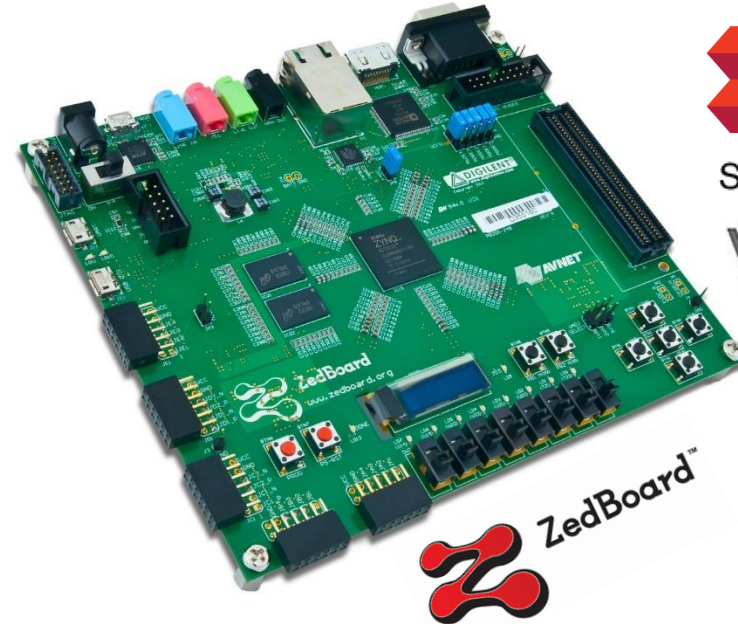




ECE573

Advanced Embedded Logic Design (AELD)

Dr. Sumit J Darak
Algorithms to Architectures Lab
Associate Professor, ECE Department
IIIT Delhi
<http://faculty.iiitd.ac.in/~sumit/>

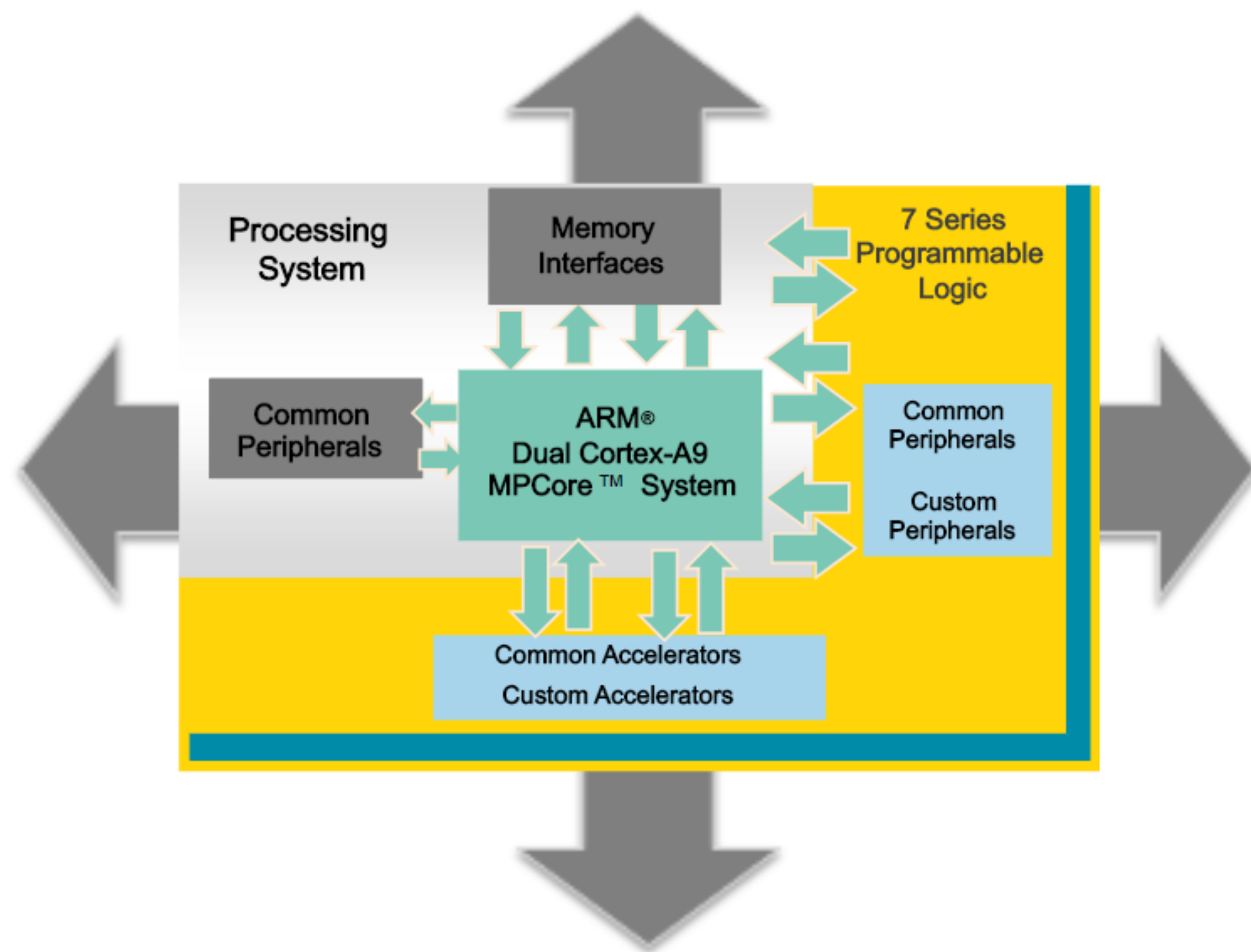


- The material for this presentation is taken from various books, courses and Xilinx/ARM XUP resources. The instructor does not claim ownership of the material presented in this class.

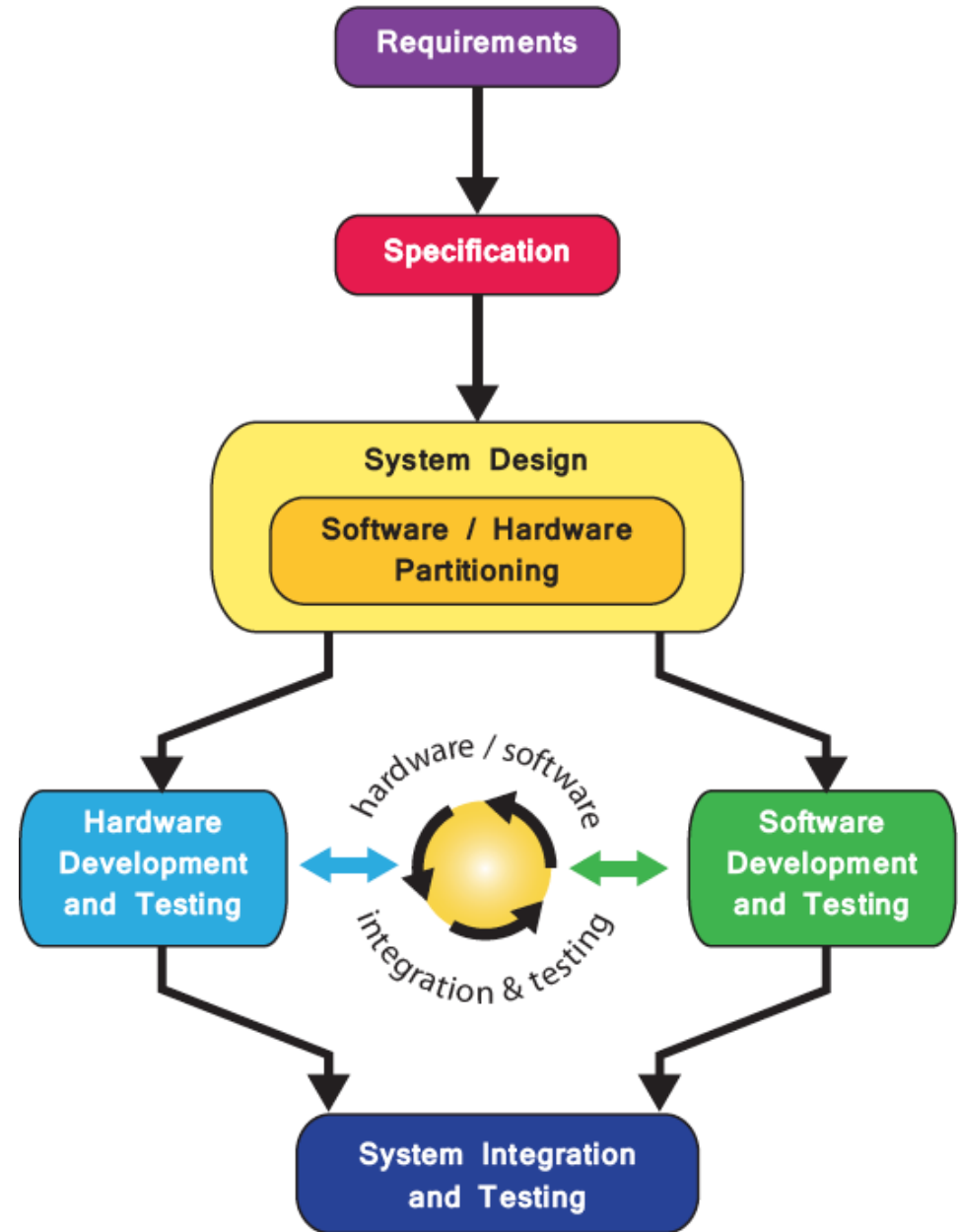
Lab 2

- In this lab, we will discuss the 8-point FFT example with a focus on debugging and execution time calculation. We will also discuss in-built code-optimization feature in SDK. Then, we will discuss generalized FFT code for large dimension
- **Topics to explore:** 1) FFT algorithm, 2) SDK debugging feature, 3) Execution time calculation using in-built timer, 3) Code-optimization on SDK for ARM processor, 4) Large size FFT, 5) Stack and heap size, Header file, Math Library and sleep function
- **Self-study:** Implement floating point arithmetic and matrix multiplication on ARM processor

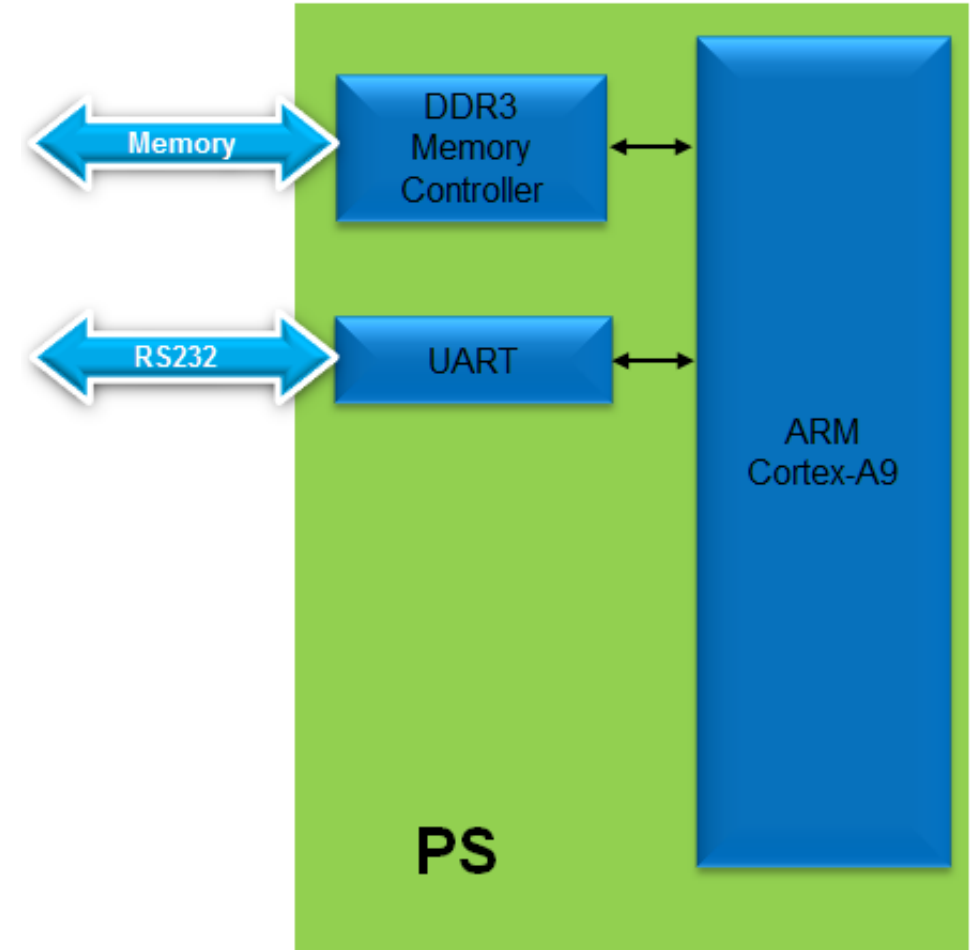
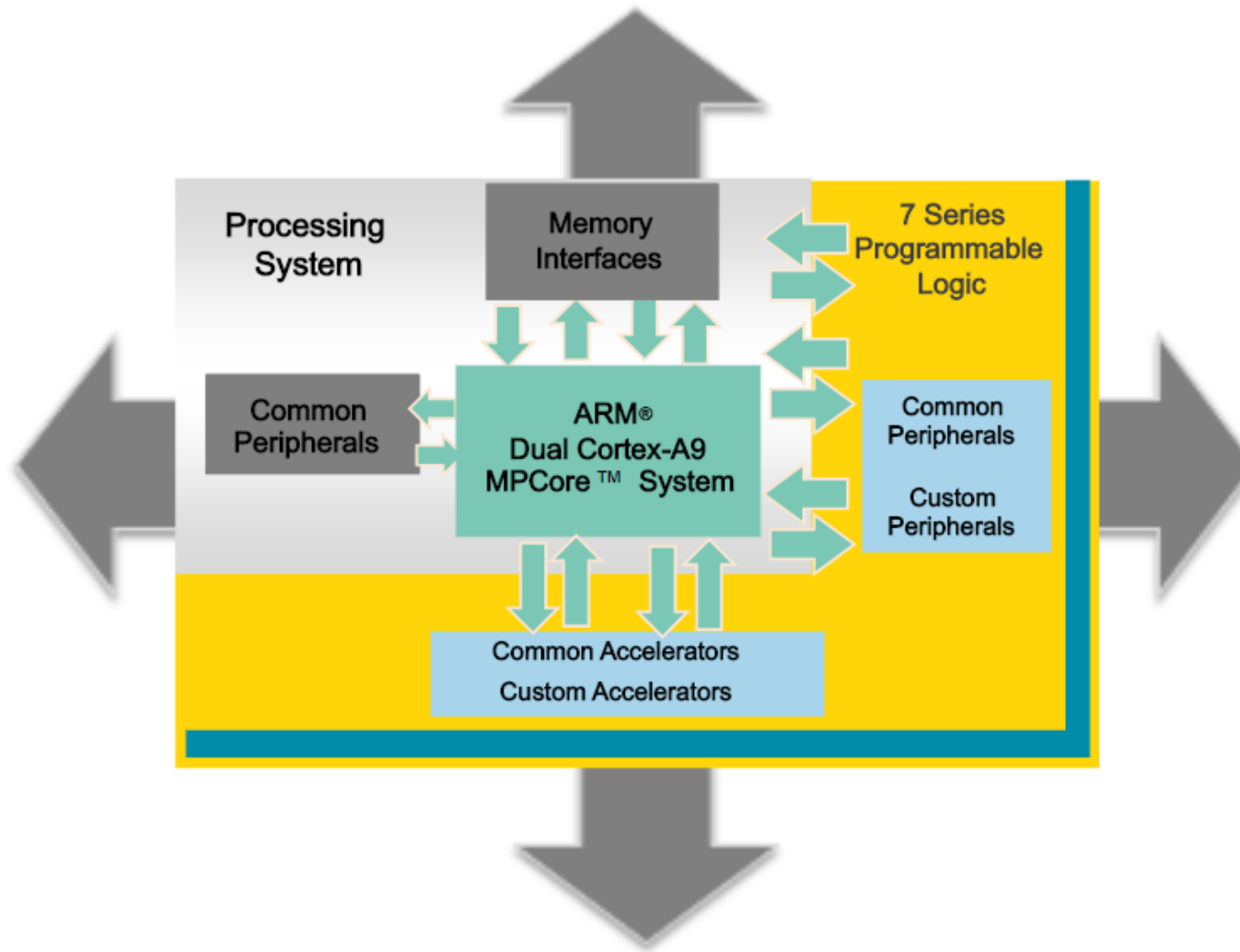
Zynq SoC



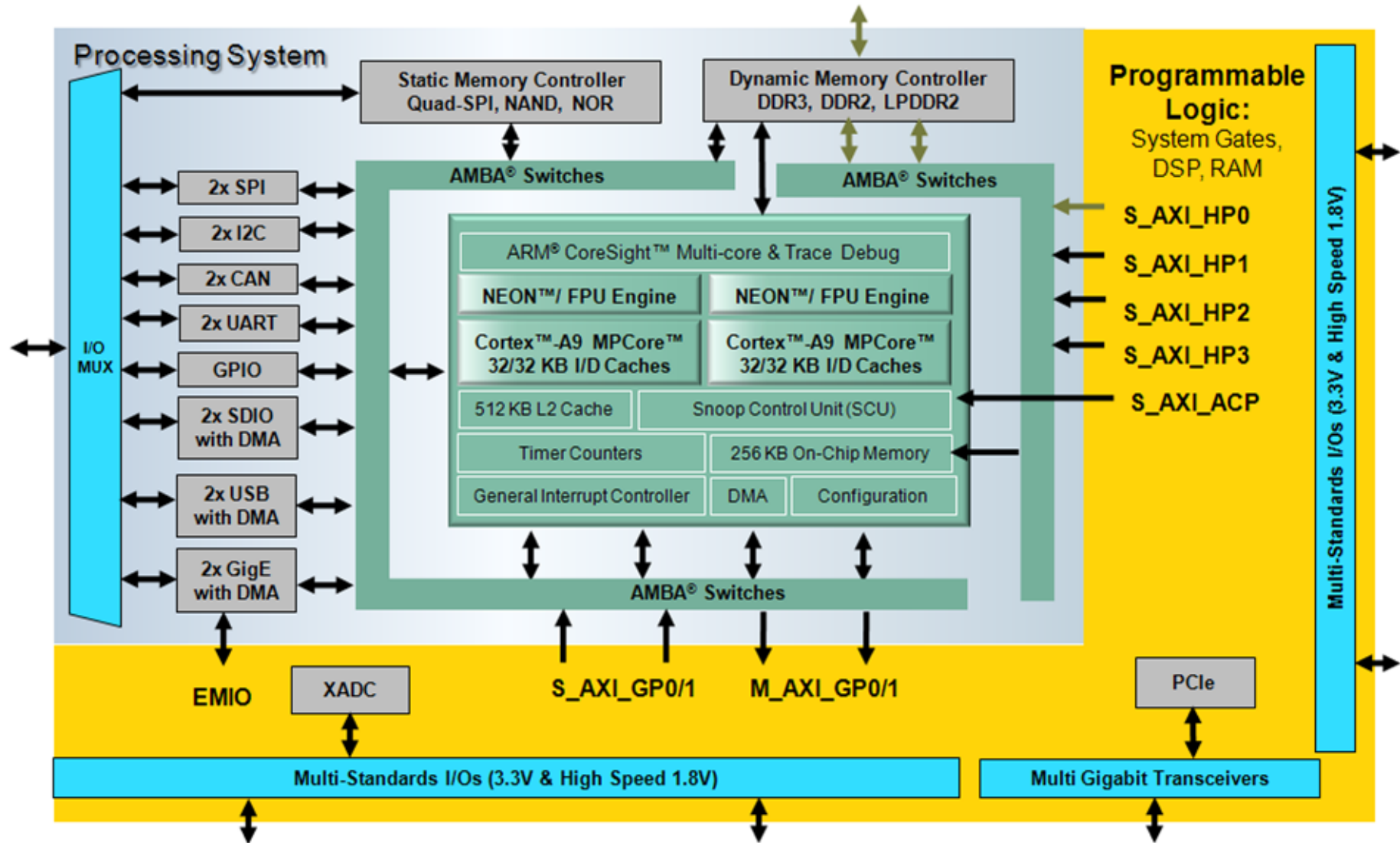
Zynq Design Flow



Lab 2



Zynq Architecture: PS and PL



Vivado Design Flow

Same as Lab 1

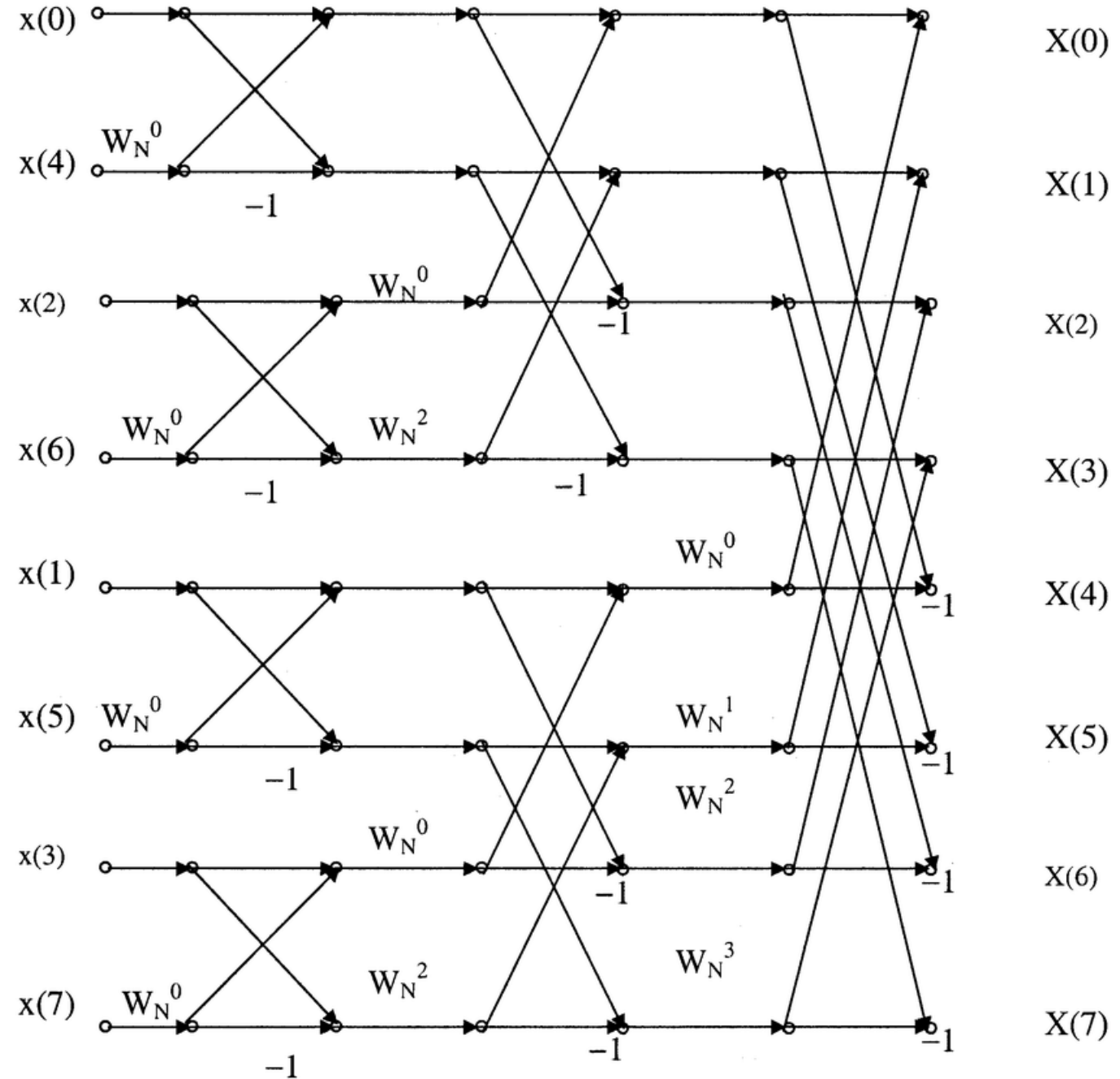
SDK Design Flow

Same as Lab 1 Except application code

FFT on ARM Processor

FFT on ARM Processor

- FFT Size = 8
- No of butter fly stages=3



$$W_N^0 = 1, W_N^1 = (1-j)/\sqrt{2}, W_N^2 = -j, W_N^3 = -(1+j)/\sqrt{2}$$

FFT on ARM Processor

```
#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
#define FFT_Size 8
const float complex twiddle_factors[FFT_Size/2] = {1-0*I, 0.7071067811865476-0.7071067811865475*I, 0.0-1*I, -0.7071067811865475-0.7071067811865476*I};
.....
int main()
{
    → // For FFT_Size point FFT, define the input.
    → // You may modify the code to take the input from user via UART
    ... float complex FFT_input[FFT_Size] = {11+23*I, 32+10*I, 91+94*I, 15+69*I, 47+96*I, 44+12*I, 96+17*I, 49+58*I};
    ... // FFT output will be stored in this variable
    ... float complex FFT_output[FFT_Size];
    ... // Variable for intermediate outputs
    ... float complex FFT_rev[FFT_Size];

    ... // Print the FFT input on the UART
    ... printf("\n FFT input:\r\n");
    > for (int i = 0; i < FFT_Size; i++)
    > {
    > → printf("%f %f\n", crealf(FFT_input[i]), cimagf(FFT_input[i]));
    > }
```

FFT on ARM Processor

- Matlab Code

```
// Equivalent Matlab Code for FFT
// clc;
// clear;
// FFT_Size = 8;
// input_real = [11,32,91,15,47,44,96,49];
// input_imag = [23,10,94,69,96,12,17,58];
// FFT_input = complex(input_real,input_imag);
// FFT_out = fft(FFT_input,FFT_Size);
// disp(real(FFT_out));
// disp(imag(FFT_out));
```

FFT on ARM Processor

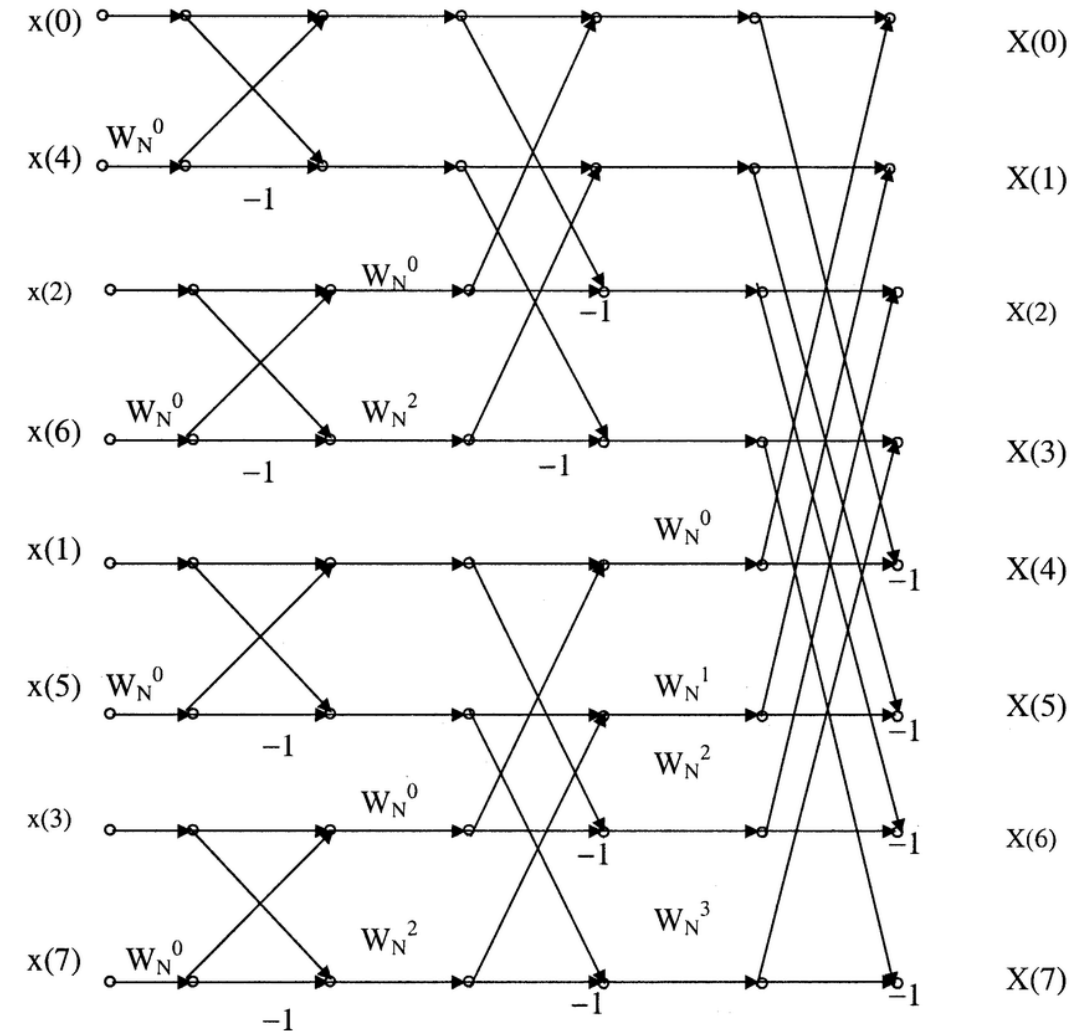
```
....// As discussed in the handout, FFT involves two tasks:
....// 1) Reorder of the inputs to get output in the normal order
....// 2) Multiplications using multi-stage butterfly approach
→ InputReorder(FFT_input, FFT_rev); // Task 1
....FFTStages(FFT_rev, FFT_output); // Task 2

....// Print the FFT output on the UART
....printf("\n FFT output: \r\n");
....for (int i = 0; i < FFT_Size; i++)
{
....printf("%f %f\n", crealf(FFT_output[i]), cimagf(FFT_output[i]));
....}

....// Modify this code for large size FFT
....// How you can generalize the code for any FFT size (limited to power of two)
....// Receive the FFT size and FFT input from User
}
```

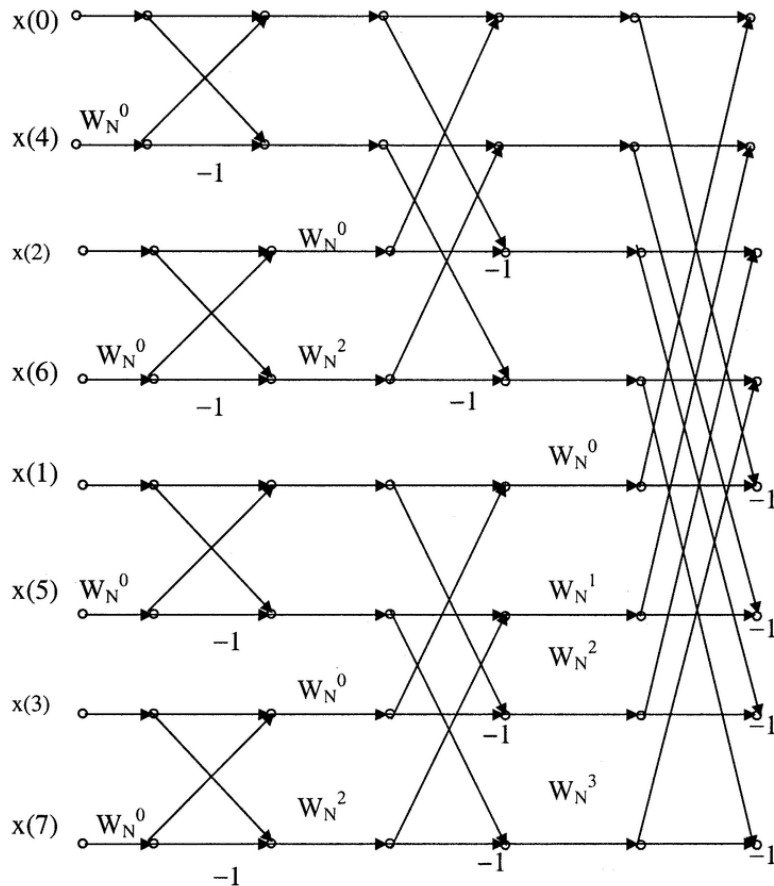
FFT on ARM Processor

```
// This function reorders the input to get the output in the normal order
// Refer the handout for the desired input order
const int input_reorder[FFT_Size] = {0, 4, 2, 6, 1, 5, 3, 7};
void InputReorder(float complex dataIn[FFT_Size], float complex dataOut[FFT_Size])
{
    for (int i = 0; i < FFT_Size; i++)
    {
        dataOut[i] = dataIn[input_reorder[i]];
    }
}
```



$$W_N^0 = 1, W_N^1 = (1-j)/\sqrt{2}, W_N^2 = -j, W_N^3 = -(1+j)/\sqrt{2}$$

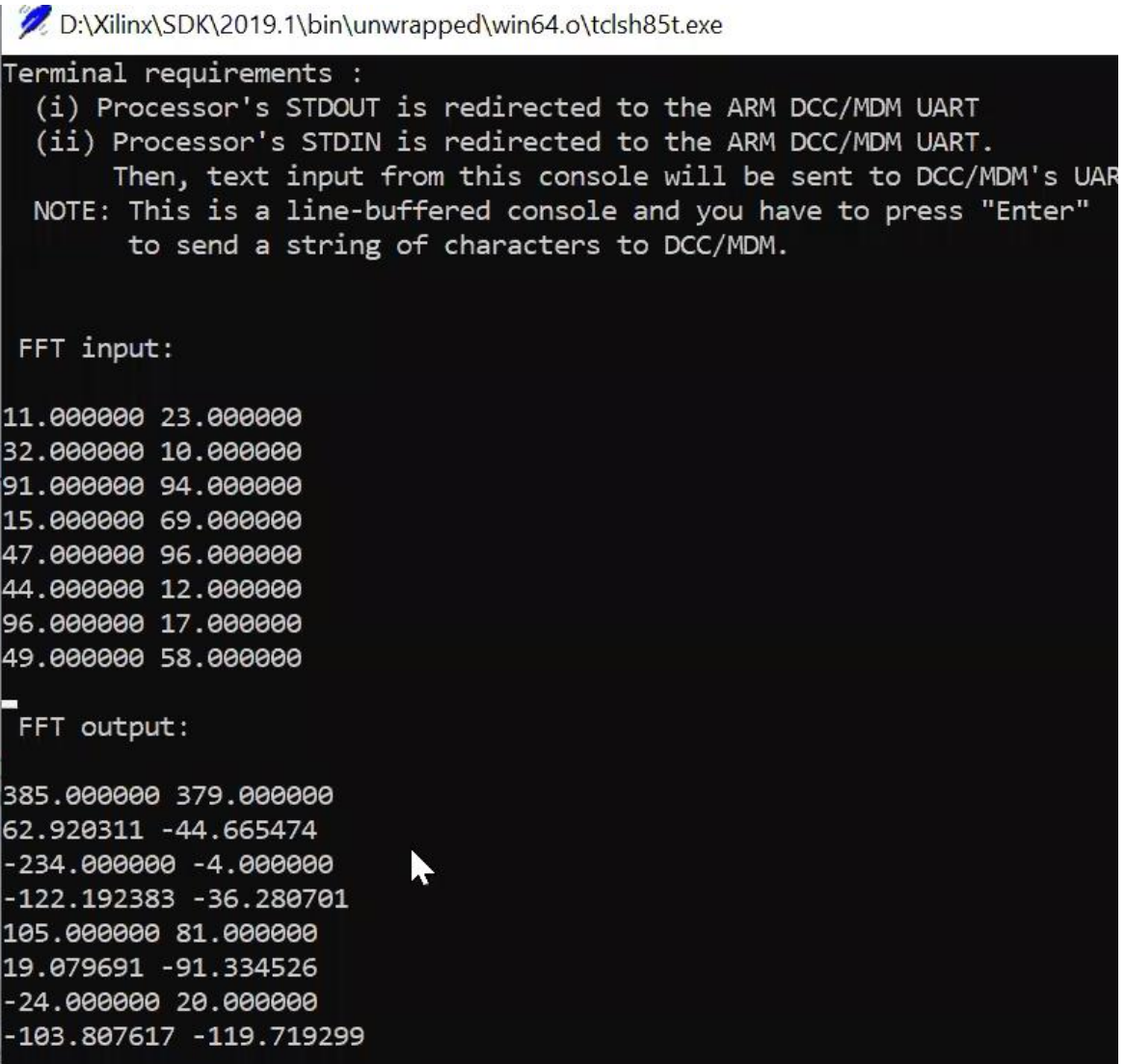
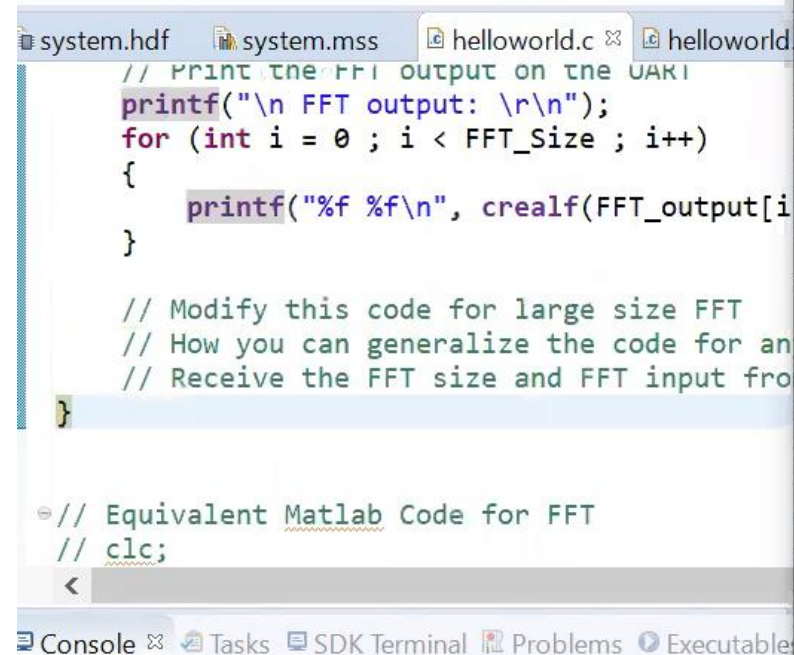
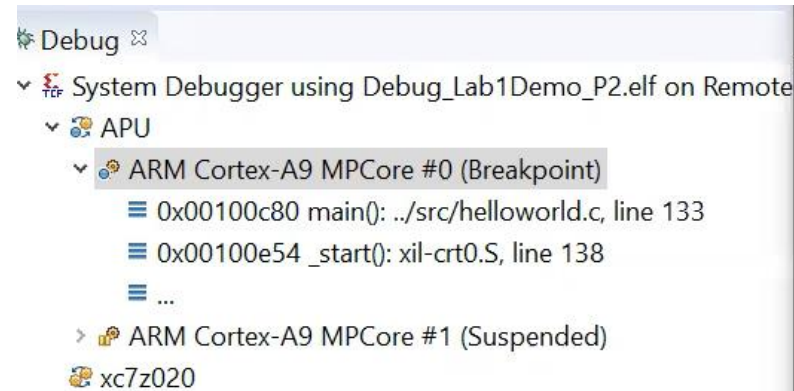
FFT on ARM Processor



$$W_N^0 = 1, W_N^1 = (1-j)/\sqrt{2}, W_N^2 = -j, W_N^3 = -(1+j)/\sqrt{2}$$

```
// For FFT of size FFT_Size, the number of butterfly stages are 2^stages = FFT_Size.
// For 8-point FFT, there are three butterfly stages.
void FFTStages(float complex FFT_input[FFT_Size], float complex FFT_output[FFT_Size])
{
    float complex stage1_out[FFT_Size], stage2_out[FFT_Size];
    // Stage 1
    for (int i = 0; i < FFT_Size; i=i+2)
    {
        stage1_out[i] = FFT_input[i] + FFT_input[i+1];
        stage1_out[i+1] = FFT_input[i] - FFT_input[i+1];
    }
    // Stage 2
    for (int i = 0; i < FFT_Size; i=i+4)
    {
        for (int j = 0; j < 2; ++j)
        {
            stage2_out[i+j] = stage1_out[i+j] + twiddle_factors[2*j]*stage1_out[i+j+2];
            stage2_out[i+2+j] = stage1_out[i+j] - twiddle_factors[2*j]*stage1_out[i+j+2];
        }
    }
    // Stage 3
    for (int i = 0; i < FFT_Size/2; i++)
    {
        FFT_output[i] = stage2_out[i] + twiddle_factors[i]*stage2_out[i+4];
        FFT_output[i+4] = stage2_out[i] - twiddle_factors[i]*stage2_out[i+4];
    }
}
```


FFT on ARM Processor



SDK Debugging

Detailed Handout is attached at the end

SDK Debugging

- Explore Variable, Breakpoints, Registers tab and step-by-step debugging.

The screenshot displays the SDK debugging interface. The 'Window' menu is open, showing options like 'New Window', 'Editor', 'Appearance', 'Show View', 'Perspective', 'Navigation', and 'Preferences'. The 'Show View' option is selected, and a submenu is visible with 'Breakpoints', 'Console', 'Debug', 'Disassembly', 'Executables', 'Expressions', 'Memory', and 'Modules'. The 'Disassembly' view is active, showing assembly code for a function. The code includes instructions like 'stm', 'ldm', 'movw', 'movt', 'blx', and 'printf'. The address 00100b5c is highlighted, corresponding to the instruction 'movw r0, #110'. The 'Disassembly' tab is selected in the bottom right corner.

Window Help

- New Window
- Editor
- Appearance
- Show View
 - Breakpoints Alt+Shift+Q, B
 - Console Alt+Shift+Q, C
 - Debug
 - Disassembly
 - Executables
 - Expressions
 - Memory
 - Modules
- Perspective
- Navigation
- Preferences

Disassembly

Enter location here Refresh View

```
00100b50: stm r12!,  
00100b54: ldm lr, {  
00100b58: stm r12,  
110 printf("\n  
00100b5c: movw r0, #  
00100b60: movt r0, #  
00100b64: blx +2052  
111 for (int  
00100b68: mov r3, #  
00100b6c: str r3, [  
00100b70: b +84  
113 print
```

The screenshot shows the 'Variables' window in the SDK debugging interface. It displays a table of variables and their values. The 'FFT_output' variable is highlighted in yellow. The 'FFT_rev' variable is expanded, showing its elements. The 'Value' column shows the values of the variables.

Name	Type	Value
> FFT_output	complex float [8]	[385.0000+3
▼ FFT_rev	complex float [8]	[11.00000+2
(x): [0]	complex float	11.00000+23
(x): [1]	complex float	47.00000+96

47.00000+96.00000i

SDK Debugging

The screenshot displays the SDK debugging environment. The top-left pane shows the 'Debug' console with the following configuration:

- System Debugger using Debug_Lab1Demo_P2.elf on RemoteBoard (RemoteBoard)
- APU
 - ARM Cortex-A9 MPCore #0 (Breakpoint)
 - 0x00100c00 main(): ../src/helloworld.c, line 124
 - 0x00100e54 _start(): xil-crt0.S, line 138
 - ...
 - ARM Cortex-A9 MPCore #1 (Suspended)
 - xc7z020

The top-right pane shows the 'Variables' window with the following data:

Name	Type	Value
FFT_output	complex float [8]	[385.0000+379.0000i, 62.0000+47.0000i, ...]
FFT_rev	complex float [8]	[11.00000+23.00000i, 47.00000+96.00000i, ...]
[0]	complex float	11.00000+23.00000i
[1]	complex float	47.00000+96.00000i

The bottom-left pane shows the source code for helloworld.c:

```
// 1) Reorder of the inputs to get output in the normal order
// 2) Multiplications using multi-stage butterfly approach
InputReorder(FFT_input, FFT_rev); // Task 1
FFTStages(FFT_rev, FFT_output); // Task 2

// Print the FFT output on the UART
printf("\n FFT output: \r\n");
for (int i = 0 ; i < FFT_Size ; i++)
{
    printf("%f %f\n", crealf(FFT_output[i]), cimagf(FFT_output[i]));
}

// Modify this code for large size FFT
// How you can generalize the code for any FFT size (limited to power of two)
```

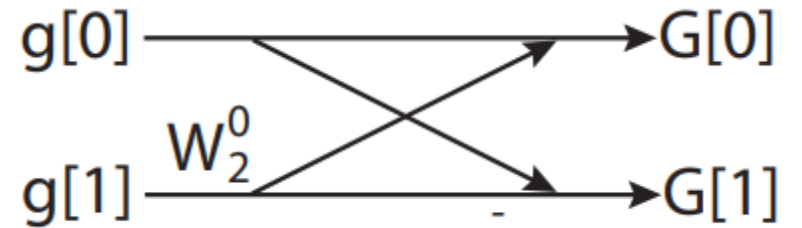
The bottom-right pane shows the 'Outline' window with the following assembly code:

```
122
123 // Print the FFT output
00100bec: sub    r2, r11, #140
00100bf0: sub    r3, r11, #204
00100bf4: mov    r1, r2
00100bf8: mov    r0, r3
00100bfc: bl     -1532 ; address
124 printf("\n FFT output
00100c00: movw   r0, #41240
00100c04: movt   r0, #16
00100c08: blx    +1888 ; address
125 for (int i = 0 ; i <
```

Large Size Fast Fourier Transform

Fast Fourier Transform

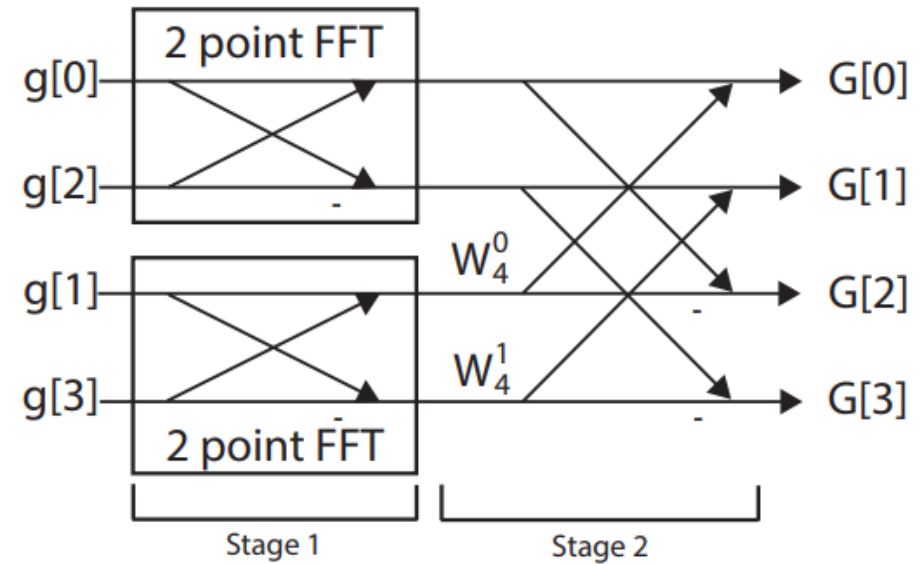
- Reduces the complexity of the DFT
- 2-point FFT: $S = \begin{bmatrix} W_2^{00} & W_2^{01} \\ W_2^{10} & W_2^{11} \end{bmatrix}$ where $W = e^{-j2\pi}$
- For inputs, $g[0]$ and $g[1]$, outputs are $G[0] = g[0] + g[1]$ and $G[1] = g[0] - g[1]$



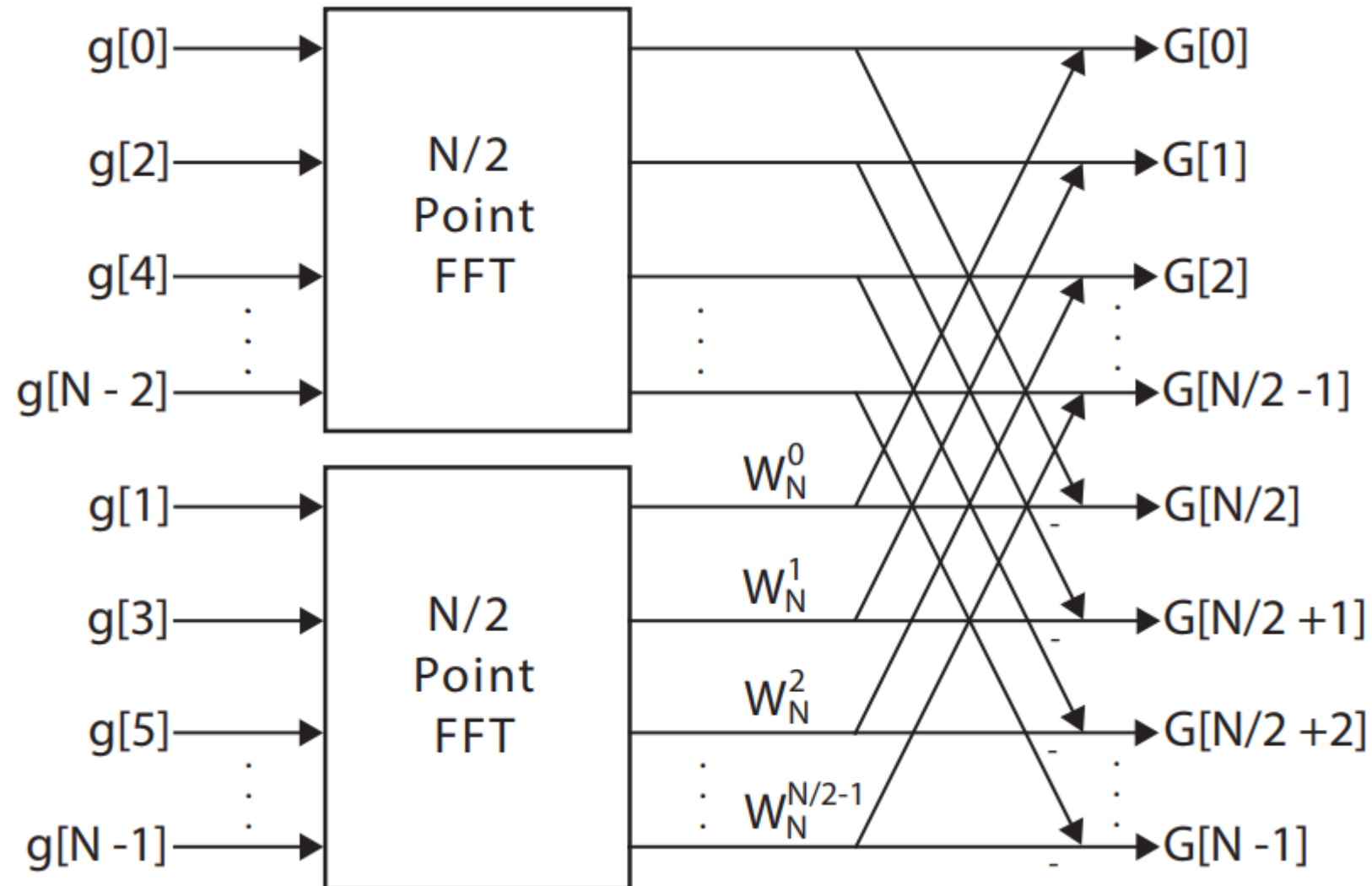
Fast Fourier Transform

$$\begin{aligned}
 G[0] &= (g[0] + g[2]) + e^{\frac{-j2\pi 0}{4}} (g[1] + g[3]) \\
 G[1] &= (g[0] - g[2]) + e^{\frac{-j2\pi 1}{4}} (g[1] - g[3]) \\
 G[2] &= (g[0] + g[2]) + e^{\frac{-j2\pi 2}{4}} (g[1] + g[3]) \\
 G[3] &= (g[0] - g[2]) + e^{\frac{-j2\pi 3}{4}} (g[1] - g[3])
 \end{aligned}$$

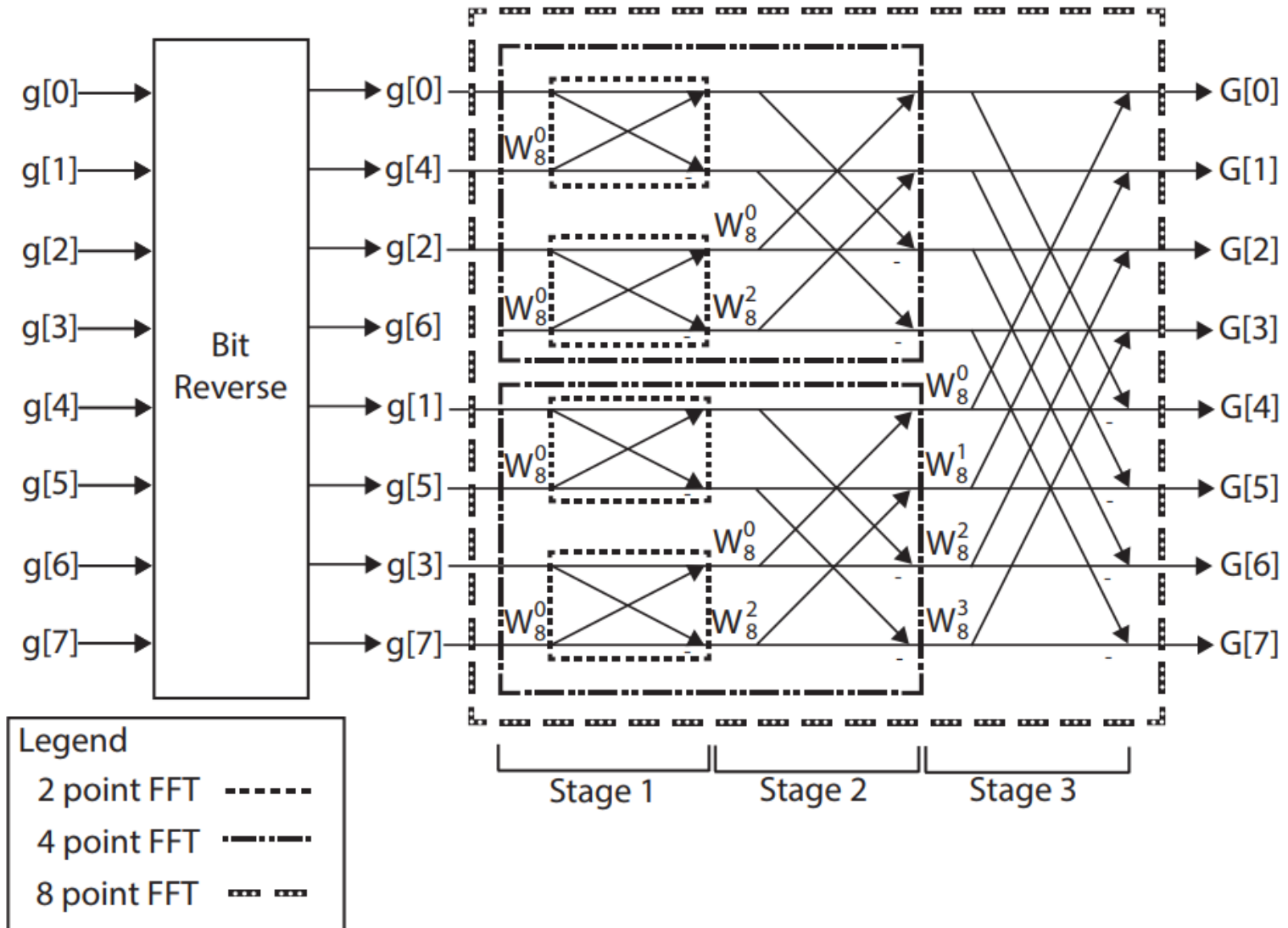
$$\begin{aligned}
 G[0] &= (g[0] + g[2]) + e^{\frac{-j2\pi 0}{4}} (g[1] + g[3]) \\
 G[1] &= (g[0] - g[2]) + e^{\frac{-j2\pi 1}{4}} (g[1] - g[3]) \\
 G[2] &= (g[0] + g[2]) - e^{\frac{-j2\pi 0}{4}} (g[1] + g[3]) \\
 G[3] &= (g[0] - g[2]) - e^{\frac{-j2\pi 1}{4}} (g[1] - g[3])
 \end{aligned}$$



Fast Fourier Transform

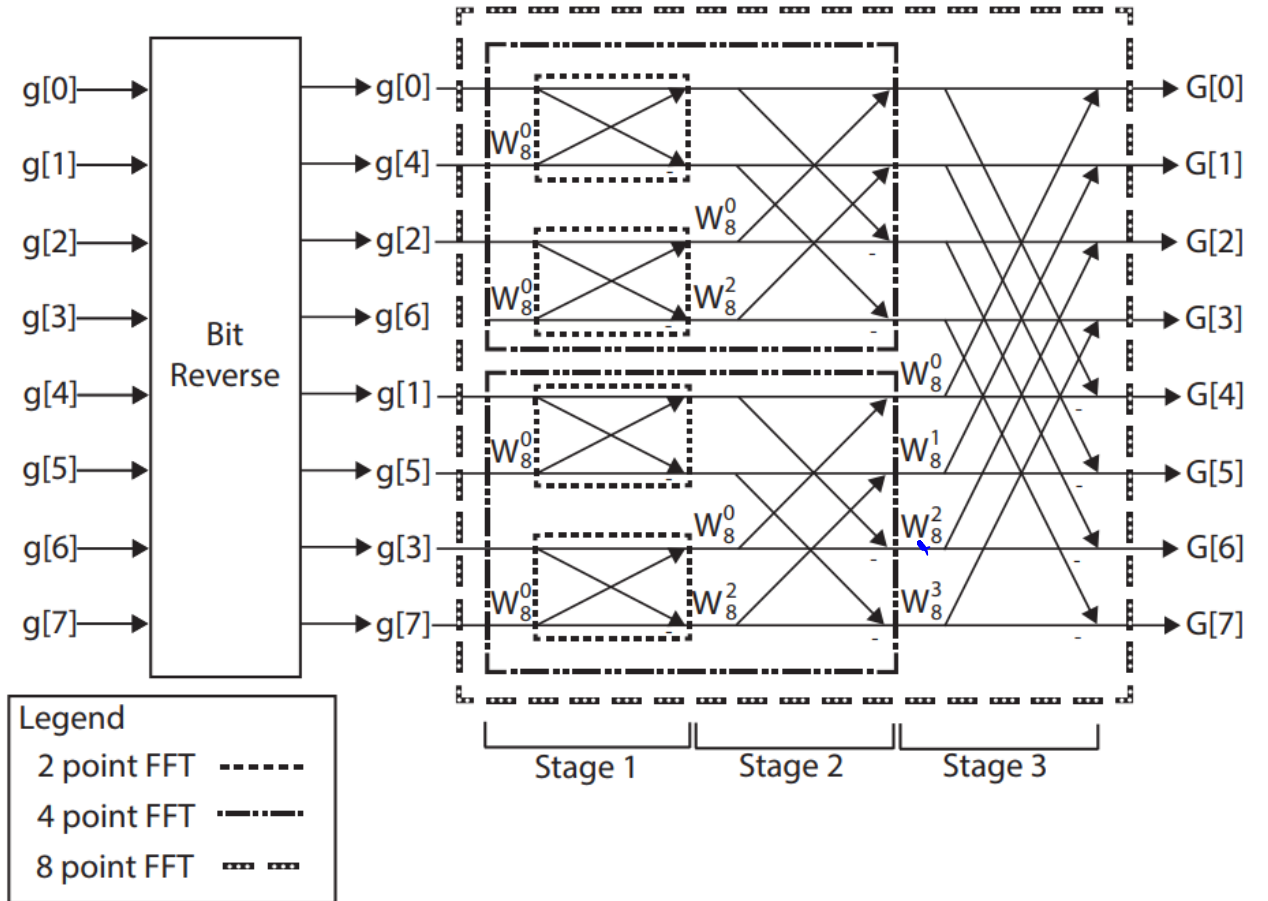


Fast Fourier Transform



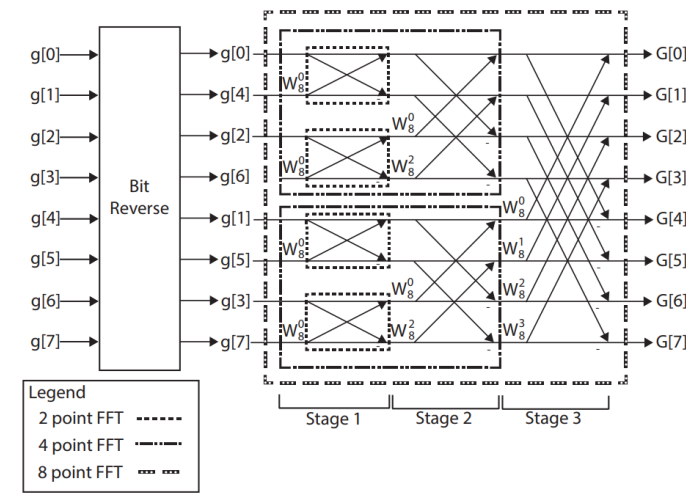
Fast Fourier Transform: Bit Reverse

Index	Binary	Reversed Binary	Reversed Index
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7



Fast Fourier Transform

- Consider each stage separately
- In each stage, fixed number of weights are used for each butterfly. For stage 1, there is only one weight. For stage two, there are two weights....
- In a stage, each weight is being used for $(\text{FFT SIZE} / \text{No. of Distinct Weights of the stage})$ times. In stage 1, there is only weight and it is reused 4 times to get 8 outputs. In stage 2, there are 2 weights which are reused 2 times to get 8 outputs...
- We can consider three FOR loops: 1) Outer For loops: Number of stages, 2) Middle For loop: Number of weights in a stage, 3) Inner For loop: Number of times a weight is reused i.e. $(\text{FFT SIZE} / \text{No. of Distinct Weights of the stage})$



HomeWork

- Compare the execution time of FFT of sizes 512 and 1024
- Realize the following function on PS:

$$Q = \frac{X}{T} + \sqrt{\frac{2 * N}{T}}$$

- Assume X,T and N are vectors of size 3 and take these values from user. Show the output on UART along with execution time.
- **Analyze the execution times for different optimization settings and different vector sizes (say 5, 10, 15).**
- Do explore various arithmetic functions on your own. It is not possible to cover all aspects in lectures and labs. Similarly, you can explore SW debugging functionality for these functions.

Math Library

- If math.h is not available, add it using following steps:

