명지 공학인 윤리 서약서

보고서 및 논문 윤리 서약

1. 나는 보고서 및 논문의 내용을 조작하지 않겠습니다.
2. 나는 다른 사람의 보고서 및 논문의 내용을 내 것처럼 무단으로 복사하지 않겠습니다.
3. 나는 다른 사람의 보고서 및 논문의 내용을 참고하거나 인용할 시 참고 및 인용 형식을 갖추고 출처를 반드시 밝히겠습니다.
4. 나는 보고서 및 논문을 대신하여 작성하도록 청탁하지도 청탁받지도 않겠습니다.

나는 보고서 및 논문 작성 시 위법 행위를 하지 않고, 명지인으로서 또한 공학인으로서 나의 양심과 명예를 지킬 것을 약속합니다.

시험 윤리 서약

1. 나는 대리시험을 청탁하거나 청탁받지 않겠습니다.
2. 나는 허용되지 않은 교과서, 노트 및 타학생의 답안지 등을 보고 답안지를 작성하지 않겠습니다.
3. 나는 타인에게 답안지를 보여주지 않겠습니다.
4. 나는 감독관의 지시와 명령에 따라 시험 과정에 참여하겠습니다.

나는 시험에 위법 행위를 하지 않고, 명지인으로서 또한 공학인으로서 나의 양심과 명예를 지킬 것을 약속합니다.

2023년 05월 05일

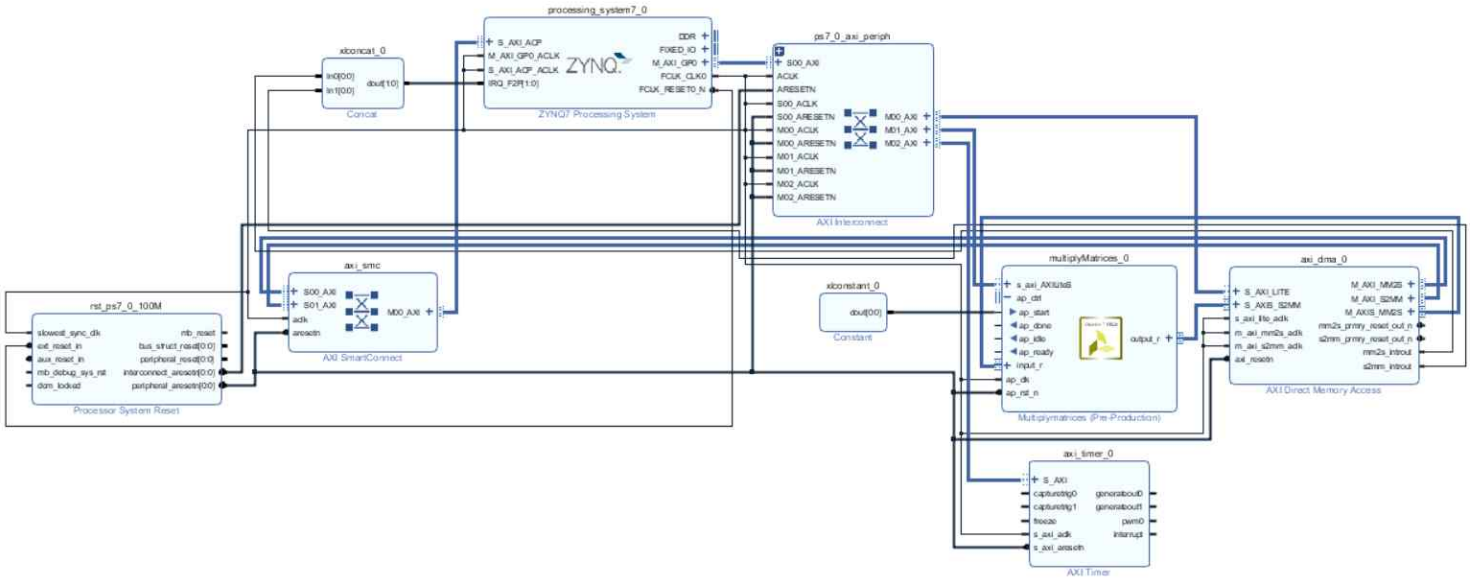
서약자

(학번) 60171878

(성명) 허무혁 **(인)**

Hardware Matrix_multiplication with HLS

60171878 허무혁



목차

1. 설계 방법

```
procedure MatrixMultiplication(A, B)
  input A, B n*n matrix
  output C, n*n matrix

begin
  for ( i = 0; i < n; i++)
    for ( j = 0; j < n; j++)
      C[i,j] = 0;
    end for
  end for

  for ( i = 0; i < n; i++)
    for ( j = 0; j < n; j++)
      for( k = 0; k < n; k++)
       C[i,j] = C[i,j] + A[i,k] * B[k,j]
      end for
    end for
  end for
end MatrixMultiplication
```

그림 3 matrix multiplication Pseudocode

위 그림은 행렬곱 알고리즘을 표현한 수도 코드이다. 다음 수도 코드를 C언어 기반으로 작성하면 다음과 같다.

입력으로 행렬 A와 B를 받고 출력으로 행렬 C를 내보낸다.

계산을 3중 for문을 돌며 행렬 A와 B의 요소를 하나씩 곱해 기존의 C값과 덧셈을 한다.

이때 행렬 A의 size가 unsigned char 일 때 255까지 표현 할 수 있고 255*255는 65025이다. 이때 행렬 C 또한 unsigned char를 사용한다면 65025까지 표현해줄 수 없으므로 행렬 C는 unsigned int 형을 이용하도록 한다.

matrix A[n][m], marix B[m][p]에서 행렬의 rows와 cols를 결정하는 n, m, p는 2^ln, 2^lm, 2^lp등으로 결정하도록 하며 ln, lm, lp의 최대값은 5까지 사용한다. 따라서 n, m, p의 최대값은 2^5 =32까지이다.

```
void MultiplyMatrices(int nCount, double **matrixA,
                      double **matrixB, double **matrixC)
{
    int i, j, k ;

    for (i = 0; i < nCount; i++)
    {
        for (j = 0; j < nCount; j++)
        {
            matrixC[i][j]=0;

            for (k = 0; k < nCount; k++)
            {
                matrixC[i][j] +=
                    matrixA[i][k]*matrixB[k][j];
            }
        }
    }
}
```

그림 4 matrix_mul C code

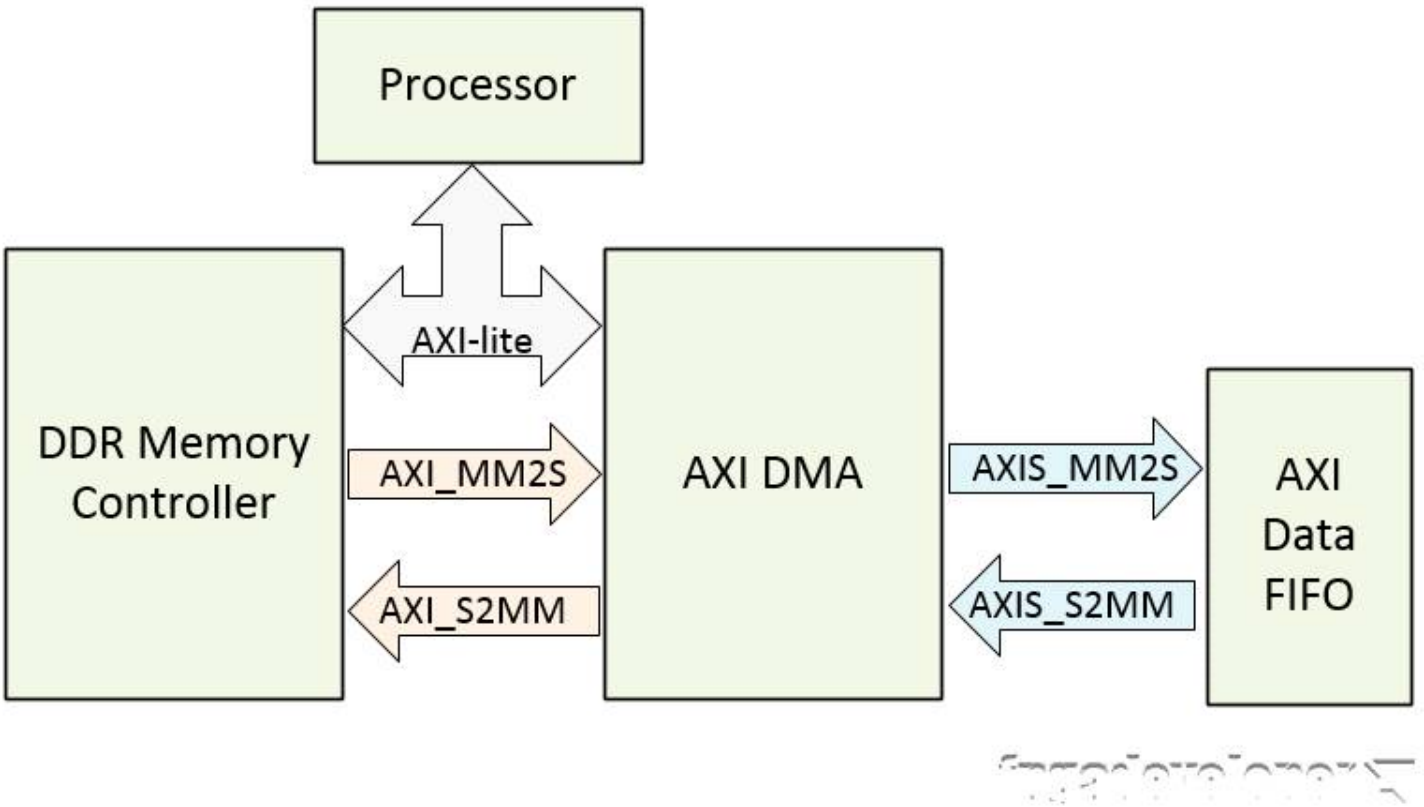


그림 5 AXI DMA

위의 내용을 기반으로 HLS를 설계한다. HLS 설계할 때 A, B는 하나의 입력신호로 만들어 줄 것이다. 하나의 입력신호로 만들어주는 이유는 A, B가 axi stream interface를 가지고 DMA ip block을 이용하게 할 것이기 때문에, 개발의 용이성을 높이기 위함이다. A, B는 AXI_MM2S를 이용한다. 출력 C는 AXIS_S2MM을 이용한다. ln, lm, lp 값은 AXI-lite를 이용한다. 단 여기서 입력 A, B는 하나의 포트에서 나가야한다.

최종 모습은 다음과 같다.

void multiplyMatrices(unsigned int input[2*MAX], unsigned int output[MAX], int ln, int lm, int lp)
다음과 같이 Function을 생성한다.

```
#pragma HLS INTERFACE s_axilite port=lp
#pragma HLS INTERFACE s_axilite port=lm
#pragma HLS INTERFACE s_axilite port=ln
#pragma HLS INTERFACE axis register both port=output
```

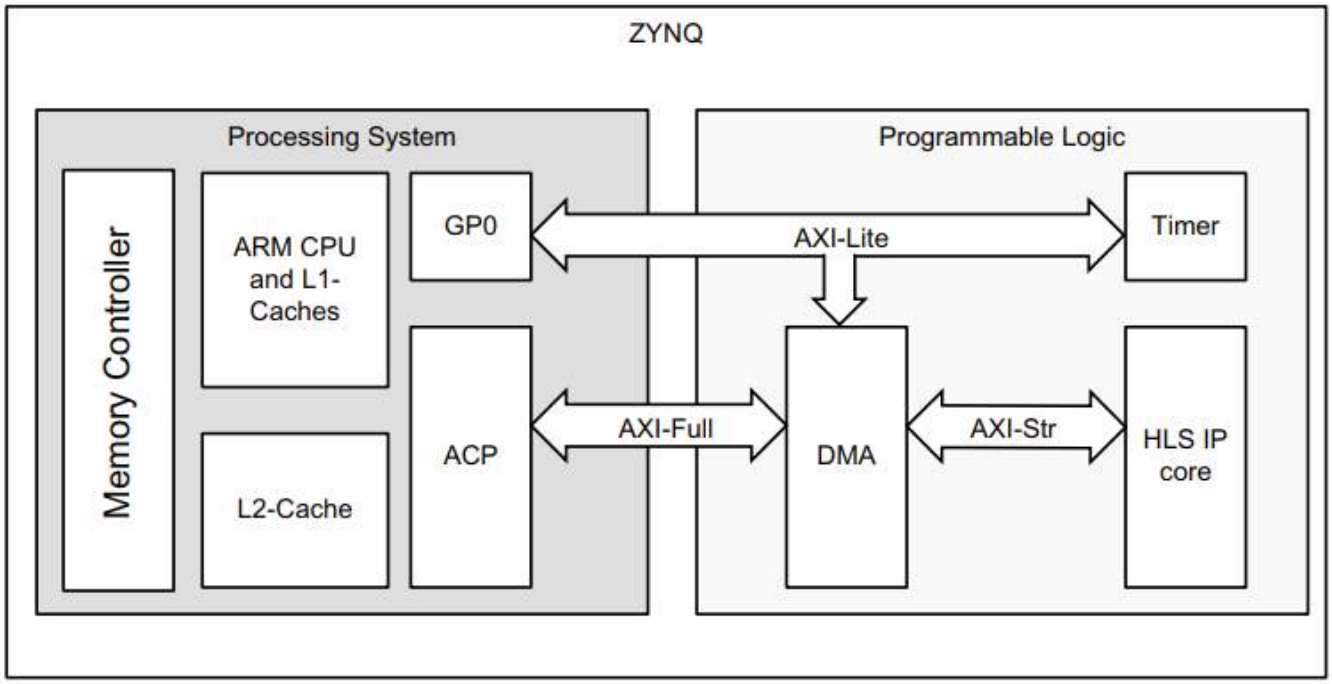


그림 6 블록

#pragma HLS INTERFACE axis register both port=input
인터페이스를 설정한다.

인풋으로 들어온 값은 다음과 같이 나누어 준다.

```
// matrixA에 변수 할당
	for (int i = 0; i < n; i++)
	{
		for (int j = 0; j < m; j++)
		{
			A[i][j] = input[count];
			count++;
		}
	}
// matrixB에 변수 할당
	for (int i = 0; i < m; i++)
	{
		for (int j = 0; j < p; j++)
		{
			B[i][j] = input[count];
			count++;
		}
	}
```

출력 AB는 다음과 같이 생성한다.

```
	for (i = 0; i < n; i++) {
		for (j = 0; j < p; j++) {
			for (k = 0; k < m; k++) {
				sum += A[i][k] * B[k][j];
			}
			output[count] = sum;
			sum = 0;
			count++;
		}
	}
```

이제 Test Bench 코드를 작성하고 HW 결과와 SW결과를 비교한다.

```
testbench
1456    1484    1512    1540
4400    4492    4584    4676
7344    7500    7656    7812
10288   10508   10728   10948
success HW and SW  result match
1456    1484    1512    1540
4400    4492    4584    4676
7344    7500    7656    7812
10288   10508   10728   10948
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

그림 7 SW_HW 비교

```
testbench
8432    16075   17956   19692
42070   61583   68813   53289
9374    20989   23441   29421
768     2913    3252    5292
36964   56135   62722   51714
18620   26695   29830   22230
15952   33500   37416   44712
4456    19676   21964   37404
success HW and SW  result match
8432    16075   17956   19692
42070   61583   68813   53289
9374    20989   23441   29421
768     2913    3252    5292
36964   56135   62722   51714
18620   26695   29830   22230
15952   33500   37416   44712
4456    19676   21964   37404
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
Finished C simulation.
```

그림 8 SW_HW 비교

HW_SW 비교를 통과했으므로 C/RTL Co-simulation을 진행한다.

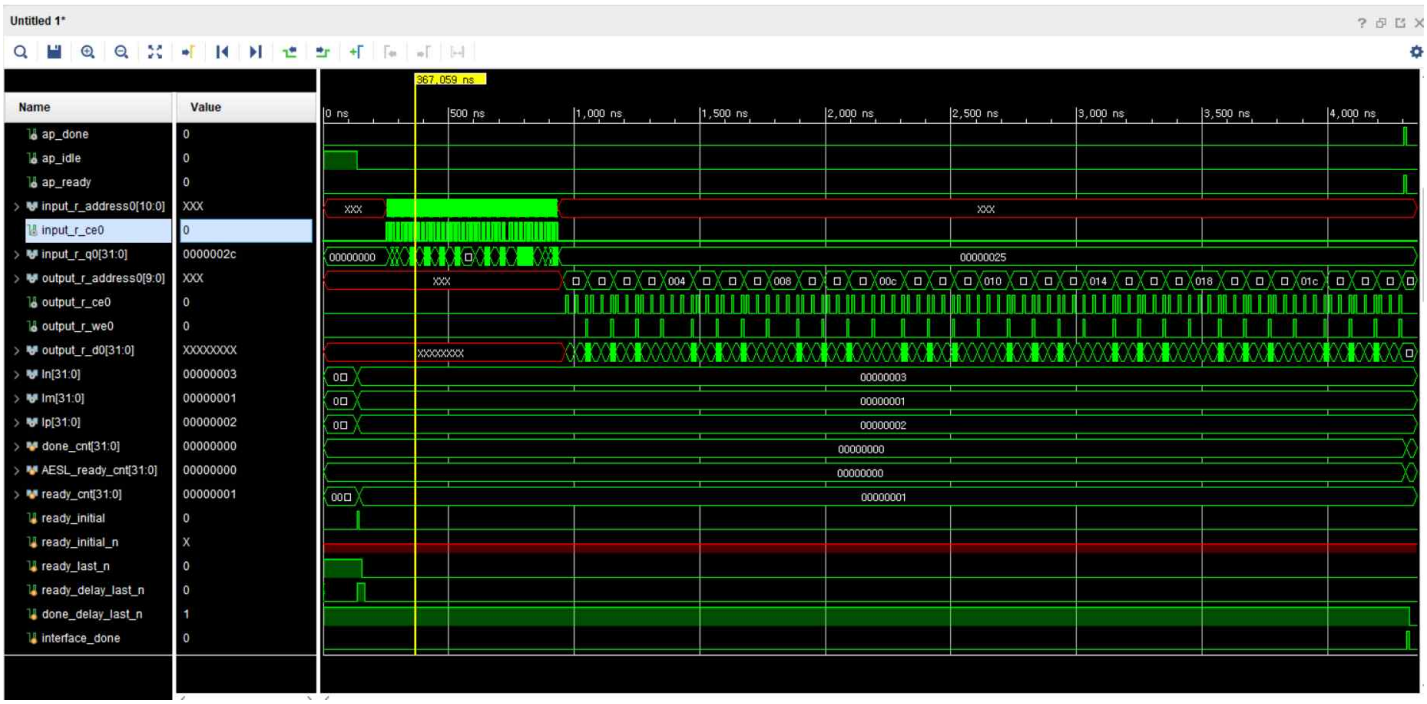
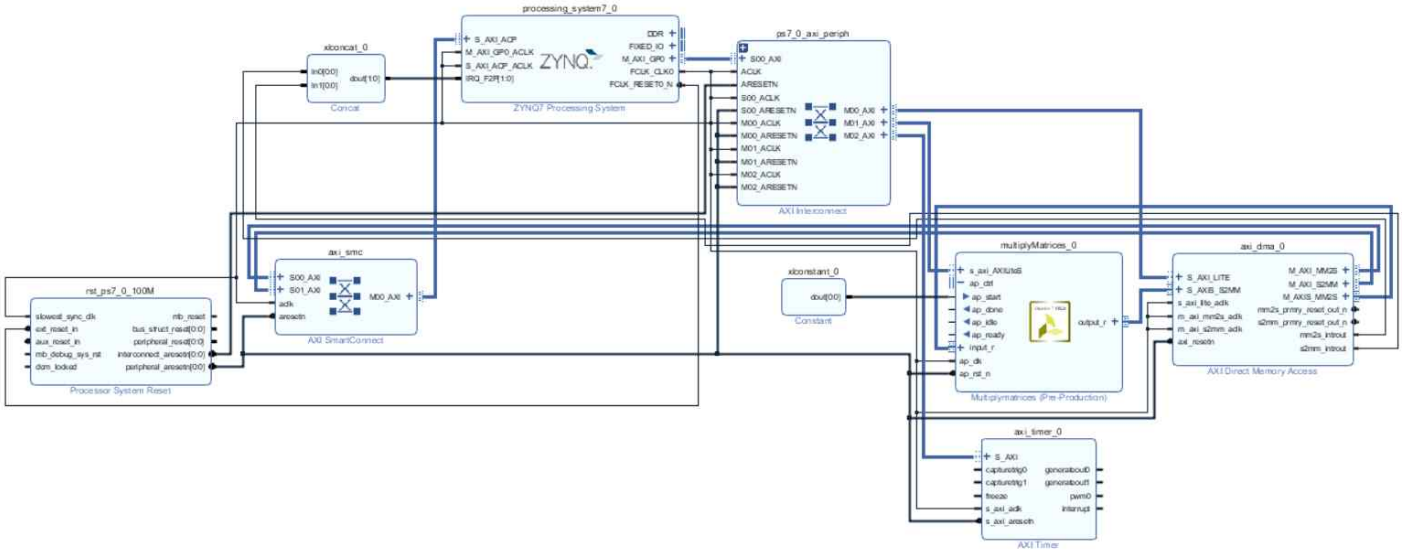


그림 9 C/RTL Co-simulation

파형을 보면 인풋 어드레스가 들어오고 무언가 연산을 끝낸뒤 아웃풋 주소가 들어오고 결과를 입력하는 것을 볼 수 있다.
이제 IP를 생성하고 그림 6에 맞추어 블록 디자인을 진행한다. 이때 timer도 추가해준다.



Bit Stream을 생성한 후 uart를 통해 결과를 검증한다.

Status = XAxiDma_SimpleTransfer(&AxiDma, (UINTPTR)TxBuffer, BYTES_TO_TRANSFER, XAXIDMA_DMA_TO_DEVICE);

Status = XAxiDma_SimpleTransfer(&AxiDma, (UINTPTR)RxBuffer, RESULT_ARRAY_TRANSFER, XAXIDMA_DEVICE_TO_DMA);
다음 DMA 코드를 통해 값을 보내고, 읽어온다.

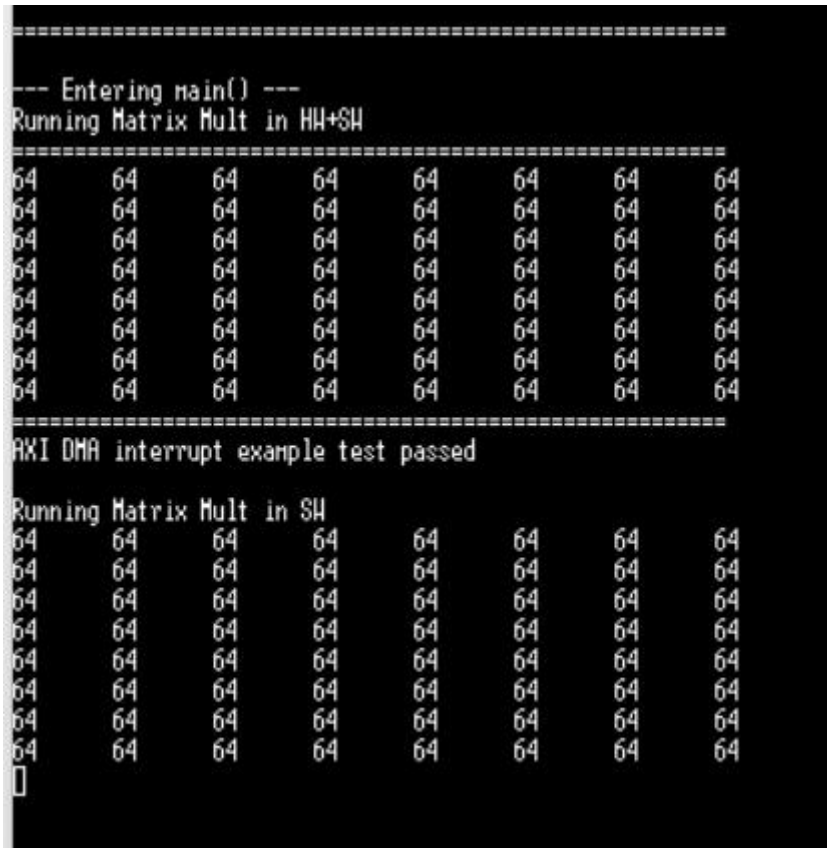
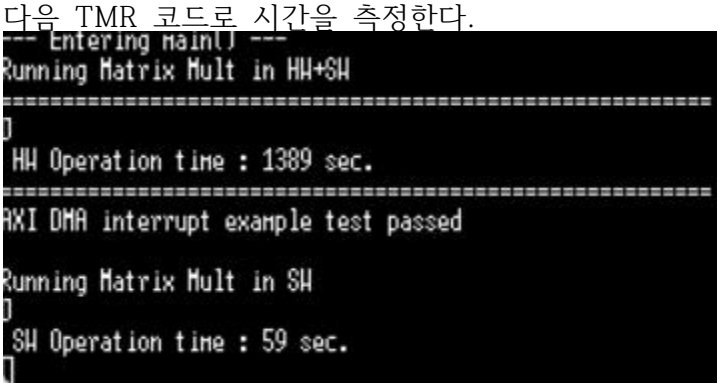


그림 11 SDK 통한 HW SW 검사

SDK로 계산한 결과 HW와 SW가 일치했다.

```
begin_time = XTmrCtr_GetValue(&timer_dev, XPAR_AXI_TIMER_DEVICE_ID);
Status = XAxiDma_SimpleTransfer(&AxiDma, (UINTPTR)TxBuffer, BYTES_TO_TRANSFER,
XAXIDMA_DMA_TO_DEVICE);
end_time = XTmrCtr_GetValue(&timer_dev, XPAR_AXI_TIMER_DEVICE_ID);
```

다음 TMR 코드로 시간을 측정한다.



2. 소스 코드 (핵심 부분만)

1. HLS

```
#define MAX             1024

void multiplyMatrices(unsigned int input[2*MAX], unsigned int output[MAX], int ln, int lm, int lp)
{
#pragma HLS INTERFACE s_axilite port=lp
#pragma HLS INTERFACE s_axilite port=lm
```

```
#pragma HLS INTERFACE s_axilite port=ln
#pragma HLS INTERFACE axis register both port=output
#pragma HLS INTERFACE axis register both port=input


	int i, j, k;
	int n = 1, m = 1, p = 1;
	int sum = 0;
	int count = 0;

	unsigned int A[32][32];
	unsigned int B[32][32];

	n = 1 << ln;
	m = 1 << lm;
	p = 1 << lp;

	// matrixA에 변수 할당
	for (int i = 0; i < n; i++)
	{
		for (int j = 0; j < m; j++)
		{
			A[i][j] = input[count];
			count++;
		}
	}
	// matrixB에 변수 할당
	for (int i = 0; i < m; i++)
	{
		for (int j = 0; j < p; j++)
		{
			B[i][j] = input[count];
			count++;
		}
	}

	count = 0;
	for (i = 0; i < n; i++) {
		for (j = 0; j < p; j++) {
			for (k = 0; k < m; k++) {
				sum += A[i][k] * B[k][j];
			}
			output[count] = sum;
			sum = 0;
			count++;
		}
	}

}
```

2. HLS_TEST

```
	printf("module\n");
	multiplyMatrices(matrix, result, ln, lm, lp);

	count = 0;
	printf("testbench\n");
	for (i = 0; i < n; i++) {
		for (j = 0; j < p; j++) {
			for (k = 0; k < m; k++) {
				sum += matrixA[i][k] * matrixB[k][j];
			}
			result_test[count] = sum;
			printf("%d\t", sum);
			sum = 0;
			count++;
		}
		printf("\n");
	}
```

```
for (count = 0; count < n * p; count++)
{
        if (result[count] != result_test[count])
        {
                printf("ERROR HW and SW results \n");
                return 1;
        }
}
printf("success HW and SW  result match \n");
```

2. SDK

<Matrix IP>

```
XMultiplymatrices_Initialize(&MultiplymatriceInstance_Ptr,
XPAR_XMULTIPLYMATRICES_0_DEVICE_ID);

XMultiplymatrices_Set_ln(&MultiplymatriceInstance_Ptr, ln);
XMultiplymatrices_Set_lm(&MultiplymatriceInstance_Ptr, lm);
XMultiplymatrices_Set_lp(&MultiplymatriceInstance_Ptr, lp);
```

<Timer IP>

```
Status = XTmrCtr_Initialize(&timer_dev, XPAR_AXI_TIMER_DEVICE_ID);

if (Status != XST_SUCCESS)
{
        print("Error: timer setup failed\r\n");
        //return XST_FAILURE;
}
XTmrCtr_SetOptions(&timer_dev, XPAR_AXI_TIMER_DEVICE_ID, XTC_ENABLE_ALL_OPTION);
```

<HW 시간측정 시작과 TX, RX 송수신>

```
XTmrCtr_Reset(&timer_dev, XPAR_AXI_TIMER_DEVICE_ID);
        begin_time = XTmrCtr_GetValue(&timer_dev, XPAR_AXI_TIMER_DEVICE_ID);
        Status = XAxiDma_SimpleTransfer(&AxiDma, (UINTPTR)TxBuffer, BYTES_TO_TRANSFER,
XAXIDMA_DMA_TO_DEVICE);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }


        Status = XAxiDma_SimpleTransfer(&AxiDma, (UINTPTR)RxBuffer, RESULT_ARRAY_TRANSFER,
XAXIDMA_DEVICE_TO_DMA);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }
        end_time = XTmrCtr_GetValue(&timer_dev, XPAR_AXI_TIMER_DEVICE_ID);
        run_time_hw = end_time - begin_time;
        xil_printf("\r HW Operation time : %d sec.\r\n",run_time_hw);
```

<SW 시간측정 시작과 SW 연산 함수 호출>

```
xil_printf("\rRunning Matrix Mult in SW\r\n");
        run_time_sw = Sw_matrixmul();
        xil_printf(" SW Operation time : %d sec. \r\n", run_time_sw);
```

<Sw_matrixmul()일부 코드와 SW시간 연산 방법>

```
        XTmrCtr_Reset(&timer_dev, XPAR_AXI_TIMER_DEVICE_ID);
        begin_time = XTmrCtr_GetValue(&timer_dev, XPAR_AXI_TIMER_DEVICE_ID);

        for (i = 0; i < n; i++) { // n = 1
                for (j = 0; j < p; j++) { // p =2
                        sum = 0;
                        for (k = 0; k < m; k++) { // m= 1
                                sum += matrixA[i][k] * matrixB[k][j];
                        }
                        result[j] = sum;
                        //xil_printf("%d \t", result[j]);
                }
                //xil_printf("\r\n");
```

```
}
end_time = XTmrCtr_GetValue(&timer_dev, XPAR_AXI_TIMER_DEVICE_ID);
run_time_sw = end_time - begin_time ;

return run_time_sw;
```

3. Discussion

1. HLS 부분 분석.

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
?	?	?	?	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	6	0	1075
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	4	-	0	0
Multiplexer	-	-	-	314
Register	-	-	1429	-
Total	4	6	1429	1389
Available	280	220	106400	53200
Utilization (%)	1	2	1	2

Detail

그림 13 Estimates

HLS 코드의 Estimate 값을 보면 timing constraints 는 10ns, Critical path delay는 8.510ns, clock uncertaintly 는 1.25ns가 나왔다. Flip flop은 1429개를 소모했으며, LUT 1389개, DSP48 6개 BRAM은 18개를 소모했다. HLS 코드에서 ln, lm, lp의 값이 미지수라 Latency 값이 ?로 출력된다.

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
137411	137411	137411	137411	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	3	0	394
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	4	-	0	0
Multiplexer	-	-	-	248
Register	-	-	421	-
Total	4	3	421	642
Available	280	220	106400	53200
Utilization (%)	1	1	~0	1

Detail

그림 14 ln, lp, lm을 5로 고정
Latency 값을 알기 위해 ln, lp, lm을 최대값(5)로 지정하고 다시 결과를 불러왔다. Latency min = 137411, Latency max = 137411, Interval = 137411, Interval = 137411 이다.

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	multiplyMatrices	return value
ap_rst	in	1	ap_ctrl_hs	multiplyMatrices	return value
ap_start	in	1	ap_ctrl_hs	multiplyMatrices	return value
ap_done	out	1	ap_ctrl_hs	multiplyMatrices	return value
ap_idle	out	1	ap_ctrl_hs	multiplyMatrices	return value
ap_ready	out	1	ap_ctrl_hs	multiplyMatrices	return value
input_r_address0	out	11	ap_memory	input_r	array
input_r_ce0	out	1	ap_memory	input_r	array
input_r_q0	in	32	ap_memory	input_r	array
output_r_address0	out	10	ap_memory	output_r	array
output_r_ce0	out	1	ap_memory	output_r	array
output_r_we0	out	1	ap_memory	output_r	array
output_r_d0	out	32	ap_memory	output_r	array
In	in	32	ap_none	In	scalar
Im	in	32	ap_none	Im	scalar
Ip	in	32	ap_none	Ip	scalar

Export the report(.html) using the Export Wizard

Open Analysis Perspective Analysis Perspective

Interface

Summary

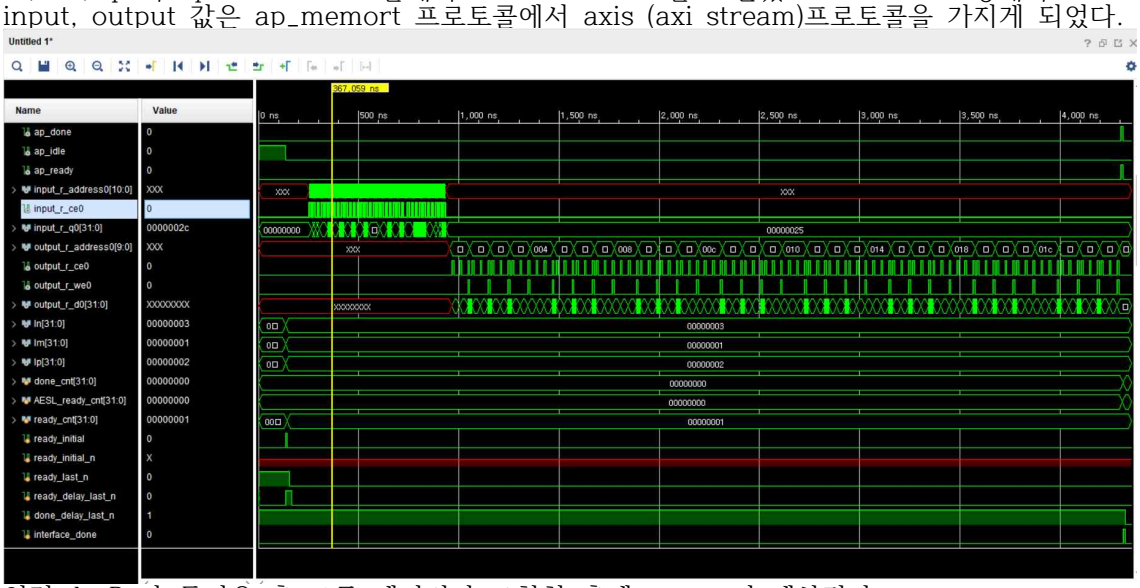
RTL Ports	Dir	Bits	Protocol	Source Object	C Type
s_axi_AXILiteS_AWVALID	in	1	s_axi	AXILiteS	scalar
s_axi_AXILiteS_AWREADY	out	1	s_axi	AXILiteS	scalar
s_axi_AXILiteS_AWADDR	in	6	s_axi	AXILiteS	scalar
s_axi_AXILiteS_WVALID	in	1	s_axi	AXILiteS	scalar
s_axi_AXILiteS_WREADY	out	1	s_axi	AXILiteS	scalar
s_axi_AXILiteS_WDATA	in	32	s_axi	AXILiteS	scalar
s_axi_AXILiteS_WSTRB	in	4	s_axi	AXILiteS	scalar
s_axi_AXILiteS_ARVALID	in	1	s_axi	AXILiteS	scalar
s_axi_AXILiteS_ARREADY	out	1	s_axi	AXILiteS	scalar
s_axi_AXILiteS_ARADDR	in	6	s_axi	AXILiteS	scalar
s_axi_AXILiteS_RVALID	out	1	s_axi	AXILiteS	scalar
s_axi_AXILiteS_RREADY	in	1	s_axi	AXILiteS	scalar
s_axi_AXILiteS_RDATA	out	32	s_axi	AXILiteS	scalar
s_axi_AXILiteS_RRESP	out	2	s_axi	AXILiteS	scalar
s_axi_AXILiteS_BVALID	out	1	s_axi	AXILiteS	scalar
s_axi_AXILiteS_BREADY	in	1	s_axi	AXILiteS	scalar
s_axi_AXILiteS_BRESP	out	2	s_axi	AXILiteS	scalar
ap_clk	in	1	ap_ctrl_hs	multiplyMatrices	return value
ap_rst_n	in	1	ap_ctrl_hs	multiplyMatrices	return value
ap_start	in	1	ap_ctrl_hs	multiplyMatrices	return value
ap_done	out	1	ap_ctrl_hs	multiplyMatrices	return value
ap_idle	out	1	ap_ctrl_hs	multiplyMatrices	return value
ap_ready	out	1	ap_ctrl_hs	multiplyMatrices	return value
input_r_TDATA	in	32	axis	input_r	pointer
input_r_TVALID	in	1	axis	input_r	pointer
input_r_TREADY	out	1	axis	input_r	pointer
output_r_TDATA	out	32	axis	output_r	pointer
output_r_TVALID	out	1	axis	output_r	pointer
output_r_TREADY	in	1	axis	output_r	pointer

Export the report(.html) using the Export Wizard

Open Analysis Perspective Analysis Perspective

그림 15 interface

인터페이스를 기본으로 세팅했을 때와 axi stream, axi lite 인터페이스를 가지게 했을 때를 비교해보면 다음과 같다. In, Im, Ip 가 ap_none 프로토콜에서 AXILITE 프로토콜로 변했고 RTL PORT 상에서도 AXI LITE에 통합되었다. input, output 값은 ap_memort 프로토콜에서 axis (axi stream)프로토콜을 가지게 되었다.



입력 A, B 가 들어온 후 모든 데이터가 도착한 후에 output이 계산된다.

2. SDK 부분 분석 (HW_SW 작동시간)

FPGA가 가지는 분명한 장점은 SW에서 CPU가 연산을 처리하는 것보다 빠른 속도로 계산을 끝마칠 수 있다는 것에 있다. AXI Timer를 이용하여 HW와 SW에서의 작동시간의 차이를 분석해본다. HW는 DMA에 데이터(input A, B)를 넣기 전부터 DMA에서 데이터를 받아온 후(output AB)를 측정한다. SW는 결과 값을 연산하는 3중 for문 연산에 걸리는 시간을 측정한다. 3중 for문의 데이터를 측정하는 이유는 입력 A, B 값을 불러오고 출력 AB를 내보내는 것이 HW연산에 걸리는 시간을 측정하는 방법과 유사하기 때문이다. 관련 코드는 10쪽 코드 설명에 나와있다.

```
--- Entering main() ---
Running Matrix Mult in HW+SW
================================================
0
HW Operation time : 1389 sec.
================================================
AXI DMA interrupt example test passed

Running Matrix Mult in SW
0
SW Operation time : 59 sec.
0
```

배열의 사이즈를 최소값 0, 0, 0으로 지정했을 때는 HW의 연산 시간이 1389로 SW의 59보다 오래걸렸다.
DMA로 보내고 받는 시간, 연산 시간이 회로로 고정되어 비슷한 시간이 걸리는 HW에 비해 SW는 받는 데이터와 연산 과정에 쓰인 시간이 데이터 크기에 따라 변한다. 따라서 이러한 차이를 보인 것으로 추측된다.

그림 18 배열 사이즈 n, m, p = 0, 0, 0

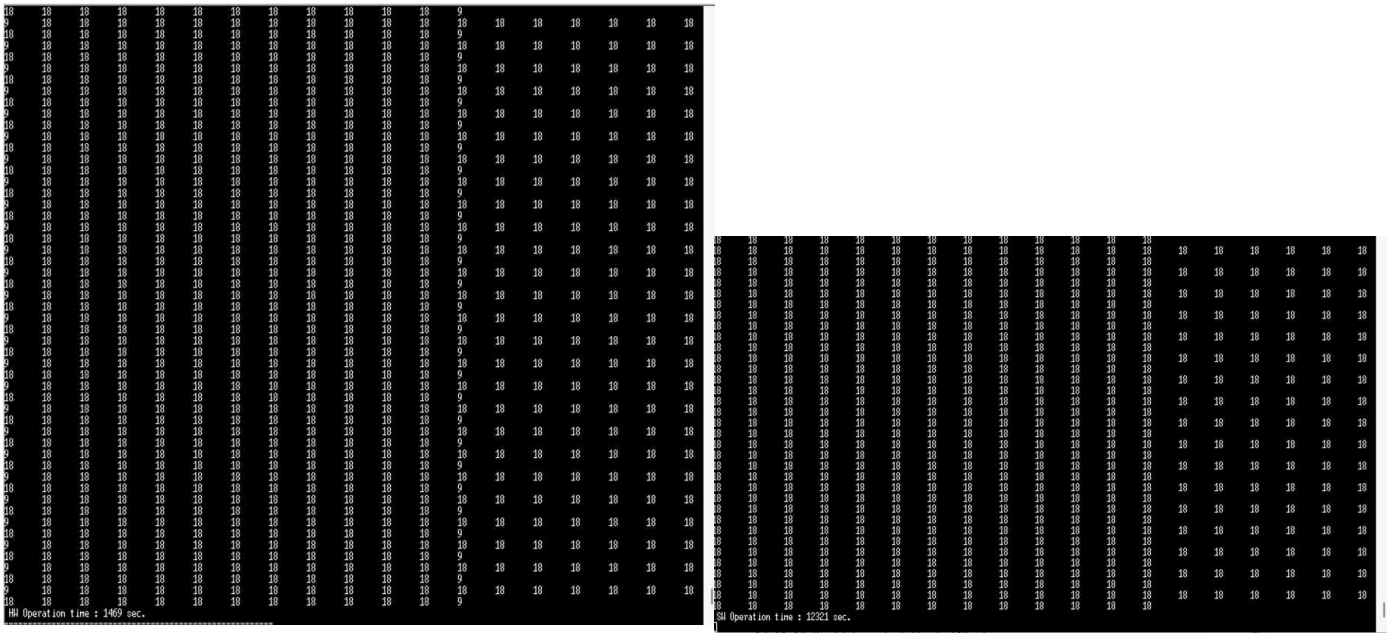


그림 19 배열 사이즈 5 1 5
이번에는 배열 사이즈를 n = 5, m = 1, p = 5로 두고 계산해 보았다. HW는 1469, SW는 12321이 도출되었다. HW는 이전 0, 0, 0 때와 비슷한 시간이 걸린 반면 SW는 연산량이 많아지자 급격히 걸리는 시간이 증가한 것이 관찰되었다.

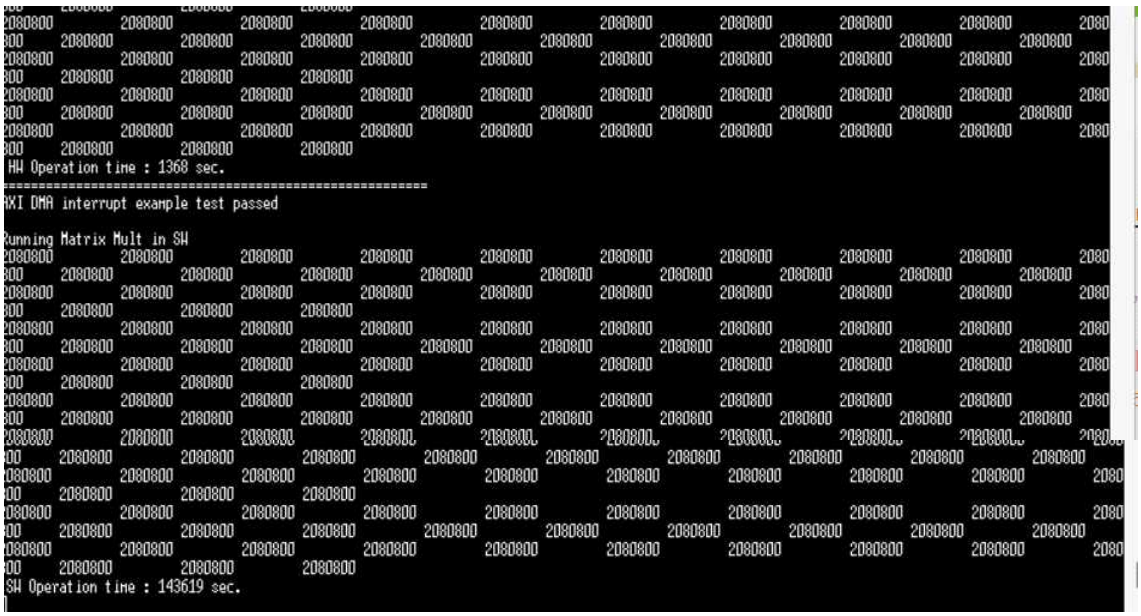


그림 22 배열 사이즈 5, 5, 5
배열 사이즈를 최대인 5, 5, 5로 두고 계산 했을 때 HW는 1368, SW는 143619가 도출되었다. 이번에도 HW의 연산시간은 별 차이가 없는 반면 SW는 매우 큰폭으로 증가하고 있다.

이것으로 봤을 때, 많은 연산을 요구하는 것일수록 CPU가 SW로 처리하기 보다는 FPGA를 사용하여 맞춤 회로를 구성하고 DMA를 이용하여 CPU 자원에 의존하지 않는 편이 긍정적이다.

3. SDK 부분 분석 (비트에 따른 표현가능한 수의 범위)

입력 A, B는 unsigned char 형으로 표현 가능한 수의 범위가 2^8 까지이다. 그러나 덧셈이나 곱셈 연산을 하게 되면 unsigned char형 8 bit를 넘어가는 문제가 생길 수 있다. 따라서 출력 AB는 unsigned int 형으로 32bit 까지 수를 표현할 수 있게 해주었다.

그림 22를 보면 입력 A, B의 요소에 모두 255로 채워주고 계산 했을 때 2080800이 출력된다. 2080800은 unsigned char형으로 표현할 수 없으며 unsigned int 형을 이용해야 overflow를 방지할 수 있다.

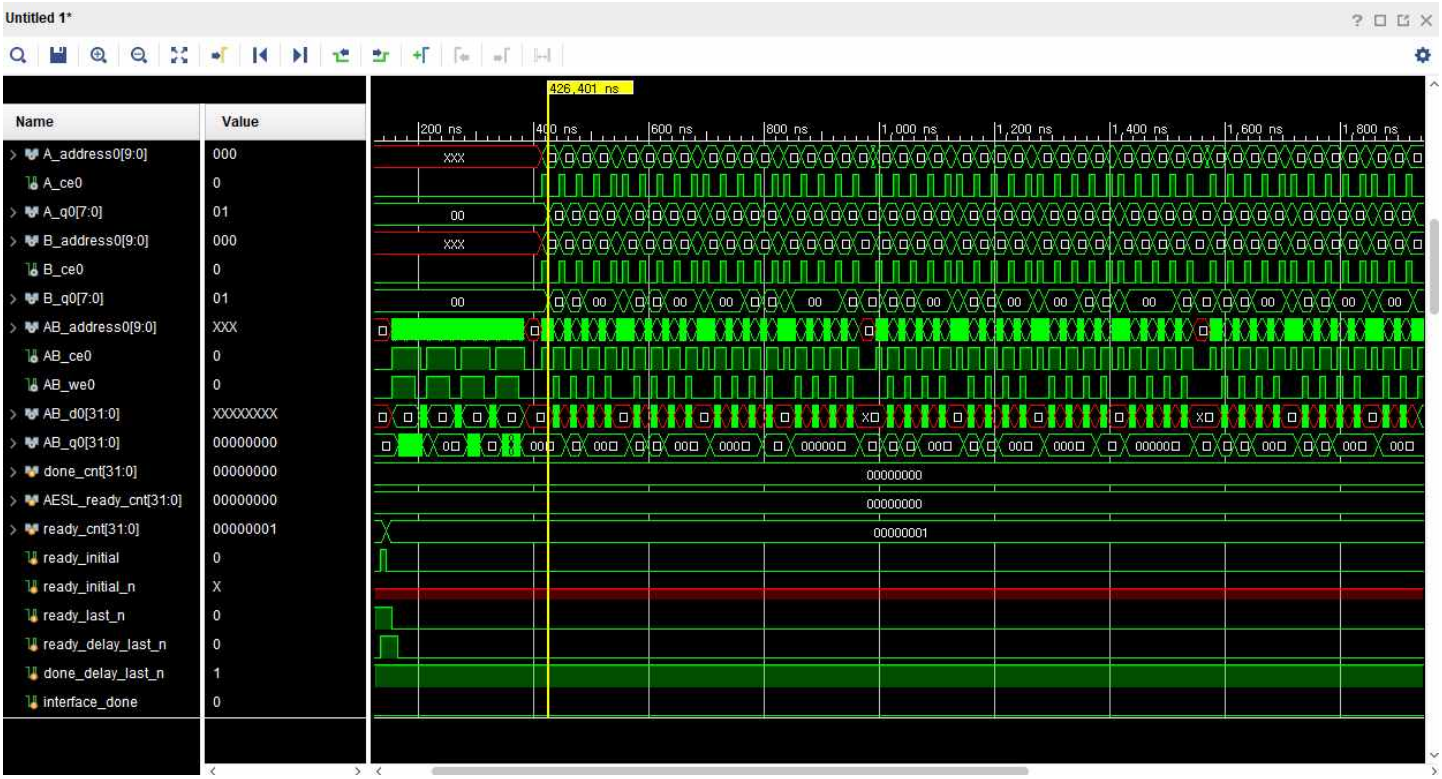
4. 발생했던 오류들과 해결방법.

1. IP에 들어오는 입력은 하나의 포트를 이용해야 한다.

처음 HLS로 IP 설계시

```
void matrixmul(unsigned int lm, unsigned int ln, unsigned int lp, mat_a_t A[MAX][MAX], mat_b_t B[MAX][MAX], result_t AB[MAX][MAX]) {
#pragma HLS INTERFACE axis register both port=AB
#pragma HLS INTERFACE axis register both port=B
#pragma HLS INTERFACE axis register both port=A
#pragma HLS INTERFACE s_axilite port=lp
#pragma HLS INTERFACE s_axilite port=ln
#pragma HLS INTERFACE s_axilite port=lm
```

위와 같이 입력 A, B를 각각 axi stream을 가지게 하였다. 입력 A, B를 DMA 2개를 구성해 입력을 받게 하고 IP를 제어하다보니 타이밍이 맞지 않아, SDK에서 연산 결과 값이 모두 0으로 출력되었다.



또한 A, B가 모두 들어온 후에 연산이 시작되는 것이 아니라 A, B의 입력을 받는 중에 AB연산도 같이 시작되었다. test bench에서는 통과하였지만, wave를 검사하고 sdk에서 검증할 때는 제대로 작동하지 않았다.

따라서 HLS에서의 문제가 인풋이 나누어져 들어오는 것이 원인이라 생각하고 코드를 수정하였다.
두 번째 수정에서는 HLS에 인풋을 하나로 받기 위해서 구조체를 사용하였다.

```
typedef struct Input_Matrix_Mul
{
        mat_a_t A[MAX][MAX];
        mat_b_t B[MAX][MAX];
```

} INPUT_MATRIX_MUL;

void matrixmul(unsigned int lm, unsigned int ln, unsigned int lp, INPUT_MATRIX_MUL input, result_t AB[MAX][MAX])
{
#pragma HLS INTERFACE axis register both port=AB
#pragma HLS INTERFACE axis register both port=input
#pragma HLS INTERFACE s_axilite port=lp
#pragma HLS INTERFACE s_axilite port=ln
#pragma HLS INTERFACE s_axilite port=lm

그러나 HLS에서 IP로 합성 시 #pragma HLS INTERFACE axis register both port=input 코드 상에서는 하나의 input이더라도 구조체를 사용하면 IP 상에서는 입력이 두 개로 쪼개지게 된다.

따라서 마지막 시도에서는

```
for (int i = 0; i < n; i++)
	{
		for (int j = 0; j < m; j++)
		{
			A[i][j] = input[count];
			count++;
		}
	}
	// matrixB에 변수 할당
	for (int i = 0; i < m; i++)
	{
		for (int j = 0; j < p; j++)
		{
			B[i][j] = input[count];
			count++;
		}
	}

	count = 0;
	for (i = 0; i < n; i++) {
		for (j = 0; j < p; j++) {
			for (k = 0; k < m; k++) {
				sum += A[i][k] * B[k][j];
			}
			output[count] = sum;
			sum = 0;
			count++;
		}
	}

}
```

코드 안에서 void **multiplyMatrices**(unsigned int input[2*MAX], unsigned int output[MAX], int ln, int lm, int lp)
사이즈가 2배인 인풋을 가지게 하고 코드 안에서 A, B의 사이즈를 분석해 요소들을 알고리즘 상에서 집어넣는 방식을 사용하였다.

2. SDK 상에서 값이 밀려서 출력되는 현상.

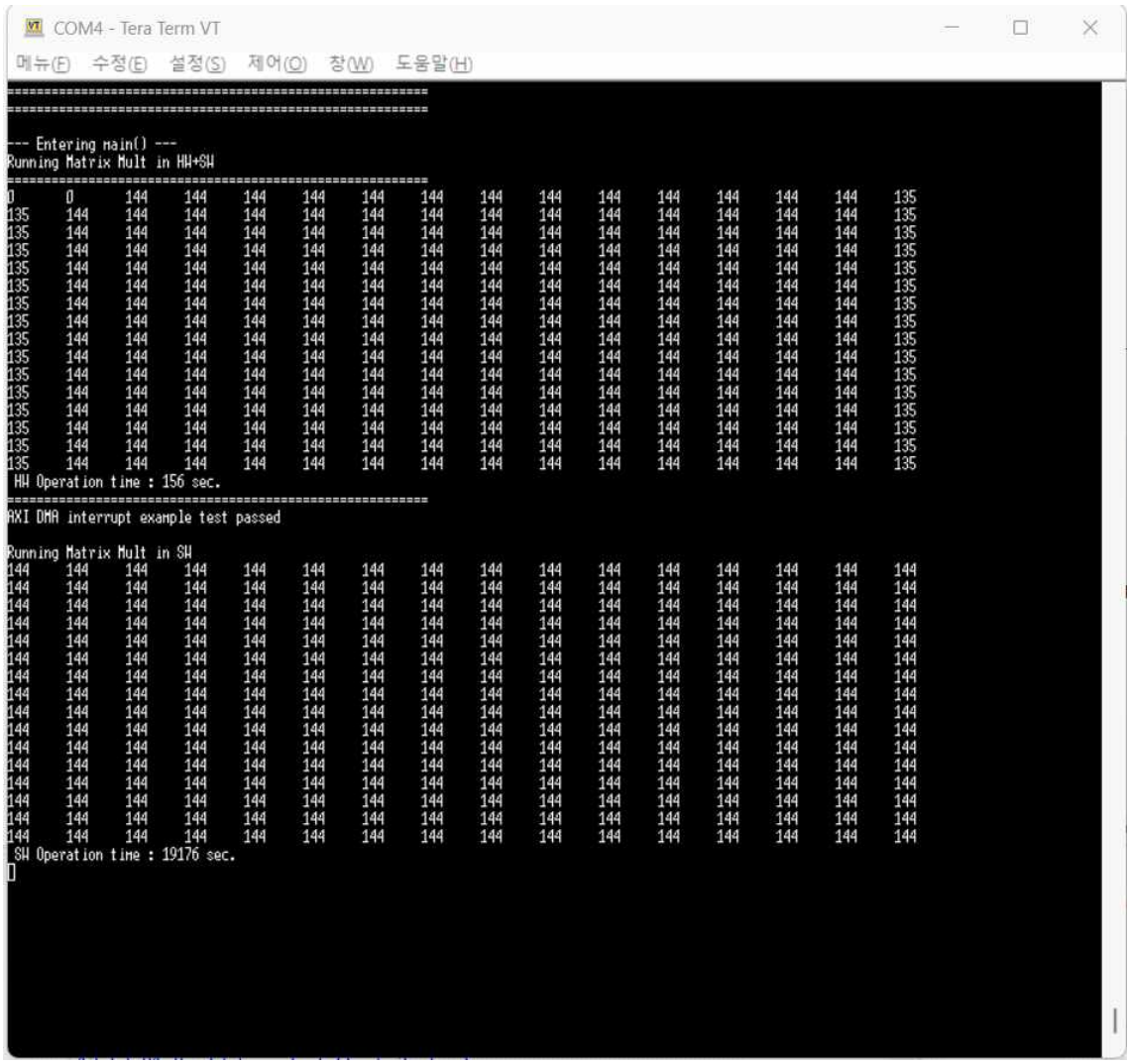
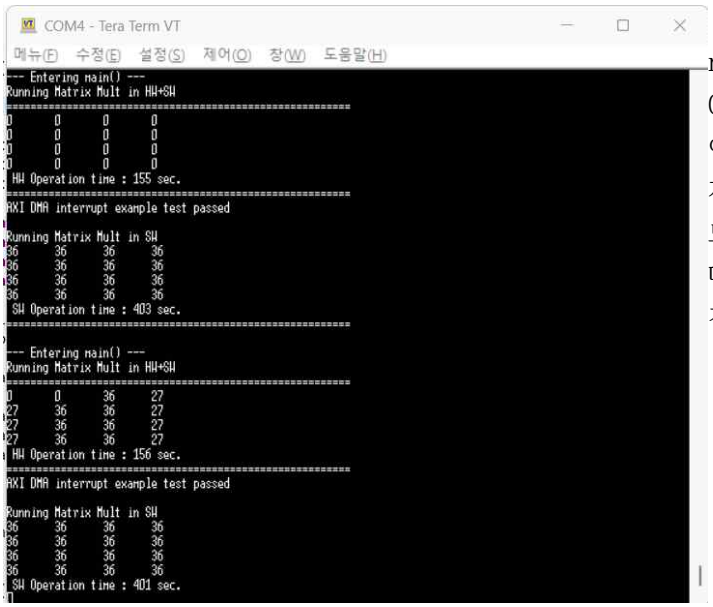


그림 24 값이 밀려서 들어옴.
사진을 보면 SW의 결과와 비교했을 때 첫 두자리가 0, 0 으로 쓰레기 값으로 시작한다. 또한 끝에서 135, 135가 출력되는 것을 보아, 인풋이 들어올 때 비트가 밀리는 것으로 추측된다. 추측하는 원인은 두 가지로 인풋값이 적힌 어드레스에서 실제로 2칸 위부터 잘못 입력 받고 있는 경우, 혹은 타이밍이 어긋나거나 원인 미상의 이유가 있는 경우이다. 따라서 SDK 코드 상에서 딜레이 함수를 이용하여 딜레이를 주어보았을 때에도 같은 현상이 발생 하는 것으로 보아, 잘못된 어드레스에서 읽고 있는 것으로 추측된다.

3. SDK 상에서 값이 모두 0으로 출력되는 현상.



다음 사진을 보면 처음에 모두 0이 출력된다. 또 두 번째 run을 시켰을 때는 위의 오류와 마찬가지로 앞의 두 자리가 0으로 채워지는데, 위에서 추측했던 것처럼 잘못된 어드레스에서 읽고 있다고 생각된다. 이유는 run을 돌리면 돌릴수록 계산 결과값이 0으로 모두 출력되는 현상이 더 빈번해진다. 또한 fpga를 전원을 꺼 초기화 시킨 후 처음 run을 시켰을 때는 0이 입력되는 것이 아니라 HW, SW의 값이 일치하는 것을 확인할 수 있기 때문이다.

4. TIMER 시간 측정시 주의 점들.

HW와 SW의 동작시간을 비교할 때에는 최대한 측정환경이 비슷하게 만들어 주어야 한다.

Xil_printf()함수는 timer 측정에 들어가지 않도록 해주어야 한다. xil_printf 함수는 꽤 많은 사이클을 잡아먹기 때문이다. xil_printf()와 같이 다른 함수들이 시간 측정할 때 끼어들지 않도록 만들어야 하며, 최대한 DMA에서 발생하는 시간을 측정하도록 노력해 주어야 한다. 그렇게 하지 않으면 HW의 시간이 더 크게 측정되는 현상이 발생하는데, 오차가 꽤 크게 측정되기 때문이다.

따라서

```
XTmrCtr_Reset(&timer_dev, XPAR_AXI_TIMER_DEVICE_ID);
	begin_time = XTmrCtr_GetValue(&timer_dev, XPAR_AXI_TIMER_DEVICE_ID);
	Status = XAxiDma_SimpleTransfer(&AxiDma, (UINTPTR)TxBuffer, BYTES_TO_TRANSFER,
XAXIDMA_DMA_TO_DEVICE);
	if (Status != XST_SUCCESS) {
		return XST_FAILURE;
	}


	Status = XAxiDma_SimpleTransfer(&AxiDma, (UINTPTR)RxBuffer, RESULT_ARRAY_TRANSFER,
XAXIDMA_DEVICE_TO_DMA);
	if (Status != XST_SUCCESS) {
		return XST_FAILURE;
	}
	end_time = XTmrCtr_GetValue(&timer_dev, XPAR_AXI_TIMER_DEVICE_ID);
	run_time_hw = end_time - begin_time;
	xil_printf("\r HW Operation time : %d sec.\r\n",run_time_hw);
```

HW 시간 측정시에는 DMA에 TX 버퍼를 보내고 RX버퍼를 받아오는 시간만 측정해 주었다.

```
XTmrCtr_Reset(&timer_dev, XPAR_AXI_TIMER_DEVICE_ID);
	begin_time = XTmrCtr_GetValue(&timer_dev, XPAR_AXI_TIMER_DEVICE_ID);

	for (i = 0; i < n; i++) { // n = 1
		for (j = 0; j < p; j++) { // p =2
			sum = 0;
			for (k = 0; k < m; k++) { // m= 1
				sum += matrixA[i][k] * matrixB[k][j];
			}
			result[j] = sum;
			//xil_printf("%d \t", result[j]);
		}
		//xil_printf("\r\n");
	}
	end_time = XTmrCtr_GetValue(&timer_dev, XPAR_AXI_TIMER_DEVICE_ID);
	run_time_sw = end_time - begin_time ;

	return run_time_sw;
```

SW에서는 3중 for문을 돌며 SW의 결과값을 연산하는 것만 측정하였는데 이는 포문을 돌며 matrixA와 matrixB에 접근할 때 cpu에서 메모리에 접근하여 값을 받아오는 것이 DMA에서 TX를 보내고 IP에서 읽는 것과 같으며, result배열에 sum 변수값을 넣는 것이 IP에서 연산하고 DMA에서 받아오는 것과 비슷하다고 생각이 들었기 때문이다.

4. 결론

DMA와 HLS IP를 이용하여 행렬 곱을 연산해보는 실습을 진행하였다. 오버플로우를 방지하기 위해 연산 결과를 저장하기 위해서는 입력보다 큰 bit를 저장할 수 있도록 해야한다. HLS에서 IP를 만들 때는 test bench를 통해 golden data를 비교해보는 작업이 필요하다. 추가로 합성후 HW/SW cosimulation을 통해 파형을 보고 HLS 코드가 잘 작동하는지도 모니터링 해야한다. DMA를 이용하고 axi stream을 이용할 때는 input 포트를 최대한 하나만 만들도록 하는 것이 유리하다. 또한 Axi timer 사용법을 익혔으며 HW와 SW를 비교할 때는 최대한 간섭하는 요소와, 오차를 배제하도록 노력해야한다.