electronics

MDPI

Article

Control System in Open-Source FPGA for a Self-Balancing Robot

Juan Ordóñez Cerezo [1,†], Encarnación Castillo Morales [2,†] and José María Cañas Plaza [1,*,†]

[1] RoboticsLab-URJC, Rey Juan Carlos University, Fuenlabrada, 28943 Madrid, Spain; jordonezcerezo@hotmail.com
[2] DiTEC Research LAB, Granada University, 18071 Granada, Spain; encas@ugr.es
* Correspondence: josemaria.plaza@urjc.es; Tel.: +34-914-888-755
† These authors contributed equally to this work.

Received: 30 December 2018; Accepted: 1 February 2019; Published: 9 February 2019

Abstract: Computing in technological applications is typically performed with software running on general-purpose microprocessors, such as the Computer Processing Unit (CPU), or specific ones, like the Graphical Processing Unit (GPU). Application-Specific Integrated Circuits (ASICs) are an interesting option when speed and reliability are required, but development costs are usually high. Field-Programmable Gate Arrays (FPGA) combine the flexibility of software with the high-speed operation of hardware, and can keep costs low. The dominant FPGA infrastructure is proprietary, but open tools have greatly improved and are a growing trend, from which robotics can benefit. This paper presents a robotics application that was fully developed using open FPGA tools. An inverted pendulum robot was designed, built, and programmed using open FPGA tools, such as IceStudio and the IceZum Alhambra board, which integrates the iCE40HX4K-TQ144 from Lattice. The perception from an inertial sensor is used in a PD control algorithm that commands two DC motors. All the modules were synthesized in an FPGA as a proof of concept. Its experimental validation shows good behavior and performance.

Keywords: robotics; open FPGAs; robot control

1. Introduction

The most common approach taken for the computing required in technological applications is using software which writes instructions for a general-purpose circuit, such as a Computer Processing Unit (CPU) or Graphical Processing Unit (GPU) [1]. Another option is designing a special circuit for this specific computation, where Application-Specific Integrated Circuits (ASICs) [2] are the traditional hardware implementation for system design. This last alternative requires more effort and has high development costs, but when an application requires real-time processing, such as a video, television, or robotic controller for real-time trajectory generation, the requirements are highly demanding and better met when implemented in hardware. However, in general, applications are more flexible when implemented in software rather than in hardware, especially when they are not computationally demanding or when they are non-critical. The emergence of Programmable Logic Devices (PLD) [3] and reconfigurable devices, such as Field-Programmable Gate Arrays (FPGAs) [4] have changed this scenario. The FPGAs are a well-established technology, not only for prototyping and development, but also as a for ASICs in a growing number of applications, as they offer benefits very similar to those of ASICs, such as high speed and reliability, but without requiring as much resources or costs as the custom ASIC design [2]. In addition, these features go far beyond those possible with microprocessor-based systems, while maintaining similar flexibility thanks to their reconfigurability. Unlike microprocessors, FPGAs perform different operations in parallel, and it is unnecessary to

compete for the same resources. Thus, incorporation of the FPGA in the industry has been driven by the fact that FPGAs are the combination of the best features of ASICs and microprocessor-based systems. Fields in which FPGAs are currently used include medical imaging, coding and encryption, aeronautics and defense, voice recognition, artificial vision, and robotics.

Thus, FPGAs are definitively established in the digital systems market, with Lattice Semiconductor Corp. [5], Xilinx [6], and Intel FPGA [7] as the main private companies. Xilinx, which recently signed a large collaboration agreement with IBM, and Intel FPGA, the new trade name of Altera after its acquisition by Intel, are market leaders. FPGA devices generally consist of a regular matrix of logic blocks and an interconnection network, both configurable, together with multiple I/Os. The high-end segment [8,9] has also integrated specific resources for digital signal processing, support for networks, or embedded microprocessors, usually being ARM cores. Thus, synthesized embedded microprocessors are included in FPGA devices, such as Nios II [10] or MicroBlaze [11]. More recently, RISC-V [12], a free and open RISC instruction-set type of architecture, has been implemented within Microsemi FPGA [13,14]. Moreover, FPGAs have a strong presence in the sector of artificial intelligence [15] providing hardware accelerators in this field that can exceed the performance of GPUs. Despite these advanced features, it is worth noting that these are proprietary FPGAs, and working with them requires a large budget that is not always feasible, such as in educational applications.

ISE from Xilinx and Quartus II from Intel FPGA are proprietary software tools offered by these companies for synthesis and analysis of designs to be implemented into its FPGAs, usually using a Hardware Description Language (HDL) [16]. These software tools enable the developer to synthesize or compile their designs, to examine RTL descriptions, to perform timing analysis, to simulate the designs, and to configure the target device using the programmer [17]. There are currently two industry standard HDLs: VHDL (very high-speed integrated-circuit Hardware Description Language) [18] and Verilog [19]. To compare these two, on the one hand, VHDL is strongly typed, it has the ability to define custom types, it can define multiple signals into one type, and the logical statement endings are clearly marked. However, it is also extremely verbose, and needs sensitivity lists and type conversions. On the other hand, Verilog is a compact language, performs logical tests on an entire array of bits with a single operator, and is adequate for low-level descriptions closer to the actual hardware. But nevertheless, it is a weakly typed language, it offers no support of custom types, the signal declarations can be confusing, and it has reduced support for asynchronous signals. This scenario has forced the search and development of new alternatives, such as SpinalHDL [20], an open-source high-level called whose goal is to use simple elements (flip-flops, gates, if/case statements) to create a new abstraction level and help the designers to reuse their code. Among their advantages over VHDL and Verilog are the evolving capabilities, the reduction of the code size, the easy type conversions, the loop detection, and that it is free and has a user-friendly IDE.

As mentioned previously, robotics is one of the application fields of FPGA [21–23]. Typically, the implementation of robot intelligence and controllers in FPGAs provides many advantages, like reliability and fast operation, which allow for better robot control. However, a large budget is required to work with proprietary FPGA software tools and it is not affordable for educational applications, such as educational robotics [24]. This paper presents a novel use of open-source FPGAs for educational robotics using a new visual language for robot programming. Concretely, an inverted pendulum real robot was developed. The main characteristics of this implementation and the performed experiments confirm the feasibility of this proposal.

The rest of the manuscript is organized as follows: Section 2 is devoted to the related works involved in open FPGA in robotics and the inverted pendulum, while Section 3 describes the design of the self-balancing robot using open FPGA. Section 4 presents the experiments, where a real implementation of the self-balancing robot over an open FPGA is carried out, confirming the feasibility of the presented proposal. Finally, the main conclusions are presented in Section 5.

2. Related Works

Three areas provide the context for the proposal in this study: the use of FPGAs in robotics, the open FPGA community, and the robotic application selected as a proof of concept—the self-balancing robot. Some key works are also reviewed in this section.

2.1. FPGA in Robotics

Robotics is one of the fastest-growing technological areas in recent years [25]. It is based on systems composed of mechanisms which are able to make movements and execute specific tasks that are programmable and intelligent. Some implementation solutions for digital control systems for robot manipulators and mobile robots proposed in the literature use hardware technologies, such as DSPs or microprocessors [26]. These solutions allow for real-time control, but since the DSP has limited output ports, applications for control of humanoid robots, for instance, are not suitable. FPGA technology avoids this limitation, ultimately reducing size and weight, and therefore, costs. In addition, due to the efficient integration of embedded processors' intellectual properties (IPs) into a FPGA, the highly sophisticated algorithms with heavy computations required by robotic controllers can be performed by software in an FPGA. Thus, many FPGA-based solutions have been implemented in the field of robotics, such as a static gesture recognition system [21], an algorithm for collision detection between Oriented-Bounding-Boxes (OBBs) [22], and an embedded, robust adaptive controller for mobile robotics [23]. Many different works have shown that FPGA implementation of robotic applications is the best solution for optimum performance. Robotics may generate benefits not only in the industry, but also in classrooms [24], enabling the emergence of new learning systems. In addition, in a future world where robots will be used in almost any activity, a learning approach using these systems in the classroom enables students' technological development at an early age, facilitating their integration into the adult world. The following are some of the educational benefits of robotics: they drive initiative and creativity; promote greater sociability; encourage algorithmic and mathematical thinking; facilitate teamwork, problem solving, and active learning; and enhance self-esteem. However, in order to facilitate educational robotics in the classroom, the systems must consider the following elements:

- A high technological integration level is not recommended;
- Robots must be sociable and fun;
- Programming frameworks should not be complex, their functionality should be limited to a certain extent, and they must attract students' attention and make them feel comfortable in the context;
- It is important for the robot to have a series of sensors and actuators, as well as inputs and outputs so that the results are visual.

A major obstacle to achieving these features in educational robotics is that most commercial educational robot platforms are closed. Thus, robot vendors do not commonly provide support for old control units, or when a deprecated robot requires an update or even simple preventive maintenance. The manufacturer tends to recommend disposing of such a unit and acquiring a new one. It is worth mentioning the well-known LEGO, PID control [27], and walking robot [28]. Open FPGAs and their new graphical IDE tools may help to avoid this obstacle, as described in the following section.

2.2. Open FPGAs

Many HDLs, as well as FPGA architectures are linked to important companies, such as Xilinx and Intel FPGA, and working with them entails high development costs. Hence, not many companies or individuals can benefit from the advantages of using FPGAs, meaning FPGA technology progresses at a slower pace. One of the keys to the success of companies like Arduino [29] is the large community of people that stands behind creating new libraries, components, etc. This is mainly due to the low price of its products and the possibility of finding all the hardware and software on the web. To understand

the creation of open FPGA [30], it is important to understand the bitstream that is used to describe the configuration with which a specific design will be implemented in a FPGA. This detailed bitstream format for a particular FPGA is typically owned by the FPGA manufacturer. This is why Cliford Wolf decided to interpret the bitstream of the Lattice iCE40 FPGA devices [5] and developed the IceStorm tool [31], a translate software from Verilog and bitstream. This translation was possible thanks to the inverse engineering, meaning that it is not the usual usage that is given, but the inverse. Thus, there is no longer dependency on any manufacturer, and all knowledge is also available. From these tools, new interfaces or new applications that were not foreseen by the manufacturer can be created. Nowadays, the focus of the IceStorm project is on HX1K-TQ144 and HX8K-CT256 devices from Lattice iCE40, but since it is an open project, a lot of people are increasing their chances. IceZum Alhambra [32] is an FPGA development board including the open FPGA iCE40HX4K-TQ144 from Lattice. It is an open hardware that is compatible with the IceStorm toolchain and with Arduino Uno shields. Figure 1 shows the Icezum Alhambra II Board. Important features of this board include the following:

- 12 MHz oscillator;
- On/off switch to enable or disable digital pins;
- 20 Input/Output 5 V pins;
- 8 Input/Output 3.3 V pins;
- Micro-B USB to program FPGA from PC;
- Reset button;
- Eight general-purpose LEDs;
- TX/RX LEDs;
- 4 analog inputs available through i2c;
- 8 K memory;
- Possibility of powering through LIPO battery.

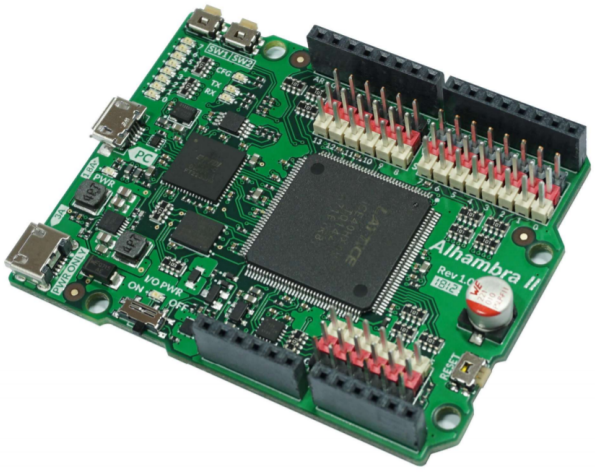


Figure 1. Icezum Alhambra II Board.

This development board can be implemented with new open tools, like IceStudio [33], a graphical IDE for free FPGAs, built on the IceStorm project. It provides simple tools to analyze and create bitstream files—that is, the lowest level of implementation for an FPGA. Boards with better features do exist, but IceZum II Alhambra provides open hardware that can be implemented with free and open software tools. This board was created with the idea of making digital electronics user-friendly for young students, allowing for a visual language for programming the FPGA [34], fulfilling the aforementioned features required for educational robotics.

2.3. Inverted Pendulum

The inverted pendulum is of one the most famous problems in terms of control theory and systems dynamics [35,36]. An inverted pendulum is represented in Figure 2, and consists of a pendulum where the center of mass is located above the balancing axis. Maintaining an upward equilibrium position is a challenge, as this equilibrium position is unstable (a system is more stable when its center of mass is closer to the supporting horizontal plane). As the inverted pendulum system is non-linear, it is well-suited for control by fuzzy logic [37]. The inverted pendulum system has significant theoretical value since it represents the basis of many complex systems, such as biped robot upright walking balance control, rocket launch vertical control, spacecraft attitude control, and offshore drilling platform stability control. Beyond its theoretical interest, the inverted pendulum is also attractive for university professors of engineering and teachers in secondary education. In this paper, a solution for the inverted pendulum problem is addressed using an open FPGA in order to correct its instability.

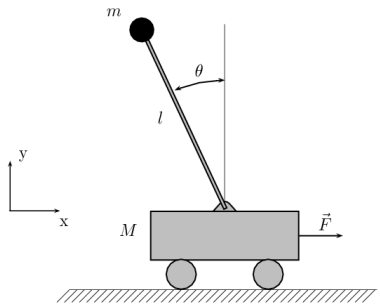


Figure 2. Representation of an inverted pendulum.

3. Self-Balancing Robot

In this section, the proposed solution for the inverted pendulum problem through the use of an FPGA in coexistence with a micro-controller is described. Several aspects are addressed, such as the physics of the self-balancing robot used in the experiments, the calculation of its structure, the sensors and actuators used, the control system, and the design and manufacture of a Printed Circuit Board (PCB) to solve certain engineering problems.

This section begins with a brief high-level description (Section 3.1) and continues with the details of the perception element (Section 3.2). Subsequently, the connection between Arduino and FPGA (Section 3.3), the PD control on FPGA (Section 3.4), and a motor block in IceStudio (Section 3.5) are also described.

3.1. Design

The hardware design of the inverted pendulum control with FPGA is shown in Figure 3. The microcontroller obtains the current angle of the system by means of i2c communication with an Inertial Measurement Unit (IMU) sensor. In the microcontroller, once the current robot vertical angle is read, a serial-type communication sends it to the FPGA in a binary format of 1 byte for the integral part and 1 byte for the decimal part. Inside the FPGA, the robot angle is read and the speed commands to the motors for correction of the angle are calculated by a basic PD controller. A shield with a DC motor driver is connected to the FPGA, and provides the possibility of varying the speed and the movement direction of two DC motors that permit the stabilization of the system.

3.2. Perception

Continuous knowledge of the angle is necessary for its analysis and correction. For this purpose, the MPU6050 sensor was used, and connected to an Arduino Nano by i2c communication. In order to correct some of the data collection problems such as noise or drift, it incorporates an internal processor

(Digital Motion Processor, DMP) that executes data fusion algorithms (Motion Fusion) to combine the measurements of the internal sensors, avoiding the necessity of performing the filters externally (Figure 4).

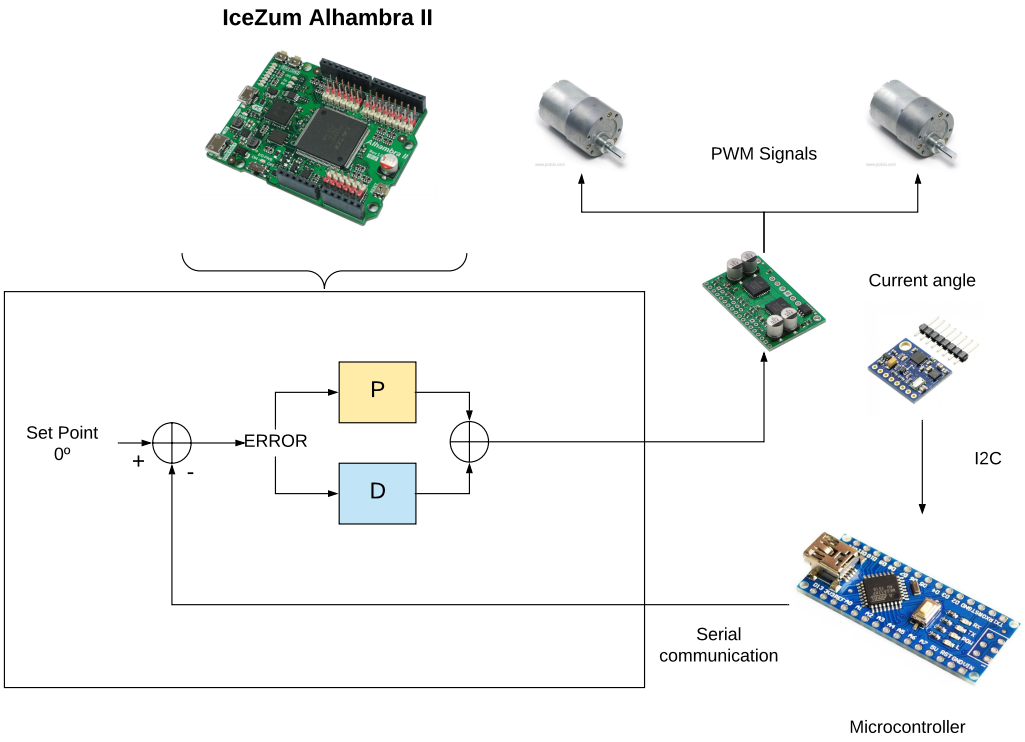


Figure 3. Hardware design of the inverted pendulum control with Field-Programmable Gate Arrays (FPGA).

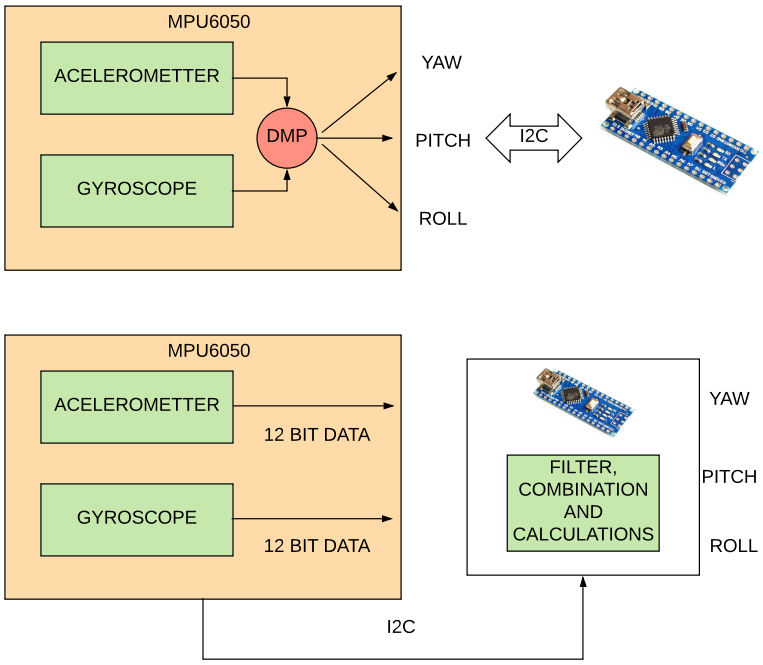


Figure 4. Advantage in the use of DMP.

3.3. Arduino-FPGA Connection

An integration between the microcontroller and the FPGA allows sequential and parallel tasks to be distinguished, assigning each process to the microcontroller if it the task needs to be sequential, or to the FPGA if the process can be parallelized, thus obtaining certain advantages. More than one option exists for the microcontroller/FPGA integration. In this work, physical coexistence with communication between them was chosen. They can also be integrated by means of several types of communication. Serial communication was selected as the most appropriate type due to the numbers of pins available in the FPGA. The communication would only be unidirectional, with the microcontroller sending information to the FPGA about the current angle of the robot in order for the FPGA to analyze and actuate starting from that angle. There are thus two parts in this serial communication: from the point of view of the microcontroller, and from the point of view of the FPGA. The sensor reading acquisition in the microcontroller is described in Section 3.2, while only the communication with the FPGA is analyzed in this section. The diagram flow on which the C-code of the microcontroller is based is shown in Figure 5.

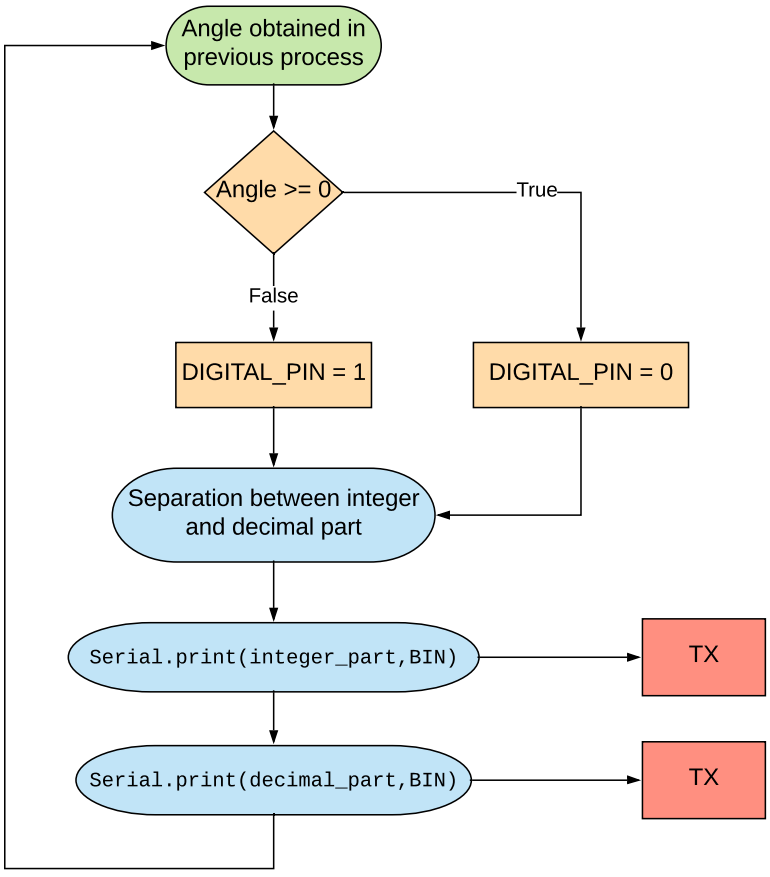


Figure 5. Diagram flow to send angle.

For correct and easy understanding by the FPGA, it is necessary to send the represented angle as several bytes in binary format, not as ASCII codes. The "Serial.println'" Arduino function was discarded, because it used ASCII codes and would even send the comma character to separate the integral and the decimal part. Instead, the Arduino function "Serial.print(Angle)" was used, which sends a binary number through the serial port. The representation and sending of the angle reading was separated into two bytes, as shown in Figure 5. The first byte is the integer part (from 0 to 255) and the second byte is the decimal part (from 0 to 100). No comma is sent over the wire. In FPGA, an

input pin will continuously enter data from the transmission pin so that it can make a correct reading of the byte. It is necessary to know:

- When a byte transmission starts;
- When a byte transmission ends;
- When a bit can be captured;
- When the necessary bits are saved in a buffer until the byte is complete.

In order to solve the previous problems and features, an intermediate module in IceStudio (Figure 6) was implemented.

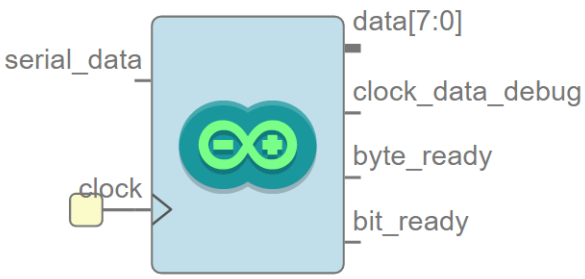


Figure 6. Appearance of Arduino Nano module in IceStudio.

This was implemented in Verilog by two machine states, with their corresponding sensitivity lists and the diagram flow represented in Figure 7.

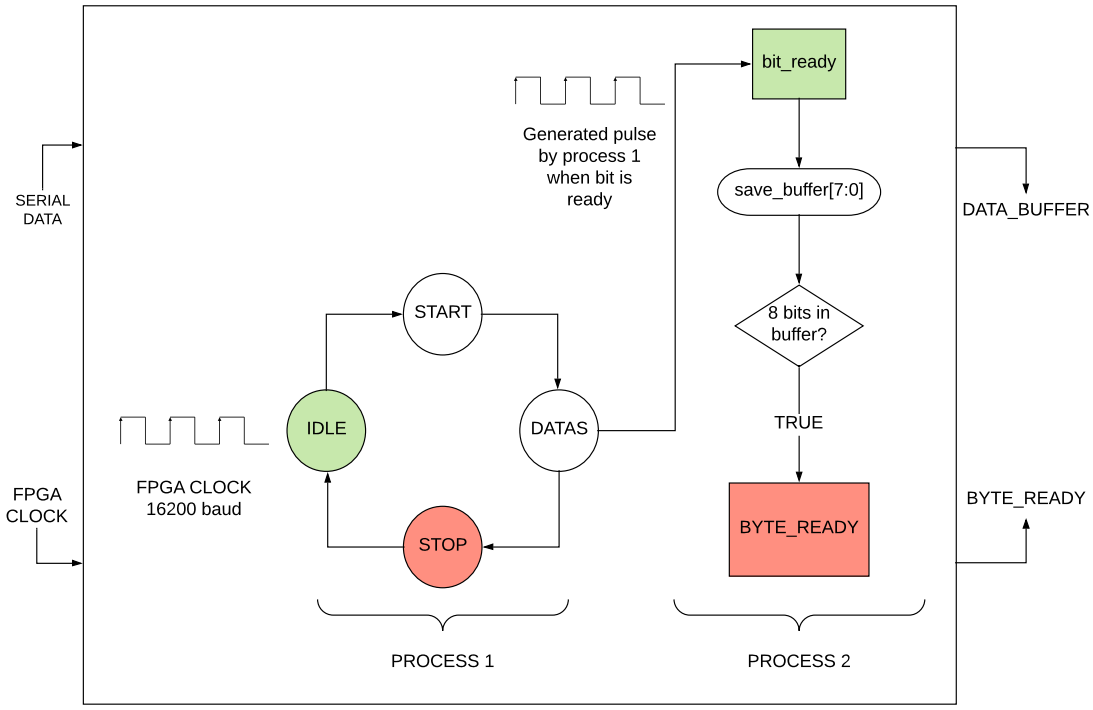


Figure 7. Diagram flow for Arduino interface.

Two distinct processes were used:

Process 1: This process only provides the next system state at the moment at which it can capture a bit and save it in the buffer. Thus, it is necessary to know the speed of the transmission. The states are the following:

- IDLE: The process remains in this state until the transmission starts, which will then lead to the next state (START);
- START: The serial transmission protocol begins with a start condition, and this state will allow recognition of when this condition ends in order to start saving bits in the buffer;
- DATA: Since the transmission speed is already known and the condition of START in the previous state has been recognized, in this state a flag will change its value when the bit is ready to be stored in the buffer, of which Process 2 will be in charge of this storage;
- STOP: In addition to a START condition, the serial transmission protocol used in Arduino has a STOP condition. This state allows recognition of the time Arduino takes to carry out this last condition—it will then return to the first state until a new transaction begins.

Process 2: This process is activated by Process 1. When Process 1 determines that a bit is available on the bus to be captured, it will set a clock flag on, initiating Process 1 through a sensitivity list. An example flow diagram could be:

- Wait until the sensibility list is activated—this will indicate that a bit can be captured;
- Bits will be stored in a buffer forming a byte, which will represent the integer or decimal part of the angle at that moment;
- When the byte is prepared to be captured by two consecutive modules, a channel will be on. Both the outputs and the buffer with the 8 bits and the "byte_ready" channel will be available.

At this point, the FPGA is able to differentiate between when it can capture a byte (BYTE_READY) and from where it has to capture the data bus (DATA_BUFFER). However, an aspect that is not part of the communication itself is that it is important to analyze whether a correct operation is required—that is, if the microcontroller has been previously told to continuously send the integer and decimal part of the angle. If this data is not correctly interpreted, it is possible that an angle on the FPGA is formed by a decimal part of an angle n, and the integer part of the angle n + 1. To do this, a module is created in IceStudio that is capable of ordering these values. The appearance of this module in IceStudio is shown in Figure 8.

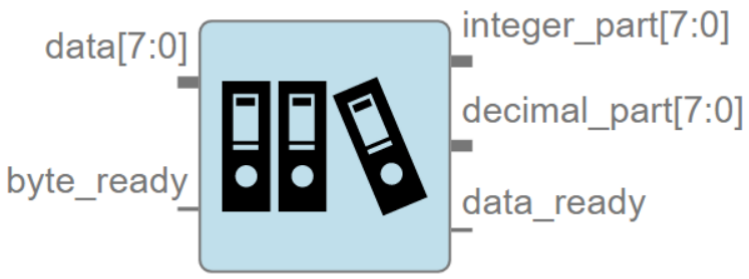


Figure 8. Module to arrange data from Arduino.

The final communication system between Arduino and IceZum Alhambra from a POV of the FPGA is represented in Figure 9.

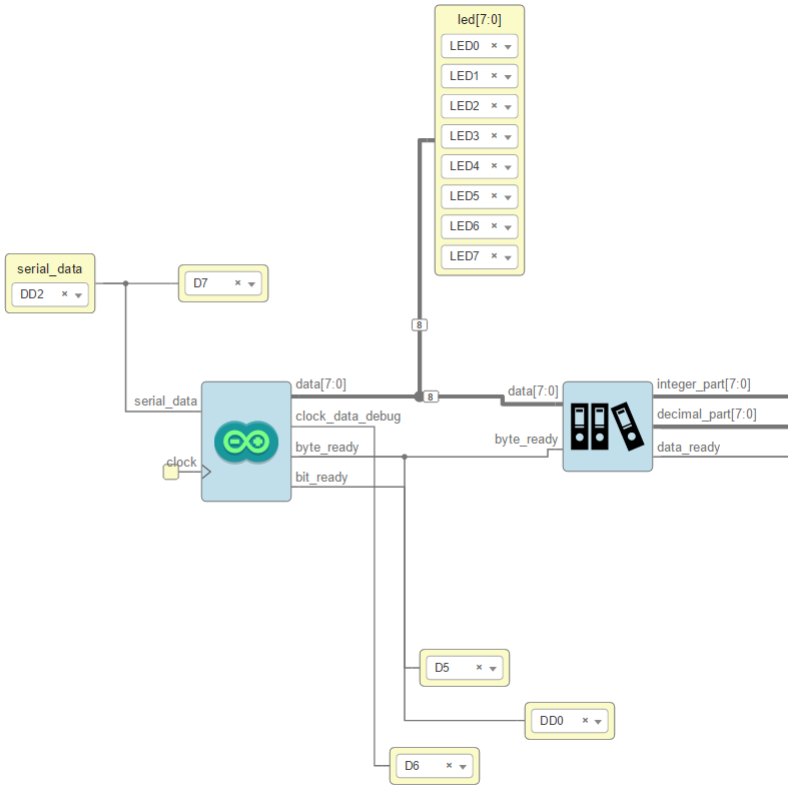


Figure 9. Communication between Arduino and IceZum Alhambra.

3.4. PD Control in FPGA

A PID controller can simply be used to control the stability of the system. One of the facilities provided by this type of controller is the ease of implementation. The flow diagram of the P controller's behavior is shown in Figure 10.

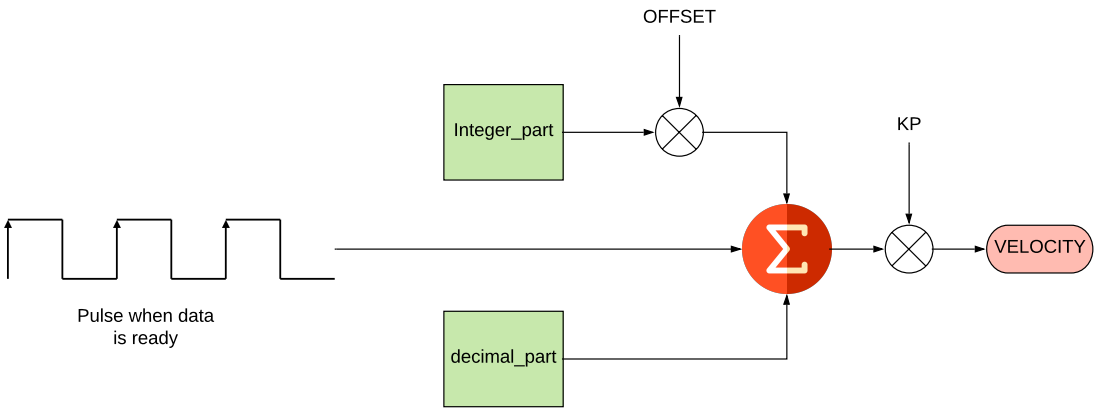


Figure 10. Flow diagram of P control.

The most important features are briefly explained as follows:

- Both the integer part and the decimal part are represented as 8-bit data without a sign. In order to give greater importance to the integer part, there is the option of dividing the decimal part by

100 (Figure 10) or of multiplying the integer part by 100. The first option does not provide good behavior due to the digital treatment of the floating comma. Thus, the second option is preferable.

- The two integer and decimal components are added, and then it is multiplied by a K_p constant, defined as a parameter which can be dynamically changed.

Referring to the D controller, the flow diagram implemented in Verilog is shown in Figure 11.

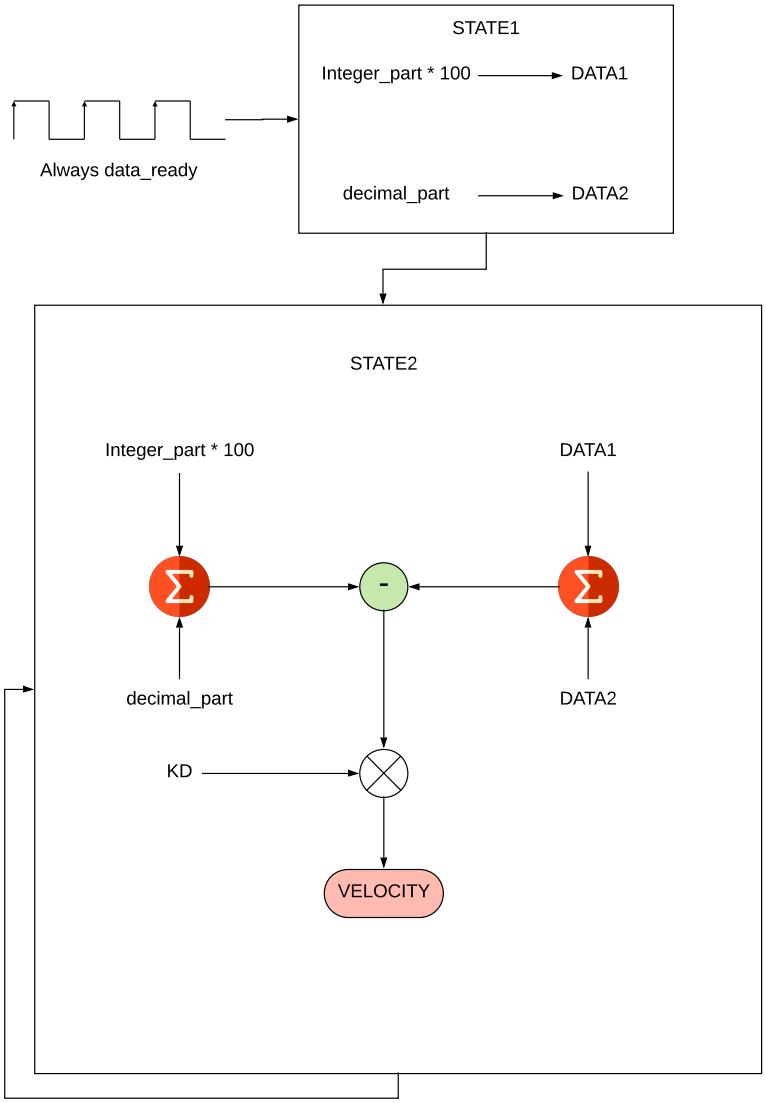


Figure 11. Flow diagram of D control.

Its implementation is composed of a state machine with two states, which will change at each pulse on the *data_ready*. This means that it will change whenever a new angle is available. The D controller is based on its operation on the prediction of future errors. The derivative control action generates a control signal proportional to the derivative of the error signal. A subtraction (derived from the error in time) is therefore carried out between the current error and the last error. Its result is multiplied by the constant K_d. Referring to the closed-loop feedback system, Figure 12 shows the final appearance of the present work developed using IceStudio.

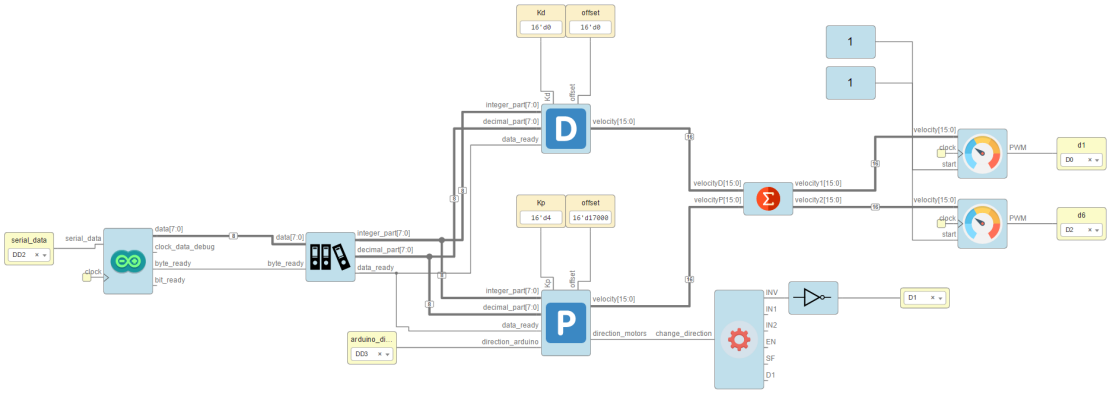


Figure 12. Final appearance of the self-balancing in IceStudio.

3.5. Motor Block in FPGA

In order to correct the current angle and obtain the stabilization, two DC motors were used. The speed of the motors was controlled by a PWM connected to the driver motor through the FPGA. Therefore, a PWM module generator, whose appearance is shown in Figure 13, is needed.

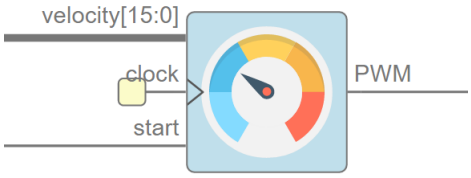


Figure 13. Appearance of PWM module in IceStudio.

Figure 14 shows the block diagram representing its behavior.

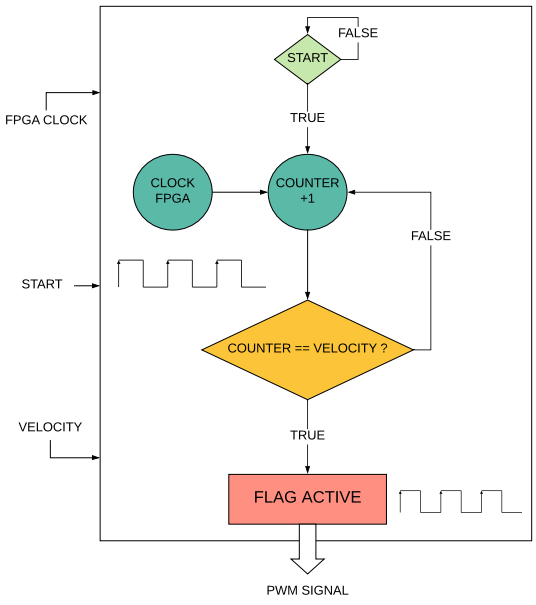


Figure 14. Flow diagram of PWM generator in Verilog.

4. Experiments

In this section, the self-balancing robot will be addressed purely from a hardware perspective describing the chosen physical model and all its components.

4.1. Physical Robot

4.1.1. IceZum Alhambra Board

The Alhambra board (Figure 1) was used as the main board and open FPGA (Section 2.2) to implement all the necessary systems that can be parallelized. For this purpose, the PD control, the calculation of the speed, and the motor control were implemented on this FPGA.

4.1.2. Arduino Nano-Processor

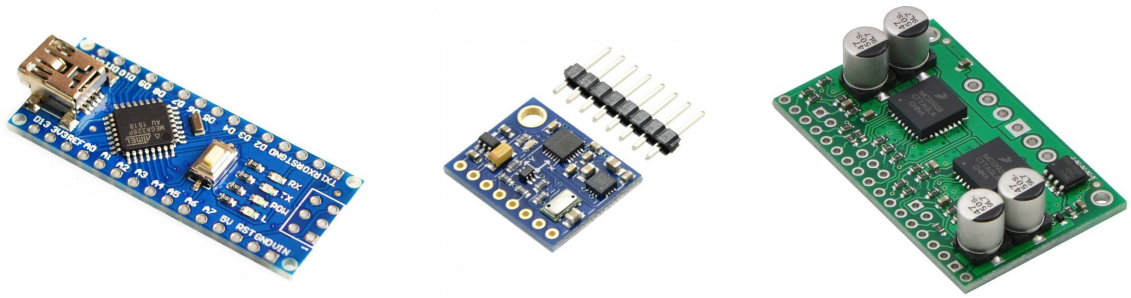
In order to allow a simple implementation of i2c communication with MPU6050 (implementation with FPGA was tested and is described in Section 4.4) and to avoid complex calculations in the FPGA, an ATMEGA microcontroller (Figure 15a) was used. Arduino Nano was chosen to develop the above features.

4.1.3. MPU6050

The MPU6050 (Figure 15b) is an Inertial Measure Unit (IMU) with six degrees of freedom (6DOF) manufactured by Invensense. It has an accelerometer and gyroscope, and allows communication by both SPI and i2c bus. To correct some of the data collection problems, it incorporates an internal processor (Digital Motion Processor, DMP) that executes data fusion algorithms (Motion Fusion) to combine the measurements of the internal sensors, avoiding having to perform the filters externally.

4.1.4. Motor Driver

For the DC motor control, which allows for robot stabilization, the MC33926 was used. It allows control of the speed and direction from up to two motors using a PWM signal which is generated by the module described in Section 3.5. The Figure 15c represents the motor driver.



(a) Arduino Nano board (b) MPU6050 IMU (c) MC33926

Figure 15. Physical components of the Self-Balancing Robot.

4.1.5. PCB Shield

After implementing the entire system and considering the necessary connection diagram between the microcontroller and FPGA and the motor driver, a printed circuit is advisable to solve some noise problems, the excessive number of cables, etc. A printed circuit board was developed using Altium Designer [38] as the design tool.

Figure 16 shows a 3D representation of the final system with all its added components.

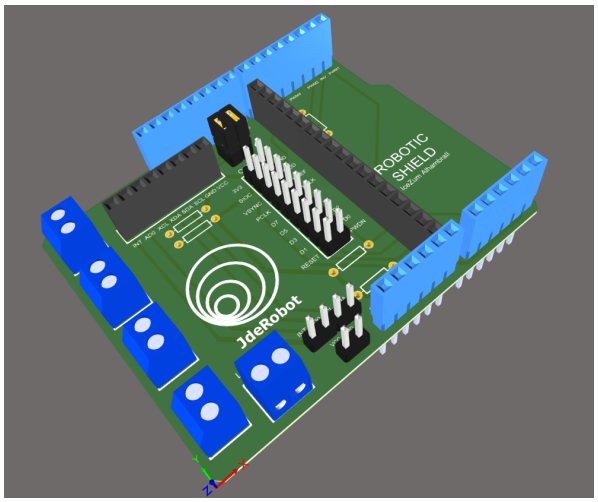


Figure 16. 3D representation of shield for IceZum Alhambra II.

4.2. Inverted Pendulum

Knowing the physics of a self-balancing robot [39] and aiming to solve the classic problem of the inverted pendulum, the mechanical structure of the Figure 17, designed with SolidWorks [39], is proposed to integrate and assemble the rest of the components.

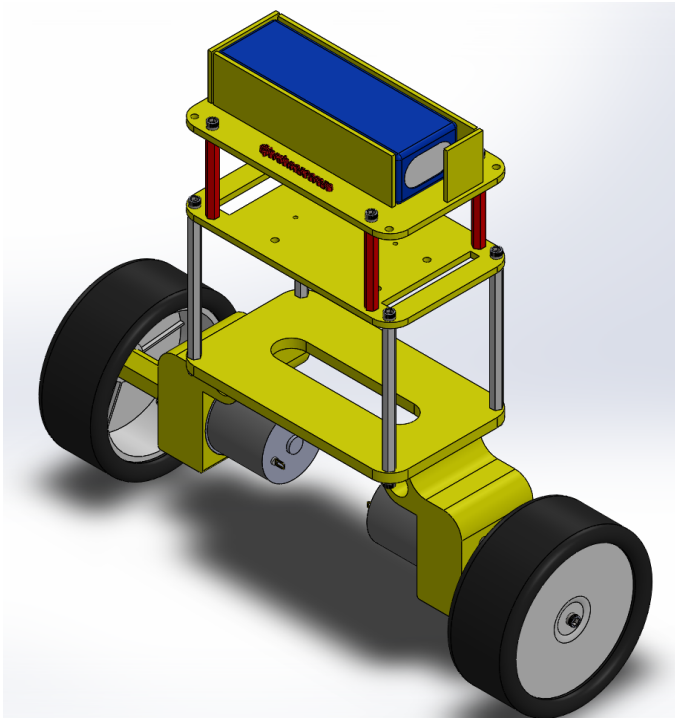


Figure 17. Balancing Robot perspective.

Different aspects of the design of this structure are considered, which are directly related to the physics of a self-balancing robot, and with it, of the inverted pendulum. As mentioned in Section 2.3, a system at rest is stable when its center of mass is closer to the horizontal plane. If we consider that the nature of the proposed system is inherently unstable, it is necessary to know the best point to situate the center of mass in order to provide better stability. According to the mathematical modeling characterization, it is assumed that, in order to achieve greater ease in stabilization, the center of mass should be placed above the midpoint of the vertical axis of our system. Therefore, in order to achieve

this positioning, we must consider the weight of all components. In Figure 18, a SolidWorks calculation is represented from this center of mass where only the heavier components of the final system are considered, including DC motors, batteries, mechanical structures, and wheels.

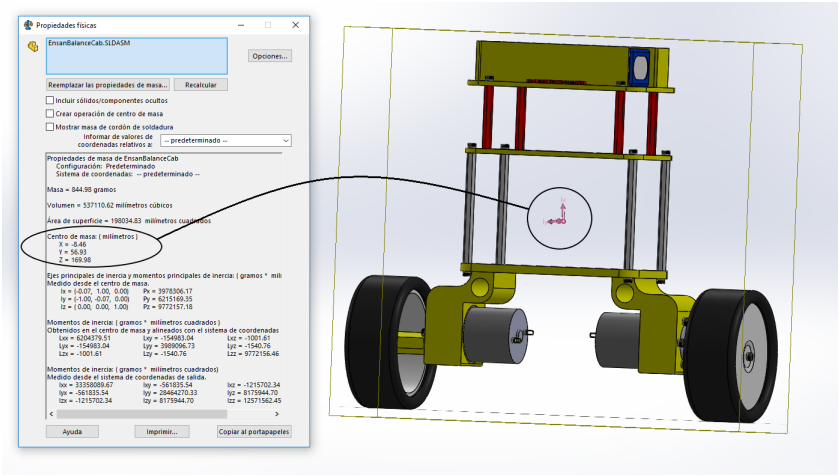


Figure 18. Final system of center of mass.

4.3. Final Results

A set of videos demonstrating the correct behavior in the Self-Balancing robot can be seen at (https://www.youtube.com/watch?v=u-KACjWmcKw). Also, the process through to the end can be found in (https://youtu.be/dQg8NQP7CfQ, https://youtu.be/d_1bnjbpQks, https://youtu.be/mLyxewOVGug). In order to manufacture the mechanical structure, a 3D printer was used (https://youtu.be/rKoIdgaJU2k). The final system is shown in Figures 19 and 20.

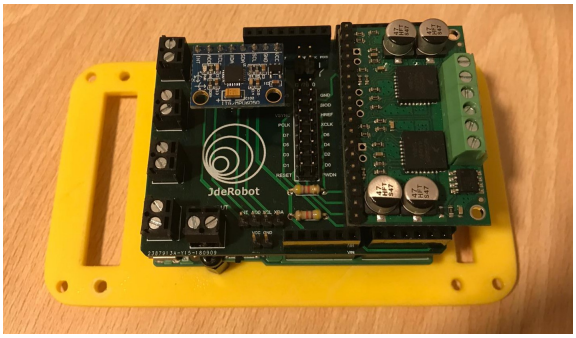


Figure 19. Final system with physical components assembled.

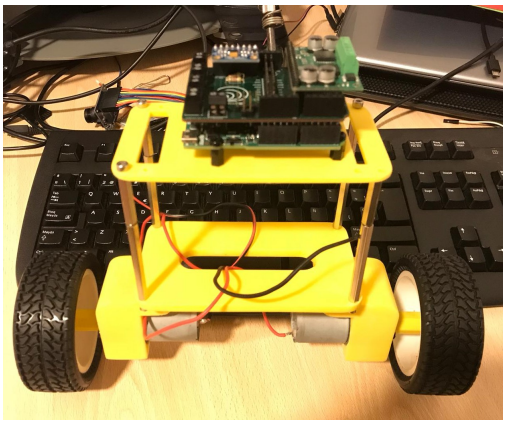


Figure 20. Final results of Self-Balancing Robot.

4.4. Alternative Design without Arduino

The proposed design before adding the microcontroller is shown in Figure 21.

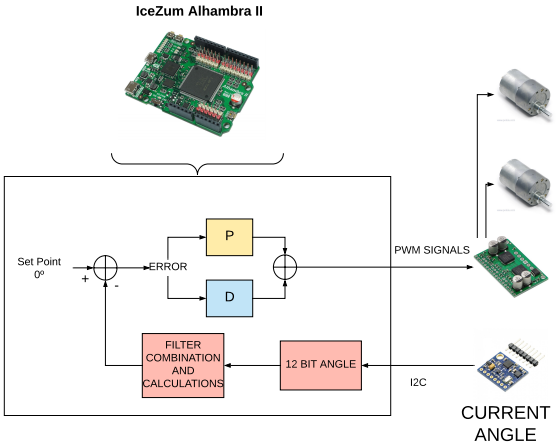


Figure 21. Hardware design of the inverted pendulum control without microcontroller.

The capture of the angle value was implemented in the FPGA. To do this, an i2c module was developed—and this presented a challenge, particularly considering the fact that a state machine and tri-state module were needed. The MPU6050 outputs were reminded without the use of DMP—2 bytes which corresponded with the accelerometer and gyroscope (12 bits for each one). The bytes had to be filtered and carefully treated in order to solve the drift problem and noise. Moreover, these values must be combined to allow reliability in terms of time and to blend the advantages of both sensors. The above development is not feasible with the number of logic gates or with the need to use sine and tangent functions. For this reason, a microcontroller with the i2c incorporated and the possibility of using DMP was clearly the best option.

5. Conclusions

FPGAs are a good intermediate option between microprocessors and ASIC for computing in many technological fields, as they combine the flexibility of software with the high-speed operation of hardware, and can keep costs low. However, most of the FPGA tools are currently proprietary and expensive. The open-source community has developed good FPGA editing and synthesis tools like IceStorm, IceStudio, and the IceZum Alhambra board. Currently supported FPGAs are not yet the most advanced models, but they already allow for the development of interesting robotic applications. A proof-of-concept robotic application was described in the present study—i.e., the inverted pendulum robot. It was fully developed using open FPGA technologies. It includes a perception module, a control module, and a motor module. Perception is based on an inertial IMU sensor. It was first developed with the sensor directly connected to the FPGA board, but there was significant noise in the data from this sensor. Finally, an intermediate Arduino processor was selected to filter out the noise and to send the filtered IMU data to the FPGA board through an i2c connection.

The control module performs a Proportional Derivative (PD) feedback algorithm inside the FPGA board. It feeds the motor drivers with the proper commands to keep the inverted pendulum robot raised and standing up even in the presence of disturbances. The FPGAs allow a new hardware approach to robot programming. Instead of a sequence of instructions, the robot logic is designed naturally in a parallel way by default. The main decomposition of robot tasks is now spatial in the FPGA circuit, which is more than sequential in the processor time. All the modules inside the FPGA hardware run concurrently at a pace of clock frequency. This can be of great use, for instance, in reactive robot behaviors. The hardware allows for continuous control instead of iteration-based software. Regarding future research, the authors are working on programming a drone with a camera

to visually follow colored objects in 3D, fully using open FPGA tools. This includes the support for image acquisition directly from the FPGA circuit and the communication with common drone flight controllers (like PX4 or ArduPilot) through PPM encoding. A second consideration to extend the current work is the development of a library of FPGA blocks which can be reused in further robotics applications.

Author Contributions: Conceptualization, J.O., E.C. and J.C.; methodology, E.C. and J.C.; software, J.O.; validation, J.O.; formal analysis, J.O., E.C. and J.C.; investigation, J.O., E.C. and J.C.; resources, J.O. and E.C.; data curation, J.O.; writing–original draft preparation, J.O., E.C. and J.C.; writing–review and editing, E.C. and J.C.; visualization, J.O., E.C. and J.C.; supervision, E.C. and J.C.; project administration, E.C.; funding acquisition, J.C.

Funding: This work was partially funded by the Community of Madrid through the RoboCity2030-III project (S2013/MIT-2748) and by the Spanish Ministry of Economy and Competitiveness through the RETOGAR project (TIN2016-76515-R).

Acknowledgments: The authors wish to thank Prof. Diego P. Morales from the Biochemistry and Electronics as Sensing Technologies Group of the University of Granada for his insightful suggestions and comments on this work.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nickolls, J.; Dally, W.J. The GPU Computing Era. *IEEE Micro.* **2010**, *30*, 56–69, doi:10.1109/MM.2010.41. [CrossRef]
2. Alkhafaji, F.S.; Hasan, W.Z.; Isa, M.; Sulaiman, N. Robotic Controller: ASIC versus FPGA—A Review. *J. Comput. Theor. Nanosci.* **2018**, *15*, 1–25. [CrossRef]
3. Sharma, A.K. *Programmable Logic Handbook: PLDs, CPLDs and FPGAs*; McGraw-Hill Handbooks: New York, NY, USA, 1998.
4. Brown, S.D.; Francis, R.J.; Rose, J.; Vranesic, Z.G. *Field-Programmable Gate Arrays*; The Springer International Series in Engineering and Computer Science; Springer: Berlin, Germany, 1992.
5. Semiconductor, L. FPGA Lattice. Available online: https://www.latticesemi.com/ (accessed on 10 September 2018).
6. Xilinx. 2018. Available online: https://www.xilinx.com/ (accessed on 20 September 2018).
7. Intel. 2018. Available online: https://www.intel.es/content/www/es/es/fpga/devices.html (accessed on 21 December 2018).
8. Intel. Stratix 10 GX/SX Device Overview. Available online: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/stratix-10/s10-overview.pdf (accessed on 20 September 2018).
9. Xilinx. Zynq-7000 All Programmable SoC Data Sheet: Overview. Available online: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf (accessed on 23 October 2017).
10. Altera. Nios II Gen2 Processor Reference Guide. Available online: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/nios2/n2cpu-nii5v1gen2.pdf (accessed on 20 June 2018).
11. Xilinx. Using the MicroBlaze Processor to Accelerate Cost-Sensitive Embedded System Development. Available online: https://www.xilinx.com/support/documentation/white_papers/wp469-microblaze-for-cost-sensitive-apps.pdf (accessed on 4 September 2018).
12. RISC-V. Available online: https://riscv.org/ (accessed on 25 December 2018).
13. Mi-V RISC-V Ecosystem. Available online: https://www.microsemi.com/product-directory/fpga-soc/5210-mi-v-embedded-ecosystem (accessed on 13 November 2018).
14. Dennis, D.K.; Priyam, A.; Virk, S.S.; Agrawal, S.; Sharma, T.; Mondal, A.; Ray, K.C. Single cycle RISC-V micro architecture processor and its FPGA prototype. In Proceedings of the 2017 7th International Symposium on Embedded Computing and System Design (ISED), Durgapur, India, 18–20 December 2017; pp. 1–5.
15. Freund, K. "Microsoft: FPGA Wins Versus Google TPUs For AI". Available online: https://www.forbes.com/sites/moorinsights/2017/08/28/microsoft-fpga-wins-versus-google-tpus-for-ai/ (accessed on 24 March 2018).
16. Ghosh, S. *Hardware Description Languages: Concepts and Principles*; IEEE Computer Society Press: New York, NY, USA, 2000.

17. Nane, R.; Sima, V.M.; Pilato, C.; Choi, J.; Fort, B.; Canis, A.; Chen, Y.T.; Hsiao, H.; Brown, S.; Ferrandi, F.; et al. A survey and evaluation of FPGA high-level synthesis tools. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2016**, *35*, 1591–1604. [CrossRef]
18. Chu, P.P. *RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability*; John Wiley & Sons: Hoboken, NJ, USA, 2006.
19. Donald Thomas, P.M. *The Verilog® Hardware Description Language*; Springer Science & Business Media: Berlin, Germany, 2008.
20. SpinalHDL User Guide. Available online: https://spinalhdl.github.io/SpinalDoc/ (accessed on 10 November 2017).
21. Raj, M.D.; Gogul, I.; Thangaraja, M.; Kumar, V.S. Static gesture recognition based precise positioning of 5-DOF robotic arm using FPGA. In Proceedings of the 2017 Trends in Industrial Measurement and Automation (TIMA), Chennai, India, 6–8 January 2017; pp. 1–6. [CrossRef]
22. Zhang, Z.; Xin, Y.; Liu, B.; Li, W.X.Y.; Lee, K.H.; Ng, C.F.; Stoyanov, D.; Cheung, R.C.C.; Kwok, K.W. FPGA-Based High-Performance Collision Detection: An Enabling Technique for Image-Guided Robotic Surgery. *Front. Robot. AI* **2016**, *3*, 51. [CrossRef]
23. Vachhani, L.; Mahindrakar, A.D.; Sridharan, K. Mobile Robot Navigation Through a Hardware-Efficient Implementation for Control-Law-Based Construction of Generalized Voronoi Diagram. *IEEE/ASME Trans. Mechatron.* **2011**, *16*, 1083–1095. [CrossRef]
24. Eteokleous, N.; Ktoridou, D. Educational robotics as learning tools within the teaching and learning practice. In Proceedings of the 2014 IEEE Global Engineering Education Conference (EDUCON), Istanbul, Turkey, 3–5 April 2014; pp. 1055–1058. [CrossRef]
25. Khatib, B.S.O. *Springer Handbook of Robotics*; Springer: Berlin, Germany, 2016.
26. Kung, Y.S.; Shu, G.S. Development of a FPGA-based motion control IC for robot arm. In Proceedings of the 2005 IEEE International Conference on Industrial Technology, Hong Kong, China, 14–17 December 2005; pp. 1397–1402. [CrossRef]
27. Nema, R.; Thakur, R.; Gupta, R. Design & Implementation of PID Controller Based On FPGA with PWM Modulator. *Int. J. Soft Comput. Eng. IJSCE* **2013**, *3*, 2231–2307.
28. Linares, J.C.; Barrientos, A.; Márquez, E.M. Hybrid Bio-Inspired Architecture for Walking Robots Through Central Pattern Generators Using Open Source FPGAs. In Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 1–5 October 2018; pp. 7071–7076. [CrossRef]
29. Arduino. Available online: https://www.arduino.cc/ (accessed on 26 May 2017).
30. Romanov, A.; Bogdan, S. Open source tools for model-based FPGA design. In Proceedings of the 2015 International Siberian Conference on Control and Communications (SIBCON), Omsk, Russia, 21–23 May 2015; pp. 1–6.
31. Wolf, C.; Lasser, M. Project Icestorm. Available online: http://www.clifford.at/icestorm (accessed on 15 January 2018).
32. Tarjeta IceZum Alhambra II. Available online: https://alhambrabits.com/alhambra/ (accessed on 20 January 2018).
33. IceStudio. Available online: https://icestudio.readthedocs.io/en/latest/ (accessed on 20 January 2018).
34. Romanov, A.; Romanov, M.; Kharchenko, A. FPGA-based control system reconfiguration using open source software. In Proceedings of the 2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), St. Petersburg, Russia, 1–3 February 2017; pp. 976–981. doi:10.1109/EIConRus.2017.7910719. [CrossRef]
35. Pathak, K.; Franch, J.; Agrawal, S.K. Velocity and position control of a wheeled inverted pendulum by partial feedback linearization. *IEEE Trans. Robot.* **2005**, *21*, 505–513. [CrossRef]
36. Orozco, L.M.L.; Lomeli, G.R.; Moreno, J.G.R.; Perea, M.T. Identification Inverted Pendulum System using Multilayer and Polynomial Neural Networks. *IEEE Latin Am. Trans.* **2015**, *13*, 1569–1576. [CrossRef]
37. Yu, L.H.; Jian, F. An Inverted Pendulum Fuzzy Controller Design and Simulation. In Proceedings of the 2014 International Symposium on Computer, Consumer and Control, Taichung, Taiwan, 10–12 June 2014; pp. 557–559. [CrossRef]

38. Altium Designer. Available online: https://www.altium.com/altium-designer/ (accessed on 19 June 2018).
39. SolidWorks. Available online: https://www.solidworks.com/es (accessed on 25 October 2018).