Lab 7 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.

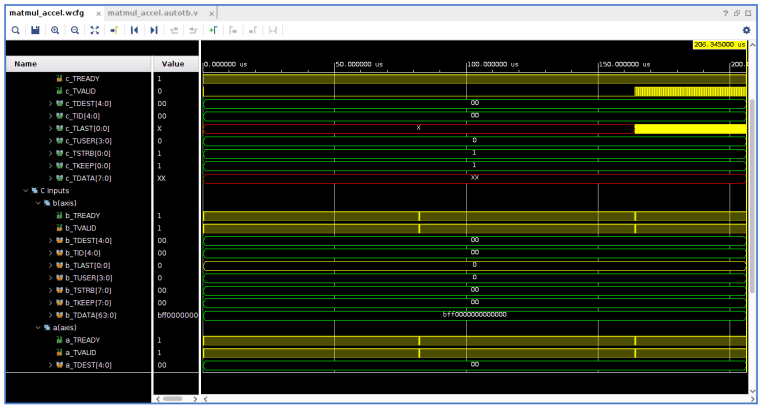
- For each parameters in matrix A and B,
  if $A_{ij}, B_{ij} \geq 0$, then replace the value as 1.
  otherwise, replace the value as -1.
- Perform matrix multiplication.
- Export the matrix C, the result as np.int8 format.
  Range of each elements in the matrix C is guaranteed to be $-128 \leq C_{ij} \leq 127$.

1.2 Optimization Detail

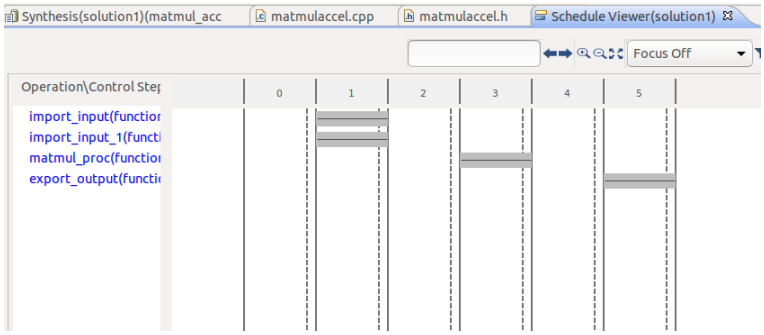
Map each parameters in A and B as 1 or -1, by its sign value.
This can be done as the example code. ap_int<2> ret = (e.data & 0x8000000000000000 ? -1 : 1);
Perform multiplication with two arrays consisted of only 1 and -1, and export the final result.
Please refer to the matmul_proc in 5.1.

1.3 Simulation Results

Below is the Cosimulation result.



Below is the C synthesis timing graph.



Below is the C synthesis report.

Synthesis(solution1)(matmul_accel_csynth.rpt)

ap_clk	10.00 ns	8.702 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
16390	16390	0.164 ms	0.164 ms	8197	8197	dataflow

Detail

Instance

Loop

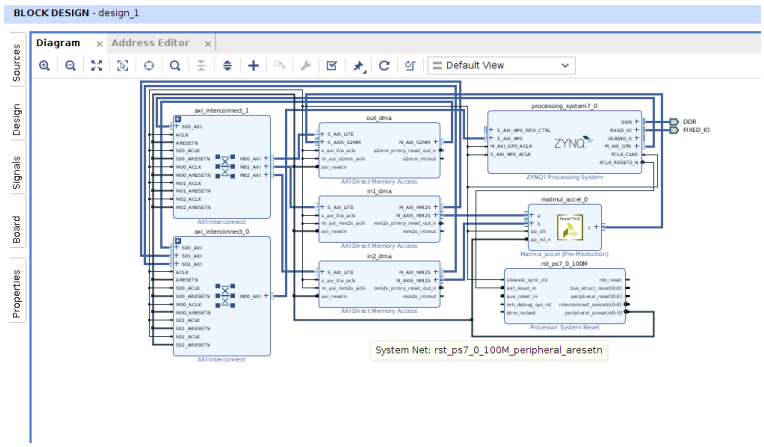
Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	262	-
FIFO	-	-	-	-	-
Instance	0	64	1464	4788	-
Memory	12	-	1024	512	0
Multiplexer	-	-	-	576	-
Register	-	-	64	-	-
Total	12	64	2552	6138	0
Available	280	220	106400	53200	0
Utilization (%)	4	29	2	11	0

Detail

Below is the top diagram of this matmul IP.



For a single matrix multiplication, HW is faster.

```
%%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 2.13 ms, sys: 294 µs, total: 2.42 ms
Wall time: 2.24 ms
```

```
%%time
sign_chk_map = np.vectorize(sign_chk)
a_sw = sign_chk_map(a)
b_sw = sign_chk_map(b)
np.matmul(a_sw,b_sw)
```

```
CPU times: user 51.1 ms, sys: 313 µs, total: 51.4 ms
Wall time: 48.1 ms
```

When matrix multiplication is inserted during the forward propagation phase, HW was slower. This is because the python code for HW resizes the 1×4096 size array into 64×64 2-d array.

```
%%time
prediction = []
for idx in range(len(X)//batch_size):
    xs = X[batch_size * idx:batch_size * idx + batch_size]
    ys = y[batch_size * idx:batch_size * idx + batch_size]
    outputs = feed_foward(xs)
    for output, yk in zip(outputs, ys):
        prediction.append(np.argmax(output) == (yk))

score = np.mean(prediction) * 100

print(score)
```

```
89.03247941445562
CPU times: user 2min 1s, sys: 9.83 s, total: 2min 10s
Wall time: 3min 6s
```

```
%%time
prediction2 = []
for idx in range(len(X)//batch_size):
    xs = X[batch_size * idx:batch_size * idx + batch_size]
    ys = y[batch_size * idx:batch_size * idx + batch_size]
    outputs2 = feed_foward2(xs)
    for output, yk in zip(outputs2, ys):
        prediction2.append(np.argmax(output) == (yk))

score = np.mean(prediction2) * 100

print(score)
```

```
89.03247941445562
CPU times: user 2min 45s, sys: 19 s, total: 3min 4s
Wall time: 7min 58s
```

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 "matmulaccel.h"



//
void matmul_proc(ap_int<2> a[SIZE][SIZE*2], ap_int<2> b[SIZE*2][SIZE], ap_int<8> c[S
//#pragma HLS ARRAY_RESHAPE variable=c complete dim=2

        //latency : 42ms
        for(int k=0;k<SIZE*2;++k){
                for(int i=0;i<SIZE;++i){
#pragma HLS PIPELINE II=1
                        ap_int<8> aik = a[i][k];
                        for(int j=0;j<SIZE;++j){
#pragma HLS UNROLL
                                c[i][j] += aik*b[k][j];
                        }
                }
        }

}


void clear_cbuf(ap_int<8> c[SIZE][SIZE]){

        for(int i=0;i<SIZE;++i){
#pragma HLS PIPELINE II=1
                for(int j=0;j<SIZE;++j){
        //#pragma HLS PIPELINE II=1
        #pragma HLS UNROLL
                        c[i][j]=0;
                }
        }
}



void export_output(AXI_INT8 *c, ap_int<8> cbuf[SIZE][SIZE]){
```

```
//#pragma HLS PIPELINE II=1
        for(int i=0;i<SIZE;++i){
                #pragma HLS PIPELINE II=1
                for(int j=0;j<SIZE;++j){
//#pragma HLS UNROLL
                        int idx = i*SIZE + j;
                        c[idx] = push_stream<8>(cbuf[i][j], idx == (SIZE*SIZE - 1));
                }
        }
}

template< int row, int col>
void import_input(AXI_F *x, ap_int<2> xbuf[row][col]){

        for(int i=0;i<row;++i){
                #pragma HLS PIPELINE II=1
                for(int j=0;j<col;++j){
//#pragma HLS UNROLL
                        int idx = i*col + j;
                        xbuf[i][j] = pop_stream<64>(x[idx]);
                }
        }
}


void matmul_init(AXI_F*a,ap_int<2> a_buf[SIZE][SIZE*2],
                AXI_F* b,ap_int<2>b_buf[SIZE*2][SIZE],
                ap_int<8> c_buf[SIZE][SIZE]){
#pragma HLS DATAFLOW
        import_input<SIZE,SIZE*2>(a,a_buf);
        import_input<SIZE*2,SIZE>(b,b_buf);
        clear_cbuf(c_buf);
}

void matmul_accel(AXI_F* a, AXI_F* b, AXI_INT8* c){
#pragma HLS INTERFACE axis port=a depth = 8192
#pragma HLS INTERFACE axis port=b depth = 8192
#pragma HLS INTERFACE axis port=c depth = 4096
//#pragma HLS INTERFACE s_axilite port=return bundle=CTRL_BUS
#pragma HLS INTERFACE ap_ctrl_none port=return

//#pragma HLS DATAFLOW
```

```
        ap_int<2> a_buf[SIZE][SIZE*2], b_buf[SIZE*2][SIZE];
//#pragma HLS ARRAY_RESHAPE variable=a_buf complete dim=2
#pragma HLS ARRAY_RESHAPE variable=b_buf complete dim=2
        ap_int<8> c_buf[SIZE][SIZE];
#pragma HLS ARRAY_RESHAPE variable=c_buf complete dim=2

        matmul_init(a,a_buf,b,b_buf,c_buf);
        //memset(c_buf, 0 ,SIZE*SIZE*sizeof(ap_int<8>));

        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 64

#include <stdio.h>
#include <iostream>
#include <string.h>
#include "matmulaccel.h"

union conv {
        double f;
        unsigned long long u;
};

int main(){

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

        AXI_F axi_a [SZ*SZ*2];
        AXI_F axi_b [SZ*SZ*2];
        AXI_INT8 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;
        }
}

for(int i=0;i<2*SZ;++i){
        for(int j=0;j<SZ;++j){
                int idx = i*SZ + j;
                conv tmp;
                tmp.f = -1.0 ;
                axi_b[idx].data = tmp.u;
                b[i][j] = -1 ;
        }
}

memset(c_sw,0,sizeof(int)*SZ*SZ);
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];
                }
        }
}

for(int i=0;i<1;++i){
        matmul_accel(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;

                if(axi_c[idx].data != c_sw[i][j]){
                        printf("%d␣%d␣%d␣%d\n", i, j, int(axi_c[idx].data),c_
                        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.float64)
        self.bbuf = allocate((SIZE*SIZE*2, ) , dtype=np.float64)
        self.obuf = allocate((SIZE*SIZE, ) , dtype=np.int8)


    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.in1_dma.sendchannel.transfer(self.abuf)
        self.in2_dma.sendchannel.transfer(self.bbuf)
        self.out_dma.recvchannel.transfer(self.obuf)
        self.in1_dma.sendchannel.wait()
        self.in2_dma.sendchannel.wait()
        self.out_dma.recvchannel.wait()

        obuf_64by64 = np.reshape(self.obuf, (64,64))
        #obuf_64by64 = obuf_64by64.astype(np.float64)
```

```
        #obuf_64by64 = self.obuf.resize((64,64))

        return obuf_64by64

    def assert_by_sw(self, x, y):
        def sign_chk(a):
            if (a>=0):
                return 1
            else:
                return -1
        import time
        sign_chk_map = np.vectorize(sign_chk)
        x_sw = sign_chk_map(a)
        y_sw = sign_chk_map(b)
        start_sw = time.time()
        xy = np.matmul(x_sw,y_sw)
        end_sw = time.time()
        start_hw = time.time()
        xy_hw = self.run(x,y)
        end_hw = time.time()
        comparison = np.isclose(xy.flatten(),xy_hw.flatten())
        for i in comparison:
            assert(i == True)

        print(end_sw-start_sw, end_hw-start_hw)

        return True
```

6 Bibliography

References