Hardware Acceleration with Runtime Reconfiguration

Zeqiang Zheng

University of California, Irvine, zeqiangz@uci.edu

Sitao Huang

University of California, Irvine, sitaoh@uci.edu

Compared with CPU-based and GPU-based computing solutions, hardware acceleration using FPGA can provide lower latency and higher energy efficiency. However, as the emerging computing applications are becoming more and more complicated, the effective utilization of FPGA on-chip resources has become more and more challenging. This project will study a universal runtime reconfigurable computing scheme on FPGA.

CCS CONCEPTS • Reconfiguration • Accelerator • High-level Synthesis

Additional Keywords and Phrases: System-on-Chip Design, Design Automation

1 INTRODUCTION

Recent years have witnessed the golden age of computing. Emerging computing applications are becoming more and more complicated and the demand of hardware acceleration is increasing very quickly. FPGA, which is able to work as an accelerator with lower latency and higher energy efficiency, attracts the attention of the industry and the researchers.

For an accelerator, the computation performance can be affected by many factors including input data distribution, energy constraints, processing throughput or latency constraints. According to

different environment conditions, we may need a customized computation mode to get adapt to the requirement of the environment. And it would help a lot if a runtime reconfiguration can be achieved.

The problem that this project deals with is to accelerate the computation of vector addition with FPGA. This project is trying to create two computation mode of FPGA to achieve vector addition acceleration with runtime reconfiguration. One mode is "high performance mode", also called hp mode in the following parts of this paper, which tries to utilize more hardware resources to get higher performance. The other one is "resource-saving mode", also called rs mode in the following parts of this paper, which can be used under a heavy resource constraint. With the runtime reconfiguration using the these designs, the system may have the ability to work well under at least to kind of environments. Therefore, the computing schema that this demo illustrates can have the potential to be widely used in many scenarios including could computing, data center, and edge computing like Internet-of-things devices, which is the main contribution of this project.

2 RELATED WORKS

As the reconfiguration has become a rising hot topic in recent years, there are already many previous research works in this fields, including using dynamic partial reconfiguration to map large workload, and the task-based scheduling as well as the ILP-based solution. However, previous attempts have focused on a particular application or domain of applications [1]. And the computation schema in this project is insensitive to application and have the potential to be widely used.

3 METHODOLOGY & EXPERIMENT

This section introduces the methodology and the experiment of the vector addition accelerator. There are 3 main development stages in the whole flow.

3.1 Application Implementation

Generally speaking, the first step is to implement the application to be a hardware. To achieve a reconfiguration, at least 2 versions are needed. Although preparing more candidate overlays is better for the real-world application, this project will just contain two candidate overlay designs as demo. Another simplification is that we choose vector addition as a sample application which is simple but good enough to show the flow and the difference between difference computation modes. To accelerate the development, the Xilinx tool "Vitis HLS " is used to create HLS hardware design for the two implementations. Using the official document as a reference[2], two solutions are created.

Here is the HLS C code of the hp mode solution, as an example.

```
#include "vector_add.h"
int vector_add(volatile int a[1024], volatile int b[1024], volatile int c[1024])
{
#pragma HLS INTERFACE m_axi depth=1024 port=a offset=slave bundle=DATA1
#pragma HLS INTERFACE m_axi depth=1024 port=b offset=slave bundle=DATA2
#pragma HLS INTERFACE m_axi depth=1024 port=c offset=slave bundle=DATA3
#pragma HLS INTERFACE s_axilite register port=return bundle=ctrl

        int i;
        int arrayA[1024];
        int arrayB[1024];
        int arrayC[1024];

        for(i = 0; i < 1024; i++)
        {
#pragma HLS PIPELINE
                arrayA[i] = a[i];
                arrayB[i] = b[i];
                arrayC[i] = 0;
        }

        for(i = 0; i < 1024; i++)
        {
                arrayC[i] = arrayA[i]+arrayB[i];
        }

        for(i = 0; i < 1024; i++)
        {
#pragma HLS PIPELINE
                c[i] = arrayC[i];
        }
        return 0;
```

}

The code shows some important features of the implementation of HP mode. To achieve high performance, HP ports should be used as the data interface. Pragmas optimization are also added to pipeline the loops. And there are internal buffers, which were arrayA, arrayB and arrayC, to cut off the complicated read-write back flow. In the rs mode, where resources are quite limited as we assumed, the implementation will not use the hp port and the pipleline as well as the internal buffers in order to use less resources. We can find the simulation report in Vitis HLS, which provides the resource usage estimation information.

3.2 SoC Design

After the development of the vector addition IP, the next step is to integrate the vector addition IP with the ZYNQ 7000 Processing system to get an entire System-on-chip design. These works are conducted on Vivado. Fig.2 is an overview of the block design. After the SoC design, the overlays will be exported and then the preparation of overlays is completed.

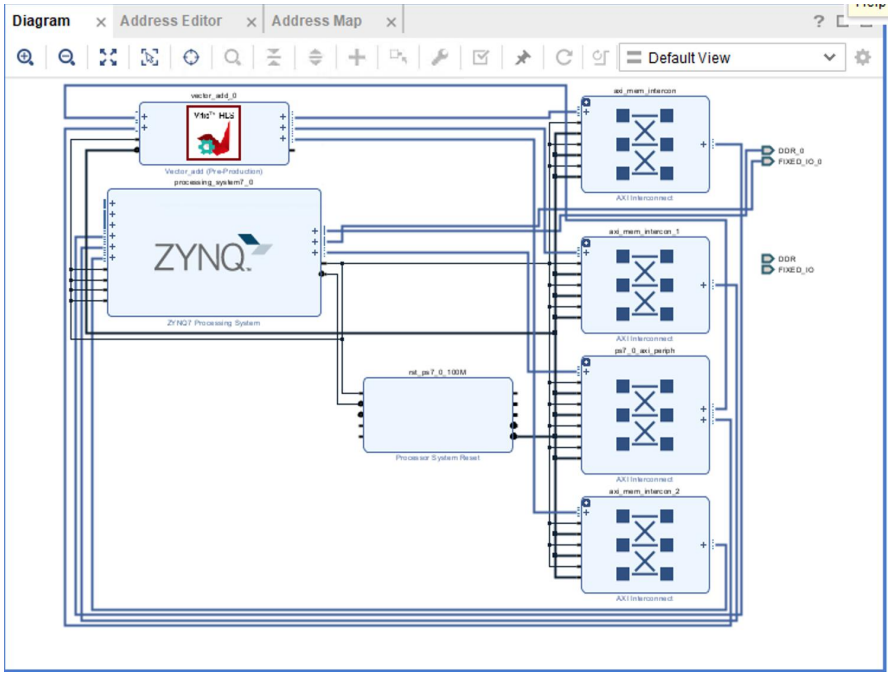

Figure 2: The block design of the System-on-chip integrated with the vector addition IP

3.3 Host programming with PYNQ

This is the most important step in the whole flow. In previous steps, two implementations are prepared and the overlays has been exported from the Vivado tool. In this step, they will run on the PYNQ–Z2 board, which is shown in Fig 3, and the PYNQ library (v2.7) [3] will be called to achieve runtime reconfiguration and test the correctness of the result.

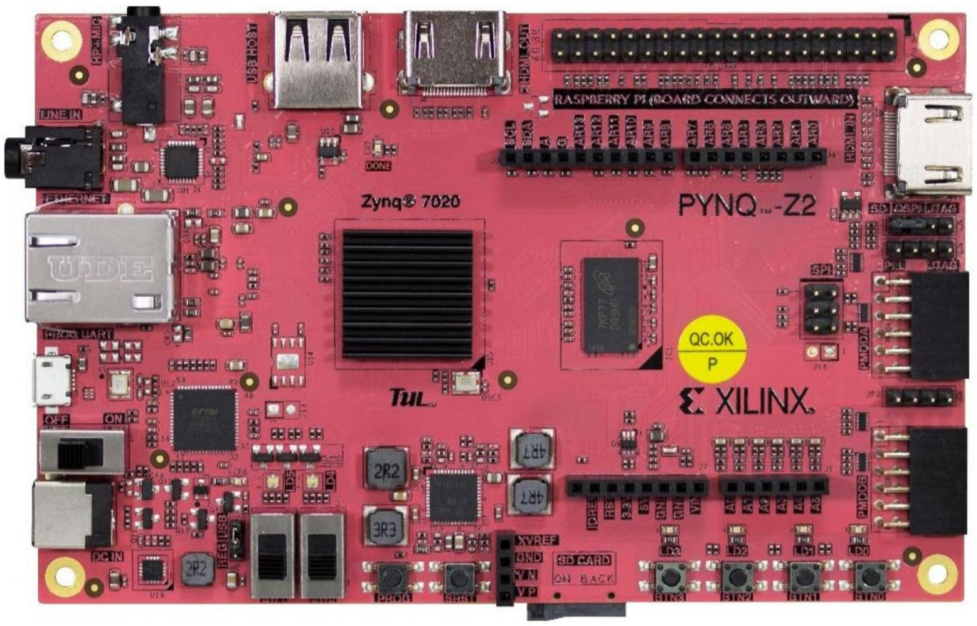

Figure 3: PYNQ–Z2 board

In this step, a python script is created on Jupyter Notebook, which call some of the pynq libraries to complete the host programming. The “Allocate” is called to allocate the shared memory that is visible to the PL at the host side in order for the preparation to feed the input data. The “overlay” library is called to operate the overlays. After the pointers to the overlays are created, the download method can be used in anywhere of the script to switch the overlay during runtime to finish the reconfiguration. As for the signal that control the start of reconfiguration in this demo, we can just use an flag argument in the if statement that has the reconfiguration inside to work as a simplified case. In the real world case, the value of this flag variable may be provided by the outside, for example, a programmable detecting

device. The python script will also work as a benchmark in which we insert time stamps before and after the acceleration to measure the actual performance.

4 CONCLUSION & FUTURE WORK

This project illustrates a universal runtime reconfiguration acceleration computing schema, where the demo deals with the vector addition acceleration problem under two scenarios. It works under two possible mode, the high performance mode under high performance demand and the resource saving mode under the resource constraint condition.

As a immature project, there are many potential optimization directions for further research. Here are three examples of them. First, a more complicated application can be considered. This will make this universal schema more meaningful as an engineering application in the real world. Potential candidates can be matrix multiplication, convolution and even machine learning applications rather than a simple vector addition. Second, more candidate FPGA overlays can be added so that there will be more flexibility for users to choose the best overlay to get closer to the bets performance or other acceleration goal. Third, partial reconfiguration is also a good further optimization choice, with which the whole computation system will not stop computation during the runtime and the reconfiguration can be faster.

REFERENCES

[1] A. Dhar, M. Yu, W. Zuo, X. Wang, N. S. Kim and D. Chen, "Leveraging Dynamic Partial Reconfiguration with Scalable ILP Based Task Scheduling," 2020 33rd International Conference on VLSI Design and 2020 19th International Conference on Embedded Systems (VLSID), 2020, pp. 201–206, doi: 10.1109/VLSID49098.2020.00052.

[2] Viitis High–level Synthesis User Guide. DOI: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_2/ug1399–vitis–hls.pdf

[3] PYNQ: Python productivity for Xilinx platforms. DOI: https://pynq.readthedocs.io/en/latest/pynq_overlays.html

[4] Chelsea Finn. 2018. Learning to Learn with Gradients. PhD Thesis, EECS Department, University of Berkeley.

A APPENDICES

You can find the project files in this link:https://github.com/Zheng–Zeqiang/vector_addition_project