ESET 349 M01
Microcontroller Architecture
Fall 2023
Multidisciplinary Engineering

Final Project

Project Report
DSTR Robotic Car With Forward Obstacle Prevention

Author's Name:
Mark Perez
Roland Alaniz

Lab Due Date: December 4, 2023

Lab Submitted Date: December 4, 2023

IS THIS LAB LATE? No

All of the information contained in this report is my own work that I completed as part of this lab assignment. I have not used results or content from any other sources or students.

Table of Contents

Chapter 1: Introduction

For our final project, we decided to build a car using the DSTR-E kit that would react to a supersonic sensor. The main objective of this project would be to extract information from a sensor that would be sending a voltage detailing the location of an object in front of the car and once the distance is close enough, the code will tell the car to stop moving and stay in place. Once the object is gone, we can continue running. When selecting this project detailed above, we were given two choices, build this project from scratch or use an existing one. We chose to pick the first one because we wanted the experience of building a project from scratch and arranging the circuity the way we wanted it to be. This project is an important one as it is very relevant to the real world. As time continues, we see that cars and car drivers tend to become more reliant on the autopilot feature that some big name car companies are coming up with such as Tesla. These autopilot cars rely on sensors to make small movements such as turns and merges. Although our car is using these ideas of using sensors to make a stop motion, it is nothing like those big name car companies and is just a small scale version of them with only one sensor. Originally there was supposed to be three sensors detailing where it would go, but due to certain issues that will be discussed earlier, we will be using only one sensor.

This project took much longer than we had initially thought. We originally thought that this project would be a couple week project but turned out to be much more than that. When we first started the planning phase we thought of it in multiple phases that consisted of a machine shop phase, build phase, circuitry phase, code phase, and a refining or testing phase.. All of these phases each have their own unique setup and tasks which will be discussed in the hardware part and software parts of this report. These phases complement each other and some of these phases cannot begin without the other such as a build phase cannot happen before a machine shop phase because we need the steel to be changed in order to build the car. This means successful completion of each phase is needed in order to proceed with the project. Also since there are only two of us working on this project, we needed to divide up the work in order to fit time constrictions. This means in one phase one of us worked on one part while the other worked on another part so we could get this project done on time.

This project is the culmination of our entire school year in ESET 349, microcontroller architecture. In this course we examine ARM architecture within the TI MSP432401R microcontroller. We look in depth of how these microcontrollers operate using assembly language to code said microcontroller. These microcontrollers are considered memory mapped I/O microcontrollers which means the inputs and outputs are directly related to the memory and location spots in the memory. This would also mean we would need to know the addresses of all the pins we decide to use for our project and set them up accordingly. We would also need to extract and store information in order to make our inputs and outputs work as intended. All of these concepts will be discussed in more detail when we discuss the software portion of this report.

Chapter 2: Description of System Design

In this portion of this report, we will be discussing the system and the phases, mentioned earlier, to our car. As we mentioned before we had five main phases, machine shop, build, circuitry, code, and refine phases, and each of these belong in their own section of this report. The machine shop, build, and circuitry belong in the hardware section of this report while the code phase belongs in the software section and the refine phase belongs in the testing section of this report. In this section, we will only be focusing on the hardware and the software portion of our project

Section 2.1: Hardware System Design

When it comes to the hardware, there was a lot that needed to be done. When designing our robot, with the instructions given from the kit we realized that the metal bars, rods, and plates needed to be machined for the construction of our build. These parts needed to have precise holes, threading and cuts to ensure proper building of the car. In order to get these holes, threads, and cuts done, we went to the machine shop. If one part is improperly built, it can result in a bad build for the car.

When the machining is done and done correctly it can result in the build being successfully put together. The build is the most important part of the car because everything is reliant on the way this car is built. Where we place our components completely relies on the body of the car. This means that the car's body is vital to the success of the project.

When connecting all the components together, we will be able to create the circuitry for the car. Figure 1 below shows the block diagram for our car.

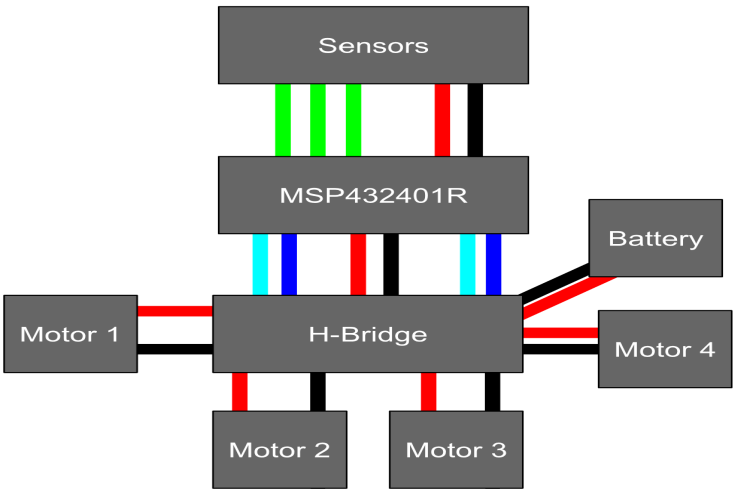

Figure 2.1: block diagram for our circuit

The block diagram above shows four motors being controlled by an H-bridge. The H bridge gets its power from a battery pack. The H bridge then sends power to the microcontroller which controls the speed of the motors through the H-bridge. The supersonic sensor sends voltage levels to the microcontroller in order to tell whether or not there is an obstacle ahead.

Essentially the battery powers everything in the circuit. The H-bridge is the driver of the motors. Once the signal within the microcontroller sends a one to the H-bridge, the motors will move allowing for motion but once it is a zero, the motors will stop. What dictates the signal being a one or a zero is all dependent on the sensors. If the sensors send out a certain voltage input, in our case 05V, stop the car, but if it is lower then keep it moving.

When creating our project, a lot of issues had occurred such as loss of time. We were given four lab days to create this project. When we did the machining of the steel parts, we ended up taking an entire lab day instead of the 40 minutes we expected. Due to this, we lost precious time we could have used for the coding section.

Section 2.2: Software System Design

In this section, we will be discussing the software portion of the system behind our design for our robotic motor vehicle. In terms of our program, we had three main sections, which included the Pulse Width Modulator (PWM), the Analog-to-Digital Converter (ADC), and our conditional statements within a loop to either stop or continue the motors based on the input from the Infrared Sensor. The full program for the software system design of the project can be seen below in Figure 2.1

```
int main(void) {
    int result;

    /* Configure P2.7 as Timer A0.4 output */
    P2->SEL0 |= 0x80;
    P2->SEL1 &= ~0x80;
    P2->DIR |= 0x80;

    /* configure TimerA0.4 as PWM */
    TIMER_A0->CCR[0] = 50000-1;     /* PWM Period */
    TIMER_A0->CCR[4] = 50000/7;     /* CCR4 PWM Around 15% Duty Cycle */
    TIMER_A0->CCTL[4] = 0xE0;       /* CCR4 reset/set mode */
    TIMER_A0->CTL = 0x0214;         /* use SMCLK, count up, clear TA0R register */


    P6->DIR |= 16;

    ADC14->CTL0 =  0x00000010;    /* power on and disabled during configuration */
    ADC14->CTL0 |= 0x04080300;    /* S/H pulse mode, sysclk, 32 sample clocks, software trigger */
    ADC14->CTL1 =  0x00000020;    /* 12-bit resolution */
    ADC14->MCTL[5] = 6;           /* A6 input, single-ended, Vref=AVCC */
    P4->SEL1 |= 0x80;             /* Configure P4.7 for A6 */
    P4->SEL0 |= 0x80;
    ADC14->CTL1 |= 0x00050000;    /* convert for mem reg 5 */
    ADC14->CTL0 |= 2;             /* enable ADC after configuration*/
    P6->DIR |= BIT5;
    P6->DIR |= BIT6;

    while (1) {
        ADC14->CTL0 |= 1;         /* start a conversion */
        while (!ADC14->IFGR0);    /* wait till conversion complete */
        result = ADC14->MEM[5];   /* read conversion result */
        if (result > 630 ) {
          P6->OUT &= BIT5;
          P6->OUT &= BIT6;
        }
        else {
          P6->OUT |= BIT5;
          P6->OUT |= BIT6;
        }
```

Figure 2.2: Software Program for ESET 349 Final Project

Our first section of code in our program consists of our Pulse Width Modulator (PWM), which allows us to alter the speed of the motors. As a team, we decided to implement the PWM based on the fact that the speed of the motors without this implementation would have the vehicle moving at too high of a speed for the operation we intended to have. In order to create a PWM using our MSP432 microcontroller, we used Timer A on the microcontroller and configured the timer to act as our pulse width modulator. After testing solely our PWM code on the motor, we found that a duty cycle from around 15% to 20% resulted in the robot traveling at a fair speed that would allow it to sense objects from our specified distance. The portion of our program that is responsible for our PWM is shown below in Figure 2.2.

```
/* Configure P2.7 as Timer A0.4 output */
P2->SEL0 |= 0x80;
P2->SEL1 &= ~0x80;
P2->DIR |= 0x80;

/* configure TimerA0.4 as PWM */
TIMER_A0->CCR[0] = 50000-1;     /* PWM Period */
TIMER_A0->CCR[4] = 50000/7;     /* CCR4 PWM Around 15% Duty Cycle */
TIMER_A0->CCTL[4] = 0xE0;       /* CCR4 reset/set mode */
TIMER_A0->CTL = 0x0214;         /* use SMCLK, count up, clear TA0R register */
```

Figure 2.3: Pulse Width Modulator (PWM) Program

Our second section of code in the program consisted of the Analog-to-Digital Converter (ADC), which allowed us to convert the analog signal from the Diligent Infrared Sensor to a Digital signal, which we were able to use within our program to allow the vehicle to either stop or continue based on whether or not it encountered an obstacle in its way. In order to do this, we had to configure the ADC on our microcontroller, in which we first had to power it on and disable it, then configure the resolution and its reference voltage level as well as the port for the input analog signal, which comes from our Infrared Sensor. Once all of the configuration has been set for the ADC, we are then able to enable the ADC after the configuration has been completed. The portion of the program responsible for the configuration of the ADC can be seen below in Figure 2.3.

```
ADC14->CTL0 =  0x00000010;    /* power on and disabled during configuration */
ADC14->CTL0 |= 0x04080300;    /* S/H pulse mode, sysclk, 32 sample clocks, software trigger */
ADC14->CTL1 =  0x00000020;    /* 12-bit resolution */
ADC14->MCTL[5] = 6;           /* A6 input, single-ended, Vref=AVCC */
P4->SEL1 |= 0x80;             /* Configure P4.7 for A6 */
P4->SEL0 |= 0x80;
ADC14->CTL1 |= 0x00050000;    /* convert for mem reg 5 */
ADC14->CTL0 |= 2;             /* enable ADC after configuration*/
```

Figure 2.4: Analog-to-Digital Converter Program

Our final section of code allows for the robot to stop or continue in motion based on the digital signal that is being outputted from our ADC. This portion of the program consists of an infinite while loop that continuously takes readings from the sensor and converts those analog readings to a digital signal. After the reading has been taken, we enter into our conditional if-else statement, which states that if the resulting value from our ADC is greater than 630 then the

motors would shut off thus stopping the robotic car at that position, and if the values is less than 630 then the motors on the robotic car would continue at the rate given by the PWM. This portion of the program can be seen below in Figure 2.4.

```
        while (1) {
            ADC14->CTL0 |= 1;          /* start a conversion */
            while (!ADC14->IFGR0);     /* wait till conversion complete */
            result = ADC14->MEM[5];    /* read conversion result */
            if (result > 630 ) {
              P6->OUT &= BIT5;
              P6->OUT &= BIT6;
            }
            else {
              P6->OUT |= BIT5;
              P6->OUT |= BIT6;
            }
        }
    }
```

Figure 2.5: ADC Reading and Conditional Program

Finally, we will be discussing some of the hardships we endured when designing the software portion of the project. The biggest challenge was determining the correct memory mapped addresses for both the ADC as well as Timer_A. After doing our research into the MSP432 microcontroller, we were able to find the documentation that contained the Bases Addresses for these two components that would allow us to configure them in our program, but were unable to run the program successfully. Due to this problem, we had to shift our focus from programming in assembly language to programming our project in C Language since we were able to obtain programs that were functioning correctly when being tested against the hardware.

Chapter 3: Test Plan & Results

In this portion of the report, we will be discussing how we as a team planned to test both the hardware and software components as well as the final design as a whole once hardware and software had been combined in order to create a fully functional robotic vehicle with IR sensor input. Along with this, we will also be discussing the final results of our robotic vehicle based on the initial plans we had made and how it compared to what we were able to achieve.

Section 3.1: Testing of Hardware Components

In order to test our hardware components, which included the full build of the robotic vehicle as well as the vehicle with the microcontroller and H-Bridge, we tested the vehicle itself to ensure that all wiring was correct. Due to the fact that the motors had to be soldered, we tested each individual motor after the wires had been soldered to ensure that the connection was correct and that it would work once the wiring was placed inside the metal components. Once the motor vehicle and its components were constructed, we tested the motors connections with the H-Bridge in order to determine the functionality as well as which pins control the direction of the motors. The last portion of testing in terms of the hardware components came when we had to

ensure that all connections were correctly made from the microcontroller to the H-Bridge, and the H-Bridge to the individual motors.

One small thing that needs to be added when discussing the sensors is that when we tried to test the sensors with operational amplifiers, for some reason these sensors did not work as expected and gave some weird results. When we expected to get a one, we got a zero, and we couldn't figure out why the op-amp was not working. We tried fixing multiple things but in the end, we could not figure it outDue to this confusion, we abandoned the op-amp idea in favor of the ADC due to it not being as complex and easier to set up.. This would also cause later issues that will be discussed in the next section of this report.

Section 3.2: Testing of Software Components

In order to test our software components, which consisted of the program with our PWM, ADC, and Conditional Statement, we tested each portion individually in order to determine whether or not each software component was functional and then tested all the software together. In order to test the PWM, we, first, completed the program for the PWM and took note of the duty cycle and period we had configured Timer A to have. Our microcontroller was then connected to the oscilloscope where we were able to view the PWM signal to ensure that the program was running correctly. In order to test our program for the ADC, we connected our microcontroller to the breadboard that had a potentiometer connected and would allow us to simulate different voltages going through the ADC in order to view the ADC output values. We also tested the voltage level that the IR sensor would output at a certain distance in order to log that as our value for our conditional.

One failure here is that when we did the ADC and the timer the addresses were not properly done, therefore when we did our testing on each one individually, it failed. We looked at datasheets and eventually could not find out why. We thought we had the right addresses and due to time restrictions we ultimately decided to swap to C language to get this project to work. properly

Section 3.3: Testing of Overall System

Once we had tested both the hardware and software components individually, we were able to combine both sets of components and test the overall system performance, which we did by loading our program onto the microcontroller that was fastened to the top of our vehicle. We, then, flipped the switch to power the vehicle with the battery pack on the bottom of the vehicle, and placed the vehicle on the floor to ensure that the speed coming from the PWM was having the vehicle move reasonable given the load of the components the vehicle was carrying as well as determining whether or not the vehicle was stopping at a reasonable distance using the ADC and the conditional statement in our program. We also tested our conditional statement in order

to determine whether we had to AND or OR the outputs based on the input from the sensor, which had to do with the fact that one of our bits was sending a high.

Section 3.4: Results

Once the testing of the overall system was completed, we were left with a completely functional robotic vehicle that would use the Infrared Sensor fixed to the front of the vehicle. When the program was loaded onto the microcontroller and placed on the floor, the robotic vehicle would move forward until it was faced with an obstacle in its way, which would then send a signal to the motors that would shut them off. Below in Figure 3.1 is an image of the final design once the vehicle had been tested.

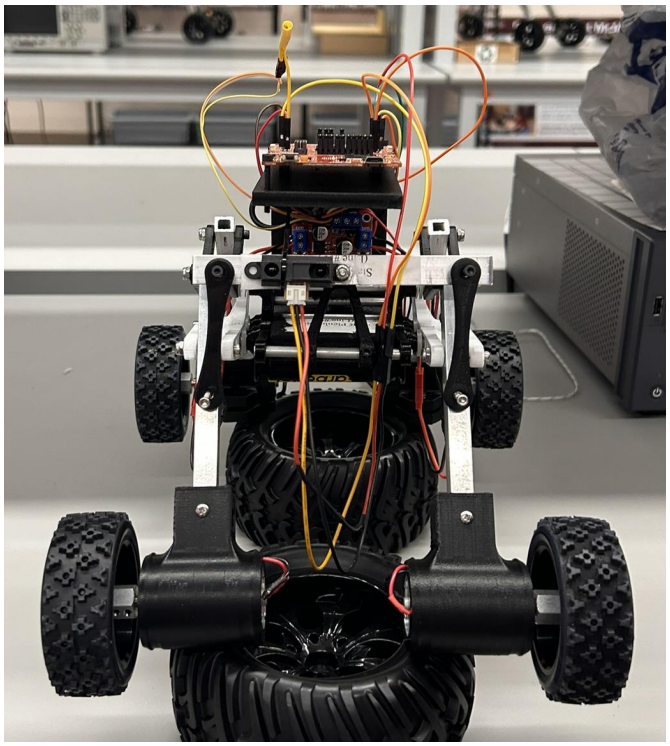

Figure 3.1: Final Completed Design of Robotic Vehicle

Chapter 4: Description of System Operations

In this particular section, we will be discussing a brief description of the system operations for our robotic vehicle. Specifically, we will be referring to the set-up requirements for our robotic vehicle as well as operating instructions and overall how the robot will operate once the program is run.

Section 4.1: Set-Up Requirements

In terms of the set-up requirements for our robotic vehicle, the only requirements for the device is that the program would need to initially be uploaded to the microcontroller that is fastened to the top of the robotic vehicle. To do this, you would first need to remove the orange wire connection on the microcontroller, which supplies the microcontroller with power when the vehicle is in motion, and it needs to be removed since we are powering the board with our computer via USB when uploading the program to the microcontroller. This is done in order to protect the microcontroller board as to not damage it by supplying too much voltage. Once the program has been successfully loaded onto the microcontroller, we would remove our cable connected to the microcontroller and plug the orange wire back into its corresponding pin, and the setup for the robotic motor vehicle is complete.

Section 4.2: Operating Instructions & Overall Operation

In terms of the operating instructions for our vehicle, the vehicle is relatively simple and easy to operate. Once the initial setup has been completed, which as stated above including loading the program onto the microcontroller, the use of the robotic vehicle would then place robot on the ground and flip the switch power switch located on the top of the vehicle near the rear end, which would turn all the vehicle's components on. Once the switch has been flipped to the on position, the vehicle would be fully functional and would move in the forward direction until it is faced with an obstacle at a particular distance and at that point would stop. The user could then pick the vehicle up and choose another direction for the vehicle to travel, and the vehicle will stop when it encounters another obstacle.

Chapter 5: Conclusion

When discussing our project there is some stuff that can be done to significantly improve our design. The main one being adding more sensors like how we originally thought. Adding more sensors such as a left or a right sensor could allow for much more creativity and complexity such as left turns, right turns, and along with reverse states. This concept would add a bit of hardware but overall it would not change the main components. On top of this, the code will be changed significantly as we would need to add the fact for two more sensors and how the turns will work. Overall, the addition of more sensors would greatly impact our design for the better.

In summary, we built a project that reacts to one sensor which will tell the car to stop when too close to an object. This design is built by both the hardware and the software working together. There are five main components to the hardware, the sensor, microcontroller, H-bridge, battery and motors and each have their own unique task such as detects objects, controls the system, the driver to the motors, delivers power to all components and the motors cause the turning of the wheels respectively. Then there is the software which utilizes two main concepts. These concepts are timers and an ADC. The ADC is what allows us to read the sensor and get a

certain value out that can tell us how far away an object is. The timer is there to synchronize everything and dictate the speed of the H-bridge.

When testing we encountered many issues. One of the main issues we encountered was the addresses. Addresses on both the timers and the ADC were difficult to get so in order to solve this issue, we decided to work in C language. C language would be noticeably easier than to use assembly language. On top of this, the sensor that we had used did not work on one of the microcontrollers so we had to use another microcontroller to test it. Overall, there were some issues we had to deal with regarding this project.

Lastly, we gained a lot of this project. We used the things we had learned throughout the course to create a robot to function as intended. One of the main things we gained is that creating projects and dealing with robots does not always go as planned. Engineers fail and take a lot more time than expected. This project was initially thought to take only a week but took us much more than that. We thought the coding would be easy but we had to simplify it by changing languages. Overall, it was a good learning experience for us as we prepare for the engineering world.

Chapter 6: Experience

To conclude our final project for ESET 349, we would like to highlight the overall experience that this project has allowed us to obtain. We will not only highlight the positive attributes that this project has allowed us to obtain, but also the negative ones that we persevered through in order to complete our project and have a working product.

Some of the positive attributes that we obtained from this project mainly consisted of knowing that you can always find a solution to your problems that you are facing in order to complete a task. We faced many challenges during the entirety of this project, but each time we faced a problem, we found a new way to go about finishing our final project. Some examples of us working around some of our problems included changing our programming language from assembly language to C, switching from an Op-Amp to ADC for our sensor input, or decreasing the amount of sensors on our project from having three sensors to now only having one sensor. Another aspect of our project that can be highlighted as positive includes the overall experience that this project has given us as engineers. Engineers that are currently in the field deal with the issues that we encountered in our project everyday, so we are gaining insight as to how to fix these issues now that we are still learning in college. This project also allowed us to create a solution to an issue that mirrors what our cars currently have on a smaller scale. Personally, my car is equipped with forward safety that alerts the driver when a car is driving away from the vehicle and our robotic vehicle practically does the same thing.

Some of the more negative attributes of our experience with this project mainly had to do with the amount of challenges and hardships we faced when trying to develop our final product.

Although we were able to find solutions to many of the problems we faced, the fact that we encountered so many challenges does impact your mindset on the project as a whole. There were many times where we felt lost and thought we were not going to be able to finish our project, but through the help from our professor and each other we were able to finish our project with some level of functionality to it.

Chapter 7: References

Sensor datasheet : Microsoft Word - IR Range Sensor rm.doc (mouser.com)
Motor datasheet : pl36675354-aslongdcmotor_com.jpg (610×800) (maoyt.com)

Chapter 8: Appendix

Below are the data sheets for the sensors

IR Range Sensor Reference Manual

DIGILENT
www.digilentinc.com

Revision: October 31, 2011

1300 NE Henley Court, Suite 3
Pullman, WA 99163
(509) 334 6306 Voice | (509) 334 6300 Fax

Overview

The IR Range Sensor is a great way to add range detection to robotic projects that use Cerebot boards. The sensor detects reflective objects 10 to 80 cm away.

Features include:

- a Sharp 2Y0A21 infrared sensor
- Futaba-style 3-pin connector
- an 8" Digilent cable

Functional Description

The IR Range Sensor is a three-terminal device that generates an analog voltage representing the distance to a reflected object.

The sensor connects to the ADC found on all Cerebot boards. The cable maps the output pins to a 3-pin connector. The yellow wire is for the analog output signal. The pin should be connected to the analog-to-digital converter (ADC) on your microcontroller. The red wire is the Vcc power supply to the sensor and is diode-protected against reverse polarization. The black wire should be connected to the ground of the ADC.

Use a 4.5V to 5.5V power supply for the IR Range Sensor. The sensor's Vcc input range is 0.3V to 7V, and the analog output is 0.4V to 3V, depending on the distance to the object.

Note: When using more than one IR Range Sensor, additional bypass capacitors should be used between Vcc and ground to filter out the switching noise generated by the sensor on the Vcc supply line. See the Sharp 2Y0A21 datasheet for recommended capacitor sizes.

Cerebot Interface

The Cerebot II's ADC analog pins are connected through servo pins S1-S8. Plug the IR Range Sensor directly into the onboard servo header, or use a PmodCON3 to connect up to four sensors via JH pins 1-6 or 7-12.

The Cerebot MX-series boards require a PmodCON3 to access the Futaba-style connectors on the analog ports. The PmodCON3 exposes four ADC analog channels to the connectors. The PmodCON3 can be used on any Pmod header with ADC analog pins.

The following table lists the PmodCON3 header pins used with different Cerebot boards.

Board	PmodCON3 Header	Pins
Cerebot II	JH	1-6 or 7-12
Cerebot 32MX4	JJ	1-6 or 7-12
	JK	1-6
Cerebot 32MX7	JA	1-6 or 7-12
Cerebot MX3cK	JE	8-12
Cerebot MX4cK	JJ	1-6 or 7-12
	JK	7-12
Cerebot MX7cK	JA	1-6 or 7-12

Note: The Cerebot II uses servo connectors S1-S8. The MX3cK PmodCON3 header J2 does not have an ADC pin, so only headers J3-J5 have ADC channels tied to the IR sensors.

Cerebot 32MX4 Sample Project

The following is the hardware configuration for using the IR Range Sensor with a PmodCON3, PmodRS232, and Cerebot 32MX4 board. The code for this project can be downloaded from the IR Range Sensor page at www.digilentinc.com.

The IR sensor is connected to header J2 on the PmodCON3. The PmodCON3 is then connected to the Cerebot 32MX4 using Pmod header JJ pins 1-6 (the top six pins of the header). Pin 1 of the PmodCON3 needs to be connected to pin 1 on the Cerebot 32MX4.

The Cerebot 32MX4 uses a 5V power supply. The Pmod header has the Vcc power selector jumper JPJ set in the VU setting; this connects the Vcc pins on the Pmod header to the boards' external power supply. The PmodCON3 has jumper JP1 set to the Vcc position to draw power from the Vcc pins on the Pmod header.

You could also use the J6 terminal block on the PmodCON3 to supply power to the servo power pins on the PmodCON3.

The PmodRS232 uses Pmod header JE pins 1-6 to send sampled data. Any serial port monitor that displays hex data will work. In the code, the UART is configured to send data at 57600 baud, 8 bits, 1 stop bit with no flow control.

Below are the data sheets for the motor

Voltage		No Load		Load Torque			Stall		Reducer
Workable	Rated	Speed	Current	Speed	Current	Torque	Torque	Current	Ratio
Range	Volt.V	rpm	MA	rpm	A	Kg.cm	Kg.cm	A	1:00
3-9V	6V	1360	≤100	1000	≤0.75	0.2	1	1.3	4.4
3-9V	6V	620	≤100	450	≤0.75	0.4	2	1.3	9.6
3-9V	6V	280	≤100	200	≤0.75	0.8	4.5	1.3	21.3
3-9V	6V	170	≤100	130	≤0.75	1.3	7.5	1.3	35
3-9V	6V	130	≤100	100	≤0.75	1.7	9	1.3	46
3-9V	6V	77	≤100	60	≤0.75	2.8	9	1.3	78
3-9V	6V	58	≤100	45	≤0.75	3.7	9	1.3	103
3-9V	6V	35	≤100	25	≤0.75	6.8	9	1.3	171
3-9V	6V	25	≤100	20	≤0.75	8	9	1.3	226
3-9V	6V	16	≤100	12	≤0.75	9	9	1.3	377
3-9V	6V	12	≤100	9	≤0.75	9	9	1.3	500
6-18V	12V	1360	≤100	1000	≤0.75	0.2	1	1.3	4.4
6-18V	12V	620	≤100	450	≤0.75	0.4	2	1.3	9.6
6-18V	12V	280	≤100	200	≤0.75	0.8	4.5	1.3	21.3
6-18V	12V	170	≤100	130	≤0.75	1.3	7.5	1.3	35
6-18V	12V	130	≤100	100	≤0.75	1.7	9	1.3	46
6-18V	12V	77	≤100	60	≤0.75	2.8	9	1.3	78
6-18V	12V	58	≤100	45	≤0.75	3.7	9	1.3	103
6-18V	12V	35	≤100	25	≤0.75	6.8	9	1.3	171
6-18V	12V	25	≤100	20	≤0.75	8	9	1.3	226
6-18V	12V	16	≤100	12	≤0.75	9	9	1.3	377
6-18V	12V	12	≤100	9	≤0.75	9	9	1.3	500
12-30V	24V	1360	≤60	1000	≤0.3	0.2	1	0.8	4.4
12-30V	24V	620	≤60	450	≤0.3	0.4	2	0.8	9.6
12-30V	24V	280	≤60	200	≤0.3	0.8	4.5	0.8	21.3
12-30V	24V	170	≤60	130	≤0.3	1.3	7.5	0.8	35
12-30V	24V	130	≤60	100	≤0.3	1.7	9	0.8	46
12-30V	24V	77	≤60	60	≤0.3	2.8	9	0.8	78
12-30V	24V	58	≤60	45	≤0.3	3.7	9	0.8	103
12-30V	24V	35	≤60	25	≤0.3	6.8	9	0.8	171
12-30V	24V	25	≤60	20	≤0.3	8	9	0.8	226
12-30V	24V	16	≤60	12	≤0.3	9	9	0.8	377
12-30V	24V	12	≤60	9	≤0.3	9	9	0.8	500