Embedded Systems Lab

Final Project Report

Design , Simulate & Build !

Riccardo Battistig
@ r.c.m.battistig@student.utwente.nl

Pietro Pennestrì, s2382660
@ p.pennestri@student.utwente.nl

Tags: FPGA Design, Jetson Nano, OpenCV, Color Tracking

Contents

1 Introduction

The project goal is to build a working vision-in-the-loop demo combination using both `C++` and `VHDL` parts.

The C++ computer vision application tracks a green rectangle and returns its center coordinates and rotation within a Cartesian reference system. The application runs on a Jetson Nano board. The FPGA (A-C4E6E10):

- through serial communication receives green rectangle position data from the Jetson Nano.
- Controls the servo motors whose shaft rotation matches the one of the green rectangle.
- Schedules-regulates the request of data from the Jetson Nano.

A physical prototype of the device has been built and tested.

1.1 Report Organization

This report describes the design phases of the prototype main components. The present document is organized as follows:

- Section 2 describes **D**esign **S**pace **E**xploration. Different trade off tables are compiled and commented to evaluate the *best* solution for different components.
- Sections 3 and 2.6 refer to software modules and hardware implementations.
- The Appendices refer to preliminary work requested by assignments[1] (already submitted on Canvas).

1.2 </> Code Source Files

All the codes related to the physical prototype are uploaded on Canvas. Alternatively, the Instructor may wish to clone the entire project folder with git:

```
git clone https://gitlab.com/pietro96/mg996r
```

Whenever necessary, the location of the source files, will be highlighted in a gray color area at the beginning of the section.

2 Design Space Exploration

Before starting our project it has been necessary to list and evaluate different feasible alternatives. In the following we will report some design implementations for the composing parts of our device. For every component, different evaluation categories are considered. The grading scale for category members goes from zero to five. Each category is also characterized by a weighting factor. At the end, each alternative will receive a total score varying from 0 to 100.

⚠ The rationale behind the scores assigned will be provided. However, some of the categories/scores considered/assigned are due to *personal* preferences and (limited) *experience*.

[1] ESL-Assignments-1920v1

2.1 The rationale of Servo Control - PWM Signal Generation

Alternatives	Cost	Dev. Time	Flexibility	Mult. Task	Easiness of Debug	Total Score
555 Timer	+++	+++	--	---	++	46
FPGA (Cyclone IV Family)	+	++	+++	+++	++	**90**
Microprocessor (8051 or AtMEGA)	+++	++	+++	---	++	52
Jetson Nano	---	++	+++	++	+++	81
Gumstix Overo	---	++	+++	+	+++	73
Weighting Factor	1	6	3	8	2	

Evaluation Grid		
+++	5	Good
++	4	
+	3	
-	2	
--	1	
---	0	Bad

Max Score: 100

Figure 1: Trade-off Table of PWM

The following categories were chosen to evaluate the PMW solution:

- Cost;
- Development time;
- Flexibility: the component can be used for other needs in the project
- Parallel: the component can be simultaneously used to perform different actions.
- Easiness of debug.

Three options were considered to realize a pulse width modulation:

- **555 Timer** this IC can be configured to generate a `PWM` signal.
  - 👍 Extremely low cost.
  - 👍 Does not require programming from the user side. Only appropriate wiring is necessary.
  - 👎 Cannot interpret the error signal from the `HBridge`.
  - 👎 Single task execution.
- **FPGA (Cyclone IV)**
  - 👍 Possibility to handle multiple tasks.
  - 👍 Can interpret the error signal from the `HBridge`.
  - 👎 Expensive compared to other options available.
- **Microprocessor**
  - 👍 Low cost device.
  - 👎 Only one task a time.

2.2 The rationale of Computer Vision Platform (Hardware)

Alternatives	Cost	Dev. Time	Flexibility	Mult. Task	Power	Easiness of Debug	Total Score
Personal Computer	---	+++	+++	++	---	+++	70
FPGA (Cyclone IV Family)	++	---	++	+++	+++	+	64
Jetson NANO	+	+++	+++	++	++	+++	89
Gumstix Overo*	-	+++	+++	++	+++	+++	**92**
Weighting Factor	1	6	3	5	4	1	

** Part not available*

Evaluation Grid		
+++	5	Good
++	4	
+	3	
-	2	
--	1	
---	0	Bad

Max Score: 100

Figure 2: Trade-off Table of Computer Vision Platform

- **Personal Computer**
  - 👍 Easiness of development.
  - 👎 Highest Power Consumption.
  - 👎 Highest Cost.
- **FPGA (Cyclone IV)**
  - 👍 Relatively low cost compared to other alternatives. A cheap Cyclone IV family board can be purchased for about 30 €.
  - 👍 Tasks can be easily parallelized .
  - 👍 Low power consumption.
  - 👎 High development time for computer vision application. No standard libraries for computer vision available comparable to OpenCV for `C++`.
- **Jetson Nano**
  - 👍 Medium Cost. Currently selled for about 108 € development board included.
  - 👍 Despite not used in this project, this board can run AI applications. It has in fact, a Maxwell 128 core GPU.
  - 👍 Big users community. The board has been tested on large scale projects.
  - 👍 Runs Ubuntu OS.
  - 👍 Power budget within the range of 10W to 5W. User can set the power profile of the board.
  - 👎 Extra cooling is recommended. In our prototype the standard heat dissipator was replaced with `ICE Tower CPU Cooling Fan`[2]. This improved cooling and performances. Extra cost was about 22 €.
- **Gumstix Overo**
  - 👍 Low power device: 250 mA @ 4V.
  - 👎 *The power management features are still in an experimental branch of Linux-OMAP, so Gumstix hasn't completed any testing*[3].
  - 👍 Can run Angstrom Linux.
  - 👎 Cannot run AI applications.

[2] https://www.seeedstudio.com/ICE-Tower-CPU-Cooling-Fan-for-Nvidia-Jetson-Nano-p-4214.html
[3] Power Specification are taken from `https://www.gumstix.com/images/Overo_Performance_Power.pdf`

2.3 The rationale of Computer Vision Software

Programming Language	OpenCV Support	GStreamer Support	Speed	Total Score
C	✗	✓	+++	50
C++	✓	✓	+++	100
Python	✓	✓	+	90
Weighting Factor	10	5	5	

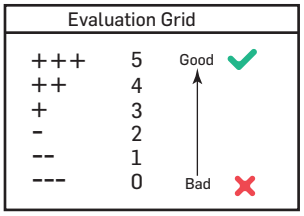


Figure 3: Trade-off Table for Computer Vision Software

The categories for evaluating the programming language used to implement the computer vision application are the followings:

- **OpenCV Support**: Most of the features of our computer vision app rely upon OpenCV functions. This means that the chosen programming language should have updated and maintained API support for this library.
- **GStreamer Support**: *is a development framework for creating applications like media players, video editors, streaming media broadcasters and so on.* [4]
- **Speed**: Compiled programming languages are preferred.

We will mention the rationale behind the scoring in Figure 3:

- **C**
  - 👍 Compiled language.
  - 👍 Extensively used in embedded systems.
  - 👎 `C` API for OpenCV are dead since 2010 [5].
- **C++**
  - 👍 Compiled language.
  - 👍 Supports OpenCV.
- **Python**
  - 👍 Supports OpenCV.
  - 👍 Easy to use.

[4] https://gstreamer.freedesktop.org/documentation/frequently-asked-questions/general.html?gi-language=c
[5] https://answers.opencv.org/question/185693/is-c-api-alive-or-not/

👎 Interpreted language.

However if we wish to develop our computer vision app directly on `FPGA`, since an HD language [6] cannot be directly compared to a programming language, the evaluation categories previously chosen can be only partially applied. Neither OpenCV nor GStreamer offer support for any HD language. At the time this report is written, few tools exist for image processing for FPGAs. Because of the possibility offered by FPGAs to parallelize computations, we assign a total score of 25 to any HD language.

2.4 Communication

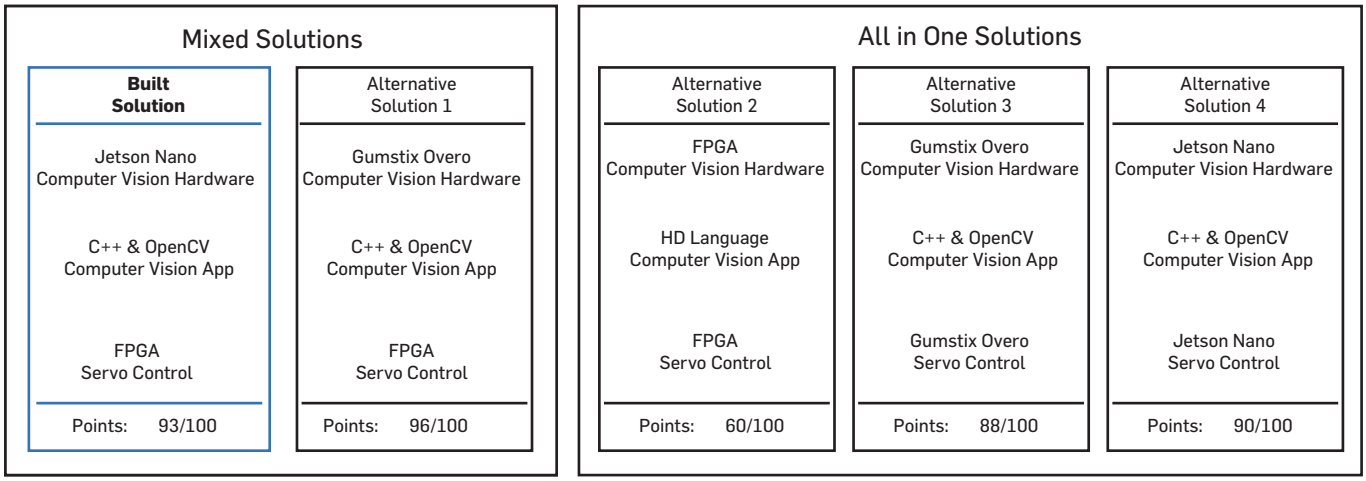
A wide set of communication approaches are available. However, to keep our hardware as simple as possible, only to alternatives are explored within our design choices:

- **Serial Communication**: this alternative requires extra hardware. Despite this limitation, such solution can be implemented in different scenarios.
- **gpio communication**: from the hardware point of view this alternative appears as the simplest one. However, the Jetson Nano currently does not offer official gpio support in `C++`

In our project, we will choose serial communication as the preferred method for exchanging data between different boards.

[6] Hardware Description language

2.5 Summary of Alternative Solutions



	PWM Servo Control	CV Platform (HD)	CV Software	Total
A. Solution 1	90	92	100	96
B. Solution	90	89	100	93
A. Solution 2	90	64	25	60
A. Solution 3	73	92	100	88
A. Solution 4	81	89	100	90
Weight	1/3	1/3	1/3	

Each Category as a weight of 1/3. Max. Points = 100

Figure 4: Comparisons of solutions

In the previous sections, the alternatives for the software and hardware components have been one by one rated. Different combinations of these components are evaluates as in Figure 4. According to our judging criteria, *Alternative Solution 1* scored best. However, due to the lack of the Gumstix Overo board at our disposal, the *Built Solution* was the one adopted for the physical prototype.

2.6 Hardware for the Physical Prototype

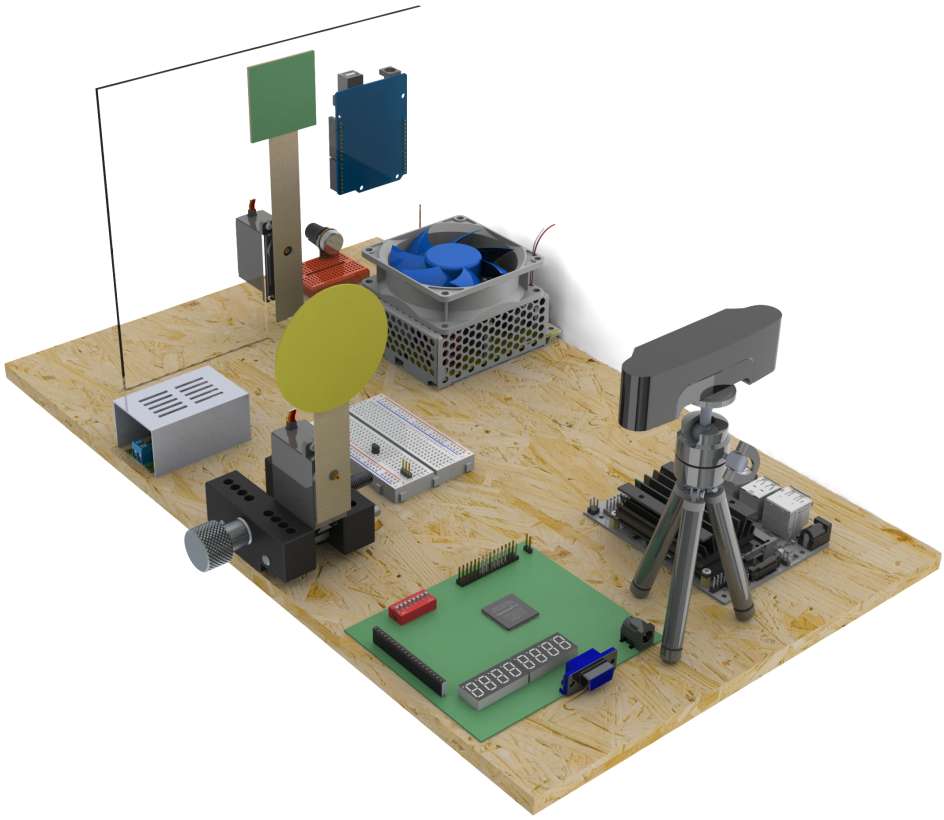


Figure 5: Rendering of the final prototype

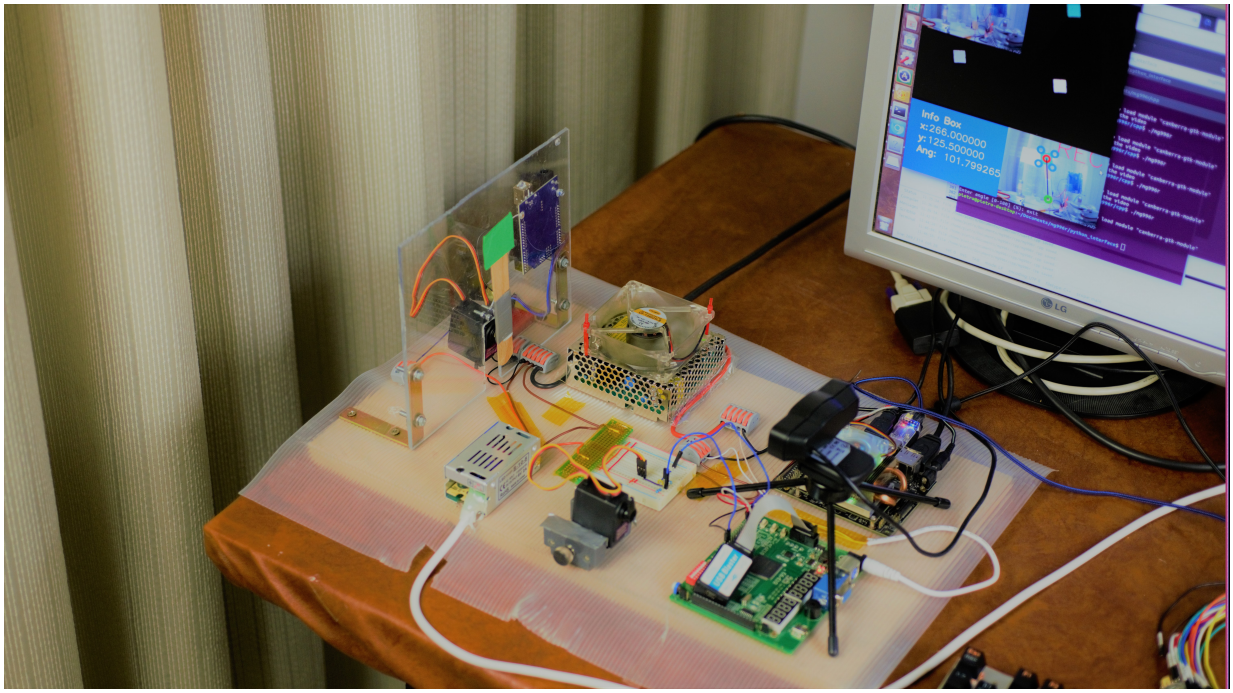


Figure 6: Picture of the prototype

A render view of the prototype and a picture of the real implementation are respectively shown in Figures 5, 6. The components used for building the demo are shown in Figure 7.

Component	Quantity
Jetson Nano	1
EP4CE6E22C8N	1
Arduino Uno	1
Mini Breadboard	1
Half Breadboard	1
2N7000 - NMOS Transistor	1
10k Resistor	2
10k Potentiometer	1
MG996R Servo Motor	2
Webcam	1
Mini Camera Tripod	1
5V - 3A Power Supply	1
5V - 2A Power Supply	1
Cooling Fan	1
Watch Case Holder	1

The hardware list also includes an `Arduino` board and a potetiometer. These components are used to manually control the servo than moves the green rectangle. The user by turning the potentiometer knob, will vary the green rectangle angular position.

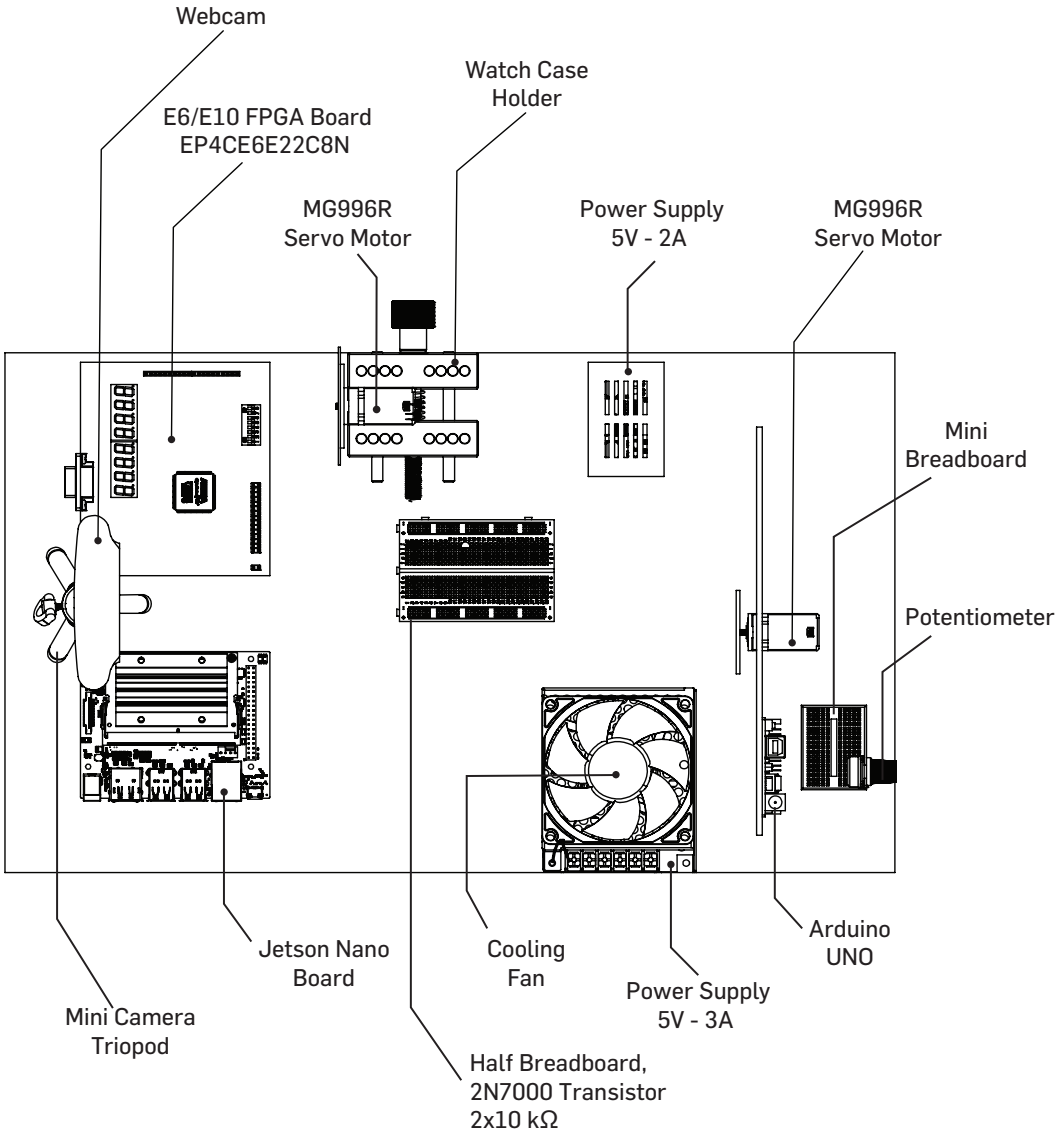


Figure 7: Components of the final prototype

3 Software Modules for the Physical Prototype

Demo Video A video showing the home built physical prototype, outcome of the present project, can be found at:

https://youtu.be/l34daSNVviE

File Reference

- IOMG996R (**Top Level Entity**): ./mg996r/IOmg996r.vhd
  - Testbench: ./mg996r/tb_IOmg996r.vhd
- MG996R: ./mg996r/mg996r.vhd
  - Testbench: ./mg996r/tb_mg996r.vhd
- UART TX Module: ./mg996r/uartTX.vhd
  - Testbench: ./mg996r/tb_uartTX.vhd
- UART RX Module: ./mg996r/uartRec.vhd
  - Testbench: ./mg996r/tb_uartRec.vhd
- Computer Vision Application: ./mg996r/cpp/mg996r.cpp

Clone GIT Project

https://gitlab.com/pietro96/mg996r/

This GIT project contains **all** the code developed to realize the physical prototype.

Since no embedded systems project is complete without a prototype, we build one. This activity allowed us:

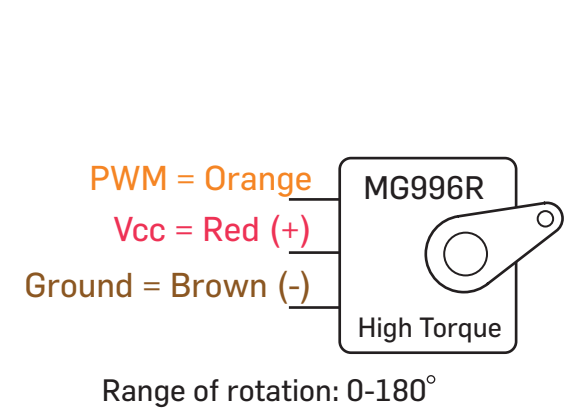
- to test out code not only in simulation, but also on the field;
- to have hands on experience on how VHDL and C++ codes work together.

The main functions of our prototype are the followings:

- Locate a green rectangle and estimate its center coordinates and rotation with respect to an arbitrary reference system. This task is done thanks to an OpenCV application written in C++ (see cpp folder). The application runs on Jetson Nano board.
- The rotation of the green rectangle is replicated by a servo motor controlled by an FPGA (A-C4E6E10). The Jetson Nano communicates the position of the green rectangle to the FPGA Board through serial communication.

Due to the tight budget, limited availability of components and tools some design choices have been almost obliged and are not optimal. For example, the motors available do not have a feedback signal accessible to users. Thus they rely upon a factory design feedback control system.

3.1 MG996R



FPGA Clock = 50 MHz

		No. Clock Cycles	
		Integer	HEX
Period	3500 µs	175000	2AB98
Min. Servo High	700 µs	35000	88B8
Max. Servo High	2400 µs	120000	1D4C0
Actuating Time	15000 µs	750000	B71B0

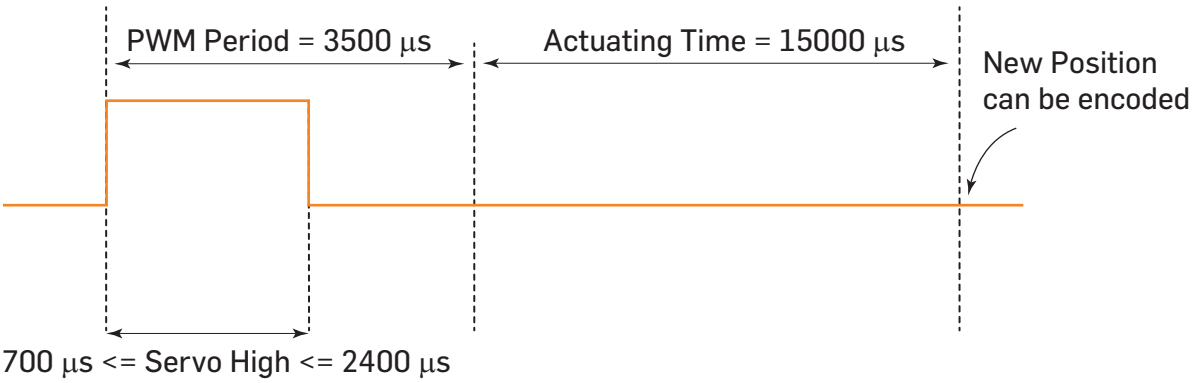


Figure 8: Experimental Characterization of the MG996R servo

The `MG996R` is the servo motor available for this application. It has 3 external terminals: 2 for the power supply and 1 for a `PWM` which encodes the position. It also has a feedback signal from a potentiometer, but this cannot be accessed from the user. A feedback controller is embedded in the `IC` wired with the servo. This device is low cost and mainly used for modelism applications. The high torque `MG996R` servo has been experimentally characterized, the results are summarized in Figure 8. The figures obtained during the characterization are used in the `MG996R VHDL` module

The `MG996R VHDL` unit is designed to work in pair with an UART unit and it has the following port mapping:

Input Ports

- `clk` : clock input for the unit;
- `rst` : active low reset signal;
- `set_position` : this signal sets the position of the servo. The user sets for how many clock cycles the `PWM` signal should stay high;
- `go` : if this signal is not raised high the servo does not update its position;
- `newDataRec` : if raised high a new position was sent to from the `UART`.

Output Ports

- `sendNewData` : if raised high the servo is asking for a new position. As soon as `newDataRec` is raised high, `sendNewData` is raised low.
- `ready` : this signal is raised high when the servo has generated the `PWM` signal and the actuating time has elapsed.
- `bad_position` : if raised high a bad position was given to the servo. The `MG996R` can only turn from 0 to 180 degree.
- `pwm`: the pulse width modulation signal.

Simulations of the `MG996R` module are depicted in Figures 9, 10 and 11.

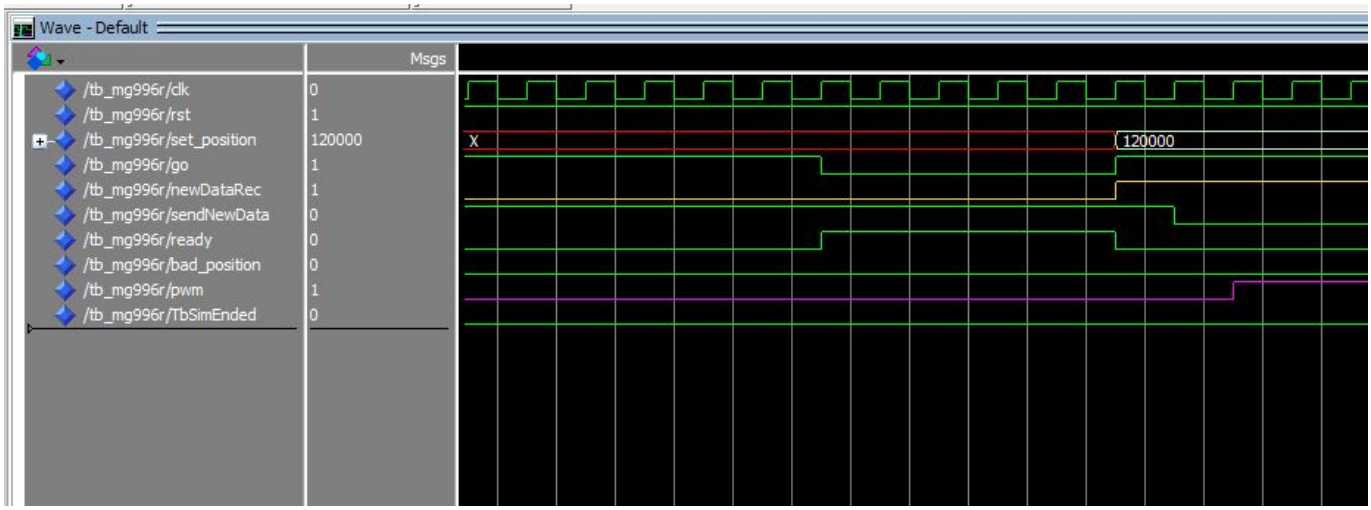


Figure 10: Detail of Simulation of the `MG996R` VHDL Module

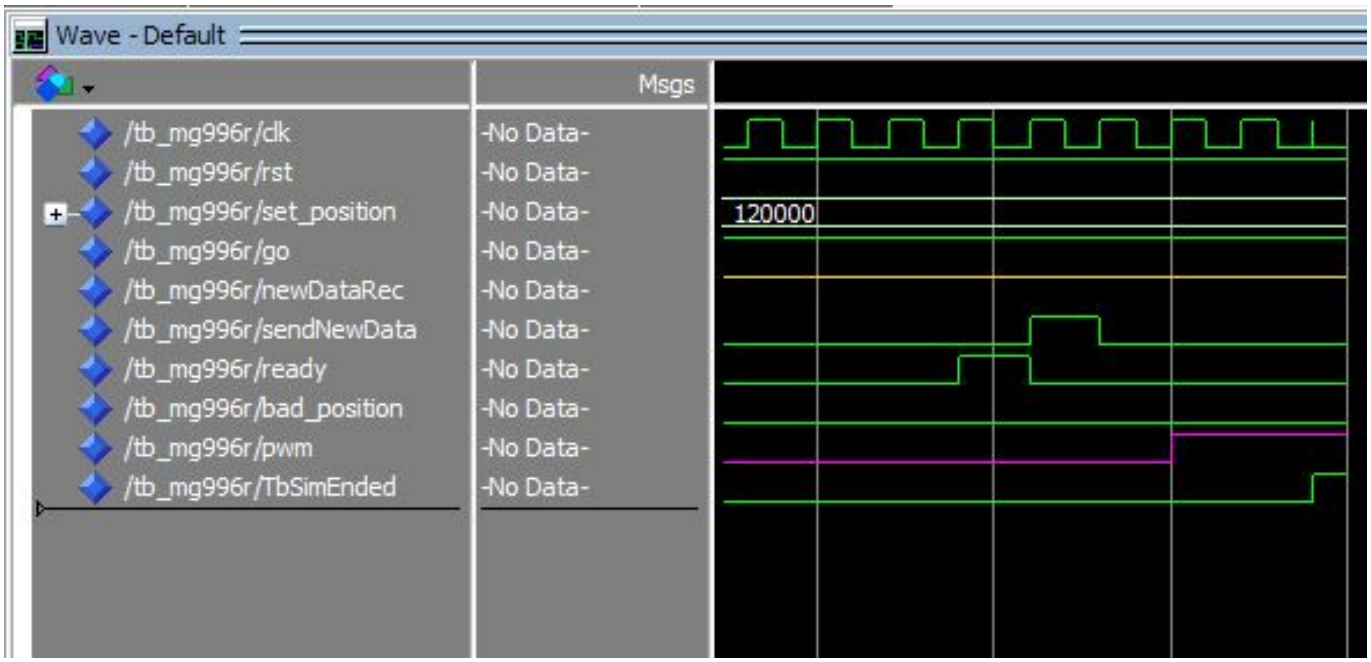


Figure 11: Detail of Simulation of the `MG996R` VHDL Module

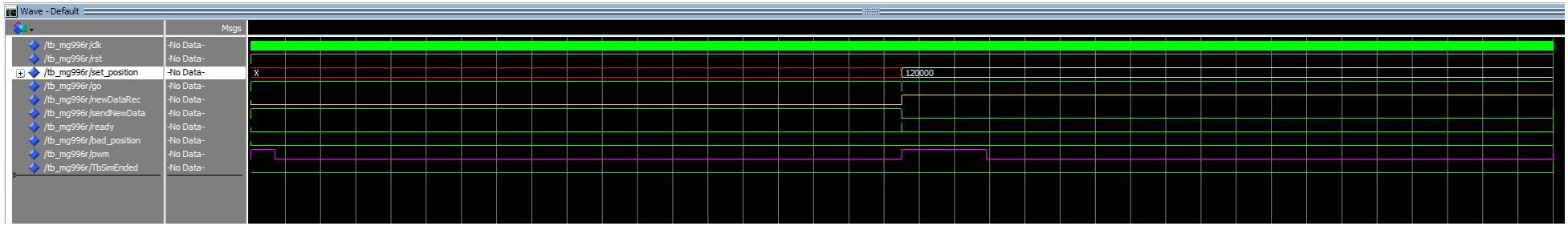


Figure 9: Simulation of the `MG996R` VHDL Module

3.2 The UART Interface

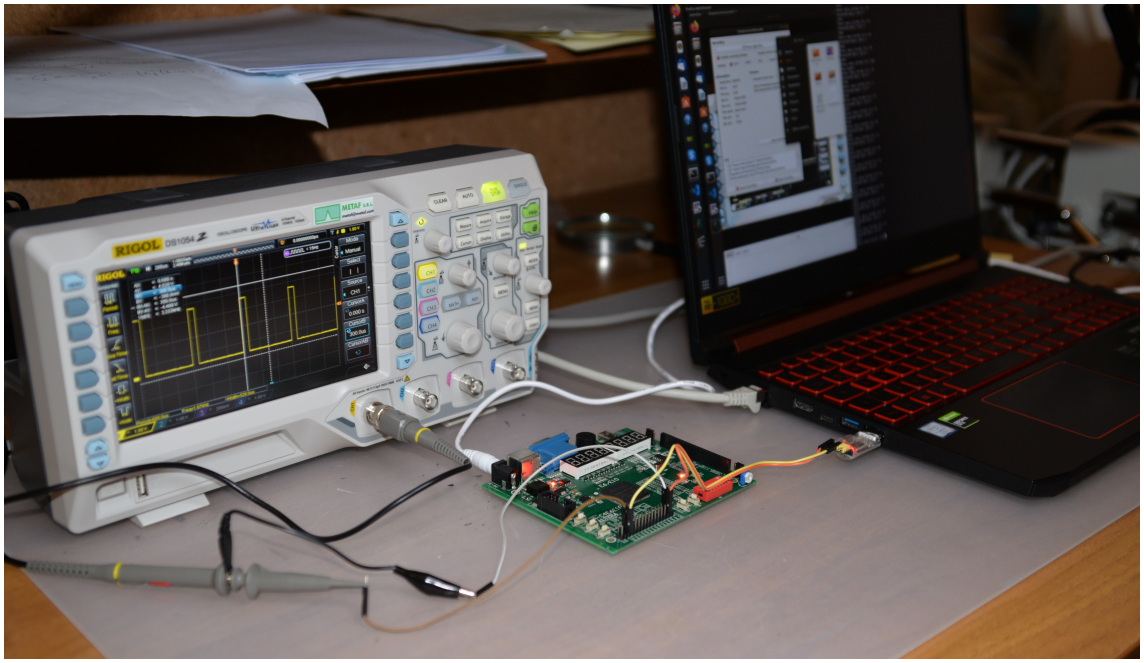


Figure 12: Debug of `UART` and `PWM` interfaces

An UART module composed of 2 units (transmitter and receiver) to communicate with the Jetson Nano has been developed. The module has a baud rate of 9600 bits per second (the baud rate can be modified by changing the generic of the two units). The transmission unit has the following port mapping:

UART TX Port Mapping

Input Ports

- `clk`: clock signal;
- `rst` : active low reset signal;
- `data2send` (8 bit data): data to send over the UART interface;
- `send`: when this signal is raised high, the transmission begins.

Output Ports

- `tx`: the UART TX signal;
- `done`: when raised high, the data transmission process is terminated.

A simulation of the transmitter is shown in Figure 13:

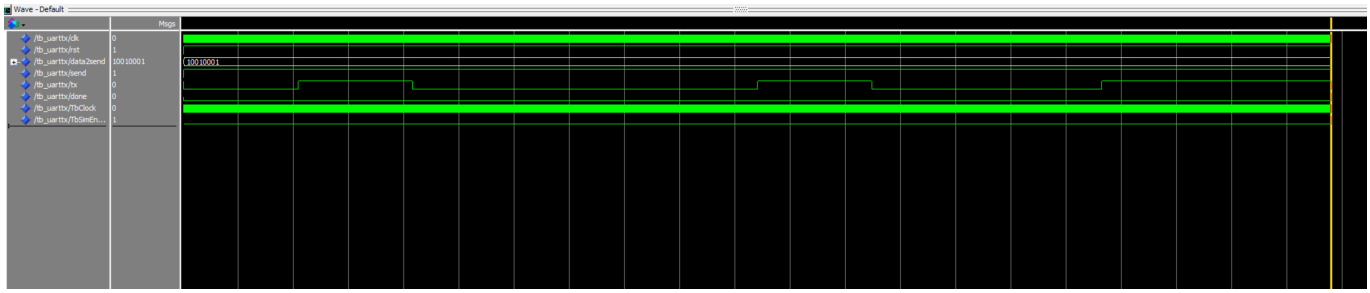


Figure 13: Simulation of the TX UART unit

The UART unit developed was published on gitlab at the following link:

`https://gitlab.com/vhdl-projects/uart`

UART RX Port Mapping
Input Ports

- clk: clock signal;
- rst: active low reset signal;
- rx: the UART RX signal;

Output Ports

- recData (8 bit): data received;
- newData: when a new data is received this signal is raised high.

3.3 I/O Wrapper

The I/O wrapper (`IOMG996R` entity) links the `MG996R` with the `UART` unit. The `RTL` is depicted in Figure 14.

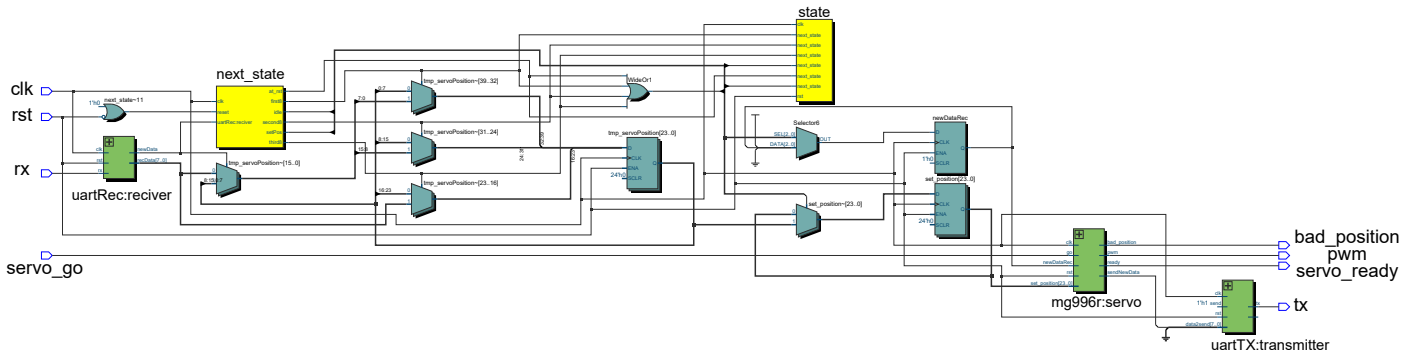


Figure 14: RTL view of top level entity

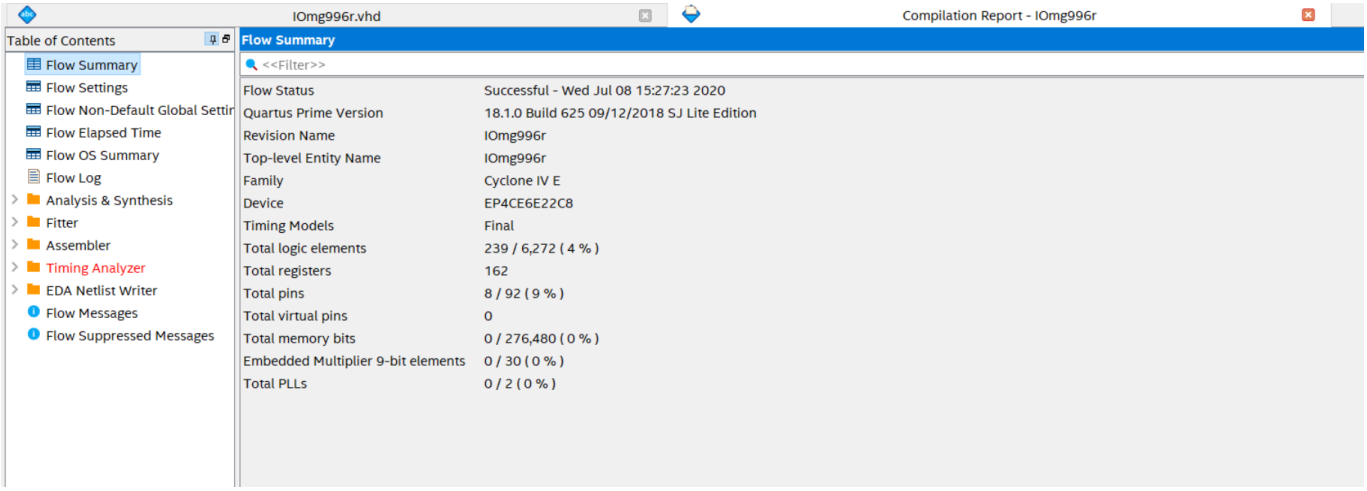


Figure 15: Flow Summary of the top level entity

The `ModelSim` simulation of the module is depicted in Figure 16.

3.4 Computer vision application

A computer vision application[7] that is able to detect a green rectangle was developed. This application:

- highlights the area where the rectangle is positioned;
- computes the coordinates of the center;
- computes the anomaly.

`OpenCV` and `gstreamer` are used to extract these features from each of the frames. The result obtained can thus be communicated through a serial port to an FPGA board.

The `UART` unit was successfully implemented and tested on simulations and hardware on a Nexys 4 DDR as well as on `A-C4E6E10` board. Live testing of the UART receiver is documented on the following YouTube Video:

`https://youtu.be/75O6jcYIYXo`

A demonstration of the video processing app is made available at the following YouTube link:
`https://youtu.be/FCx2O3vMtNI`

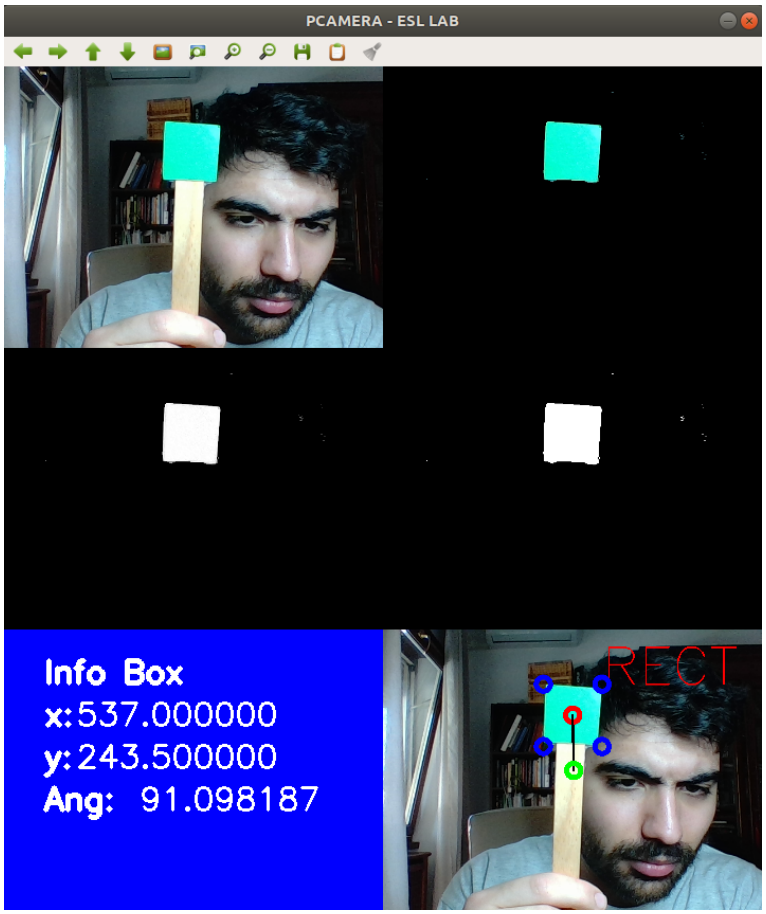


Figure 17: Frame of Demo of the computer vision application developed. See YouTube link `https://youtu.be/FCx2O3vMtNI`

[7] This application is meant to run under Ubuntu OS

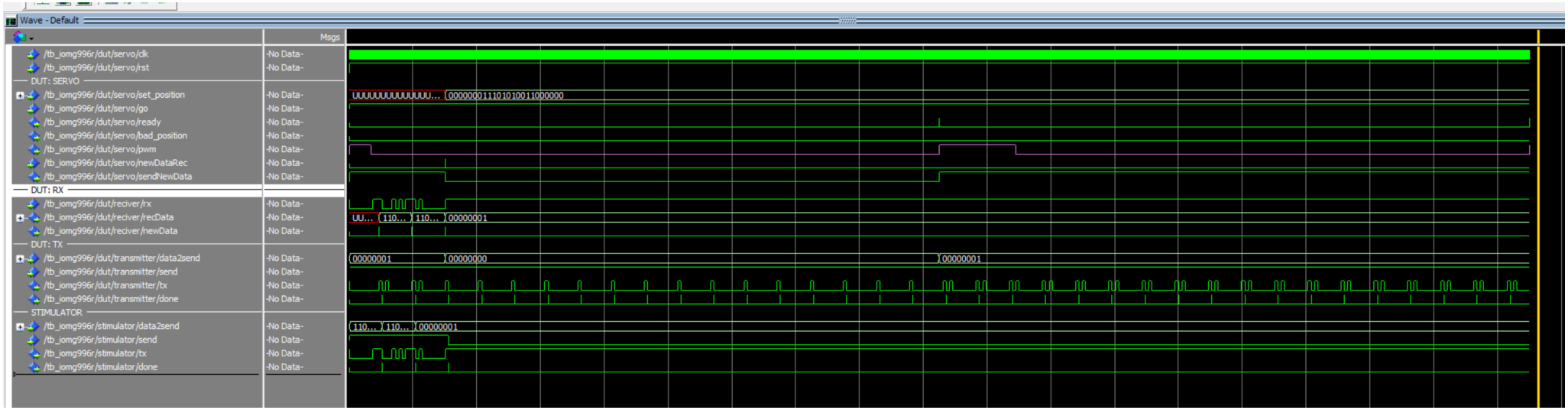


Figure 16: I/O Wrapper Modelsim Simulation

3.5 CAD Design of the Prototype

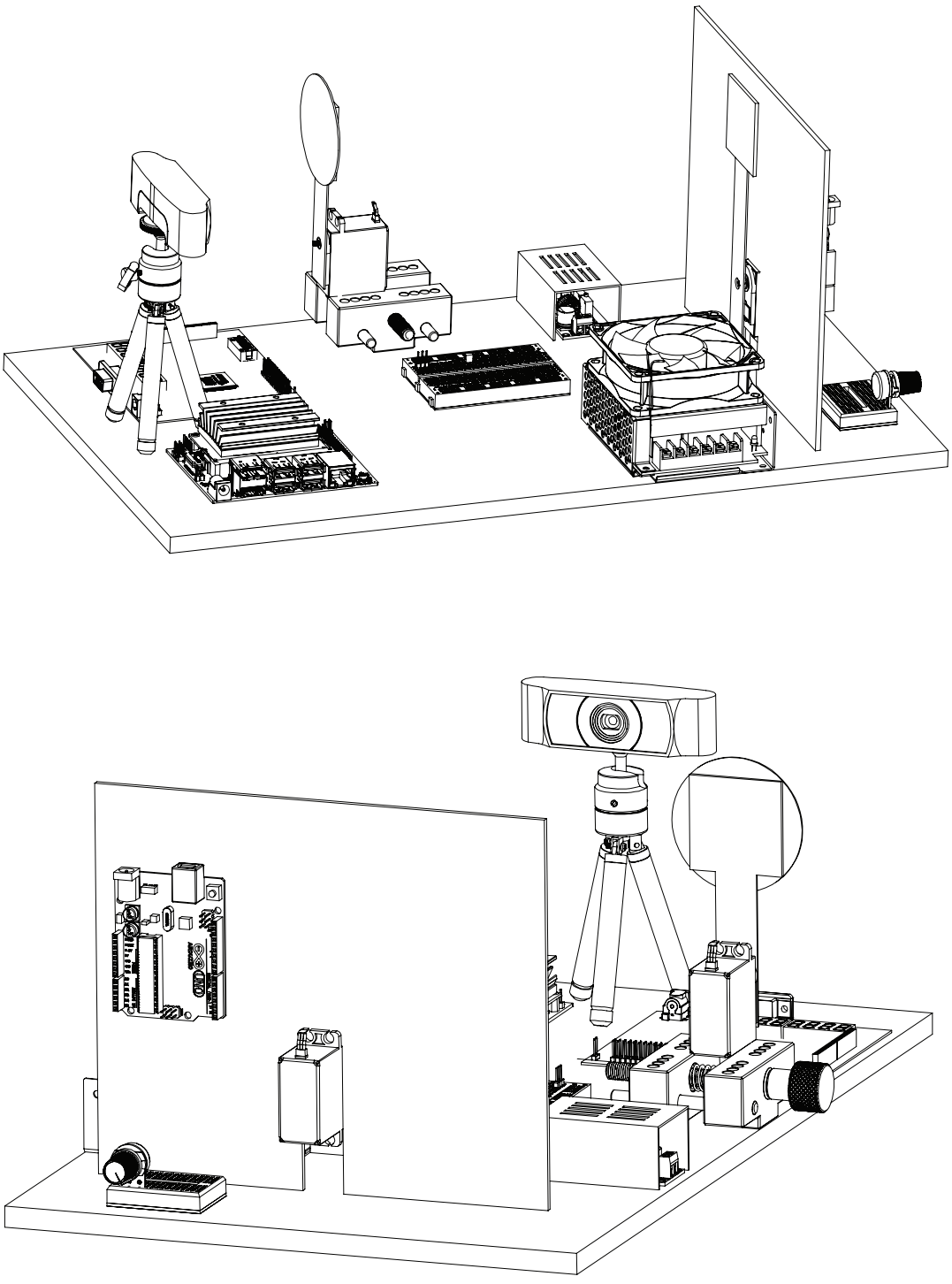


Figure 18: CAD Design Views (Part 1)

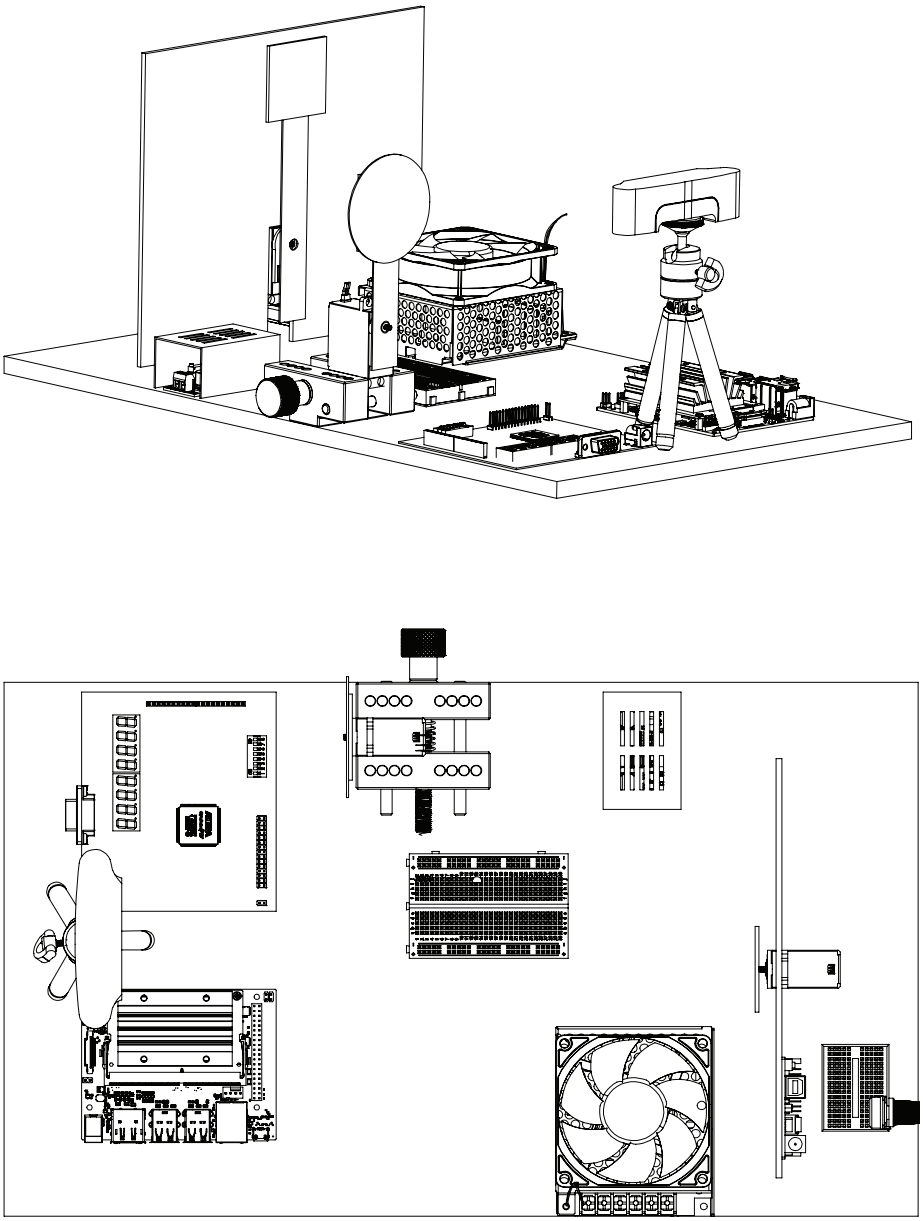


Figure 19: CAD Design Views (Part 2)

4 Conclusions

This project offered us the possibility to learn and practice many concepts related to Embedded Systems. In particular we have appreciated the formalization of design space exploration. In student project this task is often mentioned, but rarely formalized.
We considered significant to explore the interaction between an embedded processor and an FPGA. An effort was made to document the work and the results. For this purpose, we prepared different videos to share the outcome of the measurements and the experimental setup.

5 Appendix 1: Quadrature Decoder

File Reference

- `VHDL UNIT`: /ch3/quadratureDecoder/quadrature_decoder.vhd
- `TB VHDL UNIT`: /ch3/quadratureDecoder/tb_quadrature_decoder.vhd

Videos

- Quadrature signals measurements: `https://youtu.be/l5rgZYW6pMg`

The `quadrature_decoder` unit was written to decode a quadrature signal.
The logic of this unit is able:

- to detect the motor shaft sense of rotation;
- by means of a circular counter, to detect of how many steps the motor has rotated (backward or forward).

The design therefore has 3 outputs:

- `count_direction`: 1 for clockwise direction.
- `count_enable`: this signal is raised high when the counter is updated.
- `count_output`: output of the circular counter.

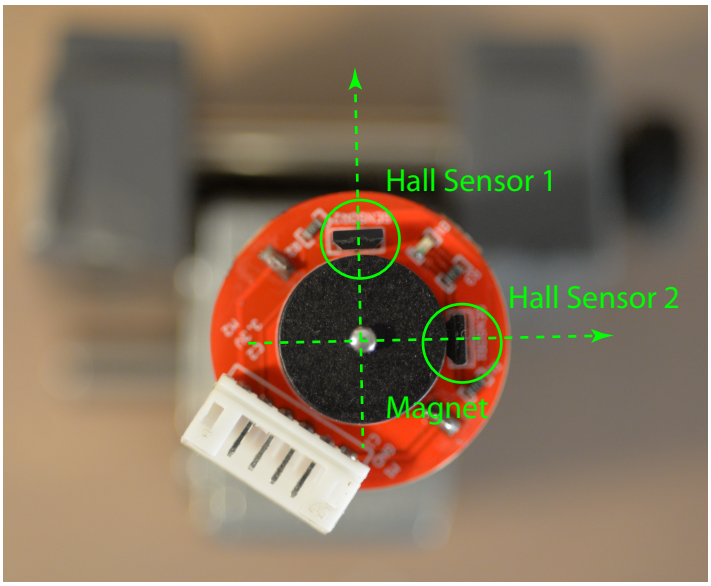


Figure 20: Hall sensors position

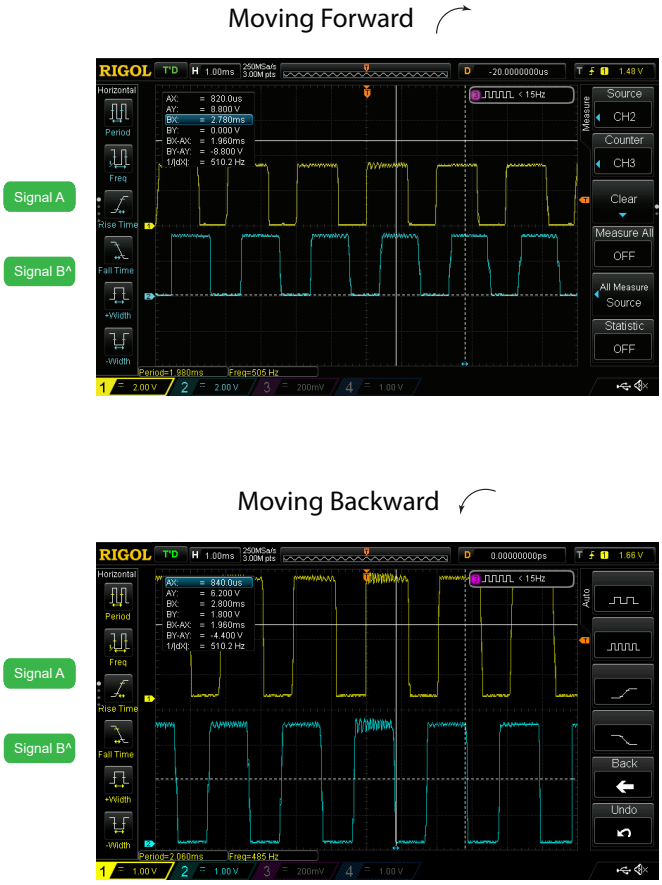


Figure 21: Hall sensors oscilloscope

When the motor shaft rotates CW (CCW) `count_direction` is increased (decreased). This behavior has been displayed in simulation as shown in Figure 22. For better understanding of the real signal from the Hall sensors, shown in Figure 20, an experimental setup has been arranged. The measurements of signals from the sensors are depicted in Figure 21. The live measurements are also displayed in the following YouTube video:
`https://youtu.be/l5rgZYW6pMg`.

6 Appendix 2: PWM Average Module

File Reference

- `VHDL UNIT`: ch4/pwm_average/pwm_average.vhd
- `TB 1 for VHDL UNIT`: ch4/pwm_average/tb_pwm_average.vhd
- `TB 2 for VHDL UNIT`: ch4/pwm_average/tb_pwm_average2.vhd

The `pwm_average` unit accepts as input a digital signal which, over the clock cycles, may vary its period and duty cycle. This unit is configured to average x number of periods, where x is a power of 2. As shown in Figure 23, this unit outputs the following signals:

- `average_high` : average number of clock cycles in which the signal stays high;

- `average_low` : average number of clock cycles in which the signal stays low;
- `average_period` = `average_high` + `average_low`
- $\texttt{average\_inverse\_duty\_cycle} = \frac{\texttt{average\_period}}{\texttt{average\_high}}$. This definition is adopted to avoid a floating point unit and use instead a simpler integer divider.

The simulation of the described unit is shown in Figure 24.

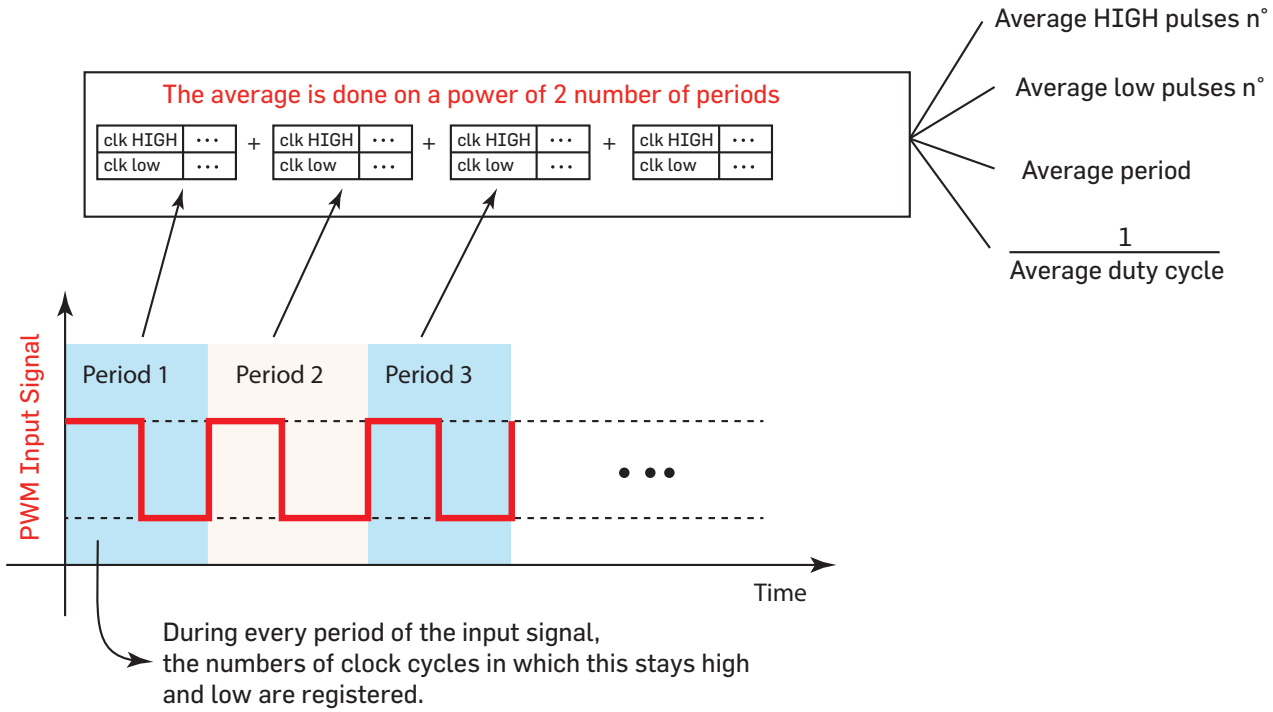


Figure 23: Visual schematics of `pwm_average` unit

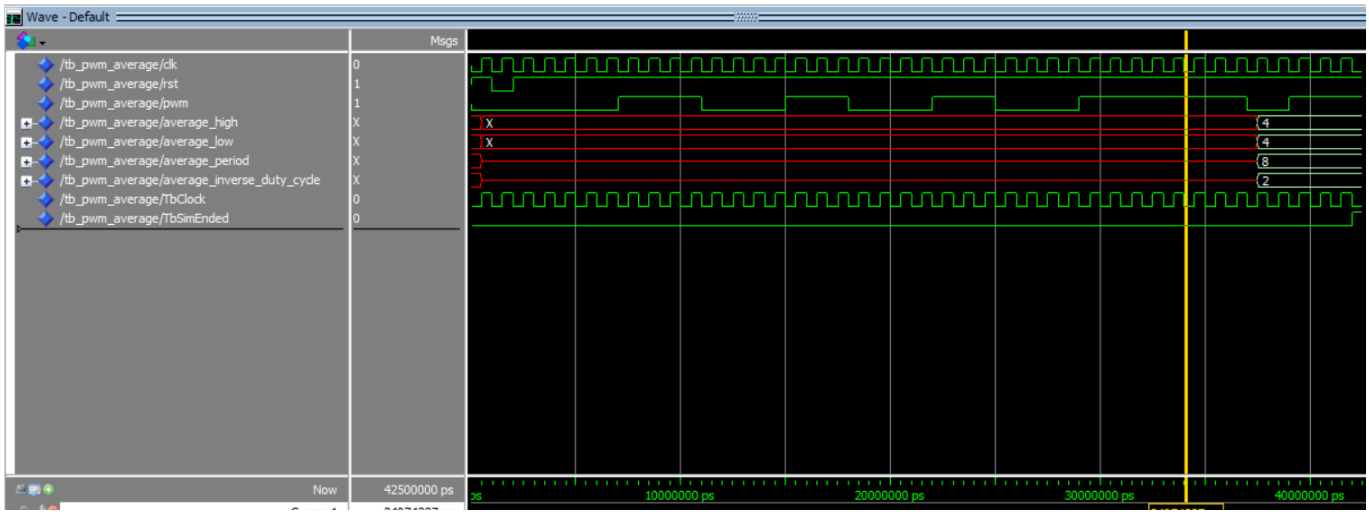


Figure 24: `pwm_average` unit sistem

Finally the developed unit was connected in a `VHDL` test bench file (`tb_pwm_average2.vhd`) to the `pwm` module. The simulation is reported in Figure 25

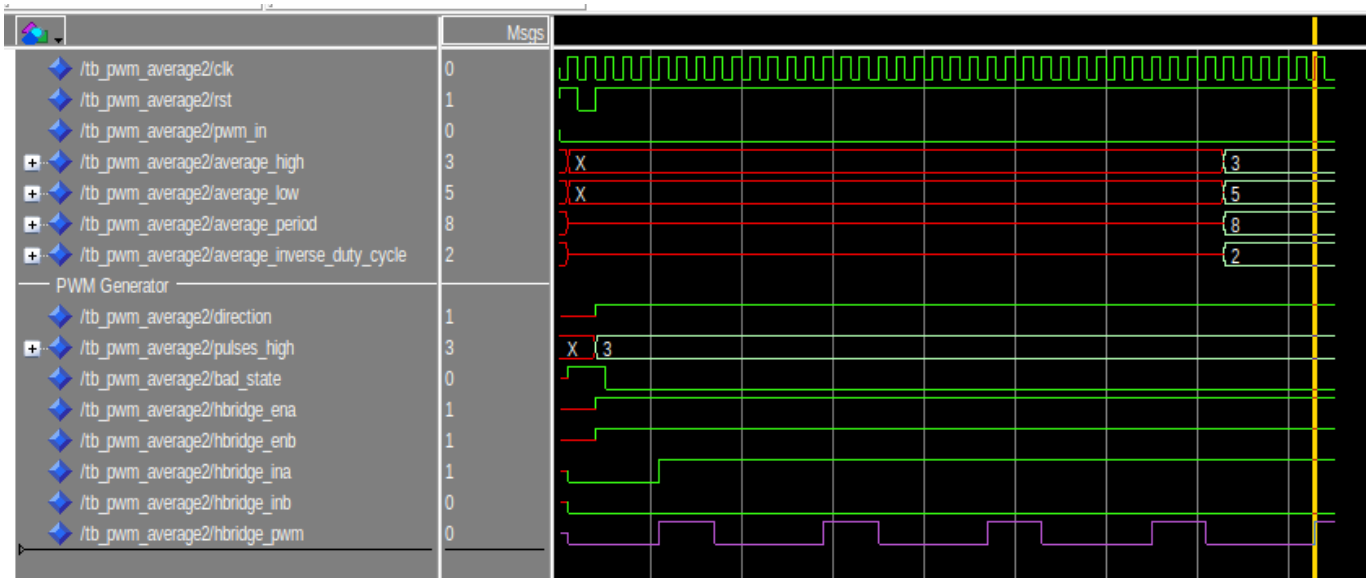


Figure 25: Pulse width average simulation

7 Appendix 3: Pulse Generator

File Reference

- `VHDL UNIT`: ch4/pulse_generator/pulse_generator.vhd
- `TB 1 for VHDL UNIT`: ch4/pulse_generator/tb_pulse_generator.vhd
- `TB 2 for VHDL UNIT`: ch4/pulse_generator/tb_pulse_generator2.vhd

The `pulse_generator` unit has been designed to accept as input the number of `PWM` pulses to be generated. Frequency, relative shift and input precision are set in the `generic`. The unit has a `go` signal which determines when the generation of the pulses can start. Once all the pulses are generated, the `all_pulses_generated` signal is asserted high. This signal stays in the high state until the `go` signal is asserted high once again. The simulation of the unit is depicted in Figure 26. The unit counts the number of pulses by counting how many times the pulse goes from logic level low to logic level high.

The `pulse_generator` unit is combined with the `quadrature_decoder`. As can be observed from the simulation of Figure 27, `count_output` value is 24. This result is explained as follow. In the `quadrature_decoder` the way pulses are counted is different: `count_output` is incremented of one unit whenever one of the input signals, namely `a` or `b`, changes its value. If we want to uniform the way pulses are counted in the two units, we simply need to divide by 4 the result of `count_output` from the `quadrature_decoder` unit. This operation is easily performed in binary by means of two bit shifts.

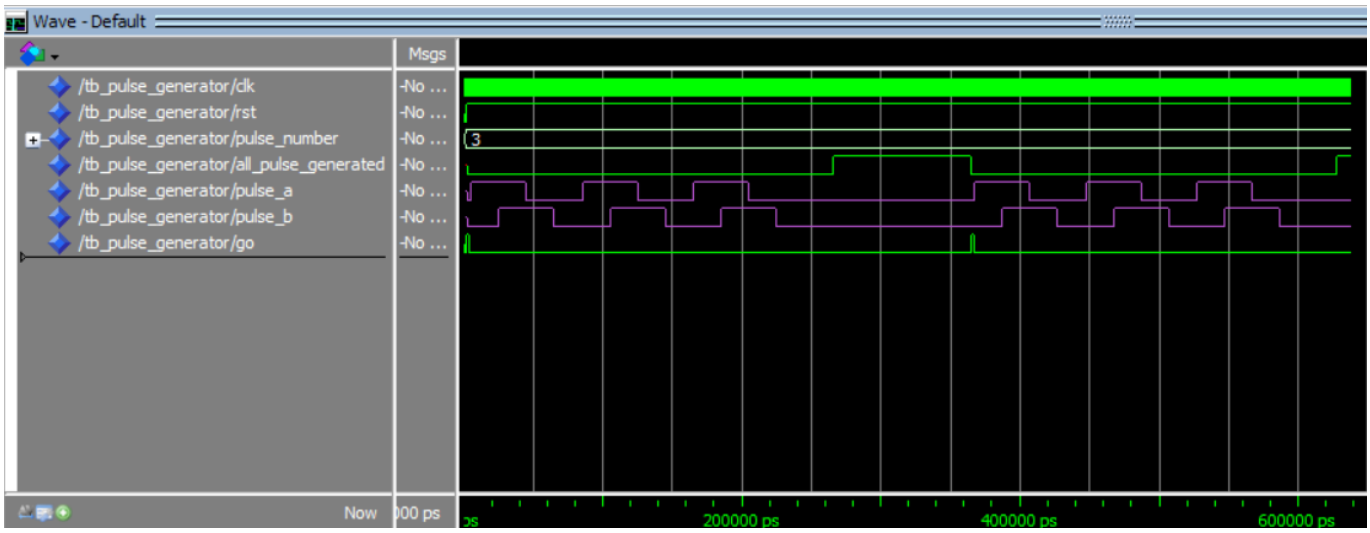


Figure 26: `pulse_generator` simulation result

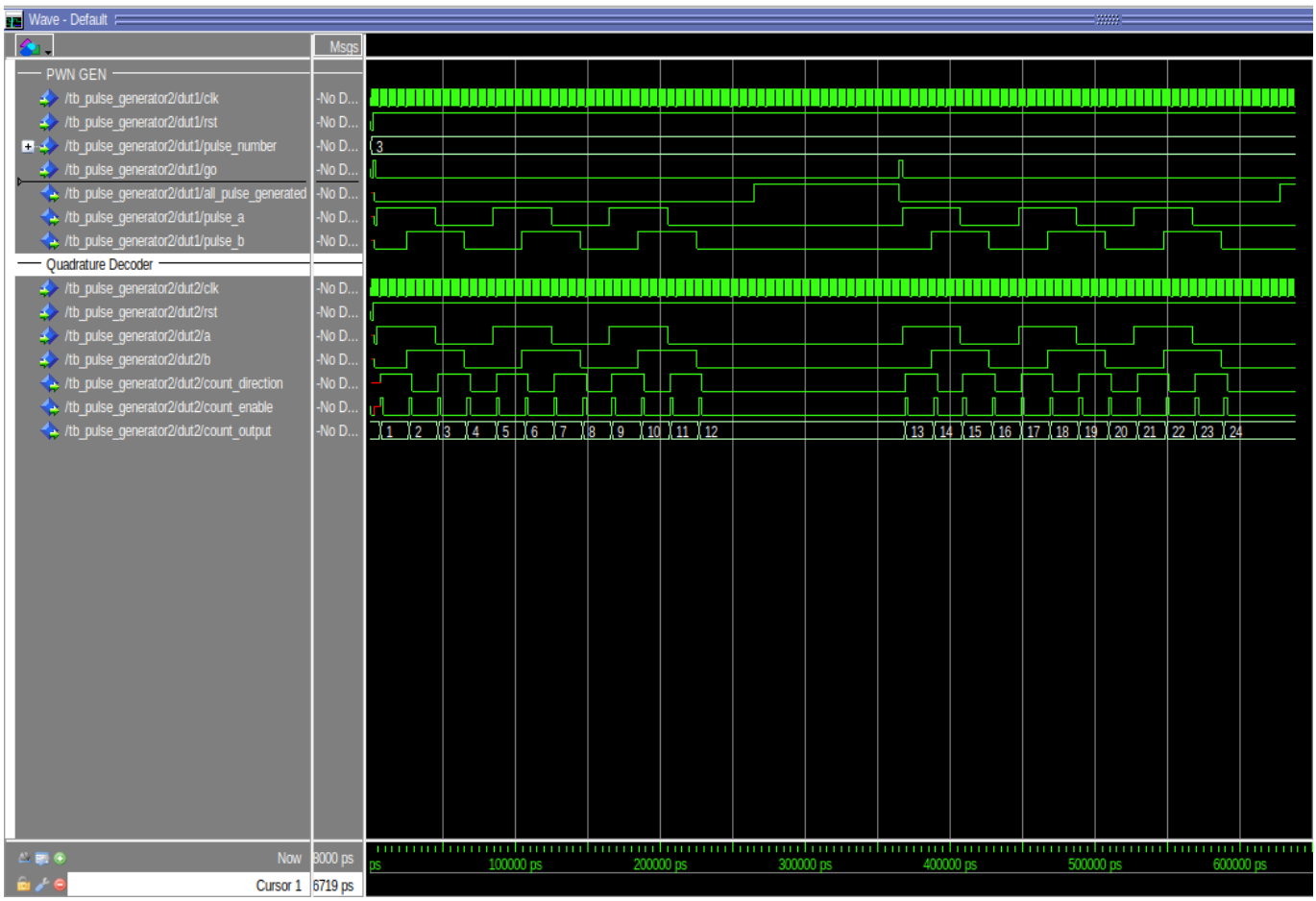


Figure 27: Combined simulation of `pulse_generator` and `quadrature_decoder`

8 Appendix 4: Plant Dynamics

In this Appendix the plant dynamics for PAN and TILT motions have been deduced. The PID parameters have been tuned to optimize response for

8.1 Observation on the 20Sim Model

We studied the 20sim model for the pan-tilt motion device (JIWY). Some inconsistencies in the numerical values of physical parameters have been detected [8]. First, from the bond graph shown in Figure 28, we tried to hint the scheme of the physical system (PAN motion only).

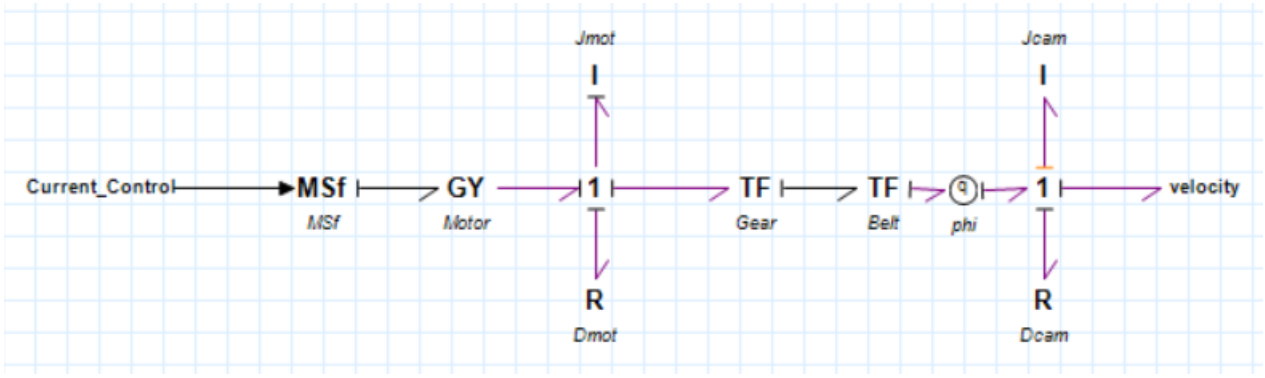


Figure 28: Bond Graph for PAN motion

- In the 20sim model provided, the transmission ratios parameters are handled by the original settings in both "Belt" and "Gear" nodes of bond graph:

  ```
  p1.e = p2.e * r;
  p2.f = p1.f * r;
  ```

  `p2.f` represents the output angular speed and `p1.f` the input angular speed and `r` is the transmission ratio. In the `20sim` model provided the parameters are:

  ```
  r=xxP[3]=4 (gear transmission ratio=omega_gear_2/omega_motor_shaft) and
  r=xxP[0]=5 (belt transmission ratio= omega_output_pulley/omega_input_pulley)
  ```

  Note that `omega_gear_2=omega_input_pulley`.

  The overall transmission ratio of the device is `omega_output_pulley/omega_motor_shaft=4*5=20` This means that, with the original program settings, the angular velocity of the output pulley is 20 times the one of the motor shaft.

- To increase the torque at the end effector (output pulley)) it is necessary to reduce the end effector output speed. This is the usual purpose of gears and belts. Therefore, in our simulation, the overall transmission ratio= `omega_output_pulley/omega_motor` should be less than 1. For this reason, we believe that the correct parameters values are `xxP[3]=1/4=0.25` (negative sign to account for verse of rotation) and `xxP[0]=1/5=0.20`.

For this reason we decided to deduce the equations of motion for PAN/TILT and validate them using `20sim` (open loop). The comparison between the results from `20Sim` parameters adjusted model and those from the deduced equation is reported in Figure 29.

[8] See discussion on CANVAS.

8.2 Description of the overall system

The general scheme for a pan and tilt motion is shown in Figure 30.

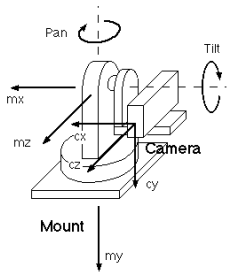


Figure 30: Representation of PAN and TILT motion

With reference to such scheme, the PAN motion refers to a rotation with respect to a vertical rotation axis and the TILT motion with respect to an horizontal rotation axis.

The overall system can be split into two subsystems: one for TILT motion and another for the PAN motion. They have the same topology, but different

In each subsystem the following main components are observed:

- the electric motor:
- a gear transmission;
- a belt transmission.

8.3 Plant dynamics equations for the PAN and TILT subsystems

In this section, the differential equation governing the plant dynamics of a subsystem is deduced. Although the analytical derivation requires an effort, the advantage is that the final outcome is a single equation instead of a linear system of equations, as generated by the 20sim software.

The subsystem modeling follows from the 20sim signal flow graph included in the files available on Canvas, as well as the numerical values (See Figure 31).

Figure 31: Original dynamic parameters

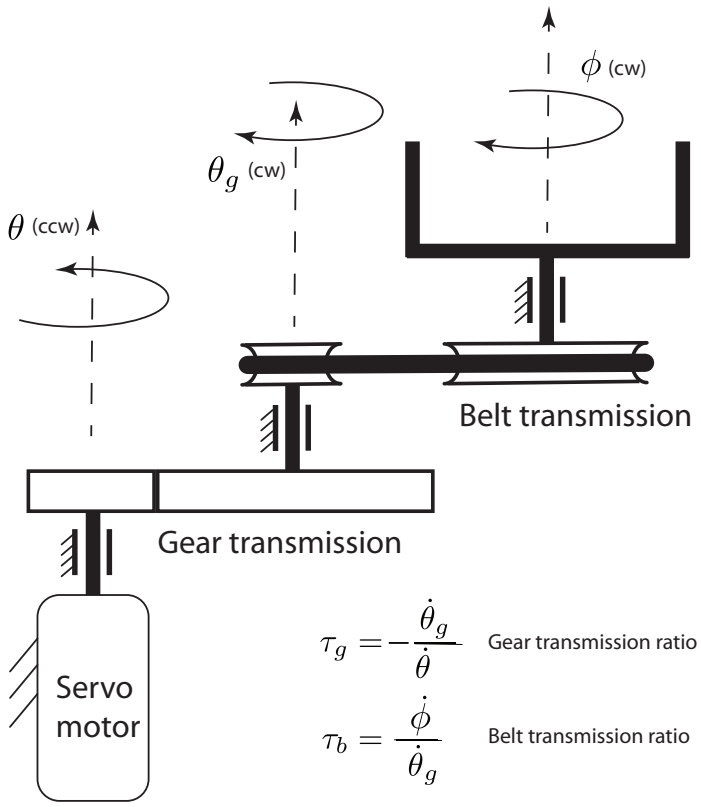


Figure 32: Physical scheme

The physical scheme is shown in Figure 32. The external generalized forces are:

- $-J_{cam}\ddot{\phi}$ inertia torque acting on the output pulley;
- $-c_{cam}\dot{\phi}$ viscous torque acting on the output pulley;
- $-J_{mot}\ddot{\theta}$ inertia torque acting on the motor shaft;
- $-c_{mot}\dot{\theta}$ viscous torque and back EMF acting on the motor shaft;
- T_a external torque on the motor shaft.

The application of principle of Virtual work gives

$$-J_{cam}\ddot{\phi}\delta\phi - c_{cam}\dot{\phi}\delta\phi - J_{mot}\ddot{\theta}\delta\theta - c_{mot}\dot{\theta}\delta\theta + T_a\delta\theta = 0 \tag{1}$$

The transmission ratios are defined as follows:

- transmission ratio of the gears:

$$\tau_g = \frac{\omega_{GEAR}}{\dot{\theta}} = 0.25 \tag{2}$$

- transmission ratio of the pulleys:

$$\tau_b = \frac{\dot{\phi}}{\omega_{PULLEY}} = 0.2 \tag{3}$$

Since

$$\omega_{PULLEY} = \omega_{GEAR} \tag{4}$$

the overall transmission ratio is

$$\tau = \tau_g \tau_b = \frac{\omega_{GEAR}}{\dot{\theta}} \frac{\dot{\phi}}{\omega_{PULLEY}} = 0.2 \cdot 0.25 = 0.05 \tag{5}$$

The following equalities give the kinematic relations between the output pulley rotation ϕ and motor shaft rotation θ:

$$\tau = \frac{\delta\phi}{\delta\theta} = \frac{\dot{\phi}}{\dot{\theta}} = \frac{\ddot{\phi}}{\ddot{\theta}} \tag{6}$$

Substituting (6) in (1) follows the differential equation of motion of the output belt:

$$\boxed{\ddot{\phi}\left(J_{cam} + \frac{J_{mot}}{\tau^2}\right) + \dot{\phi}\left(c_{cam} + \frac{c_{mot}}{\tau^2}\right) = \frac{k_T I}{\tau}} \tag{7}$$

- PAN motion (Physical parameters)

$$J_{cam} = 0.00108 \tag{8}$$
$$J_{mot} = 2.63 \cdot 10^{-6} \tag{9}$$
$$c_{cam} = 0.00985 \tag{10}$$
$$c_{mot} = 1.7 \cdot 10^{-4} \tag{11}$$
$$k_T = 0.0394 \tag{12}$$

- TILT motion (Physical parameters)

$$J_{cam} = 1. \cdot 10^{-4} \tag{13}$$
$$J_{mot} = 2.63 \cdot 10^{-6} \tag{14}$$
$$c_{cam} = 1.35 \cdot 10^{-5} \tag{15}$$
$$c_{mot} = 1.77 \cdot 10^{64} \tag{16}$$
$$k_T = 0.0394 \tag{17}$$

With the previous numerical parameters, the equation (7) can be rewritten in the following forms for the plant dynamics of the PAN and TILT motions, respectively:

- PAN motion

$$\boxed{\ddot{\phi} = -\left(369.6060 \cdot I + 36.5150 \cdot \dot{\phi}\right)} \tag{18}$$

- TILT motion

$$\boxed{\ddot{\phi} = -\left(684.0278 \cdot I + 67.5781 \cdot \dot{\phi}\right)} \tag{19}$$

As confirmed by numerical simulations both in `20sim` and Matlab, it can be observed that at steady state the magnitude of the output pulley angular speed (expressed in rad/s) is about one order of magnitude higher than the input current I (expressed in A) (*i.e.* $\dot{\phi} \approx 10 \cdot I$).

The previous differential equations can be solved numerically by means of the Euler's method.

Let ϕ_k, $\dot{\phi}_k$, I_k be the angular position, velocity and current, respectively, at time $t_k = k\Delta t$ $(k = 0, 1, 2, \dots,)$, where Δt is the integration step.

For the PAN motion, the estimated values at time $t_{k+1} = (k+1)\cdot\Delta t$, can be recursively computed as follows:

$$\ddot{\phi}_k = -\left(369.6060 \cdot I_k + 36.5150 \cdot \dot{\phi}_k\right) \tag{20a}$$

$$\dot{\phi}_{k+1} = \ddot{\phi}_k \Delta t + \dot{\phi}_k \tag{20b}$$

$$\phi_{k+1} = \phi_k + \dot{\phi}_k \cdot \Delta t \tag{20c}$$

A similar approach can be followed for the TILT motion.

Both plant dynamics equations as well as the numerical inetgration procedures have been coded in VHDL.

8.4 Tuning of PID control parameters

Because of the parameters problem previously discussed, we did not relay upon the `20sim` original parameters, but we decided to perform the PID controller tuning in Matlab. Using the previous equations as plant dynamics for the PAN and TILT motion, respectively, the tuned parameters for a PID controller have been computed by means of the `pidTuner` of Matlab (see plots of Figures 33 and 34).

Listing 1: PAN motion: tuning of PID controller parameters

```
% TUNING OF PID CONTROLLER FOR PAN MOTION
close all
clc
J_cam = 0.00108;
J_mot = 2.63e-6;
c_cam = 0.00985;
c_mot = 1.7e-4
k_t = 0.0394;
tau_belt = 0.2;
tau_gear = 0.25;
tau = tau_belt *tau_gear;
numerator = k_t/tau;
denominator = [(J_cam + J_mot/tau■2 ),(c_cam + c_mot/tau■2),0];
% Define transfer function for pan
sys = tf(numerator,denominator)
%tuning PID parameters
pidTuner(sys,'PID')
```

Listing 2: TILT motion: tuning of PID controller parameters

```
% TUNING OF PID CONTROLLER FOR TILT MOTION
close all
clc
J_cam = 1.e-4;
J_mot = 2.63e-6;
c_cam = 1.35e-5;
c_mot = 1.77e-6
k_t = 0.0394;
tau_belt = 0.2;
tau_gear = 0.25;
tau = tau_belt *tau_gear;
```

```
numerator = k_t/tau;
denominator = [(J_cam + J_mot/tau^2),(c_cam + c_mot/tau^2),0];
% Define transfer function for tilt
sys = tf(numerator,denominator)
%tuning PID parameters
sys_tuned=pidTuner(sys,'PID')
```

The tuned parameters are as follows:

- PAN MOTION

$$k_p = 7.609$$
$$k_i = 70.9964$$
$$k_d = 0.1812$$

- TILT MOTION

$$k_p = 0.0012095$$
$$k_i = 0.00019357$$
$$k_d = 0.0016793$$

By means of `20Sim`, from the original JIWY model, the `C` code describing the PAN and TILT dynamic was generated. Files are contained in the directory `/ch5/20sim`.

8.5 Programming Plant Dynamics in VHDL

File Reference

- /ch5/plant_dynamics/plant_dynamics.vhd
- /ch5/plant_dynamics/position.vhd
- /ch5/plant_dynamics/speed.vhd
- /ch5/plant_dynamics/tb_files/tb_plant_dynamics.vhd
- /ch5/plant_dynamics/tb_files/tb_position.vhd
- /ch5/plant_dynamics/tb_files/tb_speed.vhd

Two different `VHDL` module for estimating angular shaft position and velocity have been developed (`speed.vhd, position.vhd`). Both these units make use of a 32 Bit floating point library[9] Finally these two modules have been integrated into a single module (`plant_dynamics.vhd`) to simulate the plant dynamics (hardware in the loop). The result of the `VHDL` test bench is shown in Figure 35 and is consistent with the `Matlab` Simulation.

Nomenclature

Dots denote differentiation with respect to time

[9] https://github.com/xesscorp/Floating_Point_Library-JHU/blob/master/FloatPt.vhd

$\delta(\cdot)$ Virtual generalized displacement of $(\cdot)$

Δt Integration step (s)

$\dot{\theta}$ angular speed of motor shaft (rad/s)

ω_{GEAR} angular speed of output gear (rad/s)

ω_{PULLEY} angular speed of first pulley (rad/s)

ϕ angular rotation of output pulley (rad)

ϕ_k Angular position at time t_k (rad)

τ_g Gear transmission ratio

τ_m Belt transmission ratio

θ angular rotation of motor shaft (rad)

c_{cam} Damping (Nms/rad)

c_{mot} Back EMF voltage + Damping (Nms/rad)

I Current (A)

I_k Current at time t_k (A)

J_{cam} Moment of inertia of the masses attached to the output pulley (kg m^2)

J_{mot} Moment of inertia of the masses attached to motor shaft (kg m^2)

k_T Servo motor constant (Nm/A)

$T_a = k_T I$ Servo motor torque (Nm)

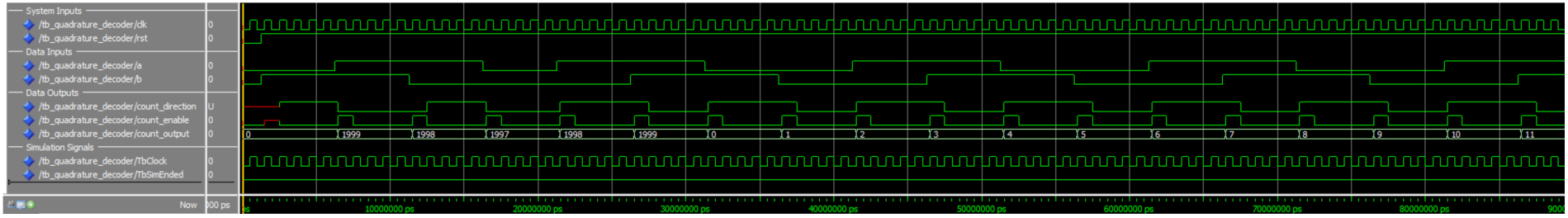
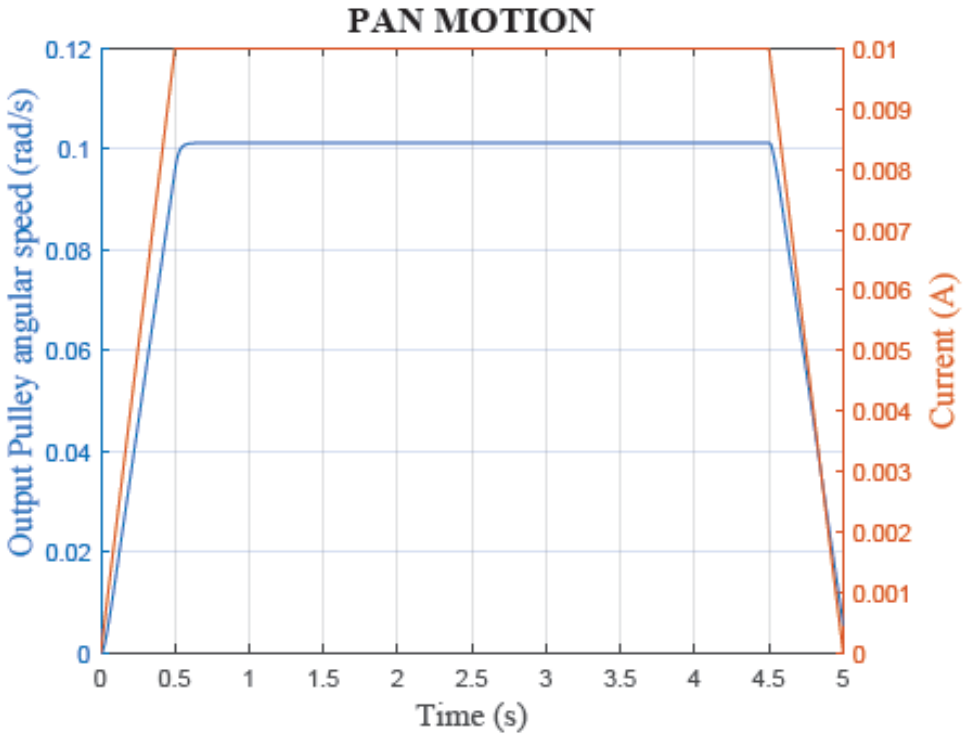


Figure 22: Simulation of Quadrature Decoder

Simulation with my parameters (Matlab)



Simulation with my parameters (20Sim)

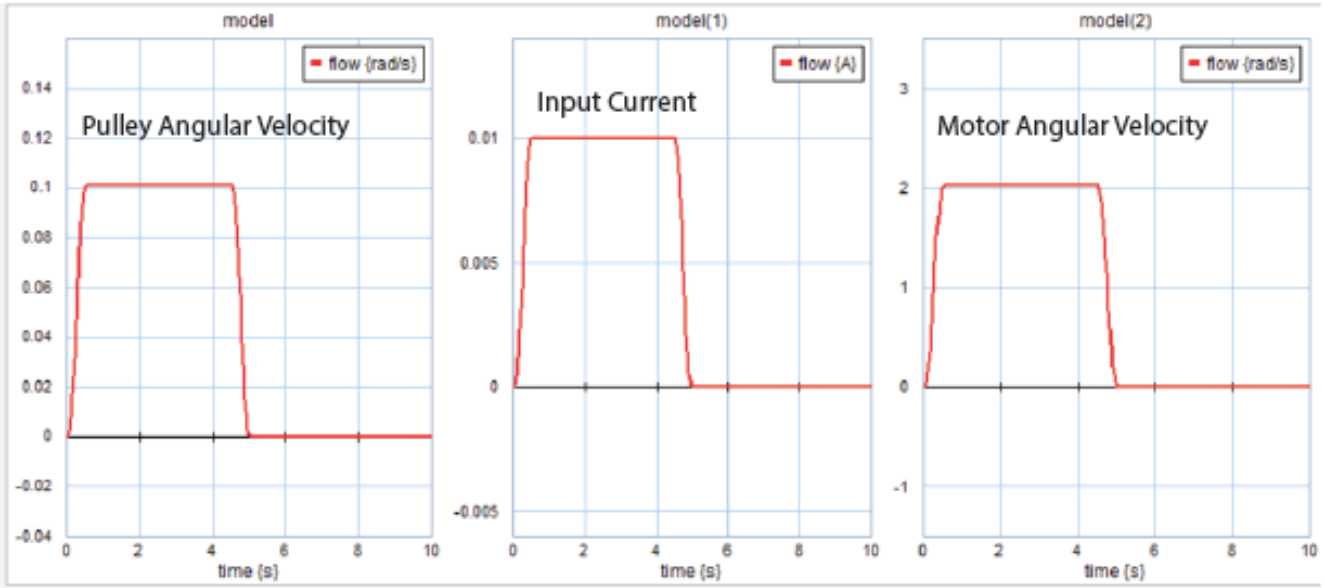


Figure 29: Comparison between `20sim` model and deduced equation (PAN Motion)

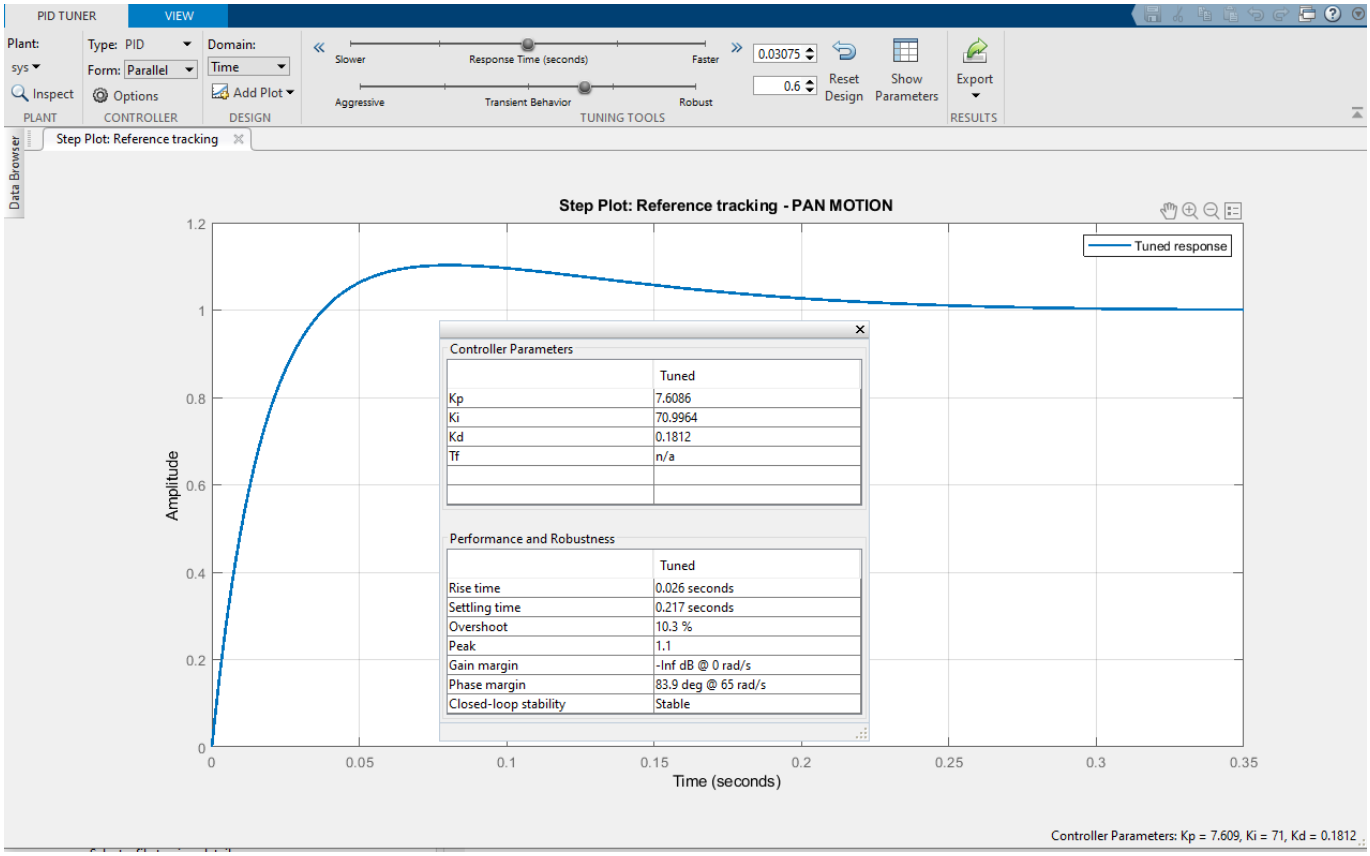


Figure 33: PID tuning for the PAN motion

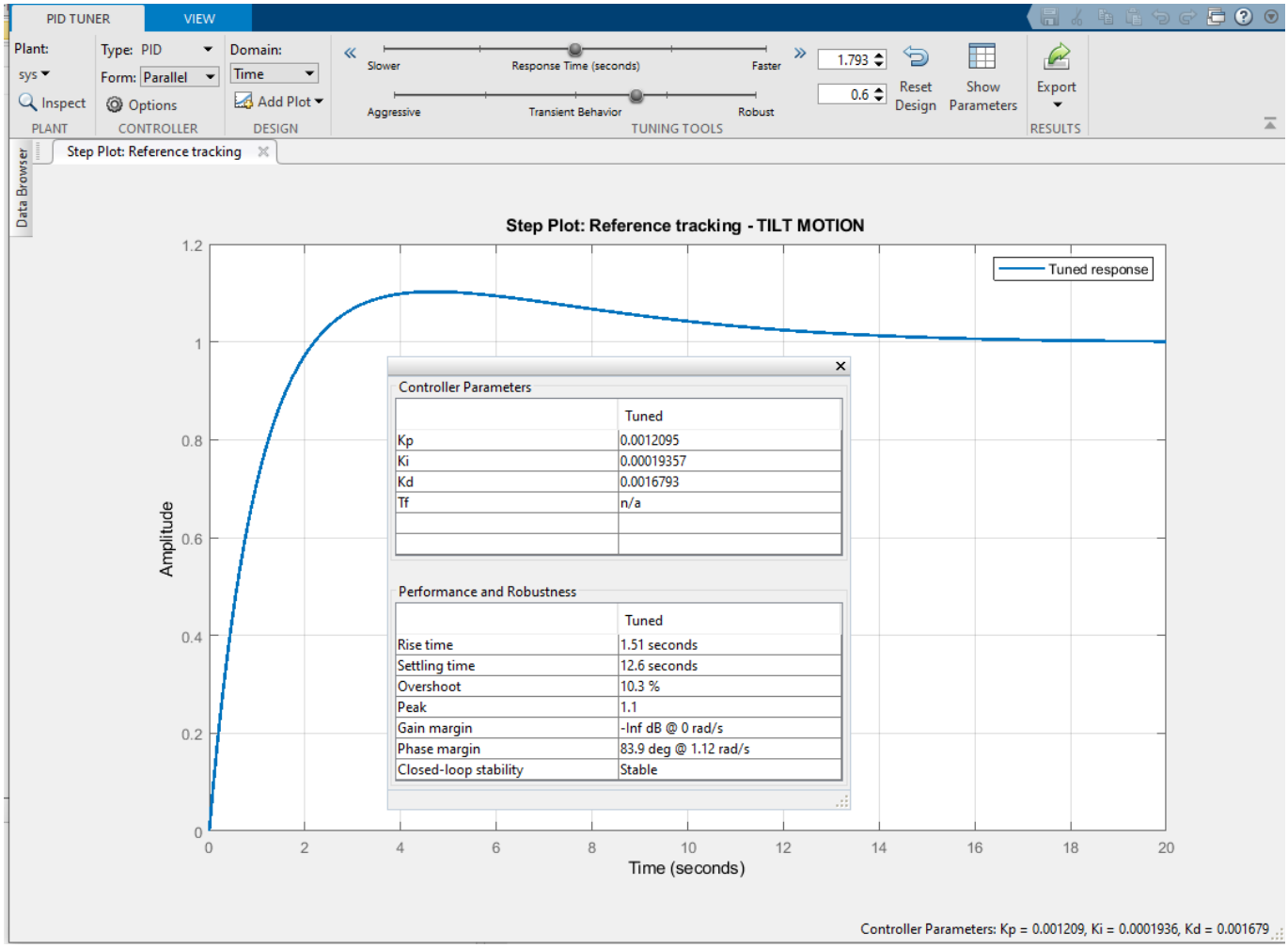


Figure 34: PID tuning for the TILT motion

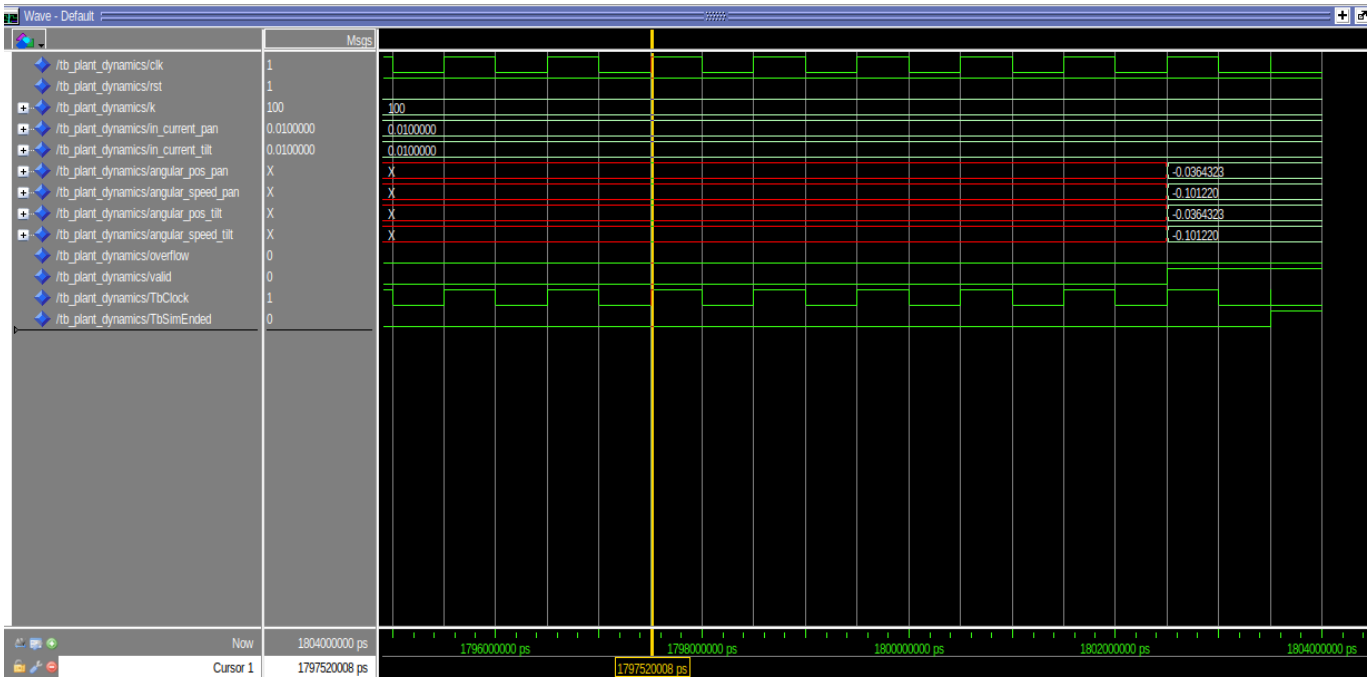


Figure 35: Plant dynamics Simulation in ModelSim