UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

Real-time display of a multiprocessor Spiking Neural Network

Master Thesis
submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by

Clément Nader

In partial fulfillment
of the requirements for the master in
ELECTRONICS ENGINEERING

Advisor: Bernardo Vallejo
Co-advisor: Dr. Jordi Madrenas Boadas
Barcelona, September 2021 - February 2022

Contents

Acknowledgments

I would like to thank all the people who helped me and contributed to my work:

- *Bernardo Vallejo for his daily advice, support and help during this thesis. Thank you for your time and the knowledge that you shared,*
- *Dr. Jordi Madrenas for his great supervision and proofreading of the thesis,*
- *Sasan Nikseresht and Diana Mata Hernández my other coworkers at the laboratory, thank you for the good working atmosphere you created and support you provided me,*
- *my flatmates Lydia Iracheta, Andrei Platonov and Rebecca Künnis for their daily psychological support,*
- *my previous coworkers and great friends Antoine Colson and Salvador Poveda for their help and conscientiousness during all these works,*
- *Mariline and Rémy Nader for their precious proofreading,*
- *and finally, my whole family and friends for their love and support.*

Abstract

Artificial Neural Networks (ANNs) are powerful computational tools that are used to solve complex pattern recognition, function estimation and classification problems not manageable by other analytical tools. They are inspired by the structure and function of the human brain and throughout their development, they have been evolving towards more powerful and more biologically realistic models.

A new generation of ANNs: Spiking Neural Networks (SNNs) have been developed. These networks emulate the neurobiological processing of information with temporal dynamics and precise timing. They are energy efficient and amenable to hardware application development. Such hardware SNNs work in real time and thus, having a real-time display makes full sense.

This current thesis presents the realization of a real-time display using High-Definition Multimedia Interface (HDMI) connected to an ongoing project. This project uses HEENS (Hardware Emulator of Evolved Neural System) architecture to implement hardware SNNs on Zynq FPGAs (Field Programmable Gate Arrays).

First of all, the communication to HDMI has been established, on two boards ZedBoard and Zynq ZC706. Screen resolution, video timings and color format have been studied. I2C (Inter-Integrated Circuit) communication has been examined and especially with the slave ADV7511, the HDMI transmitter, whose documentation has been also studied thoroughly. This transmitter has to be configured via I2C to be able to receive the HDMI signals and transmit them to the connector. Finally, all this acquired knowledge has enabled the actual implementation on the boards using VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language) of the HDMI connection. As a result, bands of colors can be shown on monitor displays.

Then, the representation of the Spiking Neural Network behavior has been made on screen, with plots to display the evolution in real time of the network. Communication has been established between the actual project architecture and the real-time display in order to receive the neural information and store it in a form which will allow their plot. At the same time, text generation has been implemented to be able to write text on screen. VHDL code has been developed to generate the plots with contours, ticks and labels as any standard chart.

The first plot that has been made is the raster plot of the neuron spikes over time. Next, in order to monitor furthermore the Spiking Neural Network, another plot has been implemented to display internal neural parameters. Among these analog parameters, one of the most important is the membrane potential, whose plot has been studied more specifically. In the end, both plots: the raster plot and the membrane potential plot (for four chosen neurons to monitor), are displayed at the same time on the screen.

To conclude, HDMI communication has been established on FPGA in order to monitor a Spiking Neural Network in real time. Two plots are displayed on the screen: the spikes over time and neural parameters for a few selected neurons of the network.

List of Figures

List of Tables

Acronyms

AER-SRT Address Event Representation over Synchronous Serial Ring.

AI Artificial Intelligence.

ANN Artificial Neural Network.

ASCII American Standard Code for Information Interchange.

BRAM Block RAM (Random-Access Memory).

CDC Clock Domain Crossing.

CEC Consumer Electronics Control.

CSC Color Space Conversion.

DDC Display Data Channel.

DDR Double Data Rate.

DMA Direct Memory Access.

DSP Digital Signal Processor.

DVI Digital Visual Interface.

FIFO First In/First Out.

FPGA Field Programmable Gate Array.

FSM Finite State Machine.

HCI HEENS Control Interface.

HDMI High-Definition Multimedia Interface.

HDTV High Definition Television.

HEENS Hardware Emulator of Evolved Neural System.

HEENS-MP HEENS Multi-Processor.

HMNR HEENS Monitored Neurons Register.

HSR HEENS Status Register.

HTR HEENS Transfer Register.

HTS	HEENS Toolchain Suite.
I2C	Inter-Integrated Circuit.
ID	Identifier.
IO	Input/Output.
IP	Intellectual Property.
LCM	Least Common Multiple.
LIF	Leaky Integrate-and-Fire.
LSB	Least Significant Bit.
MC	Master Chip.
MIF	Memory Initialization File.
MSB	Most Significant Bit.
NC	Neuromorphic Chip.
PC	Personal Computer.
PE	Processing Element.
PL	Programmable Logic.
PLL	Phase-Locked Loop.
PS	Processing System.
RAM	Random-Access Memory.
RGB	Red, Green and Blue.
ROM	Read-Only Memory.
RTL	Register Transfer Level.
SCL	Serial Clock.
SDA	Serial Data.
SDK	Software Development Kit.
SDTV	Standard Definition Television.
SIMD	Single Instruction Multiple Data.

SNAVA Spiking Neural Architecture for Versatile Applications.

SNN Spiking Neural Network.

TMDS Transition Minimized Differential Signaling.

VESA Video Electronics Standards Association.

VGA Video Graphics Array.

VHDL VHSIC (Very High Speed Integrated Circuit) Hardware Description Language.

VHSIC Very High Speed Integrated Circuit.

WNS Worst Negative Slack.

YCbCr Luma (Y), Blue-difference Chroma (Cb) and Red-difference Chroma (Cr).

1 Introduction

Artificial Neural Networks (ANNs) are powerful computational tools that allow solving complex pattern recognition, function estimation and classification problems that are not easily manageable by analytical tools outside of Artificial Intelligence (AI). They are inspired by the structure and function of the human brain and throughout their development, they have been evolving towards more powerful and more biologically realistic models. [1]

The third generation Spiking Neural Network (SNN) has been developed, which emulates the neurobiological processing of information with temporal dynamics and precise timing. In addition, with their digital signaling and sparse coding, SNNs are energy efficient and amenable to hardware application development. [1, 2]

Hardware SNNs can work in real time and thus having a real-time display of their operation makes full sense.

This current thesis presents the realization of a real-time display of SNN using High-Definition Multimedia Interface (HDMI), connected to an ongoing project.

This project is an implementation of a hardware architecture for general-purpose Spiking Neural Network. It is designed with a large scalability, reconfigurability and programmability to display the operation of neuronal models. [3]

It uses a new version of SNAVA (Spiking Neural Architecture for Versatile Applications) [4] called HEENS (Hardware Emulator of Evolved Neural System), which saves significant amount of control resources, compared to the other architectures and provides programmable connectivity. [5]

The HEENS architecture has been implemented on Zynq FPGAs (Field Programmable Gate Arrays) containing both an integrated Processing System (PS) and Programmable Logic (PL). In addition, a toolchain HTS (HEENS Toolchain Suite) has been developed and used to prototype, configure and execute the SNNs on the hardware. [5]

The goal of this thesis is to establish the HDMI communication between the PL of the FPGA and a monitor screen in order to display in real time the SNN operation.

In order to see and understand the behavior of the Spiking Neural Network with real-time display, a raster plot of the neuron spikes over time will be made, as well as the monitoring of some neurons through a plot of analog values such as their membrane potential.

2 State of the art

2.1 Biological neurons

Human brain is composed of billions of interconnected neurons that communicate via electrical spikes transmitted through synapses. Neurons are composed by three parts: dendrites, soma and axon (Figure 1). [3]

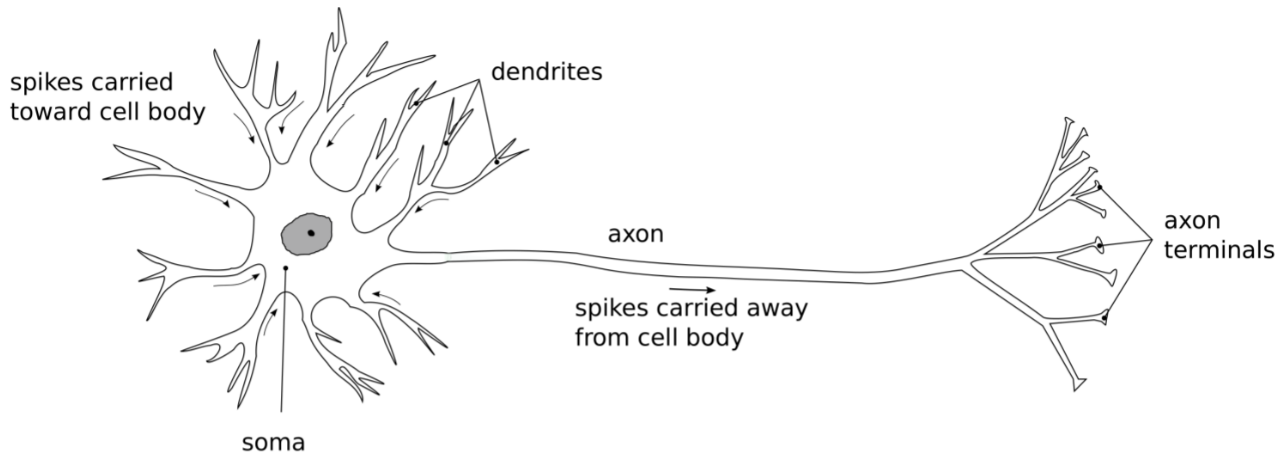

spikes carried toward cell body
dendrites
axon
axon terminals
spikes carried away from cell body
soma

Figure 1: Neuron structure. [3]

Dendrites behave as input points which receive and collect electrical signals from other neurons through synapses, and transmit them to the soma. The soma acts as the process unit of the neuron which manages the whole cell. The axon is the output canal which transmits generated spikes to its axon terminals, to connect through synapses to the next neurons. [3]

2.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are biologically inspired computational networks. They simulate the electrical activity of the biological brain. Processing Elements (PEs) are connected to each other by weighted links to inhibit or amplify the connection. It is through the adjustment of these connection weights that learning is emulated in ANNs. [6, 7, 8]

ANNs are used in a wide range of scientific fields such as finance, hydrology, or image and voice recognition. These applications can be regrouped into three categories of Artificial Intelligence: pattern classification, prediction, and control and optimization. [9]

2.3 Spiking Neural Networks

New models of Artificial Neural Networks have appeared that try to get closer and closer to the biological nervous system. During last years, a model has been the topic of much research: Spiking Neural Networks (SNNs), which is considered as the third generation of ANNs. These networks have a behavior closer to biological neurons by utilizing spikes, which enables to integrate the concepts of space and time. Their implementation is also easier than standard ANNs and can be made on fast and reliable hardware platforms. Indeed, they do not use multiplications, and the connection with spikes means only to send a single bit and not real number as other ANNs. [3, 10]

Each time a neuron receives a spike, its membrane potential increases as a function of the weight of the link (Figure 2). When it reaches the threshold (V_{thr}), it increases by an action potential (it fires) before decreasing below the resting potential (V_{res}). At the threshold potential, the neuron produces a spike. Membrane potential decays with time to converge to the resting potential.

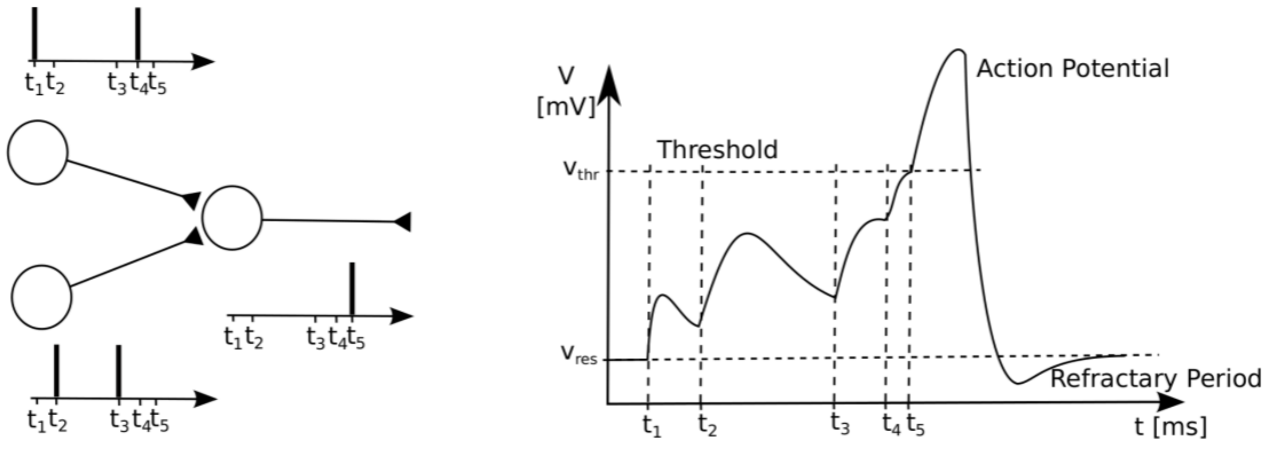


Figure 2: Spiking neurons and their membrane potential. [3]

Different neuron models exist to integrate this behavior into hardware, such as Leaky Integrate-and-Fire (LIF) model, Hodgkin–Huxley model or Izhikevich model. [10]

2.4 HEENS architecture

Hardware Emulator of Evolved Neural System (HEENS) is a hardware implementation of Spiking Neural Networks. The current project implements this HEENS architecture.

Description of the HEENS architecture

HEENS is an evolution of the SNAVA architecture [4]. It is characterized by a better resource utilization and by an enhanced programmability and scalability capabilities. Indeed, it allows the user to decide the number of Processing Elements (PEs) in the two-dimension array (rows and columns) and the number of virtual layers as a third spatial dimension. It supports also an online dynamic evolution and reconfiguration of the synapses interconnections. [3, 11]

This architecture can interconnect multiple chips in a ring topology (Figure 3). The Master Chip (MC) manages the network to configure and reconfigure the other nodes. It communicates with the general purpose processor and receives the instructions for the initialization and configurations.

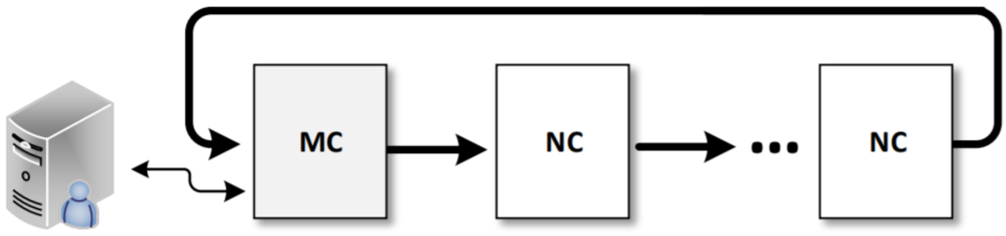


Figure 3: HEENS architecture with one Master Chip (MC) and n Neuromorphic Chips (NCs) connected as a ring. [3]

During the neural application, the MC behaves like any other chip. These other chips, called Neuromorphic Chips (NCs), process the neural algorithm. The MC collects the spikes from the other chips as well as from its own PEs. It then updates the neural and synaptic parameters such as the membrane potential. Finally, it distributes to the network the post-synaptic spikes.

HEENS process phases

The process is split in five phases (Figure 4): the initialization (*IPh*), the configuration (*CPh*), the execution (*EPh*), the evolution (*EvPh*) and the distribution (*DPh*).

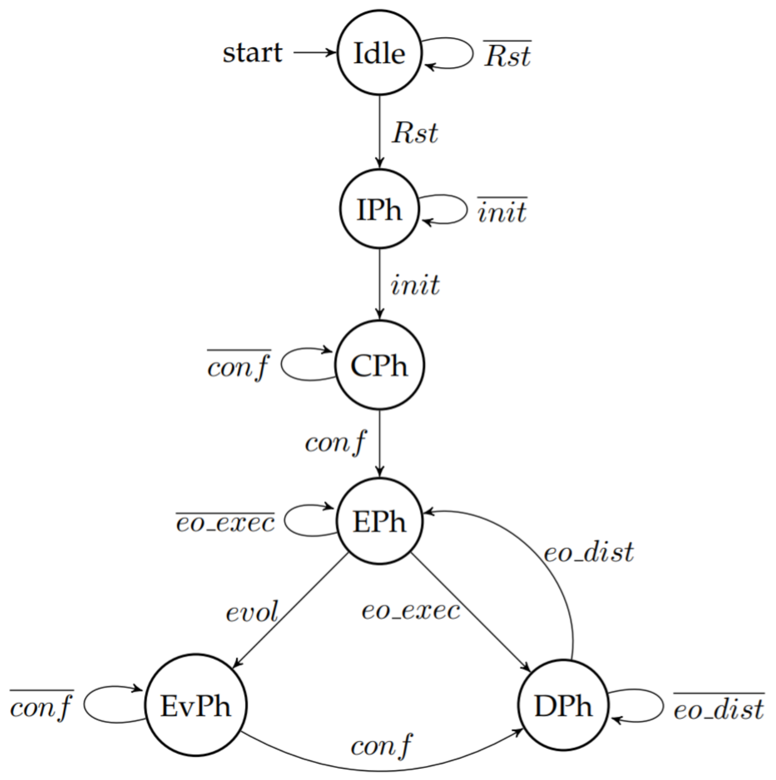


Figure 4: HEENS Finite State Machine. [3]

The process remains in the idle phase as long as the reset has not occurred. The registers are set to their default value.

- The initialization is made at a reset. The MC assigns the chip Identifier (ID) relative to each node, and transmits the ring size to the network.
- The configuration occurs after the initialization. The MC sends to the PEs their neural algorithm, their synaptic and neural parameters, as well as the map of the synaptic connections to the other PEs. Once all chips are configured, the configuration phase ends.
- In the execution phase, the neural algorithms are computed in a parallel way by each PE. Their states variables are updated. Once finished, the flag *eo_exec* is raised and the process can go to the next state.
- In the distribution phase, the spikes that have occurred are collected by the MC and broadcast to the network. Additionally, data will be sent to the PC (Personal Computer) in this state, and the communication with the real-time display that we will see in this thesis will also be made.

  Actually, the data is not sent to the PC at each distribution phase. It is stored in a FIFO (First In/First Out) memory and when the FIFO is half-full, the data is sent via Ethernet to the PC, while the other half starts to get filled in. This feature has been created instead of sending each time the data to increase the velocity, as the application development with Ethernet is slow. Thus, this thesis work is important to display the real-time behavior of the SNN, which cannot be done otherwise.
- At the end of each execution phase, the system checks if it has received an evolution command from the user. If it is the case, the system goes to the evolution phase instead of the distribution phase. The MC will send the newly received information to the selected PEs. They will then reconfigure themselves accordingly by adjusting their neural and synaptic parameters. Once all selected chips are re-configured, the evolution phase ends, and the system goes to the distribution phase.

The time unit is defined as the number of emulation cycles. These cycles include the execution and distribution phases (as well as the evolution phase if it appears), once the initialization and configuration have been realized.

HEENS Multi-Processor

The HEENS Multi-Processor (HEENS-MP) that can be seen in Figure 5 corresponds to a Neuromorphic Chip. It uses a SIMD (Single Instruction Multiple Data) computation scheme with a single control unit to achieve parallelism at data level in the PE array. This reduces area and gives a better computing efficiency. [3]

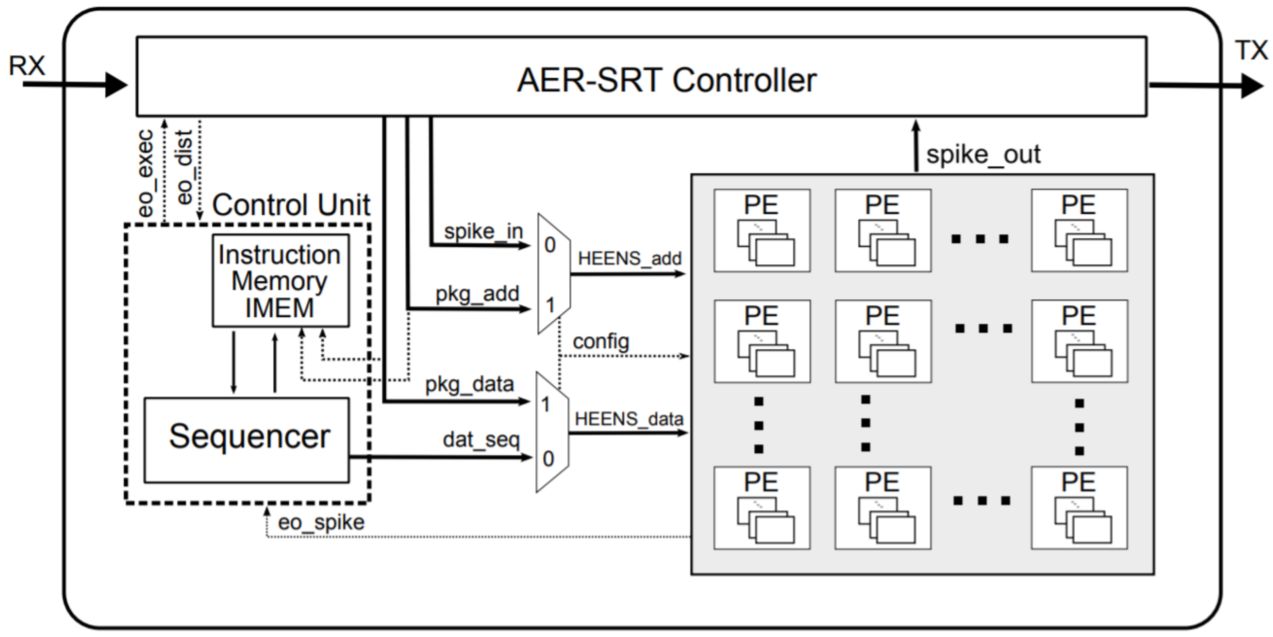


Figure 5: Block diagram of HEENS Multi-Processor. [3]

The chip is composed mainly by:

- communication buses: they permit information flow;
- the Processing Element array: each PE is a unit of SIMD-type execution, which processes the instructions of the neural algorithm;
- the control unit: it is in charge of managing the whole flow of data and instructions of the PE array, thanks to its sequencer. In addition, it generates the control signals to synchronize operations with the AER-SRT controller;
- the Address Event Representation over Synchronous Serial Ring (AER-SRT) controller: once PEs have processed their neural algorithm, a swept is made to get all the post-synaptic spikes. The spikes generated in each execution phase are encoded and gathered into a FIFO. This FIFO is read during the distribution phase and the post-synaptic spikes are transmitted to the network.

Address format of the Processing Elements

Each neuron (or PE) possesses an ID (Identifier) defined on 18 bits: 7 for the chip ID, 3 for the virtualization level, 4 for the row and 4 for the column.

The chip ID enables to know on which chip of the ring architecture the neuron is.

PEs can multiplex their behavior in time to emulate more than one neuron in an execution cycle. These additional neurons are said to be on virtualisation levels. Level 0 represents the physical layer (the HUB neurons). In addition, HEENS architecture accepts 7 other levels of virtualization (for a total of 8 levels) (Figure 6). [3, 11]

Each of these levels of virtualization has a PE array of a certain number of rows and columns.

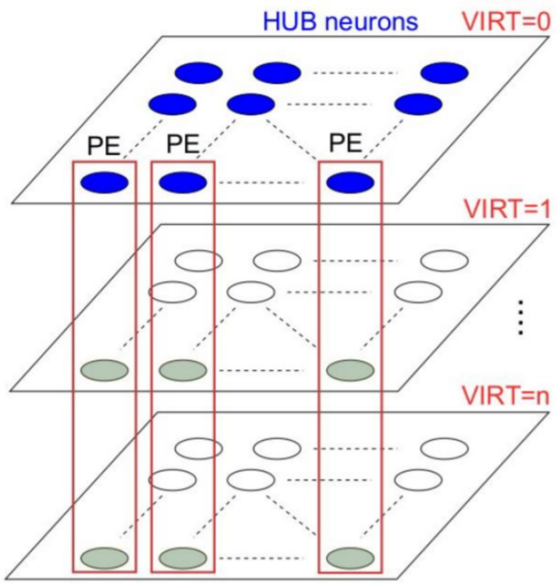


Figure 6: Virtualization of Processing Element arrays. [3]

3 Project development

3.1 Current state of the project

The project implements HEENS architecture on FPGAs and uses a toolchain (HTS) developed in Python in order to configure the Spiking Neural Network. This toolchain also provides a plot of the spiking neurons over time, but it is not in real time because of bandwidth and communication limitations, so it is necessary to wait to receive the information on the PC before displaying them.

For now, the project has been defined for two different types of boards: ZedBoard (Figure 7) and Zynq ZC706, a more complex board (Figure 8).

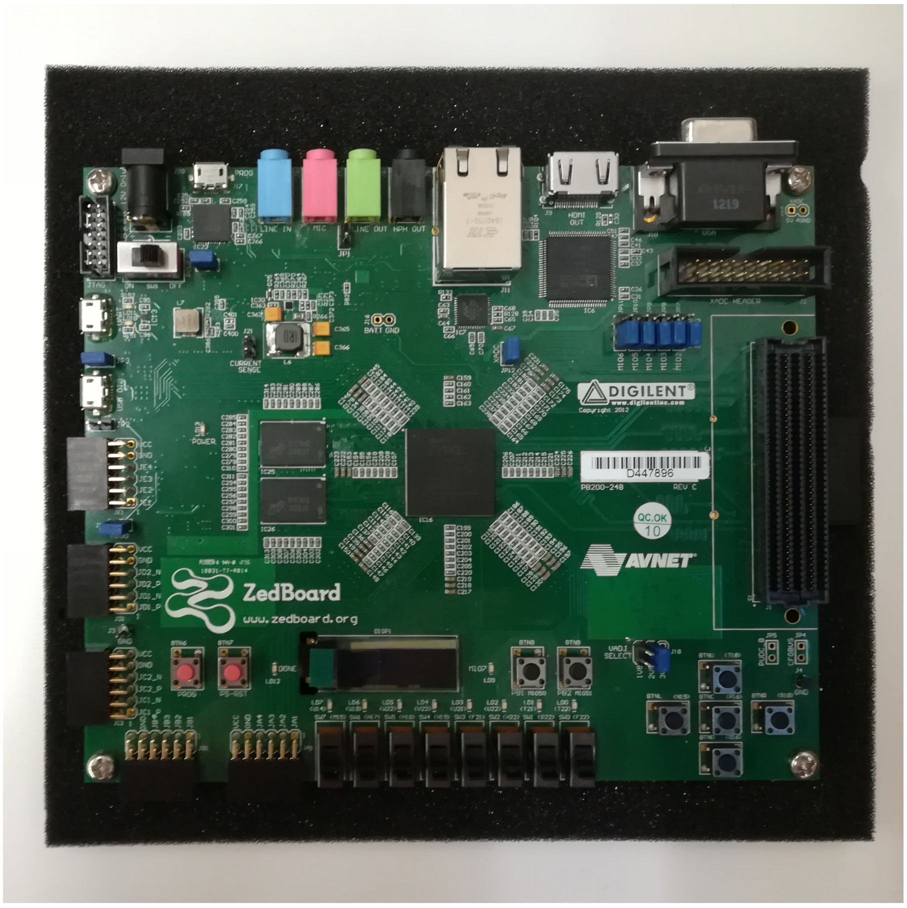


Figure 7: Picture of ZedBoard.

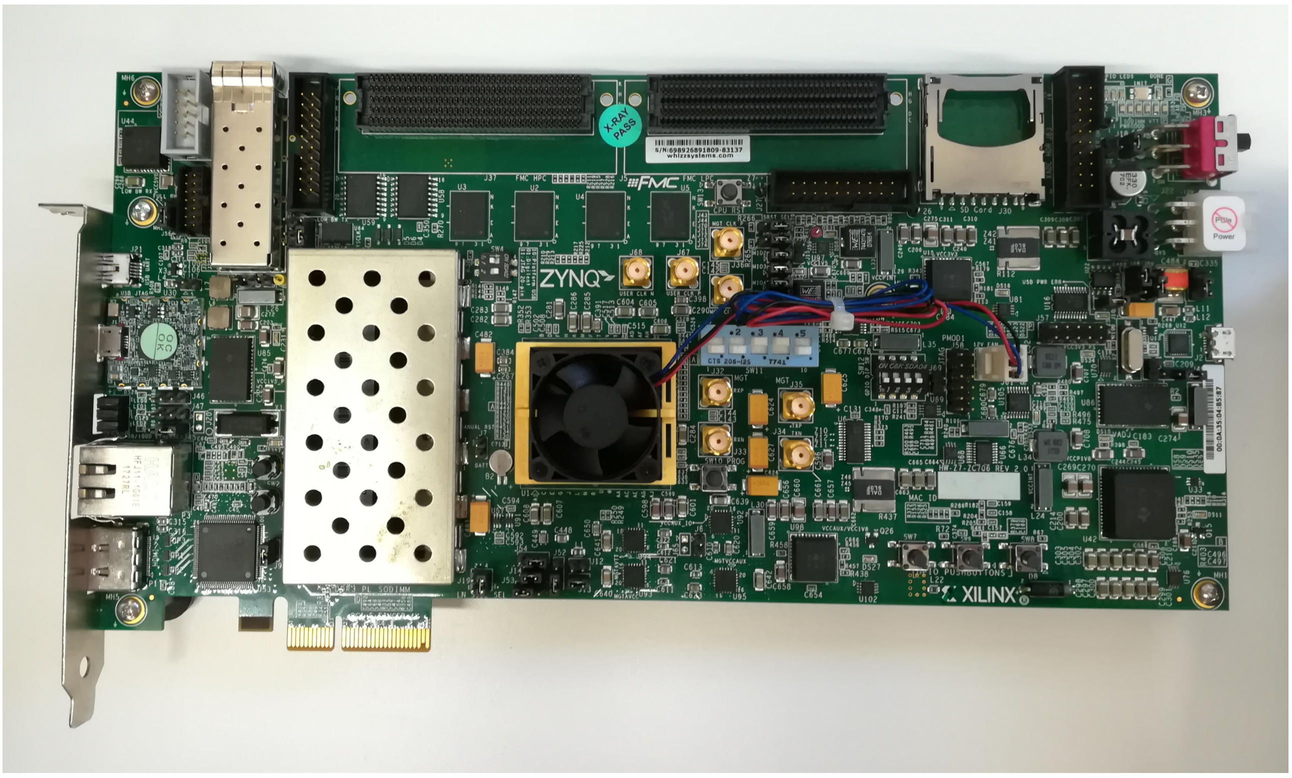


Figure 8: Picture of Zynq ZC706.

The neurons possess IDs defined on 18 bits: 4 for the column, 4 for the row, 3 for the virtualization level and 7 for the chip ID.

The project has not yet developed the virtualization levels, nor the connection to other boards. Thus, the number of neurons is only defined by the size of the Processing Element array on the Master Chip. Due to physical limits, on ZedBoard, arrays up to 5x5 can be defined, and on ZC706, up to 13x13.

The emulation time slot (the duration of the execution and distribution phases) in real time is set to 1 ms. This value has been chosen in order to have an onscreen display that is not too fast.

The data arising from the neurons is composed by the information that the neurons have produced a spike or not during the last execution phase, as well as internal neural parameters such as their membrane potential.

These spikes can be represented over time with as vertical axis the number of the neuron that has produced them. This is called a raster plot. Its display onscreen will be seen later in this document, along with the plot of neural parameters.

3.2 Vivado software and VHDL language

The thesis development has been made on the PL (Programmable Logic) in VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language). The software used is Vivado from Xilinx.

Verilog can also be used on Vivado in addition to VHDL. Besides, Xilinx provides a Software Development Kit (SDK) to create embedded applications on FPGA microprocessors (in C or C++ languages). The Processing System (PS) of both boards is a Zynq 7.

However, since in this thesis we are only working on the Programmable Logic (PL), the SDK is not necessary and all the current work can be done with Vivado.

In addition to the VHDL code, a constraint file is required (of extension .xdc for Xilinx Design Constraints). It contains the design constraints and in particular the definition of the physical pins of the board, their association to signals to be used in the code, as well as the voltage of the corresponding IO (Input/Output) pin banks. This file also contains the creation of the clock to specify to the software the clock constraints so that it is able to create timing reports and gives the slack values especially the Worst Negative Slack (WNS).

Once the Register Transfer Level (RTL) design is done (the VHDL code) and the constraints file is written, synthesis is realized to transform the RTL-specified design into a gate-level representation. Then, the implementation is performed to optimize the logic design, to place the logic cells of the design and to route the connection between the cells. Finally, a bitstream is generated to configure the device. [12, 13]

For the system verification and debugging, simulations can be done in order to check the behavior of any component with a plot of the signals against time. This helps understand precisely what happens to each signal. For example, it helps to understand the behavior of IPs (Intellectual Properties) [14] and their signals timings. Indeed, they are not defined by the user, so it can be hard to understand them fully. Simulations also helps to check that the components behave as expected and their interconnections.

To keep track of the advancement of the whole HEENS project, a Git repository was created. It contains the two different Vivado projects for the two boards, the Verilog and VHDL files for the PL, the C and C++ files for the PS, as well as the Python files of the toolchain. Git makes easier to have different people working on a single project, as each one can work on their own separated branches. And also, it allows to return to previous states, to check what changes have been made since then, and to incorporate some of these changes into the current state.

4 Screen display via HDMI

4.1 Principle

4.1.1 Introduction

The High-Definition Multimedia Interface (HDMI) is provided for transmitting digital television audiovisual signals from DVD players, set-top boxes and other audiovisual sources to television sets, projectors and other video displays. [15]

The HDMI cable and connectors carry four differential pairs that make up the TMDS (Transition Minimized Differential Signaling) data and clock channels (Figure 9). These channels are used to carry video, audio and auxiliary data.

In addition, there is a VESA (Video Electronics Standards Association) Display Data Channel (DDC) used for configuration and status exchange between a source and a sink; and there is a CEC (Consumer Electronics Control) line that is optional and provides high-level control functions.

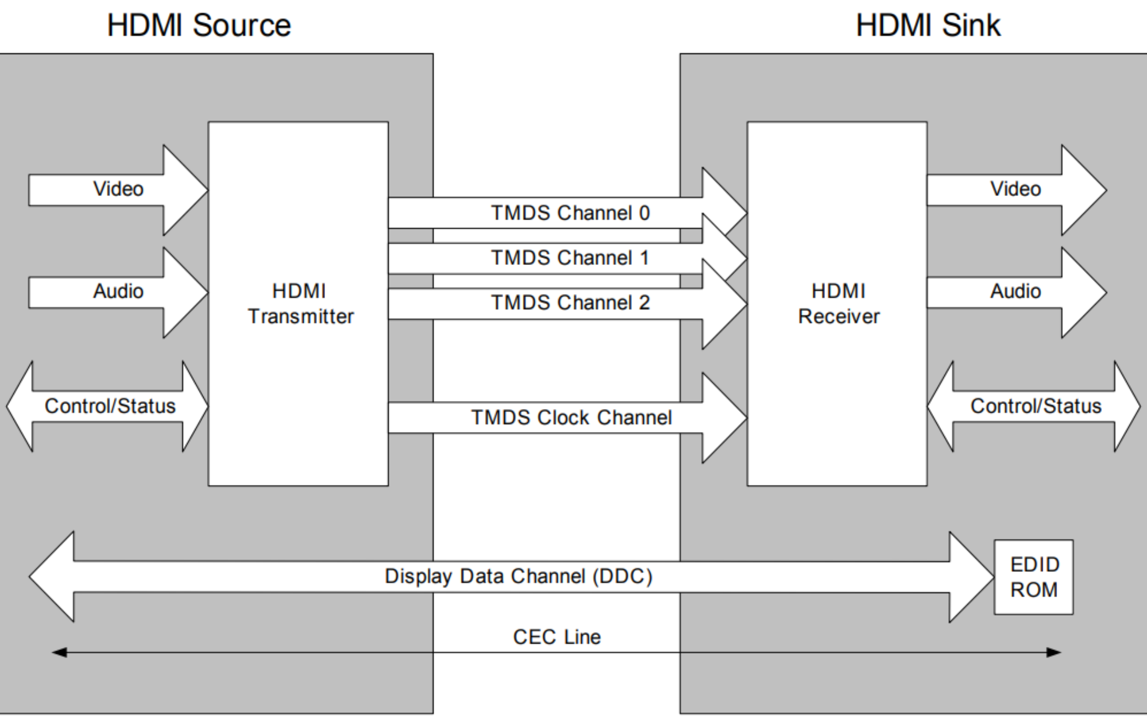


Figure 9: HDMI Block Diagram. [15]

The 3 TMDS data channels contain the audio, video and auxiliary data. The TMDS clock channel is typically at the video pixel rate, and is used as a frequency reference for the receiver to read the data channels.

In the project, audio is not used.

Video data can have a pixel size of 24, 30, 36 or 48 bits. The default 24-bits color depth requires a TMDS clock equal to the pixel clock rate, and higher color depths require faster TMDS clocks. The video pixels can be encoded in the color formats RGB, YCbCr 4:4:4 or YCbCr 4:2:2.

4.1.2 Aspect ratio and video resolution

The video resolution is the size in pixels of the height and the width of an image, video or screen. The aspect ratio corresponds to the ratio between the width and the height pixel sizes.

Monitors screen have had different aspect ratios. During 1990s, the most common one was 4:3, which corresponded also to the television and cinema formats (even though cinema would soon use 16:9). Then, in 2004, 16:10 resolution appears for computer monitors, it enables the display of two full pages of text side-by-side on one screen. [16] Finally, in 2008, television adopted the standard 16:9, followed by the computer monitors. [17]

The studied resolutions are:

- HD, High Definition (720p), 1280x720, aspect ratio 16:9;
- FHD, Full High Definition (1080p), 1920x1080, aspect ratio 16:9;
- WSXGA+, Widescreen Super Extended Graphics Array Plus, 1680x1050, aspect ratio 16:10.

The first two resolutions are the classical ones for HDMI (without going to very high definition), with 1080p being the default resolution of the project. The last one corresponds to the first screen that was used, which was older.

4.1.3 Video timings

In order to work with screens using FPGAs with either VGA (Video Graphics Array), DVI (Digital Visual Interface) or HDMI (High-Definition Multimedia Interface) protocols, video timings need to be respected. The transmission of a frame is separated in two phases: drawing pixels and blank interval. This blank interval is split, for both horizontal and vertical directions, between the back porch before the visible screen, the front porch after and finally the synchronization area (Figure 10). The horizontal synchronization signal separates two vertical lines, while the vertical signal separates two frames. [18]

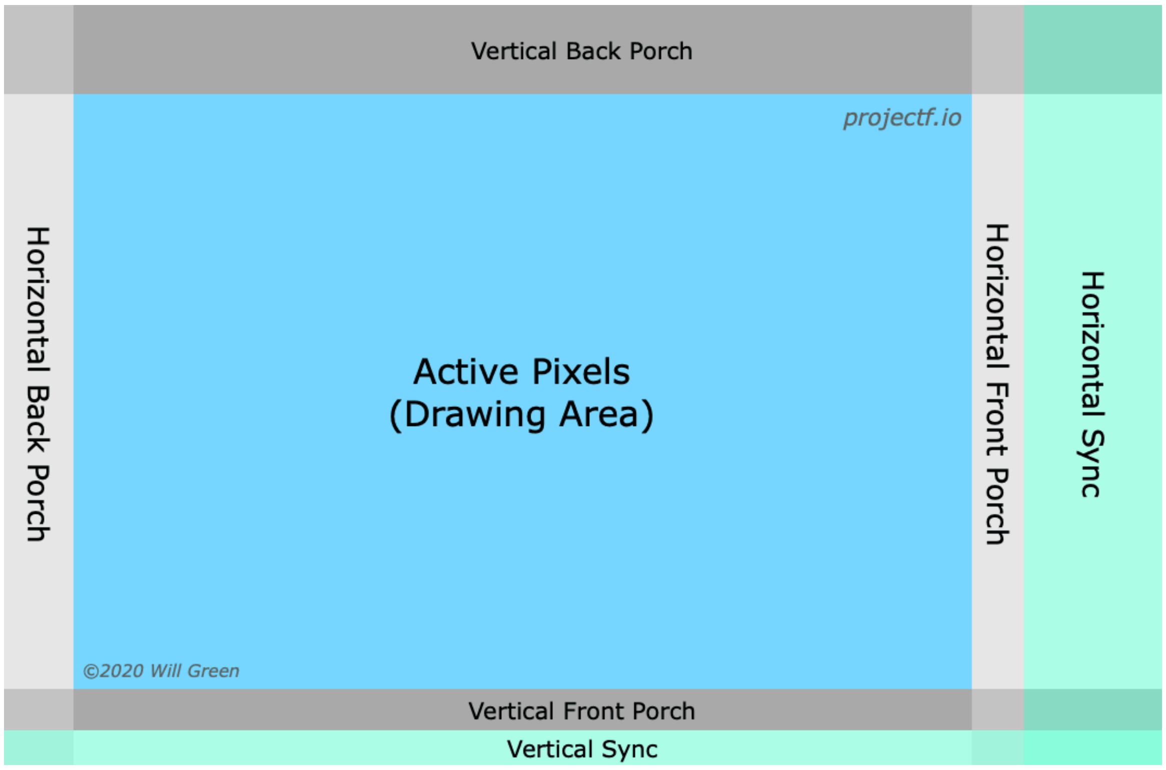


Figure 10: Screen areas (drawing pixels and blank interval). [18]

These area sizes are defined as a convention for the different resolutions (Table 1). Another important parameter is the polarity of the synchronization signals, whether they need to be high during the synchronization pulse and low the rest of time, or the contrary.

Screen resolution	Aspect ratio	Horizontal				Vertical			
		Back porch	Front porch	Sync width	Sync polarity	Back porch	Front porch	Sync width	Sync polarity
1280x720	16/9	220	110	40	positive	20	5	5	positive
1920x1080	16/9	148	88	44	positive	36	4	5	positive
1680x1050	16/10	288	104	184	negative	33	1	3	positive

Table 1: Screen areas size for 3 different resolutions. [18, 19]

The specifications of the sizes of these areas define the total number of pixels to be drawn for one frame. Then, the refresh rate of the screen defines the pixel clock at which the FPGA has to work in order to communicate with the screen (Table 2). The studied monitors work with a refresh rate of 60 frames per second.

Screen resolution	Total horizontal size	Total vertical size	Total number of pixels	Pixel clock	Typical pixel clock
1280x720	1650	750	1237500	74.25 MHz	75 MHz
1920x1080	2200	1125	2475000	148.50 MHz	150 MHz
1680x1050	2256	1087	2452272	147.13 MHz	150 MHz

Table 2: Pixel clock for 3 different resolutions.

For practical reasons and because HDMI communication still works fine, rounded pixel clocks have been selected for the project. In order to work for screen resolution of 1920x1080 which is the default one, and at 1680x1050 which is the first one used with the old screen, the pixel clock was fixed at 150 MHz.

4.1.4 Color format

To communicate via HDMI, different color formats can be used. The simplest one is RGB which uses the classical separation of the color between red, green and blue, and each of these components is defined on 8, 10 or 12 bits. Another format is YCbCr which uses also three signals on 8, 10 or 12 bits: Y is the luminance component (the brightness), and Cb and Cr are the color difference signals respectively for blue and red. [20, 21]

This second color format YCbCr is often used when noises are a prevalent problem, or simply when we want to reduce the size of color data. Indeed, human eye is very sensitive to luminance (or luma) which corresponds to Y signal, but less to chrominance (or chroma) which corresponds to Cb and Cr signals. Thus, we can dedicate more bits for Y and fewer bits for Cb and Cr, whereas with RGB we need many bits for each of the three colors to maintain a good image quality. [22]

YCbCr format enables the use of chroma subsampling. It is a compression that keeps a good luminance signal but reduces its color information. [23] As seen before, this does not affect much the quality of the image.

To talk about this compression, the terms 4:4:4, 4:2:2 and 4:2:0 are often used. The first number corresponds to the horizontal sampling reference (usually 4), the second one to the number of chroma samples in the first row, and the third one to the number of new chroma samples in the second row (usually it is either equal to the second number or zero). [23, 24, 25]

4:4:4 has no compression, 4:2:2 has a horizontal resolution halved and a full vertical resolution, and 4:2:0 has a horizontal and vertical resolution halved (Figure 11).

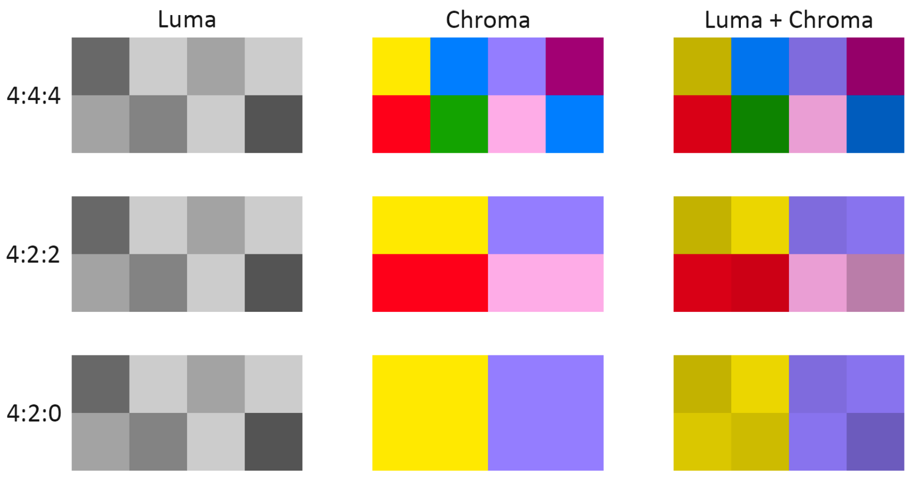


Figure 11: Chroma subsampling 4:4:4 vs 4:2:2 vs 4:2:0. [23]

In addition to subsampling considerations, there are two formats for YCbCr: SDTV (Standard Definition Television) and HDTV (High Definition Television). Since the screen resolutions for the project correspond to high definition, the format will be HDTV YCbCr.

4.2 Hardware specification

4.2.1 HDMI transmitter: ADV7511

Our FPGA cards (ZedBoard and Zynq ZC706) provide both a HDMI transmitter integrated circuit called ADV7511, for the HDMI output. This chip has first to be programmed via I2C (Inter-Integrated Circuit). Then, it will receive the pixel clock signal (*hdmi_clk*), the synchronization signals (*hdmi_hsync* and *hdmi_vsync*), the data enable signal (*hdmi_de*) and the data color signals (*hdmi_d* vector). In return, it will create the 3 TMDS data channels and the TMDS clock channel, as well as the auxiliary data channels, which will then be sent and carried by the HDMI connector.

Color signals configuration

These data color signals correspond to 36 pins on the chip. However, on the Zedboard, only 18 pins are connected (Figure 12), which correspond to pins 8 to 23, and on the ZC706, 24 pins are connected (Figure 13), which correspond to pins 4 to 11, 16 to 23 and 28 to 35.

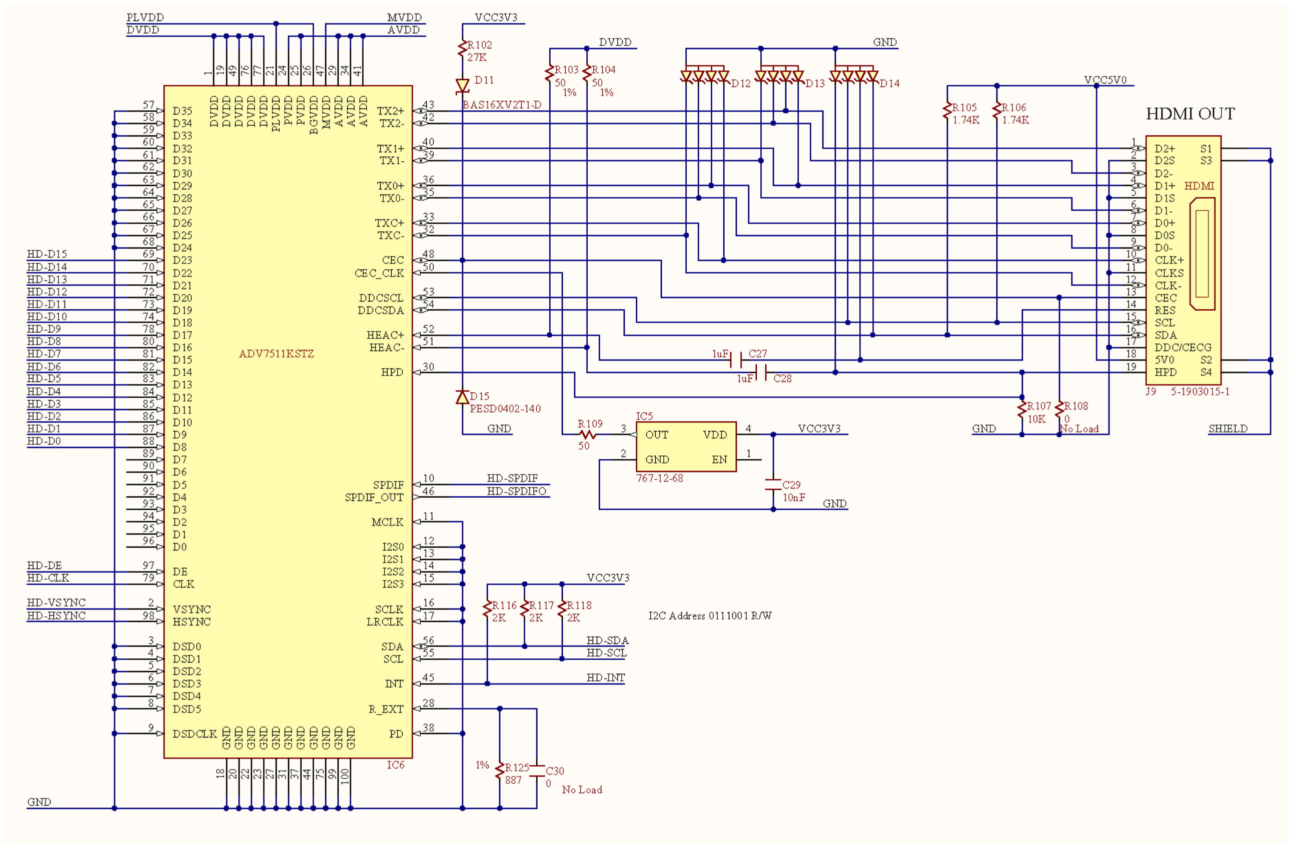


Figure 12: Schematic of the ADV7511 chip connections on ZedBoard. [26]

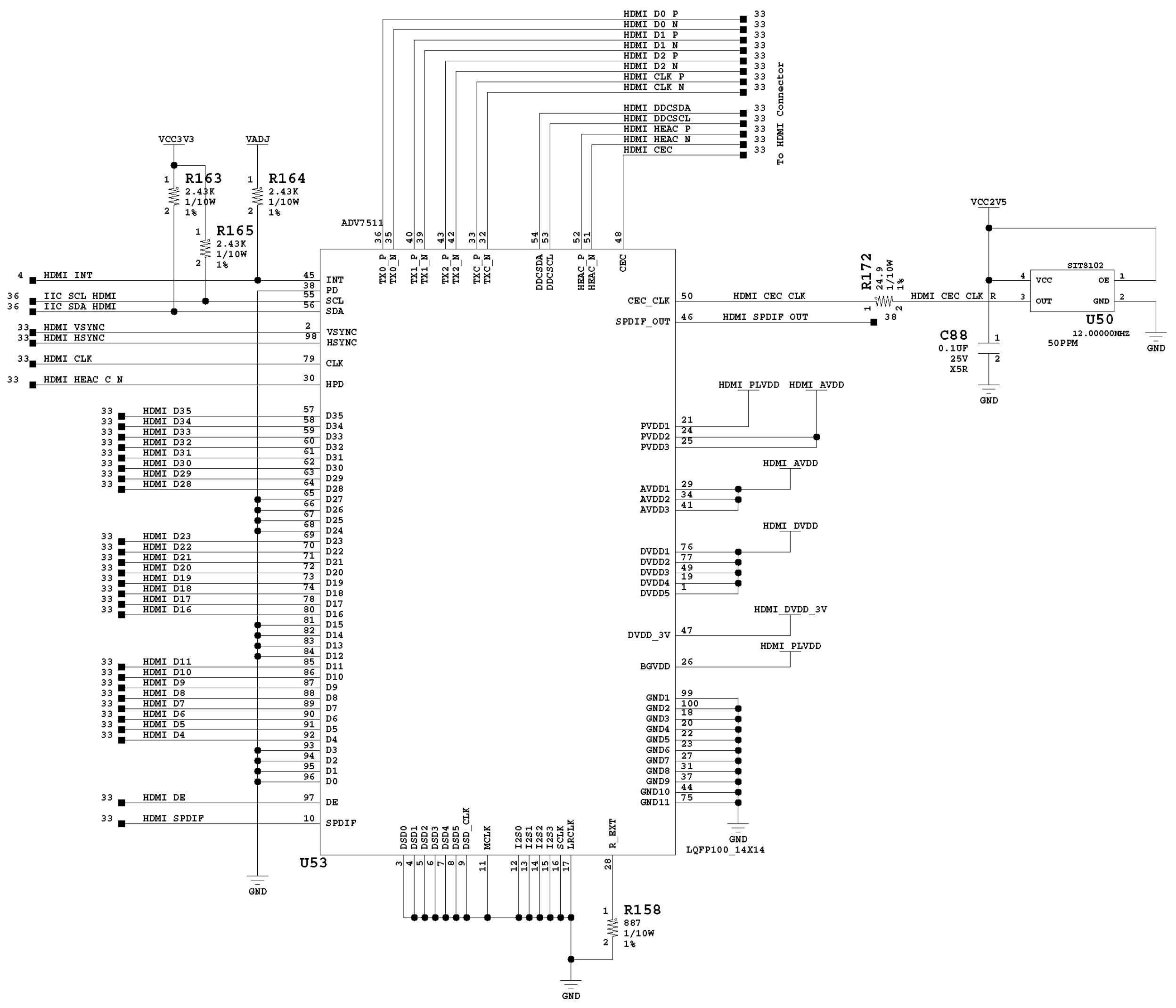


Figure 13: Schematic of the ADV7511 chip connections on Zynq ZC706. [27]

Now that we are aware of the connection available with the chip ADV7511 for both boards, we can study the manual of this HDMI transmitter [28] in order to configure it.

The ZedBoard that uses the data pins from 8 to 23 corresponds to Table 18 of the datasheet [28] (Figure 14).

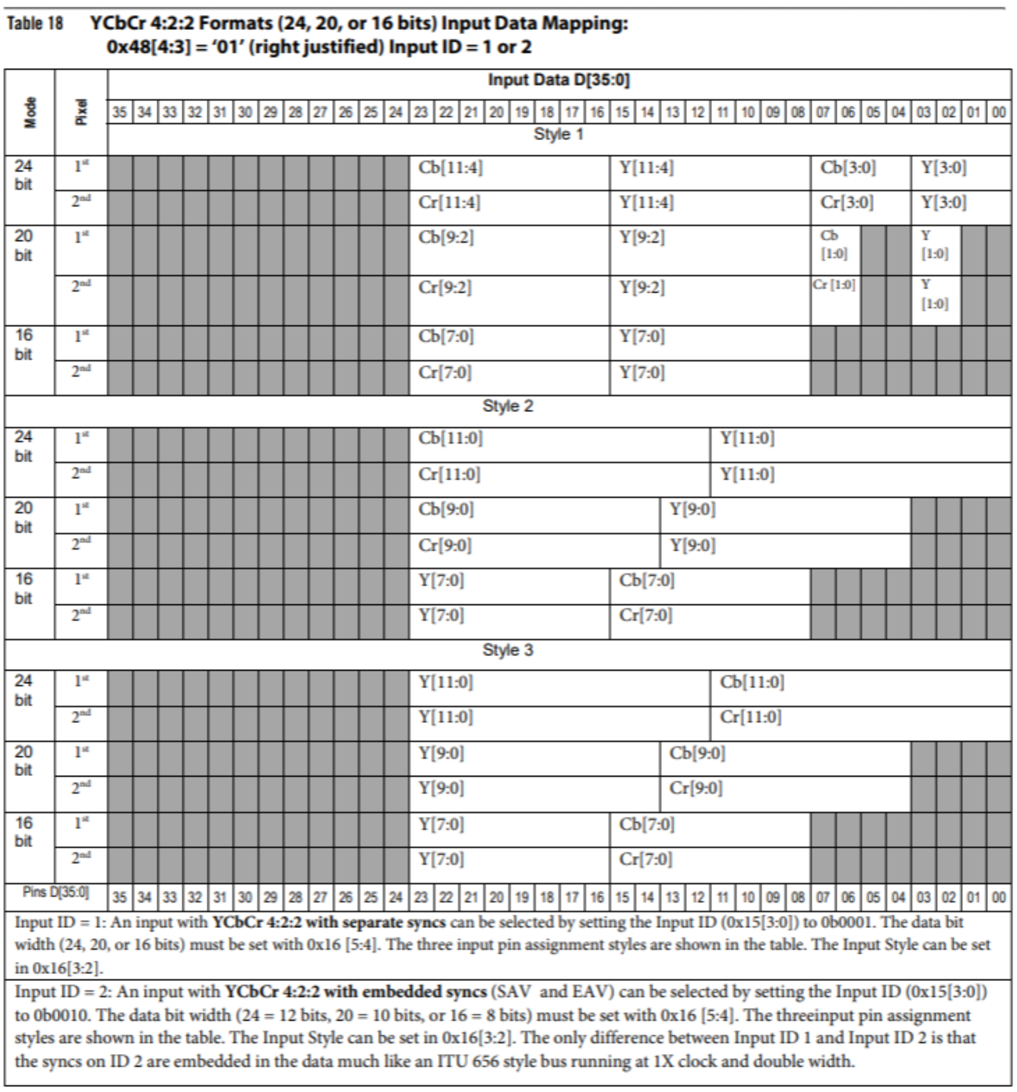


Table 18 **YCbCr 4:2:2 Formats (24, 20, or 16 bits) Input Data Mapping: 0x48[4:3] = '01' (right justified) Input ID = 1 or 2**

Mode	Pixel	Input Data D[35:0]																																			
		35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
		Style 1																																			
24 bit	1st	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Cb[11:4]								Y[11:4]								Cb[3:0]				Y[3:0]			
	2nd	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Cr[11:4]								Y[11:4]								Cr[3:0]				Y[3:0]			
20 bit	1st	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Cb[9:2]								Y[9:2]								Cb [1:0]		▨	▨	Y [1:0]		▨	▨
	2nd	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Cr[9:2]								Y[9:2]								Cr [1:0]		▨	▨	Y [1:0]		▨	▨
16 bit	1st	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Cb[7:0]								Y[7:0]								▨	▨	▨	▨	▨	▨	▨	▨
	2nd	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Cr[7:0]								Y[7:0]								▨	▨	▨	▨	▨	▨	▨	▨
		Style 2																																			
24 bit	1st	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Cb[11:0]												Y[11:0]											
	2nd	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Cr[11:0]												Y[11:0]											
20 bit	1st	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Cb[9:0]										Y[9:0]										▨	▨	▨	▨
	2nd	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Cr[9:0]										Y[9:0]										▨	▨	▨	▨
16 bit	1st	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Y[7:0]								Cb[7:0]								▨	▨	▨	▨	▨	▨	▨	▨
	2nd	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Y[7:0]								Cr[7:0]								▨	▨	▨	▨	▨	▨	▨	▨
		Style 3																																			
24 bit	1st	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Y[11:0]												Cb[11:0]											
	2nd	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Y[11:0]												Cr[11:0]											
20 bit	1st	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Y[9:0]										Cb[9:0]										▨	▨	▨	▨
	2nd	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Y[9:0]										Cr[9:0]										▨	▨	▨	▨
16 bit	1st	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Y[7:0]								Cb[7:0]								▨	▨	▨	▨	▨	▨	▨	▨
	2nd	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	Y[7:0]								Cr[7:0]								▨	▨	▨	▨	▨	▨	▨	▨
Pins D[35:0]		35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Input ID = 1: An input with **YCbCr 4:2:2 with separate syncs** can be selected by setting the Input ID (0x15[3:0]) to 0b0001. The data bit width (24, 20, or 16 bits) must be set with 0x16 [5:4]. The three input pin assignment styles are shown in the table. The Input Style can be set in 0x16[3:2].

Input ID = 2: An input with **YCbCr 4:2:2 with embedded syncs** (SAV and EAV) can be selected by setting the Input ID (0x15[3:0]) to 0b0010. The data bit width (24 = 12 bits, 20 = 10 bits, or 16 = 8 bits) must be set with 0x16 [5:4]. The threeinput pin assignment styles are shown in the table. The Input Style can be set in 0x16[3:2]. The only difference between Input ID 1 and Input ID 2 is that the syncs on ID 2 are embedded in the data much like an ITU 656 style bus running at 1X clock and double width.

Figure 14: Table of color data pins corresponding to input ID 1 and 2 on ADV7511. [28]

We need a color format YCbCr 4:2:2 with 16 bits. I have decided to use style 3, with Y being sent on bits 16 to 23, and on bits 8 to 15: Cb for the first pixel and Cr for the second. The chroma signals have half horizontal resolution.

ZC706 with its 24 pins corresponds to Table 16 of the datasheet [28] (Figure 15).

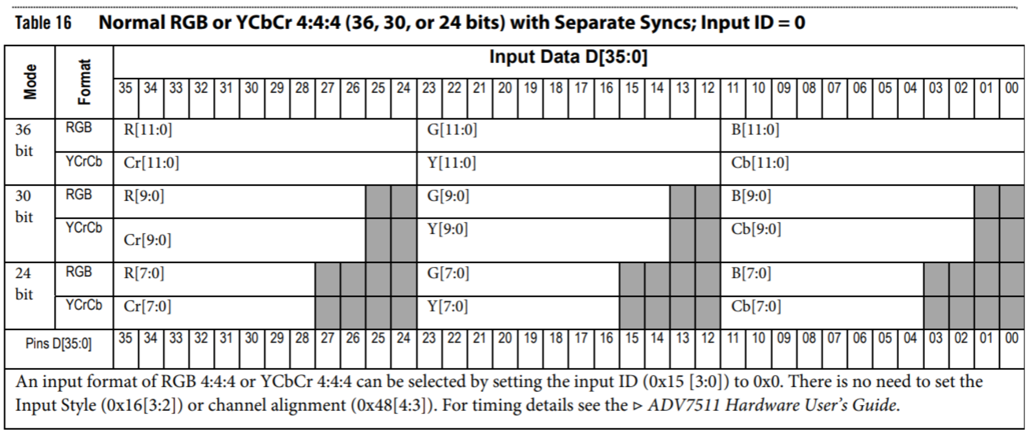


Table 16 Normal RGB or YCbCr 4:4:4 (36, 30, or 24 bits) with Separate Syncs; Input ID = 0

Mode	Format	Input Data D[35:0]																																			
		35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
36 bit	RGB	R[11:0]												G[11:0]												B[11:0]											
	YCrCb	Cr[11:0]												Y[11:0]												Cb[11:0]											
30 bit	RGB	R[9:0]										▨	▨	G[9:0]										▨	▨	B[9:0]										▨	▨
	YCrCb	Cr[9:0]										▨	▨	Y[9:0]										▨	▨	Cb[9:0]										▨	▨
24 bit	RGB	R[7:0]								▨	▨	▨	▨	G[7:0]								▨	▨	▨	▨	B[7:0]								▨	▨	▨	▨
	YCrCb	Cr[7:0]								▨	▨	▨	▨	Y[7:0]								▨	▨	▨	▨	Cb[7:0]								▨	▨	▨	▨
Pins D[35:0]		35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

An input format of RGB 4:4:4 or YCbCr 4:4:4 can be selected by setting the input ID (0x15 [3:0]) to 0x0. There is no need to set the Input Style (0x16[3:2]) or channel alignment (0x48[4:3]). For timing details see the ▷ *ADV7511 Hardware User's Guide.*

Figure 15: Table of color data pins corresponding to input ID 0 on ADV7511. [28]

For this board, either RGB format or YCbCr 4:4:4 (with no subsampling) can be chosen. I have decided to use RGB because it is the easiest one, as the generated colors data will be directly in this format, therefore no color conversion will be required.

As separate syncs are used for ZC706, I have chosen input ID 1 (and not 2) for ZedBoard in order to use also separate syncs, and have a more similar behavior.

The colors for each board have to be defined with each component on 8 bits.

Registers map

For the ADV7511 to work as a HDMI transmitter, some of its registers have to be set. When using I2C to set them, the whole 8 bits that compose the register must be sent. So even though only few bits of a register must be set to a value different from the default one, the whole byte information must be provided.

Fixed registers that must be set

There are some registers that have to be set to defined value for the chip to work properly (Figure A.1). When the whole 8 bits are not specified in the table from the documentation [28], these unspecified bits may have to be set to their default value that may be different from zero, and so they need to remain at this value. Actually, it is the case for one of these fixed registers: 0x9D (Figure A.2).

Main Power Up

In order to power up the transmitter, register 0x41(6) needs to be set to zero (Figure A.3).

HDMI or DVI mode

For the output mode of the chip, there is a choice between HDMI and DVI modes (Figure A.4). I have chosen DVI mode. And then 0xAF(1) has to be set to 0. However, the rest of this register 0xAF cannot be simply set to zero (Figure A.5).

HDMI is backward compatible with Digital Visual Interface (DVI), without any signal conversion or loss of video quality. It just necessitates that the video pixel encoding is RGB because DVI only supports this color format. [15]

Thus, video data needs to be converted to RGB color format, which can be done directly by the transmitter using Color Space Conversion (CSC).

Input Formatting Related Registers

For the input mode, there will be two different configurations for ZedBoard or ZC706. Besides, both use separate syncs. And the data length of each color component is of 8 bits.

For ZedBoard, as seen in figure 14, input ID is 1, data is right justified, and input style 3 has been chosen.

For ZC706, as seen in figure 15, input ID is 0, there is no data alignment and no input style to be set.

Then, the input formatting related registers can be mapped following the documentation [28] (Figures A.6 to A.9).

CSC related registers

Finally, registers have to be mapped for the Color Space Conversion (CSC).

For ZedBoard, colors data need to be converted from HDTV YCbCr to RGB (Figure A.10).

For ZC706, colors data is already in format RGB, so the identity matrix needs to be used (Figure A.11).

Summary registers map tables

Finally, all the registers to be set are summarized in Tables 3 and 4.

Register address	Register value		
	ZedBoard	ZC706	
Fixed registers to be set			
0x98	0x03		
0x9A	0xE0		
0x9C	0x30		
0x9D	0x61		
0xA2	0xA4		
0xA3	0xA4		
0xE0	0xD0		
0xF9	0x00		→ Fixed I2C Address
Main Power Up			
0x41	0x10		→ Main Power Up
Input Mode			
0x15	0x01	0x00	→ Input ID: for ZedBoard: 1: 16, 20, 24 bit YCbCr 4:2:2 (separate syncs) for ZC706: 0: 24 bit RGB 4:4:4 or YCbCr 4:4:4 (separate syncs)
0x16	0x3C	0x30	→ Output Format: 4:4:4, Input Color Depth: 8 bit, Input Style: for ZedBoard: 3, for ZC706: not defined
0x17	0x00		→ Vsync and Hsync Polarities: pass through, 4:2:2 to 4:4:4 Up Conversion Method: zero order interpolation
0x48	0x08	0x00	→ Normal Video Input Bus Order, Video Input Justification: for ZedBoard: right justified, for ZC706: not defined
Output Mode			
0xAF	0x04		→ DVI mode

Table 3: Register map for ADV7511 for both ZedBoard and ZC706 boards.

Register address	Register value		
	ZedBoard	ZC706	
Color Space Conversion			
0x18	0xE7	0xA8	→ A1
0x19	0x34	0x00	→ A1
0x1A	0x04	0x00	→ A2
0x1B	0xAD	0x00	→ A2
0x1C	0x00	0x00	→ A3
0x1D	0x00	0x00	→ A3
0x1E	0x1C	0x00	→ A4
0x1F	0x1B	0x00	→ A4
0x20	0x1D	0x00	→ B1
0x21	0xDC	0x00	→ B1
0x22	0x04	0x08	→ B2
0x23	0x1D	0x00	→ B2
0x24	0x1F	0x00	→ B3
0x25	0x24	0x00	→ B3
0x26	0x01	0x00	→ B3
0x27	0x35	0x00	→ B4
0x28	0x00	0x00	→ C1
0x29	0x00	0x00	→ C1
0x2A	0x04	0x00	→ C2
0x2B	0xAD	0x00	→ C2
0x2C	0x08	0x08	→ C3
0x2D	0x7C	0x00	→ C3
0x2E	0x1B	0x00	→ C4
0x2F	0x77	0x00	→ C4

Table 4: Register map (CSC) for ADV7511 for both ZedBoard and ZC706 boards.

4.2.2 I2C bus switch on ZC706

On Zynq ZC706, an I2C bus switch exists and it needs to be configured first before communicating with ADV7511. 8 different slaves are connected to this bus switch.

I2C communication can be made via the PS (Processing System) or the PL (Programmable Logic) (Figure 16). However, I wanted to try to have everything directly on the PL, so the I2C has been programmed on it.

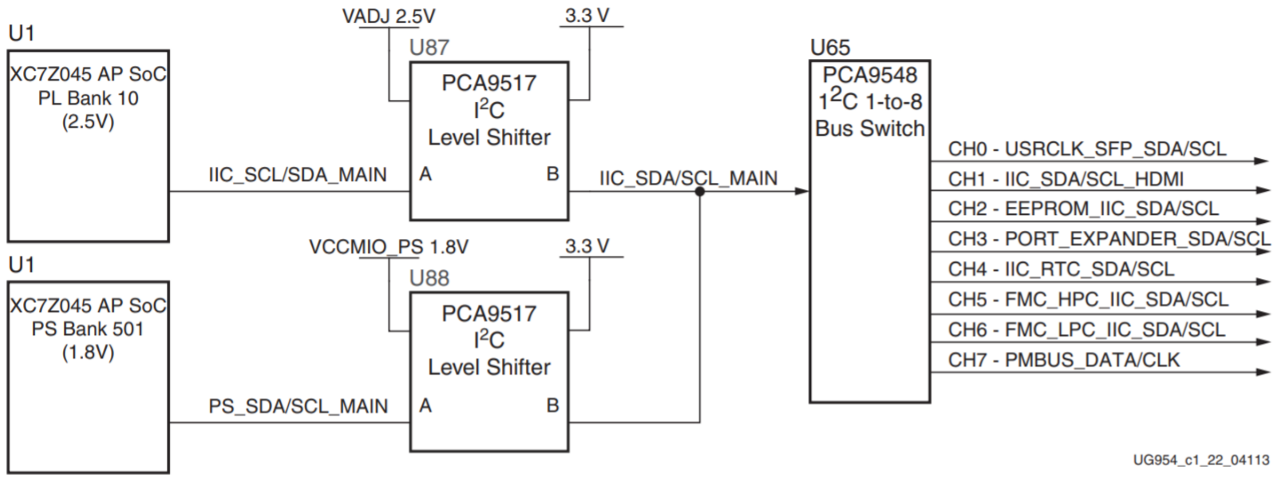


Figure 16: I2C bus topology on ZC706. [29]

The I2C address of the bus switch is 0b1110_100 [29]. A first I2C message has to be sent to the bus switch on 8 bits with a '1' at the position that corresponds to the channel(s) to be enabled. For example, in the case where HDMI transmitter ADV7511 (which corresponds to channel 1) is the only slave to be enabled, the I2C message 0b0000_0010 has to be sent. [29, 30]

Then, the communication can be made with the enabled slaves at their corresponding address (Figure A.12).

Actually, the address of ADV7511 on the I2C bus 0b0111_001 is the same as on the ZedBoard, which on its side does not possess an I2C bus switch, and the communication can directly be made to the HDMI transmitter using the corresponding address. [26]

4.3 Programming

4.3.1 Introduction

As an example to understand HDMI and its connection from ZedBoard, the work of Mike Field [31] was very helpful.

He was actually using only 8 pins on the ZedBoard: pins 8 to 15 with the same color format YCbCr 4:2:2, with each data on 8 bits. In order to send both luma (Y) and chroma (Cb or Cr) information in one clock period, he used DDR (Double Data Rate). This corresponds to the input ID 7 (Table 21 of the datasheet [28]), with style 1 (actually style 2 could also have been used but with pins 0 to 7).

In order to test the HDMI connection, an example was made only to show on the screen vertical bands of color. Two projects for this HDMI connection test have been created: for ZedBoard and Zynq ZC706. The sources used by the projects are located outside of the projects and are split between the common files used by both platforms, and specific files for each of the two platforms.

A Git repository has been created, different from the HEENS project one, for this HDMI test including both projects. It helped me while coding to easily check the differences made between the current code and a previous state, as well as to return to a previous state if necessary.

In order to find the pins of the different HDMI IOs, the schematics of both boards have been studied [26, 27] to create the two constraints files (.xdc).

4.3.2 Clock generation

The general clock on ZedBoard is single-ended and has a frequency of 100 MHz. On ZC706, it is differential and has a frequency of 200 MHz.

To generate a single-ended clock from a differential one, I have used an IP (Intellectual Property) from Xilinx library: a Differential Input Buffer.

To generate the pixel clock at 150 MHz, a Base PLL (Phase-Locked Loop) is used, with a first clock division by 2 for ZC706, to go to the frequency of 100 MHz that corresponds to the one of ZedBoard. Then, the clock is multiplied by 15, and the divider is 10 for the pixel clock. Actually, the PLL was also used to generate the 125 MHz HEENS clock using a 12 divider in the simpler project that was created to simulate the behavior of HEENS, before the incorporation to the actual code where HEENS clock is generated by the PS. This explains the value of 15 as the LCM (Least Common Multiple) between 3 and 5 corresponding to the numerator of the fractions $3/2$ ($= 150/100$) and $5/4$ ($= 125/100$).

4.3.3 Screen resolution

In order to keep the values of the sizes of the different parts of the screen (both visible and the blanking interval), a VHDL package has been created (*hdmi_resolution_pkg*). It contains the table for the three chosen resolutions (Table 1), as well as a constant to determine which resolution is chosen. It then contains the calculation of the different ranges of areas and defines them as constant for the other components to use them. It defines as well the sync polarities.

4.3.4 HDMI transmitter configuration via I2C

To program of the HDMI transmitter ADV7511, Inter-Integrated Circuit (I2C) is used.

Principle of I2C

I2C is a synchronous bus, defined at the start of 1980s by Philips company, in order to connect integrated circuits of a same automation application. [32]

It uses three cables: the ground GND, a data signal SDA (Serial Data) and a clock signal SCL (Serial Clock).

As the master transmits the clock to the slaves, any frequency can be used (as long as it is not too fast).

Every station connected on this bus possesses a unique address defined on either 7 or 10 bits. Addresses on 7 bits will be seen because they correspond to what we work with.

Stations are either master or slave. There is always only one master. Communications are initiated by the master and consist of frames.

I2C protocol uses 4 different events based on transitions on the two lines SDA and SCL (actually a fifth exists for repeated start but it is not used in this project) (Figure 17).

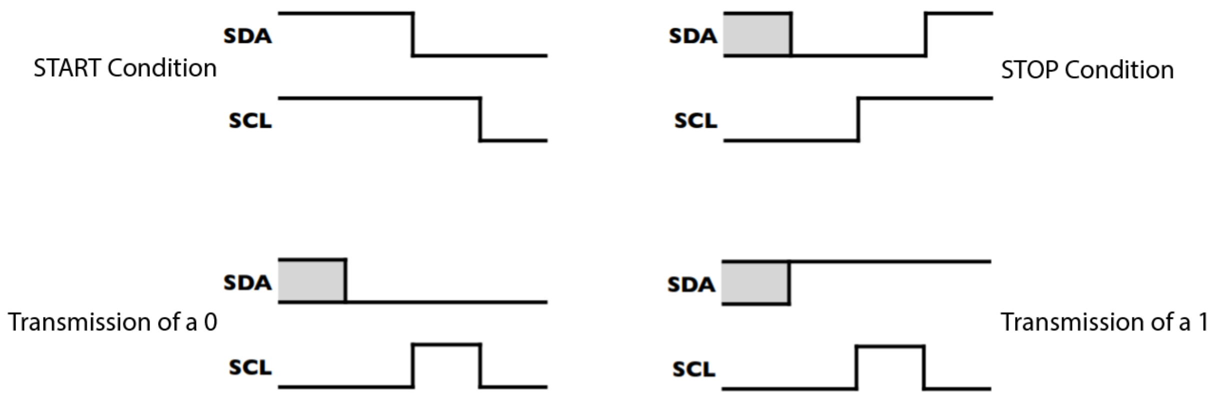


Figure 17: I2C events. [32]

The master can ask either for a writing or a reading frame.

The master sends the start condition (S), then the address (on 7 bits) of the slave with which it wants to communicate, then the last bit of this first byte is filled with either a zero (W) for a writing frame or a one (R) for a reading one. The slave of the corresponding address responds with an acknowledgment signal (A) at zero. After this, the length of the transmitted data can be zero, one or more bytes.

The data signal SDA can be controlled either by the master (in blue on the figures) or by a slave (in green).

For writing frames (Figure 18), for every data byte, the master sends its eight bits of data, and the slave responds with a A bit at zero. When the master has finished, it sends the stop condition (P).

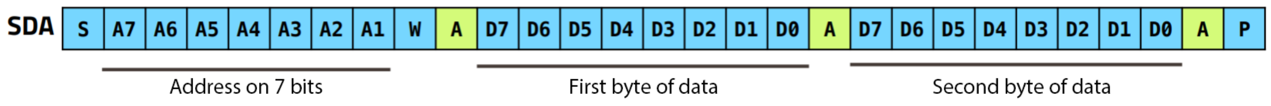


Figure 18: I2C writing frame with two bytes of data. [32]

For reading frames (Figure 19), for every data byte, the slave sends its eight bits of data, and the master responds with a A bit at zero. When the master has finished, it sends the stop condition (P).

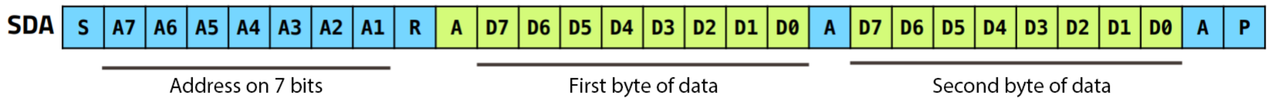


Figure 19: I2C reading frame with two bytes of data. [32]

Implementation of an I2C sender

I have implemented an I2C sender component in VHDL in order to program the HDMI transmitter. It only writes because the only needed direction is the master (the PL) which will write registers in the ADV7511 slave. This component was mainly inspired from Mike Field's work on ZedBoard communication with VGA and HDMI. [31]

The size of the data is two bytes, one for the register address and the other one for the register value to be written. Actually, at one given time, the data is only one byte: on ZC706 when the I2C bus switch has to be programmed to enable the HDMI transmitter slave. However, in order to avoid having two different components and thus to simplify the program, the same I2C sender of two-byte length has been used by simply doubling the data: the same information to enable the slave is sent two times in the message.

I2C does not work at high frequencies, so the clock which is at 150 MHz has to be divided. I have chosen to divide it by 512 ($= 2^9$) down to 293 kHz, which works to communicate with I2C. Actually, this data rate of 293 kbit/s corresponds to the fast mode in I2C which can go up to 400 kbit/s, while the standard mode is only up to 100 kbit/s. [33]

But in order to create the SCL signal, a clock two times faster is needed. So, actually, the clock divider will only be on 8 bits first to get this faster clock, and then it will be divided by two to generate the data clock. Indeed, it is much easier to generate a slower clock from a faster one than the contrary.

The implementation of the clock divider by 256 ($= 2^8$) is made with a counter on 8 bits which will increment at each rising edge of the input clock, and when this counter reaches 0b1000_0000, the divided clock is set to zero, and when it reaches 0b0000_0000, the divided clock is set to one.

Now that the faster clock is generated, in order to get a clock twice as slow, a small process is needed where at each rising edge of the faster clock, the data clock is inverted (from 0 to 1, and 1 to 0).

With the two clocks now created, the I2C sending process can be done. It uses a Finite State Machine (FSM) with three states (Figure 20): the ready state, the setting state when the shift registers are set, and then the registers shifting state when the data is transmitted and the registers are shifted. This FSM is sequenced at the data clock.

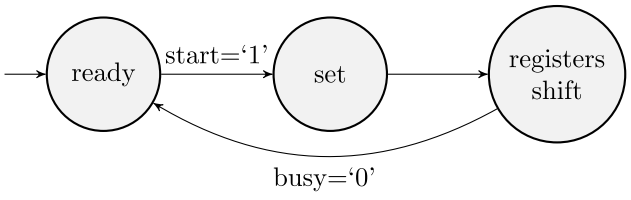


Figure 20: FSM of the I2C sending process.

When the component is ready, it can receive a start signal to go to the setting state. This setting state lasts only one clock period, and then the system goes to the shifting state, and when the component is not busy anymore and all the data is sent, it returns to the ready state.

The process uses shift registers on 29 bits (1 for the start event, 7 for the address, 1 for the writing bit, 1 for the acknowledgment, 8 for one data byte plus 1 for the acknowledgment so 9, times the number of data bytes, so here 2, and finally 1 for the stop event, which makes $1 + (7 + 1 + 1) + (8 + 1) \times 2 + 1 = 29$. There are 5 different shift registers.

The first is *data_sr* which contains the values for SDA: a zero for the start and the stop conditions, the value address, the writing bit at zero, as well as the actual data bytes to send. During the acknowledgments, a one is put. At each clock period during the register-shift state, it is shifted and filled with a one at the Least Significant Bit (LSB), while the Most Significant Bit (MSB) is read to give the value of SDA.

In order to have the SDA signal to be set at high impedance during the acknowledgments, another shift register *ack_sr* is implemented which will be one during the acknowledgments and zero the rest of the time. And at each clock period, it is filled with a zero, so that when all data has been sent, SDA signal is not at high impedance but takes the value of *data_sr* which will be one.

The third is *busy_sr* which is set to all ones during the setting state, and is filled with a zero at each clock period, and when the MSB is zero, it means that all the data has been sent and the component can return to the ready state.

The fourth and fifth shift registers are used to program SCL. They are the first clock quarter and the last clock quarter shift registers. In fact, with the clock two times faster, the data clock period can be cut in four intervals: when both clocks are at one, when only the data clock is at one, when only the faster clock is at one, when both clocks are at zero (Figure 21).

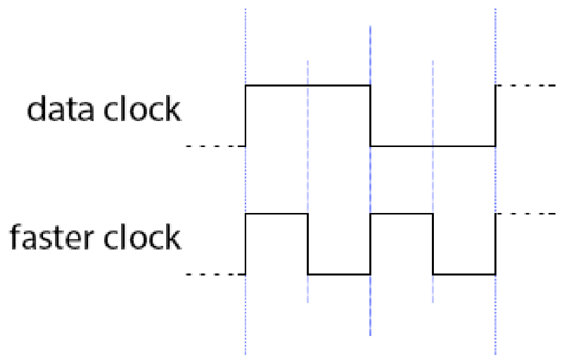


Figure 21: I2C clocks.

And to program SCL in order to make it possible to have the I2C events (Figure 17), during the second and the third intervals, SCL is set to one, during the first interval, it is set to the value of the *clk_first_quarter_sr* shift register, and during the last interval to the value of *clk_last_quarter_sr*.

The first quarter register is set to a one at the MSB and zero otherwise. The last quarter register is set to a one at the LSB and zero otherwise. At each clock period, they are filled with a one. Thus, during the first clock period, the SCL signal will be one during the three first intervals and zero during the last, so the start condition is fulfilled. During the last clock period, SCL will be zero during the first interval and one during the three last (to fulfill the stop condition). And in the middle of the transmission, it will be zero during the first and the last intervals, and one in the middle intervals, in order to be able to send a one or a zero accordingly to the SDA value (Figure 22).

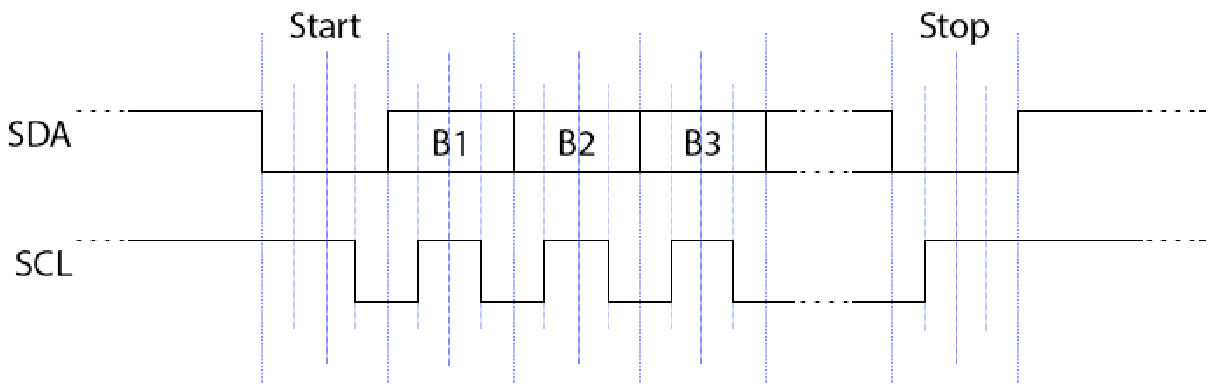


Figure 22: SDA and SCL signals during an I2C frame example.

Programming of the HDMI transmitter

In order to program ADV7511, two different descriptions have been made: one for ZedBoard and another for ZC706. The reasons are that they use different configuration of registers, and that for ZC706, the I2C bus switch needs to be programmed first.

The component uses an array of pairs of two bytes: the first for the register address and the second for the register value to be set. This array will be stored in ROM (Read-Only Memory).

Again, a Finite State Machine is used (Figure 23). For ZedBoard, there are five states.

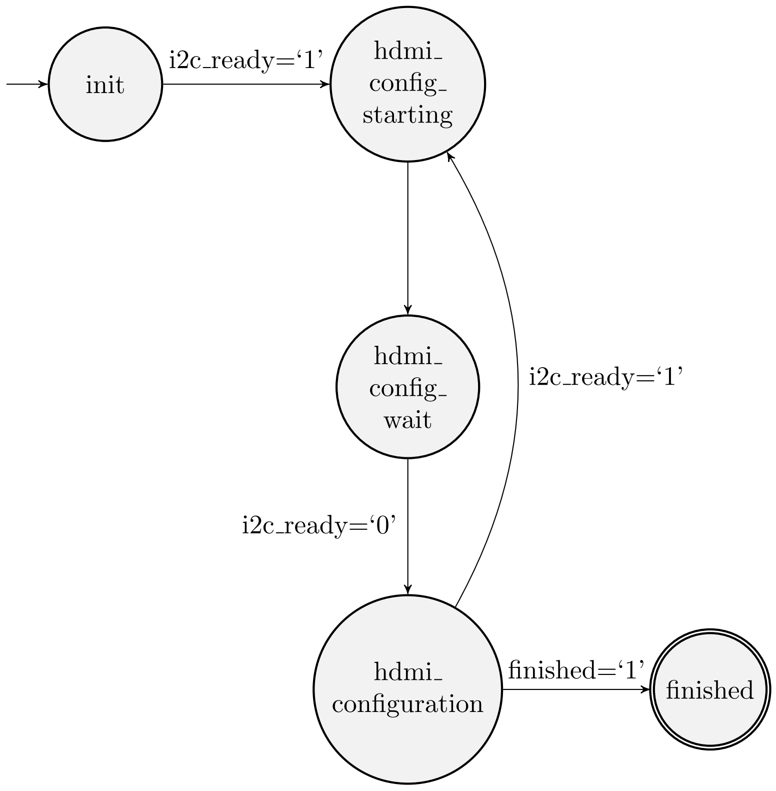


Figure 23: FSM of the HDMI transmitter configuration process for ZedBoard.

The initial state is where the process starts and returns to in case of reset. In this state, the system waits for the I2C sender component to be ready.

Then, the configuration starting state is where the address, data and start signals are given to the I2C sender. The address is the I2C address of ADV7511. The data is set to the corresponding value inside the array. The start signal is set to one.

The next state is to wait for the ready signal of the I2C sender to return to zero (so the transmission will have begun). This enables to check in the next state when it will return to one which will mean that the I2C sender has finished to transmit the data and is ready to send new one.

Then, there is a state to wait for the end of the configuration and for the ready signal to be one (as said just before). When the I2C sender is ready, the process returns either to the starting state if there is more data to send, or otherwise to the finish state. In parallel, the pointer in the array is incremented when it returns to the starting state in order to send the next data.

For ZC706, the FSM is changed to introduce new states before the HDMI configuration states in order to configure the I2C bus switch (Figure 24). Three new states are created that are the same as for the HDMI configuration but for the bus switch configuration.

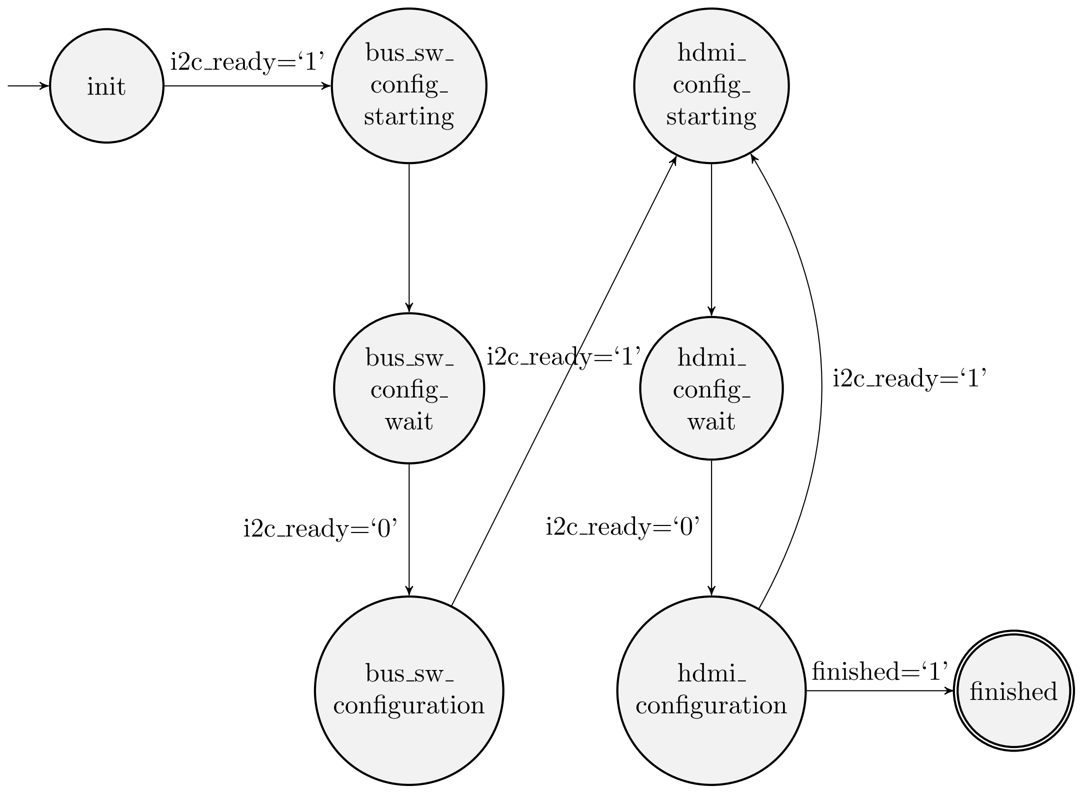


Figure 24: FSM of the HDMI transmitter configuration process for ZC706.

From the initial state, the system now goes to the bus switch configuration starting state when the I2C sender is ready. In this starting state, the address, data and start signals are given to the I2C sender. The address is the I2C address of the bus switch. The data is set to twice the byte 0b0000_0010 to enable ADV7511 as slave. The start signal is set to one.

Then, there is again a state to wait for the ready signal to return to zero as in the HDMI configuration. This will mark that the sending process has begun.

And the last new state is to wait for the sending to finish and for the I2C sender to be ready. Then, the process goes directly to the starting state of the HDMI configuration. Indeed, in this bus switch configuration, there is only one message to send. Therefore, there is no need to return to its starting state (and no pointer to increment).

4.3.5 Conversion from RGB to YCbCr

As the generated colors data will be in RGB format, and now that we know that each component of colors has to be on 8 bits, we can study the conversion from RGB to HDTV YCbCr, needed on ZedBoard.

In order to transform RGB to YCbCr for HDTV and with a full range from 0 to 255 for RGB signals, there are equations to follow. [21]

$$\begin{cases} Y = 16 + 0.183R + 0.614G + 0.062B \\ Cb = 128 - 0.101R - 0.338G + 0.439B \\ Cr = 128 + 0.439R - 0.399G - 0.040B \end{cases} \tag{1}$$

However, these equations have to be transformed to be implemented in a FPGA. [20]

As multiplying by value lower than one (or dividing) is not easy with FPGA, a better and easier way is to multiply this factor by a power of two (large enough), and then we can multiply with the integer value of this obtained number. Finally, in order to get the desired value, the result needs to be divided by the power of two that has been introduced, which will be done by a right bit shift (by discarding the LSBs).

I have chosen here 8 as the power of two, so the multiplication by 256 ($= 2^8$) has to be done:

$$\begin{cases} Y = 16 + \frac{46.848}{256}R + \frac{157.184}{256}G + \frac{15.872}{256}B \\ Cb = 128 - \frac{25.856}{256}R - \frac{86.528}{256}G + \frac{112.384}{256}B \\ Cr = 128 + \frac{112.384}{256}R - \frac{102.144}{256}G - \frac{10.240}{256}B \end{cases} \tag{2}$$

Then, the multiply factors have to be rounded in order to work with integers.

$$\begin{cases} Y \approx 16 + \frac{47}{256}R + \frac{157}{256}G + \frac{16}{256}B \\ Cb \approx 128 - \frac{26}{256}R - \frac{87}{256}G + \frac{112}{256}B \\ Cr \approx 128 + \frac{112}{256}R - \frac{102}{256}G - \frac{10}{256}B \end{cases} \tag{3}$$

Now, we can re-write the multiplications as only bit shift and sum/difference operations. In order to do so, factors need to be decomposed in powers of two.

For example, the factor 47 in binary is 0b0010_1111, from which it can be deduced the decomposition in powers of two: $47 = 2^5 + 2^3 + 2^2 + 2^1 + 2^0$. Or rather we can remark that when there are consecutive '1's (or consecutive powers of two), it corresponds also to the difference between the maximum power of two plus one and the minimum power of two. That is in the example: $2^3 + 2^2 + 2^1 + 2^0 = 2^4 - 2^0$. Therefore, it reduces the number of bit shift operations and sums/differences that will be needed. It is only when there are more than two consecutive powers of two that it permits to reduce the number of operations. To conclude, the example results to $47 = 2^5 + 2^4 - 2^0$, and then $47R = (R << 5) + (R << 4) - (R << 0)$.

Actually, this method is called the Booth's multiplication algorithm. [34]

This can be done for every factor:

Sign	Factor	Binary value	Decomposition in powers of two	
		Y		
$+$	47	0b0010_1111	$2^5 + 2^4 - 2^0$	$\rightarrow$ R
$+$	157	0b1001_1101	$2^7 + 2^5 - 2^2 + 2^0$	$\rightarrow$ G
$+$	16	0b0001_0000	2^4	$\rightarrow$ B
		Cb		
-	26	0b0001_1010	$2^4 + 2^3 + 2^1$	$\rightarrow$ R
-	87	0b0101_0111	$2^6 + 2^4 + 2^3 - 2^0$	$\rightarrow$ G
$+$	112	0b0111_0000	$2^7 - 2^4$	$\rightarrow$ B
		Cr		
$+$	112	0b0111_0000	$2^7 - 2^4$	$\rightarrow$ R
-	102	0b0110_0110	$2^6 + 2^5 + 2^2 + 2^1$	$\rightarrow$ G
-	10	0b0000_1010	$2^3 + 2^1$	$\rightarrow$ B

Table 5: Factors decomposition in powers of two, for RGB to HDTV YCbCr conversion.

Finally, all multiplications and divisions can be transformed to bit shift operations and additions/subtractions.

$$\begin{cases} Y \approx 16 + (((R << 5) + (R << 4) - (R << 0)) \\ \quad + ((G << 7) + (G << 5) - (G << 2) + (G << 0)) \\ \quad + ((B << 4))) >> 8 \\ Cb \approx 128 + (-((R << 4) + (R << 3) + (R << 1)) \\ \quad - ((G << 6) + (G << 4) + (G << 3) - (G << 0)) \\ \quad + ((B << 7) - (B << 4))) >> 8 \\ Cr \approx 128 + (((R << 7) - (R << 4)) \\ \quad - ((G << 6) + (G << 5) + (G << 2) + (G << 1)) \\ \quad - ((B << 3) + (B << 1))) >> 8 \end{cases} \tag{4}$$

In Mike Field's project [31], the conversion was made by multiplying the factors value by 32768 ($= 2^{15}$) instead of 256 ($= 2^8$). It enables having more precision in the result, but with tests on screen, the difference was not visible. Besides, the multiplication was made using DSPs (Digital Signal Processors) instead of the decomposition in powers of two and bit shifts. DSPs are specialized microprocessors that are used to complete complex calculations with the use of adders and multipliers. [35]

4.3.6 Generation of the screen

The HDMI connection schematic can be seen in Figures 25 for ZedBoard and 26 for ZC706.

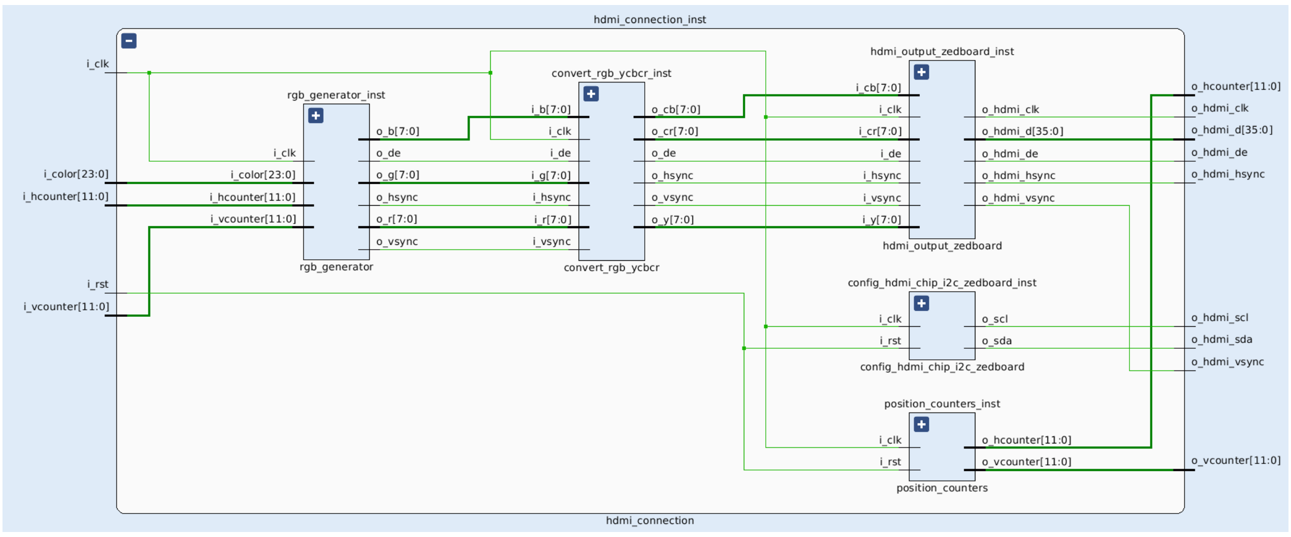


Figure 25: Schematics of the hdmi_connection component on ZedBoard.

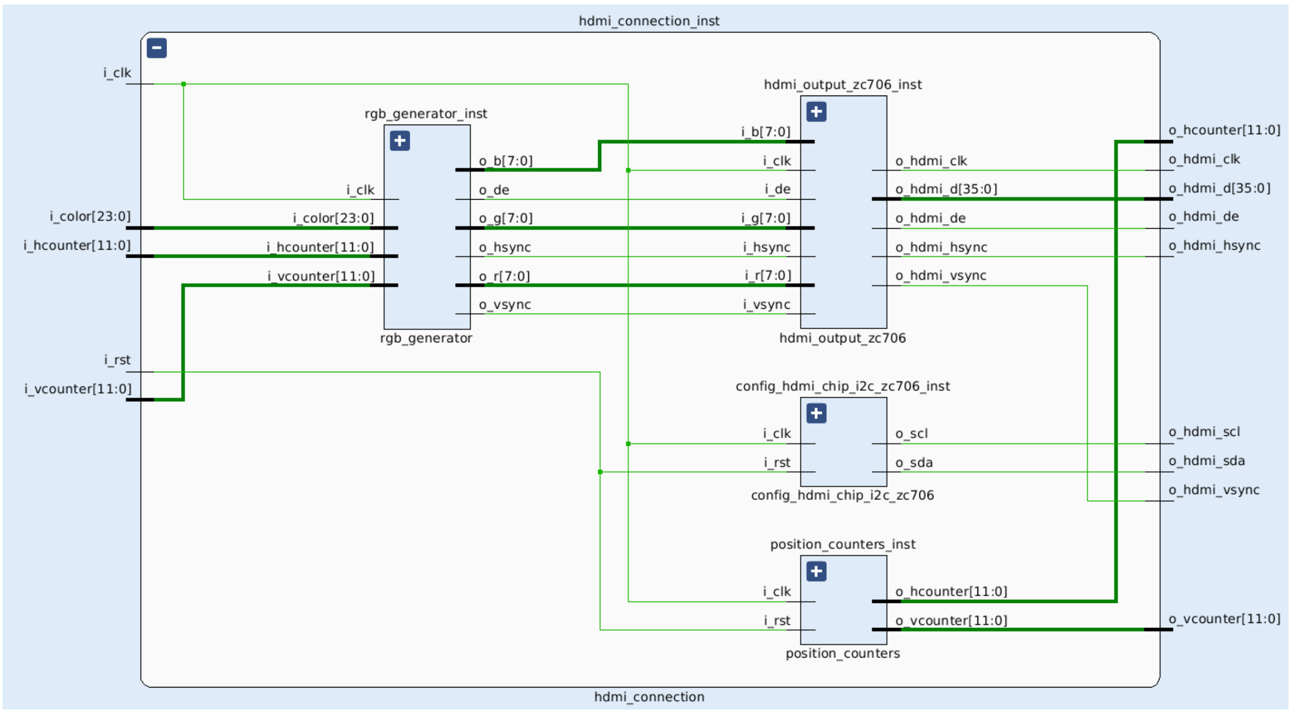


Figure 26: Schematics of the hdmi_connection component on ZC706.

The component *config_hdmi_chip_i2c* handles the I2C communication to configure the HDMI transmitter ADV7511.

Position Counters Generation

In order to generate a screen, two counters are needed: a vertical and a horizontal one to describe the whole screen including the visible area and the blank one.

Instead of having the back porches before the visible screen, I have changed the reference of the screen to start directly by the visible area. Therefore, the back porches are now at the end of the screen after the synchronization areas. This enables to have the counters between zero and the classical resolution lengths for the active pixels (for example the horizontal counter between 0 and 1919, and the vertical between 0 and 1079 for a 1920x1080 screen resolution), and to have larger values in the blank interval.

The counters start at the top left of the screen. The horizontal one is the first to increment. When the total horizontal size is reached, the vertical counter is incremented by one and the horizontal counter returns to zero: a new line begins. At the end of the screen, when both counters reach the total screen size, they return to zero: a new frame begins.

Color Generation

These two counters (outputs of the *hdmi_connection* block) are sent to a component in order to generate the colors data in the visible area. Colors are generated in the format RGB as it is the most practical format to understand. It is important that HDMI information to be sent remains consistent in terms of clock period. For example, here, as colors are generated in one clock period, the counters need to be also delayed by one clock period.

HDMI signals generation

This color data, alongside the delayed horizontal and vertical counters (that are now inputs of the *hdmi_connection* block), is sent to another component which will create the HDMI signals:

- the data enable signal that is one inside the visible area and zero elsewhere,
- the synchronization signals which will be equal to the specified polarity when the counters are in the corresponding area, and the negated value elsewhere,
- and the RGB color signals are equal to the incoming colors data inside the visible area and are set to zero elsewhere.

RGB to YCbCr conversion

Now, the color data needs to be converted from RGB to YCbCr only for ZedBoard following equations obtained in section 4.3.5. As seen above, at each clock period of the conversion, the data enable and the syncs signals need to go through a latch to keep time-consistent signals.

Actually, when the conversion is made in only one clock period, the WNS is negative, corresponding to a timing violation.

In order to fix this, the conversion is split into a two-clock-period pipeline instead. During the first clock period, the bit shifts and the first sums are done for each Y, Cb and Cr component for every component R, G and B (it corresponds to the multiplication of the RGB components by the factors), which makes nine intermediate calculations and variables. During the second clock period, the sum of the three RGB related variables is made for each YCbCr component, and finally the 8 last significant bits are gotten rid of in order to divide by 256 ($= 2^8$), and the result value is added to the constant value.

HDMI output

Now that the color format is correct (directly RGB for ZC706, and YCbCr for ZedBoard) and that the other signals are generated, these signals can be connected to the HDMI output.

For ZC706, the clock that is used for the whole screen part at 150 MHz is connected to the output pixel clock *hdmi_clk*. The generated synchronization signals are connected to *hdmi_hsync* and *hdmi_vsync*. The generated data enable signal is connected to *hdmi_de*.

Finally, for the data color signals: the R signal on 8 bits is connected to *hdmi_d(35 downto 28)*, the G signal to *hdmi_d(23 downto 16)* and B to *hdmi_d(11 downto 4)*.

For ZedBoard, *hdmi_clk*, *hdmi_hsync*, *hdmi_vsync* and *hdmi_de* signals are connected the same way: directly to the generated signals.

However, for the data color signal, it is more complicated. As the system uses 4:2:2 chroma subsampling, a ping-pong needs to be implemented to send Cb for the first pixel and Cr for the second one, then we return to Cb and so on. To implement this feature, a bit signal is used. It will be set to zero when the generated data enable signal is zero. When data enable is one, the bit signal switches at each clock period between zero and one. Thus, for the first pixel in the visible area the bit signal equals zero and then it switches. Therefore, we can connect Cb to *hdmi_d(15 downto 8)* when this bit signal equals zero, otherwise Cr is connected to the same pins. For the luma signal, Y is connected all the time to *hdmi_d(23 downto 16)*.

4.4 Results

On ZedBoard, the example project [31] has provided much help. The setup was working fine at the first attempt, and it was easy to incorporate the changes little by little. The final result of the HDMI test gives the following image on the screen (Figure 27), which is exactly what was expected. The screen displays vertical bands of width 256 pixels, colored as follows:

- in red for the first band, by setting the Red signal at the maximum value (0xFF),
- in yellow, with both Red and Green signals,
- in green, by setting the Green signal at the maximum and the other colors at zero,
- in cyan, with both Green and Blue signals,
- in blue, with only the Blue signal,
- in magenta, with Blue and Red,
- finally, the rest of the screen is painted in white with all Red, Green and Blue signals to 0xFF.

This test was working well on both screen resolutions: 1680x1050 on the first monitor used, as well as 1920x1080 (the target resolution) on the next monitor.

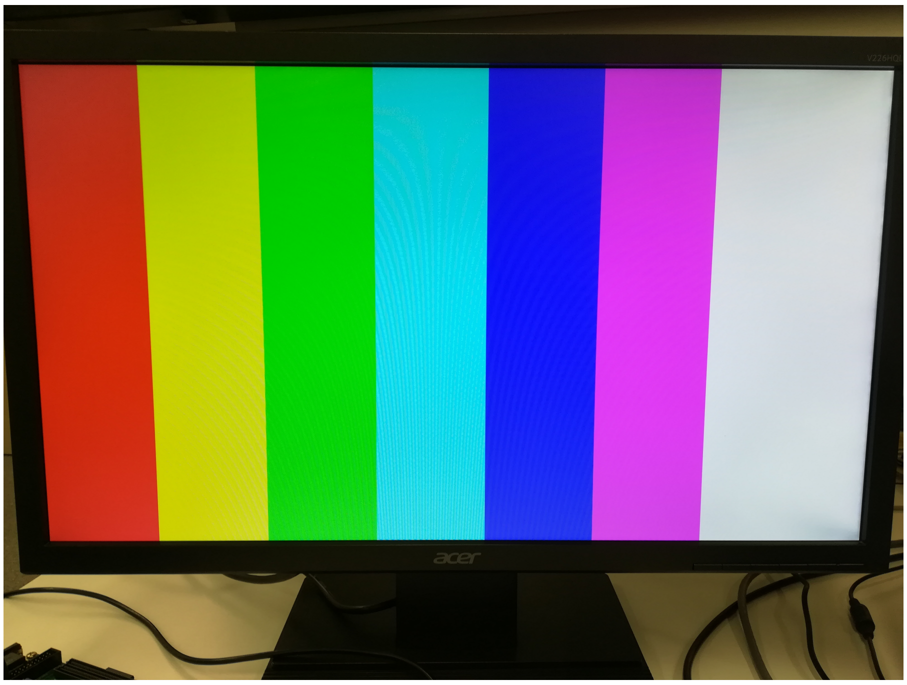


Figure 27: Result screen of the HDMI test.

With the other board ZC706, the implementation has been more complex. The code was not providing any image on the screen, which remained black. I initially thought that it was the I2C bus selector with which I had issue. Therefore, I tried to implement the I2C communication on the PS (Processing System) (in C language, using functions for the I2C already made in the libraries). Firstly, the PS only did the configuration of the I2C bus selector while the PL was still doing the configuration of the HDMI transmitter. Secondly, the PS was doing the whole I2C process with both the communication with the bus selector and ADV7511. However, the results were disappointing, with the same black screen.

In order to solve the issue, I did several research online. In the end, I found a project online using HDMI on several boards including ZC706. The part of this project that was helpful was the constraints file [36]. Indeed, it contained the pins definition and more importantly, the voltage of the corresponding banks. I noticed that they were using 2.5 V for the HDMI pins instead of the 1.8 V that I was using, set by default within the software Vivado (in the Package Pins window of the Synthesis Design). Therefore, I changed these values in my constraints file to 2.5 V.

After making this change, the display was working well, while the change of the whole I2C process on the PS was still applied. But then, when I tried to implement back my I2C on the PL, it failed.

By studying what could possibly go wrong in my I2C sending procedure, I tried to modify a feature that I had implemented differently from Mike Field's example [31]. Indeed, I had decided to fill the shift register *ack_sr* with ones instead of zeros when the I2C communication was not in use. This had been done to set the SDA signal in high impedance when I2C was not in use, to release the I2C lines for other communications. And it was actually working fine on ZedBoard.

However, on ZC706, it was causing issue. When reverting back the behavior to fill *ack_sr* with zeros, I2C communication was finally working. This change makes the SDA signal take (instead of high impedance) the value of *data_sr*, which is set to one when the I2C is not used.

With this fix on the I2C on the PL, the HDMI connection was eventually working as expected, and I got the same result onscreen as on ZedBoard (Figure 27). This HDMI test with ZC706 was only made on the 1920x1080 monitor because I was already working on the second screen, which possesses the target resolution.

Now that the HDMI connection has been established between the PL and the monitor for both boards (ZedBoard and ZC706), we can study the real-time display of the SNN (Spiking Neural Network).

5 Raster plot

5.1 Communication with HEENS

5.1.1 Principle

The raster plot is meant to show the spiking neurons with time. In the following sections, I will call HEENS all the part of the code that manages the neural network, and with which the work described in this thesis will communicate in order to generate the real-time display.

In order for the screen generation part to receive the neurons information, a FIFO memory is used. It is filled by HEENS during the execution phase, and needs to be read during the distribution phase. This FIFO contains all the neurons (actually their ID) that have produced a spike during the last execution phase. The neurons IDs are stored on 18 bits: 4 for the column, 4 for the row, 3 for the virtualization level and 7 for the chip ID.

However, all of the possible neurons (on 18 bits) are not used by HEENS. Indeed, the boards are limited in number of neurons: at maximum 5x5 rows and columns for ZedBoard, and at maximum 13x13 for ZC706 (plus the 8 levels of virtualization for both). First, I focused on ZedBoard. For this board, the maximal number of neurons is 200 ($= 5 \times 5 \times 8$) (without being connected to other boards).

Therefore, in order to represent the neurons, we only want to display these 200 values. Consequently, it is needed to convert the neuron ID on 18 bits to a value between 0 and 199. First, the 18 bits are split between the chip ID (*chip_id*), the virtualization level (*virt*), the row (*row*) and the column (*column*). Then, we follow the equation below, where the number of used column, row and virtualization levels of the PE array in the current application is used for the computation of the id value:

$$\begin{aligned} id\_value = {} & column \\ & + NB\_COLUMN \times row \\ & + NB\_ROW \times NB\_COLUMN \times virt \\ & + NB\_VIRT \times NB\_ROW \times NB\_COLUMN \times chip\_id \end{aligned} \tag{5}$$

This equation can be rewritten as follows:

$$\begin{aligned} id\_value = {} & column \\ & + NB\_COLUMN \times (row \\ & + NB\_ROW \times (virt \\ & + NB\_VIRT \times (chip\_id))) \end{aligned} \tag{6}$$

The current time is defined as the number of working phases: it starts at zero and increments at the end of the distribution phase (which represents the end of a whole working phase). This value is stored into a 32-bit register.

It was decided to display on the screen 1024 timestamps at a time (approximately one second of real-time operation), and then to shift the plot by introducing the new values at the right. This value of 1024 has been chosen because it fits within the screen width of 1920 pixels for the target resolution, as well as the 1680 pixels width of the resolution used for the first screen. In addition, 1024 is a power of two, which facilitates work.

In order to store the neurons spike information, the initial idea was to keep directly in registers the spiking neuron IDs on 18 bits plus the current time on 32 bits. However, it is required to decide the maximal size of this array of registers, so the maximal number of spikes to keep in memory, which means that when it is full, the oldest information is lost. In addition, it requires a lot of time because at each position of screen counters inside the plot, the whole memory has to be read. Besides, it is needed to check if the horizontal position corresponds to the time of the spikes. Then, it is required to do the conversion from the neuron ID on 18 bits to the value between 0 and 199. Finally, we can check if it corresponds to the vertical position.

To reduce this time, the conversion of the neuron ID can be made prior to the storage in memory and it will also reduce the memory size. However, this solution of keeping the events with the spikes and their generation time is still not satisfying.

The next idea was to keep in memory the visible screen with an array of the screen size filled by ones and zeros to decide if a neuron is spiking or not, and then if the pixel needs to be painted or not. But, a better option is to keep in memory only the useful part: only the 200 neurons and the 1024 timestamps.

Thanks to a block memory component that represents storage as BRAM (Block RAM (Random-Access Memory)), this array will be saved. And rather than shifting the data in this memory at every new working phase, a pointer is used and actually it already exists: the current time can be used. Only its 10 Least Significant Bits are used (it corresponds to $1024 = 2^{10}$ timestamps), and in that case, having a power of two as the number of timestamps to print reveals itself very practical. The column of 200 neurons will be written inside the memory at the corresponding position given by this pointer (with ones to indicate that the neuron is spiking). When the memory is full and the pointer reaches 1023, it will naturally return to 0. At that stage, we can write again in this circular memory.

As this pointer will give the position of the current time, when the memory is not filled yet, the data to be printed is between 0 and this pointer. When it is full and that we have started to re-write information in the memory, the oldest written information corresponds to the pointer plus one. Then, all the following columns can be plotted as the next timestamps until we reach the value of the pointer which represents the last written information.

5.1.2 FIFO signals

A FIFO possesses as ports (Figure 28): a clock signal that can be different for the writing and the reading, a reset signal, writing ports as well as reading ports.

The writing ports are the following: the input data to be written, a writing enable signal to set to one to write the input data inside and a full signal that is one when full.

The reading ports are the following: the output data from the FIFO, a reading enable to set to one to read this data and a empty signal that is one when empty.

In addition to these signals, there can be data counts (for the reading or writing, clocked at the corresponding clock) and a valid signal that is equal to one when, after the read enable has been set to one, the output data is valid to be read.

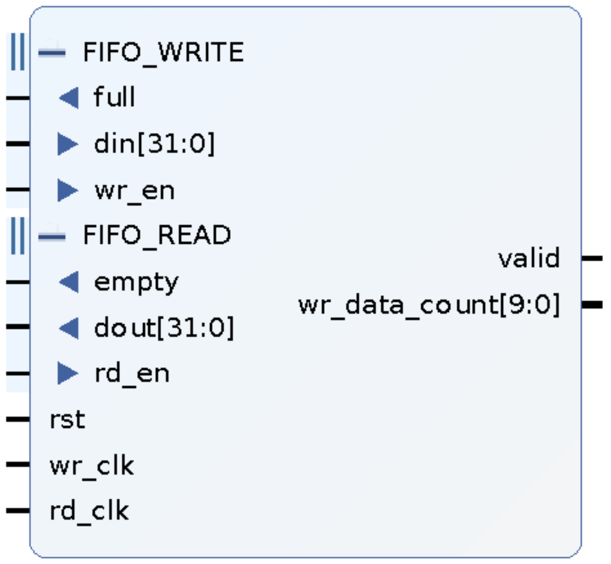


Figure 28: FIFO input/output ports of the spikes.

The PE array generates and encodes the spikes, which are written into the FIFO. Consequently for this communication, there is only to care about the reading related ports (including the valid signal). The reading clock is the HEENS clock, set at 125 MHz.

The screen part possesses a flag to tell that it is ready to read data. And then, HEENS will activate the read enable signal. This behavior has be done in order for HEENS to remain in control of the reading process, as actually this FIFO is also used in order to send this spikes information to the PC, so HEENS has to know when it can read data from this FIFO. This also explains why this information is on 32 bits and not 18 only, which is the size of the neuron ID, because additionally, the spikes time stamps are sent. Nonetheless, only the 18 Least Significant Bits will be connected to the HDMI display part.

5.1.3 Block memory signals

A block memory supports two ports (Figure 29) at two different clocks (or only one). For each, there are:

- the clock signal,
- the enable signal to activate the memory for both reading and writing,
- the writing enable to activate in addition to the previous signal for the writing,
- the address signal to specify at which position in the memory we want to read or write,
- the input data to be written in the memory at the address when both enable signals are set to one,
- and finally the output data that can be read from the memory at the address when the enable signal is one.

This output data signal equals the expected value only a few clock periods after the enable signal and address are set. This behavior has to been taken care of when trying to read from such block memory.

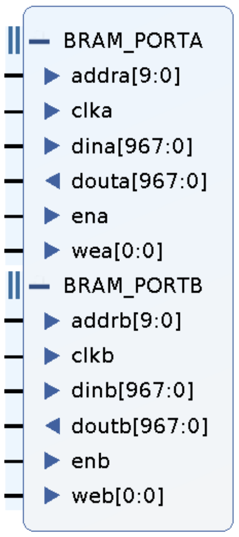


Figure 29: Block memory input/output ports of the spikes information.

This memory will store the spikes information, which is 1024 columns of neurons (values between 0 and 199), written by the FIFO reading process, which communicates with HEENS at the HEENS clock. Then, the memory will be read by the screen generator process at the pixel clock.

Actually, the size of the memory has been set to 968 and not 200 in order to have the same component for any row and column configuration. The levels of virtualization are set to 8 for the display (even though in the current software version, they are not supported yet). In case of neurons array of 12x12, the number of neurons is $12 \times 12 \times 8 = 1152$, which is a value larger that the height of the screen (1080) at the target resolution 1920x1080. Then, we limit the display to a 11x11, which makes $11 \times 11 \times 8 = 968$, hence the width of the memory.

5.1.4 Reading the FIFO and storing the information

In order to read the spikes FIFO from HEENS and to write this information into the memory, a Finite State Machine has been used (Figure 30).

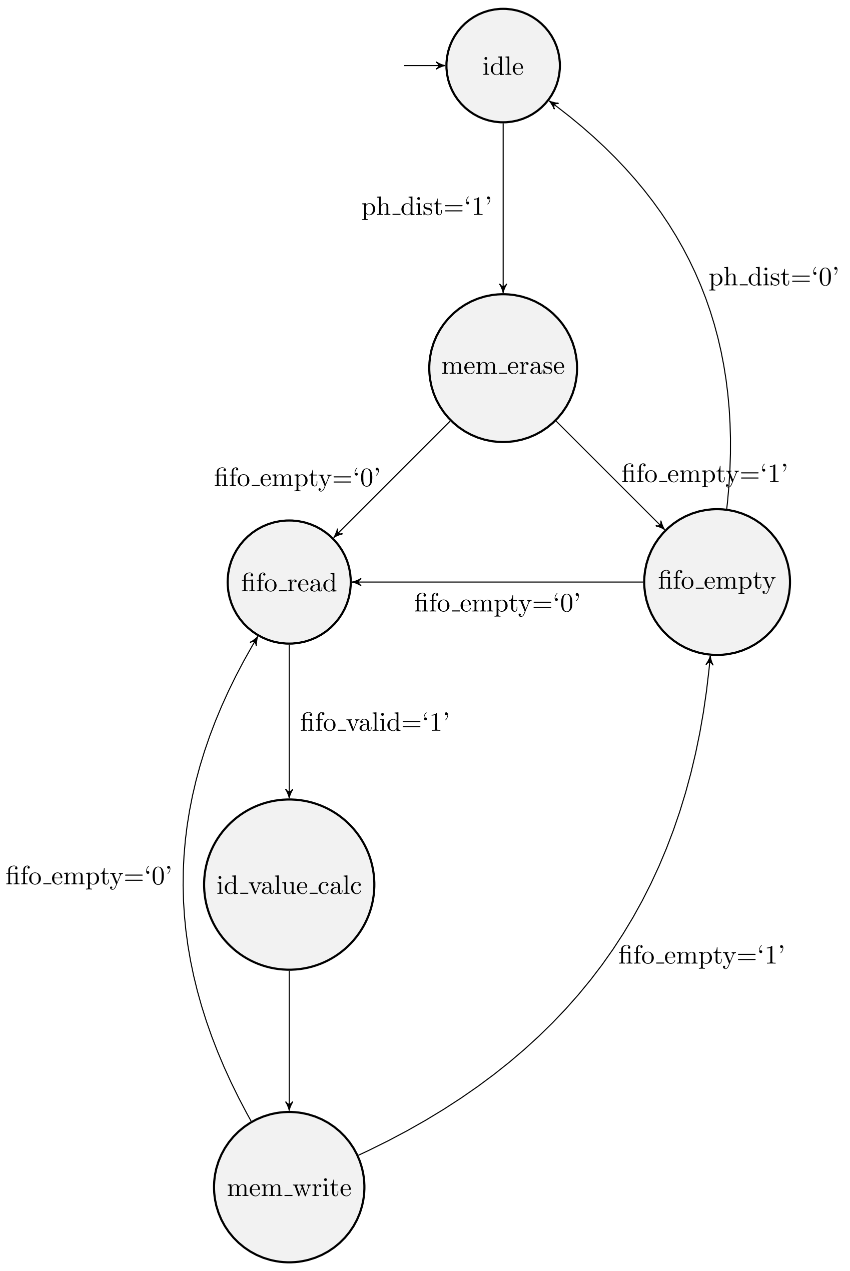


Figure 30: FSM of reading the FIFO and writing in memory the spikes information.

Idle state

The first state is where we wait when HEENS is not in distribution phase. The address at which information will be written is set when the distribution phase begins to be the current time (modulus 2^{10} to get a number between 0 and 1023).

Memory writing procedure

In order to write in the memory, the neuron ID value we get between 0 and 199 represents the position in the memory column where we need to put a one. However, as multiple spikes can occur during one execution phase, we need to add a one at the correct position while keeping the values at other positions in the column as they are. Consequently, we need to first read the memory at the defined address, second add a one at the correct position and third write the new column at the same address. Besides, for each new distribution phase, as we will only append information to the column, we need first to erase it.

Memory erasing state

Hence, this is the state where the memory is erased at the start of the distribution phase. In this state, the memory enable and write enable signals are set to one, and the input data to be written is set to a column of all zeros.

FIFO empty state

Then, the new state will be FIFO_EMPTY when the FIFO is empty. In this state, we wait to return to the first idle state at the end of distribution, or to go to the same state when the FIFO is not empty as from MEM_ERASE: the FIFO_READ state.

FIFO reading state

In the FIFO reading state, the flag to tell that the reading procedure is ready is set to one, and it waits for the valid signal. When valid is one, the output data of the FIFO is read and kept in memory, and in parallel, the memory enable signal is set to one in order to be able to read the memory later (the memory address has already been set). Indeed, we have seen that the output data signal from a memory can be read only a few clock periods after the enable signal and address are set.

ID value calculation state

The new state when the valid signal is one is the calculation state where the conversion from the vector on 18 bits to an integer from 0 and 199 is done. The memory enable signal remains at one. This state lasts one clock period.

Memory writing state

Finally, information can be written in the memory. In this one-clock-period state, the output data of the memory is read and a one is put at the position in the read column that equals the integer value gotten from the previous calculation. Then, this new data is set as the input data of the memory, while the memory enable remains one and the writing enable is set to one.

The conversion has been split in two steps: first the data is read from the FIFO, and then the calculation is done during another clock period, in order to prevent negative slack.

In addition, this conversion that includes multiplication has been made directly and not using decomposition of the factors in powers of two and bit shift operations as we have seen in the conversion from RGB to YCbCr. Indeed, the factors depend of the number of columns, rows (and virtualization levels but this is always set to 8), and they will be set to different values for different PE array configurations. Therefore, the decomposition cannot be easily done automatically.

5.1.5 Implementation of the buffer

HEENS has been cadenced for its working phase (execution plus distribution) to last one millisecond in order to have a display readable with around one second (1.024 s exactly) between the time a spike appears at the right of the screen and the time when it leaves it at the left.

This implies that each new millisecond, a new data is written in the memory and the screen needs to shift to the left introducing the new data at the right. However, with a screen refreshing at 60 frames per second (fps) (actually with the rounding made on the pixel clock at 150 MHz, 60.606 fps), a whole screen to be displayed requires 1/60 second, which is 16.67 ms. Thus, during one display between the first point of the plot and the last, memory data can have changed.

In order to have consistent data during one display, a buffer has been implemented between the FIFO and the memory. The buffer will store the data from the FIFO as the memory was doing before, and when a display is finished, the buffer data will be written into the memory. The size of this buffer memory has been set to 32 (times 968) because 16 is not sufficient as the whole display takes more than 16 milliseconds.

In order to synchronize the writing of the buffer in the actual memory, I have decided that the screen generator will raise a flag when it reaches the end of the visible screen. This way, there is enough time with the blanking interval, for the buffer to be written in memory before the next display. However, this signal will be generated at the pixel clock and it has to be read at HEENS clock, so it needs to be synchronized beforehand. For this purpose, an IP of Xilinx was used, called CDC (Clock Domain Crossing) in order to ensure the traversal between the two clock domains. [37]

To implement this buffer feature, two new states were introduced in the state machine. In FIFO_EMPTY state, when the flag of the end of the visible screen is one, the new state is WAIT_BEFORE_TRANSFER where we set the buffer enable signal to one in order to read it later and wait two clock periods. Then, the next state is TRANSFER_WRITE, where the output data from the buffer can be read and written directly in the actual memory (with the two enable signals being set to one). The state remains the same until having read and written all data from the buffer into the memory. Finally, it returns to the FIFO_EMPTY state.

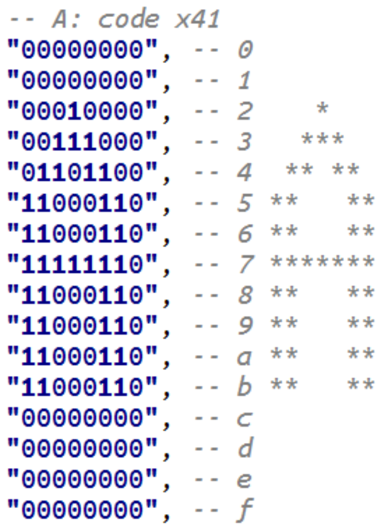
5.2 Creating the plot

5.2.1 Information to show

There is some useful information that is useful to display. So, I decided that on the top left corner of the screen we will write the name of the board (either ZedBoard or Zynq ZC706), the number of columns, rows, virtualization levels and chips we are working with, as well as the execution time.

Text font definition

In order to write anything on the screen, a font has to be defined. I found examples that displays text using VGA [38, 39], and I used the font they have defined. It is a mono-spaced font where each character is inside a rectangle of width 8 and height 16. It defines the 128 ASCII (American Standard Code for Information Interchange) characters. The information is stored in an array of 8-bit width (which corresponds to the font width) and of depth 2048 ($= 128 \times 16$, for the 16 font height of the 128 ASCII characters). The first 16 lines correspond to the first ASCII character (of code 0x00), the next 16 correspond to the second character (of code 0x01), and so on (Figure 31). I have defined this array as a constant in a new VHDL package (*character_definition_pkg*).



```
-- A: code x41
"00000000", -- 0
"00000000", -- 1
"00010000", -- 2    *
"00111000", -- 3   ***
"01101100", -- 4  ** **
"11000110", -- 5 **   **
"11000110", -- 6 **   **
"11111110", -- 7 *******
"11000110", -- 8 **   **
"11000110", -- 9 **   **
"11000110", -- a **   **
"11000110", -- b **   **
"00000000", -- c
"00000000", -- d
"00000000", -- e
"00000000", -- f
```

Figure 31: Example of character representation: the 'A' character.

Text generator

Now, to write a string of characters on the screen, a component is created. I used one of the example cited above [38] as a starting point. It uses the length of the text to display as a generic (so it will be constant for an instance of this component). Its inputs are the pixel clock, the horizontal and vertical positions of the current pixel in the screen, and the horizontal and vertical positions of the top left corner of the text to display. Its output is a Boolean to tell if the current pixel should be ON or OFF. In addition, there is another input: a Boolean to activate or not the display.

First, the horizontal and vertical positions of the current pixel are shifted so that their reference is the top left corner of the text. In order to display something, we check if this shifted position is positive in both directions, and that the vertical component is lower than the font height and the horizontal one lower than the text length times the font width. Actually, in order not to work with the signed type, these tests are made before the shift of the reference.

Then, the shifted horizontal position is used to determine to which character in the string correspond the pixel position. This is done by a right bit shift of 3 (by discarding the three Least Significant Bits) in order to divide by 8 (the font width). In addition, to get the bit column of this character in order to draw it, a modulus 8 is made (by keeping only the three Least Significant Bits).

With the character position in the string, its ASCII code can be obtained by using the *pos* attribute of the *character* type (however, it only works if the string is a constant). Then, to get the corresponding row in the font array, this ASCII code is multiplied by 16 (the font height) and added to the shifted vertical position, which is the bit row of the character. This is done by concatenating the ASCII code and the last 4 bits of the shifted vertical position.

A process is then used to read from the array (which will be stored in ROM) the row (of 8 bits, with 8 being the font width) of the character corresponding to the vertical position.

Finally, another clocked process is used to check if there is a one at the bit column of this row in order to tell if the pixel should be ON.

Rotated text generator

Another version of this component has been made to write rotated text for vertical axis labels. The roles of the horizontal and vertical positions are inverted. Besides, in order to get the character in the string, we have to count from the end as the left of the string will be at the bottom and not top. The same applies when we have to check if there is a one in the character bit row at the right column, because the left of this character will be at the bottom.

Integer text generator

A third version has been made in order to write the time, because the strings to show will not be constant anymore. This component receives the number of digits of the input integer instead of the text length, and the string has been replaced by a digit array.

The process is similar to the classical text generator, it is only to get the ASCII code of the current digit that we use the zero ASCII code (0x30) and add to it the value of this digit. Actually, the digits array allows also in addition to numbers between 0 and 9, the value -1, which signifies not to display this character. It corresponds to the leading zeros of the integer, in order not to display "06" for example but only " 6" (with a leading space).

Writing time

The time to be written on the screen corresponds to the time that increments at the end of each distribution phase. The system was cadenced to have each end of distribution phase every millisecond. Then, with a counter of 1000, we can get seconds and thus display a classical time with days, hours, minutes and seconds. For the day, I have decided to make it range from 0 to 63 because it is a power of two, and it is with two digits in order to keep the same behavior as for the other time units.

These counters (except for the milliseconds one that will not be displayed) will not be integers but digits arrays that we have defined before. They will be initialized to [-1, 0], as they are all on two digits (and with -1 for the leading zero as said before).

At each change in the current time value, the milliseconds counter increments, it ranges from 0 to 999. Then, when it reaches 999, it returns to 0 and the seconds counter starts to increment. The digit at position 0 (with 0 being at the right) increments first, and when it reaches 9, the digit at position 1 increments (and if it is equal to -1, it goes to 1 directly). When the seconds counter reaches [5, 9], it goes down to [-1, 0] and the process continues with the minutes and so on.

I have decided to write the current time aligned to the left (with "Execution Time: " written before), with the fact that when leading time units are zero, they are not shown. For example, when the day, hour and minute are equal to zero, only the seconds will be shown and on the left. Then, when time has exceeded 60 seconds, the minute counter will appear at the left, and the seconds will appear at the right of the minutes even when they are equal to zero. For example, at 1 hour and 3 seconds, the 0 minute will be shown.

In order to do so, arrays of horizontal positions have been created. The one for the days will be of length one because either days do not appear or they do so totally at the left. The one for the hours will be of length two because when they appear it is either totally at the left or they are the days counter before. In that case, the hours horizontal position is shifted by 2 for the day counter, plus 1 for the "d" that will be written for day, plus 1 for the space after, so by 4. Then, the array for the minutes will be of length 3, and the one for the seconds of length 4.

Now, there is to design the pointers to the value to use in these arrays. For the days, there is none because it is only of length 1. For the hours, when the days counter is zero (actually [-1, 0]), the pointer is set to 0 (there is nothing before), and when the days counter is different from zero, the pointer equals 1 (there is one value before). For the minutes, when both days and hours counters are zeros, it is 0, when the days counter is zero but the hours is not, it is 1, and if both are not zero, it is 2. And we follow the same logic for the seconds.

To know if a time unit value should appear or not, we can simply check if the next time unit has its pointer which is different from 0, then it has to be written. Besides, the seconds are always shown.

5.2.2 Reading the memory and creating the plot

To be more visible, the dots on the screen will be plus sign and not just a pixel. These plus signs are formed with one pixel at the center, top, bottom, left and right. Therefore, when reading the memory, we need to know the current column but also the previous one and the next one. The previous column corresponds to the time stamp before, and permits to know if the current position is the right of a plus sign. The next column permits to know if it corresponds to the left of a plus sign. For the top and bottom pixels, the current column is what is needed.

In order to get the current neuron (which represents the vertical position), an integer signal is created. It is initialized at the maximal value (199) when the vertical counter reaches the top of the plot. Then, it is decremented at each new line. Indeed, the vertical position is defined with the top left of the screen as a reference, but a plot has its reference at the bottom left.

Three clock periods are needed between the time the memory address and the enable signal are set and the time the output data is correct and can be read.

Thus, four clock periods before the plot left limit, the address is set to the oldest value in the memory and the read enable signal is set to one. This oldest value is zero when the memory is not full yet, and the current time value plus one otherwise. Then, at each clock period, the address is incremented until it reaches the current time value.

One clock period before the plot left limit, the output data is stored in the current memory column, and the previous column is set to zero. Then, at each clock period, the current column will be set to the output data, the previous column to the current value and the next column is directly the output data of the memory.

With the buffer introduced and to have a consistent plot in the end, time-related signals should only be updated once every display. These signals are used as a reference to know what data from the memory represents the oldest and the last values. That is why, in the screen generation component, the current time value is only updated when we receive the confirmation from the FIFO reading process (through another CDC in the direction from the HEENS clock to the pixel clock) that the buffer has been transferred into the memory, which means that new data are available to read and display.

Now, the plus sign can be drawn by checking for the current column that at the current neuron position there is a one (for the center point), or that at the neuron position plus or minus one there is a one (for the bottom and top points), or that at the current neuron position but in the previous column or the next column (which is directly the output data from the memory), there is a one. And if one of these assumptions is true, then the pixel should be colored.

In addition to the actual dots, in order to draw a plot, I have added plot contours, with ticks along the axes, with label on them and title for the axes. For the vertical axis, the rotated text generator was used. Otherwise, it was the classical text generator (even for the tick labels, which are integers, because they are constant and the *pos* attribute works fine then).

5.2.3 Extended plot

Actually, with only 200 pixels, the plot is not even occupying a fourth of the screen vertically. Therefore, I decided to extend the plot using a button as a switch from the normal plot to the extended one. The extension factor is 4, which is practical because first, it still fits the screen even with the labels around the plot, and then it is a power of two so it is easier to work with when using bits signals. This also allows, because it is greater than 3, to have the top and bottom points of the plus sign to be painted only for the actual neuron position, and not carrying over the other vertical positions.

In order to implement this extension, another component was created and a button on the board will be used to switch between the two configuration and determine which of these two components will have control on the communication with the memory, as well as the actual painting of the pixels.

The middle button on both ZedBoard and ZC706 is used for this feature. The left button is already used for the reset. And the right one will be used on ZC706 by HEENS (in the multi-board configuration-). ZedBoard possesses 5 buttons (top, left, middle, right and bottom) and ZC706 3 (left, middle and right). Then, to have a consistent behavior, only 3 buttons can be used. In the end, only the middle one was available.

In order to avoid the bouncing when reading the button input, the clock divider component is used with a clock divider of 2^{21} on the pixel clock, so up to a clock period of 14 ms. With this slower clock, the input is only read once in a while and rebounds are not seen at this rate. Then, to convert the behavior of this button to that of a switch, a flip-flop is used to alternate between a zero and a one each time a rising edge is detected in the button signal, which is each time the button is pressed.

Now, a extended vertical counter is used along with an intermediate counter on 2 bits to count up to 4. This extended counter is initialized at zero when the real vertical counter is zero. When its value is outside of the range of the previous smaller plot, it increases normally. However, inside this range, the intermediate counter increments and when it is 3, it returns to zero and the extended vertical counter increases by one. Also in this range, when the intermediate counter is 3, the neuron value that represents the position (which was already set to the maximal neuron value 199) will decrement (so only once every four vertical pixels).

Thus, the same values of plot limits as the standard plot can be used. Moreover, the same components to draw the contours can be used using this extended vertical counter, except for the labels (for which a new component has been created). Indeed, for the text generation, the vertical counter needs to update with each line, otherwise the text will also extend.

Additionally, not to extend the ticks along the vertical axis, a Boolean signal has been used in the contours component to know when the intermediate counter is equal to 2, which will be the center of the plus signs. And for the standard plot component, this Boolean will always be set to True.

To paint the dots, the same previous, current and next memory columns system is used as in the not-extended plot. However, now, for the top, center and bottom points, we check that the current column has a one at the current neuron position and that the intermediate counter is either 1, 2 or 3. For the left and right points, while reading the value of the next and previous columns, we have to check also that the intermediate counter equals 2.

5.2.4 Created packages

In order to define the constants that will be used and to facilitate their utilization by different components, a few packages have been created.

First, the package that contains the HDMI resolution information (*hdmi_resolution_pkg*) is still used for the HDMI communication part.

Second, a package (*neurons_pkg*) has been created in order to store the constants that will be used for the neurons information. There is also the conversion function from the neuron ID on 18 bits to an integer between 0 and the maximum neuron value. In order to do this conversion and also calculate this maximum value, the maximal numbers of virtualization levels, of rows and of columns are defined first. In addition, this file also contains the ranges of the two raster plot: for the small plot not extended, it is the minimum between 200 and the number of neuron IDs; for the extended plot, this range is the minimum between 968 and the number IDs. If this range of the extended plot is lower than 200, the extended plot will have the space between two neurons of 4 instead of 1, with the same neuron range as the small plot. Otherwise, the extended plot displays all the neurons up to a maximum of 968 (with the space between two neurons of 1).

Third, another package (*character_definition_pkg*) has been created to store the information for the text generator. It contains the definition of the digits array type (with digits between 0 and 9, and the value -1). This file also defines the size of the characters (8 pixels in width and 16 in height). Finally, it contains the table of all ASCII characters in order to display them onscreen.

Fourth, a package (*plot_pkg*) that contains the information relative to the creation of the plots has been created. It contains the definition of the colors, as well as values to position the plots.

Finally, two other packages have been created that will only define the names of the concerned board in order to write them on the screen. The two packages will possess the same name (*neurons_platform_spec_pkg*) in order for the other components to be identical for any board and include the same package. Then, each of the two Vivado projects will only include their corresponding package.

5.3 Results with emulated neural network

In order to check and debug the code, simulations were very helpful especially to understand the timings of the IPs: FIFO and memory block, specifically the clock periods there are to wait before reading a correct value from both components.

As it is a visible result which is created, the actual tests onscreen were very helpful to check the good behavior of the dots of the plot, as well as the contours and the text generation (for example for the rotated version, in order to write the text properly).

Additionally, in the standard version of the text generator, the last column of the characters was not written. The issue was that the horizontal counter updates every clock period and there is no latch is the text generation component. The test that the character column from the ROM font array possesses a one at the right position, is made in a process, introducing one clock delay. However, the extraction of the right positions is made combinational. Thus, there was a mismatch in the timings. To solve this problem, simply a minus one is made on the current column position in the character signal.

In addition, to decrease the time of the experimental verification and reduce the synthesis, implementation and creation of the bitstream, an example project has been created. It emulates HEENS behavior by managing and writing in the spikes FIFO, and by creating the phases (initialization, configuration, execution and distribution).

In order to write into the spikes FIFO, a memory block was used with pre-charged data. To create this data, an Excel file was used to generate a file where every neuron from 0 to 199 will be written under the neuron ID form on 18 bits. This simulates a process where the first neuron is spiking at the first instant of time, then the second neuron at the second instant and so on, up to the neuron number 199 at the 200th instant. After these 200 time instants, the memory is filled with zero for the rest of the 1024 time stamps.

Then, in the HEENS simulator component, this memory is read in its execution phase, one element at each phase, and written in the FIFO. In the distribution phase, the reading of the FIFO is managed, and when it receives the ready flag from the FIFO reading process, it activates during one clock period the read enable signal.

5x5 configuration on ZedBoard

With the 5x5 array, thus with 200 neurons, the results were as shown below with the designed simulator of HEENS, in the standard configuration with the plot not extended (Figures 32, 33 and 34).

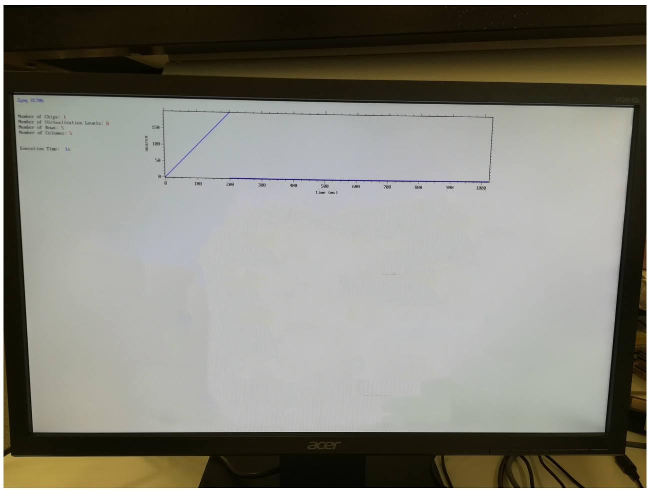
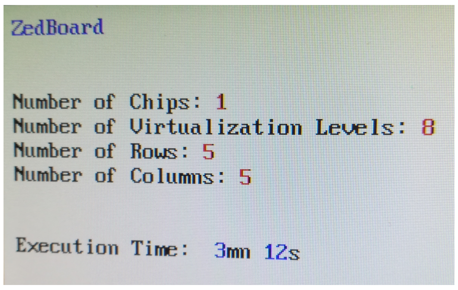


Figure 32: Result screen of the raster plot implementation for ZedBoard with 5x5 array.

The information is written on the top left of the screen. Here, the example is with the 5x5 array on ZedBoard. And the raster plot is placed on the top of the screen in the middle.



```
ZedBoard

Number of Chips: 1
Number of Virtualization Levels: 8
Number of Rows: 5
Number of Columns: 5

Execution Time:  3mn 12s
```

Figure 33: Result screen of the raster plot implementation for ZedBoard with 5x5 array (zoomed on the top left information).

The name of the board, the number of columns, rows, virtualization levels and chips are written, as well as the execution time that increases at the correct pace to follow the real time. Different colors are used to highlight the values.

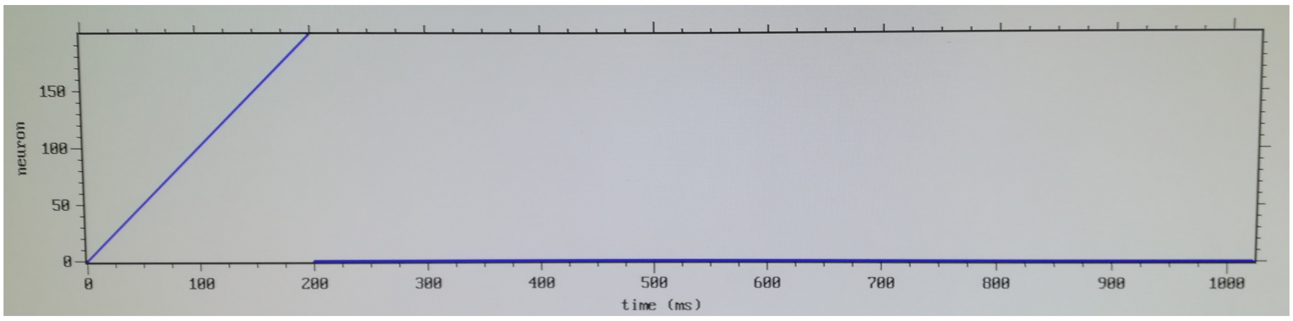
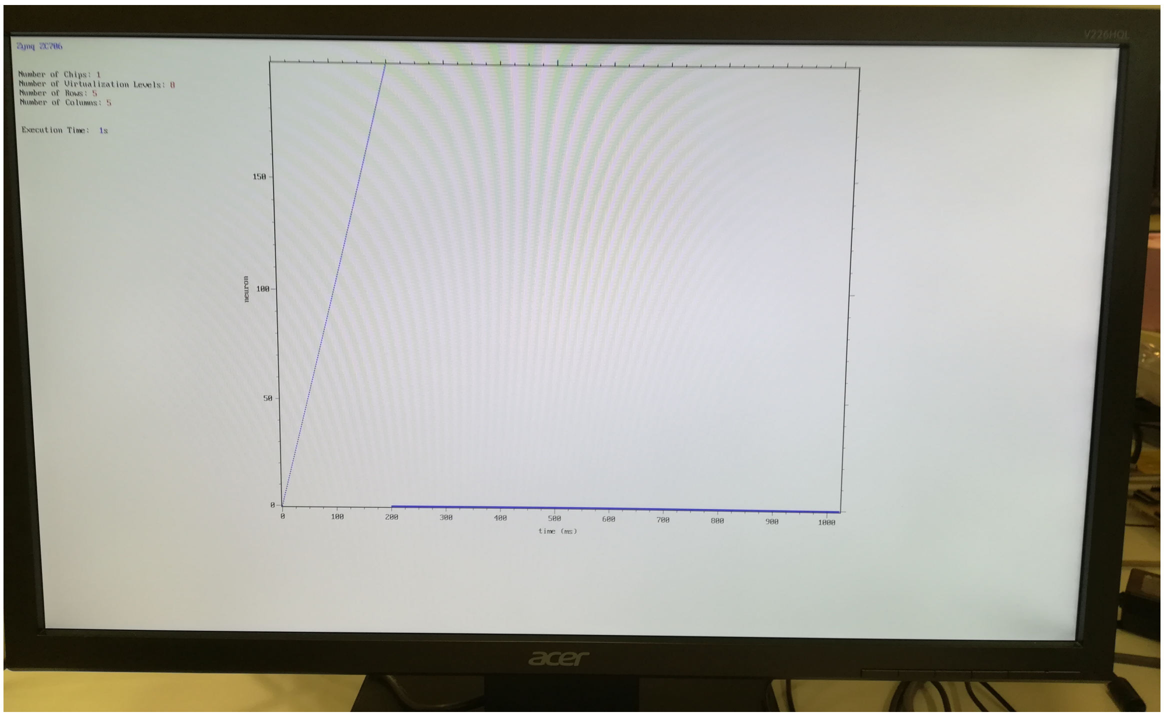


Figure 34: Result screen of the raster plot implementation for ZedBoard with 5x5 array (zoomed on the plot).

The plot appears correctly with the curve in blue and the contours in black. The vertical axis is labeled "neurons" with ticks every 10 neurons, bigger ones every 50 and biggest ones every 100. Tick labels are written every 50 neurons. Here, the axis ranges from 0 to 199. The horizontal axis is labeled "time (ms)" as every time stamp has been designed to last one millisecond. Ticks are every 25 ms, bigger every 100 and biggest every 500. Tick labels are written every 100 ms. This axis ranges from 0 to 1023.

The curve corresponds to what was expected: during the first 200 time stamps, the neurons are spiking one after the other, and then the example memory was filled with zeros. So zeros were written in the FIFO and then in the spikes memory, which explains the line from time 200 to 1023 where the neuron 0 is spiking.

As we can see with 200 neurons, the screen feels a bit empty (Figure 32), hence the idea of the extended plot, which gives the following results (Figure 35), with the same configuration of neurons, just pressing the middle button from the previous state.



```
Zynq ZC706

Number of Chips: 1
Number of Virtualization Levels: 0
Number of Rows: 5
Number of Columns: 5

Execution Time:  1s
```

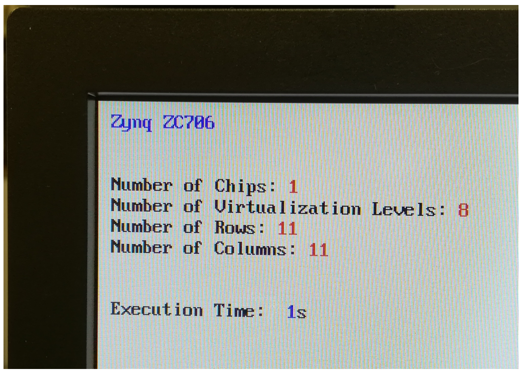
Figure 35: Result screen of the raster plot implementation for ZedBoard with 5x5 array with the extended plot.

Each neuron are now every 4 pixels on the vertical axis, and the plot occupies a larger part of the screen vertically.

11x11 configuration on ZC706

Then, the behavior has been tested with other configurations than 5x5 and on the other board. For a number of neurons bigger than 200, the extended plot cannot be done. Thus, it has been decided in that case to keep only the first 200 neurons on the first switch configuration because another plot will be drawn at the bottom, and on the second configuration, instead of the extended plot multiplied by 4, the total plot will be shown (up to 968 neurons).

With the 11x11 configuration on ZC706, the results are shown in Figures 36 and 37 in the first configuration with 200 neurons. The same example memory as for the 5x5 was used in the HEENS simulator component.



```
Zynq ZC706

Number of Chips: 1
Number of Virtualization Levels: 8
Number of Rows: 11
Number of Columns: 11

Execution Time:  1s
```

Figure 36: Result screen of the raster plot implementation for ZC706 with 11x11 array (zoomed on the top left information).

The corresponding information is correctly shown.

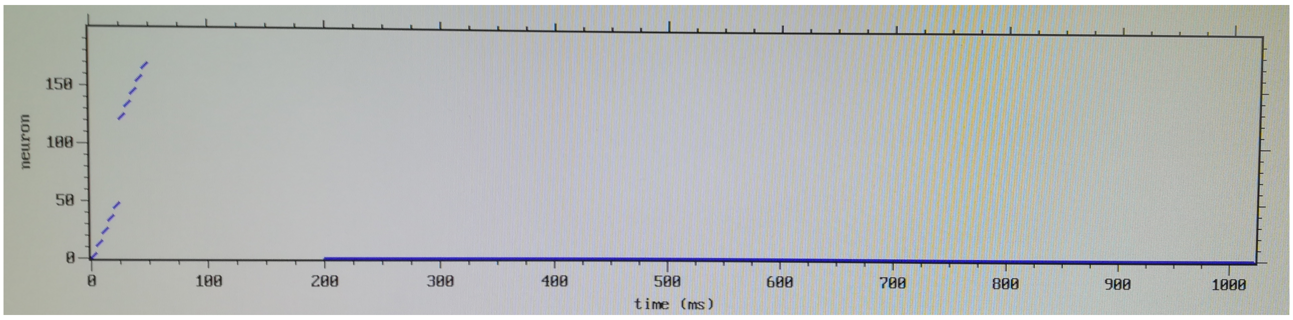


Figure 37: Result screen of the raster plot implementation for ZC706 with 11x11 array (zoomed on the plot).

The plot ranges only from 0 to 199 in this configuration while the neurons values are up to 967 ($= 11 \times 11 \times 8 - 1$).

With the extended configuration (by pressing the button), the results of the full raster plot are the following (Figure 38).

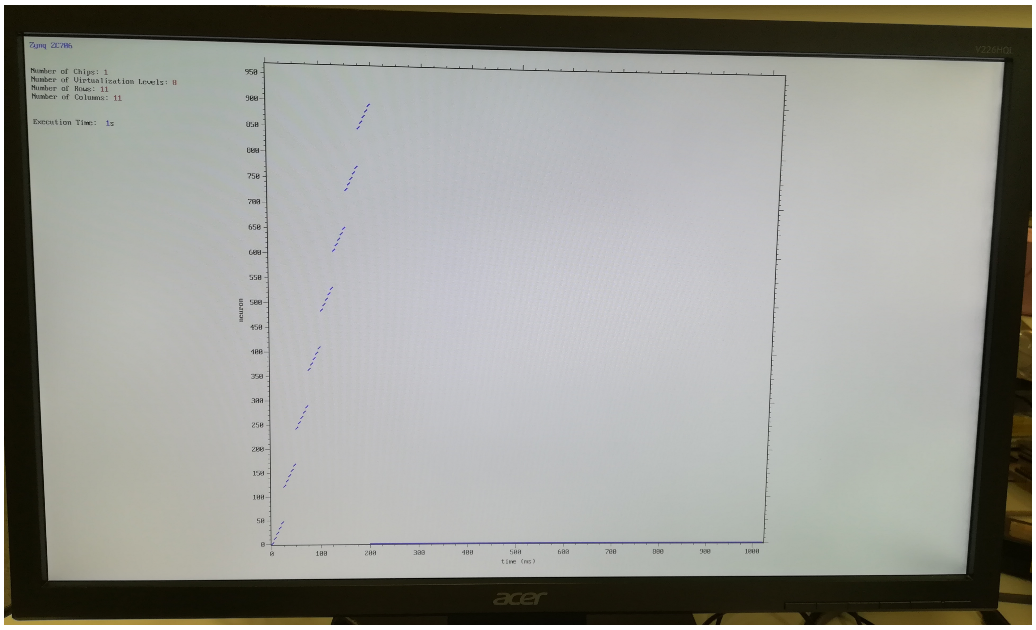

```
Zynq ZC706

Number of Chips: 1
Number of Virtualization Levels: 8
Number of Rows: 11
Number of Columns: 11

Execution Time:  1s
```

Figure 38: Result screen of the raster plot implementation for ZC706 with 11x11 array with the extended plot.

The plot now ranges correctly from 0 to 967. As the memory was designed for the 5x5 configuration, we will have on each line only neurons from 0 to 4 spiking and not from 0 to 10, and this will apply for the first 5 lines (and not 11). This explains the obtained plot: for 5 pixels we have the spiking neuron number incrementing, then a vertical jump of 6 pixels, and all this 5 times (for 5 lines), and then a larger vertical jump (of 66 pixels, for the 6 left line of 11 columns), and then it repeats for the next virtualization level, and so on for a total of 8 levels of virtualization.

The results for two other configurations (2x2 and 6x6) can be found in Appendix (Raster plot results for 2x2 and 6x6 configurations).

The next part is dedicated to the plot of neural parameters that will be drawn in the first switch configuration under the raster plot on (maximum) 200 neurons.

6 Plot of neural parameters

6.1 Communication with HEENS

6.1.1 Principle

In many cases, it is interesting to watch, besides the spikes, internal neural parameters. One of the most important is the membrane potential. These parameters are not binary anymore, but they are multivalued, thus they should be displayed as analog values.

The new plot is meant to show the membrane potential value of 4 chosen neurons as a function of the time.

This plot will be seen alongside the raster plot (even though because the screen is not infinite, a maximal value of neurons to plot for the raster plot had to be decided: 200) and it will occupy the bottom part of the screen.

For the communication between HEENS and the screen generation part, a FIFO memory is used, and the whole communication system is very similar to the one for the raster plot. However, the values inside the FIFO will actually be signed values on 16 bits (with a range from -32768 to 32767).

At present, the membrane potential only varies from $-80\,\mathrm{mV}$ to $-30\,\mathrm{mV}$, which are represented by signed integers between -8000 and -3000, the precision being ten microvolts.

In order that everything (the raster plot of maximum 200 neurons and the four plots to be created for every selected neurons, as well as the plot contours) fits in the screen, I decided that each of the four plots will have a height of 180 pixels.

Therefore, the values between -8000 and -3000 have to be converted to values between 0 and 179, which is done by the following equation:

$$\begin{aligned} plot\_value &= \frac{signed\_value + 8000}{(-3000) - (-8000)} \times (179 - 0) \\ &= \frac{signed\_value + 8000}{5000/179} \\ &= \frac{signed\_value + 8000}{27.932} \end{aligned} \tag{7}$$

Once again, the division will be transformed by multiplying by a power of two: $2^{16} = 65536$, here the value is larger than in the conversion from RGB to YCbCr in order to increase the precision in the division.

$$plot\_value \approx \frac{signed\_value + 8000}{65536} \times 2347 \tag{8}$$

Here, the multiplication will not be transformed into bit shift operations because the range of values between -8000 and -3000 may change and the transformation to bit shift cannot simply be automatized in VHDL.

Then, values are saturated in the range between 0 and 179, to be sure that they will appear correctly on the screen.

The system to read the FIFO is quite the same as for the raster plot. However, in order to write the information in the memory, as there are 4 neurons for which we study the membrane potential, the FIFO needs to be read 4 times, and each time the value will be stored. At the fourth time, these four values are concatenated into the data to write inside the memory.

The value, as it ranges from 0 to 179, will be stored on 8 bits (which goes up to 255). And then, the memory will be of depth 1024 and of width 32 for the 4 values on 8 bits (for each of the 4 neurons).

All these constants for the conversion from the analog values on 18 bits into an integer between 0 and 179 are stored in the package *neurons_pkg*. This package contains the starting range between -8000 and -3000 as well as the target range between 0 and 179. It defines the number of neurons to display, which is 4. It also determines the width of the memory in order to store these values, which is 32. Finally, it calculates the constants used for the conversion: the value to add (8000), the number of bits for the division precision (16), as well as the multiplier (2347).

6.1.2 Reading the FIFO and storing the information

In order to read the membrane potential FIFO from HEENS and then to write this information into the memory, a Finite State Machine has been used (Figure 39) which is a bit different than the one used for the spikes FIFO reading mechanism.

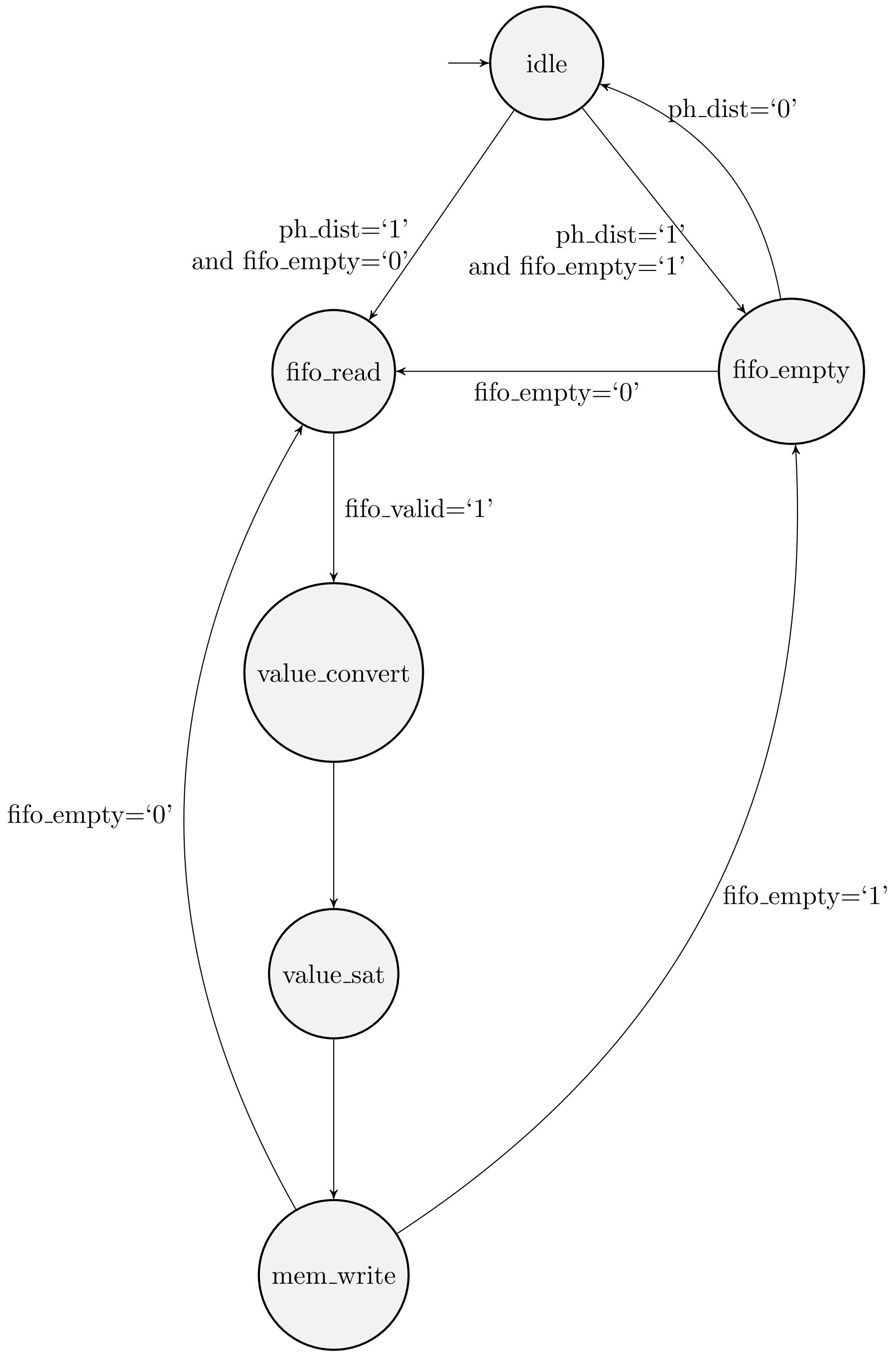


Figure 39: FSM of reading the FIFO and writing in memory the potential information.

As there is only one value for each time instant, we will write directly in memory and not append. Therefore, there is no need to erase the memory as a first state.

Besides, another state had to be created as the conversion operations were taking too much time, and WNS was negative. Consequently, the operations are split in 2: the conversion operation *per se* is made during one clock period (and one state), and the saturation of the value between 0 and 179 in another.

In addition, the MEM_WRITE state has changed. Indeed, we need to read the FIFO 4 times for the values of the 4 neurons and each of the three first times, the value is stored in a register (of length $3 \times 8 = 24$), and at the fourth time, the obtained value from the FIFO (that corresponds to the fourth neuron) is concatenated with the register and all the data is written in the memory.

6.2 Creating the plots

Four plots need to be done, each above another. The contours components have changed compared to the ones of the raster plot. Here, the horizontal ticks are only at the bottom and the top of all of the four charts. However, vertical ticks have to be drawn for each of the four plots.

The horizontal axis represents the time as well as in the raster plot. This way, the different curves evolve at the same pace. The scale remains the same as the one of the raster plot for this horizontal axis. Then, the same axis label will be used, and the tick labels will be displayed on the top of the membrane potential charts.

Actually, in order to use the same components to draw the ticks and the black contours as the raster plot, a new input has been created for these components: a Boolean to tell when the vertical counter is between the plots. Thereby, we can draw the border between two consecutive plots of two pixels to paint in black. Moreover, in order to draw the ticks along the vertical axis for the four plots, the counters need to be reset between two plots to be used for the next plot.

However, the component to draw text for the axes label and the tick labels has to be different. As the plots are somewhat small, only two vertical tick labels will be written: for $-80\,\mathrm{mV}$ and $-55\,\mathrm{mV}$. In addition, for $-55\,\mathrm{mV}$, as it corresponds to the threshold potential, the word "threshold" will be written at the right, and a dotted line will be drawn. For the dotted line, we can simply use the horizontal counter, take one of the LSBs and check if it is zero or one. If we consider the LSB, it will change between zero and one for every pixel. If we take the second LSB, it will change every two pixels, and so on. I chose to check for the third Least Significant Bit, which makes dashes of length 4 ($= 2^{3-1}$) as a dotted line.

In addition, to differentiate each plot even more, the color of the curve and of the tick labels will be different for each plot. This has been done by, instead of having a Boolean to tell if the pixels need to be ON or OFF, having an array of Booleans of length 4 (for each of the four plot) to tell with which color the pixels have to be painted.

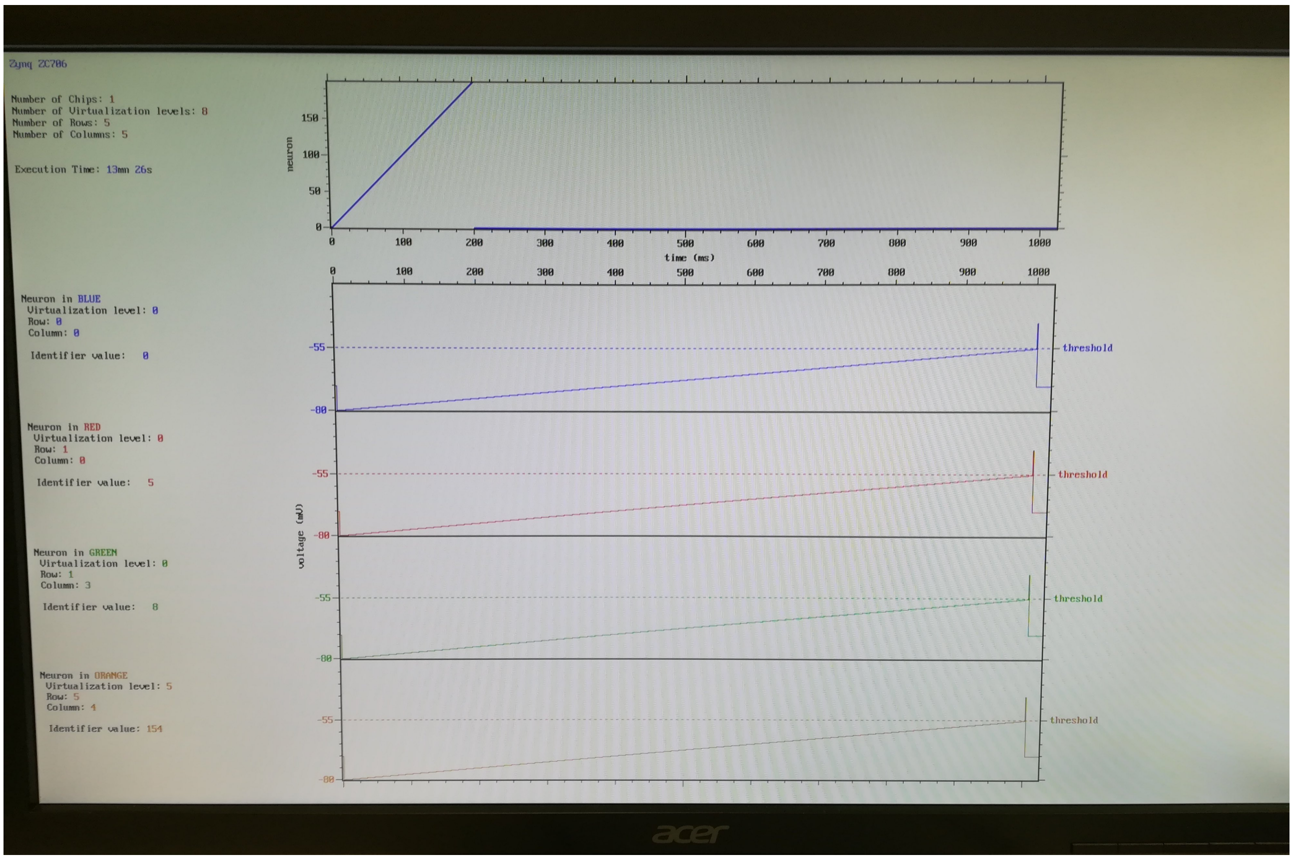
As it is an analog curve that is plot, it is made continuous. To do so, when the position counters are inside one plot, the last voltage value is stored in a register. And instead of just checking if the current pixel corresponds to the current voltage value, we check if the current pixel is in the range between the last voltage value and the current one. Thereby, we draw vertical lines to connect the dots. Besides, dots are only one pixel and not plus sign as in the raster plot.

Apart from the actual charts, information relative to each selected neuron will be written on the screen. This information enables to identify which neurons are monitored. It will include the virtualization level, the row and the column. In addition, the neuron identifier value will be written. It corresponds to the value used as the vertical coordinate in the raster plot.

6.3 Results with emulated neural network

An example was made using the HEENS simulator that now reads another memory in order to write it in another FIFO. The memory has been created with Excel and it starts by the minimum value of −80 mV and is incremented each time by 0.025 mV. This last value permits that the potential is increased by 25 mV after 1000 time stamps and reaches −55 mV: the threshold value. Then, it increases by 10 mV directly to simulate the firing, and it returns to the resting potential −70 mV. These values are simply duplicate three times in order to get the 4 example values for the 4 selected neurons.

With the example project, the results are the following (in the standard configuration of the switch, else only the extended raster plot appears) (Figures 40 and 41).

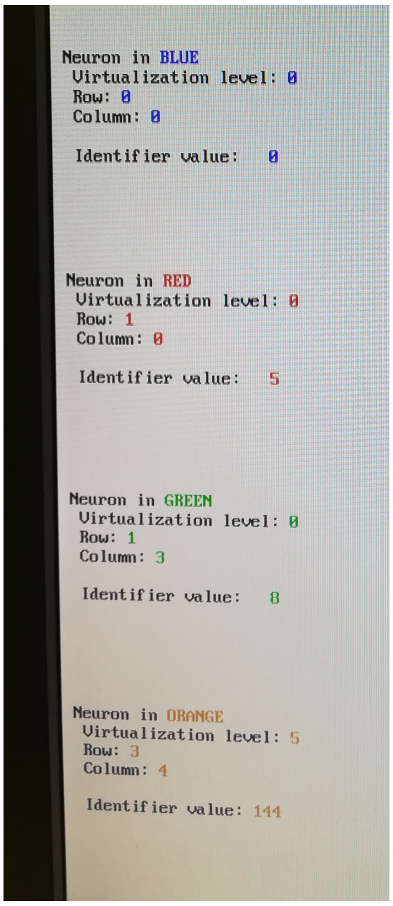


```
Zynq ZC706

Number of Chips: 1
Number of Virtualization levels: 8
Number of Rows: 5
Number of Columns: 5

Execution Time: 13mn 26s

Neuron in BLUE
 Virtualization level: 0
 Row: 0
 Column: 0

 Identifier value:   0

Neuron in RED
 Virtualization level: 0
 Row: 1
 Column: 0

 Identifier value:   5

Neuron in GREEN
 Virtualization level: 0
 Row: 1
 Column: 3

 Identifier value:   8

Neuron in ORANGE
 Virtualization level: 5
 Row: 5
 Column: 4

 Identifier value: 154
```

Figure 40: Result screen of the membrane potential plot implementation for ZedBoard with 5x5 array.

We can see the disposition of the whole screen with the raster plot at the top with the 200 neurons, and below the 4 plots of the membrane potential of 4 selected neurons, one above the other. In addition, we can see the global information at the top left corner of the screen, and the information of each selected neurons at the left of its corresponding plot.

We can see that each neuron plot is well represented with different colors: the top one in blue, the second one in red, the third one in green and the fourth one in orange. All plots are identical. The dotted line appears well, and with the example, we see that when the voltage crosses this threshold at time stamp 1000, it increases by 10 mV before returning back to the resting potential.



```
Neuron in BLUE
 Virtualization level: 0
 Row: 0
 Column: 0

 Identifier value:   0

Neuron in RED
 Virtualization level: 0
 Row: 1
 Column: 0

 Identifier value:   5

Neuron in GREEN
 Virtualization level: 0
 Row: 1
 Column: 3

 Identifier value:   8

Neuron in ORANGE
 Virtualization level: 5
 Row: 3
 Column: 4

 Identifier value: 144
```

Figure 41: Result screen of the membrane potential plot implementation for ZedBoard with 5x5 array (zoomed at the left of the plots).

We can see that the monitored neuron information is displayed on the left on each plot. In order to check the behavior, I have chosen some random neurons (numbers 0, 5, 8 and 144) as an example for the signal which selects the neurons to monitor. As information, we have the virtualization level, row and column, as well as the identifier value (the value used for the raster plot). This last value is calculated as a function of the PE array size (here 5x5).

7 Experimental results

7.1 Principle

In this section, the experimental results with the actual HEENS architecture will be seen.

The size of the PE arrays is defined in the VHDL package *SNN_pkg* of HEENS.

In order to implement HEENS in the target, once the bitstream has been generated by Vivado, it can be exported alongside the hardware. These files can then be used in the SDK (Software Development Kit) from Xilinx. The executable can be run on the Processing System, once the FPGA fabric (the Programmable Logic) has already been programmed.

There are some useful registers for HEENS Control Interface (HCI) that have to be written with the PS, such as the HTR (HEENS Transfer Register) and HMNR (HEENS Monitored Neurons Register). The operations regarding these registers can be found in Appendix (HEENS Control Interface registers).

Then, it can already be seen on the monitor screen the frame of the plots. In order to launch a simulation, the HEENS Toolchain Suite (HTS) is used. A Python executable called *creanet.py* is used in order to assemble the sequencer program (which contains the neural model), and to generate the Memory Initialization Files (MIFs) and the configuration files.

In addition to the assembler file, there is an application file also to give to the Python executable. This application file contains the connection between the PEs, as well as their weight. Two different applications have been studied in this thesis.

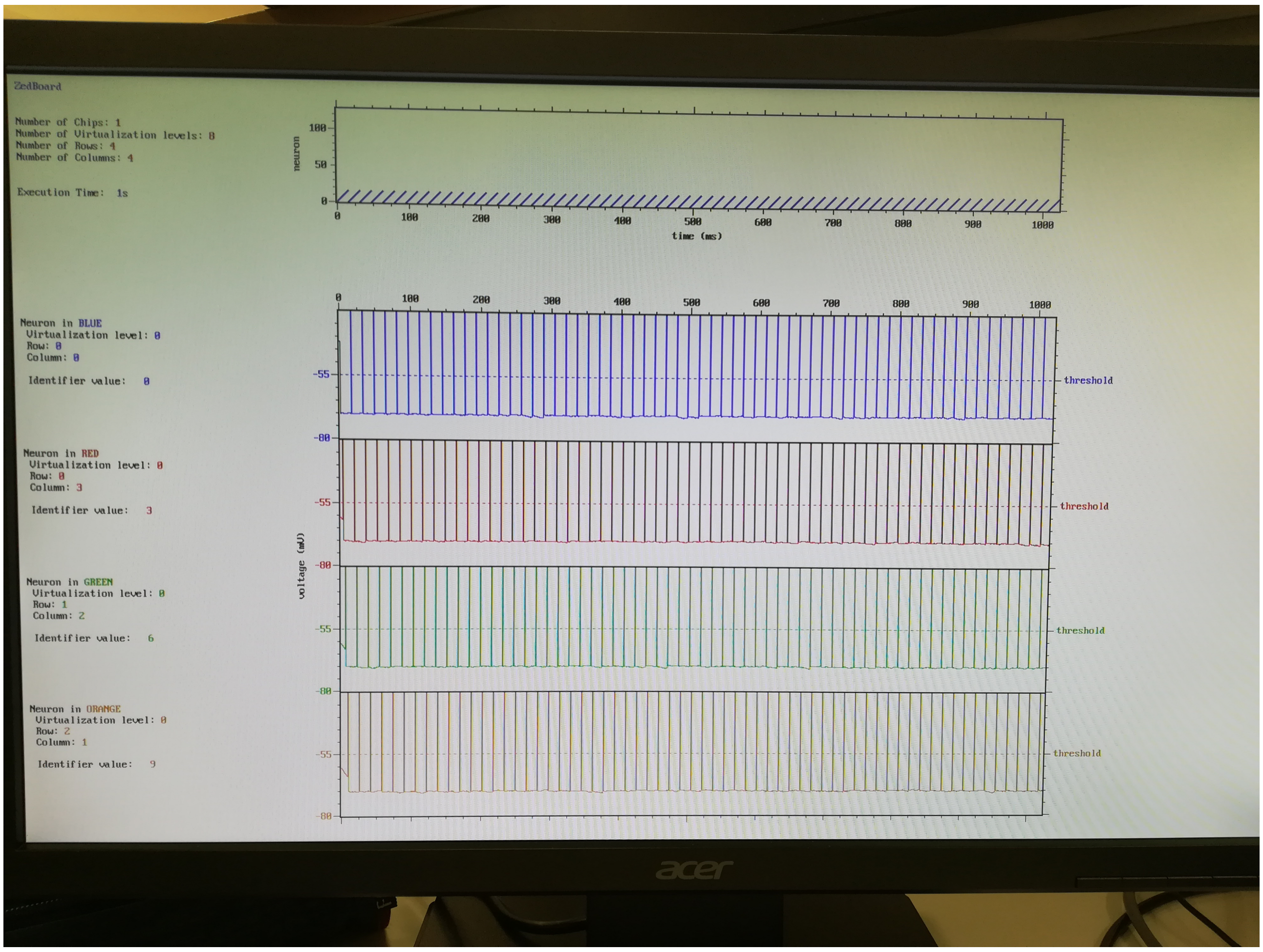
With the addition of the screen generation and HDMI communication parts, ZedBoard does not support the 5x5 configuration anymore due to a lack in BRAM memory.

In these tests, to be able to take pictures and see things on the screen, the simulation has been limited to 1024 time stamps. However, this number (which is defined thanks to the HTR register in the PS code) can be set to a much larger value to have longer simulations and a continuous real-time display.

7.2 Ring oscillator

One of the simplest application is the oscillator. Each neuron are connected to the next one with a chosen weight (here 5500) so that the first spike in neuron 0 will lead to a spike in neuron 1, which will then lead to another spike in neuron 2, and so on. In order to have the ring behavior, the last neuron is simply connected to neuron 0 with the same weight as for the other connections. It corresponds to the behavior that was emulated when testing the raster plot implementation (except that it was not looping).

The results obtained on the monitor screen are presented in Figures 42 and 43, with the 4x4 configuration on ZedBoard.



```
ZedBoard

Number of Chips: 1
Number of Virtualization levels: 8
Number of Rows: 4
Number of Columns: 4

Execution Time:  1s

Neuron in BLUE
 Virtualization level: 0
 Row: 0
 Column: 0

 Identifier value:   0

Neuron in RED
 Virtualization level: 0
 Row: 0
 Column: 3

 Identifier value:   3

Neuron in GREEN
 Virtualization level: 0
 Row: 1
 Column: 2

 Identifier value:   6

Neuron in ORANGE
 Virtualization level: 0
 Row: 2
 Column: 1

 Identifier value:   9
```

Figure 42: Result screen of the ring oscillator example on ZedBoard with 4x4 array.

The global information can be seen in the top left, and the selected neurons information is at the left of each of the 4 plots. For the oscillator, the selection of the neurons to monitor is not very important as they all have the same behavior. I have chosen neurons 0, 3, 6 and 9.

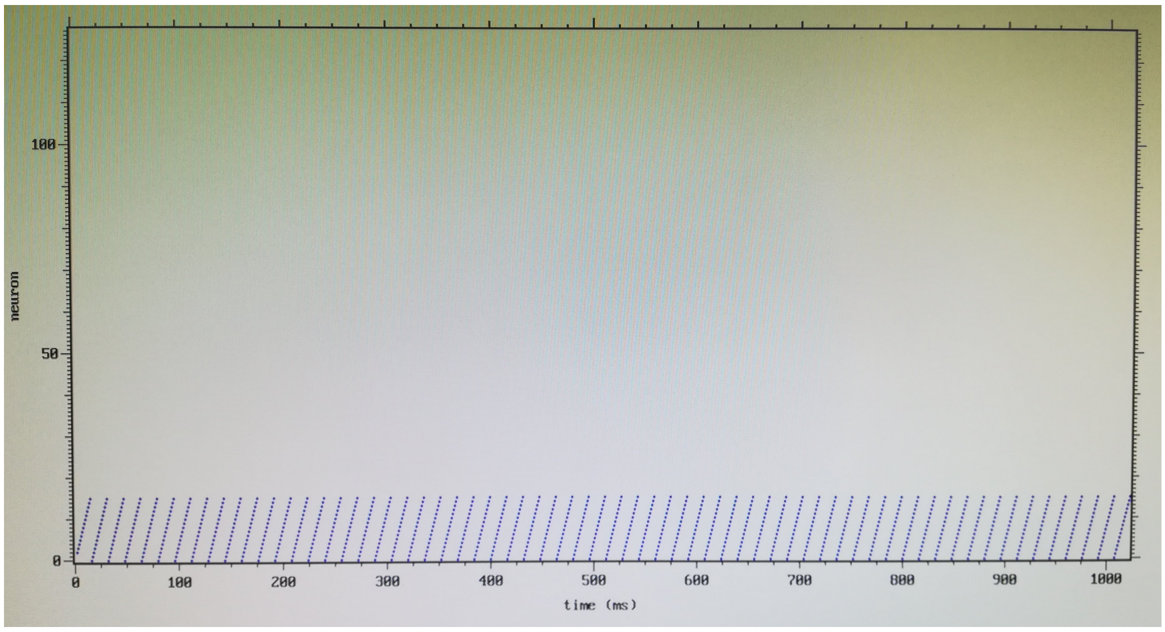


Figure 43: Result screen of the ring oscillator example on ZedBoard with 4x4 array, in the extended configuration of the raster plot.

Neurons from 0 to 15 generate a spike in a oscillating way. For the membrane potentials, we can see that there is a shift in time between the different neurons, but otherwise, the plots are the same.

With the very high weight (5500) that was set on the connections, the membrane potential increases drastically when a neuron receives a spike from the previous neuron. It actually saturates at $-30\,\mathrm{mV}$, and then returns to the resting potential very abruptly as well.

The spacing between each spike of a certain neuron is of 16 time stamps, the time for the ring oscillator to do a loop.

7.3 Delay lines

The topology of the delay lines can be seen in Figure 44. Neuron 0 spike triggers the delay lines and time base.

For delay line 1, neurons 1 and 2 have positive feedback, so they oscillate, until neuron 3 fires and inhibits them.

Delay line 2 operates with the same topology, but its delay will be bigger because neuron 6 will take more time to fire due to the smaller weight (150) between neurons 5 and 6, than between neurons 2 and 3 (200).

The time base oscillates continuously with a programmable rate defined by the weight (300) between neurons 8 and 9.

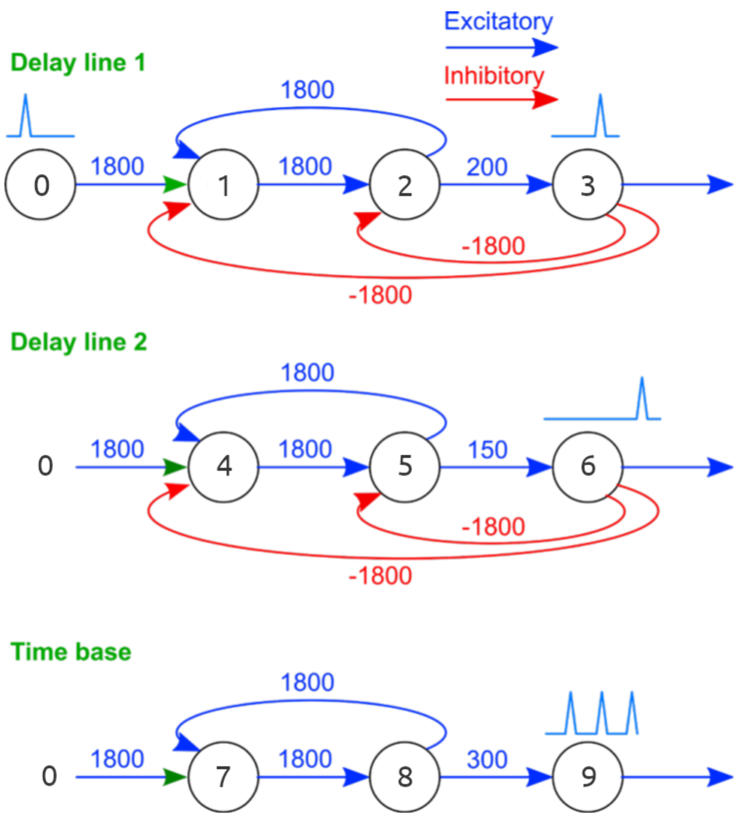


Figure 44: Example of two delay lines with short and longer delays, and time base. [40]

Consequently, the most interesting neurons to monitor are neurons 3, 6 and 9. We decided to monitor neuron 0 as the fourth neuron.

The onscreen results can be seen in Figures 45 and 46.

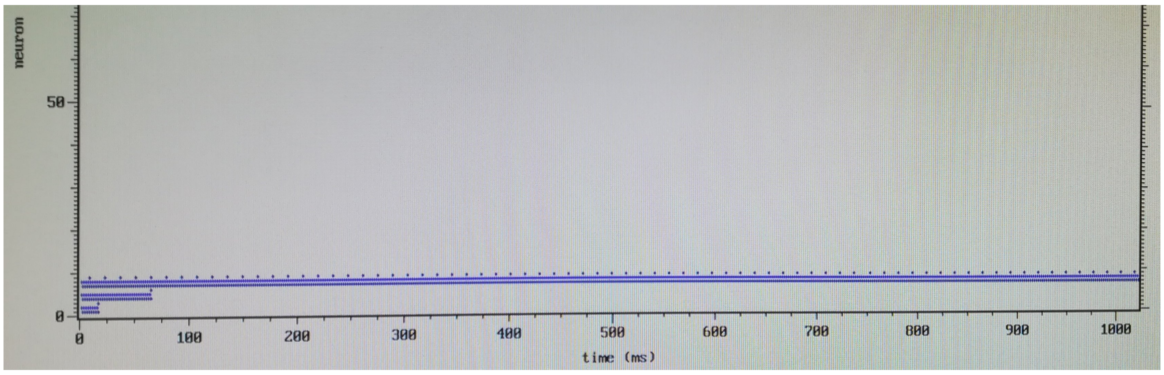
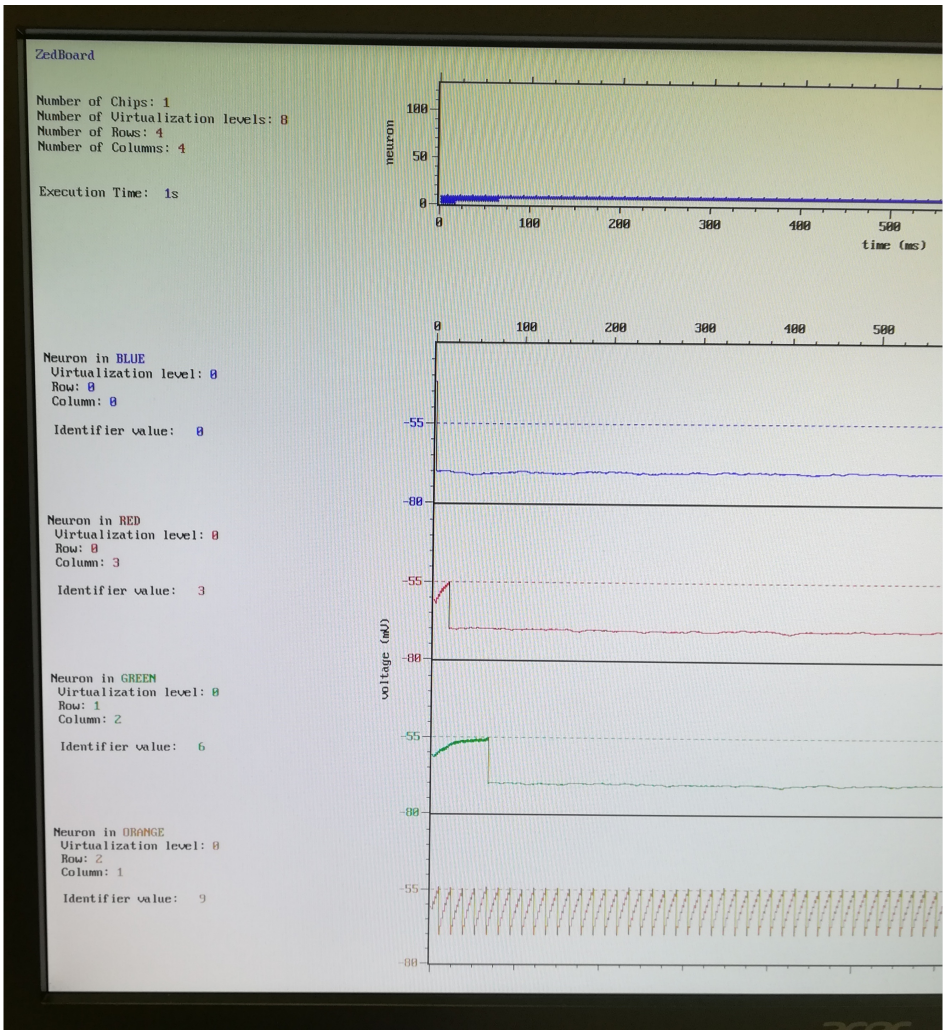


Figure 45: Result screen of the delay lines example on ZedBoard with 4x4 array, in the extended configuration of the raster plot.



```
ZedBoard

Number of Chips: 1
Number of Virtualization levels: 8
Number of Rows: 4
Number of Columns: 4

Execution Time:  1s

Neuron in BLUE
 Virtualization level: 0
 Row: 0
 Column: 0

 Identifier value:   0

Neuron in RED
 Virtualization level: 0
 Row: 0
 Column: 3

 Identifier value:   3

Neuron in GREEN
 Virtualization level: 0
 Row: 1
 Column: 2

 Identifier value:   6

Neuron in ORANGE
 Virtualization level: 0
 Row: 2
 Column: 1

 Identifier value:   9
```

Figure 46: Result screen of the delay lines example on ZedBoard with 4x4 array, zoomed on the left of the screen.

Neuron 0 membrane potential remains around $-70\,\mathrm{mV}$, which is the defined resting potential. Its first spike at the very beginning of the simulation cannot be seen in the raster plot, but its membrane potential is going down from above the threshold value (of $-55\,\mathrm{mV}$).

Neurons 1 and 2 can be seen oscillating in the raster plot as expected, until neuron 3 fires. We can see that its potential increases little by little until it reaches the threshold value, which corresponds to the time of its spike.

The behavior is very similar for the second delay line, except that it lasts longer, and neuron 6 potential grows slower, which is logical with the lower weight than in the first delay line.

For the base time line, the behavior is as expected: neuron 9 produces a spike even sooner than neuron 3, with its membrane potential increasing faster, which is logical with the larger weight than in the first delay line. Besides, as there is no inhibition system, the behavior keeps repeating, creating this base time.

8 Conclusion and future development

8.1 Summary

One of the goals of this thesis was to establish the communication via HDMI to a monitor screen. This was successfully achieved. Both FPGAs ZedBoard and Zynq ZC706 are able, using only their Programmable Logic, to first program the HDMI transmitter ADV7511 using I2C, and then to generate bands of colors in RGB format and send this color information (by converting them into the YCbCr format in the case of ZedBoard) to the transmitter. The latter will then send this information to the HDMI connector and finally the screen. This display feature is working on screens of both 1680x1050 and 1920x1080 resolutions.

The display of text on the screen was also successfully achieved, and all required information is now shown on the screen: the name of the board, the number of chips, the number of virtualization levels, as well as the number of rows and columns used. The execution time is also displayed in the form of seconds, minutes, hours and days.

In order to monitor the SNN, the raster plot of the spiking neurons over time is being displayed in real time on the screen. This plot possesses the features of a classical chart with drawn axes, ticks on the axes, labels on these ticks to show the values on the axes, as well as axis labels.

This plot is limited to 200 neurons in order to be able to show other plots below. However, in order to show a plot with more neurons or on a bigger scale, a button used as a switch has been used. If more than 200 neurons have been defined, pressing the button permits to switch to a display of only the raster plot with all the defined neurons (the maximum is 968 neurons due to screen height limitations). If less than 200 or 200 neurons are defined, it switches to a display with only the raster plot but on a bigger scale: with every neuron being positioned every four vertical pixels.

The reserved space below the maximum-200-neuron raster plot is used in order to show an internal parameter, the membrane potential, of 4 chosen neurons. These four plots are directly one above the other and use the same horizontal axis as the raster plot. They are displayed using four different colors. In addition to the chart features that are also used for the raster plot, a threshold is displayed thanks to a dotted line, as well as a “threshold voltage” label at the right of the plots. In order to know which neurons are the ones selected to be monitored, their relative information is written at the left of each plot.

8.2 Future work

The utilization of divided clocks (for instance in the I2C sending protocol) could lead to metastability as the processes are synchronized on created clocks and not the main generated clock (here the pixel clock), which leads to time-domain crossing which is something to be avoided. Instead, one can use counters also to divide the clock but in order to generate pulses as enable signals (and not clocks). Then, processes will use these enable signals while still being synchronized at the general clock.

The protocol used in order to read the data from the neural parameters FIFO is not robust as it has to wait for four values before writing in the memory, and if it receives fewer than these four values, nothing is written. This case can occur when the signal used to know which neurons to monitor contains twice the same value (which will designate the same neuron). Then, the neural parameter data will be written only once into the FIFO. Consequently, the FIFO will contain one value less than expected. In order to fix this issue, the system to read the FIFO can write each time in memory, which means that it needs to read also before writing. This procedure will resemble that of the spikes FIFO reading, which appends the information and includes a first erasing state.

A problem is occurring on Zynq ZC706: the cooling fan stops working when the FPGA is programmed. It seems to be a problem linked to the voltage definition of IO banks [41]. However, the fan was still working during the first HDMI test on ZC706, which uses the same IO definition. In the end, I have not been able to find and solve the issue.

A point of concern is regarding the display of the neurons to monitor and their internal parameters. Indeed, as HEENS works, the sending of the analog values into the FIFO is made in neurons ascending order. However, the array that contains the identifiers of the neurons to monitor (which is set in the PS with HMNR register) can be in any order. And it is based on this array that the information to identify the neuron is written on the left of each plot. Thus, the neuron information may not match the parameters plot on its right. Therefore, for now, we have to take care that the neurons to monitor are entered in ascending order. Nonetheless, it would be a good idea to have a procedure that sorts this array, and rather in the PS because it would be easier.

The programming was made using many constants in VHDL. Thus, to change their value, it cannot be done during the process by console input for example, and it requires a new run of synthesis, implementation and bitstream generation. Being able to change these values by serial communication could be very handy for instance for the display of other analog neural parameters such as the firing rate or the synaptic weights. This would also mean changing the conversion from the analog parameters into easy-to-plot values. Indeed, now it is from a value between -8000 and -3000 to an integer between 0 and 179. Actually, the target range of 180 pixels will remain the same if we stick to the idea of displaying information for four different neurons, while conserving the raster plot above (on maximum 200 neurons). However, the first range of values (here from -8000 to -3000) will be different for other neural parameters.

In addition, many more features can be implemented for the display. For example, if we are working with a 12x12 or 13x13 array size, the additional neurons above 968 will not be represented at all. An idea could be to have via the console (all the three available buttons are already used on ZC706) the possibility to choose to display the large-ID neurons or the low-ID neurons, or directly to specify the range to be displayed. This would mean an increase in the size of the spikes memory, whose width is currently 968, as well as an increase in the size of the components that will have to handle larger arrays.

To conclude, HDMI display has been established on FPGA and it enables the monitoring of the Spiking Neural Network in real time thanks to two plots: the raster plot and internal neural parameters such as the membrane potential for some chosen neurons of the network. However, it remains a first work and more development can be done to further improve the real-time display.

References

[1] Samanwoy Ghosh-Dastidar and Hojjat Adeli. “Third Generation Neural Networks: Spiking Neural Networks”. In: *Advances in Computational Intelligence* 116 (2009), pages 167–178. DOI: 10.1007/978-3-642-03156-4_17 (see p. 1).

[2] Micah Richert, Jayram Nageswaran, Nikil Dutt, and Jeffrey Krichmar. “An Efficient Simulation Environment for Modeling Large-Scale Cortical Processing”. In: *Frontiers in Neuroinformatics* 5 (2011). DOI: 10.3389/fninf.2011.00019 (see p. 1).

[3] Mireya Zapata Rodríguez. “Arquitectura Escalable SIMD con Conectividad Jerárquica y Reconfigurable para la Emulación de SNN”. Spanish. PhD thesis. Universitat Politècnica de Catalunya, Departament d’Enginyeria Electrònica, September 2017 (see pp. 1–5, 7, 8).

[4] Athul Sripad, Giovanny Sanchez, Mireya Zapata, Vito Pirrone, Taho Dorta, Salvatore Cambria, Albert Marti, Karthikeyan Krishnamourthy, and Jordi Madrenas. “SNAVA - A real-time multi-FPGA multi-model spiking neural network simulation architecture”. In: *Neural Networks* 97 (2018), pages 28–45. DOI: 10.1016/j.neunet.2017.09.011 (see pp. 1, 4).

[5] Josep Angel Oltra-Oltra, Jordi Madrenas, Mireya Zapata, Bernardo Vallejo, Diana Mata-Hernandez, and Shigeo Sato. “Hardware-Software Co-Design for Efficient and Scalable Real-Time Emulation of SNNs on the Edge”. In: *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2021, pages 1–5. DOI: 10.1109/ISCAS51556.2021.9401615 (see p. 1).

[6] Steven Walczak and Narciso Cerpa. “Artificial Neural Networks”. In: *Encyclopedia of Physical Science and Technology (Third Edition)*. Edited by Robert A. Meyers. Third Edition. Academic Press, 2003, pages 631–645. DOI: 10.1016/B0-12-227410-5/00837-1 (see p. 2).

[7] Y.-S. Park and S. Lek. “Chapter 7 - Artificial Neural Networks: Multilayer Perceptron for Ecological Modeling”. In: *Ecological Model Types*. Edited by Sven Erik Jørgensen. Volume 28. Developments in Environmental Modelling. Elsevier, 2016, pages 123–140. DOI: 10.1016/B978-0-444-63623-2.00007-4 (see p. 2).

[8] Fatemeh Falah, Omid Rahmati, Mohammad Rostami, Ebrahim Ahmadisharaf, Ioannis N. Daliakopoulos, and Hamid Reza Pourghasemi. “14 - Artificial Neural Networks for Flood Susceptibility Mapping in Data-Scarce Urban Areas”. In: *Spatial Modeling in GIS and R for Earth and Environmental Sciences*. Edited by Hamid Reza Pourghasemi and Candan Gokceoglu. Elsevier, 2019, pages 323–336. DOI: 10.1016/B978-0-12-815226-3.00014-4 (see p. 2).

[9] Arash Malekian and Nastaran Chitsaz. “Chapter 4 - Concepts, procedures, and applications of artificial neural network models in streamflow forecasting”. In: *Advances in Streamflow Forecasting*. Edited by Priyanka Sharma and Deepesh Machiwal. Elsevier, 2021, pages 115–147. DOI: 10.1016/B978-0-12-820673-7.00003-2 (see p. 2).

[10] Jesus L. Lobo, Javier Del Ser, Albert Bifet, and Nikola Kasabov. "Spiking Neural Networks and online learning: An overview and perspectives". In: *Neural Networks* 121 (2020), pages 88–100. DOI: `10.1016/j.neunet.2019.09.004` (see p. 3).

[11] Corrado Bonfanti. "Parameter Monitoring and Communication in a Ring-Topology-Based SNN Emulator Hardware". Master's thesis. Politecnico di Torino, Universitat Politècnica de Catalunya, November 2020 (see pp. 4, 8).

[12] *Vivado Design Suite User Guide. Synthesis.* v2013.2. Xilinx, June 19, 2013. `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_2/ug901-vivado-synthesis.pdf` (visited on 13/1/2022) (see p. 11).

[13] *Vivado Design Suite User Guide. Implementation.* v2020.2. Xilinx, February 26, 2021. `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_2/ug904-vivado-implementation.pdf` (visited on 13/1/2022) (see p. 11).

[14] *Vivado Design Suite User Guide. Designing with IP.* v2019.2. Xilinx, March 3, 2020. `https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug896-vivado-ip.pdf` (visited on 13/1/2022) (see p. 11).

[15] *High-Definition Multimedia Interface.* Version 1.3a. Hitachi, Matsushita, Philips, Silicon Image, Sony, Thomson, Toshiba, November 10, 2006 (see pp. 12, 21).

[16] *Monitor Technology Guide.* NEC Display Solutions, March 15, 2007. `https://web.archive.org/web/20070315085244/http://www.necdisplay.com/support/css/monitortechguide/index05.htm#aspectratio` (visited on 31/12/2021) (see p. 13).

[17] Teoalida. *Screen resolution statistics.* The world of Teoalida. January 2016. `https://www.teoalida.com/webdesign/screen-resolution/` (visited on 31/12/2021) (see p. 13).

[18] Will Green. *Video Timings: VGA, SVGA, 720p, 1080p.* Project F - FPGA Development. June 26, 2020. `https://projectf.io/posts/video-timings-vga-720p-1080p/` (visited on 29/12/2021) (see p. 14).

[19] *VESA Signal 1680 x 1050 @ 60 Hz timing.* TinyVGA.com. 2008. `http://tinyvga.com/vga-timing/1680x1050@60Hz` (visited on 29/12/2021) (see p. 14).

[20] Nelson Campos. *RGB to YCbCr conversion. Playing with bits and pixels.* sistenix.com. August 21, 2016. `https://sistenix.com/rgb2ycbcr.html` (visited on 30/12/2021) (see pp. 16, 34).

[21] Keith Jack. *Video Demystified. A Handbook for the Digital Engineer.* Fourth Edition. Newnes, Elsevier, 2005. ISBN: 0-7506-7822-4 (see pp. 16, 34).

[22] *The Important Role of Luminance and Chrominance in Noise Reduction.* Neatlab. September 12, 2018. `https://blog.neatvideo.com/post/luminance-and-chrominance-1` (visited on 30/12/2021) (see p. 16).

[23] Adam Babcock. *Chroma Subsampling. 4:4:4 vs 4:2:2 vs 4:2:0.* RTINGS.com. March 4, 2019. `https://www.rtings.com/tv/learn/chroma-subsampling` (visited on 1/1/2022) (see p. 16).

[24] *Chroma Subsampling – 4:4:4 vs 4:2:2 vs 4:2:0.* Learn Media Tech. `https://learnmediatech.com/chroma-subsampling-444-vs-422-vs-420/` (visited on 1/1/2022) (see p. 16).

[25] Charles Poynton. *Chroma subsampling notation*. January 24, 2008. `http://poynton.ca/PDFs/Chroma_subsampling_notation.pdf` (visited on 1/1/2022) (see p. 16).

[26] *ZedBoard*. Revision D.2. Digilent, March 4, 2013. `https://www.xilinx.com/support/documentation/university/XUP%20Boards/XUPZedBoard/documentation/ZedBoard_RevC.1_Schematic_130129.pdf` (visited on 1/1/2022) (see pp. 17, 24, 25).

[27] *ZC706 Evaluation Platform*. Version 1.1, Revision 02. Xilinx, October 31, 2012. `https://www.xilinx.com/support/documentation/boards_and_kits/zynq-7000/zc706-schematic-xtp215-rev1-1.pdf` (visited on 1/1/2022) (see pp. 18, 25).

[28] *ADV7511. Programming Guide*. Revision G. Advantiv, Analog Devices, March 2012. `https://www.analog.com/media/en/technical-documentation/user-guides/ADV7511_Programming_Guide.pdf` (visited on 1/1/2022) (see pp. 18–21, 25, 84–88).

[29] *ZC706 Evaluation Board for the Zynq-7000 XC7Z045 SoC. User Guide*. v1.8. Xilinx, August 6, 2019. `https://www.xilinx.com/content/dam/xilinx/support/documentation/boards_and_kits/zc706/ug954-zc706-eval-board-xc7z045-ap-soc.pdf` (visited on 5/1/2022) (see pp. 24, 89).

[30] *PCA9548A Low Voltage 8-Channel I2C Switch with Reset*. Revised March 2021. Texas Instruments, June 2009. `https://www.ti.com/lit/ds/symlink/pca9548a.pdf` (visited on 11/1/2022) (see p. 24).

[31] Mike Field. *Zedboard VGA HDMI*. September 13, 2013. `http://web.archive.org/web/20190821200752/http://hamsterworks.co.nz/mediawiki/index.php/Zedboard_VGA_HDMI` (visited on 5/1/2022) (see pp. 25, 29, 35, 39, 40).

[32] Pierre Molinaro. *Systèmes interconnectés (SYSIT). Cours, Option C2Syst2E*. French. Centrale Nantes, January 2019 (see pp. 27, 28).

[33] *Fast Mode, I2C Bus*. I2C Bus. `https://www.i2c-bus.org/fastmode/` (visited on 7/1/2022) (see p. 29).

[34] *Booth's multiplication algorithm*. Wikipedia. December 2, 2021. `https://en.wikipedia.org/wiki/Booth's_multiplication_algorithm` (visited on 20/1/2022) (see p. 34).

[35] *Choosing FPGA or DSP for your Application*. Hunt Engineering. May 5, 2009. `http://hunteng.co.uk/info/fpga-or-dsp.htm` (visited on 13/1/2022) (see p. 35).

[36] Analog Devices, Inc. *HDL. hdl/projects/common/zc706/zc706_system_constr.xdc*. Github. August 8, 2019. `https://github.com/analogdevicesinc/hdl/blob/master/projects/common/zc706/zc706_system_constr.xdc` (visited on 11/1/2022) (see p. 40).

[37] *XPM CDC Generator. LogiCORE IP Product Guide, Vivado Design Suite*. v1.0. Xilinx, February 9, 2021. `https://www.xilinx.com/support/documentation/ip_documentation/xpm_cdc_gen/v1_0/pg382-xpm-cdc-generator.pdf` (visited on 2/2/2022) (see p. 49).

[38] Derek-X-Wang. *VGA-Text-Generator*. Github. December 5, 2015. `https://github.com/Derek-X-Wang/VGA-Text-Generator` (visited on 12/1/2022) (see pp. 50, 51).

[39] MadLittleMods. *FP-V-GA-Text*. Github. December 16, 2013. `https://github.com/MadLittleMods/FP-V-GA-Text` (visited on 12/1/2022) (see p. 50).

[40] Jordi Madrenas, Mireya Zapata, Josep Àngel Oltra, Bernardo Vallejo, Satoshi Moriya, and Shigeo Sato. *Efficient Digital Spike Processing of Frequency-Encoded Sensor Signals for Edge Computing*. 2020 (see p. 74).

[41] *Zynq-7000 SoC ZC706 Evaluation Kit, ZC706 Fan stops working when the Vadj voltage is set to 1.8V*. Xilinx. September 23, 2021. `https://support.xilinx.com/s/article/61712?language=en_US` (visited on 1/2/2022) (see p. 78).

[42] *HEENS Architecture. Datasheet, FPGA PU*. Departament d'Enginyeria Electrònica, Universitat Politècnica de Catalunya, 2021 (see pp. 93, 95).

Appendix

ADV7511 registers map

The following tables (Figures A.1 to A.11) give the different values to write in some chosen registers of the HDMI transmitter ADV7511.

Fixed registers that must be set

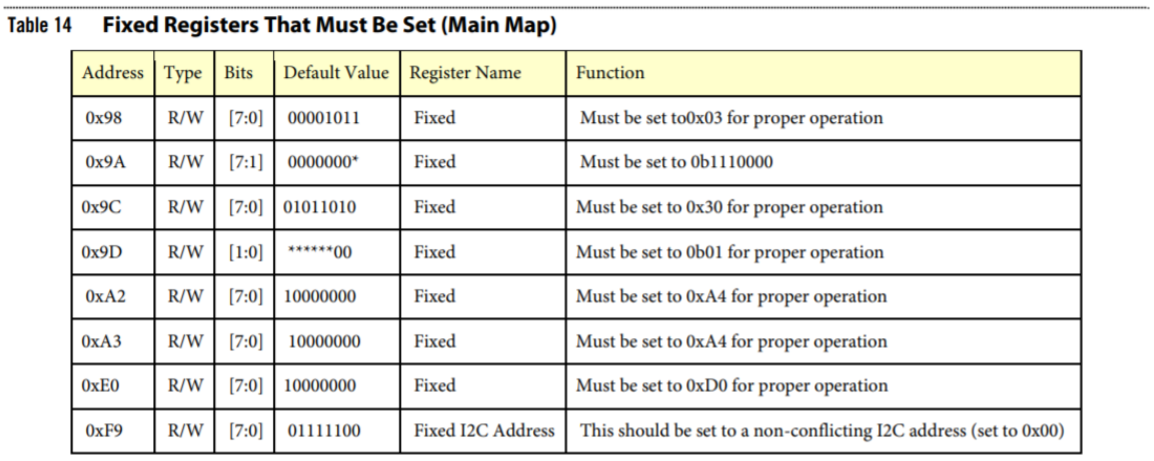
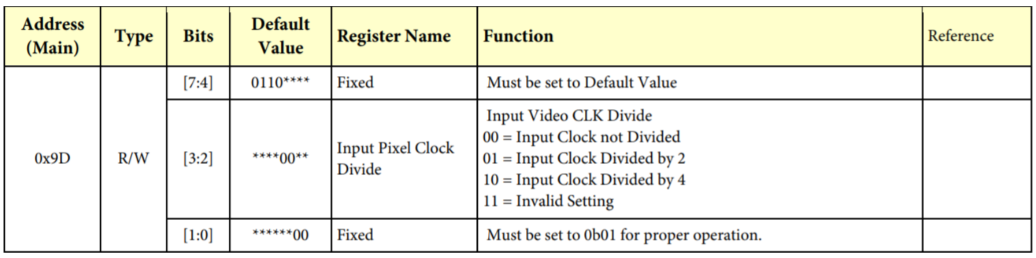


Table 14 Fixed Registers That Must Be Set (Main Map)

Address	Type	Bits	Default Value	Register Name	Function
0x98	R/W	[7:0]	00001011	Fixed	Must be set to0x03 for proper operation
0x9A	R/W	[7:1]	0000000*	Fixed	Must be set to 0b1110000
0x9C	R/W	[7:0]	01011010	Fixed	Must be set to 0x30 for proper operation
0x9D	R/W	[1:0]	******00	Fixed	Must be set to 0b01 for proper operation
0xA2	R/W	[7:0]	10000000	Fixed	Must be set to 0xA4 for proper operation
0xA3	R/W	[7:0]	10000000	Fixed	Must be set to 0xA4 for proper operation
0xE0	R/W	[7:0]	10000000	Fixed	Must be set to 0xD0 for proper operation
0xF9	R/W	[7:0]	01111100	Fixed I2C Address	This should be set to a non-conflicting I2C address (set to 0x00)

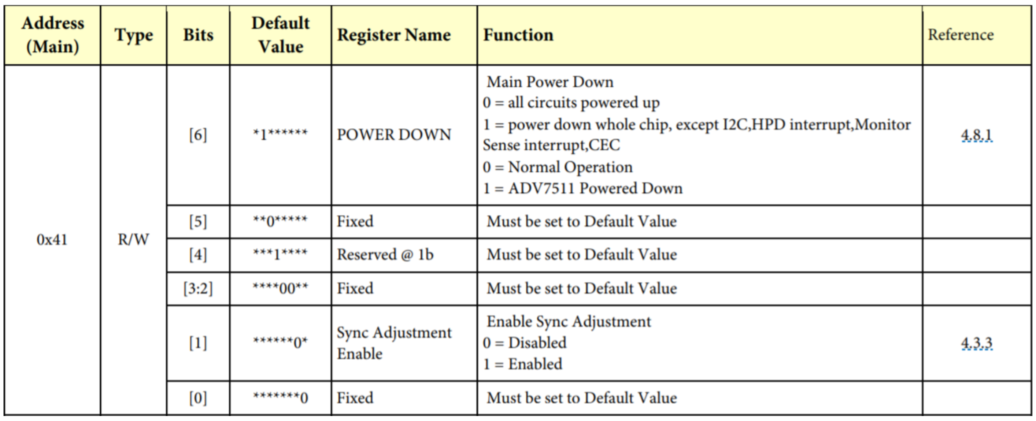
Figure A.1: Table of fixed registers that must be set. [28]



Address (Main)	Type	Bits	Default Value	Register Name	Function	Reference
0x9D	R/W	[7:4]	0110****	Fixed	Must be set to Default Value	
		[3:2]	****00**	Input Pixel Clock Divide	Input Video CLK Divide 00 = Input Clock not Divided 01 = Input Clock Divided by 2 10 = Input Clock Divided by 4 11 = Invalid Setting	
		[1:0]	******00	Fixed	Must be set to 0b01 for proper operation.	

Figure A.2: Register map of 0x9D. [28]

Main Power Up



Address (Main)	Type	Bits	Default Value	Register Name	Function	Reference
0x41	R/W	[6]	*1******	POWER DOWN	Main Power Down 0 = all circuits powered up 1 = power down whole chip, except I2C,HPD interrupt,Monitor Sense interrupt,CEC 0 = Normal Operation 1 = ADV7511 Powered Down	4.8.1
		[5]	**0*****	Fixed	Must be set to Default Value	
		[4]	***1****	Reserved @ 1b	Must be set to Default Value	
		[3:2]	****00**	Fixed	Must be set to Default Value	
		[1]	******0*	Sync Adjustment Enable	Enable Sync Adjustment 0 = Disabled 1 = Enabled	4.3.3
		[0]	*******0	Fixed	Must be set to Default Value	

Figure A.3: Register map of 0x41. [28]

HDMI or DVI mode

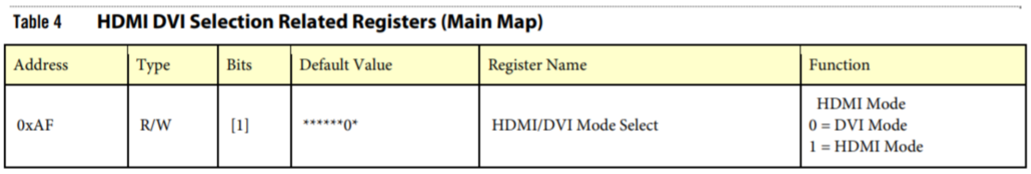
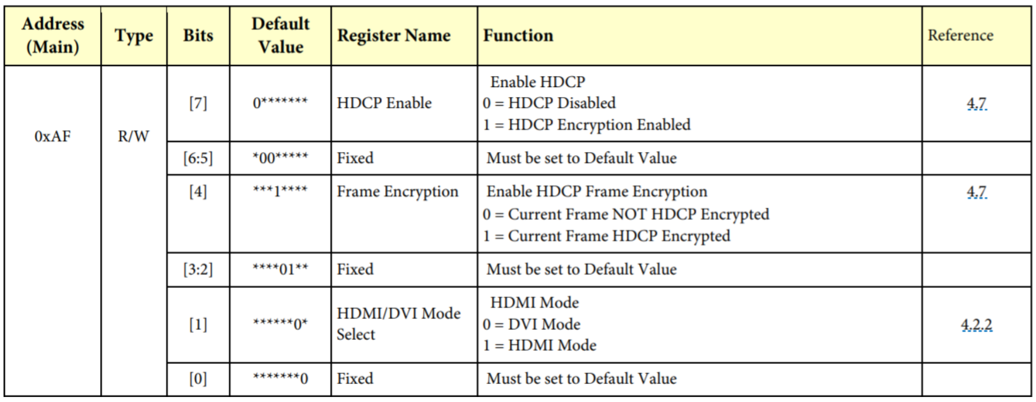


Table 4 HDMI DVI Selection Related Registers (Main Map)

Address	Type	Bits	Default Value	Register Name	Function
0xAF	R/W	[1]	******0*	HDMI/DVI Mode Select	HDMI Mode 0 = DVI Mode 1 = HDMI Mode

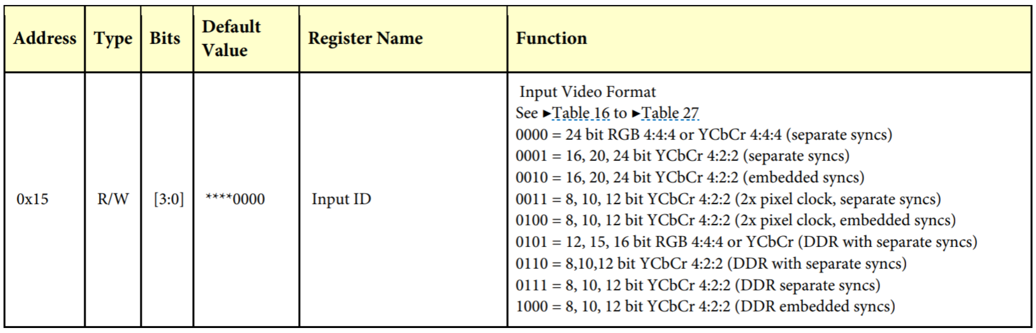
Figure A.4: Table of HDMI or DVI mode. [28]



Address (Main)	Type	Bits	Default Value	Register Name	Function	Reference
0xAF	R/W	[7]	0*******	HDCP Enable	Enable HDCP 0 = HDCP Disabled 1 = HDCP Encryption Enabled	4.7
		[6:5]	*00*****	Fixed	Must be set to Default Value	
		[4]	***1****	Frame Encryption	Enable HDCP Frame Encryption 0 = Current Frame NOT HDCP Encrypted 1 = Current Frame HDCP Encrypted	4.7
		[3:2]	****01**	Fixed	Must be set to Default Value	
		[1]	******0*	HDMI/DVI Mode Select	HDMI Mode 0 = DVI Mode 1 = HDMI Mode	4.2.2
		[0]	*******0	Fixed	Must be set to Default Value	

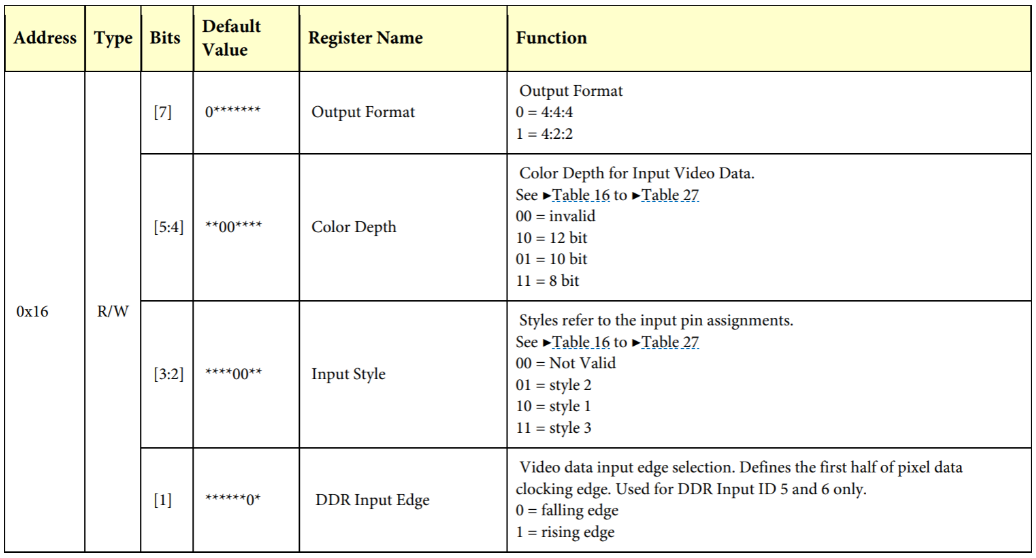
Figure A.5: Register map of 0xAF. [28]

Input Formatting Related Registers



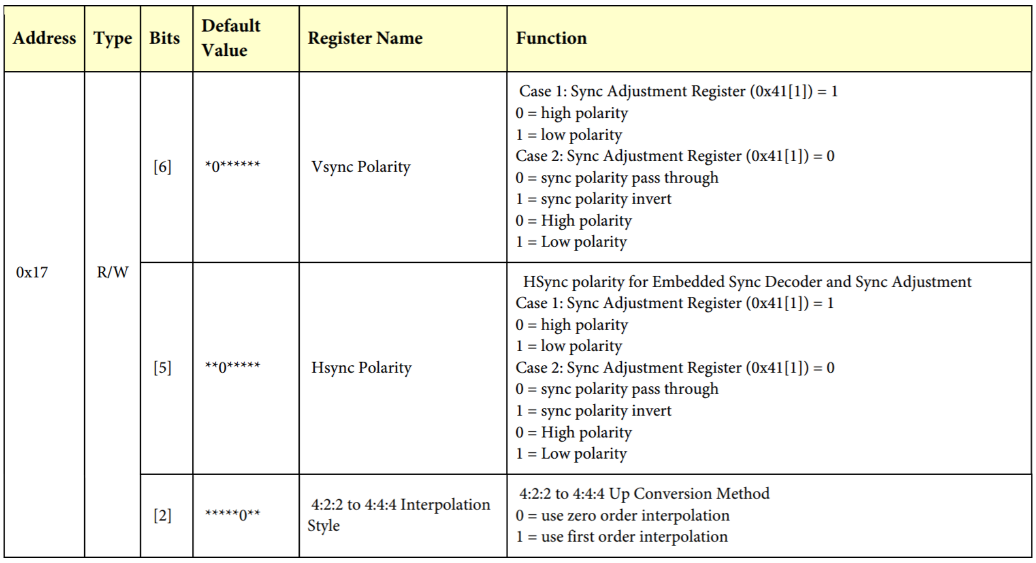
Address	Type	Bits	Default Value	Register Name	Function
0x15	R/W	[3:0]	****0000	Input ID	Input Video Format See ►Table 16 to ►Table 27 0000 = 24 bit RGB 4:4:4 or YCbCr 4:4:4 (separate syncs) 0001 = 16, 20, 24 bit YCbCr 4:2:2 (separate syncs) 0010 = 16, 20, 24 bit YCbCr 4:2:2 (embedded syncs) 0011 = 8, 10, 12 bit YCbCr 4:2:2 (2x pixel clock, separate syncs) 0100 = 8, 10, 12 bit YCbCr 4:2:2 (2x pixel clock, embedded syncs) 0101 = 12, 15, 16 bit RGB 4:4:4 or YCbCr (DDR with separate syncs) 0110 = 8,10,12 bit YCbCr 4:2:2 (DDR with separate syncs) 0111 = 8, 10, 12 bit YCbCr 4:2:2 (DDR separate syncs) 1000 = 8, 10, 12 bit YCbCr 4:2:2 (DDR embedded syncs)

Figure A.6: Register map of 0x15. [28]



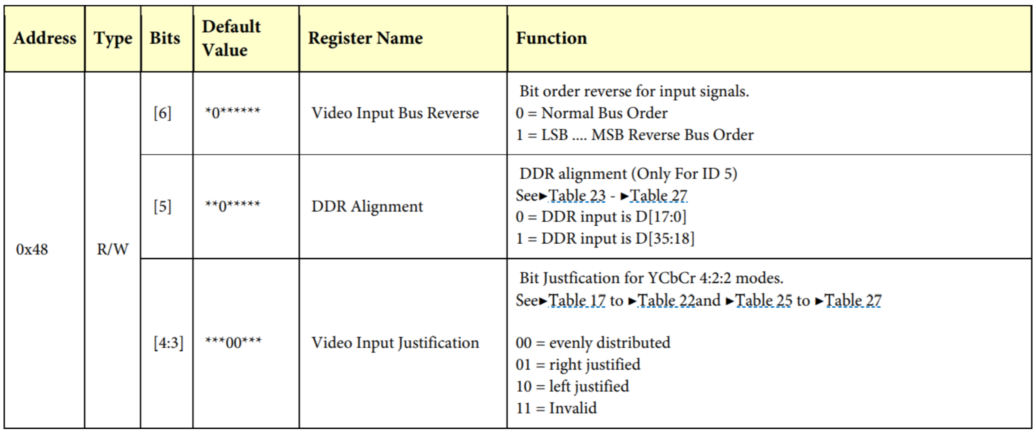
Address	Type	Bits	Default Value	Register Name	Function
0x16	R/W	[7]	0*******	Output Format	Output Format 0 = 4:4:4 1 = 4:2:2
		[5:4]	**00****	Color Depth	Color Depth for Input Video Data. See ►Table 16 to ►Table 27 00 = invalid 10 = 12 bit 01 = 10 bit 11 = 8 bit
		[3:2]	****00**	Input Style	Styles refer to the input pin assignments. See ►Table 16 to ►Table 27 00 = Not Valid 01 = style 2 10 = style 1 11 = style 3
		[1]	******0*	DDR Input Edge	Video data input edge selection. Defines the first half of pixel data clocking edge. Used for DDR Input ID 5 and 6 only. 0 = falling edge 1 = rising edge

Figure A.7: Register map of 0x16. [28]



Address	Type	Bits	Default Value	Register Name	Function
0x17	R/W	[6]	*0******	Vsync Polarity	Case 1: Sync Adjustment Register (0x41[1]) = 1 0 = high polarity 1 = low polarity Case 2: Sync Adjustment Register (0x41[1]) = 0 0 = sync polarity pass through 1 = sync polarity invert 0 = High polarity 1 = Low polarity
		[5]	**0*****	Hsync Polarity	HSync polarity for Embedded Sync Decoder and Sync Adjustment Case 1: Sync Adjustment Register (0x41[1]) = 1 0 = high polarity 1 = low polarity Case 2: Sync Adjustment Register (0x41[1]) = 0 0 = sync polarity pass through 1 = sync polarity invert 0 = High polarity 1 = Low polarity
		[2]	*****0**	4:2:2 to 4:4:4 Interpolation Style	4:2:2 to 4:4:4 Up Conversion Method 0 = use zero order interpolation 1 = use first order interpolation

Figure A.8: Register map of 0x17. [28]



Address	Type	Bits	Default Value	Register Name	Function
0x48	R/W	[6]	*0******	Video Input Bus Reverse	Bit order reverse for input signals. 0 = Normal Bus Order 1 = LSB .... MSB Reverse Bus Order
		[5]	**0*****	DDR Alignment	DDR alignment (Only For ID 5) See►Table 23 - ►Table 27 0 = DDR input is D[17:0] 1 = DDR input is D[35:18]
		[4:3]	***00***	Video Input Justification	Bit Justfication for YCbCr 4:2:2 modes. See►Table 17 to ►Table 22and ►Table 25 to ►Table 27 00 = evenly distributed 01 = right justified 10 = left justified 11 = Invalid

Figure A.9: Register map of 0x48. [28]

CSC related registers

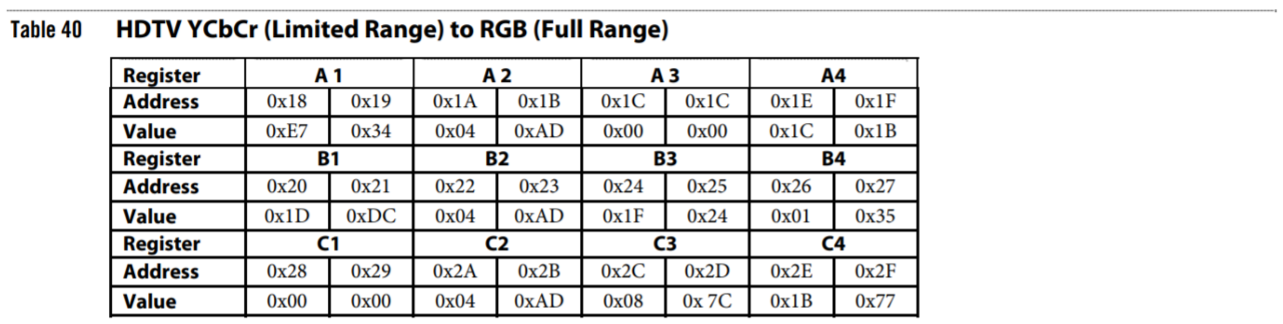


Table 40 **HDTV YCbCr (Limited Range) to RGB (Full Range)**

Register	A 1		A 2		A 3		A4	
Address	0x18	0x19	0x1A	0x1B	0x1C	0x1C	0x1E	0x1F
Value	0xE7	0x34	0x04	0xAD	0x00	0x00	0x1C	0x1B
Register	B1		B2		B3		B4	
Address	0x20	0x21	0x22	0x23	0x24	0x25	0x26	0x27
Value	0x1D	0xDC	0x04	0xAD	0x1F	0x24	0x01	0x35
Register	C1		C2		C3		C4	
Address	0x28	0x29	0x2A	0x2B	0x2C	0x2D	0x2E	0x2F
Value	0x00	0x00	0x04	0xAD	0x08	0x 7C	0x1B	0x77

Figure A.10: Table of CSC - HDTV YCbCr (Limited Range) to RGB (Full Range). [28]

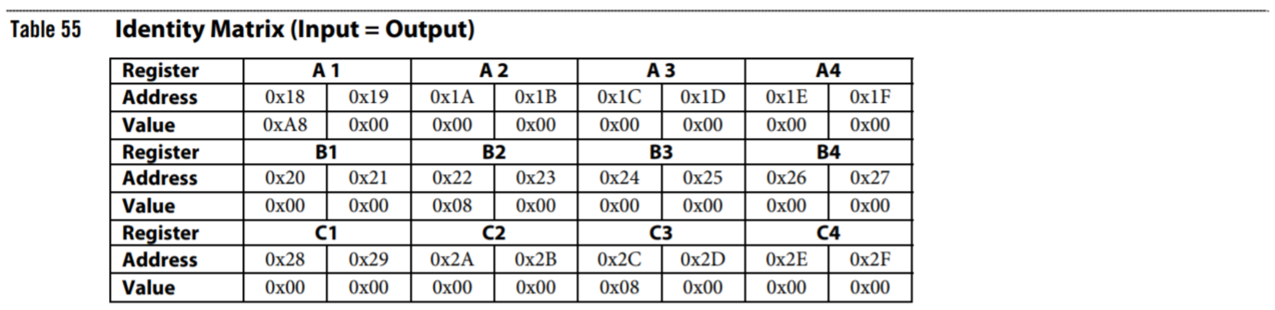


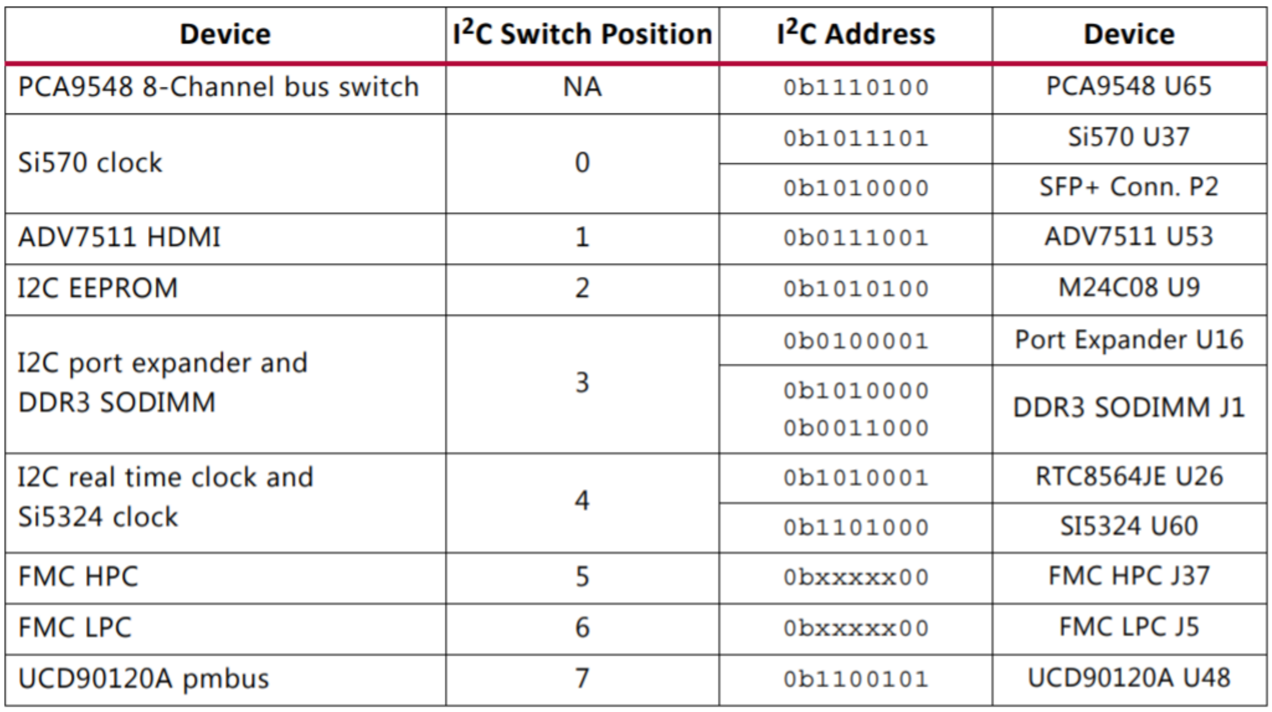
Table 55 **Identity Matrix (Input = Output)**

Register	A 1		A 2		A 3		A4	
Address	0x18	0x19	0x1A	0x1B	0x1C	0x1D	0x1E	0x1F
Value	0xA8	0x00	0x00	0x00	0x00	0x00	0x00	0x00
Register	B1		B2		B3		B4	
Address	0x20	0x21	0x22	0x23	0x24	0x25	0x26	0x27
Value	0x00	0x00	0x08	0x00	0x00	0x00	0x00	0x00
Register	C1		C2		C3		C4	
Address	0x28	0x29	0x2A	0x2B	0x2C	0x2D	0x2E	0x2F
Value	0x00	0x00	0x00	0x00	0x08	0x00	0x00	0x00

Figure A.11: Table of CSC - Identity Matrix (Input = Output). [28]

ZC706 I2C bus switch

This table on Figure A.12 gives the I2C addresses of the different slaves on the I2C bus on ZC706. The first line corresponds to the bus switch itself. The third line with the I2C switch position of 1 is the HDMI transmitter ADV7511.



Device	I^2C Switch Position	I^2C Address	Device
PCA9548 8-Channel bus switch	NA	0b1110100	PCA9548 U65
Si570 clock	0	0b1011101	Si570 U37
		0b1010000	SFP+ Conn. P2
ADV7511 HDMI	1	0b0111001	ADV7511 U53
I2C EEPROM	2	0b1010100	M24C08 U9
I2C port expander and DDR3 SODIMM	3	0b0100001	Port Expander U16
		0b1010000 0b0011000	DDR3 SODIMM J1
I2C real time clock and Si5324 clock	4	0b1010001	RTC8564JE U26
		0b1101000	SI5324 U60
FMC HPC	5	0bxxxxx00	FMC HPC J37
FMC LPC	6	0bxxxxx00	FMC LPC J5
UCD90120A pmbus	7	0b1100101	UCD90120A U48

Figure A.12: I2C slaves address on ZC706. [29]

Raster plot results for 2x2 and 6x6 configurations

6x6 configuration

With the 6x6 configuration on ZC706, the results are the following in the first configuration of the switch with 200 neurons (Figure A.13). The same example memory as for the 5x5 was used in the HEENS simulator component.

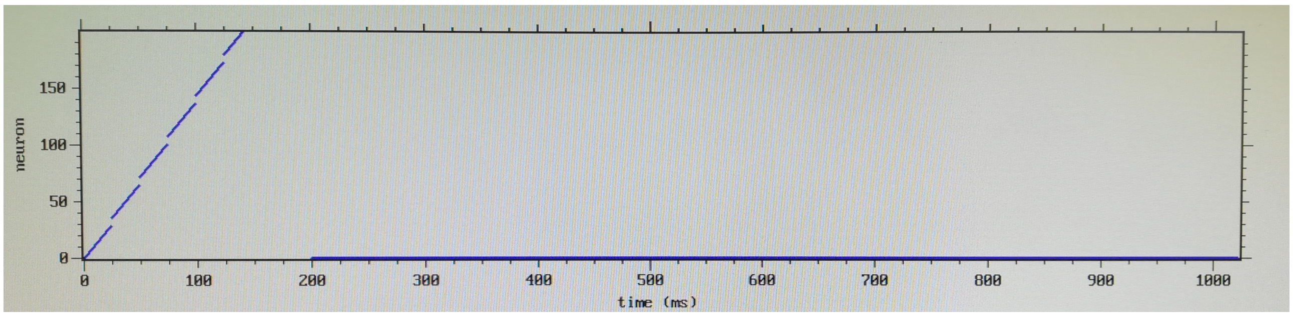


Figure A.13: Result screen of the raster plot implementation for ZC706 with 6x6 array (zoomed on the plot).

The plot ranges only from 0 to 199 in this configuration while the neurons values are up to 287 $(= 6 \times 6 \times 8 - 1)$.

With the extended configuration (by pressing the button), the results of the full raster plot are the following (Figure A.14).

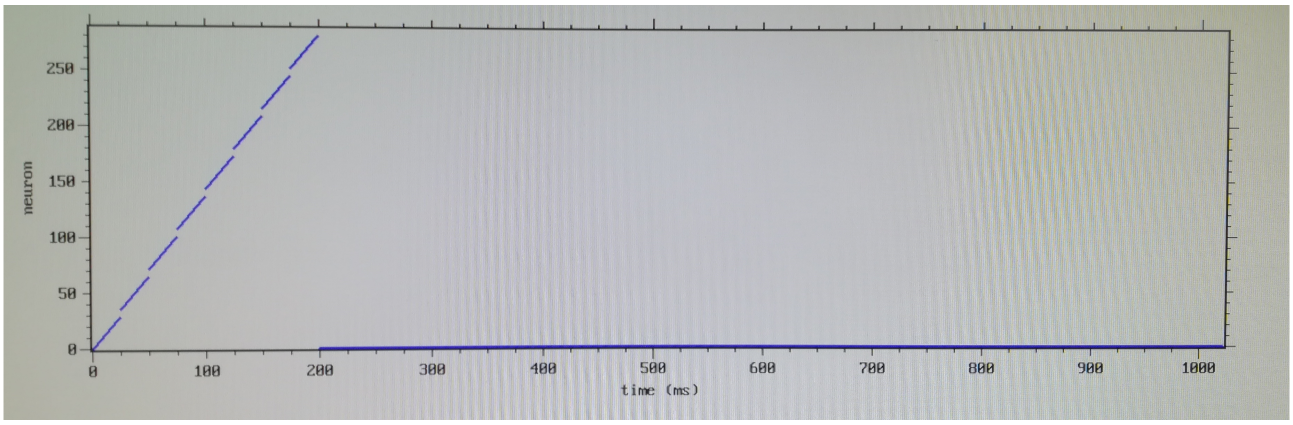


Figure A.14: Result screen of the raster plot implementation for ZC706 with 6x6 array with the extended plot.

The plot now ranges correctly from 0 to 287. As the memory was designed for the 5x5 configuration, we will have on each line only neurons from 0 to 4 spiking and not from 0 to 5, and this will apply for the first 5 lines (and not 6). This explains the obtained plot: for 5 pixels we have the spiking neuron number incrementing, then a vertical jump of 1 pixel, and all this 5 times (for 5 lines), and then a larger vertical jump (of 6 pixels, for the left line of 6 columns), and then it repeats for the next virtualization level, and so on for a total of 8 levels of virtualization.

2x2 configuration

With the 2x2 configuration on ZC706, the results are the following in the first configuration of the switch (Figure A.15). The same example memory as for the 5x5 was used in the HEENS simulator component.

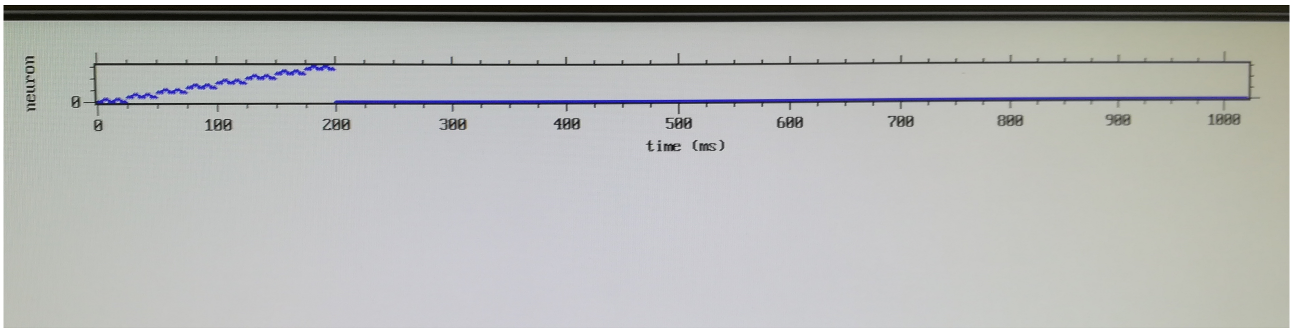


Figure A.15: Result screen of the raster plot implementation for ZC706 with 2x2 array (zoomed on the plot).

The plot ranges only from 0 to 31 ($= 2 \times 2 \times 8 - 1$). The 32-neuron range is lower than the 200 value we have fixed, then the switch will extend the plot by increasing the space between two neurons in the vertical axis up to 4.

With the extended configuration (by pressing the button), the results are the following (Figure A.16).

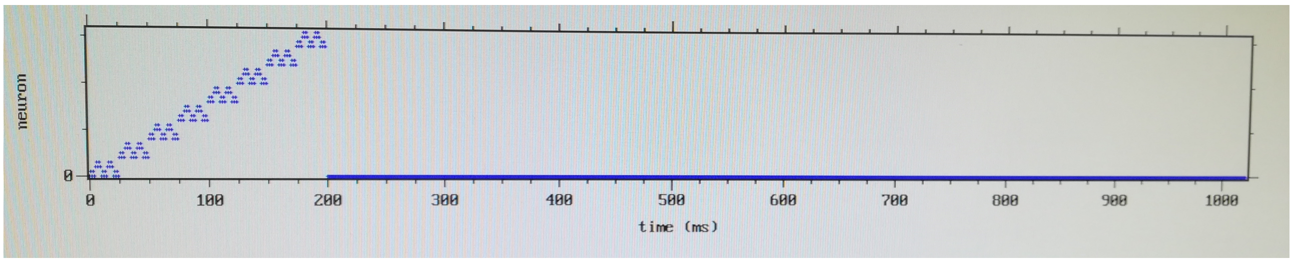


Figure A.16: Result screen of the raster plot implementation for ZC706 with 2x2 array with the extended plot.

The neurons are now every 4 pixels vertically. As the memory was designed for the 5x5 configuration, we will have the first two neurons correctly spiking. As the values are modulus 2, the third neuron corresponds to the first one spiking, and so on, we thus obtain this ‘M’ shape. This is for the first line so the first 2 neurons in this configuration. Subsequently, we have the same on the second line. The third line will correspond to the first line and so on, hence this ‘M’ shape also appears at an upper scale. Finally, this behavior is repeated for all of the 8 virtualization levels.

HEENS Control Interface registers

HEENS Control Interface register map

The registers for HEENS Control Interface (HCI) are listed in the Table A.1.

Offset	Register	Name	Access	Reset
0x00	HEENS Transfer Register	HTR	Read-Write	0x0
0x04	HEENS Status Register	HSR	Read-Write	0x0
0x08	HEENS Monitored Neurons Register	HMNR	Write	0x0
0x0C	Reserved	-	-	-

Table A.1: HCI Registers map. [42]

These registers length is 32 bits. In this document, we will only see HTR and HMNR registers and, in particular, their write operations.

HEENS Transfer Register write operation

The format of the register HEENS Transfer Register in write operation can be seen in Table A.2.

31	30	29	28	27	26	25	24
DT		-	-	-	-	WLEN	

23	22	21	20	19	18	17	16
WLEN							

15	14	13	12	11	10	9	8
WLEN							

7	6	5	4	3	2	1	0
WLEN							

Table A.2: HTR register (write operation). [42]

- DT: Datatype

  0b00: Configuration length
  0b01: Length of DMA (Direct Memory Access) packets to device
  0b10: Number of execution cycles
  0b11: Execution cycle duration time (clock cycles)

- WLEN: Word length

  0x0000000-0x3FFFFFF: Word length of the data being transferred

The number of execution cycles can be set for example to 1024 (0x400) in order to execute only 1024 cycles, which will be fully displayed on the monitor and the display will be still. It can also be set to much larger values in order to increase the number of execution cycles, and the display will be in real time and starts to scroll.

In order to have cycles of 1 ms, the execution cycle duration time has to be set to 125000 (0x1E848) with HEENS clock at 125 MHz. Actually, HEENS clock is at this frequency only on ZC706. On ZedBoard, it is at 50 MHz, so the duration time has to be set at 50000 (0xC350).

Nonetheless, this duration time can be set to different values. We can set larger values to slow down the execution speed, which also reduces the speed of the display scrolling and makes easier to see the real-time display. This value can also be reduced. However, the display may not work anymore due to the limitation in size of the buffers. Indeed, they have a size of 32, and with a speed of 1 ms and a screen refresh rate of 60 Hz, we already fill more than 16 slots of the buffers ($1/(60\,\mathrm{Hz}) \approx 16.67\,\mathrm{ms}$). Then, for example, we cannot reduce the cycle duration time by two down to 500 µs because we would need more than 32 slots in the buffer. An option can be simply to increase the size of the buffers if we want to have a working real-time display with this reduced cycle duration time.

Besides, in all the cases where the cycle duration time is not set to 1 ms, the display of the execution time on the screen and the label of the horizontal axis of the plots, which is *time (ms)*, will not be correct anymore. The displayed time will go slower or faster than the actual real time.

HEENS Monitored Neurons Register write operation

The format of the register HEENS Monitored Neurons Register in write operation can be seen in Table A.3.

31	30	29	28	27	26	25	24
DT		-	-	-	-	-	-

23	22	21	20	19	18	17	16
-	-		VIRT2			ROW2	

15	14	13	12	11	10	9	8
ROW2		COL2				VIRT1	

7	6	5	4	3	2	1	0
	ROW1				COL1		

Table A.3: HMNR register (write operation). [42]

- DT: Datatype

  0b00: Reserved
  0b01: First and second neurons to be monitored
  0b10: Third and fourth neurons to be monitored
  0b11: Reserved

- COL1: Column of the first neuron

  0b0000-0b1111: Column number of the first (or third) neuron to be monitored

- ROW1: Row of the first neuron

  0b0000-0b1111: Row number of the first (or third) neuron to be monitored

- VIRT1: Virtualization level of the first neuron

  0b000-0b111: Virtualization level of the first (or third) neuron to be monitored

- COL2: Column of the second neuron

  0b0000-0b1111: Column number of the second (or fourth) neuron to be monitored

- ROW2: Row of the second neuron

  0b0000-0b1111: Row number of the second (or fourth) neuron to be monitored

- VIRT2: Virtualization level of the second neuron

  0b000-0b111: Virtualization level of the second (or fourth) neuron to be monitored