DSP in VLSI

Homework 5

Fast Fourier Transform (FFT)

I. Purpose

In this homework., we will learn the principle of FFT operation and to realize how to use hardware to implement discrete Fourier transform efficiently.

II. Principle of FFT

Discrete Fourier transform is widely used in signal processing to transform periodic discrete signal between time domain and frequency domain. Given time-domain signal x_n and frequency-domain signal X_k, its equations are provided as follows.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j2\pi nk/N},\ k = 0,1,\dots,N-1 \quad (1)$$

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k e^{j2\pi nk/N},\ n = 0,1,\dots,N-1 \quad (2)$$

where x_n and X_k are time-domain and frequency-domain signals, N is the period. The computation complexity of discrete Fourier transform is quite large. Its multiplication complexity is $O(N^2)$, which is proportional to the DFT size. When N is power of 2, we have a fast algorithm to implement the discrete Fourier transform so that the multiplication complexity can be reduced to $O(Nlog_2N)$. In this homework, we will implement 32-point fast Fourier transform.

A. Fast Fourier transform (FFT) algorithm

In this lab., we will use the radix-2 FFT algorithm. The discrete Fourier transform can be written as $X_k = \sum_{n=0}^{N-1} x_n W_N^{nk}$, where $W_N = e^{-j2\pi/N}$. Separating the frequency-domain signal into the even-indexed part and the odd-indexed part, we can obtain $X_{2r} = \sum_{n=0}^{N-1} x_n W_N^{2rn}$ and $X_{2r+1} = \sum_{n=0}^{N-1} x_n W_N^{(2r+1)n}$. The time-domain signal is then further classified into the first half and the second half. Thus,

$$\begin{aligned} X_{2r} &= \sum_{n=0}^{\frac{N}{2}-1} x_n W_N^{2rn} + \sum_{n=\frac{N}{2}}^{N-1} x_n W_N^{2rn} \\ &= \sum_{n=0}^{N/2-1} x_n (W_N^2)^{rn} + \sum_{n=0}^{N/2-1} x_{(n+\frac{N}{2})} (W_N^2)^{r(n+\frac{N}{2})}. \end{aligned} \quad (3)$$

Because $(W_N^2)^{r(n+\frac{N}{2})} = W_N^{2rn} W_N^{rN} = W_N^{2rn}$, Eq. (3) can be further simplified as

$$X_{2r} = \sum_{n=0}^{N/2-1} x_n (W_N^2)^{rn} + \sum_{n=0}^{N/2-1} x_{(n+\frac{N}{2})} (W_N^2)^{rn}$$

$$= \sum_{n=0}^{N/2-1} \left(x_n + x_{(n+\frac{N}{2})} \right) W_{N/2}^{rn} \tag{4}$$

It can be regarded as $N/2$-point discrete Fourier transform of the new samples that are generated by adding the first half to the second half. Similarly, the frequency-domain odd-indexed part can be written as

$$X_{2r+1} = \sum_{n=0}^{N/2-1} x_n W_N^{(2r+1)n} + \sum_{n=N/2}^{N-1} x_n W_N^{(2r+1)n}$$

$$= \sum_{n=0}^{N/2-1} x_n W_N^{(2r+1)n} + \sum_{n=0}^{N/2-1} x_{(n+\frac{N}{2})} W_N^{(2r+1)(n+\frac{N}{2})}. \tag{5}$$

Because $W_N^{n+N/2} = -W_N^n$, Eq. (5) becomes

$$X_{2r+1} = \sum_{n=0}^{N/2-1} x_n W_N^{(2r+1)n} - \sum_{n=0}^{N/2-1} x_{(n+\frac{N}{2})} W_N^{(2r+1)n}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} \left(x_n - x_{\left(n+\frac{N}{2}\right)} \right) W_N^n W_{N/2}^{rn}. \tag{6}$$

It can be interpreted as the $N/2$-point discrete Fourier transform of the new samples that are generated by multiplying W_N^n to the addition output of the first half and the second half. W_N^n is called twiddle factor. Fig. 1 shows the data flow of decomposition of one 8-point discrete Fourier transform into two 4-point discrete Fourier transforms.

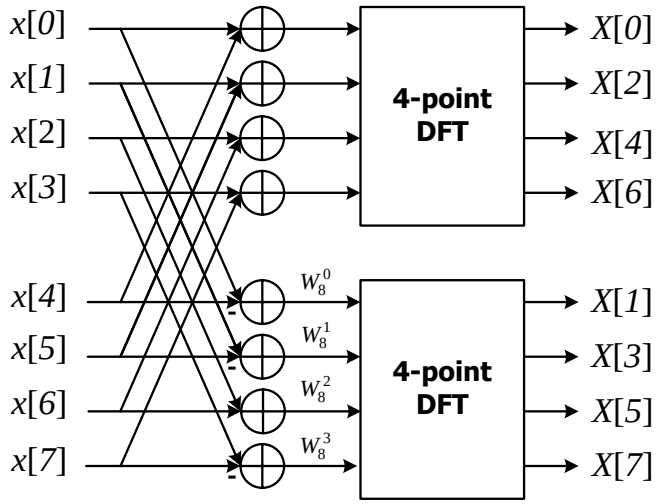

Fig. 1 Decomposition of 8-point discrete Fourier transform

If we proceed the decomposition to compute every $N/2$-point discrete Fourier transform by two $N/4$-point discrete Fourier transform, then we can obtain the fast algorithm to compute N-point discrete Fourier transform. Let $g_n = x_n + x_{(n+\frac{N}{2})}$ and $r = 2m$. From (4)

$$X_{4m} = G_{2m} = \sum_{n=0}^{\frac{N}{2}-1} g_n W_{N/2}^{2mn} = \sum_{n=0}^{\frac{N}{2}-1} (g_n + g_{n+N/4}) W_{N/4}^{mn} \tag{7}$$

$$X_{4m+2} = G_{2m+1} = \sum_{n=0}^{\frac{N}{2}-1} g_n W_{N/2}^{2mn+n} = \sum_{n=0}^{\frac{N}{2}-1} (g_n - g_{n+N/4}) W_{N/2}^{n} W_{N/4}^{mn}. \quad (8)$$

Fig. 2 shows the signal flow graph (SFG) of an 8-point fast Fourier transform. It is called decimation-in-frequency FFT because the frequency domain samples do not appear in the normal order.

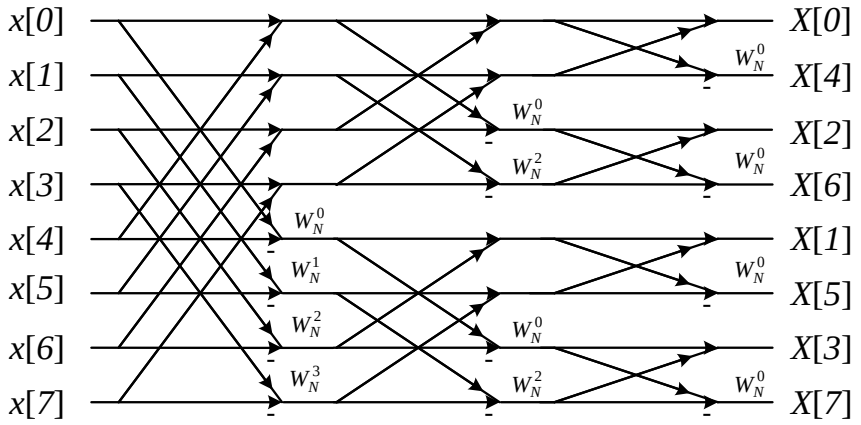

Fig. 2 Signal flow graph (SFG) of an 8-point fast Fourier transform.

To make the frequency-domain samples output in order, we need to re-arrange the decimated frequency-domain samples. Observing the sequence indexes, we can realize that the outputs are in an order that uses the bit-reversed binary representation, as shown in Fig. 3. Consequently, if we want to get output in order, we only need to reverse the binary representation of the index. This operation is called bit-reversal or bit-reverse re-ordering.

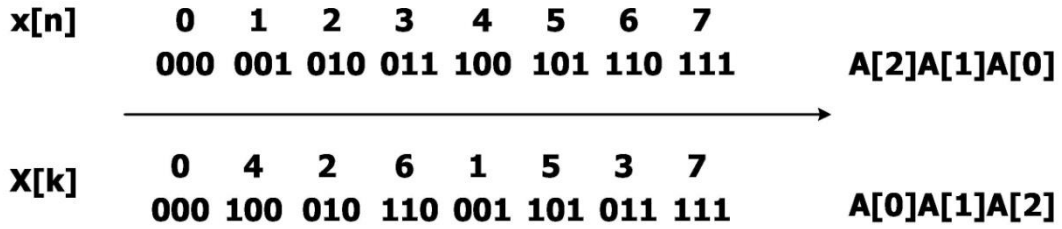

Fig. 3 Frequency-domain samples in bit-reversed order.

B. Pipelined architecture for FFT

In this homework., we will implement one of the famous pipelined fast Fourier transform architectures, named multi-path delay commutator (MDC) architecture, which processes two inputs and generates two outputs at each clock cycle. Fig. 4 is the block diagram of an 8-point MDC FFT by cascading three stages. Each processing stage contains delay buffers, a butterfly unit, and a multiplier. The butterfly unit

computes addition and subtraction of two input signals. The commutator and delay buffers adjust the input sequence. The multiplier handles the twiddle-factor multiplication. We also need to control the multiplicand to the multipliers to arrange the correct twiddle factor for multiplication.

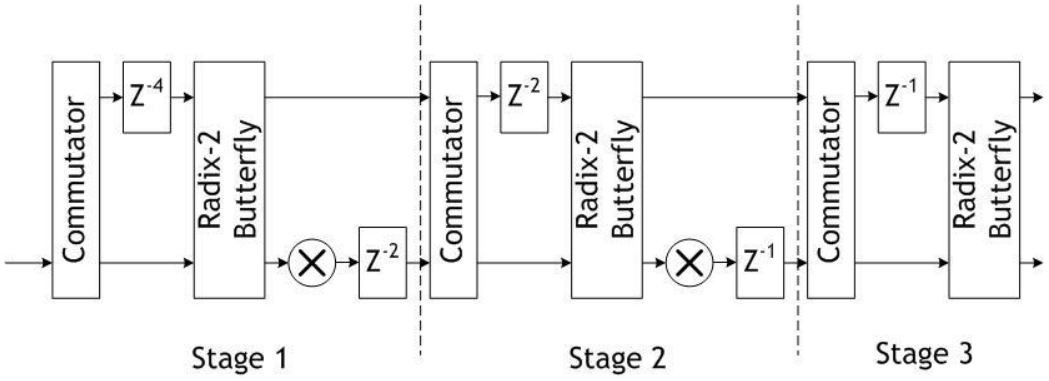

Fig. 4 pipelined architecture for 8-point FFT.

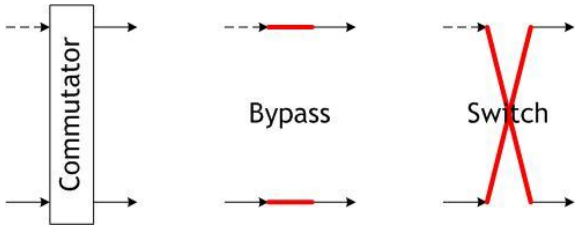

Fig. 5 Two modes of the commutator.

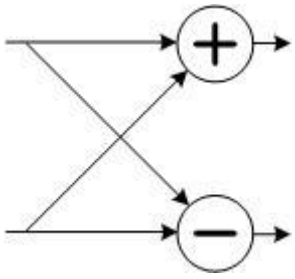

Fig. 6 Butterfly unit

Using 8-point FFT as an example, we can realize the operation of the radix-2 multi-path delay commutator architecture with the signal-flow graph. As shown in Table I, two inputs of the butterfly unit are labeled as the upper input and the lower input. Similarly, it has the upper output and the lower output. Assume that input signals are denoted as $x_0,\ x_1, \ldots,\ x_7$. The sum of x_k and $x_{k+N/2}$ is denoted as g_k. The difference of x_k and $x_{k+N/2}$ is denoted as h_k. p_k is the sum of g_k and $g_{k+N/4}$. q_k is the difference of g_k and $g_{k+N/4}$. m_k is the sum of h'_k and $h'_{k+N/4}$. n_k is the difference of h'_k and $h'_{k+N/4}$. h'_k is the output of h_k multiplied by the twiddle factor.

Table I The sequence of 8-point FFT implemented by MDC architecture

	Time	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Stage 1	Commutator LI	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	y_0	y_1	y_2	y_3	y_4	y_5	y_6
	Commutator UO	x_0	x_1	x_2	x_3					y_0	y_1	y_2	y_3			
	Commutator LO					x_4	x_5	x_6	x_7					y_4	y_5	y_6
	Butterfly UI					x_0	x_1	x_2	x_3					y_0	y_1	y_2
	Butterfly LI					x_4	x_5	x_6	x_7					y_4	y_5	y_6
	Butterfly UO					g_0	g_1	g_2	g_3							
	Butterfly LO					h_0	h_1	h_2	h_3							
	Multiplier					W_8^0	W_8^1	W_8^2	W_8^3					W_8^0	W_8^1	W_8^2
Stage 2	Commutator UI					g_0	g_1	g_2	g_3							
	Commutator LI							h'_0	h'_1	h'_2	h'_3					
	Commutator UO					g_0	g_1	h'_0	h'_1							
	Commutator LO							g_2	g_3	h'_2	h'_3					
	Butterfly UI							g_0	g_1	h'_0	h'_1					
	Butterfly LI							g_2	g_3	h'_2	h'_3					
	Butterfly UO							p_0	p_1	m_0	m_1					
	Butterfly LO							q_0	q_1	n_0	n_1					
	Multiplier							W_8^0	W_8^2	W_8^0	W_8^2					

Stage 3	Commutator UI							p_0	p_1	m_0	m_1					
	Commutator LI								q'_0	q'_1	n'_0	n'_1				
	Commutator UO							p_0	q'_0	m_0	n'_0					
	Commutator LO								p_1	q'_1	m_1	n'_1				
	Butterfly UI								p_0	q'_0	m_0	n'_0				
	Butterfly LI								p_1	q'_1	m_1	n'_1				
	Butterfly UO								X_0	X_2	X_1	X_3				
	Butterfly LO								X_4	X_6	X_5	X_7				

C. Lookup table for twiddle factor

Twiddle factor can be saved in a table with the possible sin and cosine values. Use the counter to control the table output as the multiplicand to the multiplier.

D. Inverse Fourier transform

From the duality of DFT and IDFT in Eqs. (1) and (2), we can see that the inverse FFT can be interpreted as the following operation.

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k e^{j2\pi nk/N} = \frac{1}{N}\left(\sum_{k=0}^{N-1} (X_k)^* e^{-j2\pi nk/N}\right)^*$$

It can be interpreted as using the complex conjugate frequency domain signal $(X_k)^*$ as the input and then sending it to the FFT module. The time-domain signal x_n can be obtained by the complex conjugate of the FFT output divided by N. However, since N is power of 2, it is not necessary to implement a divider. We only need to change the scaling when we translate the fixed-point integer to the true values. Hence, the IFFT can be implemented by the FFT module with change of the sign values of the imaginary part of the FFT inputs and outputs.

We can extend the concept of the 8-point FFT architecture to the 32-point FFT operation. The hardware architecture of 32-point FFT is shown in Fig. 6.

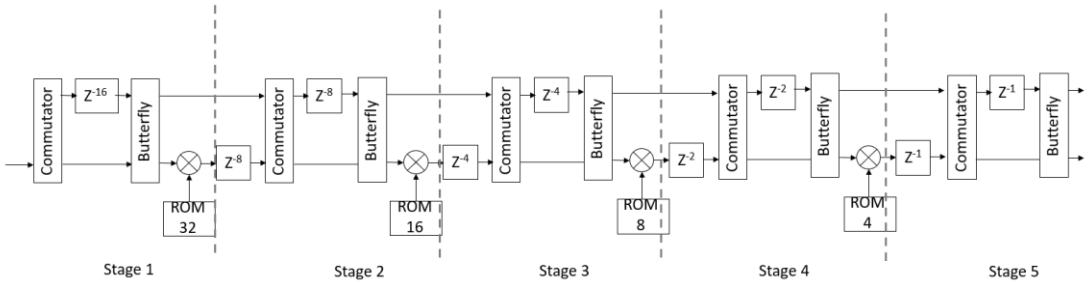

Fig. 7 Architecture of the 32-point FFT operation.

III. Design Procedures

A. Fixed-point representation for a given signal-to-quantization noise ratio (SQNR) requirement

1. In an FFT processor, usually the quantization error accumulates as the butterfly operations proceed. Hence, it is quite common to decide the fixed-point representation for FFT processors stage by stage. Given a set S={1+j, 1-j, -1+j, -1-j}, please randomly generate 32 samples from the set S, denoted as $A_0 \sim A_{31}$. Use Matlab/Python command "ifft" to transform $A_0 \sim A_{31}$ into the time-domain signal $x_0 \sim x_{31}$. Write Matlab program to implement MDC FFT. Use $x_0 \sim x_{31}$ as the MDC FFT input. Denote the MDC FFT output as $X_0 \sim X_{31}$.
2. Observe the bit-reversal property of decimation in frequency. Write a Matlab/Python program to change the bit-reversed order into normal order. Check the difference between $X_0 \sim X_{31}$ and $A_0 \sim A_{31}$.
3. The SQNR requirement of the FFT processor is **30dB**.

$$\mathrm{SQNR} = 10\log\left(\frac{\mathrm{E}\{|\mathrm{X_i}|^2\}}{\mathrm{E}\{|\widehat{\mathrm{X}}_\mathrm{i}\text{-}\mathrm{X_i}|^2\}}\right)$$

where $\widehat{\mathrm{X}}_\mathrm{i}$ denotes the quantized output of $\mathrm{X_i}$. Now decide the fractional part word-length at every stage **sequentially**. Namely, when you decide the fractional part word-length of the second stage, fix the fractional part wordlength setting that you determine for the first stage. If you use w_f for stage i, it means the butterfly output and twiddle factor multiplication output at stage i are truncated to w_f bits in the fractional part.

B. Module design

4. Design the commutator and butterfly in Fig. 5 and Fig. 6 so that the commutator and butterfly can perform switch/calculation and bypass operation. Given that there is a counter counting from 0 to $N-1$ when the first valid input enters into the N-point FFT module. Show your control signal that sends to the commutator module at each stage to control two operation modes. (0: switch, 1: bypass operation) as

well as the control signal to the butterfly unit (0: computation, 1: bypass)

5. From Table I, derive your control signal to the complex multiplier block to control the multiplication and bypass operation. (0: Multiplication, 1: bypass operation)
6. Since sine and cosine functions have symmetry property, for the ROM table with 32 phases, we will only save the sine and cosine values in the first quadrant. Please design the lookup table and the way to generate correct results for four quadrant.
7. Design the bit-reversal module so that the decimated frequency-domain samples can be sent out in order. You can allocate 2 memory banks, each having 32 elements. The FFT output is first saved in the memory bank A and the normal-order output is then fetched from the memory bank B. We call it ping-pong access as shown in Fig. 8. Now, write Verilog to design the ping-pong access bit-reversal module. Use bit reversed indices as the inputs. Verify the correctness of the output as given in Fig. 9.

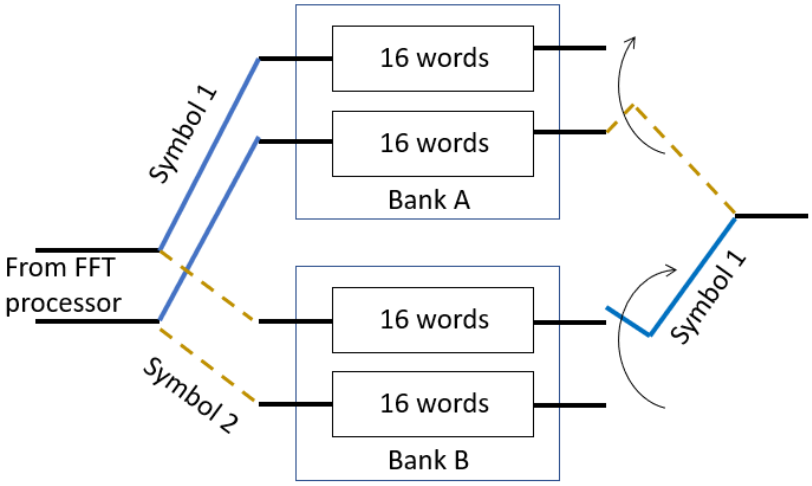

Fig. 8 Ping-pong access

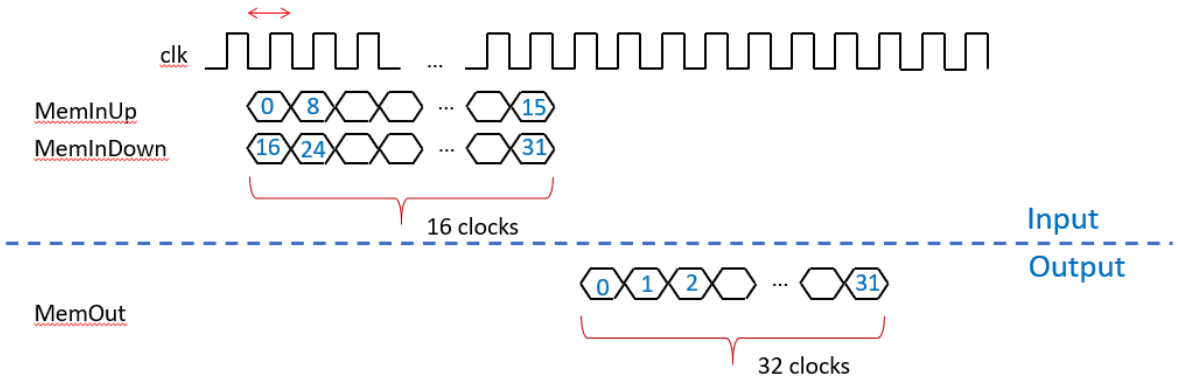

Fig. 9 Timing diagram for verification

C. FFT Functional Verification

8. Connect the modules in part B and then first check the output of the FFT processor using MDC architecture.
9. Now, check the steaming-in streaming out capability of your architecture with the following timing diagram. Feed 3 FFT symbols (96 samples) and observe the outputs.

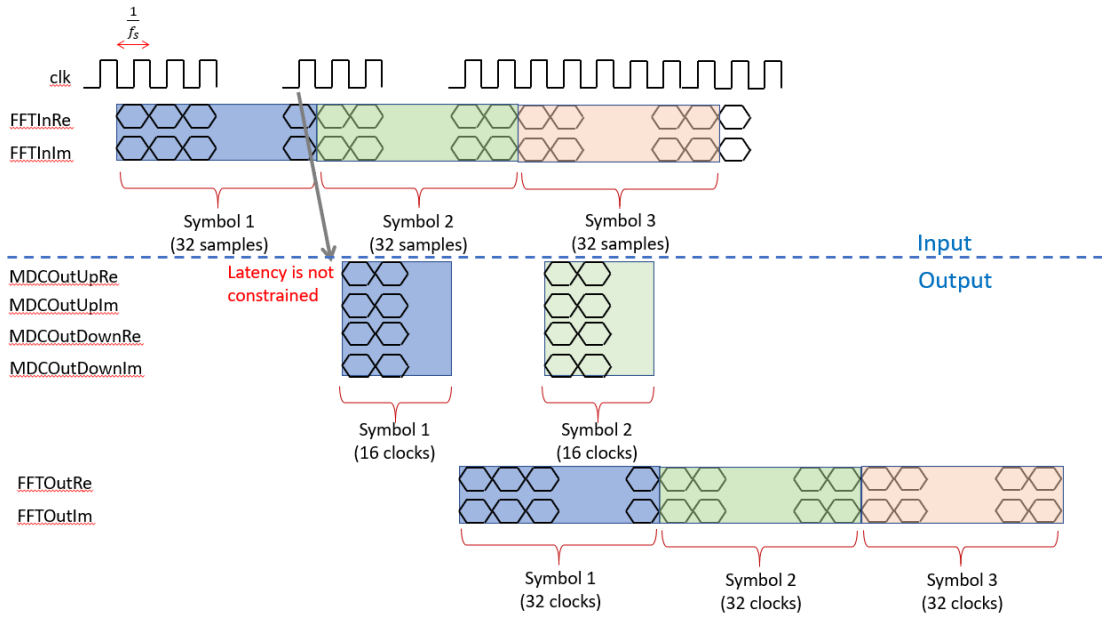

Fig. 10 Timing diagram for MDC FFT verification.

IV. Required Results

1. **(Step 1)** Please use **FFTInput32.mat** as the inputs. Use your MDC FFT Matlab/Python program to generate the 32 FFT outputs. The program outputs should be in a 2×16 array. Because they are in bit-reversed order, write a program to generate their associated frequency domain indices. Now, you have an 4×16 array containing the frequency domain indices and the MDC outputs. List the array. (Maybe you can copy the results in Work Space and paste onto your report.) (10%)
2. (**Step 2**) Please use **FFTInput32.mat** as the inputs. Show that your program for output re-ordering is correct. Draw the real part and imaginary part of $X_0 \sim X_{31}$. (10%) Compare the results of $X_0 \sim X_{31}$ to $A_0 \sim A_{31}$. Draw the absolute error of each sample. The error should be small than 10^{-10} because of double-precision floating-point operations. If the error is not acceptable, the first 20 points will be deducted, too. (10%)
3. (**Step 3**) Generate your own inputs of 96 samples. Draw the SQNR versus fractional part word-length N stage by stage. (30%)

(a) Quantize the first stage. Evaluate the FFT output SQNR versus fractional part word-length N for N=7, 8, 9, 10,..., 13, 14,15 for stage 1. Draw the figure.

(b) Fix the setting for stage 1. Quantize the second stage. Evaluate the FFT output SQNR versus fractional part word-length N for N=7, 8, 9, 10,..., 13, 14,15 for stage 2. Draw the figure.

(c) Fix the setting for stage 1 and 2. Quantize the second stage. Evaluate the FFT output SQNR versus fractional part word-length N for N=7, 8, 9, 10,..., 13, 14,15 for stage 3 and so on. Draw the figure

(d) Repeat until stage 5. So, you have five figures.

(e) Then, decide the fractional part word-length for twiddle factors of all the stages. Draw the FFT output SQNR versus fractional part word-length of twiddle factors.

4. **(Step 4)** Now, use a 5-bit counter counting from 0 to 31, which is synchronized to the input index. Generate the control signals of commutator and butterfly units for each stage. Show how you generate them and why? (20%)
5. **(Step 5)** Generate the control signals of complex multipliers for each stage. Show how you generate them and why? (10%)
6. **(Step 6)** Explain the way that you use the sine/cosine values in the first quadrant to generate the required 32 phases for ROM 32. Draw the block diagram (10%)
7. **(Step 7)** Show the timing diagram of behavior simulation for bit-reverse re-ordering with ping-pong accessed bit-reversal buffer as in Fig. 9. (15%)
8. **(Step 8)** Please use **FFTInput32.mat** as the inputs. Quantize them using the wordlength selected by yourself. Show the timing diagram of behavior simulation for MDC FFT. Compared the results to your answer in Q1(Step 1). Draw the error for 32 samples of real part and imaginary part. (25%)
9. **(Step 9)** Use your own inputs of 96 samples in Q3 (Step3). Show the timing diagram of behavior simulation with streaming-input and streaming-output. Draw the error for 96 samples of real part and imaginary part. (25%) Calculate the SQNR of 96 samples.
10. **(Step 10)** Insert D flip-flop at the input and output. Synthesize your design. Provide the report of max delay. Note that if you did not insert internal pipeline registers, the critical path is long and the operating frequency is not high. (10%)
11. **(Step 11)** Insert pipeline registers to accelerate your design. For FPGA flow, the target operating clock frequency (f_s) is 75MHz. For cell-based design flow, the target operating frequency (f_s) is 130MHz. Show the timing diagram of **post-synthesis** simulation results with proper clock period settings $(1/f_s)$. Draw the error for 96 samples of real part and imaginary part. (25%)

12. Please upload your report containing the required results and your Verilog codes as well as Matlab/Python codes to NTUCool.

A gentle reminder:
If you can not find your bugs in either Matlab/python program or Verilog code for 32-point operation, try to debug in 8-point FFT operation and then 16-point FFT operation. You can check the operation results stage by stage in 8-point FFT as described in Table I. The operations are regular. Usually, if you figure out the regularity, you can make it.