本科生毕业设计（论文）

题　　目：

Runtime Environment Aware Self-Adaptive Hardware Acceleration With Dynamic Reconfiguration

姓　　名：	郑泽强
学　　号：	11812716
系　　别：	深港微电子学院
专　　业：	微电子科学与工程
指导教师：	陈全、黄思陶

2022 年 5 月 16 日

诚信承诺书

1.本人郑重承诺所呈交的毕业设计（论文），是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料均真实可靠。

2.除文中已经注明引用的内容外，本论文不包含任何其他人或集体已经发表或撰写过的作品或成果。对本论文的研究作出重要贡献的个人和集体，均已在文中以明确的方式标明。

3.本人承诺在毕业论文（设计）选题和研究内容过程中没有抄袭他人研究成果和伪造相关数据等行为。

4.在毕业论文（设计）中对侵犯任何方面知识产权的行为，由本人承担相应的法律责任。

作者签名 郑泽强

2022 年 5 月 16 日

Runtime Environment Aware Self-Adaptive Hardware Acceleration With Dynamic Reconfiguration

郑泽强

深港微电子学院　指导教师：陈全

加州大学欧文分校工程学院 联合指导教师：黄思陶

[摘要]：与基于 CPU 和 GPU 的计算解决方案相比，使用 FPGA 进行硬件加速可以提供更低的延迟和更高的能效。然而，随着新兴的计算应用变得越来越复杂，FPGA 片上资源的有效利用变得越来越具有挑战性。本项目将研究一种基于 FPGA 的通用运行时可重构计算方案，达到运行环境自适应的硬件加速效果。该方案包括一个基于高级综合的优化硬件加速器设计、一个完整的硬件设计与系统编程工作流程和一个通用的动态运行时重构调度策略，以实现环境自适应。

[关键词]：异构计算；可重构计算；软硬件协同设计；高层次综合；电子设计自动化；硬件加速

[ABSTRACT]: Compared with CPU-based and GPU-based computing solutions, hardware acceleration using FPGAs 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 studies a universal runtime reconfigurable computing scheme on FPGA which can achieve hardware acceleration while getting adapt to the runtime environment changes. The scheme includes an optimized hardware accelerator design based on high level synthesis(HLS), a complete hardware design and host programming workflow and a universal scheduling strategy for dynamic runtime reconfiguration to achieve environment-aware accelerator self-adaption.

[Keywords]: Heterogeneous Computing; Reconfigurable Computing; Hardware Software Co-design; High Level Synthesis; Electronic Design Automation；Hardware Acceleration

Contents

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 can work as an accelerator with lower latency and higher energy efficiency compared to CPUs and GPUs, attracts the attention of the industry and the academia.

For an accelerator, FPGA's computation performance can be affected or constrained by many factors including input data distribution, energy constraints, processing throughput or latency constraints, etc. An accelerator with fixed architecture may not be able to remain its great performance under varying environment conditions. 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.

Dynamic Partial Reconfiguration(DPR) technology enables the implementation of adaptive hardware systems that can modify their behavior dynamically during runtime at the hardware level to adapt to their surroundings[1], thus it is adopted in this project to implement the environment self-adaptive system.

This project is trying to create a scheme to apply runtime dynamic reconfiguration to a generic scenario. In the prototype demo, vector addition acceleration is chosen as an example of application scenario. The prototype system includes candidate FPGA overlays that can accelerate vector addition, software controller that can control the reconfiguration and a reconfiguration scheduler to select the best overlay among candidate FPGA overlays.

The main contributions of this project can be summarized as follows:

1) a universal and expansible runtime reconfigurable computing scheme on FPGA. The computing scheme is for vector addition, as an example in this project. The hardware part is an optimized FPGA accelerator for vector addition while the software part schedules and controls the reconfiguration.

2) a complete and flexible hardware design and host programming workflow that

can apply the proposed computing scheme rapidly to a wide range of applications. For hardware part, the high-level syntheses tool reduce the development time greatly. For software part, the structure of scheduler is relatively fixed and the crossover points estimation can be automated.

3) a universal scheduling strategy for dynamic runtime reconfiguration to achieve environment-aware self-adaption. The only input features of the scheduler are environment vectors, which means the scheduling is also not application-specific.

The advantages of the scheme are universality and scalability. First, the scheme is insensitive to applications. The computing schema that this demo illustrates have the potential to be widely used in many scenarios including could computing, data center, and edge computing like Internet-of-Things(IoT) devices. Moreover, the scheme can be expanded easily by creating more candidate overlays for reconfiguration. This advantage will be even more significant when the prototype system are moved into a more complex working environment. The challenges of this project may include the latency caused by the reconfiguration and the difficulty of finding optimal reconfiguration scheduling strategy.

The rest of paper is organized as follows. Section 2 introduces and compare this project with previous attempts of applying dynamic partial reconfiguration in hardware acceleration and environment-aware self-adaption. Some directions of further optimization are also discussed according to related works. Section 3 defines the problem that this project is dealing with and explained the key points of the project. Section 4 explains the methodology of the prototype system design, which includes each part of the hardware design-host programming design flow: application implementation based on high level synthesis, System-on-chip(SoC) design based on Vivado and the host programming based on PYNQ. The key point is the accelerator controller and scheduler design. Section 5 introduces how the experiment platform is built and how the data are collected and arranged. Section 6 discusses the challenges in this project and the limitations of the current prototype and proposes some directions of further optimization. Section 7 concludes the whole paper.

2. Related Works

As the FPGA-based hardware acceleration is becoming a rising hot topic in recent years, there are some FPGA-based runtime reconfiguraton works in this fields. Dynamic Partial Reconfiguration has been applied for environment-aware runtime adaption, system cost reduction with time multiplexing, reliability and integration mechanism for hardware accelerators within general purpose computing systems[1]. The previous attempts of using Dynamic Partial Reconfiguration to implement environment-aware self-adaptive systems are successful in many fields including software defined radio, data analytics and driver assistance systems[1]. Varying compute modules of Partial Reconfiguration systems can be loaded as needed at runtime, which enables computing systems that deal with changing environments.

Examples of combining dynamic partial reconfiguration with accelerator includes [2] and [3]. In [2], the authors design a flexible system that can easily modify the CNN accelerator at run time using dynamic partial reconfiguration to adapt to different networks. In [3], the author further combined the dynamic partial reconfiguration with high level synthesis and hardware/software co-design. The paper present a workflow designing CNN layer specific hardware blocks and reconfiguring them during runtime, which demonstrate the possibilities of our hardware/software codesign approach and hardware reconfiguration in the Deep Landmark Detection application.

The idea of [2] that modifying the accelerator during runtime and the idea of hardware/ software co-design in [3] are adopted in the project. In [2] and [3], the reconfiguration is conducted according to layer characteristics of neural networks, and the neural networks can be revised for hardware acceleration. In our project, the software application algorithm will not be revised for hardware acceleration, which increases the flexibility of the prototype system when a new application comes up.

Some previous works also show the direction of further optimization of this project which includes reconfiguration partition, task scheduling optimization and hardware/software co-design workflow optimization. In [4], Reconfigurable Partitions

(RP) allows the system to get adapted for an optimized execution according to the application, as well as the application phase. Together with partial reconfiguration, task scheduling are explored in [5]-[10]. [9] and [10] shows the Integer Linear Programming (ILP) based scheduling and further optimization with heuristic approach. [11] proposed a Python-based FPGA programming and synthesis flow with much higher level of abstraction than all existing hardware design flows that can greatly decrease the development time.

3. Problem Definition

Imagine an application scenario where the user needs to accelerate vector addition under an environment with continuously changing power constraints and throughput expectations. The purpose of this project is to develop a flexible computing solution for the this type of application scenarios. The final prototype is expected to be a computing solution for vector addition acceleration consisting of an FPGA based hardware accelerator and corresponding software controller for environment-aware self-adaption and scheduling.

The change of environment will be simulated by an environment simulator, which will provide environment vector that contains selected environment variables including power requirement and throughput expectation to the environment variables. This project aims to find an effective scheduling strategy and implement an effective corresponding hardware design and host programming workflow to accelerate the computation of vector addition with FPGA.

The key problem and the difficulty of this project is to find the proper reconfiguration scheduling strategy, which means a proper function to map an environment vector to a number that represents a candidate overlays. The design of the scheduler will be discussed in Section 4 and 5.

4. Methodology

4.1 Overview

This section introduces the methodology of the vector addition acceleration in different stages of the hardware/software co-design flow. There are three main development stages in the whole flow. The whole structure overview of the system including hardware part and software part are shown in Figure 1 below.

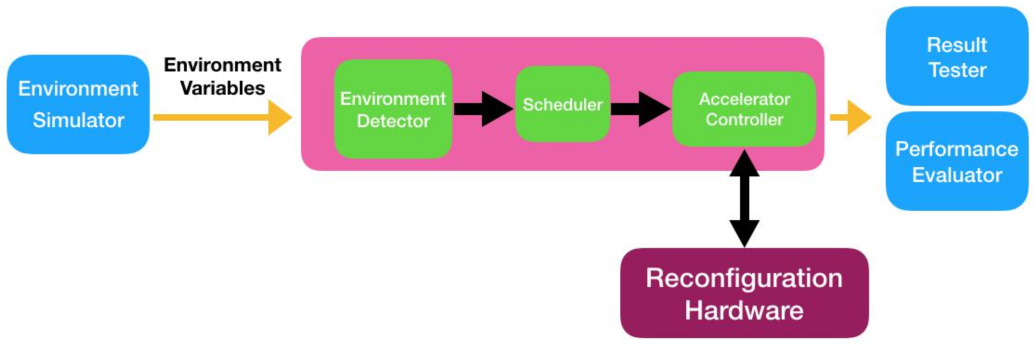

Figure 1: System Overview

In the figure 1, the purple part is hardware and the others are software. The pink block is implemented in host programming with the PYNQ framework and the purple part is designed by Xilinx Vitis High Level Synthesis (HLS) and Vivado design tools. The blue parts are external test environment. They are not part of the prototype product.

As for the hardware design, the overview of the workflow and corresponding data interface are shown in Figure 2 below. The details will be discussed in the following sections.

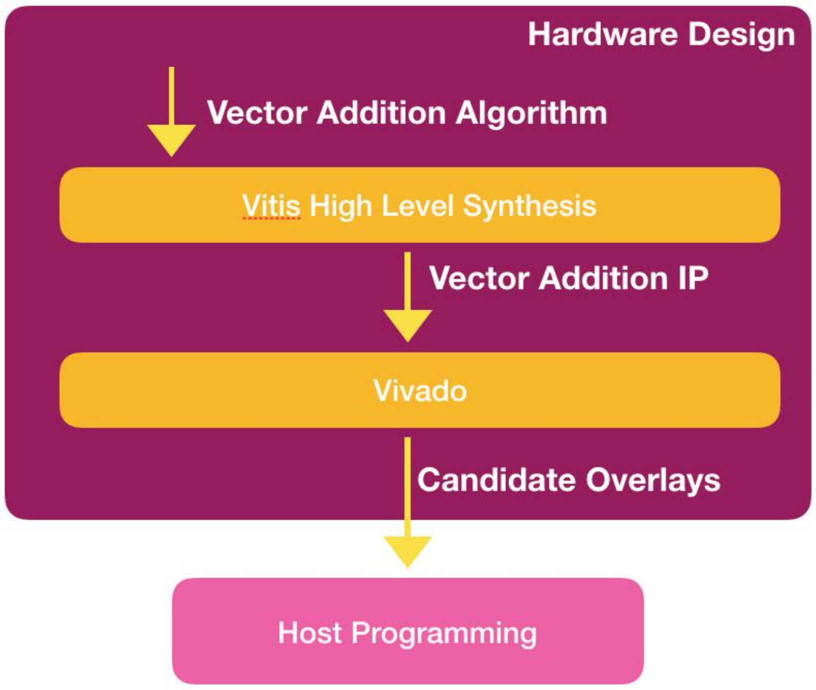

Figure 2: Overview of hardware development flow and corresponding data interface

4.2 Application Implementation with HLS

4.2.1 General Structure

This project is trying to create two computation mode of FPGA accelerators to achieve vector addition acceleration with runtime reconfiguration. One mode is "high performance mode", also called *HP* mode, which tries to utilize more hardware resources to get higher performance. The other one is "resource-saving mode", also called *RS* mode, which can be used under a heavy resource or power constraint.

According to the design flow, the first step is to implement the application with hardware accelerators, that is, to implement the vector addition algorithm in hardware using Vitis High Level Synthesis(HLS). To achieve runtime reconfiguration, at least 2 candidate computing modules are needed. Although preparing more candidate overlays is better for the real-world application, this project will use two candidate overlay designs for demonstration purpose. 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. With the official document as a reference[12], two solutions are created.

With an optimized vector addition accelerator (HP mode) as the example, Figure 3 shows its brief overview of the Finite State Machine with Datapath(FSMD) of the vector addition hardware accelerator.

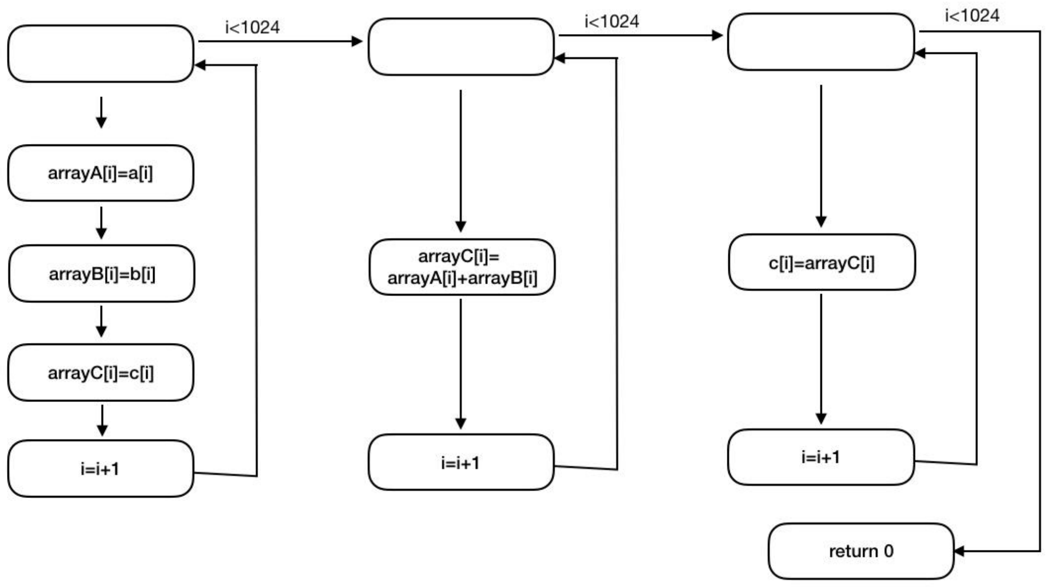

Figure 3: FSMD of the vector addition hardware accelerator

4.2.2 Optimization

In order for higher performance, some HLS pragmas are applied to optimize the design, which include high performance interfaces settings as below and input&output buffers as well as the parallelization pragmas in the buffer loop. With internal buffers, memory access can be pipelined in burst mode and the performance can be improved further.

```
#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
#pragma HLS PIPELINE
```

Code 1: Pragmas to optimize application implementation

While in the resource-saving mode, where resources are quite limited as we assumed, the implementation will not use the HP port, pipleline as well as the internal buffers in order to use less resources.

4.3 System-on-Chip 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(SoC) design. This integration can be done in Xilinx Vivado tool. Figure 4 is an overview of the block design. After the SoC design, the overlays will be exported and then the overlays are ready to use.

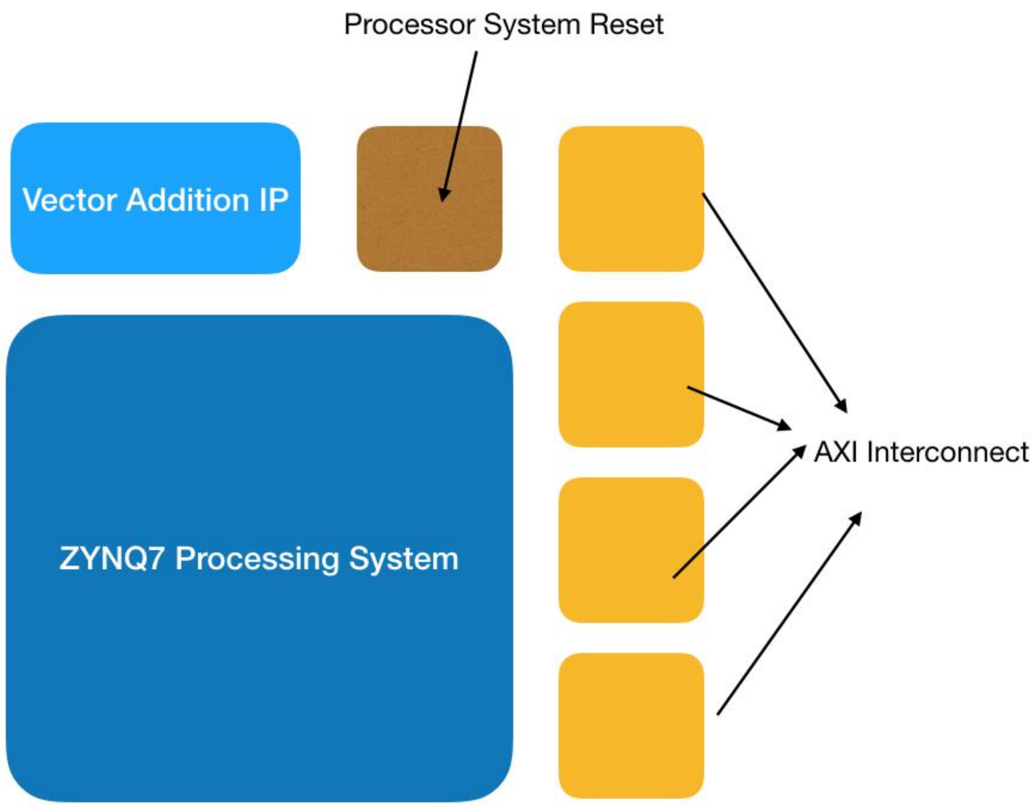

Figure 4: The Overview of SoC block design without connection

4.4 Host programming with PYNQ

4.4.1 Overview

The system is implemented with a PYNQ-Z2 board, as shown in Figure 5. The host programming part is based on PYNQ programming framework.

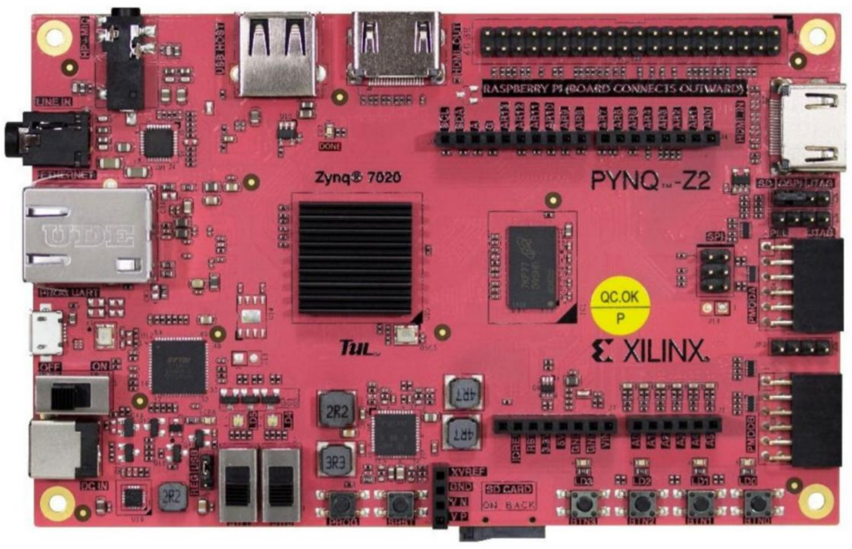

Figure 5: PYNQ-Z2 board

The host programming environment includes PYNQ library v2.7.0 and Jupyter Notebook[13]. Jupyter Notebook is integrated for host programming with Python. Host programming is done within Jupyter Notebook. PYNQ is an open-source software framework that provides a development platform based on Python for Xilinx's FPGA SoC including ZYNQ-7000. PYNQ-Z2 board, as the experiment platform in this project, is supported by PYNQ.

The internal data interface of the prototype between different internal modules are shown in Figure 6.

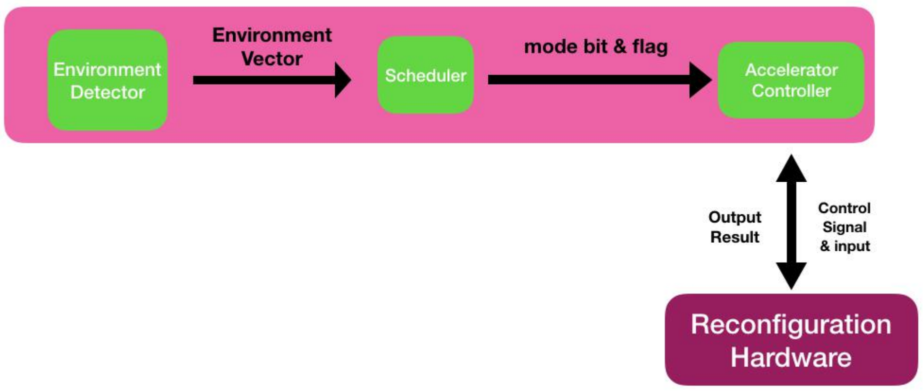

Figure 6: Overview of Internal Data Interface of Host Programming

As shown in Figure 6, the scheduler decides the mode bit that represents the computing mode with corresponding overlay and the flag value, which is a Boolean value that tells the system whether to change the current computing mode or not. The hardware communicates control signals and input/output data with the accelerator controller through shared memory and registers. The simulator provides the environment vector whose elements are environment variables like resource limit and temperature. When the Python script starts to run, environment simulator will start first. If the tool is working in the real world, this simulator will receive the observation results from the environment detectors or sensors.

The external data interface of the prototype system is shown in Figure 6 as below.

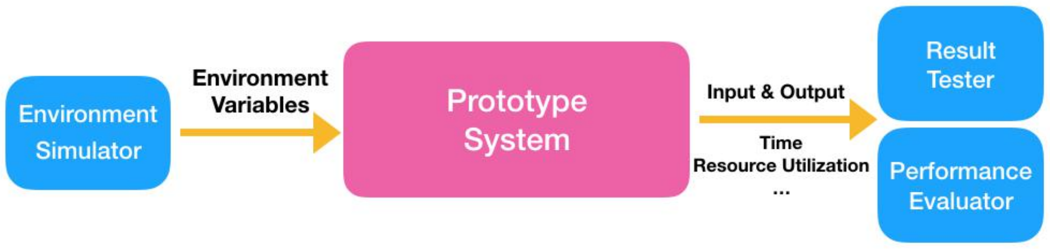

Figure 7: Overview of External Data Interface of Host Programming

As shown in Figure 7, blue parts are artificial test environment. The Environment Simulator, result tester and performance evaluator are all a part of the host programming. More details will be discussed in the experiment section.

4.4.2 Accelerator Controller

The accelerator controller is one of the most important parts in the whole system. Figure 8 shows the behavior of the accelerator controller.

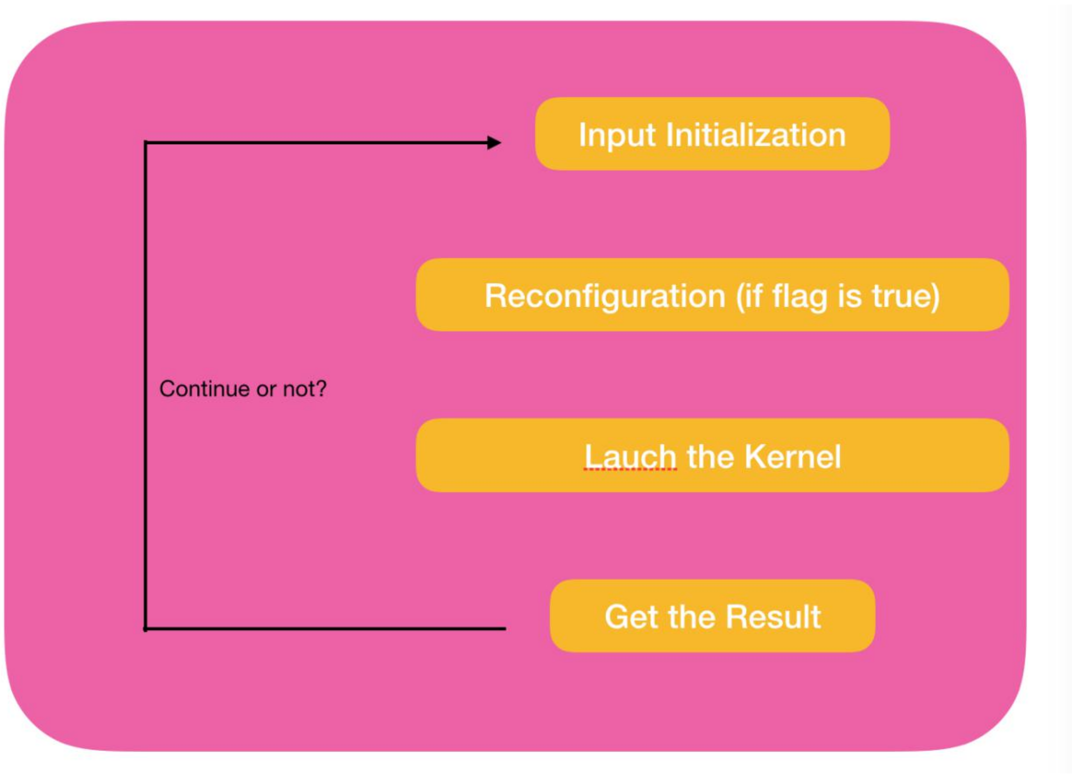

Figure 8: behavior of the accelerator controller

There are four main steps in the accelerator controller. In “Input initialization”, the Processing System(PS) loads the input vector into the PS-PL shared memory, that is to copy the contents of the host buffer into the mirrored device buffer. The accelerator controller will also get the scheduling result from the scheduler. If the flag, which is an output of scheduler, is true, then the hardware reconfiguration is needed. After that, the kernel is launched through writing start signal into the corresponding register. The PS need to wait for the PL until the computation of this batch of input is competed. After the acceleration is done, in the “get the result” step, the contents of the device buffer, which is the output result, will be copied into the mirrored host buffer. All these steps are contained in a loop that will continue until the break condition, for example, the preseted maximum work time is reached, are met.

PYNQ libraries are called frequently in the design of acceleration controller. to support the host programming. The “allocate” function is called to allocate the shared memory that is visible to both the PL and 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 a flag argument in the if statement that has the reconfiguration inside to work as a simplified case. In real-world cases, the value of this flag variable may be provided by the environment, 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.

4.4.3 Scheduler Design Methodology

In previous steps, two implementations are prepared and the overlays that has been exported from Vivado will run on the PYNQ-Z2 board. The scheduler provides a scheduling function to decide which candidate to choose next. The scheduler function can remain undetermined in experiments, where the environment will continue changing as experiment variables and the computation performance will vary according to the experiment variables. Then the scheduling function can be specified according to the reconfiguration policy.

The scheduler contains a scheduling function that can decide the best overlay to choose under given environment vector. The function can map the environment vector to a value of mode bit and may not be explicit. The function is implemented by an if-statement decision branch like the following structure:

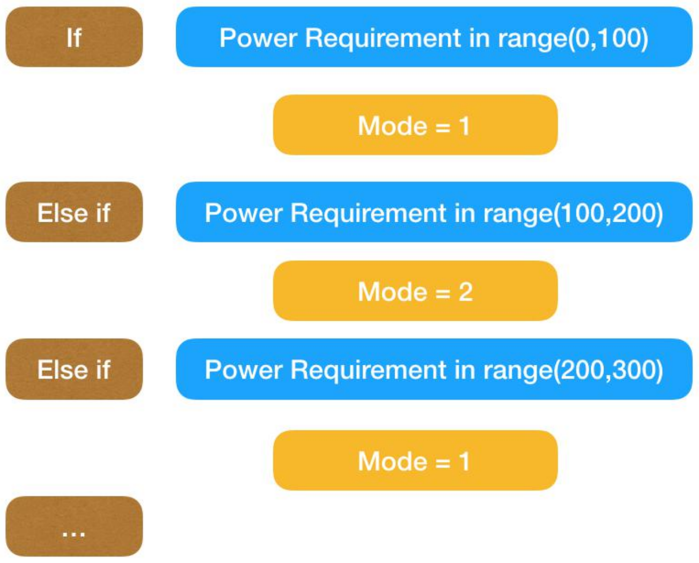
If
Power Requirement in range(0,100)
Mode = 1
Else if
Power Requirement in range(100,200)
Mode = 2
Else if
Power Requirement in range(200,300)
Mode = 1
...

Figure 9: Decision Branch that maps environment variable to mode bit

Figure 9 shows the core part of scheduler, the decision branch that maps environment vector to the mode bit. In this example, the selected environment variable under consideration is the power requirement. The range of selected variables in the if statement condition is divided by crossover points, which are 100, 200, 300... in this example. Crossover points are critical points where the best candidate overlay changes, and its position is defined to be an environment vector.

It is hard to find the specific crossover points, but an estimation of range is feasible under a simplified environment-performance relation model. A simple diagram of this model is shown in Figure 8 below.

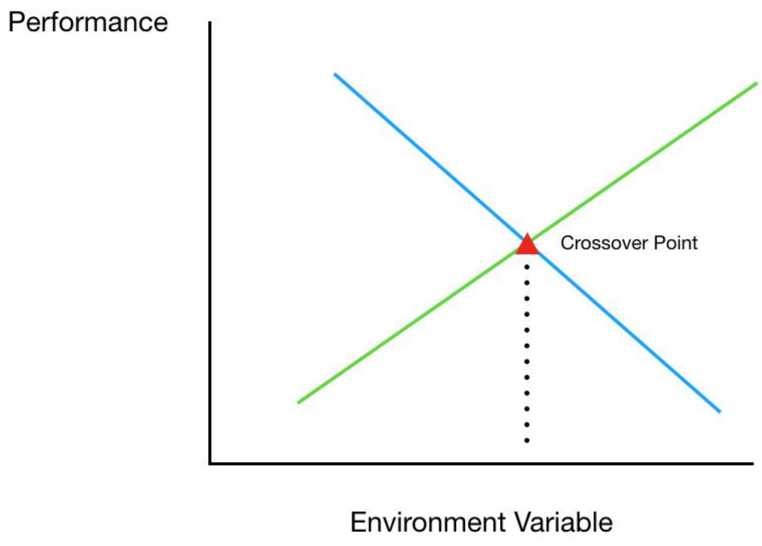

Figure 10: a simplified environment-performance relation model

Figure 10 describes a simplified case of local parts near the crossover point in the environment-performance relationship model. The environment variable and the variable to describe performance have not been determined. The idea is that, for example, if mode 1 works better with large input size and mode mode 2 works better with small input size, and the crossover must happen in some place when the input size is becoming larger. Therefore, it is feasible to choose test points in Figure 8 to estimate the range of the location of crossover point. The details of crossover point estimation will be further discussed in the Experiment Section 5.4.

5. Experiment

5.1 Experiment Design Overview

As shown in Figure 6 above, the artificial experiment platform is built in host programming framework with Python. The platform includes an environment simulator to provide environment variables, a result tester to compare the input and output vectors and decide the correctness and a performance evaluator to measure the time and resource utilization as well as some other aspects of performance of the prototype system. The implementation is shown in the host programming code above.

The flow of experiment includes the following steps. First, the two candidate overlays need to be tested separately until the result of vector addition is correct and their performance information will be recorded. This information will be compared along with the variation of selected environment variables and the scheduling strategy will be decided. The final step is to test the reconfiguration as well as the scheduler and compare its performance with the case without reconfiguration.

5.2 High Level Synthesis Result

According to the report of VItis HLS, the expected hardware parameters of two computing modes are shown in the table below. The comparison shows that the input size of 1024 is not large enough to utilize the advantage on performance of high performance mode.

Computing Mode	Target	Estimated	Uncertainty
Plain Mode	10.00ns	5.806ns	2.70ns
High Performance Mode	10.00ns	7.300ns	2.70ns

Table 1: Timing Estimate Comparison with input size 1024

Computing Mode	Latency Cycles	Latency/ns	Interval	BRAM	DSP	FF	LUT	URAM
Plain	1027	1.027E4	1028	0	0	71	103	0
High Performance	3097	3.097E4	3098	18	0	2464	2892	0

Table 2: Performance&Resource Utilization Comparison with input size 1024

5.3 Result Test

The design of result tester is a part of host programming based on Python as shown below. The output of the hardware will be compared with the result of regular addition. The correctness Boolean value will be true only when each element of the output array is correct.

```
correctness = True
```

```
          for i in range(1024):
              if
      (array_c[i]==array_a[i]+array_b[i]):
                  continue
              else:
                  correctness = False
```

Code 2: Pseudo code of result tester

5.4 Performance Evaluation

The performance evaluator starts to run after the result tester provide the correctness Boolean variable with True value.

As mentioned in problem definition section, the latency caused by reconfiguration can be significant. The latency provided by Vitis HLS synthesis report is not comprehensive. To evaluate the latency of the whole acceleration system, time stamps are inserted before the kernel is launched and after the acceleration is completed. To statistically reduce the error of stochastic factors, test input vectors should be large enough. The latency of accelerator can then be calculated by subtraction of the two time stamps. The data that needs to be collected includes latency, resource utilization, input size, power requirement and throughput requirement. Repeat this step for the two overlays.

5.5 Scheduler Design

To find the crossover points for an environment variable, rearrange the data and build a list, of which each element, as an object, has two members (or 5 members): an environment vector and a performance vector. The list is arranged in ascending order of an chosen environment variable.

Compare each element of the two lists, when the environment vector is the same, leave only the record with better performance and add it into a new list. For each element in the new list, a new member should be added to record which computing mode the remaining element is related to. The structure of the new list is shown below in Table 3.

Chosen Mode	Performance Vector: (Latency, Resource Utilization)	Environment Vector: (Input Size, Power Requirement (Ascending), Throughput Requirement)

Table 3: Structure of Arranged Data

With the data arranged in a list in the ascending order of one environment, a function find_element() can then be built. The input is list index while the output is all the other members. Then an algorithm can be conducted to automatically find crossover points. Pseudo code in the case of environment variable1 ascending is shown below.

```
i = 1
While i < 1000
   If Chosen_mode(i-1) ==Chosen_mode(i)
      Break
   Else
      i ++
find_element(i)
```

Code 3: Pseudo code to find crossover points

This process needs to be repeated for any change in the environment vector, thus it would be much more convenient to have a Python script to deal with the data collection.

6. Future Works

There are still some challenges remaining to be solved in this project. One of the challenges during the prototype system development may be the latency caused by the reconfiguration. Partial Reconfiguration(PR) may be a good choice to reduce the latency caused by reconfiguration. The candidate overlays are all vector addition accelerators, thus the reconfiguration region can be limited to some parts of the whole accelerator. With partial reconfiguration, the whole computation system will not stop computation during the runtime and the reconfiguration can be faster.

Another difficulty is to finding optimal reconfiguration scheduling strategy. The current method is dependent on a simplified environment-performance model and can not deal with environment variables with more complex relationship. Machine Learning may help in dealing with complex relationship between environment variables and performance parameters.

To further explore the potential of the computing scheme we proposed, a more complicated application scenario 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. We can also add more candidate overlays to scale the system and make it more powerful.

7. Conclusion

This project illustrates a universal runtime reconfiguration acceleration computing flow, where the prototype system implements the vector addition acceleration problem under two types of 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. The prototype system shows the ability of environment-aware self-adaption. The hardware design and host programming workflow that we proposed is generic, and not restricted to a particular domain and have good scalability. To achieve environment-aware self-adaption, a scheduling strategy for dynamic runtime reconfiguration is proposed based on a simplified environment-performance relation model.

References

[1] Kizheppatt Vipin and Suhaib A. Fahmy. 2018. FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications. ACM Comput. Surv. 51, 4, Article 72 (July 2019), 39 pages. https://doi.org/10.1145/3193827

[2] H. Irmak, D. Ziener and N. Alachiotis, "Increasing Flexibility of FPGA-based CNN Accelerators with Dynamic Partial Reconfiguration," 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), 2021, pp. 306-311, doi: 10.1109/FPL53798.2021.00061.

[3] F. Kästner, B. Janßen, F. Kautz, M. Hübner and G. Corradi, "Hardware/Software Codesign for Convolutional Neural Networks Exploiting Dynamic Partial Reconfiguration on PYNQ," 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2018, pp. 154-161, doi: 10.1109/IPDPSW.2018.00031.

[4] 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.

[5] G. Charitopoulos and et. al., "Hardware Task Scheduling for Partially Reconfigurable FPGAs," in Applied Reconfigurable Computing, 2015.

[6] A. Purgato and et. al., "Resource-Efficient Scheduling for Partially- Reconfigurable FPGA-Based Systems," in 2016 IEEE International Parallel and Distributed Processing Symposium Workshops.

[7] E. A. Deiana and et. al., "A multiobjective reconfiguration-aware sched- uler for FPGA-based heterogeneous architectures," in 2015 International Conference on ReConFigurable Computing and FPGAs.

[8] R. Cordone and et. al., "Partitioning and Scheduling of Task Graphs on Partially Dynamically Reconfigurable FPGAs," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2009.

[9] F. Redaelli and et. al., "Task Scheduling with Configuration Prefetch- ing and Anti-Fragmentation techniques on Dynamically Reconfigurable Systems," in 2008 Design, Automation and Test in Europe.

[10]F. Redaelli and et. al., "An ILP Formulation for the Task Graph Schedul- ing Problem Tailored to Bi-dimensional Reconfigurable Architectures," in 2008 International Conference on Reconfigurable Computing and FPGAs.

[11] S. Huang, K. Wu, H. Jeong, C. Wang, D. Chen and W. -M. Hwu, "PyLog: An Algorithm-Centric Python-Based FPGA Programming and Synthesis Flow," in IEEE Transactions on Computers, vol. 70, no. 12, pp. 2015-2028, 1 Dec. 2021, doi: 10.1109/TC.2021.3123465.

[12]Vitis High-level Synthesis User Guide. DOI:https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_2/ug1399-vitis-hls.pdf

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

Appendix

Appendix A

This is an open-source project. You can find the project files in this link: https://github.com/Zheng-Zeqiang/vector_addition_project

Acknowledgements

This research was conducted under the co-supervision of Professor Sitao Huang in University of California, Irvine and Professor Quan Chen in Southern University of Science and Technology. All the work was done in University of California, Irvine. The FPGA development kit and the Linux server are provided by Professor Sitao Huang. Thanks for the great contribution from Prof Sitao Huang and the support from The Henry Samueli School of Engineering at UC Irvine and the School of Microelectronics at Southern University of Science and Technology.