FPGA-Implementation of an online Symbolic Controller Synthesizer using OpenCL and the Atlas-SoC board

Bachelor Thesis

Scientific work to obtain the degree
B.Sc. Electrical Engineering and Information Technology

Department of Electrical and Computer Engineering
Technische Universität München

Supervised by Mr Mahmoud Khaled
Assistant Professorship of Hybrid Control Systems

Submitted by Low Jian How Leon
106B Punggol Field
Apt. 17-534 Singapore 822106
+65 97614376

Filed On München, on 18 June 2018

Abstract

This thesis introduce the use of De0-Nano-SoC board with OpenCL functionality to synthesize online symbolic controller. OpenCL was used due to its computational speed that allows system to be able work efficiently in real time.

Developed by my supervisor Mahmoud Khaled, the OpenCL kernel function is to construct symbolic abstraction of dynamical systems using growth bound and synthesis symbolic controllers using fixed point operations on the symbolic model through over-approximating the reachabile sets.

The De0-Nano-SoC was used due to its easy configuration and avaliability. In this thesis, we would aim to try understand the OpenCL programming framework and coding style and change the source code provided by my supervisor so as to allow it to be implemented on the board.

Acknowledgement

I would first like to thank my thesis advisor Mahmoud Khaled of the Hybrid Control Systems group, Department of Electrical Engineering and Information Technology at the Technical University of Munich(TUM). The door to his office was always available whenever I run into difficulties or questions pertaining to the thesis. He will steer me in the right direction whenever help is needed from him.

I would also like to express profound indebtedness to my parents and friends for providing me with unwavering support and continuous encouragement through the course of researching and writing this thesis. This accomplishment would not have been possible without them.

Contents

Chapter 1

Introduction

1.1 Motivation

For all hardware and software products that are developed, the most important ingredient that determine the success of the products are how reliable and fast it function in real-world applications. In the hybrid control field, the significance of the synthesis of controllers for any hybrid systems has become of utmost importance.

With the continuous fervor to improve the synthesis of controllers, we will look into using different alternatives to expand the options that would benefit the functionality. In this thesis, we will focus on primarily on Open Computing Langauge (OpenCL) [3] a free and widely used Application programming interface(API), which work in tandem with the Computer Processing Unit(CPU) to provide better computation speed. Next we would also look into using Altas-SoC board, De0-Nano-SoC as the base to find out how well and reliable it works for the synthesis of controllers.

1.2 Related Work

Firstly before discussing the usage of the FPGA implementation of an online symbolic controller synthesizer using OpenCL and the Altas-SoC board, there are other softwares that implement the online symbolic controller synthesis that help us understand more on the synthesis of online symbolic controller.

One example would be the open software tool called SCOT[4] that is developed by Dr Matthias Runnger and Dr. Majid Zamani from the department of the Hybrid Control Systems Group in Technical University of Munich (TUM). By using Lipschitz type estimate on the right-hand-side of the differential with different parameters, it is able to create symbolic models for the facilitation of the synthesis of controllers.

Furthermore, it uses C++ as the main language and MATLAB as an user interface for the generation of simulation for the code which help in visualising the controller in movement. This makes it easier for other users to further develop their programs.

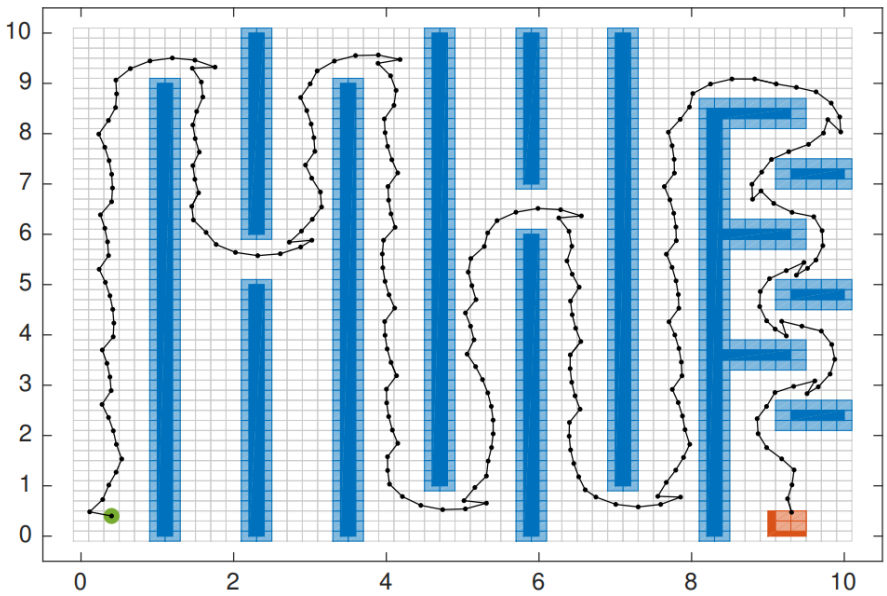

Figure 1.1: Example of the output of the online symbolic controller[4]

1.3 Overview

This paper is structured in the following outline. Chapter 2 gives an short overview on the Altas-SoC board that we are using for the OpenCL program to function as automatically symbolic controllers.

Chapter 3 gives an overview on OpenCL for its usage and its general framework. We would further discuss in details about how we intend to use it for the synthesizes of the the controllers.

In Chapter 4, we will discuss on how we ensure that the OpenCL files could be compiled, and steps undertaken to make the executable file.

Chapter 2

DE0-Nano-SoC

2.1 Introduction

Developed by Terasic.Inc, the DE0-Nano-SoC Development Kit showcase a sturdy hardware design built for the Altera System-on-Chip(SoC) FPGA. It uses the dual-core Cortex-A9 embedded cores with industrial-standard programmable logic to make designing more flexible. [1]

Furthermore, it has a processor, peripherals and memory interface that are able to communicate easily with the FPGA fabric using a high-bandwidth interconnect backbone. Moreover, it has lots of other functions such as ethernet, analog functions.

2.2 Specifications

∗FPGA Device

- Cyclone V SoC **5CSEMA4U236N** Device
- Dual-Core ARM CORTEX-A9(HPS)

∗Configuration sources

- Serial configuration flash-**EPCS128**
- Embedded USB-Blaster II**(JTAG)**

∗Memory Device

- 1GB(2x256Mx16) DDR3 SDRAM

- MICRO SD Card Socket

∗Communication Port

- USB OTG Port(USB Micro-AB connector)
- UART to USB(USB Mini-B Connector)
- 1 Gigabit Ethernet PHY with RJ45 Connector

∗Connectors

- Two 40-pin Expansion Headers
- One 10-pin ADC I/P Header
- One LTC Connector

∗Switches, Buttons and Indicators

- 3 User Keys
- 4 User switches
- 11 User LEDs
- 2 HPS Reset Buttons

∗Sensors

- G-Sensor on HPS

Layout Block Diagram

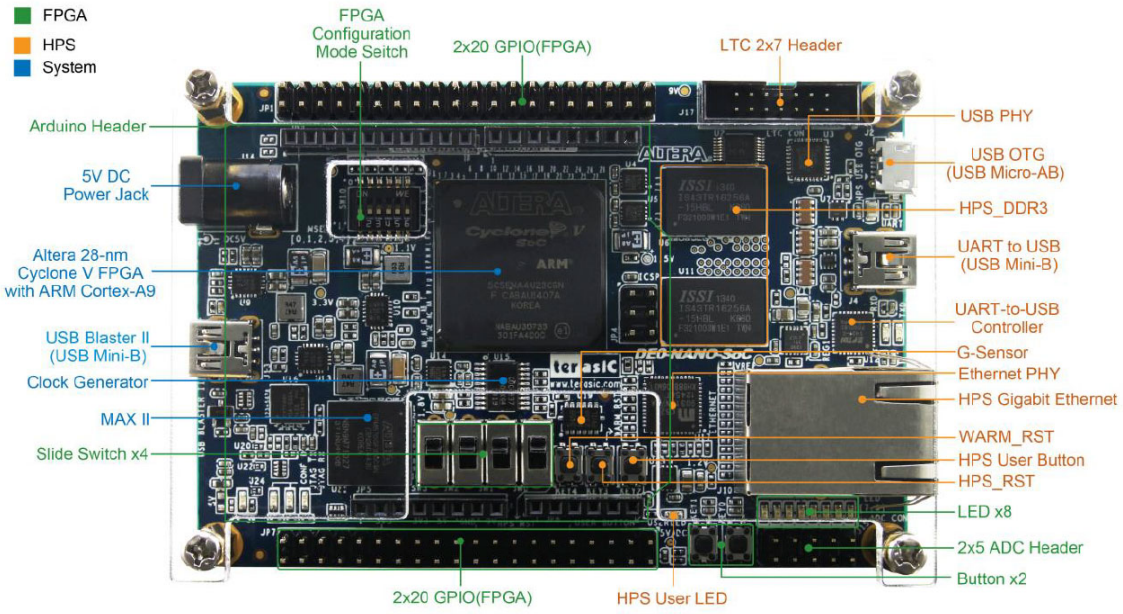

Figure 2.1: Front layout of DE0-Nano-SoC

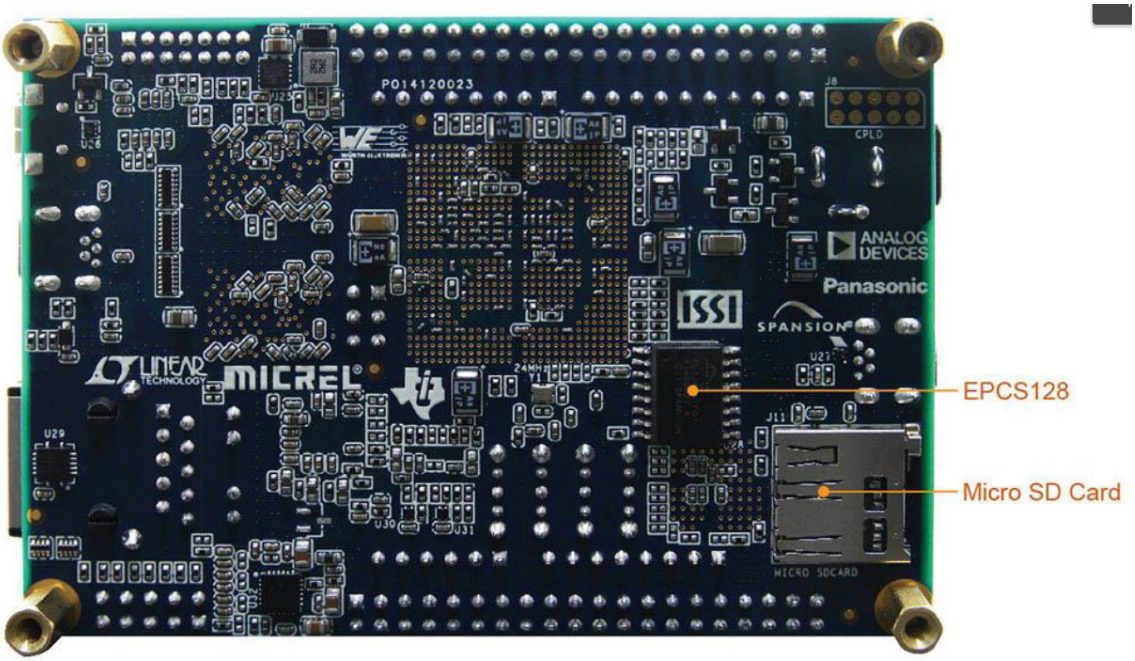

Figure 2.2: Back layout of DE0-Nano-SoC

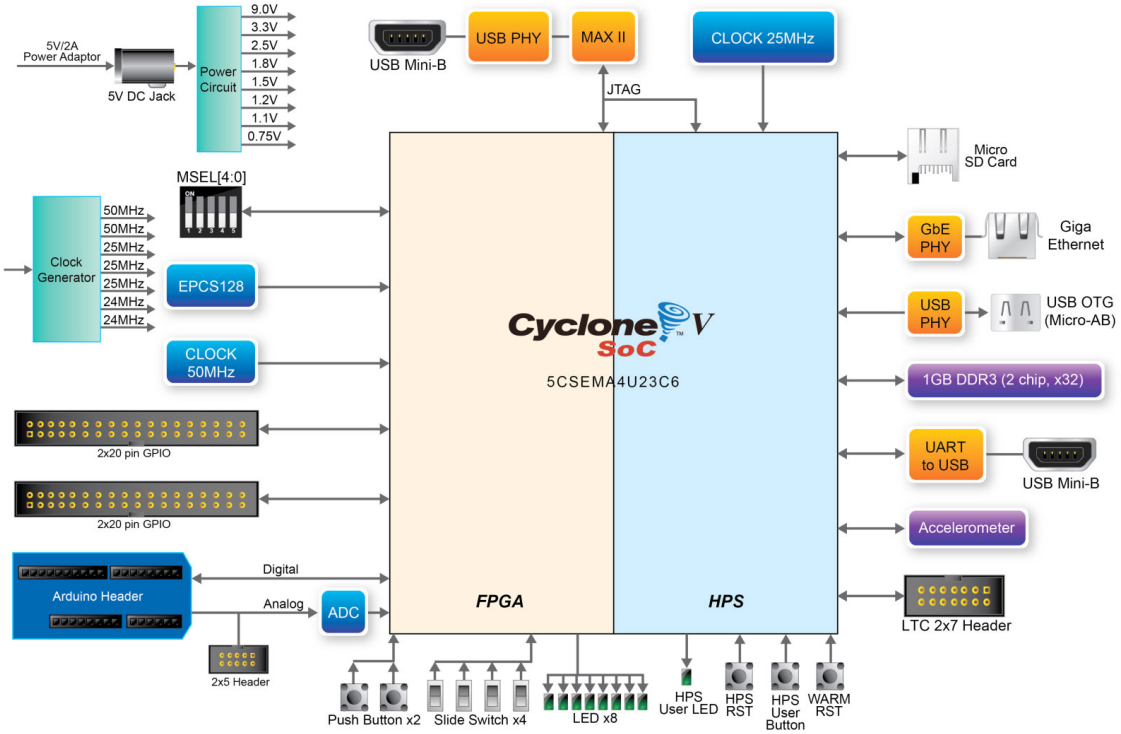

Figure 2.3: Back layout of DE0-Nano-SoC

2.3 Applications

There are many different applications that we can use to create with the De0-Nano-SoC board. Some examples would be creating a Graphical User Interface(GUI) Paint whereby user can draw on a touchscreen board using the De0-Nano-SoC board as the processor.

However what we are more focused on is the FPGA portion of the board.Thus we will delve in deeper on the interaction between OpenCL and the FPGA part of the De0-Nano-SoC. One of the many application used to illustrate the interaction would be the computation speed to show the images of a Mandelbrot set zoom sequence.

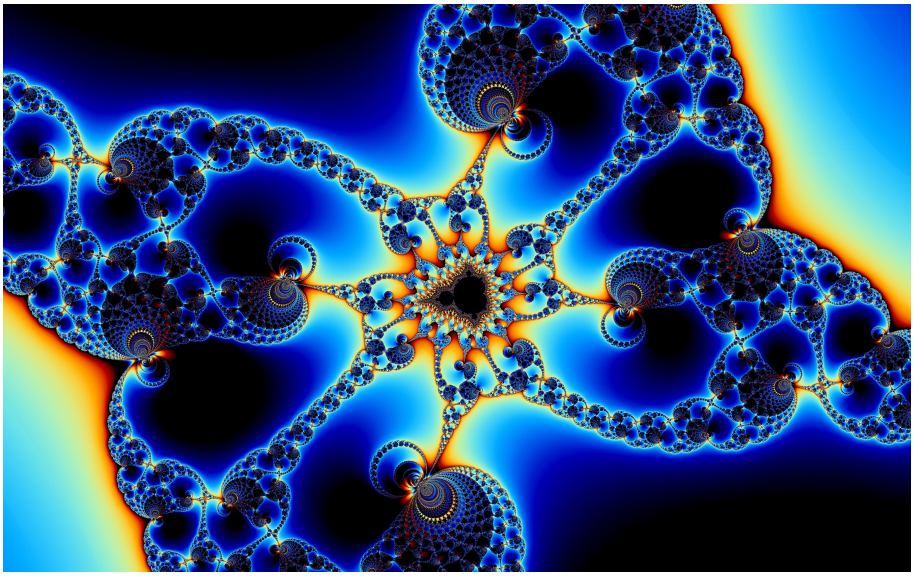

Figure 2.4: Mandelbrot set zoom Sequence[5]

By using the board attached with a display screen board, the hardware configuration file(.aocx) and the executable file which were implemented on the board will accelerate the computation speed of the Mandelbrot set zoom sequence.

Chapter 3

Open Computing Language(OpenCL)

3.1 Introduction

OpenCL(OpenCL) is a framework for accelerating software algorithms programs that can be execute in parellelism across multiple platforms such as central processing units (CPUs), graphics processing units (GPUs) , digital signal processors(DSPs)[3]and much more.

The first version,OpenCL 1.0 was developed in 2008 by the Khronos Group, an independent standards consortium. Programming language is called OpenCL C based on C99[2]which is C-like. With it being developed since 2008, there are lots of application being created. To name a few like in graphics, there are Adobe Photoshop, Photoscan and etc. Furthermore for CAD and 3D Modelling would be programs like Autodesk Maya and LuxRender.

3.2 Architecture of OpenCL

In this section we would briefly describe the OpenCL Programming Model to further understand what we are working with, in the architecture there are 3 main modelsref:bopencl

•Platform Model - host interwined with OpenCL device(s) which consists of compute units which are further divided into processing Elements

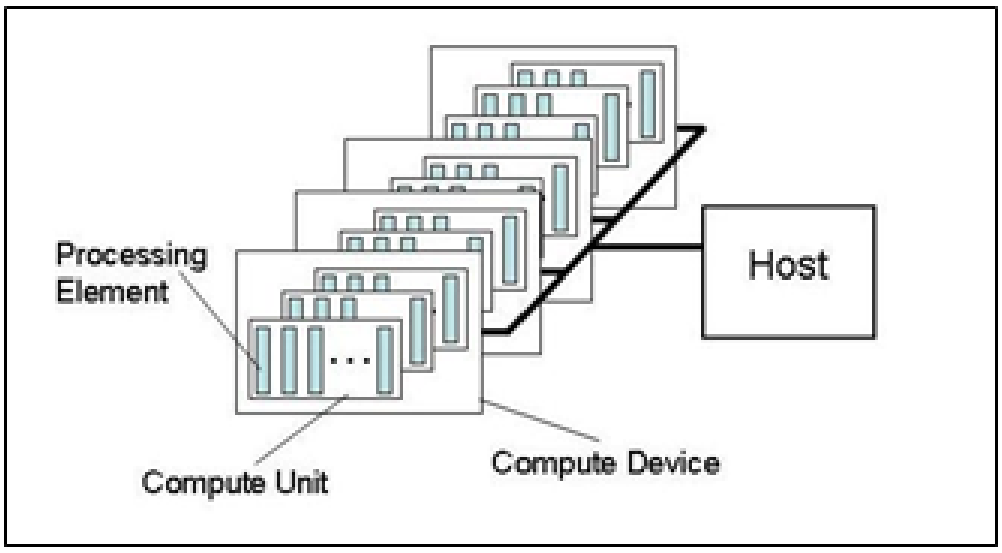

Figure 3.1: Platform Model[3]

•Execution Model - consists of two different types of execution, kernels for the OpenCL file and host programs that is executed on the host. The space of the execution is defined by the NDRange

•Memory Model - divided into two parts , one for the host and the other for the memory device

∗**Host**

- Host Memory : available to host and is defined outside of OpenCL, memory objects are transported from the OpenCL API
- Device Memory : available to kernels executing on OpenCL devices

∗**Device**

- Global : all work-items and work-groups have read and write access on both host and device
- Constant : only work-items have read access, but host have both read and write access
- Local : for data-exchange for work-items in a work group
- Private : only accessible for one work-item

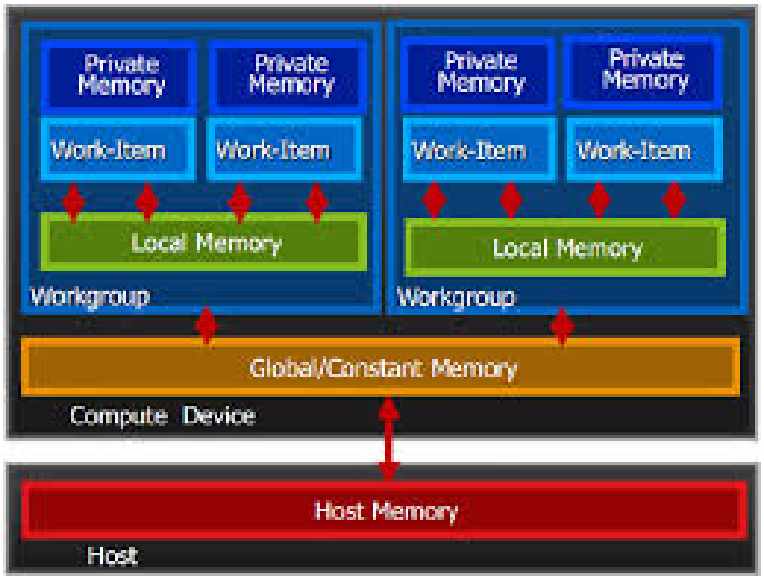

Figure 3.2: OpenCL Memory Model[3]

Chapter 4

Implementation

4.1 VectorAdd Example

In this section I am going to implement an example to familiarise ourselves with the OpenCL FPGA by creating the required files and the setting up of the environment. There will be an compilation on the host before executing it on the board.

4.1.1 Setting up of the environment

For different Cyclone V devices, they are different board support packages(BSP) for OpenCL. The BSP that was available for the DE0-Nano-SoC only support the FPGA 14.0 SDK for OpenCL. Environment variables and license file should be saved to the /.bashrc file to be set as permanent variables for the OpenCL SDK.

```
export ALTERAOCLSDKROOT="/opt/altera/14.0/hld"
export QSYS_ROOTDIR="/opt/altera/14.0/quartus/sopc_builder/bin"
export LM_LICENSE_FILE=/home/fmlab1/Downloads/1-JZ8EZ5_License.dat
export AOCL_BOARD_PACKAGE_ROOT=$ALTERAOCLSDKROOT/board/de0_nano_soc_openCL_bsp
export LD_LIBRARY_PATH=$ALTERAOCLSDKROOT/host/arm32/lib
```

Figure 4.1: Environment Variables on /.bashrc

Figure 4.1 This figure shows all the parameters to be be inputted for the OpenCL SDK.

4.1.2 Compilation of the vectorAdd example

In order to execute the program on the board, we would need two kind of files, one .aocx file and one executable file. For the .aocx file, we have to compile the OpenCL file(.cl) to produce it.

```
// ACL kernel for adding two input vectors
__kernel void vectorAdd(__global const float *x,
                        __global const float *y,
                        __global float *restrict z)//restrict to optimize the code
{
    // get index of the work item
    int index = get_global_id(0);

    // add the vector elements
    z[index] = x[index] + y[index];
}
```

Figure 4.2: vectorAdd Kernel Function

In the figure, the kernel function have 3 global arrays, 2 of the global arrays are for the input and the last array is for the output. In the function, you need to get the index of the work item so that it figure out which element array to sum. To compile the .aocx file, we need to type the following command lines to the terminal, which are **source $ALTERAOCLSDKROOT\init_opencl.sh** to go to the host OpenCL environment and

aoc device/vectorAdd.cl -o bin/vectorAdd.aocx –board de0_nano _soc _sharedonly

for compilation.

Once the .aocx file is complied, we have to direct path with the following command line **/opt/altera/14/0/embedded/embedded_commandshell.sh** in the terminal to access the cross compiler in the FPGA 14.0 SDK for OpenCL. Using GNU Makefile, we create an executable file by cross compiling the AOCL libraries and the host code.

4.1.3 Results

As discussed in the previous sections, once the files are made we have to transfer the .aocx file(hardware configuration file) and the executable file into the board and run it on the terminal with **minicom**. Using the BSP provided by the manufacturer the SD card image thats supports the OpenCL functionality was load onto the board, this allow us to test if the program was able to communicate with the board itself.

```
root@socfpga:~/ownvector_add2# aocl program /dev/acl0 vectorAdd.aocx
aocl program: Running reprogram from /home/root/opencl_arm32_rte/board/c5soc/arm32/bin
Reprogramming was successful!
root@socfpga:~/ownvector_add2# ./vectorAdd
Initializing OpenCL
Platform: Altera SDK for OpenCL
Using 1 device(s)
  de0_nano_soc_sharedonly : Cyclone V SoC Development Kit
Using AOCX: vectorAdd.aocx
Launching for device 0 (1000000 elements)

Time: 100.742 ms
Kernel time (device 0): 6.918 ms

Verification: PASS
root@socfpga:~/ownvector_add2#
```

Figure 4.3: Results for the vectorAdd

In order to run the executable file, we need access the OpenCL functionality in the board itself by typing **source ./init_opencl.sh** in the terminal. Furthermore, we need to reconfigure the board to know which .aocx file we are using by typing **acol program /dev** As shown in the figure above, after executing the vectorAdd is shows the kernel time use to compute the vector addition with specific number of elements. In conclusion we are able to familiarise ourselves with the functionality of the OpenCL and the FPGA board.

4.2 Analysis of Developed OpenCL Program

Based on what we are had learned on the concept on OpenCL and De0-Nano-SoC using the vectorAdd example. We will try to implement the already developed OpenCL program provided by my supervisor Mahmoud Khaled. Firstly we would need to create the .aocx files from the kernel files.

The OpenCL program kernel function is to construct symbolic abstraction of dynamical system using growth bound and synthesis symbolic controllers using fixed-point operations on the symbolic model constructed by over-approximating the reachabile sets.

4.2.1 Challenges Faced

∗Kernel file design cant be fit on the device

While compiling the kernel file, one of the challenges was spilt the kernel function into 4 different functions as the board specification cant fit the design of the kernel file.

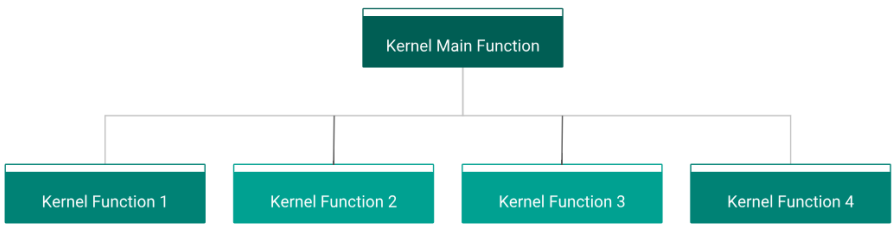

Figure 4.4: Diagram for Kernel Function

The first function would be the initialisation, the next two functions would be to construct the system with growth bound and synthesis symbolic controllers. The last kernel function would be check the functionality of the program.

In order for the kernel files to be able to interact with each other as a whole kernel function, I need to include all variables into the memory bag. Furthermore I need to change some variables to memory access so that all kernel files could be linked.

∗Altering the main.cpp file

As mention in the previous chapter for the program to run on the board, we need 2 kinds of files in order for it to work. Thus we have to also change the environment of the developed host code(main.cpp) to the same one given from the vectorAdd example.

Thus I need to include more variables into the struct data of the host code, moreover I need to reserve the memory correctly for the newly modified structure so that when examining the execution time of each kernel, the results would be more accurate.

4.2.2 Conclusion

Using vectorAdd example, i was able to sucessfully implement the kernel file on the board. From the example, my knowledge improved greatly on how to create an environment whereby I could use OpenCL on the board. Without the board support package for the board and the cross compiler for the De0-Nano-SoC, I was unable to move further in runnning the example.

From the understanding that i gained from the example. I extend it to the OpenCL kernel provided, i was able to sucessfully compile 4 .aocx files for the hardware configuration of the board. However I was from the the executable file due the difference in ultilities files and header files for the OpenCL kernel. Due to that those errors, it affect the whole structure of the code. However I believe that once the errors are rectified, the synthesis of the online symbolic controller would be possible.

Bibliography

[1] De0-nano-soc board specifications.

[2] Introduction to opencl™ programming.

[3] Kronos Group. Opencl.

[4] Matthias Rungger and Majid Zamani. Scots: A tool for the synthesis of symbolic controllers.

[5] https://rocketboards.org/foswiki/view/Projects/OpenCLMandelbrotDemoOnAtlasSoC