Radix-2 Single Path Delay Feedback (SDF) 128pt-FFT

1. 서론

디지털 신호 처리(DSP)에서 신호 변환은 매우 중요한 과정이다. **ADC(Analog to Digital Conversion)**를 통해, 아날로그 Data 입력들을 디지털 Data로 변환하여 컴퓨터에서 분석 처리가 가능하게 변환 시킨다. 또한 이 데이터를 사용자가 원하는 Data만 뽑아내기 위해서는 시간 영역에서 주파수 영역으로 변환할 필요가 있다. 이를 위해 **DFT(Discrete Fourier Transform)**를 사용하며, 신호의 주파수 성분을 분석할 수 있다. 그러나 DFT 과정은 연산량이 많아 속도가 느리고 필요 HW도 많아지는 단점이 있다. 이를 해결하기 위해 **FFT(Fast Fourier Transform**)를 사용하여 연산량과 HW를 줄이고 있다. 이 FFT처리된 Data는 이후 신호 처리, 영상 및 음성 분석, 통신 시스템, 레이더, 의료 영상 등의 다양한 분야에서 사용된다. 이번 실험을 통해 DFT에 비해 FFT가 얼마나 연산량이 줄어들고 어떻게 FFT의 HW를 설계하는 지 알 수 있게될 것이다

2. DFT

2-1. DFT연산

$$X_k \cong X\left(K\frac{2\pi}{N}\right) = \sum_{K=0}^{N-1} x(n) \bullet e^{\left(-jk\frac{2\pi}{N}n\right)}$$

DFT의 수식으로써, 위 수식 중 $e^{\left(-jk\frac{2\pi}{N}n\right)}$는 Twiddle Factor로써 W_N^{nk}로 나타낸다.
예시로써 8pt DFT연산을 해보면 아래와 같은 식이 나온다.

$$X[K] = \sum_{n=0}^{7} x(n) \bullet W_8^{nk}$$

$$X[0] = x(0) \bullet W_8^0 + x(1) \bullet W_8^0 + \cdots + x(7) \bullet W_8^0$$

$$X[1] = x(0) \bullet W_8^0 + x(1) \bullet W_8^1 + \cdots + x(7) \bullet W_8^7$$

$$X[2] = x(0) \bullet W_8^0 + x(1) \bullet W_8^2 + \cdots + x(7) \bullet W_8^{14}$$

$$\vdots$$

$$X[7] = x(0) \bullet W_8^0 + x(1) \bullet W_8^7 + \cdots + x(7) \bullet W_8^{49}$$

라는 식이 나오고 8pt -DFT의 연산 양을 알 수 있다.

2-2. DFT의 연산 량

이산 푸리에 변환(DFT)는 많은 연산을 요구하는데, 위의 연산에서 확인할 수 있듯이,

Npt-DFT	Ex) 8pt-DFT
Multiplier : N^2개	Multiplier : 64개
Adder : $N(N-1)$개	Adder : 56개

8pt DFT만해도 총 **120**개의 연산을 요구한다. 이를 해결하기 위해 고속 푸리에 변환(FFT) 알고리즘을 사용하고 있다. 현실에서 사용하는 DFT는 더 큰pt의 연산을 하는데, 그렇게 되면 너무 많은 연산을 요구하게 되고 그에 따른 더 많은 HW를 요구하게 되는 것이다. 이를 해결하기 위해 우리는 고속 푸리에 트랜스폼**(FFT)** 알고리즘을 사용하게 된다.

※ Twiddle Factor(회전 인자)

일정한 주기(N)를 가지고 반복되는 인수로써 Npt의 N의 주기성을 띈다.

Ex) 8pt DFT의 Twiddle Factor(W_8^{nk})

$$W_8^0 = e^{\left(-jk\frac{2\pi}{N}0\right)} = W_8^8 = W_4^0$$

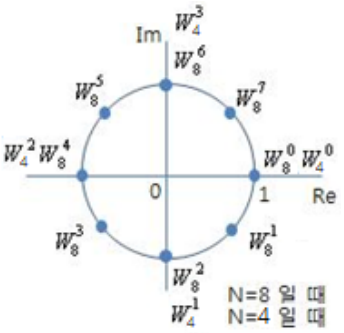
$$W_8^1 = e^{\left(-jk\frac{2\pi}{N}1\right)} = W_8^9$$

$$W_8^2 = e^{\left(-jk\frac{2\pi}{N}2\right)} = W_8^{10} = W_4^1$$

$$\vdots$$

$$W_8^6 = e^{\left(-jk\frac{2\pi}{N}2\right)} = W_8^{14} = W_4^3$$

$$W_8^7 = e^{\left(-jk\frac{2\pi}{N}7\right)} = W_8^{15}$$

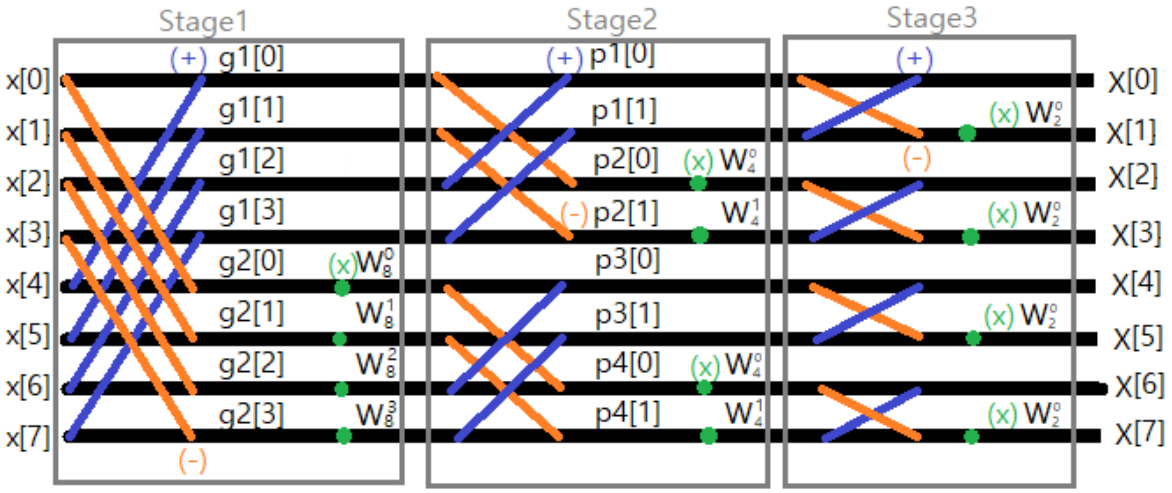


추후에 미리 Twiddle Factor를 계산해 Memory에 저장해 연산에 사용한다.

3. FFT

3-1. FFT연산

고속 푸리에 변환(FFT)은 Butterfly 연산을 사용하는데, 아래 8pt butterfly 예시가 있다.



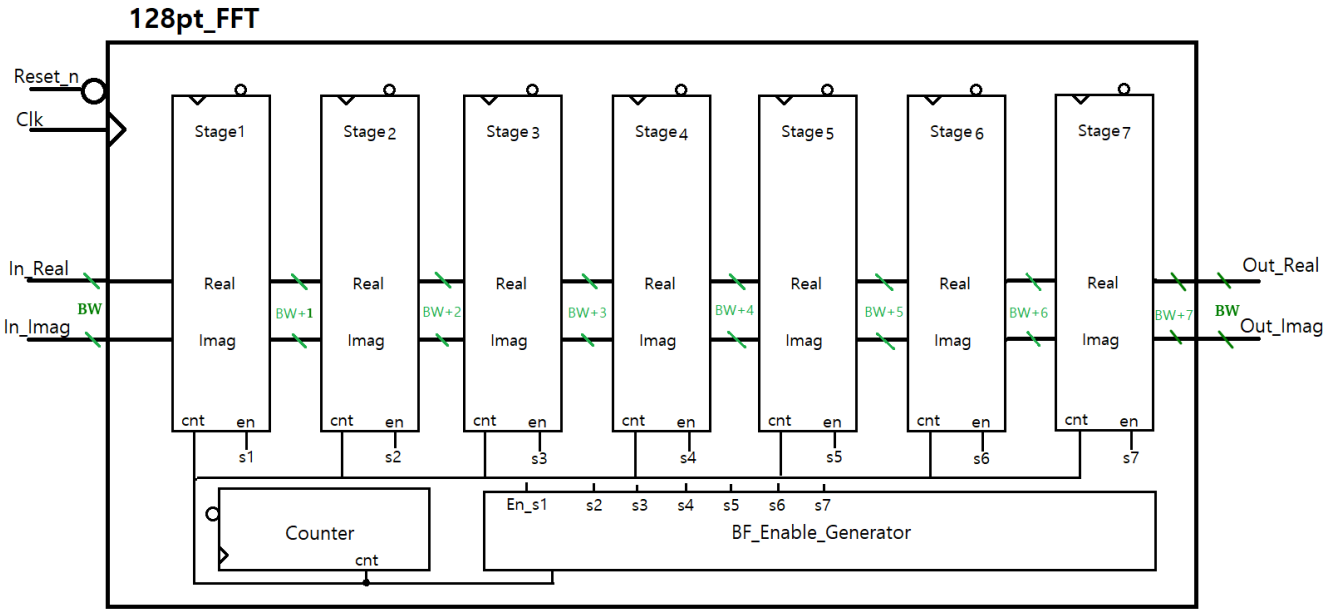
3-2. FFT의 연산 량

8pt-FFT	Npt-FFT
Multiplier : 12개	Multiplier : $\frac{N}{2}log_2N$개
Adder : 24개	Adder : $Nlog_2N$개

8pt DFT가 **120**개의 연산을 요구하는 것에 비해, **8pt FFT**는 **36**개의 연산을 요구한다. 이와 같이 확연한 연산 량의 차이가 나는 것을 알 수 있다. 그에 따라 필요한 HW의 수 또한 줄어들게 된다.

4. HW설계 (Block Diagram)

4-1. 128pt FFT의 Block Diagram



아래 Block Diagram은 상위 Block으로서 하위 Block인 Stage들은 다음 목차에서 자세히 설명된다.

- Stages (Top_FFT.v)
  1) 목적
     - Pipeline(SDF)을 고려하여 설계되고, 제어 신호(from BF_Enable_Generator)에 따라 Stage가 Butterfly 연산 또는 Twiddle Factor Multiplication의 값을 출력시키는 Block
  2) 설계
     - 128pt FFT를 만들기위해 필요한 Stage는 $log_2 128 = 7(stages)$개가 필요하다.
     - Bit Size : Stage를 지나칠 때 마다 (현재 $Bit\ size$) + 1이 된다.

- Counter (Counter.v)
  1) 목적
     - 현재의 동작 Cycle을 카운팅 하기위한 Block
       1. BF_Enable_Generator(128 카운팅)
       2. Stage의 Multiplier(64 카운팅)

2) 설계

- 1clk마다 +1씩 되는 카운터
- 최대 128까지 셀 수 있는 카운터

- BF_Enable_Generator (BF_En_Gen.v)

1) 목적

- Stage 내부의 Mux들을 제어하는 신호를 만들어내는 Block

2) 설계

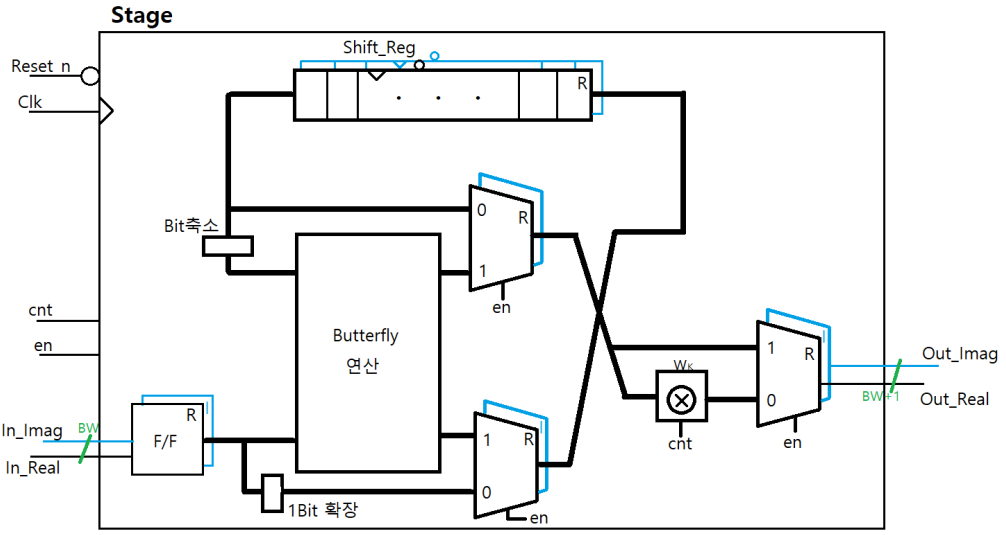
- Stage 내부 Mux들을 Timing에 맞게 제어하기 위해, 카운터의 cnt signal로부터 bit를 뽑아와 제어신호로 사용한다. Timing Diagram을 보면 Mux에
  ① En_s1(7bit) : 64Clk마다 BF <-> TM으로 변경되므로 cnt[6] 사용
  ② En_s2(6bit) : 32Clk마다 BF <-> TM으로 변경되므로 cnt[5] 사용

  ⋮

  ⑦ En_s7(1bit) : 1Clk마다 BF <-> TM으로 변경되므로 cnt[0] 사용

- Timing Diagram을 보면 Enable 신호로 Mux를 제어함에 따라 Stage의 출력 값이 달라지는 것을 확인할 수 있다. 또한, Stage를 통과할 때마다 제어 신호와 데이터 간의 동기화가 1클럭씩 지연되는 현상이 발생한다. 따라서 Stage의 수가 증가할수록 Enable 신호의 타이밍을 1클럭씩 추가로 조정해야 올바르게 동작한다. (참고 Timing Diagram)

4-2. Stage의 Block Diagram



- Shift Register (Shift_Reg.v)

1) 목적

- 입력을 $\frac{N}{2}(N$은Npt를뜻함)clk 뒤에 출력을 내보내는 Block

※ Butterfly연산을 할 때, 1clk마다 입력이 들어가게 되는데, Butterfly연산은 위에서 확인되듯이 $x[k] \pm x\left[k + \frac{N}{2}\right](k = 0 \sim N - 1)$ 해당 연산을 하게 된다. 즉, $x\left[k + \frac{N}{2}\right]$ 이 입력인 시간에 Butterfly연산을 하게 되는 것이다.
128pt-FFT의 식 $x[0] \pm x\left[0 + \frac{128}{2}\right]$을 보면, x[0]가 Butterfly연산을 하게 되는 시간은 x[64]이 입력으로 들어오는 시간이다. 즉, x[64]가 들어오는 시간(64clk)까지 x[0]의 데이터를 저장해둘 Shift Register가 필요하다.

2) 설계

- Butterfly 연산에서는 1클럭마다 입력 데이터가 들어오며, 연산 과정에서 알 수 있듯이 $x[k] \pm x\left[k + \frac{N}{2}\right](k = 0N - 1)$의 계산이 이뤄진다. 즉, $x\left[k + \frac{N}{2}\right]$가 입력되는 시간에 Butterfly연산이 된다.
128pt FFT에서의 $x[0] \pm x\left[0 + \frac{128}{2}\right]$ Butterfly연산을 설계하고자 하면, $x[64]$가 입력일 때가 x[0]의 Butterfly연산이 가능해지는 시점이다. 따라서 $x[64]$가 입력으로 들어오는 시점(64클럭)까지 $x[0]$의 값을 저장할 수 있는 레지스터가 필요하다.
- Stage를 지날수록 Shift Register의 Register수가 $\frac{1}{2}$로 줄어든다.

Ex) Stage1(64개), Stage2(32개) ~ Stage1(1개)

- Butterfly calculator (BF_Calc.v)

1) 목적

- Butterfly 연산을 구현한 HW

2) 설계

- $x[k] \pm x\left[k + \frac{N}{2}\right]$연산 설계
- 두 입력의 연산(±)의 출력(연산 결과)의 Bit Width는 Bit이 된다.

- Twiddle Factor(W_k) Multiplier (Mult.v)

1) 목적

- 입력 signals에 대해 W_k를 곱해주는 HW

2) 설계

- W_k를 미리 계산하여 Memory에 저장해두고, 입력이 들어오면 해당 입력에 맞는 W_k와 곱해주는 형태이다. 이때의 W_k는 cnt signal로부터 가져와 짝을 지어주는 형태이다.
- W_k는 **10**번째 고정소수점 형태로 나타낸다.

Ex) W_{128}^{1}(10번째 고정소수점)

$$W_{128}^{1} = exp\left(-j1\frac{2\pi}{128}n\right)$$

$$= cos\left(\frac{2\pi}{128}\right) - jsin\left(\frac{2\pi}{128}\right)$$

$$= W_{128}^{1}[Re] + W_{128}^{1}[Im]$$

$W_{128}^{1}[Re] \cong 0.998$, 이를 고정소수점 10을 이용해 2진수로 나타내면

$$0.998 * 1024 \cong 1022_{(10)}$$

$$= 0011\ 1111\ 1110_{(2)}$$

위와 같은 형태로 나머지 W_k도 계산하여 저장해두어 이로 인해 빠른 계산을 가능케 한다.

- 출력 = 입력$[Real + Imag] \times W_k[Real + Imag]$

  $= (Real \times Real) + (Real \times Imag) + (Imag \times Real) + (Imag \times Imag)$

  실수(Out_Real) : $(Real \times Real)$ - $(Imag \times Imag)$

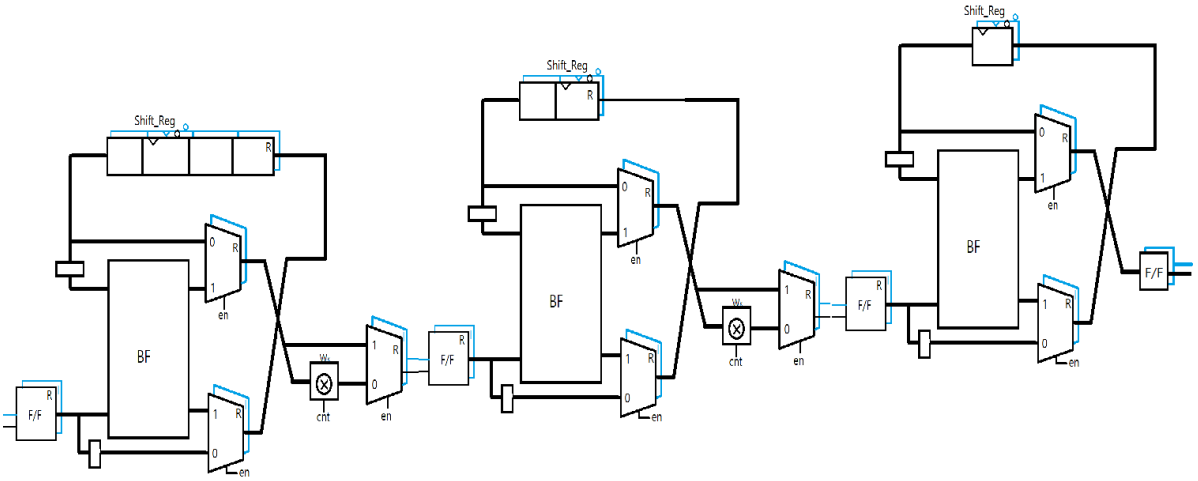
  허수(Out_Imag) : $(Real \times Imag) + (Imag \times Real)$

  ※ 두 signal의 곱의 Bit size는 두 신호의 Bit size의 합

- 출력의 Bit Size = Bit size[입력 + W_k]인데 데이터의 사이즈가 너무 크므로, 필요 없는 Bit를 Truncation한다

5. Stage의 Timing Diagram

Ex) 8pt-FFT



- 목표 : 매 클럭마다 들어오는 입력($x[0] \sim x[N]$)을 파이프라인에 맞게 차례대로 출력을 내보내는것
- 설계
  - Stage내부 Mux의 제어신호(En) :

    각 Stage의 Mux는 제어 신호(En)를 통해 Mode(TM, BF)를 결정하며, 아래와 같이 동작한다.

    1) TM(Twiddle Factor Multiplication) Mode :

    - 시간 : BF_Calc의 입력 Data가 x[0] ~ x[3](=x[$\frac{N}{2} - 1$])까지의 시간 동안 활성화

    2) BF(Butterfly Calculation) Mode :

    - 시간 : BF_Calc의 입력 Data가 x[4](= $x[\frac{N}{2}]$) ~ x[7](= $x[N - 1]$)까지의 시간동안 활성화

  - 제어 신호(En)와 Cnt 관계 :

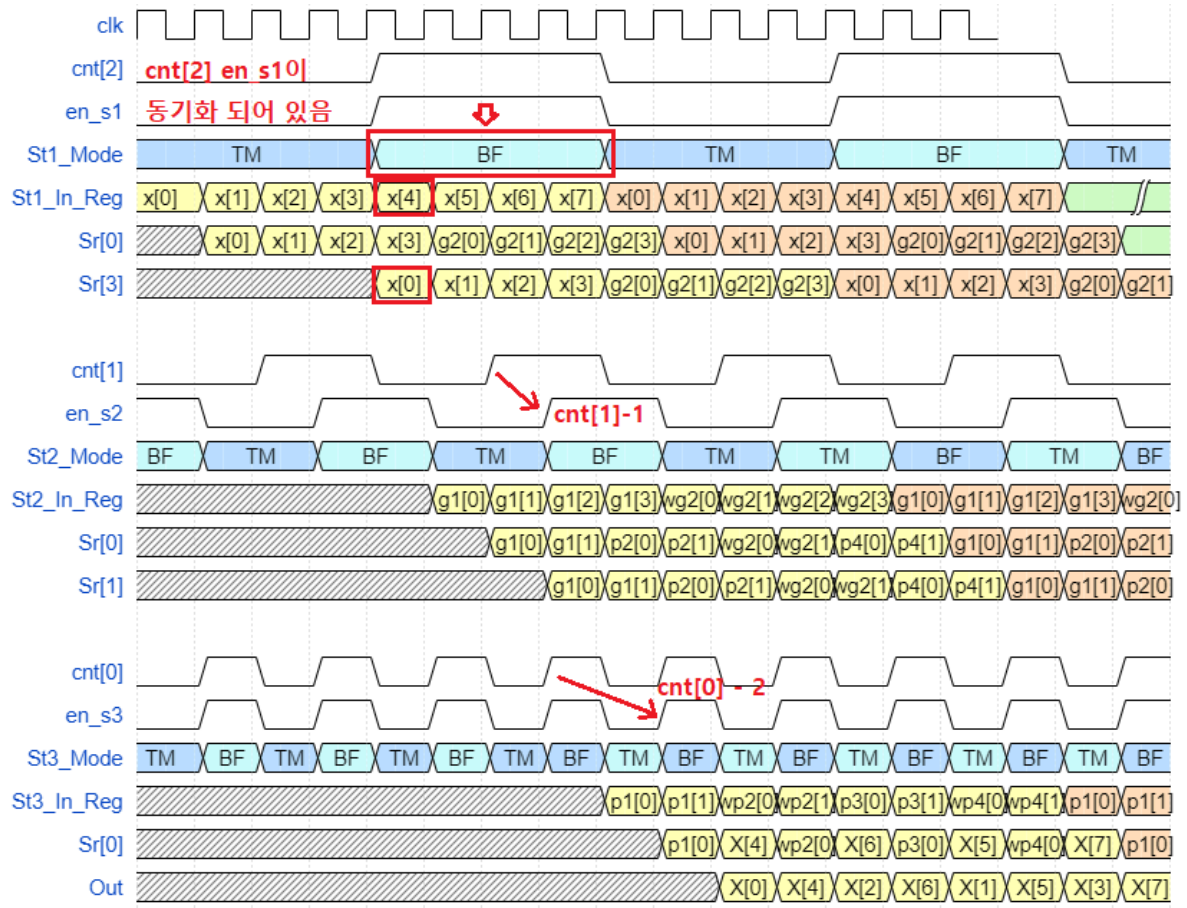
    각 Stage의 제어 신호(En)신호는 카운터(cnt)로 부터 생성된다.

    1) Stage1의 En_s1신호와 cnt[2] 신호는 정상적으로 동기화 되어 있음

    2) Stage2의 En_s2신호는 cnt[1] 신호 보다 1클락 늦게 활성화 됨

    3) Stage3의 En_s3신호는 정상 동기화 처럼 보이지만 cnt[0] 보다 3클락 늦게 활성화 된다고도 볼 수 있음.

    ∴ 즉, Stage를 통과할 수록 En신호가 1clk씩 지연 되므로, 각 **Stage**의 **cnt[n]**신호를 **1**클락씩 지연 시켜 연결해야된다. (BF_En_Gen.v 참고)

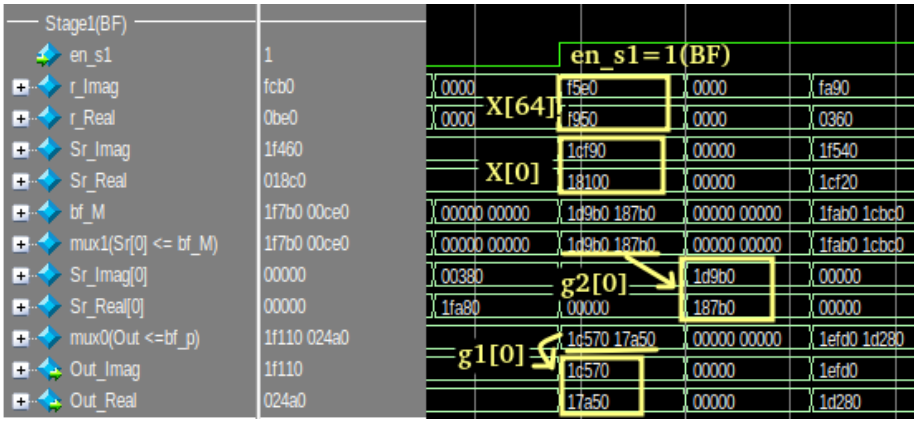

clk
cnt[2]
cnt[2] en_s1이
en_s1
동기화 되어 있음
St1_Mode
TM
BF
TM
BF
TM
St1_In_Reg
x[0] x[1] x[2] x[3] x[4] x[5] x[6] x[7] x[0] x[1] x[2] x[3] x[4] x[5] x[6] x[7]
Sr[0]
x[0] x[1] x[2] x[3] g2[0] g2[1] g2[2] g2[3] x[0] x[1] x[2] x[3] g2[0] g2[1] g2[2] g2[3]
Sr[3]
x[0] x[1] x[2] x[3] g2[0] g2[1] g2[2] g2[3] x[0] x[1] x[2] x[3] g2[0] g2[1]
cnt[1]
en_s2
cnt[1]-1
St2_Mode
BF TM BF TM BF TM TM BF TM BF
St2_In_Reg
g1[0] g1[1] g1[2] g1[3] wg2[0] wg2[1] wg2[2] wg2[3] g1[0] g1[1] g1[2] g1[3] wg2[0]
Sr[0]
g1[0] g1[1] p2[0] p2[1] wg2[0] wg2[1] p4[0] p4[1] g1[0] g1[1] p2[0] p2[1]
Sr[1]
g1[0] g1[1] p2[0] p2[1] wg2[0] wg2[1] p4[0] p4[1] g1[0] g1[1] p2[0]
cnt[0]
cnt[0] - 2
en_s3
St3_Mode
TM BF TM BF TM BF TM BF TM BF TM BF TM BF TM BF TM BF
St3_In_Reg
p1[0] p1[1] wp2[0] wp2[1] p3[0] p3[1] wp4[0] wp4[1] p1[0] p1[1]
Sr[0]
p1[0] X[4] wp2[0] X[6] p3[0] X[5] wp4[0] X[7] p1[0]
Out
X[0] X[4] X[2] X[6] X[1] X[5] X[3] X[7]

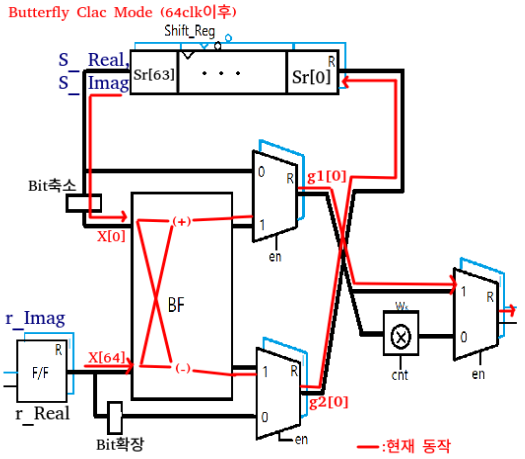
6. 결과

6-1. Synthesis 전 Behavioral Simulation 결과

- 출력(Stage내부)

▫ Butterfly Calc Mode





- 해석 :

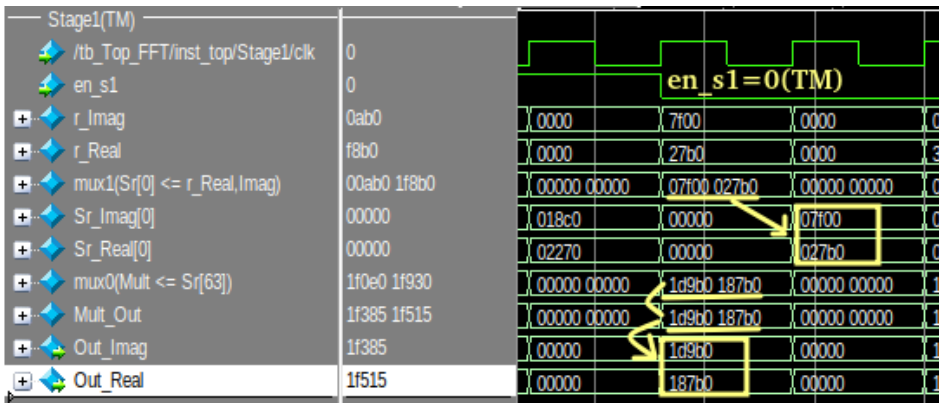
128pt FFT로써 64clk(cnt : 64~127)동안 en_s1=1(BF하는 모습을 확인할수 있고,

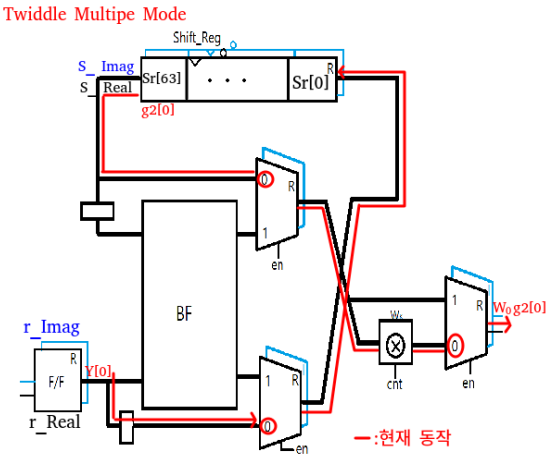
동작 1) g1[0](= x[64] + x[0])가 Stage1의 출력으로 나감

동작 2) g2[0](= x[64] – x[0])가 Shift Register(Sr[0])로 들어감

정상 동작함을 위 Simulation결과로 확인할 수 있다.

▫ Twiddle Factors Multiplicate Mode



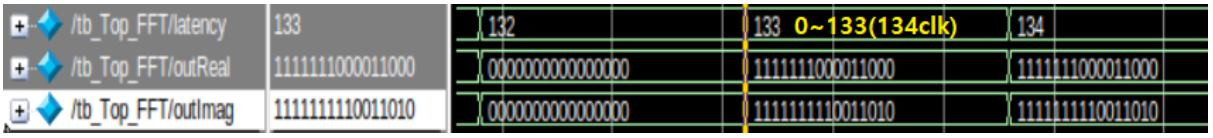


- 해석 :

128pt FFT로써 64clk동안 TM Mode(bf_en = 0)로 동작 하는 모습을 확인할수 있고,

g2[0] (=x[64] - x[0])이 Twiddle Factor(W0)와 곱셈을 하여 W0g2[0]값이 출력되어 나가는 것을 확인 할 수 있다.

▫ 상위 System(Top_FFT)의 Simulation결과



- latency : 134 클락

Stage1 : 64clk(Shift_Register) + 1clk(F/F)

Stage2 : 32clk(Shift_Regsiter) + 1clk(F/F)

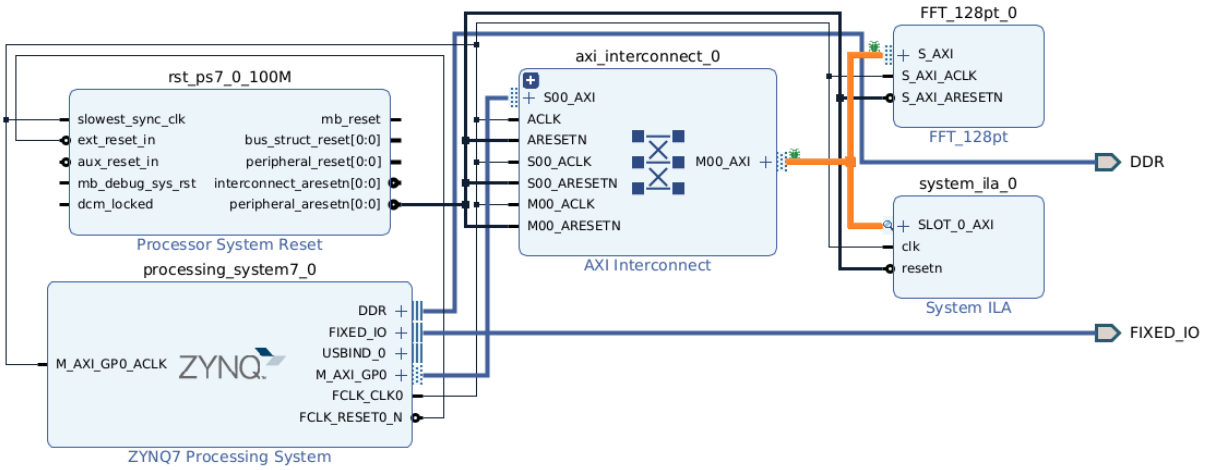
⋮

Stage7 : 1clk(Shift_Regsiter) + 1clk(F/F)

파이프라인으로 인해 latency는 증가했지만 Throughput이 줄어들게 됨으로써 빠른 처리속도를 보여준다.

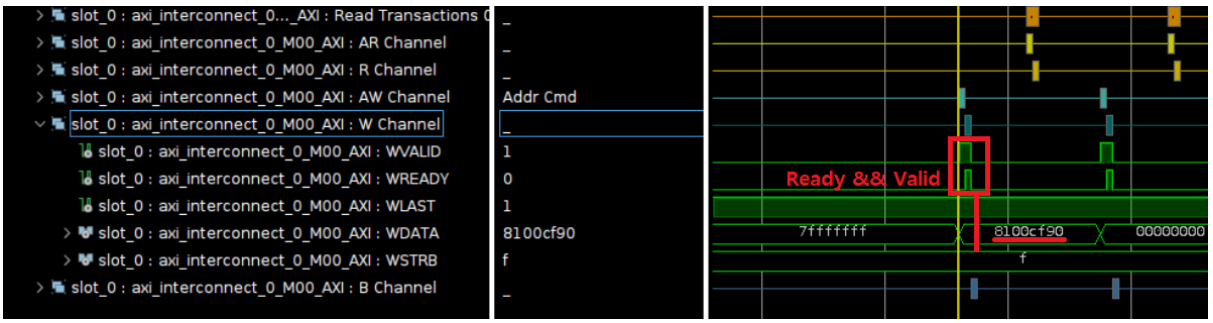
6-2. 실제 동작(Synthesis 이후)

- 사용 칩 : Arty z7-20보드의 Zynq-7000 Chip 을 사용.
- 설명 :Axi Bus를 사용했으며, PS에 입력(input.h) Data를 주어 합성된 PL(FFT)이 Simulation에서 보였던 결과와 같은지 ILA와 Uart Serial 통신을 통해 확인한다.



- ILA

<AXI_BUS_WDATA>



설명 :

PS로 입력해준 Data(input.h)가 Axi-bus를 통해 PL로 들어가고 있음을 확인 할 수 있다.

DATA : 8100[real]CF90[imag]$_{(16)}$ -> 0001_1111_1010_0100[real] 1100_1111_1001_0000[Imag]$_{(2)}$

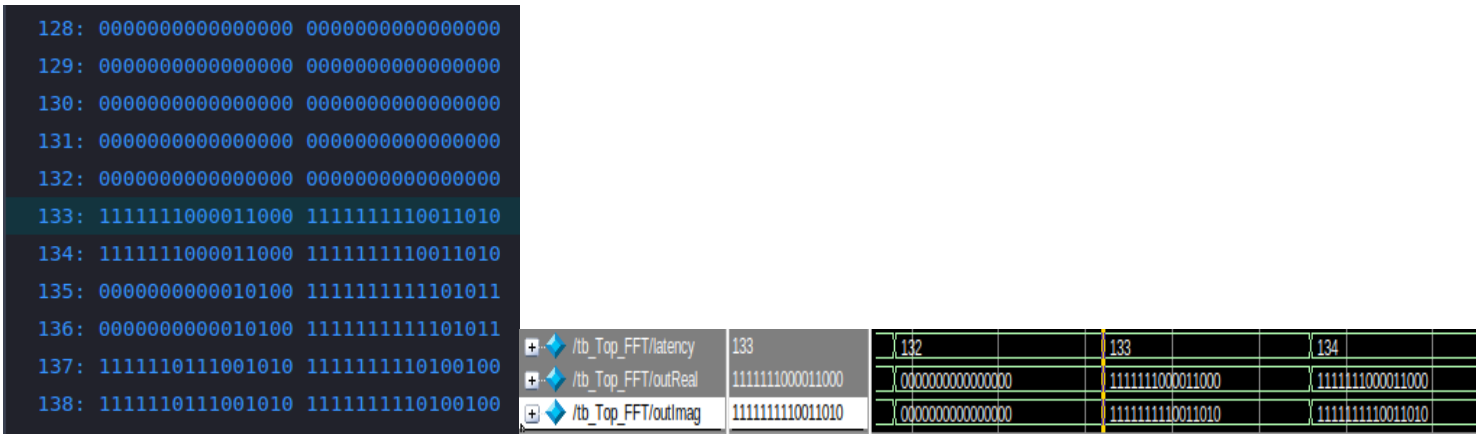
<AXI_BUS_RDATA>

설명 :

Rdata를 통해 PL(FFT)의 출력이 Axi-bus를 통해 Read되고 있음을 알 수 있다.

Data : fe18[real]ff9a[imag]$_{(16)}$ -> 1111_1110_0001_1000[real] 1111_1111_1001_1010[Imag]$_{(2)}$

• UART Serial Monitor로 출력 확인



<합성 후 출력 결과> <Simulation 출력 결과>

Simulation에서 확인했던 출력 값과 latency가 합성 후의 출력 결과와 일치하는 것을 확인 할 수 있다.

7. 소스코드

https://github.com/Hs-Eom/FFT_128pt.git