Lab 6 Report

Jeon Woongbae, Park Sangbeen

UNIST

EE32601

PYNQ Access : `http://tpugroupd.iptime.org:777`, PW : xilinx

1 Optimization

Below is the implementation and optimization approach.

1.1 Implementation Detail

Implemented source code is available at 5.1.

Overall flow of code is listed as below.

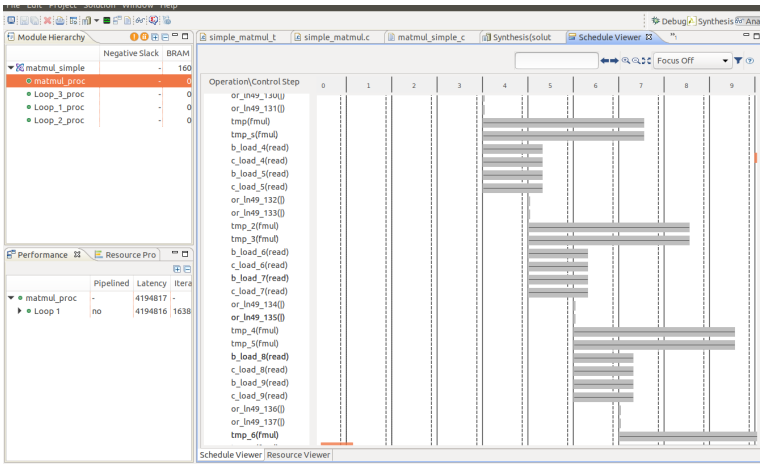
- Receive input via import_input, store them in buffers.
  This is similar to the previous SAXPY project.
- Perform matmul via matmul_proc.
  Matrix multiplication in matmul_proc is written in a way that maximizes cache hit rate.
- Export output via export_output, which transfers a buffer into axis format.

1.2 Optimization Detail

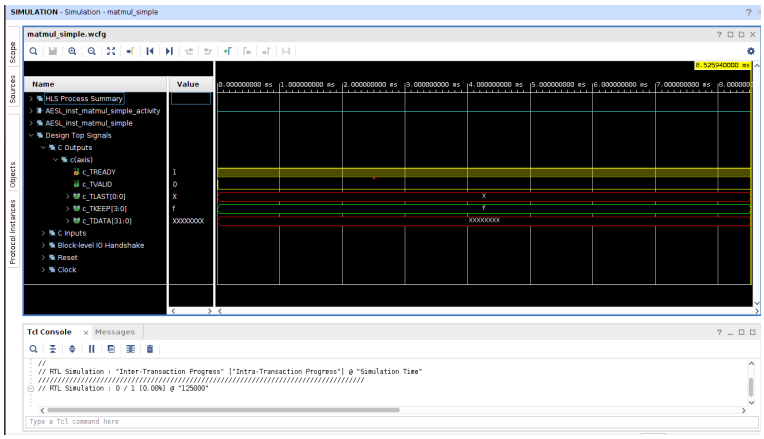
Matrix multiplication is written in a way that maximizes cache hit rate.
Naive approaches compute multiplication in i, j, k index order.
However, cache friendly way takes k, i, j index order.
Cache friendly method is about 10 times faster than the naive method.
Please refer to the matmul_proc in 5.1.

1.3 Simulation Results

Below is the C simulation result.



Below is the Cosimulation result.



Below is the C synthesis results.

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.372 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
532995	532995	5.330 ms	5.330 ms	524546	524546	dataflow

Detail

Instance

Loop

Utilization Estimates

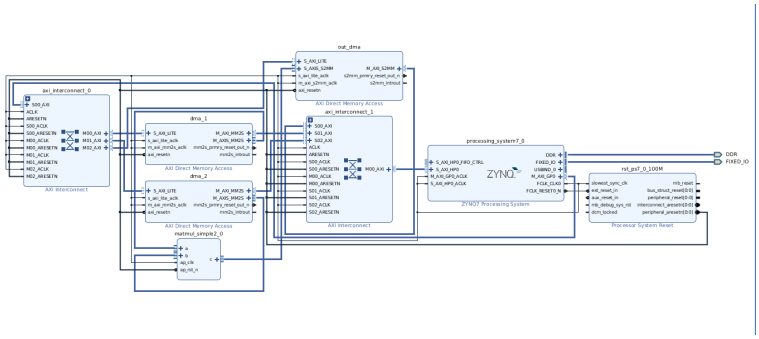
Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2	-
FIFO	-	-	-	-	-
Instance	-	10	4348	5283	-
Memory	40	-	0	0	0
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	40	10	4348	5285	0
Available	280	220	106400	53200	0
Utilization (%)	14	4	4	9	0

Detail

Below is the top diagram of this matmul IP.

For C [64][64], A [64][128], B [128][64], HW is faster.



[16]:
```
%%time
np.matmul(a,b)
```

```
CPU times: user 10 ms, sys: 2 µs, total: 10 ms
Wall time: 8.15 ms
```

[16]:
```
array([[30.49638689, 31.37053253, 30.59561202, ..., 28.46146702,
        30.01720131, 29.46309113],
       [32.37445954, 33.68166486, 34.11580266, ..., 31.17309627,
        34.05335407, 31.89066384],
       [32.66588963, 33.14602452, 31.12623184, ..., 28.2857328 ,
        32.85944191, 30.5411617 ],
       ...,
       [33.76716411, 33.21389784, 31.4241097 , ..., 30.5353657 ,
        33.07130232, 31.5157654 ],
       [33.07990562, 33.53627194, 32.88372353, ..., 30.03490454,
        33.39403257, 33.55540399],
       [32.11478557, 34.73122557, 31.33148286, ..., 29.36215169,
        31.67757991, 31.87542109]])
```

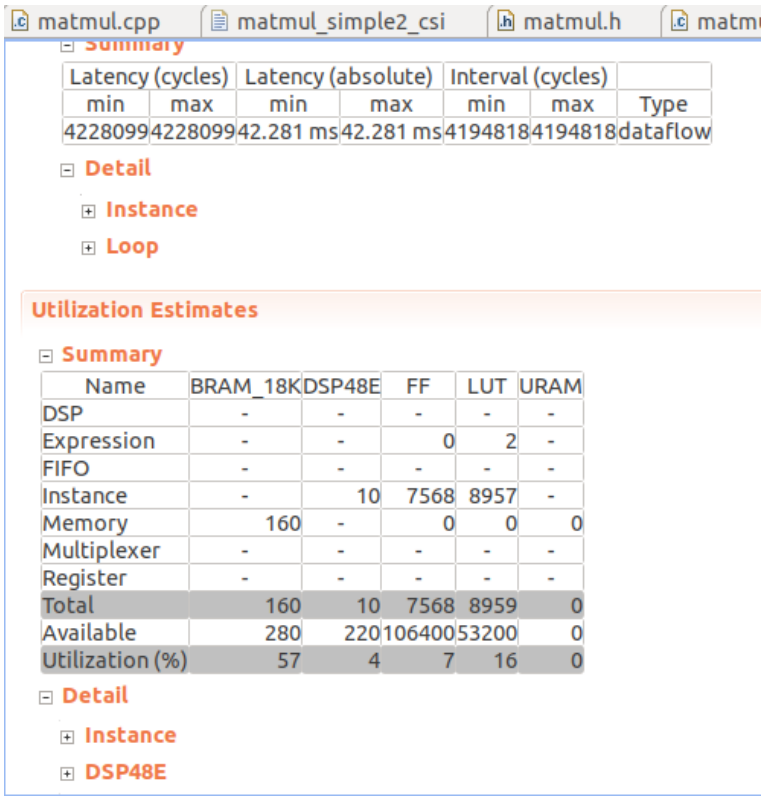
[17]:
```
%%time
dma1.sendchannel.transfer(abuf)
dma2.sendchannel.transfer(bbuf)
out_dma.recvchannel.transfer(obuf)
dma1.sendchannel.wait()
dma2.sendchannel.wait()
out_dma.recvchannel.wait()
```

```
CPU times: user 7.21 ms, sys: 0 ns, total: 7.21 ms
Wall time: 7.04 ms
```

1.4 Difference when size is 128

When size is doubled to 128, the amount of BRAM needed to fully synthesize the whole system including Zynq processor, DMAs, and the matmul IP is 320.

This exceeds the size of BRAM offered by the Pynq board, which is 280.
Although the BRAM needed to synthesize the IP is only 160, additional components like DMAs take extra amount of BRAM usage for themselves and increase the BRAM utilization.


matmul.cpp | matmul_simple2_csi | matmul.h | matm

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
4228099	4228099	42.281 ms	42.281 ms	4194818	4194818	dataflow

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2	-
FIFO	-	-	-	-	-
Instance	-	10	7568	8957	-
Memory	160	-	0	0	0
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	160	10	7568	8959	0
Available	280	220	106400	53200	0
Utilization (%)	57	4	7	16	0

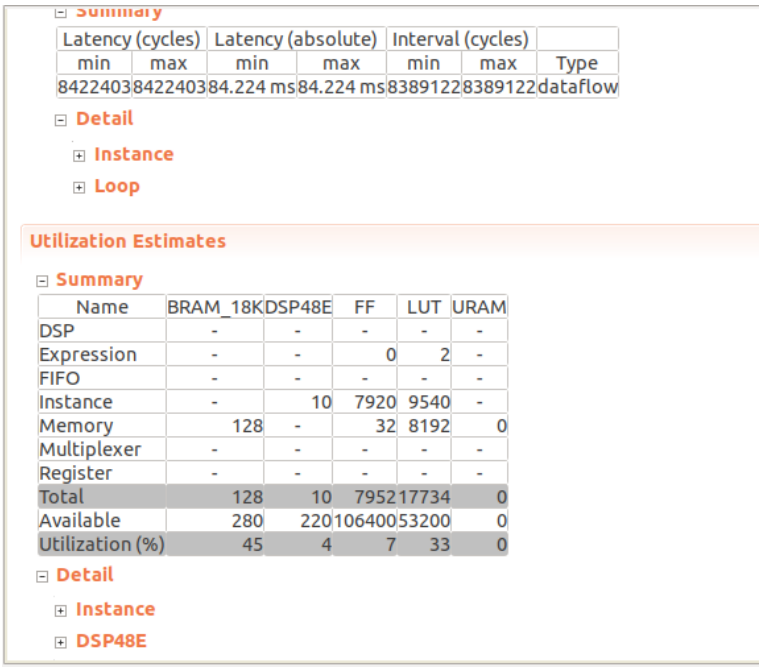
Detail

Instance

DSP48E

Inorder to reduce the BRAM utilization of matmul IP, pragma HLS resource variable=a core=RAM_1P_LUTRAM can be used.
This is a HLS pragma that forces the HLS to implement arrays in LUTs and make it as a LUTRAM, instead of the default option which is BRAM.

This pragma does reduce the BRAM utilization.
This is the result of replacing C[128][128] into LUTRAM as shown in the figure below.



Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
8422403	8422403	84.224 ms	84.224 ms	8389122	8389122	dataflow

Detail

Instance

Loop

Utilization Estimates

Summary

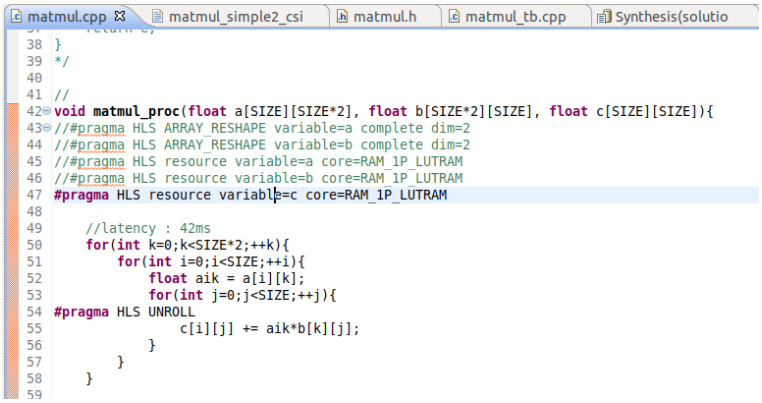
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2	-
FIFO	-	-	-	-	-
Instance	-	10	7920	9540	-
Memory	128	-	32	8192	0
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	128	10	7952	17734	0
Available	280	220	106400	53200	0
Utilization (%)	45	4	7	33	0

Detail

Instance

DSP48E

However, this leads to a latency increase.



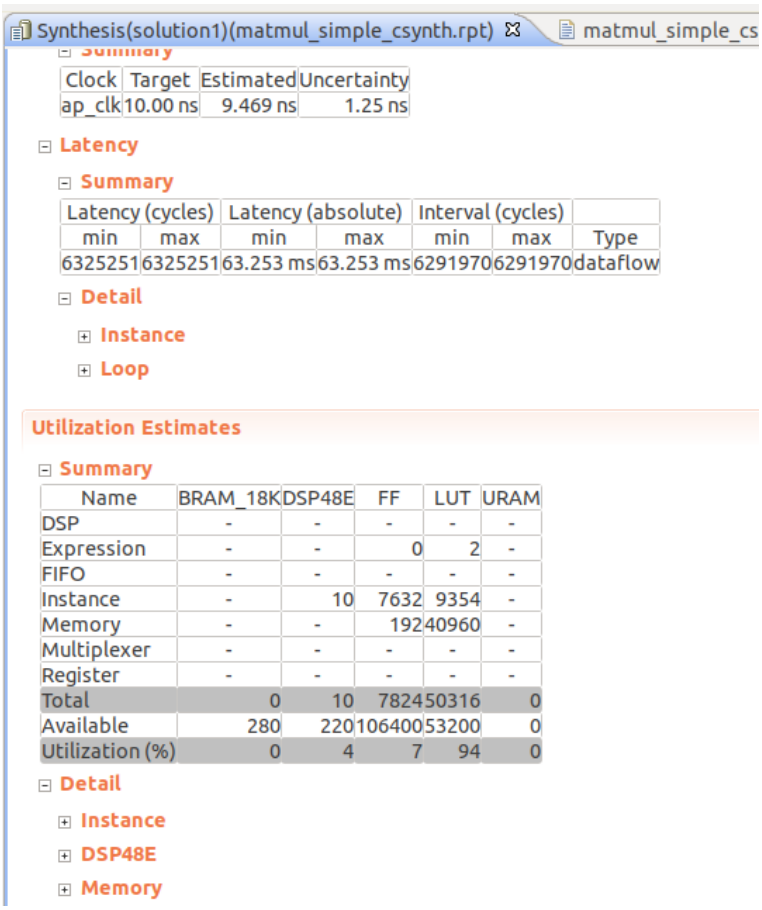
matmul.cpp | matmul_simple2_csi | matmul.h | matmul_tb.cpp | Synthesis(solutio

```
38 }
39 */
40
41 //
42 void matmul_proc(float a[SIZE][SIZE*2], float b[SIZE*2][SIZE], float c[SIZE][SIZE]){
43 //#pragma HLS ARRAY_RESHAPE variable=a complete dim=2
44 //#pragma HLS ARRAY_RESHAPE variable=b complete dim=2
45 //#pragma HLS resource variable=a core=RAM_1P_LUTRAM
46 //#pragma HLS resource variable=b core=RAM_1P_LUTRAM
47 #pragma HLS resource variable=c core=RAM_1P_LUTRAM
48
49     //latency : 42ms
50     for(int k=0;k<SIZE*2;++k){
51         for(int i=0;i<SIZE;++i){
52             float aik = a[i][k];
53             for(int j=0;j<SIZE;++j){
54 #pragma HLS UNROLL
55                 c[i][j] += aik*b[k][j];
56             }
57         }
58     }
59
```

This is due to the LUTRAM usage instead of BRAM.
Because of the latency increase, for size 128 case, HW is slower than SW.

```
[16]: %%time
      np.matmul(a,b)

      CPU times: user 31.3 ms, sys: 1.21 ms, total: 32.5 ms
      Wall time: 27.8 ms

[16]: array([[64.96428756, 66.91436225, 66.73916609, ..., 73.14234228,
              61.38856116, 67.89229004],
             [61.60006931, 68.22690556, 64.1571937 , ..., 70.5851868 ,
              61.74231791, 65.88577291],
             [60.61420992, 62.27654513, 63.51443211, ..., 69.02459687,
              60.39315881, 65.34860171],
             ...,
             [61.15406102, 64.15427857, 63.86326607, ..., 67.36606392,
              62.10560524, 64.95093586],
             [60.73866967, 62.68358155, 61.97021216, ..., 67.71417621,
              59.69572674, 64.8889588 ],
             [59.62176912, 62.41084906, 62.90943882, ..., 67.38605054,
              59.43076064, 65.29316386]])

[17]: %%time
      dma1.sendchannel.transfer(abuf)
      dma2.sendchannel.transfer(bbuf)
      out_dma.recvchannel.transfer(obuf)
      dma1.sendchannel.wait()
      dma2.sendchannel.wait()
      out_dma.recvchannel.wait()

      CPU times: user 78.8 ms, sys: 9.87 ms, total: 88.6 ms
      Wall time: 86.1 ms
```

Below is the result when all the buffers are replaced with LUTRAM.



Synthesis(solution1)(matmul_simple_csynth.rpt) matmul_simple_cs

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	9.469 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
6325251	6325251	63.253 ms	63.253 ms	6291970	6291970	dataflow

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2	-
FIFO	-	-	-	-	-
Instance	-	10	7632	9354	-
Memory	-	-	192	40960	-
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	0	10	7824	50316	0
Available	280	220	106400	53200	0
Utilization (%)	0	4	7	94	0

Detail

Instance

DSP48E

Memory

2 Python Programming on Pynq

This section briefly explains about key points in Python Pynq programming.
Check more details in the submitted Jupyter Notebook,
or access our Pynq board via here : `http://tpugroupd.iptime.org:777`.

2.1 Overlay Driver Source Code

Pynq Overlay driver for the matmul IP is available at 5.3.

2.2 Flatten 2D Arrays

Original inputs are 2D arrays. However, in order to cast this input into a format that supports AXI Streraming, 2D arrays are flatten into 1D arrays and transferred into each DMAs.

2.3 Use of DMA

Matmul IP receives its input A and B from two DMAs and returns its output C with a single DMA.
Matmul IP transfers its dynamically allocated data array into both DMAS, set an output_dma as a receiving channel, and issue wait() for all 3 DMAs.

3 Bit Stream, Jupyter Notebook, GCD Vivado IP

Bit stream, Jupyter Notebook, GCD Vivado IP are submitted via BB.
Please refer to the files submitted.

4 Contribution

Jeon Woongbae and Park Sangbeen equally contributed.

5 Source Codes

5.1 matmul cpp

Below is the matmul.cpp code.

```
#include "matmul.h"

/*
float pop_stream(AXI_F const &e) // type casting AXI_F -> int
{
#pragma HLS INLINE

    union conv {
        float f;
        unsigned int u;
    };

    conv tmp;
    tmp.u = e.data;

    float ret = tmp.f;
    volatile ap_uint<4> keep = e.keep; // -1 : 1111
        volatile ap_uint<1> last = e.last; // only asserted at the last element
        return ret;
}

AXI_F push_stream(float const ret, int last) // type casting AXI_F -> int
{
#pragma HLS INLINE
    union conv {
        float f;
        unsigned int u;
    };

    conv tmp;
    tmp.f = ret;

    AXI_F e;
    e.data = tmp.u;
    e.keep = -1;            // 1111
    e.last = last ? 1 : 0; // 1 when last
        return e;
```

```
}
*/

//
void matmul_proc(float a[SIZE][SIZE*2], float b[SIZE*2][SIZE], float c[SIZE][SIZE]){
//#pragma HLS ARRAY_RESHAPE variable=a complete dim=2
//#pragma HLS ARRAY_RESHAPE variable=b complete dim=2
//#pragma HLS resource variable=a core=RAM_1P_LUTRAM
//#pragma HLS resource variable=b core=RAM_1P_LUTRAM
//#pragma HLS resource variable=c core=RAM_1P_LUTRAM

	//latency : 42ms
	for(int k=0;k<SIZE*2;++k){
		for(int i=0;i<SIZE;++i){
			float aik = a[i][k];
			for(int j=0;j<SIZE;++j){
#pragma HLS UNROLL
				c[i][j] += aik*b[k][j];
			}
		}
	}

	/*
	// latency : 0.4 s
	for(int i=0;i<SIZE;++i){
		for(int j=0;j<SIZE;++j){
			float ctemp = 0;
			for(int k=0;k<SIZE*2;++k){
#pragma HLS UNROLL
				ctemp += a[i][k]*b[k][j];
			}
			c[i][j] = ctemp;
		}
	}
	*/

}

void export_output(AXI_F *c, float cbuf[SIZE][SIZE]){
//#pragma HLS INLINE
	for(int i=0;i<SIZE;++i){
		for(int j=0;j<SIZE;++j){
```

```
                        int idx = i*SIZE + j;
                        c[idx] = push_stream<float>(cbuf[i][j], idx == (SIZE*SIZE -
                }
        }
}

template< int row, int col>
void import_input(AXI_F *x, float xbuf[row][col]){
//#pragma HLS INLINE
        for(int i=0;i<row;++i){
                for(int j=0;j<col;++j){
                        int idx = i*col + j;
                        xbuf[i][j] = pop_stream<float>(x[idx]);
                }
        }
}

void matmul_simple2(AXI_F *a, AXI_F *b, AXI_F *c){
#pragma HLS INTERFACE axis port=a depth = 32768
#pragma HLS INTERFACE axis port=b depth = 32768
#pragma HLS INTERFACE axis port=c depth = 32768
//#pragma HLS INTERFACE s_axilite port=return bundle=CTRL_BUS
#pragma HLS INTERFACE ap_ctrl_none port=return

#pragma HLS DATAFLOW
        float a_buf[SIZE][SIZE*2], b_buf[SIZE*2][SIZE], c_buf[SIZE][SIZE];

        import_input<SIZE,SIZE*2>(a,a_buf);
        import_input<SIZE*2,SIZE>(b,b_buf);

        matmul_proc(a_buf,b_buf,c_buf);
        export_output(c,c_buf);
}



/*
#include "simple_matmul.h"

float pop_stream(AXI_F const &e) // type casting AXI_F -> int
{
#pragma HLS INLINE
```

```
    union conv {
        float f;
        unsigned int u;
    };

    conv tmp;
    tmp.u = e.data;

    float ret = tmp.f;
    volatile ap_uint<4> keep = e.keep; // -1 : 1111
        volatile ap_uint<1> last = e.last; // only asserted at the last element
        return ret;
}

AXI_F push_stream(float const ret, int last) // type casting AXI_F -> int
{
#pragma HLS INLINE
    union conv {
        float f;
        unsigned int u;
    };

    conv tmp;
    tmp.f = ret;

    AXI_F e;
    e.data = tmp.u;
    e.keep = -1;            // 1111
    e.last = last ? 1 : 0; // 1 when last
        return e;
}

void matmul_proc(float a[SIZE][SIZE*2], float b[SIZE*2][SIZE], float c[SIZE][SIZE]){
#pragma HLS ARRAY_RESHAPE variable=a complete dim=2
#pragma HLS ARRAY_RESHAPE variable=b complete dim=2

#pragma HLS DATAFLOW

        float a_buf[SIZE][SIZE*2];
#pragma HLS ARRAY_RESHAPE variable=a_buf complete dim=2
        float b_buf[SIZE*2][SIZE];
```

```
#pragma HLS ARRAY_RESHAPE variable=b_buf complete dim=2
        float c_buf[SIZE][SIZE];

        for(int i=0;i<SIZE;++i){
                for(int j=0;j<SIZE*2;++j){
#pragma HLS UNROLL factor = 4
                        a_buf[i][j] = a[i][j];
                }
        }

        for(int i=0;i<SIZE*2;++i){
                for(int j=0;j<SIZE;++j){
#pragma HLS UNROLL factor = 4
                        b_buf[i][j] = b[i][j];
                }
        }

        for(int i=0;i<SIZE;++i){
                for(int j=0;j<SIZE;++j){
#pragma HLS PIPELINE II=256
                        float c_tmp = 0;
                        for(int k=0;k<SIZE*2;++k){
#pragma HLS UNROLL factor = 4
                                c_tmp += a_buf[i][k]*b_buf[k][j];
                        }
                        c_buf[i][j] = c_tmp;
                }
        }

        for(int i=0;i<SIZE;++i){
                for(int j=0;j<SIZE;++j){
#pragma HLS UNROLL factor = 4
                        c[i][j] = c_buf[i][j];
                }
        }

}

void export_output(AXI_F *c, float cbuf[SIZE][SIZE]){
#pragma HLS INLINE
        for(int i=0;i<SIZE;++i){
                for(int j=0;j<SIZE;++j){
```

```
#pragma HLS PIPELINE II=4
                        int idx = i*SIZE + j;
                        c[idx] = push_stream(cbuf[i][j], idx == (SIZE*SIZE - 1));
                }
        }
}

template< int row, int col>
void import_input(AXI_F *x, float xbuf[row][col]){
#pragma HLS INLINE
        for(int i=0;i<row;++i){
                for(int j=0;j<col;++j){
#pragma HLS PIPELINE II=4
                        int idx = i*col + j;
                        xbuf[i][j] = pop_stream(x[idx]);
                }
        }
}

void matmul_simple(AXI_F *a, AXI_F *b, AXI_F *c){
#pragma HLS INTERFACE axis port=a
#pragma HLS INTERFACE axis port=b
#pragma HLS INTERFACE axis port=c
//#pragma HLS INTERFACE s_axilite port=return bundle=CTRL_BUS
#pragma ap_ctrl_none port=return

#pragma HLS DATAFLOW
        float a_buf[SIZE][SIZE*2], b_buf[SIZE*2][SIZE], c_buf[SIZE][SIZE];

        import_input<SIZE,SIZE*2>(a,a_buf);
        import_input<SIZE*2,SIZE>(b,b_buf);

        matmul_proc(a_buf,b_buf,c_buf);
        export_output(c,c_buf);
}
*/
```

5.2 matmul testbench

Below is the matmul_tb.cpp test bench code.

```
#define SZ 128
```

```
#include <stdio.h>
#include <iostream>
#include <string.h>
#include "matmul.h"

union conv {
        float f;
        unsigned int u;
};

int main(){

        float a[SZ][SZ*2];
        float b[SZ*2][SZ];
        //float c[SZ][SZ];
        float c_sw[SZ][SZ];

        AXI_F axi_a [SZ*SZ*2];
        AXI_F axi_b [SZ*SZ*2];
        AXI_F axi_c [SZ*SZ];

        for(int i=0;i<SZ;++i){
                for(int j=0;j<2*SZ;++j){
                        int idx = i*SZ*2 + j;
                        conv tmp;
                        tmp.f = 1.0;
                        axi_a[idx].data = tmp.u;
                        a[i][j] = 1.0;
                }
        }

        for(int i=0;i<2*SZ;++i){
                for(int j=0;j<SZ;++j){
                        int idx = i*SZ + j;
                        conv tmp;
                        tmp.f = 2.0;
                        axi_b[idx].data = tmp.u;
                        b[i][j] = 2.0;
                }
        }

        for(int k=0;k<SZ*2;++k){
```

```
                for(int i=0;i<SZ;++i){
                        float tmp = a[i][k];
                        for(int j=0;j<SZ;++j){
                                c_sw[i][j] += tmp*b[k][j];
                        }
                }
        }

        matmul_simple2(axi_a,axi_b,axi_c);

        int ret=0;
        for(int i=0;i<SZ;++i){
                for(int j=0;j<SZ;++j){
                        int idx = i*SZ + j;
                        conv tmp;
                        tmp.u = axi_c[idx].data;
                        if(tmp.f != c_sw[i][j]){
                                ret=1;
                                break;
                        }
                }
        }

        return ret;

}
```

5.3 Overlay Python Code

Below is the Overlay for matmul IP.

```
class matmul_overlay(Overlay):
    def __init__(self, bitfile_name, SIZE=64):
        super().__init__(bitfile_name)
        #self.saxpy_hw = self.krnl_saxpy2_0
        self.abuf = allocate((SIZE*SIZE*2,) , dtype=np.float32)
        self.bbuf = allocate((SIZE*SIZE*2,) , dtype=np.float32)
        self.obuf = allocate((SIZE*SIZE,) , dtype=np.float32)
        #interrupt not implemented

    def python_float_to_uint(self, num):
        return ctypes.c_uint.from_buffer(ctypes.c_float(num)).value
```

```
    def register_map(self):
        return self.register_map

    def ip_dict(self):
        return self.saxpy_hw.ip_dict()

    def run(self, x, y):
        np.copyto(self.abuf, x.flatten())
        np.copyto(self.bbuf, y.flatten())
        #self.saxpy_hw.write(saxpy_hw.register_map.size.address, SIZE)
        #self.saxpy_hw.write(saxpy_hw.register_map.a.address, self.python_float_to_u

        self.dma_1.sendchannel.transfer(self.abuf)
        self.dma_2.sendchannel.transfer(self.bbuf)
        self.out_dma.recvchannel.transfer(self.obuf)
        self.dma_1.sendchannel.wait()
        self.dma_2.sendchannel.wait()
        self.out_dma.recvchannel.wait()

        return self.obuf

    def assert_by_sw(self, x, y):
        xy = np.matmul(x,y)
        xy_hw = self.run(x,y)
        comparison = np.isclose(xy.flatten(),xy_hw)
        for i in comparison:
            assert(i == True)

        return True

MATMUL = matmul_overlay("matmul3.bit")
MATMUL.run(a,b)
MATMUL.assert_by_sw(a,b)
```

6 Bibliography

References