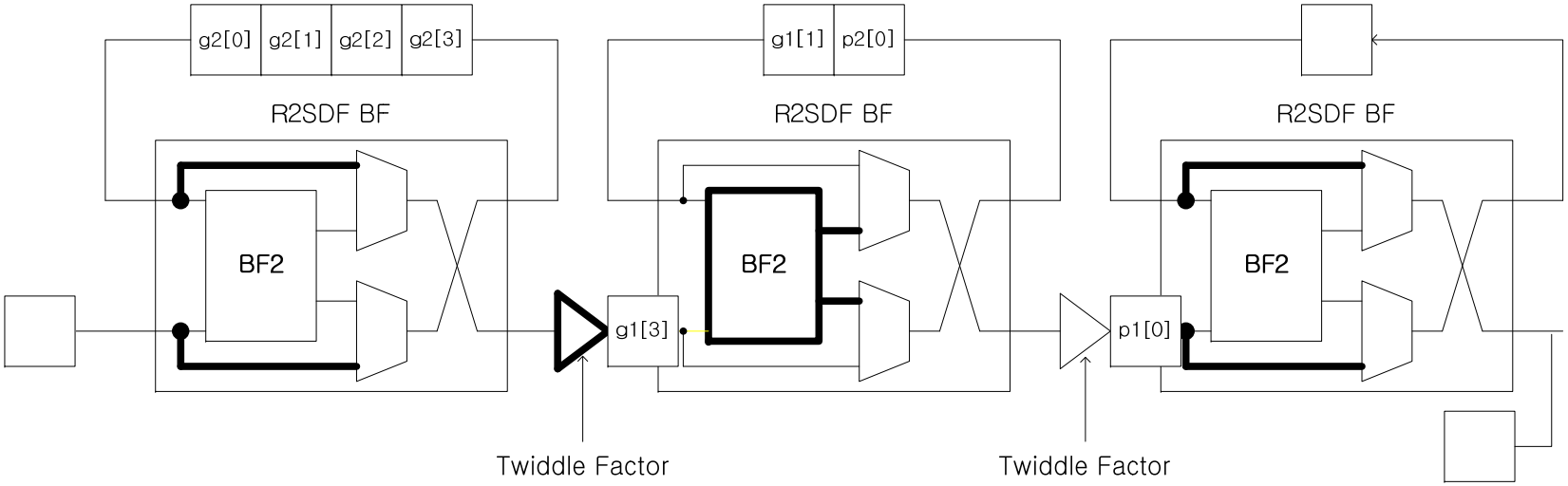




Single-Path Delay Feedback

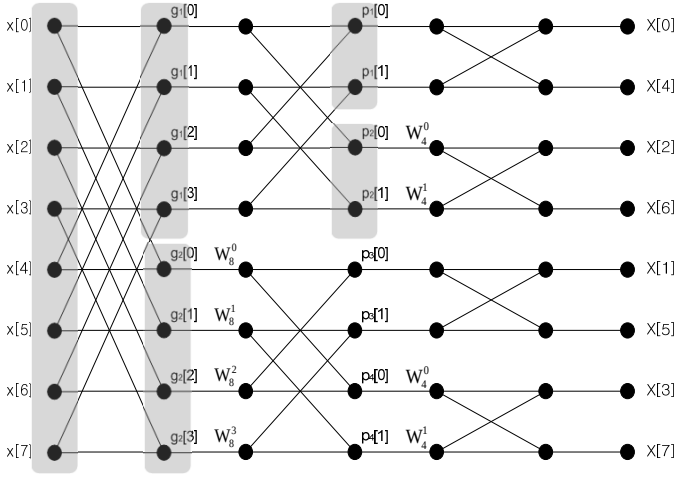
- Pipeline operation
  - Clock cycle 8

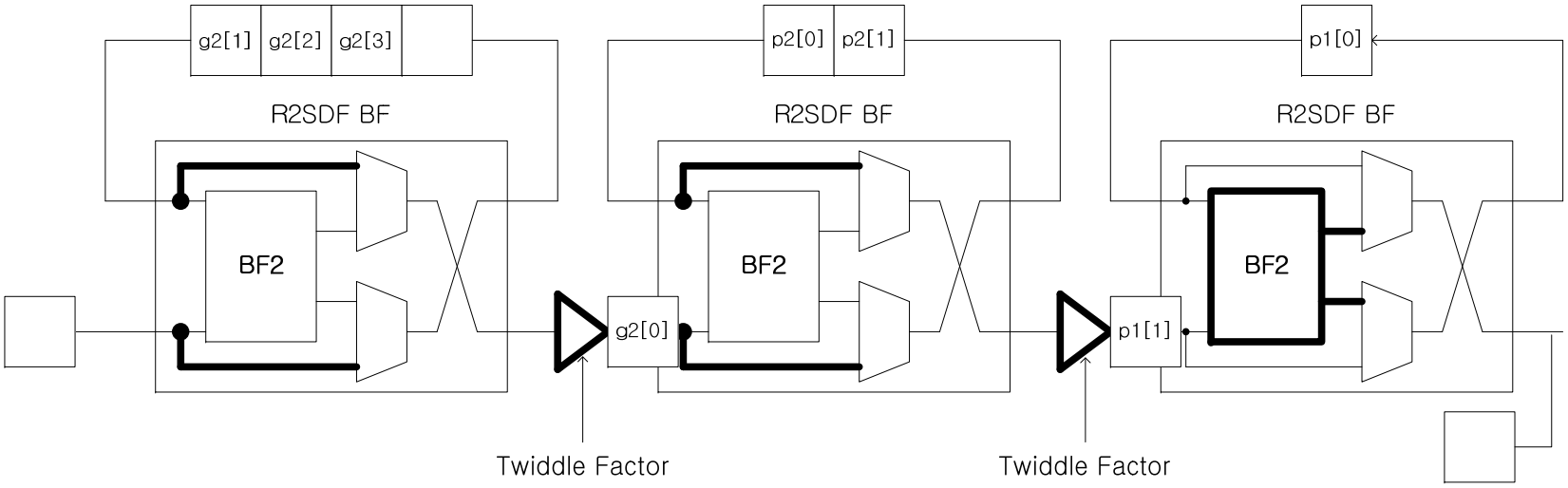




Single-Path Delay Feedback

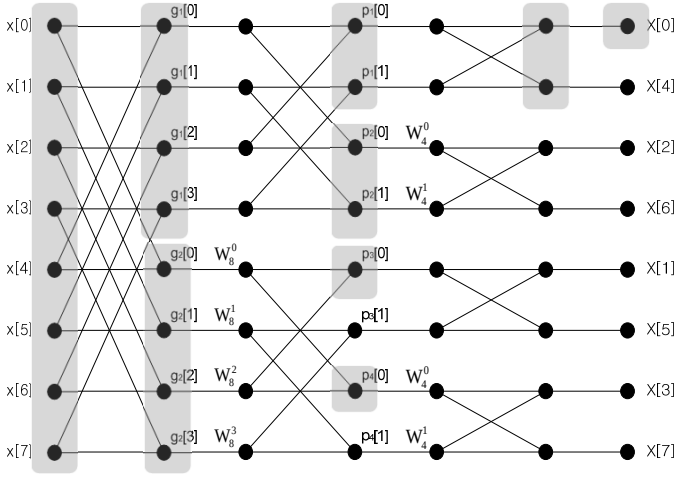
- Pipeline operation
  - Clock cycle 9

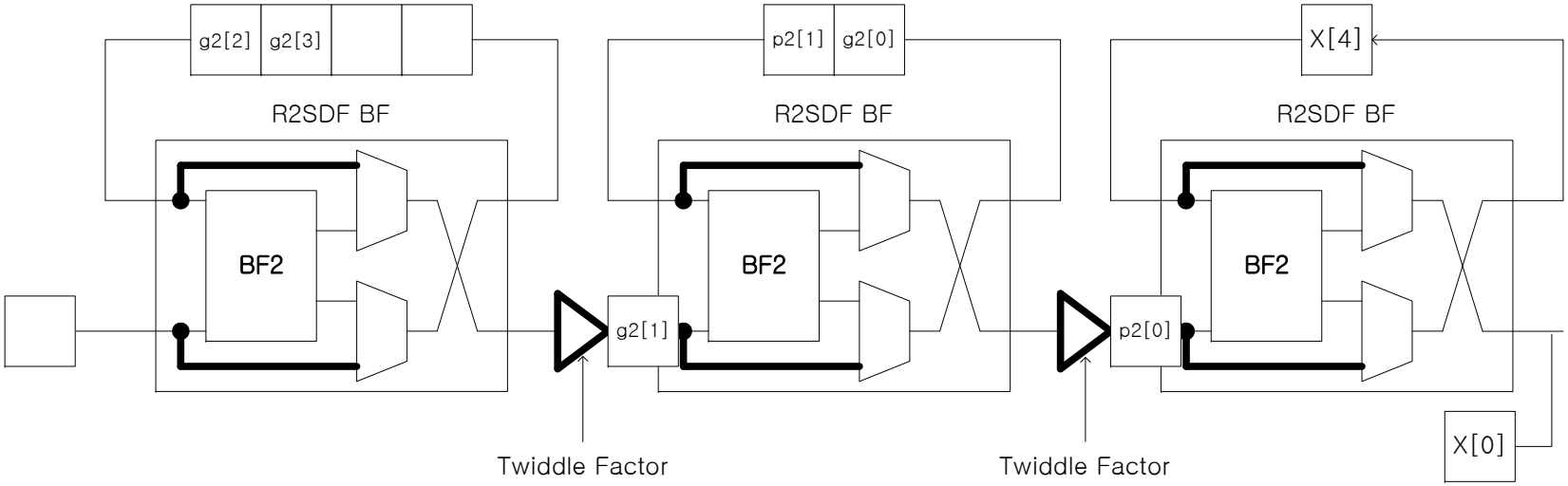




Single-Path Delay Feedback

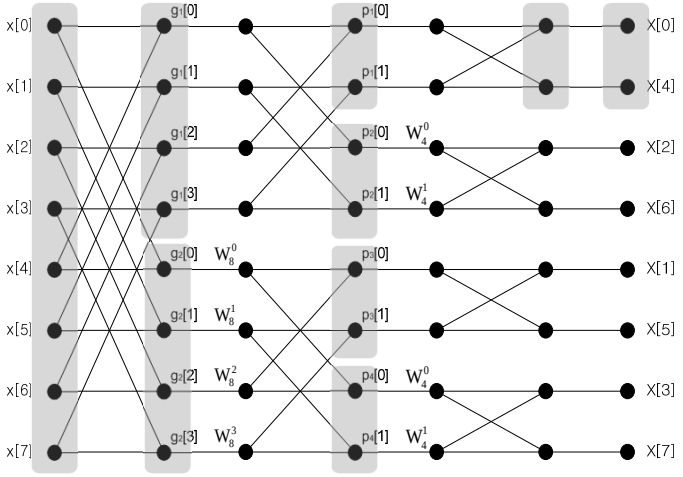
- Pipeline operation
  - Clock cycle 10

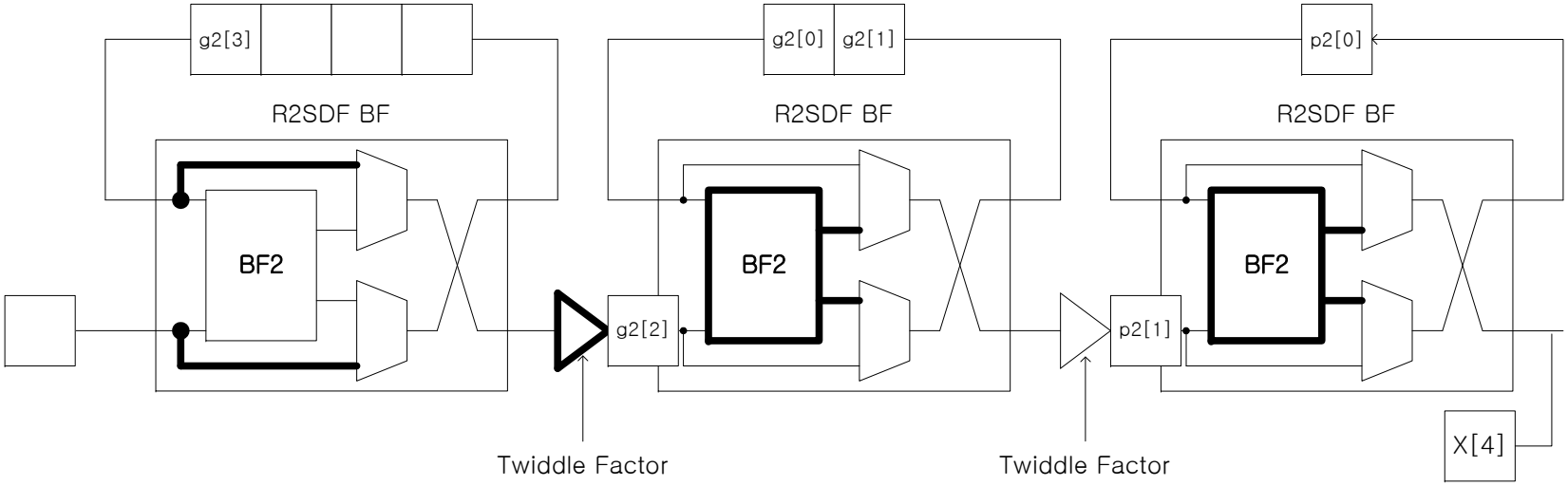




Single-Path Delay Feedback

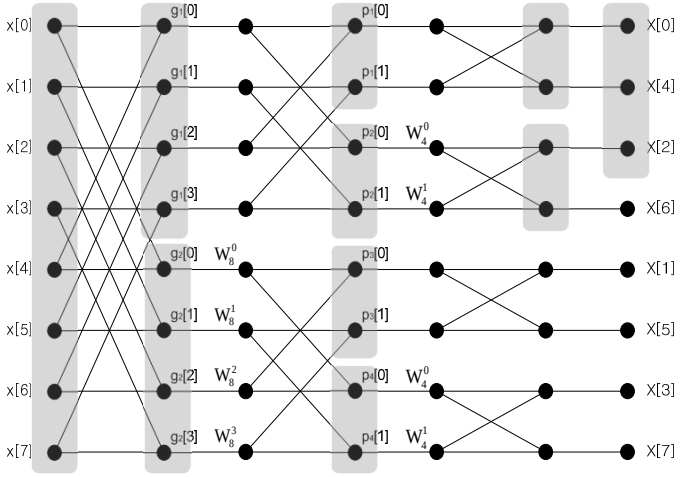
- Pipeline operation
  - Clock cycle 11

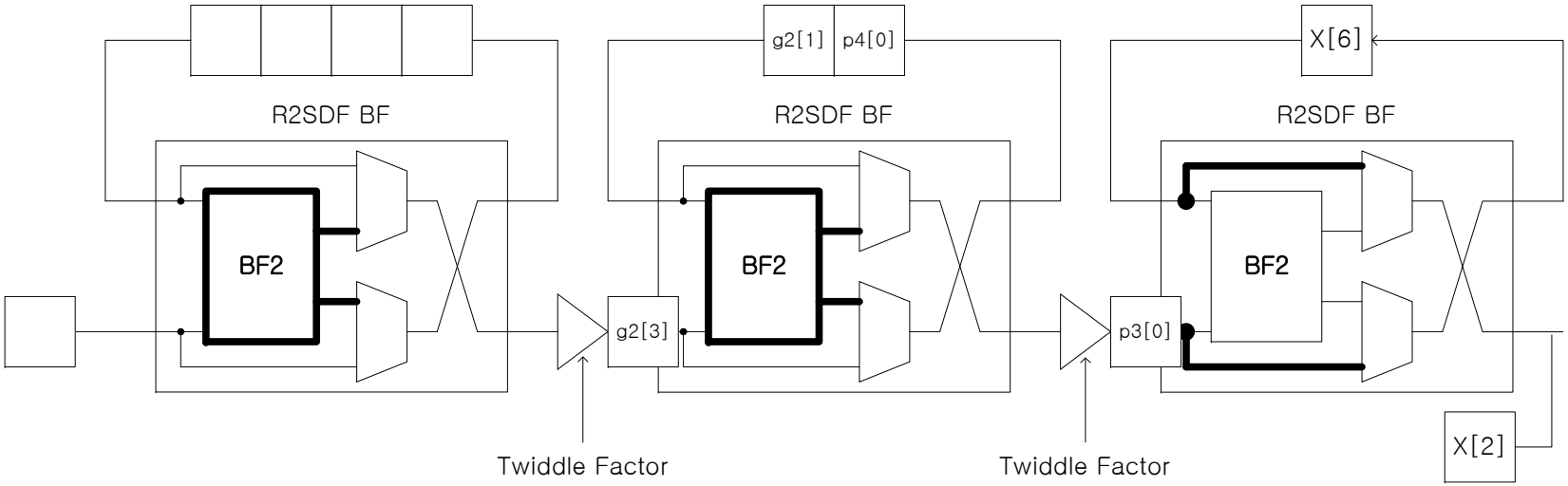




Single-Path Delay Feedback

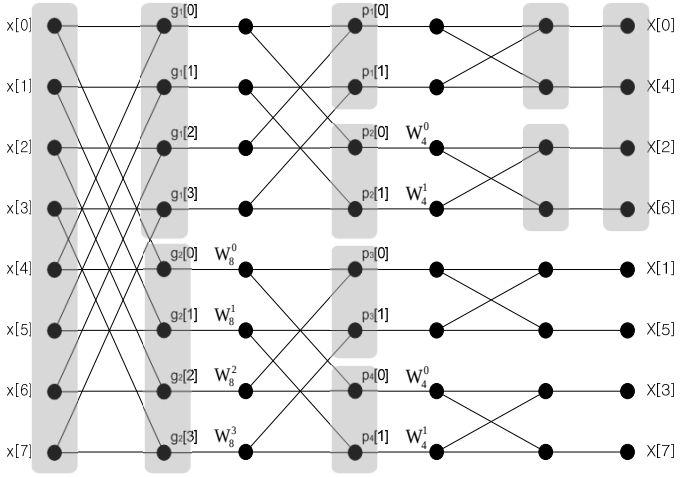
- Pipeline operation
  - Clock cycle 12

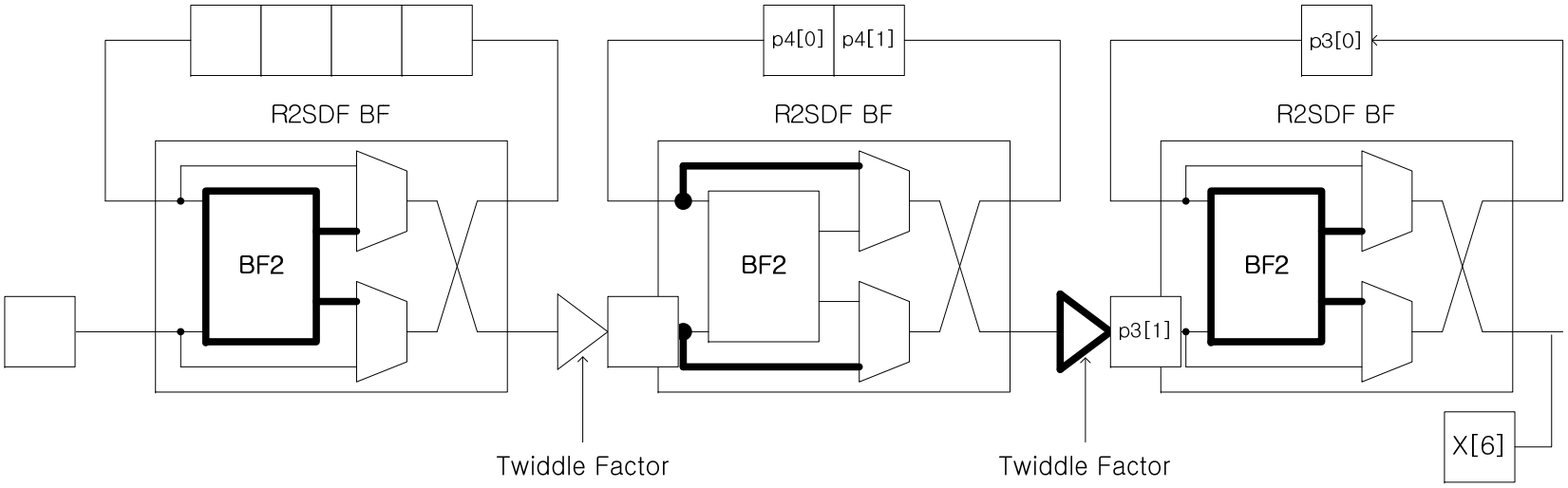




Single-Path Delay Feedback

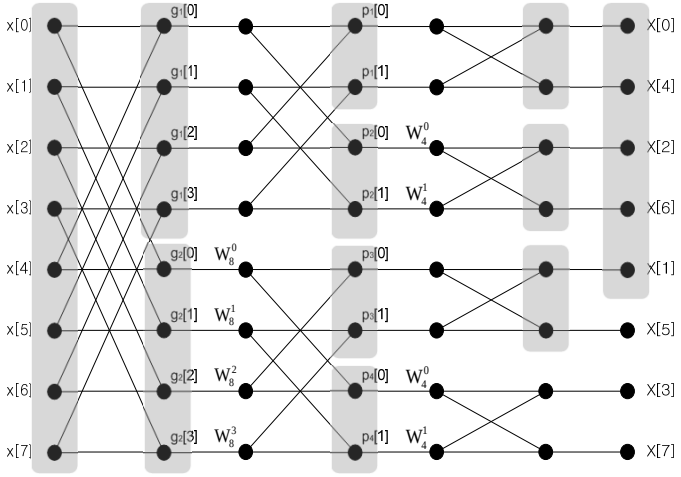
- Pipeline operation
  - Clock cycle 13

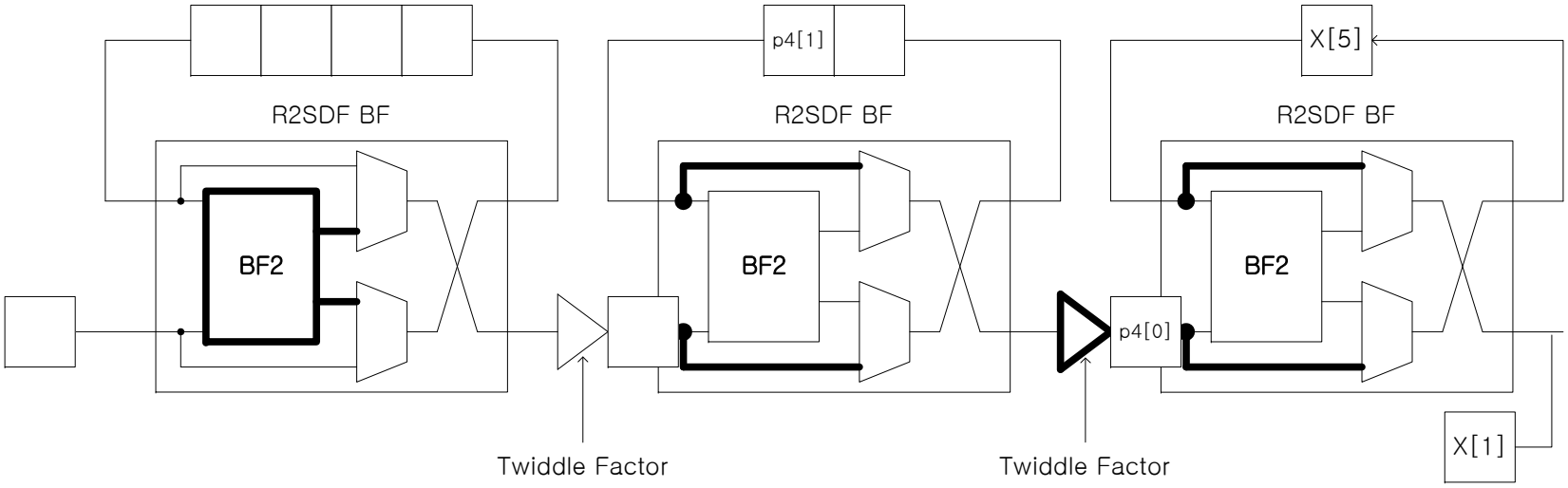




Single-Path Delay Feedback

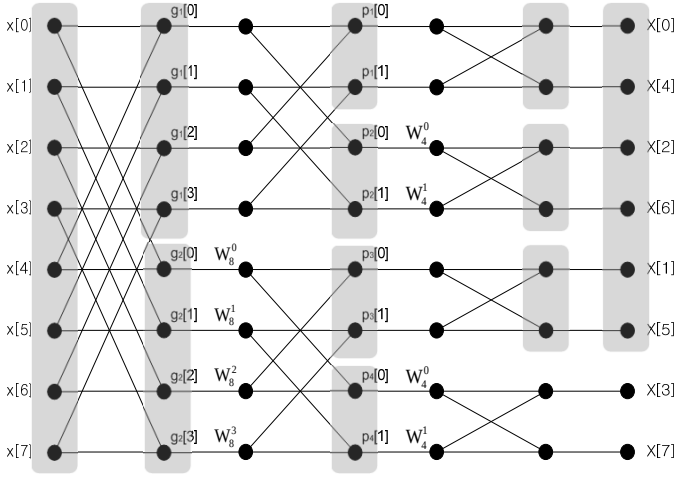
- Pipeline operation
  - Clock cycle 14

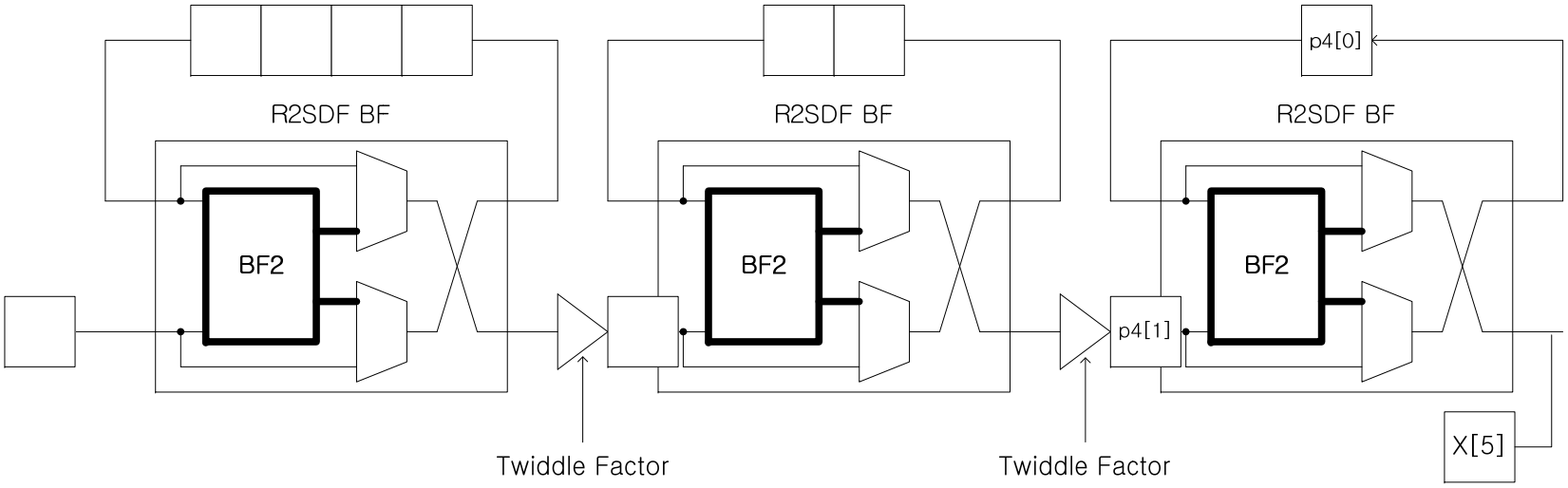




Single-Path Delay Feedback

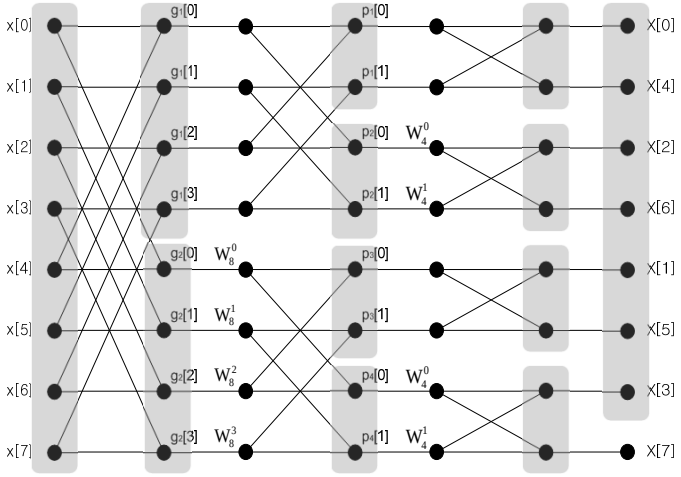
- Pipeline operation
  - Clock cycle 15

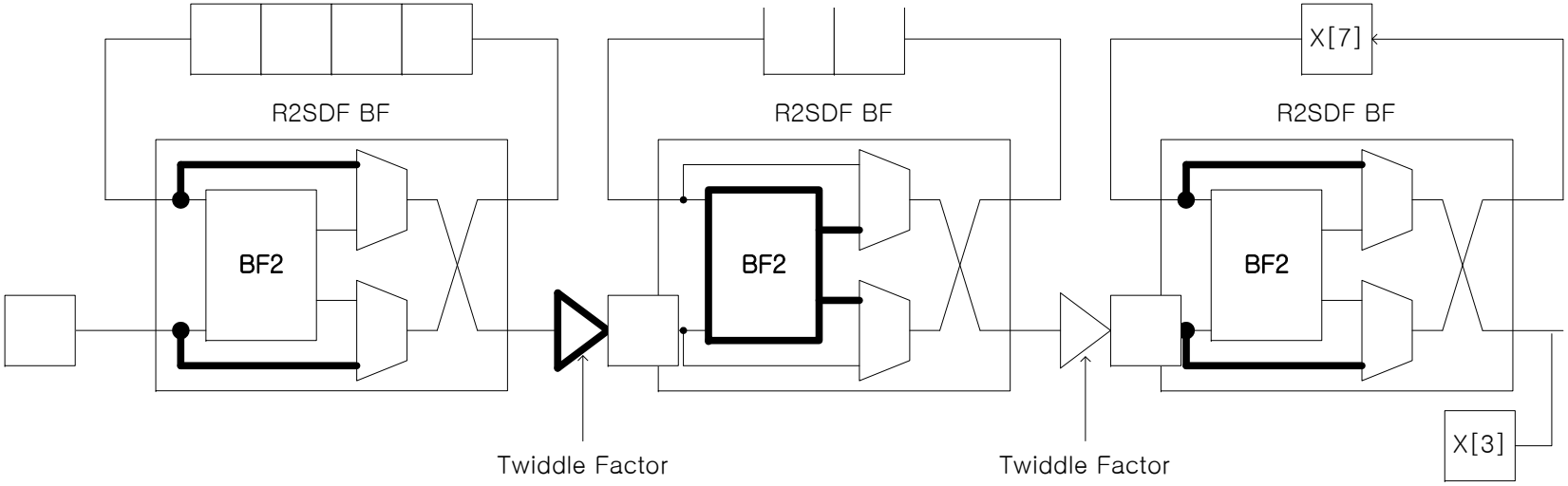




Single-Path Delay Feedback

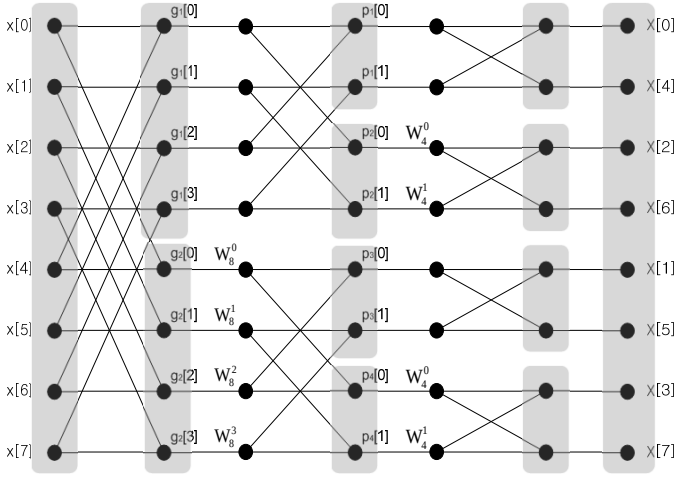
- Pipeline operation
  - Clock cycle 16

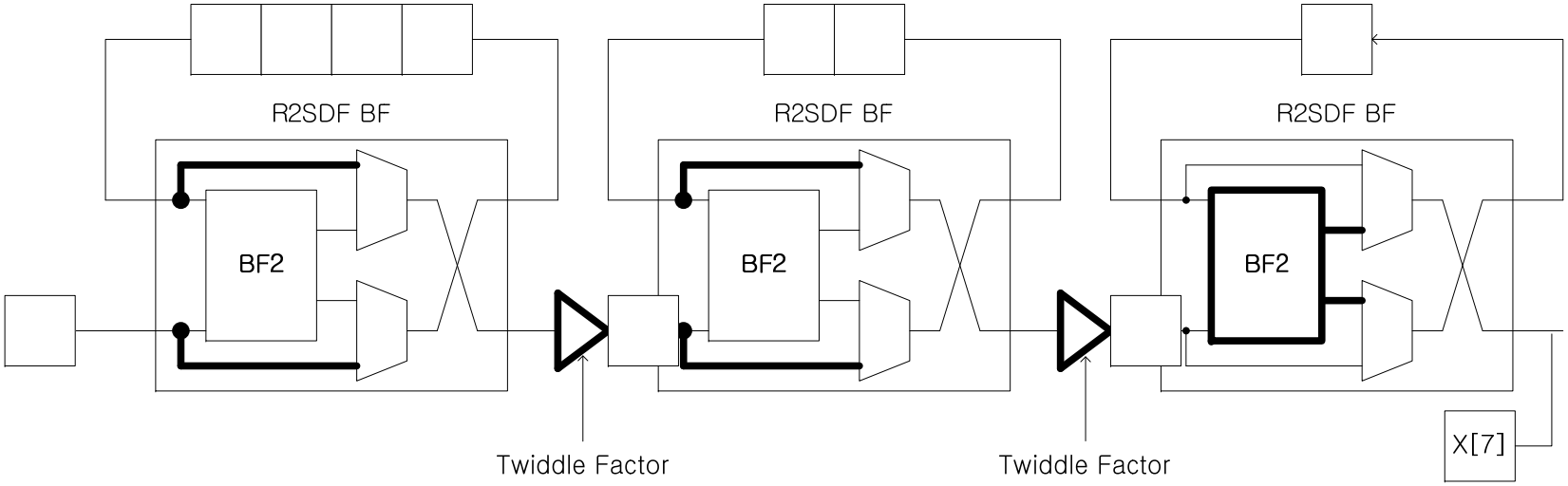




Single-Path Delay Feedback

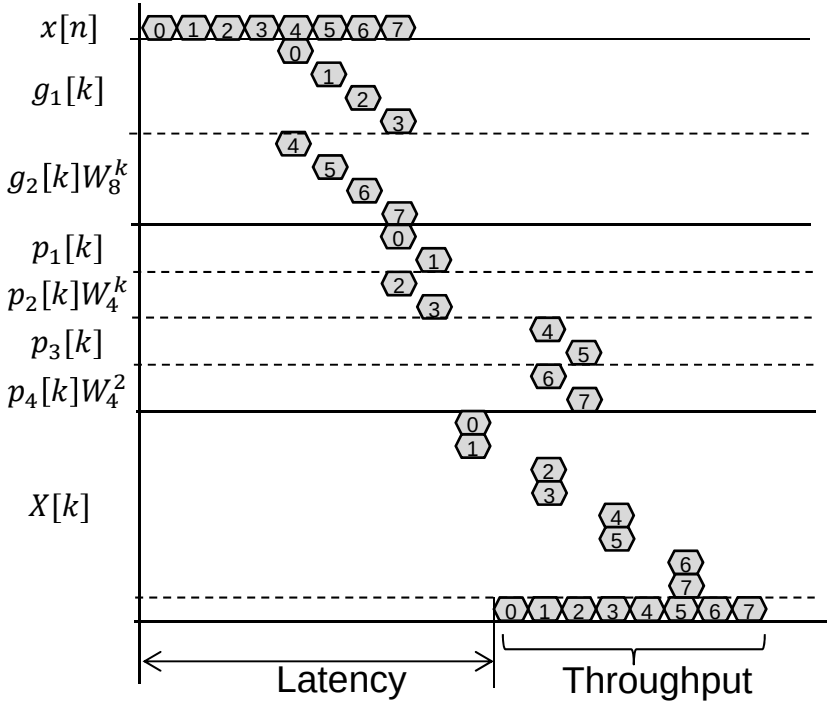
- Pipeline operation
  - Clock cycle 17





Single-Path Delay Feedback

- Performance metrics
  - Latency: $\left((N-1)+\log_2 N\right)T_{clk}\ [sec]$
  - Throughput: $1/T_{clk}\ [samples/sec]$
  - Example: 8-pt FFT ($N = 8$)



H/W utilization rate:
- BF: 50%
- Register: 100%

Single-Path Delay Feedback

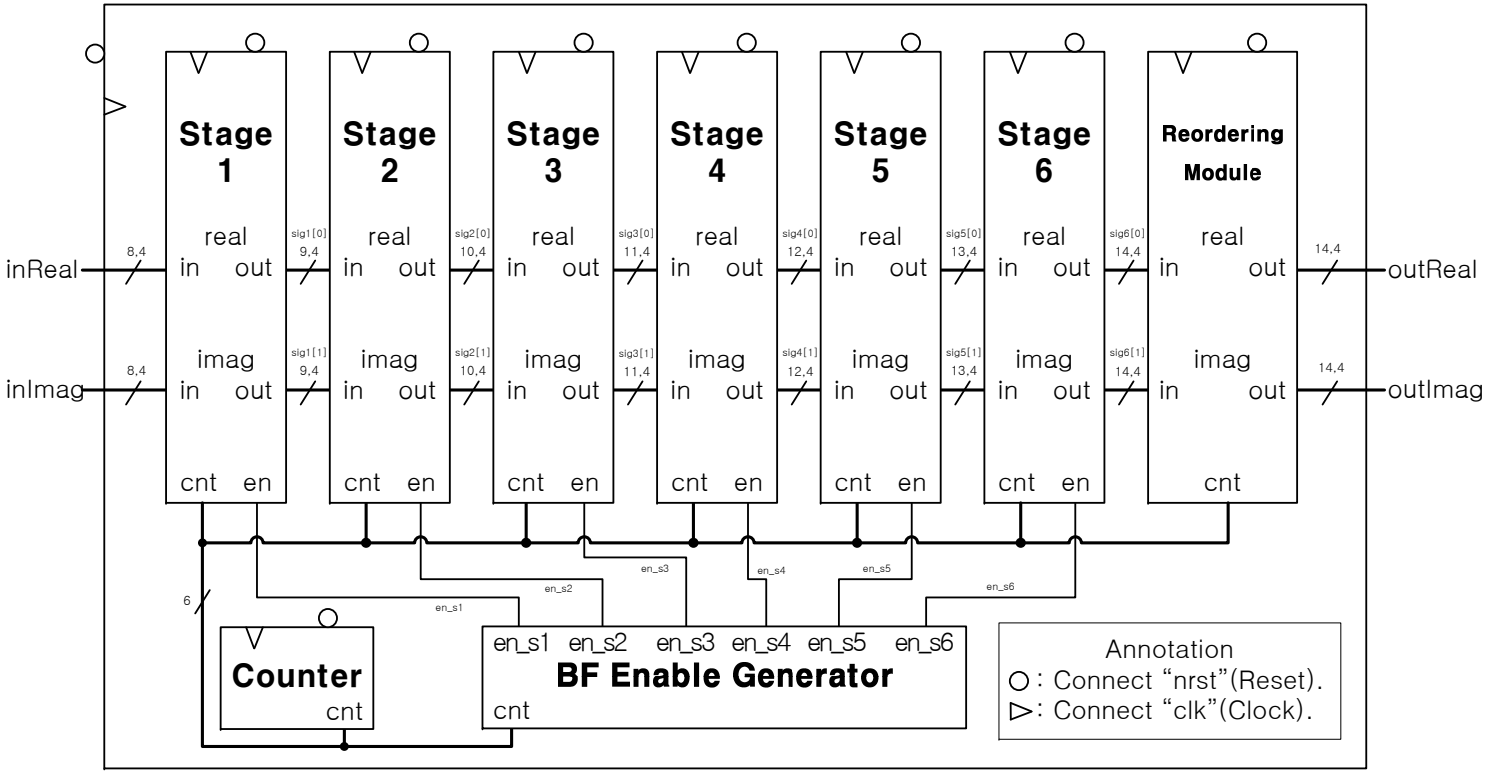
- Complexity metrics
  - Area: Comparison with parallel architecture

	Parallel		Pipeline	
	BF-PE	Register	BF-PE	Register
4	4	8	2	5
8	12	24	3	10
16	32	64	4	19
64	192	384	6	69
N	$(N/2)\ log_2 N$	$N\ log_2 N$	$log_2 N$	$(N-1)$ $+ log_2 N$

Complexity reduction by $\sim\frac{2}{N}$!

Single-Path Delay Feedback

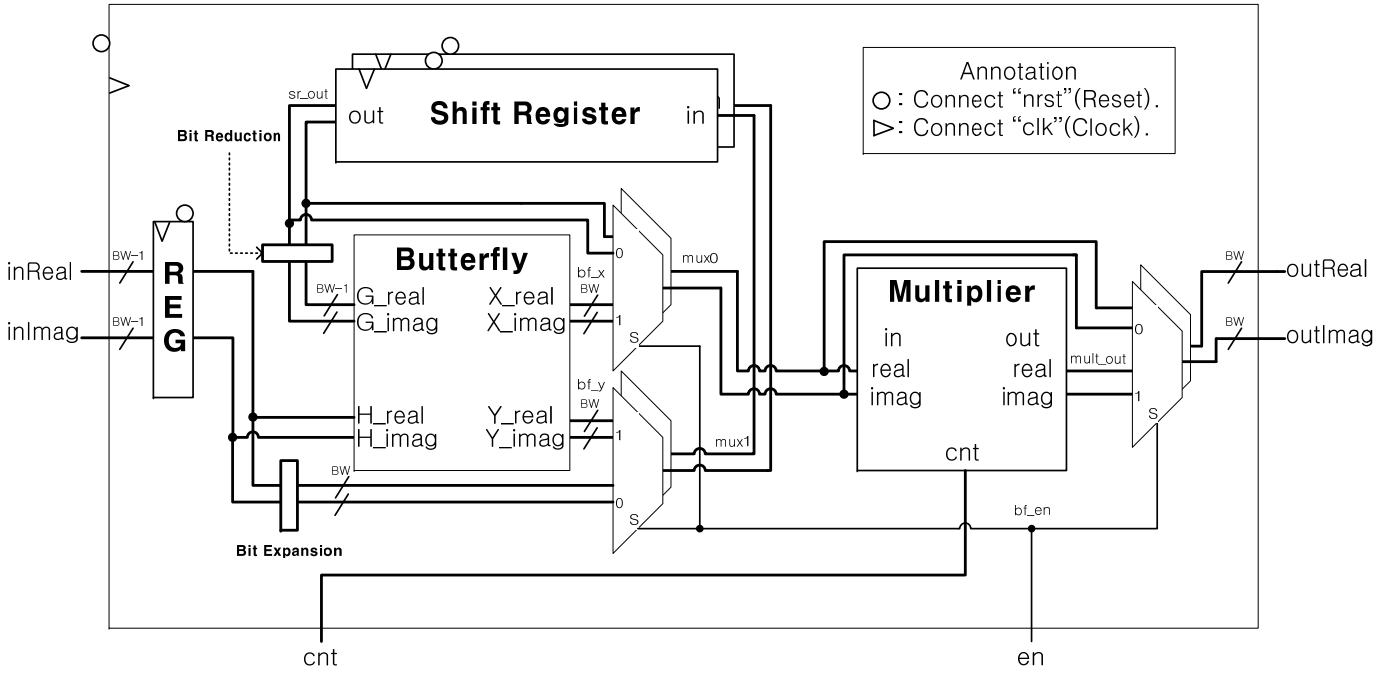
- Complexity metrics
  - Example: 64-pt Radix-2 SDF (CMOS 180nm)



Courtesy: S.W. Kim and J. H. Wang, Konkuk Univ.

Single-Path Delay Feedback

- Complexity metrics (cont'd)
  - Example: 64-pt Radix-2 SDF (CMOS 180nm)



Courtesy: S.W. Kim and J. H. Wang, Konkuk Univ.

Single-Path Delay Feedback

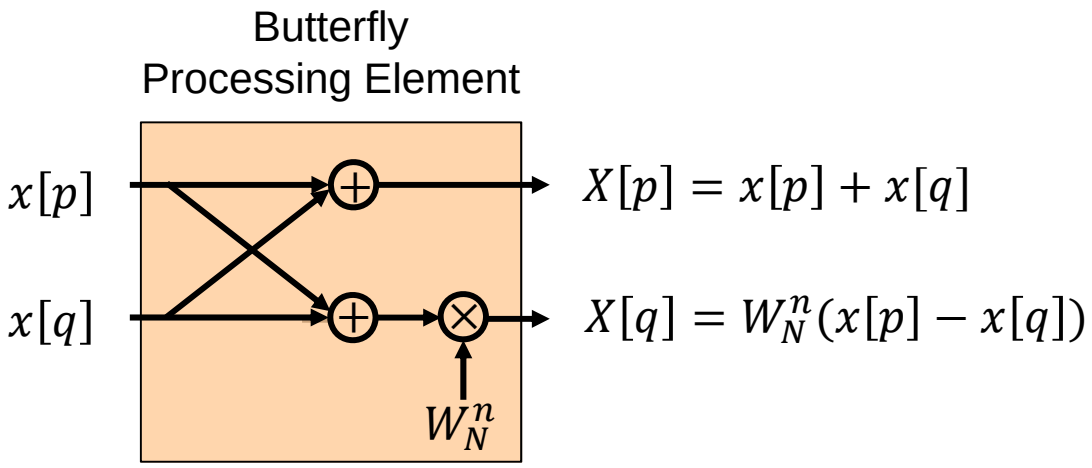
- Complexity metrics (cont'd)
  - Example: 64-pt Radix-2 SDF (CMOS 180nm)

Area (um^2)						
	Stage1	Stage2	Stage3	Stage4	Stage5	Stage6
Butterfly	3632	3925	4217	42031	4803	5096
Multiplier	33430	34907	35339	25995	4617	-
Shift Reg0	23923	12799	6819	3639	1922	-
Shift Reg1	23923	12799	6845	3639	1922	-
Stage Area	88314	68123	57196	42031	17769	11688
RvReg	392875					
Cnt	592					
Total Area	679000 (Stage1+..+Stage6+RvReg+Cnt)					

Courtesy: S.W. Kim and J. H. Wang, Konkuk Univ.

Fixed-Point Arithmetic

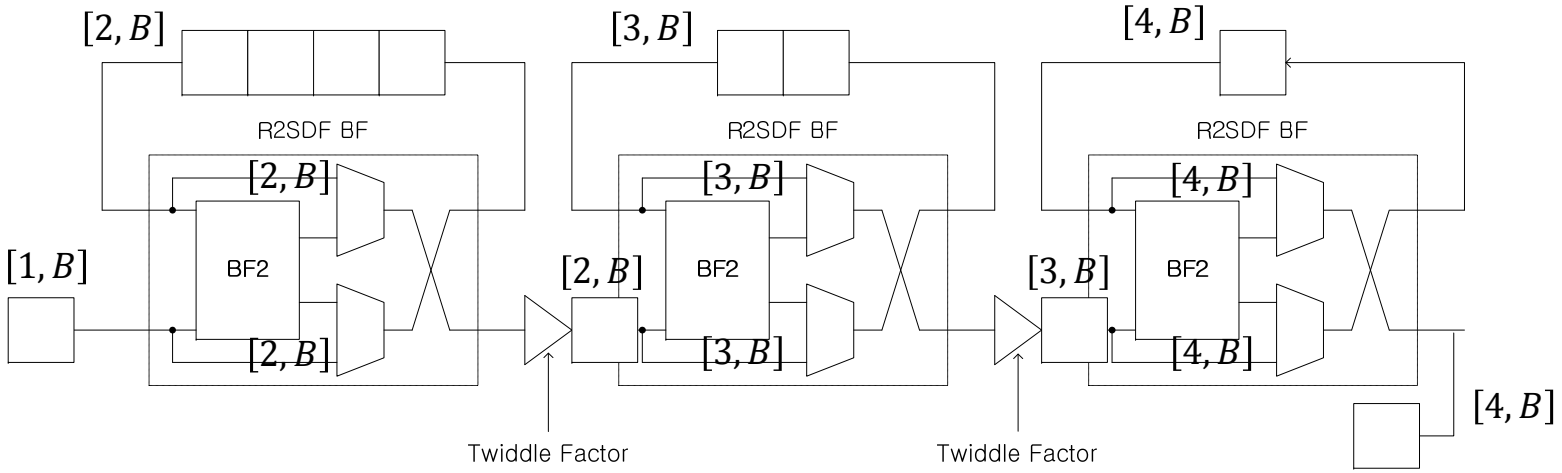
- Full-precision BF-PE
  - Equal word length assumed for real & imaginary parts
    - ✓ $x_r[p], x_i[p], x_r[q], x_i[q] \sim [1, N_x]$, $W_{Nr}^n, W_{Ni}^n \sim [1, N_W]$
  - Neither quantization nor overflow allowed
    - ✓ $X_r[p], X_i[p] \sim [2, \max(N_x, N_W)]$
    - ✓ $X_r[q], X_i[q] \sim [2, \max(N_x, N_W) + N_W]$



  - Word length expansion
    - ✓ One extra integer bit + many extra fractional bits

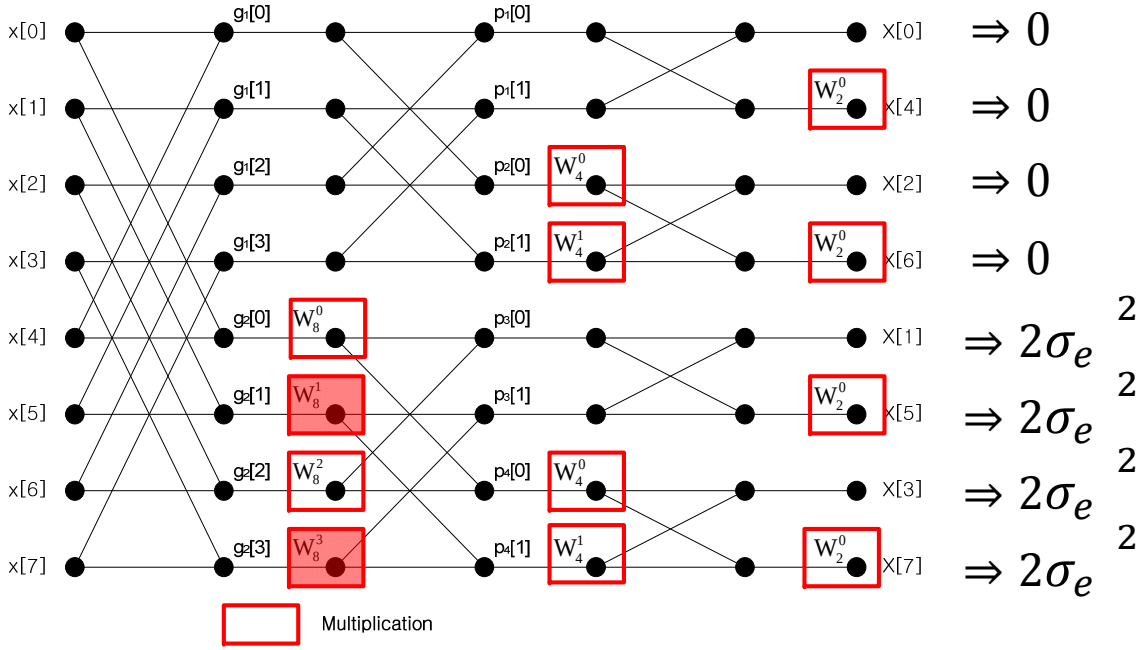
Fixed-Point Arithmetic

- Finite-WL BE-PE
  - 1 bit expansion per stage in MSB → No overflow
  - No expansion in LSB → Quantization
    - ✓ $X_r[p], X_i[p], X_r[q], X_i[q] \sim [2, \max(N_x, N_W)]$
  - Example: 8-pt FFT ($B = 8$)
    - ✓ $N_x = N_W = B$



Fixed-Point Arithmetic

- Quantization error (rounding)
  - Increase by $\sigma_e^2 = \frac{2^{-2B}}{12}$ per (nontrivial constant) multiplication
  - Example: 8-pt FFT ($N = 8$)



$\Rightarrow 0$

$\Rightarrow 0$

$\Rightarrow 0$

$\Rightarrow 0$

$\Rightarrow 2\sigma_e^2$

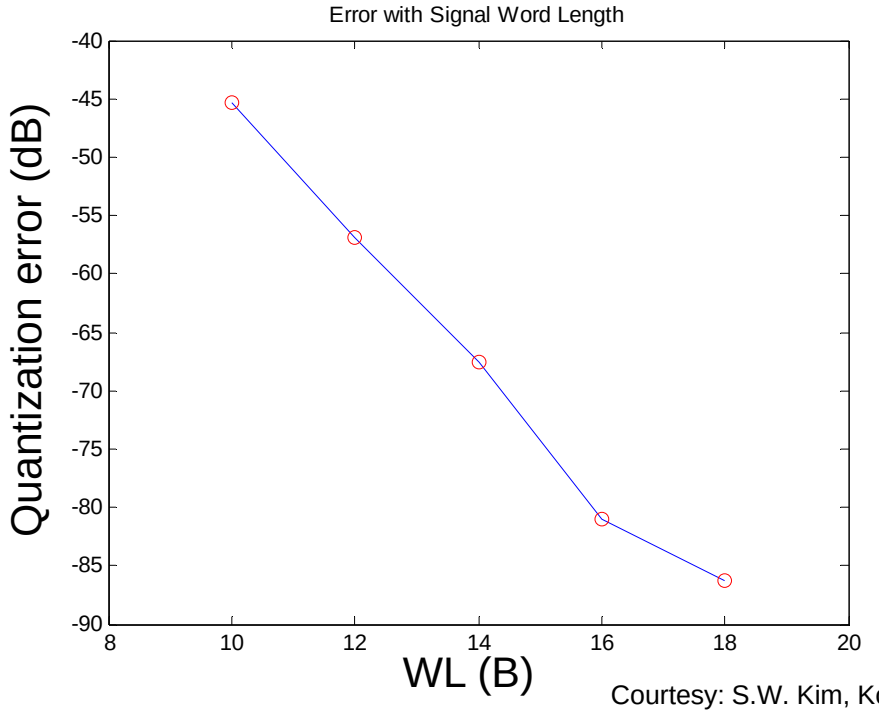
$\Rightarrow 2\sigma_e^2$

$\Rightarrow 2\sigma_e^2$

$\Rightarrow 2\sigma_e^2$

Performance Metrics

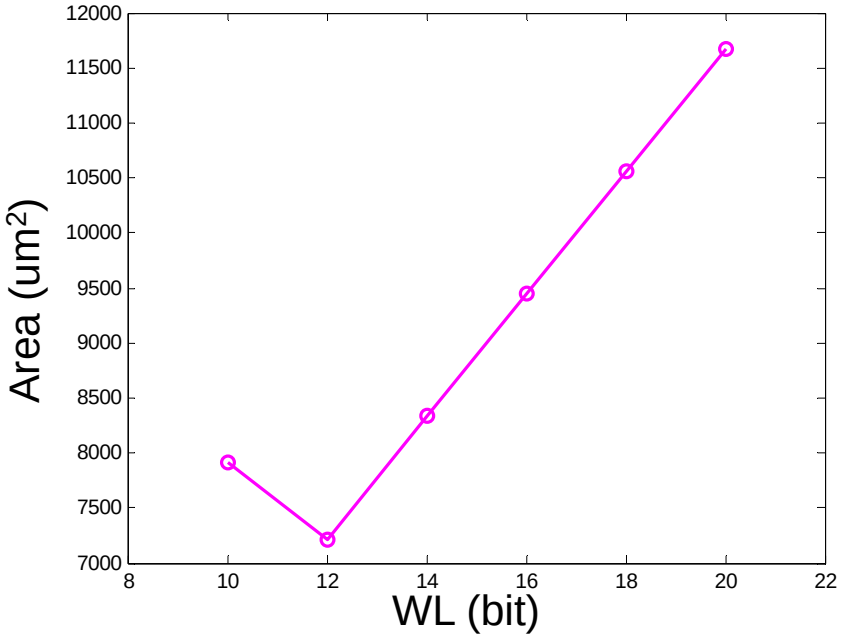
- Quantization error
  - Quantization-error-to-signal ratio: $\frac{\sigma_e^2}{\sigma_s^2} = 4N \times 2^{-2B}$
    - ✓ Proportional to FFT size (N)
    - ✓ Exponential with word length (B): -6 dB/bit

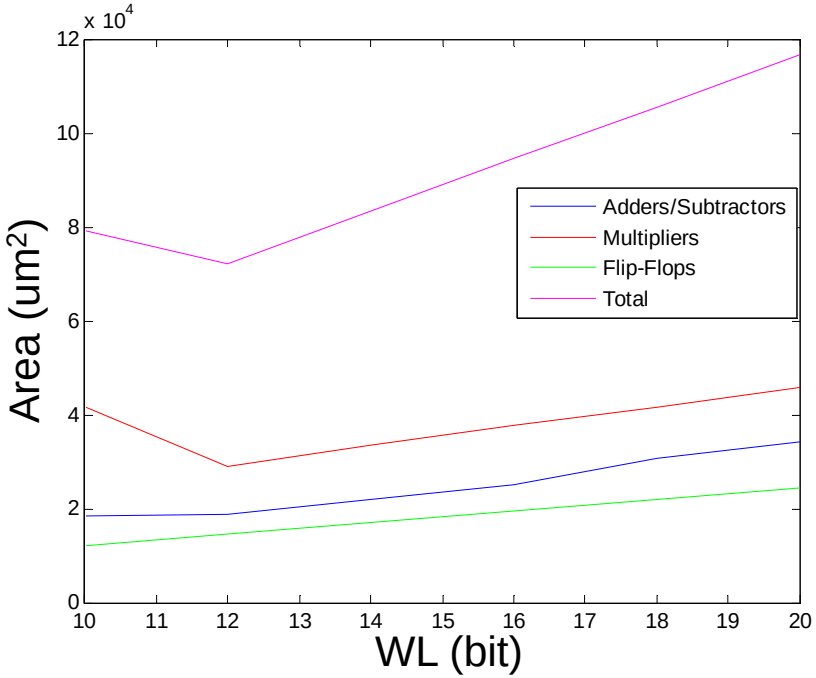


Courtesy: S.W. Kim, Konkuk Univ.

Complexity Metrics

- Area (CMOS 180nm)

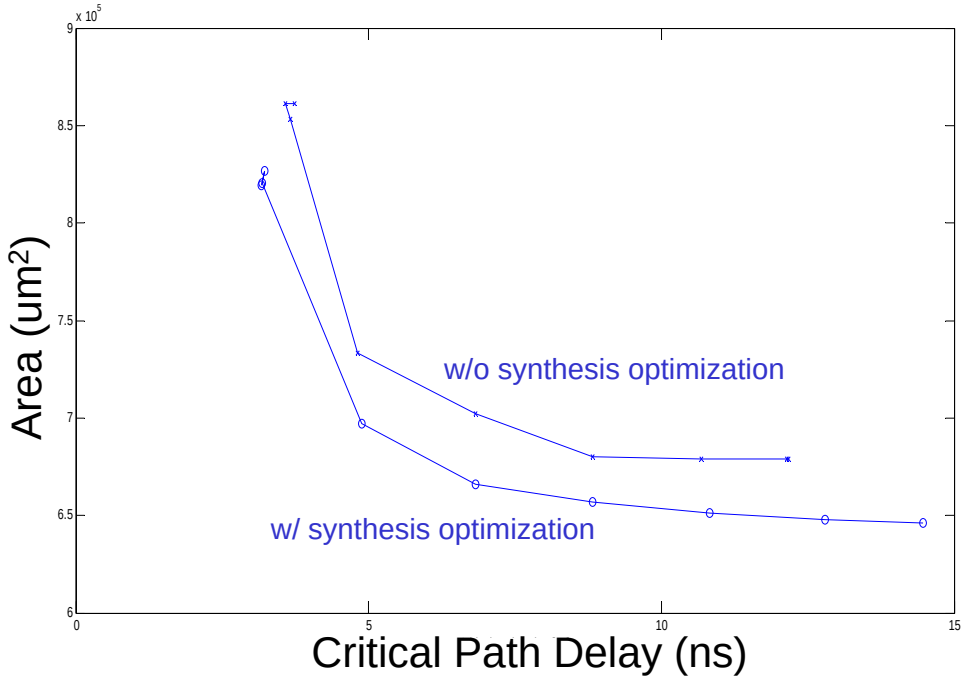




Courtesy: S.W. Kim and J. H. Wang, Konkuk Univ.

Performance-Complexity Trade-off

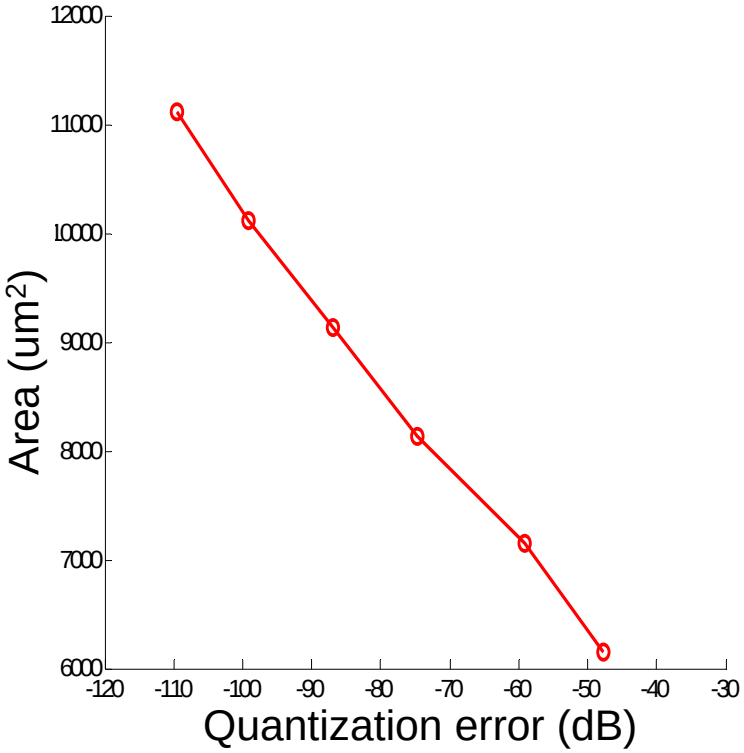
- Area vs. Delay
  - 64-pt Radix-2 SDF (CMOS 180nm)



Courtesy: S.W. Kim and J. H. Wang, Konkuk Univ.

Performance-Complexity Trade-off

- Quantization error vs. Area (CMOS 180nm)
  - With respect to word length



Courtesy: S.W. Kim and J. H. Wang, Konkuk Univ.

Lab 2: HW Design

- Creating RTL projects
- Programming in RTL
- Running Behavioral Simulation
- Running Synthesis
- Running Implementation

Source Codes

- tb_Top_FFT.v

```
1   `timescale 1ns/1ps
2   `define p 2
3   module tb_Top_FFT;
4
5   reg nrst,clk;
6
7
8   reg [15:0] input_re[191:0];
9   reg [15:0] input_im[191:0];
10
11  reg [15:0] output_re[191:0];
12  reg [15:0] output_im[191:0];
13
14  wire [15:0] inReal,inImag;
15  wire [15:0] outReal,outImag;
16
17  integer clkcnt;
18
19  reg start;
20
21
22  Top_FFT top0(.nrst(nrst),.clk(clk),.start(start),.valid(1'b1),
23               .inReal(inReal),.inImag(inImag),
24               .outReal(outReal),.outImag(outImag));
25
26  always
27    #(`p/2)  clk = !clk;
28
29  always@(negedge clk)
30    clkcnt = clkcnt +1;
31
```

```
32  initial begin
33    clk  = 0;
34    nrst = 0;
35    clkcnt=-103;
36    start = 0;
37    $readmemb("binary_in_real.txt", input_re);
38    $readmemb("binary_in_imag.txt", input_im);
39
40    #(100*`p) start = 1'b1;
41    #(`p/2+1) nrst = 1;
42  end
43
44  assign inReal = clkcnt > -2? input_re[clkcnt+1] : 0;
45  assign inImag = clkcnt > -2? input_im[clkcnt+1] : 0;
46
47  always @ (posedge clk) begin
48  output_re[clkcnt+1] <= outReal;
49  output_im[clkcnt+1] <= outImag;
50  end
51
52
53  integer dumpfile, i;
54  initial begin
55
56
57    #(292*`p +1) dumpfile = $fopen("binary_out_real.txt","w");
58    for(i = 0; i<192;i=i+1)begin
59    $fwrite(dumpfile,"%b\n",output_re[i]);
60    end
61    $fclose(dumpfile);
62
63      dumpfile = $fopen("binary_out_imag.txt","w");
64      for(i = 0; i<192;i=i+1)begin
65      $fwrite(dumpfile,"%b\n",output_im[i]);
66      end
67      $fclose(dumpfile);
68
69    $stop;
70  end
71
72
73  endmodule
```

Source Codes

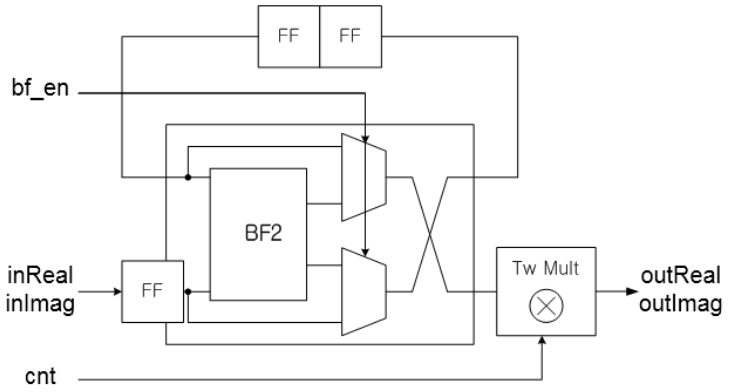
- Top_FFT.v

```
module Top_FFT #(
  parameter in_BW = 16,
  parameter out_BW= 22,
  parameter cut_BW= 6
) (
  input nrst,clk,start,
  input valid,
  input [in_BW-1:0]  inReal,inImag,
  output[out_BW-cut_BW-1:0] outReal,outImag
);

wire [5:0] cnt;

wire [in_BW  :0] sig1[1:0];
wire [in_BW+1:0] sig2[1:0];
wire [in_BW+2:0] sig3[1:0];
wire [in_BW+3:0] sig4[1:0];
wire [in_BW+4:0] sig5[1:0];
wire [in_BW+5:0] sig6[1:0];

wire  en_s1,en_s5,en_s6;
reg en_s2;
reg [2:0] en_s4;
reg [1:0] en_s3;

Counter cnt0(nrst,clk,start, valid,cnt);
Stage #(in_BW+1,32) stage1(nrst,clk,en_s1,cnt,inReal,inImag, valid, sig1[0],sig1[1]);
Stage #(in_BW+2,16) stage2(nrst,clk,en_s2,cnt,sig1[0],sig1[1], valid, sig2[0],sig2[1]);
Stage #(in_BW+3,8 ) stage3(nrst,clk,en_s3[1],cnt,sig2[0],sig2[1], valid, sig3[0],sig3[1]);
Stage #(in_BW+4,4 ) stage4(nrst,clk,en_s4[2],cnt,sig3[0],sig3[1], valid, sig4[0],sig4[1]);
Stage #(in_BW+5,2 ) stage5(nrst,clk,en_s5,cnt,sig4[0],sig4[1], valid, sig5[0],sig5[1]);
Stage6 #(in_BW+6,1 ) stage6(nrst,clk,en_s6   ,sig5[0],sig5[1], valid, sig6[0],sig6[1]);

assign outReal = sig6[0][in_BW+5 : cut_BW]; //added
assign outImag = sig6[1][in_BW+5 : cut_BW]; //added
```

Source Codes

- Stage.v

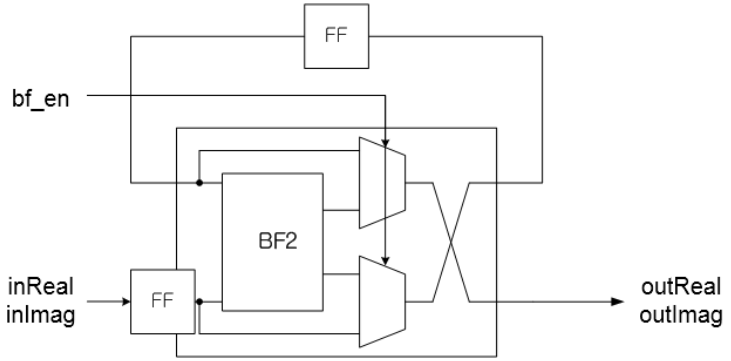
```
module Stage(nrst,clk,bf_en,cnt,inReal,inImag,valid,outReal,outImag);
parameter BW=16;
parameter N =32;

input           nrst,clk,bf_en;
input [BW-2:0]  inReal,inImag;
input [5:0]     cnt;
input           valid;
output[BW-1:0]  outReal,outImag;

reg  [BW-2:0] rReal,rImag;

wire [BW-1:0] bf_x[1:0];
wire [BW-1:0] bf_y[1:0];

wire [BW-1:0] mult_out[1:0];

wire [BW-1:0] sr_out[1:0];

wire [BW-1:0] mux0[1:0];
wire [BW-1:0] mux1[1:0];

assign mux0[0] = bf_en? bf_x[0] : sr_out[0];
assign mux0[1] = bf_en? bf_x[1] : sr_out[1];

assign mux1[0] = bf_en? bf_y[0] : {rReal[BW-2],rReal};
assign mux1[1] = bf_en? bf_y[1] : {rImag[BW-2],rImag};

Shift_Reg #(BW,N) sr0(nrst,clk,mux1[0],valid,sr_out[0]);
Shift_Reg #(BW,N) sr1(nrst,clk,mux1[1],valid,sr_out[1]);

BF #(BW)bf0({sr_out[0][BW-1],sr_out[0][BW-3:0]},{sr_out[1][BW-1],sr_out[1][BW-3:0]},rReal,rImag,bf_x[0],bf_x[1],bf_y[0],bf_y[1]);

MULT #(.BW(BW),.N(N))mult0(mux0[0],mux0[1],cnt[4:0],mult_out[0],mult_out[1]);

assign outReal = bf_en? mux0[0] : mult_out[0];
assign outImag = bf_en? mux0[1] : mult_out[1];


always@(posedge clk) begin
    if(!nrst) begin
      rReal <= 0;
      rImag <= 0;
    end
    else if(valid) begin

      rReal <= inReal;
      rImag <= inImag;

    end
end

endmodule
```



Source Codes

- Stage6.v

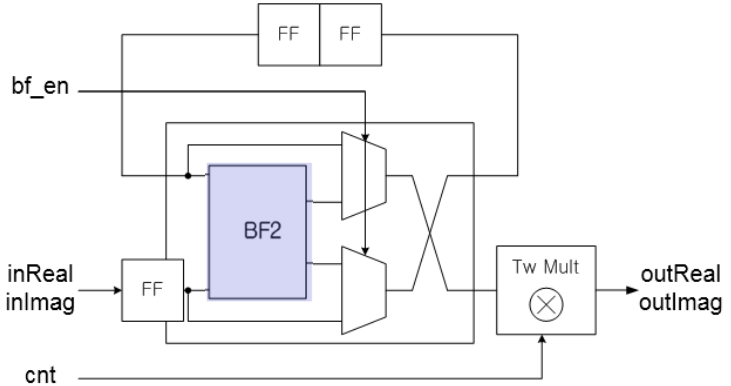
```
module Stage6(nrst,clk,bf_en,inReal,inImag,valid,outReal,outImag);
parameter BW=16;
parameter N =1;

input           nrst,clk,bf_en;
input [BW-2:0]  inReal,inImag;
input           valid;
output[BW-1:0]  outReal,outImag;

reg   [BW-2:0]  rReal,rImag;

wire  [BW-1:0]  bf_x[1:0];
wire  [BW-1:0]  bf_y[1:0];

reg   [BW-1:0]  sr_out[1:0];

wire  [BW-1:0]  mux0[1:0];
wire  [BW-1:0]  mux1[1:0];

assign mux0[0] = bf_en? bf_x[0] : sr_out[0];
assign mux0[1] = bf_en? bf_x[1] : sr_out[1];

assign mux1[0] = bf_en? bf_y[0] : {rReal[BW-2],rReal};
assign mux1[1] = bf_en? bf_y[1] : {rImag[BW-2],rImag};


////////////////////////////////////////////////////
//////////     Fill your code here    //////////////
////////////////////////////////////////////////////

endmodule
```



Source Codes

- BF.v

```
1   module BF (Gr,Gi,Hr,Hi,Xr,Xi,Yr,Yi);
2
3   parameter BW =16;
4
5   input   signed [BW-2:0] Gr,Gi,Hr,Hi;
6   output  signed [BW-1:0] Xr,Xi,Yr,Yi;
7
8   assign Xr = Gr+Hr;
9   assign Xi = Gi+Hi;
10  assign Yr = Gr-Hr;
11  assign Yi = Gi-Hi;
12
13  endmodule
```



Source Codes

- Shift_Reg.v

```
1   module Shift_Reg
2   #(
3       parameter BW=16,
4       parameter N =32
5   ) (
6       input nrst,clk,
7       input [BW-1:0] inData,
8       input valid,
9       output[BW-1:0] outData
10  );
11
12  reg     [BW-1:0] sr[N-1:0];
13  integer i;
14
15  always@(posedge clk)
16      if(!nrst)
17          for(i=1;i<N;i=i+1)
18              sr[i] <= 0;
19      else if (valid) begin
20          for(i=1;i<N;i=i+1)
21              sr[i] <= sr[i-1];
22      end
23
24
25  always@(posedge clk)
26      if(!nrst)
27        sr[0] <= 0;
28      else if (valid) begin
29        sr[0] <= inData;
30      end
31
32  assign outData = sr[N-1];
33
34  endmodule
```