[SoC Design]

Lab 1: SW Design

Chester Sungchung Park (박성정)

SoC Design Lab, Konkuk University

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

Teaching Assistants

- Youngho Seo (younghoseo@konkuk.ac.kr), M.S. candidate
- Sanghun Lee (sanghunlee@konkuk.ac.kr), M.S. candidate

Outline

- Objectives

- SW design (DFT)
  - Creating projects
  - Adding math library
  - Running C applications
  - Debugging C applications

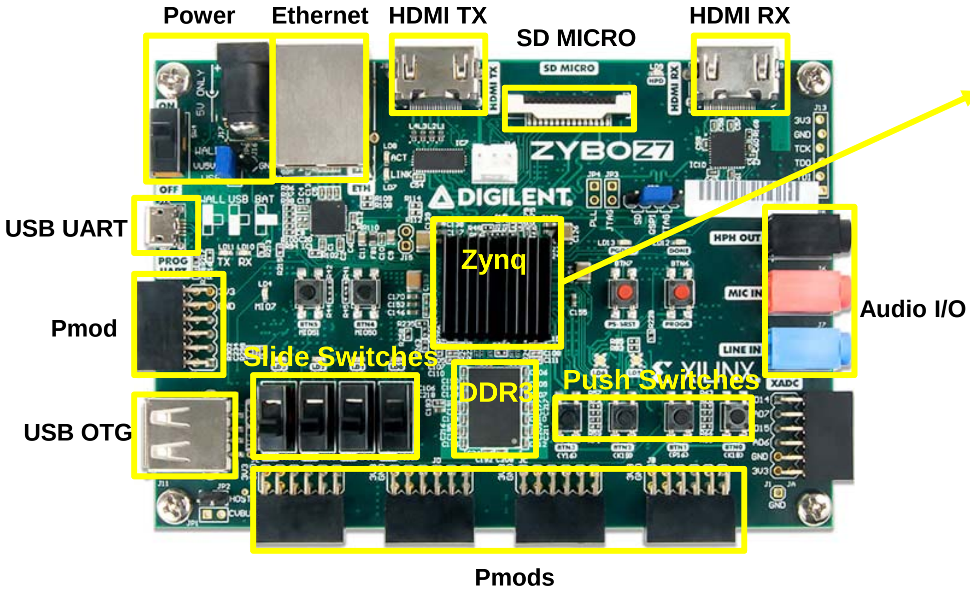
- SW optimization (FFT)
  - Programming in C
  - Programming in assembly
  - Setting optimization level

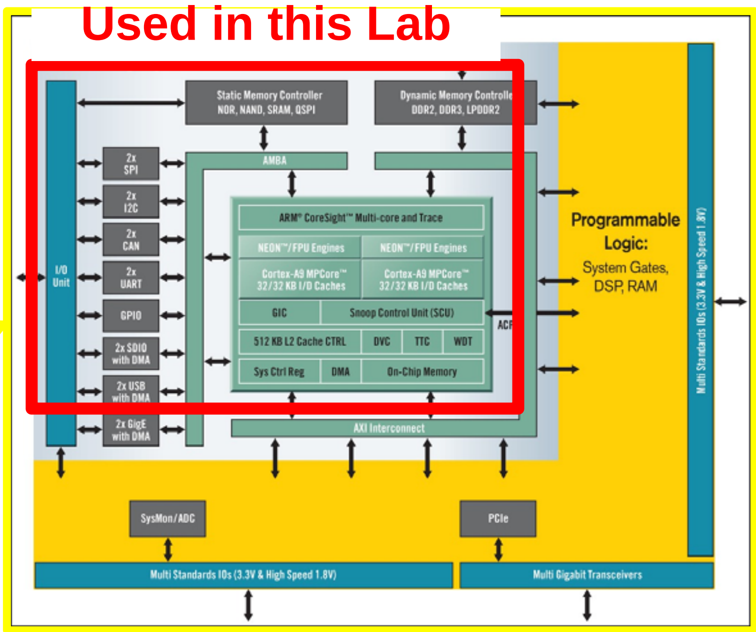
Objectives

- After completing this lab, you will be able to :
  - Program an application in either C or assembly
  - Run an application
  - Debug an application
  - Measure the execution time of an application
  - Optimize the performance of an application in either C or assembly

Introduction

- ZYNQ





Introduction

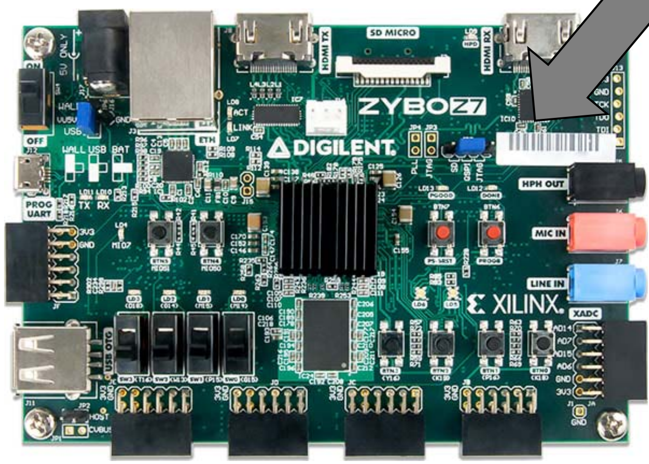
- ❑ Design flow

Used in this Lab

Vivado

SDK

ZYNQ/Zybo



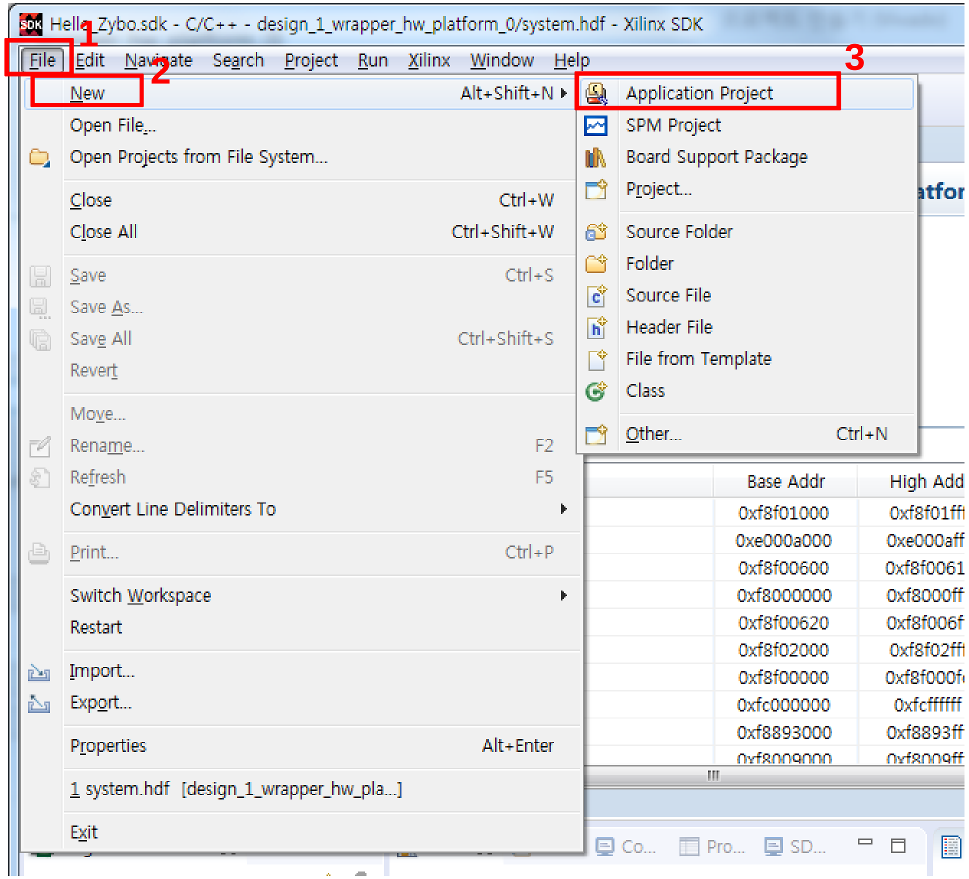
S/W Design (DFT)

Creating Projects

- Repeat the previous steps
  - Follow pp. 6~26 of the following lab workbook: **Lab_SD2019_0w.pdf**

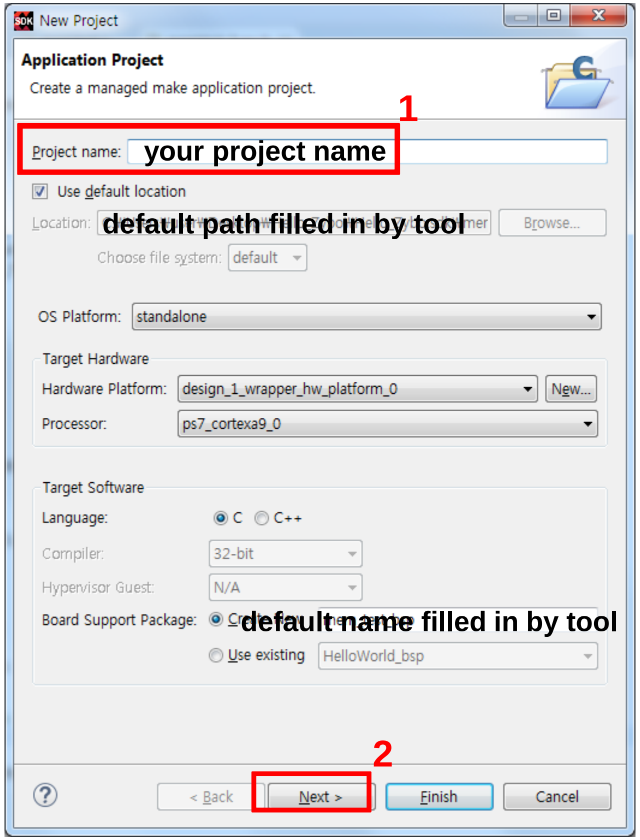
Running C Applications

- Create a C application project
  - Click ***'File' > 'New' > 'Application Project'***



Running C Applications

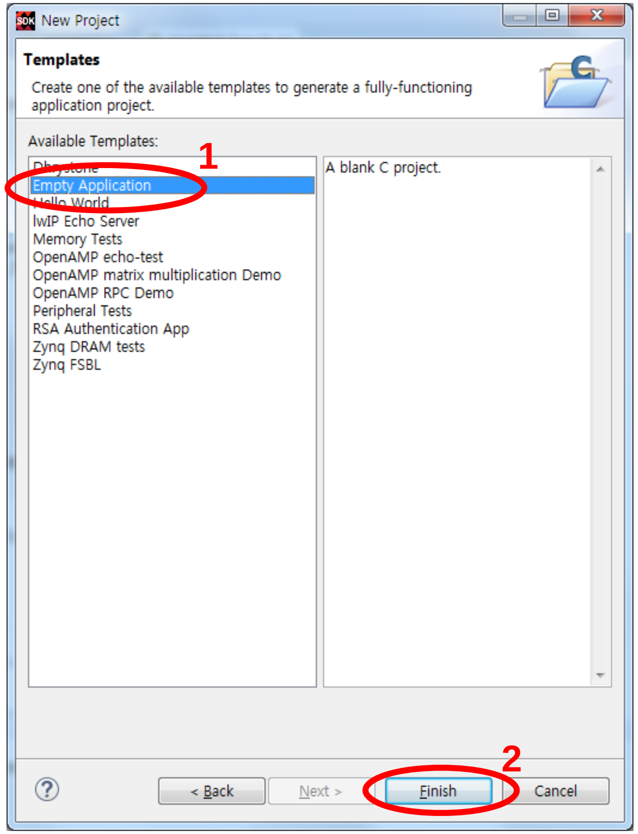
- Create a C application project (cont'd)
  - Type ***<your project name>*** in the Project name field
  - The ***'Board Support Package'*** field can be set up to use an existing BSP or a new BSP can be created based on the project name. (Do not modify)



Running C Applications

❑ Create a C application project (cont'd)

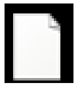
- Select ***'Empty Application'*** from the Template list
- Click ***'Finish'***



Creating Projects

❑ Add source files

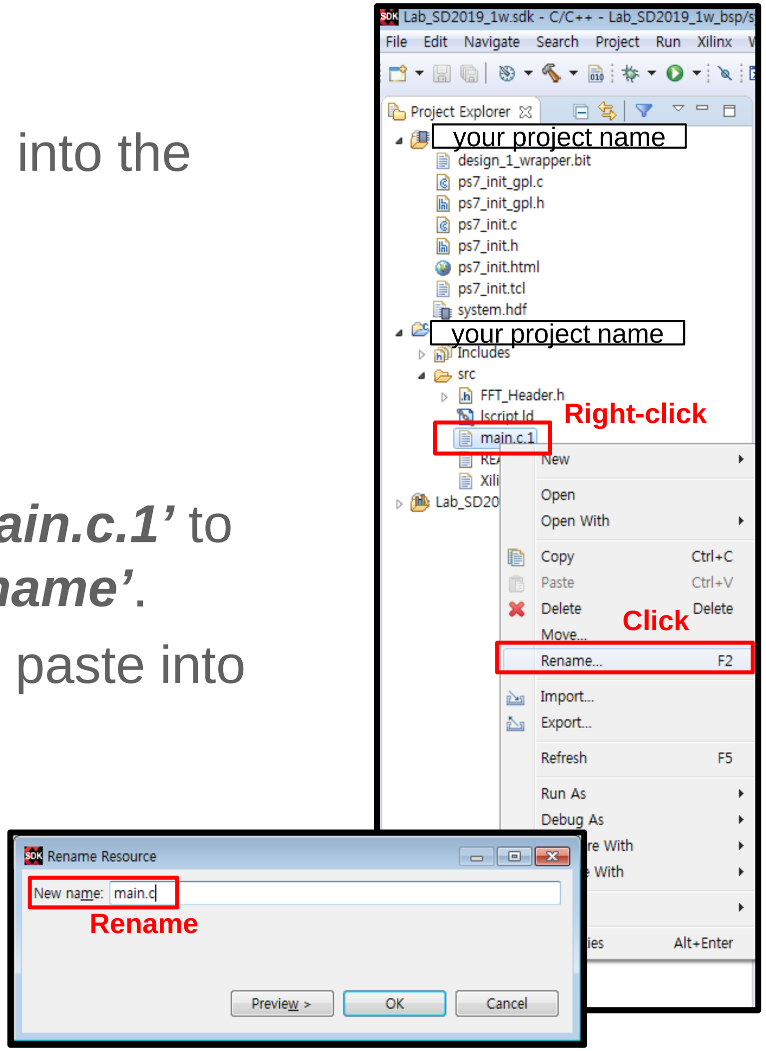
- Copy ***'main.c.1'*** and paste into the ***'src'*** folder.



main.c.1

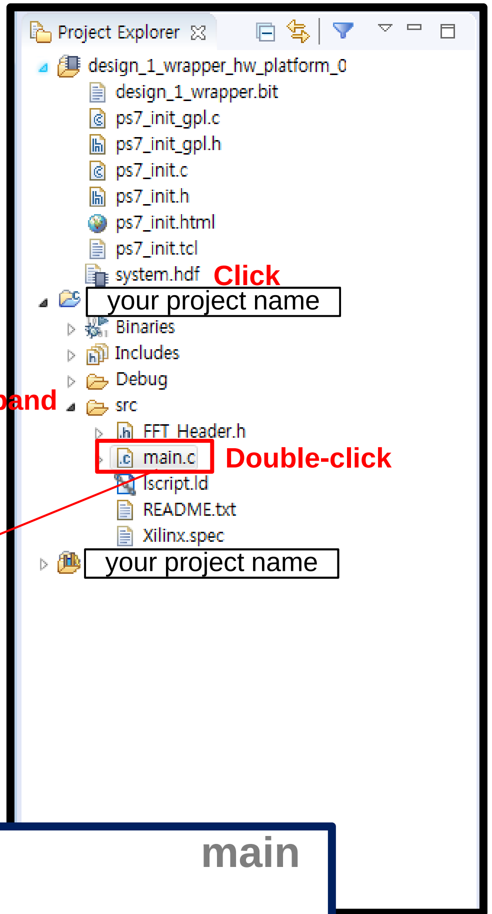
- Change the name from ***'main.c.1'*** to ***'main.c'*** after clicking ***'Rename'***.
- Copy ***'FFT_Header.h'*** and paste into the ***'src'*** folder

FFT_Header.h



Creating Projects

- Check the C application program
  - Expand ***'your project name > src'*** to see all of the source files included in the project by clicking the arrow next to ***'src'***.
  - Double-click the ***'main.c'*** file to open it



Function

```c
int Re_ordering(int x) {
    return 32 * (x % 2) + 16 * ((x % 4) / 2) + 8 * ((x % 8) / 4)
            + 4 * ((x % 16) / 8) + 2 * ((x % 32) / 16) + x / 32;
}

void DFT()
{
    int n = 0, i = 0 ,k = 0;

    complex input[N];
    complex temp_mult[N];

    int out_re[N] = {0,};
    int out_im[N] = {0,};

    for (n = 0; n<N; n++)
    {
        input[n].re=in_real[n];
        input[n].im=in_imag[n];
    }

    for (i = 0; i<N; i++)
    {
        X_DFT[i] = add_cal(init1_int,init2_int);
        for (k = 0; k<N; k++)
        {
            temp_mult[k] = multiple(input[k],W[(k*i)%64]);
            X_DFT[i] = add_cal(temp_mult[k],X_DFT[i]);
        }
    }

    for (n = 0; n<N; n++)
    {
        out_re[n] = X_DFT[n].re;
        out_im[n] = X_DFT[n].im;
    }

}
```

Header files & global variables

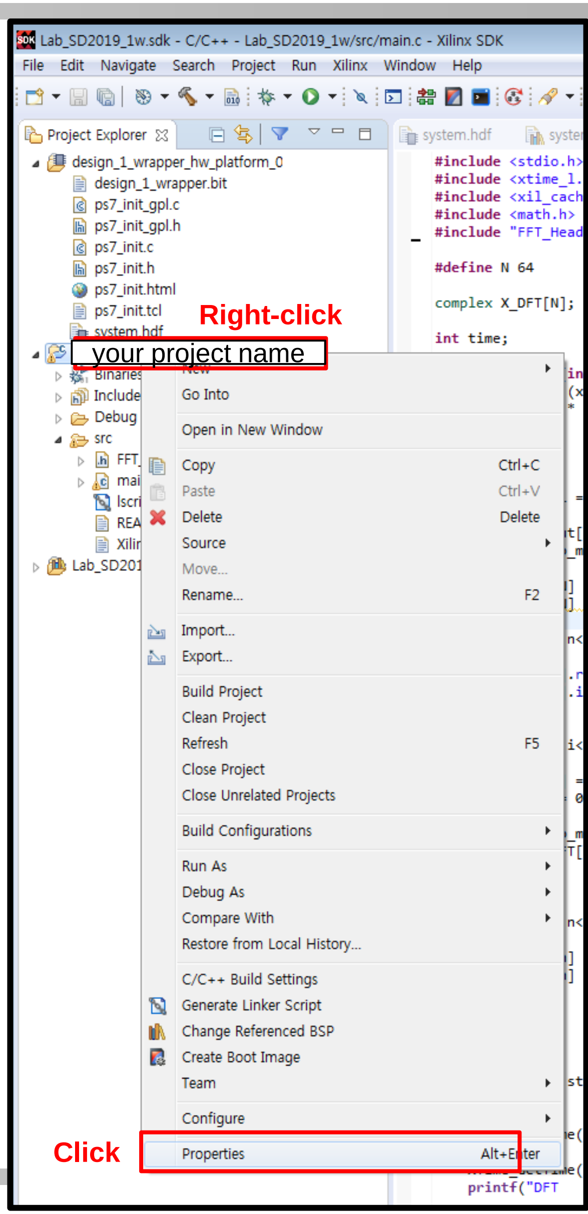
```c
#include <stdio.h>
#include <xtime_l.h>
#include <xil_cache.h>
#include <math.h>
#include "FFT_Header.h"

#define N 64

complex X_DFT[N];

int time;
```

main

```c
int main() {
    XTime start,stop;
    int i = 0;

    XTime_GetTime((XTime*)&start);
    DFT();
    XTime_GetTime((XTime*)&stop);
    printf("DFT          %8.3f us\n",((float)stop - (float)start)/COUNTS_PER_SECOND*1000000);

    return 0;
}
```

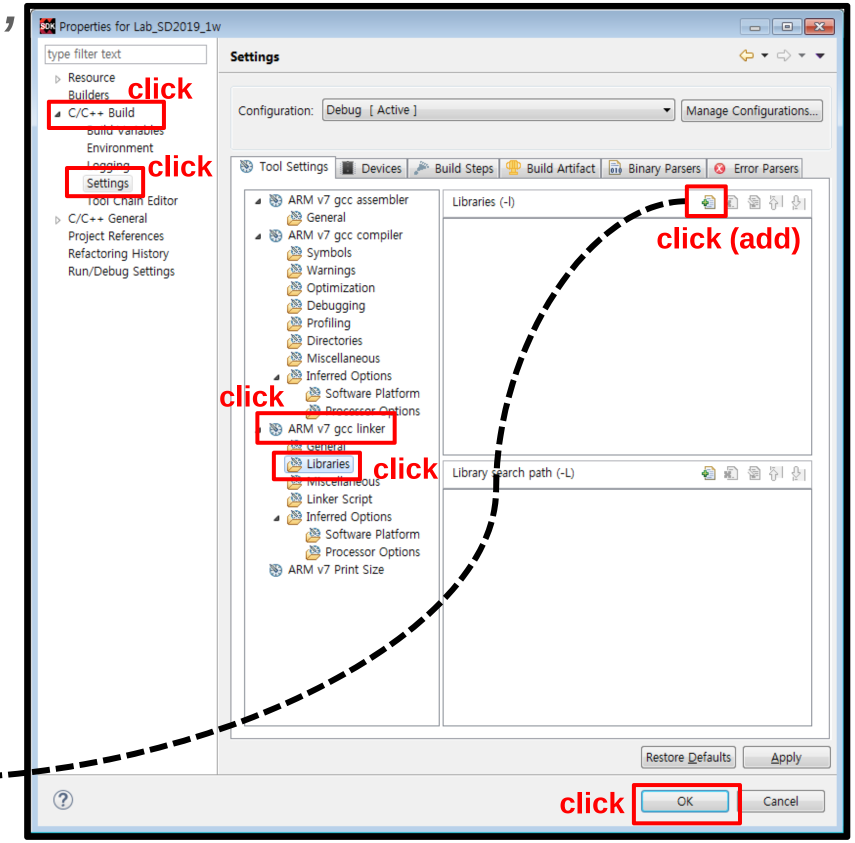
Adding Math Library

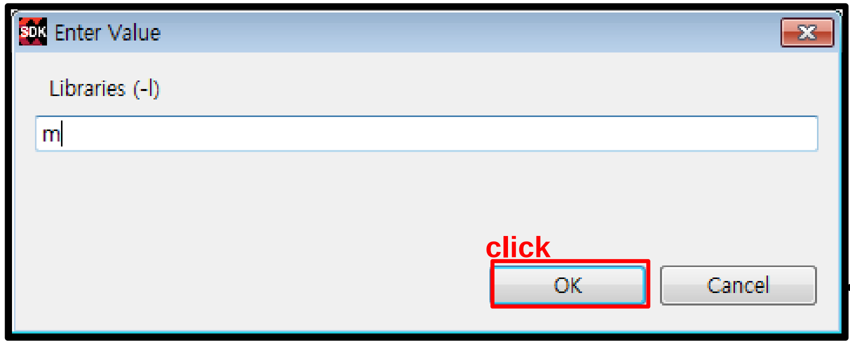
- Add math library to linker
  - Choose ***'your project name' > Properties***



Adding Math Library

- Add math library to linker (cont'd)
  - Click ***'C/C++ Build*** > ***Settings*** > ***ARM v7 gcc linker*** > ***Libraries*** > ***add'***
  - Add the value ***'m'*** to ***'Libraries'***
  - Click **'*OK*'.**





Running C Applications

- Review the function: ***'main()'***
  - ① Measures the start time (***'start'***)
  - ② Calls '***DFT()***'
  - ③ Measures the stop time (***'stop'***)
  - ④ Prints the execution time

```
int main() {
    XTime start,stop;
    int i = 0;

 ①  XTime_GetTime((XTime*)&start);
 ②  DFT();
 ③  XTime_GetTime((XTime*)&stop);
 ④  printf("DFT       %8.3f us\n",((float)stop - (float)start)/COUNTS_PER_SECOND*1000000);

    return 0;
}
```

Running C Applications

- ❑ Review the function ***'DFT()'***
  - ① Takes the input (from ***'header'***)
  - ② Performs DFT
  - ③ Generates the output

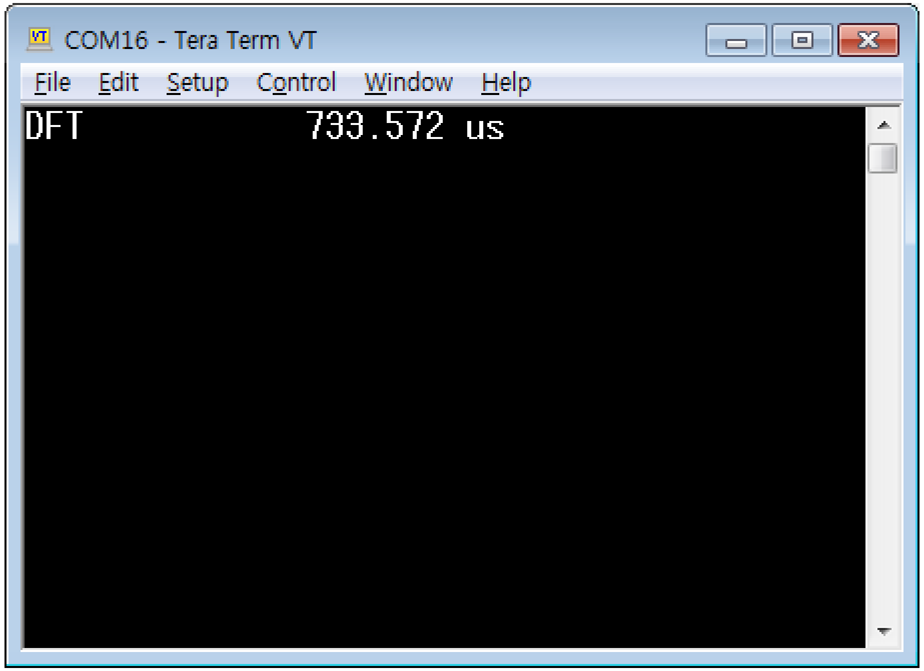
```c
void DFT()
{
    int n = 0, i = 0 ,k = 0;

    complex input[N];
    complex temp_mult[N];

    int out_re[N] = {0,};
    int out_im[N] = {0,};

    for (n=0; n<N; n++)                                          ①
    {
        input[n].re=in_real[n];
        input[n].im=in_imag[n];
    }

    for (i=0; i<N; i++)                                          ②
    {
        X_DFT[i] = add_cal(init1_int,init2_int);
        for (k=0; k<N; k++)
        {
            temp_mult[k] = multiple(input[k],W[(k*i)%64]);
            X_DFT[i] = add_cal(temp_mult[k],X_DFT[i]);
        }
    }

    for (n=0; n<N; n++)                                          ③
    {
        out_re[n] = X_DFT[n].re;
        out_im[n] = X_DFT[n].im;
    }

}
```

Running C Applications

- Repeat the previous steps
  - Follow pp. 31~34 of the following lab workbook: **Lab_SD2019_0w.pdf**

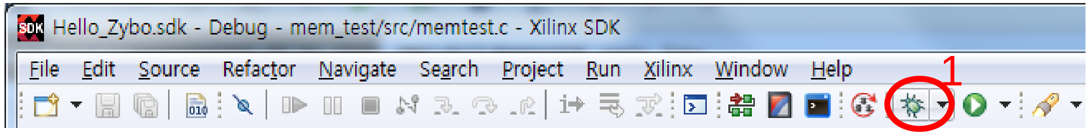
Running C Applications

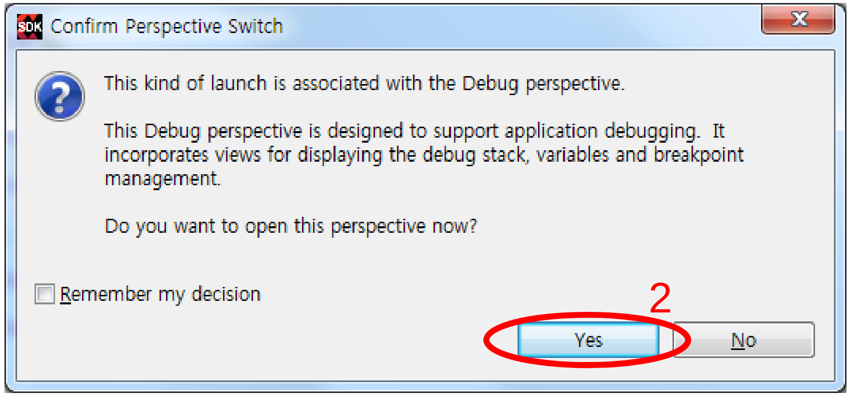
- ❑ Run the application Check the output of the application on ***'Tera Term'***
  - ✓ You should see the execution time as shown below.



Debugging C Applications

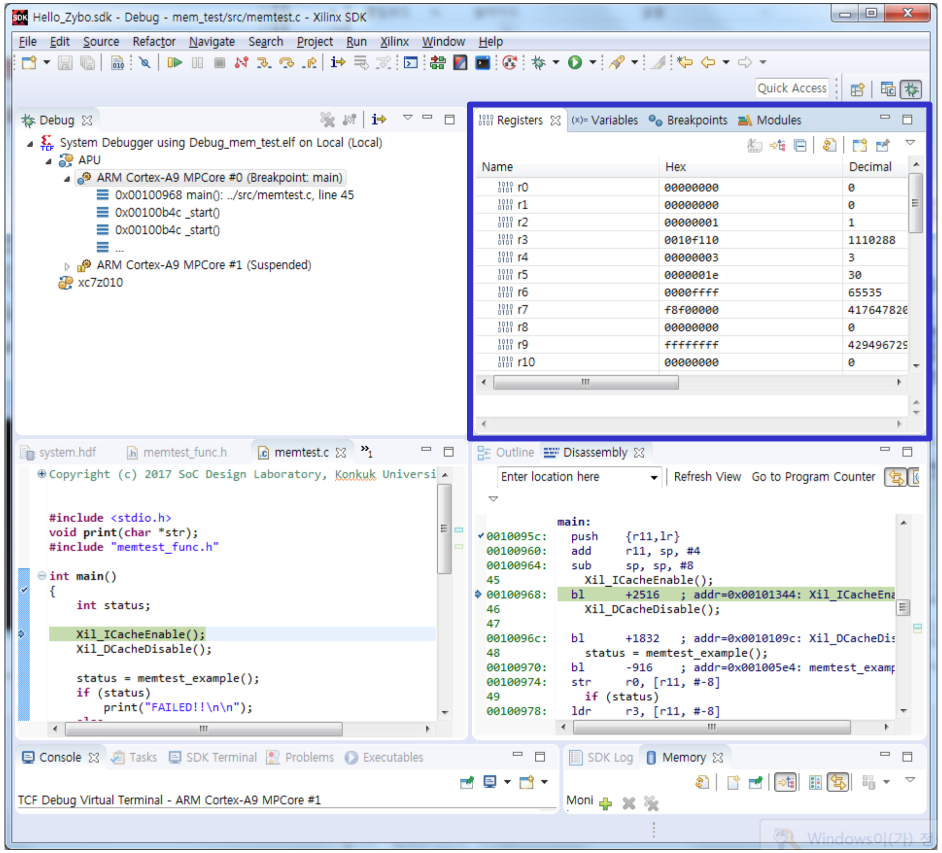
- Debug the application
  - Click the ***'Debug System Debugger'*** icon and then click ***'Yes'***





Debugging C Applications

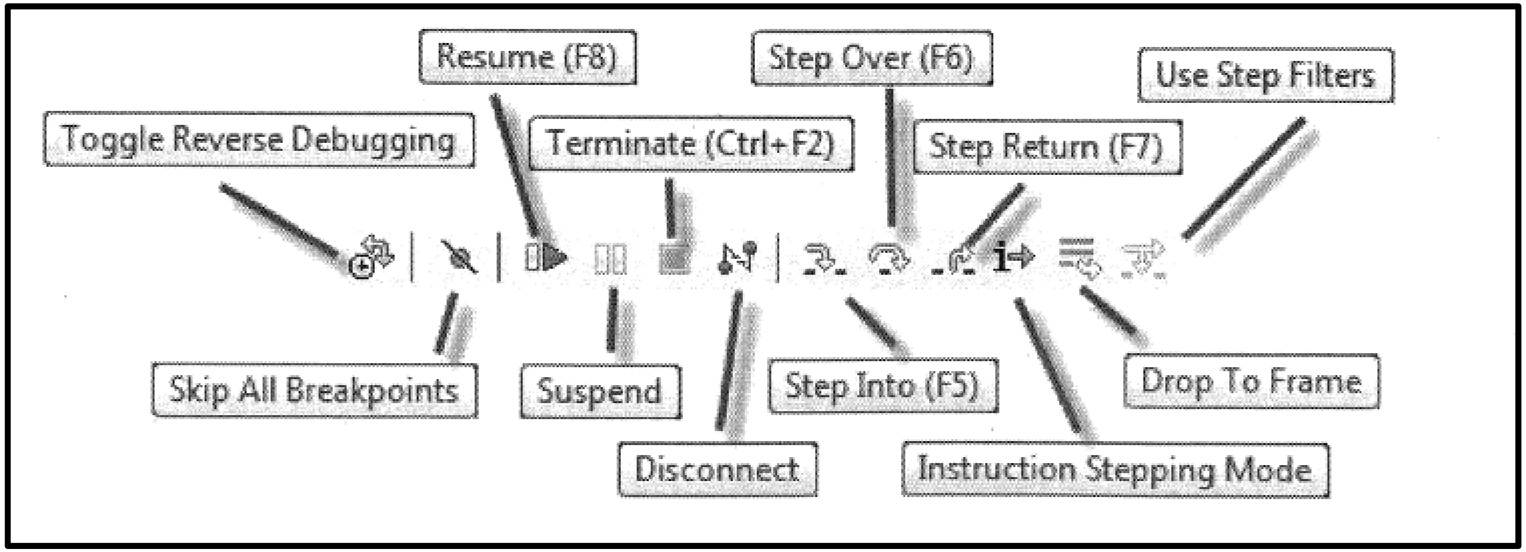
- ❑ Debug the application (cont'd)
  - Choose the ***'Registers'*** or ***'Variable'*** tab



Debugging C Applications

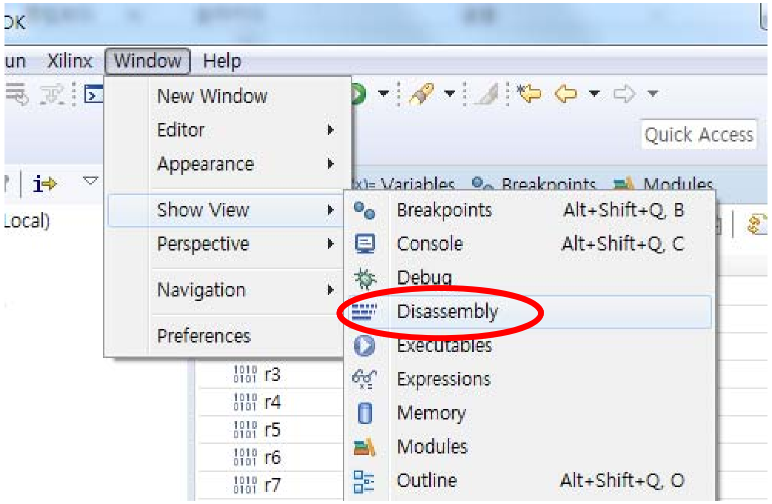
- ❑ Debug the application (cont'd)

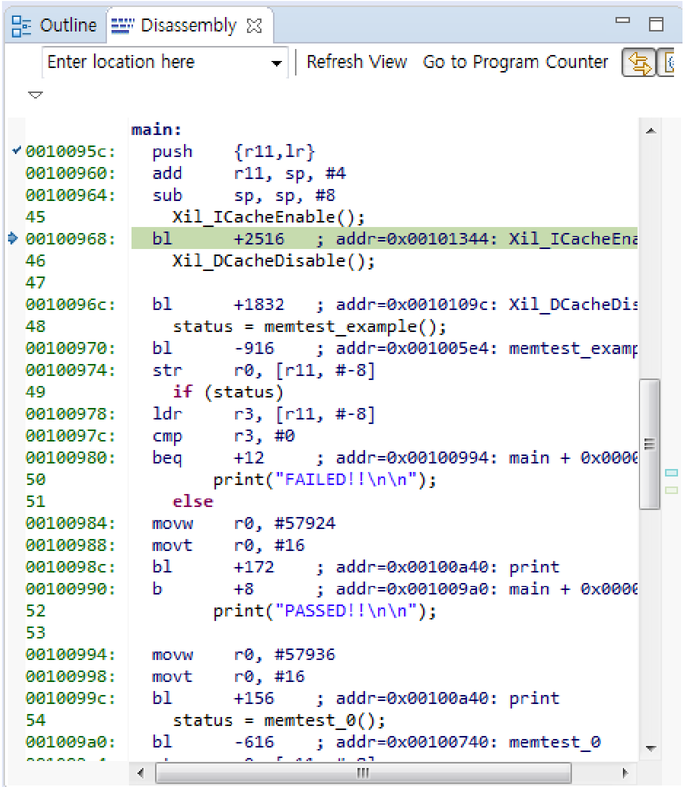
Debug Tool Bar



Debugging C Applications

- ❑ Review the disassembly
  - Open the ***'Window' > 'Show View'*** menu and then click ***'Disassembly'***

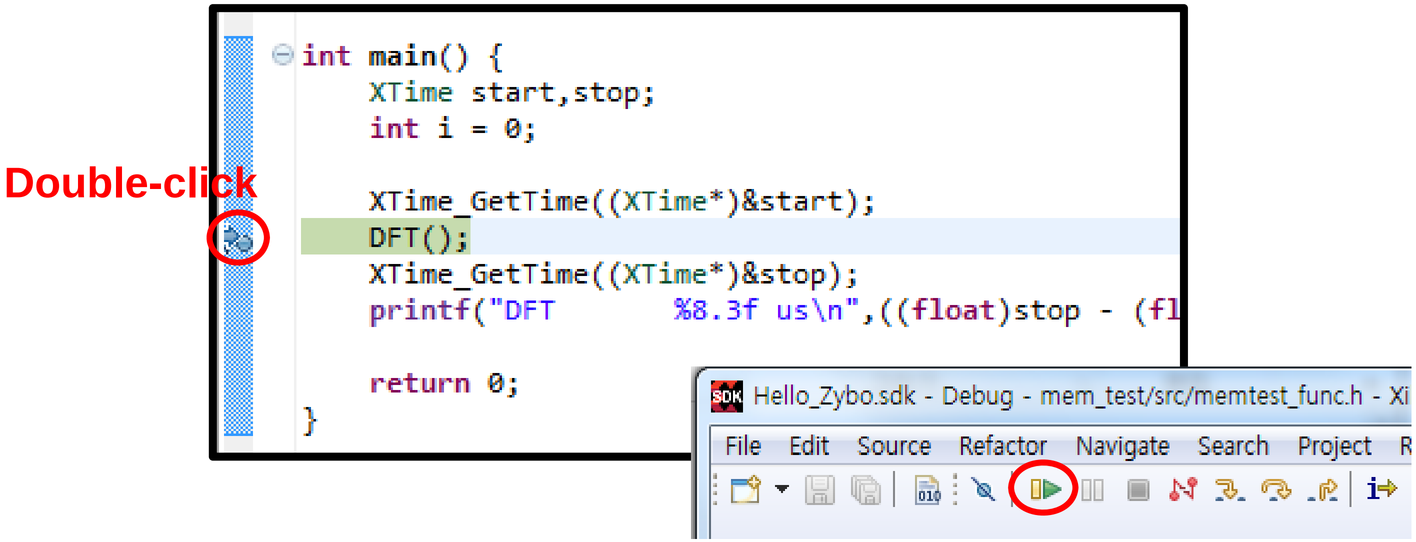




Debugging C Applications

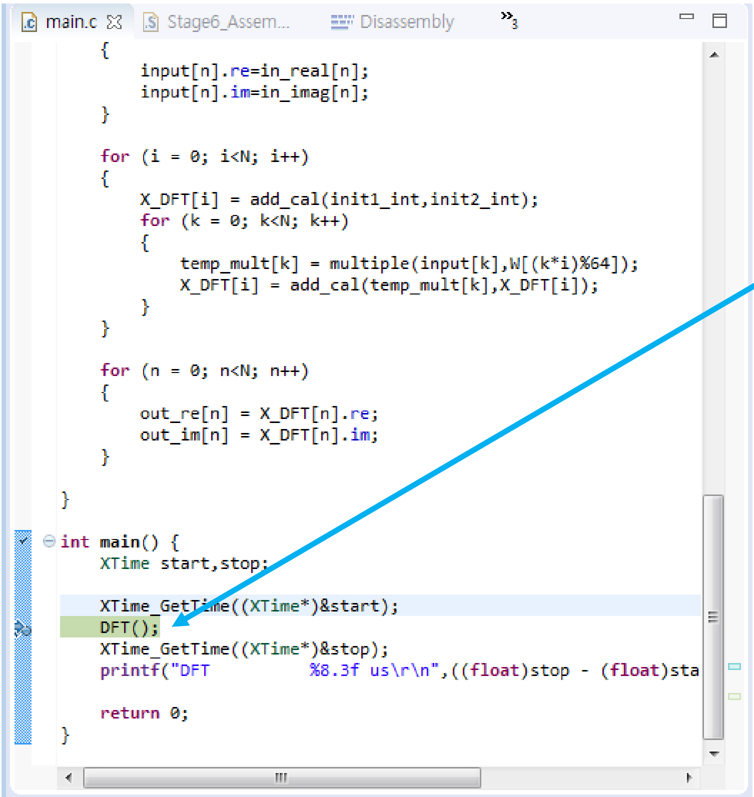
❑ Review the disassembly (cont'd)

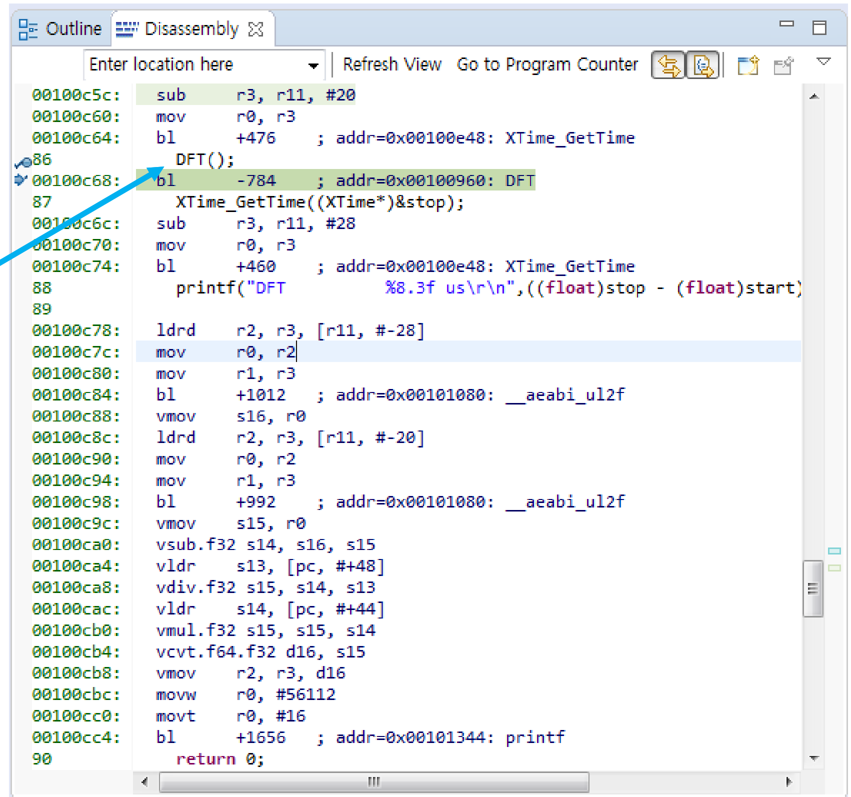
- Double-click on the left side of a code line to add a breakpoint
- Click the ***'Resume'*** icon to continue debugging



Debugging C Applications

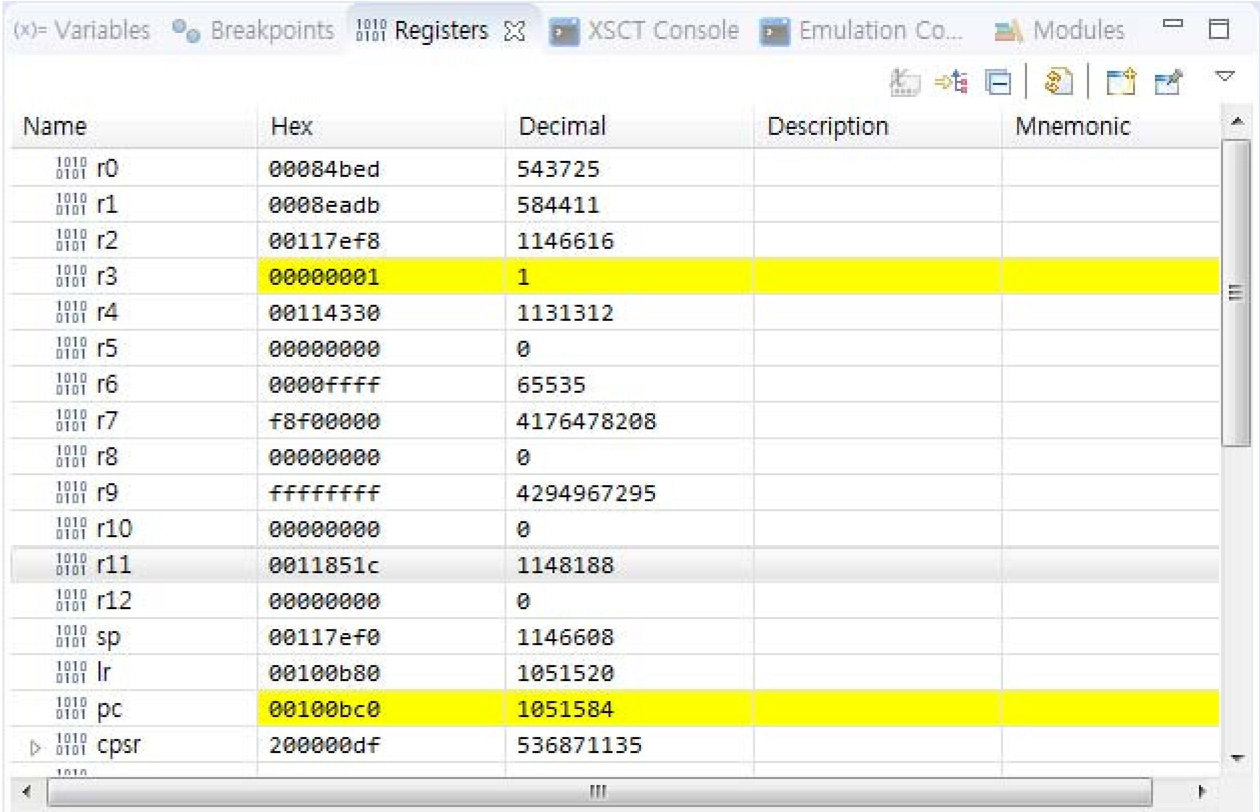
❑ Review the disassembly (cont'd)





Debugging C Applications

- ❑ Check the content of a register

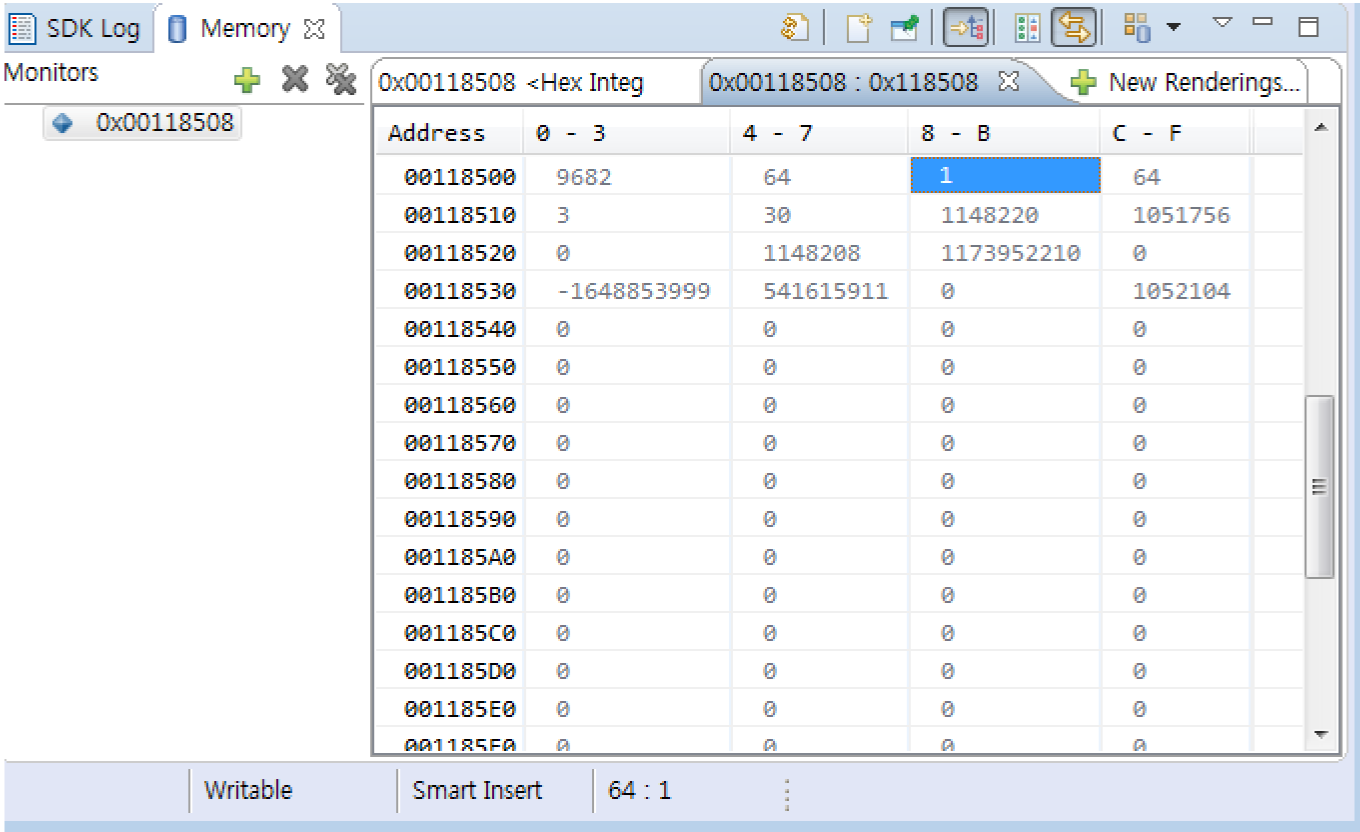


Name	Hex	Decimal	Description	Mnemonic
r0	00084bed	543725		
r1	0008eadb	584411		
r2	00117ef8	1146616		
r3	00000001	1		
r4	00114330	1131312		
r5	00000000	0		
r6	0000ffff	65535		
r7	f8f00000	4176478208		
r8	00000000	0		
r9	ffffffff	4294967295		
r10	00000000	0		
r11	0011851c	1148188		
r12	00000000	0		
sp	00117ef0	1146608		
lr	00100b80	1051520		
pc	00100bc0	1051584		
cpsr	200000df	536871135		

Debugging C Applications

❑ Check the content of a memory location

- Location for the loop variable (i)

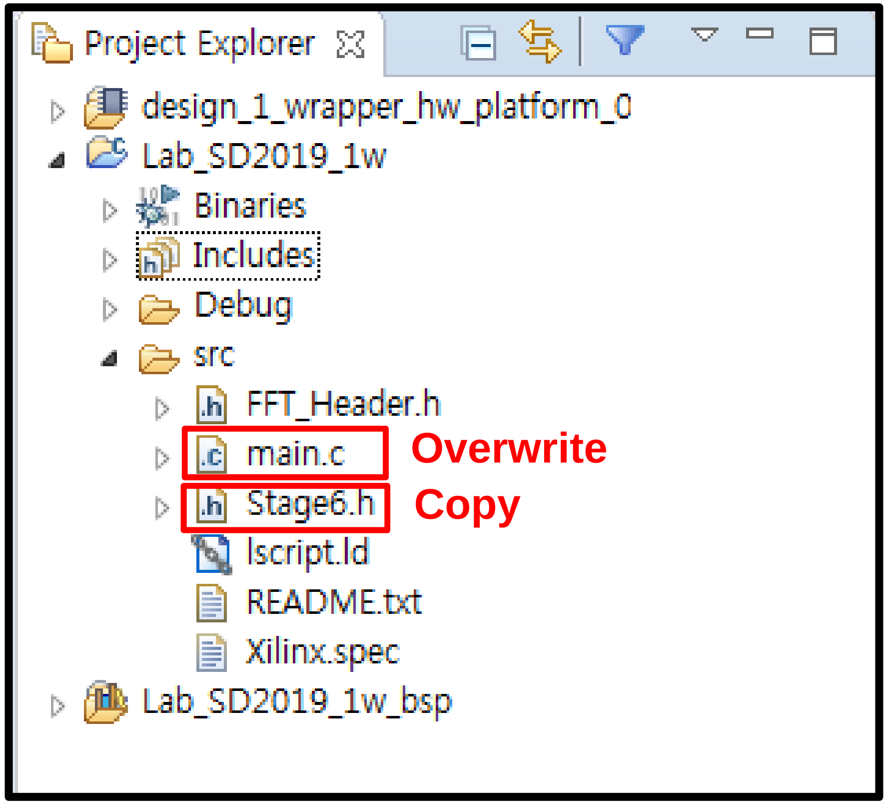


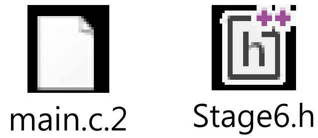
Address	0 - 3	4 - 7	8 - B	C - F
00118500	9682	64	1	64
00118510	3	30	1148220	1051756
00118520	0	1148208	1173952210	0
00118530	-1648853999	541615911	0	1052104
00118540	0	0	0	0
00118550	0	0	0	0
00118560	0	0	0	0
00118570	0	0	0	0
00118580	0	0	0	0
00118590	0	0	0	0
001185A0	0	0	0	0
001185B0	0	0	0	0
001185C0	0	0	0	0
001185D0	0	0	0	0
001185E0	0	0	0	0
001185F0	0	0	0	0

S/W Optimization (FFT)

Programming in C

- Copy the following files and paste them into the ***'src'*** folder.





Programming in C

- Review the function ***'main()'***
  - ① Calls '***FFT()***'
  - ② Compares the output of '***FFT()***' with that of '***DFT()***'

①

②

```c
int main() {
    XTime start,stop;
    int i = 0;

    float error_total, error_real, error_imag;
    float sig_total;
    float SNR;

    XTime_GetTime((XTime*)&start);
    DFT();
    XTime_GetTime((XTime*)&stop);
    printf("DFT        %8.3f us\n",((float)stop - (float)start)/COUNTS_PER_SECOND*1000000);

    XTime_GetTime((XTime*)&start);
    FFT();
    XTime_GetTime((XTime*)&stop);
    printf("FFT        %8.3f us\n",((float)stop - (float)start)/COUNTS_PER_SECOND*1000000);

    error_total = 0;
    sig_total = 0;
    for(i = 0; i<N; i++){
        error_real = (X_DFT[i].re)-(X_FFT[i].re);
        error_imag = (X_DFT[i].im) -(X_FFT[i].im);

        error_total += error_real*error_real + error_imag*error_imag;

        sig_total += (X_DFT[i].re)*(X_DFT[i].re) + (X_DFT[i].im)*(X_DFT[i].im);
    }
    SNR = 10*log10(sig_total/error_total);
    xil_printf("FFT model SNR : %d dB\n",(int)SNR);

    return 0;
}
```

Programming in C

- Review the function ***'FFT()'***
  - ① Butterfly: Stage 1 ~ Stage 5
  - ② Butterfly: Stage 6 (*incomplete*)
  - ③ Reordering

```c
#ifndef STAGE6_H_
#define STAGE6_H_


void Stage6(complex *output, complex *input)
{
    int n;

    /////////////////////////////////////////////
    //
    //
    //      Fill Your Code Here.
    //
    //
    /////////////////////////////////////////////
}



#endif /* STAGE6_H_ */
```

```c
void FFT()
{
    complex input[N],temp[N];

    int out_re[N];
    int out_im[N];

    int data;
    int n,k;
```

①

```c
    for (data=0;data<N;data++)
    {
        input[data].re=in_real[data];
        input[data].im=in_imag[data];
    }

    for (n=0;n<32;n++) //stage1
    {
        temp[n]=add_cal(input[n],input[n+32]);
        temp[n+32]=multiple(sub_cal(input[n],input[n+32]),W[n]);
    }

    for (n=0;n<16;n++) //stage2
    {
        for (k=0;k<2;k++)
        {
            input[n+(32*k)]=add_cal(temp[n+(32*k)],temp[n+((32*k)+16)]);
            input[n+((32*k)+16)]=multiple(sub_cal(temp[n+(32*k)],temp[n+((32*k)+16)]),W[2*n]);
        }
    }

    for(n=0;n<8;n++) //stage-3
    {
        for (k=0;k<4;k++)
        {
            temp[n+(16*k)] = add_cal(input[n+(16*k)],input[n+((16*k)+8)]);
            temp[n+((16*k)+8)] = multiple(sub_cal(input[n+(16*k)],input[n+((16*k)+8)]),W[4*n]);
        }
    }

    for (n=0;n<4;n++) //stage4
    {
        for (k=0;k<8;k++)
        {
            input[n+(8*k)] = add_cal(temp[n+(8*k)],temp[n+((8*k)+4)]);
            input[n+((8*k)+4)] =multiple(sub_cal(temp[n+(8*k)],temp[n+((8*k)+4)]),W[8*n]);
        }
    }

    for (n=0;n<2;n++) //stage5
    {
        for (k=0;k<16;k++)
        {
            temp[n+(4*k)]=add_cal(input[n+(4*k)],input[n+(4*k+2)]);
            temp[n+(4*k+2)]=multiple(sub_cal(input[n+(4*k)],input[n+(4*k+2)]),W[16*n]);
        }

    }
```

②

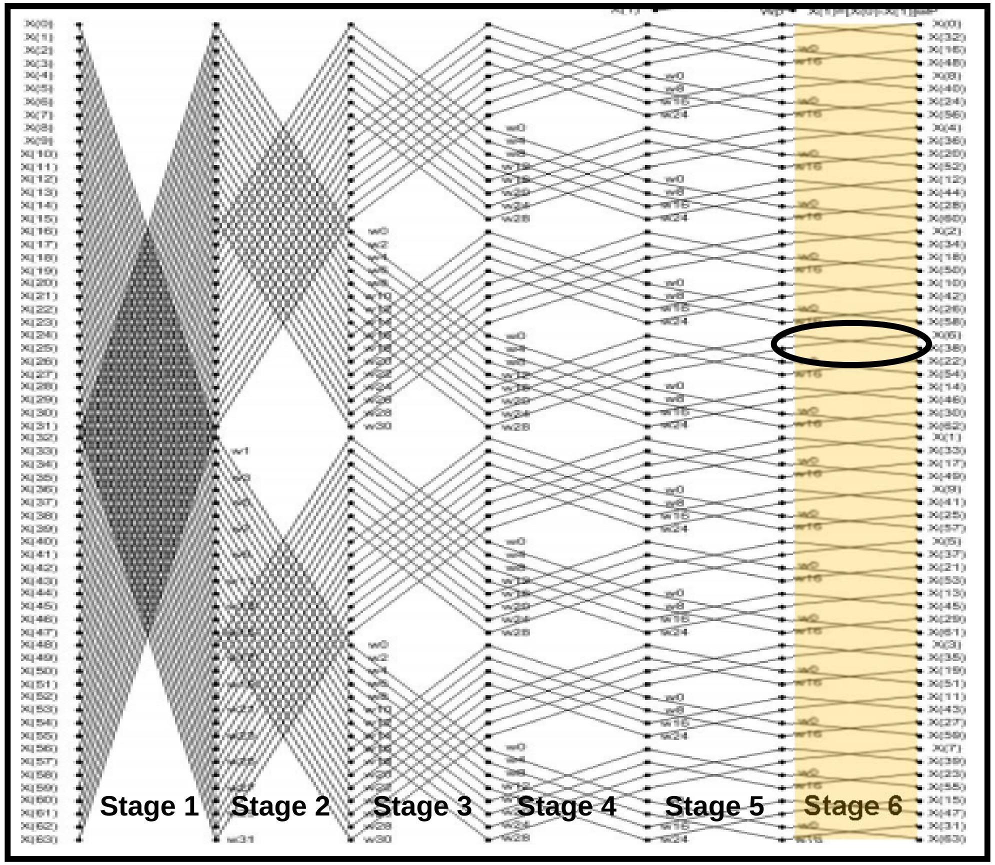
```c
    // Stage 6
    Stage6(input, temp);
```

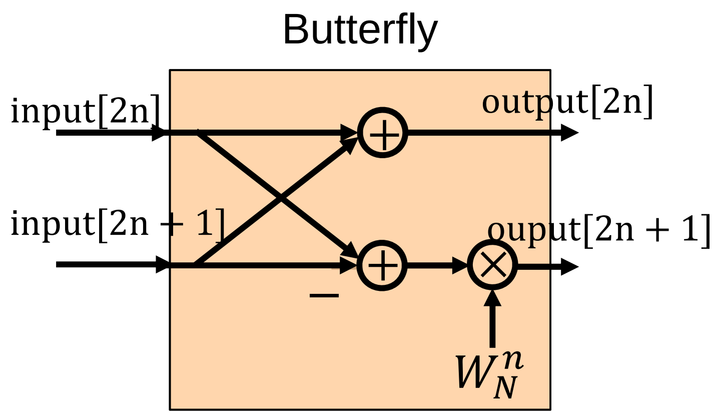
③

```c
    for(n=0;n<N;n++)
    {
        X_FFT[n]=input[Re_ordering(n)];
    }

    for (n=0;n<N;n++)
    {
        out_im[n]=(X_FFT[n].im)>>10;
        out_re[n]=(X_FFT[n].re)>>10;
    }
}
```

Programming in C

- ❑ Add lines to the ***'Fill Your Code Here'*** section in ***'Stage6.h'***

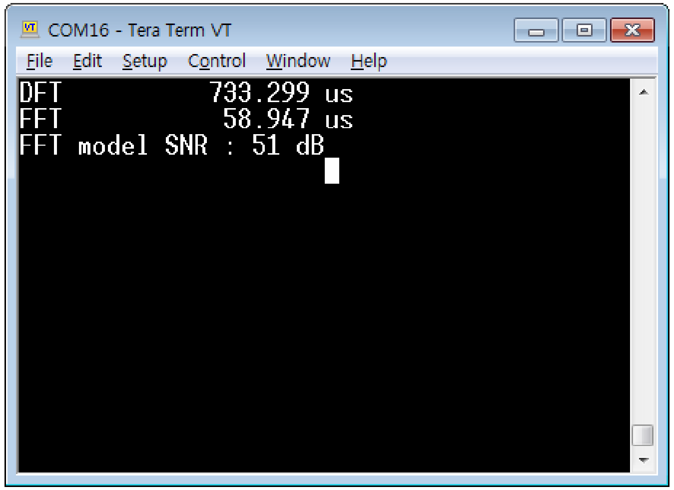




Programming in C

❑ Run the application

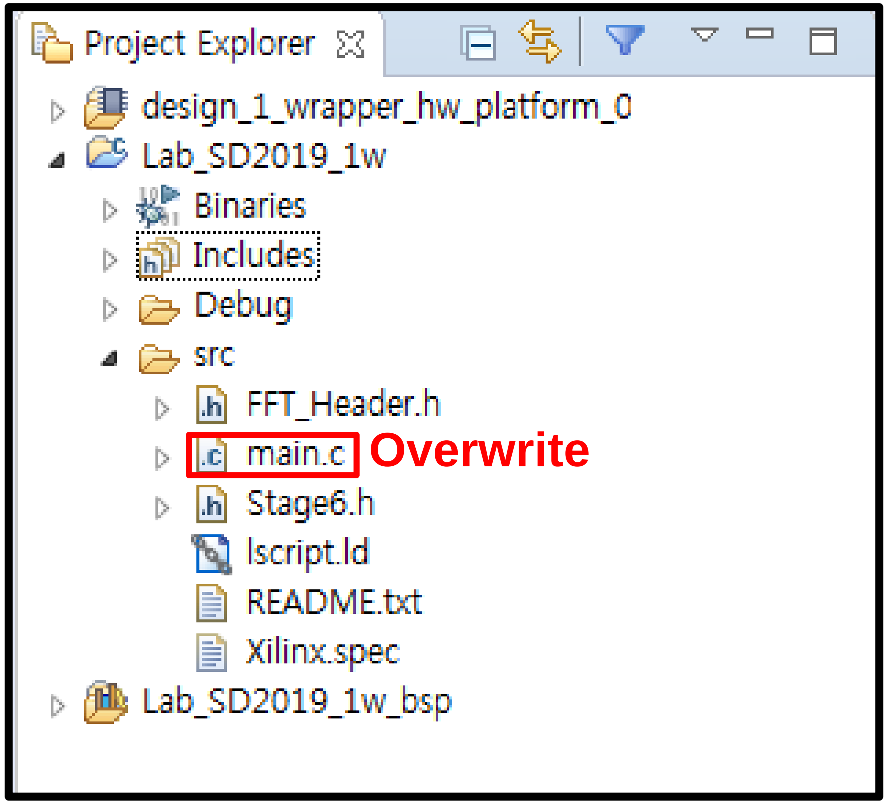
- Check the output of the application on ***'Tera Term'***
  - ✓ Check the performance gain.
  - ✓ Check the accuracy given by SNR

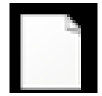


- Figure out the reason for the performance gain.

Programming in Assembly

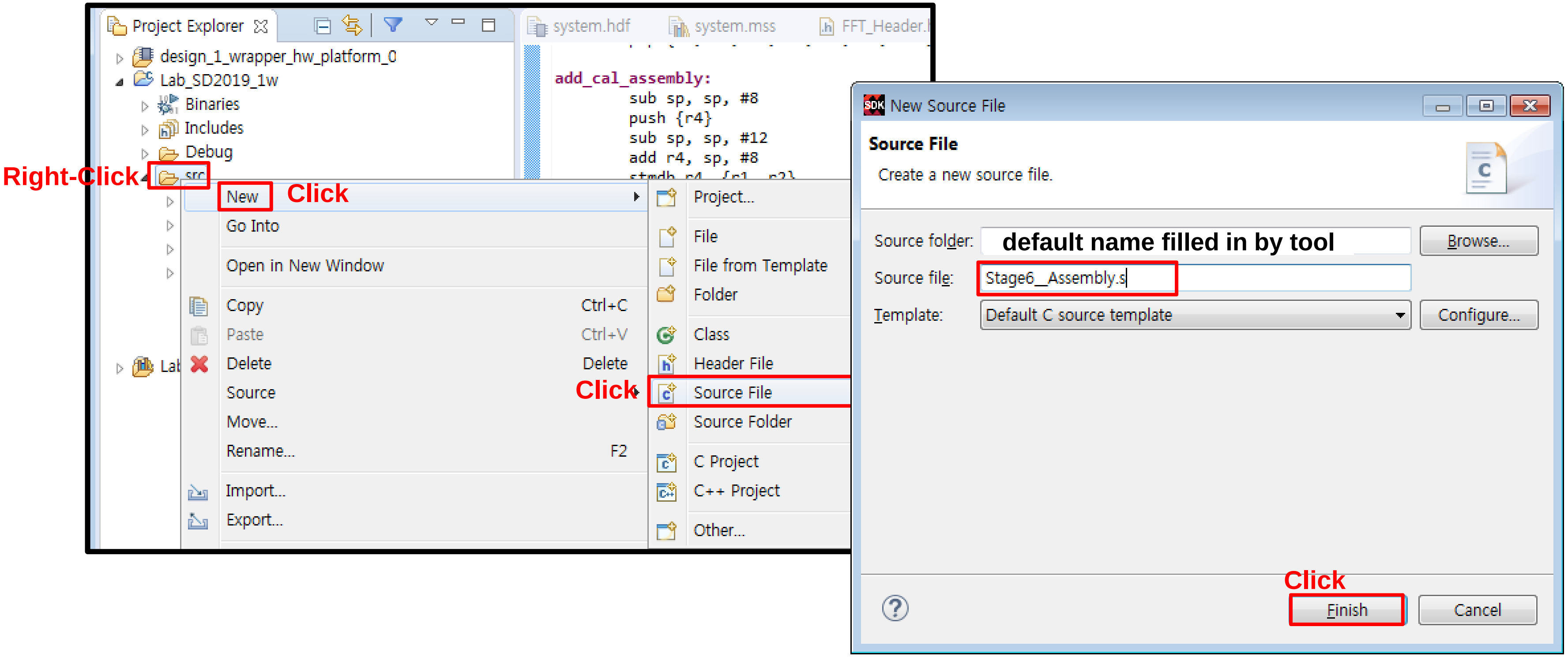
- ❑ Copy the following file and paste it into the ***'src'*** folder.





Programming in Assembly

- Add assembly source files
  - Click ***'src > New > Source File'***.
  - Type ***'Stage6_Assembly.s'*** (using file extension '***.s***')



Programming in Assembly

- Review the function ***'main()'***
  - ① Calls '***FFT_Assembly()***'
  - ② Compares the output of '***FFT()***' with that of '***DFT()***'

```c
int main() {
    XTime start,stop;
    int i = 0;

    float error_total, error_real, error_imag;
    float sig_total;
    float SNR;

    XTime_GetTime((XTime*)&start);
    DFT();
    XTime_GetTime((XTime*)&stop);
    printf("DFT          %8.3f us\n",((float)stop - (float)start)/COUNTS_PER_SECOND*1000000);

    XTime_GetTime((XTime*)&start);
    FFT();
    XTime_GetTime((XTime*)&stop);
    printf("FFT          %8.3f us\n",((float)stop - (float)start)/COUNTS_PER_SECOND*1000000);

    XTime_GetTime((XTime*)&start);
    FFT_Assembly();
    XTime_GetTime((XTime*)&stop);
    printf("FFT Assembly %8.3f us\n",((float)stop - (float)start)/COUNTS_PER_SECOND*1000000);


    error_total = 0;
    sig_total = 0;
    for(i = 0; i<N; i++){
        error_real = (X_DFT[i].re)-(X_FFT[i].re);
        error_imag = (X_DFT[i].im) -(X_FFT[i].im);

        error_total += error_real*error_real + error_imag*error_imag;

        sig_total += (X_DFT[i].re)*(X_DFT[i].re) + (X_DFT[i].im)*(X_DFT[i].im);
    }
    SNR = 10*log10(sig_total/error_total);
    xil_printf("FFT model SNR : %d dB\n",(int)SNR);


    error_total = 0;
    sig_total = 0;
    for(i = 0; i<N; i++){
        error_real = (X_DFT[i].re)-(X_FFT_Assembly[i].re);
        error_imag = (X_DFT[i].im) -(X_FFT_Assembly[i].im);

        error_total += error_real*error_real + error_imag*error_imag;

        sig_total += (X_DFT[i].re)*(X_DFT[i].re) + (X_DFT[i].im)*(X_DFT[i].im);
    }
    SNR = 10*log10(sig_total/error_total);
    xil_printf("FFT Assembly model SNR : %d dB\n",(int)SNR);

    return 0;
}
```

①

②

Programming in Assembly

- Review the function ***'FFT_Assembly()'***
  - ① Butterfly: Stage 1 ~ Stage 5
  - ② Butterfly: Stage 6 (*incomplete*)
  - ③ Reordering

```c
#ifndef STAGE6_H_
#define STAGE6_H_


void Stage6(complex *output, complex *input)
{
    int n;

    ///////////////////////////////////////////
    //
    //
    //      Fill Your Code Here.
    //
    //
    ///////////////////////////////////////////
}



#endif /* STAGE6_H_ */
```

```c
void FFT()
{
    complex input[N],temp[N];

    int out_re[N];
    int out_im[N];

    int data;
    int n,k;
```

①

```c
    for (data=0;data<N;data++)
    {
        input[data].re=in_real[data];
        input[data].im=in_imag[data];
    }

    for (n=0;n<32;n++) //stage1
    {
        temp[n]=add_cal(input[n],input[n+32]);
        temp[n+32]=multiple(sub_cal(input[n],input[n+32]),W[n]);
    }

    for (n=0;n<16;n++) //stage2
    {
        for (k=0;k<2;k++)
        {
            input[n+(32*k)]=add_cal(temp[n+(32*k)],temp[n+((32*k)+16)]);
            input[n+((32*k)+16)]=multiple(sub_cal(temp[n+(32*k)],temp[n+((32*k)+16)]),W[2*n]);
        }
    }

    for(n=0;n<8;n++) //stage-3
    {
        for (k=0;k<4;k++)
        {
            temp[n+(16*k)] = add_cal(input[n+(16*k)],input[n+((16*k)+8)]);
            temp[n+((16*k)+8)] = multiple(sub_cal(input[n+(16*k)],input[n+((16*k)+8)]),W[4*n]);
        }
    }

    for (n=0;n<4;n++) //stage4
    {
        for (k=0;k<8;k++)
        {
            input[n+(8*k)] = add_cal(temp[n+(8*k)],temp[n+((8*k)+4)]);
            input[n+((8*k)+4)] =multiple(sub_cal(temp[n+(8*k)],temp[n+((8*k)+4)]),W[8*n]);
        }
    }

    for (n=0;n<2;n++) //stage5
    {
        for (k=0;k<16;k++)
        {
            temp[n+(4*k)]=add_cal(input[n+(4*k)],input[n+(4*k+2)]);
            temp[n+(4*k+2)]=multiple(sub_cal(input[n+(4*k)],input[n+(4*k+2)]),W[16*n]);
        }

    }
```

②

```c
    // Stage 6
    Stage6(input, temp);
```

③

```c
    for(n=0;n<N;n++)
    {
        X_FFT[n]=input[Re_ordering(n)];
    }

    for (n=0;n<N;n++)
    {
        out_im[n]=(X_FFT[n].im)>>10;
        out_re[n]=(X_FFT[n].re)>>10;
    }
}
```

Programming in Assembly

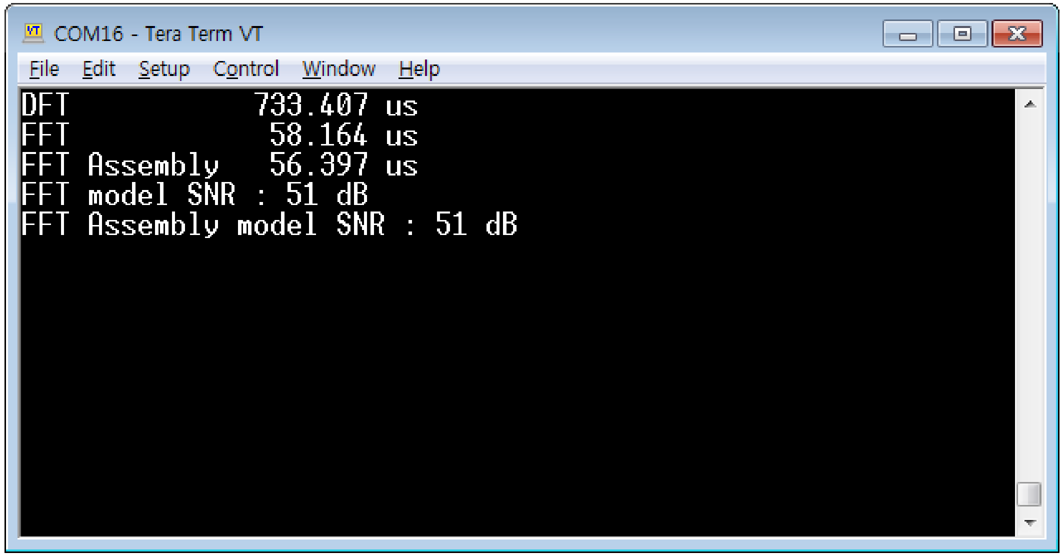
- ❑ Use the following assembly program

Stage6_Assembly.s

```
        .text
        .syntax   unified

        .align  4
        .global Stage6_Assembly
        .arm


Stage6_Assembly:
            push {r4, r5, r6, r7, r8, r9, r10, r11, lr}
            sub sp, sp, #20
            mov r9, r0
            mov r10, r1
            mov r4, #0
            add r7, r1, #8
            add r5, sp, #8
            add r6, r4, r10
            add r8, r7, r4
            ldr r3, [r8, #4]
            str r3, [sp]
            ldr r3, [r7, r4]
            mov r0, r5
            ldm r6, {r1, r2}
            bl add_cal_assembly
            add r3, r4, r9
            ldm r5, {r0, r1}
            stm r3, {r0, r1}
            add r11, r3, #8
            ldr r3, [r8, #4]
            str r3, [sp]
            ldr r3, [r7, r4]
            mov r0, r5
            ldm r6, {r1, r2}
            bl sub_cal_assembly
            ldm r5, {r0, r1}
            stm r11, {r0, r1}
            add r4, r4, #16
            cmp r4, #512
            bne Stage6_Assembly+28
            add sp, sp, #20
            pop {r4, r5, r6, r7, r8, r9, r10, r11, pc}
```

Programming in Assembly

- Run the application
  - Check the output of the application on ***'Tera Term'***
    - ✓ Check the performance gain.
    - ✓ Check the accuracy given by SNR

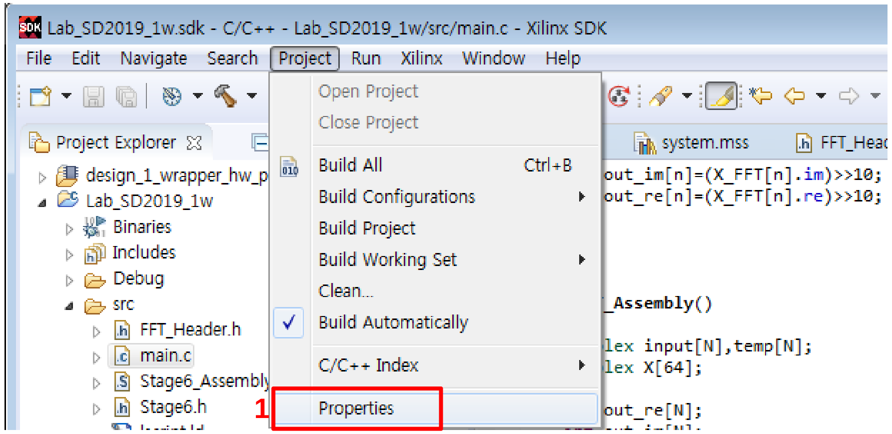


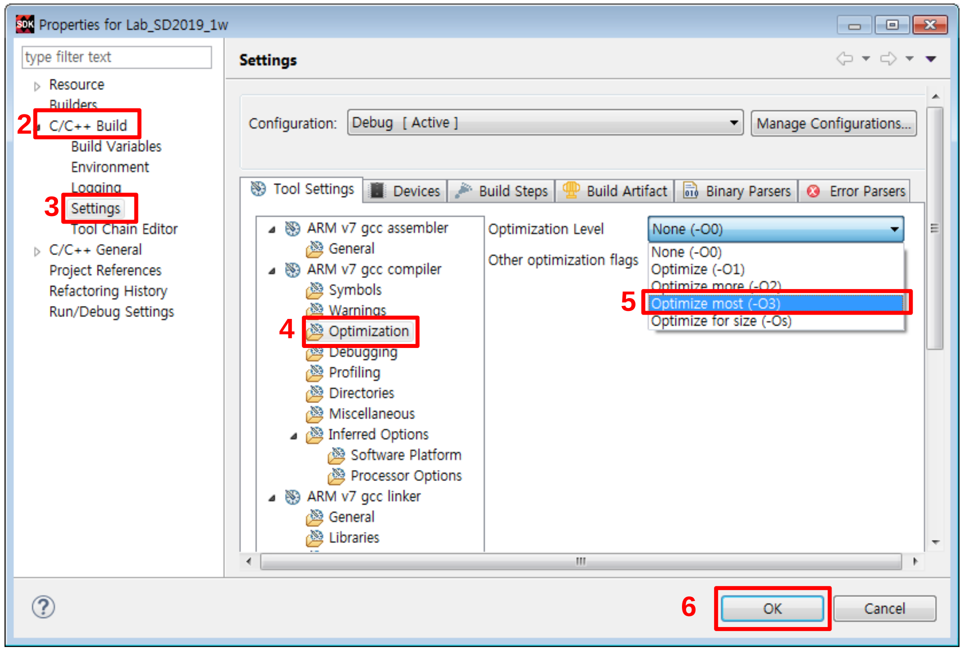
Programming in Assembly

- Debug the application
  - Repeat the steps depicted in pp. 20~27 to reason for the performance gain.

Setting Optimization Level

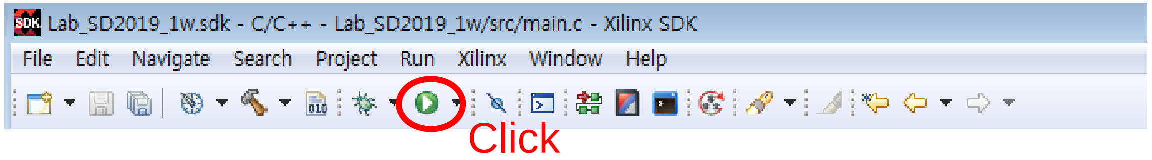
- Set the compiler optimization level
  - Select ***'Project'*** menu and click ***'Properties'***
  - Select ***'Settings'*** tap and click ***'ARM v7 gcc compilier > Optimization'***
  - Select ***'Optimization most (-O3)'*** in the dropdown menu of ***'Optimization Level'***
  - Click ***'OK'***



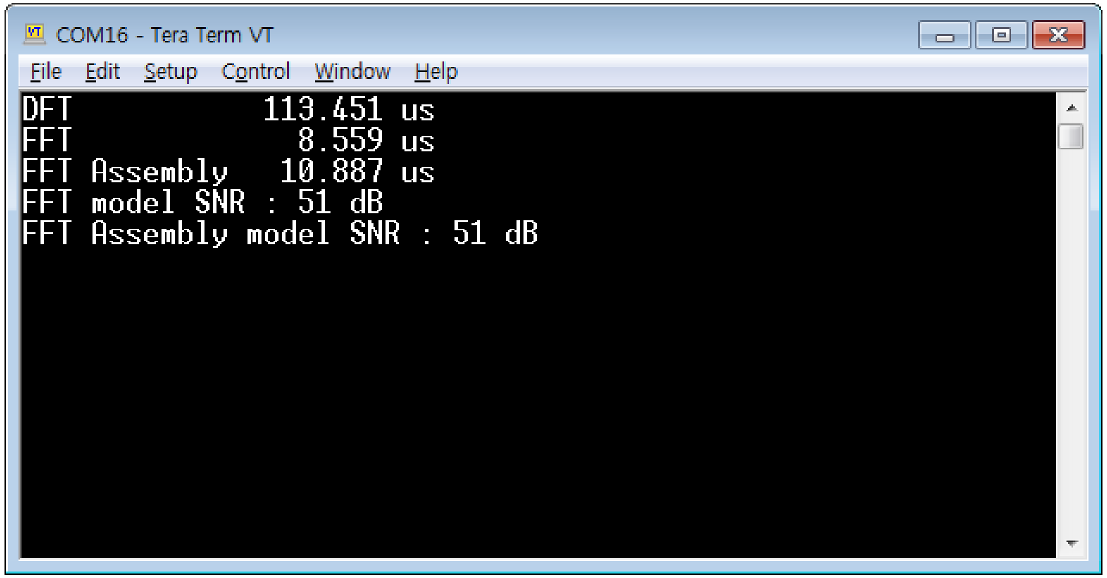


Setting Optimization Level

- ❑ Run the application
  - Click the ***'Run As…'*** icon to run the application again



  - Check the output of the application on ***'Tera Term'***
    - ✓ Check how much it accelerates the application



Setting Optimization Level

- Debug the application
  - Repeat the steps depicted in pp. 20~27 to figure out the impact of the compiler optimization level on the assembly codes

Demo

- ❑ Compare the (normalized) execution times of all the **three** FFT implementations as follows
  - Optimization level: O0
    - ✓ D-cache disabled/enabled
  - Optimization level: O3
    - ✓ D-cache disabled/enabled
- ❑ Figure out the reasons for the measured speedup