Embedded Real Time Systems

Particle Swarm Optimization

Final Project
Group 1

David Jensen
Henrik Bagger Jensen

Supervised by
Kim Bjerge, Jalil Boudjadar

AARHUS UNIVERSITY

January 2, 2019

Contents

1 Introduction

This report describes a final project of the course Embedded Real Time Systems. The final project is a mandatory part of the course that needs to be carried out and is done so in a group of two; David Jensen and Henrik Bagger Jensen. This project addresses the concept of hardware accelerating an optimization algorithm to present the readers of the report, that the course description of qualifications have been effectuated:

"This course covers topics for design of embedded real-time systems. It has focus on software architectures, software design patterns and methods for scheduling and analysing embedded real-time software. It provides a thorough introduction to hardware-software co-design including methodologies for partitioning, hardware architecture modelling and methods for implementing System on a Programmable Chip designs (CPU+FPGA)..."[1]

Furthermore this introduction will inform the reader how to read the document.

Throughout the project, code snippets will be used to explain in detail, how they function and what certain code will do. In Listing 1, an example of such code snippet is shown.

```
1  #include <stdio.h>
2  #define N 10
3  /* Block
4  * comment */
5
6  int main()
7  {
8    int i;
9
10   // Line comment.
11   puts("Hello world!");
12
13   for (i = 0; i < N; i++) {
14     puts("LaTeX is also great for programmers!");
15   }
16
17   return 0;
18 }
```

Listing 1: Example listing.

This project follows the guide on what to include in the final project[1]. This short document will be referred to often, and a conventional way of referring the document is established. In the start of most chapters a description will follow. The description is a snippet of the final project guide, in case the text is in a square box. Example follows.

Description:

> In this exercise, you should propose and demonstrate the use of a methodology based on SysML/UML and related profiles for the design of an Embedded Real Time project. The methodology must be applied for the design of a system that you define and specifies. The project should include a SysML/UML model with architectures for alternative SoPC solutions. A model should be developed to evaluate and verify the system in terms of hardware and software processing modules for a selected architecture. Finally part of the system should be implemented and verified on the ZYBO board.[2]

[1] [6] 2018 - Kursuskatalog.au.dk.
[2] [1] 2017 - Bjerge. |Page 1|

2 Description

The purpose of this project is to take the well-known Particle Swarm Optimization algorithm[3] and hardware accelerate it on the Zybo xc7z010clg-400-1 board. Achieving this by using the board and accompanying Xilinx tools, the algorithm will be split into parts, that run in software and others in hardware.

As seen in the rich picture on figure 1. A researcher often sits in their lab solving problems of mathematical nature. The projects varies from groundbreaking university research to helping the industry partners solve problems. An ongoing theme is the need to find a objective function hence the desire to maximize or minimize.

In this example, the researcher's preferred tool is the particle swarm optimization algorithm, because it can find the global maximum or minimum, the drawback however is that it takes a long time to do the calculations on the devices they have available. Therefore using a Zybo xc7z010clg-400-1 board where the critical parts of the algorithm will be hardware accelerated. Using this solution they can optimize problems much faster saving time that can be spent elsewhere.

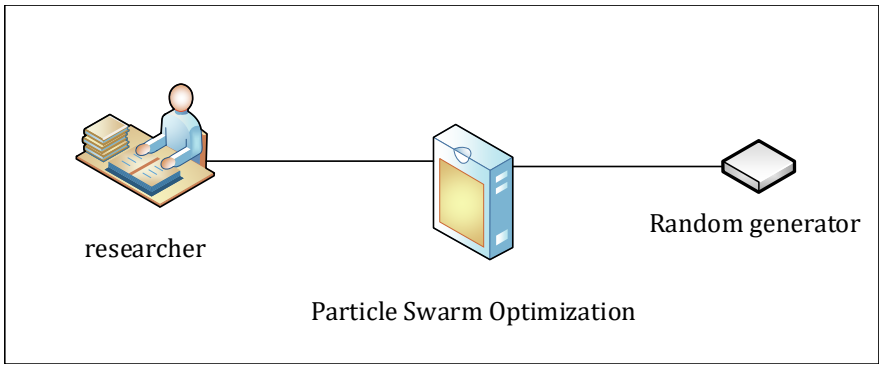


Figure 1: Researcher who is using the Particle Swarm Optimization System and a device to generate the random element.

[3] [2] 2009 - Blondin.

3 Methodology

3.1 Description

1. Define a methodology[1] using SysML/UML diagrams for the development of your system. Specify SysML/UML diagrams that need to be made in the different phases of the project. Make a short description of each design phases and the SysML/UML diagrams and profiles you decide to use in the methodology. Remember to use references to the papers you have use as inspiration for your work. Decide on an UML[2] tool. [4]

3.2 SysML

In this project SysML standard will be used to describe the system as a whole, but also to help describe core functionality down to the wire. The usage of SysML standard is decided upon because of the structural approach to the analysis. To ensure cohesion throughout the usage of SysML and UML, the *"A Practical Guide to SysML: The Systems Modeling Language"*[4] document will be used. This report will follow a Top-Down approach as stated below:

- Analyzing the Problem
- Model Organization
- Operation
  - Use Cases
  - Requirements
- Structure
  - Block Definition Diagram (bdd)
  - Internal Block Diagram (ibd)
- Behavior
  - Sequence Diagram (sd)
  - Activity Diagram (act)
  - State Machine Diagram (smd)

Microsoft Visio, with the included stencil toolbox[5], and Visual Studio 2017 Enterprise will be used as UML tools.

3.3 Analyzing the Problem

When starting a project it is important to analysis the problem description. For this project it can be found on page 4. Important information can be extracted from the description and will assist in building the boundary between the system of interest, external systems and stakeholders who interact directly or indirectly.

[4] [1] 2017 - Bjerge. |Page 1|
[5] [5] 2018 - Hruby.

3.4 Model Organization

To manage the complete model and sub-models of the full system, the package diagram[6] will be used. A package diagram is a system organization model of SysML standards. In this project the package diagram will be used to formulate and verify which diagrams and models are used as a methodology to describe the system of interest. Another effect the package diagram have on this project, is the organization structure it brings.

Multiple package diagrams are used to explain dependencies of components and sub-systems in a complex system. Since this project is of a smaller scale, only a single package diagram is needed. However this feature can be useful in case this project is used as a component or sub-system in a complex system.

3.5 Operation

The operations contains both the Use Cases and Requirements because they both describe how the system operates. They both also reflect the established requirements.

3.5.1 Use Cases

Use cases[7] represent the goals the system is intended to solve. A use case also has the notion of actors who are users of the system. An example is a researcher who access a system of interest to help solve some problem. It can also have actors that indirectly use the system such as managers or industry partners.

3.5.2 Requirements

The Requirements is presented as a requirements list. Requirements specify conditions that must be fulfilled. Requirements are goals that should be achieved.

3.6 Structure

To explain the structure of the system, the usage of both block definition diagram (bdd) and internal block diagram (ibd) will be used. Both bdd and ibd will be used as structure diagrams and won't include context to the environment because of the system scale.

3.6.1 Block Definition Diagram

The block definition diagram[8] (bdd) is used to display a high-level top view, of the overall system. In this project the bdd will be used to make a simple structure model. The bdd will be used to explain core behavior and relationship between blocks.

3.6.2 Internal Block Diagram

The internal block diagram[9] (ibd), is used to explain internal and external connections between blocks, from the prior block definition diagram. In this project, it will be used to make a more detailed structure model, on top of the information gathered by the block definition diagram.

3.7 Behavior

In SysML it is possible to model the behavior of the system using Activity Diagrams which describe the actions in a given Activity that controls the flow of input/output from one Activity to another. Sequence Diagrams main focus is the communication between different objects in time. State Machines main focus is the different states the system can obtain.

[6] [4] 2014 - Friedenthal, Moore, and Steiner. |Page 103|
[7] [4] 2014 - Friedenthal, Moore, and Steiner. |Page 303|
[8] [4] 2014 - Friedenthal, Moore, and Steiner. |Page 119|
[9] [4] 2014 - Friedenthal, Moore, and Steiner. |Page 119|

3.7.1 Activity Diagram

Activity diagram created to show the control flowing from one Activity to another and they are good at visualizing the logic such as of conditional structures, loops and concurrency. A way to look at it is to think about it as a complex flowchart.

3.7.2 Sequence Diagram

Sequence diagrams are used to model how parts of a block interact by the means of API calls signals or messages. This is useful to get an overview of how software works by using methods or functions between software components. But be noted that Sequence Diagrams are not strictly for software.

3.7.3 State Machine Diagram

State Machine Diagram is used to model state behaviors throughout a blocks life cycle. This is useful when a system or sub-system has different states of operation, to structure and show the functionality that is required in different states of the system.

4 Requirement specification

In this section a package diagram will give an overview of the different diagrams included and where they reside. It also cover a use case diagram showing the different use cases that has been extracted from the project description found on page 4. At the end of the section use case descriptions and a list of requirements will be displayed.

4.1 Description

> **2.** Write a requirement specification with functional and non-functional requirements especially with focus on performance like throughput and latency. The functional requirements can be described in terms of use cases.[10]

4.2 Analyzing the Problem

To make get a better understanding of the problem the description in section 2 Description, on page 4, has been analyzed. The result of the analyze have produced nouns which can be utilized to fabricate an understanding between problem and model, and will be used to generate the use cases and requirements list.

4.3 Model Organization

The package diagram can be seen in figure 2. Observing the Figure indicate that the top level is named PSOS (Particle Swarm Optimization System), note that the other packages connect to this namespace.

A brief description of the content:

- Operation contains Use Cases and Requirements diagrams, see section 4.4 Operation, page 9.
- Structure contains Block Definition and Internal Block diagrams, see section 5.2 Structure, page 12.
- Behavior contains Activity, Sequence and State Machine diagrams, see section 5.3 Behavior, page 14.

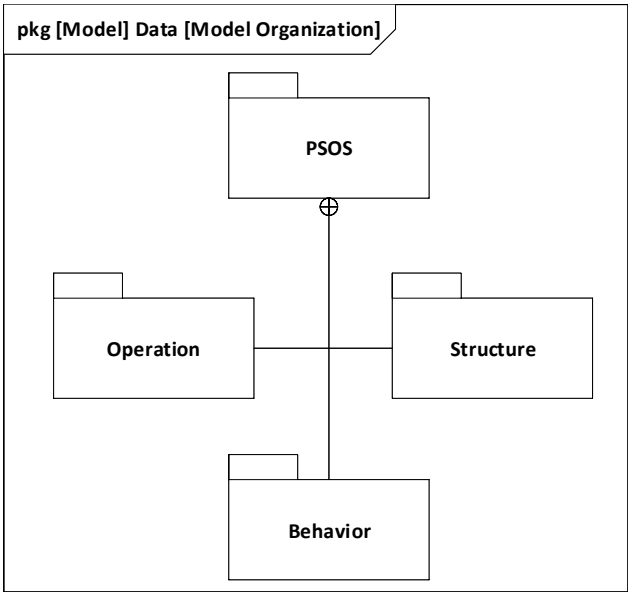


Figure 2: Package Diagram with the namespace PSOS

[10] [1] 2017 - Bjerge. |Page 1|

4.4 Operation

4.4.1 Use Case Diagram

The Use Case Diagram found on Figure 3 show the different use cases of the system. UC1: Setup where the setup of the system happens, here various parameters can get changed to fit the researchers needs. UC2: Find Minima is where the software and hardware will attempt to find the global minimum of the given problem. UC3: Find maxima is where the software and hardware will attempt to find the global maximum of the given problem. In both UC2 and UC3 the researcher should always be able to return to UC1.

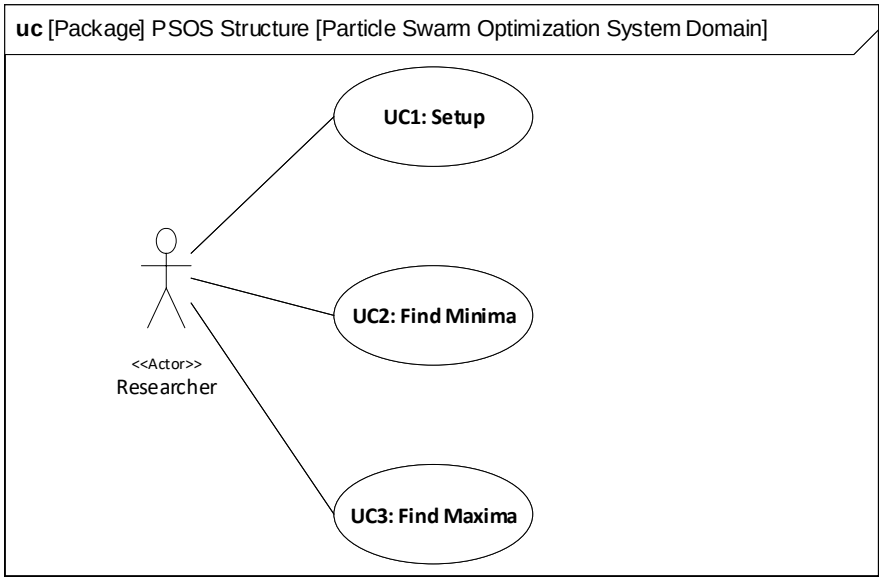


Figure 3: Use Case Diagram of Particle Swarm Optimization

4.4.2 Use Case descriptions

The use case descriptions are created as fully dressed use cases.

Name:	UC1: Setup
Goal:	Configure the board via the interface
Initering:	Researcher
Actor:	Researcher (primary)
Reference:	
Number of simultaneous occurrences:	One
Pre-condition:	The board powered on and has software installed
Result:	The board has been setup
Main scenario:	1. Changes the value for c1 to 1 using the buttons on the Zybo board. • [Ext 1.a : User changes state by using the switchs to Find Maxima.] • [Ext 1.b : User changes state by using the switchs to Find Minima.] 2. The chnages are saved by using the buttons on the Zybo board. 3. UC1 finishes.
extensions:	**[Ext 1.a : User changes state by using the switches to Find Maxima.]** 1. The system continues at Find Maxima using default values. 2. UC1 finishes. **[Ext 1.b : User changes state by using the switches to Find Minima.]** 1. The system continues at Find Minima using default values. 2. UC1 finishes.

Table 1: UC1: Setup

Name:	UC2: Find Minima
Goal:	Find the minima of a function
Initering:	Researcher
Actor:	Researcher (primary)
Reference:	UC1: Setup
Number of simultaneous occurrences:	One
Pre-condition:	UC1 has been done
Result:	Found the minima of a function
Main scenario:	1. Presses start and waits for the system to compute a solution. 2. The system show the final position with best result. 3. UC2 finishes.

Table 2: UC2: Find Minima

Name:	UC3: Find Maxima
Goal:	Find the maxima of a function
Initering:	Researcher
Actor:	Researcher (primary)
Reference:	UC1: Setup
Number of simultaneous occurrences:	One
Pre-condition:	UC1 has been done
Result:	Found the maxima of a function
Main scenario:	1. Presses start and waits for the system to compute a solution. 2. The system show the final position with best result. 3. UC3 finishes.

Table 3: UC3: Find Maxima

4.4.3 Requirements

The requirements for the project are listed below.
Functional requirements

1. The system must have a GUI.
2. The system must make use of the hardware buttons in the GUI.
3. The system must make use of the hardware switches in the GUI.
4. The system must use particle swarm optimization to find the minima.
5. The system must use particle swarm optimization to find the maxima.
6. The GUI must allow the researcher to change C1 and C2 in the algorithm.

Non-functional requirements

1. The system must work with floating point precision in the search algorithm.

5 Design Methodology

5.1 Description

3. Use your design methodology found in **1**. to describe a SysML/UML model of your system in terms of structure and behavior. Make a suggestion for alternative HW/SW architectures. Decide on which parts of the functionality should be mapped to hardware components and software processes. [11]

5.2 Structure

5.2.1 Block Definition Diagram

The system consist of three parts, a random generator, an user interface and the particle swarm logic. on Figure 4 the block definition diagram can be seen including these three parts, as part of the overall Particle Swarm Optimization System.

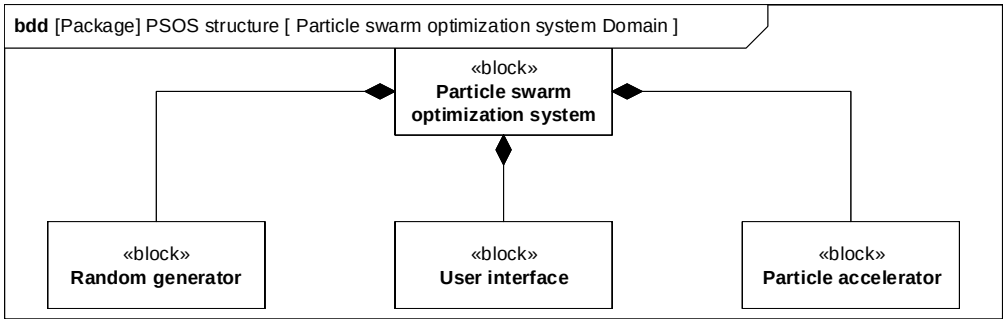


Figure 4: Block Definition Diagram of Particle Swarm Optimization System

Random Generator:
The Random Generator block is needed, to generate random seeds or values for use in the particle swarm algorithm.

User Interface:
The User Interface block needs core functionality to communicate with the system. It needs to be able to take inputs regarding the particle swarm algorithms variables and create a graphical representation of the results to the user, after a successful particle swarm analysis.

Particle Accelerator:
The Particle Accelerator block has the functionality to perform the particle swarm optimization algorithm. Variables for the algorithm, given by the user, are communicated via the User Interface block. The random aspects for the algorithm is given by the Random Generator block.

5.2.2 Internal Block Diagram

Continuing with the data from the Block Definition Diagram, an Internal Block Definition Diagram (ibd) is created. Showing connections and dataflows with the ibd as seen on Figure 5.

[11] [1] 2017 - Bjerge. |Page 1|

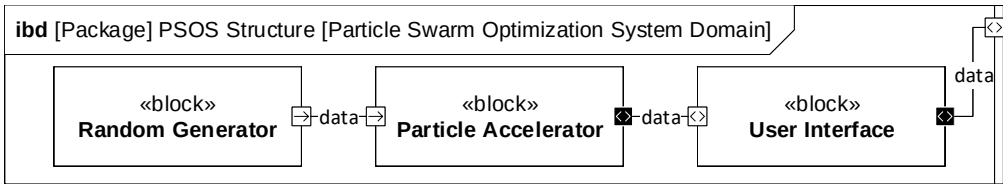


Figure 5: Internal Block Diagram of Particle Swarm Optimization System

The researcher uses the User Interface block, as a medium to interact with the system. The User input data to be used as variables in the particle swarm algorithm through the User Interface via a GUI. When the particle swarm algorithm has been performed by the system, the output is given to the researcher by the User Interface Block via the GUI.

The User Interface block communicates the desired variables to be used in the particle swarm algorithm to the Particle Accelerator block. When the Particle Accelerator block has performed the particle swarm algorithm, it sends the data to the User Interface block.

The Random Generator delivers a random seed or value, when needed, to the Particle Accelerator block, to be used in the particle swarm algorithm.

Two different designs of the Particle Accelerator have been created. Both designs are designed as IP Cores in mind. The first design, shown on Figure 6, is a simplistic design, in which all functionality is formed inside a single instance of a Particle Accelerator.

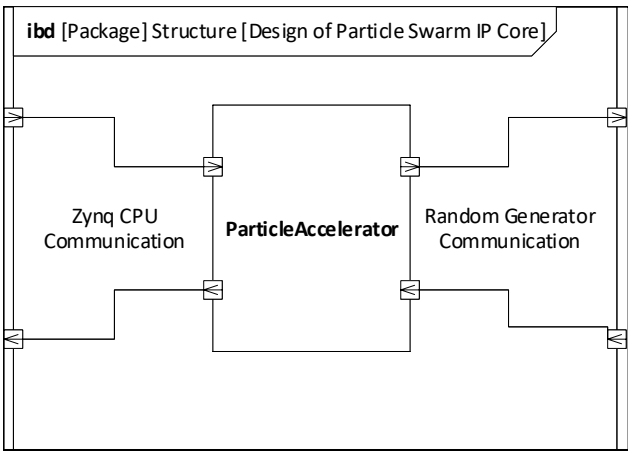


Figure 6: Simple IP Core design of the Particle Accelerator

The second design, shown on Figure 7, contains the Particle Master and up to n particles. The Particle Master controls the particles and acquire each particle's result. This is the design that will be further elaborated upon.

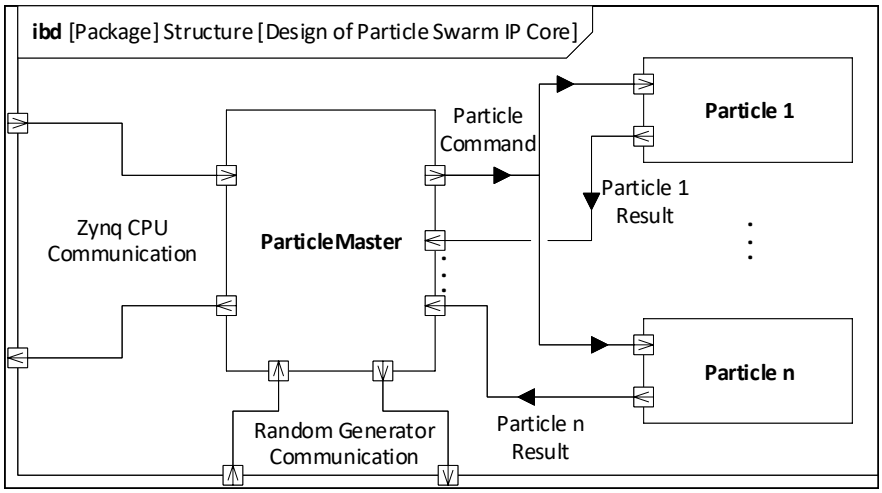


Figure 7: IP Core design of the Particle Accelerator with

5.3 Behavior

5.3.1 Activity Diagram

The Activity diagram, shown on Figure 8, shows how the different parts of the project work together. When the board has been started and software installed, the researcher is asked to configure the values $c1$ and $c2$, these are also known as the acceleration coefficients. $c1$ expresses how much confidence a particle has in itself, the cognitive aspect. $c2$ expresses how much confidence a particle has in its peers, the social aspect. A search for the minima can then be started and random values for the algorithm will be requested. When the random values has been obtained, the calculation occur and a result is returned.

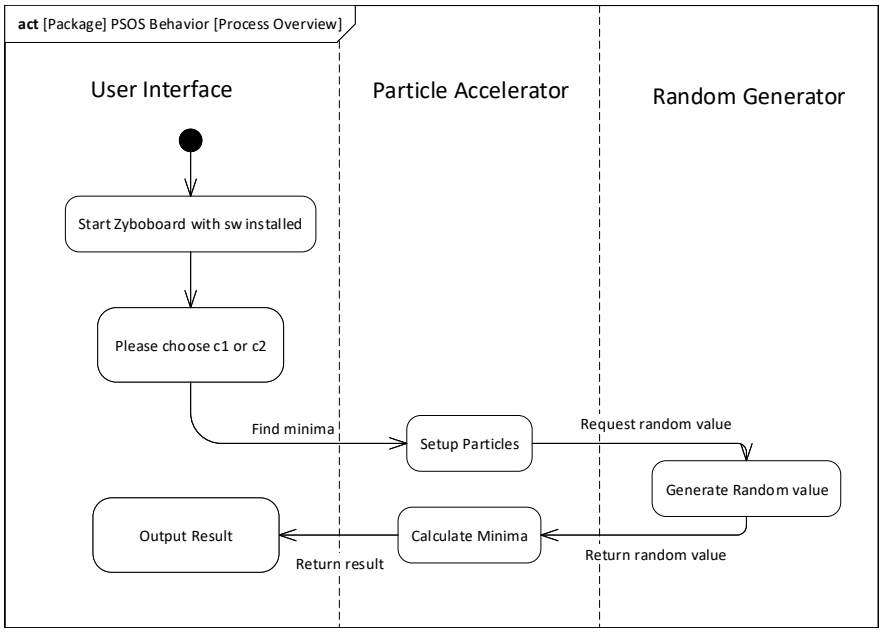


Figure 8: Activity diagram for PSOS

5.3.2 Sequence Diagram

The sequence diagram in Figure 9, shows the Sequence Diagram for the Zynq CPU. It has a context, that is used to control the state pattern. Different actions and transitions are called.

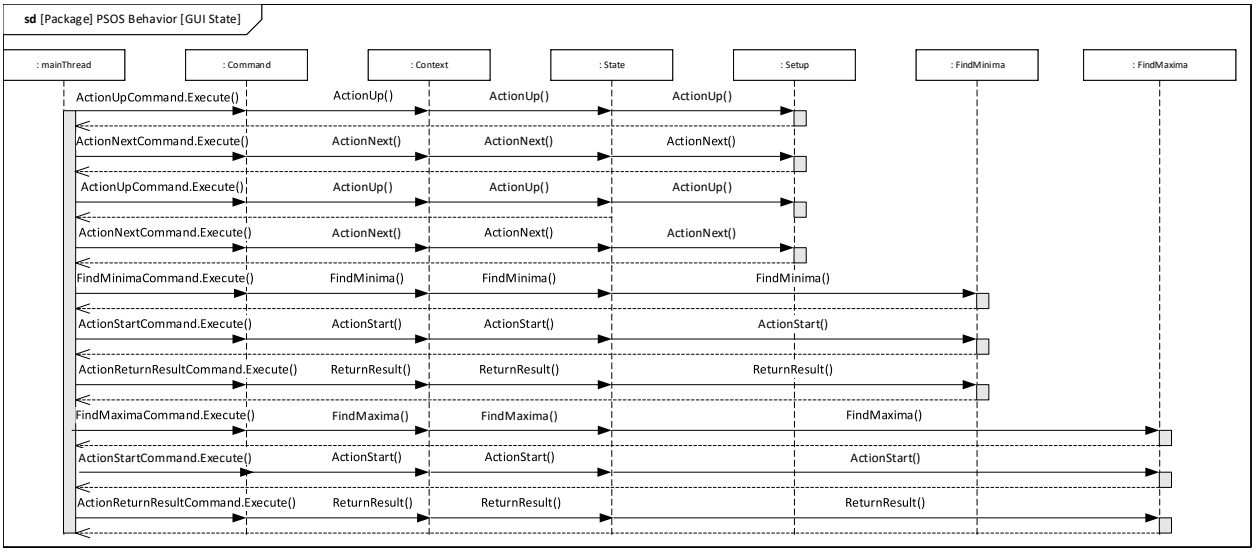


Figure 9: Sequence Diagram for PSOS

5.3.3 State Machine Diagram

The State Machine in Figure 10 shows the different states that are allowed and how to traverse them. As can be observed, specific actions and transitions are allowed in the current state.

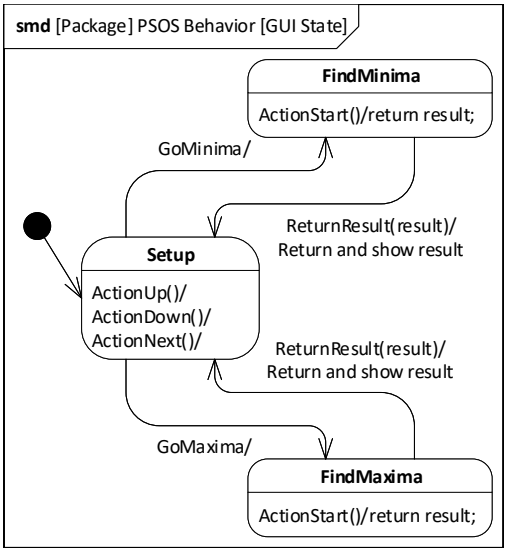


Figure 10: State Machine Diagram of the Graphical User Interface

6 Design Patterns

6.1 Description

> **4.** Design the software and apply design patterns that are suitable for your project, and motivate the choice of the used patterns. Use the Two-Part Architecture Model if relevant for the problem. Use the abstract OS package for the ZYBO board.

[12]

In the following solution the patterns used are:

- State Pattern
- Singleton
- Command Pattern

In this project it has been concluded that the implementation of the State Pattern, Singleton and Command Pattern[13] profits the project. Each of the patterns are covered in the sections following.

[12] [1] 2017 - Bjerge. |Page 1|
[13] [7] 1995 - Ralph Johnson, Erich Gamma, John Vlissides.

6.2 GOF State Pattern

The intent of the state pattern is to grant an object the ability to change its behavior when it's internal state changes hence the object will therefore change its class. Consider the state machine diagram in Figure 10, page 15, that represents the GUI of the PSOS. The PSOS can be in one of the distinct states: Setup, FindMinima, FindMaxima.

When the Zynq CPU starts it enters the Setup state. This receives actions from other classes such as `ActionUp`, `ActionDown`, `ActionNext` and the respond will differ base on the action. For example, the result of an `StartAction` depends on what the current state is in. In the FindMinima or FindMaxima state the action can occur, the Setup state however does not accept it.

As seen in Figure 11, is the state machine in the project. It was a good choice to use the GOF State Pattern because of the ease to model a state machine and bring it into code.

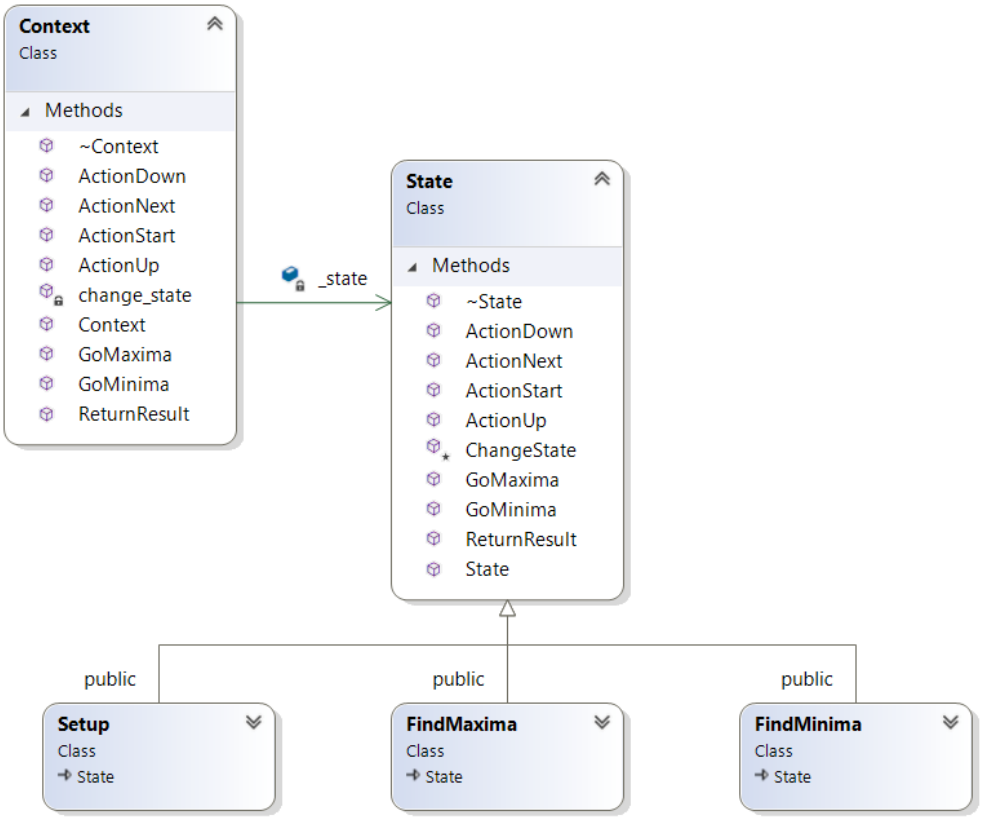


Figure 11: An overview of the state machine and it's states.

6.3 GOF Singleton

The intent of the singleton is to guarantee that a class only has one instance therefor providing a single global point of access to it. This is accomplished by making the class itself responsible for keeping track of its instance. The class can guarantee that no other instance can be constructed and therefore it can only have a single way to access the instance.

In the project, the GOF Singleton was used to acquire deep history in order for the researcher's variable settings to be kept in the Setup class. It can be seen in the class diagram in Figure 12, where the constructor is private and the only way to get access to the instance of the class, is by using its `Instance()` method. This guarantees that the acquired instance of the object is always of the same object, unless the destructor have been called.

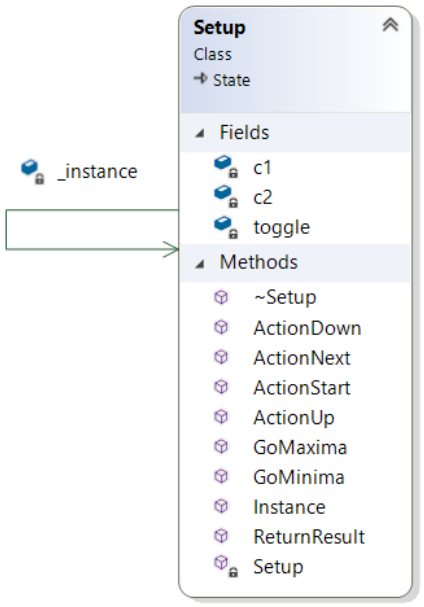


Figure 12: The setup class makes use of the Gof singleton pattern.

6.4 GOF Command Pattern

The intent of GOF Command Pattern is to Encapsulate a request and therefore letting one set different parameters for clients with different requests types. An example could be a message, log requests or other requests. It has support undoable operations (operations that can't be undone).

This allows software engineers to create objects that can interact with the application in a parameterize manner. The command can be stored and moved around like other objects. The main thing about this pattern is the abstract Command class which other commands inherit from. The abstract Command has a interface for executing operations hence an abstract Execute method. Concrete Command subclasses specify a Execute method with the needs for the specific Command.

In Figure 13 the commands in the PSOS is shown. The Actions are `ActionDownCommand`, `ActionNextCommand`, `ActionStartCommand` and `ActionUpCommand`. The Actions are used to change settings in the setup state and start the search for FindMinima and FindMaxima. The transitions commands are `FindMaximaCommand`, `ReturnResultCommand` and `FindMinimaCommand`. By using commands the code handling specific functionality are streamlined and only exist one place, hence it makes it easier to maintain and guards against code duplication.

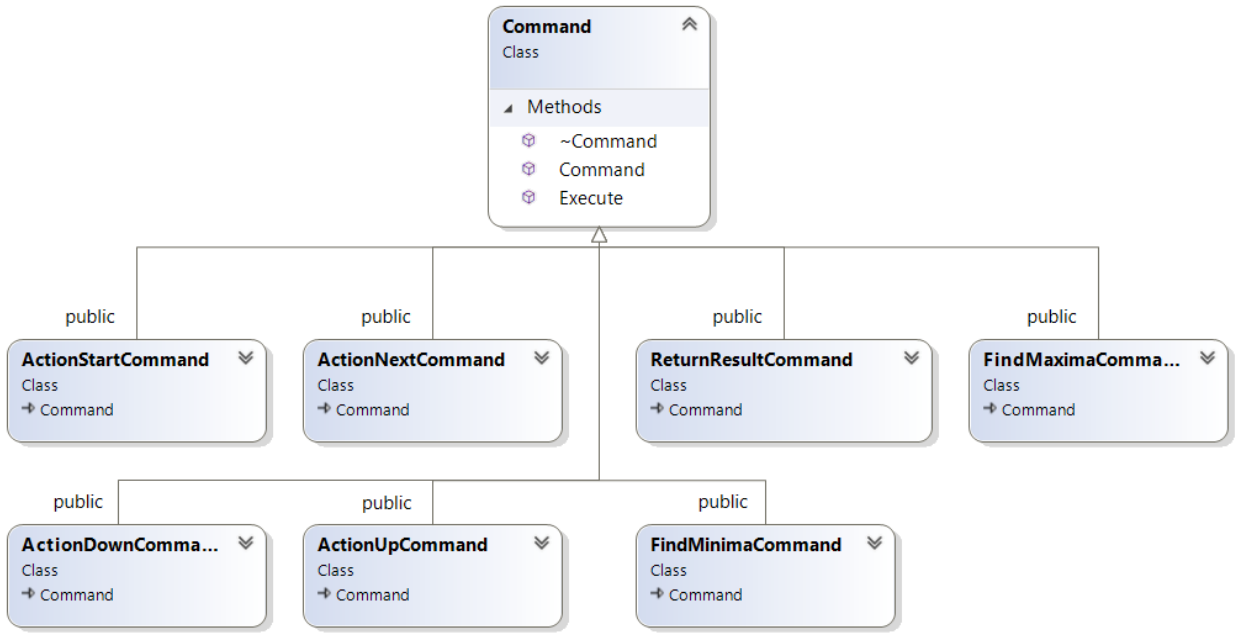


Figure 13: Different Commands in the PSOS.

7 Implement and Test

7.1 Description

5. Select the SysML/UML model for your preferred architecture suggestions in **3.** Choose a part of the functionality including both hardware and software components to create a model and a testbench in HLS using SystemC or C-code. Simulate and validate your design model.

Argue for your choice of modeling language and abstraction level of modeling. Use the reports from the HLS tool to evaluate performance of the design. Assess whether the design is able to fulfill your requirements and constraints.[14]

6. Implement and test a part of your system using the ZYBO platform including at least one IP core written and verified with the HLS tool.

7.2 Float as a Datatype

Throughout this project, it has not been stated how essential it is that the data worked on is as close to real values as possible. Using long double as the used datatype both for calculation and evaluation for the particle swarm optimization algorithm is preferred. This is the floating point with most precision supported in SystemC. Working with Vivado however, it is known that only fix point as a datatype is supported. To keep it simple the students have created a test program in the Vivado HLS tool that uses floats as a datatype both for calculations and in data transfers, to check if it is possible to use floating point, without conversion, when trying to hardware accelerate. The program that will be introduced in this section is floatPrototype.

The minimal design of floatPrototype is a IP core that take in two inputs, of type float, make a simple calculation with the data and returns it as a float. The outcome will conclude whether all the basic building blocks of the desired functionality in the PSOS can be achieved or not. On Figure 14 is a basic drawing of the desired flow of the floatPrototype IP core.

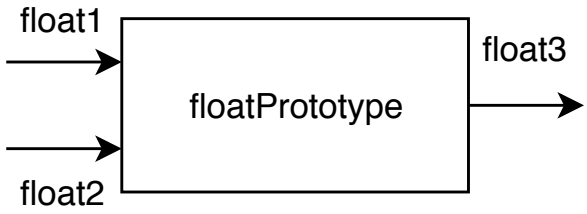


Figure 14: Figure showing the dataflow of floatPrototype.

```
1 [...]
2 SC_MODULE(floatPrototypec){
3 [...]
4   sc_in<float> float1;
5   sc_in<float> float2;
6   sc_out<float> float3;
7
8   void multiply();
9 [...]
```

Listing 2: Simplified view of the header file of floatPrototype.

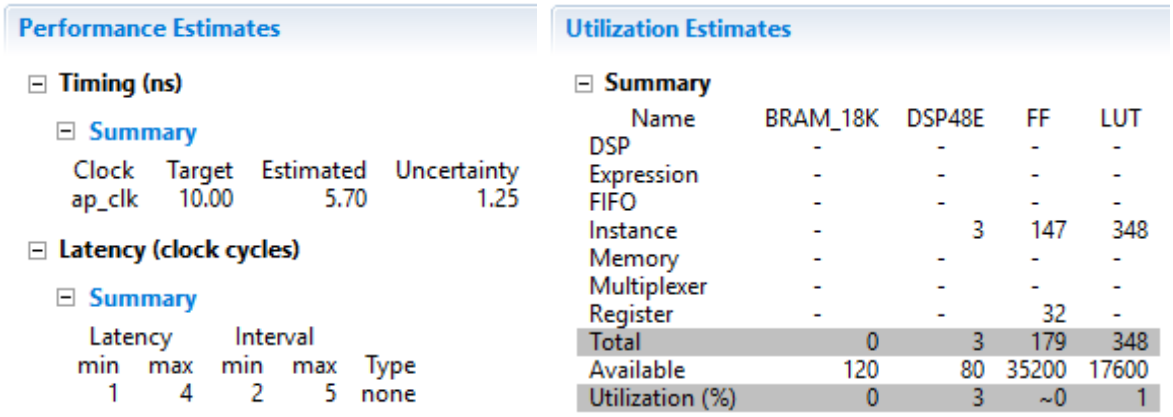
[14] [1] 2017 - Bjerge. |Page 2|

On Listing 2 and 3 is the SystemC implementation of floatPrototype, scaled down to the core of the IP core test. As can be seen on Listing 2 the IP core that is being created has two float inputs, `float1` and `float2`, as well as a float output, `float3`. It has a function, `multiply()`, which simply multiplies `float1` with `float2` and writes the result onto `float3`.

```
1 #include "floatPrototype.h"
2
3 void floatPrototypec::multiply(){
4   #pragma HLS resource core=AXI4LiteS metadata="-bus_bundle slv0" variable=float1
5 [...]
6   float3.write( float1.read() * float2.read() );
7 }
```

Listing 3: SystemC file of floatPrototype.

A simulation of the code is constructed and conducted in the Vivado HLS tools. The specific test is not included in this section. If study of the test is desired, the reader is referred to the HLS appendix that can be found in the "appendix.zip". The outcome of the test suit is the expected outcome in the case that the program functions using floats. The summary after the synthesis of floatPrototype can be seen on Figure 15.



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	5.70	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
1	4	2	5	none

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	-	-
FIFO	-	-	-	-
Instance	-	3	147	348
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	32	-
Total	0	3	179	348
Available	120	80	35200	17600
Utilization (%)	0	3	~0	1

Figure 15: Autogenerated summary after synthesis of floatPrototype

To further elaborate upon the float as a datatype test, the floatPrototype is exported as an RTL (Register-transfer level) from the Vivado HLS tool, and included into a simple program. The program has tests that are conducted to check whether it is possible to utilize floats in an IP core with the datatype being transferred directly. The open block design in Vivado can be seen on Figure 16.

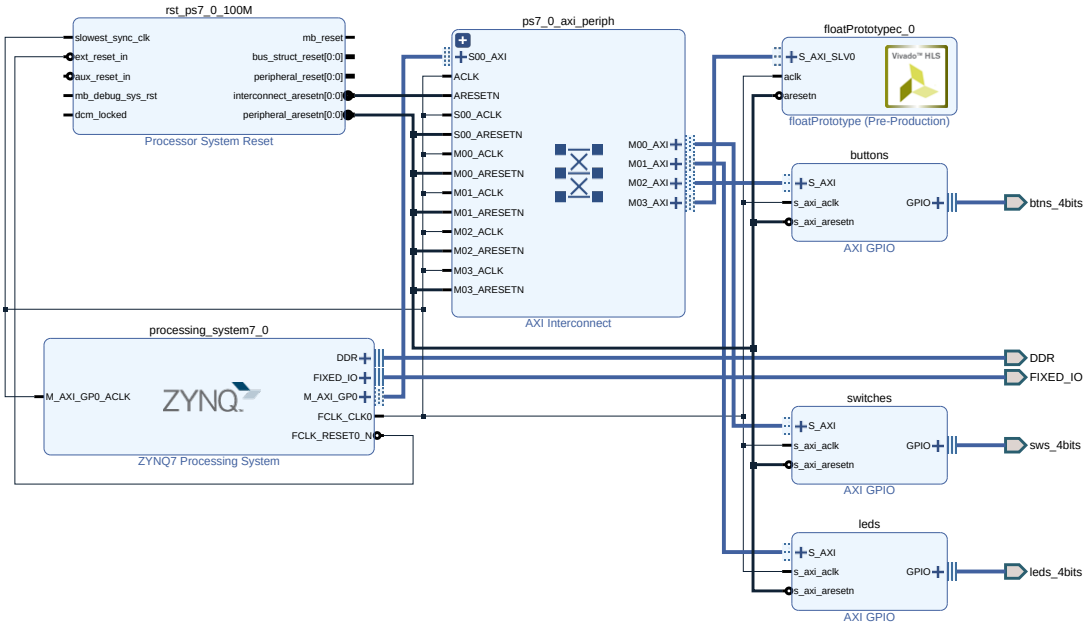


Figure 16: Vivado open block design with floatPrototype as an IP core.

At first glance it doesn't seem possible to send floats directly to and from the IP core, instead the core communicates using u32's in it's Axi Lite interface. Since a float use 32bits like an u32 it is possible to cheat the system to think that the datatype send is a u32, by making a u32 pointer point at a float object, when in fact it is a float. This is a bit of a hack, telling the system that it is of another datatype than what it is. By testing the IP core with a specific input values the output show that the IP core does indeed use floats. Results of a testprogram, performed on the Zybo board, can be seen on Figure 17.

```
Welcome to float multiply build
The first test does 5*5 and the result should be 25
The result of this test was: 25
The second test does 2.5*2.5 and the result should be 6.25
The result of this test was: 6.25
The third test does 9.9302902903*9.832932892389 and the result should be 97.64387802646198
The result of this test was: 97.6439
As it can be seen, it can't handle so many decimals
```

Figure 17: Result of floatPrototype IP core test, run on Zybo Board.

With these tests conducted, it can be verified that it is indeed possible to hardware accelerate mathematical expressions in hardware.

7.3 Vivado HLS

With the knowledge acquired from section 7.2 Float as a Datatype the creation of a Particle Swarm Optimization algorithm on hardware seems possible, however dependent on the problem to solve it may be incredibly demanding. The problem that was chosen to use as a test is the "peaks" function[15] and the algorithm that was implemented was first tested and verified in matlab and can be found in the appendix folder.

To see the complete code, the reader is referred to the appendix where the complete HLS code can be found.

In the design of the particles it is known that there will be two bottlenecks in which data can be problematic to handle. The first is the construction of a random value, it is not possible to create a random value inside an IP core with the HLS tools. Therefore the used Zynq CPU or another interface is needed to create a random value. This feature is abstracted away in this concept to simplify the project, but it is a known bottleneck that can slow the particles down. The second bottleneck is the global best solution after each iteration of movement of the particles. A central or master unit needs to check which solution of all the particles are the best solution, a simple version of this is implemented however this is a part of the system that can be heavily optimized in future versions of the system.

The PSOS HLS is divided into two different classes, a Particle and a ParticleMaster class. The Particles are doing the individual particle calculations while the master is the global calculation. The master also handles the communication to and from the Zynq CPU. On Figure 18 the connections of the HLS component are shown.

[15] [3] 2013 - Chong and Zak. |Page 290|

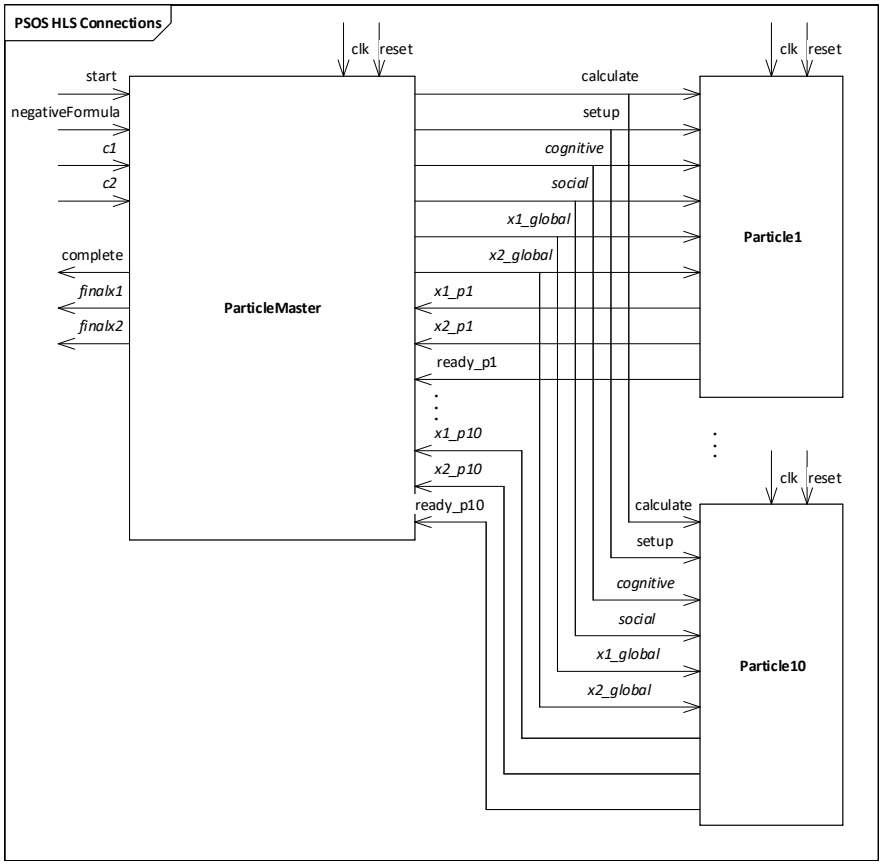


Figure 18: Diagram showing the connections of the HLS program. Normal text indicate that the datatype transfered is a boolean. Italic text indicate that the datatype transfered is a float.

7.3.1 ParticleMaster

The ParticleMaster has two threads running. `Setup` and `ReadCalculations`. `Setup` awaits the *start* connection to go high. When this event occurs it will start to setup the ten particles that is attached to it with the different variables that are used in the particle swarm optimization algorithm. When this has been done it starts the particles by letting the *setup* connection go high. The code of the `Setup` can be seen in Listing 4.

```
1  void particlemaster::Setup(){
2    while(true){
3      while(!start.read() || setupDone){
4        wait();
5      }
6
7      setup.write(false);
8      setupDone = false;
9      wait();
10
11     // reading and setting up c1 and c2
12     cognitive.write(c1.read());
13     social.write(c2.read());
14     wait();
15
16     // Checking if we are looking for maximum or minimum and writing it
17     negFormula = negativeFormula.read();
18     wait();
19     maximum.write(negFormula);
```

```
    wait();

    iterations = 60;
    setupDone = true;

    setup.write(true);
    wait();
  }
}
```

Listing 4: Setup thread of ParticleMaster.

`ReadCalculations` awaits that all the particles are done with their calculation of each iteration before checking which of the particle has the best global position. If it has not run all the iterations through it will continue to make the particles calculate a new position after giving them the global best position. After the last iteration it will send the best position found back through the *finalx1* and *finalx2* connections and end with setting the *complete* connection high as well. The code can be seen on Listing 5.

```
void particlemaster::ReadCalculations(){
  while(true){
    while(!(ready_p1.read() && ready_p2.read() && ready_p3.read() && ready_p4.read() &&
        ready_p5.read() && ready_p6.read() && ready_p7.read() && ready_p8.read() && ready_p9
        .read() && ready_p10.read())){
      wait();
    }

    // Read values
    float x1s[10];
    float x2s[10];

    x1s[0] = x1_p1.read();
    x2s[0] = x2_p1.read();
    wait();
[...]
    x1s[9] = x1_p10.read();
    x2s[9] = x2_p10.read();
    wait();

    calculate.write(false);
    wait();

    float bestValue = Equation(x1_best, x2_best);

    for (int i = 0; i < sizeof(x1s); i++) {
      #pragma HLS INLINE
      if(Equation(x1s[i],x2s[i]) < bestValue)
      {
        bestValue = Equation(x1s[i],x2s[i]);
        x1_best = x1s[i];
        x2_best = x2s[i];
        wait();
      }
    }

    iterations--;

    if(iterations > 0)
    {
      x1_global.write(x1_best);
      x2_global.write(x2_best);
      wait();

      calculate.write(true);
      wait();
```

```
45      }
46      else
47      {
48        finalx1.write(x1_best);
49        finalx2.write(x2_best);
50        wait();
51
52        complete.write(true);
53        wait();
54      }
55    }
56  }
```

Listing 5: ReadCalculations thread of ParticleMaster.

7.3.2 Particle

The ParticleMaster has ten particles attached. This is hardcoded as of now, but should be more than enough to create a proof of concept.

Like the ParticleMaster, each particle has two threads; `Setup` and `Execute`. Like the `Setup` thread from ParticleMaster, this thread uses the data communicated from the ParticleMaster to the Particles, to setup the different variables that can be modified. in this case it is only possible to change $c1$ and $c2$ which are the cognitive and social modifiers of the PSO. When the particles are setup they will set their start position at a random value with a random velocity. When this have been done it sends back the position coordinates to the master, through the *x1_out* and *x2_out* connections, and sets the *ready* connection to high afterwards. Code can be seen on Listing 6.

The particle is then setup and ready to start using execute.

```
void particles::Setup(){
  while(true){
    while(!setup.read() || setupDone){
      wait();
    }

    ready.write(false);
    wait();

    setupDone = false;
    calculationDone = false;

    w = 0.8;
    ax = 10;
    av = 1;

    negativeFormula = maximum.read();
    c1 = cognitive.read();
    c2 = social.read();

    wait();

    // Make first position and velocity
    x1 = Random_position(0,ax);
    x2 = Random_position(0,ax);

    v1 = Random_velocity(av);
    v2 = Random_velocity(av);

    // Set first position as best position
    x1_best = x1;
    x2_best = x2;

    x1_out.write(x1);
    x2_out.write(x2);
    wait();

    setupDone = true;
    ready.write(true);
    wait();
  }
}
```

Listing 6: Setup thread of Particle.

`Execute` awaits the internal `setupDone` flag as well as the *calculate* connection to go high before starting the calculation process. The calculation process that is followed, is the standard equation of PSO shown in Equation 1.

$$v_i(t+1) = wv_i(t) + c_1 r_1(\hat{x}_i(t) - x_i(t)) + c_2 r_2(g(t) - x_i(t)) \tag{1}$$

Every time a new position has been calculated it is tested whether or not the position is a more optimized position than the old best cognitive position. If so, the particle overrides the old best position with the new position. After this process, the Particle set the *ready* communication to high and awaits the next command.

```
void particles::Execute(){
  while(true){
    while(!(setupDone && calculate.read()) || calculationDone){
      if(!calculate.read())
      {
        calculationDone = false;
      }
      wait();
    }

    ready.write(false);
    wait();

    v1 = w*v1 + c1*Randval()*(x1_best-x1) + c2*Randval()*(x1_global.read()-x1);
    v2 = w*v2 + c1*Randval()*(x2_best-x2) + c2*Randval()*(x2_global.read()-x2);
    wait();

    x1 = x1+v1;
    x2 = x2+v2;

    if (Equation(x1,x2) < Equation(x1_best,x2_best))
    {
      x1_best = x1;
      x2_best = x2;
    }

    x1_out.write(x1);
    x2_out.write(x2);

    calculationDone = true;

    ready.write(true);
    wait();
  }
}
```

Listing 7: Execute thread of Particle.

After synthesizing the SystemC model, it is obvious that the IP Core is far too demanding for the FPGA on the Zybo board in this project. Summaries of the results can be seen on Figure 19. These results will be elaborated further upon in section 8 Conclusion, on page 32.

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	10.47	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
0	50600	1	50601	none

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	-	-
FIFO	-	-	-	-
Instance	30	7873	757964	305280
Memory	-	-	-	-
Multiplexer	-	-	-	9
Register	-	-	294	-
Total	30	7873	758258	305289
Available	120	80	35200	17600
Utilization (%)	25	9841	2154	1734

Figure 19: Autogenerated summary after synthesis of PSOS HLS.

7.4 Vivado

The implementation of the PSO through the HLS tool have shown that the Zybo board used in this project doesn't have the necessary FPGA to implement the IP core on the board. Therefore the implementation of the GUI design won't include the IP Core generated in the Open Block design. The PSO have been abstracted away to test the design on the board.

This section show the implementation of the different software patterns mentioned in section 6 Design Patterns, page 16, on the Zynq CPU. The design makes use of the GPIO ports to access the hardware buttons and switches, to control the state of the PSOS, as shown on Figure 10, on page 15. The buttons are used to control the actions in the application ActionUp, ActionDown, ActionNext and ActionStart. The switches are used to control the different of the PSOS; Setup, FindMinima and FindMaxima. The open block design used to test the GUI software can be seen on Figure 20.

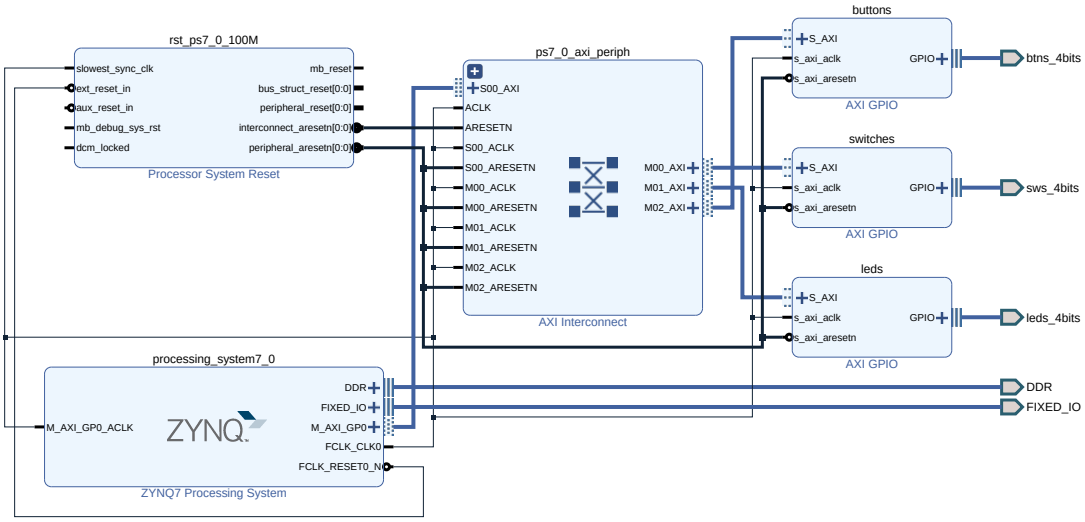


Figure 20: Vivado open block design of design used to test GUI software.

The Software is made using FreeRTOS, where the mainThread controls the application. As it can be seen in Listing 8 it initializes the buttons and switches, and starts checking on button and switch events. In the case a new event occurs, it sends a command via command pattern to the current state.

```
1  [...]
2  void MainThread::run(){
3    xil_printf("Welcome to Particle Swarm Optimization Embedded \r\n");
4
5    XGpio dip, push;
6
7    XGpio_Initialize(&dip, XPAR_SWITCHES_DEVICE_ID);
8    XGpio_SetDataDirection(&dip, 1, 0xffffffff);
9
10   XGpio_Initialize(&push, XPAR_BUTTONS_DEVICE_ID);
11   XGpio_SetDataDirection(&push, 1, 0xffffffff);
12
13   xil_printf("Use the switches to control the menu \r\n");
14
15   _context = new Context;
16   //Transitions
17   FindMinimaCommand findMinimaCommand = FindMinimaCommand(_context);
18   FindMaximaCommand findMaximaCommand = FindMaximaCommand(_context);
```

```
19    ReturnResultCommand returnResultCommand = ReturnResultCommand(_context);
20
21    while (true){
22      switch (XGpio_DiscreteRead(&dip, 1)) {
23        case 0:
24          returnResultCommand.returnedResult = 10;
25          returnResultCommand.Execute();
26          buttonCommand(_context, XGpio_DiscreteRead(&push, 1));
27          break;
28        case 1:
29          findMinimaCommand.Execute();
30          buttonCommand(_context, XGpio_DiscreteRead(&push, 1));
31          break;
32        case 2:
33          findMaximaCommand.Execute();
34          buttonCommand(_context, XGpio_DiscreteRead(&push, 1));
35          break;
36        default:
37          break;
38      }
39      for (int i = 0; i < 9999; ++i);
40    }
41  }
42  [...]
```

Listing 8: MainThread run thread.

As mentioned in section 6.4 GOF Command Pattern, on page 19, the usage of the GOF Command Pattern will be used to execute commands on the state pattern. An example of an implemented command can be seen on Listing 9.

```
1  [...]
2  FindMaximaCommand::FindMaximaCommand(Context* c) {
3    _context = c;
4  }
5  void FindMaximaCommand::Execute(){
6    _context->GoMaxima();
7  }
```

Listing 9: FindMaxima command implementation.

In section 6.2 GOF State Pattern, on page 17, it is decided that the usage of GOF State Pattern shall be utilized to maintain the states of the GUI. On Listing 10 the implementation of an action and a transition can be seen as an example from the Setup state. `ActionUp()` changes the internal value $c1$, `GoMaxima(context*)` changes the state from *Setup* to *FindMaxima*.

```
1  [...]
2  void Setup::ActionUp(){
3    if(toggle == true)
4    {
5      std::cout << "ActionUp called c1: " << ++c1 << std::endl;
6    } else {
7      std::cout << "ActionUp called c2: " << ++c2 << std::endl;
8    }
9  }
10 void Setup::GoMaxima(Context* c){
11   std::cout << "GoMaxima" << std::endl;
12   ChangeState(c, new FindMaxima());
13 }
14 [...]
```

Listing 10: Actions implemented in the Setup state.

Because of the need to remember the values $c1$ and $c2$ inside the Setup state, the Setup state is implemented as a singleton. This implements deep history into the class as stated in section 6.3 GOF Singleton, on page 18. The implementation of the singleton pattern can be seen on Listing 11.

```
1  [...]
2  Setup* Setup::_instance = 0;
3
4  Setup* Setup::Instance() {
5    if (_instance == 0) {
6      _instance = new Setup();
7    }
8
9    std::cout << "Please set c1 and c2" << std::endl;
10   return _instance;
11 }
12
13 Setup::Setup() {
14   toggle = true;
15   c1 = 1;
16   c2 = 1;
17 }
18 [...]
```

Listing 11: Singleton implementation in the Setup state.

Testing the functionality of the software on the Zybo Board gives the expected result and the output from the console can be seen in Figure 21.

```
Welcome to Particle Swarm Optimization Embedded
Use the switches to control the menu
Please set c1 and c2
ActionUp called c1: 2
Now you can change c2
ActionDown called c2: 0
ActionDown called c2: -1
ActionDown called c2: -2
GoMinima
Started to FindMinima
ReturnResult
Please set c1 and c2
GoMaxima
ReturnResult
Please set c1 and c2
ActionUp called c2: -1
ActionDown called c2: -2
Now you can change c1
ActionUp called c1: 3
ActionDown called c1: 2
ActionDown called c1: 1
ActionUp called c1: 2
ActionUp called c1: 3
```

Figure 21: Console output of Vivado test program.

8 Conclusion

This project have been conducted in order to analyze if the particle swarm optimization algorithm can be optimized using hardware acceleration. The entire algorithm is implemented in hardware, this could be the reason why so many hardware components are used. A possible way to reduce the hardware requirements would be to only move part(s) of the algorithm onto hardware. Some of parts of particle swarm would be better suited for the Central Processing Unit, such as the generation of random values. But this would also bring trade-offs because of the incensed communication between the hardware and CPU. As for having a random generator in hardware the students didn't know of method to achieve this, though as an afterthought it could properly be implemented by getting random noise from the environment.

The math function that was used as the problem to optimize is the peaks function therefore no general functions was used. The peaks function is a rather large, hence hardware requirements would be less if a simpler function was used. Another way to enhance performance would be to have utilized bit wise operations and thereby making the calculations faster.

The high level synthesis tools, proved to be very slow, using 2 hours to compile and do synthesis. The code in HLS would need to be optimized to get a more manageable experience. Due to the excessive synthesis time, the choice was made to stop the development of more designs.

A key part here was also the use of floating point precision in the search algorithm. As shown in the synthesizer we need quite a many lookup table (LUT), Digital Signal Processor (DSP) and flip-flops. But it is indeed still a design that would possible to run on a bigger FPGA. The ultrascale+ VU37P[16] can handle the designs shown in Figure 19, but the VU37P was not available in the university lab hence it was not possible to test the real scenario.

[16] [8] 2017 - Xilinx Inc.. |Page 3|

9 Bibliography

References

[1] Kim Bjerge. Project in Embedded Real Time Systems, 2017. URL https://blackboard.au.dk/bbcswebdav/pid-1860576-dt-content-rid-5344534_1/courses/BB-Cou-UUVA-75742/Project/ERTS_Project.pdf.

[2] James Blondin. Particle swarm optimization: A tutorial. ... *Site: Http://Cs. Armstrong. Edu/Saad/Csci8100/Pso Tutorial ...*, pages 1–5, 2009. URL http://cs.armstrong.edu/saad/csci8100/pso_tutorial.pdf.

[3] Edwin K.P Chong and Stanislaw H. Zak. *An introduction to optimization*. 2013. ISBN 9781118279014.

[4] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML: The Systems Modeling Language*. 2014. ISBN 978-0-12-800800-3. doi: 10.1016/B978-0-12-800202-5.00002-3. URL https://books.google.com/books?id=Ze60AwAAQBAJ.

[5] Pavel Hruby. Visio Stencil and Template for UML 2.5, 2018. URL http://www.softwarestencils.com/uml/#Visio2013.

[6] Kursuskatalog.au.dk. Kursuskatalog: Embedded Real Time Systems, 2018. URL https://kursuskatalog.au.dk/da/course/83379/Embedded-Real-Time-Systems.

[7] Richard Helm Ralph Johnson, Erich Gamma, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. ISBN 0201633612, 9780201633610. URL https://books.google.dk/books?id=iyIvGGp2550C.

[8] Xilinx Inc. UltraScale+ FPGA Product Tables and Product Selection Guide, 2017. URL https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf.