RC-Car (KART)

Lecture Summerschool 1 (SS1)

Orientation: Systems Engineering (Synd)
Specialisation: Infotronics (IT)
Course: Summerschool 1 (SS1)
Authors: Silvan Zahno, Axel Amand, François Corthay, Charles Praplan

Date: 01.07.2024
Version: v1.0

Contents

1 Security Guide

In the pursuit of creating a small remote-controlled car, it is imperative to consider the security-aspect of the project. Ensuring the integrity of the hardware, preventing damage to components and personal safety are vital. This chapter outlines the security measures and protocols that must be adhered to throughout the course of the project. These measures have been put in place to protect both the project's hardware and the participants involved.

Some security rules are also mentioned again in the followoing chapters, where they are relevant.

Think before doing
Certainly, thinking before taking action is a fundamental principle of effective project management and security.

No Hardware to Leave the Premises
To safeguard the project's hardware components, it is strictly forbidden for any team member to take project-related hardware home.

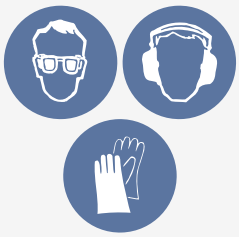
All hardware must be properly stored in the designated laboratory cabinets.

Behaviour in the Labors
To safeguard the project's hardware components, it is strictly forbidden to have any food and drinks close to the project-related hardware.

In addition protection for other equipments such as tables, instruments must be used.

The Labors needs to be kept clean and tidy.

Mechanical Precautions
The motors used in the remote-controlled car can pose a physical risk, and special attention should be given to prevent accidents and injuries. Team members must exercise caution when handling the motors, ensuring that they are properly secured, and that all moving parts are well-guarded to prevent accidental contact.



Personal Protective Equipment (PPE)
When working with mechanical tools and machines, it's imperative to prioritize the safety of all team members. PPE, such as safety goggles, gloves, ear protection, and dust masks, should be worn as appropriate when operating machinery. Always remember, safety first.

Power Disconnection Protocol
Completely disconnect any power source before making changes or modifications on the hardware.

Battery Precautions
Before connecting batteries to the remote-controlled car, comprehensive functional testing is mandatory. The following precautions must be taken:

- Prior to connecting the batteries, all functionalities of the car must be tested using a laboratory power supply limited to $0.05A$, excluding the motors. When attaching the motors and other custom equipment, the power supply limit must be increased to $1.2A$. This precaution prevents sudden surges of current from damaging the circuitry during the initial testing phase.
- Make sure to shut the circuit off before charging to avoid higher voltages on the 12V rail. The charge rate should be around 0.4C => 1500mA here. Use only dedicated Ni-Mh chargers.

Secure SODIMM Connector
The FPGA board is a critical component of the remote-controlled car project. To ensure its proper functioning, the SODIMM connector must be inspected at each test to verify that the FPGA board is securely connected. Loose connections can result in system malfunctions, data corruption, and potential damage to the FPGA.

Help is available
In case of any questions or concerns regarding the security of the project, the team members are encouraged to contact assistants and professors.

1.1 Consequences

It is crucial to emphasize that any deviation from the security measures outlined in this chapter will result in consequences. It will involve the **deduction of points from the final project grade**. This penalty serves as a reminder of the importance of adhering to the security protocols and maintaining the integrity of the project.

In conclusion, security is a paramount consideration in the development of the remote-controlled car project. The security measures presented in this chapter are designed to protect the hardware and people, prevent damage, and ensure the project's success. By following these protocols and cooperating with professors, the team can create a safe and secure environment for project development while minimizing the risk of damage.

2 Introduction

The Kart module (SS1) is a Summer School module for students between the 2nd and the 3rd semester. It is a home-made model car remotely controlled by a smartphone. The Appendix C - Inspiration gives an overview of previous year's karts.

The work of the students can be summarized in four main tasks:
- design and assembly of the chassis and the body
- analysis of the Direct Current (DC) motor [1]
- configuration of the controlling Field Programmable Gate Array (FPGA) [2]
- completion and extension of the control Graphical User Interface (GUI) on the smartphone

This document only covers the configuration and programming of the control electronics.

2.1 Objective

For the control electronics part there are three mandatory objectives:
- Control block for the DC Motor, see Section 5.1 - PWM Modulator
- Control block for the stepper motor, see Section 5.2 - 4 coils sequence generator
- Hall-Sensor counter, see Section 5.3

2.2 Evaluation

Fullfilling all mandatory objectives mentioned in Section 2.1 will result in a grade of 4.0. The students are free to implement additional features. For every added feature, the grade will be increased. Depending on the complexity of the feature between 0.5 and 1.0. until the maximum grade of 6.0 is reached.

Optional features can be:
- Emergency Stop using an ultrasonic distance sensor
- Animate Kart parts with servomotors and others I/Os
- Other improvements outside the scope of the predefined objectives

2.3 Files

All necessary files can be collected via two methods:
1. Either downloaded via a zip file at the following link https://github.com/hei-synd-ss1/ss1-vhdl/archive/refs/heads/main.zip [3] directly from github and extract it into your preferred location.
2. Alternatively a group specific repository can be created via github classroom by using the following invitation link: https://classroom.github.com/a/pgjx0vpr. You repository can then be cloned with **`git`**.

```
git clone https://github.com/hei-synd-ss1-stud/2024-ss1-vhdl-<groupname>.git
```

Make sure there is no space characters in the full projects path. HDL may hang while booting or files may not be loaded/saved correctly.

2.4 Tools

The design environment for the control electronic consists of several tools.

- Mentor HDL designer for graphical design entry [4]

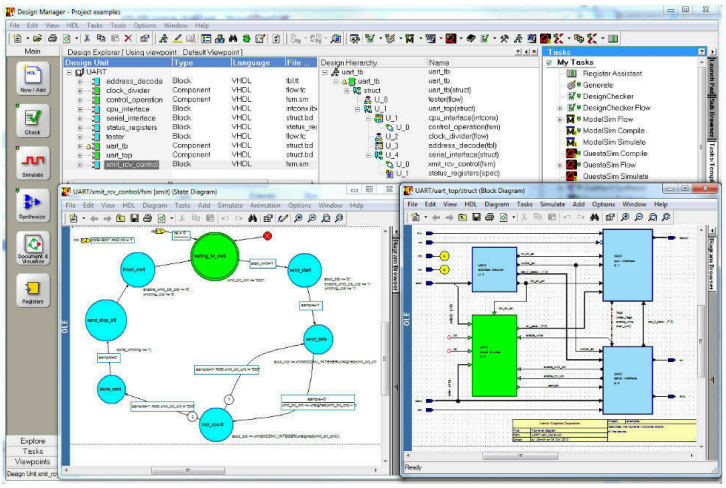


Figure 1: Mentor HDL Designer

- Mentor ModelSim for simulation [5]

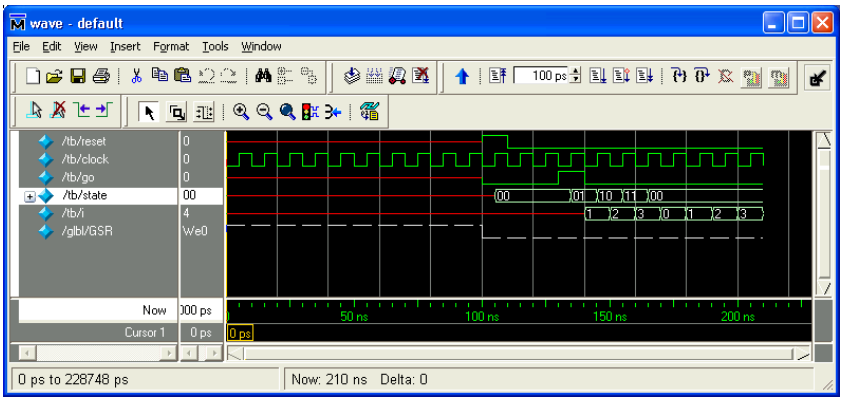


Figure 2: Mentor Modelsim

- Microchip Libero IDE for synthesis and programming[6]

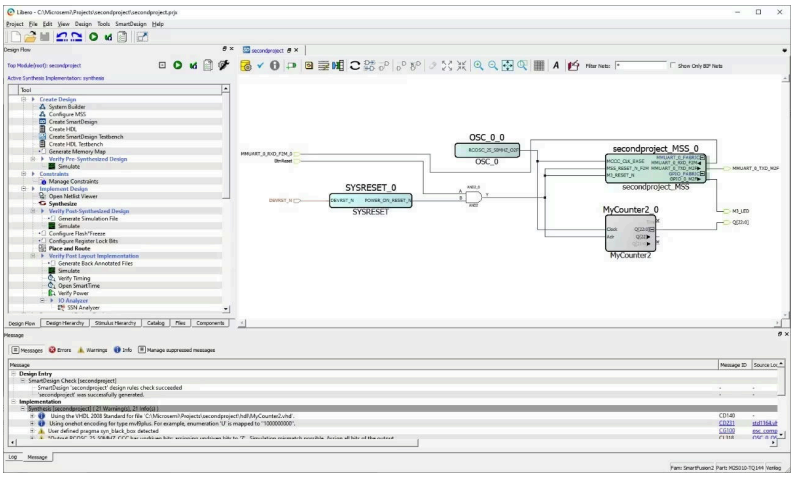


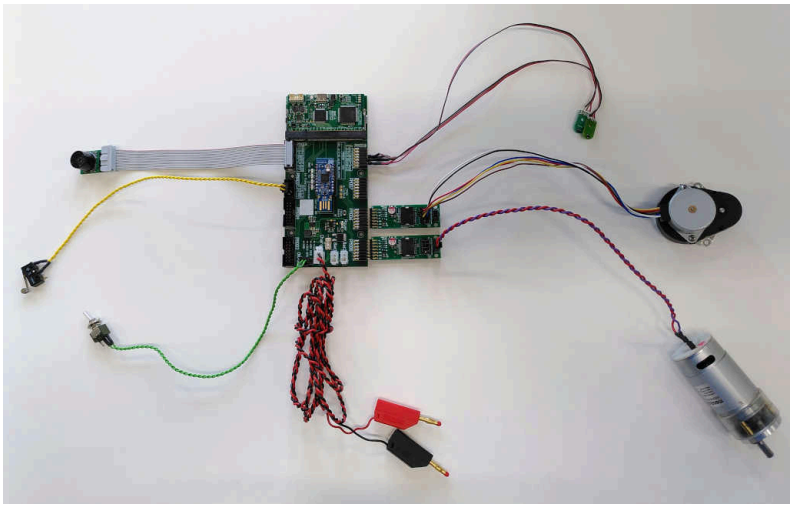
Figure 3: Microchip Libero

2.5 Cabling

Since every RC Kart is unique, you will probably use different input and output signals. This needs to be reflected in the Pin Constraint file which can be found under **`Board/concat/Kart.pdc`**.

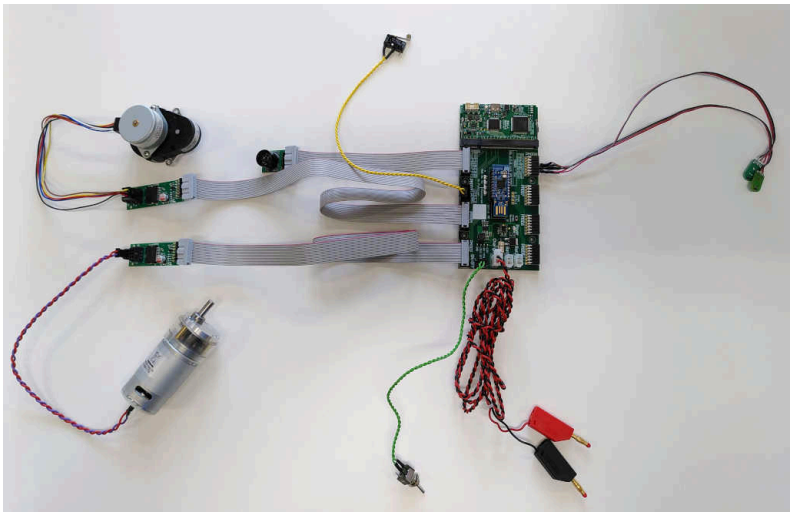
The master solution already programmed on the board for tests uses the direct connection schematic.

Hereafter two examples:



Module	Pmod Nb
Range Sensor	PM1 (8)
Stepper End	PM2 (1)
Leds out.	PM4
DC Motor	PM5 (5, 6)
Stepper Motor	PM6 (5 to 8)
Buttons in.	PM7 (5 to 8)
Hall Sensor	PM8 (4)

Figure 4: Hardware Cabling (direct connection, **`Kart.pdc`**)



Module	Pmod Nb
Range Sensor	PM1 (8)
Stepper End	PM2 (1)
DC Motor	PM3 (5, 6)
Stepper Motor	PM4 (5 to 8)
Leds out.	PM6
Buttons in.	PM7 (5 to 8)
Hall Sensor	PM8 (4)

Figure 5: Hardware Cabling (flat cables, **`Kart_FlatCables.pdc`**)

3 System Architecture

The architecture is essentially split between two parts: the embedded electronic which drives the kart and read the various sensors, communicating with a smartphone to be controlled remotely. In this section only the mandatory functions are described.

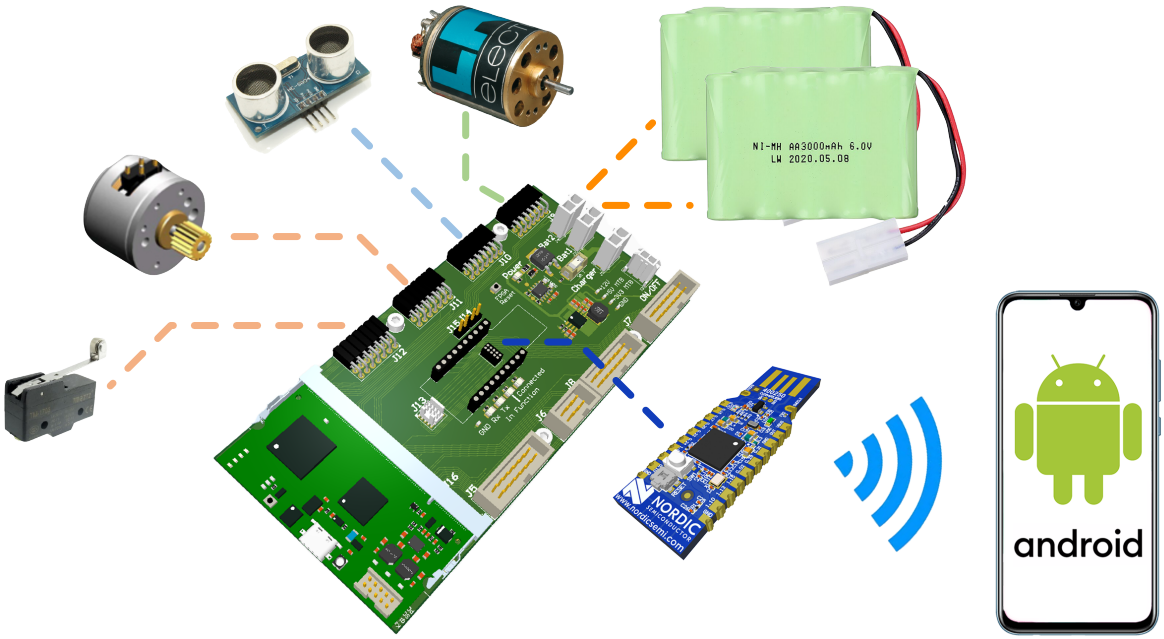


Figure 6: System Architecture Overview

(The shown config is custom and does not correspond to Figure 4 or Figure 5)

Contents

3.1 Block diagram

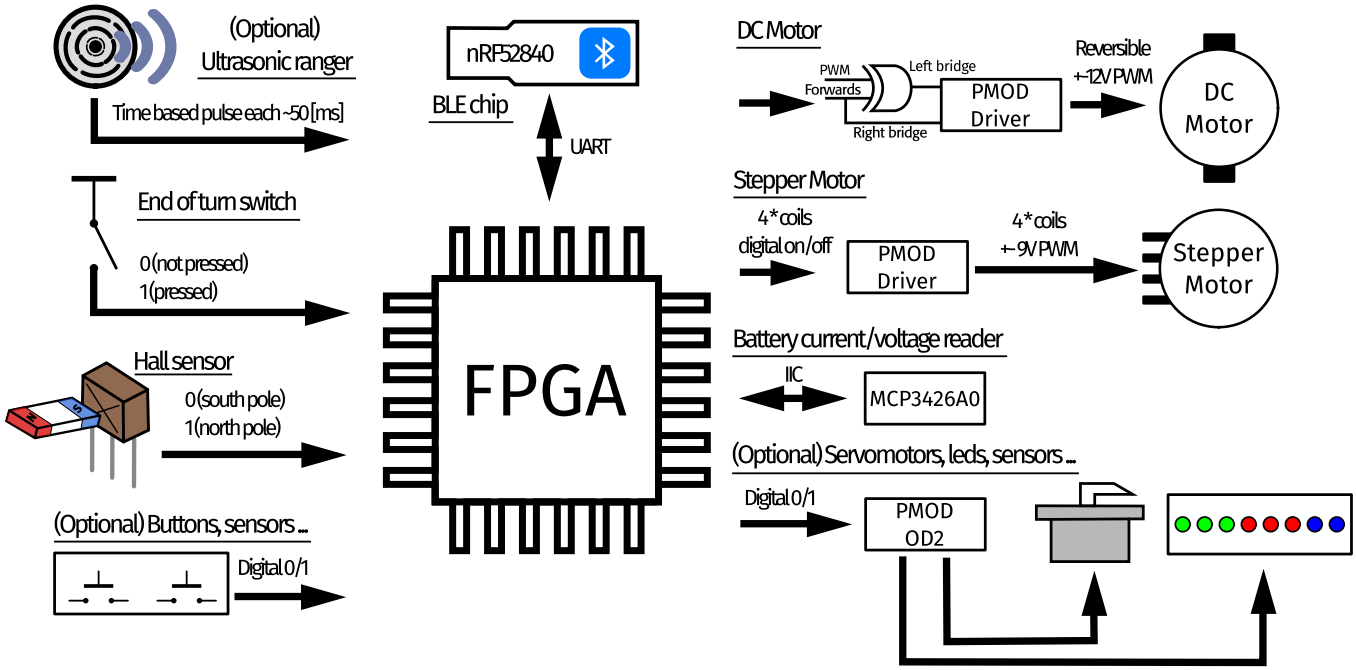


Figure 7: Electronic Architecture

The system is centered around the FPGA which gets the following inputs:

- **Ultrasonic ranger (optional)**: permits to detect distances and brake in case of an obstacle
- **End of turn switch**: used to zero the wheels angle of the Kart
- **Hall sensor**: combined to magnets, allow to calculate the car speed
- **Buttons, sensors ... (optional)**: any extra hardware which creates 0-3.3V digital inputs can be wired to the FPGA and used internally for extra features

The followings are available on the board itself:

- **User leds (optional)**: 3 user leds can be freely controlled for debug purposes
- **BLE chip**: the FPGA communicates through UART with a nRF52849 BLE chip to link with the smartphone
- **Battery reader**: a MCP3426A0 chip allow to read through I2C both the current and voltage from the system

Finally, outputs are:

- **DC motor**: the system is propulsed by a 12V brushed DC motor
- **Stepper motor**: the steering of the car is carried out by a 4 coils stepper motor
- **Servomotors, leds, sensors ... (optional)**: any digital output can be wired to the FPGA through the PMOD-OD2 board. This board can translate outputs into higher voltages and currents. The selectable output voltages of this board are +3.3, +5 or +12 V

3.2 Functions

The minimal system allows to communicate with the smartphone as well as operate all required sensors and actuators. The system needs to:

- Propel the kart forward and backward with the help of the DC motor - Section 4.4.1 - and motor driver PMOD - Section 4.4.3
- Steer the kart with the help of the stepper motor - Section 4.4.2, a motor driver PMOD board - Section 4.4.3 - and the end of turn switch - Section 4.5
- Count the hall sensor - Section 4.6 - pulses to measure the speed
- Set registers correctly to communicate through the UART serial link with a custom communication protocol - Section 7 - with the smartphone over bluetooth - Section 4.7.

4 Hardware Components

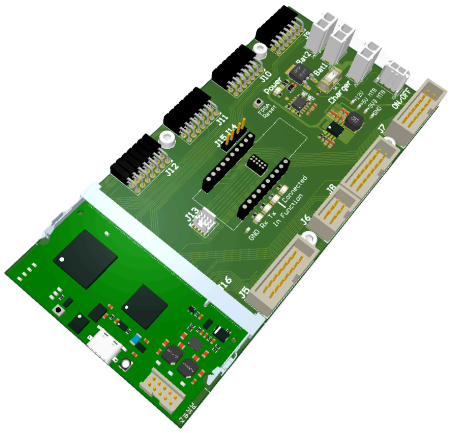


Figure 8: Kart PCB

Contents

4.1 Good Practices

In order not to damage the hardware, strictly follow the Section 1 - Security Guidelines.

4.2 Motherboard (MB)

The Kart motherboard can receive any compatible FPGA daughterboard - Section 4.3 such as the AGLN250 [7] one used during the Kart project. The Motherboard connects all pheripherals to the I/Os and powers the system at the same time [2].

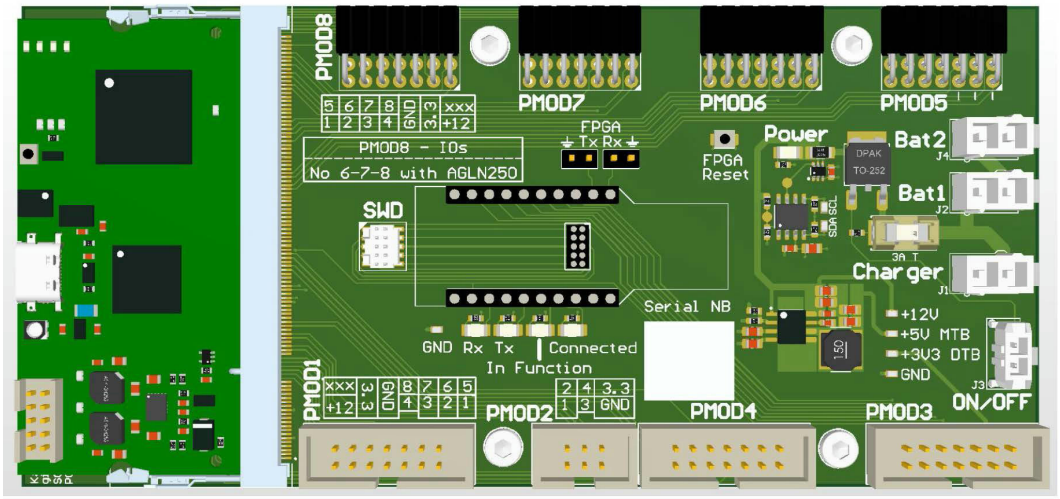


Figure 9: Motherboard PCB

4.2.1 Power

The main power entry point of the system is through the motherboard, either by using:
1. the two battery connectors with two +6V / 2400mAh packs put in series
2. the charger port with a +12V input from a regulated DC supply.

The +12V is then reduced to a +5V rail through a buck converter. Finally, the daughterboard is fed with the +5V to provide a +3.3V rail.

4.2.1.1 Charging

The Charger connector is used combined with a DC supply to test the system, but also for an NiMh charger to be connected.

Charging batteries is handled by the electronic lab directly. Simply ask and hand them your packs.

4.2.1.2 Power-on

A switch must be connected to the corresponding port to power the board. The +12V is then transported to the PMODs and the buck converter through a 1.25A-T fuse. A green LED shows the board power status.

Follow the tests guideline given under Section 6 before powering anything.
If the fuse breaks, check for any short-circuit before replacing it and power-cycling the circuit.

4.2.1.3 Power State

An I2C dual-inputs ADC converter (MCP3426A0) [8] is present on the board to read both the battery voltage and the current consumption.

Access the data

The chip is read from the FPGA each second through dedicated I2C lines. The information can be read from the smartphone at will by accessing the corresponding registers - Section 7.2.

4.2.2 SODIMM Daughterboard Connector

The daughterboard - Section 4.3 is connected to the motherboard through a SODIMM-200 (DDR2 RAM) connector.

4.2.3 I/Os

The board allows for multiple I/Os to be plugged following the PMOD wiring [9], slightly modified to add a +12V rail, under the following form:

- 4 dual connectors for direct plug (PMODs 5 to 8)
- 3 dual connectors for flat-cables (PMODs 1, 3, 4)
- 1 single connector for flat-cable (PMOD 2)

PMOD8 signals P6 to P8 cannot be used with the AGLN250 FPGA. Use only the upper row (PM8_1 to PM8_4).

The pins are described on the board itself and correspond to the following:

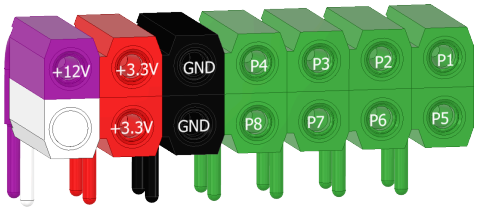


Figure 10: PMOD pinning (header)

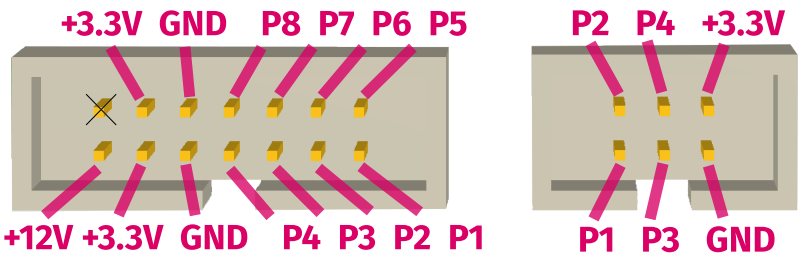


Figure 11: PMOD pinning (flat)

At all time, ensure there is **NO VOLTAGE FEEDBACK** from anything else than 3.3V on the I/Os. Use a dedicated PMOD board if needed. They are presented in the Appendix B.

4.2.4 FPGA Reset

A small button allows the user to reset the FPGA from the motherboard.

4.2.5 UART Sniffer

Both Tx from the FPGA and the BLE module can be sniffed by wiring a dedicated UART-USB chip to the provided headers or directly with an oscilloscope.

4.2.6 BLE Socket

The Bluetooth ⟺ USB dongle - Section 4.7 [10] can be inserted in its dedicated socket to control the Kart with a smartphone, or easily removed to be plugged in a PC directly. One can emulate the BLE module with the help of a custom serial interpreter - Section 6.5 by simply plugging the daughterboard in a PC through the USB-C. The communication is merged between both the PC and the BLE module.

Trying to communicate simultaneously from the BLE module and the PC will result in undefined behavior (surely scrambled and wrong data read by the FPGA).

One can listen to what the FPGA communicates to the BLE module by opening a serial terminal on the USB-C COM port, but not the other way around.

4.3 Dautherboard (DB)

The FPGA daughterboard embeds an Igloo **AGLN250** chip [7] in a **VQ100** package, driven by a **10MHz** clock.

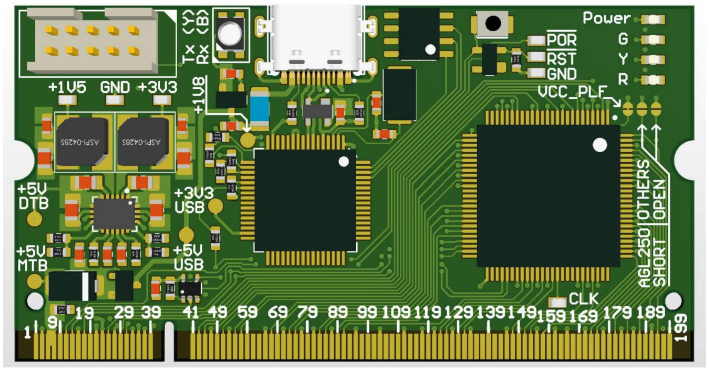


Figure 12: Daughterboard PCB

4.3.1 Power

The board is powered either through an USB-C connector (+5V, providing a JTAG access along an USB-UART converter [11]) or through the motherboard - Section 4.2 via an external +5V rail.
It will automatically resolve the path if both supplies are wired simultaneously, choosing the motherboard rail in priority.
Internally, it creates a +3.3V rail used by both the FPGA and the motherboard, plus another +1.5V rail for the FPGA core supply.

4.3.2 Programming

The board can be programmed by using Libero IDE - Appendix III and plugging a Microsemi FlashPro 4 dongle on the dedicated 10 pins header.
It is also possible to use the USB-C connector with the help of OpenOCD and custom scripts.
Both the USB and the FlashPro can be plugged at the same time. The FlashPro gains priority over the USB JTAG signals.

While using the FlashPro without the motherboard powered, it is necessary to plug both the USB-C and the FlashPro to be able to program the card.

4.3.3 Connection with Motherboard

The board is linked to the motherboard - Section 4.2 through an SODIMM-200 connector (the 200 gold fingers on the FPGA board) [2]. By design, it cannot be inserted the wrong way.
The connector pining is shown in the Kart Pinning Datasheet [12].

4.3.4 I/O

Reset

A button is found on the board to reset the FPGA.

Be careful when pushing it while it is inserted in a motherboard. Do not apply force on the SODIMM connector.

LEDs

A blue LED indicates that the board is powered (top right of the board), while a second found near the USB connector shows in and out transaction over UART.

The red led indicates if the stepper end switch is pressed.
The yellow led toggles on and off when a magnet is rotated in front of the hall sensor.
The green led is on when the smartphone is connected to the BLE module.

PoR

The board also embeds a Power on Reset (PoR) circuitry that will detect low FPGA voltage and reset it (discharged batteries, too high current consumption ...).

4.4 Motors

4.4.1 DC-Motor

The DC Motor is used to propel the kart forward. It is a brushed DC motor, running on +12V and drawing a current of $I_{\text{max}} = 0.7A$ and $I_{\text{idle}} = 0.32A$ [1]. The PMOD DC-Stepper Motor Driver allows to control the motor via a PWM Signal [13] through a H-Bridge [14].

Color	Pin
Red	+
Violet	-

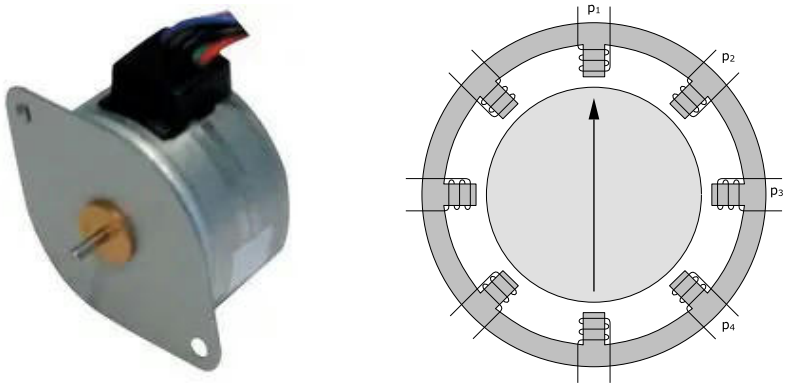
Figure 13: Modelcraft RB350018-2A723R DC Motor and its pining

4.4.2 Stepper-Motor

The Stepper Motor is used for steering the kart. It is a bipolar stepper motor with a step angle of 7.5° - 48 steps per rotation and a nominal current of $I = 0.86A@5V$ with a $R = 5.8\Omega$ [15].
It is attached to a 100:1 reductor gear which leads to an output axis with a step angle 0.075° - 4800 steps per rotation.

The motor is controlled with the PMOD DC-Stepper Motor Driver - Section 4.4.3 hosting a dual full H-Bridge to control the 4 coils of the stepper motor.

The calibration can be performed using the End-of-Turn switch - Section 4.5.



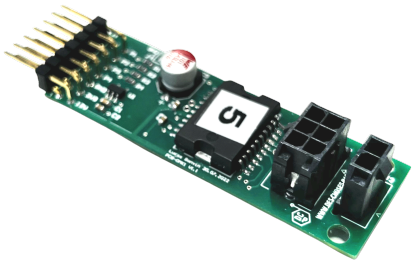
Color	Pin
White	Coil 1
Blue	Coil 2
Black	GND
Brown	GND
Red	Coil 3
Yellow	Coil 4

Figure 14: Nanotec SP3575M0906-A Steppermotor, coils and pining

4.4.3 PMOD DC-Stepper Control board

The control board hosts a dual full H-Bridge and a circuitry to create the switching dead-times [16].
It allows to control both the DC motor or the stepper motor.
The 6-pins connector is used to connect a stepper motor and the 2-pins one the DC Motor.

> ⚠ Only one motor can be connected at any given time.



Row	Pin	Descr.	Row	Pin	Descr.
	2	P1 NC		1	P5 In Left Bridge A
	4	P2 NC		3	P6 In Right Bridge A
	6	P3 NC		5	P7 In Left Bridge B
Top	8	P4 NC	Bottom	7	P8 In Right Bridge B
	10	GND		9	GND
	12	3.3V		11	3.3V
	14	12V		13	NC

Figure 15: PMOD DC-Stepper Control board and pining

The Figure 16 shows the bloc diagram of the main component, the L298P:

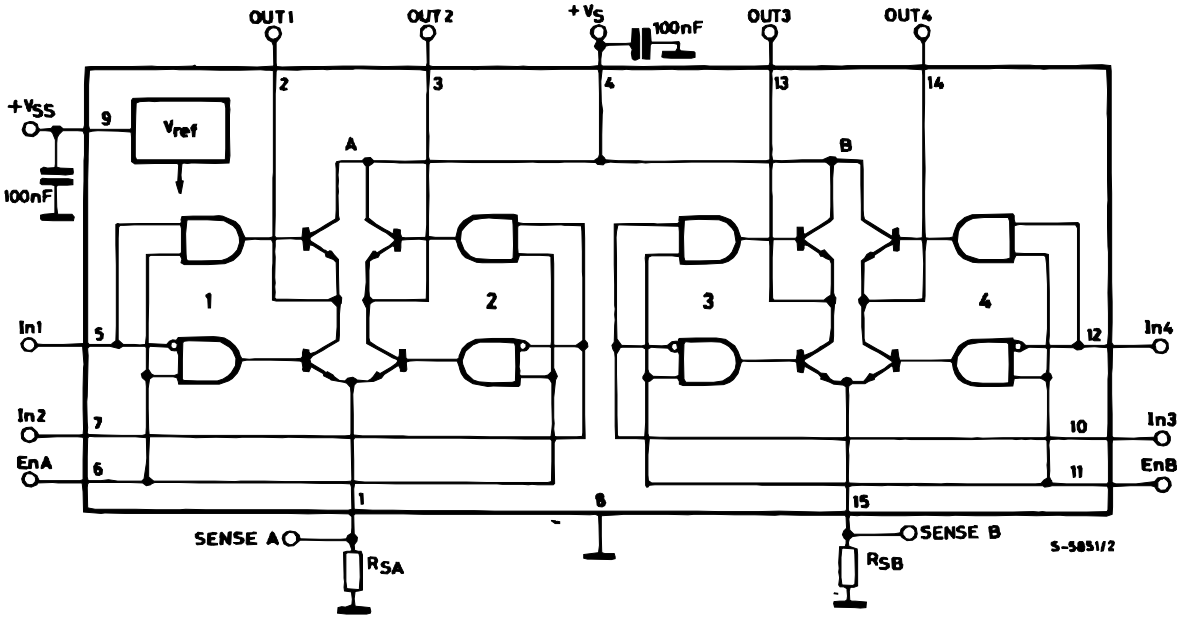
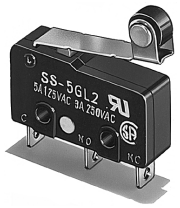
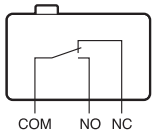


Figure 16: PMOD DC-Stepper Control board

4.5 End of turn switch

The end of turn switch can be plugged into any PMOD connector and is used to identify the steering "zero" position. The used switch is a Omron miniature high reliability and security switch [17].





Color	Pin
Yellow	PMOD I/O
Yellow	GND

Figure 17: Omron SS-5T Miniature High Reliability and Security Switch and its pining

An internal pull-up must be enabled on the FPGA side.

4.6 Hall Sensor

One or two Hall sensors are used to track the distance driven by the kart. The SS311PT/SS411P digital Hall-effect sensors [18] are operated by a magnetic field and designed to respond to ***alternating*** North and South poles with their Schmitt-trigger [19] output.

They can be powered between `2.7Vdc` to `7Vdc` with an open collector output integrating a 10kΩ pull-up resistor already.

Color	Pin
Red	3.3 V power supply
Brown	Hall output
Black	GND

Figure 18: Hall Sensor Honeywell SS311PT and its wiring

No internal pull resistor should be enabled on the FPGA side.

4.7 Bluetooth Dongle NRF52840

The nRF52840 Dongle is a small, low-cost USB dongle that supports Bluetooth 5.4, Bluetooth mesh, Thread, Zigbee, 802.15.4, ANT and 2.4 GHz proprietary protocols [10], [20]. In this project it is used to communicate with the smartphone. The output of the dongle is an UART serial link that is connected to the FPGA. The communication protocol is defined in Section 7.

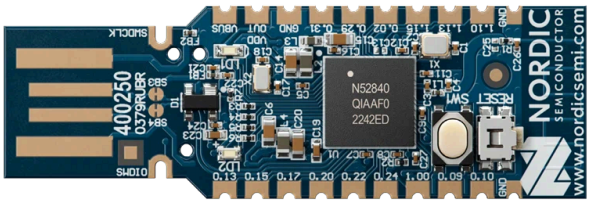


Figure 19: NRF52840 Bluetooth Dongle

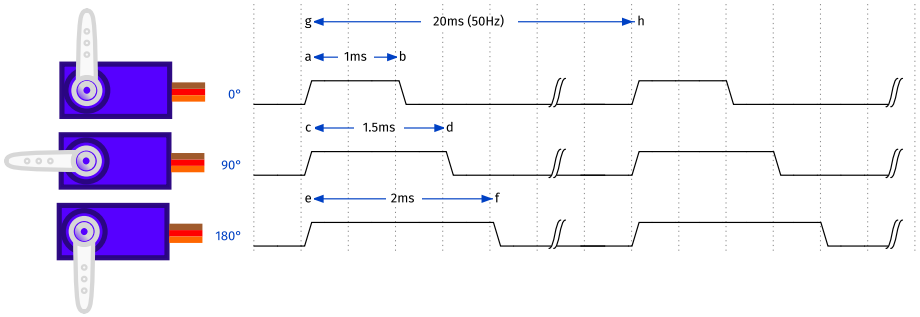
4.8 Sensors & I/Os

Various sensors can be mounted on the motherboard - Section 4.2 through the exposed PMOD - Appendix B connectors.
Any +3.3V I/O which consumes less than 8 mA can be connected on any PMOD connector (either by directly plugging them into the pin headers or through the IDC cables).

Only **pins 6, 7 and 8 of the PMOD8 CANNOT be used** with the current FPGA.

4.8.1 Servo Motor

Servos are a great choice for robotics projects, automation, RC models and so on. They can be used to drive custom mechanical parts of your RC-Car. The angle is controlled with the control pin. The pulse width of a $f = 50$Hz signal defines the angle, see Figure 20.



Color	Pin
Yellow	Control
Red	5V
Black	GND

Figure 20: Servo Motor Control and pining

Use boards like the PMOD-OD2 - Appendix iii to control such devices.

5 FPGA Design

At least the three different modules must be completed:

- The DC motor controller (Section 5.1) receives a **`prescaler`** and a **`speed value`** to build the corresponding PWM and **`direction`** signals.
- The stepper motor controller (Section 5.2) receives a **`prescaler`** and the desired **`angle`** and builds the **`coil`** controls signals.
- The sensor controller (Section 5.3) manages I/O comprising the **`hall sensors`** to retrieve the driving speed and the **`range finder`** to get the distance from an obstacle (optional).

Contents

5.1 DC Motor Controller

The DC motor controller is in charge of the Kart's propulsion, both in forward and reverse.

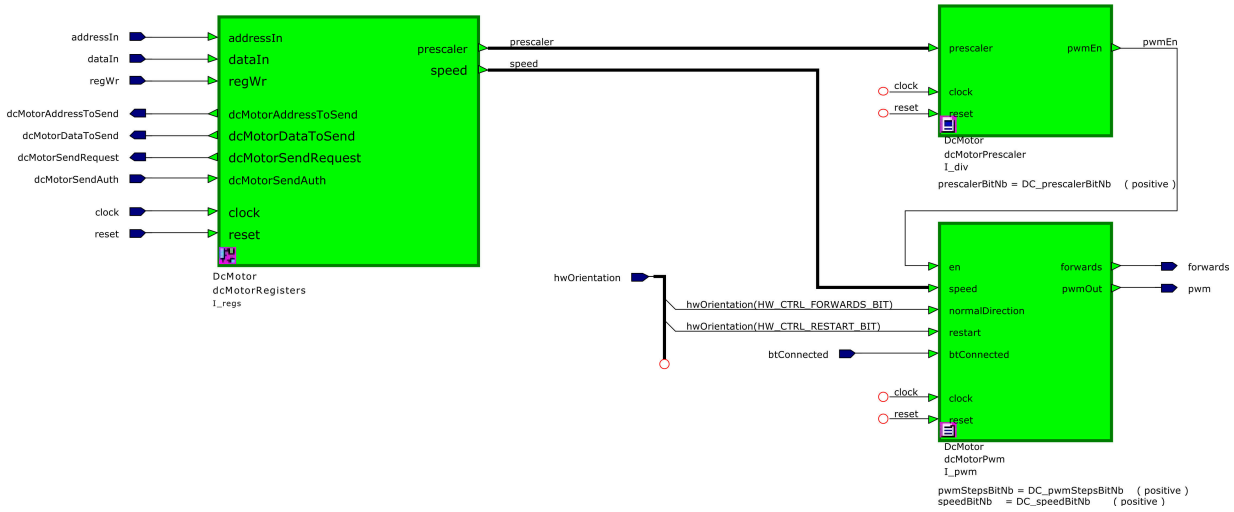


Figure 21: DC module top-level

For this, two signals are generated :

- A **`pwm`** with a controllable frequency whose duty-cycle is modified to control the speed
- A **`forwards`** signal to drive either forward or backward.

The speed is set in the DC motor speed register - Section 7.2, ranging from **`-15=0b1001`** to **`15=0b01111`**.

The frequency is given by the **`prescaler`** in the DC motor prescaler register - Section 7.2 following the formula:

$$f_{\text{PWM\_DC}} = \frac{f_{\text{clk}}}{\text{PWM}_{\text{steps}} * \text{prescaler}} = \frac{10\text{MHz}}{16 * \text{prescaler}} \tag{1}$$

The minimal value of the PWM signal is studied in another part of the project.

5.1.1 Functionality

The **`dcMotorPwm`** block receives the **`speed`** signed number and has to drive the DC motor with the **`pwm`** and **`forwards`** signals. These are then converted to two driving signals controlling a H-Bridge [14].
The mean amplitude of the DC motor's voltage is set by a PWM:

- The **`forwards`** signal is derived from the sign of the **`speed`** control.
- The **`pwm`** signal is derived from the absolute value of **`speed`**.

The PWM signal is implemented with the help of a free-running counter and a comparator. Since the power transistors cannot switch at too high frequencies, the PWM period must be controlled.
This is achieved with the help of an **`en`** (enable) signal generated by the **`dcMotorPrescaler`** block, dividing the clock frequency. The **`dcMotorPwm`** counter must only increment when this signal is **`en='1'`**.

5.1.1.1 PWM Generation

Create a structure which is able to generate the PWM on 16 steps, cadenced by the **`en`** signal and whose duty is set by the absolute value of the **`speed[MSB-1 : 0]`** signal.

Set the **`forwards`** signal **`'1'`** if the Kart should drive forward, **`'0'`** otherwise.

Draw the circuit of the **`dcMotorPwm`** block.

5.1.1.2 Hardware orientation

The mechanical design can either lead the Kart to drive forward or backward when a positive voltage is applied to the DC motor.

In order to cope with this, a setup signal, **`normalDirection`**, is provided to the block. **`normalDirection = '1'`** means that a positive voltage applied to the DC motor lets the kart drive forwards.

Update the circuit in order to cope for the different mechanical design possibilities.

The setup bit is configured in the hardware control register - Section 7.2.

5.1.2 Bluetooth connection

When the Bluetooth connection is lost, the DC motor should not turn to prevent any damage.

A control signal, **`btConnected`**, is provided to the block. When **`btConnected = '0'`**, the DC motor must stop.

If the **`btConnected`** then rises back to **`'1'`**, the motor must not move until the speed register is modified. Else the Kart would dangerously resume moving without the user being ready.

The BT connection bit is given by the hardware control register - Section 7.2.

Update the circuit in order to stop the motor on connection loss and resume only after the connection is resumed AND the speed modified.

5.1.3 Tests

Refer to Section 6.1.1 - DC Motor testing to test your block fully before deploying it on the FPGA.

5.2 Stepper Motor Controller

The stepper motor controller is in charge of the Kart's steering to reach the desired angle.
For this, four signals are generated to control the four coils of the motor:

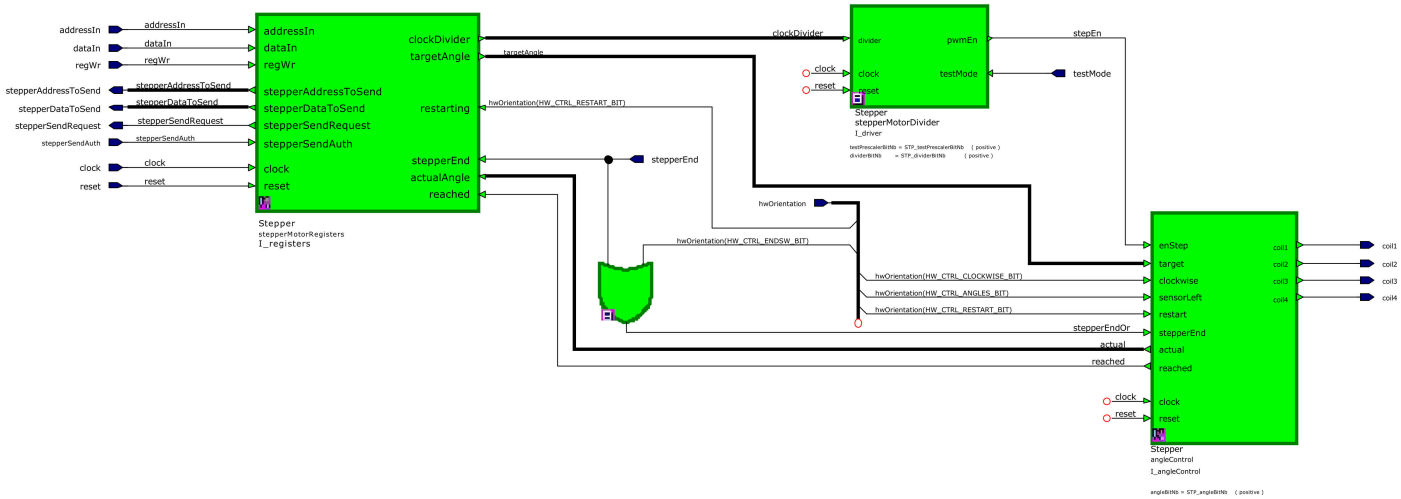


Figure 22: Stepper module top-level

The **`angleControl`** block is in charge to generate the 4 phases to turn the kart's steering wheels to the desired angle.

The unsigned signal **`targetAngle`** is set in the stepper motor target angle register - Section 7.2 and the current angle written to the **`actualAngle`** register.

The frequency is given by the **`prescaler`** register following the formula:

$$f_{\text{PWM\_step}} = \frac{f_{\text{base}}}{\text{prescaler}} = \frac{100\text{kHz}}{\text{prescaler}} \tag{2}$$

Too high frequencies will result in the motor tripping or slipping.

5.2.1 Functionality

The **`angleControl`** block receives a **`targetAngle`** and has to step the coils at each occurrence of the **`enStep`** signal until the desired angle is reached. This signal is generated by the **`stepperMotorDivider`** block.
The coils are controlled through the 4 outputs **`coil1`** to **`coil4`**. When a **`'1'`** is applied, the corresponding coil draws current and becomes magnetized.
The taken steps are updated live on the **`actual`** vector to reflect the current Kart's steering and the flag **`reached`** set to **`'1'`** when the requested position is reached. The information is then automatically sent to the smartphone.

The angle values represent the number of steps of the steppermotor. The maximum angle is 360°, which corresponds to 4800 steps see Section 4.4.2

5.2.2 Driving coils

To control the stepper, one coil at a time should be magnetized, known as wave drive.

Other driving methods such as full-step, half-step and microstepping are further ideas which can be counted as extra features for your Kart, as well as reducing the current consumption with some PWM s.

Each time the signal **`stepEn`** is **`'1'`**, the coils must change state as long as the current angle has not reached the target. For that, a counter must remember how many steps have been taken until now and write it in the vector **`actual`**.

Keep the flag **`reached`** to **`'0'`** as long as the target is not reached, else indicate it with a **`'1'`**.
At this moment, cut off all coils to limit the current consumption. Failing to do so will result in overheating motors.
Finally, ensure that both the first and last pulse of the coils (from resuming the movement to ending it when the target is reached) have the same duration as all the others. Otherwise, the motor will not be correctly magnetized and trip.

The vector **`actual`** and the flag **`reached`** are both used internally to trigger events which will transmit their values to the smartphone. A mishandling of those signals will result in a flood of the communication system, which may in turn imply loss of data with the smartphone, commands not updating and the GUI freezing because of too many interrupts.

Draw the circuit of the **`angleControl`** block.

5.2.3 Zero position

The signal **`stepperEnd`** is linked to the mechanical switch mounted on the Kart which indicates when the steering is at 0.
When this signal rises, the **`actual`** vector should be reset to 0.
With this, the steering motor will not try to turn further than what the kart's mechanical structure allows it to do in one of the directions. In the other direction, it is the programmer's task not to request a too large **`target`** angle. This also means that the angles are always considered as positive numbers, the zero position being given by the **`stepperEnd`** switch.

Update the circuit to integrate the reset based on **`stepperEnd`**.

5.2.4 Hardware orientation

The mechanical design allows for the following variations:

- the **`stepperEnd`** switch can be placed such as to detect the maximal steering angle either on the left or on the right side
- switching the coil controls in the sequence 1 ⇨ 2 ⇨ 3 ⇨ 4 ⇨ 1 ⇨ 2 ⇨ … can result in the kart to turn either to the left or to the right

In order to cope with all possibilities, 2 setup signals are provided to the block:

- **sensorLeft** being **'1'** means that the **stepperEnd** switch is placed on the left side
- **clockwise** being **'1'** means that the sequence 1 ⇨ 2 ⇨ 3 ⇨ 4 ⇨ 1 ⇨ 2 ⇨ ... lets the steering turn to the right

The corresponding setup bits are configured in the hardware control register - Section 7.2.

Update the circuit of the **angleControl** block in order to cope for the different mechanical design possibilities.

5.2.5 Initial position

The hardware control register - Section 7.2 also contains the bit **restart** indicating that the stepper motor has to turn all the way back until reaching the **stepperEnd**.

This bit is set by the remote control smartphone after a successful Bluetooth pairing, together with the appropriate **sensorLeft** and **clockwise** setup bits, in order for the FPGA hardware to discover the zero angle position. This should only be done after the stepper motor period register has been set to a proper value.

Resetting to the zero position can also prove beneficial in the event of excessive current draw from the batteries, leading to a power dip in the FPGA that triggers a reset.

The **restart** signal remainds in a high state until either the **stepperEnd** button is pressed or it is reset by the smartphone.

- Consequently, upon detecting a restart, the system needs to transistion into a restart state and proceed until the **stepperEnd** button is pressed.
- If the **restart** signal falls while the **stepperEnd** has still not been pressed, the restart must continue.
- If the switch is already pressed when **restart** rises, it stays on for only one clock period.

The stepper motor must wait for the **restart** signal to be released before acting normally. Also, it should not turn while a first **restart** has not been performed.

Update the circuit of the **angleControl** block in order to turn all the way to the **stepperEnd** position once the **restart** signal is set.

5.2.6 Tests

Refer to Section 6.1.2 - Stepper Motor testing to test your block fully before deploying it on the FPGA.

5.3 Sensors Controller

The sensors controller handles the other I/Os of the system:

- Generic, 0-3.3V digital inputs. *Up to 16 inputs.*
- Generic, 0-3.3V digital outputs. *8 available outputs.*
- Dedicated I2C battery voltage and current reader.
- Dedicated hall sensors reader.
- Dedicated supersonic range finder pulse reader.

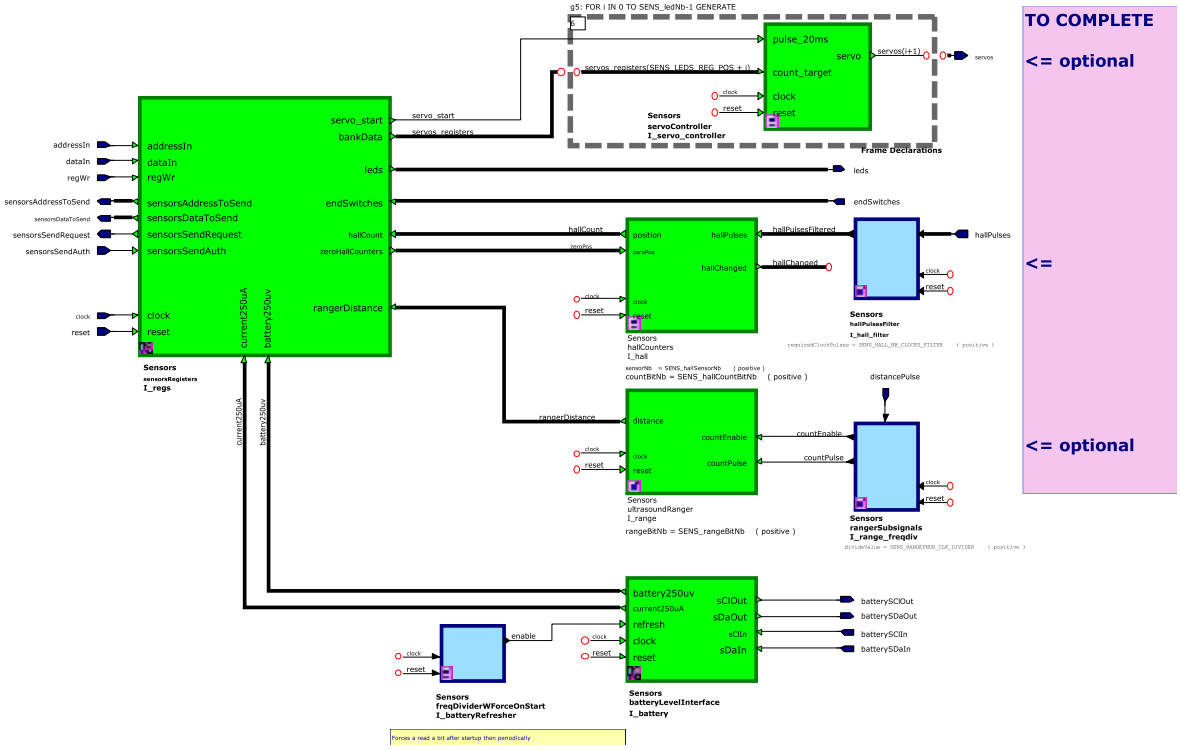


Figure 23: Sensors module top-level

5.3.1 Functionality

The **`leds`** vector is controlled by the smartphone and used for generic outputs at will. It can simply set those on or off, but also handle toggling them at predefined frequencies. See the LEDx registers - Section 7.2.

The **`endSwitches`** vector is used as generic inputs. Each transition of their value is forwarded to the smartphone.

As the state changes of the **`endSwitches`** are transmitted to the smartphone, these inputs are not designed for rapid fluctuations. Swift changes could potentially overwhelm the communication system, leading to issues such as data loss with the smartphone, commands failing to update, and the graphical user interface (GUI) freezing due to an excessive number of interrupts.

The **`freqDividerWForceOnStart`** along the **`batteryLevelInterface`** blocks handle communication with the motherboard's I2C voltage and current reader - see Section 4.2.1.3. Both values are read

periodically, typically each second, and forwarded to the smartphone when they change from at least a predetermined value.

5.3.2 Hall counter

The hall sensors presented in Section 4.6 create pulses based on the speed of the Kart. They are first passing through the **`hallPulsesFilter`** block which is implemented and does the following:

- Get the raw pulses from the hall sensors
- Debounces the input to avoid false transitions
- Transmits the filtered pulses to the **`hallCounters`** block

The **`hallPulses`** vector contains one signal per hall pulse sensor.

Even if you only use one hall sensor, the second exists, just held to **`'0'`** *at all time.*

The following behavior must be implemented:

- The **`hallCounters`** block receives the **`hallPulses`** vector and has to count the number of pulses for each sensor.
- If the vector **`zeroPos`** is set to **`'1'`** for a particular sensor, the corresponding **`hallCount`** value is reset. Subsequently, if the zeroPos vector is **`'0'`**, the dedicated counter should be incremented either on:
  - The rising edge of the pulse
  - Both the rising and falling edges of the pulse

How the counting behavior should be processed depends on your Kart. If you prefer counting on both edges for better precision, set the corresponding value in the **Kart/Kart_Student** package. Adjust the variable **`HALLSENS_2PULSES_PER_TURN`** to **`'1'`** if the signal is counted on each edge, or **`'0'`** if counted only on one edge.

The signal ***`hallChanged`*** *is currently unused.*

The two hall counters values must be concatenated to the **`hallCount`** vector. The second counter needs to be shifted by 16bit: $\text{hallCount} = (\text{hallCount}_2 \ll 16) + \text{hallCount}_1$. It must **AT ALL TIME** reflect the current counters values.

Draw the circuit of the **`hallCounters`** block.

5.3.2.1 Tests

Refer to Section 6.1.3 - Sensors testing to test your block fully before deploying it on the FPGA.

5.3.3 Ultrasound Ranger *(Optional)*

An ultrasound ranger is useful to detect obstacles in the front or back of the Kart. Based on the PMOD-MAXSONAR board from Digilent, it can be plugged into any one-row PMOD connector.

Beware not to wire it on the +12V pin !

The ranger outputs a pulse named **PW** whose length is to be counted. The distance to an object is then determined following the rule $147\frac{\mu s}{\text{inch}} = 57.84\frac{\mu s}{\text{cm}}$.

There is no start/stop indication: the sensor continuously outputs pulses between 0.88 and 37.5 ms long, each 49 ms.

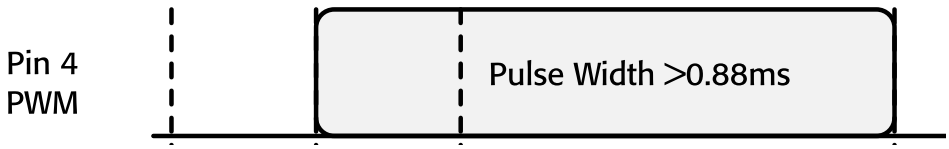


Figure 24: MaxSonar PW pulse

The pulse first goes through the **`rangerSubsignals`** block which does the following:

- Gets the raw **`distancePulse`** signal, continuously coming in as the sensor is configured to always take measurements
- Around each 333 ms, it waits for a fresh measurement, i.e. **`distancePulse`** falling then rising again
- The **`countEnable`** signal will then rise to indicate that **`countPulse`** must be counted
- **`countPulse`** creates a pulse every 1 us, as counting the clock directly is impossible since an overflow of the register may occur
- **`countEnable`** finally falls at the end of the measurement

So the block **`ultrasoundRanger`** must implement the following behavior:

- Wait for the signal **`countEnable`** to be **`'1'`**
- Count the pulses from **`countPulse`** as long as **`countEnable`** is **`'1'`**
- When it goes back to **`'0'`**, the counter value must be output to the **`distance`** signal

The signal **`distance`** must be updated only when **`reset = '1'`** or when **`countPulse`** falls, the signal being kept intact until the next measurement is completed.

Draw the circuit of the **`ultrasoundRanger`** block.

5.3.3.1 Tests

Refer to Section 6.1.3 - Sensors testing to test your block fully before deploying it on the FPGA.

5.3.4 Servomotors controller *(Optional)*

Servomotors are easy to control and allow for many applications requiring circular motions. Those can also be transformed easily into linear ones with the help of a bit of mechanic.

A typical servomotor requires a pulse each 20 [ms] whose duration ranges from 1 (-90°) to 2 [ms] (90°) as shown in the following figure:

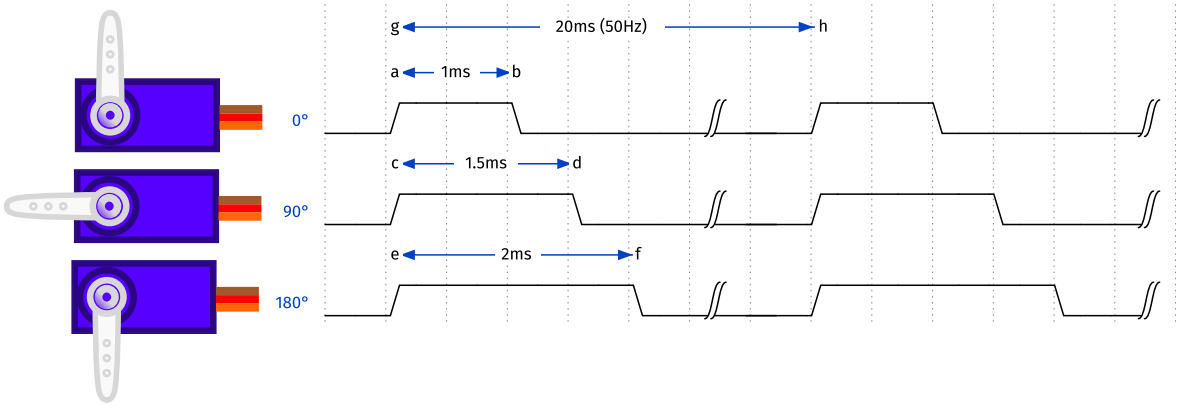


Figure 25: Servo Motor Control and pining

The block **`servoController`** must implement the following behavior:

- Set the output **`servo`** to **`'0'`**
- Wait for the signal **`pulse_20ms`** to be **`'1'`** *(only lasts for one clock period)*
- Set the output to **`'1'`** and begin counting
- Once the count value corresponds to the one given in **`countTarget`**, set the output back to **`'0'`**
- Wair for the next **`pulse_20ms`** to start again

Draw the circuit of the **`servoController`** block.

5.3.4.1 Tests

Refer to Section 6.1.3 - Sensors testing to test your block fully before deploying it on the FPGA.

6 Testing

Three types of testers are available to fully validate the design before flashing the FPGA.
Per module simulation - specific functionalities of the circuit:

- DC Motor Module - Section 6.1.1
- Stepper Motor Module - Section 6.1.2
- Sensors Module - Section 6.1.3

Circuit simulation - overall circuit behavior:

- Overall circuit (modules only) - Section 6.2.1
- Full circuit (with COM emulation) - Section 6.2.2

Finally, a USB tester - Section 6.5 allows to test and control the Kart by using a PC to emulate the smartphone by connecting it directly via USB.

Always complete simulations tests before any wiring and programming of the board. Always use a stabilised DC power supply while developing.

Contents

6.1 Per module

Each module can be tested individually.

6.1.1 DC Motor testing

The DC motor functionality can be tested through the **`DCMotor_test`** ⇨ **`dcMotorController_tb`** block.

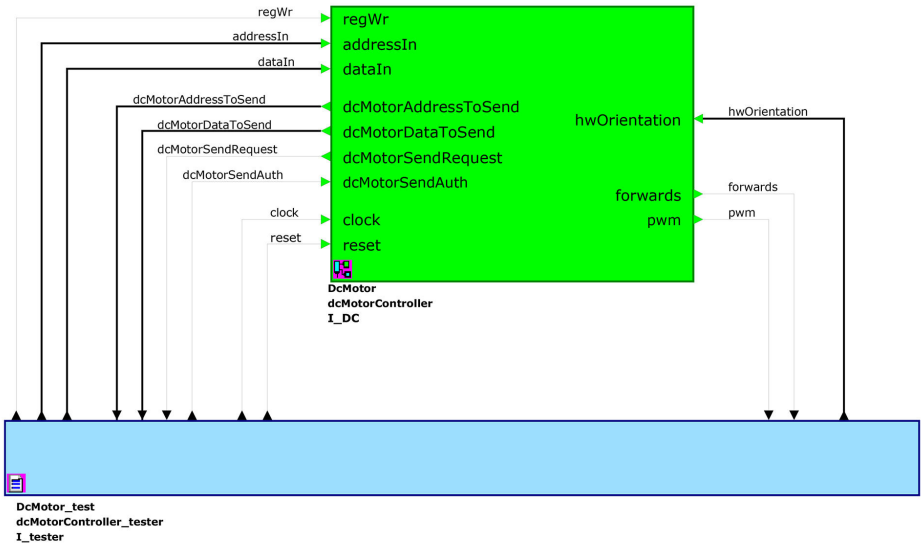


Figure 26: DC Module Testbench

The corresponding simulation layout file for Modelsim is available under **`$SIMULATION_DIR/DCMotor/dcMotorController.do`**. It will automatically run the simulation if it has not been modified.

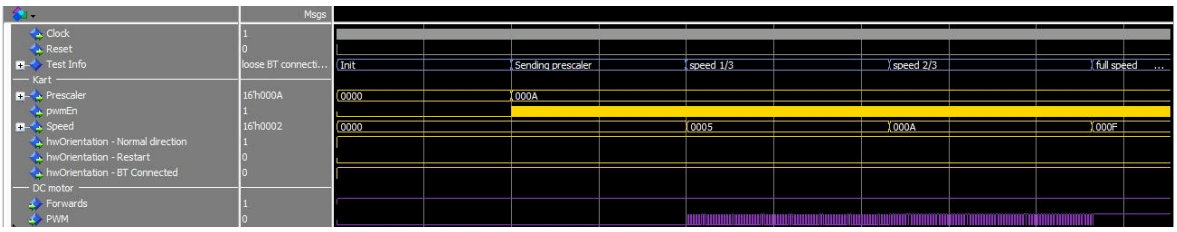


Figure 27: DC Module Simulation

- The blue header shows which test is performed
- The yellow signals are those generated by the tester
- The purple signals are the one generated by your implementation

6.1.1.1 Testing

The following tests are performed:

- The **`prescaler`** signal is set to **`10`**
- Positive speeds of **`33%`**, **`66%`** and **`100%`** are tested, expecting a correct **`PWM`** period and **`forwards = '1'`**
- Negative speeds of **`33%`**, **`66%`** and **`100%`** are tested, expecting a correct **`PWM`** period and **`forwards = '0'`**

- The **`normalDirection`** is inverted
- Positive speeds of **`50%`** and **`100%`** are tested, expecting a correct **`PWM`** period and **`forwards = '0'`**
- Negative speeds of **`50%`** and **`100%`** are tested, expecting a correct **`PWM`** period and **`forwards = '1'`**
- **`Restart`** is held at **`'1'`** while **`normalDirection`** is inverted again, expecting **`PWM = '0'`** and **`forwards = '1'`**
- **`Restart`** is released, expecting correct **`PWM`** and **`forwards = '1'`**
- The **`btConnected`** is lost, expecting **`PWM = '0'`**
- The **`btConnected`** is retrieved, expecting **`PWM = '0'`** (by security, the Kart should not restart until receiving a new **`speed`** command)
- A **`speed`** of **`0`** is set, expecting **`PWM = '0'`** and **`forwards = '1'`** (in the real world, it is automatically done by the smartphone once reconnected)
- A **`speed`** of **`50%`** is set, expecting a correct **`PWM`** period and **`forwards = '1'`**

Transcript

The transcript window gives you details on if tests passed or not:

- For the **`PWM`** signal, you get **`PWM OK`** or **`PWM Error`**.
- For the **`forwards`** signal, you get **`Direction OK`** or **`Direction Error`**.

6.1.2 Stepper Motor testing

The stepper motor functionality can be tested through the **`StepperMotor_test`** ⇨ **`stepperMotorController_tb`** block.

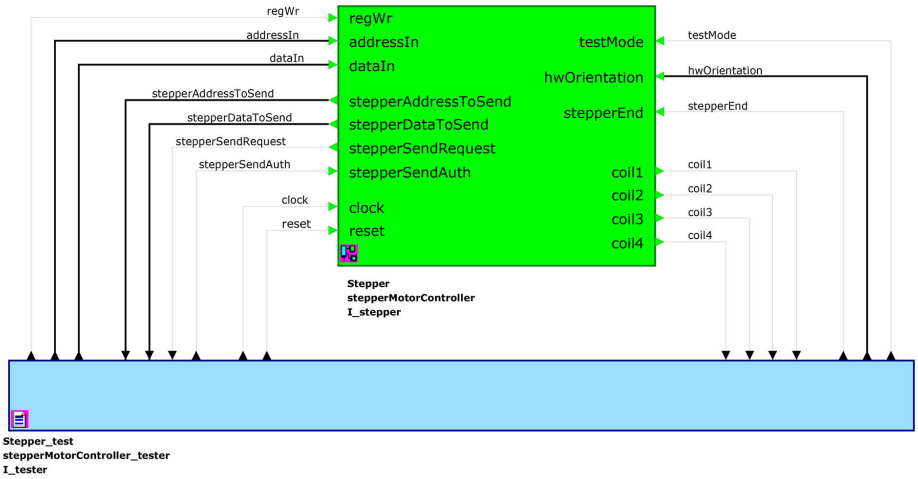


Figure 28: Stepper Module Testbench

The corresponding simulation layout file for Modelsim is available under **`$SIMULATION_DIR/Stepper/stepperMotorController.do`**. It will automatically run the simulation if it has not been modified.

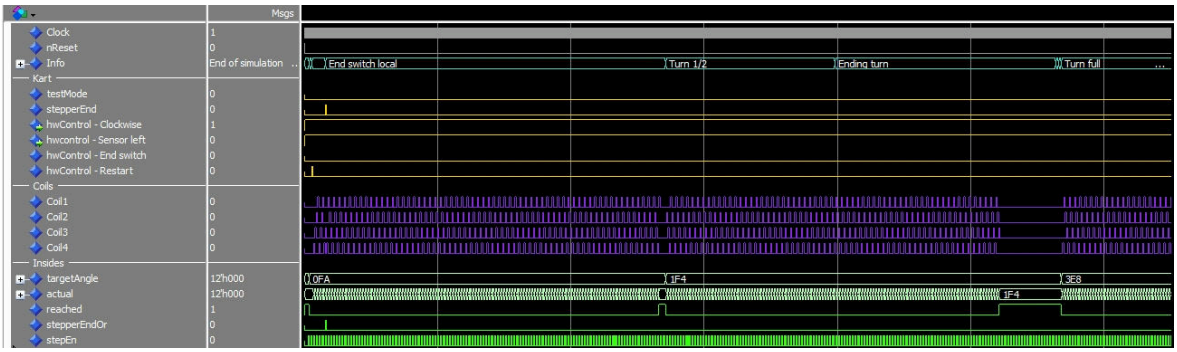


Figure 29: Stepper Module Simulation

- The blue header shows which test is performed
- The yellow signals are those generated by the tester
- The purple signals are the one generated by your implementation
- The green signals are internal ones

6.1.2.1 Testing

The following tests are performed :

- The **`prescaler`** is set, outputting pulses on **`stepEn`**
- The **`restart`** signal pulses (both one pulse or constant **`'1'`** are valid), expecting the coils to turn as 4 ⇨ 3 ⇨ 2 ⇨ 1 ⇨ 4...
- The **`restart`** signal is released, expecting the coils to keep turning
- The **`stepperEnd`** is pressed, expecting the coils to stop
- A **`target`** of **`250`** is set, expecting the coils to move in order 1 ⇨ 2 ⇨ 3 ⇨ 4 ⇨ 1...
- The **`reached`** signal should be **`'1'`** when **`250`** pulses have been made

- A **target** of **500** is set, expecting same behavior as before
- The **target** suddenly changes back to **0**, expecting the coils to move in the reversed sequence of 4 ⇨ 3 ⇨ 2 ⇨ 1 ⇨ 4…
- The **reached** signal should be **'1'** when **actual** reaches **0**
- A **target** of **250** is set
- A new **restart** is issued as before, while the sensor left signal is set to **'0'**, expecting the coils to turn as 1 ⇨ 2 ⇨ 3 ⇨ 4 ⇨ 1…
- A **target** of **250** is set, expecting the coils to move in order 4 ⇨ 3 ⇨ 2 ⇨ 1 ⇨ 4…
- The **reached** signal should be **'1'** when **250** pulses have been made
- A **target** of **0** is set, expecting the coils to move as 1 ⇨ 2 ⇨ 3 ⇨ 4 ⇨ 1…
- The **reached** signal should be **'1'** when **actual** reaches **0**
- A new **restart** is issued as in the beginning after trying to reach a **target** of **500**, expecting the coils to turn as 4 ⇨ 3 ⇨ 2 ⇨ 1 ⇨ 4…
- The **hwControl - end switch** (emulated **stepperEnd**) is pressed, expecting the coils to stop

Transcript

The transcript window gives you details on if tests passed or not:

- For the **Coil1...4** signals, you get **Coil direction OK** or **Coil direction error**.
- For the **reached** signal, you get **Reached flag OK** or **Reached flag error**.
- For the **actual** signal, you get **Position readback OK** or **Position readback error**.

6.1.3 Sensors Controller testing

The Hall Sensor, Ultrasound Ranger and servomotors outputs can be tested through the **`Sensors_test`** ⇨ **`sensors_tb`** block.

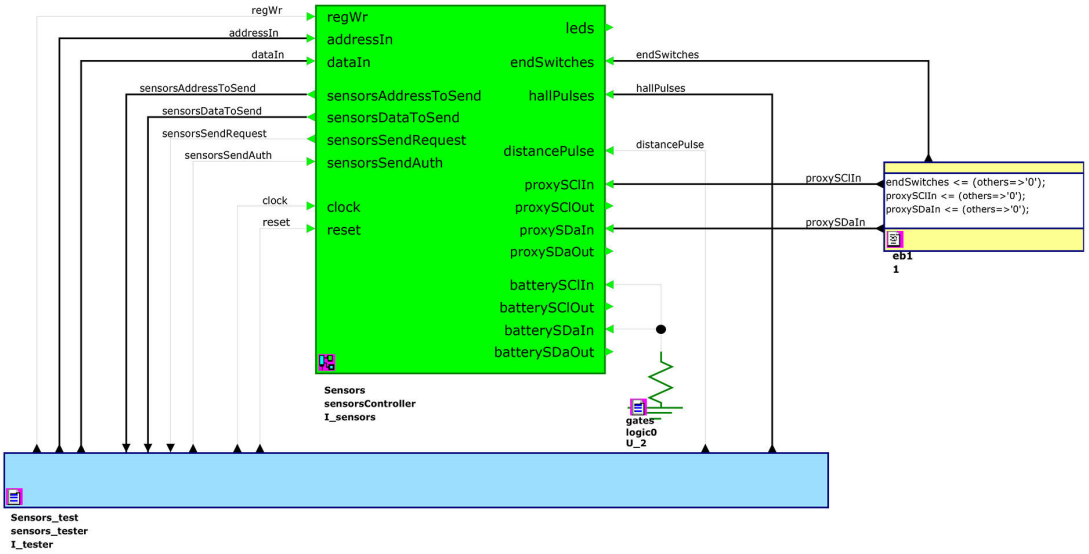


Figure 30: Sensors Module Testbench

The corresponding simulation layout file for Modelsim is available under **`$SIMULATION_DIR/Sensors/sensors.do`**. It will automatically run the simulation if it has not been modified.

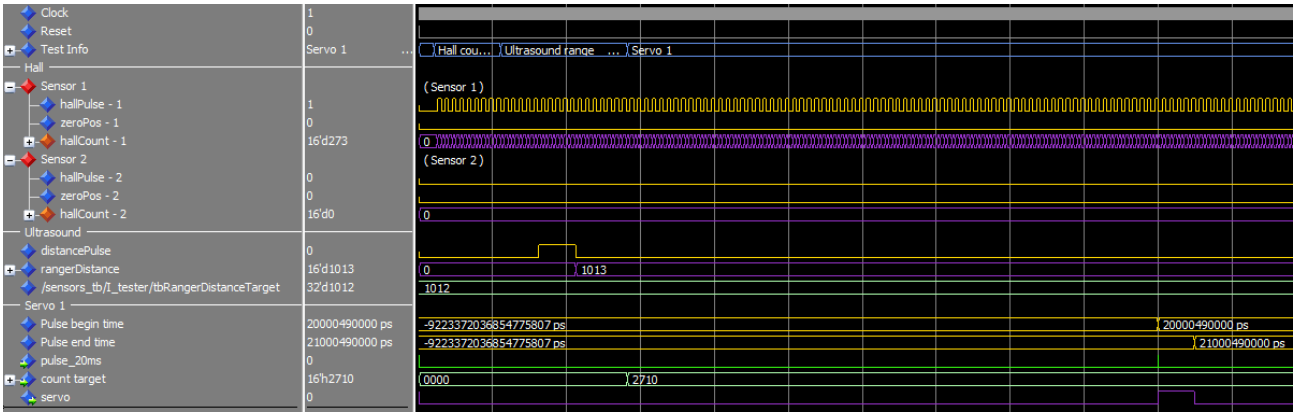


Figure 31: Sensors Module Simulation

- The blue header shows which test is performed
- The yellow signals are those generated by the tester
- The purple signals are the one generated by your implementation
- The green signals are internal ones

6.1.3.1 Testing

The following tests are performed for the Hall sensors:
- For each sensor, a pulse of a fixed period (n times slower for each new sensor) is generated
- The **`hallCount`** is compared to an internal counter, depending on the **`HALLSENS_2PULSES_PER_TURN`** constant
- When **`zeroPos`** goes to **`'1'`**, the **`hallCount`** should be reset

The following tests are performed for the Ranger:
- A pulse of $\mathrm{tbRangerDistanceTarget}(1012) * \left(\frac{T_{\mathrm{clk}}}{10}\right)$ is generated
- When the signal **`distancePulse`** falls, the **`rangerDistance`** should be a value within 1011 to 1013

The following tests are performed for the Servomotors:
- Register **`LED1`** is loaded with value **`10'000`**, corresponding to a 1 [ms] pulse.
- The tester waits for the **`servo`** signal to rise, saving the current time value
- When the **`servo`** signal falls, the time value is saved
- Pulse duration should be 1 [ms]

Transcript

The transcript window gives you details on if tests passed or not:
- For the **`hallCount`** signal, you get **`Hall count n OK`** or **`Hall count n error`**.
- For the **`rangerdistance`** signal, you get **`Ultrasound ranger OK`** or **`Ultrasound ranger error`**.
- For the **`servo`** signal, you get **`Servo 1 pulse is (not) 1 ms long`**, with the found value written if not OK.

6.2 Whole circuit

6.2.1 Modules Simulation

In addition to the dedicated modules testers, the overall behavior can be tested through the **`Kart_test`** ⇨ **`kartController_tb`** block.

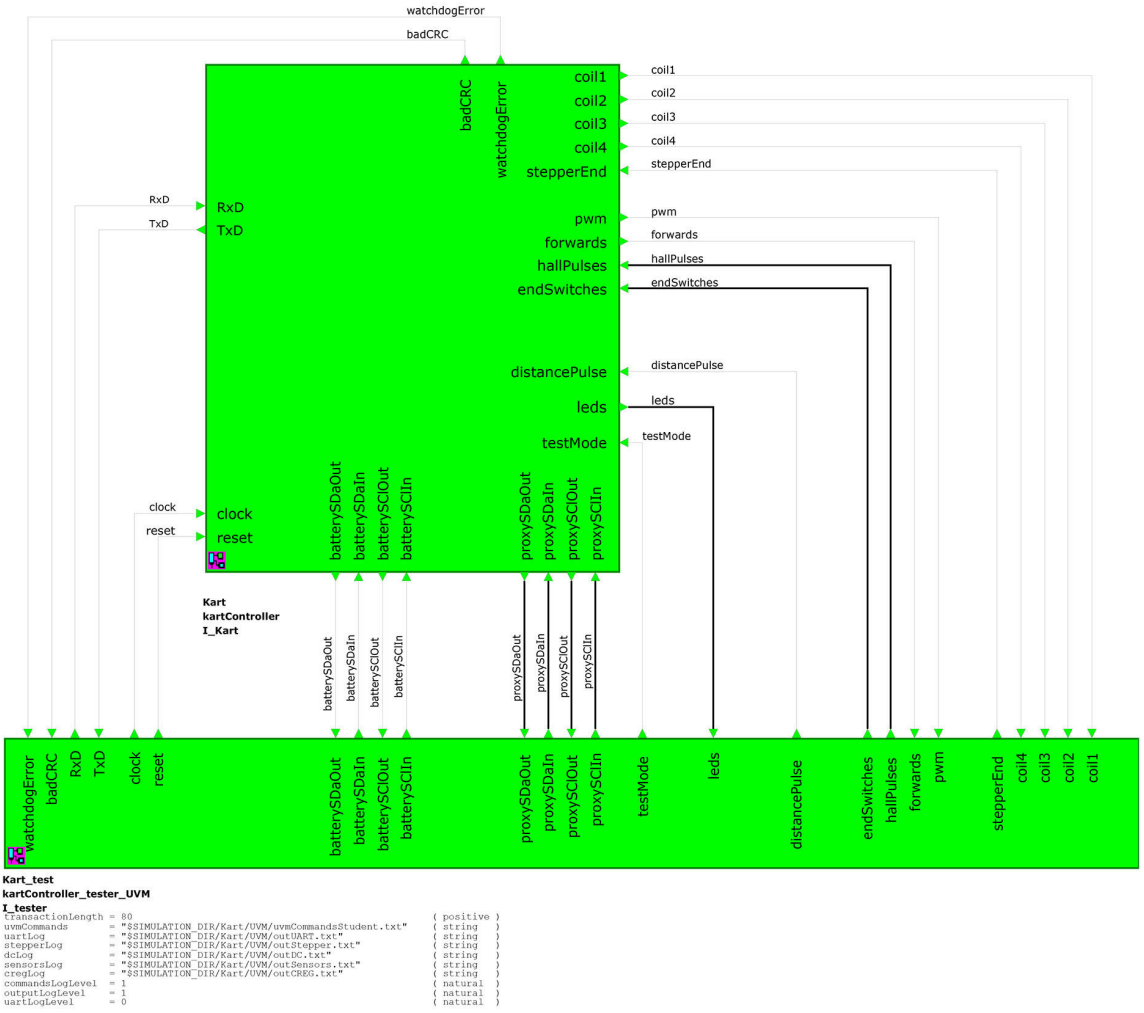


Figure 32: Kart Toplevel Testbench

The tester internal layout differs because it makes use of the UVM technology - see Figure 33.

The tester reads the commands given in **`$SIMULATION_DIR/Kart/UVM/uvmCommandsStudent.txt`** and creates different logs under **`$SIMULATION_DIR/Kart/UVM/outXXX.txt`**. The file can be modified without the circuit being recompiled. The corresponding simulation layout file for Modelsim is available under **`$SIMULATION_DIR/Kart/kartStudent.do`**. It will automatically run the simulation if it has not been modified.

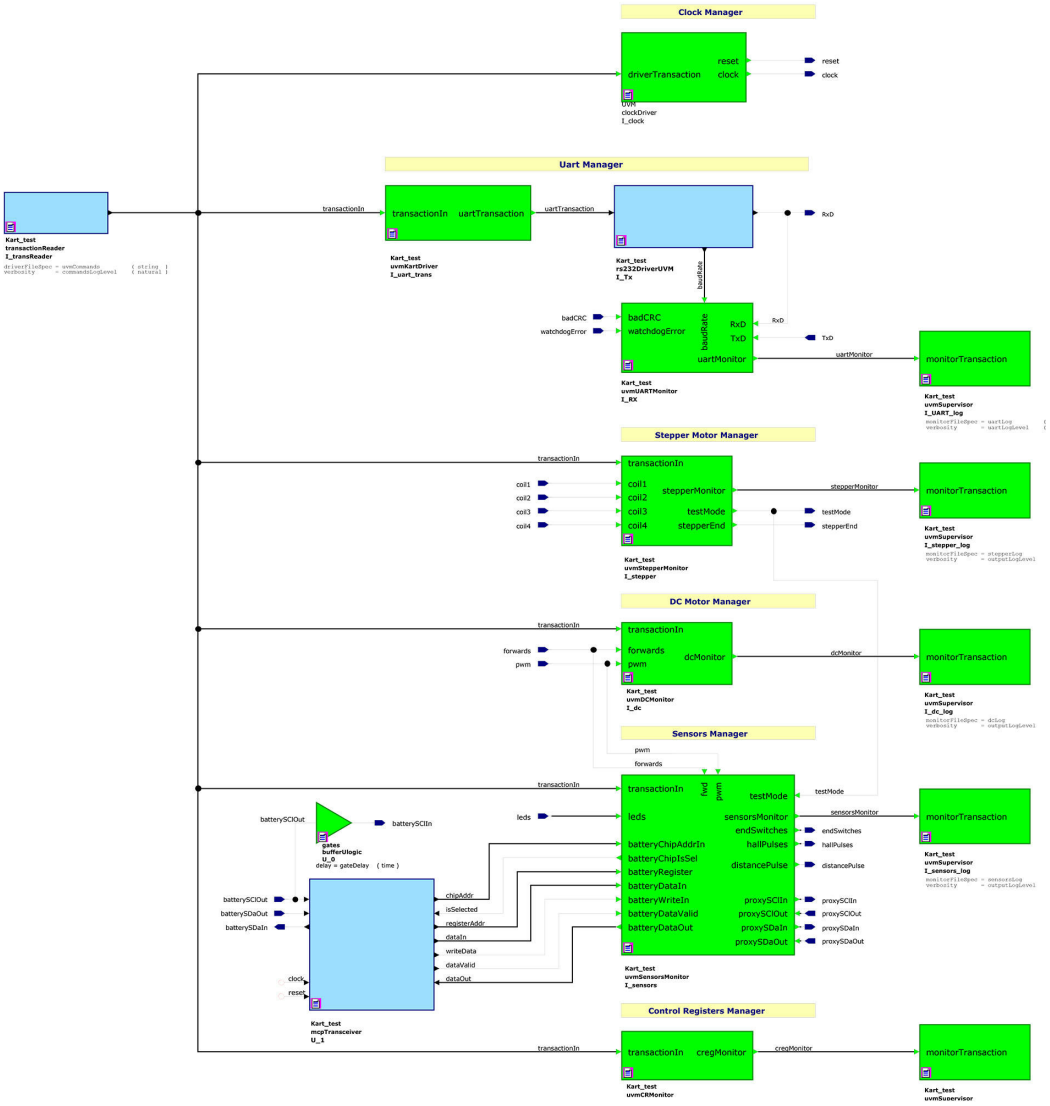


Figure 33: Kart Toplevel Tester

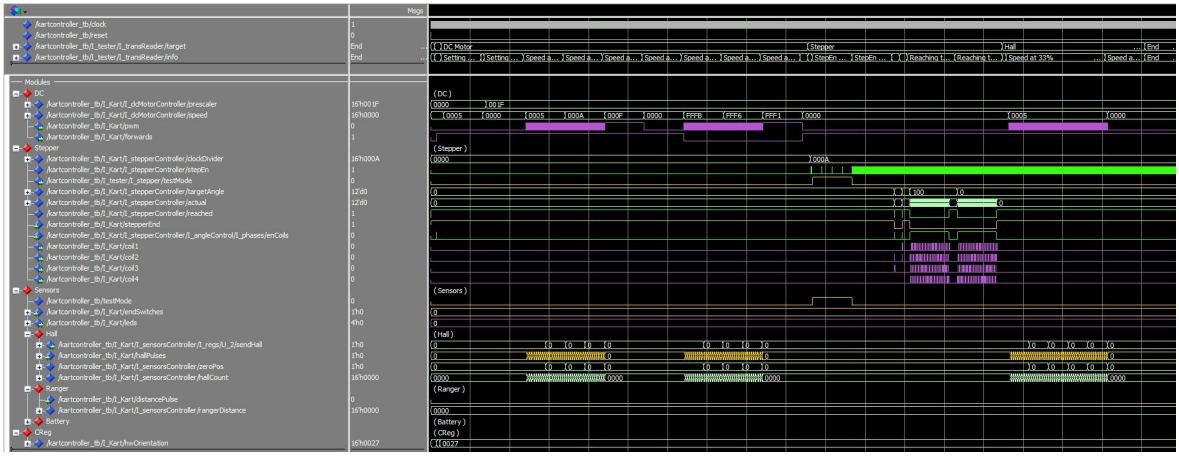


Figure 34: Kart Toplevel Simulation

6.2.1.1 Tests

All the student-designed blocks are tested (check both **`target`** and **`info`** signals on the simulation). Contrary to the other testers, there is no direct error logging. One must check the functionality "by hand" in the simulation window (by correlating signals with the info from the transcript window or the log files).

6.2.2 Full-board

Another tester, checking the whole system (including Rx/Tx frames, registers managers ...) can be loaded through the **`Kart_test`** ⇨ **`kartController_full_tb`** block and the **`$SIMULATION_DIR/Kart/kartStudent.do`** file.

It is mostly intended for people developing the full circuit, but left there for curious people:

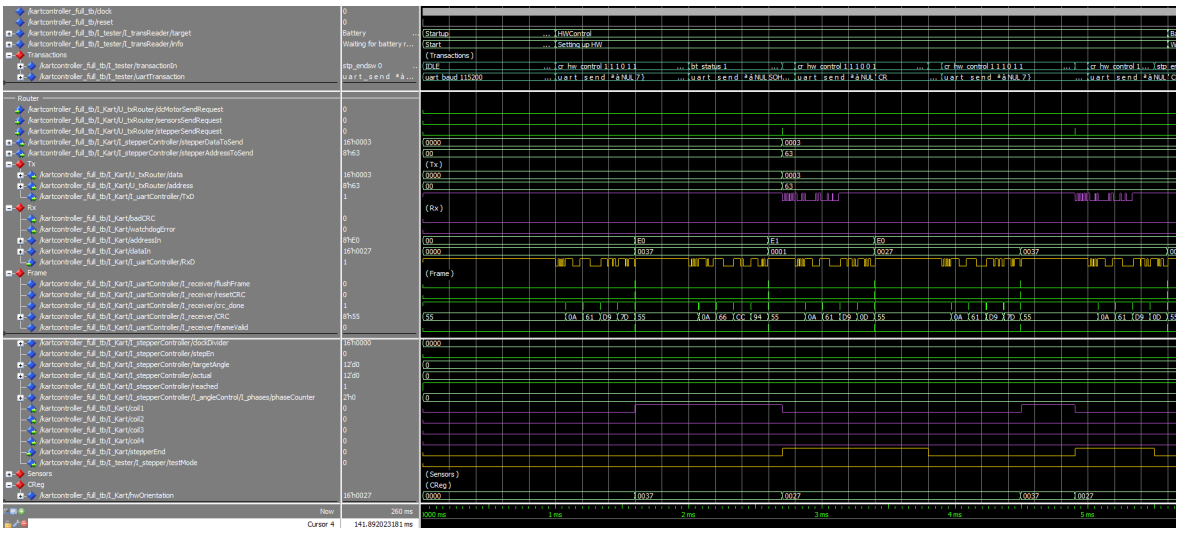


Figure 35: Kart Full Simulation

6.3 Setting up the board

Once your whole circuit is working in simulation, the last step is to tell the FPGA which physical pin corresponds to which signal and setup how many inputs and outputs you really use.

6.3.1 I/Os number configuration

Open the **`Kart/Kart_Student`** package, and set the following constants:

- **`HALLSENS_2PULSES_PER_TURN`**: **`'0'`** if you count only one pulse per turn on the hall sensor, else **`'1'`** if you count twice per turn
- **`STD_HALL_NUMBER`**: the number of hall sensors used, either **`1`** or **`2`**
- **`STD_ENDSW_NUMBER`**: the number of inputs used, named **`endSwitches`** at the board level, up to **`16`**
- **`STD_LEDS_NUMBER`**: the number of outputs used, named **`leds`**, **`servos`** and **`servos_inverted`** at the board level, up to **`8`**

LEDs and Servos
LEDs and SERVOs are shared among the system. Here is an example:
- I have two 'leds' type outputs, and one servomotor.
- I set **`STD_LEDS_NUMBER`** to **`3`**.
- In the pining (see next chapter), I assign **`{leds[1]}`** and **`{leds[2]}`** to my two outputs, and **`{servos_inverted[3]}`** to the servomotor output *(**`_inverted`** because I use the PMOD-OD2 board).*
- From my application, I set registers **`LED1`** and **`LED2`** as defined in Table 7 for leds, and **`LED3`** as defined for servos.

6.3.2 Pining setup

This is done by altering the constraints file found under **`Board/concat/Kart.pdc`**.

All PMODs are listed, along with other signals such as the clock, I2C, UART ...

A simple signal is defined with its name like **`stepperEnd`**, and signals from a vector are written such as **`{leds[1]}`**.

Warning
Do not modify the signals which are not linked to PMODs I/Os.

Inputs
To wire an input, set the IO to the correct signal name such as:
`set_io myVHDLSignalName -pinname 97 -fixed yes -DIRECTION Input`
You can also append **`-RES_PULL Up`** and **`-RES_PULL Down`** to the end of the line to enable a pull-up or pull-down resistor.

Outputs
To wire an output, set the IO to the correct signal name such as:
`set_io myVHDLSignalName -pinname 97 -fixed yes -DIRECTION Output`

Refer to the PMOD - Appendix B pages to know how boards should be used, with or without pull resistors ...

Valid signals

Only signals found in the **Board/Kart_Board** VHDL block can be used:

Inputs

- **stepperEnd**: where the stepper end switch is connected
- **distancePulse**, *opt*: where the ultrasound ranger PWM pin is connected
- **{halls[x]}**, from 1 to STD_HALL_NUMBER: where the hall sensors are connected
- **dc_A**, **dc_B**: where the DC motor control pins are connected
- **{endSwitches[x]}**, from 1 to STD_ENDSW_NUMBER, *opt*: where digital, 3.3V inputs are connected

Outputs

- **coil1** to **coil4**: where the coils of the stepper motor are connected
- **{leds[x]}**, **{servos[x]}** and **{servos_inverted[x]}**, from 1 to STD_LEDS_NUMBER, *opt*: where the digital outputs are connected

6.3.3 Onboard LEDs

The red led indicates if the stepper end switch is pressed.
The yellow led toggles on and off when a magnet is rotated in front of the hall sensor.
The green led is on when the smartphone is connected to the BLE module.

6.4 Programming the board

Professor validation
Your design, constraints file and overall wiring must have been validated before any FPGA's programming.

Refer to Libero - Appendix III to use and deploy your design thanks to the Libero IDE.

Feel free to ask your professors for a quick introduction to the tool flow.

6.5 USB commands emulation

Power Precautions
Use a laboratory power supply limited from $0.05A$ (no motors) up to $1.2A$ (with motors).

To test the communication directly from a PC, the tool **EBS3 UART Interpreter** is available in the project folder under **CommandInterpreter** either as a Windows executable, Linux executable or Python script.

Hereafter the steps to follow in order to communicate with the Kart:

- Power the circuit off
- Remove the BLE module - Section 4.7 from the motherboard - Section 4.2
- Power the motherboard - Section 4.2 with a regulated DC voltage supply with +12V
- Wire the USB-C present on the daughterboard - Section 4.3 to your PC
  - Two new UART COM ports should be detected
- Download and/or open the Kart Command Interpreter utility (available in the VHDL project in the folder **`CommandInterpreter/`**)
  - Linux
  - Windows
  - Source code
- In the top menu **`Serial`** ⇨ **`Port`**, select the correct COM port
  - should be the biggest of the two new ports

To test the connection, click the **`Read`** button. The **`Tx`** and **`Rx`** values should change, with **`Rx`** becoming green (frame correctly received), and a text added to the text area.

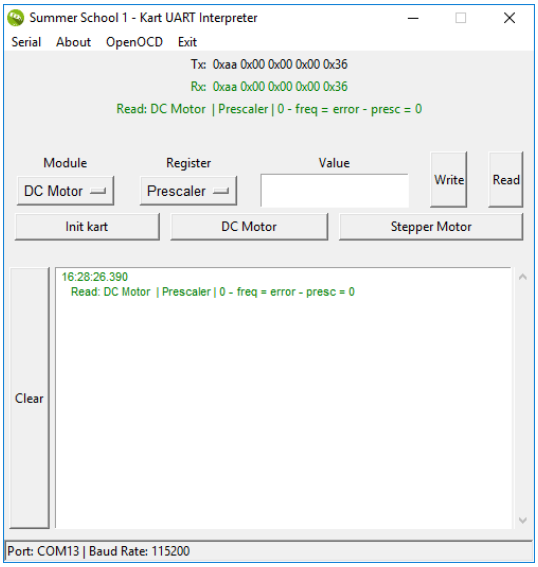


Figure 36: Kart EBS3 UART Interpreter

6.5.1 Quick Test

The simplest way to test both motors are the three following buttons:

Button	Effect	Output
Init Kart	• Set the DC prescaler, stepper prescaler, and execute the restart sequence. **Must be clicked first.** • Answer the 4 prompts following your hardware configuration.	Init Kart DC Prescaler to 31 Stepper Prescaler to 500 BT as connected CReg to reset w. stepper end (0b111111) CReg to normal mode (0b100111) Init done
DC Motor	• Set the speed to full for 2s • Set it to 0 for 2s • Set it to full in reverse for 2s	DC test DC speed tp 15 DC speed tp 0 DC speed t0 −15 DC speed to 0 DC test done *For karts with mounted hall sensors, extra messages will tell the speed.*
Stepper Motor	• Set the stepper to **400** (**30°**) • Detect angle reached • Set the stepper to **0** • Detect angle reached	Stepper test Stepper tp 400 (30°) 10:46:33.169 Read: Stepper Motor | Actual Angle | 28 ... 10:46:34.804 Read: Stepper Motor | Actual Angle | 364 10:46:34.999 Read: Stepper Motor | Stepper HW | stepper open - position reached Stepper to 0 10:46:36.207 Read: Stepper Motor | Actual Angle | 364 ... Read: Stepper Motor | Actual Angle | 28 10:46:38.029 Read: Stepper Motor | stepper open - position reached Stepper test done

Figure 37: Quick Test buttons

6.5.2 Registers R/W

Each register can be read and/or written by hand following their data description - Section 7.2. For this, select the **`Module`** first, then which **`Register`** to access.

Read

To read, simply click the **`Read`** button. Successful read will be shown in green (CRC is ok) and logged, with extra computed informations. For example the DC prescaler logs the motor frequency.

Write

To write, enter a value in the value box such as:

- Direct integer *(only DC speed may be negative)*
- **`0bxxxx`** binary values
- **`0xxxxx`** hexadecimal values

Then click on the **`Write`** button.

Reset
Simply power-cycle the FPGA board to reset all registers.

7 Communication

This section defines the serial link protocol [21] used to communicate between the Kart and the BLE Module - Section 4.7 [10] to the PC or Android Smartphone.

Contents

7.1 General Principle

The system incorporates a BLE module - see Section 4.7. To prevent line congestion, data transmission from the Kart to the User occurs exclusively during specific events or upon the User's request. This strategy helps optimize communication by minimizing unnecessary data transfer and enhancing overall system efficiency.

7.1.1 Serial Port Configuration

The module communicates with the FPGA through UART with the following settings:

Reading State	Data bits	Parity	Stop bits	Handshake	Baudrate
`HIGH`	`8`	`NONE`	`1`	`NONE`	`115'200`

Table 1: Serial Port Configuration

7.1.2 Message format

SoF (1byte)	Address (1byte)	Data (2bytes)	EoF (1byte)
`0xAA`	`UINT8`	`UINT16 / INT16 / VECTOR16` (MSB First)	`CRC8 / ITU`

Table 2: Message Format

The address is decomposed as follows: `0bMMWRRRRR`

- `MM` : targeted module
  - `0b00` : DC Motor
  - `0b01` : Stepper Motor
  - `0b10` : Sensors
  - `0b11` : Control Registers
- `W` : defines if the data is saved to (`'1'`) or read from (`'0'`) the FPGA
  - The FPGA will respond to a request with the exact same address when `W = '0'`
  - The FPGA will save incoming data in the targeted register when `W = '1'`
  - The FPGA will send data on predefined events with the `W` bit set to `'0'`
- `RRRRR` : targeted register

7.1.2.1 Frame example

For the BLE module to light LED1 with it changing each 500 ms, the following frame is sent:

SoF (1byte)	Address (1byte)	Data High (1byte)	Data Low (1byte)	EoF (1byte)
`0xAA`	`0b10100001`	`0b10000001`	`0b11110100`	`0x74`

Table 3: Frame example LED+ 500ms

7.2 Registers

Device	Access	from	to
DC Motor	**Read**	**0x00**	**0x1F**
	Write	**0x20**	**0x3F**
Stepper Motor	**Read**	**0x40**	**0x5F**
	Write	**0x60**	**0x7F**
Sensors	**Read**	**0x80**	**0x9F**
	Write	**0xA0**	**0xBF**
Control	**Read**	**0xC0**	**0xDF**
	Write	**0xE0**	**0xFF**

Table 4: Memory Map

DC Motor

Addr	Name	Type	Description	Direction	Event
0	Prescaler	UINT16	DC PWM frequency f_{PWM} $\frac{f_{\text{clk}}}{\text{PWM}_{\text{steps}} * \text{ prescaler}} = \frac{10\text{MHz}}{16 * \text{ prescaler}}$		
1	Speed	INT5	Desired speed from **-15** (**0xFFF1**) to **15** (**0x000f**) negative = backwards		

Table 5: DC Motor Registers

Stepper Motor

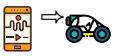
Addr	Name	Type	Description	Direction	Event
0	Prescaler	UINT16	Stepper switching frquency $f_{\text{step}} = 100\frac{\text{kHz}}{\text{prescaler}}$		
1	Target angle	UINT16	Desired steering angle in motor steps 0 = end switch		
2	Actual angle	UINT16	Actual steering angle in motor steps 0 = end switch		When a delta of at least **STP_ANGLE_DELTA_DEG** (2°) from the last registered value happens
3	Stepper HW	UINT14 + Vector2	Bit[0] : stepper end Bit[1]: position reached Bits[15:2]: actual steering angle		Sent when stepper end is pressed (rising edge) or position reached (rising edge)

Table 6: Stepper Motor Registers

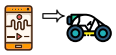
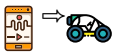
Sensors					
Addr	**Name**	**Type**	**Description**	**Direction**	**Event**
0	LED1	BIT + UINT15	Bit[15]: on - off Bits[14:0]: half-period in ms if 0, led status = bit 15		
	LEDx	...	...	...	
7	LED8	BIT + UINT15	Bit[15]: on - off Bits[14:0]: half-period in ms if 0, led status = bit 15		
0	SERVO1	UINT16	Servo target pulse duration in clock pulses (10´000 = 1 ms)		
	SERVOx	...	...	...	
7	SERVO8	UINT16	Servo target pulse duration in clock pulses (10´000 = 1 ms)		
8	Voltage	UINT16	Battery Voltage $U = \text{register} * 250e^{-6} * 7.8V$		When a delta of at least **SENS_BATT_DELTA_MV** (50) from the last registered value happens
9	Current	UINT16	Consumed current $I = \text{register} * \frac{250e^{-6}}{100*5e^{-3}}$		When a delta of at least **SENS_CURR_DELTA_MA** (50) from the last registered value happens
A	Range finder	UINT16	Distance to sensor $D = \text{register} * \frac{25.4}{147e^{-6}*\left(\frac{10M}{10}\right)}$ Register zeroed if less than 152mm (sensor min distance) or greater than 1500mm (arbitary max distance) Event not sent in such case		When a delta of at least **SENS_RANGEFNDR_MM** (60) from the last registered value happens
B	End Switches	Vec-tor16	Sensors current values Right justified (sensor 1 is bit 0)		On any edge change of any sensor
C	Hall 1	UINT16	Hall pulses count Zeroed on overflow of the register		Each **SENS_HALL_OLD_SEND_TIMEOUT_MS** (100ms) if value changed from last time
D	Hall 2	UINT16	Hall pulses count Zeroed on overflow of the register		Each **SENS_HALL_OLD_SEND_TIMEOUT_MS** (100ms) if value changed from last time

Table 7: Sensors Registers

The **LEDx** and **SERVOx** registers are shared. Use either of the register format and root the pins to the **leds** signals for leds, and **servos** signals for servomotors. See Board Setup - Section 6.3 for more information.

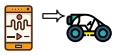
Control					
Addr	**Name**	**Type**	**Description**	**Direction**	**Event**
0	Hardware Control	Vector6	Bit[0]: when **'0'**, the Kart goes backwards when the motor turns forward Bit[1]: when **'1'**, the Kart turns to the right as the stepper coils go from 1 to 4 Bit[2]: when **'1'**, the angles are measured clockwise Bit[3]: emulates the end switch contact for the stepper motor Bit[4]: restart the stepperMotor module and stop the DC motor when **'1'**		The end sensor always defines angle 0. Angles are always positive numbers in registers. If bits 1 and 2 are different, the stepper motor phases sequence has to be inverted.
1	BT Status	Vector1	Bit[0]: when **'0'**, the smartphone is disconnected	NRF⇨ 	The register is set by the NRF itself, since it is not possible to foresee the disconnection. If the bluetooth connection is lost, the kart must stop.

Table 8: Control Registers

7.3 Initialisation Sequence

Multiple registers must be set before the Kart can drive. The following sequence is used by the EBS3 serial interpreter - Section 6.5 :

- Write **`DC Motor`** ⇨ **`Prescaler`** to **`31`** (around 21 kHz PWM frequency)
- Write **`Stepper Motor`** ⇨ **`Prescaler`** to **`400`** (250 Hz coil switching frequency)
- Tell the smartphone it is connected by writing **`Control Registers`** ⇨ **`BT status`** to **`1`**
- Write **`Control Registers`**⇨ **`Hardware Control`** to **`0b10xxx`** to restart the system
  - The stepper should turn until hitting the end switch, except if already on it
- Read **`Stepper Motor`**⇨ **`Stepper HW`** and check the last bit
  - If is **`'1'`**, it means we are already zeroed
  - If not, wait for an event from this register to tell the reset is complete
- Write **`Control Registers`**⇨ **`Hardware Control`** to **`0b00xxx`** to deassert the reset

The **`Control Registers`** ⇨ **`restart bit`** is automatically reset back to **`'0'`** when the **`stepper_end`** input is activated.

The Kart is now ready to function !

Appendices

Contents

A Tools

I HDL Designer

Mentor HDL designer is the tool for graphical design entry as used during laboratories [4]

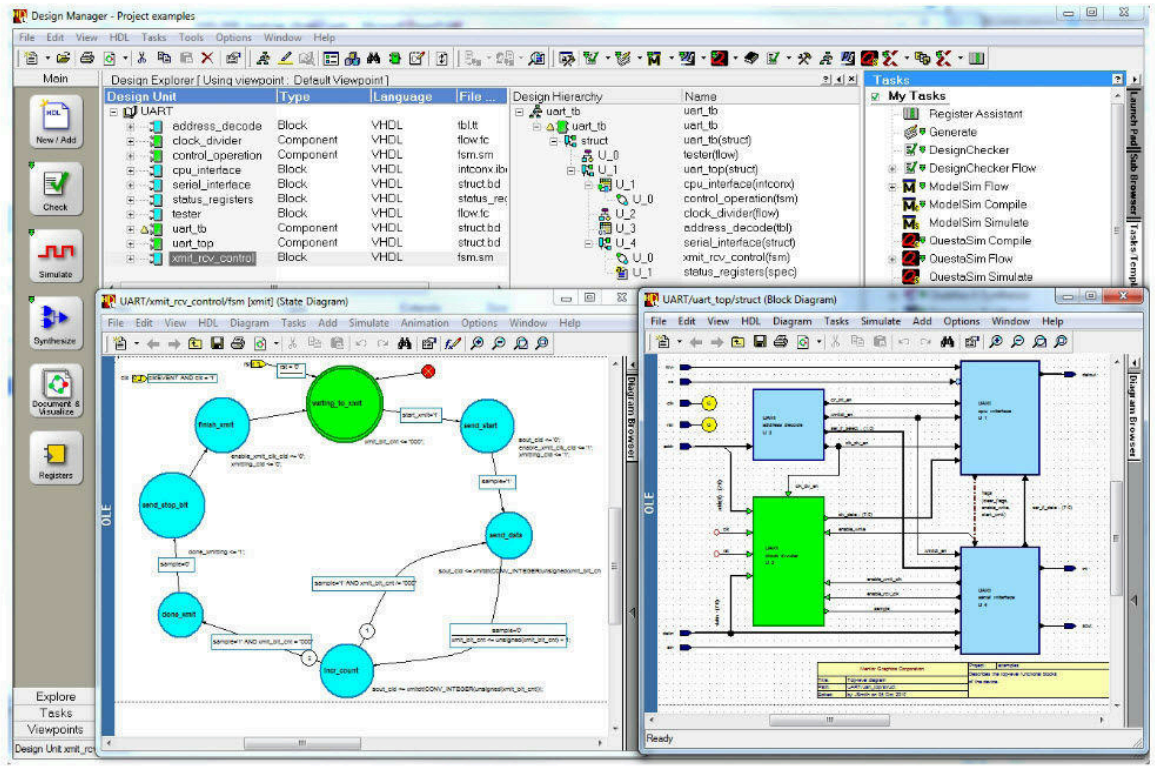
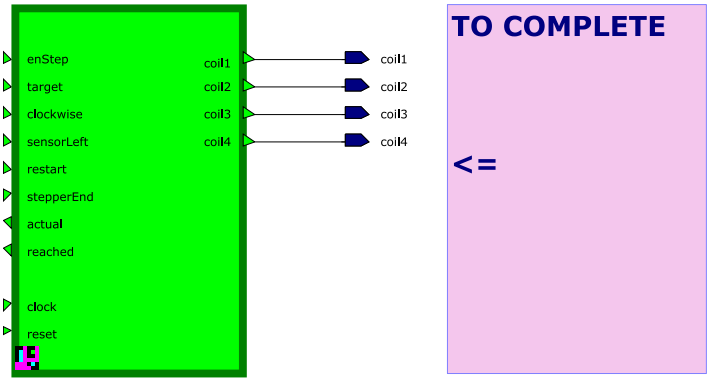


Figure 38: Mentor HDL Designer

Always run the **`kart.bat`** file to launch the project.

Parts you must complete are pointed by purple text blocks:



A cheatsheet is available online under https://github.com/hei-synd-ss1/ss1-docs/blob/main/control-electronics/EDA_Tools_Cheatsheet.pdf.

II Modelsim

Mentor ModelSim for simulation [5]

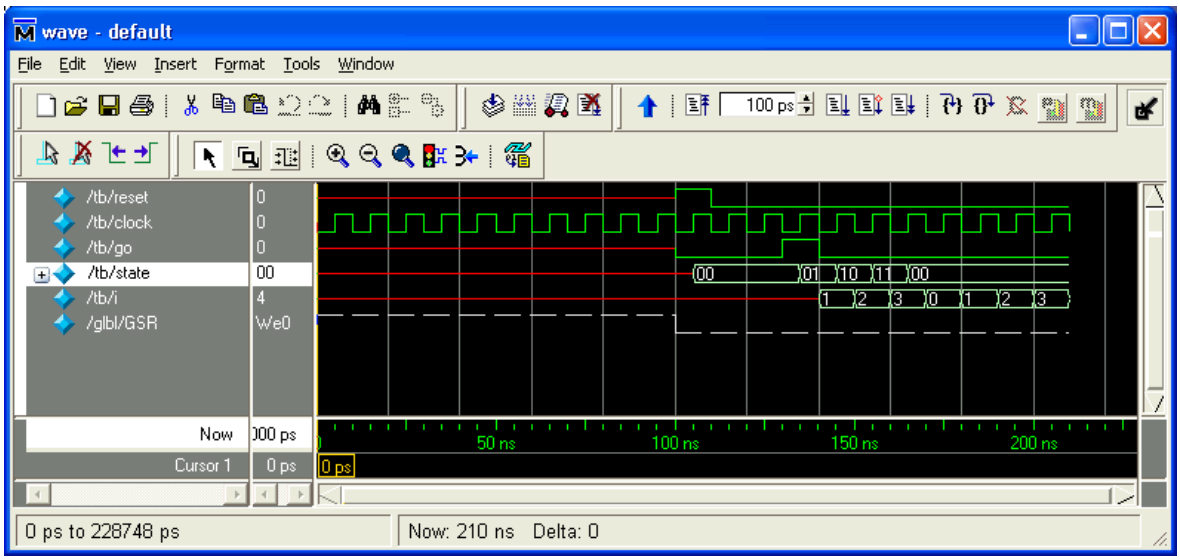


Figure 39: Mentor Modelsim

The simulations are explained under Section 6. They allow to test on the module level itself or the complete circuit in operation by simulating commands received from the smartphone.

The simulations files are available under the **`Simulation`** folder and contains among others the **`.do`** waveforms files related to all VHDL tester.

III Microchip Libero

Microchip Libero IDE for synthesis and programming[6]

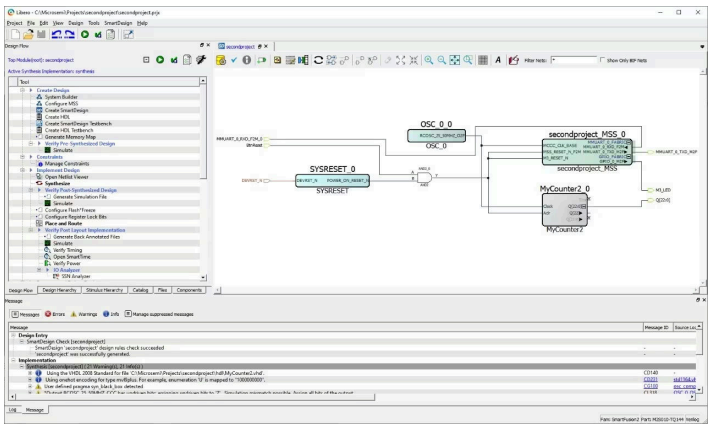


Figure 40: Microchip Libero

Libero SoC is a design software from Microchip (former Actel) for FPGA.

i Overview

1 Synthesis

Libero SoC can be launched as a standalone or from one of its ***.prjx** project file to complete the synthesis process.

One can launch Libero directly from the Kart project by running the correct task.

- On HDL Designer, open the **`Board`** library
- Highlight the top-level block **`Kart_Board`**
- On the tasks list on the right, first run **`Prepare for synthesis`** then **`Libero Project Navigator`**

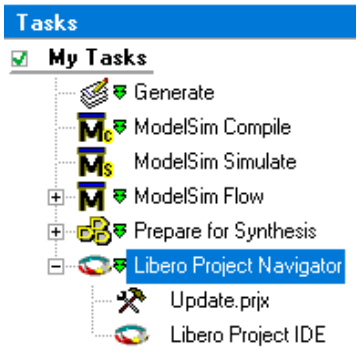


Figure 41: HDL Designer - Tasks

2 Flash

The FPGA flash is done through the FlashPro software included with Libero through the generated ***.pdb** bitfile thanks to a dedicated programmer such as the FlashPro4.

It can be launched as a standalone or directly from within Libero.

FlashPro can also be used to generate ***.svf** files which can then be used with OpenOCD to flash the FPGA by its USB port.

ii Synthesis

1 Prepare project

After running the HDL task, the project window opens. On the list located on the left, locate the **Compile ⇨ Constraints ⇨ yourConstraintsFile.pdc** ⇨ right-click ⇨ **Mark as used**:

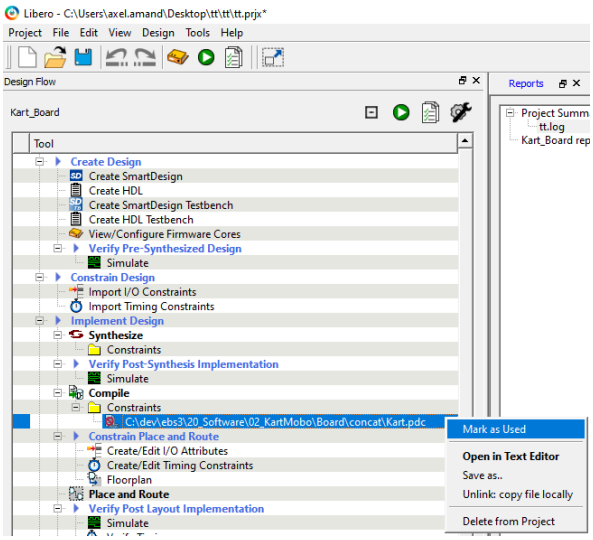


Figure 42: Libero - Use constraints

Right click on **Synthesize ⇨ Open Interactively**. In the newly opened window, on the left, set the correct clock frequency and exit while saving:

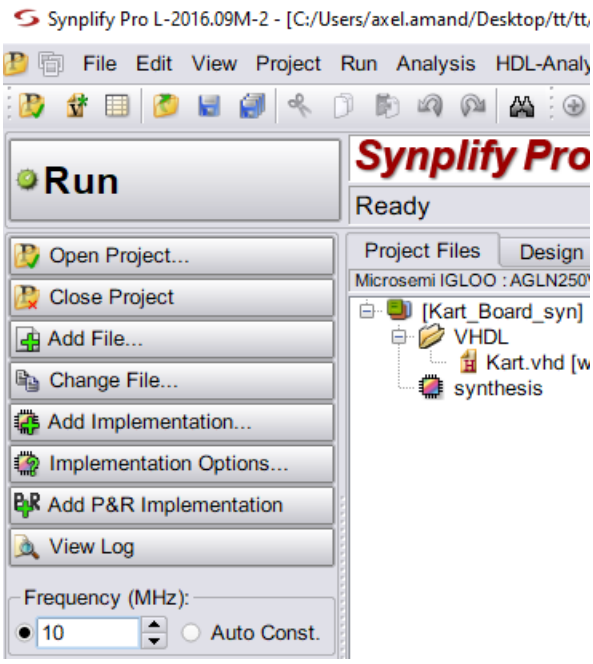


Figure 43: Libero - Clock setup

This step is required for the program to estimate if the implementation reaches the correct timings.

Right click on **Compile ⇨ Open Interactively ⇨ I/O attribute editor** and check that pins are correctly linked to the internal signals with correct settings:

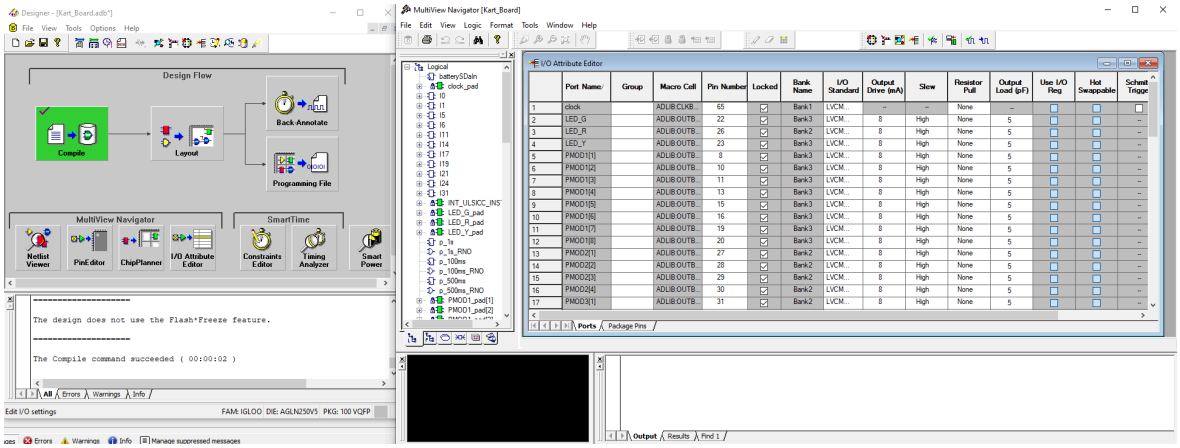


Figure 44: Libero - Constraints check

2 Synthesize

When the project is ready, the VHDL file must be synthesized, compiled and rooted on the chip.

Either click on **Layout** in the previously opened window (**Compile** ⇨ **Open Interactively** ⇨ **Layout** ⇨ **Ok**) or double click on **Place and Route** in the task list.

The **Message** window on the bottom of Libero can be used to check for errors and warning (both should always be checked). Some parts of the circuit may be pruned, clocks inferred unintentionally, unused signals found ...

In addition, reports can be browsed in the **Reports** tab, notably:

- **Synthesize - prjName.srr**:

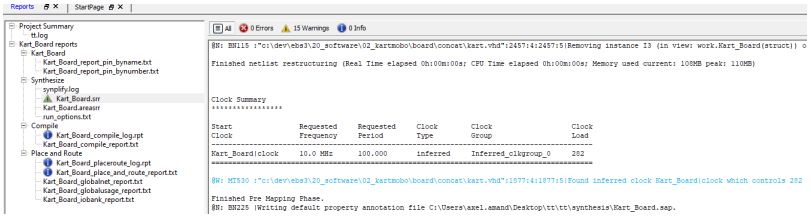


Figure 45: Libero - Logfile

  which contains informations on:

  - **Clock summary**: should only show the actual clocks - if some inferred ones are found, there is a design problem in the VHDL code
  - **Performance summary**: shows the worst slacks, if the timings are met and the potential fastest clock usable
  - **Core Cells** and **RAM/ROM** usages: how full the FPGA is
- **Compile - prjName_compile_log.rpt** shows the following:
  - **Compile Report**: more detailed view of used cells, BRAM block, I/Os ...
  - **I/O Technology**: ensure the standard is set to **LVCMOS33**
  - **I/O Placement**: ensure I/Os are all locked (Placed and UnPlaced ones may indicate errors or that the compilation used a pad to root a signal more easily because said pad was not locked even if not used ⇨ user must ensure the pad is not rooted to anything on the board or lock it beforehand)

- **Place and Route - prjName_globalnet_report.txt** shows global clocks and reset signals found under **Nets Sharing Loads**. In most cases, only a clock and a reset should be shown.

3 Bitfile

If the compilation is successfull, double click on **Generate Programming Data** to generate a ***.pdb** bitfile.

FlashPro can be launched directly with a right click on **Program Device** ⇨ **Open Interactively**.

4 Tips

Locked pins

All I/Os referenced in the VHDL file should be linked to a pad of the chip. If when trying to compile the following window appears, the ***.pdc** constraints file may be missing some I/Os:

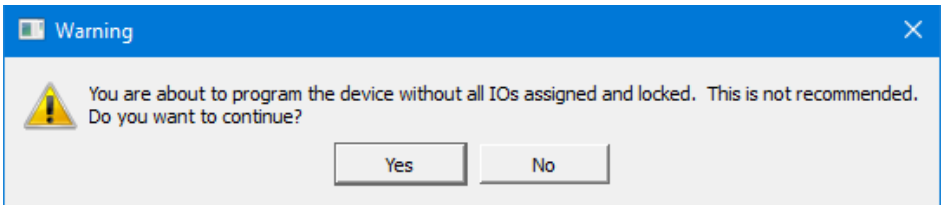


Figure 46: Libero - PDC warning

Click No and correct it.

Inferred clocks

Incorrect designs may lead the synthesizer to infer some clocks (e.g. may appear on state machines depending on a signal to trigger each state).

If the warning is ignored, it may replace the clock with said signal. Incorrect behavior may result (desynchronized state changed, no state changes at all ...). Here is an example of inferred clock:

```
Clock Summary
*****************

Start                                Requested     Requested     Clock        Clock                Clock
Clock                                Frequency     Period        Type         Group                Load
-----------------------------------------------------------------------------------------------------------
Kart_Board|clock                     10.0 MHz      100.000       inferred     Inferred_clkgroup_0  1328
coilControl|stepdelayed_inferred_clock  10.0 MHz   100.000       inferred     Inferred_clkgroup_1  4
===========================================================================================================
```

Figure 47: Libero - PDC warning

iii Flashing

1 FlashPro

Overview

FlashPro is the official tool supported for the Karts FPGA. *It is required to create compatible programming files for OpenOCD*

Usage Launch FlashPro directly or from within the Libero project with a right click on **Program Device ⇨ Open Interactively**.

Wire a compatible programmer (such as a FlashPro4) and click on **Refresh/Rescan for Programmers** which should show found devices:

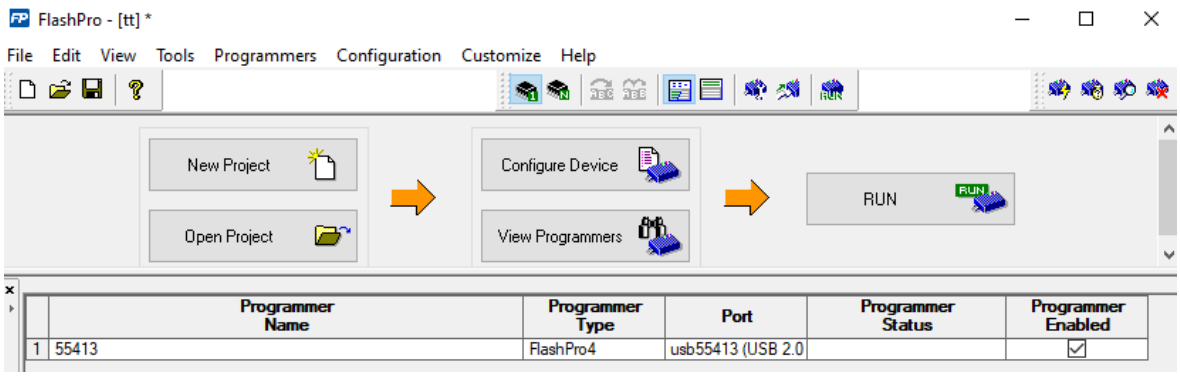


Figure 48: Flashpro - Programmers

Under **Configuration ⇨ Load Programming File** select the previsouly created ***.pdb** file.

Wire the board, connect the programmer, then click on **PROGRAM**. The advancement is shown in column **Programmer Status**.

The programmer does not supply the daughterboard with power. An external power source (motherboard or USB-C) is needed.

I/Os states normally remain in high-impedance state (with potential pull resistors) while the chip is being programmed. For sensitive applications, disconnect it from the motherboard beforehand.

2 OpenOCD

The board can be programmed without any third-parties hardware through OpenOCD thanks to the embedded FT2232HL chip on the daughterboard which offers both an UART and a JTAG interface.

This method is not currently supported by your teachers. You have no guarantee to receive any help from them.

If interested, refer to the **`doc/Kart_AGLN250.pdf`** - section **OpenOCD**.

To not run flash commands by hand, once OpenOCD is installed correctly with its extension files and added to the path, you can run the **EBS3 UART Interpreter** - Section 6.5 and click on **OpenOCD** in the toolbar. Select any of your **.svf** file for the tool to locate all required files. It will then launch the programming and logs are output in the textbox.

B PMod boards

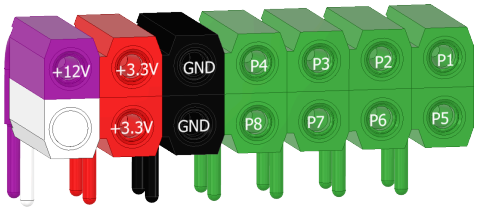


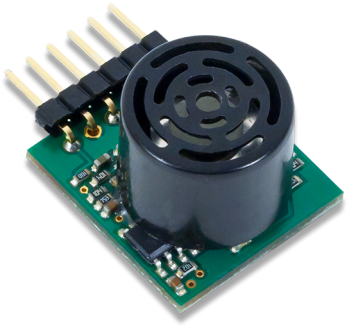
Figure 49: PMOD Pinning [22], [9]

I Inputs

The sink per pin cannot be higher than 8 mA.
Never ever input a voltage different than +3.3V.
Internal pull-up/down can be enabled at will on the FPGA.

i Ultrasound Ranger

An ultrasound ranger can detect if there is an obstacle at the front or back of the kart. It is based on the PMOD-MAXSONAR board from Digilent [23], and can be plugged into any one-row PMOD connector.



Pin	Descr.
1	*AN (Unused)*
2	*RX (Unused)*
3	*TX (Unused)*
4	PWM
5	GND
6	3.3V

Figure 50: PMOD MAXSONAR

Use th **`PWM`** pin with no internal pull resistor.
Beware not to wire it on the +12V pin !

See Section 5.3.3 for more informations on the generated pulse.

ii Buttons / Digital Inputs

The **PMOD-CON1** - *Wire terminal connectors* - [24] and the **PMOD-TPH** - *Pin headers* - [25] can both be used to interface digital inputs.

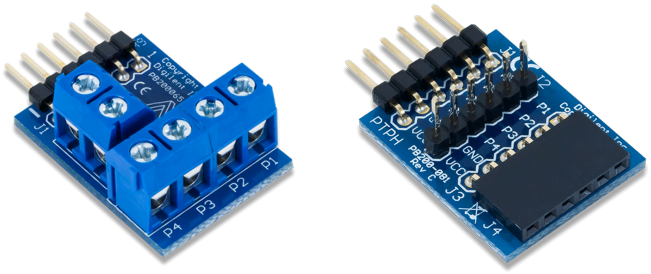


Table 9: PMOD-CON1 and PMOD-TPH boards

The first connects inputs thanks to screw terminals, the second with pin headers.

Enable pull resistors on the FPGA side based on your input type (push-pull, open-drain ...).

II Outputs

The source or sink per pin cannot be higher than 8 mA.
The outputs are always in the +3.3V range and there is no voltage feedback protection !

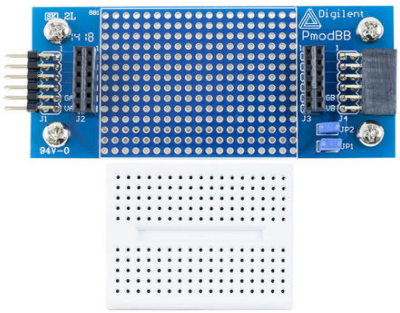
i Digital Signals

Direct drive from the FPGA is only possible for signals attacking high-resistance circuits like MOSFETs gates.
Such elements may be wired directly, or by using the PMOD-CON1 / PMOD-TPH - chapter ii boards.

You must at all time respect the previous warning.

ii Breadboard

The **PMOD-BB** board is intended for tests. It is a small breadboard which allows to plug components in and test a small circuit before designing a custom circuit.



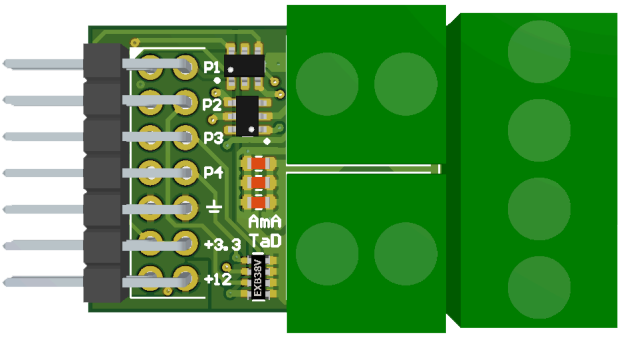
> ⚠ Use the given breadboard and do not solder on the holes directly.

Figure 51: PMOD Breadboard

iii PMOD-OD2 board

The **PMOD-OD2** ia a custom board allowing to control up to 4 outputs with a selectable voltage level thanks to open-drain outputs.

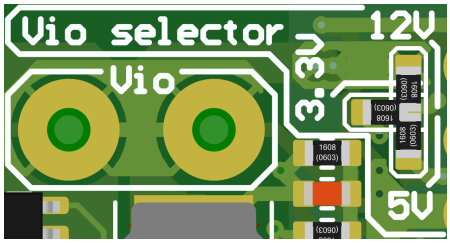
The double-rows PMOD connector is used to avoid confusion when plugging the board thanks to its keying pin. The lower row is unused.



Pins		Descr.	
1	2	X	P1
3	4	X	P2
5	6	X	P3
7	8	X	P4
9	10	X	GND
11	12	X	3.3V
13	14	X	12V

Table 10: PMOD-OD2 board and pining

Output Voltage
The voltage is selected by soldering one of the three following resistors on the backside of the board to switch between **+3.3V**, **+5V** or **+12V**:



> ⚠ Only one resistor must be soldered at any given time.

Table 11: Vio selection

Terminal
There are three screw terminals:

- A double terminal for Vio
- A double terminal for GND
- A quad terminal for the four outputs

They are all indicated on the back of the board.

Always use the terminals of the same board for **Vio**, **Px** and **GND**. Do not root either of those from another source to avoid destroying the protections in place.

The board uses negative logic.
When setting up your constraint file, use the signals **`{servos_inverted[x]}`** to use the PMOD-OD2 board.
For other usages with non-inverting logic boards, the signal **`{servos[x]}`** is available.

Wiring loads
Since outputs are open-drains, two wiring methods can be used:

- **Leds, relays, small DC motors ...**:
  - If needed (e.g. LEDs), put a resistor in series with the load.
    - For inductive loads, the circuit is already protected with flyback diodes.
  - Set the Vio jumper to the desired voltage
  - Wire the positive side of the load on the Vio terminal (J1) and the negative one to Px (J3)
  - Ensure no resistor from R2 to R5 is soldered.
  - When Cx is `'0'`, the output is left floating and there is no conduction.
  - When Cx is `'1'`, the transistor is driven and the output conducts.
- **Pseudo push-pull, servos control ...**:
  - Basically, it is only possible to either close the transistor (output a '0') or left it open (output a 'Z'). Some loads require a well-defined '0' or '1'.
  - Add a resistor either:
    - Between the Vio terminal (J1) and the Px terminal (J3) - *external resistor*
    - Solder one on the R2 to R5 pads - *soldered resistor*
  - Set the Vio jumper to the desired voltage.
  - When Cx is `'0'`, the output is `'1'`.
  - When Cx is `'1'`, the output is `'0'`.

DO NOT TRY TO DRIVE CURRENT THROUGH THE Px PIN.
All the current would flow through the resistor, creating a voltage drop and power loss, leading to a fire hazard.

Both methods are shown in Figure 52 :

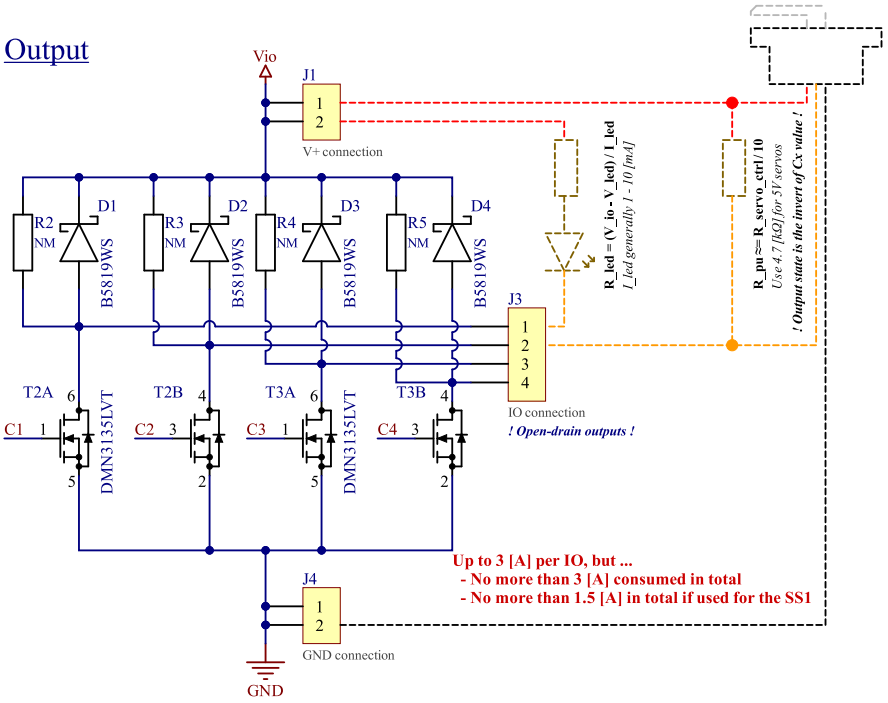


Figure 52: Driving loads with PMOD-OD2

C Inspiration

Figure 53: Summerschool 2004

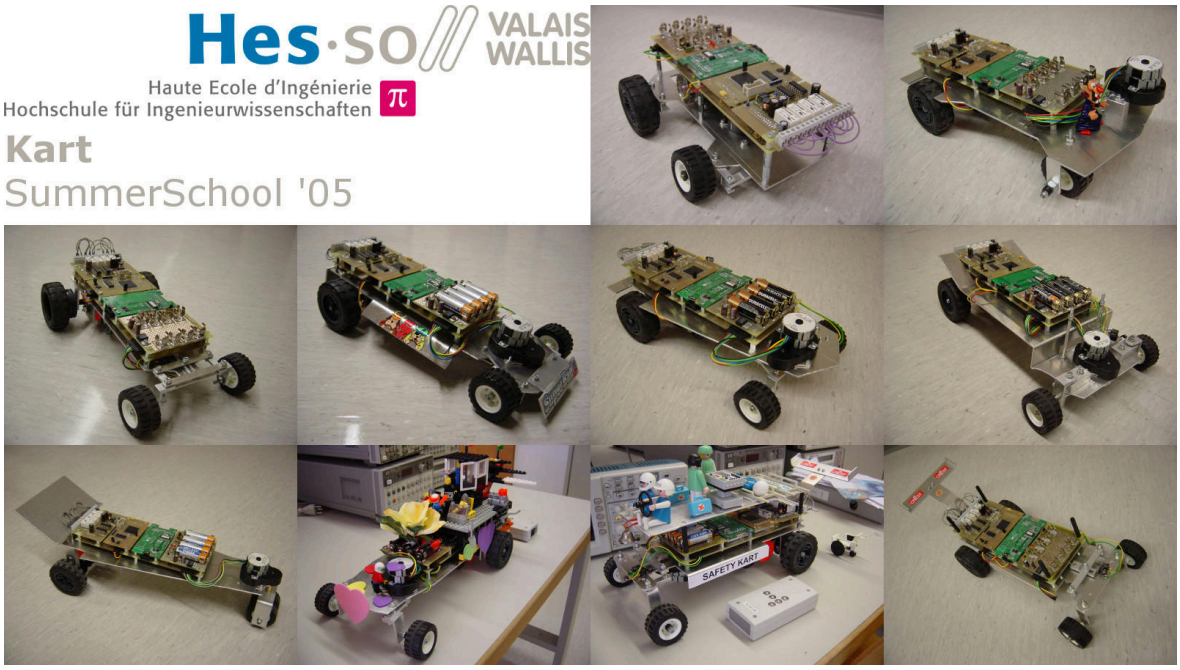


Figure 54: Summerschool 2005

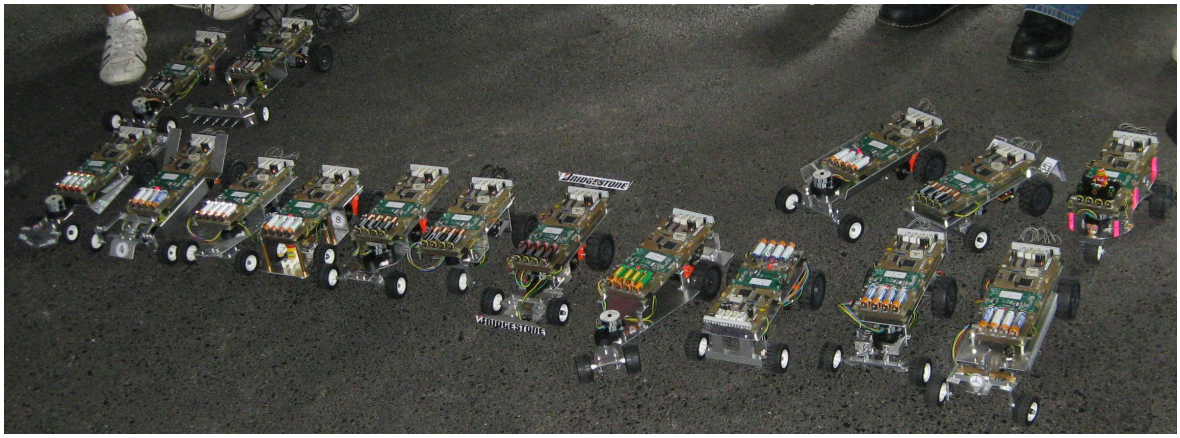


Figure 55: Summerschool 2009

Figure 56: Summerschool 2012

Figure 57: Summerschool 2013

Figure 58: Summerschool 2015

Figure 59: Summerschool 2017

Figure 60: Summerschool 2018

Figure 61: Summerschool 2020

Figure 62: Summerschool 2022

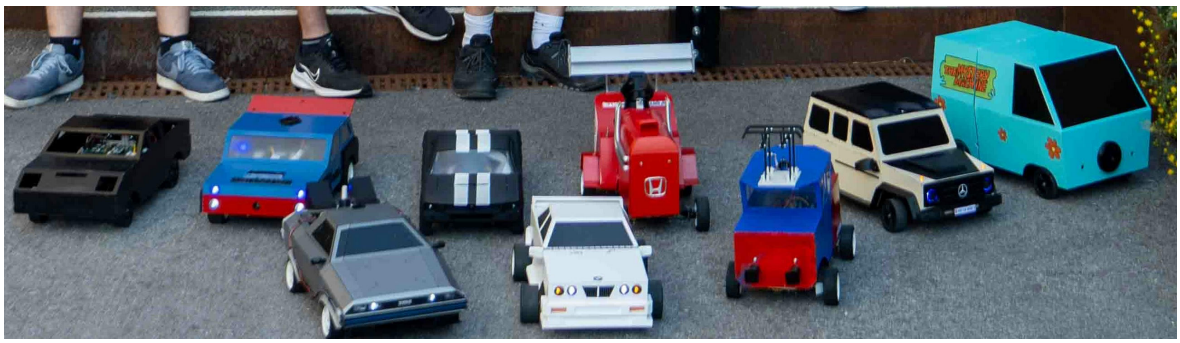


Figure 63: Summerschool 2023

Bibliography

[1] S. ye industrial, "DC Motor Modelcraft RB350018-2A723R 12V Technical Datasheet." 2006.

[2] A. Amand and S. Zahno, "FPGA-EBS3 Electornic Technical Documentation." 2022.

[3] "Hei-Synd-SS1/ss1-vhdl: HEVS SS1 Kart Summerschool Project Based on the EBS3 Igloo Board.." Accessed: Sep. 01, 2023. [Online]. Available: https://github.com/hei-synd-ss1/ss1-vhdl

[4] "HDL Designer Visualizing Complex RTL Designs." Accessed: Sep. 01, 2023. [Online]. Available: https://eda.sw.siemens.com/en-US/ic/hdl-designer/

[5] "ModelSim HDL Simulator." Accessed: Sep. 01, 2023. [Online]. Available: https://eda.sw.siemens.com/en-US/ic/modelsim/

[6] "Libero® IDE | Microchip Technology." Accessed: Sep. 01, 2023. [Online]. Available: https://www.microchip.com/en-us/products/fpgas-and-plds/fpga-and-soc-design-tools/fpga/libero-ide

[7] "IGLOO Nano Low Power Flash FPGAs with Flash*Freeze Technology Datasheet." 2019.

[8] M. Technologies, "Microchip MCP3426/7/8, 16-Bit, Multi-Channel ΔΣ Analog-to-Digital Converter with I2CTM Interface and On-Board Reference." 2009.

[9] D. Inc, "Digilent Pmod Inteface Specification v1.2.0." 2017.

[10] "nRF52840 Dongle." Accessed: Sep. 01, 2023. [Online]. Available: https://www.nordicsemi.com/Products/Development-hardware/nRF52840-Dongle

[11] F. INC, "FTDI FT2232H Dual High Speed USB to Multiporpose UART/FIFO IC." 2019.

[12] A. Amand and S. Zahno, "Kart SODIMM-200 Pinning." 2022.

[13] "Pulse-Width Modulation." Aug. 29, 2023. Accessed: Sep. 15, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Pulse-width_modulation&oldid=1172794179

[14] "H-Bridge." Aug. 07, 2023. Accessed: Sep. 15, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=H-bridge&oldid=1169162530

[15] N. Inc, "Nanotec SP3575M0906-A Stepper Motor Datasheet." 2006.

[16] D. Inc, "Digilent Pmod DC Stepper Schematic." 2022.

[17] Omron, "Datasheet Omron Subminiature Basic Switch Offers High Reliability and Security." 2005.

[18] Honeywell, "Honeywell SS311PT/SS411P Bipolar Hall-effect Digital Position Sensors with Build-in Pull-up Resistor." 2009.

[19] "Schmitt Trigger." Aug. 28, 2023. Accessed: Sep. 15, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Schmitt_trigger&oldid=1172685652

[20] N. Semiconductor, "Datasheet Nordic NRF52840 Dongle." 2020.

[21] "Universal Asynchronous Receiver-Transmitter." Aug. 28, 2023. Accessed: Sep. 01, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Universal_asynchronous_receiver-transmitter&oldid=1172648211

[22] “Pmod™ - Digilent Reference.” Accessed: Sep. 01, 2023. [Online]. Available: https://digilent.com/reference/pmod/start

[23] D. Inc, “Pmod MAXSONAR Reference Manual.” 2014.

[24] D. Inc, “Digilent Pmod CON3 Schematic.” 2015.

[25] D. Inc, “Pmod CON3 Reference Manual.” 2016.