Self-Balancing Robot Based on the Terasic DE10-Nano Kit

Contents

Introduction

In this article, we describe the inner workings of a self-balancing robot based on the Terasic DE10-Nano Kit. Standing at 13 cm tall and 19.5 x 7.5 cm wide, this small, two-wheeled robot can make corrections to its upright position through the use of Proportional-Integral-Derivative (PID) controllers—to create a closed-loop (meaning constant) feedback system to maintain stability. An ultrasonic sensor functions as the ears of our robot and enables it to move around without bumping into obstacles. And we use Hall effect sensors on the DC motors to determine motor speed and direction.

We begin with an overview of the robot's basic architecture that comprises the balance and control system and provide reasoning for some of the architecture decisions. We then give an explanation of the basic operation of the robot (inner workings) and discuss what you can do with the robot (expansion possibilities). Throughout the article we also highlight important advantages (virtues) of developing with a Field Programmable Gate Array (FPGA) for embedded applications (using our self-balancing robot as the target application to demonstrate those virtues).

Audience

This article is for developers who'd like to learn more about the advantages of FPGAs. We've chosen robotics as the target application but the motivation here is not to promote FPGAs for robotics. While robotics is a great application area for custom microprocessors, (what we ultimately configure an FPGA to be), for developers to know how take advantage of an FPGA for a robotics project depends on their experience level and whether or

Commented [GE1]: X? y?

Commented [GE2]: Maybe reference echolocation.

Commented [GE3]: Make sure to give a brief explanation of Hall effect sensors.

Commented [GE4]: Still need to address why the developer would purchase the robot in the first place. I assume they want to develop on it. How can they use it? What can the robot be used for? It's not an appliance but a research tool? What's the purpose of purchasing the robot?

Commented [GE5]: Add a better transition. From microprocessors to for developers.

not they have clear understanding of the benefits they'd like to gain from using an FPGA. For the purposes of this article, we'd like to make it clear what those FPGA advantages are (what we refer to as "virtues") and how the robotics application we've chosen, a self-balancing robot, demonstrates those virtues.

However, developers have the option to use the robot as a development platform for robotics applications.

Prerequisites for developers

Before developing robotics applications on this platform, we recommend that users meet the following prerequisites:

- Knowledge of FPGA development tools and processes
- Master the use of the Terasic DE10-Nano board which includes knowing how to create projects and using the Nios® embedded processor tools and Platform Designer (formerly Qsys) tools in the Intel® Quartus® Prime software

Recommended tutorials

- Use the Quartus software to program your first FPGA device
- Use the Qsys tool to create a custom FPGA hardware design
- Learn how to instantiate and configure a Hard Processor System (HPS) component using Qsys
- Create your first Nios II software design

Commented [GE6]: Most readers will probably scan this area.

Architecture Overview

Note: The Self-balancing Robot is based on a System on Chip (SoC) FPGA which means that an FPGA and ARM processor are on the same chip. With the SoC FPGA, we have two processor options available to control the robot: 1) we can use the ARM processor or 2) implement a Nios® II processor onto the FPGA. For the purposes of this article, we present information based on the configuration option that implements a Nios II, a 32-bit reconfigurable embedded processor. For more information see section **3.1 Configuration Mode Switches** of the *Self-Balancing Robot User Guide* available on the Terasic website.

For the architecture overview we describe the robot's set of hardware and software components, the key functions of the two main hardware boards and how they communicate with sensors (which provide input to the balance algorithm). In the section that follows, **Architecture Decisions and Implementation**, we provide further detail as well as the reasoning (the *why*) for the architecture decisions.

The architecture of the self-balancing robot system is built on two main boards:

1) Terasic DE10-Nano board
2) Motor Driver board

These two boards work together to comprise the balance and control system of our self-balancing robot.

Commented [GE7]: Understanding the self-balancing robot. The why is important here. What the product does + purpose of the hardware and software. For example, **why** are there sensors? Sensors are used for… And **why** did we use two boards here? A high level overview but no the details (those come later).

Commented [GE8]: The architecture of a system is its 'skeleton' – the highest level of abstraction of a system. How do modules interact with each other, what boards are present… The design of the entire system. The architecture deals with the reasons for the decisions – *what* and *why* something's been done but not *how*.

Hardware

The DE10-Nano board is responsible for the control/balance system of the robot (that is, it's dedicated to processing the functions of the balance algorithm). To control the rotation of the motors (and thus the wheels), the DE10-Nano board sends control signals (via GPIO) to the Motor Driver board (which then drives the motors). Sensors are included on the Motor Driver board and provide inputs (sensor data) to the balance algorithm which can then determine what adjustments need to be made for the robot to keep its balance.

Terasic DE10-Nano board

The DE10-Nano board is based on a Cyclone® V System on Chip (SoC) FPGA which contains both a hard processor (ARM*) and the FPGA (Cyclone V). That means we have two processor options to control the robot.

Users can choose to execute the balance algorithm entirely through the ARM processor or implement a Nios II soft processor within the FPGA fabric. See the *Self-Balancing Robot Getting Started Guide* from Terasic for further details on how to switch between these two processor configuration modes. For the self-balancing robot we describe in this article, we've chosen to implement a Nios II processor.

Commented [GE9]: Three different options (architectural decisions) for implementing the PI/PD loops (balance control): ARM* processor, Nios® II processor (IP core, soft processor), and Custom HW (use tool called HLS to convert the code written in C (PID algorithm)… and then turn that into RTL…).

ARM processor

The ARM processor is responsible for running Linux*.

Nios® II "soft" processor

With the configuration mode we've selected, the Nios II processor is responsible for controlling the robot. That is, we dedicate the Nios II to processing the functions of the balance algorithm for the robot. We have a dual purpose in implementing a Nios II within the FPGA. First, we want to offload the workload from ARM processor (to boost the overall system performance) and second, using a Nios II can be easier to use for those getting started with FPGAs (ease of use).

Commented [GE10]: Easier than creating a bit stream. Someone new to FPGAs won't be developing their own bitstreams (blueprint written in an HDL than tells the FPGA how to configure or reconfigure itself).

Note: Here "soft" means that the Nios II processor and all peripherals (I/O) are written in a Hardware Description Language (Verilog or VHDL) **–** that is, we can implement the processor architecture entirely within the programmable logic of the FPGA.

Motor Driver board

Lorem ipsum. Receives control signals from the FPGA.

Motors

Commented [GE11]: Motor control signals drive the H-bridge…

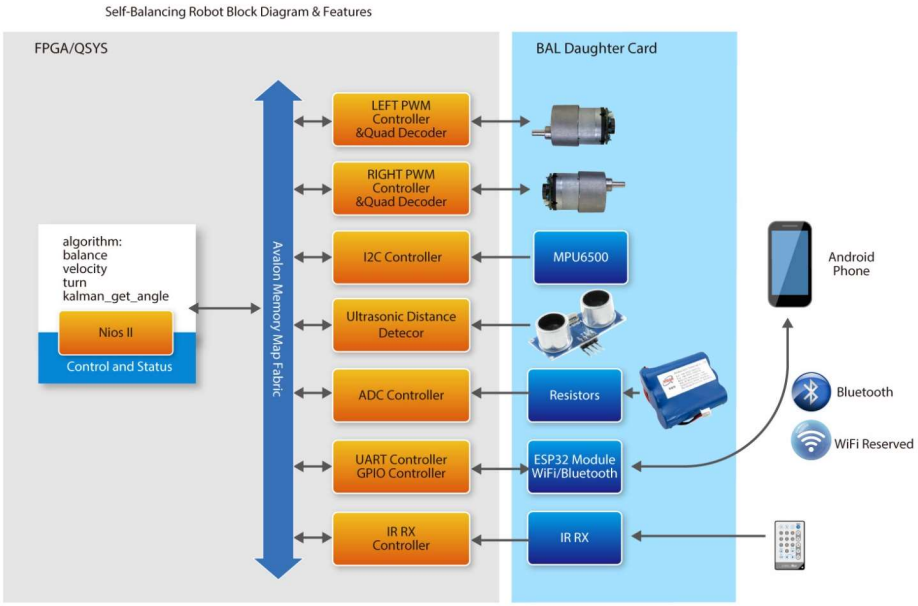
The motors drive the wheels of the robot and are controlled by a chip (Toshiba TB6612FNG).

Bluetooth Module

The Bluetooth module enables users to remote control the robot by a smartphone app. Users can download the app… Yes, this is available!

Sensors

We include sensors on the Motor Driver board to enable the robot to maintain its upright position, avoid obstacles, and determine wheel speed and direction. And we interface with the sensors on the Motor Driver board by adding or creating soft IP within the FPGA. The orange blocks in the figure below represent soft IP. With this design, we demonstrate the advantage of being able to expand I/O using an FPGA. If we'd like to include more sensors for additional functionality, we can simply reconfigure the FPGA to include the interface for that sensor.



MPU-6500 (accelerometer and gyroscope)

The MPU-6500 is a single chip that integrates a 3-axis accelerometer and 3-axis gyroscope and we use the data from this sensor to determine the tilt angle of the robot.

Note: While the DE10-Nano board has a built in 3-axis accelerometer (ADXL345), it is not used here for the Terasic Self-Balancing Robot. Instead, the Motor Driver board uses a MPU-6500 to determine the tilt angle of the robot. This sensor provides a more accurate measurement of the tilt angle because of the additional data that comes from the gyroscope.

Ultrasonic sensor

We use an ultrasonic sensor to measure distance and we use the values from this sensor to enable the robot to avoid obstacles (10 cm from an object) or follow an object (10-20 cm from an object).

Hall Effect sensor

Hall Effect sensors are positioned on the motors to determine the direction and speed of the wheels.

The measured values from the accelerometer + gyroscope sensor module are inputs to the balance algorithm (which is implemented as PI or PD loops).

Sensors	**Purpose**	**What it measures**	**Location**
MPU-6500 (accelerometer + gyroscope)	Provides input data to maintain upright position (when stationary or in motion)	Tilt angle, θ	Motor Driver board
Ultrasonic sensor	Measures distance from an object to avoid obstacles or follow objects	Distance	Motor Driver board
Hall effect sensors	Provides input data used to control the motors	Rotational speed and direction of the wheels	Attached to the DC motors

Software

Linux OS

The ARM processor runs the Linux operating system.

Applications

The balance algorithm runs on the Nios II processor.

Architectural Decisions & Implementation

We implement a Nios II processor, a 32-bit reconfigurable processor, within the FPGA and then customize our peripheral set (based on the sensors and communication modules we'd like to include). This enables us to easily configure the FPGA to be a custom microcontroller. For robotics, a standard microcontroller is often used (especially for those getting started in robotics). And robotics is often an appropriate application area for custom microprocessors (i.e., microcontrollers) when a generic, off-the-shelf microcontroller does not meet your application requirements.

Commented [GE12]: Fix. Reader gets confused.

Choosing to implement a soft processor

Nios II "soft" processor

The Nios II processor core is soft Intellectual Property (IP) that you download onto an Intel® FPGA.

Commented [GE13]: Do we have any data to show the actual numbers for a boost in system performance?

Commented [GE14]: Reword this using your own understanding and mention something about soft processors and IP cores.

In the case of the self-balancing robot, we don't replace the main processor (ARM) with the soft processor (Nios II) but instead make the Nios II responsible for controlling the robot and the ARM continues to run the Linux OS. We configure the Nios II to be dedicated exclusively to sending control signals to the Motor Driver board, which then adjust the direction and speed of the robot's wheels (turning, back and forth motion, spinning, etc.). By implementing a soft processor within the FPGA, we can offload work from the main processor (ARM) which can then focus on what it does best (tasks such as running the operating system, network stacks and file systems). And the result is often a boost in the overall system performance.

We demonstrate three key advantages (virtues) to using a soft processor (Nios II) within the FPGA and by using the FPGA in general:

1. **I/O Expansion**: With our robot, we expand I/O by extending the basic functionality of the Nios II processor. And we tailor the I/O to our specific application by defining custom peripherals (UART, PWM, CAN bus, etc.).
2. **Boost Performance**: When we offload the ARM processor by configuring the FPGA to be responsible for the balance and control system of the robot, what results is a boost in system performance.
3. **Adapt to change**: Lorem ipsum.

Commented [GE15]: Add camera card; adapting to changing requirements (not technology specific); Let's come back to this.

Reason for the use of two boards to control the robot

A Cyclone V SoC FPGA is the key component we use on the DE10-Nano board. We implement a Nios II processor within the FPGA and then appoint the Nios II processor to run our balance algorithm. Because the General Purpose I/O (GPIO) pins on the FPGA are unable to drive the motors of the robot, we need to include an additional board to deliver enough current to drive the DC motors. So we've added a Motor Driver board (which is responsible for receiving control signals from the FPGA) that uses a control circuit to power the motors.

Commented [GE16R15]: How is this better than doing it in software? Re-program the FPGA. It is never easier to do something in hardware than to just do it in software. Couldn't you just do it on a CPU in just software?

Commented [GE17]: Need a better transition here.

Commented [GE18]: Why? What are the I/O requirements to drive the motors?

Note: The GPIO pins on the FPGA are appropriate for signaling (for current requirements in milliamps) but they don't have enough current to drive the motors (which requires Amps). And while the GPIO by themselves don't have the amperage to drive the motors of the robot, the GPIO on the FPGA are still responsible for sending the control signals to the Motor Driver board. It doesn't provide enough power.

The role of the Motor Driver Board

The Motor Driver board receives control signals from the FPGA and contains an additional circuit (a Toshiba TB6612FNG) to control the DC motors of our two-wheeled robot – the rotation of wheels (clockwise or counterclockwise) and applying brakes to either slow (decelerate) or stop the robot. The Motor Driver board also contains sensors (which provide data inputs to the balance algorithm) and communication components (Bluetooth and IR receiver for remote control operation).

Basic operation: Balance control system

The balance control system of the robot enables the robot to dynamically respond to a rapidly changing environment (what balance really means here). The sensors on the Motor Driver board provide the input data to our balance algorithm (running on the Nios II) which is implemented as Proportional-Integral-Derivative (PID) controller.

How balancing works

Commented [GE19]: Explain what we mean by 'balancing' here. Balance means what for the robot?

To understand what's required for balancing, we have to appreciate the physics of an inverted pendulum – a pendulum with its center of mass above its pivot point. Our self-balancing robot is similar to an inverted (or upside down) pendulum. That means the majority of the robot's weight, or its center of mass, is located above its pivot point (in this case, somewhere near wheels).

What results from this setup is an inherently unstable robot that must be actively balanced to maintain its upright position. What we need for balance is a dynamic response to a rapidly changing environment. And that requires us to implement a feedback system (a constant feedback loop) to monitor our robot's tilt angle and speed and then make adjustments (the instant an error happens) to keep it from taking a tumble or crashing into a wall.

Why not just use a three or four-wheeled robot?

You may wonder why we don't just design a robot with three or four wheels. That would take care of the balance problem. But the robot would. Well, with a two-wheeled design our robot can rotate in place (spin) to change direction instantly and navigate tight spaces with precision (something it's three of four-wheeled counterparts are often clumsy at doing).

What we do to counteract a fall

The action we take to counteract a fall is to drive the motors in the direction of the fall – our goal being to maintain the robot's center of mass above its pivot point (physics speak). This action takes a cue from human behavior – to avoid a fall, you would take a step in the direction of your fall (whether that's forward or backward).

To drive the motors (to make corrections), we first need to know a few things about the state of our robot:

1) Angle of inclination (tilt)
2) Direction of the fall (forward or backward)
3) Speed at which it's falling

We obtain this information from the MPU-6050, a sensor combining a 3-axis accelerometer and 3-axis gyroscope, located on the Motor Driver board – we combine the values of both the accelerometer and gyroscope and pass them through a Kalman filter to get the most accurate measurement of tilt angle for the robot.

Balance algorithm

The balance algorithm runs on the Nios II processor and we implement the algorithm as closed-loop controls (PID controllers). The ability to detect and control its tilt angle, speed and turn from a closed-loop feedback system (the balance system) results in a robot that can dynamically respond to a rapidly changing environment (thus, it can "self-balance").

Main algorithm functions

The main algorithm functions are a Balance, Kalman, Turn and Velocity.

Balance

The balance function is implemented as a PD controller.

Kalman filter

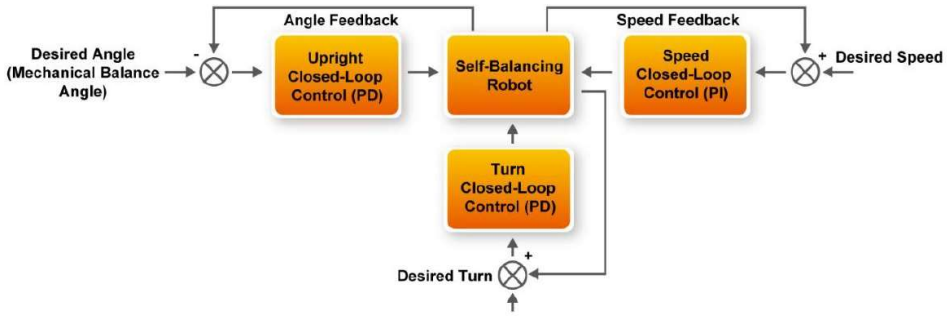
We use a Kalman filter to process the combined values of the accelerometer and gyroscope for a more accurate measurement of the tilt angle.

Turn

The balance function is implemented as a PD controller.

Velocity

The balance function is implemented as a PI controller.



Closed-loop system for feedback and correction

PID Controllers and Sensor Data

Through the use of PID controllers, we establish a system of feedback and correction to allow the robot to continuously adjust itself (to enable it with a dynamic response). The PID controllers that make up the balance algorithm read sensor data (**feedback**) and determine an desired output values (for example, desired angle or turn) by first calculating and then summing the Proportional (P), Integral (I) and Derivative (D) terms. The balance algorithm can then figure out what corrections need to be made and then sends those adjustments to the motors (**correction**). For our robot, Proportional Integral (PI) and Proportion Derivation (PD) controllers (variations of a PID controller) are used to generate the desired system response.

Accelerometer + gyroscope: how we measure tilt angle

We use a Motion Tracking Device, MPU-6500, on the Motor Driver board which combines a gyroscope and accelerometer to measure the tilt angle, θ, of the robot. And combining these values provides are more accurate measurement. Essentially, these values help us to determine the orientation of the Motor Driver board (the bottom half of the robot's body) which then enables us to figure out whether the robot is level or at an angle.

Ultrasonic sensor: how we measure distance

The ultrasonic sensor functions as the ears of our robot (yes, ears and not eyes!) and enables us to measure distance. Based on the physics of reflected waves, these sensors work by emitting ultrasonic sound waves and listening for an echo. That is, if the sound wave is reflected back, we know that there is an object in front of the robot. And the time it takes for the sensor to transmit and then receive that sound wave is how we determine the distance between our robot and an object. And unlike proximity sensors (which use reflected electromagnetic waves, such as infrared, to measure distance), the ultrasonic sensor is reliable in dark environments (such as a dimly lit room).

Note: ultrasonic denotes frequencies above 20,000 Hertz (the upper audible limit for human ears meaning that ultrasonic sound waves are imperceptible to humans). So you won't hear the robot as it emits ultrasonic sound waves and listens for an echo.

There are two operation modes based on data from the ultrasonic sensor. Users can choose to operate a mode that only performs obstacle avoidance or a mode that includes object following (in addition to obstacle avoidance). See the *Terasic Self-Balancing Robot Getting Started Guide* for more information on how to select these modes of operation.

Obstacle avoidance

For objects that are within a distance of 10 cm from the robot, our robot will autonomously avoid the object by maneuvering around it.

Object following

For the object following mode, the robot is not controlled by the smartphone app or the IR remote control. For objects that are a distance between 10-20 cm and moving (but maintaining a distance within the specified range), the robot will follow the object.

Hall sensors on the DC motors: how we detect motor speed and direction

Bluetooth module: how we communicate with the robot

Expansion possibilities (FPGA and HPS [Hard Processor System] I/O)

We can add vision to our robot through the use of a camera card.

Add a camera card

Camera color recognition

Next Steps

- Go to maker.io (part of Digi-Key*) for a step-by-step tutorial on how to setup the self-balancing robot
- https://www.digikey.com/en/maker/projects/189047050e514baf9f5c1c45428623e3
- https://eewiki.net/pages/viewpage.action?pageId=71958550

Open-source examples

Resources

Self-Balancing Robot Schematics

http://mail.terasic.com.tw/~johnny/2018/bal/balance_car_c0.pdf

The Terasic Self-Balancing Robot Kit Unboxing Video is now live at:

- https://youtu.be/k_ykIVfpmZw
- https://bcove.video/2INZdDE

Commented [GE20]: What is it that we hope the developer/reader does after learning about this robot (architecture overview, design decisions and implementation)? What do we want to inspire them to do? What are trying to help them understand?

Commented [GE21]: How you can implement. See it in action and then learn how to do it for yourself.

Commented [GE22]: Tutorial of what?

Commented [GE23R22]: Unboxing video and technical deep dive and algorithm implementation (follow the orange ball) – TBD stuff

The Terasic Self-Balancing Robot Kit Product Landing page is not yet live outside the US:

- https://www.digikey.com/products/en?keywords=P0509&keywords=&pkeyword=P0509&v=771