Design and Validation of a model-based development process for distributed control algorithms with different target platforms

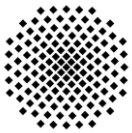
Master Thesis

Autor: **Villar Rio, Laura**
Enrollment number: 3292459
Master in Industrial Engineering

Institut für Steuerungstechnik
der Werkzeugmaschinen und Fertigungseinrichtungen

Supervisor: M. Sc. Caren Dripke

Universität Stuttgart



Universität
Stuttgart

Datum: 29.09.2017

Acknowledgments

I would like to pay special thankfulness, warmth and appreciation to the persons below who made my research and personal development during this long journey successful and assisted me at every point to cherish my goal:

My Supervisor, M.Sc. Caren Dripke for giving the opportunity to join her project during this period. Her support and assistance made it possible to achieve the goal.

My Assistant Supervisor, Dipl.-Ing Florian Frick, whose help at every point during my research helped me to work in time.

To the University of Stuttgart and the people I met during this excepcoinal period, for making me feel at home and let me discover a new range of cultures, which will be really helpful in my future challenges.

My Mom, Dad and Brother, family members and friends, without whom it could not have been possible; they believed in my possibilities even when I didn't. They do not only assisted me morally and emotionally but also financially.

Contents

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

CNC	Computer Numerical Control
FPGA	Field Programmable Gate Array
CPU	Central Processing Unit
ISW	Institute for Control Engineering of Machine Tools and Manufacturing Units
HDL	Hardware Description Language
KS	Knowledge Source
DCS	Distributed Control System
DEVEKOS	Consistent engineering for safe, distributed and communicating multi-component systems
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
I/O	Input / Output
MATLAB	MATrix LABoratory
MOSFET	Metal–Oxide–Semiconductor Field-Effect Transistor
NFETS	n-channel FET (Field-Effect Transistor)
PFETS	p-channel FET (Field-Effect Transistor)
CMOS	Complementary Metal-Oxide Semiconductor
IC	Integrated Circuit
TTL	Transistor-Transistor Logic
PROM	Programmable Read-Only Memory
EPROM	Erasable Programmable Read-Only Memory
DST	Depleted-Substrate Transistor
PLA	Programmable Logic Array
PAL	Programmable Array Logic
EEPROM	Electrically Erasable Programmable Read-Only Memory
GAL	Generic Array Logic
ANSI	American National Standards Institute
DSP	Digital Signal Processing
ASIC	Application Specific Integrated Circuit
ALM	Adaptive Logic Module
PLL	Phase-Locked Loop

1 INTRODUCTION

1.1 MOTIVATION/BACKGROUND

This master's thesis is written at the Universität Stuttgart and submitted to Universitat Politècnica de Catalunya in partial fulfillment of the requirements for Master of Science degree in Industrial Engineering. The thesis is carried out by the student Laura Villar Rio, participating in the Erasmus+ exchange program. It is supervised by M.Sc Caren Dripke, researcher of the Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW).

The thesis is also framed within the DEVEKOS project, a project carried out by several organizations such as the ISW. The context of this project is the age of digitization and industry 4.0. Thanks to the miniaturization of the electronics, control, actuators and sensors now the components can be integrated into a tight space. The embedded software makes such components intelligent and they offer their own abilities in the form of manufacturer-standardized automation functions. These skills are composed in the engineering of machine integrators to higher-quality skills until a level of skill is reached that corresponds to the process of the product being produced. This results in a component-based automation with a suitable component-oriented engineering approach - both prerequisites for the implementation of the currently emerging industry 4.0 standards.

A Distributed Control System (DCS), which will be implemented in this thesis, increases reliability and reduces installation costs by localizing control functions near the process plant, but enables monitoring and supervisory control of the process remotely. The key attribute of a DCS is its reliability due to the distribution of the control processing around nodes in the system. This mitigates a single processor failure. If a processor fails, it will only affect one section of the plant process, as opposed to a failure of a central computer, which would affect the whole process, increasing also productivity. This distribution of computing power local to the field Input/Output (I/O) field connection racks also ensures fast controller processing times by removing possible network and central processing delays. Another advantage, which makes these systems appropriate for the Industry 4.0, is that the introduction of distributed control allowed easy interconnection and re-configuration of plant controls such as cascaded loops and interlocks, and easy interfacing with other production computer systems.

In line with the digitization that Industry 4.0 heralds, future control system functionality will be decided by software and firmware upgrades and be completely independent of hardware in the future. This will allow the user to purchase the control system hardware with limited functionality and as requirements increase they can continue to upgrade to add additional functionality as required — such as faster cycle times, additional communication interfaces etc. To achieve this goal this project is going to use Field Programmable Gate Arrays (FPGAs) completely reprogrammable and with high capabilities.

It also allows faster update times when new functionality is developed as this functionality can be available as soon as the firmware is upgraded, in the same way as a smartphone.

1.2 OBJECTIVE

This Master Thesis should simulate a basic distributed interpolation task with Simulink, where multiple axes move in a group with distributed interpolation.

Distributed interpolation is the base of Distributed Control Systems (DCS). This is a type of automated control system that is distributed throughout a machine to provide instructions to different parts of the machine. Instead of having a centrally located device controlling all machine components, each section of a machine has its own computer that controls the operation. A DCS is commonly used in manufacturing equipment and utilizes input and output protocols to control the machine components. [1]

A concept of a control algorithm for distributed interpolation shall be developed that connects basic axis interpolation techniques with a communication concept for distributed negotiation on motion tasks. All axes should have the same set up with three basic functionalities. First, each axis issues a valid set point for each interpolation cycle. Second, the axis reacts and responds to inquiries from an external master that coordinates and negotiates a motion sequence for all axes only when all participating axes are ready for movement and can achieve the requested motion dynamics. Lastly, it communicates errors as well as a successful finish of a requested motion. Task times are crucial in this setup, as the communication/negotiation dialog needs to happen at a higher frequency as several rounds of inquiries might be needed until an agreement on the motion is ready to be executed. The result of this task should be a fully autonomously running motion execution of all axes complete with necessary negotiations and possible error states of axes, preferably paired with a behavior visualization of the separate components in order to track the internal negotiations. The modularity and extensibility of all components are presupposed.

This set up of modules should then be converted into C or VHDL Code using the Workflow described in the documentation of the MATLAB (HDL) Coder. The development process should be thoroughly documented and analyzed especially on hardships with the automated code conversion. A specific focus should be taken onto the offered top level in/outputs to the modules and how the user can configure them. The converted modules are then deployed on appropriate hardware with a sensible Input / Output visualization of the internal processes.

Algorithms for distributed interpolation need to be deployed on either Central Processing Units (CPUs) or FPGAs. As the algorithm development and validation is mostly taking place in MATLAB, it is necessary to convert the code into embedded programming languages. MATLAB offers conversion software for both cases, called MATLAB Coder and MATLAB HDL Coder, but many constraints must be considered in order to successfully convert algorithms.

1.3 SCOPE

In terms of time scope and as this thesis is developed in the context of the European exchange program (Erasmus+) its duration it is limited until end of September 2017.

The specific time flow of this project can be checked in more detail in Fig.1 with the representation of the Gantt chart:

Tasks	Duration (days)	Date of Start	Date of End	01.April	01.May	01.June	01.July	01.August	01.Sept.
State of the Art	29	01/04/2017	30/04/2017						
CNC Simulation	18	01/05/2017	19/05/2017						
Negotiation Protocol	30	20/05/2017	19/06/2017						
Distributed Simulation	30	20/06/2017	20/07/2017						
Conversion to C & HDL Code	31	20/07/2017	20/08/2017						
Implementation	20	21/08/2017	10/09/2017						
Conclusions & Summary	5	10/09/2017	15/09/2017						
Writing Final Thesis Report	14	01/09/2017	15/09/2017						
Submission Thesis Report	14	15/09/2017	29/09/2017						
Thesis Presentation	11	15/09/2017	26/09/2017						

FIGURE 1: GANTT CHART

2 STATE OF THE ART

This chapter is a summary of the state of the art of the different elements which will be used in this thesis.

As it has been mentioned in Chapter 1 a distributed control algorithm will be developed, validated and implemented. In order to obtain a successful algorithm, it is necessary to implement a proper negotiation technique, which fulfills the requirements of time and precision. In this context, a basic explanation of negotiation protocols can be found at the beginning of this chapter.

It has also been considered necessary to speak about the hardware, which will be used to achieve the goal, FPGAs. A brief explanation about what they are, why they have been chosen above microcontrollers and its coding language (HDL) will be introduced. The section will also refer to C language (the coding language for microcontrollers) because even though they are not going to be tested in this case, they are widely used in the field. Once the algorithm has been designed and validated with MATLAB, it will be converted into the embedded languages C/HDL through MATLAB Coder and HDL Coder, which will be also described.

2.1 NEGOTIATION

"A negotiation is a bargaining (give and take) process between two or more parties (each with its own aims, needs, and viewpoints) seeking to discover a common ground and reach an agreement to settle a matter of mutual concern or resolve a conflict. The noun form of the verb negotiate."

Business Dictionary, May 2017

In the context of this thesis, negotiation is carried out between the axes and the external master. The process consists of 3 steps: each axis receives a valid set point for each interpolation cycle. Later, axes communicate and answer to inquiries from the external master that coordinates and negotiates a motion sequence for all axes and orders them to move only when all are ready for movement. Third, each axis communicates possible errors as well as a successful finish of a requested motion. This process is crucial for the correct development of the project. For this reason, in this section, several architectures and negotiation techniques, which are considered of interest, are explained and context is also provided.

2.1.1 AGENT ARCHITECTURES

An Agent is a person/object which acts on behalf of another person or group. In this case, the axes and the external master are the agents of the system.

The internal architecture of an agent is essentially the description of its modules and how the selected modules are organized in relation to each other. There are some remarkable desired characteristics that these architectures should have, such as: network-centric, communicative (axes have to communicate with each other and with the master), semi-autonomous (axes are able to know if they can achieve a determined set point and communicate its decision to the master), proactive, adaptable (axes give a set point alternative if they cannot achieve the desired one), flexible, mobile, deliberative… [3]

Agent architectures can be described using two types of classification: by behavior and by internal organization. In this case, the focus will be made in the behavior organization because it adjusts better to the project characteristics:

- By behavior: [4]
  - *Deliberative:* Also called cognitive agents. They have domain knowledge and planning capability to undertake a series of actions and cooperate with other agents to achieve a task. One well-known deliberative agent architecture is the BDI (Beliefs, Desires, and Intentions). The beliefs are the expectations of the agent about a current state of the world, desire is an abstract notion which specifies preferences for future world states and intentions are what the agent is committed to doing.
  - *Reactive:* Or behavior-based architectures. They respond in an event-condition-action mode. They do not have internal models of the world. They only respond to external stimulation and the information available from the sensing of the environment. The use of symbolic representation is neglected, they only share low-level data. The deliberative reasoning is replaced by emerging behavior, which adapts to changes in the real-world environment in a timely way. It has a quick response time.
  - *Hybrid:* The most popular combination is to join deliberative and reactive types.

The architecture that will be implemented is hybrid, a mix between deliberative and reactive. The axes are deliberative because they have previous knowledge of their capabilities (position range, maximum velocity, acceleration, jerk…). But the local master is reactive, because it only gathers information from the axes and reacts to it, it does not know, which are their capabilities.

2.1.2 SYSTEM ARCHITECTURES

A multi-agent system architecture is a pattern of relationships among the agents (considering agents in the large). [3]

It is possible to classify agent-based system architectures into three categories:

1. *Hierarchical:* centralized behavior.
2. *Federated:* As it solves the problem related to the hierarchical, they are increasingly being considered as a solution. There is no explicit shared facility for storing active data. Several approaches have been considered such as the facilitator, the broker, and the mediator.
3. *Autonomous:* Any other software or a human being does not control it. It can communicate directly with any other agent of the system or external, has its own goals, and associated set of motivations.

The system architecture approach that will more likely be considered is the federated with the external master acting as a mediator. Axes respond to the requests of the master but they have their own capabilities.

2.1.3 COMMUNICATION, COORDINATION AND COOPERATION

Communication, coordination, and cooperation are three keys issues in terms of negotiation and in multi-agent systems:

Communication [3]

Communication enables agents to exchange information and to coordinate their activities. It is necessary to speak about levels of communication, modes and speech act theory.

Communication can exist in levels of sophistication as it is listed below:

- *No communication:* communication is not always necessary and it can be solved through speculation. However, if all the agents apply that method the possibilities can be infinite.
- *Message and plan passing:* agents communicate with each other sending messages. In the plan-passing approach agent A communicates its total plan to B and B does the same to A. Cooperation can be achieved but the computational cost is expensive, there is no guarantee that the resulting plan will be compatible with the recipient's knowledge. Total plans are difficult to derive in real world applications because of the existent uncertainty about the present state of the world.
- Information exchange through a *shared data repository.* The shared data repository is used for agents to write messages to or post partial results on, and obtain information from.
- *High-level of communication:* natural language understanding, speech act theory, conversations, and other formal theories. In this current case, this level of communication is needed. Axes should negotiate and constantly send messages to the master in order to achieve a successful movement.

Agent communication can be implemented in different ways depending on the nature of the agents, the global architecture of the system, timing or number of receivers of the message [3] [4]:

- *Direct/indirect:* direct communication is realized through message passing and indirect using a shared data repository.
- *Synchronous / asynchronous:* synchronous mode means working in the same location at the same time and asynchronous in the same location but at different times.
- *Single / multiple recipients:* the communication possibilities are point-to-point (one to one), broadcast (one to all others) and multicast (one to a selected group).

The system studied is direct, synchronous and multicast / point-to-point.

Many communication theories are based in Speech act theory and it is primarily concerned with natural language utterances. The categorization of speech acts by message types is shown below [5]:

- *Assertive:* "The production line has been improved."- Information.
- *Directive:* "Work to improve the production line." – Order.
- *Commissive:* "I will work to improve the production line" – Future.
- *Permissive:* "You may work to improve the production line" – Permission.
- *Prohibitive:* "The production line will be stopped next month." – Declaring decision/announcement.

To speak about communication, it is essential to understand what a conversation in terms of agents is:

"A conversation in this context, it is understood as a sequence of messages between two agents, taking place over a period of time, bounded by certain termination conditions for any given occurrence."

Shen, Weiming et al. *Multi-agent systems for concurrent intelligent design and manufacturing. [3]*

Finite State Machines, Petri Nets or Enhanced Dooley graphs, can represent a conversation as in Figure 2.

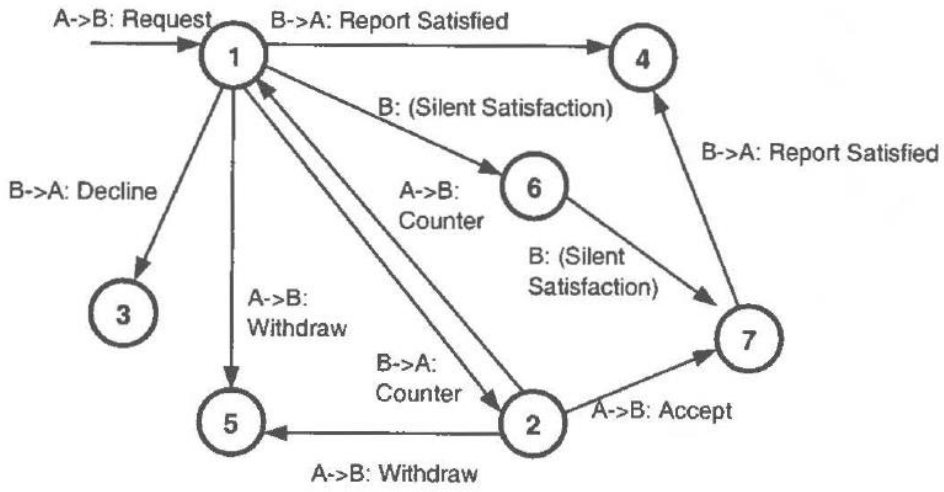


FIGURE 2: REPRESENTATION OF THE CONVERSATION POLICY FOR A REQUEST [3]

Coordination

The coordination is the process by which an agent reasons about its local actions and the anticipated action of others to try to ensure the community acts in a coherent manner. Therefore, coordination among agents is essential for achieving the goals and acting in a coherent manner. Coordination implies considering the actions of the other agents in the system when planning and executing one agent's actions. Coordination is also a means to achieve the coherent behaviour of the entire system. Coordination may imply *cooperation* and in this case the agent society works towards common goals to be achieved, but may also imply *competition*, with agents having divergent or even antagonistic goals. In our case, we can speak about cooperation because they try to achieve a same global goal, which can only be achieved working together (global Set Point). [3][6].

Cooperation

"Cooperation is the act of working together with someone or doing what they ask you."

Cambridge Dictionary, September 2017

Cooperation is necessary due to complementary abilities, to the interdependency that exists among agent actions and to the necessity to satisfy some global restrictions or criteria of success. In a cooperative model of problem solving the agents are collectively motivated or collectively interested, therefore they are working to achieve a common goal. [6]

2.1.4 NEGOTIATION AND CONFLICT RESOLUTION

The first step to resolve a conflict will be to detect it. Then, agents will use communication channels and try to eliminate them. Negotiation is done by exchanging messages among agents. Since the process involves several messages, a discussion will take place in which each agent's attitude will be an important factor. This attitude is governed by an agent's beliefs and goals and by global situation. The negotiation process follows rules, which together implement a strategy.

For instance, the master will detect that not all axes can achieve the set point given and start a negotiation with them until reaching the closest one. The axes will use their knowledge about their maximum velocity and acceleration and offer to the master a possible set point solution.

The negotiation strategies are the ones who decide the position of a particular agent in the negotiation process. In the past, agents were restricted to a fixed strategy which consisted in acting according to their beliefs and did not have a global view of the problem. The evolution moves into flexible strategies such as [3][7]:

- *Contract based negotiation:* each agent (manager), having some work to subcontract, broadcasts an offer and waits for the other agents (contractors) to send bids. After some delay, the best offers are retained and contracts are allocated to one or more contractors who process their subtasks.
- *Market-based approaches:* the goal is to solve a distributed resource allocation problem. Agents are classified as producers and consumers. Equilibrium is reached when the prices of goods are such that all resources are being used up.
- *Game theory based negotiation:* [8] the global outcome is given in a table showing the results of combined decisions. Each player, however, makes decisions independently. The key concepts are utility functions, a space of deals and strategies and negotiation protocols. One of the most well-known example of this theory is the prisoner's dilemma.
- *Plan based negotiation:* it is based on cooperation strategies for resolving conflicts among plans of a group of agents. Agents plan activities separately and, then, they coordinate their plans.
- *AI based negotiation:* as almost every human interaction requires negotiation, this new technique seems to suit perfectly.

Market-based approach and Game theory based negotiation seem to fit into the requirements. A deeper analysis of its implementation should be carried out.

2.2 FPGA

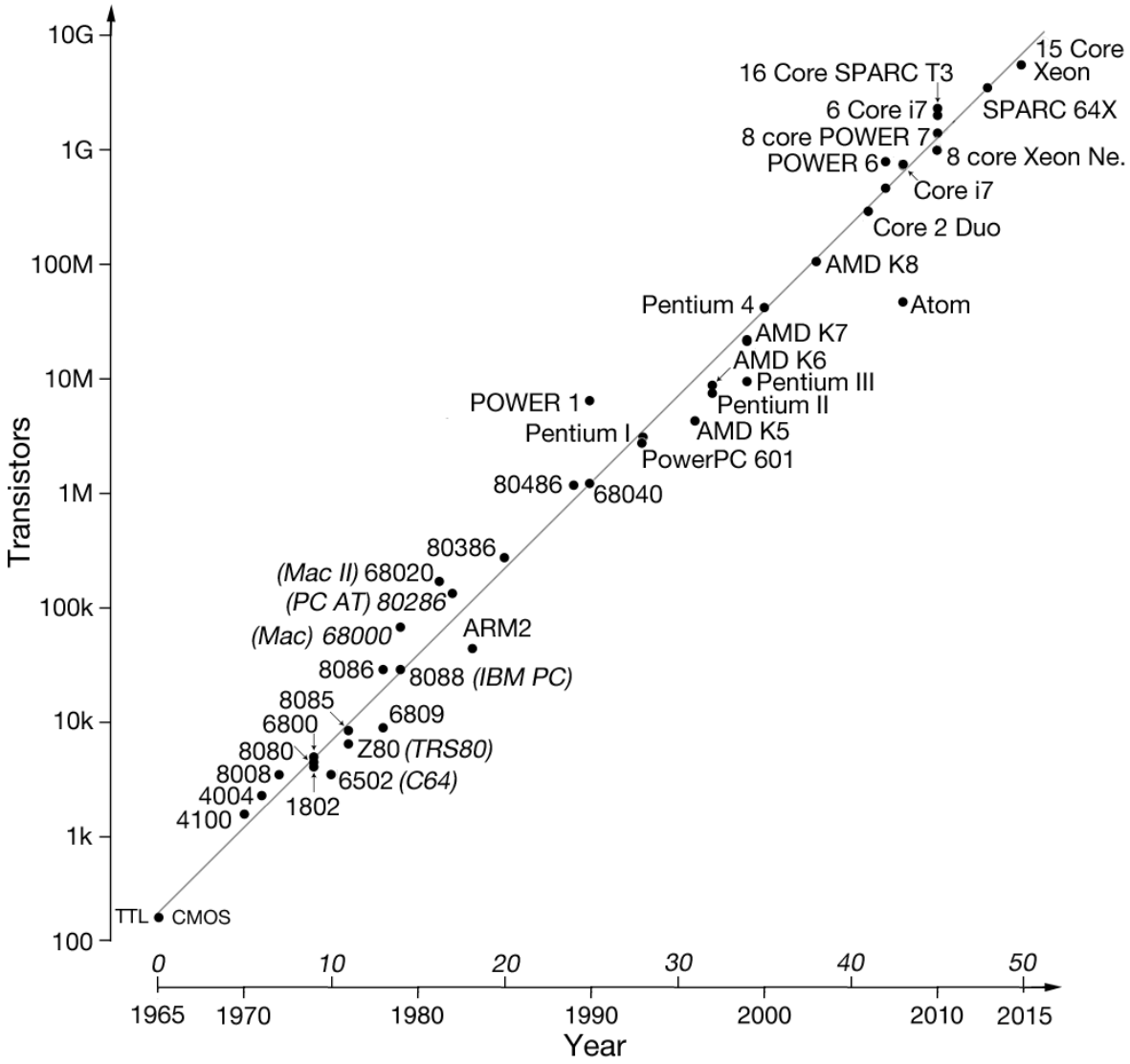
A Field Programmable Gate Array is a prefabricated silicon device that can be customized for a specific application. Unlike traditional CPUs, FPGAs are "field-programmable," meaning they can be configured by the user after manufacturing.

A better way to understand how and why the FPGAs where developed, a brief resume of electronics field development is carried out [10]:

- 1960 — MOSFET: Metal–Oxide–Semiconductor Field-Effect Transistor is one of the most basic elements in a FPGA. In the world of FPGAs, they are usually referred to as gates. They are used similarly to switches. Depending on what value they receive, 0 or 1, they will

either block or allow the flow of current. There are two main types of MOSFETs, PFETS and NFETS.

- 1961 — IC: An Integrated Circuit is a small chip that has a set circuit on it. ICs were an upgrade from using transistors to build circuits because they are generally printed. This way you can print an entire circuit in one shot, rather than having to assemble the circuit with possibly thousands of transistors. This saves time and space.
- 1962 — TTL: A Transistor-Transistor Logic is a family of ICs. Rather than being made of MOSFETs like the ICs, a TTL is made with BJTs (Bipolar Junction Transistors). The main advantage of the TTL over the IC is that it consumes less power and is less sensitive to damage from electrostatic discharge.
- 1963 — CMOS: A Complementary Metal-Oxide Semiconductor is another improvement on



the IC technology in the construction part. They reduce waste heat as there is no standing current during the transition.

- 1965 — Moore's Law: This law (see Fig. 3) is based on the prediction that the number of transistors in each circuit doubles every year. It is now used in industry and research to set goals. Unfortunately, soon Moore's Law may not hold, as technology is getting to the point where transistors are a number of silicon atoms wide.

FIGURE 3: REPRESENTATION OF MOORE'S LAW. [10]

- 1970 — PROM: Programmable Read-Only Memory was one of the first types of programmable memory. Previously there had been ROM (Read-Only Memory) where the data was set at manufacturing. Although PROM was a huge step in programmable technology, there was still improvement to be made, as it wasn't reprogrammable. Once you set the memory, it was permanent.
- 1971 — EPROM: Erasable Programmable Read-Only Memory was the second step in programmable memory, in that it could be erased. To erase an EPROM, you had to shine UV-light directly on the Eprom. However, this type of erasing was not the most practical, as it was necessary to take the EPROM out of the circuit to erase it.
- 1972 — DST: A Depleted-Substrate Transistor is similar to a MOSFET, except that it has no voltage difference from its gate to source. This provides higher gain and low noise as compared to a normal MOSFET.
- 1975 — PLA: A Programmable Logic Array is made up of programmable AND gate planes and programmable OR gate planes, connected to product a desired output. With this type of programmable logic you could most directly implement POS (Product Of Sums), OR gates then AND gates, and SOP (Sum Of Products), AND gates then OR gates, which are now only used as a formalism in design, and teaching tool. This allowed for programmable logic, but nothing near the complexity of design that FPGAs allow.
- 1978 — PAL: Programmable Array Logic is similar to PLA; however, rather than having two programmable planes, it instead has one PROM array, a fixed OR plane, and a programmable AND plane. This only allows for the sum of products logic equations with feedback from the outputs.
- 1983 — EEPROM: Electrically Erasable Programmable Read-Only Memory offered an extensive improvement from EPROM, it was erasable electrically. This allowed for data to be read, erased, and reprogrammed. However, EEPROMs only allow for a certain number of times it can be reprogrammed.
- 1983 — GAL: Generic Array Logic was a major improvement on PAL. GAL is completely erasable and re-programmable.
- 1984 — FLASH: Flash memory is a type of EEPROM, and it is the most commonly used non-volatile memory. One major improvement from the EEPROM is that FLASH memory can be erased in blocks. So, rather than having to erase all of the memory and rewrite it, you can choose what memory to erase.
- 1985 — FPGA: The first FPGA was developed.

A FPGA has an internal structure which can be divided in four main components (see Fig.4) [11] [12]:

1) Configurable Logic Blocks (CLB): Hundreds or thousands of these blocks that can be interconnected between them and the I/O blocks through the Programmable Interconnects. Inside each of them basically, there is a lookup table, one or several flip-flops with clock and reset function and the possibility to choose any output.
2) Programmable Interconnects: Programmable routing that connects these logic functions created with CLB.
3) I/O Blocks: Input / Output blocks. They are the components, which allow data transfers in and out of the FPGA. The inputs and outputs of the chip go through component groups called I/O banks, which consists of 50 individual I/O blocks. The

I/O blocks themselves are configurable in a number of ways depending on the type of data the user is either expecting to receive or transmit. These are similar to transceivers but operate at lower speeds and can maintain more functional flexibility. A simple analogy to distinguish the two would be to consider having a choice of the vehicle between a car (I/O block) and a jet (transceiver) for a commute. Even if the distance allowed to get the take off speed of the jet, it would be wildly impractical to operate.

4) Flash Memory: A configurable external Flash Memory is necessary because the FPGA is volatile, that means when the FPGA is disconnected it becomes blank again.

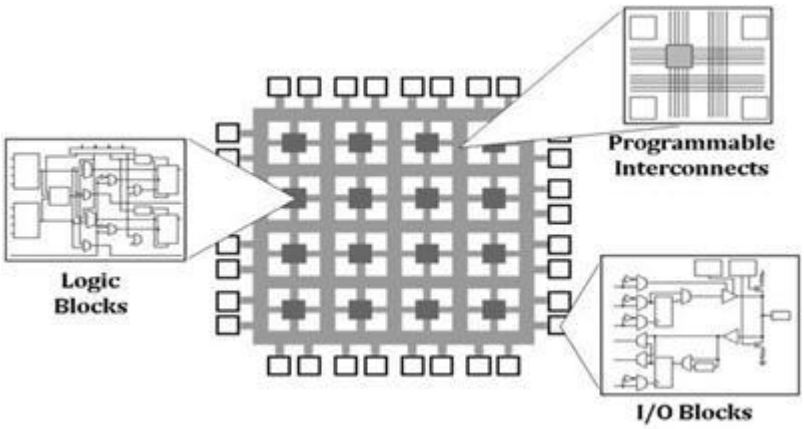


FIGURE 4: INTERNAL STRUCTURE OF AN FPGA (NATIONAL INSTRUMENTS TUTORIAL – 2010).

The advantages of a FPGA among a microcontroller are several and they are listed in the Table 1 below with the disadvantages:

Advantages	Disadvantages
Everything can be programmed.	High price.
Extremely fast.	Large use of power.
Massive parallel calculation (with separate processing).	Volatile.
High I/O count.	High pin count.
Higher memory resources	Complicated programming language (HDL).
Not fixed hardware structure	Difficult of comparison because of the lack of standardization.

TABLE 1: ADVANTAGES AND DISADVANTAGES OF A FPGA

But despite these disadvantages, FPGAs present a compelling alternative for digital system implementation of large data due to their less time to market and low volume cost and also because of their widely ranged possibilities.

2.3 HDL CODE

Hardware Description Language (HDL) [14] is a specialized computer language used to program electronic and digital logic circuits. The structure, operation, and design of the circuits are programmable using HDL. HDL includes a textual description consisting of operators, expressions, statements, inputs, and outputs. Instead of generating a computer executable file, the HDL

compilers provide a gate map understood as netlist describing the hardware (i.e., a list of gates and the wires connecting them).

The gate map obtained is then downloaded to the programming device to check the operations of the desired circuit. The language helps to describe any digital circuit in the form of structural, behavioral and gate level and it is found to be an excellent programming language for FPGAs and Complex Programmable Logic Devices (CPLDs), as it will be referred in this thesis.

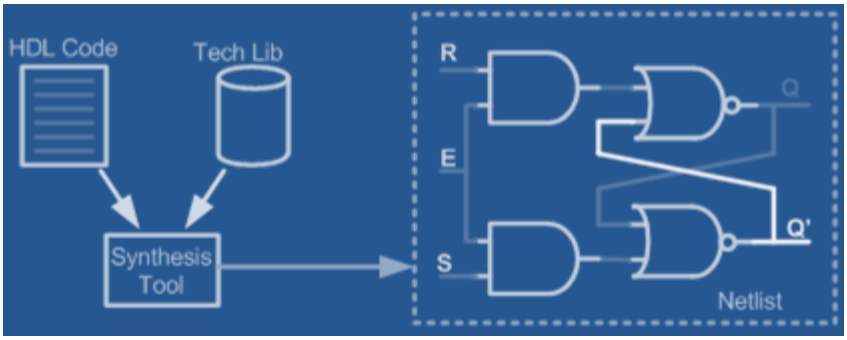


FIGURE 5: EXAMPLE OF GATE MAP (TALA, D. K.URL: HTTP://WWW.ASIC-WORLD.COM/VERILOG/SYNTHESIS1.HTML)

The three most used HDL languages are Verilog, VHDL (Very High Speed Integrated Circuit Hardware Description Language), and SystemC. Of these, SystemC is the newest. HDL code appeared as a solution to Moore's law because it supplied the absence of a better programming language allowing hardware and software co-design. Complex digital circuit designs require more time for development, synthesis, simulation and debugging. The arrival of HDLs has helped to solve this problem by allowing each module to be worked by a separate team.

All the goals like power, throughput, latency (delay), test coverage, functionality and area consumption required for a design can be known by using HDL. As a result, the designer can make the necessary engineering tradeoffs and can develop the design in a better and efficient way. Simple syntax, expressions, statements, concurrent and sequential programming is also necessary while describing the electronics circuits. All these features can be obtained by using a hardware description language. Now while comparing HDL and C languages, the major difference is that HDL provides the timing information of a design.

In most of the industries, Verilog and VHDL are the most common ones. Verilog consists of modules and the language allows Behavioral, Dataflow, and Structural Description. On the other hand, VHDL's design is composed of entities consisting of multiple architectures. SystemC consist a set of C++ classes and macros.

The designing of an entire application in HDL can be very difficult and time-consuming because of the typical level of abstraction of the high-level languages. Also, a program made in HDL can be complex to modify. For these reasons, more people are using applications to convert the code to HDL as the HDL Coder that will be used in this thesis.

In this project, reference will be made only to Verilog and VHDL because the programming of HDL code it is going to be made through the conversion of MATLAB code to HDL code using the MATLAB HDL Coder which offers the possibility to choose one of these two embedded languages.

2.4 MATLAB

MATLAB is a commercial software developed by the American company MathWorks to solve mathematical problems and to graphically display the results. It is also the software chosen for the implementation of that project. When the simulation is already developed in MATLAB it needs to be converted into embedded languages such as C and HDL in order to start the hardware implementation. During this conversion, it is important to check the fixed and float points that will be described in the first part of this section. Later, a brief summary is done on how to use/operate the conversion applications provided by MATLAB to pass from a Simulink model to C or HDL code.

2.4.1 FIXED POINT VS FLOATING POINT

Digital signal processing can be separated into two categories: fixed point and floating point. These designations refer to the format used to store and manipulate numeric representations of data. Fixed-point DSPs are designed to represent and manipulate integers – positive and negative whole numbers – via a minimum of 16 bits. Floating-point DSPs represent and manipulate rational numbers via a minimum of 32 bits in a manner similar to scientific notation, where a number is represented with a mantissa and an exponent (e.g., A·2B, where 'An' is the mantissa and 'B' is the exponent). [15]

The exponentiation inherent in floating-point computation assures a much larger dynamic range – the largest and smallest numbers that can be represented - which is especially important when processing extremely large data sets or data sets where the range may be unpredictable. It is not the case, because the path is decided and predictable (it has to be inside the position range of the axes) and the data sets are not so big.

Floating-point processing has much greater precision than fixed-point processing, distinguishing floating-point processors as the ideal DSP when computational accuracy is a critical requirement.

Dynamic range and precision considerations typically define the criteria used by designers to determine whether fixed-point or floating-point processors are ideally suited for an application – where computational demands are high, floating point is favored. For general purpose applications with a high-volume fixed point are chosen.

However, the design will be converted to Fixed-Point because HDL Coder requires it to work but the system should not have any problem because it is not such a big application. A summary of the different characteristics explained can be seen in Table 2 [16]:

Fixed Point	Floating Point
16- or 24-bit Devices	32-bit Devices
Limited Dynamic Range	Large Dynamic Range
Overflow and quantization errors must be resolved	Easier to program since no scale is required
Long product development time	Quick time-to-market
Faster clock rate	Slower clock rate
Cheaper	More expensive
Lower power consumption	Higher power consumption

TABLE 2: COMPARISON BETWEEN FIXED POINT AND FLOATING POINT

2.4.2 MATLAB TO C

C, is a programming language of general purpose and high-level, originally developed by Dennis M. Ritchie to evolve the UNIX Operating system. It is a successor of B language introduced around 1970 but it was not formalized until 1988 by the American National Standard Institute (ANSI). Nowadays is the most widely System Programming Language. [17]

C was initially used for system development work, particularly the programs that make-up the operating system. It was later adopted as a system development language because it produces code that runs nearly as fast as the code written in assembly language. In our case of study, the conversion from MATLAB to C is used to implement the system into CPUs.

To accomplish this goal from a developed APP, MATLAB offers two options. The first one is to convert the entire APP directly using MATLAB compiler. In this case, a short library which will need MATLAB runtime to work is created. The second option is to convert only the code, this can be done automatically using MATLAB coder and the result will be independent of MATLAB.

A manual translation is not advisable because MATLAB and C are two completely different languages. It is not an easy process and there are high possibilities of losing valuable time. It will be hard to modify the requirements later and it will also have coding errors. Summing up, it is a time-consuming and expensive process.

MATLAB Coder helps to do this process automatically but it is necessary to follow a specific workflow [18] and taking into account some requirements:

1. *Prepare:* In this step, it is necessary to prepare the MATLAB algorithm for code generation through implementation choices and the usage of supported language features.
2. *Test:* Verify if MATLAB code is ready for code generation. First of all, validate that MATLAB program generates code: if it is successful MATLAB will generate an .m file. Then, accelerate execution of the user-written algorithm.
3. *Generate:* It can generate source code or MEX for final use. Iterate your MATLAB program to optimize and implement as a source, executable or library.

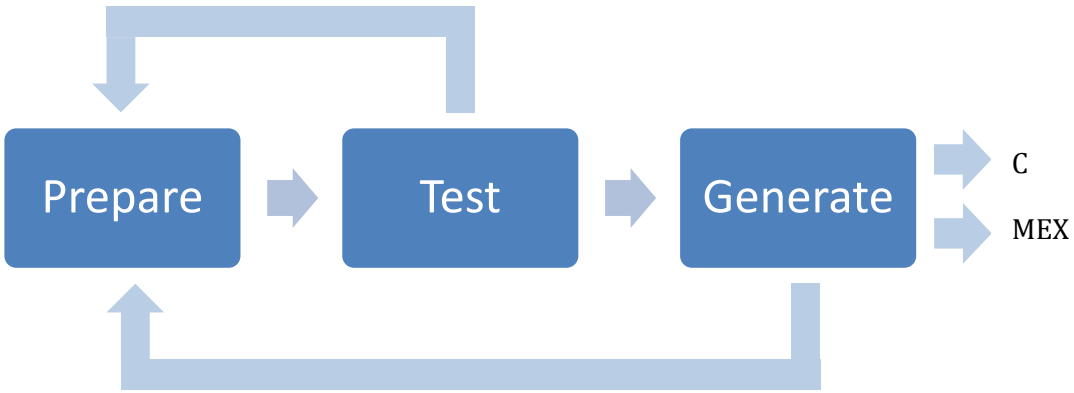


FIGURE 6: THREE-STEP WORKFLOW

During this implementation, it is important to look at 4 different points that differ from MATLAB to C:

- *Polymorphism: the* same function of MATLAB code can be used for an element by element multiplication (with logical, integer, real, complex…), matrix multiplication or dot product. However, for C code it is needed to create a specific function for each operation and to specify the entry types (see Fig. 7)[18]:

MATLAB

```
function a = foo(b,c)
a = b*c;
```

C element multiplication →

```
double foo (double b, double c)
{
return b*c;
}
```

C matrix multiplication ↓

```
Void foo (const double b[15], const double c[30], double a[18])
{
        int i0, i1, i2;
        for (i0=0; i0<3; i0++){
                for (i1=0; i1<6; i1++){
                        a[i0 + 3*i1]=0.0;
                        for (i2=0; i2<5; i2++){
                                a[i0 + 3*i1]+=b[i0 + 3*i2]*c[i2 + 5*i1];
                        }
                }
        }
}
```

FIGURE 7: EXAMPLE OF POLYMORPHISM

- *Memory allocation:* it is mandatory to specify if an element is statically or dynamically allocated in the memory. MATLAB is a dynamical type, which means you do never have to worry declaring memory for the data that you create. On the other hand, C is a statically type so you always have to explicitly specify the size of your data or otherwise you have to use dynamic memory allocation using specific commands.
- *Processing matrices and arrays:* In arrays/matrixes MATLAB used compact whereas C code uses for loops. Another difference is that MATLAB is 0 based indexed and C is 1 based indexed.
- *Fixed-point data types:* in C is necessary to specify the data types of your inputs and outputs as well as the internal variables.

These considerations can mean a relevant increase in the number of lines comparing C code to MATLAB Code.

2.4.3 MATLAB TO HDL CODE

This conversion is made using HDL Coder but it is also necessary to follow Model-Based Design to make it work properly.

Model-Based Design is a mechanism that will show how to bridge the gap between the algorithmic description and the hardware implementation.

One easy way to verify that hardware implementation matches system specifications is using Model-Based Design and HDL Coder (MATLAB application) together. It can also be useful to iterate designs faster.

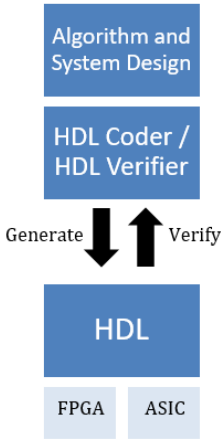


FIGURE 8: MODEL-BASED DESIGN WORKFLOW

There are few steps [19] from algorithm through Hardware implementation that will make the process successful:

- The author writes in MATLAB already thinking in the ultimate Hardware implementation. During this thesis, the author will try to clarify which are the main issues that have to be taken into account before starting to implement a simulation that will be converted to HDL Code using Matlab/Simulink.
- Leverage HDL coder workflow: launch the HDL Workflow Advisor and follow the steps:
    1. *Definition of Input types:* it can be done by running a floating-point simulation using the Test Bench or manually defining variable by variable.
    2. *Fixed-Point conversion:* Floating to Fix-Point validation which defines the simulation maximum and minimum values. These values and the proposed fix-point results can be easily modified and iterated again.
    3. *HDL code generation:* it can be chosen between VHDL or Verilog language and the coding standards.
    4. *HDL verification:* it can be done through 3 different ways:
        - HDL test bench
        - With Cosimulation: The signals can be checked graphically with ModelSim.
        - With FPGA-in-the-loop: Using MATLAB against a real FPGA board. This option will be the one implemented in this project in 4.3 Implementation.
    5. *Synthesis and Analysis:* to get an accurate picture of the results. The design can be integrated into a larger project using Altera. This step is not considered in the scope of this project.
- Use synthesis and implementations tools.

2.4.4 USING HDL CODER AND HDL VERIFIER FOR FPGA AND ASIC DESIGNS

There are a few practices [20] that should be followed:

1. Use modeling and simulation to optimize at the system level:
    - Convert floating point to optimized fixed-point model: it will be done by automatically tracking of signal range (also in intermediate quantities) and with that propose to make a recommendation on word/fraction lengths.

- Bit-true models in the same environment: as a method to be able to quantify the error occurred during the conversion.

2. Automatically generate readable, traceable HDL code for FPGA and ASIC designs:
   - In a Simulink model, open HDL Workflow Advisor, set Target Device as Generic ASIC/FPGA and choose the synthesis tool for your specific FPGA target.
   - Execute all the next steps to generate the code (Prepare Model for HDL Code Generation, HDL Code Generation and FPGA Synthesis and Analysis) where options such as the programming language, results presentation and others can be chosen.
   - The report generated can be useful for analyzing the number of resources used in the implementation, for instance, the number multipliers, multiplexers, registers… that have been created and in which sections. This report will be useful in order to give a reasoned conclusion of the utility of this HDL conversion. It is important to know if the converted code employs many resources and how are they used because this project should not need a high number of them.
   - Optimization adds Path Delay with should be checked in order not to affect badly to the simulation.
   - Use pipeline to improve the speed and identify the true bottlenecks of the system through highlighting the critical paths.

3. Re-use System Level Test Bench for HDL Verification: the problem with Test Bench is the difficulty of re-using them between different levels of abstraction (Model, VHDL, FPGA). For making that possible HDL Cosimulation can be really helpful.

   - Open HDL Workflow Advisor: in HDL Code Generation, use Generate Cosimulation and run the test.

4. Enable regression testing with FPGA-in-the-loop simulation:

   - Re-use test benches for regression testing.
   - Inside HDL Workflow Advisor – FPGA Turnkey.

3 SYSTEM MODELLING

This chapter is showing the simulations implemented with MATLAB/Simulink during the development of the thesis. These simulations represent the environment that would be used in a real project and try to be visual and easy to understand, apart from providing significant data. The system simulation has been evolved from a first centralized system where only a representation of the CNC machine workflow until getting the final system, decentralized, where the axes should accomplish 3 basic functions: each axis has a valid set point for each interpolation cycle; second, they negotiate a motion sequence with an external master; third, they communicate errors or the end of a successful movement.

3.1 CNC MACHINE SIMULATION

In a CNC machine, the Master has as an input a set of points, it performs a path planning limiting the top velocities or accelerations of the axes. At that point, it sends the points of the planned path to the axes, which just make a simple linear interpolation between them before moving. This system is centralized because all the information is in possession of the master and it is also the one which gives the orders.

The layout can be seen in the Fig. 12 and it will be explained more in detail during this chapter. Some constant values, as the frequencies at which the simulation was running, needed to be defined and they were chosen to be kept as accurate as possible compared to a real CNC machine.

Variable	Value
Communication frequency	f = 1000 Hz
Interpolation frequency	f_{int} = 16000 Hz
Global frequency	f_g = f/20 Hz
Simulation time	simt = 12/f_g Hz

TABLE 3: VALUES ARE GIVEN TO THE FREQUENCY VARIABLES

The simulation has 3 different parts: the generation of the desired trajectory, the master system, and the slaves X and Y. Only two axes have been implemented in order to make the system easier to understand but adding a third one should not be a problem, it would only be necessary to add another Lookup table.

- Desired trajectory: the first part of the system is the generation of the trajectory. The path is determined using two lookup tables, one for each axis. The Set Points resulted from the lookup tables are in Cartesian coordinates.
  The outputs, *x* and *y*, are the inputs of the Master system. This system gives a Set Point every f_g , which is controlled by the Rate Transition block named “RT fg1”.

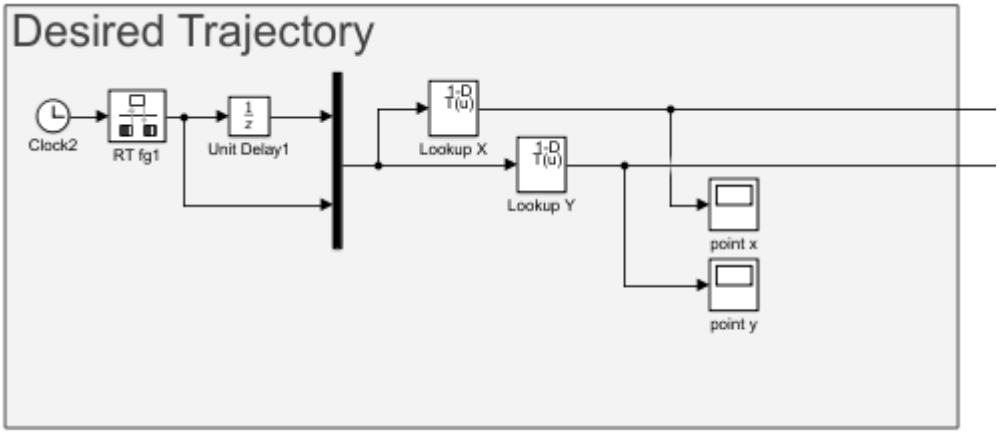


FIGURE 9: DESIRED TRAJECTORY

- Master subsystem: this system is in charge of the interpolation process and is the one with all the information about the axes (maximum position range, velocity, acceleration…), the intelligent part of the system. When it receives the Set Points, it applies a 7th degree polynomial interpolation to create a path and divide it into small Set Points that are going to be sent to the axes. The use of this high degree polynomial is justified by the search of a smooth trajectory, where the axes have no issues in the transition from one Set Point to the other. This polynomial secures a velocity, acceleration, and jerk equal to zero at the end of every Set Point. The Set Point given by the trajectory generator has to be divided into small ones because of the limitations, it can create a path that does not imply breaking them by imposing them as a condition. However, it was decided not to apply any limitation in this early stage of the system modelling.
  This system works at two different frequencies, every Set Point is given at a f_g frequency and it generates a new small Set Point at f frequency, as it can be seen by the inputs of the both Rate Transitions blocks.

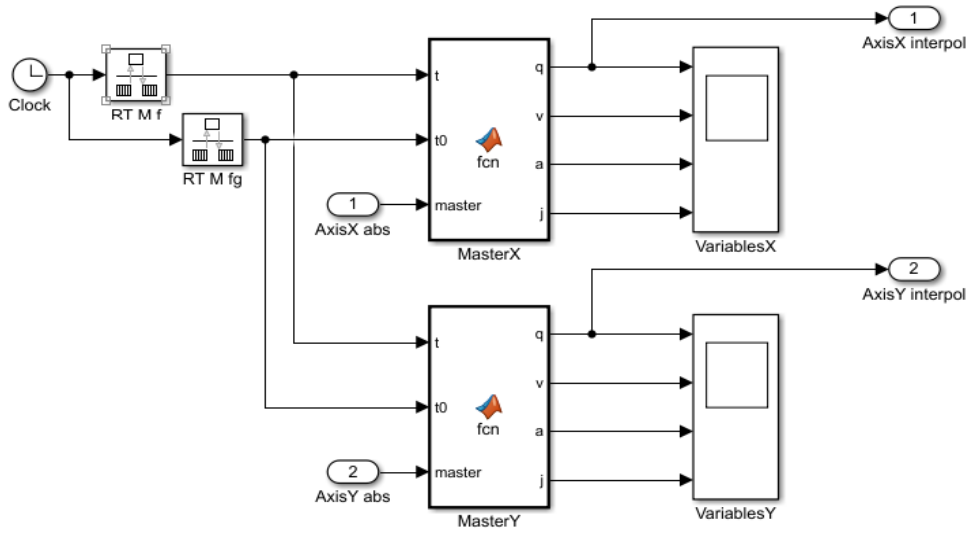


FIGURE 10: LAYOUT OF THE MASTER SYSTEM

- Slave X/Y and final transformation: The slaves receive the coordinates given by the master and apply a linear interpolation (Subsystem Axis X/Y) in order to calculate the velocity and acceleration that they need to accomplish the Set Point given. They would never exceed their limitations because the master has already made the calculations for them, this is the main difference from the distributed interpolation simulation that will be implemented after.
  After that, the output position q_x, which is the desired position, enters in the subsystem X (layout in Fig. 11). This subsystem makes the conversion into the real position applying the saturation limits in acceleration, velocity, and position and the transfer function.

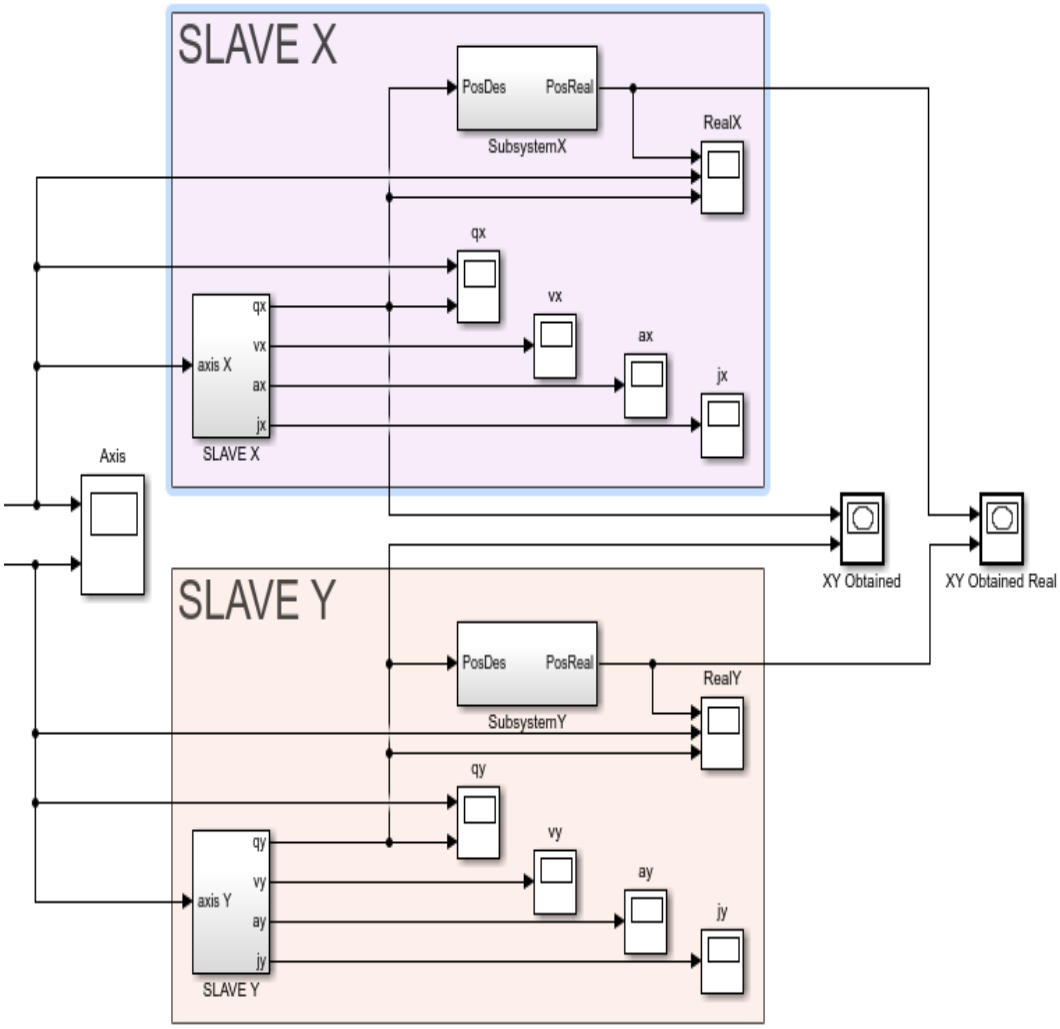


FIGURE 11: LAYOUT SLAVES

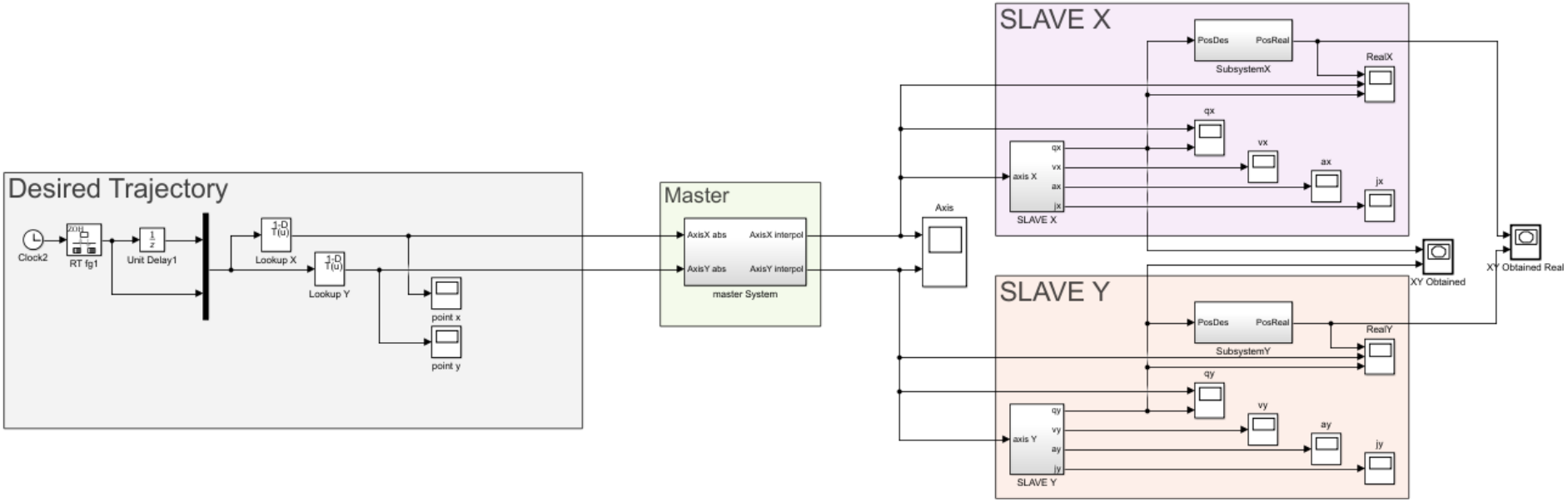


FIGURE 12: LAYOUT OF CNC SIMULATION

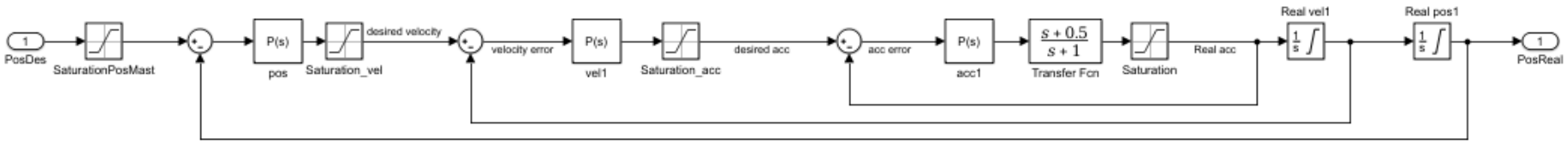


FIGURE 13: LAYOUT OF THE SUBSYSTEM X (SAME AS SUBSYSTEM Y).

The results of the real system can be seen in the Figure 14 and 15 below. The velocity (limited at 1 m/s and position of x and y is limited to 0.15 m are affected for the saturation for this trajectory. Hence, there is no difference in the position between the path followed by the desired trajectory and the real trajectory. In the Fig. 14 we can see the desired position, which is defined until 0.2 m and in the Fig.15 the error between both of them, which is noticeable after the system goes over the 0.15m, until 0.05m.

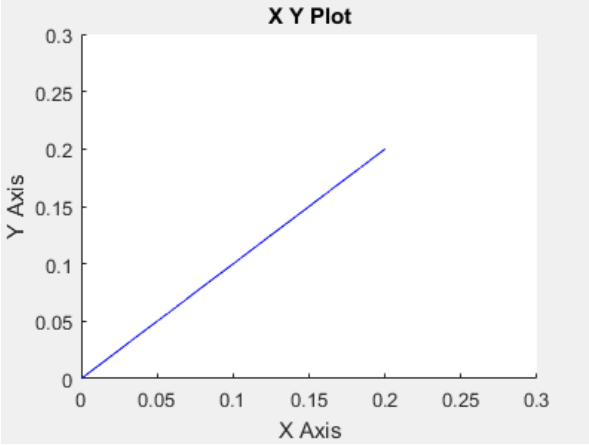


FIGURE 14: DESIRED POSITION

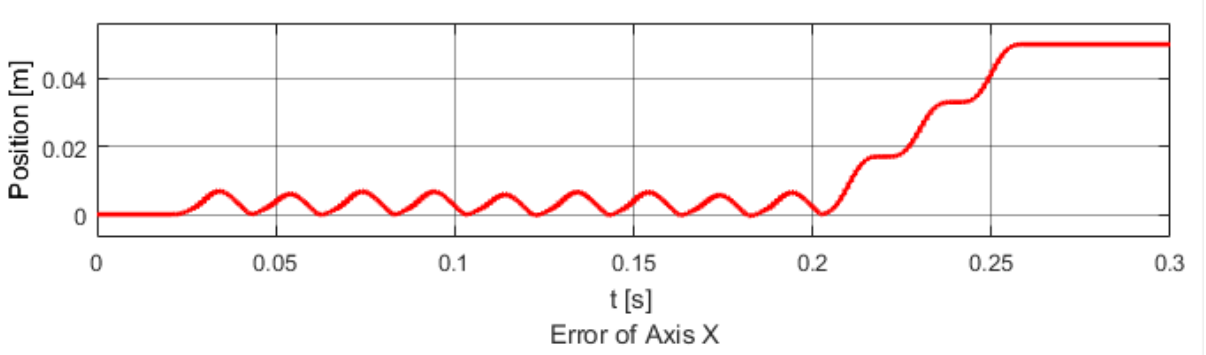


FIGURE 15: ERROR OF THE AXIS X

More representative plots are drawn in the Fig.16, where it is possible to see more deeply the evolution of the position, velocity and acceleration and how the velocity and acceleration are reduced to zero after the maximum position (the saturation at 0.15m) is reached.

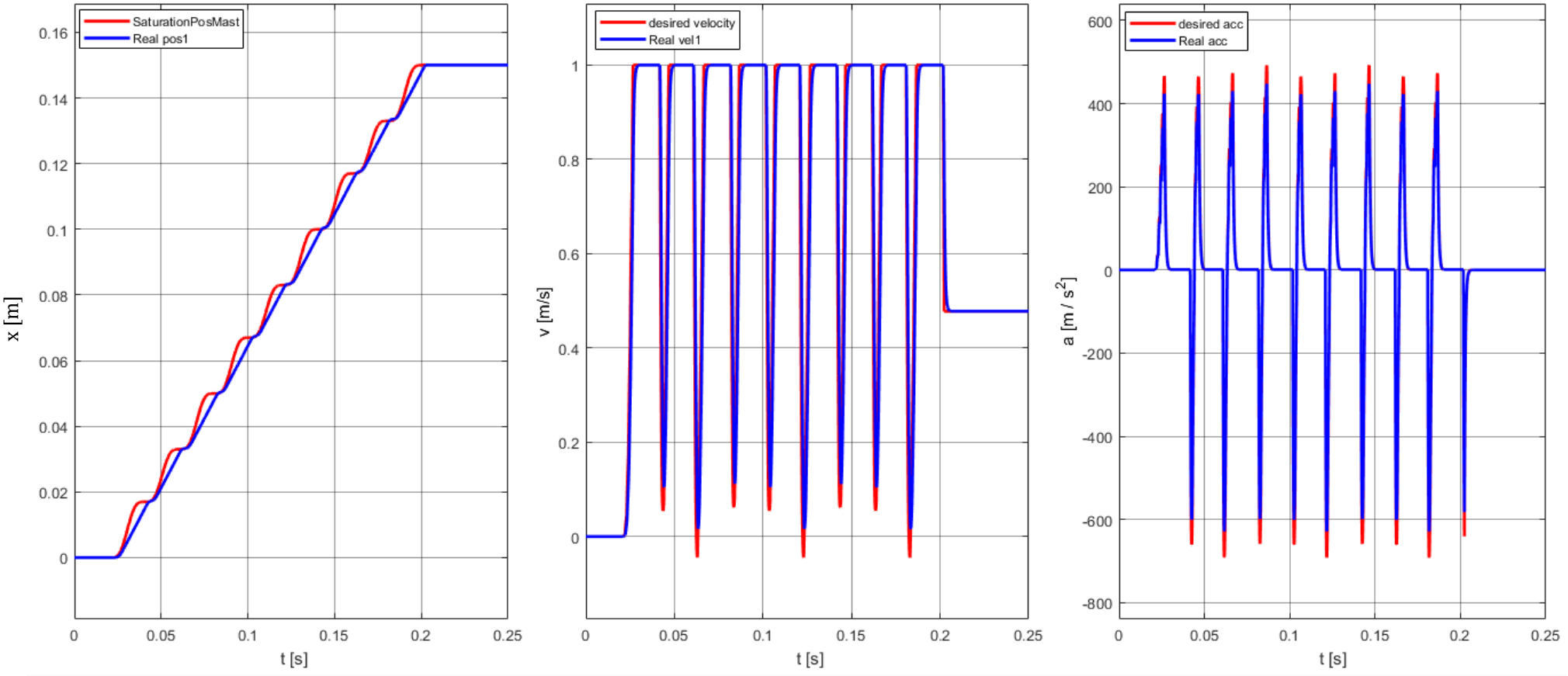


FIGURE 16: EVOLUTION OF POSITION, VELOCITY AND ACCELERATION

3.2 NEGOTIATION PROTOCOL

For the final simulation, a negotiation protocol has been implemented which explains how/when the axes communicate with each other and with the master. The goal of this protocol was to reduce the communication at a minimum level but maintain its effectiveness. All the necessary communication until reaching an agreement should be encapsulated in the time period of a master process.

In order to develop this distributed principle, it has been considered necessary to create seven State Machines which will coordinate the actions: each axis has two State Machines, one for the communication and another one for motion. In addition, one of the axis can be randomly chosen as a Local Master. In order to perform this task, that axis will have a third State Machine to control and supervise the whole system. The functions of that machines are described below:

- Global State Machine (GSM): located in the axis chosen as the master. It communicates with the COMs and the global master. Its task consist in gathering the information given by the global master (the goal Set Points) and transmit this information to each axis. Axes will calculate if they can achieve this Set Point and send this information back to the GSM that will be in the lead of the negotiation.

- Communication State Machine (COM): it is located in every axis and can communicate with the Motion State Machine of the same axis and with the GSM. This machine receives the goal SetPoint and gives a positive or negative answer back to the GSM depending on whether it can or cannot reach the goal. It also transmits the OK to start the motion and the Set Point negotiated to the Motion State Machine when the GSM gives to it the validation.

- Motion State Machine (MOT): located on every axis, it has a single communication channel with the COM of its axis. Its function is to give the order of movement to the axis motor and communicate the COM machine when the movement has finished or there is any error.

The communication through these machines is made with 3 different types of messages which have an agreed format:

1. **Message Type 1:** manages the communication between the global master and the GSM.

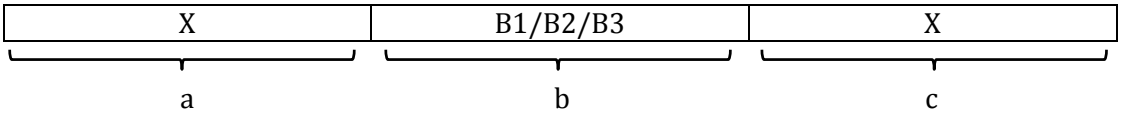


FIGURE 17: MESSAGE TYPE 1

a: Sender address – in order to know who sends the message, each participant is identified with a number. In this case, 0 (Global Master) and 1 (GSM).
b: Target / SetPoint goal – B1 refers to Axis 1, B2 to Axis 2 and B3 to Axis 3.
c: State status – gives necessary information about the condition of the message. 0 (all in order), 1 (Error), 3 (Confirmation of lecture).

2. **Message Type 2:** message used in the communication between GSM and COMs.

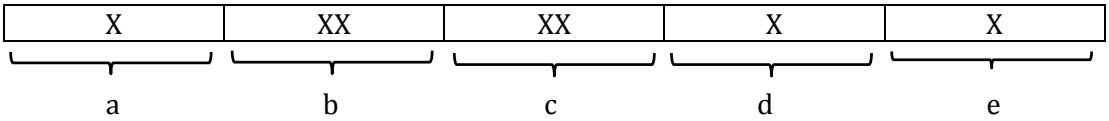


FIGURE 18: MESSAGE TYPE 2

a: Sender address. 0 (GSM), 1 (COM axis 1), 2 (COM axis 2), 3 (COM axis 3).
b: Target / SetPoint goal.
c: Maximum Point that can be reached by axes.
d: State status. 0 (all in order), 1 (Error), 2 (NOK – axis can't reach the SetPoint goal), 5 (Communication has exit Ready State).
e: Actual Position

3. **Message Type 3:** it corresponds to the communication between COMs and MOTs.

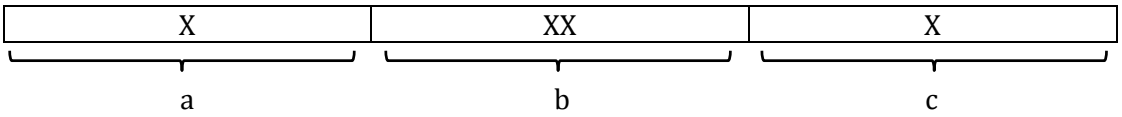


FIGURE 19: MESSAGE TYPE 3

a: Sender address. In this case, 0 (MOT) and 1 (COM).
b: Target / SetPoint goal.
c: State status. 0 (OK), 1 (Error).

In the next Figure 20, it can be observed more clearly how is the topology of the system and which communications are internal or external, which implies a different time of communication.

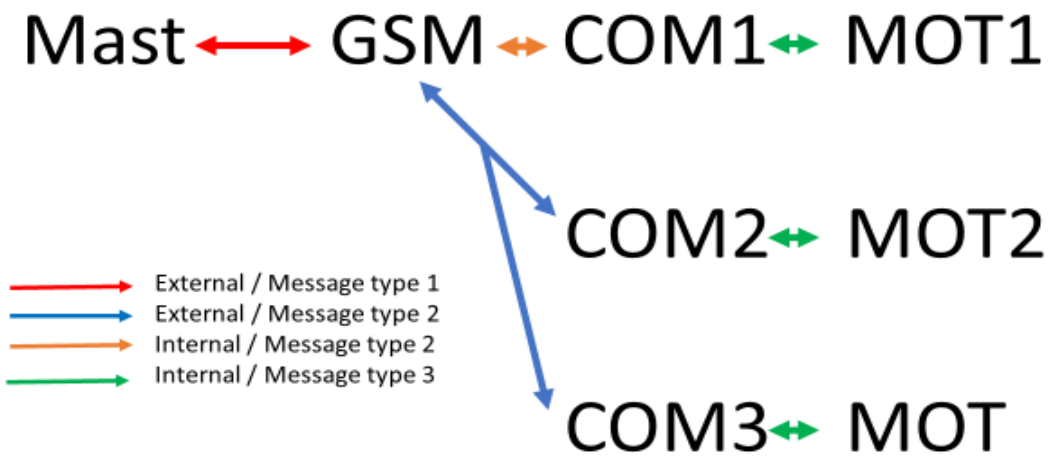


FIGURE 20: TOPOLOGY OF THE SYSTEM

Before explaining in more detail how this protocol works, it is necessary to enumerate the assumptions that have been considered:

- Global Master is always working properly. On this basis, there are no communication problems related to it.
- All axes will start to move synchronously because a cyclic state machine[1] will control their movement.
- The communication during RUN state (when the axes are moving) will be checked by the cyclic state machine mentioned above.
- A communication problem is possible between the axes.
- Internal communications are reliable.
- Communication takes 10% of the computation frequency if it is external and 5% if it is internal.
- Movement is defined as 20 times the operation time of axes
- More than one SetPoint is sent to Local Master.
- Negotiation process begins when the previous movement has started.

During the conception of this protocol, it has also been considered necessary to define the possible problems that may occur in a real system and how the system is going to confront them.

- Lack of answer: it can happen between the COMs and the GSM. The system has a lack of answer if the AnswerTime[2] is past and no answer is received. If this event occurs the GSM will resend the SetPoint two times, if there is still no answer, the system will understand that there is an error and all state machines will go to Error state in order to prevent the system from a higher damage.

- Loss of communication: Axes do not communicate to the cyclic state machine during moving phase. Axes will move to Error state.

- Mechanical Error: This error can happen during the moving phase and, therefore, axes will also move to Error state to stop the system.

The different states of the State Machines have been defined as well as the transitions between them through State Diagrams (default State appears in blue) as it can be seen below:

[1] The cyclic state machine is just described in a functional way but not developed in this thesis because it corresponds to another part of the DEVEKOS project.
[2] Answer Time: defined as the time to get an answer after having sent a SetPoint from the GSM. AT = communication time + computation time + communication time.

COM:

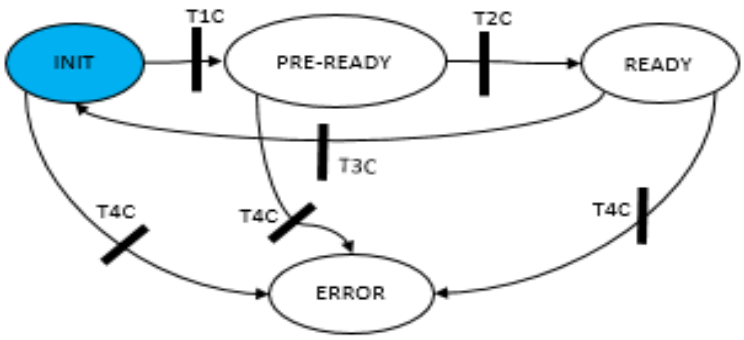


FIGURE 21: STATE DIAGRAM OF COM

Transitions	Meaning
T1C	SetPoint received.
T2C	OK received from GSM & Interp done & Movement finished.
T3C	SetPoint sent to Motion and answer from motion indicating beginning.
T4C	Error message received (from Motion or Communication).*

TABLE 4: EXPLANATION OF THE TRANSITIONS OF COM MACHINE

During the PRE-READY state the interpolation of the SetPoint will be made to check if the axis can or can't reach the goal.

*If the Error message arrives from Motion (MOT) it has to be sent to the GSM and if it comes from GSM, COM has to inform the MOT in order to stop the axis.

MOT:

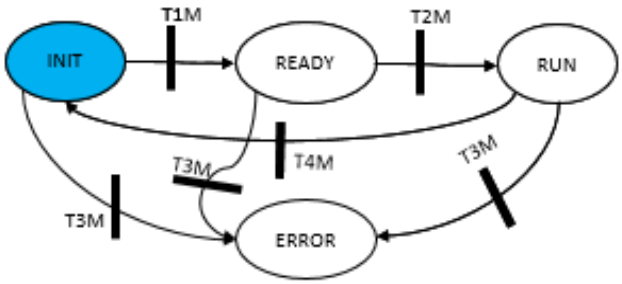


FIGURE 22: STATE DIAGRAM OF MOT

Transitions	Meaning
T1M	Path received (T3C) & Axis OK.
T2M	Axes coordinated.
T3M	Error message received OR error from the axis.
T4M	Movement is finished & message OK to Communication is sent.

TABLE 5: EXPLANATION OF THE TRANSITION OF MOT MACHINE

The RUN state implies the movement of the axis. The transition T2M is managed by the cyclic state machine.

If the ERROR message is received from the COM, MOT State Machine will stop the axis. Otherwise, if it is internal it will stop and also send a message to the COM.

GSM:

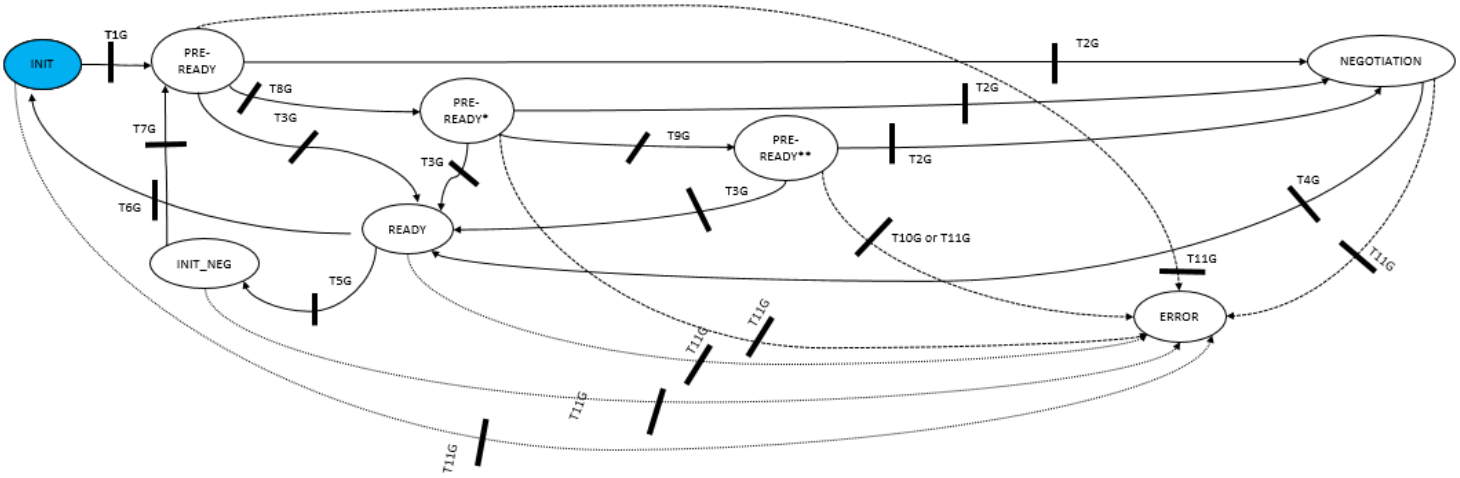


FIGURE 23: STATE DIAGRAM OF GSM INCLUDING THE RETRIES

Transitions	Meaning
T1G	SetPoint received from global Master & Confirmation of Lecture sent.
T2G	If one/more axes are NOK with the SetPoint.
T3G	All axes are OK with the new SetPoint (setpoint sent).
T4G	OK, confirmation from Axes (set n=1).
T5G	Time for new SetPoint & n = 1.
T6G	Time for new SetPoint & n = 0.
T7G	Verify SetPoint and path are not the same.
T8G	AnswerTime is passed and no answer (Set c = 1)
T9G	AnswerTime is passed and no answer (Set c = 2)
T10G	AnswerTime is passed and no answer.
T11G	Error message from Local Axes.

TABLE 6: EXPLANATION OF THE TRANSITIONS OF GSM

It is worth to point out that every state can jump to the Error State because the GSM is managing the whole system, therefore the error can be related to any of the axis state machines and the GSM need to inform the other axes.

As this protocol contains a big amount of information, which could be difficult to process, different time diagrams considering several possibilities has been developed. Due to its large dimensions, it has not been considered appropriated to add it here but it can be found under the name 'NegotiationProtocol.xslsx' in the same folder of this document. In this file, it is possible to verify that the communication has been kept under the limits (less than the 1Khz frequency of the Global Master), concretely 0.000924 s in the longest possibility considered, which corresponds to a negotiation with 2 retries.

3.3 DISTRIBUTED SIMULATION

Once the negotiation protocol is defined, the simulation has been implemented using the MATLAB tool Simulink. During this section the ins and outs of this simulation will be developed in order to allow the readers understand it.

3.3.1 EXPLANATION OF THE SIMULATION

First of all, it is appropriate to make a small summary of the desired behavior. In the real workflow, the Set Points (path to follow) will be generated by a Global Master, which will send them to a Local Master inside one of the Axis (chosen by the user / randomly), this Local Master will make a first and smooth interpolation and generate small Set Points which will be sent to all the axis, including itself. Then, the axes will check if they can reach the goal Set Point and answer positively. If they cannot, the negotiation will take place and finally the GSM will resend a new goal Set Point reachable by all the axes along with a motion order. After, it checks if the system has evolved correctly.

With the above workflow in mind, one can make a quick view of the layout of the simulation that can be found in Fig. 27. The simulation starts with the generation of the Set Point (at the left corner). The Set Points are then connected to the Inside Local Master (red in the center). These 3 functions generate the small set points and should be located inside the GSM State Machine. This cannot be done because State Machines are implemented using State Flow chars which do not admit MATLAB functions external to them. Their outputs are connected to Axis 1, which has been assigned as Local Master. The first part makes the first interpolation and sends the generated small Set Points to the second part, the Axis 1, where all three State Machines are implemented (GSM, COM, and MOT).

The communication between all the elements (Global Master <–> Local Master, Local Master <–> Other Axes…) is carried out by the messages defined Negotiation protocol (see section 3.2). The messages are not intermittent because they are represented by signals that have always have a value. In order to know when the System has to look at them, the simulation is governed by triggers, which is the representation of the communication times. There are two types of trigger, named ‘TrigBig” and “TrigMin”. The “TrigBig” is activated every time that the Local Master has verified that the System has realized a Set Point and needs that the Global Master provides a new one. The “TrigMin” refers to the small Set Points, product of the first interpolation realized by the Local Master.

Following the first peek of the system, a deeper explanation of every part will be performed in the next parragraphs.

The first task should be carried by an external Global Master (blue sections). The Set Points in the simulation are created by a MATLAB function (Fig. 25), which divides them in Cartesian global coordinates (X,Y,Z). The Memory Block is used to remember the last Set Point given and create a new in reference to that one. The path created has ups and downs in order to test that the simulation works properly in both cases.

The Global Master only communicates with the Local Master and it uses the Message Type 1, which apart from Set Point it also requires sending the Sender and the Status. In this simulation is

considered that the global master does not have any error. Consequently, the Status and the Sender are generated as constant values (see Fig. 26).

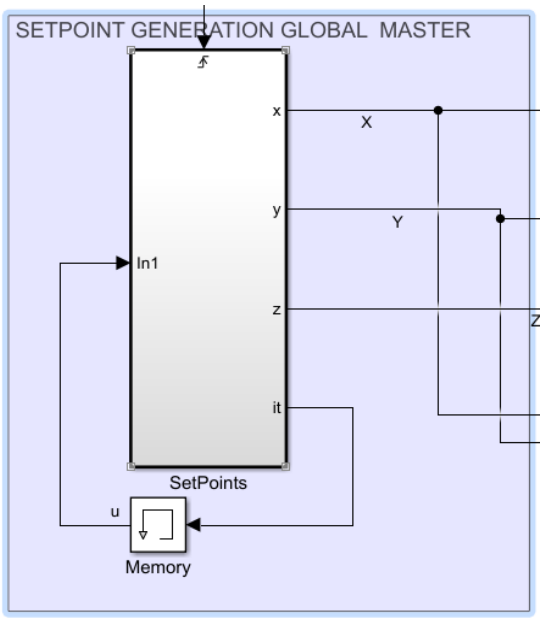


FIGURE 24: SUBSYSTEM SET POINT GENERATION

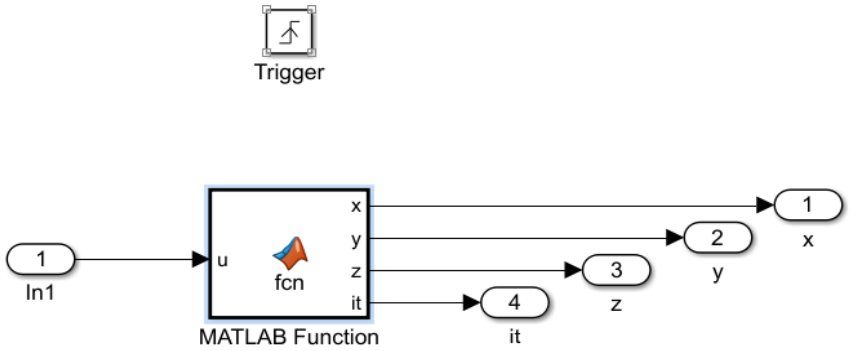


FIGURE 25: VIEW OF THE INSIDE OF THE SUBSYSTEM

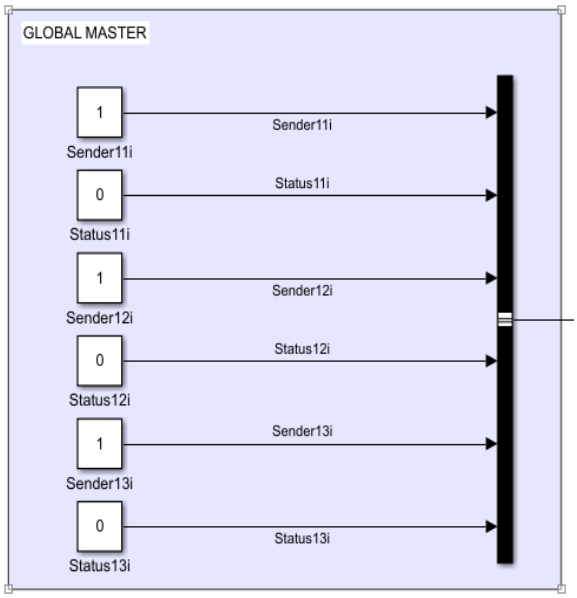


FIGURE 26: SENDER AND STATUS OF GLOBAL MASTER

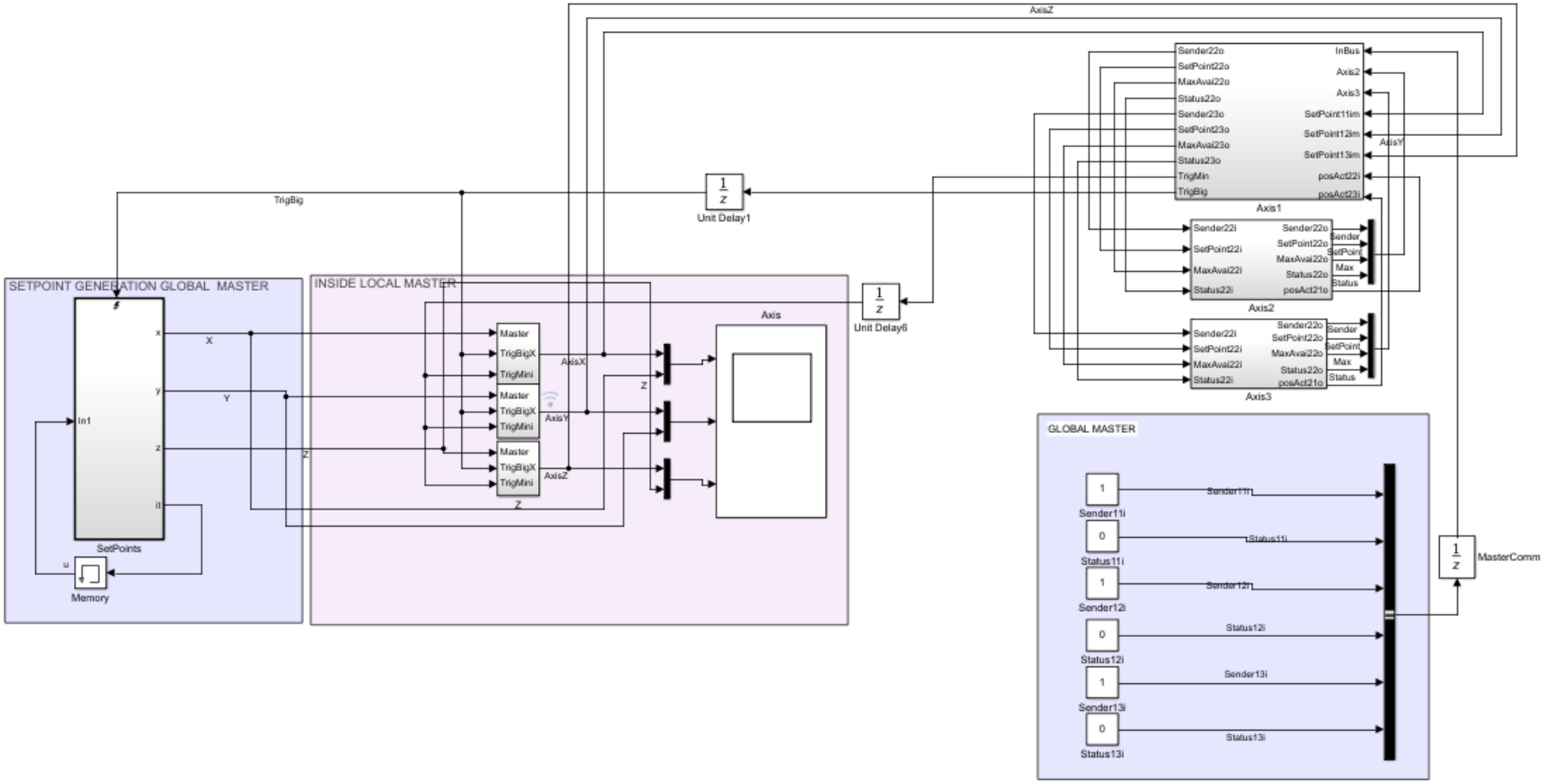


FIGURE 27: LAYOUT DISTRIBUTED SIMULATION

These Set Points are connected to the first part of the Local Master (see Fig. 28), which consists of three different subsystems (X, Y, Z) each of them realizes the interpolation for its coordinate.

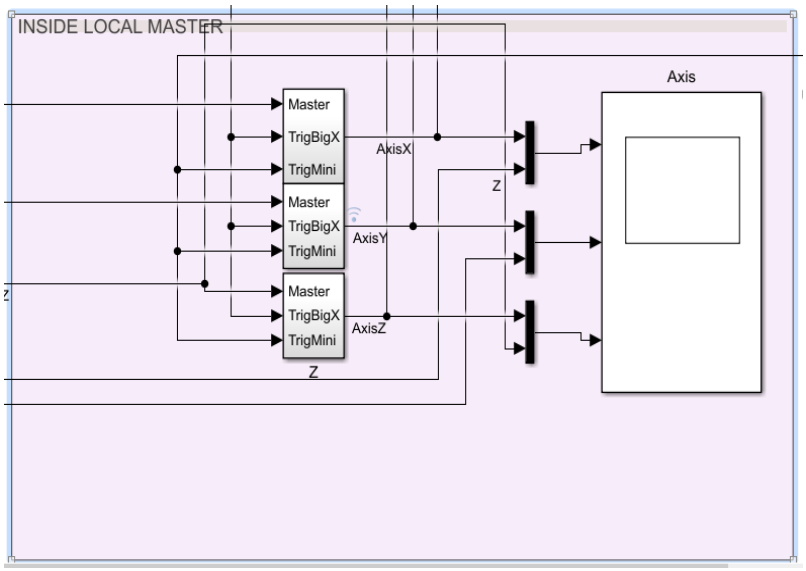


FIGURE 28: LAYOUT OF THE FIRST PART OF THE LOCAL MASTER

This section has as external inputs, the two triggers and the correspondent coordinate, and as external outputs, the position of the small Set Point (q) and also velocity (v), acceleration (a) and jerk (j), these last ones are only used for the internal update of the function. It has also two variables *i* and *init,* that correspond to the outputs *new_init* and *i_sort* and are used to keep the value of these variables in the next iterations

.

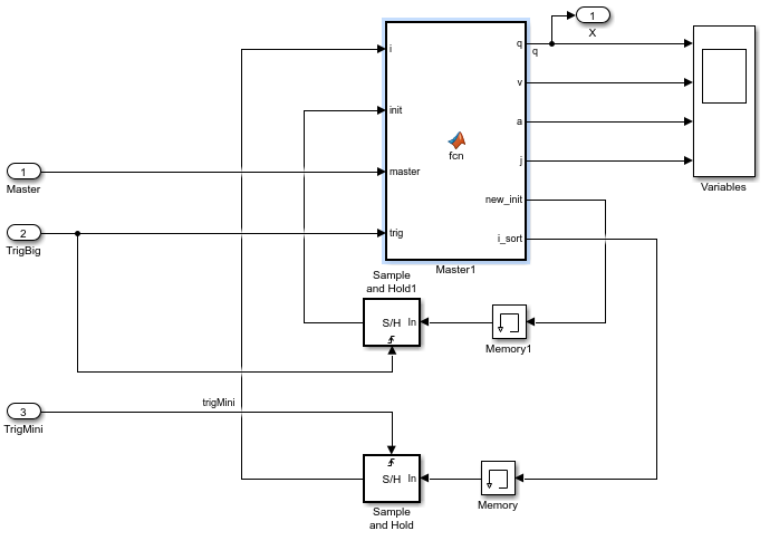


FIGURE 29: LAYOUT OF THE X SUBSYSTEM

The function calculates the path between the Set Points given by the Global Master and it ensures that they follow a smooth path using a 7th degree polynomial (Eq.1), which not only assures the continuity of the position but also of the velocity, acceleration, and jerk.

$$\boldsymbol{q} = a_0 + a_1 \cdot \left(\frac{t_1}{x} \cdot i\right) + a_2 \cdot \left(\frac{t_1}{x} \cdot i\right)^2 + a_3 \cdot \left(\frac{t_1}{x} \cdot i\right)^3 + a_4 \cdot \left(\frac{t_1}{x} \cdot i\right)^4 + a_5 \cdot \left(\frac{t_1}{x} \cdot i\right)^5 + a_6 \cdot \left(\frac{t_1}{x} \cdot i\right)^6 + a_7 \cdot \left(\frac{t_1}{x} \cdot i\right)^7; \quad \textbf{(Eq.1)}$$

$$\boldsymbol{v} = a_1 + 2 \cdot a_2 \cdot \left(\frac{t_1}{x} \cdot i\right) + 3 \cdot a_3 \cdot \left(\frac{t_1}{x} \cdot i\right)^2 + 4 \cdot a_4 \cdot \left(\frac{t_1}{x} \cdot i\right)^3 + 5 \cdot a_5 \cdot \left(\frac{t_1}{x} \cdot i\right)^4 + 6 \cdot a_6 \cdot \left(\frac{t_1}{x} \cdot i\right)^5 + 7 \cdot a_7 \cdot \left(\frac{t_1}{x} \cdot i\right)^6;$$

$$\boldsymbol{a} = 2 \cdot a_2 + 2 \cdot 3 \cdot a_3 \cdot \left(\frac{t_1}{x} \cdot i\right) + 3 \cdot 4 \cdot a_4 \cdot \left(\frac{t_1}{x} \cdot i\right)^2 + 4 \cdot 5 \cdot a_5 \cdot \left(\frac{t_1}{x} \cdot i\right)^3 + 5 \cdot 6 \cdot a_6 \cdot \left(\frac{t_1}{x} \cdot i\right)^4 + 6 \cdot 7 \cdot a_7 \cdot \left(\frac{t_1}{x} \cdot i\right)^5;$$

$$\boldsymbol{j} = 2 \cdot 3 \cdot a_3 \cdot \left(\frac{t_1}{x} \cdot i\right) + 3 \cdot 4 \cdot 5 \cdot a_5 \cdot \left(\frac{t_1}{x} \cdot i\right)^2 + 4 \cdot 5 \cdot 6 \cdot a_6 \cdot \left(\frac{t_1}{x} \cdot i\right)^3 + 5 \cdot 6 \cdot 7 \cdot a_7 \cdot \left(\frac{t_1}{x} \cdot i\right)^4;$$

The interpolation is divided in 15 points, which are represented by *x*. *i* is the iteration variable and $t_1 = T = \frac{x}{f_g}$ expresses the part of the iteration that has already been executed.
And the value of the coefficients:

$$a_o = q_0;$$

$$a_1 = v_0;$$

$$a_2 = a_{c_0};$$

$$a_3 = \frac{j_0}{6};$$

$$a_4 = \frac{210 \cdot h - T \cdot \left(30 \cdot a_{c_0} - 15 \cdot a_{c_1}\right) \cdot T + (4 \cdot j_0 + j_1) \cdot T^2 + 120 \cdot v_0 + 90 \cdot v_1}{6 \cdot T^4};$$

$$a_5 = \frac{-168 \cdot h - T \cdot \left(20 \cdot a_{c_0} - 14 \cdot a_{c_1}\right) \cdot T + (2 \cdot j_0 + j_1) \cdot T^2 + 90 \cdot v_0 + 78 \cdot v_1}{2 \cdot T^5};$$

$$a_6 = \frac{420 \cdot h - T \cdot \left(45 \cdot a_{c_0} - 39 \cdot a_{c_1}\right) \cdot T + (4 \cdot j_0 + 3 \cdot j_1) \cdot T^2 + 216 \cdot v_0 + 204 \cdot v_1}{6 \cdot T^6}$$

$$a_7 = \frac{-120 \cdot h - T \cdot \left(12 \cdot a_{c_0} - 12 \cdot a_{c_1}\right) \cdot T + (\cdot j_0 + j_1) \cdot T^2 + 60 \cdot v_0 + 60 \cdot v_1}{6 \cdot T^6}$$

The initial conditions q_0,v_0, a_{c_0} and j_0 are equal to the variable *init* and the final conditions are all equal to zero except $q_1 = master$ (X input) because position has to be conserved, in order to generate the smooth path. The function is governed by if-statements and the process is show in the diagram below:

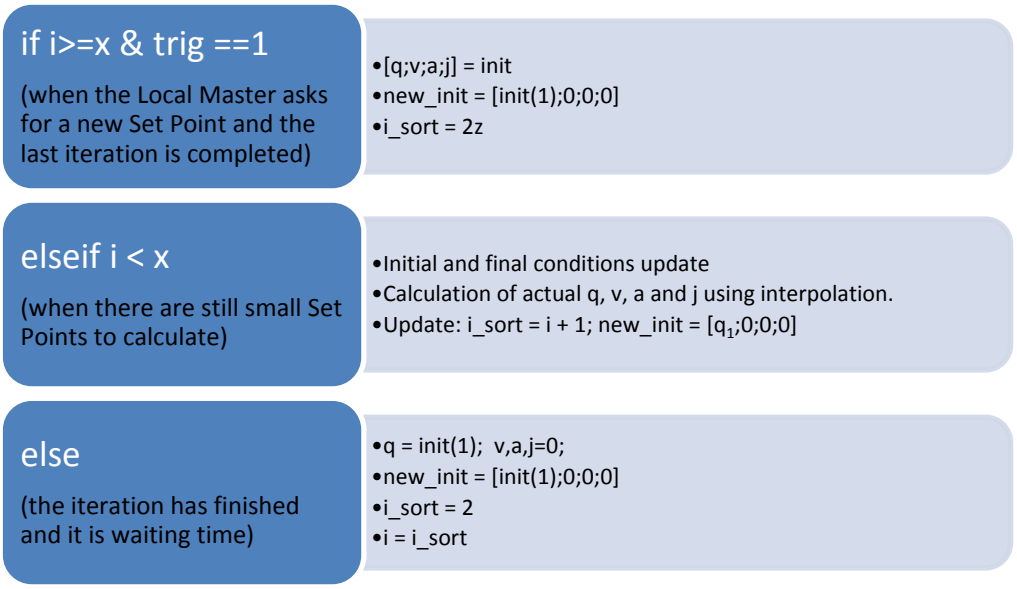


FIGURE 30: WORKFLOW OF THE INTERPOLATION FUNCTION

This function is updated every time the TrigMin changes.

Once the interpolation has been done, these new small SetPoints (named SetPoint11im, SetPoint12i and SetPoint13im) are sent to the Axis 1 (see Fig.31) and the Stateflow begins to work.

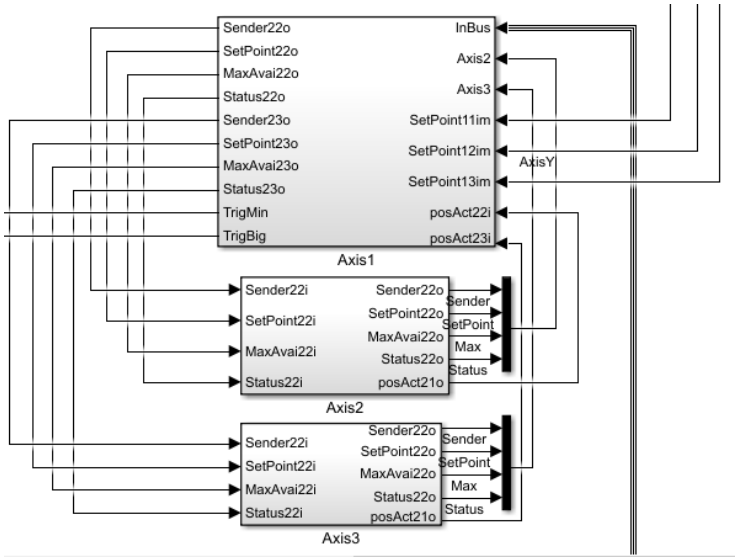


FIGURE 31: LAYOUT OF THE AXES

Axis 2 and 3 have the same layout with COM and MOT state machines and Axis 1 has also the GSM state machine (see Fig.32). If the GSM state machine will be changed to another random Axis it will also work as it will be tested in the next chapters.

Inputs and Outputs of Axis 2 and 3 correspond to the Message Type 2 and as it can be seen, these Axes only communicate with the Local Master and not between them. Axis 1 needs the inputs of the other both Axes and also the triggers to control the rhythm of operation of the system.

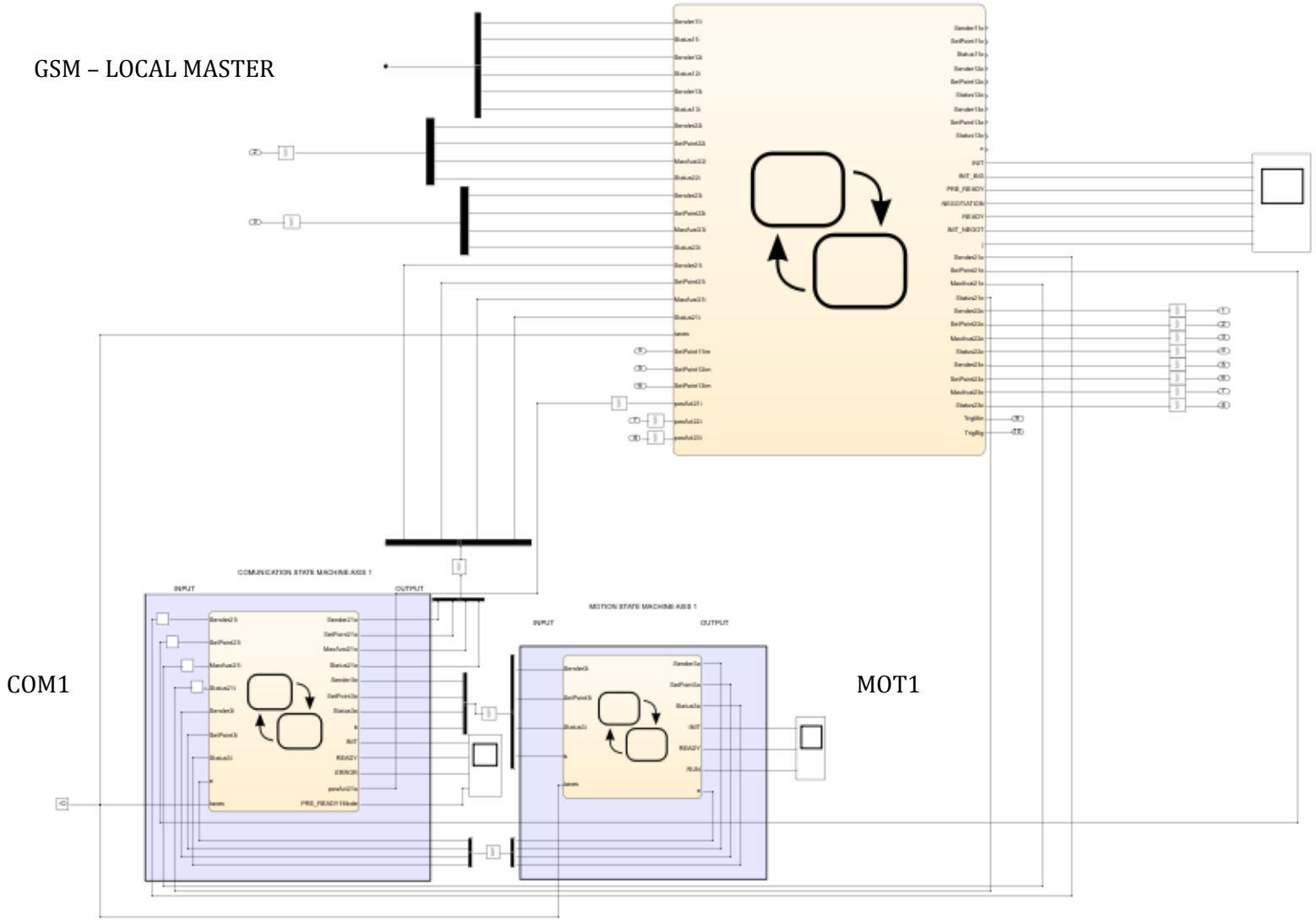


FIGURE 32: LAYOUT OF AXIS 1

The states of the State Machine are the ones explained in Negotiation protocol (see section 3.2) and they will not be deeply explained but some points of interest that can be difficult to understand will be described:

ALL STATE MACHINES

- Delays between connections:
  To simulate the communication delays, there are 3 types of delay between the connections (external, internal and Global Master).
- States with named proc:
  These states represent the processing time of the FPGA.
- Outputs disconnected:
  There are few outputs that they are not connected to anything. They are mainly the outputs for visualization of the different states.
- Error state: all states can go to an error state and they will only go out when the Local Master sends to them everything is OK again (State different than 1).

GSM-LOCAL MASTER

- Initialization: How to control the both triggers :
  The system starts in the INIT state as it can be distinguished in the Fig.33 for the arrow with the sphere. As the variable *j* has an initial value of 15, the system will directly go to INIT_BIG, where the TrigBig is initialized and then come back to INIT state. In this state, the TrigMin is activated and a new small point is asked for.
  The system will go 15 times through the INIT state until coming back to the INIT_BIG for asking for a new Set Point.

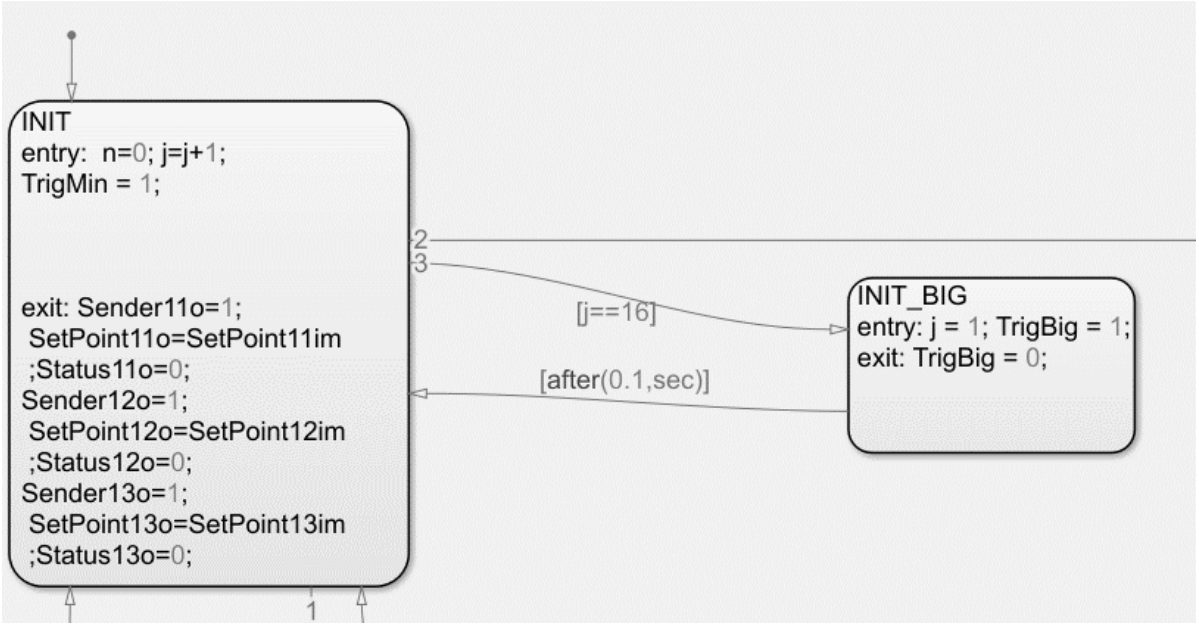


FIGURE 33: INITIALIZATION OF THE GSM

- Explanation of transitions with functions:
  There are 3 transitions that are represented with functions called (T1G, T2G, and T5G) because these functions were too long to be written (see Fig. 34). The functions do not have the proper name of the inputs because that can only be done when they are called, not when we write them in Stateflow. Consequently, their code it's difficult to be understood because of the lack of context.
  - T1G: Sender11i & Sender12i & Sender13i == 1 &SetPoints > 0. Global Master is sending and the Set Point is more than 0. The system is limited to positive values of Set Points.
  - T2G: One or more States are equal to 2. Axes are not OK with the SetPoint and the negotiation has to start.
  - T5G: All Status sent from Axes is equal to 5 (That means RUN state of the MOT machine has started).

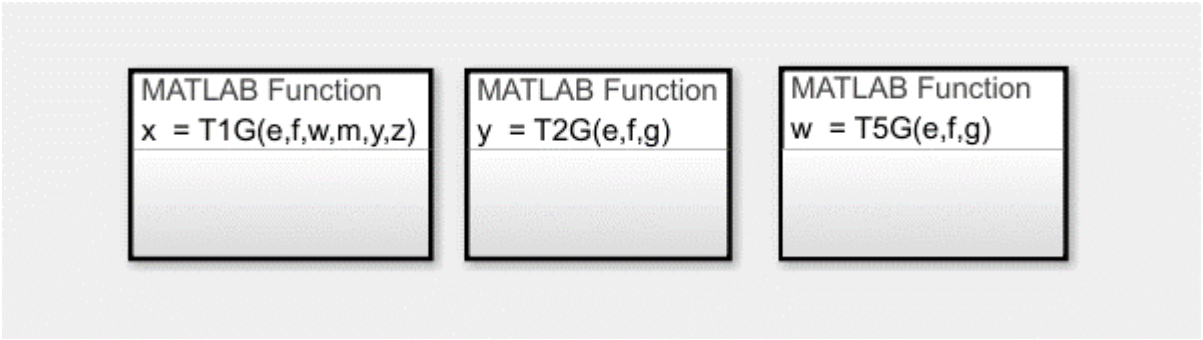


FIGURE 34: THE FUNCTIONS THAT GOVERN THE TRANSITIONS

- Explanation of the used variables:
  The only variable used in this State Machine is the *n* variable. *n* = 0 means that the system has not gone through the negotiation state, *n* = 1 is updated when the system enters in the Negotiation state. Thus, the system to go to the INIT_NEG state and not to the normal INIT because the small Set Point has not been reached yet.

- Max Avai function:
  This is the function used in the Negotiation state for calculating the maximum point reachable by each axis. Hence, the necessary inputs are the actual position, the Set Point and the *MaxAvai* (maximum point reachable calculated by each axis).
  The goal of the function is to calculate the new Set Point for each axis in order to reach the minimum maxim available to keep a synchronous movement. The argumentation for that is to keep a synchronous movement.
  The function can look longer because the minimum and absolut functions cannot be used inside Stateflow. The output is the *MaxAvai21o* that is sent to every axis.
  It is important to notice that the percentages are initialized at 1 because if the *MaxAvai* is already the SetPoint the Axis has reached the 100% of the path and does not have to have any influence on the final result.

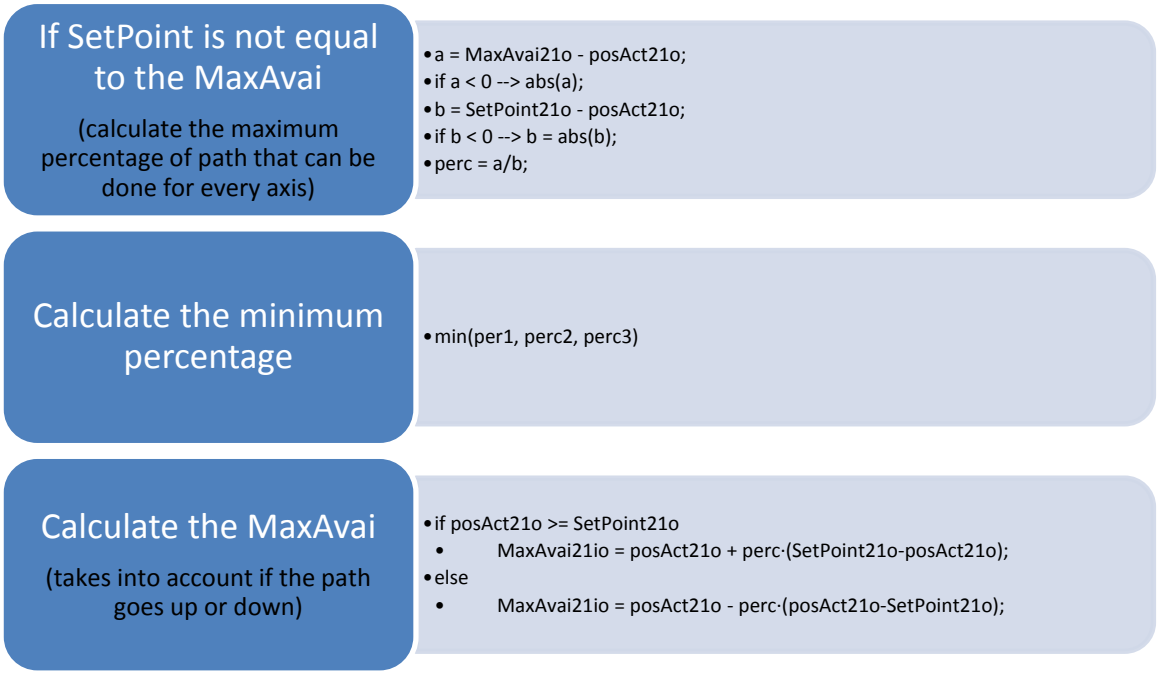


FIGURE 35: WORKFLOW OF THE MAXAVAI FUNCTION

COM

- PRE_READY State:
  This is the state which sends the information to the Negotiation state of the GSM State Machine. It is composed of 5 different states: INIT, CORRECT, NEGOTIATION, WAIT, and LASTSTEP.

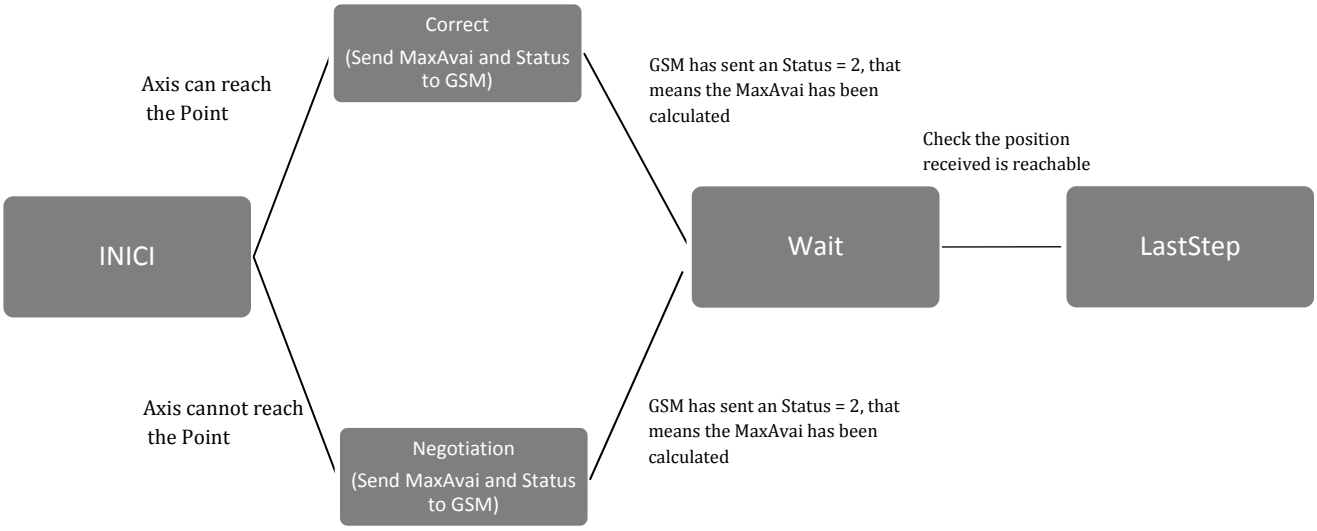


FIGURE 36: PROCESS EXPLANATION OF THE PRE_READY STATE

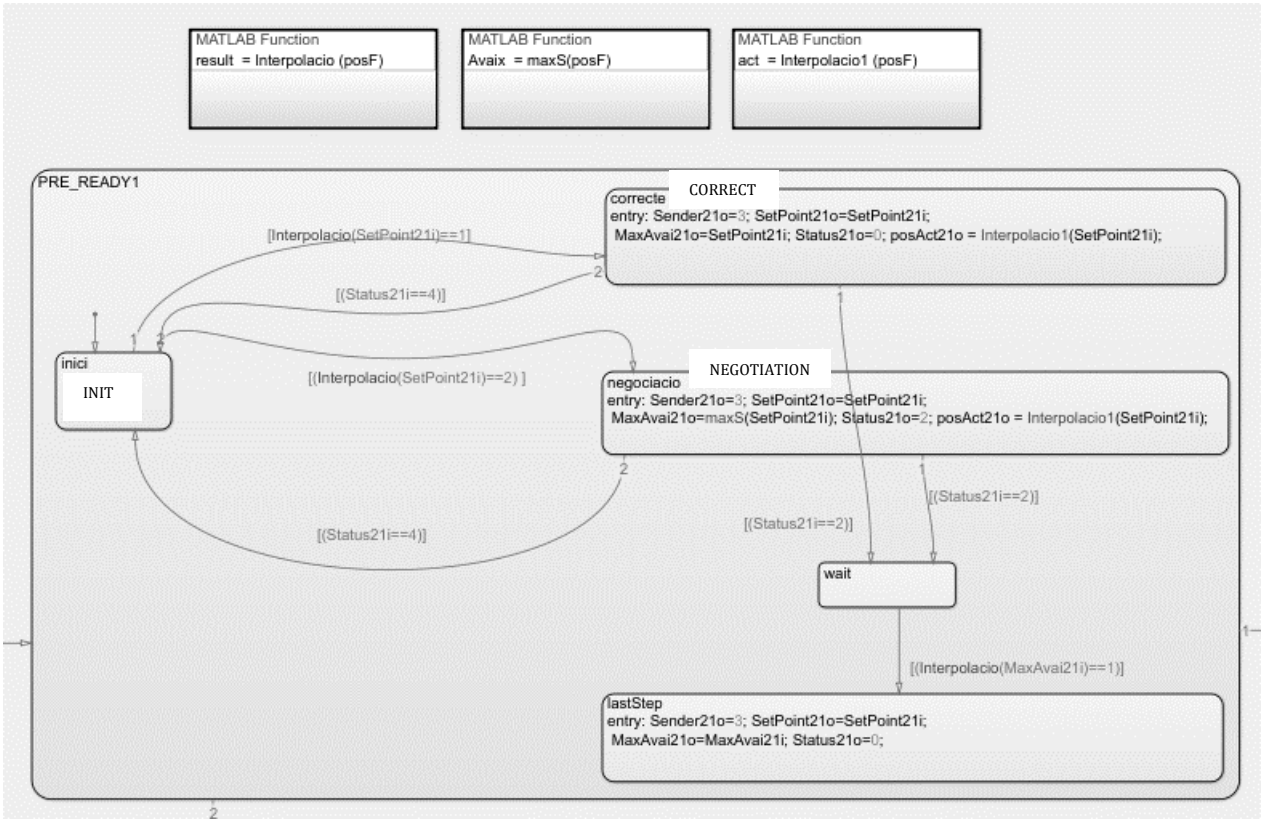


FIGURE 37:LAYOUT OF PRE_READY STATE OF COM STATE MACHINE

The Axis goes out of the PRE_READY when all the Axis have arrived at the LASTSTEP state.

There are 3 functions used inside this state: Interpolacio, maxS, and Interpolacio. The three of them have the same code but different outputs due to Stateflow restrictions (is not possible to assign a value of a function with more than one output). Hence, only one is described.

The function realizes a linear interpolation and calculates the necessary velocity for reaching the next Point. This velocity is compared to the maximum velocity defines and gives a result of 1 if it is OK and 2 if it is not reachable. The restriction of the position will also give a 2 but has not been fully implemented.

In order to perform this computation is necessary to know the actual position of the Axis. This parameter has to be kept to the next runs of the function. Consequently, has been defined as a *persistent variable* [3] and also a local variable in the Stateflow because it is necessary to be updated after solving the negotiation, the value of the actual position will be the *MaxAvai* output of the *MaxAvai* function of the GSM, which is the final set point that will be sent to motion.

[3] A persistent variable is a variable variables that is local to the function in which it is declared; yet its value are retained in memory between calls to the function.

The *maxS* function uses the actual position to calculate the maximum reachable position with the help of the maximum velocity and *Interpolacio1* returns the actual position.

- Variables used in this machine:
  The only variable applied is *b* which is a method of coordination between COM and MOT. When *b* = 1 COM has just entered into the READY state and *b* = 0 COM has exit READY state.

MOT

- Time transitions different than taxes
  Until that point, all the transition related to time were *taxes*, which is the time of computation attributed to the axes. Nevertheless, in this State Machine we have two different ones:
  - The transition between READY and RUN: defined as 2 seconds, it is the time necessary for the Axis to begin the movement.
  - The transition between RUN and INIT: it has been defined as 2.5 seconds and it is the time attributed to the movement of the Axis.

- Variables used in this machine:
  The only variable applied is *e* which is a method of coordination between MOT and COM. *e* = 1 MOT has activated the RUN state) and *e* = 0 MOT has deactivated RUN state. When MOT has activated the RUN state, the MOT can start to negotiate the next Point.

3.3.2 RESULTS

In this subsection, the results of the simulation will be presented.

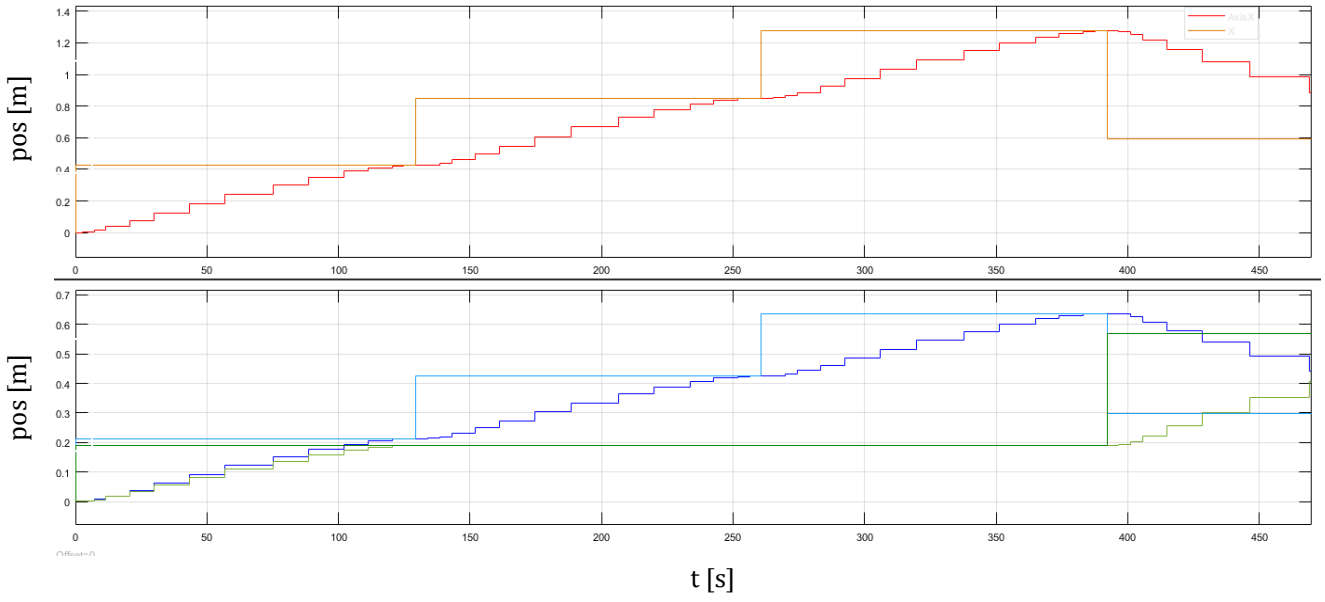


FIGURE 38: ABOVE X PATH AND BELOW Y AND Z

As it has been mentioned before. The path generated grows at the beginning and decreases at the end. In the Fig.38, which shows the outputs of the X,Y,Z subsystems it is easy to appreciate

how the model generates smaller steps near the change of Set Point to avoid big changes that could affect negatively to the Axis.

To check how the Axes evolve is interesting to plot of the PRE_READY state of the COM state machine (see Fig.39 and Fig.40). These two figures show the differences between the trajectory of X and Y axis. The X path has a greater difference between the Set Points and, consequently, the model needs to enter in the NEGOTIATION intermediate state of COM1 more often. Conversely, COM2 does not enter the NEGOTIATION state directly, at least at the beginning, is the GSM which tells it to go to the WAIT state because the negotiation due to Axis 1 has started.

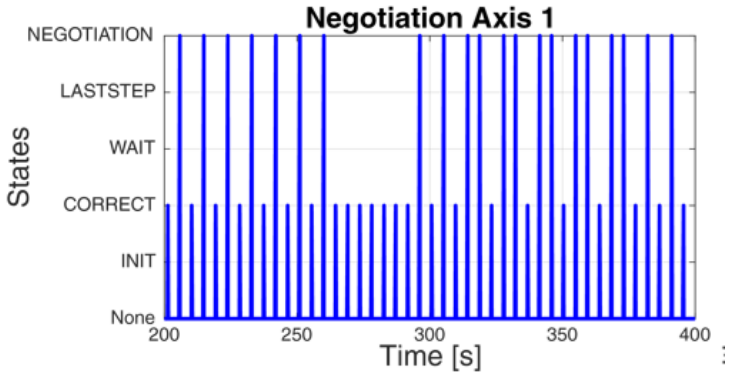


FIGURE 39: REPRESENTATION OF THE PRE_READY OF THE COM1 STATE MACHINE

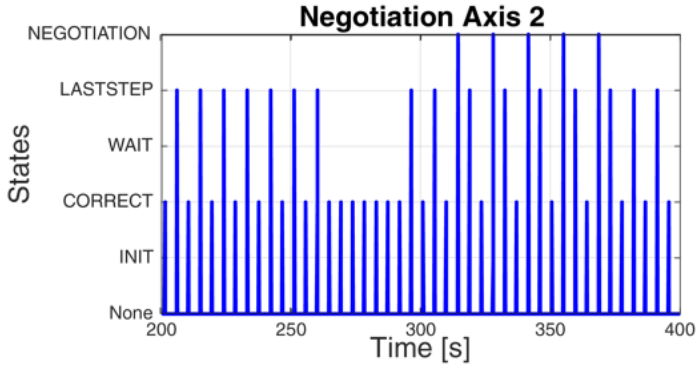


FIGURE 40: REPRESENTATION OF THE PRE_READY OF THE COM2 STATE MACHINE

The last interesting graphic is the output of the RUN state, which means the path that will be sent to the physical Axis (see Fig.41). A point of attention is the shape of the path and the number of small Set Points between the big Set Points, several more than the 15 result of the first interpolation, due to the negotiation processes.

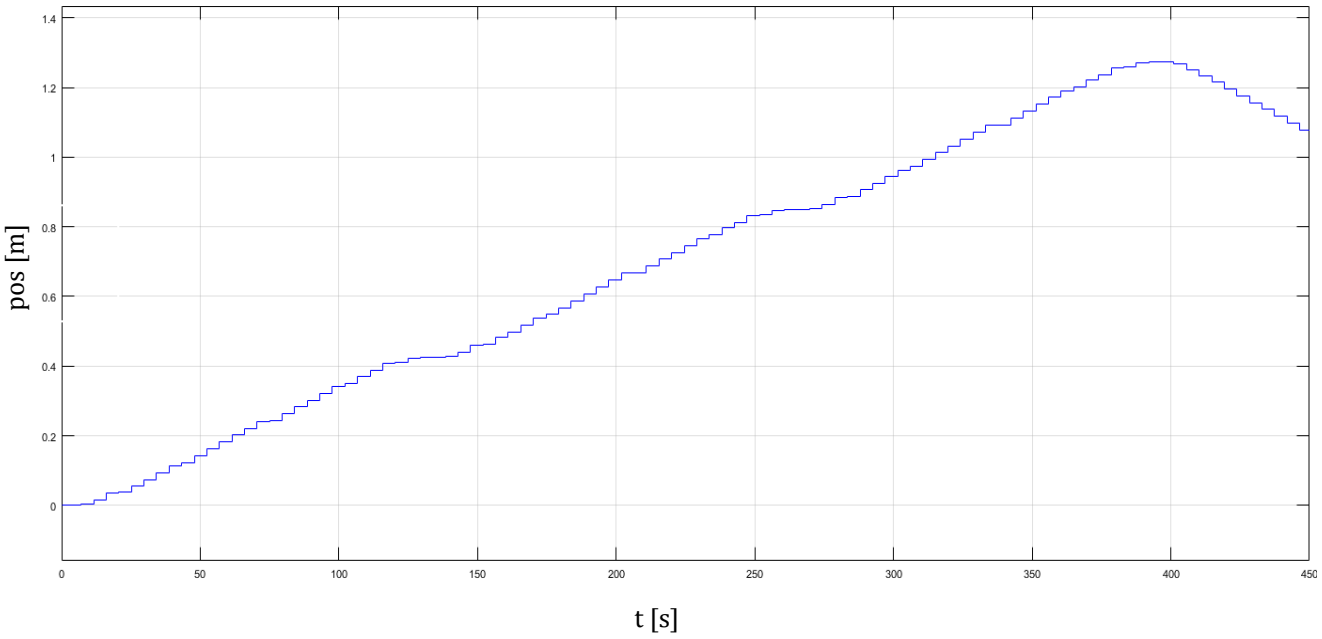


FIGURE 41: OUTPUT OF THE POSITION OF THE MOT STATE MACHINE OF AXIS 1 – PATH GENERATED

3.3.3 IMPROVEMENTS

After the simulation was implemented and tested, two noticeable improvements to make it more realistic and easier to read were done.

- <u>All Axis can be the Local Master:</u>
  The GSM State Machine was implemented also in the Axis 2 and 3 in order to make them able to perform also as Local Master.
  This change also needed the creation of more inputs and outputs for the three Axis and the implementation of a system of switches that will be controlled by a constant variable *l* that corresponds to the number of Axis chosen to be the Master. This variable is run in the MATLAB .m file of constants that has to be executed before the simulation.
  This execution meant a great increase in the number of connections and made the model more difficult to read for a foreign eye.

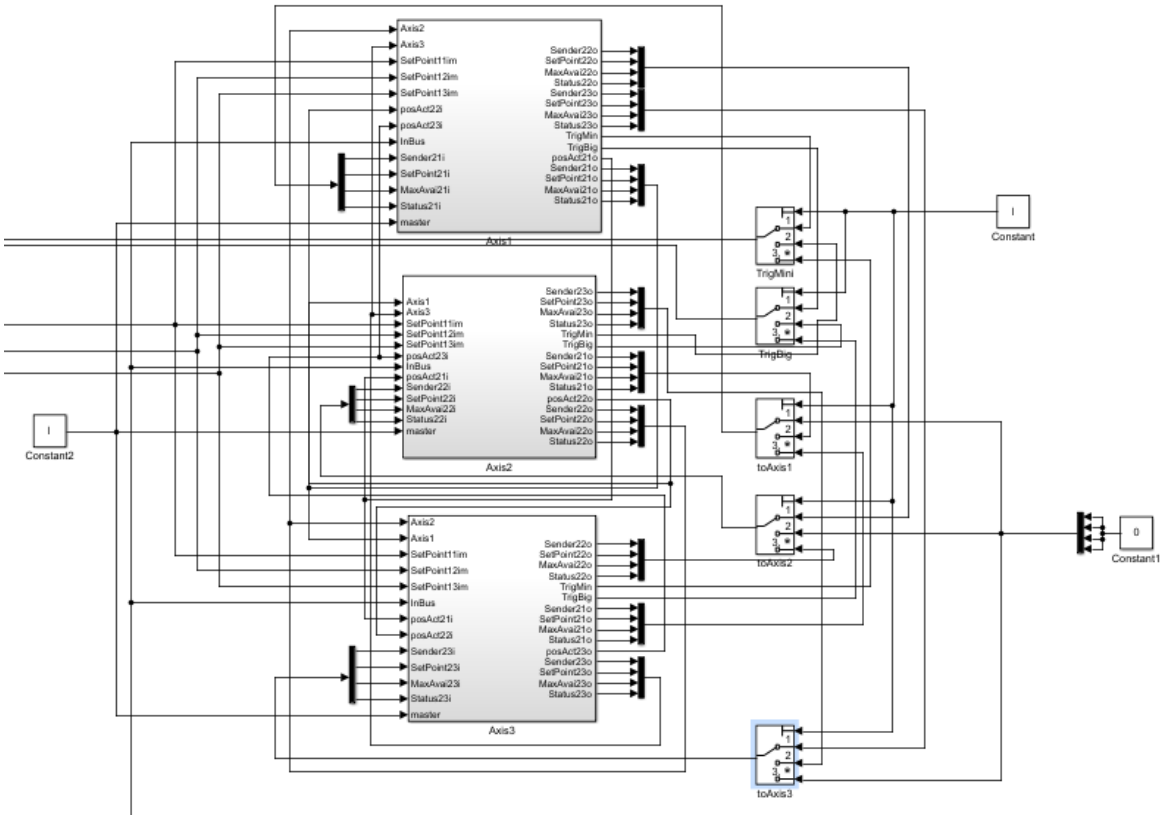


FIGURE 42: LAYOUT OF THE MODIFIED AXES

It was also necessary to create a new state inside the GSM State Machine of every Axis. This state is in charge of deactivating the State Machine if the Axis is not the chosen one to be the Local Master. The state called BREAK has an entry but no output, which is the way to assure that the model does not continue to make any calculations inside this State Machine.

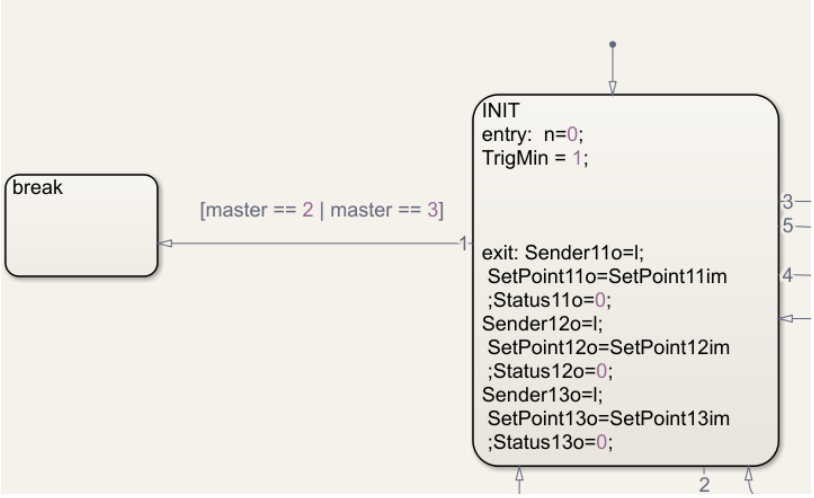


FIGURE 43: BREAK STATE INSIDE THE GSM MACHINE OF AXIS 1

- Implementation of a mask:
  A mask that allows the user to choose from a list which of the three Axes is going to be the Local Master has been programmed.
  This list allows the user to not have to make any modifications of the code to run all of the possible options.

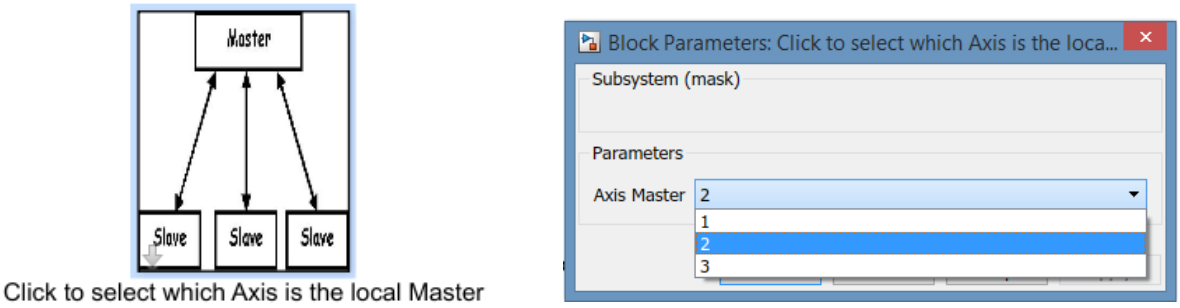


FIGURE 44: MASK TO SELECT THE LOCAL MASTER

4 CONVERSION TO C/HDL CODE AND IMPLEMENTATION

In this chapter, the simulation done with MATLAB in Chapter 3 will be converted into C and HDL Code with the help of MATLAB Coder and HDL Coder. The steps followed and the problems encountered will be precisely documented. It will have a greater focus into the top level inputs/outputs to the modules and how they can be configurated by the user. The converted modules are then deployed on appropriate hardware with a sensible I/O visualization of the internal processes.

4.1 CONVERSION TO C

In this section, it will be explained how the Simulink simulation was converted into C code using MATLAB Coder and which are the considerations to take into account. This C Code will not be implemented because the target platform is a FPGA.

First of all, it is necessary to download and install the MinGW-w64 compiler for MATLAB. To execute this compiler, we will write the next command in the MATLAB workspace [21,22]:

>> mex – setup C (or mex – setup C++)

Before checking if the Simulink Simulation can be directly converted to C, we can check some parameters in Simulation > Configuration Parameters > C Code. This tab will allow us to make some early customization.

The next step is to execute the Code Generation Advisor (Code > C/C++ Code > Code Generation Advisor):

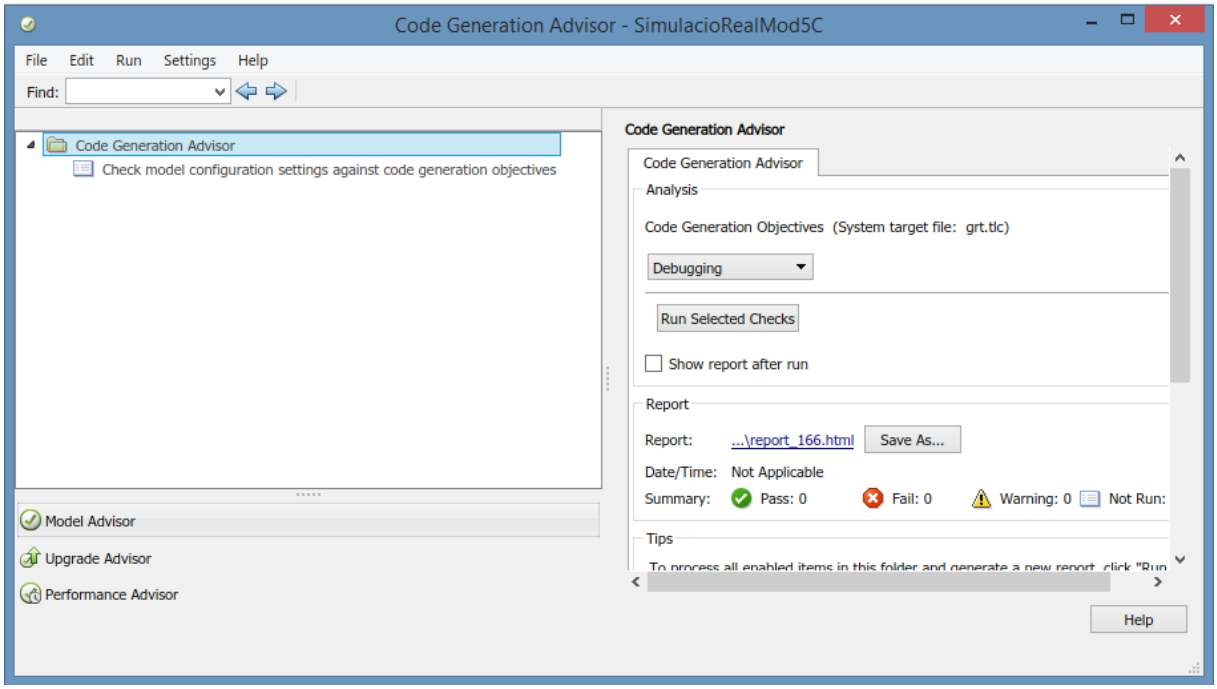


FIGURE 45: CODE GENERATION ADVISOR

In the right window, we select the Code Generation Objectives, in our case, Debugging.

We run the selected checks and click on the Upgrate Advisor which will run several more checks.

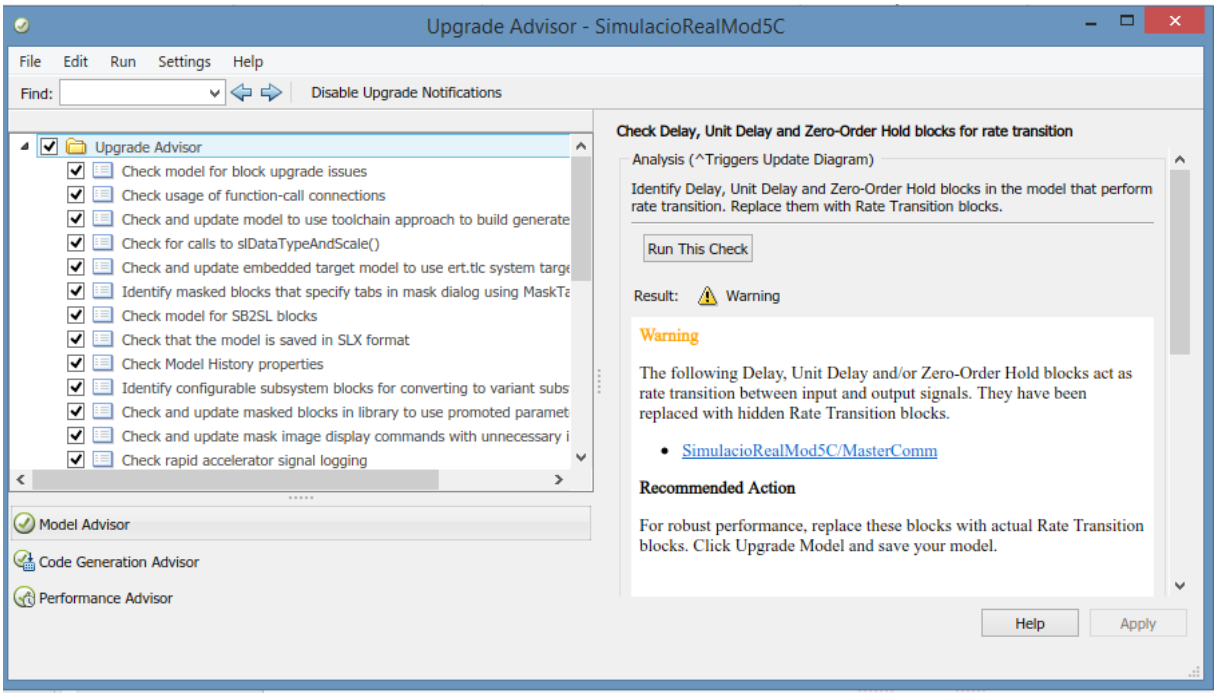


FIGURE 46: UPGRATE ADVISOR

Now, the C generation can start, Code > C/C++ Code > Build selected subsystem. The simulation gives an error because, for C generation, the solver type has to be Fixed-Step. After that, we update the simulation and Build again.

MATLAB will create a folder with the same name of the simulation but add '_grt_rtw' at the end. In this folder, it can be found all the files generated, including the code.

For running the code, we have to introduce the name of the simulation in the Command Window adding a '!' at the beginning.

>> !SimulacioRealMod5C

This command will generate a MAT-File which contains the same variables as those generated by simulating the model.

4.2 CONVERSION TO HDL

This section is a guide to the procedure necessary to convert the distributed simulation into HDL Code, only one of the Axis will be converted and later tested. The errors that have occurred will be described and also the solutions implemented. The process has need longer time and effort than the C conversion and it is one of the goals of the thesis because it is necessary to run the FPGA.

It is important to notice that Diagnostic Viewer works with cascading errors, every time a problem is solved another one appears.

4.2.1 HDL CODE VERIFICATION

The first time it was checked the possibility to convert the system using the option Code > HDL Code > Generate Code, the operation gave several errors. In the next lines the problems found during the conversion will be explained as well as the solutions implemented:

The errors can be classified into two types: the first ones are the errors related to State Flow and the second ones, to the Simulink in general.

4.2.1.1 Stateflow errors

A few considerations have to be taken into account when we try to implement this chars in HDL code.

1. Chars must be initialized at the beginning of the simulation.
   This problem can be solved using the Model Explorer (View > Model Explorer > Model Explorer). The Chars have to be selected in the left part of the window and the option can be found in the right part.

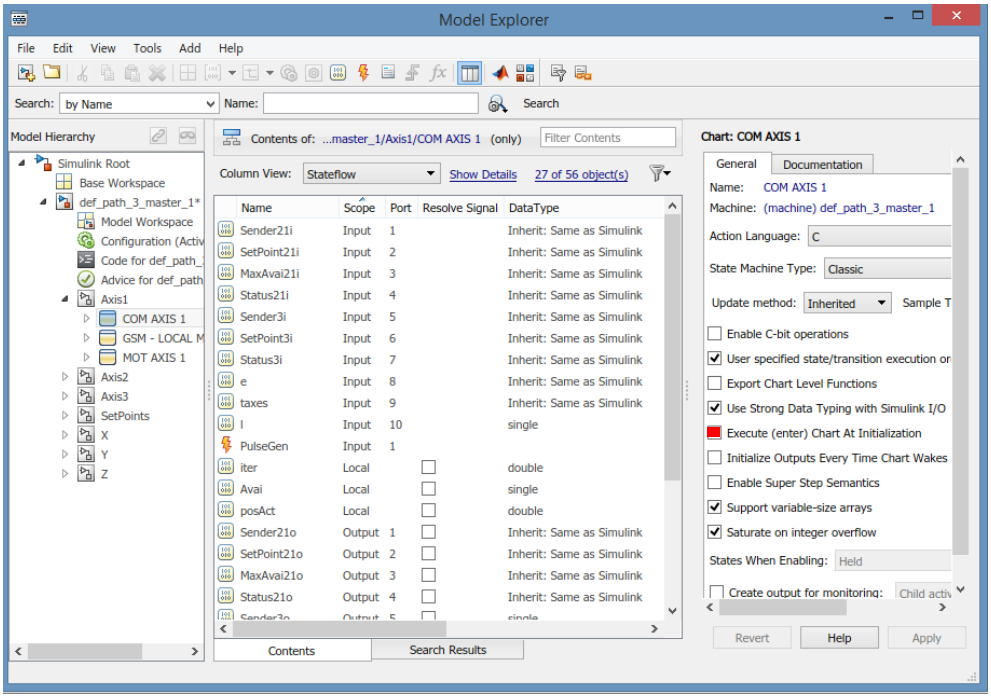


FIGURE 47: MODEL EXPLORER PANEL

2. Subsystems can't have absolut temporal logic:
   Temporal logic controls the execution of a chart in terms of time. In state actions and transitions, you can use two types of temporal logic: event-based and absolute-time. Event-based temporal logic keeps track of recurring events, and absolute-time temporal logic defines time periods based on the simulation time of your chart. To operate on these recurring events or simulation time, you use built-in functions called temporal logic operators.[23]
   Absolut-time temporal logic is used on the transitions that represent the running time of the Axis and also the time of the process. It is only allowed to use this transitions if they are related to events of the own simulation and not to a period of time.
   The solution executed consisted on converting the charts in Triggered charts adding and Event Input from Simulink. The trigger chosen was a binary signal which changed every

0.01 seconds. This solution failed because the only signal block that can be used in HDL is the Constant block (see Fig. 48).
A good advice to follow during the implementation is to check if the blocks used can be found inside the HDL section in the Simulink Library Browser.

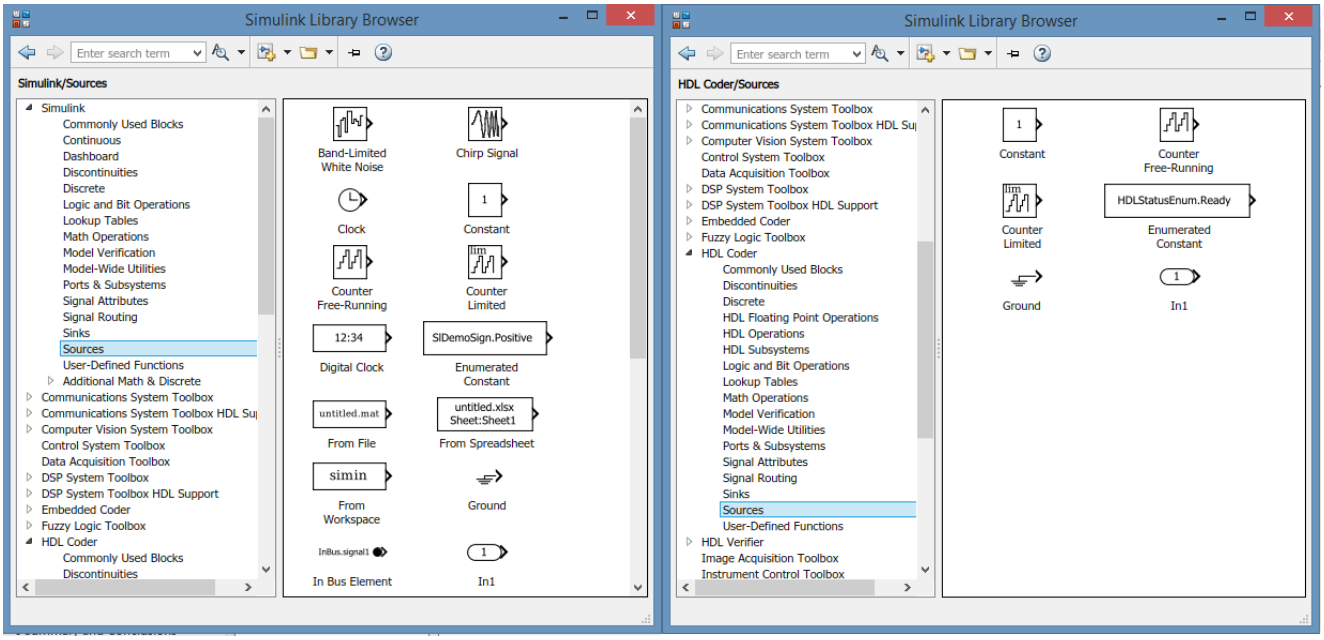


FIGURE 48: COMPARISON OF LIBRARY BROWSER OF HDL AND GENERAL SIMULINK

The final solution implemented is based on the Counter Limited block. This block goal is to be a counter but in this occasion, we will cut out it on 1, so it will actuate as a binary signal of period 0.01 seconds. All the charts will have the same Input Event but we also need to change the transitions which will change from *after(taxes,sec)* to *after(number, PulseGen), PulseGen* is the trigger event. If it is *after(2, PulseGen),* the transition will be fired after 0.02 seconds.

3. Illegal Data access or computation for the Char given that 'Execute at INI' must be enabled.
This problem was more difficult to solve because it is meaning it is not easy to be understood. MATLAB is saying that arithmetic operations can not be done in Chars (only assignments). These operations can be performed in MATLAB functions and transitions.
In our case, we have moved the arithmetic operation to a condition of the transition next to the char as it can be seen in the Fig.49 below:

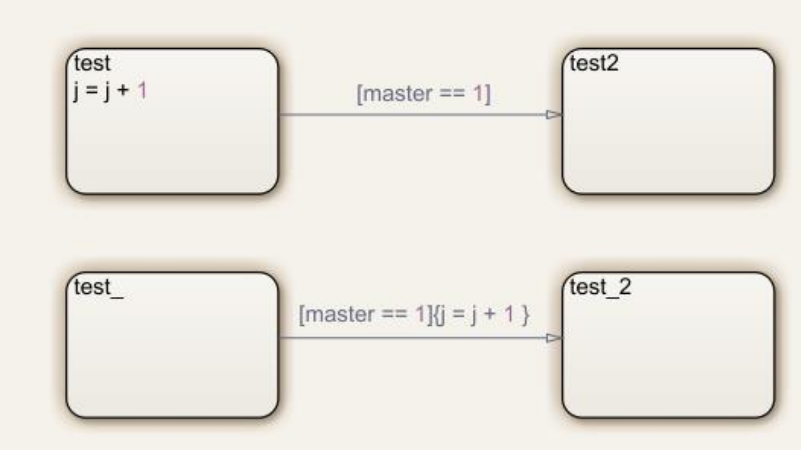


FIGURE 49: CONVERSION APPLIED

4.2.1.2 Errors related to Simulink

Errors related to the general system:

1. Functions like: sin, cos, abs, round, min or max or exponents can't be used.
   These basic functions can not be used unless we activate one of the Floating Point Libraries of the Fig.50. In this case, the appropriate library is one of the Altera ones, because the FPGA used will be from that brand. In order to execute them, it is necessary to install the Quartus II software and run it on the command window. These options were tested and discarded because they do not accept triggered subsystems as the ones that govern the model.

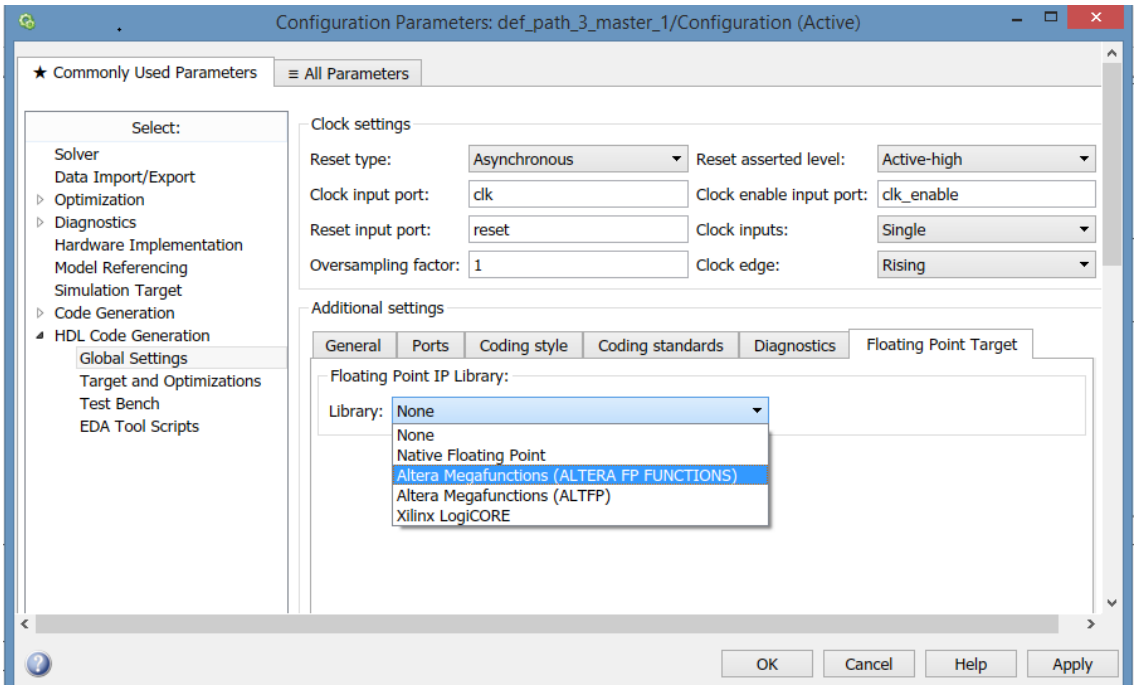


FIGURE 50: FLOATING POINT LIBRARIES

For this reason, all code was changed and the functions become longer and less readable as it is showed in the examples below, which is a variable of X subsystem:

```
a7 = (-120*h + T*((12*ac0 - 12*ac1)*T + (j0 + j1)*T^2 + 60*v0 +
60*v1)) / (6*T^6);
```

It was converted into:

```
a7 = (-120*h + T*((12*ac0 - 12*ac1)*T + (j0 + j1)*T*T + 60*v0 +
60*v1)) / (6*T*T*T*T*T*T*T);
```

2. Fixed-Step:
   The solver type has to be Fixed-Step as in C code. Fixed Step size: 0.005, the same as the smallest one in the system. Also the option "Treat each discrete rate as a separate task" has to be activated.

3. Only one rate transition is accepted. All delays must have inherit time.
   All the delays that they were used for external/internal communication delays have to change to -1, inherit time. These delays would not be necessary in the implementation but they are kept because MATLAB requires them in order to avoid algebraical loops.
   This means, there is no difference of communication between internal and external. In order to keep the coordination of the system, delays in series were added to the Axes and the path generation system.

4. All initial values must be zero.
   HDL does not admit to having Initial Conditions different to 0.
   This issue entailed a change in the initialization of the GSM state because the *j* variable had an initial value of 15 in the first simulation. The solution is shown in the next Fig. 51 and it implicated the creation of a new variable *temp*.

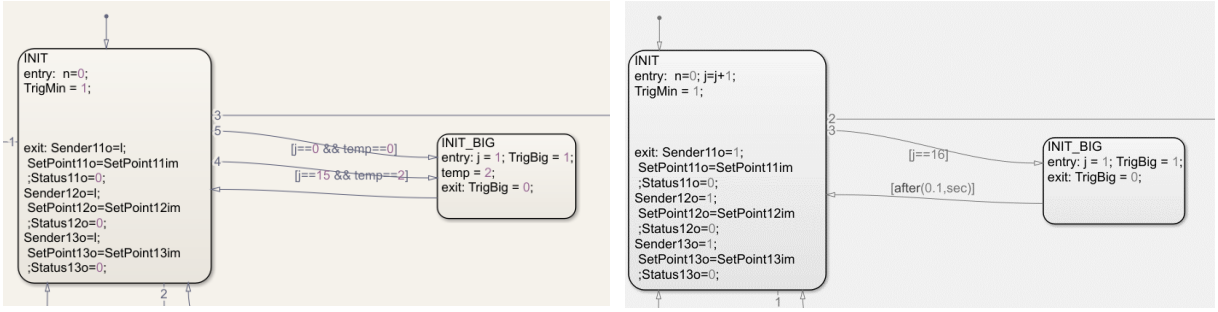


FIGURE 51: CHANGES APPLIED TO AVOID AN INITIAL VALUE OF J.

This variable is only initialized at the first iteration and allows the system to have the same behavior as before.

5. The trigger has to be boolean or uint16.
   All the subsystems employed are triggered by one signal. All that signals have to be converted to boolean or uint16 avoiding the use of the Convert block, which is not admitted in HDL. For this purpose, a small function was implemented:

```
function y = fcn(u)
if u == 1
y = true;
else
y = false;
end
end
```

FIGURE 52: CODE FOR THE MATLAB FUNCTION TO CONVERT INTO BOOLEAN

6. Bus signals
   The bus signals are also not admitted in HDL Code. They were changed to usual signals.

7. Double (warning):
   This last one is not an error but a warning. Although could seem less important, we have to resolve it before the next step because it will become an error there. The exact MATLAB warning is: "Double types are not supported; consider using single types". This seems to indicate that we should use the Single precision variables but it refers to Fixed-Point variables. The next subchapter will be dedicated to resolving this point.

4.2.2 *FIXED-POINT TOOL*

To convert all the variables to Fixed-Point, it has been done by the MATLAB Fixed-Point Tool, which can be found in Analysis > Data Type Design > Fixed-Point Tool.

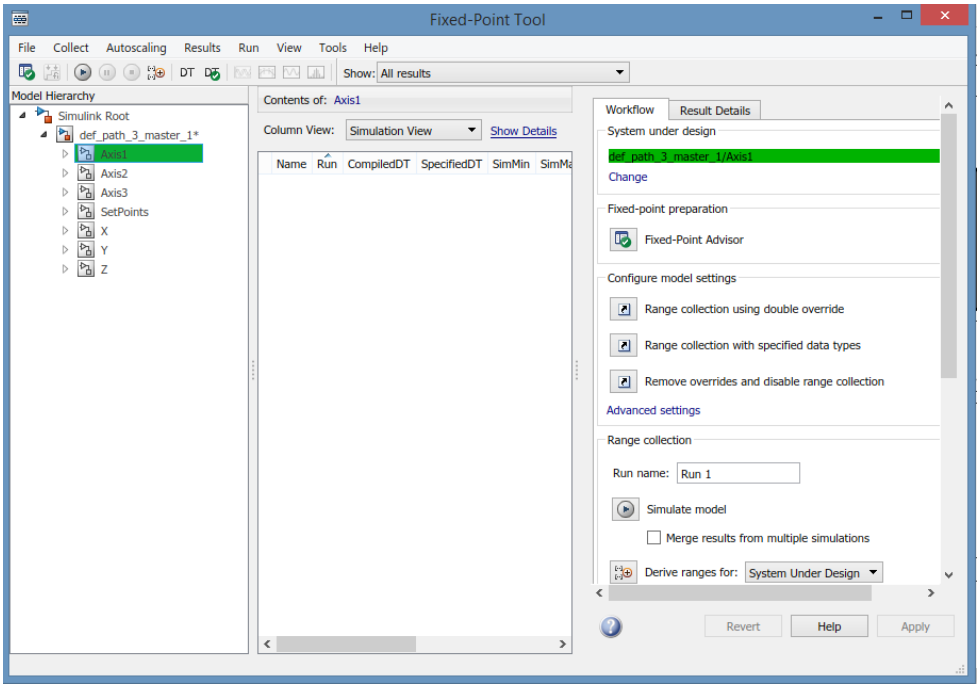


FIGURE 53: FIXED-POINT TOOL

4.2.2.1 Explanation of choice of Fixed-Point worth and fraction length

The fixed-point are a representation of binary numbers that have 3 parameters on that order: signedness, worth length and fraction length.

And it is represented in MATLAB [24] :

$$fixdt(1, 32, 22)$$

Fraction length defines the scaling of the stored integer value and word length limits the values the stored integer can take but has no effect on the fraction length. It is easier to see it with some examples of an unsigned number with 4 bits of worth length:

$$aaaa. = aaaa. \cdot 2^0 \ (Fraction\ length = 0)$$

$$aaa.a = aaaa. \cdot 2^{-1} \ (Fraction\ length = 1)$$

$$aa.aa = aaaa. \cdot 2^{-2} \ (Fraction\ length = 2)$$

$$aa00 = aa. \cdot 2^{-2} \ (Fraction\ length = -2)$$

After checking all the maximum and minimum values that take all the variables in the simulation, it was decided to apply a fixed-point of worth length of 32 and fraction length of 22. This point is able to arrive 512 and has a precision of 10^{-4}.

4.2.2.2 Change of structure of the functions

One of the most problematic issues of the Fixed-Point advisor was an error related to the functions inside State Flow. These functions are not allowed [25], which implies to eliminate all the transitions that were represented by functions but, also, and more importantly, to remove the MaxAvai and Interpolacio functions.

These functions have to be implemented outside of the State Flow as it can be seen in the Fig.54:

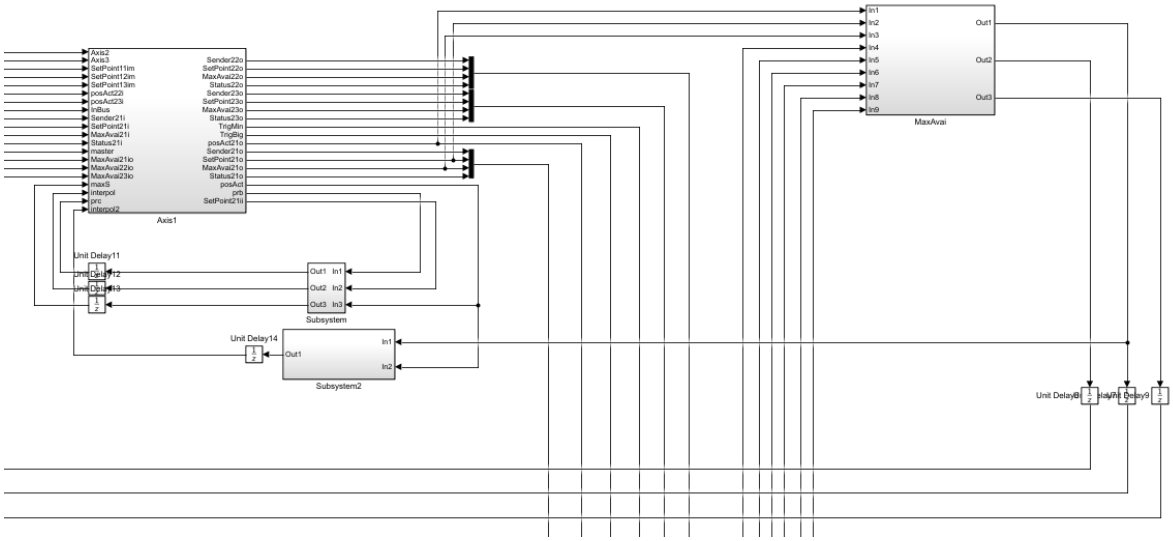


FIGURE 54: NEW LAYOUT WITH THE FUNCTIONS OUTSIDE OF THE AXIS

A big change was needed for every function:

- Function MaxAvai: this is the function, which governs the Negotiation state inside the Local Master and its goal is to calculate the maximum reachable point for each Axis. The function structure is the same but has been necessary to add some more outputs/input to the Axis as *MaxAvai21io, MaxAvai22io, MaxAvai23io* which correspond to the outputs of this function.
- Interpolacio / maxS / Avai: These 3 functions will be converted into two: Interpol and Interpol2.
    a. Interpol: inputs *posAct, posF* = SetPoint21i and *prb*. One of the main variables *posAct*, which was used to keep the actual position of the Axis can not be implemented on the same way: it was a persistent variable and this is not possible for this conversion. It has been created as an output variable inside the State Flow, which allows giving this variable an initial value of 0, and it is defined also as an input of this function. The structure of the function is kept except adding the *prb* variable, which is related to the *prc* variable to coordinate the system. The outputs are *prc, maxS* (corresponds to the earlier *maxS* function) and *Interpolacio* (which is the prior result).
    b. Interpol2: same as Interpol but with different input, instead of SetPoint21i it is MaxAvai21i. It is the function used between WAIT and LASTSTEP to check the point is reachable.

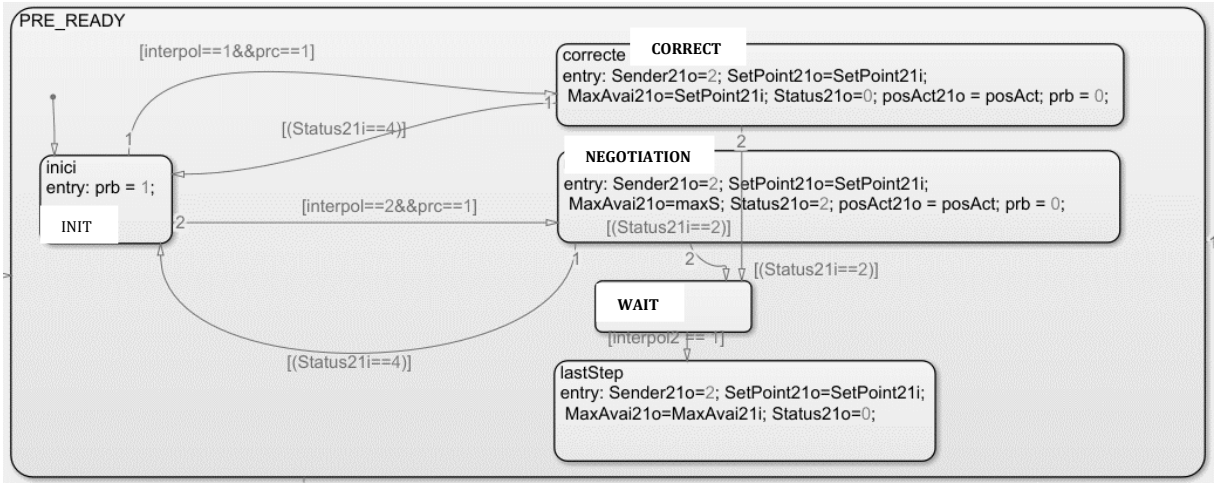


FIGURE 55: PRE_READY STATE AFTER THE MODIFICATIONS

4.2.2.3 Fixed-Point Tool procedure

The procedure to use this tool[4] is the following (before ensure that the model is updated):

1. Select the subsystem to convert.

[4] Before starting to use this tool, it would be interesting to read: https://de.mathworks.com/help/fixedpoint/ug/best-practices-for-using-the-fixed-point-tool-to-propose-data-types-for-your-simulink-model.html

2. Run the Fixed-Point Advisor:

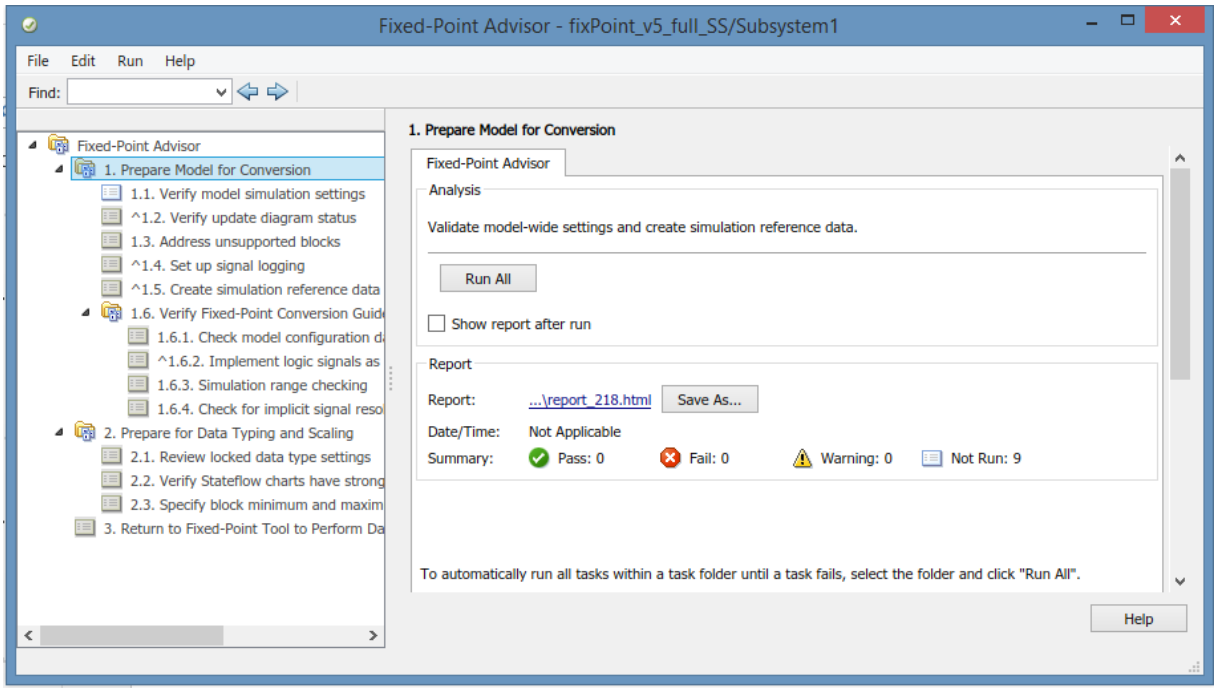


FIGURE 56: FIXED-POINT ADVISOR

Run the steps one by one because it is required to make some changes. Accept the changes of configuration that the Advisor will propose.

- In step *1.4 Set up signal logging*: choose the signals that will be checked by the tool in order to compare if the conversion is correct.
- In step *2.3 Specify block minimum and maximum*: specify the maximum and minimum at least of the Import blocks.

When all the steps have been executed the Advisor returns to the Fixed-Point tool.

3. Select the option “Range collection using double override” in Configure Model Settings. The run will be changed to “DoubleOverride”. [26]
4. Inside Range collection, select “Simulate Model” with the option “Merge results from multiple simulations select”.
5. In Automatic Data Typing, select which Default work length is required, if we would like to have variables signed or not and, then, click on Propose Data Types.
   MATLAB will generate Data Types and we have to accept is we want the proposed ones or propose new ones. It will probably also says that there are some conflicts that have to be solved.
6. Accept the Data Types and close the tool.
7. Insert Convert blocks in the inputs and outputs of the subsystem converted and simulate. If we have accepted the Data Types proposed by the tool, we will realize, at least in this simulation, that the system continues to give errors. MATLAB will tell us there a few mismatch errors, that is because some of the variables are changed in one of the State Machines but not in the others. Thus, it has been decided not to accept the Data Types propose by the tool and use the one explained (fixdt(1,32,22)) in 4.2.2.1 Explanation of choice of Fixed-Point worth and fraction length to all the variables. All variables will have

the same fixed-point, although this is not totally optimised because of the high number of possible mismatches due to bad performance of the application.
If the simulation does not run still, open the Model Explorer and check if some of the inputs or outputs blocks are in *inherit type*. If it is the case, change them to fixdt(1,32,22).

8. Open the Fixed-Point tool again and, in this case, select the option "Range collection with specified Data Types" in Configure Model Settings. The run will be changed to "NoOverride".
9. Close the tool and run. The system is fully converted to Fixed-Point.

The steps explained have to be followed in order and the range collections can not be changed because they are the only ones accepted for HDL conversion.

After all errors were solved, the generation of the Code was made through the HDL Workflow Advisor where the specific characteristics of the FPGA used could be chosen that is explained in the next chapter.

4.3 IMPLEMENTATION

This section describes the process of HDL Code generation with the HDL Workflow Advisor, the loading and testing of this code into the FPGA board and it also takes a deep look on how efficient in terms of FPGA usage is the code generated. For this last step, several tests with different implementations have been realized.

When the simulation has passed the HDL Code > Generate HDL for Subsystem without errors and warnings the generation of HDL Code can start. It is important to eliminate also the warnings because they will cause errors in the next step.

4.3.1 SOFTWARE INSTALLATION AND TECHNICAL INFORMATION

Before starting the code generation an inspection of the FPGA Board chosen for this implementation it is necessary. In our case, the board is the DE1-SoC from Altera. A compilation of some required technical details has been made using the User Manual [27]:

Name of the Board	DE1-SoC
Brand	Altera
Family	Cyclone V
Device	5CSEMA5F31C6
JTAG Chain Position[5]	2
Clock Frequency	50 MHz
Clock Pin Number	AA16
Clock Type	Single-Ended

TABLE 7: TECHNICAL INFORMATION ABOUT THE DE1-SOC BOARD

As it is an Altera Board the Quartus Prime Software Lite edition[6] has to be downloaded and installed. After that, it has to be initialized in the workspace of MATLAB using the following commands:

[5] This feature was not found in the user manual. The information was extracted from the MATLAB sample boards from the same family.
[6] This program can be downloaded in http://dl.altera.com/?edition=lite. 10 September 2017.

>> Path = 'C:\intelFPGA_lite\17.0\quartus\bin64'

>> hdlsetuptoolpath('ToolName', 'Altera Quartus II', 'ToolPath', Path)

The variable Path can change depends on where the software has been installed. We also need to download the HDL Verifier Support Package for Altera FPGA Boards. [7]

4.3.2 HDL WORKFLOW ADVISOR

Before to start a last parameter has be activated: Simulation > Model Configuration Parameters > All parameters > Scalarize vector ports.

At this point, we have to check that the simulation is updated and the HDL Workflow Advisor can be run selecting the Subsystem we would like to convert.

This Workflow Advisor is divided into four sections as it can be seen in the Fig.57: Set Target, Prepare Model for HDL Code Generation, HDL Code Generation and FPGA-in-the-Loop Implementation.

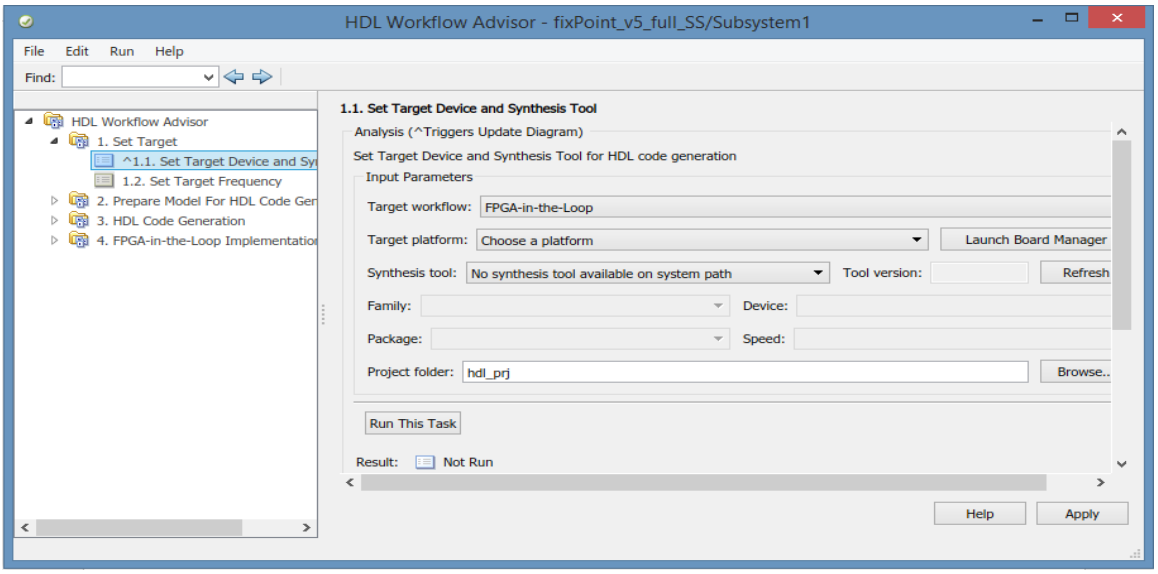


FIGURE 57: HDL WORKFLOW ADVISOR

The steps follow are described next:

1. **Set Target:**
    1. <u>Set Target Device and Synthesis Tool:</u>
        i. Target Workflow: FPGA-in-the-loop
        ii. Target Platform: as the board does not match any of sample one, we will use Launch Board Manager and define a New Board with the details searched in 4.1.
        iii. Synthesis Tool: Altera QUARTUS II
        iv. Project folder: this code generation has problems if the folder used is in the cloud or this process has been done more than one time with the same

[7] It can be downloaded in the next link: https://www.mathworks.com/help/supportpkg/alterafpgaboards/index.html. 20 August 2017.

name of the folder. For this reason, it is better to erase all the non-successful attempts.

2. Set Target Frequency:
    i. Target Frequency: 50MHz

2. **Prepare Model for HDL Code Generation:** Run all of them, accept the changes proposed. An error called 'Abnormal exit' can be detected during these steps, this problem is related to the Scope and when they are erased the problem disappears.
    1. Check Global Settings
    2. Check Algebraic Loops
    3. Check Block Compatibility
    4. Check Sample Times
    5. Check FPGA-in-the-Loop Compatibility: in case the conversion to Fixed Point is not fully achieved, this step will fail.
3. **HDL Code Generation:** run these steps. They enable us to optimize the code and a deeper look will be done in the next section.
    1. Set Code Generation Options
        i. Set Basic Options
        ii. Set Advanced Options
        iii. Set Optimization Options
        iv. Set TestBench options
    2. Generate RTL code and Testbench
4. **FPGA-in-the-Loop Implementation:**
    1. Set FPGA-in-the-Loop Options: this section lets to choose customization options for FPGA-in-the-Loop. One can add additional files to table below for HDL Coder black-box source code, custom FPGA design tool scripts, and/or constraint files.
    2. Build FPGA-in-the-Loop: the last step of the process where the HDL Code is generated. This step can give several problems, although having check all the steps of HDL generation before. The problems were related to VHDL syntaxes errors, division capacity or rights to write in the folder.
        i. Error (10500): VHDL syntax error at COM_AXIS_1.vhd(341) near text "wait"; expecting "(", or an identifier ("wait" is a reserved keyword), or unary operator: As it written there are several words that can not be used in VHDL as 'wait'. The solution of this problem is easy, change this word for another one not reserved, after checking the list of reserved words [28].

        ii. Error (272006) altera lpm divide must be less than or equal to 64 file: in this case the easiest solution after identifying the values which are given the issue, we can lower their precision, split the division into different steps (if more than two values are involved) or use an expression of the type (a/b = a * (1/b)).

        iii. Errors related to lack of authority to write in a file or corrupted files. These errors can be easily solved saving the file, not in a cloud and erase old files.

If the program runs without errors, it will generate a FIL file that looks like the Fig. 58 below. This file is the one used later for the loading and running onto the FPGA.

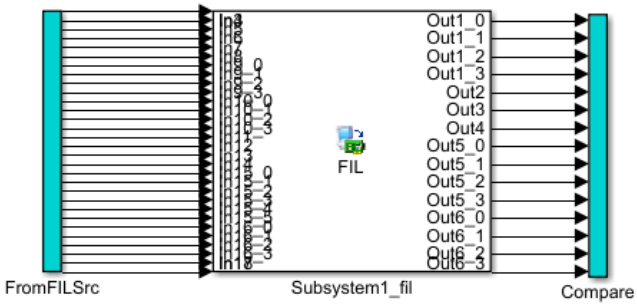


FIGURE 58: SUBSYSTEM GENERATED AFTER THE CONVERSION

It will also open a command window that gives information about the progress of the code generation and informs when the process is finished. The rest of the files generated can give information about, for example, how much usage of the FPGA is necessary to implement the code (FIL.fit):

Family	Cyclone V
Device	5CSEMA5F31C6
Logic Utilization (in ALMs)	9291 / 32070 (29%)
Total registers	6106
Total pins	1 / 457 (< 1%)
Total block memory bits	153488 / 4065200 (4%)
Total RAM Blocks	50 / 397 (13%)
Total DSP Blocks	32 / 87 (37%)
Total PLLs	1 / 6 (17%)

TABLE 8: USAGE OF THE FPGA RESOURCES

In this case, two parameters attract the attention for being particularly high: Logic utilization (29%) and Total DSP Blocks (37%). This is important because we have not converted all system, just only one part. The evolution of this usage will be deeply examined in 4.4.

4.3.3 LOADING PROGRAMMING FILE ONTO FPGA AND RUN SIMULATION

The code is already generated and it is the time to test it with the FPGA with the help of the FIL simulation. For this purpose, the first step is to load the programming file onto the FPGA [29]. Double click on the FIL subsystem and click on the load option.

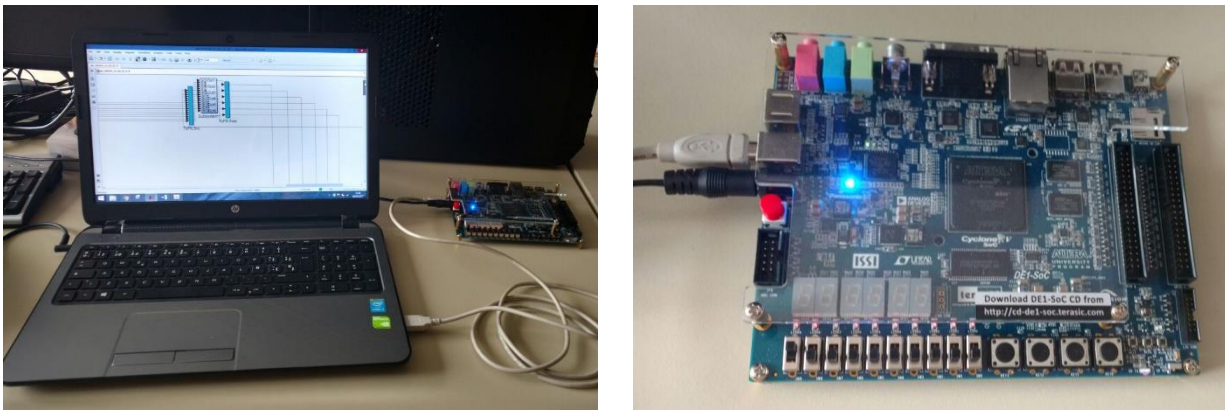


FIGURE 59: RUNNING SIMULATION ON THE RIGHT AND DETAIL OF THE FPGA ON THE LEFT.

Once the code is load, it is only necessary to run the simulation as in a normal Simulink file. The simulation runs normally but slower, it took almost two hours in front of the 20 minutes of the Simulink simulation, which indicates the communication (USB Port) it is not working correctly and it would have needed a faster channel or a revision of configuration. In the next Fig.60 it can be seen the output position of the Axis 1 during this implementation.

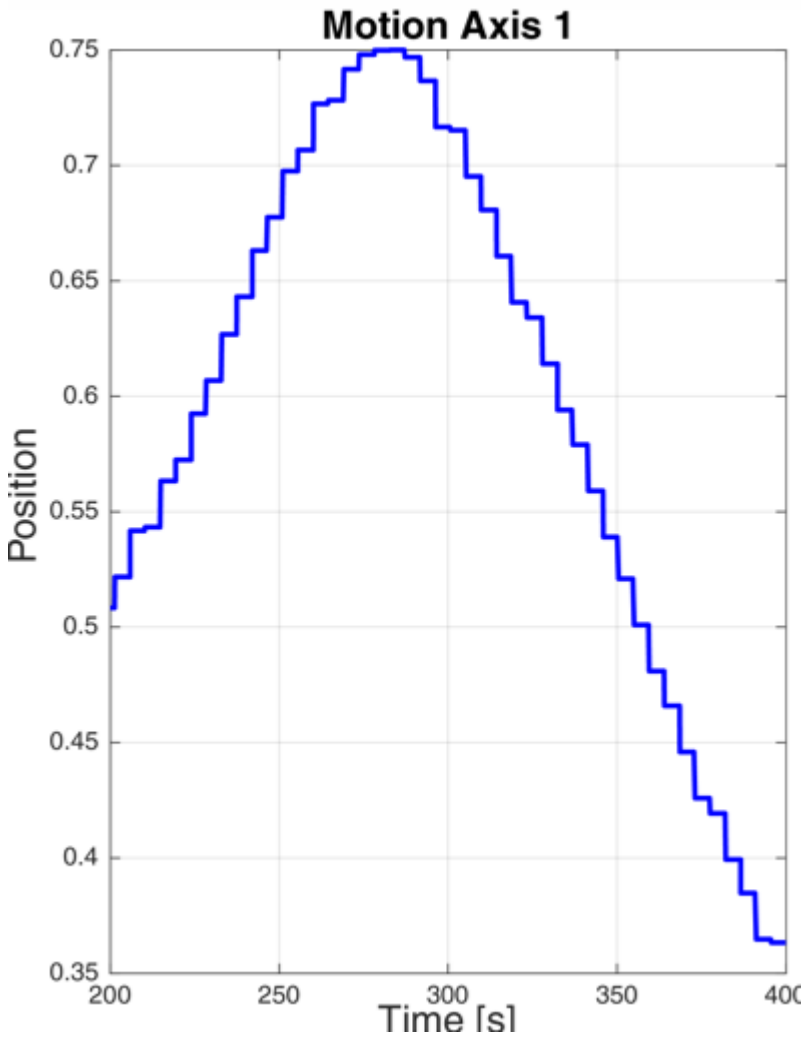


FIGURE 60: OUTPUT POSITION OF AXIS 1

4.4 INSPECTION OF THE HDL CODE GENERATED

During this section a deeper look on how the HDL code has been generated and which are the resources used. This will help to make a reasonable conclusion of the positive and negative aspects of the HDL Coder MATLAB too.

4.4.1 OVERVIEW OF THE CODE

The code generated was analized in order to see how the conversion was done. HDL Coder generated a folder with the name we decided in the first step of the HDL Advisor. Inside of this folder, we can find two subfolders called *fil*_prj and *hdlsrc*.

The first step was to identify how the hierarchy was defined, this can be found inside the *hdlsrc* > *'nameofsimulation'.* In this folder, we can see all the different components that HDL Coder converted separately. The main subsystem converted was named 'Subsystem1' and if we open the .vhd archive, we find at the beginning the name of the file and when it was generated and also which version of MATLAB and HDL Coder was used.

```
-- File Name: 5\hdlsrc\fixPoint_v5_full_SS\Subsystem1.vhd
-- Created: 2017-09-08 15:21:49
--
-- Generated by MATLAB 9.2 and HDL Coder 3.10
--
```

FIGURE 61: FIRST LINES OF .VHD CODE

It is possible to see how this system is defined as the zero level of hierarchy:

```
-- -------------------------------------------------------------
--
-- Module: Subsystem1
-- Source Path: fixPoint_v5_full_SS/Subsystem1
-- Hierarchy Level: 0
--
-- -------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
```

FIGURE 62: HIERARCHY DEFINITION OF MAIN SUBSYSTEM

If we choose some of the others subsystems as the subsystem which enclose one of the functions, we can check it is in the next level. This is exactly what was expected because they are inmediately after the subsystem1.

```
-- -------------------------------------------------------------
--
-- Module: Subsystem
-- Source Path: fixPoint_v5_full_SS/Subsystem1/Subsystem
-- Hierarchy Level: 1
--
-- -------------------------------------------------------------
```

FIGURE 63: SUBSYSTEM AFTER MAIN SUBSYSTEM

From the main subsystem (Subsystem1.vhd) we can notice more important details as the clock frequency at which the system is working.

```
-- -------------------------------------------------------------
-- Rate and Clocking Details
-- -------------------------------------------------------------
-- Model base rate: 0.005
-- Target subsystem base rate: 0.005
--
--
-- Clock Enable  Sample Time
-- -------------------------------------------------------------
-- ce_out        0.005
-- -------------------------------------------------------------
--
```

FIGURE 64: RATE AND CLOCKING DETAILS

The system clock is running at the same frequency of the simulation (20Khz) and not at the 50MHz that FPGA works. This means the usage of the capabilites of the FPGA is reduced to 0.0004%, which is totally inefficient. This observation can be done tracking the enable signals through all the subsystems and it can be seen they are only working at this sample time.

Following the examination of the code, it was possible to understand the code generation related to the Subsystems and the State Machines because HDL Coder made an almost one to one conversion. It started with the definition of the inputs and ouputs of the MOT subsystem as it can be seen in the Fig.65. It defined them as inputs or outputs and also the class.

```
ENTITY MOT_AXIS_1 IS
  PORT( clk                               :   IN    std_logic;
        reset                             :   IN    std_logic;
        enb                               :   IN    std_logic;
        Sender3i                          :   IN    std_logic_vector(31 DOWNTO 0);
        SetPoint3i                        :   IN    std_logic_vector(31 DOWNTO 0);
        Status3i                          :   IN    std_logic_vector(31 DOWNTO 0);
        b                                 :   IN    std_logic_vector(31 DOWNTO 0);
        taxes                             :   IN    std_logic_vector(31 DOWNTO 0);
        PulseGen                          :   IN    std_logic;  -- ufix1
        Sender3o                          :   OUT   std_logic_vector(31 DOWNTO 0);
        SetPoint3o                        :   OUT   std_logic_vector(31 DOWNTO 0);
        Status3o                          :   OUT   std_logic_vector(31 DOWNTO 0);
        alphaINIT                         :   OUT   std_logic;
        alphaREADY                        :   OUT   std_logic;
        alphaRUN                          :   OUT   std_logic;
        e                                 :   OUT   std_logic_vector(31 DOWNTO 0)
        );
END MOT_AXIS_1;
```

FIGURE 65: DEFINITION OF ENTITIES OF COM

Then, it is possible to see (Fig.66) the if statements that are used to move from one State to another:

```
WHEN IN_READY =>
  IF temporalCounter_i1_temp >= to_unsigned(16#C8#, 8) THEN
    READY_reg_next <= '0';
    is_MOT_AXIS_1_next <= IN_RUN;
    temporalCounter_i1_temp := to_unsigned(16#00#, 8);
    RUN_reg_next <= '1';
    e_reg_next <= to_signed(4194304, 32);
    Sender3o_reg_next <= to_signed(0, 32);
    SetPoint3o_reg_next <= SetPoint3i_signed;
    Status3o_reg_next <= to_signed(0, 32);
  ELSIF Status3i_signed = to_signed(4194304, 32) THEN
    READY_reg_next <= '0';
    is_MOT_AXIS_1_next <= IN_ready_proc;
    temporalCounter_i1_temp := to_unsigned(16#00#, 8);
  END IF;
WHEN IN_RUN =>
  Sender3o_reg_next <= to_signed(0, 32);
  IF temporalCounter_i1_temp >= to_unsigned(16#FA#, 8) THEN
    RUN_reg_next <= '0';
    is_MOT_AXIS_1_next <= IN_INIT;
    INIT_reg_next <= '1';
  ELSIF Status3i_signed = to_signed(4194304, 32) THEN
    RUN_reg_next <= '0';
    is_MOT_AXIS_1_next <= IN_run_proc;
    temporalCounter_i1_temp := to_unsigned(16#00#, 8);
```

FIGURE 66: IF STATEMENTS RELATED TO THE STATES OF THE MOT STATE MACHINE

This kind of translation it is quick to check but not the most optimal. Then, it was possible to check the connections that were made but this part is more difficult to understand.

But the biggest problem started when the functions were checked. The HDL Coder creates really complex code lines when a mathematical operation has to be implemented, as the one that can be seen below:

```
 ELSE
      cast_5 := unsigned(mul_temp_1(59 DOWNTO 44)) + ('0' & (mul_temp_1(43) AND (( NOT mul_temp_1(75)) OR
(mul_temp_1(42) OR mul_temp_1(41) OR mul_temp_1(40) OR mul_temp_1(39) OR mul_temp_1(38) OR mul_temp_1(37)
 OR mul_temp_1(36) OR mul_temp_1(35) OR mul_temp_1(34) OR mul_temp_1(33) OR mul_temp_1(32) OR mul_temp_1(31)
OR mul_temp_1(30) OR mul_temp_1(29) OR mul_temp_1(28) OR mul_temp_1(27) OR mul_temp_1(26) OR mul_temp_1(25)
OR mul_temp_1(24) OR mul_temp_1(23) OR mul_temp_1(22) OR mul_temp_1(21) OR mul_temp_1(20) OR ...
```

FIGURE 67: MATHEMATICAL OPERATIONS IMPLEMENTED IN HDL

These type of lines also mean a great use of material FPGA resources. To check which resources are used in each entity, we can take a look at the "Analysis & Synthesis Resource Utilization by Entity" inside of the map file. We can find that 7 DSP (Digital Signal Processing) blocks are used for this only function, which is just a simple linear interpolation. This function is repeated twice which makes 14 DSP plus the 18 used on the other function, a total of 32 DSP blocks.

4.4.2 FPGA USE OF MATERIAL RESOURCES

In order to analize the use of material resources, 2 tests more were performed. The Subsystem used in the first conversion was copied one time in the first test and two in the second. The logic behind this process is to have a clear view of the evolution of the resources employed, we wanted to know if the resources will double or the rise will be not so significative.

FEATURES	1 Axis	2 Axis	3 Axis
Family	Cyclone V	Cyclone V	Cyclone V
Device	5CSEMA5F31C6	5CSEMA5F31C6	5CSEMA5F31C6
Logic Utilization (in ALMs)	9291 / 32070 (29%)	13,789 / 32,070 (43 %)	18,432 / 32,070 (57 %)
Total registers	6106	8884	11613
Total pins	1 / 457 (< 1%)	1 / 457 (< 1 %)	1 / 457 (< 1 %)
Total block memory bits	153488 / 4065200 (4%)	160,784 / 4,065,280 (4 %)	168,080 / 4,065,280 (4 %)
Total RAM Blocks	50 / 397 (13%)	61 / 397 (15 %)	73 / 397 (18 %)
Total DSP Blocks	32 / 87 (37%)	64 / 87 (74 %)	87 / 87 (100 %)
Total PLLs	1 / 6 (17%)	1 / 6 (17 %)	1 / 6 (17 %)

TABLE 9: COMPARISON OF THE FPGA USAGE OF RESOURCES IN THE DIFFERENT TESTS

As it can be observed in Table 9 the results were not fully positive. Specifically in two features: Logic Utilization in ALMs (Adaptive Logic Modules) and Total DSP Blocks. The value of Logic Utilization was already high and its increase (not linear) let it in a use of more than 55%. The second one, DSP Blocks has become exactly the double in the second test and arrived to 100% in the last one, which can cause problems in the correct implementation of the FPGA. We are considering values relatively high since 40% because the program developed is not large, so that could mean a not optimised use of the FPGA resources by the HDL converter. The first two conversions lasted around 20-30 minutes but the last test took almost 3 hours.

These tests were also useful to understand where the resources of the FPGA were used. The DSP blocks are most probably dedicated to generate the subsystem that we have converted (in our

case Subsystem1), our Axis. The ALMs are used for the set up configuration (JTAI, system initialization…) and also for the Subsystem. For this reason, the increase of the last ones is not doubled every time.

It is also considered important to mention that the 3 conversions finished succesfully without errors but the number of warnings was being increased, most of them related to timing constraints.

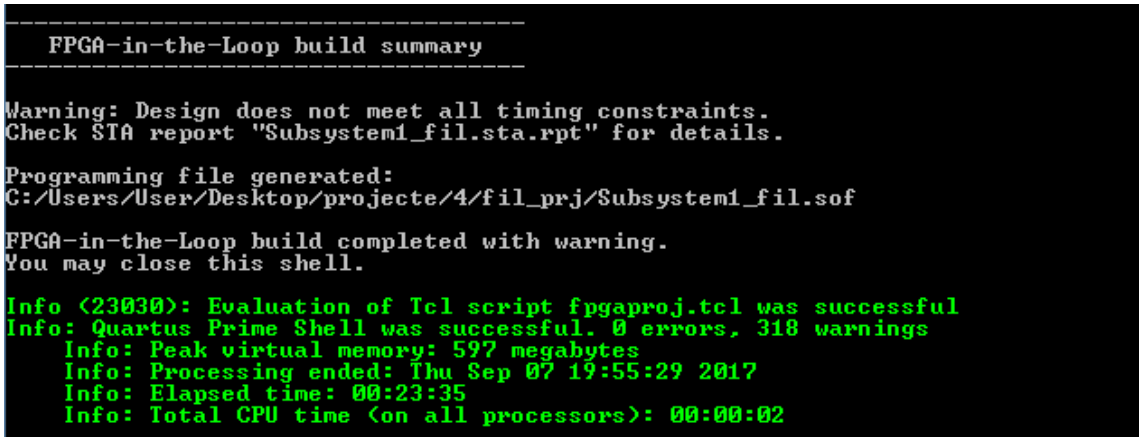


FIGURE 68: CONVERSION FINISHED SUCCESSFULLY BUT WITH WARNINGS

4.4.3 TIME CONSTRAINTS

These timing constraints warnings were considered important to analyze. Hence, the 'TimeQuest Timing Analyzer report" was examined in deeper detail. This report contains a lot of warning but what is more worrying is the high negative slack:

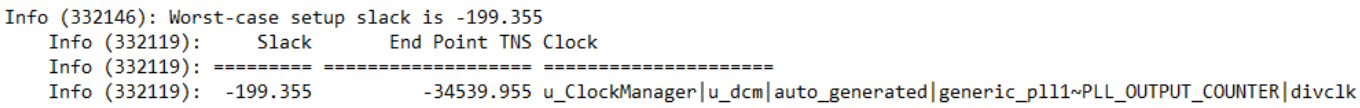


FIGURE 69: HIGH SLACK FOUND IN THE TIME REPORT

"Slack time can be defined as the amount of time an task can be delayed without causing another task to be delayed or impacting the completion date of your project."

TechTarget, September 2017

In terms of our system, it refers to whether timing is met along a timing path. If this slack is positive it means that the signal can get from the beginning until the end of the timing path fast enough for the circuit to operate correctly. [30] If the slack is negative, which is our case, can be understood as an inability of the signal to reach the end of the timing path in a correct time to ensure correct circuit operation. For this reason, a negative slack adds randomness to a system and makes the design unacceptable in terms of reliability.

5 BUDGET AND ENVIRONMENTAL IMPACT

5.1 BUDGET

The cost of the development of this thesis can be divided between the material/software cost and the cost of the human resources employed during the project.

	Academic Cost	**Commercial Cost**
FPGA Board	145€	210€
MATLAB + Simulink Software	Free	5000€
Quartus II Lite	Free (Trial Version)	2800€
Total Cost	145€	8010€

TABLE 10: MATERIAL AND SOFTWARE COST

The duration of the project has been 6 months and the time invested by the 6,5h/working day.

Person	**Hours invested [h]**	**Price / hour [€/h]**	**Cost [€]**
Student	780h	15€/h	11700€
Professor	30h	40€/h	1200€
		Total Cost	12900€

TABLE 11: HUMAN RESOURCES COST

The total cost of the project is 145€ + 12900€ = 13045€. In a non-academical environment, this cost would rise until 20910€, a remarkable increase.

5.2 ENVIRONMENTAL IMPACT

The supplies used during this project are mainly related to software and human resources. For this reason, the level of environmental impact is really low.

The only significant material with a high environmental impact is the FPGA Board. This board needs a lot of electronic components which can have a deep impact on the environment during its production and, at the end of their life, if they are not recycled in an appropriate way. However, this board is totally reprogrammable and has an academic purpose, so we can assure high numbers of usages before its withdrawal, which would not happen in the normal microcontrollers.

6 SUMMARY, CONCLUSIONS AND FUTURE GOALS

6.1 SUMMARY AND CONCLUSIONS

This chapter summarizes the work done during this Master Thesis and gives a general view of the most important points. It will also explain which would be the direction to continue working.

The thesis started with the information gathering that is summarized in the first chapter, the State of the Art. In it some negotiation techniques are discussed, which have been later applied: the architecture chosen is hybrid, axes act as mainly deliberative because they have total knowledge of their limits and local master is reactive because it uses the information given by the axis and reacts in base of this. The error state it is a good example of this hybrid approach as it can be deliberative or reactive, if the axes are the ones having the error it will be deliberative because it will come from their knowledge of the world but if this error comes from another axes, they will react and stop the system. The system architecture considered is the federated with the external master acting as a mediator. Axes respond to the requests of the master but they have their own capabilities. The communication through messages can be characterized as a high level of communication, where axes only give part of the information they have (they do not share their capabilities). The system studied is direct, synchronous and has point-to-point communication/broadcast communication. Broadcast from local master to axes and rest point-to-point.

In this chapter, there is also a brief review of what is an FPGA (target hardware of the thesis), how it works and which is its programming language, HDL. It ends with a concise explanation of MATLAB, the main software used and the tools that will be necessary. This chapter meet the goal of providing a context to the frame of the thesis and the steps that will be implemented but a deeper research of MATLAB tool has needed to be done due to the errors occurred during the next chapter.

The next chapter, System modelling, gives an overview of the two systems implemented: a CNC Machine and distributed simulation. The first one is used as a basis for the development of the second, the main goal of the chapter. This is the simulation that will be converted to HDL but in order to implement it, it is necessary to define a negotiation protocol, which has to be simple and effective. The objectives were accomplished with the implementation of a simulation and a protocol plain and fast enough.

The fourth chapter explains how the distributed simulation was converted to HDL and C code. This chapter was the most challenging and time-consuming of all. The conversion to C using MATLAB Coder it is quite direct as it is widely used tool, therefore, it is well-optimized. Nevertheless, the conversion from Simulink to HDL Coder requires of a high level of knowledge about the HDL Coder tool and also the Fixed-Point tool. Furthermore, the employment of these applications requires to low the simulation of level, meaning that not even a quarter part of the resources of Simulink / MATLAB can be used and, for instance, functions have to written without the help of any subfunction as maximum, minimum or absolut.

Particularly, the Fixed-Point Tool shows an unexpected bad performance because their conversions have to be manually modified in order to achieve a successful simulation without errors.

For these later reasons, the simulation of chapter 3 and 4 have almost the same result but their implementations differ in many ways. This FPGA system is bigger and more difficult to understand but matches the main goal of the project.

Then, a small explanation of the implementation is done and its results are examined. In this last step, it was observed an unexpected used of a high number of the FPGA resources by the HDL Coder, which can implicate a lack of optimizing during this conversion.

The last chapter is a small summary of the budget and the environmental impact of this thesis. None of them are remarkable in this scenario but it is interesting to notice that the budget will increase enormously if the project was not carried out in an academic context.

To sum up, the goals of the project were accomplished but the conversion was not as easy or successful as it was expected, resulting in a lack of time to perform wider experiments. The MATLAB tools used require having some experience working with them and a big change from the way simulations are implemented in Simulink. If this project was started again, the author would have designed the simulation on a completely different way, thinking about the problems occurred. Finally, the lack of optimization on the existing tool resulted in a big resource dedication solving secondary issues, less resources were possible to dedicate to the optimization of the conversion.

6.2 FUTURE WORKS

The next steps of this research could have different directions:

- Continue testing different simulations in order to see how the FPGA resources evolve and change the parameters of the HDL implementation as pipeling/optimization to check how they affect into the conversion until finding a good one. It is also necessary to test if the HDL code generated by MATLAB can be used without this application.
- Check if it is possible to use the clock of the FPGA and not the clock of the simulation, which will increase the use of the capabilities of the FPGA and could reduce the timing problems.
- Solve the timing constraint problems in order to increase the reliability of the system.
- Design a simulation from the beginning taking into account the issues encountered along the development of the thesis.
- Implement the distributed interpolation directly in HDL.

7 BIBLIOGRAPHY

[1] *What Is Distributed Control System (DCS)? Definition and Meaning.* BusinessDictionary.com. URL: http://www.businessdictionary.com/definition/distributed-control-system-DCS.html. 11 May 2017.

[2] *The Future of Control Systems and Industry 4.0*. Wnipt Mast Web. URL: http://processonline.com.au/content/industrial-networks-buses/article/the-future-of-control-systems-and-industry-4-0-1365495420#ixzz4gZNBTPzJ. 11 May 2017.

[3] Shen, Weiming; Norrie, Douglas H.; Barthès, Jean-Paul. *Multi-agent systems for concurrent intelligent design and manufacturing.* 2013. CRC press. 1 April 2017.

[4] Professor Jeffrey S. Rosenschein. "PowerPoint Lecture Slides for An Introduction to Multiagent Systems." An Introduction to Multiagent Systems/Mike Wooldridge/Powerpoint Slides, Hebrew University of Jerusalem, www.cs.ox.ac.uk/people/michael.wooldridge/pubs/imas/distrib/powerpoint-slides/. 5 April 2017.

[5] Tsovaltzi, Dimitra et al. "Searle's Classification of Speech Acts." 3.1.3 Searle's Classification of Speech Acts, Computational Linguistics and Phonetics, www.coli.uni-saarland.de/projects/milca/courses/dialogue/html/node66.html. 5 April 2017.

[6] Florea, Adina Magda. "Introduction to Multi-Agent Systems". International Summer School on Multi-Agent Systems, Bucharest, 1998. 6 April 2017.

[7] Cui, R., Guo, J., & Gao, B. (2013). Game theory-based negotiation for multiple robots task allocation. Robotica, 31(6), 923-934. doi:10.1017/S0263574713000192. 11 May 2017.

[8] Satell, Greg. "A Guide to Game Theory and Negotiations." Digital Tonto, Nov. 2009, www.digitaltonto.com/2009/game-theory-guide-to-negotiations/. 13 May 2017.

[9] Bradshaw, J.M., Dutfield, S., Benoit, P. and Woolley, J.D. *KAoS: Toward an industrial-strength open agent architecture.* Software Agents, AAAIIMIT Press. 1997. pp. 375-418. 14 May 2017.

[10] Scherer, Thomas. 50 Years of Moore's Law. 2015. Elektor. URL: https://www.elektormagazine.com/articles/moores-law. 15 May 2017.

[11] Eastland, Nate. Structure of an FPGA. Digilent Inc. Blog. 13 Aug. 2015. URL: https://blog.digilentinc.com/structure-of-an-fpga/. 11 May 2017.

[12] Jones, David. *What is an FPGA?* 2013. Youtube. URL: https://www.youtube.com/watch?v=gUsHwi4M4xE. 01 May 2017.

[13] Farroq, U; Marrakchi, Z; Mehrez, H. Three-based FPGA Architectures Application Specific Exploration and Optimization. 2012. Springer. 02 May 2017.

[14] *Hardware Description Language.* 2014. Mepits. URL: https://www.mepits.com/tutorial/143/VLSI/Hardware-Description-Language. 11 May 2017.

[15] Smith, Steven W. *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego, CA: California Technical Publ., 1999.

[16] *Fixed-Point vs. Floating-Point Digital Signal Processing*. AnalogDevices. URL: http://www.analog.com/en/education/education-library/articles/fixed-point-vs-floating-point-dsp.html. 11 May 2017.

[17] *C Language Overview*. www.tutorialspoint.com. 2013. URL: https://www.tutorialspoint.com/cprogramming/c_overview.htm. 11 May 2017.

[18] Chou, Bill. *MATLAB to C Made Easy*. 2016. MathWorks. URL: https://www.mathworks.com/videos/matlab-to-c-made-easy-81870.html

[19] Sinha, Udayan; Anderson, Robert. *Targetting MATLAB Algorithms to FPGAs*. MathWorks. URL: https://www.mathworks.com/videos/targeting-matlab-algorithms-to-fpgas-81968.html. 01 May 2017.

[20] Van Beek, Stephan. *Using HDL Coder and HDL Verifier for FPGA and ASIC Designs*. MathWorks. 2012. URL: https://de.mathworks.com/videos/using-hdl-coder-and-hdl-verifier-for-fpga-and-asic-designs.html. 01 May 2017.

[21] *C Code Generation from Simulink* - MATLAB & Simulink, MathWorks, URL: html https://www.mathworks.com/help/rtw/gs/model-and-simulation.html. 20 August 2017.

[22] *Generate C Code for Model*- MATLAB & Simulink, MathWorks, URL: https://de.mathworks.com/help/rtw/gs/model-and-simulation.html. 20 August 2017.

[23] *Control Chart Execution Using Temporal Logic* - MATLAB & Simulink, MathWorks, URL: https://www.mathworks.com/help/stateflow/ug/using-temporal-logic-in-state-actions-and-transitions.html. 5 August 2017.

[24] *Fixed-Point numbers*- MATLAB & Simulink, MathWorks, URL: https://de.mathworks.com/help/simulink/ug/fixed-point-numbers.html#bu7krc6-1. 15 August 2017.

[25] *Simulink Functions in Stateflow* - MATLAB & Simulink, MathWorks, URL: https://de.mathworks.com/help/stateflow/ug/simulink-functions-in-stateflow.html. 20 August 2017.

[26] *Convert Floating-Point to Fixed Point*. - MATLAB & Simulink, MathWorks, URL: https://de.mathworks.com/help/fixedpoint/ug/tutorial-steps.html. 28 August 2017.

[27] *User Manual DE1-SoC User Manual* .1 www.terasic.com April 8, 2015. 01 September 2017.

[28] *VHDL Reserved Words."*VHDL Reserved Words, www.csee.umbc.edu/portal/help/VHDL/reserved.html. URL: https://www.csee.umbc.edu/portal/help/VHDL/reserved.html. 10 September 2017

[29] *FIL Simulation with HDL Workflow Advisor.* - MATLAB & Simulink, MathWorks, URL: https://de.mathworks.com/help/hdlverifier/ug/fil-simulation-with-hdl-workflow-advisor-for-simulink.html. 10 September 2017.

[30] *What is a negative Slack?.* URL: www.edaboard.com. 20 September 2017.