Lab 4: Parallel Port and Timer

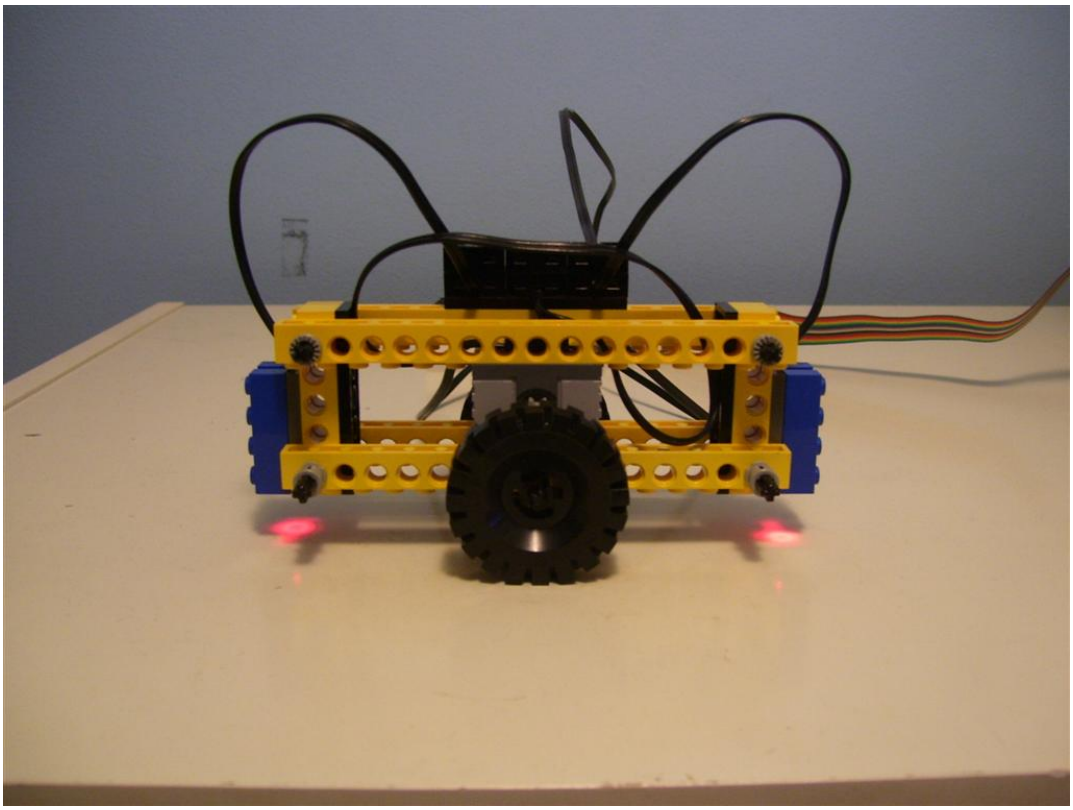
Description

In this lab you will learn how to use the Parallel Interface (JP1/JP2 connectors on the DE1-SoC board) to communicate with another piece of hardware, the *Lego controller*. The Lego controller connects to *motors* and *sensors*, allowing your program to interact with the physical world. You will also learn how to use the *Timer* peripheral on the DE1-SoC computer to measure delay intervals.

You will be given a self-balancing device made from Lego which will be controlled by the Lego controller and the DE1-SoC Nios II processor. In Part 1 of this lab you will implement a basic balancing algorithm to read the sensors and control the motor. In Part 2 you will improve your algorithm by slowing down the motor, using the timer to implement Pulse Width Modulation (PWM).

Self-Balancing Device Description

The device consists of a frame and a Lego motor balanced on a single axle with two wheels, with a sensor at each end of the frame. Each sensor measures its distance from the ground based on the reflection of a built-in LED. The closer the sensor is to the ground, the more light it senses, and the higher the value it reports. By comparing the values of both sensors, the device can determine if it is tilting in one direction or another. The goal of the lab is to receive this information, and determine how much to roll the device, using the motor, in the direction that will correct that tilt.



Part 1

In Part 1 you will write an assembly program that will read the values of the two sensors. If the values of the sensors indicate the device is tilting, you should drive the motor either clockwise or counter-clockwise, so that the device rolls in the direction that will correct that tilt. Otherwise, if the sensor values indicate the device is not tilting, you can turn off the motor. **You must use both sensors.** Using just one sensor tends to perform poorly as it can't adjust to changing lighting and reflectivity conditions.

The Lego Controller Device Documentation describes how you can control the motors and read the sensor values. As described in that document, the sensor interface works either in value mode or state mode. For this lab, **use the value mode of the sensor interface**.

Notes

- Be careful to not set the motor to always-off when reading the sensors. Doing so will result in the motor being off most of the time. (Why?)
- Setting the direction bit to 0 for a motor does not always mean clockwise rotation. The direction of rotation depends on the polarity of the connection, i.e., which way it is plugged in. Rotating the connector reverses the motor direction.
- Be careful when you test your device and make sure the motor behaves the way you expect it to. If you guessed the direction bit wrong or there is an error in your code, the device could drive off in one direction.
- Not all sensors are identical. It is likely that your two sensors will report two slightly different values when placed the same distance from the same surface. You can assume the sensors are identical when you write your code, but in rare cases, you may want to modify your code after you test your two sensors.
- When you read from a sensor, make sure the corresponding Sensor Ready bit is valid (low), or else the value you read will be invalid.

Part 2

Once you test your code from Part 1, you may see that your device is not able to achieve a perfect balance. Instead it rapidly oscillates back and forth. This is because the motor is stronger than we want it to be, which causes it to "overcompensate" when trying to correct a tilt. We now want to reduce the strength of the motor by rapidly turning the motor on and off (pulse width modulation).

You will use the Timer peripheral to accurately time the on and off durations. Although this could be done (with less accuracy) using program-based delay loops, **you are required to use the timer as described here**. Write a Nios II assembly language subroutine that, when called with a parameter, `N`, waits for `N` Timer cycles before returning, using the timer peripheral. Incorporate this subroutine into your code for the Part I program to implement the pseudocode described below. **This must be a proper subroutine** (call, return, parameter passing, etc.)

```
/* Pseudocode: */
Turn Motor ON
Delay 262150 cycles using the Timer
Turn Motor OFF
Delay 262150 cycles using the Timer
```

The pseudocode uses a 50% duty cycle (on=262150, off=262150). This is a good starting point but you are encouraged to experiment with different values of the duty cycle (on/(on+off)) to change the strength of the motor.

Preparation (2 Marks)

The Lego balancing device will be provided **in the lab.** Do the following as part of your preparation:

1. Read the following documents to learn how to interface with both the Lego controller and the timer:
    - Lego Controller Device Documentation
    - Timer Documentation

These documents also include a sample assembly code that demonstrates how to properly use the Lego controller and the timer. **You are allowed to reuse this code, but you must understand it.** For help on how to read the documentation you may wish to consult the Device Documentation Template.

2. Since we will be using the stack in this lab, the stack pointer must be initialized (In Lab 3, the C Runtime Library did this for you). What is a good value to initialize it to, and why? (The computer's memory map may be useful.)
3. Answer the following questions about bit masking:
    a. Write assembly instruction(s) to set bit 11 of r2 to 1, without changing other bits
    b. Write assembly instruction(s) to set bit 8 of r2 to 0, without changing other bits
4. Answer the following questions, assuming the Lego controller is plugged into JP1:
    a. To properly configure the Lego controller I have to write the value _____ to the direction register, which is located at address _____.
    b. To turn on motor 2 I have to set bit ___ at memory location _____ to the value ___.
    c. Give an example of a 32-bit value that will enable sensor 1 for reading when written to the GPIO port: _____ This 32-bit word will implicitly turn motor 2 ___ (on/off/unchanged/undefined/?).
    d. To check if sensor 1 is giving a valid data I have to read bit ___ from address _____ and test if the bit has the value ___.
    e. To read the current value of sensor 1, I have to read bits ___ to ___ from address _____.
5. Answer the following questions about the Timer:
    a. Does the timer count up or down?
    b. How do you initialize the timer's period?
    c. How can you check if the timer has completed its period?
    d. Suppose the processor executes one instruction every clock cycle, and you start the timer at cycle 7, as shown in the table below. At what time does the next instruction execute?

Time (cycles)	Instruction
1	`movia r3, 0xff202000`
3	`movi r2, 10000`
4	`stwio r2, 8(r3)`
5	`stwio r0, 12(r3)  # 10000 cycle period`
6	`movi r2, 4`
7	`stwio r2, 4(r3)  # Start timer`
???	`ldwio r2, 0(r3)`

6. Write the timer delay subroutine used to be used in Part 2. This must be a proper subroutine that follows the NIOS II ABI
7. Write a simple test program that uses your timer subroutine to blink an LED with a period of 1 second. (Recall: Red LEDs are at location `ff200000`)
8. Write the assembly language programs for part 1 and part 2 described above. Your code must be well commented. Compile your code using the Altera Monitor program and fix any compilation errors.

In Lab (2 Marks)

- Demonstrate both your working programs (separately) to your TA.
    - Part 1 (1 mark)
    - Part 2 (1 mark). Use a PWM period of 1 second to prove that you have implemented PWM correctly.
        - If you don't get this part working, you may show the working timer subroutine test program from part 7 of the preparation for 0.5 marks
- Put all source files in code.zip and **submit it on blackboard.**

Tips

- Place the device on a uniform light-coloured surface, this will help the sensors measure distance more accurately, since sensor readings depend on the reflectivity of the surface.
- Don't forget to correctly set the direction register for the Lego Controller.
- Make sure the Lego controller is connected to the same 40 pin expansion header (JP1 or JP2) that you use in your program.
- When you try to run your program make sure the slider switch on the Lego controller is in the middle position.
- To work with individual bits within a word, use the bitwise AND and OR operations.
- Remember to use the I/O versions of load/store assembly instructions (e.g., ldwio, stwio) when you are reading/writing from/to an I/O peripheral.
- To speed up marking in the lab, have both projects open in two instances of the Monitor Program. "Disconnect" from one, then "Connect" on the other one to switch (You will still need to "Load" the program).

Quiz (1 Mark)

Be prepared to answer any questions about your code, and about how the Lego motors, the sensors and the timer work.