Evaluation Sheet: Design Proposal
ECE 2031: Digital Design Laboratory

Print clearly and legibly.

Students' Names: Joel Bailey
                  David Thomas
                  Jacob Nahe
                  Chris Shaw

Course and Lab Section: 2071 L06

Lead Undergraduate TA's Name: Julian Curran

Graduate TA's Signature: ____________________________

Pledge of Academic Honesty: On my honor, I pledge that I have neither given anyone else assistance or information on this lab report nor have I received assistance or information from anyone other than Professors Collins or Bourgeois, the UTA lab assistants, or the GTA writing consultants. I further pledge that I am in full compliance with every statute and codicil of the Georgia Tech Honor Code and that all of the work that appears in this report is my own, unless it is attributed to another source. I understand and agree that any violations of the Georgia Tech Honor Code will be forwarded to the Dean of Students for adjudication.

Students' Signatures and ID Numbers: ____________________

Date: ________________________

Grading Rubric: 300 points

<table>
<thead>
<tr>
<th>SECTION</th>
<th>POINTS POSSIBLE</th>
<th>POINTS EARNED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executive Summary</td>
<td>10</td>
<td>.1</td>
</tr>
<tr>
<td>Technical Approach</td>
<td>40</td>
<td>37</td>
</tr>
<tr>
<td>Management Plan</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>Grammar/Mechanics/Punctuation</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Writing Style</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Overall Effectiveness of Proposal</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Final Grade</td>
<td>300 (100 x 3)</td>
<td>97</td>
</tr>
</tbody>
</table>

Comments: Well-written, good technical info!
Executive Summary

This project seeks to design a modular and comprehensive suite of hardware devices to allow precise control of robot motion both in closed-loop and direct duty-cycle manipulation forms. To accomplish these tasks, a set of single-purpose modules will be designed and combined into the final velocity controller device. A pulse-wave-modulation (PWM) signal generator will need to be implemented to drive the robot motors, a velocity averager is needed to establish a stable velocity signal, and finally the closed-loop control device itself will act as a bridge between these two devices, reading information from the velocity averager and modifying the duty cycle of the PWM signal generator to match the robot velocity to the desired velocity. The user will be able to select between modes and interface with the velocity controller through special registers in SCOMP. The modularity of the approach is its main strength because it combines extensibility and ease of debugging, allowing users to specify new velocity control paradigms and rapidly test them if it is so desired. This design does not only encompass a velocity controller, it can be framework for a wide variety of future devices that deal with robot motion and control.
A Modular PWM-Based Closed-Loop Controlled Velocity Controller for the AmigoBot

Technical Approach

Problem Description

The goal of this project is to design a VHDL device to control the AmigoBot motors, meeting the following specifications:

1. Produces two motor control signals for the left motor, one for motor activation and one for motor direction
2. Accepts a 16-bit two's complement velocity command that
   a. Fully enables the motor when the command is either 32767 or -32767
   b. Disables the motor when the command is either 0 or -32768
   c. Drives the motor with a variable duty cycle equal to the magnitude of the input divided by 32767 when the input is some other value
3. Drives the motor counterclockwise when the velocity is positive and clockwise when it is negative

In addition, a SCOMP program which performs the following tasks is also necessary:

1. Reads the a velocity from the 16 switches on the DE2 board and passes this velocity to the velocity controller
2. Runs the left motor only while KEY3 on the DE2 board is pressed to protect the robot from damage if it were to fall off its pedestal
3. Outputs an average velocity from the optical encoder onto the LCD or 7-segment display

Proposed Overall Design

In order to fulfill the requirements mentioned in the previous section, the team decided to adopt a modular approach, splitting the design into smaller parts so that testing can be done separately for each individual part. This design approach makes debugging more efficient because errors can be resolved to a single component rather than a complicated system of components. In addition, designing smaller single-purpose modules allows for the reuse and recombination of the modules at a future time.

The basic velocity controller will be composed of an input handler, an input decoder and a PWM generator. The input handler will primarily be used to latch the data when the SCOMP writes to the velocity controller. This latched data will then be directly fed to the input decoder (seen in Figure 2) which will transform the data from a 16-bit two's complement number to a direction and a twelve-bit magnitude signal. The direction signal will be connected to the motor direction signal to control the
motors motion. The twelve most significant bits of the magnitude will be passed to the PWM generator. The most significant twelve bits were chosen because the PWM generator can only produce a PWM signal with $2^{12}$ different duty cycles evenly spaced from 0% to 100%. More details on the PWM generator are explained in a later section. A schematic of the overall design can be seen in Figure 1.

Figure 1. Velocity controller schematic that connects the input handler, input decoder and the PWM generator together.

```vhdl
-- input_decoder.vhd (VHDL)
-- This code implements a device which transforms a 16-bit twos complement number into a 12-bit sign-magnitude number for motor control.
-- David Inouye
-- ECE2031 L06
-- 03/02/2011

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;

-- Define the ports of the PWM generator
ENTITY Input_Decoder IS
  PORT( data : IN std_logic_vector(15 DOWNTO 0);
        duty : IN std_logic_vector(11 DOWNTO 0);
        direction : IN std_logic
```

```vhdl
  OR std_logic_vector(0);
  PWM_signal : OUT std_logic;
  MOTOR_en : OUT std_logic
);```
); -- The pulse width modulated signal
END Input_Decoder;

ARCHITECTURE a OF Input_Decoder IS

signal neg_data : std_logic_vector(15 DOWNTO 0);

BEGIN
-- checks the signed bit of the 2's compliment number and sends as
direction
  direction <= data(15);
-- takes the reverse of 2's compliment
  neg_data <= NOT(DATA - 1);

PROCESS(data)
BEGIN

Figure 2. The VHDL code for the input decoder which transforms its input from a twos complement number to a direction and magnitude output.

Pulse Width Modulation Design

Since a simpler Pulse Width Modulation (PWM) generator was constructed in a previous project, the team decided to simply alter the original PWM generator to save time. The original PWM generator only generated different positive duty cycles in increments of one eighths from one eighth to seven-eighths. For the proposed design, however, a greater degree of precision is needed so that the closed-loop control mechanism described later in the document can be more accurate.

The primary constraint for choosing the amount of resolution possible for the PWM generator is the maximum clock speed of the DE2 board. Using a phase locked loop, the 50 MHz built in clock signal of the DE2 board can be transformed to at most a 400 MHz clock frequency. From this clock speed, the team calculated that in order to meet the specification that the frequency of the motor signal needs to be approximately 100 kHz, the PWM generator could have $2^{12}$ or 4,096 different duty cycles spaced out evenly between a 0% and 100% duty cycle. This implies a PWM frequency of 97.67 kHz. Because of this design, the closed loop controller mechanism can adjust the duty cycle by as little $1/(2^{12}) \approx 0.01\%$. Though it may not be necessary to control the duty cycle with this amount of precision even for the closed loop controller mechanism, the team decided to design the PWM generator with the maximum resolution possible in order to accommodate future unforeseen uses. If the user desires less resolution, a different input decoder can be implemented that maps ranges of inputs to single duty cycles rather than the the one-
to-one mapping between input and duty cycle that is currently implemented.

A VHDL file was created that consists of a state machine—which keeps track of whether the signal should be low or high—and an internal counter to provide the correct timing for the duty cycle modulation. The UML state diagram can be seen in Figure 3. The initial state is L, or low, since the motor enable signal is active low. Then when the counter is equal to the input value—denoted by “duty” in Figure 1—the state transitions to state H, or high. When the counter overflows to zero, the state switches back to state L. The equality comparison of the input signal and the counter is performed by computing an internal XOR signal of the current count and the input signal. In this way, the XOR signal will be all zeros when the signals are equal and therefore the constant zero can be compared to XOR signal for an equality comparison. The full VHDL code can be found in the Appendix A.

![State Diagram](image)

**Figure 3.** The state diagram for the PWM generator that alternates between the L and H state based on the count.

**Velocity Averager**

It is necessary to calculate an average velocity both to meet the project requirements and to allow for closed-loop control. The calculation of the average velocity can be performed in hardware without use of SCOMP. This frees SCOMP for other tasks such as user interaction and robot pathfinding. A simple finite impulse response (FIR) filter will be designed and implemented in hardware to perform a running average on the data. In designing this FIR filter, testing will have to be performed to find the best balance
of filter length and performance. A longer filter length potentially sacrifices performance to generate a more stable average velocity, whereas a shorter filter might use fewer board resources while reporting a less stable average velocity. Ideally, the average velocity would change like the reading on a speedometer in a car, smoothly transitioning between stable numbers without tiny digital oscillations.

The filter will be composed of a circular buffer designed in VHDL, an adder, and a bank of shift registers. Initially, there will be eight registers in the circular buffer and correspondingly an eight-input adder and eight shift registers, as can be seen in Figure 4 below.

![Diagram of velocity averager](image)

**Figure 4.** Proposed design for the velocity averager that latches the last eight velocity values, shifts them by three, and then adds them up which results in an average velocity.

Further testing will determine whether eight points is long enough. A fully general FIR filter can be implemented with a multiplier and divider instead of a shift register, but since this problem only requires the implementation of a device to calculate running averages, a full FIR filter is most likely not necessary. As long as the length of the filter is a power of two and the sampled velocities are equally weighted, the shift register will function as an integer divider.

In operation, the velocity averager will receive a single 32-bit velocity word from the optical encoder and store that word in the circular buffer. The circular buffer will contain as many registers as the length of the running average desired, and will advance one register each time it is written to, overwriting
the oldest value stored in the buffer. Shifters will then shift the velocity words from the buffer as many bits down as necessary and hand the shifted values to the adder to be summed up. The resulting value will be latched in an output register. Shifting must be performed before addition to minimize overflow conditions.

Closed-Loop Control (CLC) Design

The closed-loop control device will be implemented in hardware by combining the optical encoder, velocity averager and velocity controller into one larger IO device. The device will encapsulate all three of these other smaller devices so that the SCOMP only has to communicate with one larger device. A basic diagram of the main connections between components can be seen in Figure 5.

![Composite Device Diagram]

Because it will be a combination of the other devices, each of the other devices can be tested and designed separately. The composite device will accept both read and write commands from the SCOMP. If SCOMP reads from this composite device, the average velocity will be written to the IO bus, but if SCOMP writes to the device, the input velocity will be fed to the CLC controller. The CLC controller will have continuous input from the velocity averager via direct connection and will adjust the motor dynamically to attempt to control the actual speed of rotation rather than just torque. When designing this CLC device, the max wheel speed will be found by running the motor at full speed and reading the max
counts per second from the velocity averager. In this way, the velocity SCOMP input, which is simply in magnitude from 1 to 32767, can be scaled to the correct target velocity in counts per second, which will be from 1 to the max velocity, so the CLC knows when to increase the overall power or decrease the overall power going to the motor.

Internally, the CLC device will be designed as a proportional-integral-derivative (PID) controller. The output gain of the motor can be thus described as a linear combination of the error, the derivative of the error, and the integral over all past errors, where the error is the difference between the current motor output and the desired output. The error can be found by subtracting the reading from the velocity averager from the desired velocity, which is read in from the switches. The derivative of the error is the derivative of the velocity and the integral over the error is similarly the integral of the velocity, or the position. Both the velocity and acceleration are provided by the optical reader. The system should be easy to implement with most of its development time spent on finding constants that cause the system to stabilize quickly to the desired value.

**SCOMP Program**

A simple SCOMP program will demonstrate our design and add the functionality to gradually increase or decrease the velocity based on which push button keys are being pressed. For safety reasons, a watchdog timer implemented in the hardware design will make sure that KEY3 is being pressed before allowing the motor to be enabled. If KEY3 is being pressed, the program will read the switches as a sign magnitude number and convert it to a two's complement number and then write this two's complement number to the composite device as a velocity. If KEY1 is also being pressed, the program will gradually increase the velocity written to the composite device. On the other hand, if instead KEY2 is also being pressed, the program will gradually decrease the velocity written to the composite device. In addition, the program will be reading an average velocity value from the composite device, which includes the optical encoder, and outputting this average velocity in counts per second to the seven segment display. These steps will be contained in a continuous loop within the SCOMP program.

**Management Plan**

The design will be executed according to the Gantt Chart provided in Appendix B.

**Contingency Plans**

In the event that time becomes a limiting factor and the team is unable to complete all aforementioned tasks, the velocity controller, basic SCOMP program, and velocity averager will
be completed. The CLC composite device is mainly a combination of these devices, so it can be implemented after these fundamental items are completed. In this way, the main requirements can be met first and additions can be added afterwards. Also, if designing the CLC mechanism in hardware becomes too difficult, a SCOMP-based program can be implemented that will read the velocity from the velocity averager and use its output to adjust the velocity being written to the velocity controller. The added feature of the SCOMP program which can gradually increase or decrease the velocity will be implemented after the basic SCOMP program that fulfills the requirements is completed.