Evaluation of the FIR Example using Xilinx Vivado High-Level Synthesis Compiler

Argonne Leadership Computing Facility
About Argonne National Laboratory
Argonne is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC
under contract DE-AC02-06CH11357. The Laboratory’s main facility is outside Chicago, at
9700 South Cass Avenue, Argonne, Illinois 60439. For information about Argonne
and its pioneering science and technology programs, see www.anl.gov.

DOCUMENT AVAILABILITY
Online Access: U.S. Department of Energy (DOE) reports produced after 1991 and a
growing number of pre-1991 documents are available free via DOE’s SciTech Connect
(http://www.osti.gov/scitech/)

Reports not in digital format may be purchased by the public from the
National Technical Information Service (NTIS):
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312
www.ntis.gov
Phone: (800) 553-NTIS (6847) or (703) 605-6000
Fax: (703) 605-6900
Email: orders@ntis.gov

Reports not in digital format are available to DOE and DOE contractors from the
Office of Scientific and Technical Information (OSTI):
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
www.osti.gov
Phone: (865) 576-8401
Fax: (865) 576-5728
Email: reports@osti.gov

Disclaimer
This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor UChicago Argonne, LLC, nor any of their employees or officers, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of document authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof, Argonne National Laboratory, or UChicago Argonne, LLC.
Evaluation of the FIR Example using Xilinx Vivado High-Level Synthesis Compiler

prepared by Zheming Jin, Hal Finkel, Kazutomo Yoshii, Franck Cappello

Argonne Leadership Computing Facility, Argonne National Laboratory

July 28, 2017
Introduction

Compared to central processing units (CPUs) and graphics processing units (GPUs), field programmable gate arrays (FPGAs) have major advantages in reconfigurability and performance achieved per watt. This development flow has been augmented with high-level synthesis (HLS) flow that can convert programs written in a high-level programming language to Hardware Description Language (HDL). Using high-level programming languages such as C, C++, and OpenCL for FPGA-based development could allow software developers, who have little FPGA knowledge, to take advantage of the FPGA-based application acceleration. This improves developer productivity and makes the FPGA-based acceleration accessible to hardware and software developers.

Xilinx Vivado HLS compiler is a high-level synthesis tool that enables C, C++ and System C specification to be directly targeted into Xilinx FPGAs without the need to create RTL manually. The white paper [1] published recently by Xilinx uses a finite impulse response (FIR) example to demonstrate the variable-precision features in the Vivado HLS compiler and the resource and power benefits of converting floating point to fixed point for a design.

To get a better understanding of variable-precision features in terms of resource usage and performance, this report presents the experimental results of evaluating the FIR example using Vivado HLS 2017.1 and a Kintex Ultrascale FPGA. In addition, we evaluated the half-precision floating-point data type against the double-precision and single-precision data type and present the detailed results.

Experimental Setup

For a discrete-time FIR filter of order N, each value of the output is a weighted sum of the most recent N input values:

\[ y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + b_2 \cdot x[n-2] + \ldots + b_N \cdot x[n-N] \]

where \( x[n] \) is the value of the input signal, \( y[n] \) is the value of the output signal, N is the filter order, \( b_i \) is the coefficient of the filter. The \( x[n-i] \) in these terms are commonly referred to as taps. An N-th order filter is also referred to as an (N+1)-tap filter.

Figure 1 shows the floating- and fixed-point C++ implementations [1] of the FIR example. Users can change the data type of the input, output signals and the coefficient by specifying the values of “fp_data_t”, “fp_coef_t” and “fp_acc_t” in a C++ header file. Both the input signals and the coefficients are stored in one-dimensional arrays. A chain of shift registers stores the successive input signal.

```c
// Top-level function for the floating-point FIR
fp_acc_t fp_FIR(fp_data_t x) {
    static CFir<fp_coef_t, fp_data_t, fp_acc_t> fir1;
    return fir1(x);
}
```

// Top-level function for the fixed-point FIR
fx_acc_t fx_FIR(fx_data_t x) {
    static CFir<fx_coef_t, fx_data_t, fx_acc_t> fir1;
    return fir1(x);
}

// FIR function
template<class coef_T, class data_T, class acc_T>
acc_T CFir<coef_T, data_T, acc_T>::operator()(data_T x) {
    int i;
    acc_T acc = 0;
    data_T m;

    loop: for (i = N-1; i >= 0; i--) {
        if (i == 0) {
            m = x;
            shift_reg[0] = x;
        } else {
            m = shift_reg[i-1];
            if (i != (N-1)) {
                shift_reg[i] = shift_reg[i - 1];
            }
        }
        acc += m * c[i];
    }
    return acc;
}

Figure 1. The floating- and fixed-point implementations in C++ of the FIR example

We evaluated the example with Vivado HLS 2017.1 and the XCKU115-FLVA1517-2E (KU115) FPGA device. Table 1 lists a few major features of the two FPGA devices. VU9P is based on Virtex UltraScale+ architecture while KU115 is based on Kintex Ultrascale architecture. VU9P has approximately 1.78X more logic resources and 1.23X more DSPs than KU115.

Table 1. Features of the two FPGA devices VU9P and KU115 [2]

<table>
<thead>
<tr>
<th></th>
<th>System logic cells</th>
<th>CLB Flip-flops</th>
<th>CLB LUTs</th>
<th>DSPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCVU9P-2FLGB2104</td>
<td>Virtex UltraScale+</td>
<td>2586,150</td>
<td>2364,480</td>
<td>1182,240</td>
</tr>
<tr>
<td>XCKU115-FLVA1517-2E</td>
<td>Kintex UltraScale</td>
<td>1451,100</td>
<td>1326,720</td>
<td>663,360</td>
</tr>
</tbody>
</table>
Experimental Results

The white paper evaluated the 85-tap finite impulse response example with Vivado HLS 2016.4 and the XCVU9P-2FLGB2104 (VU9P) FPGA device. The target frequency is 500 MHz.

Tables 2 lists the maximum frequency (Fmax) and FPGA resource usage of the single-precision floating-point and fixed-point implementations of a single FIR in [1]. For the fixed-point implementation, the data types are 18-bit coefficient with 1 integer and 17 fractional bits, 27-bit input signal with 15 integer and 12 fractional bits and a 48-bit accumulator with 19 integer and 29 fractional bits. It should be noted that the experimental results in this report are based on the same source program and script as in [1].

Table 2. FPGA implementation results of the floating-point and fixed-point FIRs in [1]

<table>
<thead>
<tr>
<th></th>
<th>Single-precision Floating Point</th>
<th>Fixed Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fmax (MHz)</td>
<td>500</td>
<td>580</td>
</tr>
<tr>
<td>Latency</td>
<td>91</td>
<td>12</td>
</tr>
<tr>
<td>Iteration Interval</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DSP48E2</td>
<td>423</td>
<td>85</td>
</tr>
<tr>
<td>LUTs</td>
<td>23106</td>
<td>1973</td>
</tr>
</tbody>
</table>

Table 3. FPGA implementation results of the floating-point and fixed-point FIRs

<table>
<thead>
<tr>
<th></th>
<th>Single-precision Floating Point</th>
<th>Fixed Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fmax (MHz)</td>
<td>443</td>
<td>366</td>
</tr>
<tr>
<td>Latency</td>
<td>91</td>
<td>8</td>
</tr>
<tr>
<td>Iteration Interval</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DSP48E2</td>
<td>423</td>
<td>81</td>
</tr>
<tr>
<td>LUTs</td>
<td>21572</td>
<td>1151</td>
</tr>
<tr>
<td>CLBs</td>
<td>5939</td>
<td>631</td>
</tr>
<tr>
<td>FFs</td>
<td>49481</td>
<td>4383</td>
</tr>
<tr>
<td>Power(W)</td>
<td>4.31</td>
<td>1.577</td>
</tr>
</tbody>
</table>

Tables 3 lists the Fmax and resource usage of the floating-point and fixed-point implementations using Vivado HLS 2017.1 and the KU115 device. The iteration interval is 1 for all the implementations. Compared to the Fmax results from VU9P, the KU115 device cannot achieve 500MHz frequency for the same 91-cycle latency. While the number of DSPs for the single-precision floating point are the same for both devices, Vivado HLS 2017.1 generates the fixed-point implementation with 81 DSPs instead of 85 DSPs. In addition, the latency of the fixed-point implementation is 8 instead of 12. Due to the low latency, Fmax of the fixed-point version cannot achieve the target timing 400 MHz\(^1\).

For the single-precision floating-point implementation, the tool instantiates 84 floating-point add operators and 85 floating-point multiply operators in the RTL design. These operators are

---

1 The timing is met using Vivado HLS 2016.4
then transformed into 423 DSPs by Vivado. For the fixed-point implementation, 81 DSPs are composed of 28 DSPs for integer multiply and 53 DSPs for integer multiply and add.

To achieve the required 400 MHz timing, we increase the latency required to produce an output by adding the Vivado HLS latency pragma [3], “#pragma HLS latency min=<int> max=<int>”, in the FIR function in Figure 1. When the latency pragma is specified with a specific latency that is greater than the minimum latency, Vivado HLS extends the latency to the specified value.

Table 4 lists the FPGA implementation results with latency ranging from 9 to 12. Note the number of DSPs (not shown in the Table) is always 81. As shown in Table 4, when the latency is larger than 8, all the implementations meet the timing requirement. Increasing the latency increases the number of CLBs from 592 to 810, the number of FFs from 4287 to 4338. The total power consumption, from Vivado power report, increases very slightly from 1.589 W to 1.599 W.

Table 4. FPGA implementation results of the fixed-point FIRs (different latency)

<table>
<thead>
<tr>
<th>Latency</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fmax (MHz)</td>
<td>418</td>
<td>412</td>
<td>419</td>
<td>414</td>
</tr>
<tr>
<td>LUTs</td>
<td>1361</td>
<td>1361</td>
<td>1362</td>
<td>1362</td>
</tr>
<tr>
<td>CLBs</td>
<td>592</td>
<td>621</td>
<td>775</td>
<td>810</td>
</tr>
<tr>
<td>FFs</td>
<td>4287</td>
<td>4290</td>
<td>4294</td>
<td>4338</td>
</tr>
<tr>
<td>Power (W)</td>
<td>1.589</td>
<td>1.593</td>
<td>1.598</td>
<td>1.599</td>
</tr>
</tbody>
</table>

Starting from the version 2015.3, Vivado HLS natively supports a half-precision (16-bit) floating-point data type. This data type provides the strength of standard C float types but uses fewer hardware resources when synthesized. The half-precision floating-point data type provides a smaller dynamic range than the standard 32-bit float type. The data type has 1 signed bit, 5 exponent bits, and 10 mantissa bits.

Table 5. Double-, single-, and half-precision floating-point implementations of the FIR example

<table>
<thead>
<tr>
<th></th>
<th>Double-precision</th>
<th>Single-precision</th>
<th>Half-precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fmax (MHz)</td>
<td>351</td>
<td>443</td>
<td>433</td>
</tr>
<tr>
<td>Latency</td>
<td>118</td>
<td>91</td>
<td>86</td>
</tr>
<tr>
<td>Iteration Interval</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DSP48E2</td>
<td>847</td>
<td>423</td>
<td>338</td>
</tr>
<tr>
<td>LUTs</td>
<td>67119</td>
<td>21572</td>
<td>11105</td>
</tr>
<tr>
<td>CLBs</td>
<td>18436</td>
<td>5939</td>
<td>4082</td>
</tr>
<tr>
<td>FFs</td>
<td>147094</td>
<td>49481</td>
<td>29514</td>
</tr>
<tr>
<td>Power (W)</td>
<td>8.262</td>
<td>4.31</td>
<td>3.13</td>
</tr>
</tbody>
</table>

We extended the source program to evaluate the resource and performance results of double- and half-precision implementations. As shown in Table 5, Fmax of the double-precision implementation is 351 MHz, lower than the target frequency. The number of DSPs decreases from 847 (double-precision) to 423 (single-precision) to 338 (half-precision). The resource usage (LUTs, CLBs and FFs) of the double-precision implementation is 2X more than the single-precision implementation. Compared to the resource usage of the single-precision floating-point implementation, the half-precision floating-point implementation decreases the number of DSPs
by 20%, the number of LUTs by 48%, the number of CLBs by 31%, and the number of FFs by 40%. The power decreases from 8.262W (double-precision) to 4.31W (single-precision) to 3.13W (half-precision).

For the single-precision floating-point FIR, increasing the latency makes the FPGA implementations meet the timing requirement. For the double-precision, Table 6 shows the latency increase from 118 cycles to 138 cycles does not always improve Fmax.

| Fmax (MHz) | 351 | 388 | 340 |
| Latency    | 118 | 128 | 138 |
| Iteration Interval | 1   | 1   | 1   |
| DSP48E2    | 847 | 847 | 847 |
| LUTs       | 67119 | 67190 | 67193 |
| CLBs       | 18436 | 17846 | 17206 |
| FFs        | 147094 | 147050 | 147070 |
| Power (W)  | 8.262 | 8.4 | 8.339 |

Table 6. Double-precision floating-point implementations of the FIR example (different latency)

Conclusion

The report evaluates the FPGA implementations of the FIR example using Vivado HLS 2017.1 and the KU115 FPGA device. The maximum frequency of the FIR implementation is less than 450 MHz. The single-precision implementation can reduce the FPGA resource usage by approximately 50% compared to the double-precision implementation. Compared to the resource usage of the single-precision floating-point implementation, the half-precision floating-point implementation reduces the number of DSPs by 20%, the number of LUTs by 48%, the number of CLBs by 31%, and the number of FFs by 40%. A developer may improve Fmax of an implementation by increasing the latency of a function in the source program.

Acknowledgements

This material is based upon work supported by the U.S. Department of Energy Office of Science, under contract DEAC02-06CH11357.

Reference
