**FPGA CONTROLLED MAZE SOLVING ROBOTIC ARM**

Thomas Edmonds

May 2024

Remember to try and merge github branches so all work is in a single tag

**Abstract**

This project examined the idea of using an FPGA as the main controller for a robotic arm. The idea behind this is to prototype a self-contained arm controller that can quickly respond to its environment with minimal outside control or input.

**Acknowledgements**

Nick & Lab Technicians maybe guy who’s code I took from github

**Contents**

**Background**

Explain all the key mechanics so they don’t have to be explained again later

Q-Learning

FPGA

Inverse Kinematics

HDL

**Glossary**

FPGA: Field Programmable Gate Array

Q-Learning:

inverse kinematics

microcontroller

EMF: electromotive force

PCB: Printed Circuit Board

GPIO: General Purpose Input Output

DOF: Degrees of Freedom

**1 Introduction**

Robotic arms are often used in industry to complete repetitive tasks such as assembling components or packing items into boxes, but what if the box is off centre or the component is upside down? With typical dumb robotic arms, the component would be welded upside down and the item would be put next to the box or in the wrong place.

This project hopes to provide a solution to these problems by creating a prototype robotic arm controller that can react and adapt to its surroundings on the fly with little to no external control. As a proof of concept, the arm should be able to use the colour data from a photo to map a maze, Q-Learning to solve it and inverse kinematics to move through it.

An FPGA is used for three main reasons. Firstly, they are significantly faster than microcontrollers particularly for complex mathematics due to their parallel processing capabilities. Secondly, they are far more flexible, where additional peripherals or microcontrollers may have been required to solve new problems or meet requirements as they appeared, an FPGA can do pretty much anything on its own. Lastly, while this project could have been completed easier and faster using a microcontroller the results wouldn’t have been as good/useful as the controller would have been slower to adapt/respond to its environment.

**2 Project management & Planning**

ASK YOURSELF QUESTIONS why did you do this etc

The majority of this project was to be done during the 4 month period from the 22nd of January to the 2nd of May, with some initial planning done during October/November.

Breakdown of each task,

This project can be broken down into six main tasks:

1. Setting up and testing the robotic arm,
2. Simple control for the arm using the FPGA,
3. Calculating inverse kinematics and sending that data to the arm,
4. Setting up Q-learning on the FPGA,
5. Setting up the camera and sending image data from the microcontroller to the FPGA,
6. Combining all of the above work.

Why I chose the arm (mention problems later) and other componets

During the initial planning period a SainSmart robotic arm was selected from the project store as there wasn’t enough budget to buy a better one and wasn’t enough time to build a new one. Unfortunately the SainSmart wiki where most of the documentation for the arm can be found seems to have gone down and doesn’t seem to be archived on the Wayback Machine See what Paul says to do about this Paul answered CHECK EMAIL

A DEO-CV Cyclone V development board was also booked out of stores for the FPGA as it was the most powerful available and buying one of equivalent or greater power/speed would have taken the project significantly over budget. Add reference and maybe go into detail about specification

It was decided that as the camera and microcontroller or processor? were not going to be used until the final stages of the project and were mostly an extension of the original plan? Specification? that they would be acquired closer to the end of the project when a clearer picture of requirements could be found.

Gannt chart?

As part of preparing for the project this Gannt chart was created to help with project management, unfortunately due to numerous unforeseen delays and extenuating circumstances this original schedule couldn’t be followed.

A screenshot of a computer

Description automatically generated

Risk assessment

A risk assessment was also completed in preparation for the project. Thankfully this project was relatively low risk with no real standout precautions that needed to be taken. Put risk assessment here or link to appendices

If more words/pages needed could mention older plans here (find in older meeting notes/initial proposal)

Rough overview of each meeting

Maybe mention practice fixed point module writing?

**3 Initial arm build**

Once project work fully started in January the first step was to test the arm’s servos and find out what needs to be done to control it with the FPGA, by controlling it with the included Arduino.

The first step was downloading example code and the user guide from the SainSmart wiki. Unfortunately as mentioned earlier, as of the time of writing the SainSmart wiki is no longer available. As such these files have been clearly marked and bundled in the top level of the project GitHub repository.

Unfortunately, it was immediately clear that there was a problem with the original design which was confirmed with the help of the lab technicians. The power supply board included with the arm was not capable of protecting the Arduino from the back-emf generated by the noisy servo motors and as a result two Arduino boards were overloaded before this issue was found. Because of this fault an isolator circuit was required, especially because while replacing the cheap Arduino was easy, replacing the FPGA would have been expensive and probably taken too long. Between finding the required isolators, completing work for other modules, and getting them delivered it took almost a whole month to get these isolators with them arriving on the 22nd of February.

Maybe add small section explaining what magnetic isolators do and how they work? Maybe put this in the glossary/background section?

As a result of the power supply problem described above it was never possible to test the arm with the provided example code for more than a few seconds and as a result all testing had to be done using the FPGA.

**3.1 Basic Testing HDL**

To test the arm and isolation circuit some simple HDL was written using Quartus as can be seen in the appendices [X] or in version 0.01 of the GitHub repository. This code simply outputs different PWM signals to the output GPIO pin depending on which switches are active and works by using a simple timer to enable the output when the value state\_1 is greater than the maximum value minus the angle multiplied by 515 which is the number of microseconds per degree of movement divided by the period of the clock where the maximum value is the 20ms period of the PWM signal converted to microseconds and divided by the period of the clock.

The input angle value was determined by the position of switches 1-4 on the FPGA, each angle was associated with a combination of switches for example 0 degrees was 0000 and 180 degrees was 1111. This was done rather than giving each angle its own switch because if each angle was given its own switch, then if multiple switches were active only the first angle in the else if chain would have been sent.

A computer diagram with many lines

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**3.2 Isolator circuit design/build/testing**

Magnetic isolators were chosen for their low cost and the circuit seen in figure X below was built.

A diagram of a circuit

Description automatically generated

Initially a test circuit was built on a breadboard to confirm that the circuit would work as intended, which was proven when tested first by using a simplified version of the testing HDL written earlier where a switch on the FPGA was connected to an LED, and then using the full testing code descirbed in section 3.1 with an osciliscope as seen in the videos found in the refrences [X]

Initially a test circuit was built on a breadboard to confirm that the circuit would work as intended. This circuit was tested, first by using a simplified version of the testing HDL written earlier where a switch on the FPGA was connected to an LED, and then using the full testing code descirbed in section 3.1 with an osciliscope as seen in the videos found in the refrences [X]

A white circuit board with wires

Description automatically generated

Once this simple circuit had been tested a full circuit with four isolators for the four key servomotors was soldered on a perfboard. Perfboard was used here rather than a PCB as getting a PCB made would have taken a lot of time and planning/testing this isolator circuit had already used up a lot of time and none of the other project work could be tested until it was completed on the 6th of march.

A machine with many wires

Description automatically generated

Unfortunately when testing this circuit with the FPGA it no longer worked, this fault was noticed when the isolator circuit was built on the 7th of march as a result much of the lab time was spent trying to fix it. Initially each soldered wire was connectivity tested using a multimeter however no faults were found. So over the next week finishing on the 12th the HDL code which had been modified to add four separate outputs was tested using testbenches in preperation for the lab session on the 13th.

**3.3 Modified HDL, Testbenches & Further Testing**

As mentioned at the end of section 3.2 a modifed version of the testing HDL was created; as before this modified code can be found either in the appendices or in version 0.0.4 of the GitHub repository. This modified version of the HDL used three extra values which were compared to the same timer as before rather than creating three new timers. This modified HDL also used four extra switches to determine which servo should be moved.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

Once it became clear that there was something wrong with the modified modules two testbenches were written to test the individual modules and then a single complete module was created by combining the HDL from them. This testbench ANGLE\_PWM\_COMBINED\_TB initially tests each motor to confirm that the correct PWM signal is being output, it then tests each angle to make sure the correct output signal is produced for each combination of switches, as can be seen in the waveform screenshot figure [X] below. This result meant that the fault had to be either in the wires connecting the FPGA to the isolator circuit, on the isolator circuit itself or on the physical FPGA hardware.A screenshot of a computer

Description automatically generated

During the lab session on the 13th the connecting wires were connectivity tested to remove that possibility and then to make sure that there was no problem with the FPGA hardware the isolator circuit was tested using the signal generator on an oscilliscope. This revealed that three of the isolator circuits actually worked with the first one having the output wire connected to the output enable pin as seen in figure[X], this mistake was quickly fixed by resoldering the wire. Unfortunately this meant that the only remaining potential causes of the problem were either the FPGA hardware or Quartus not synthesising properly.

A circuit board with wires and wires

Description automatically generated

After many hours of testing the problem was identified as Quartus not correctly assigning pins, this can be seen in figure[X] where the left side of the image shows the previous working versions pin assignments, and the right side shows the new version’s pin assignments. To try and fix this the project supervisor Nicholas Outram was asked for help but as he was out of office at the time he could only provide limited support. Following his advice a separate LED pin was defined and tested this pin was assigned correctly and the LED worked, to get further support a meeting was booked the next day to discuss the issue further.

A screenshot of a computer

Description automatically generated

During the meeting Nick was unable to identify any issues with the project files that could have been causing this problem putting it down to most likely being caused by the changing of versions between the lab where Quartus 20.1.1 is used and home where Quartus 23.1 is used, as a result he was only able to offer the solution of manually assigning each pin which is a work intensive task. Fortunately shortly after the meeting while preparing to start manually assigning pins it was noticed that the compilation log had warnings about an incomplete GPIO bus, as a result of noticing this the missing pins were defined and on the next compilation they were assigned perfectly as can be seen in figure[X].

A screenshot of a computer

Description automatically generated

This was then checked using a Pico Scope and this testing proved that the fix had worked as seen in figure [X]. The FPGA was then connected to the isolation circuit and the arm worked near perfectly.

A screenshot of a computer

Description automatically generated

**3.4 Initial Build Final Touches**

While testing the now working circuit it was noted that when the arm moved it dragged the isolator circuit with it, sometimes even dragging it over the FPGA causing a short. Luckily this never caused any major damage, but it was identified as a significant problem and as such a GPIO header was soldered to perfboard which could then be attached to the FPGA using spacers and a ribbon cable as seen in figure [X]. This was then connected, using long wires with a lot of slack to the isolation circuit which had been attached to the arm’s power supply board again using spacers as seen in figure[X]. This allowed the arm to move freely without dragging around the FPGA or isolator circuit. It was also one of the last pieces of project work completed before the original project supervisor left the university.

A circuit board with wires and a ruler

Description automatically generated A machine with wires on a table

Description automatically generated

**5 Machine learning attempts**

The initial plan for finding the angle values required to control the arm and send the end effector to specific states was to use machine learning to estimate the inverse kinematics values. This plan was based on the coursework for the machine learning module ROCO351 where a simulated method for doing this was used to calculate the inverse kinematics for a simpler 2DOF arm.  
  
The initial concept was fairly simple and with some help from Ian Howard the forward kinematics for the 6DOF arm were quickly found this time using a method based on the coursework for ROCO224 . This work can be seen in the MATLAB files Forward\_Kinematics\_Test\_main, Forward\_Kinematics\_test and TVec on the GitHub repository. These modules create the kinematic matrices and multiply them to find the end effector location, for any number of randomly chosen angle values. When plotted on a scatter diagram these end effector location values seem to show a fairly accurate depiction of the arm’s reach as seen in figure[X] below.

A red circle on a white background

Description automatically generated

Unfortunately adapting the existing algorithm (found in the MATLAB file Main\_P2\_TrainNeuralNetwork2023 in the ROCO351 folder) for a 6DOF arm and three-dimensional movement proved extremely challenging and quite time consuming. As a result after almost a week of struggling, with less than a month left on the project and a failed attempt to create a new algorithm from scratch this part of the project was scrapped.

Don’t know how much to put here, probably just a basic outline of how it would have worked with limited details.

**6 Late arm build**

Building base board

Once the attempts at machine learning were finished it was noticed that unless the project was on a base board the ink from the pen could soak through the paper and mark the desk and also that the required angle values could potentially change if the arm was knocked, or the maze was in the wrong place.

Three pieces of MDF were found and holes were drilled for the FPGA, arm and for wire holders? clamps? Larger holes were then drilled on the back so the nuts could sit flush and the board would sit level, this was important as if the board had wobbled then the arm’s movement could have caused greater inaccuracy in the end effectors position.

Finishing the board took roughly two days due to issues getting the right size drill bit and work on the base board finished on the 24th

A close-up of wires on a white surface

Description automatically generated

Issues with servos

Issues with arm (faulty servo’s etc)

3d printing/design of pen holder.

Issues with pen (too firm/not flexible enough)

Replacing arm

**7 Manual Control HDL**

Instead of using inverse kinematics, a set of HDL modules were created that allowed the servos to be manually controlled using buttons on the FPGA with the current angle output on the seven-segment display. This HDL was written between the 25th and the 27th of April.  
  
First there needed to be a way to see what the current angle value of each servo was so that the angle values for each state could be written down, unfortunately at this point there was less than a week until project showcase and with a lot of work that needed to be done the HDL for the seven segment display was sourced from GitHub ([arnaudeveloper](https://github.com/arnaudeveloper/Display_7seg), 2018) [X] This code simply uses case statements to assign the correct binary value to each output and form specific letters/numbers by lighting up the correct segments on the display. The pins for the seven-segment display and push buttons were also defined on the top-level schematic diagram and were assigned correctly on compilation thanks to the lessons learned in section 3.3.

Next KEY\_INPUT a module to register button presses was written. This module was based on and used many of the same principles as the ANGLE\_OUTPUT\_UNIT module, the main difference being that rather than using pre set angle values it starts with angle values of zero and increments or decrements them with each button press. Thankfully according to the DEO CV user manual [X] , the push buttons are already debounced using a Schmitt trigger circuit which simplifies the module. The push buttons are high by default and pushed low when pressed, so the HDL checks on the positive edge of the clock which button is pressed and then uses a set of else if statements to check which motor’s angle value should be changed, if the new angle value would be over 180 or under 0 and if the current real angle is equal to the previous temporary value. Once one set of these checks have been passed a temporary holding value for the respective angle gets set to the current value ±3, this new value is then passed to a second always block.

The second always block checks on each positive clock edge if the push buttons are all unpressed, if they are it checks which servo should be moved and that the temporary value is different to the old one, if so It is assigned otherwise the previous value is held. The check to see if the temporary value is different to the current value is not really necessary, but it was one of many attempts to stop the angle value from instantly hitting the maximum or minimum value as soon as the button is pressed it was kept as it shouldn’t cause problems and there wasn’t much time left once this was working so it wasn’t worth optimising at the risk of breaking things.

The final module for manual control ANGLE\_DECODE takes the angle value and breaks it down into its individual digits, again it uses a case statement to determine which motor is being moved based on the active switch. The angle values for the current angle are divided into their individual digits by calculating the modulo of the angle for the hundreds and then taking the modulo of the angle divided by 10 and 100 for the tens and ones respectively. The modulo command is very resource intensive on FPGA’s and so an alternative method that has been used before would be to use a very long case statement that checks if each number is within a range of numbers for each digit, this method could potentially be used to optimise the design however it is very time consuming to write and it’s very easy to make mistakes during the process, which is why in this situation where time was limited modulo was used instead.

**6 Q-Learning**

Final stretch, discus how the Q-Learning did and didn’t work.

Discus potential issues and solutions. Duplicated below?

Using parallel capabilities maybe could have calculated the new-Q for all 4 actions at once

**7 Overall testing**

Failure to move arm using Q-learning, solutions and fixes. Duplicate?

Mention Initial incorrect PWM signals causing servos to be noisy at rest and correcting it

**8 Conclusion**

Summarise everything could maybe be merged with overall testing depending on how much stuff fits/doesn’t fit.

**Recommendations**

With more time would have been doable too much work for one person in the time frame

Lack of peer to review of work due to lack of FPGA experienced technicians/peers in the lab.

With hindsight would have reduced scope of project, replaced arm with remote controlled buggy/car and used a bigger floor-based maze. Removes kinematics, use of MATLAB and worrying about how to communicate kinematics to the FPGA, reduced number of tasks/required skills makes project significantly more manageable.

**References**

isolator LED test <https://youtu.be/WXU1UwsgJGA>

isolator PWM test <https://youtu.be/epZFnG3XhxA>