Lab 4 Report

Jeon Woongbae, Park Sangbeen

UNIST

EE32601

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

1 Optimization

Below is the optimization approach.

1.1 Naive Implementation (Not Optimized)

1.1.1 Not Optimized Code

Below is the naive implementation of SAXPY.

```
void saxpy(float* x, float* y, float a){
#pragma HLS INTERFACE m_axi port=x offset=slave depth=20
#pragma HLS INTERFACE m_axi port=y offset=slave depth=20
#pragma HLS INTERFACE m_axi port=a offset=slave depth=20

#pragma HLS INTERFACE s_axilite port=x bundle=control
#pragma HLS INTERFACE s_axilite port=y bundle=control
#pragma HLS INTERFACE s_axilite port=a bundle=control

#pragma HLS INTERFACE s_axilite port=return bundle=control

#pragma HLS dataflow

    for(int i=0;i<SIZE;++i){
                y[i] += a*x[i];
        }

}
```

Below result shows that it takes 229 cycles for 10 inputs.

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.317 ns	2.70 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
229	229	2.290 us	2.290 us	230	230	dataflow

Detail

Instance

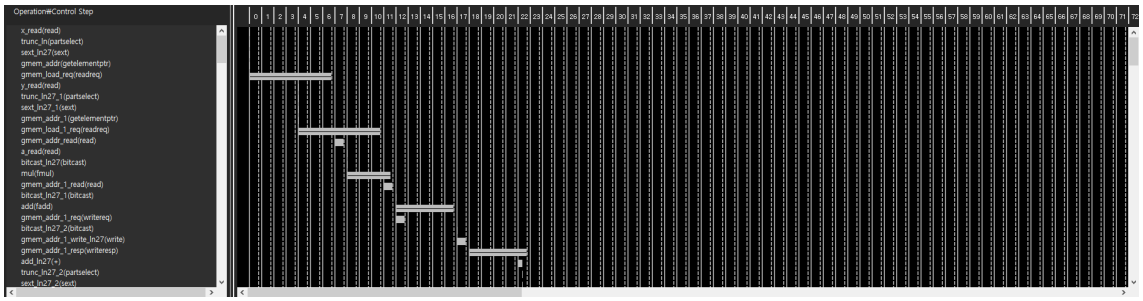
Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	-	-	-
FIFO	-	-	-	-	-
Instance	2	5	3225	4275	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	2	5	3225	4275	0
Available	280	220	106400	53200	0
Utilization (%)	~0	2	3	8	0

Below is the timing graph.
It can be shown that it is not optimized.



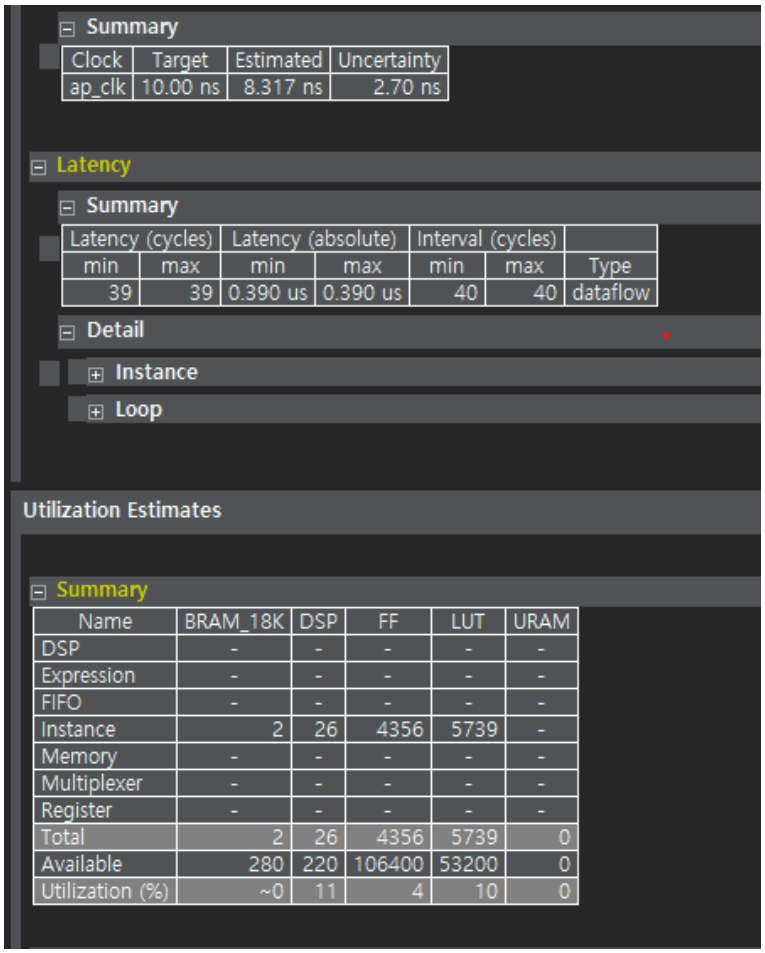
1.1.2 Optimized Approach

Below is the optimized code for SAXPY.
We used two buffers for X and Y, and a single size buffer for a number a.
Then, we imported two vectors X and Y into their own buffers,
calculated aX, added aX to the ybuf, and copied ybuf into the target Y.

```
void saxpy(float* x, float* y, float a){
#pragma HLS INTERFACE m_axi port=x offset=slave
#pragma HLS INTERFACE m_axi port=y offset=slave
#pragma HLS INTERFACE m_axi port=a offset=slave

#pragma HLS INTERFACE s_axilite port=x bundle=control
#pragma HLS INTERFACE s_axilite port=y bundle=control
#pragma HLS INTERFACE s_axilite port=a bundle=control

#pragma HLS INTERFACE s_axilite port=return bundle=control

#pragma HLS dataflow

        float buf[SIZE];
        float ybuf[SIZE];
        float abuf;
        abuf = a;


        for(int i=0;i<SIZE;++i){
#pragma HLS UNROLL
                buf[i] = x[i];
        }

        for(int i=0;i<SIZE;++i){
#pragma HLS UNROLL
                ybuf[i] = y[i];
        }

        for(int i=0;i<SIZE;++i){
#pragma HLS UNROLL
                buf[i] *= abuf;
        }
```

```
        for(int i=0;i<SIZE;++i){
#pragma HLS UNROLL
                ybuf[i] += buf[i];
        }

        for(int i=0;i<SIZE;++i){
#pragma HLS UNROLL
                y[i] = ybuf[i];
        }


}
```

Below is the execution result of synthesis. It takes 39 cycles.
Compared with the non-optimized version, which takes 230 cycles.
Optimized version is about 6 times faster than the non-optimized version.



Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.317 ns	2.70 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
39	39	0.390 us	0.390 us	40	40	dataflow

Detail

Instance

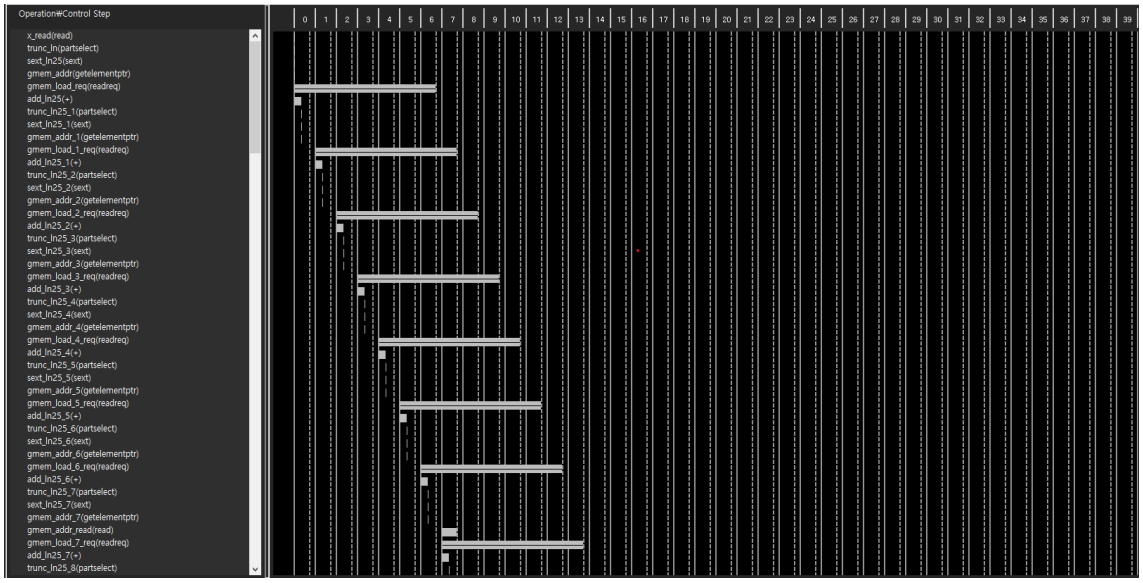
Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	-	-	-
FIFO	-	-	-	-	-
Instance	2	26	4356	5739	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	2	26	4356	5739	0
Available	280	220	106400	53200	0
Utilization (%)	~0	11	4	10	0

Below is the timing graph.
It can be shown that operations are parallelized.



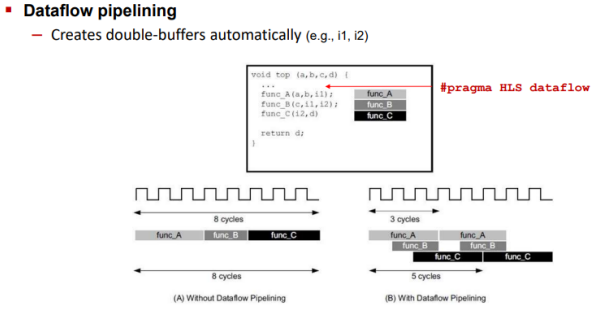
1.1.3 Explanation

Buffer is used to serialize data read requests.
If data read requests are not serialize, then each arithmetic operations have to wait for their data to be fetched.
When buffers are used, all the data are loaded and arithmetic operations do not have to wait for data fetch.

Arithmetic operations are disassembled.
Instead of doing $Y+ = aX$, we disassembled them in to two pieces; computing aX and saving them into buffers, and execute $Y+ = aX$.
By disassembling the operation into two parts, computing aX requires only floating point multiplication operations, and $Y+ = aX$ requires only floating point addition operations.
This unifies the class of arithmetic operations, making it parallelizable.

Loop unrolling with #pragma HLS UNROLL is enabled.
This enables loop unrolling, parallelizing the for loops.

#pragma HLS dataflow is used to parallelize buffer creation.
buf for X, ybuf for Y are created in parallel by using the #pragma HLS dataflow command.
Using this reduces 4 cycles, making 43 cycles to 39 cycles.



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 Single Scalar Value Input

We acknowledge that we got some hint from Hong Junwha, from other group.
Pointer-format for a single value was not successful, and with the hint,
we enabled to make a convert logic between Python float32 and uint32.

Converting original a as uint32 format and sending that uint32 value to the SAXPY Overlay made it work.
Below is the implementation of conversion logic.

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

2.2 Usage of Interrupts

saxpy_hw.register_map.CTRL.AP_START = 1 immediately executes the SAXPY logic.
However, accessing the result right after the start of execution does not guarantee the correctness of the result, because there is no guarantee that the computation logic has finished.
Thus, we explicitly made an infinite loop that checks whether the SAXPY chip has entered into an idle state, which indicates the end of execution.
Below is the part of implemented logic in SAXPY driver.

```
saxpy_hw.register_map.CTRL.AP_START = 1
while(saxpy_hw.register_map.CTRL.AP_IDLE != 1):
    continue
```

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.
Hong Junwha gave a brief hint about single scalar input.

5 Bibliography

References