Optimizing FFT Resource Efficiency on FPGA using High-level Synthesis

Guanglin Xu, Tze Meng Low, James C. Hoe, Franz Franchetti

Contact: guanglinxu@cmu.edu

Carnegie Mellon University

Introduction
HLS: improved productivity in FPGA programming

```
#include "C/C++ Code"

// High-level Synthesis

Implement FFT in HLS often causes inefficient use of resources

BRAM Utilization Rate
(Higher is better)

Peak Frequency
(Higher is better)

Can we change the way using HLS for retaining the resource efficiency?

Yes! Make the way using HLS for retaining the resource efficiency?

Make Permutation Explicit
HLS cannot handle port conflict automatically

Results in long latency

<table>
<thead>
<tr>
<th>6</th>
<th>4</th>
<th>2</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Bit Reversal Permutation

Implementations

RTL: HLS:

Can we change the way using HLS for retaining the resource efficiency?

Yes! Make the way using HLS for retaining the resource efficiency?

Vivado HLS Code and Pragmas

```
void switch(int T in[2], out[2], bool on) {
    if (on == true) {
        out[0] == in[on==true][1];
        out[1] == in[on==true][0];
    }
}
```

```
void pease_fft(type X[N], type Y[N]) {
    #pragma HLS INTERFACE axis port=X,Y
    #pragma HLS ARRAY_PARTITION variable=X,Y cyclic factor=2
    for (i=0; i<N/2; i++) {
        #pragma HLS PIPELINE
        in[1] = X[2*j+i];
        write_stage_bitrev(buf, in);
        for (j=0; j<N/2; j++) {
            #pragma HLS PIPELINE
            if (i==0) read_stage_bitrev(buf, in);
            else read_stage_stride2(buf, in);
            twid mul-(in, twiddled, 4);
            radix2_butterfly(twiddled, out);
            write_stage_stride2(buf, out);
        }
    }
    #pragma HLS PIPELINE
    read_stage_stride2(buf, out);
    Y[2*j+1] = out[0]; Y[2*j+2] = out[1];
}
```

Map HLS Loop to Feedback Path

Original FFT feedback path: from streaming to streaming

No direct mapping from HLS code

```
proc
```

Iterative Datapath for the Pease FFT Algorithm

Revised feedback path: from BRAM to BRAM

Natural to map from a HLS loop

```
proc
```

Experimental Results

Latency of radix-2 FFT

| Device: Xilinx ZC706 |

<table>
<thead>
<tr>
<th>Latency (thousand cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve HLS vs. Xilinx HLS</td>
</tr>
</tbody>
</table>

Reduce latency significantly

Resource Utilization of radix-4 FFT-1k

<table>
<thead>
<tr>
<th>Clock Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 MHz</td>
</tr>
<tr>
<td>200 MHz</td>
</tr>
<tr>
<td>300 MHz</td>
</tr>
<tr>
<td>400 MHz</td>
</tr>
</tbody>
</table>

Much better than naïve; comparable to RTL

Conclusion

- Make the FFT datapath explicit improves the design quality
- Achieve similar performance and resource utilization to the Xilinx RTL reference
- Could potentially apply to other algorithms

Acknowledgements

This work was sponsored partly by the DARPA BRASS program under agreement FA8750-16-2-003. The content, views and conclusions presented in this document are those of the author(s) and do not necessarily reflect the position or the policy of the sponsoring agencies.