Département de Génie Électrique

École Polytechnique de Montréal

Automne 2017

- ELE 8307 -

Prototypage rapide de systèmes numériques

Rapport de Projet : *Logic Brain*

Fait par :

Sivakumar Chidambaram
(1898504)

Abidin Talebna
(1678162)

Table of Contents

1. Introduction

Logic Brain implements the forward propogation of a Neural network. A neural network consists of a set of neurons arranged in layers one after another. There is an input layer, followed by many hidden layers and finally an output layer. The output of each neuron is the weighted summation of all the inputs of previous layer as shown in equation 1. Then all the weighted inputs are summed along with a bias. The sum is then compared with a threshold and the output is assigned a ternary value {-1 ,0, 1}. The first input layer consists of 8 bits integer inputs and integer weights. The hidden layers consist of ternary inputs and ternary weights. The outputs are ternary for all the layers. The input is an image and the neural network is applied on a sub-image window. This sub-image window is moved along the horizontal and vertical axis until the whole image is covered. The output image is displayed on the VGA monitor. The project aims to implement this network calculations on Cyclone V development kit and improve the speed as compared to implementing the algorithm on software. This report explains in details the architecture in section II, followed by results in section 3 and conclusions in section III.

2. Architecture

The proposed architecture takes advantage of the reconfigurable and flexible nature of the Field Programmable Gate Arrays (FPGA). Hence, it can dynamically accommodate upto a maximum of 1024 neurons in each layer and upto 16*16 sub-image size. However, the number of layers is hard coded to 3. The proposed architecture consists of 3 main parts:

1. Avalon Slave
2. Avalon Master
3. Neural Network Calculation.

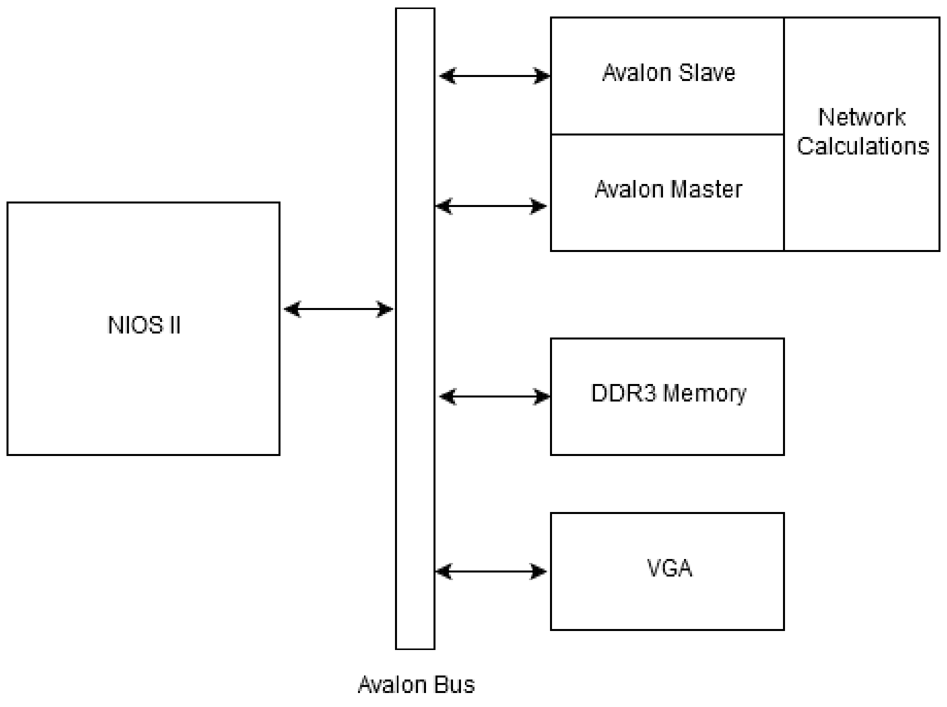

Fig 1: Top View

The Top view of the important components are shown in Fig 1. The NIOS II processor controls the overall flow of the program. The Avalon Slave/Master module implements the Neural Network calculations and writes the result into the DDR3 memory. The VGA displays the result on the VGA monitor.

2.1 Avalon Slave

The Avalon slave is an Avalon Memory Mapped Interface. It is interfaced with the NIOS II processor through the Avalon bus. It has a read and write interface. The read interface is used to obtain the dynamic parameters from the NIOS II processor. As these parameters are user defined in the C++ program of the NIOS processor, they can be changed 5 mins before. The address mapping of the data is given in Table 1.

Slave Address	**Value**
Base + 0	Number of neurons in layer 1
Base + 2	Number of neurons in layer 2
Base + 4	Number of inputs in layer 1
Base + 6	Number of inputs in layer 2
Base + 8	Positive threshhold of layer 1
Base + 10	Negative threshhold of layer 1
Base + 12	Positive threshhold of layer 2
Base + 14	Negative threshhold of layer 2
Base + 16	Sub-Image starting pixel source base address
Base + 18	Output layer writing base address
Base + 20	Layer 1 bias base address
Base + 22	Layer 1 weight base address
Base + 24	Layer 2 bias base address
Base + 26	Layer 2 weight base address
Base + 28	Layer 3 bias base address
Base + 30	Layer 3 weight base address
Base + 32	Start signal
Base + 34	Number of neurons in layer 3
Base + 36	Number of inputs in layer 3
Base + 38	Positive threshhold of layer 3
Base + 40	Negative threshhold of layer 3

Table 1: Avalon Slave Read Interface Values

The Avalon Slave Read interface is mainly used to read the done signal so at to know when the slave has finished implementing all the state machine. Also, it is used to read different register values for debugging.

2.2 Avalon Master

The Avalon master's functionality is similar to that of the Direct Memory Access. It also utilises the Avalon Memory Mapped Interface to communicate with the DDR3 memory. It also has both read and write interface. The read interface is used to read data from the memory and the write interface is used to write data into the memory. The address of the memory location is obtained from the Avalon slave interface.

2.2.1) Read Interface

The read interface of the Avalon master is used to read the data from the DDR3 memory. The appropriate address is set to av_address for reading the source,bias and weights. The Byte_Enable is set to high and the data is read in the av_readdata. The readdata is 128 bits longs. Thus 4 integers can be read at a time. The readdata is read into a FIFO buffer. All the data (for ex. source pixels) is pushed into the buffer, then the data is popped from the buffer one by one and stored in appropriate registers. Then these data are used for numerical calculations. The advantage of having such an interface is that it allows the NIOS II processor to work on other tasks while the Avalon master fetches data. An ASM is shown in Fig 2.

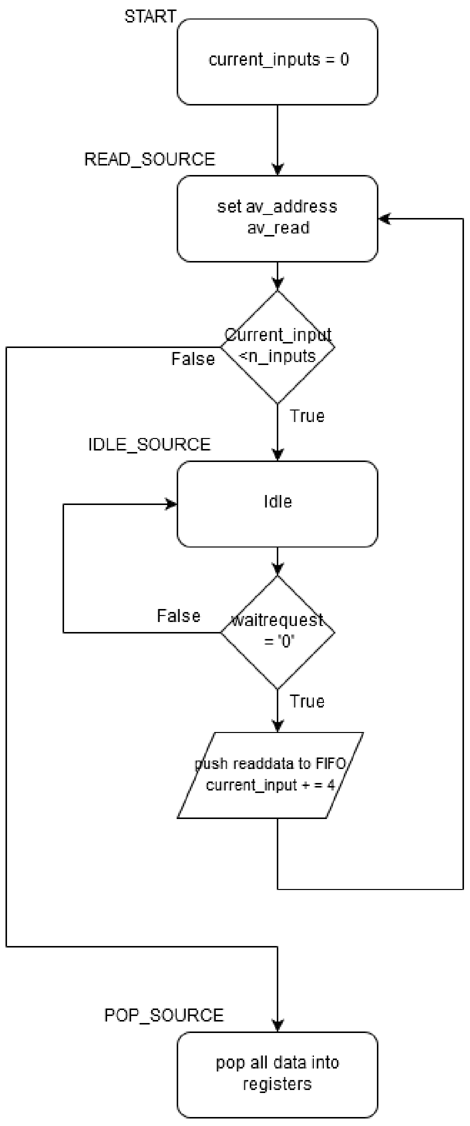

Fig 2: ASM for Reading Data

2.2.2) Write Interface

The write interface of the Avalon master is used to write data from the registers in the Avalon master/slave device to the DDR3 memory. The data that needs to be written are first pushed into the FIFO. Then the av_address is set to address of the memory location, av_write is asserted and the byte_enable is made high. Then the data from the FIFO is popped to the Avalon writedata. The writedata is 128 bits wide, hence 4 integers can be written at a time. The ASM for writing interface is given in Fig 3.

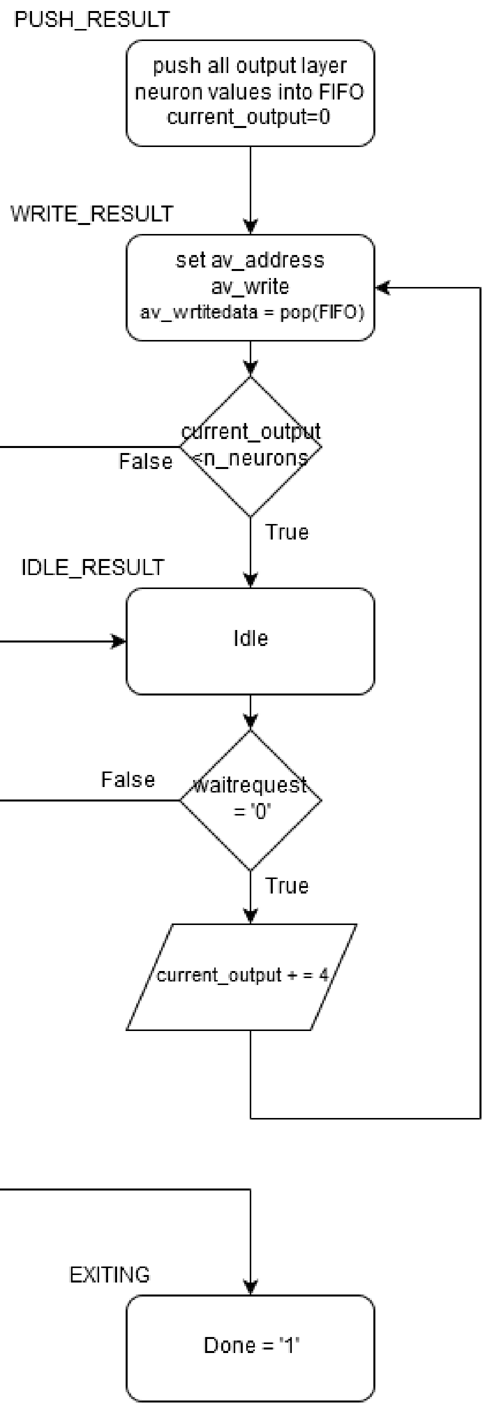

Fig 3: ASM for writing data

2.3 Numerical Calculations

The numerical calculations form the main part of the state machine design. The numerical calculation is where the parallelised processing power of the FPGA is harnessed. The multiplication calculations are done in parallel. The algorithm is explained in a series of step and through a flowchart [Fig 4]. The overall datapath is illustrated in Fig 5.

STEP 1: First all the source data is read from the memory into source registers (reg_source). Then the current layer counter is set to 0.

STEP 2: All parameters for the current layer such as the number of neurons, number of inputs, threshholds, bias base address and weights base address are set.

STEP 3: Then, the bias value is read from the memory. As the readdata is 128 bits wide, the bias values for 4 neurons are read together. Hence the bias is read after a gap of 4 neurons.

STEP 4: The accumulator is set with the bias value of current neuron.

STEP 5: Then, the weights need to be read. Here, the read operation is performed twice. Thus 8 weights are obtained.

STEP 6: These 8 weight values and the first 8 source pixel values are multiplied and summed. This sum is added to the accumulator. The multiplication is an 8 bit multiplication using the <*> operator of IEEE_NUMERIC_STD package.

STEP 7: Then the current input counter is increased by 8.

STEP 8: Then the next 8 weight values are read and is multiplied with the following 8 source pixel values. The sum of these 8 products is added to the accumulator.

STEP 9: This process continues till the current input is less than the number of inputs. Once the current input is greater than the number of inputs then the accumulator value is compared with the thresh holds in the comparison state. Based on the threshholds, the output value is stored in the intermediate output register.

STEP 10: The current neuron counter is incremented by 1 and the control goes to the set accumulator state.

STEP 11: Then the same steps 4 to 11are followed for the subsequent neurons. It continues till the current neuron counter is less than the number of neurons.

STEP 12: Once it is greater, then all the values from intermediate output register is copied to the intermediate input register. The current layer is incremented by 1.

STEP 13: The control transfers to the set parameters state. For the hidden layers, the calculation state inputs are taken from the intermediate input register. The multiplication is ternary multiplication, which is explained in detail in section 3.1. The output is stored in the intermediate output register. Again, the intermediate output is copied to the intermediate input register.

STEP 14: For the final output layer, the whole process remains the same except for the last step. The values in the intermediate output register are pushed into the FIFO of the Avalon master write interface instead of writing it back to intermediate input register. Then the values from the FIFO are written into the memory. Since the Avalon data bus width is 128 bits, 4 outputs are written at a time.

STEP 15: Once all the data is written, the state machine outputs asserts the done signal. This done signal is continuously read by the NIOS II processor. Once the done signal is high, the processor de-asserts the start signal, which in turn de-asserts the done signal. Hence the state machine goes back to the INIT state and stays in the INIT state until the next sub image is loaded into the memory and the start signal is once again asserted by the processor.

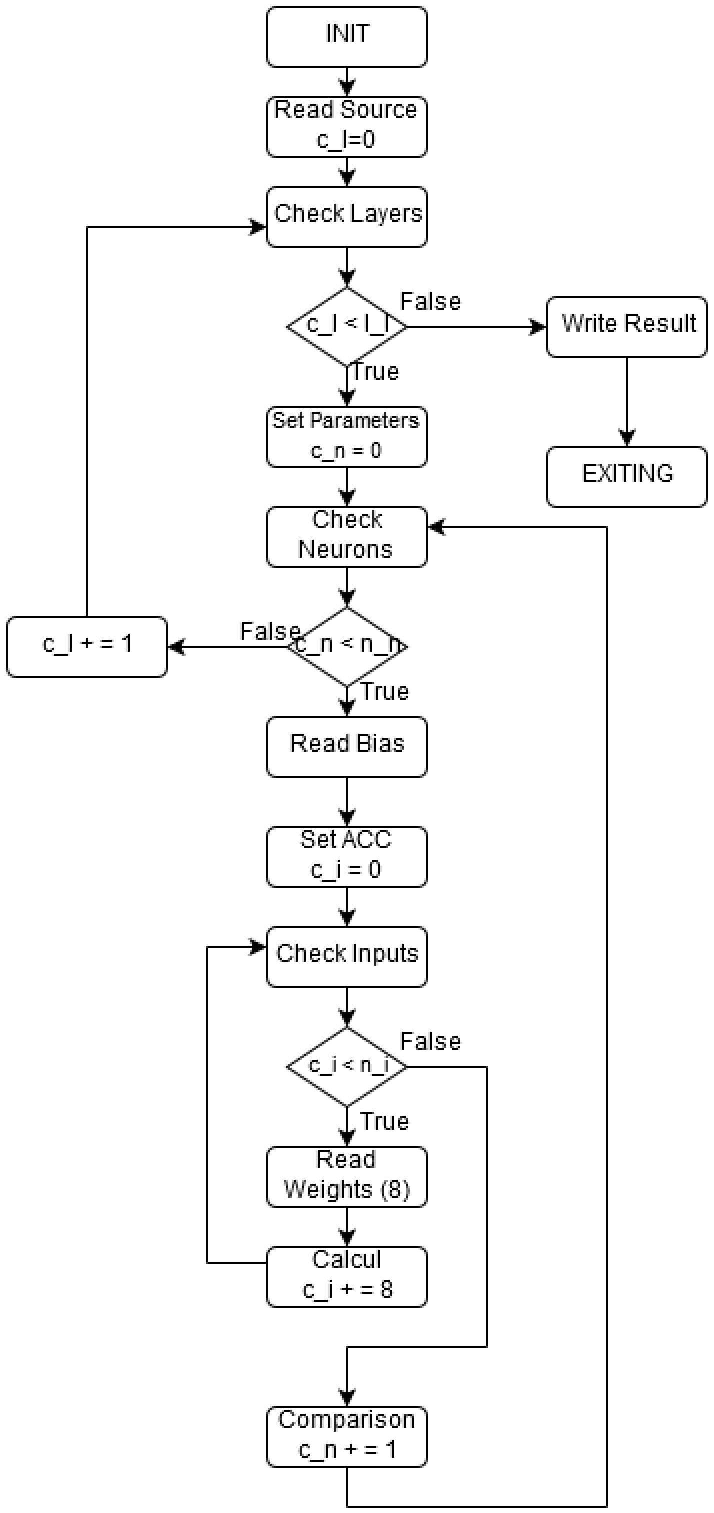

Fig 4: Overall ASM

Legends: c_i = current input, c_n = current neurons, c_l = current layer, n_i = number of inputs, n_n = number of neurons, n_l = number of layers

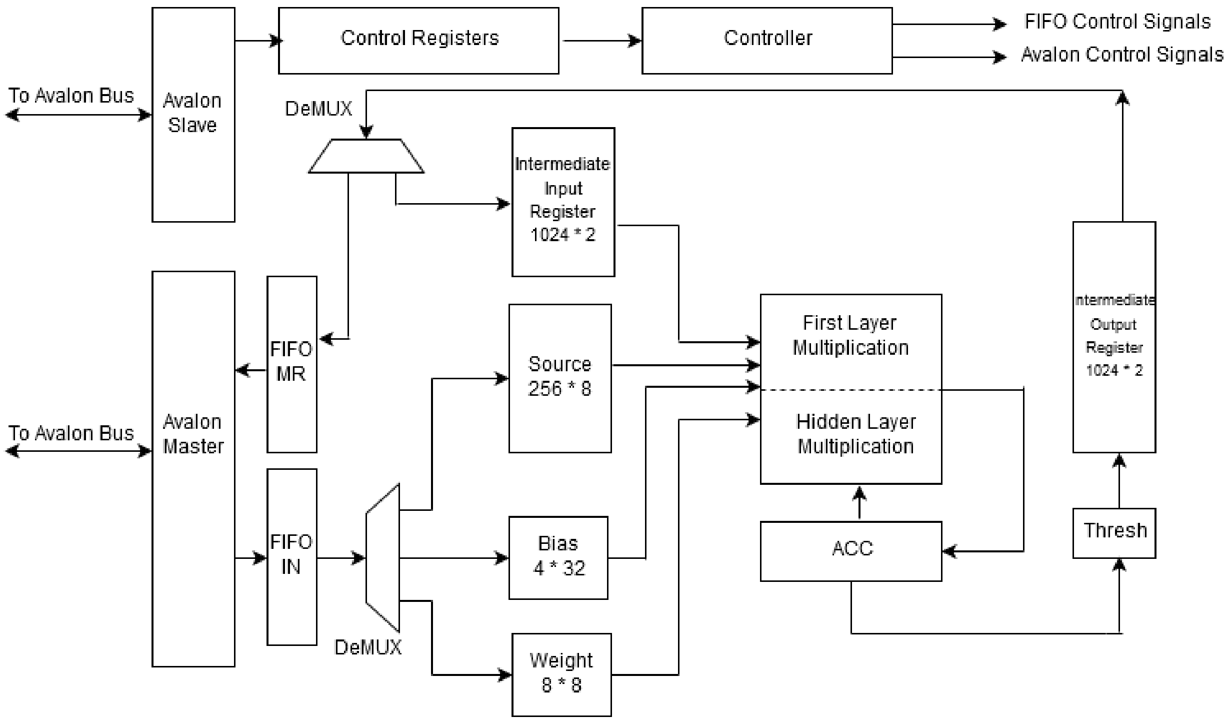

Fig 5: Datapath

2.4 Salient Features

2.4.1 Ternary Multiplication Unit

Ternary values consist of {-1,0,1} which are represented in 2's complement form as {11,00,01} respectively. The Table 2 gives the all possible combinations of inputs and their corresponding outputs. From this table the Karnaugh Map (K-Map) [Fig 6] is derived which gives the reduced form of ternary multiplication. The circuit derived from the K-Map is shown in Fig 7. The circuit is entirely made of basic AND and OR gates and there are no multiplexers. This not only improves the speed but also reduces the area and power as well. This circuit is instantiated 8 times to process the data in parallel.

Inputs (X)	Weights (W)	Outputs (O)
00	00	00
00	01	00
00	11	00
01	00	00
01	01	01
01	11	11
11	00	00
11	01	11
11	11	01

Table 2: Ternary Multiplication

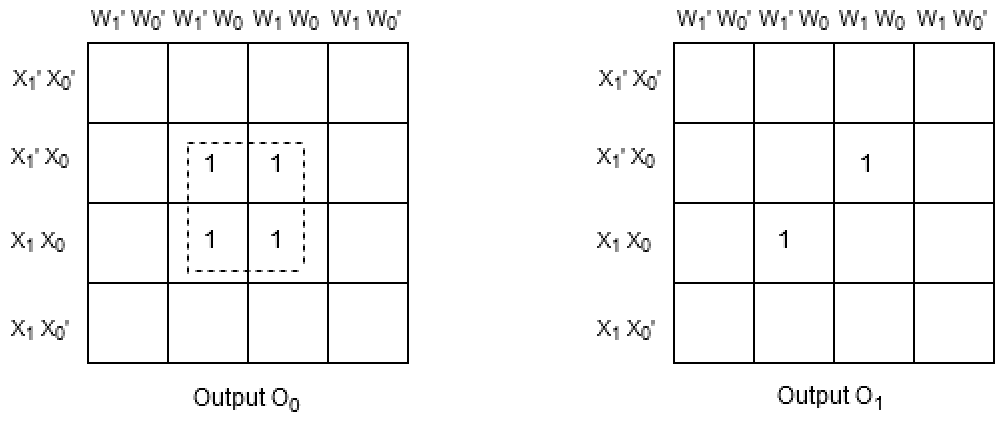

Fig 6: Karnaugh Map

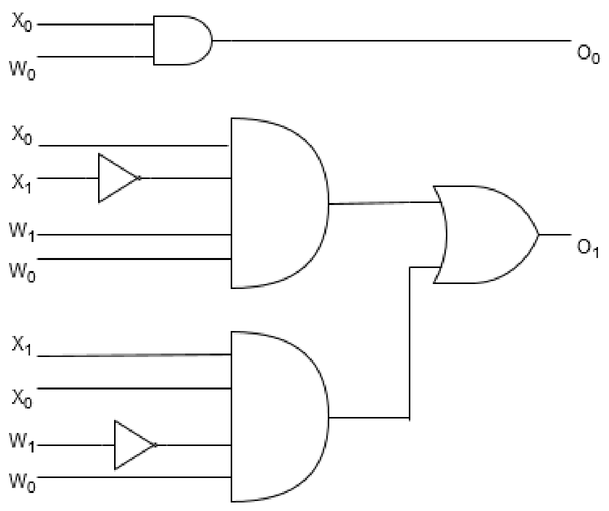

Fig 7: Ternary Multiplication Unit

2.4.2) Intermediate Output Registers

In the above architecture the output of the layers are written in an intermediate register and used as input for the next layer. This saves time to write and read the intermediate values from the memory.

2.4.3) Parallelization

The multiplication in both the input and hidden layers are done parallelly in groups of 8. The Avalon master has a width of 128 bits, so 4 integers can be read and written at a time.

2.4.4) Configurability

The circuit can be programmed 5 mins before. That is dynamic change of base address of data stored in memory is accommodated. The number of neurons in each layer can vary from 1 to 1024.

2.5 Changes from Initial Architecture

1. In the initial proposed architecture the ternary multiplication unit consisted of 9 to 1 Multiplexer. In the new architecture the ternary multiplication unit is derived from K-Maps.

2. In the initial architecture the output of each layer was written into the memory and the same values were read back as inputs for next layer. In the present architecture, the values are written back in local registers.

3. In the initial architecture, all the weights of the neuron were read and the multiplication and summation was done parallelly for all inputs. Now, only 8 weights are read and 8 multiplications are done parallelly.

3. RESULTS

The output matrix and the original matrix match to an extent of around 30%. Some features such as edges are visible. The system provides a 90% increase in speed with respect to optimized software solution.

4. CONCLUSIONS

This project has been a steep learning curve. First, I learnt how a hardware product is planned and implemented. I also learnt to write a comparatively large ASM design and verify its functioning at each level. The most important take away from this course was working on a mixed hardware and software design. There were a couple of places where I spent a lot of time on the project. First, validating the circuit on test benches took a lot of time as in the final ASM there were many states and many signals. Checking each signal was asserted and de-asserted at the correct instant took some time. Secondly, after each minor change in the VHDL code, I had to wait 25 mins to compile the design. In general the course was well paced. I faced a lot of difficulties during the lab sessions as the manuals were in French, and I always missed some important steps due to my limited knowledge of French. Hence, if there is an English translation of lab manuals, it would have been a bit easier. In the case of project, it would have been better, if there were groups of 4. The lectures were well oriented for labs and project.

I allow Professor Jean Pierre David to use this work and the associated source codes.

References

1. Avalon Memory Mapped Interface Specification