Design Document for the DC motor controller and driving simulator

Alexandre Msellati (am5692), Ayman Aslam Talkani (aat2193)

31 March 2022

Contents

1 Introduction

We want to build a DC motor controller that is also linked to a VGA display so that to simulate a first-person driving car, where the speed of the real DC motor and the simulated car can be controlled by a (physical) potentiometer. In order to achieve this, we will need to approximate the model of the DC motor and to build a PI controller based on its features. We will implement Hardware as well as Software for the DC motor controller,while also feeding the calculated speed of the DC motor to our VGA-display code, which will alter the speed at which the sprites move accordingly. We will also try to have certain messages pop to instruct the driver on how to optimally drive in order to conserve energy, such as infrequent changes in speed.

We have divided the work amongst the two team members, such that Alexandre will primarily focus on the implementation the motor controller, while Ayman will work on the driving simulator. We will also try to assist each other if required, and finally work on the integration of these two components.

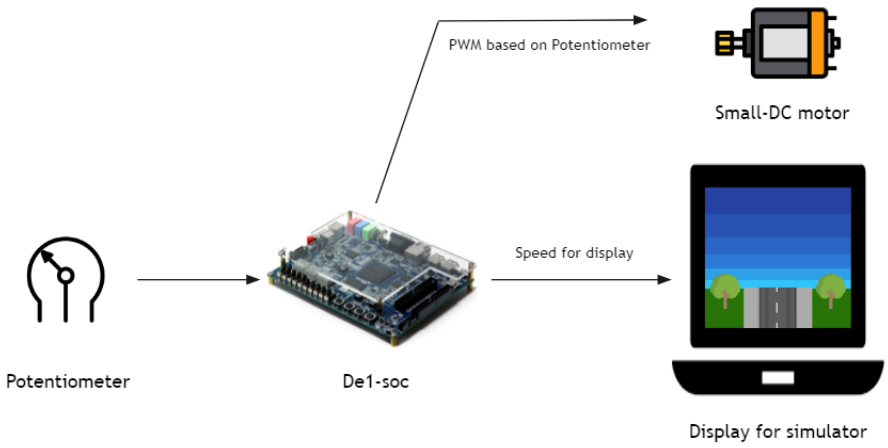

Figure 1: Overview of application. Note that display graphics shown may change

2 System Block Diagram

The DC motor controller will embed Hardware that is described figure 2. The ADC of the FPGA will be used to retrieve the signal of the armature voltage of the actual DC motor and the voltage from the dimmer. The sample rate will be of 50 MHz, the clock the of the FPGA. These signals from the ADC are then processed to give the speed reference of the user, ω_{ref}, and the effective speed of the motor, ω. These signals are then processed by the Motor controller module, that is connected to the avalon bus to send to and receive data from the software. Basically, you want to be able to change the parameters of the motor through software so that you do not have to always issue a new hardware just for changing the parameters. We could also implement the control laws in software and just give values to the controller. The motor controller output, V_{ref} is the control input voltage for the PWM, which will issue a signal V_o with a varying DC value. The signal V_o will be a square wave.

It should be noted that a voltage divider will probably be used on the ADC to sample the voltage input of the dimmer.

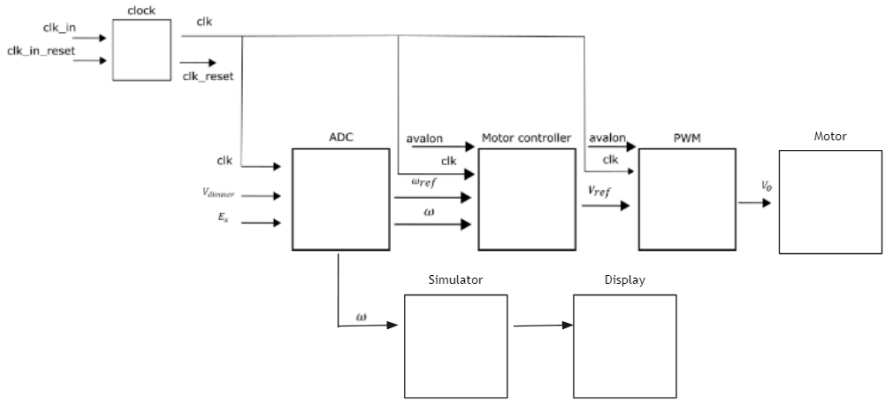

Figure 2: Block diagram of the hardware for the DC motor controller

As seen in this block diagram, we can also separately send the speed w of the motor to the simulator files. Our simulator will essentially display a moving scenery and a road on the VGA display. This will display the sprites of trees and other objects on the screen, which will be moving according to the speed of the motor, thus giving the impression that as the motor speed increases, the driver in the simulator is going at a faster speed. We may also include a text box at the bottom of the screen, which will show messages/suggestions from our simulator on how the driver could improve their driving, such as reducing the frequency of braking, keeping speed consistent if possible, etc.

3 Algorithms

All the computations are based on integers. The following table 1 is used to give the unit of values:

Time	us
Voltage	mV
Current	mA
Speed	rad/s

Table 1: Caption

3.1 DC motor controller

Basically, the role of the DC motor controller is to solve differential equations in order to determine the control input. Before that, the ADC will compute speed of reference in the following way:

$$\frac{V_{dimmer}}{V_{max}} = \frac{\omega_{ref}}{\omega_{max}} \tag{1}$$

Moreover, the speed of the motor can be estimated by:

$$\omega = \frac{E_a}{K\phi} \tag{2}$$

This would be directly done in hardware. The motor controller will be a proportional-integral controller that has yet to be defined:

$$G_c = K_p + \frac{K_I}{sT_I} \tag{3}$$

The choice of the DC motor will guide the selection of the parameters for the controller. The differential equations will be solved using Euler's explicit scheme: by using small time steps, Δt, we can approximate the derivative of a function:

$$\frac{d\omega}{dt}(t_k) = f(t_k) \Longrightarrow \frac{\omega(t_k)}{\Delta t} = f(t_k) \quad \text{where} \quad \Delta t = t_k - t_{k-1} \tag{4}$$

Since this will be implemented through hardware, Δt is the period of the clock or maybe a multiple of it. Finally, the reference voltage computed by the controller supplies the PWM module, that generates the square wave whose DC value, V_o, equals the reference voltage, V_{ref}. This will be handled by the duty cycle of the PWM. Moreover, V_{ref} will be sent the software to model the speed through the DC motor model and compared with the effective value, eventually both shown on the display. The motor model will be computed by the software, using the block diagram of the DC motor, figure (3). The differential equations will be solved just as the hardware, and the parameters will depend on the choice of the

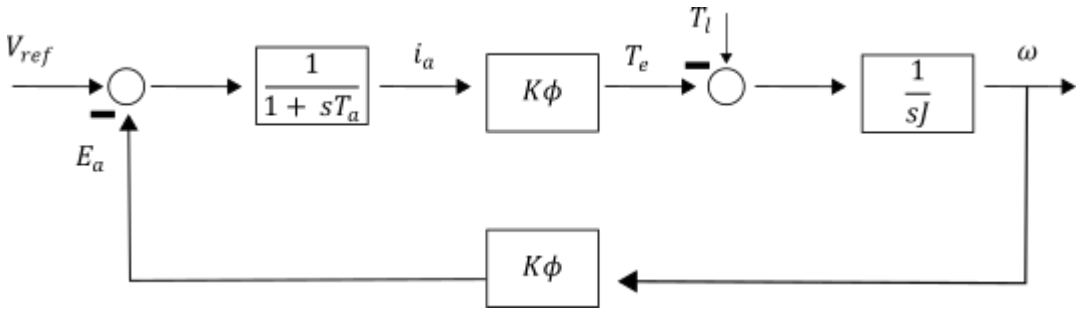

Figure 3: Block diagram of the DC motor model

motor. A MATLAB script might be written first to ensure that the controller actually does its job.

In order not to have to change the hardware, the parameters of the controller and the motor will be defined in the software and sent to the hardware. As a result, we can think of one file main.c, which would work the following way:

- Initialize the parameters and send them to the hardware;
- Check that everything is working:
  - ADC values are fine;
  - the controller sends appropriate values;
  - waveform output of the PWM module is relevant.
- Launch the threads of monitoring and let the hardware do its job.

In the hardware, the user would have to press a button to start the motor, and another one to stop it.

3.2 Simulator

For the simulator, we will be making use of the platform designer in quartus, and assign relevant signals to our new peripheral component. We will then communicate with the peripheral through our simulator software, such that it can communicate with the device driver and send values from our software such as calculated speed of the motor. This will then change the speed at which the driving scenery(seen from a first-person-perspective) moves depending on motor speed.

The x and y coordinates of the scenery would both have a fixed value added to them to move diagonally until they reach screen borders, after which we would add smaller versions of object further away from the driver, thus giving an illusion that the driver is proceeding on a simulated road. We will also have a fixed scaling factor added to the width of the pixel art of the object each

time it moves, thus giving the impression that it will be moving closer. The landscape.c file would serve as a device driver for the VGA video generator, while the sprite-location.c file would send the new coordinates for objects in the scenery based on the calculated speed of the motor. Note that the width of the object pixel-art will also be changed by a scaling factor proportional to the speed.

If time permits, we will also try to implement a text box at the bottom of the VGA screen, which will show us instructions from the driving simulator to improve fuel efficiency, along with other suggestions for improved driving. We have attached an image we prepared to help for visualization purposes below. Note that this is not coded on FPGA yet, and will be attempted by us in a future milestone.

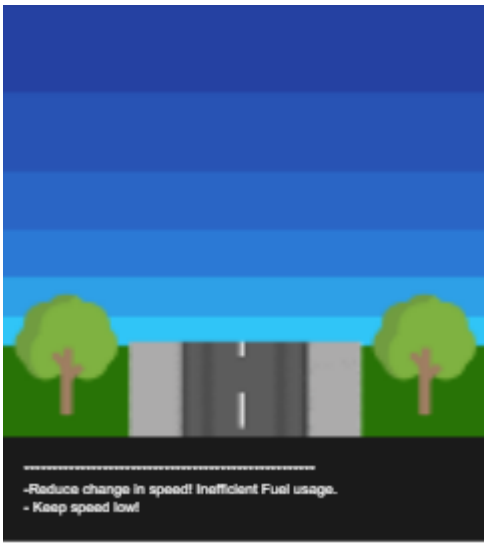

Figure 4: Example of a type of simulator we will be trying to make. Note that this is not an actual simulator on FPGA, just one made for understanding purposes

4 Resource budget

The total RAM of the HPS is 64 Kbyte while the FPGA has 4,450 Kbits (556 Kbyte). This is probably where we will need to optimize the software and maybe do trade-offs between hardware and software (relieve the software of some functions and give it to the hardware). The software and hardware variables are defined in the two tables 2 and 3 below. Some values for the motor controller, as well as the simulator have yet to be selected and will be defined later. Once we decide exactly what to display on our scenery for the simulator, we will include that in resource budget.

Variable	Size (bits)	Range value
ω_{ref}	11	[0, 15,000] RPM $\Longleftrightarrow$ [0, 1570] rad/s
ω	11	[0, 15,000] RPM $\Longleftrightarrow$ [0, 1570] rad/s
V_{ref}	13	[0, 5000] mV
V_o	13	[0, 5000] mV
V_{dimmer}	13	[0, 5000] mV
E_a	13	[0, 5000] mV
K_I		
K_P		
$K\phi$		
Δt		

Table 2: Hardware variables

Variable	Size (bits)	Range value
ω_{ref}	11	[0, 15,000] RPM $\Longleftrightarrow$ [0, 1570] rad/s
ω	11	[0, 15,000] RPM $\Longleftrightarrow$ [0, 1570] rad/s
V_{ref}	13	[0, 5000] mV
V_o	13	[0, 5000] mV
V_{dimmer}	13	[0, 5000] mV
E_a	13	[0, 5000] mV
K_I		
K_P		
$K\phi$		
J		
L_a		
R_a		
T_e		
T_l		
Δt		

Table 3: Software variables

5 The Hardware/Software Interface

The Hardware/Software interface is summarized in the figure 5 below.

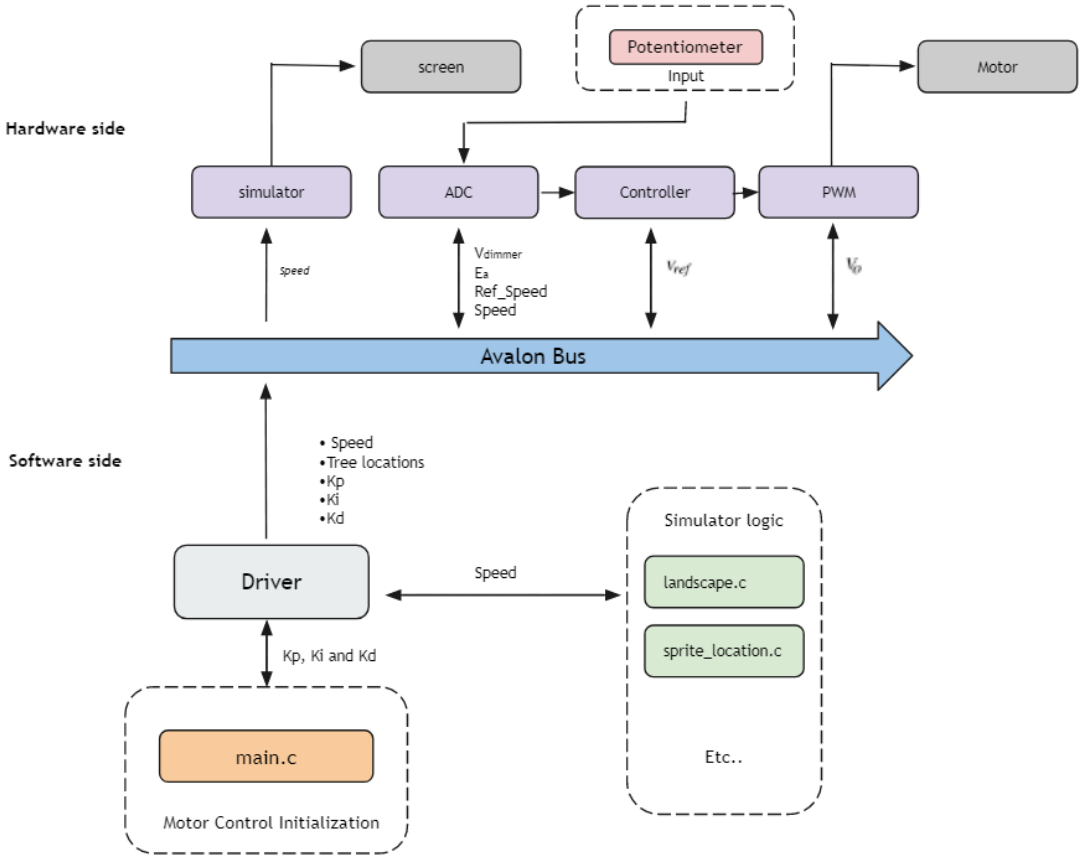

Figure 5: Hardware and Software Interface diagram. Note that this diagram is still in the early stages and might be altered while we are proceeding with the milestones of the project

6 Milestones

Here are the different milestones we have set to achieve this project:

- Milestone 1:
  - Setup of the HW for the controller and the PWM;
  - Display values of interesting parameters for the driver through VGA;
- Milestone 2:
  - Setup of the HW for the ADC with the DC motor and the dimmer;
  - Display a still version of the landscape through VGA;
- Milestone 3:
  - Interaction between HW and SW with the DC motor model built in SW: the motor should be able to turn!
  - The landscape is now moving. Also try to add text box for simulator instructions.