Driver for Fingerprint Scanner

Nelson Talukder
CID 01055592
Blackett Laboratory, Imperial College London

23rd November, 2017

Abstract

The project produced a device capable of fingerprint recognition, consisting of security features to verify matching and non-matching fingerprints with the additional capability of storing new fingerprints. The main user output was an LCD, which was able to communicate to the user the results. A total of 6 features were available to the user. Extensions to the security aspect were options to check a specific memory location for a fingerprint or determine where in the memory it was located; both unique functions. A final feature was the use of an alarm to alert of a possible intruder. An investigation into the reliability of the scanner returned the figure of up to 90% accuracy depending on the kind of finger used. Reasons for variations in accuracy were also examined.

1 Introduction and Theory

1.1 Introduction

The main objective was the production of a device, designed so a user could have their identity verified for security purposes. Many such devices already exist using, for example Iris scans and facial recognition. The advantage of using a fingerprint scanner however is the relative price and complexity [1].
It was decided to go beyond this particular objective and produce a multi-purpose device that utilised additional hardware to enhance the user experience with extra features.
Building a driver, which could control the scanner, required extensive coding. The microprocessor had to have all the instruction codes ready to transmit to the scanner and a way of dealing with the verification signals sent back. This required the use of various on-board Hardware such as long term storage in SRAM and the serial to parallel conversion within USART1.
A major focus of the project was also user interaction; whether the user would be able to understand the capabilities of the device and know exactly how to access different features. There were multiple options presented on the LCD display and a clear welcome screen to explain how they could access these options. These screens may be viewed in Appendix C.

1.2 Theory

The type of sensor used here was optical. The device worked in a similar way to a camera taking a picture with the print recorded by CCD's. There are a few types of Optical Fingerprint Scanner. The one here used Frustrated Total Internal Reflection or FTIR [1]. This used the reflection of light from the fingerprint to produce an image of the finger, as demonstrated in Figure 1. The varying reflectivity of the light from a groove compared to a ridge provided a method to distinguish between the two. Light was reflected from the air-glass boundary and absorbed at the boundary between the ridge and the glass. This meant reflected light would show the pattern of ridges as dark lines.

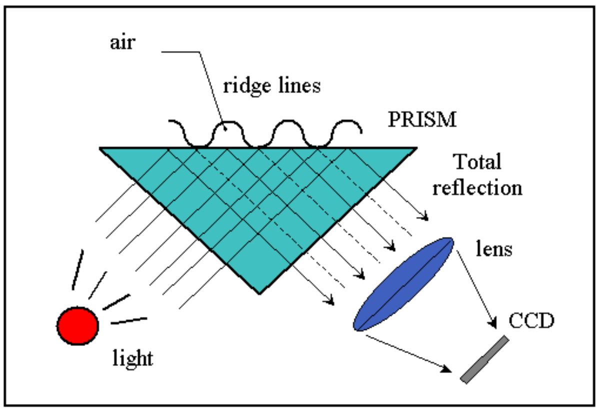


Figure 1: The reflection of light from the of a finger ridge compared to a groove.

The Scanner analysed certain features, or minutiae [2], in the image of the finger. The distance between these features was recorded as a template. Something that was expected to reduce the accuracy of the scanner was distortion on the surface of the sensor taking the print. This included the presence of grease or dust which would create either an out of focus stored image or a poor representation of any new fingerprints. Previous studies have highlighted this as an issue that standard optical scanners do not resolve [3]. This was not investigated, however was suggested as a reason for limited reliability in matching.

2 High and Low level, Software and Hardware design

Throughout the project, signals received were examined by storing them in internal SRAM and checking them through Atmel Studio on the computer. All programming was done in AVR Assembly. Results were originally displayed on LED's however this was removed when the LCD display was initialised. Constant reference to the atmega128 datasheet helped to ensure that every step could be analysed for what was going on computationally.

2.1 High level Design

A basic overview of how the microprocessor dealt with the various hardware can be observed in Figure 3. Each input or output device had to be connected to the processor via a port, which may be viewed in Figure 4, on the following page.
How the whole program worked can be viewed in Figure 2. This shows how the program displayed options to the User and asked for input via the LCD.

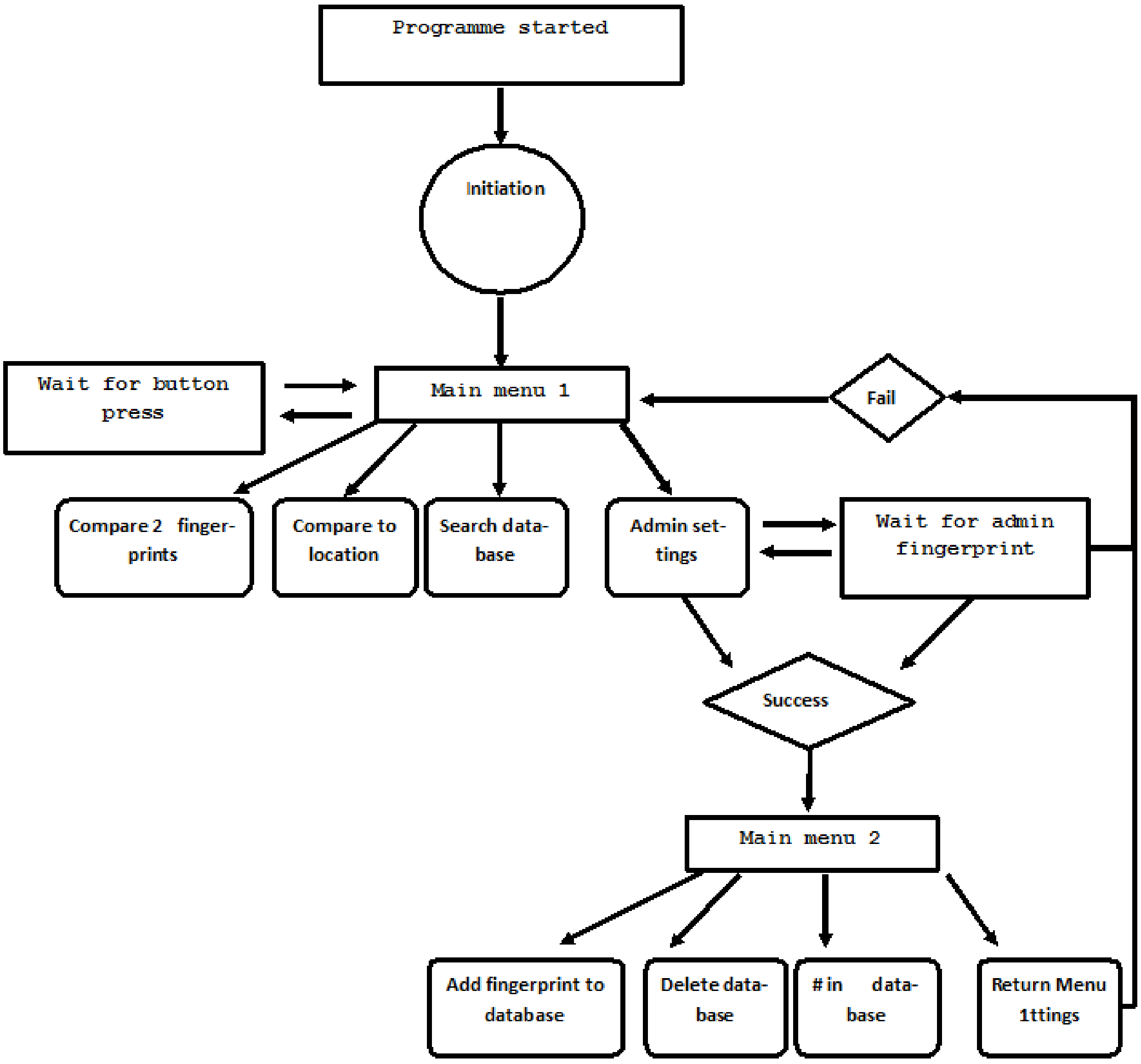


Figure 2: Flow map of the entire operation of the device.

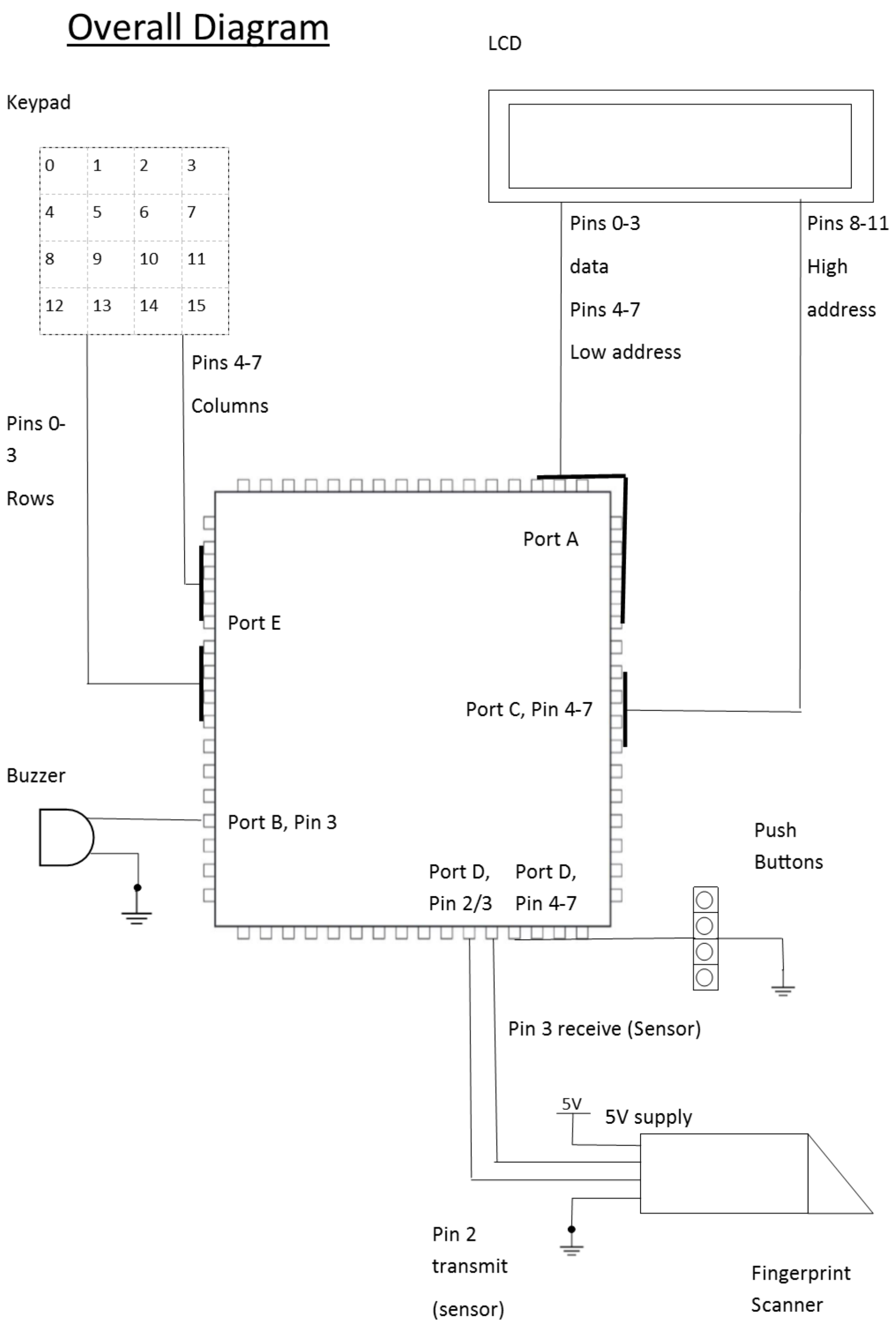


Figure 3: Diagram of overall setup. The full operation of each pin can be viewed in figure 4.

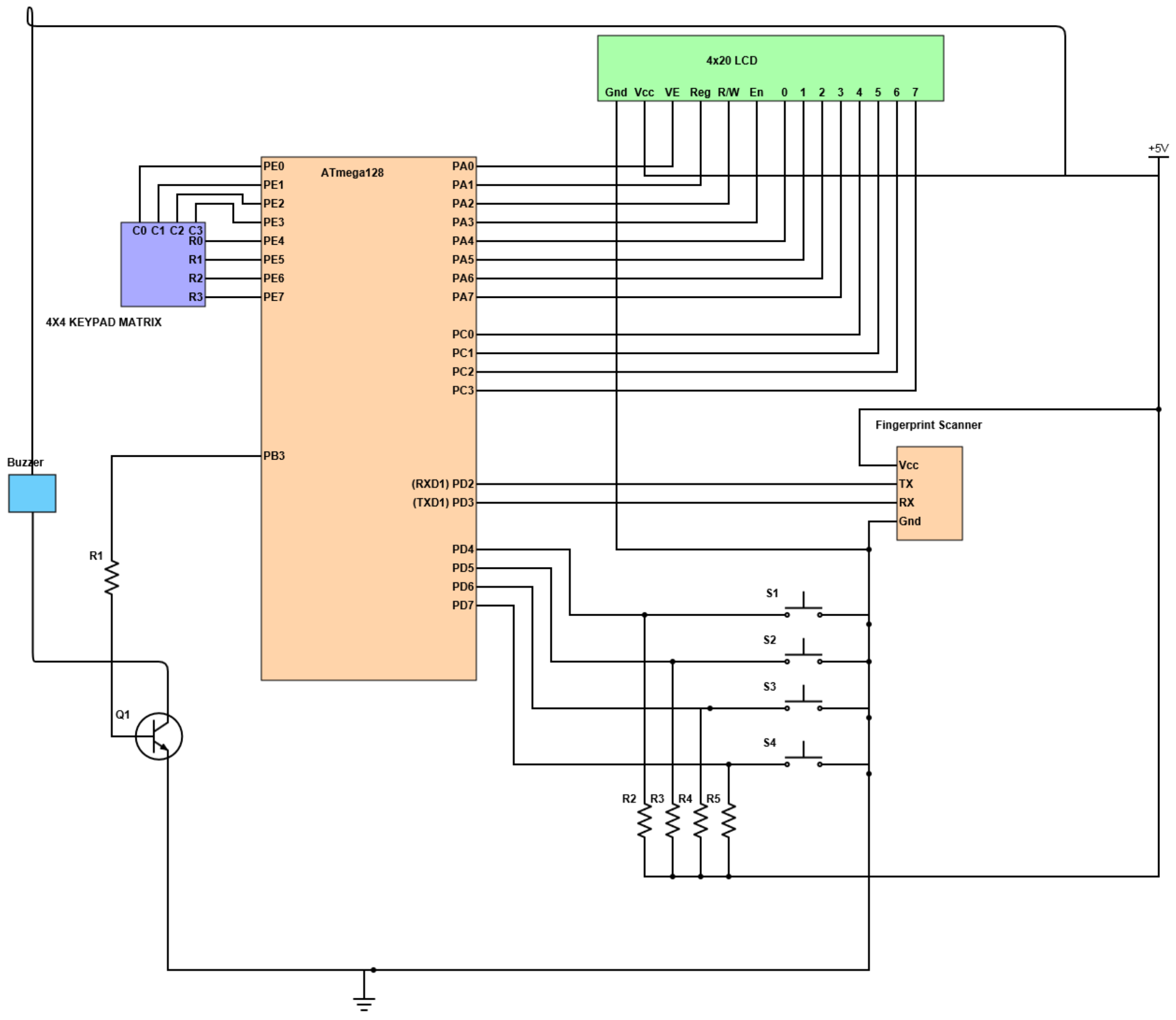


Figure 4: Schematic of the Hardware used.

2.2 Individual Hardware

2.2.1 Keypad

This was solely an input device. The user could select a memory locations, listed from 0 to 15. The keyboard provided a single byte when the get_input subroutine was called, viewable on lines 39-64, Appendix D. The program could then identify which button had been pressed. The keypad worked by running a current through a grid of wires, viewable in figure 5. If a button was pressed the voltage was pulled down on a particular wire resulting in a 0 for the pin attached to it. A current was first run into pins 4-7 and out of 0-3 giving the 4 highest bits. The current inputs and outputs were then switched to get the 4 lowest bits. These to value were added together by software and saved.

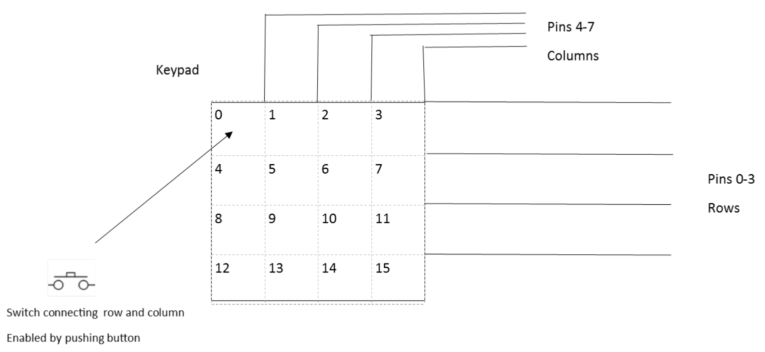


Figure 5: Diagram of the Keypad used.

2.2.2 Buzzer

The buzzer was connected to pin 3, Port B. Software controlled whether it received a DC voltage signal, enabling it to transmit an audible signal. The buzzer was used as an additional output device to improve the security aspect where the device could immediately signal an unrecognized person.

2.2.3 LCD

The type of LCD used was a Hitachi with a 20 × 4 display. This allowed up to 4 options to be displayed above each other. The microprocessor used Ports A and C to communicate with the LCD. The complete set up of the pins can be viewed in Figure 4. It required 4 data and 8 address pins [4]. Loading a value to the external SRAM location $C000 would lead to the value being displayed in the LCD. The LCD presented a more user-friendly experience where a person could be presented with options on how to interact with the device as shown in Figure 6. It also meant a more varied way to display whether a fingerprint was credible or fraudulent. It allowed the specific ID of the person to be presented to the user. A major advantage of the LCD however was that it made it a completely stand-alone device that could communicate features directly, not via a handbook.

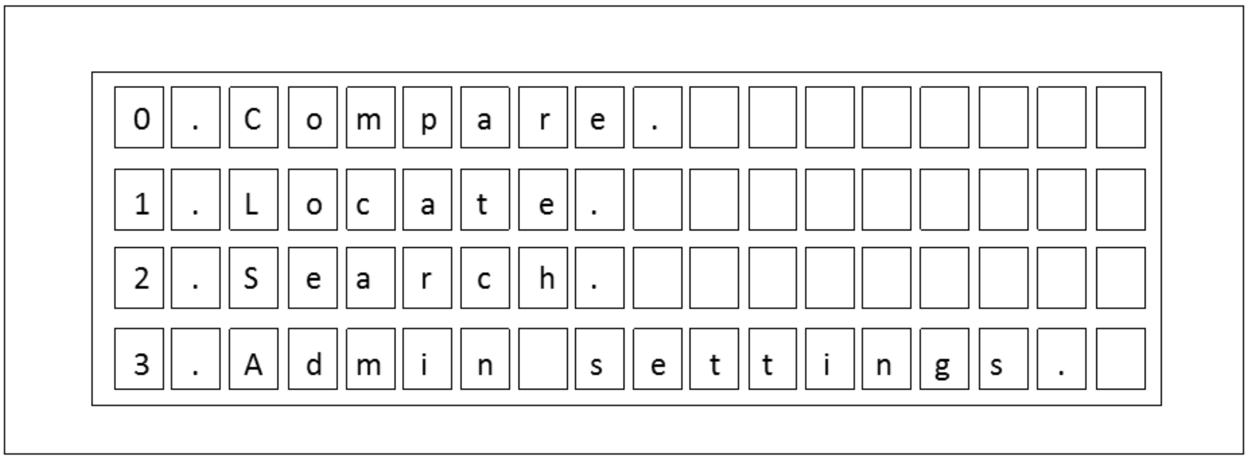


Figure 6: The layout of the options on the LCD.

2.2.4 Input Pins

These were inputs that allowed the user to select a feature. Pins 4-7 of port D were set up as an input port for this purpose. The ones used here were pull down pins, as can bee seen from Figure 3. This meant they redirected power from the pins when they were pressed resulting in a different value being read from the pins. For example, pressing the last pin would change the output from Port D from $FF to $FE. An index of four different values could be looked up by the program to determine which pin had been pressed hence which option the User had selected, the code may be viewed in Appendix E, lines 740 to 765.

2.2.5 USART1 - On Board

The USART1 was set by the Atmega128 to be controllable through port D, as shown in Figure 7. Information received was placed in the USART buffer. The microprocessor was programmed to collect the data and to store it in various SRAM locations.

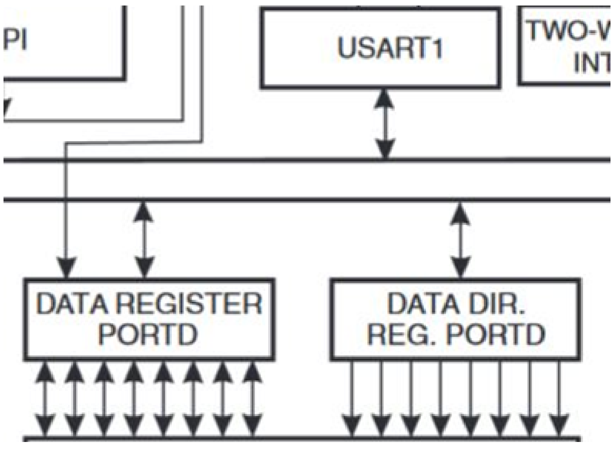


Figure 7: USART1 communication with Port D.

2.2.6 Fingerprint Scanner

The type of Fingerprint Scanner used was the ZFM-206, produced by Hangzhou Zhian Tec. This module had its own processor and storage which dealt with the requested function on command performed .

2.3 Data Transfer

The fingerprint scanner interacted with the Microprocessor via asynchronous serial communication.

This meant transmission had to be one bit at a time by the Microprocessor. Serial data from the Scanner to the Microprocessor had to be converted to parallel so the it could be stored by the microprocessor.

2.3.1 Transmission

Software was used to manually send serial signals through a wire to the fingerprint Scanner via pin 3 of Port D. The sendUART subroutine was called to send each individual bit of every byte. Loops sent the bits at carefully calculated increments so that they could be sent with the correct frequency.

2.3.2 Reception

Originally, a similar method was employed to receive data where it was collected as individual bits. Pin 2 was polled for a period after each transmission. This however gave an issue were the Atmega128 board had to be prepared all the time and was not always quick enough to switch to a state where it could receive data. Later the USART1 interrupt allowed any signal received into Pin 2 of Port D to be saved in the memory, whenever data was available. The previous error meant there were occasions where confirmation data on whether a fingerprint had indeed been recorded was not picked up by the Microprocessor. The hardware USART had to be set up to enable interrupts and included the use of the subroutine save_data_start352. Every Byte of data that was received in Port D was transferred to the USART1 buffer where it could be collected by the microprocessor before another byte was received by the buffer.

2.3.3 Setting BAUD rate

The rate at which bits were sent and received had to be set up by software. The required rate was 57600 bits/s for the Fingerprint Scanner to register it as a Command Package. For transmission, the time of each bit sending loop was calculated to ensure the bits were sent at the correct rate. This involved calculating a delay that ensured the loop would wait for the correct time to send the next bit. It was calculated the loop would require 138 cycles to send a bit, given the oscillator freqeuency of the Atmega128 was 8Mhz [5]. The number of cycles taken by commands was 30. Therefore 108 cycles of delay used in the transmission cycle. It was simpler for the USART1 BAUD rate. This followed equation 2, obtained from the Atmega128 datasheet. UBRR was a quantity that could be set in the initialisation of USART1. UBRR was calculated to be 8.

$$UBRR = \frac{f_{OSC}}{16(BAUD)} - 1 \tag{1}$$

2.4 Software

2.4.1 Initialisation

The Ports were initialised as either input or output within an Initialisation section. This included setting up the correct delay to obtain the correct frequency for the manual transmission of data. The entire initialisation sequence is given in Appendix E, lines 21-107. Figure 8 gives an overview of the main stages of initialisation.

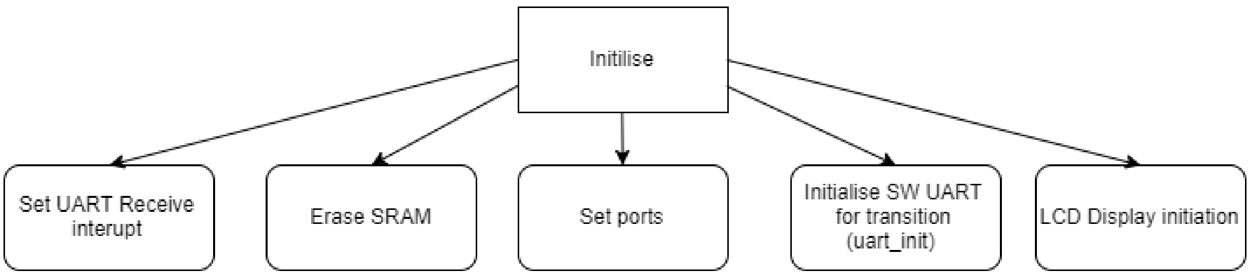
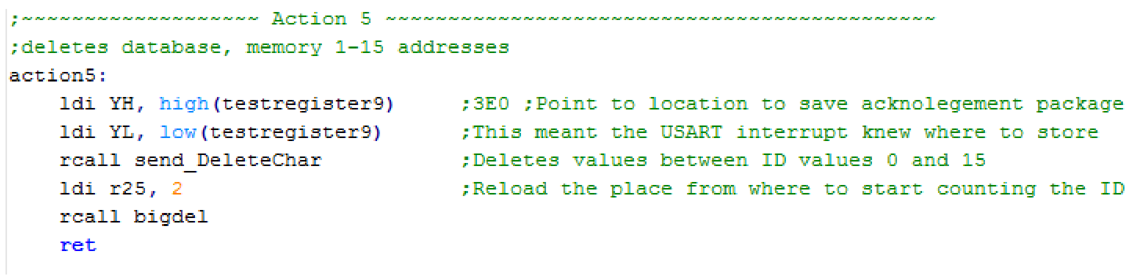

Initilise
Set UART Receive interupt
Erase SRAM
Set ports
Initialise SW UART for transition (uart_init)
LCD Display initiation

Figure 8: Mid level sketch of the initialisation

2.4.2 Actions

The program was split into various modules, called actions. These were responsible for the features the User could access. The actions were kept in a separate include file. They would call shared subroutines from the main file to operate. An example of an Action is given in Figure 9. This action was responsible for erasing the memory.

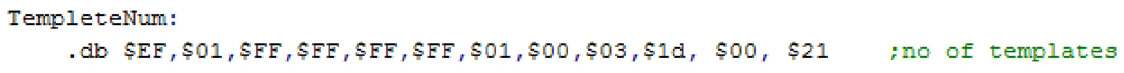


```
;~~~~~~~~~~~~~~~~~~ Action 5 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;deletes database, memory 1-15 addresses
action5:
    ldi YH, high(testregister9)      ;3E0 ;Point to location to save acknolegement package
    ldi YL, low(testregister9)       ;This meant the USART interrupt knew where to store
    rcall send_DeleteChar            ;Deletes values between ID values 0 and 15
    ldi r25, 2                       ;Reload the place from where to start counting the ID
    rcall bigdel
    ret
```

Figure 9: Action5, Actions were the largest blocks the program could be split into.

2.4.3 Databases

Various databases with the different Command Packages split up into bytes had to be stored in the Atmega128. This meant various commands could be sent at various stages in the program to the Fingerprint Scanner. Figure 10 below shows the database required for Action counting the number of stored fingerprints. The form of the command packages is provided in the datasheet [6].

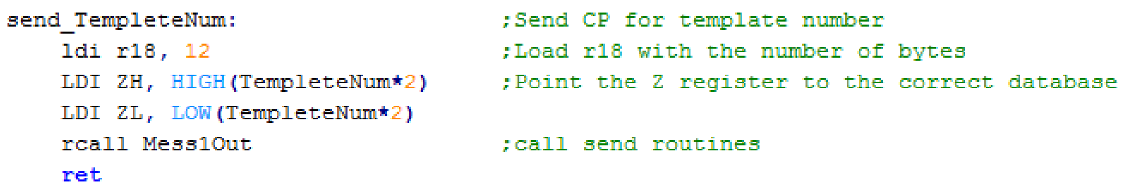


```
TempleteNum:
    .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$03,$1d, $00, $21    ;no of templates
```

Figure 10: A database with a Command Package split into individual bytes.

2.4.4 Sending Subroutines

A number of subroutines were used, each of which sent a different Command Package to the Fingerprint Scanner. Figure 11 below shows the subroutine which sent the Command for counting the number of stored templates, or fingerprints.

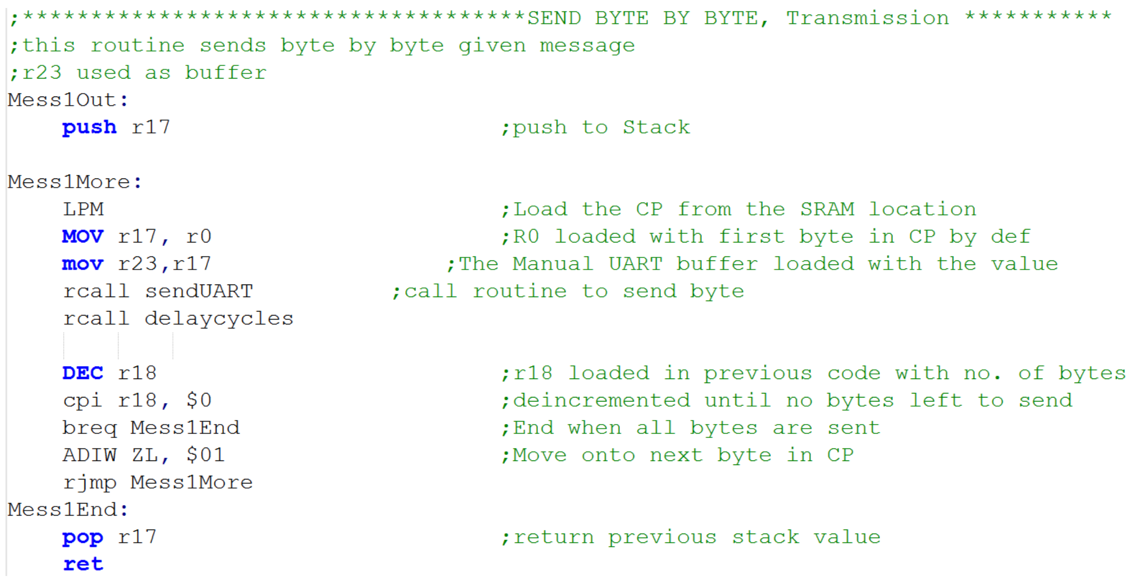


```
send_TempleteNum:                ;Send CP for template number
    ldi r18, 12                  ;Load r18 with the number of bytes
    LDI ZH, HIGH(TempleteNum*2)  ;Point the Z register to the correct database
    LDI ZL, LOW(TempleteNum*2)
    rcall Mess1Out               ;call send routines
    ret
```

Figure 11: Snippet of the sending routine responsible for outputting the data from the Atmega128.

2.4.5 Mess1out

This function allowed the data to manually be sent to the device, byte by byte. It was called as a subroutines by the send_TempleteNum in Figure 11, that pointed to the correct database. Mess1out then sent the bytes. Figure 12 shows how it looped through whole Command Package, sending the data byte by byte.

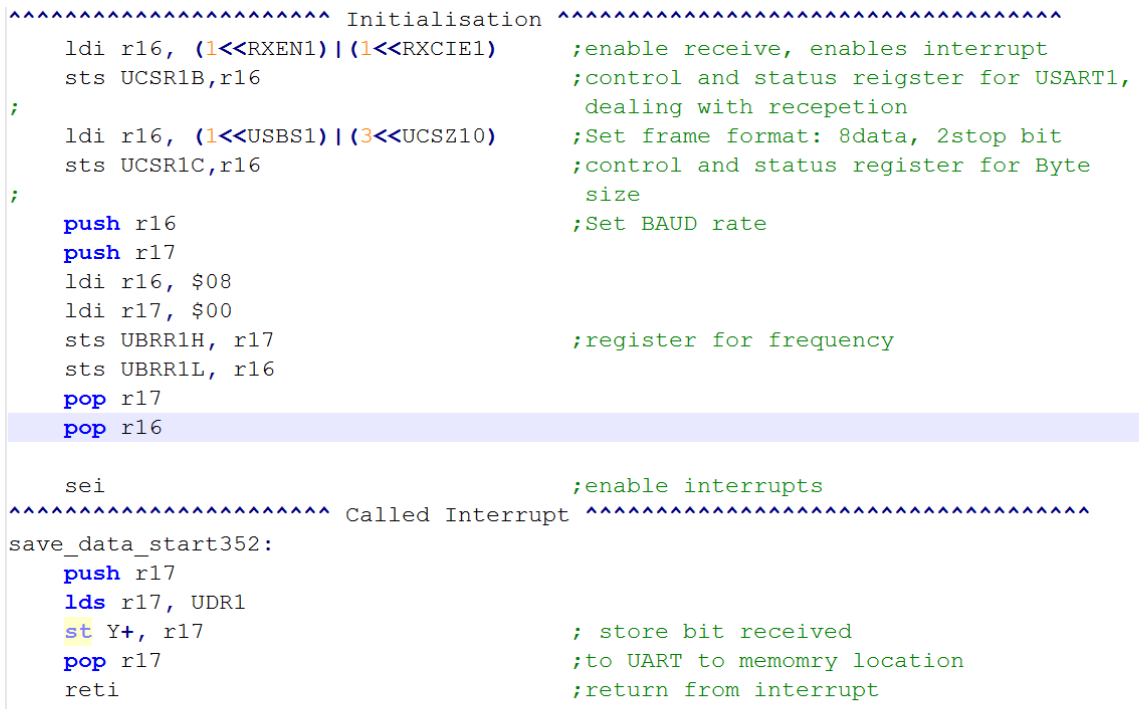


```
;*************************************SEND BYTE BY BYTE, Transmission ***********
;this routine sends byte by byte given message
;r23 used as buffer
Mess1Out:
    push r17                        ;push to Stack

Mess1More:
    LPM                             ;Load the CP from the SRAM location
    MOV r17, r0                     ;R0 loaded with first byte in CP by def
    mov r23,r17                 ;The Manual UART buffer loaded with the value
    rcall sendUART              ;call routine to send byte
    rcall delaycycles

    DEC r18                         ;r18 loaded in previous code with no. of bytes
    cpi r18, $0                     ;deincremented until no bytes left to send
    breq Mess1End                   ;End when all bytes are sent
    ADIW ZL, $01                    ;Move onto next byte in CP
    rjmp Mess1More
Mess1End:
    pop r17                         ;return previous stack value
    ret
```

Figure 12: Snippet of the code responsible for byte transmission.

2.4.6 Save_data_start352

This interrupt sequence allowed any data that was captured in the USART buffer to be stored in the previously pointed location in SRAM. It required USART1 to be set up in a way that allowed it to store data to the USART1 buffer whenever data was received into pin 3 of Port D. The initialisation and routine can be viewed in Figure 13.



```
^^^^^^^^^^^^^^^^^^^^^^^ Initialisation ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    ldi r16, (1<<RXEN1)|(1<<RXCIE1)     ;enable receive, enables interrupt
    sts UCSR1B,r16                      ;control and status reigster for USART1,
;                                        dealing with recepetion
    ldi r16, (1<<USBS1)|(3<<UCSZ10)     ;Set frame format: 8data, 2stop bit
    sts UCSR1C,r16                      ;control and status register for Byte
;                                        size
    push r16                            ;Set BAUD rate
    push r17
    ldi r16, $08
    ldi r17, $00
    sts UBRR1H, r17                     ;register for frequency
    sts UBRR1L, r16
    pop r17
    pop r16

    sei                                 ;enable interrupts
^^^^^^^^^^^^^^^^^^^^^^^ Called Interrupt ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
save_data_start352:
    push r17
    lds r17, UDR1
    st Y+, r17                          ; store bit received
    pop r17                             ;to UART to memomry location
    reti                                ;return from interrupt
```

Figure 13: Code snippets of the set-up and routine for the USART1 interrupt subroutine.

2.4.7 sendUART

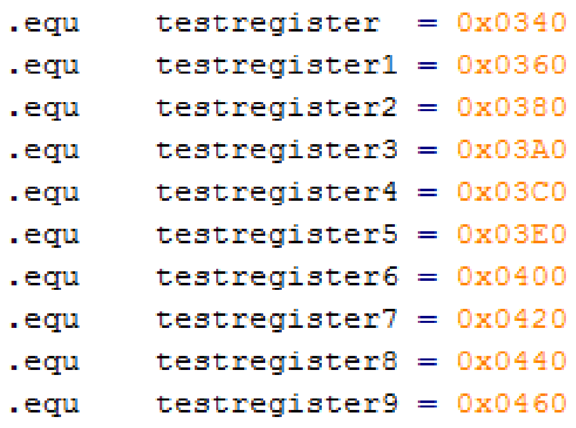
This Subroutine was set up to manually send bits of data. It can be seen in Appendix E, lines 530 to 565. It consisted of 138 cycles, including a set delay which ensured that bits were sent at the frequency of 57600 bits/s.

2.4.8 Compare_basic2

Any Acknowledgement Package indicating the success of a matching had to be compared to a database. This database had a unique sequence that determined whether the package indicated a correct Match. Comparing the package to the sequence of a correct Match allowed the program to conclude whether the scanner had found a Match for the fingerprint. This section of code and may be viewed in appendix E, lines 328 to 395.

2.5 Testing the Software

Each sent Command Package would generate an Acknowledgement Package which would state what the Fingerprint Scanner had done, these varied in Length between 12 and 16 bytes. The Acknowledgement Package always included a byte of confirmation code that would verify the success of a task or an inability to read the Command Package. These codes were given in the datasheet [6]. Throughout the production of the overall product, these were saved to SRAM. The various locations in SRAM where the first byte would be saved are shown in Figure 14. Several locations were required as several Acknowledgement Packages, generated by each respective Command Package, had to be stored in SRAM.



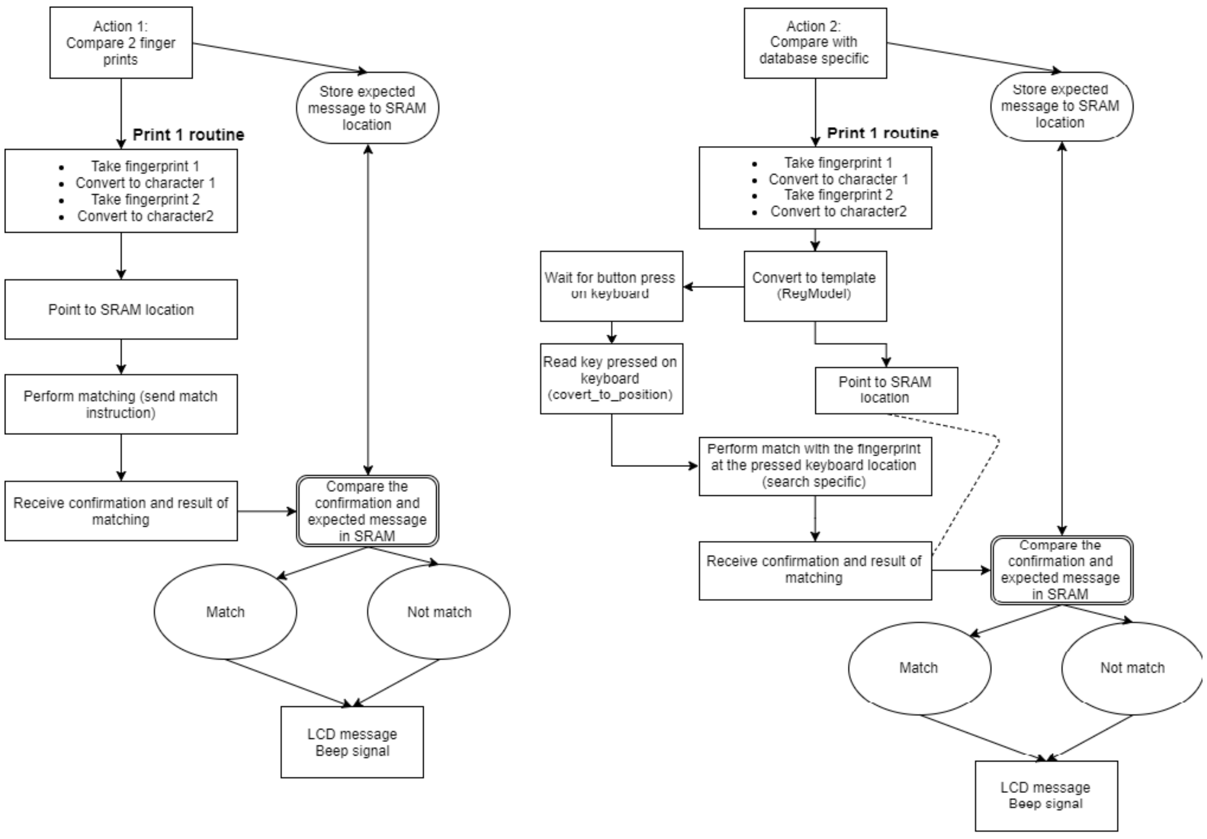
```
.equ    testregister  = 0x0340
.equ    testregister1 = 0x0360
.equ    testregister2 = 0x0380
.equ    testregister3 = 0x03A0
.equ    testregister4 = 0x03C0
.equ    testregister5 = 0x03E0
.equ    testregister6 = 0x0400
.equ    testregister7 = 0x0420
.equ    testregister8 = 0x0440
.equ    testregister9 = 0x0460
```

Figure 14: SRAM locations for storing various Acknowledgement packages.

2.6 Flowcharts of individual actions

The compare function, Action 1, attempted to take two fingerprints and compare them for similarity.

The use of this would be a quick test to compare a real to a possible forged fingerprint to see the reliability of the fake.
Action 2, was called 'locate'. It performed the function of getting the fingerprint, before asking the user for a location against which to compare the fingerprint. The flowcharts of actions 1 and 2 can be seen in Figure 15.
Actions 4, 5 and 6 were set as administrative options as they could alter the memory of the Scanner or provide information that could be used by an intruder. They were placed on a separate screen. Figure 2 shows how it could be accessed from screen 1.
Action 3 performed the general database search. The on screen option for it was 'Search Database'. Action 4, named 'Store', was a function that stored a fingerprint and allocated it a specific memory location. Action 3 and 4 can be seen in Figure 16.
The Template number counter, Action 6 requested a total for the number of used memory locations, this required one sent package and 1 received package. Memory clearance was performed by Action 5, or 'Delete'. It sent one Command Package to the device to delete memory locations 0 to 15.



Action 1: Compare 2 finger prints
Store expected message to SRAM location
Print 1 routine
Take fingerprint 1
Convert to character 1
Take fingerprint 2
Convert to character2
Point to SRAM location
Perform matching (send match instruction)
Receive confirmation and result of matching
Compare the confirmation and expected message in SRAM
Match
Not match
LCD message Beep signal
Action 2: Compare with database specific
Store expected message to SRAM location
Print 1 routine
Take fingerprint 1
Convert to character 1
Take fingerprint 2
Convert to character2
Wait for button press on keyboard
Convert to template (RegModel)
Read key pressed on keyboard (covert_to_position)
Point to SRAM location
Perform match with the fingerprint at the pressed keyboard location (search specific)
Receive confirmation and result of matching
Compare the confirmation and expected message in SRAM
Match
Not match
LCD message Beep signal

Figure 15: Flow Charts for the Compare and Locate functions.

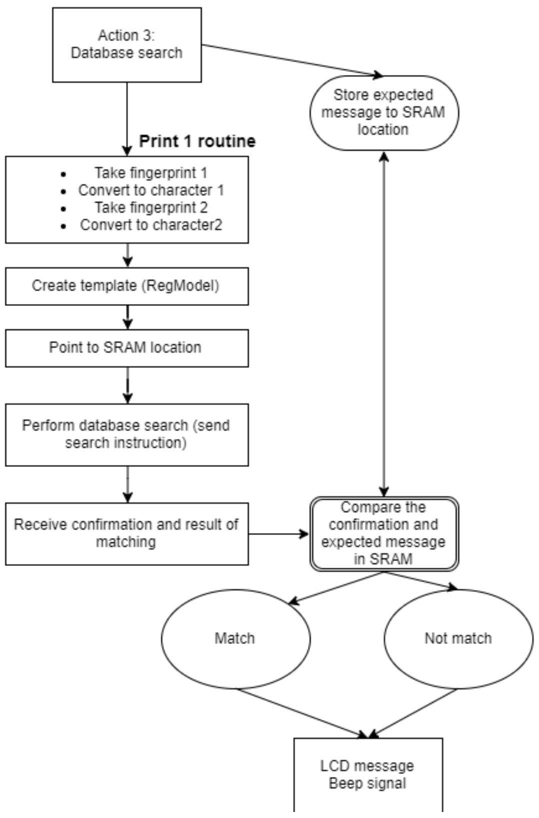
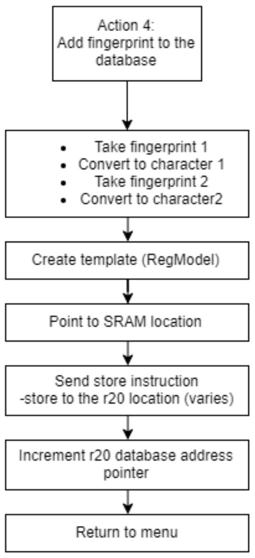

Action 3:
Database search
Store expected message to SRAM location
Print 1 routine
Take fingerprint 1
Convert to character 1
Take fingerprint 2
Convert to character2
Create template (RegModel)
Point to SRAM location
Perform database search (send search instruction)
Receive confirmation and result of matching
Compare the confirmation and expected message in SRAM
Match
Not match
LCD message
Beep signal


Figure 16: Flow Charts for the Search and Store functions.

3 Results and Performance

3.1 Results

The device performed all the actions mentioned in the plan. This includes the ability to recognize a fingerprint and to alert of a possible fraudulent identity via an alarm.

3.1.1 Reliability

The reliability of the fingerprint scanner was measured for different fingers to check the variation of different sizes and type of finger. Each finger of Experimenter 1 were stored on the device. The locate function was then used 10 times for each finger to check each finger against it's stored template. The results of the investigation can be viewed in Figure 17. This concluded in the hypothesis that the scanner recognized certain fingers more easily than others. It was very capable of measuring the thumb, index and middle fingers at around 80-90% accuracy. It struggled more with the fourth and fifth fingers with less than 70% accuracy, compromising the reliability of the scanner.

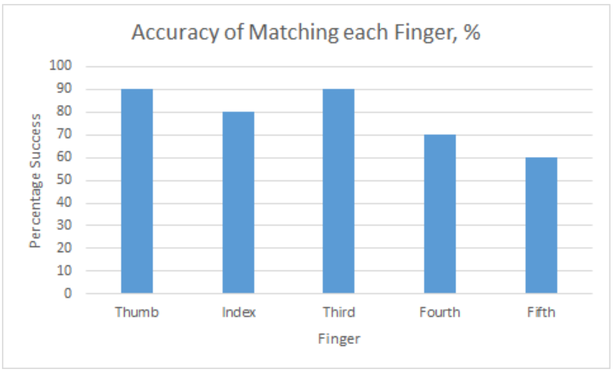


Figure 17: The varying reliability of matching different fingerprints.

The quoted failure rate from the datasheet of the fingerprint scanner was 1% [6]. It was not mentioned if this was just for a particular person or a particular finger and whether they were instructed to present their finger in a certain way. This result seemed to be an over optimistic estimation by the manufacturer. The conclusion drawn was the variation in size of the fingerprint appeared to be factor from experiment. Investigating false positives resulted in the conclusion that it was highly unlikely to get a false positive with none returned in all forms of testing, this however was not recorded. The figure provided for the rate of false positives was 0.001% or 1 out of 100,000 by the Scanner's datasheet [6]. Given that it was unlikely more than 1000 prints would be taken over the course of the experiment, this was concluded to be futile to experiment formally.

3.2 Performance

The general search of the Scanner's database, the basic security feature from the plan, was only limited by the speed of searching. According to the datasheet, the fingerprint scanner could take up to 1 second to search the database. This caused a significant delay in receiving a result. As this was a problem with a specific piece of hardware, it was concluded that it would be difficult to resolve without changing the Scanner used. The Fingerprint Scanner always did exactly what the User expected. Achieving all goals set out in the Product Plan. A picture of the complete set up of the device can be viewed in Appendix B. Action 1 worked successfully. The challenge however was to produce match-able files. There were 3 different files type, an image, a character file and a template. An error in the Datasheet [6] said that templates needed to be matched, however it was the case character files needed to be matched. This was resolved by trial and error, converting the files between different file types and matching them. An adjustment also allowed the program to continue if the fingerprints were not presented in the time. In which case the program simply went back to the option screen.
Action 2 performed exactly the task expected. The only issue in this case being if the User did not press the keypad for a sufficient length of time, around 0.1s, it would not register the correct value. This could be improved by changing the delay values. More of a challenge would be to solve the multiple finger press, when more than 1 key is pressed simultaneously. This could be solved by not acknowledging the result and asking for the User to resubmit a fingerprint. Actions 4, 5 and 6 were successfully set up as administrative options. They required the User to already be registered as an administrator. Only Experimenter 1 and 2 were set as administrators however if a new person was registered, they could gain temporary administrative access. Action 4 could store a fingerprint if it was presented and wouldn't take one if it was not presented. The only issue here being that the memory location for the next fingerprint would be incremented, regardless of whether a fingerprint was taken or not. This could be resolved with modification [1].
The challenge in getting this to work fully was tailoring the Command Package to constantly change memory location. How this was implemented can be seen in Appendix E, lines 456 to 487. This required two bytes to vary; the part of the package responsible for memory location and the Checksum, a verification byte set by the Scanner [6]. It also involved incrementing a value in SRAM to change the value for memory location.
Action 5 was able to check how many Fingerprints were already stored in the memory. This was a useful feature which allowed the User to know if the database had been previously cleared of non-permanent identities. The device effectively cleared memory locations when asked for by Action 5. The LCD was able to display every option the User could access, all of these can be viewed in Appendix C. There was also a possibility to return to the first option screen from the second, displayed on the second screen as 'return'.

[1]See 4.2 Modifications.

3.2.1 Extensions

The project went beyond the plan set out by including extra capabilities. This included administrative setting where the device did not just recognize the specified user; it could add new fingerprints, remove old ones and check the memory usage. A final extra adaption was a function to check the memory manually.

3.3 Errors

Multiple factors could have contributed to the discrepancy between the stated failure rate and the observed failure rate. This included the position of the fingerprint over the sensor and distortion from dirt and grease on the surface of the scanner. These could warrant future investigation. A further unresolved issue could be the fact that the device used a small area scanner that did not scan the whole fingerprint. This required the same area of a finger to be presented every time. Previous investigation [1] suggested that the area of the sensor is the most likely factor in misidentifying a correct finger and is especially limited in small area scanners. As this was a small area Scanner, its was likely affected by this.

3.3.1 Effect of Moisture and dust

The most productive future investigation would likely be a check the effect of a dry and wet fingerprint to determine if there was a noticeable difference. Previous study [3] has indicated that it may be a significant detractor of the reliability of the Scanner.

4 Updates, Modifications and improvements

4.1 Updates

Additional functions could be added to a third screen. An easy one to implement would toggle the alarm, disabling security. This would just require some coding to disable Port B, the Alarms output port in certain situations. Another feature would be to add a pass-code override for the admin settings when a fingerprint is not recognised. This would be relatively easy to implement as the keypad is already set up and a simple routine could be called from the main screen which allows the admin settings to be accessed via a password. Another possible improvement would deal with the time between the User selecting an action and an output being displayed on the LCD. Information on the screen whilst a search occurs could help the user to know how long they will be waiting for a result. This would be in the form of a progress bar on the LCD.

4.2 Modifications

An issue with the current set up is that if the same finger is presented for storage, it will be stored in a new location. The result of which is several copies of the same fingerprint some situations. This uses up storage space for new fingerprints and lets someone have multiple ID's meaning they can't be uniquely identified by the device. A useful modification to the store function therefore would be a check to see if the print already existed in the database before it was stored. A message could be displayed on the LCD to communicate the issue to the user.

4.3 Improvements

The main issue was the accuracy meaning that someone would either have to repeatedly present their fingerprint to reduce the chance of false rejection or they would have to only present certain fingers. To circumvent this, multiple prints of one finger could be taken for a verification. This would involve a loop that kept taking prints and searching the database for a match. If a successful print was found it would exit the loop. A timer could be implemented to end the cycle once 5 seconds had elapsed without a successful attempt. This would change chance of false rejection rate to 9% from 30% for the Fourth Finger, based on estimations from the data in Figure 17. This assumes 2 cycles could be implemented in the time. This also assumes the false positive rate is small enough to be ignored.

5 Conclusion

The project managed to produce a security device as planned, one that could alert of a possible intruder via a buzzer. It also went significantly beyond this capability however could also strictly be used as intruder identification system. Extra features were also successfully implemented and tested. The reliability was investigated and clear methods for improvement identified.

References

[1] Maio D. Jain A. Prabhakar S. Maltoni, D. *Handbook of Fingerprint Recognition.* London: Springer London, 2 edition, 2009.

[2] Biometrika. Basics of fingerprint recognition technology and biometric systems, 2017. Available from: http://www.biometrika.it/eng/wp_fingintro.html.

[3] Lee S Son G Back S, Lee Y. Moisture - insensitive optical fingerprint. *Optics Express*, 24(17), 2016.

[4] Protostack. Hd44780 character lcd displays - part 1 - protostack, 2017. https://protostack.com.au/2010/03/character-lcd-displays-part-1/.

[5] Atmel Corporation. Atmega128 datasheet, 2011. Available from: http://www.atmel.com/images/doc2467.pdf.

[6] Hangzhou Zhian Tec. Corp. Zhiantec zfm206 series datasheet, 2011. http://wiki.seeed.cc/Grove-Fingerprint_Sensor/.

Specification Table

Number of keypad accessible prints	15
Maximum Capacity	120 [6]
Function	**Times**
Compare	1s
locate	<2s from Keypad Input
Search Database	<3s
Store	<2s
Template Number	1s
Delete	0.5s
Power	DC, 5V
False Read Rate (FRR)	10-20% Thumb, Index and Middle finger
FRR	30-40% Fourth and Fifth fingers
DPI	434
Image Capture	CCD [6]
Storage	Flash