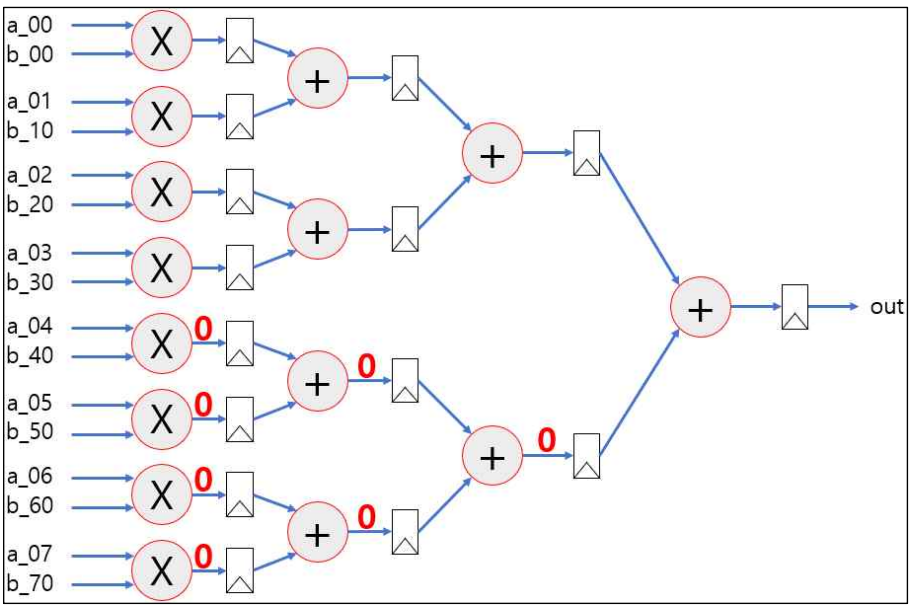
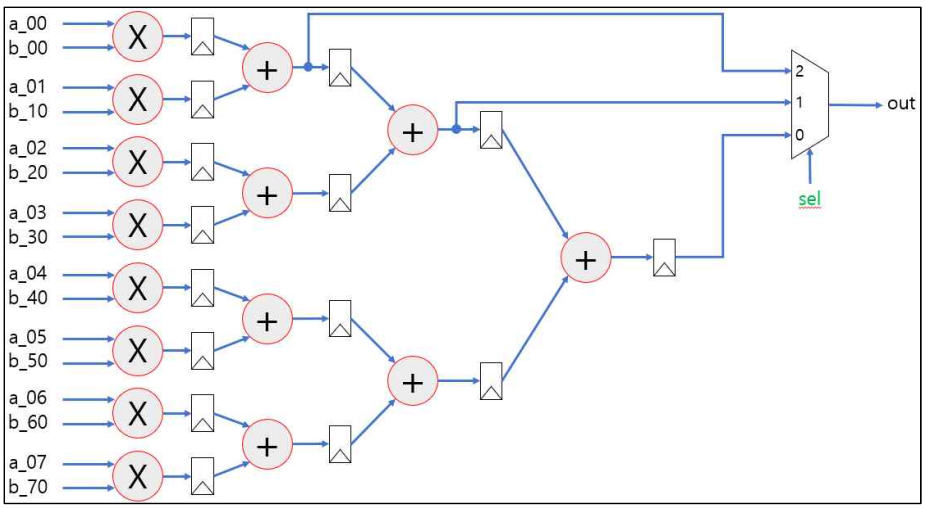
Team Project 최종 보고서

2022221762 서주원
2022222447 박주동

1. 설계할 HW의 목적 및 구조



- matrix multiplication의 핵심인 MAC 연산을 수행하기 위한 Adder tree 기반의 HW 가속기 구조를 마련함 (8x8 size)
- 그러나 최근 matrix의 sparsity가 높은 특징 때문에 0 값이 많이 존재하여 위의 그림처럼 tree 일부의 연산 결과가 0이어도 사이클을 그대로 소모해야 하는 문제가 있음



- 이를 보완하기 위해 그림과 같은 구조를 설계함
- a, b의 sparsity에 따라서 output 값이 더 빠르게 빠져나올 수 있도록 별도의 path와 mux를 추가함
- 또한, 어떠한 output을 선택할지 결정하도록 select 신호를 추가함
- 위에서부터 adder tree의 "**1/4만 사용하는 경우** (-2 사이클)", "**1/2만 사용하는 경우** (-1 사이클)", "**전부 사용하는 경우**"로 구분됨

2. HLS에서 C로 모듈 제작

mac8x8.cpp | Synthesis(solution1)

```
void mac8x8(
    int a[8*8],
    int t_b[8*8], //transposed b
    int sel[8*8],
    int out[8*8])
{
#pragma HLS INTERFACE s_axilite port=return bundle=CRTL_BUS
#pragma HLS INTERFACE bram port=a
#pragma HLS INTERFACE bram port=t_b
#pragma HLS INTERFACE bram port=sel
#pragma HLS INTERFACE bram port=out

    int res_mul[8];
    int res_add0[4];
    int res_add1[2];
    int res_add2;

    Col: for (int i=0; i<8; i++) {
        Row: for (int j=0; j<8; j++) {
            Mult: for (int k=0; k<8; k++) {
                res_mul[k] =a[i*8 + k] * t_b[j*8 + k];
            }

            Add0: for (int l=0; l<4; l++) {
                res_add0[l] = res_mul[2*l] + res_mul[2*l+1];
            }
            if (sel[i*8 + j] == 2)
                out[i*8 + j] = res_add0[0]; //-2 cycles

            else {
                Add1: for (int m=0; m<2; m++) {
                    res_add1[m] = res_add0[2*m] + res_add0[2*m+1];
                }
                if (sel[i*8 + j] == 1)
                    out[i*8 + j] = res_add1[0]; // -1 cycle

                else {
                    Add2: res_add2 = res_add1[0] + res_add1[1];

                    if (sel[i*8 + j] == 0)
                        out[i*8 + j] = res_add2;
                    else
                        out[i*8 + j] = 0;
                }
            }
        }
    }
}
```

- BRAM 사용을 효율적으로 하기 위해서 HLS에서 먼저 해당 HW를 구현하였음
- 행렬 데이터 관리를 편하게 하기 위해서 2차원 배열을 flatten시켜 1차원 배열로 선언 및 관리함 (a[8][8] -> a[64])
- 각 port를 bram과 연동하여 동작하도록 directive 설정을 추가함

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.510	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
2897	3025	2897	3025	none

Detail

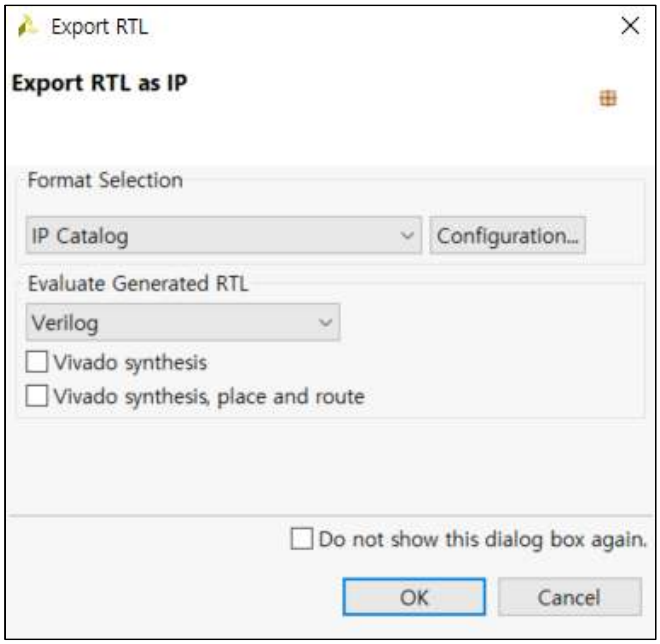
Instance

Loop

Utilization Estimates

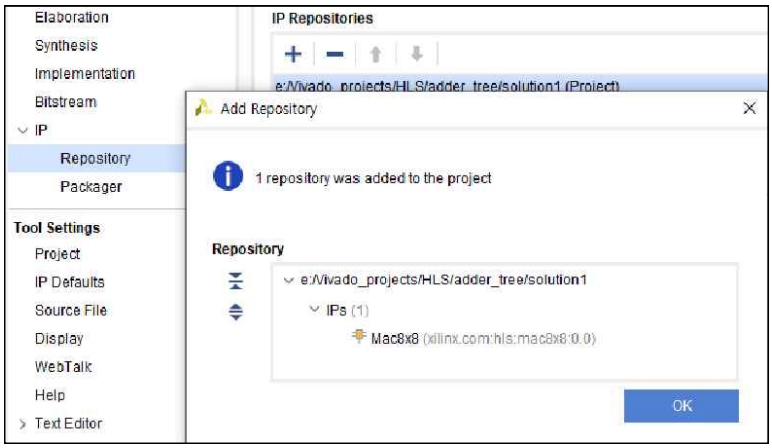
Summary

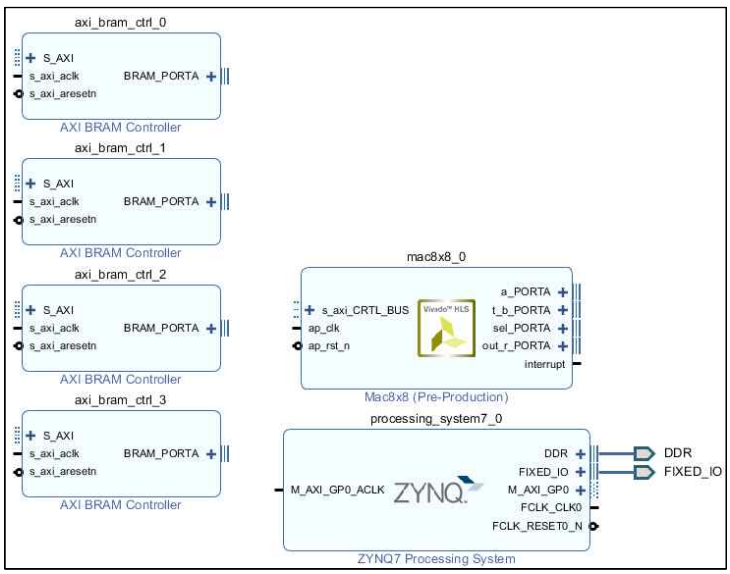
Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	3	0	529
FIFO	-	-	-	-
Instance	0	-	36	40
Memory	0	-	64	4
Multiplexer	-	-	-	164
Register	-	-	382	-
Total	0	3	482	737
Available	280	220	106400	53200
Utilization (%)	0	1	~0	1



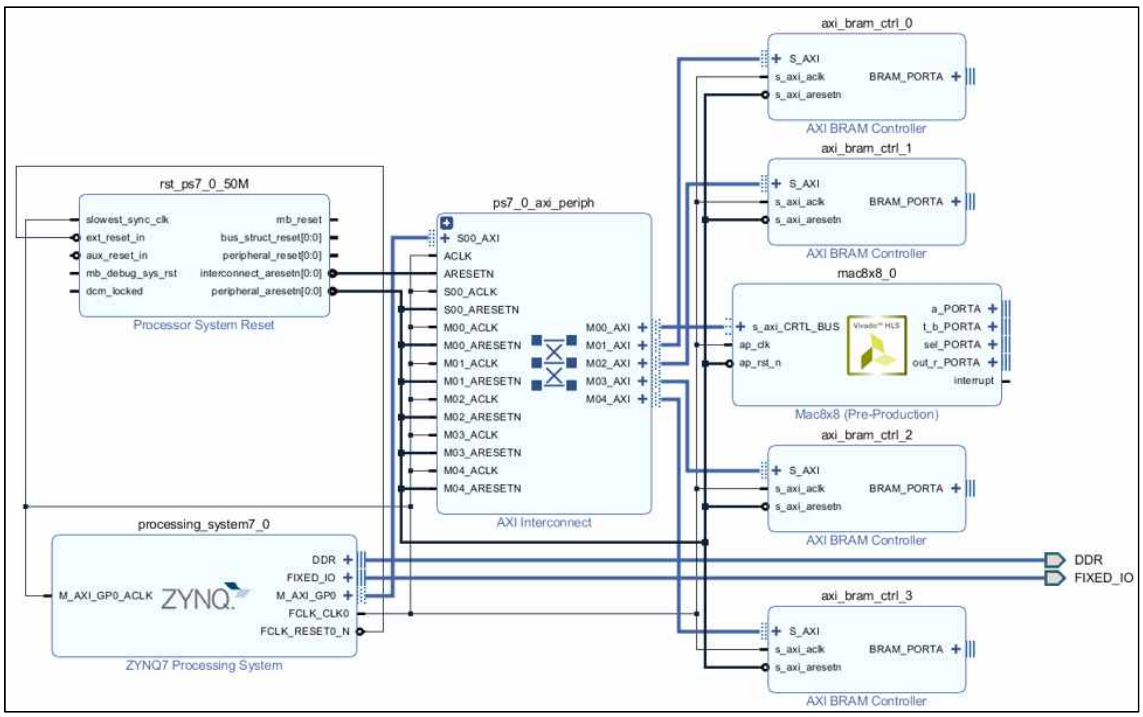
- 해당 함수(mac8x8)를 합성 및 export하여 custom IP를 생성함

3. Vivado에서 블록 디자인 설계

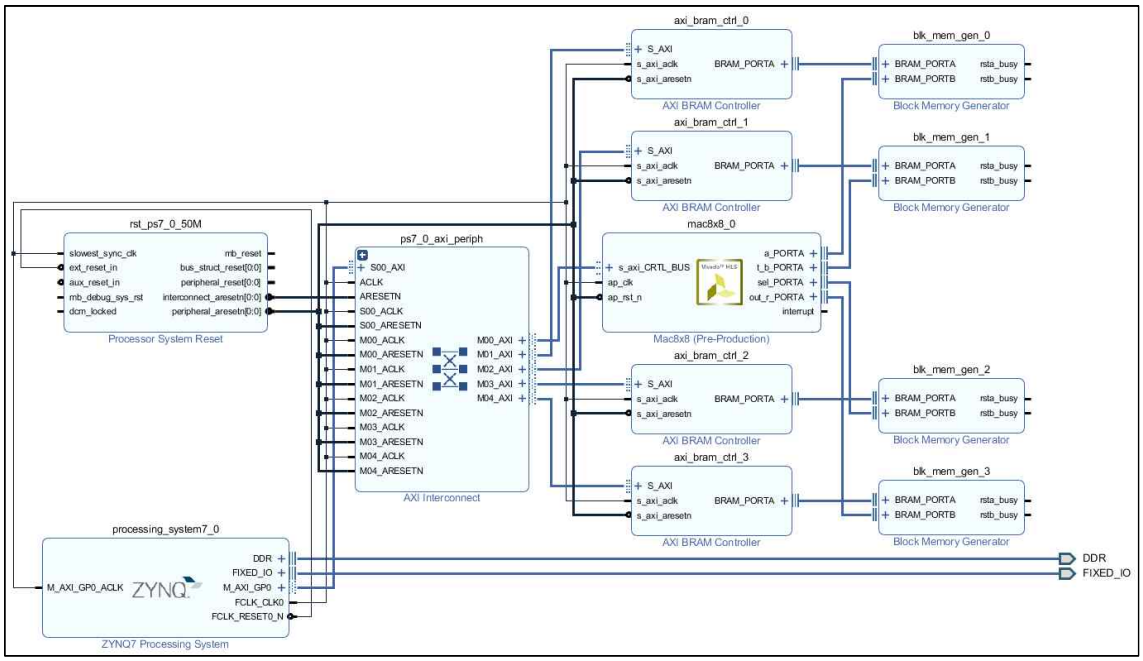




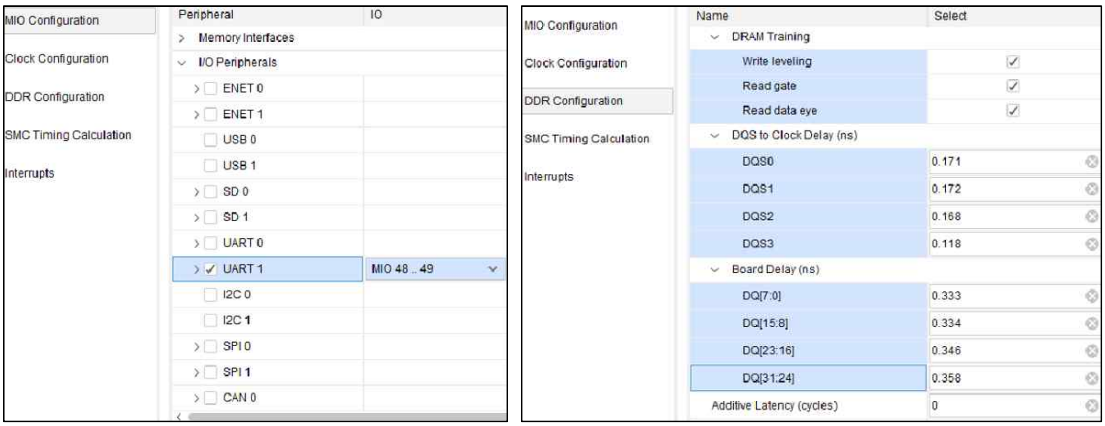
- mac8x8의 IP 블록을 Settings에서 등록한 후, Block design에서 불러와 배치함
- 각 port와 연결할 AXI BRAM Controller를 4개 생성



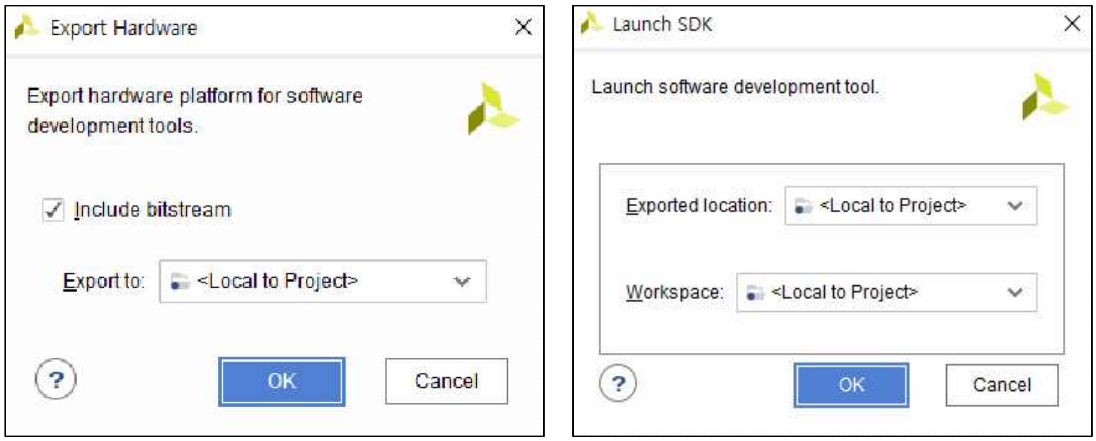
- AXI port들에 대하여 Connection Automation을 실행한 결과, 위 그림과 같이 정렬됨



- 이어서 BRAM Memory Generator 블록을 4개 생성하여 True Dual Port RAM으로 설정한 후 위 그림과 같이 port를 연결함

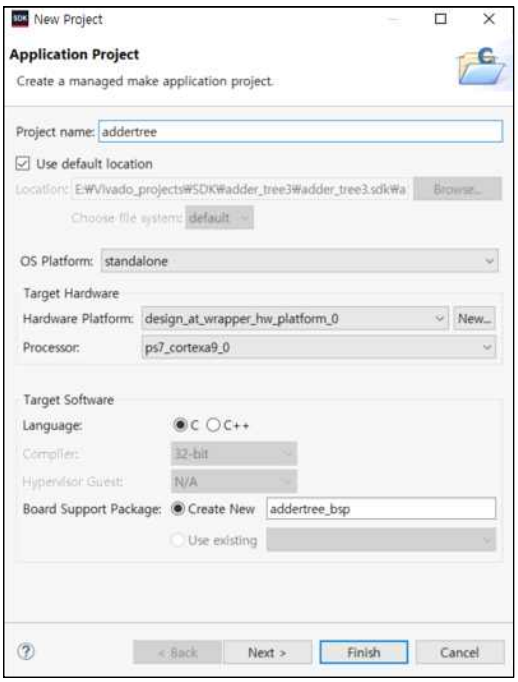


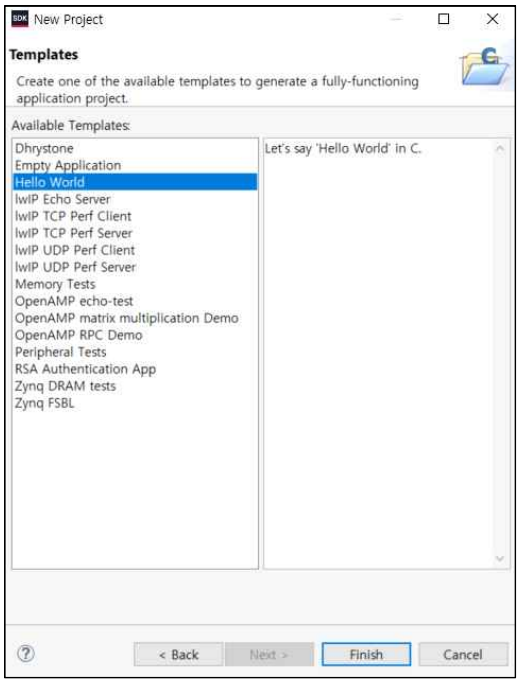
- ZYNQ7 PS를 더블 클릭하여 UART 활성화 및 DDR Configuration 진행



- Validation 진행하여 error가 없는지 확인함
- 해당 design에 대한 Bitstream을 생성한 후 Launch SDK

4. SDK에서 검증하기





- New application project를 생성 (Hello World 템플릿)
- 이후 helloworld.c 내용을 다음과 같이 수정함

```c
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xparameters.h"
#include <xmac8x8.h>

//for timing test
#include "xtime_l.h"
float test_time;
float total_time=0;
XTime time_before_test;
XTime time_after_test;

//bram address
int * bram_a_addr   = XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR; // 0x4000_0000
int * bram_tb_addr  = XPAR_AXI_BRAM_CTRL_1_S_AXI_BASEADDR; // 0x4200_0000
int * bram_sel_addr = XPAR_AXI_BRAM_CTRL_2_S_AXI_BASEADDR; // 0x4400_0000
int * bram_out_addr = XPAR_AXI_BRAM_CTRL_3_S_AXI_BASEADDR; // 0x4600_0000

XMac8x8 Xmac;
XMac8x8_Config *Xmac_Config;

void init_XMac_Core()
{
   int status = 0;

   Xmac_Config = XMac8x8_LookupConfig(XPAR_MAC8X8_0_DEVICE_ID);
   if(Xmac_Config)
   {
      status = XMac8x8_CfgInitialize(&Xmac, Xmac_Config);
      if(status != XST_SUCCESS)
      {
         printf("Failed to initialize\n");
      }
   }
}
```

```c
int main()
{
    init_platform();
    init_XMac_Core();

    printf("\n\r-------PROGRAM START-------\n\r");

    int a [8][8] = {
            {1,2,3,4,0,0,0,0},
            {1,2,0,0,0,0,0,0},
            {1,2,3,4,0,0,0,0},
            {1,2,3,0,0,0,0,0},
            {1,2,0,0,0,0,0,0},
            {1,0,0,0,0,0,0,0},
            {1,2,3,0,0,0,0,0},
            {1,2,3,4,0,0,0,0}
    };

    int tb [8][8] = { //transposed b
            {1,1,1,1,1,0,0,0},
            {1,1,1,1,0,0,0,0},
            {1,1,1,1,1,1,0,0},
            {1,1,1,1,1,1,0,0},
            {1,1,1,1,1,1,1,1},
            {1,1,1,0,0,0,0,0},
            {1,1,1,1,1,1,1,1},
            {1,1,1,1,1,0,0,0}
    };

    int sel [8][8] = { // 0=orig, 1=50%, 2=25%
            {1,1,1,1,1,1,1,1},
            {2,2,2,2,2,2,2,2},
            {1,1,1,1,1,1,1,1},
            {1,1,1,1,1,1,1,1},
            {2,2,2,2,2,2,2,2},
            {2,2,2,2,2,2,2,2},
            {1,1,1,1,1,1,1,1},
            {1,1,1,1,1,1,1,1}
    };

    int out [8][8];

    int i, j, k;
```

- (왼쪽 그림)
- 수행 시간 측정을 위한 header 및 변수 선언
- Bram address를 Vivado의 Address Editor에서 참조하여 a, tb, sel, out 변수에 알맞게 할당
- Initializing을 위한 함수는 기존 수업에서 사용한 코드를 참조함
- (오른쪽 그림)
- 연산에 사용할 행렬 a, tb(transposed b), selection 신호 sel, output 행렬 out, loop index 값들을 미리 선언함

```
    printf("<a, tb, sel -> BRAM>\n\r");
    for (i=0; i<8; i++) {
        for (j=0; j<8; j++) {
            bram_a_addr[i*8 + j] = a[i][j];
            printf("a[%d] = %2d ", (i*8+j), bram_a_addr[i*8 + j]);

            bram_tb_addr[i*8 + j] = tb[i][j];
            printf("tb[%d] = %2d ", (i*8+j), bram_tb_addr[i*8 + j]);

            bram_sel_addr[i*8 + j] = sel[i][j];
            printf("sel[%d] = %2d \n", (i*8+j), bram_sel_addr[i*8 + j]);
        }
        printf("\n");
    }

    for (k = 0; k < 500; k++) {
        //for timing test
        XTime_GetTime(&time_before_test);

        // Read & Write Reg from BRAM
        XMac8x8_Start(&Xmac);
        while(!XMac8x8_IsDone(&Xmac));

        //for timing test
        XTime_GetTime(&time_after_test);

        test_time = (float)(time_after_test - time_before_test)/(COUNTS_PER_SECOND*0.000001);
        total_time += test_time;
        printf("\n<Loop %d time: %.3f us>\n", k+1, (total_time/(k+1)));
    }

    printf("\n<BRAM -> out>\n\r");
    for (i=0; i<8; i++) {
        for (j=0; j<8; j++) {
            out[i][j] = bram_out_addr[i*8 + j];
            printf("%3d ", out[i][j]);
        }
        printf("\n");
    }

    //for timing test
    //test_time = (float)(time_after_test - time_before_test)/(COUNTS_PER_SECOND*0.000001);
    printf("\n<Execution time: %.3f us>\n", (total_time/k));

    cleanup_platform();
    return 0;
}
```

- 먼저 a, tb, sel 값들을 먼저 bram에 담음
- HW를 동작 시키기 직전과 동작완료 직후의 시간을 측정함
- 총 500번의 동작시간의 평균을 구함
- 동작이 완료되면 output 이 담긴 bram으로부터 값들을 가져오며, print를 통해 값을 확인
- 최종적으로 수행 시간(의 평균)을 print

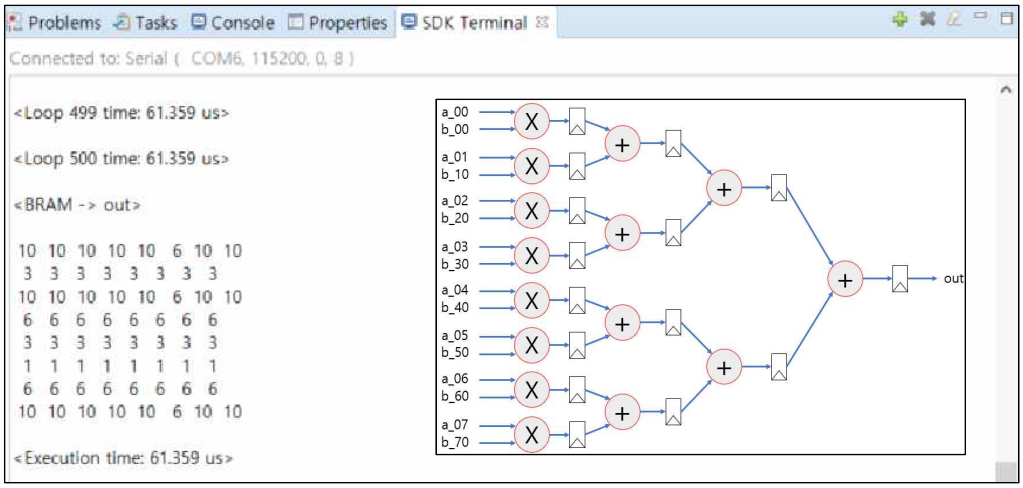
$$\begin{pmatrix} 1 & 2 & 3 & 4 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 & 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 10 & 10 & 10 & 10 & 10 & 6 & 10 & 10 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 10 & 10 & 10 & 10 & 10 & 6 & 10 & 10 \\ 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\ 10 & 10 & 10 & 10 & 10 & 6 & 10 & 10 \end{pmatrix}$$

(실제 행렬 곱셈 결과)

5. 검증 결과

```
bram_sel_addr[i*8 + j] = 0; //sel[i][j];
printf("sel[%d] = %2d \n", (i*8+j), bram_sel_addr[i*8 + j]);
```

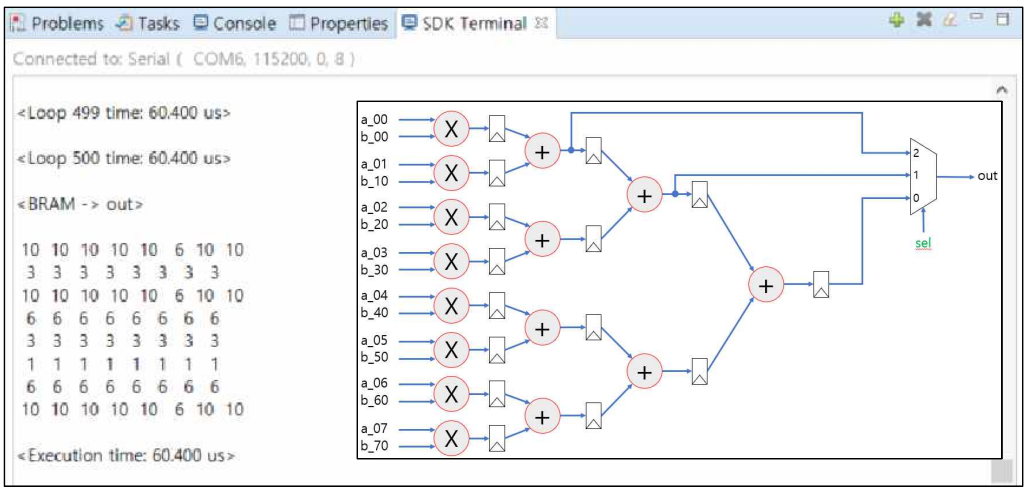
- 먼저 selection 값을 전부 0으로 넣어서 sparsity에 상관없이 모든 adder tree가 동작하도록 한 후 SW를 실행함 (기존 구조로 동작)



- 터미널을 확인한 결과 500번 동안 평균 61.359 us 소요된 것을 확인함

```
bram_sel_addr[i*8 + j] = sel[i][j];
printf("sel[%d] = %2d \n", (i*8+j), bram_sel_addr[i*8 + j]);
```

- 이후 다시 sel 값을 sparsity에 알맞게 넣어주어 output이 더 빠르게 나오도록 함 (개선된 구조로 동작)



- 500번 동안 평균 60.400 us 소요된 것을 확인함

- 개선된 구조로 동작시켜서 61.359 - 60.400 = **0.959 us** 정도의 수행 시간 감소 효과를 얻을 수 있었음
- 그리고 두 HW의 연산 결과가 같은 것을 확인하여 연산 결과에도 이상이 없음을 확인하였음.