Optimizing FFT Resource Efficiency on FPGA using High-level Synthesis

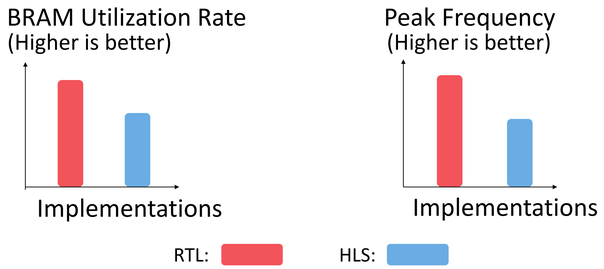
Guanglin Xu, Tze Meng Low, James C. Hoe, Franz Franchetti Contact: guanglinxu@cmu.edu Carnegie Mellon University

Electrical & Computer ENGINEERING

Introduction

HLS: improved productivity in FPGA programming

Implement FFT in HLS often causes inefficient use of resources



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

Yes! Make the datapath explicit in HLS code!

Make Permutation Explicit

HLS cannot handle port conflict automatically

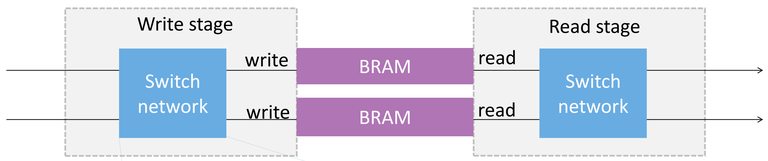
Results in long latency

Bit Reversal Permutation

Port Conflict on BRAM

Make it explicit

Use a systematic method from *Püschel et al. J. ACM. 2009, 56, 1*



Datapath Skeleton for Permutations

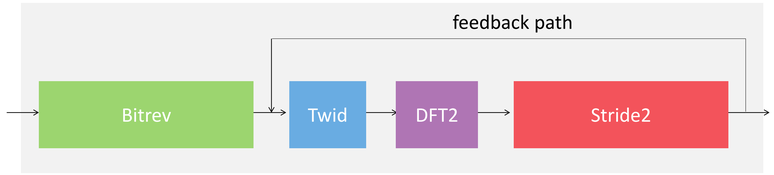
```
void switch(T in[2], T out[2], bool on) {
 out[0] = in[(on==true)?(1):(0)];
 out[1] = in[(on==true)?(0):(1)];
}
```

Simplify array indexing

Map HLS Loop to Feedback Path

Original FFT feedback path: from streaming to streaming

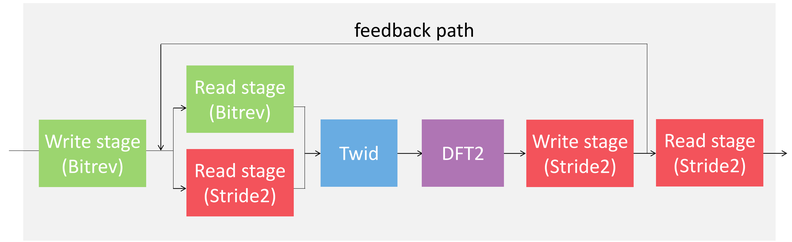
No direct mapping from HLS code



Iterative Datapath for the Pease FFT Algorithm

Revised feedback path: from BRAM to BRAM

Natural to map from a HLS loop



Revised Block Diagram

Easier to map from a loop

Vivado HLS Code and Pragmas

Stream interface
Stream width=2

Pipeline

Disable conservative dependency analysis

Fuse computation and permutation into one loop

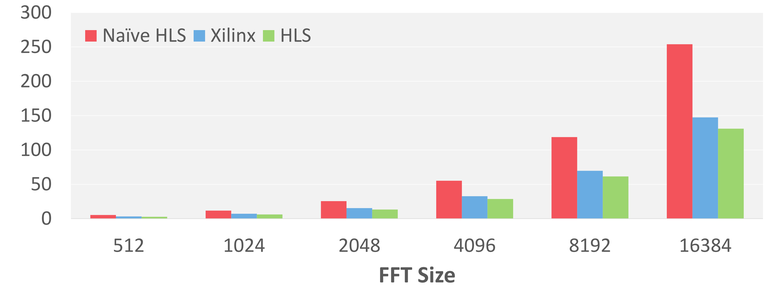
```
void pease_fft(type X[N], type Y[N]) {
#pragma HLS INTERFACE axis port=X,Y
#pragma HLS ARRAY_PARTITION variable=X,Y \
          cyclic factor=2
 for (j=0; j<N/2; j++) {
#pragma HLS PIPELINE
   in[0] = X[2*j+0]; in[1] = X[2*j+1];
   write_stage<bitrev>(buf, in);
 }
 for (i=0; i<LOG2N; i++) {
   for (j=0; j<N/2; j++) {
#pragma HLS DEPENDENCE variable=buf inter false
#pragma HLS PIPELINE
     if (i==0) read_stage<bitrev>(buf, in);
     else read_stage<stride2>(buf, in);
     twid_mult(in, twiddled, i);
     radix2_butterfly(twiddled, out);
     write_stage<stride2>(buf, out);
   }
 }
 for (j=0; j<N/2; j++) {
#pragma HLS PIPELINE
     read_stage<stride2>(buf, out);
     Y[2*j+0] = out[0]; Y[2*j+1] = out[1];
 }
}
```

Experimental Results

Latency of radix-2 FFT

Naïve HLS: Naïve HLS implementations
Xilinx: RTL reference from ___, *FFT LogiCORE IP, 2017. [Online]*
HLS: Implementations with explicit datapath

Latency
(thousand cycles)

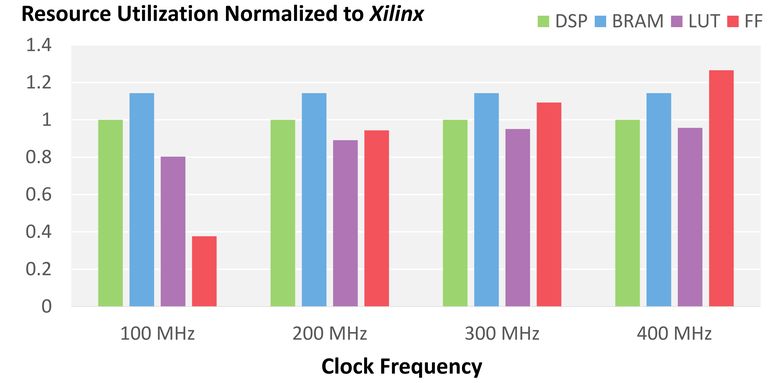


Reduce latency significantly

Resource Utilization of radix-4 FFT-1k

Device: Xilinx ZC706

Resource Utilization Normalized to *Xilinx*



- Peak clock frequency: 400 MHz vs. Naïve HLS ~200 MHz
- Similar resource consumption, fewer FFs at 100 MHz

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 authors(s) and do not necessarily reflect the position or the policy of the sponsoring agencies.