High Level Synthesis of Neville Interpolation on an Embedded FPGA Platform using SDSoC

Kyeong-Bin Park[1], Jung-Hyun Hong[2] and Ki-Seok Chung[3]
Department of Electronics and Computer Engineering, Hanyang University
Haengdang 1-dong, Seongdong-gu, Seoul, 133-791, Korea
E-mail: [1]eurakb@gmail.com, [2] jhhong34@gmail.com, [3]kchung@hanyang.ac.kr

Abstract: Today, high-level synthesis (HLS) has emerged as a widely used method for various digital systems. In this paper, we propose a the line interpolator implemented on Xilinx Zynq SoC using a high-level synthesis tool called Software-Defined SoC (SDSoC). There are lots of iterative calculations in the line interpolation algorithm; therefore, the hardware acceleration is preferred over the software-only implementation. We implement the Neville interpolation algorithm and apply the loop unrolling and pipelining techniques to fully utilize the target FPGA. When the utilization techniques are applied, the area is increased, but the total execution time is reduced by 10 times. The proposed implementation using SDSoC shows better performance compared to the software-only one, and fast design space exploration is achieved using HLS compared to the conventional RTL design.

1. Introduction

As technology progresses and systems become rapidly complex, it has become almost inevitable to use high-level abstractions and synthesis methods in designing digital circuits. High-level synthesis (HLS) is the automatic synthesis of a design structure from a behavioral specification, at levels above the logic level [1]. The Software-Defined SoC (SDSoC) development environment is one of the HLS development tools [2]. SDSoC is designed to be used mainly for a heterogeneous target platform to implement embedded systems. One of the biggest advantages of using SDSoC is that the compiler generates the complete hardware and software system based on the user-specified target platform and the required design functionality through fast design space exploration.

There are lots of iterative calculations in the line interpolation algorithm; therefore, the hardware acceleration is preferred over the software-only implementation. Neville's algorithm, for example, is one of the popular line interpolation algorithms, and it uses the prior calculation results in the following step [4]. In this interpolation algorithm, initially, the 0^{th} order polynomial for a point is calculated. Then, the 1^{st} order polynomial for the two points obtained by the 0^{th} polynomial is extracted. So this algorithm can extract the n^{th} order polynomial by repeating this process. In this paper, we propose an implementation of the iterative Neville interpolation algorithm on the Xilinx Zynq SoC platform using SDSoC [3].

Since the hardware logic generated by the default compilation by SDSoC may not meet the performance requirement, optimization through parallelization and pipelining should be taken into consideration to achieve the satisfactory target performance. To improve the performance of the interpolation, techniques such as loop-unrolling and loop pipeling have been employed. A detailed description of the optimizing techniques will be explained in Section 3.

2. Neville Interpolation Algorithm

Interpolation is a method of obtaining an approximate function from known statistical or experimental data points (x_i's) and finding new data points by the approximate function (g(x)) within the range of the known data points. There are several algorithms such as linear interpolation, Lagrange interpolation, Neville interpolation and Newton interpolation, etc. In this paper, we implement the Neville interpolation algorithm on a Zynq SoC platform using SDSoC. The Neville interpolation algorithm uses the prior calculation results in the following step. Equation (1), (2), and (3) show the order of calculating polynomials in the Neville interpolation. (1) is used when the number of the given point is one. (2) is used when the number of the given points is two, and so on.

$$g_0(\mathrm{x}) = f(x_0),\ g_1(\mathrm{x}) = f(x_1), \cdots \quad (1)$$

$$g_{0,1}(\mathrm{x}) = \frac{(x-x_0)g_1(\mathrm{x})-(x-x_1)g_0(\mathrm{x})}{x_1-x_0}, \cdots \quad (2)$$

$$g_{0,1,2}(\mathrm{x}) = \frac{(x-x_0)g_{1,2}(\mathrm{x})-(x-x_2)g_{0,1}(\mathrm{x})}{x_2-x_0}, \cdots \quad (3)$$

3. Implementation of Neville Interpolation

In this paper, the Neville interpolation algorithm is implemented on a Zynq SoC platform using SDSoC. A pseudo code of the Neville algorithm is given in Figure 1.

To optimize the hardware logic generated by SDSoC to meet the performance requirement, the loop unrolling and loop pipelining techniques to fully utilize the target FPGA are employed. Loop unrolling is a technique that exploits parallelism across multiple loop iterations. It creates multiple copies of the loop body and adjusts the loop iteration counter accordingly. Loop pipelining is a technique that allows the operations in a loop to be executed in an overlapping manner as shown in Figure 2. Without pipelining, six clock cycles are needed to carry out two consecutive READ operations. However, with pipelining, it requires only four clock cycles. The initiation interval (II) is the number of cycles that must elapse between the start of successive loop iterations.

There is no dependency in the for-loops of the algorithm shown in Figure 1. Therefore, the loop unrolling and loop pipelining may be applied using SDSoC pragmas in order to generate the optimized synthesis results.

```
for(n=0; n<N_pos; n++){          // # of interpolated point
    for(i=0; i<REF_point; i++){          // # of given point
#pragma HLS unroll factor= REF_point
#pragma HLS pipeline II=1
        g[0] = y_ref[i];
        for(j=i, k=1; j>=1; j--, k++){
            output[n] = calculation using upon equation;
            g[i-j+1] = output[n];
        }
        for(j=0; j<=i; j++)   g_present[j] = g[j];
    }
}
```

Figure 1. Pseudo-code of the Neville interpolation algorithm

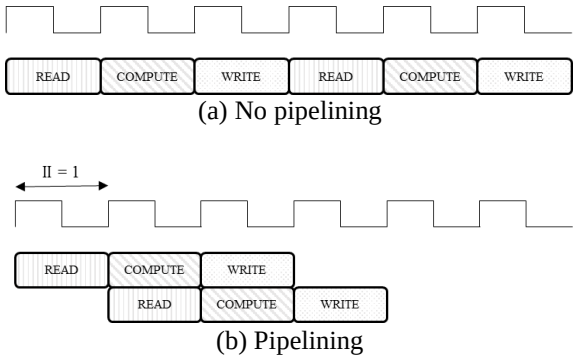

Figure 2. Loop pipelining when the initiation interval is 1

4. Experimental Results

In this paper, we propose a hardware implementation of the Neville interpolation algorithm on a Zynq SoC plaform. The Zynq SoC is equipped with a processing system (PS) based on a dual ARM Cortex^{TM}-A9 core and a programmable logic (PL) with a Xilinx FPGA device in a single chip. PL consists of configurable logic blocks (look-up tables (LUTs), flip-flops, and cascadable adders), Block RAM, DSP blocks and etc. By using PL, behavioral description of a design can be compiled into the target specific custom hardware.

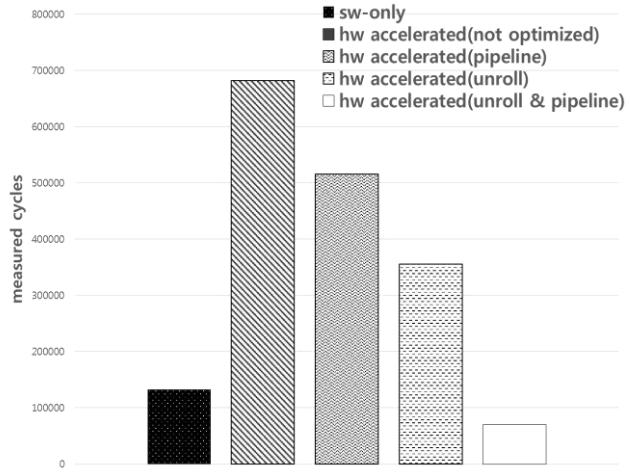

Figure 3. Performance comparison of interpolators

The interpolator finds 30 interpolated points when the number of the given points is 10. We compare the performance and the required hardware area for various implementations. The number of execution cycles of each implementation is used for performance comparison and the required area for each hardware implementation is estimated based on the utilization number of the LUTs in PL.

Experimental results on performance are shown in Figure 3. When hardware is not optimized, performance is worse than the software-only implementation. It is mainly because PL resources are not efficiently utilized without optimization. Figure 4 shows area comparison results. The required area of the fully optimized implementation is more than 5 times larger than the one without optimization. However, the performance is improved approximately by 10 times. In addition, the fully optimized implementation is nearly twice faster than the software-only one.

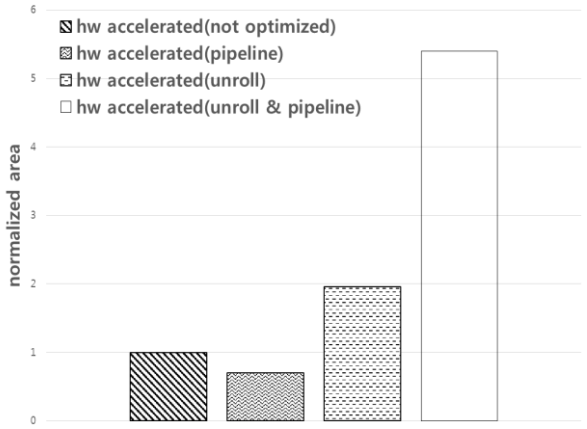

Figure 4. Area comparison of hardware implementations

5. Conclusion

We implement the Neville interpolation algorithm on a Xilinx Zynq SoC platform by using a high-level synthesis tool called SDSoC. Optimization of hardware implementation is important to improve the performance of the synthesized design. There is a trade-off between the performance and the area when the hardware is optimized. With optimization applied, the required area increases while the performance is improved. The proposed Neville interpolator implementation with the optimization techniques shows 1.88 times better performance than the software-only implementation. We conclude that the applied high-level synthesis method is very effective in finding the optimal design through fast design space exploration.

References

[1] P. Coussy and A. Morawiec, *High-Level Synthesis*, Springer, 2010.

[2] V. Kathail *et al*, "SDSoC: A Higher-level Programming Environment for Zynq SoC and Ultrascale+ MPSoC," *in Proc. of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 4, Feb. 2016.

[3] Xilinx, *Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*, 2016

[4] E. H. Neville, *Iterative interpolation*, St. Joseph's IS Press, 1934.++