

Project Final Report

Lego Moving Robot

27 April 2007

Sue Vargo, thanyen87@comcast.net

Project Abstract:

This Lego Robot will move forward, backward and then turns in a different direction to continue in a different path. The Zilog Z8 Microcontroller is used instead of the Lego RCX brick to move the Lego build robot. The Pololu Micro Dual Serial Motor Controller is used to drive the Lego motors to move the robot. This particular motor controller is used because it is very easy to use and is build by Pololu to be used with the Lego motors. Pololu will even show you how to build your very own Lego moving robot. The Z8 Microcontroller board offers a wider capability to operate the Lego robot then the old Lego RCX brick. Sure, the new Lego NXT brick offers more range and capability then the old brick, but who has that kind of money? The Pololu build robot is less expensive and more fun to build and operate.

Status

The robot did work as planned. The only change I made was not to add a sensor to the robot because I did not have time. The only thing the sensor will add to the robot is that it is suppose to guide the robot along a certain pathway. I am happy to get the robot to move and, for me, that is a huge accomplishment. I didn't have enough time because I thought I would be able to easily program the robot for various movements and functionality. As it turns out, I had more problems with the hardware then I anticipated and was not able to build in fancy movements to the robot's programming.

Specification

Hardware

Zilog Z8 Encore! MicroController circuit board.

Pololu Micro Dual Serial Motor Controller, part # 0411, \$24.95 + shipping

Lego robot kit with Lego 2 Lego motors (#43362)

Lego RCX compatible electric connector with wire (wires optional)

(For connecting the Lego motors to the motor controller. This item is actually discontinued by Lego)

Or various Lego parts, 4 wheels and 2 Lego motors (#43362)

A 9v battery pack

A breadboard

Various wires

Design Description

A large Lego board is needed to hold the Z8 board. The robot can be built on top of this Lego board, complete with wheel attachments and the two Lego motors. The breadboard will also need to be on the board. The breadboard will allow the motor controller to be connected to the Z8 board. Also leave room on the board to hold the 9v battery pack.

With the Lego robot built and the Z8 board firmly attached to the robot, it is now ready to be wired to the motor controller. Attach the Pololu motor controller to the breadboard. If the motor controller is held such that the components on the motor controller are facing the user, the pins are numbered from 1 through 9 starting on the left side. Pin 1 will be the left most pin and pin 9 will be the right most pin. Connect pin 1 from the motor controller on the breadboard to the VCC pin on the Z8 board. Pin 1 is the power source for the motor controller and the motors. Connect pin 2 on the motor controller to a ground pin on the Z8 board. Connect pin 3 on the motor controller to a VDD pin on the Z8 board. Pin 3 is the logic source for the motor controller. The motor controller's pin 4 is its serial input line. Pin 4 will allow the Z8 board to communicate with the motor controller via the serial uart port. Connect pin 4 to one of the serial pin, either pin A5 for TXD0 or pin D5 for TXD1. Pin A5 for TXD0 was used for this project. Pin 5 is used to reset the motor to its original setting, which is all off and waiting for commands. Connect pin 5 of the motor controller to any general purpose IO pin on the Z8 board. For this project, pin A2 on the Z8 was used, mainly because it has no alternate function which could be inadvertently turned on.

Figure 1 shows an illustration of the pins on the motor controller and Figure 2 shows the connection diagram.

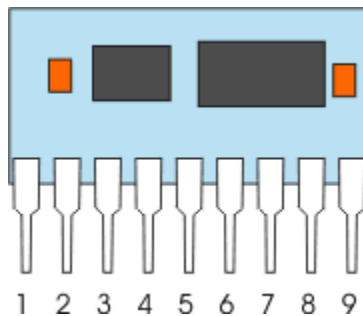


Figure1: Pin configuration on the motor controller

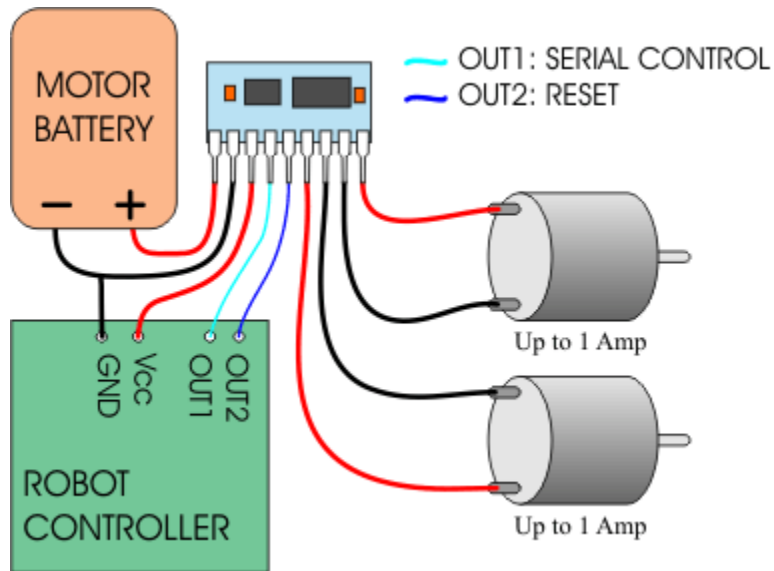


Figure 2: Wire connections

Since the Lego motors do not have pins, as shown in the picture below, it is necessary to use other attachments to connect to Lego motor to non-Lego parts.



Figure 3: Lego 43362 motor

The Lego RCX compatible electric connector (Figure 4) is used to connect the Lego motors to the motor controller. The connector is a square with 4 holes to fit on top of the black area of the Lego motor. The black area on the motor has plates for connectivity. The holes are wired to 2 pins on the connector. The connector can come with its own wire that has female connector at both end of the wire. These wires are not used for this project. Connect pin 6 and pin 7 of the motor controller to the 2 pins on one motor. Connect pin 8 and pin 9 to the 2 pins on the other motor. The pins on the motor controller have positive and negative sides, but the Lego motors do not have positive and negative sides. If the motors do not rotate in the expected direction of choice, then switch the wires on the motors, or reverse the direction in the software to get the desired direction.

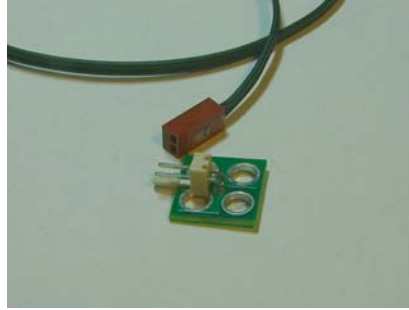


Figure 4: Lego RCX compatible electric connector

The squared connector comes at a set dimension. It may not fit firmly on top of the black area on the motor. I shave one side of the square to make it fit on the motor. This does take time and a lot of elbow grease. An alternate way to connect the square connector to the motor is to use the standard Lego connector, shown in Figure 5 below. The Lego connector has connecting plates on the two ends of the wire. The plates are cross-wired so that any connection orientation will work. Connect one side of the Lego connector to the motor. Connect the green square connector with pins to the other side of the Lego connector. Connect the wires from the pin of the connector to the pins on the motor controller as described previously. This prevents from having to make the square connector fit on the Lego motors.



Figure 5: Standard Lego connector

Software

The software used for this project is the Zilog ZDS II Encore!, version 4.10.0. A brand new motor controller needed to be configured before being used. It can be reconfigured to fit different design if the design of the project changed. Although the motor controller can be reconfigured thousands of times, it is not recommended that the user reconfigured the motor controller unless it needs to be reconfigured. The configuration for the motor controller is stored in a small area of rom and space is limited. The new configuration does not overwrite the old configuration. Each new configuration is stored in rom. Thus the more the user reconfigures the motor controller, the less rom is available for the next reconfiguration.

For my project, a separate program is written to configure the motor controller. The configuration program simply resets pin 5 and waits 100 ms before sending commands to the motor controller. For this project, the configuration of the motor controller is for 2 motors and the motors are numbered 0 and 1. The commands to reconfigure are, in this order, a start command, a configuration command and the motor setting command. The start command is always the byte 0x80. The configuration command is the byte 0x02. The setting command is a single byte specified in Figure 6 below, with the left most bit being the most significant bit.

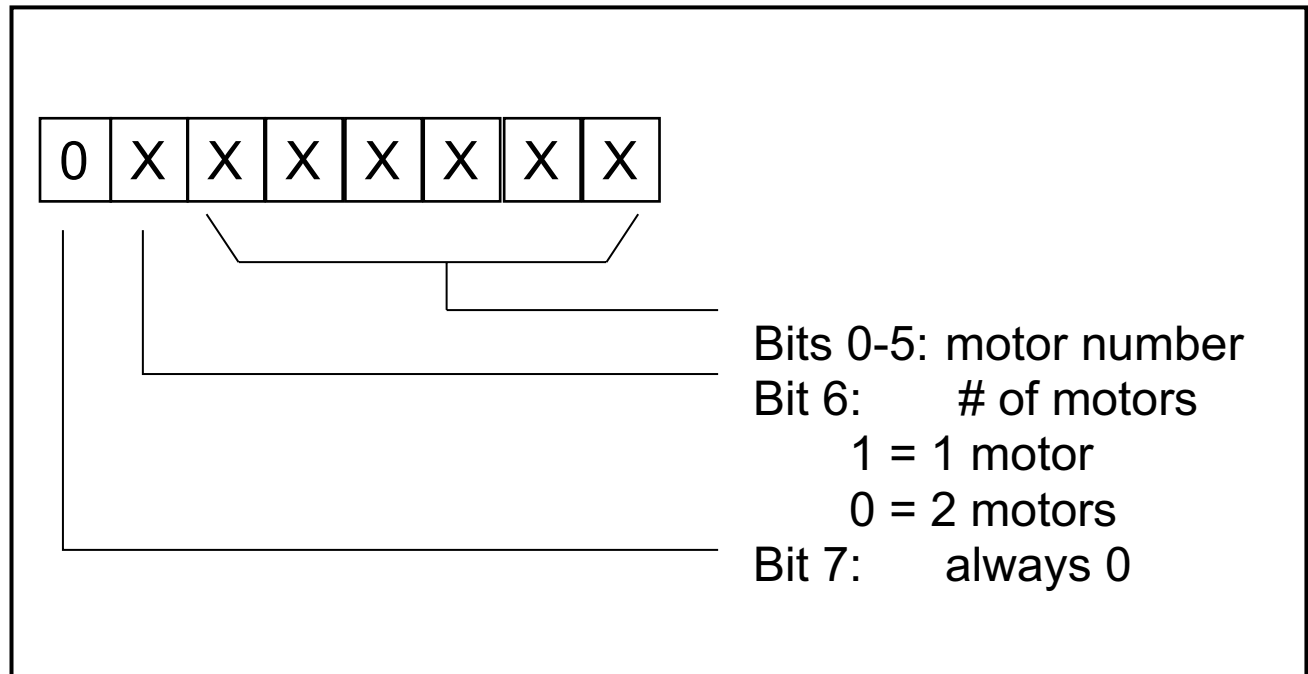


Figure 6: The configuration command for reconfiguring the motor controller.

The program that controls the robot is a separate program. The program starts out waiting 100ms, which is the required time delay before data can be received by the motor controller's serial line, before sending any commands. Each movement of the robot is controlled by 3 commands, the start command, the device type command and the speed and direction command. The start command is always the byte 0x80. The device type is specific to each device Pololu manufactured. For the Micro Dual Serial Motor Controller, the device type is 0x00. The speed and direction command is illustrated in Figure 6 below, with the left most bit being the most significant bit.

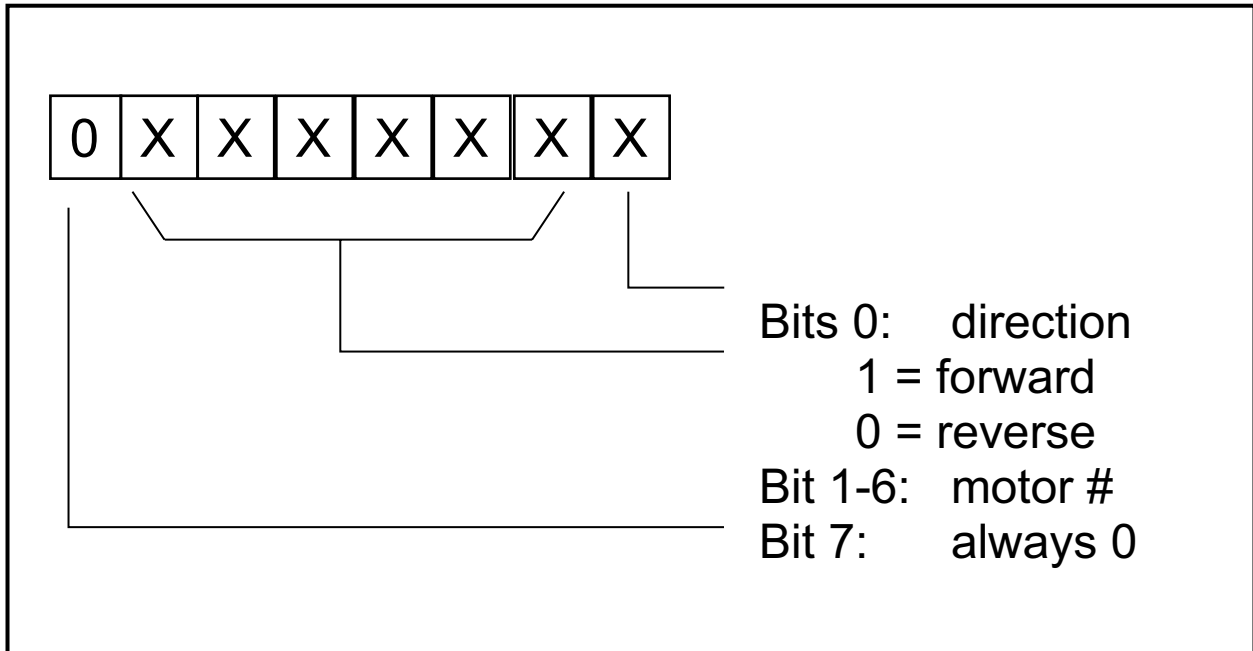


Figure 7: The speed and direction command

With the knowledge of what commands to use, and the motor controller configured for use, the user is now ready to write the software to control the robot. When the robot is first turned on, the motor controller needs a period of 100 milliseconds before being ready to receive commands. One way of doing this is to implement a timer for over 100ms. The motor controller needs at least 100ms so any amount of time over 100ms is sufficient. Set the timer for over 100ms to ensure that the motor controller is ready. This only needs to be run once, at the beginning of the program, and any time the motor controller needs to be reset.

The next function that is useful is the reset function. This is a simple function that takes the general purpose IO pin low for at least 2 microseconds and then back up again. This function resets the motor controller back to its initial states.

For controlling the robot, the serial line needs to function at a setting that matches the setting for the motor controller. The motor controller requires non-inverted, logic-level transmission at baud rates between 1200 and 19200. The bits are transmitted 8 bits at a time with no parity and one stop bit.

The software can be as simple as using the build-in uart initialization function and the transmit function, or as hard as writing your own code to transmit and initialize the uart. I tried to write my own functions to transmit and initialize the uart, but I soon find that this task is much harder than I anticipated. My ending software uses the build-in functions. To use the Z8's build-in function, the library `sio.h` needs to be included with the main program that runs the robot. The uart can be initialized with the function:

```
init_uart(_UART0, _DEFFREQ, 19200ul);
```

The first parameter is the uart, which can be UART0 or UART1, depending on which serial pin the user wishes to use. The second parameter is the default frequency. The third parameter is the baud rate, which should be between 1200ul and 19200ul.

The build-in function for transmitting bits through the serial line is:

```
putch(char ch);
```

The parameter `ch` is the command byte being transmitted to the motor controller.

Now it is just a matter of sending the desire commands to the motor controller. A slight pause included between changes of direction is a good idea to ensure that the motor won't overheat. A simple program to start the motors and slowly increasing and decreasing the speed is illustrated here in pseudo code:

```
Variables:      start = 0x80, device =0x00;
                motor0reverse = 0x0; motor0forward = 0x01;
                motor1reverse = 0x02; motor1forward = 0x03;
```

Start program

Initialize all necessary pins, ports and timers

Timer set to wait for over 100ms

Loop through speed from 0 to 127

```
    Putch(start);
    Putch(device);
    Putch(motor1forward);
    Putch(speed);
    Pause for a few seconds
```

Loop through speed from 127 to 0

```
    Putch(start);
    Putch(device);
    Putch(motor1forward);
    Putch(speed);
    Pause for a few seconds
```

Implementation & Construction

Hardware Diagram

Block diagram of the connections for this project

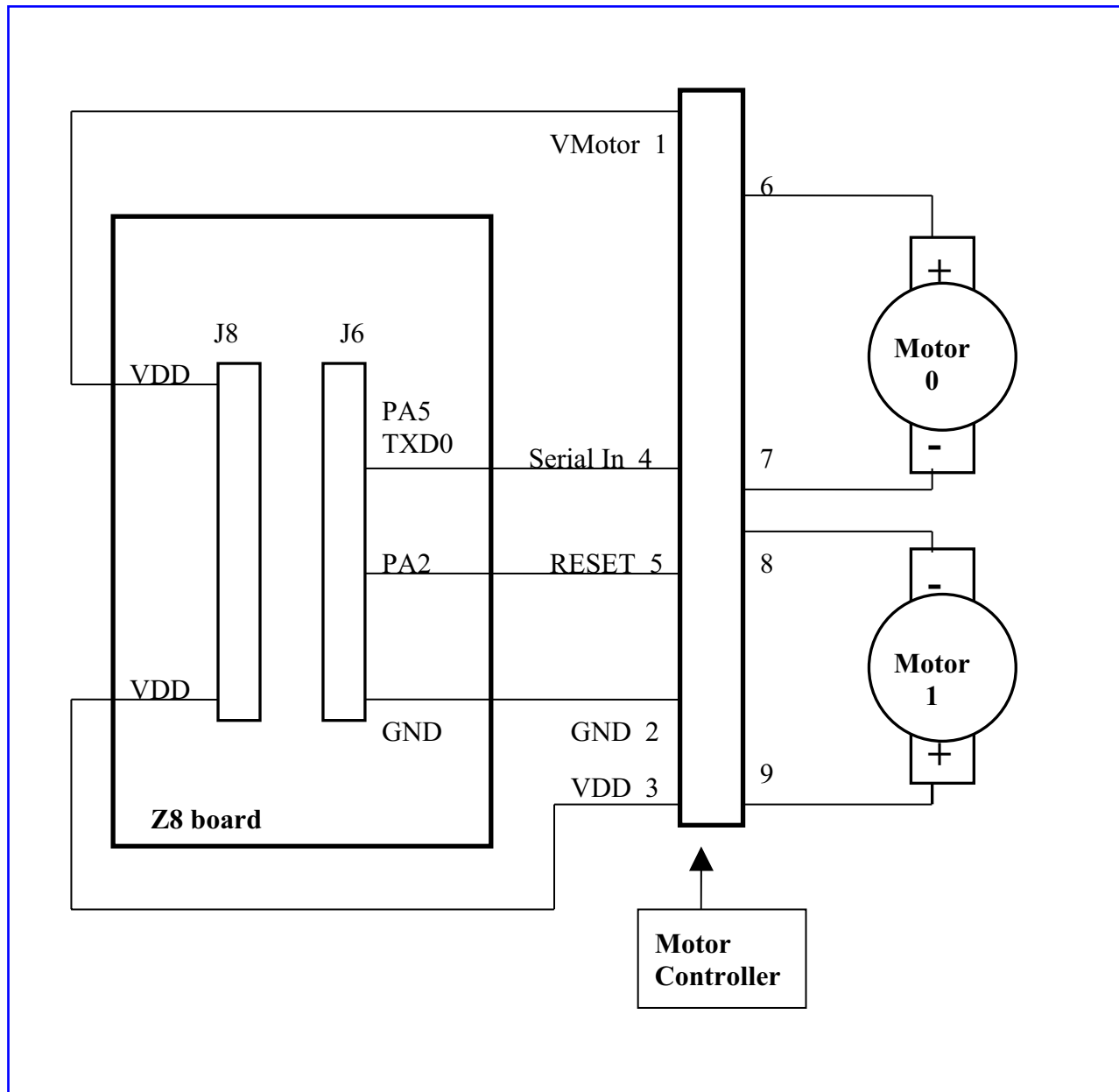


Figure 8: Connection diagram

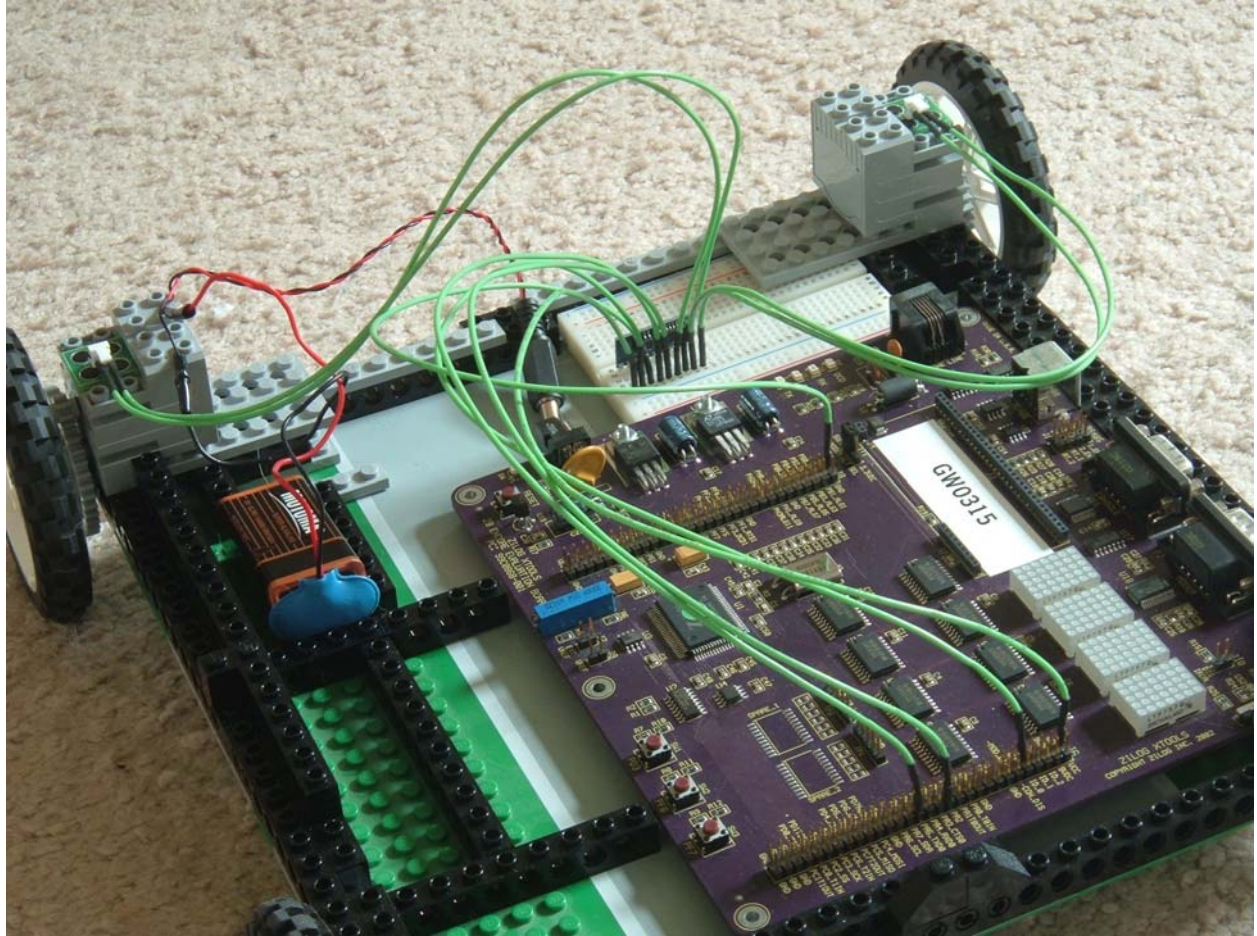


Figure 9: Connection illustration

Software Diagram

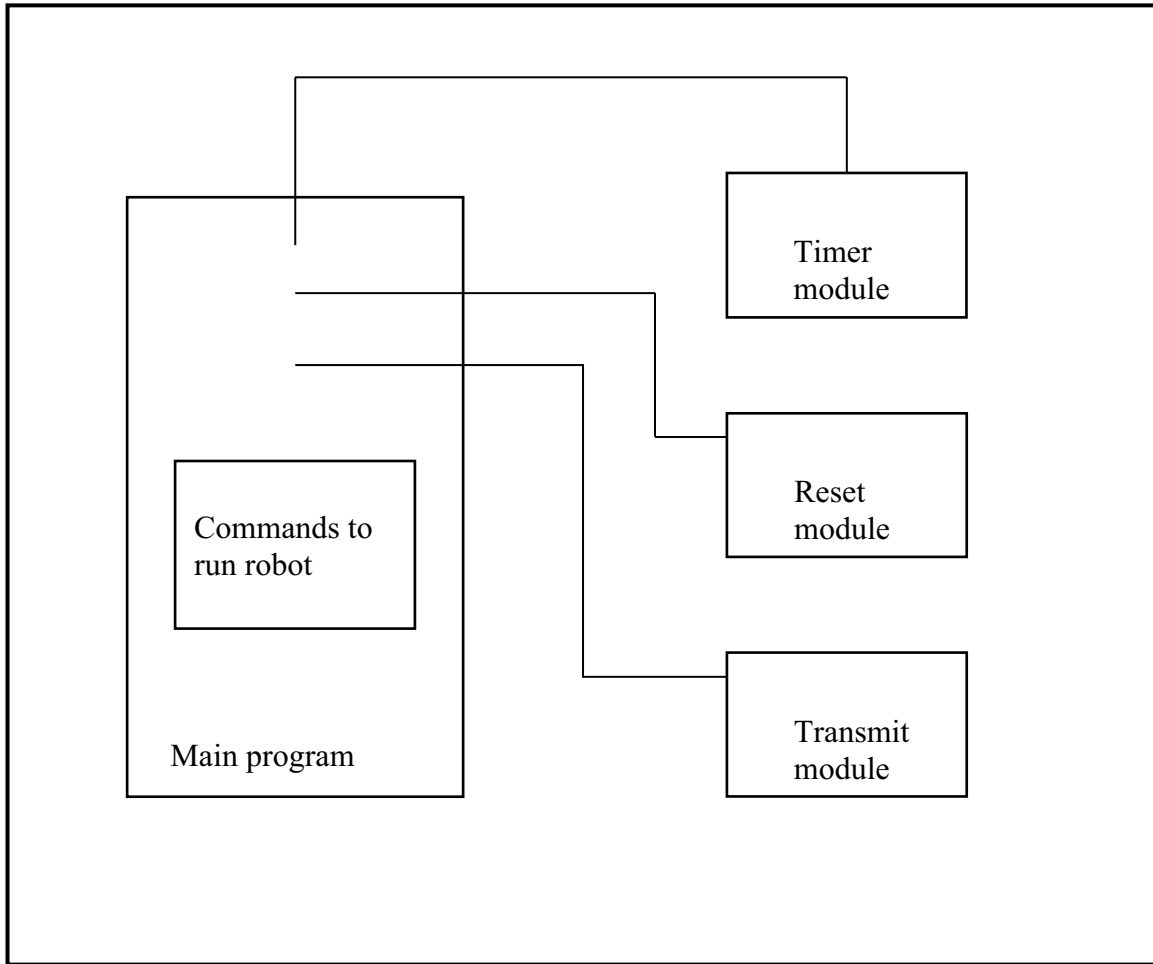


Figure 10: Software block diagram

Restrospective

The main thing I learned from this project is that, for me, it isn't that easy trying to translate everything you learn from class into a real project with no prescribed parameters. It was pretty open ended and how you set up your project depended on what you want at the end. It was hard to read a data sheet or a manual for a device that said what the device needed and then trying to figure out how to implement that on the Z8 board and figuring out which parts to use. There can be numerous ways to implement something and the ease of implementation depends on the design choose.

When I started this project I had all kind of ideas as to what I could do and what direction I wanted to take the project in. As time goes on, I had to scale back what I wanted to do to what I could do. In the end I have a working moving robot, but not one that could do all the things I had imagine it to do when I started the project.

The other hard thing for me was asking the right question, other than "how do you do this" kind of question. I feel that I can learn more trying to figure out my mistakes and asking specific questions rather than general "how to" questions. I won't be learning much if the answers are all thrown at me and I don't have to figure out anything on my own. Because of this, the project took me longer to put together than anticipated, but in the end I did learn a lot.

Attachments

A program for the configuration of the motor controller

A program for running the robot.

A brochure of pictures of the project. This is why pictures are not on this report.