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 [3].
It was also decided to also go beyond this particular objective and produce a multi-purpose device that utilised additional hardware to enhance the user experience with additional 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 of 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 the different features. There were multiple options presented on the LCD display and a clear welcome screen which explained 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 use Frustrated Total Internal Reflection or FTIR [3]. 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.

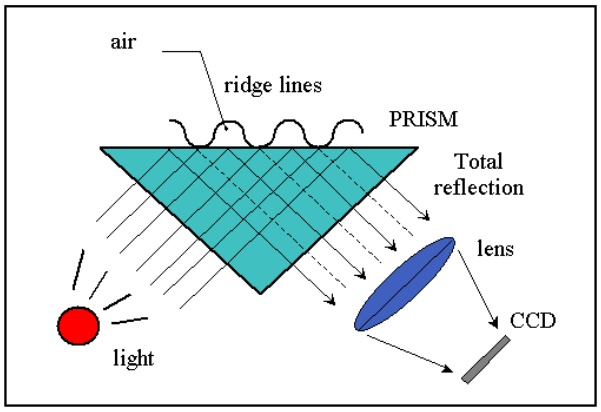

Figure 1: The reflection of light from the of a finger ridge compared to a groove.

The device used image analysis, which checked certain features of the finger, or minutiae [2], for recognition such as the distance between certain lines. 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 [1]. This 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 via storage internal SRAM and checking the value through Atmel Studio on the computer. All comparison outputs were also displayed on the LED's however this became redundant when the LCD display was initialized and could be used as an output and has since been deleted. Constant reference to the atmega128 datasheet helped to ensure that every step could be analyzed 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.

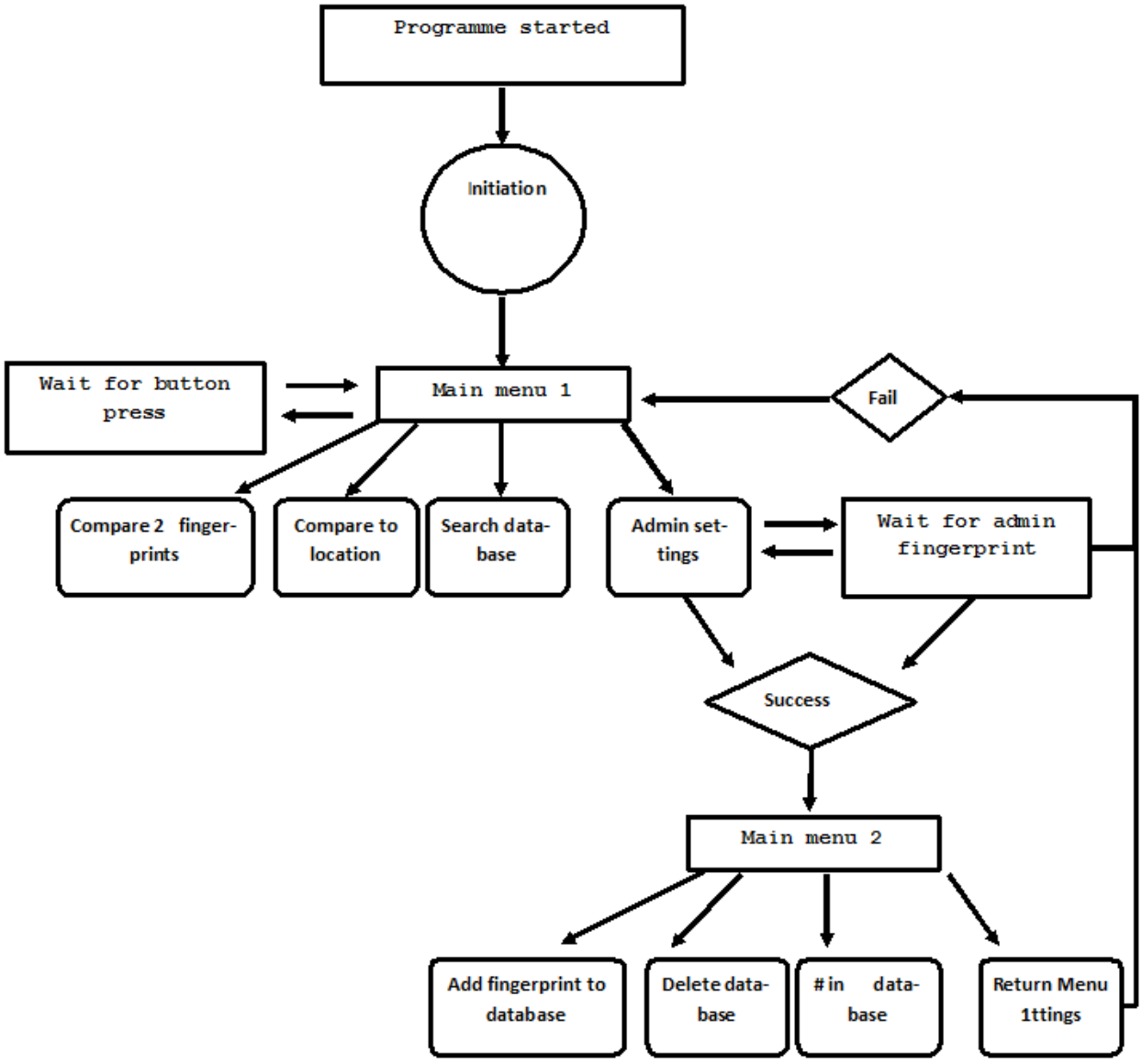

Figure 2: Flow map of the entire operation of the device.

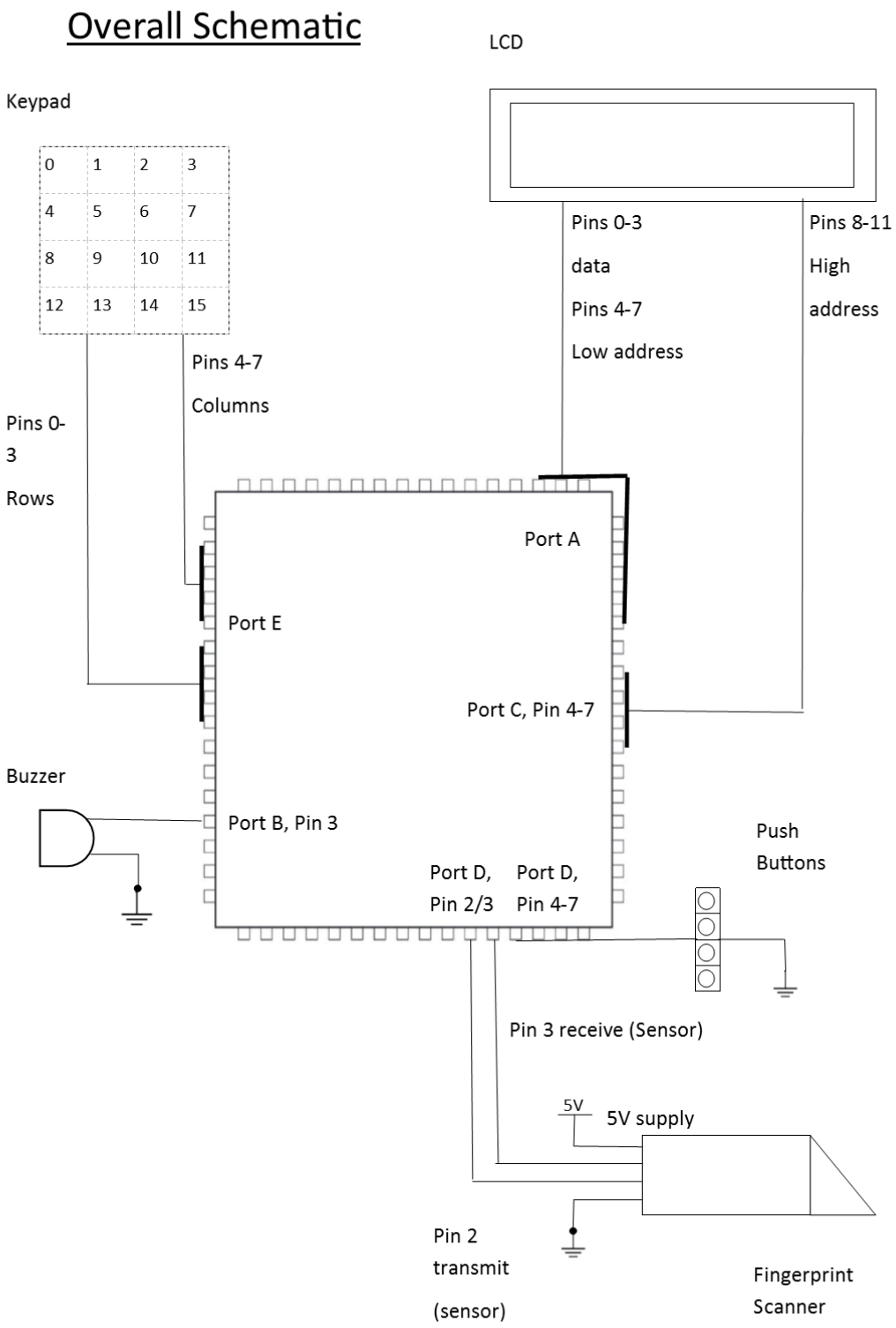

Figure 3: Diagram of overall setup. The full operation of each pin can be viewed in figure 4.

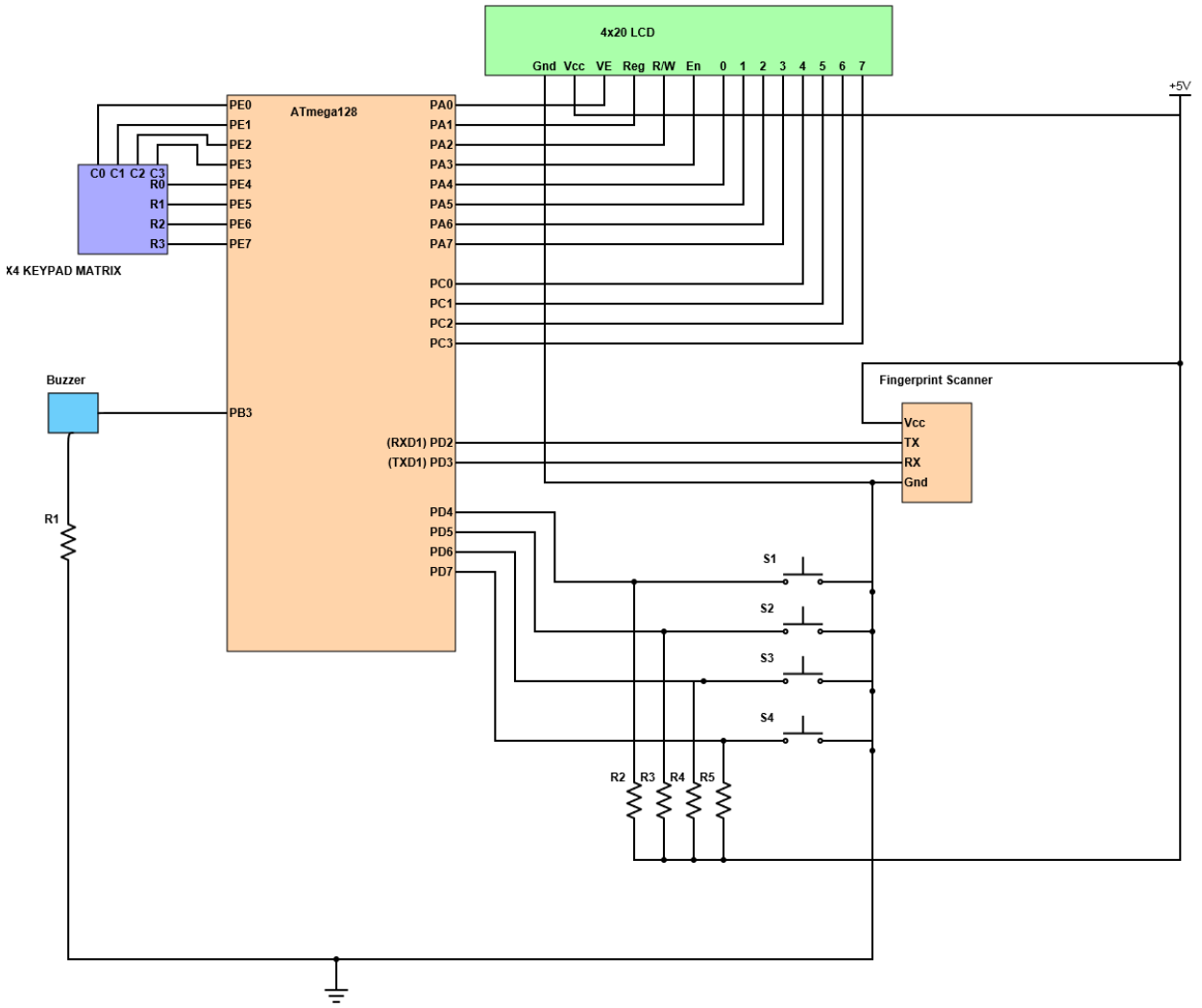

Figure 4: Schematic of the Hardware used.

2.2 Individual Hardware

2.2.1 Keypad

This was solely an input device, letting the user select from 16 memory locations, listed from 0 to 15. When asked, the keyboard would provide a single byte, from which allowed the program to 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.

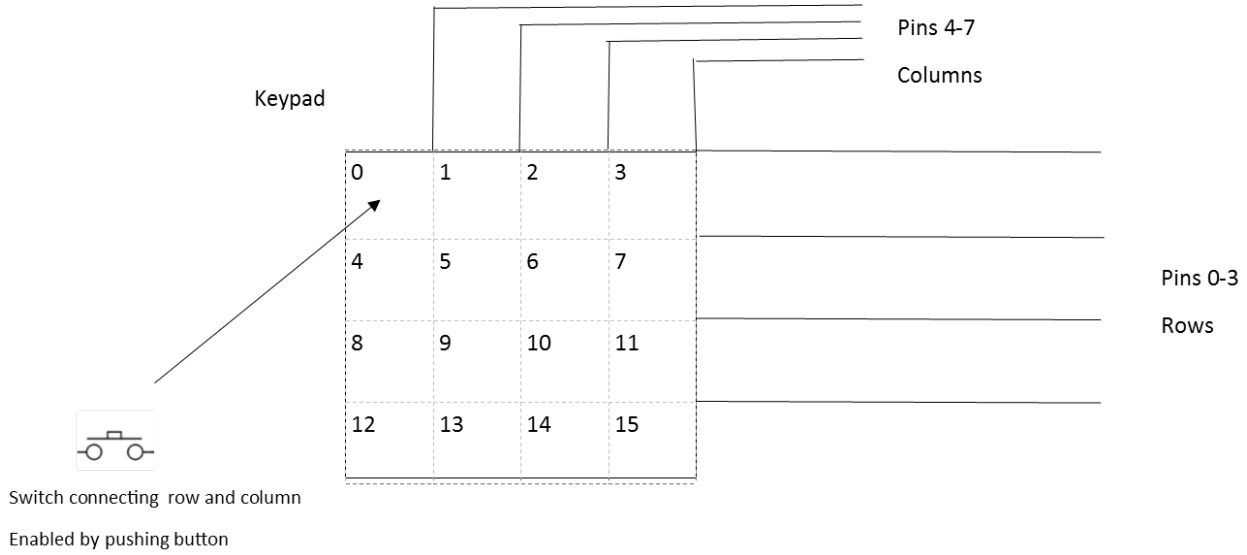

Figure 5: Diagram of the Keypad used.

2.2.2 Buzzer

The buzzer was connected to Port B, where a specific pin controlled whether it received a voltage signal enabling it to transmit an audible signal. The one used here required a DC 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 allowed the user simple way on how to set up who they wished to be recognized as credible, giving them complete control over the device. This complemented the basic security features.

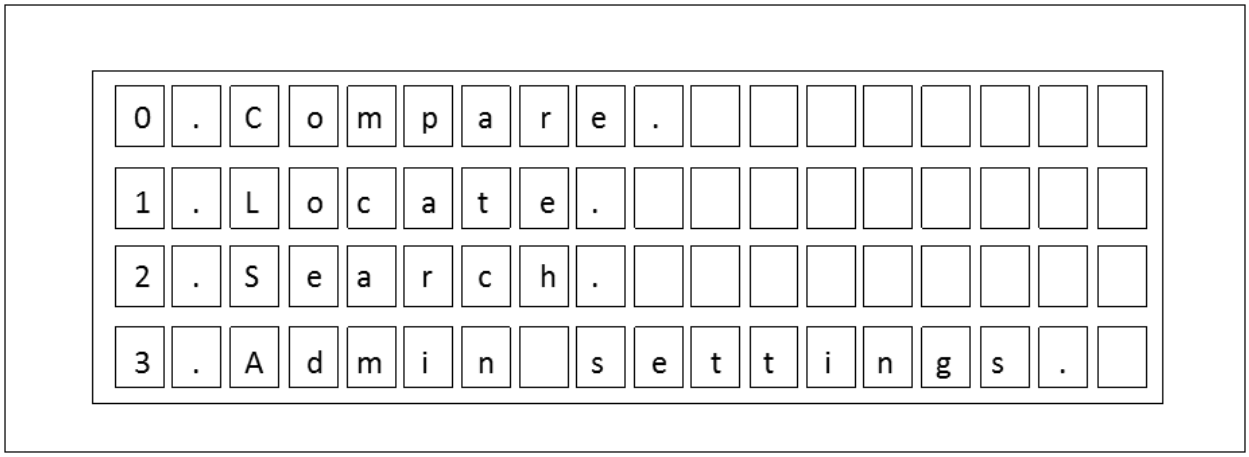

Figure 6: The layout of the options on the LCD.

2.2.4 Input Pins

These were an input that allowed the user to select which action the User wanted to access. 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 result-

ing in a different value being read from the pins. For example, pressing the last pin would change the output from the pins from $FF to $FE. An index of four different values could be looked up by the progrmam to determine which pin had been pressed hence which option the User had selected.

2.2.5 USART1 - On Board

The USART1 was set by the Atmega128 to be controllable through port D. Information received was placed in the USART buffer. The microprocessor was programmed to collect the data and to store it in various SRAM locations. This can be seen in figure 7.

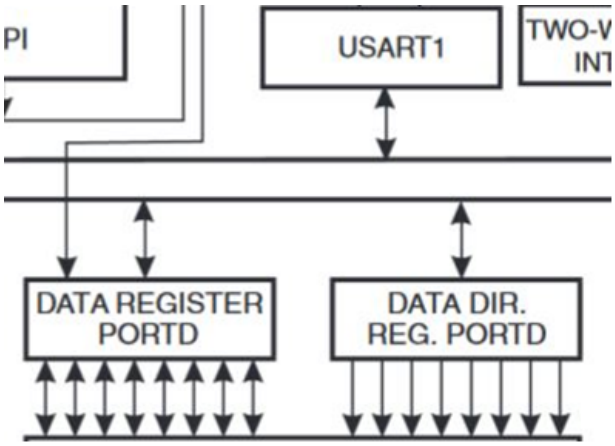

Figure 7: USART1 communication with Port D.

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 which on command performed the requested function.

2.3 Data Transfer

The fingerprint scanner itself interacted with microprocessor via asynchronous serial communication. This meant transmission and reception had to be a single bit at a time. Pin 2 was set up for reception and Pin 3 for transmission. For transmission the bits were sent manually via software, using the UART_send_byte subroutine, which enabled a serial signal to be sent through port D. For reception, the USART1 hardware was employed.

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. Subroutines were called that sent 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 and pin 2 was constantly polled for data. 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 into the buffer.

2.3.3 Setting BAUD rate

The rate at which bits were sent and received had to be set up by software. This had to be 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. 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.

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

2.4 Software

2.4.1 Initialisation

The Ports all 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 can be found in the code in Appendix E, lines 21-107. figure 8 gives an overview of the main stages of initialisation.

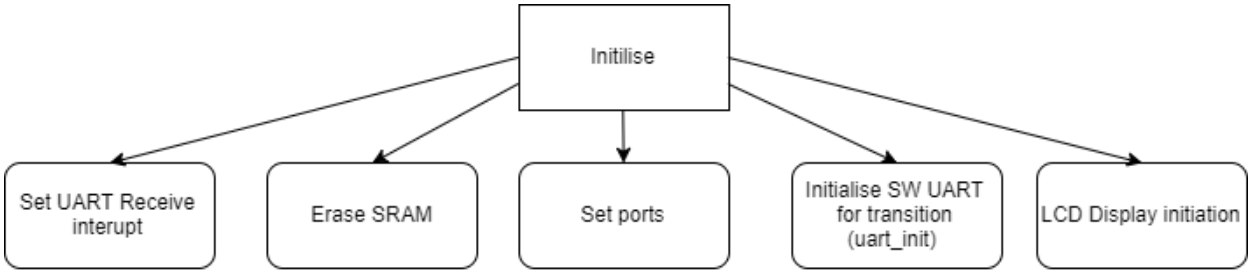

Figure 8: Mid level sketch of the initialisation

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 is action5, given in figure 9. This action was responsible for counting the number of stored Fingerprints.

```
;~~~~~~~~~~~~~~~~~~~ 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 telling the Fingerprint Scanner to count the number of Fingerprint currently stored on the device. The form of the command packages is provided in the datasheet for the Fingerprint Scanner.

```
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 sub-

routines 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 to 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 which 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 input recorded indicating the success of a matching had to be compared to a database. This database had a unique sequence that determined whether the input indicated a correct Match. Comparing the input sequence to that of a sequence of a correct Match allowed the program to fully recognize whether the scanner had found a Match for the fingerprint. This was a relatively long 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. The acknowledgement package always included a byte of confirmation code that would verify a successful task or an inability to read the Command Package. These codes were given in the datasheet and throughout the production of the overall product, these were saved to SRAM. The various locations in SRAM where these were saved were 20 locations apart, the various locations can be viewed in figure 14. Several locations were required as several commands were sent so each step needed to be verified with a separate confirmation, each of which had to be stored in SRAM independently.

```
.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

Action 1 attempted to take two fingerprints and compare them for similarity. The use of this would be a quick test to see if someone had successfully forged a fingerprint.
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 was a function that stored a fingerprint and allocated it a specific memory location. Action 3 and 4 can be seen in figure 16.
Action 5 requested a total for the number of used memory locations, this required one sent package and 1 received package. Action 6 was a simple delete function. It sent one Command Package to the device to delete memory location 0 to 15.

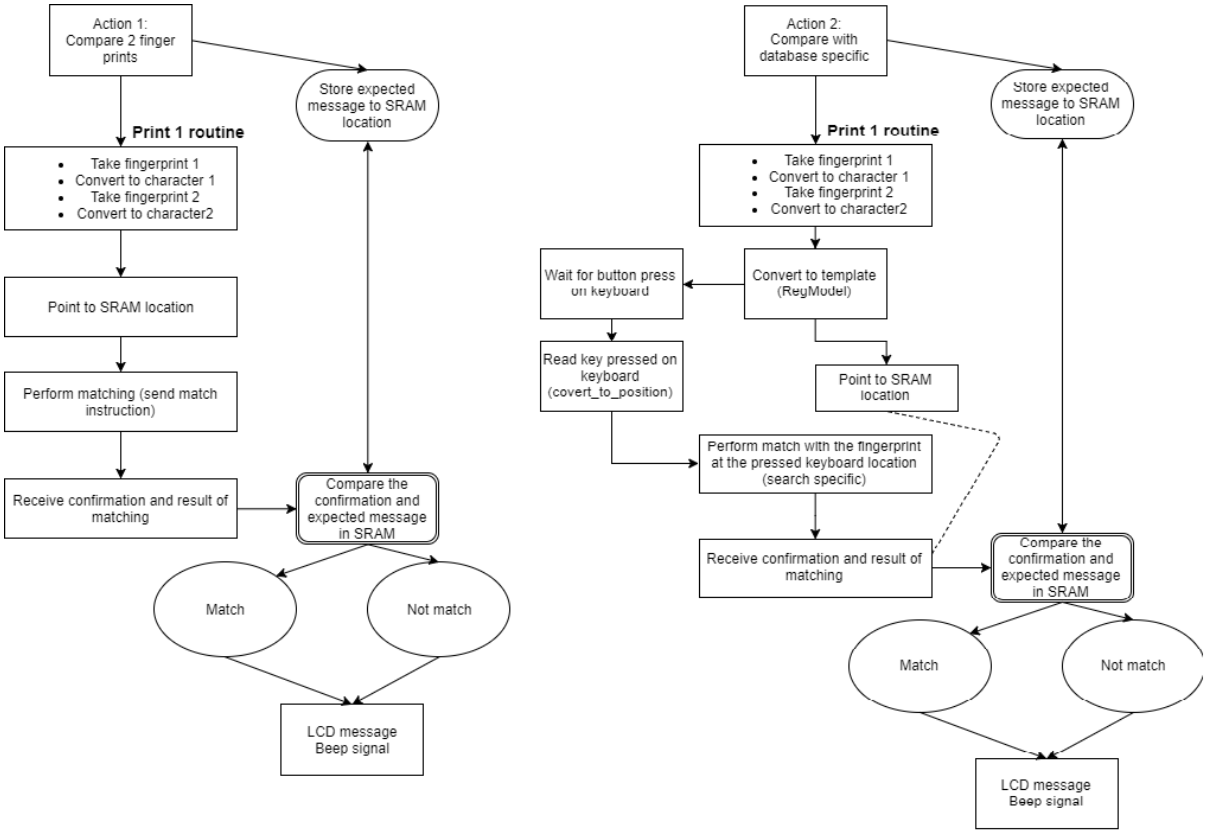

Figure 15: Flow Charts for the Compare and Locate functions.

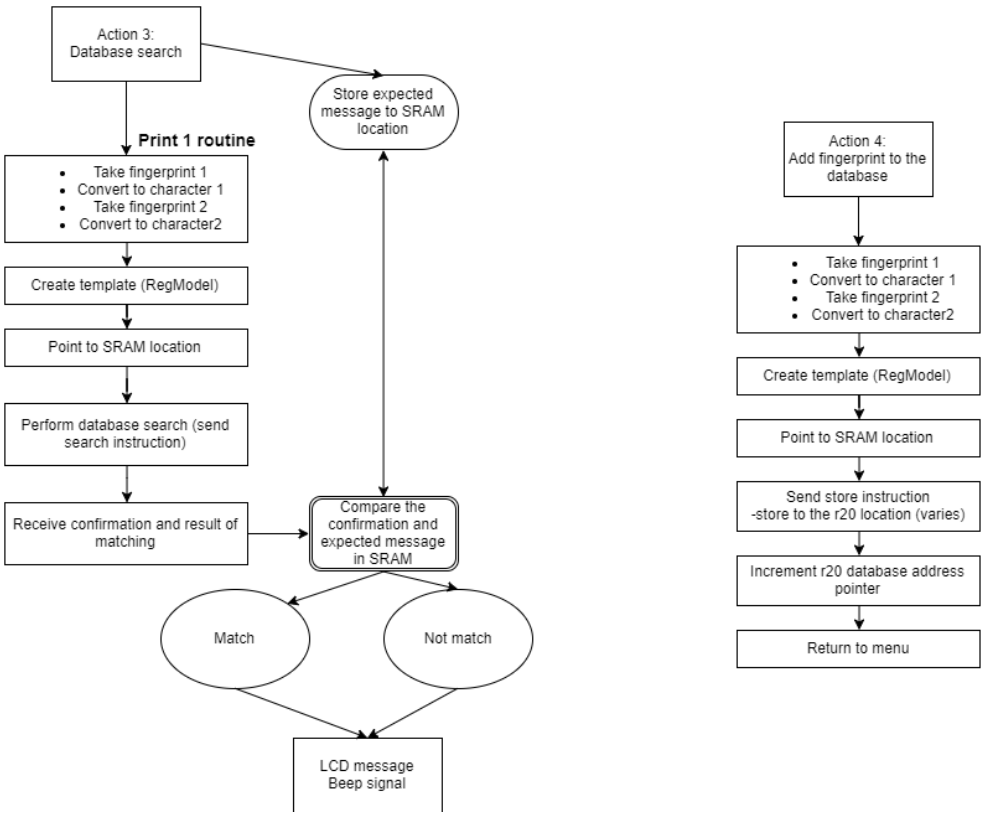

Figure 16: Flow Charts for the Search and Store functions.

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 for was measured for different fingers to check the variation of different sizes and type of finger. This concluded in the observation 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% accuracy. It struggled more with the fourth and fifth fingers with less than 70% accuracy, compromising the reliability of the scanner. The results of the investigation can be viewed in figure 17.

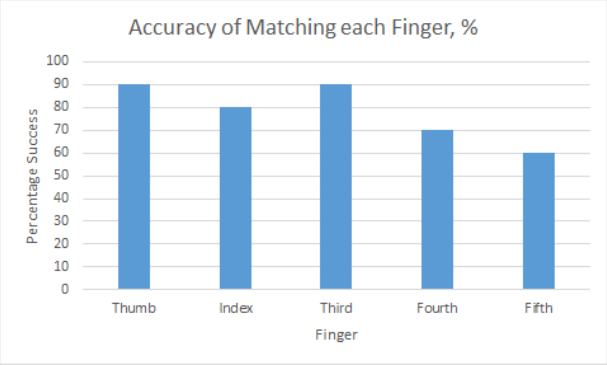

Figure 17: The varying reliability of matching different fingerprints.

The quoted failure rate from the datasheet of the fingerprint scanner was 1% however, 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. The variation in size of the fingerprint appeared to be factor from experiment. Testing for 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. This was also provided as 0.001% or 1 out of 100,000 by the Scanner's datasheet. 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

All of the, time the Fingerprint scanner did exactly what the User told it to do. A complete set up of the device can be viewed in Appendix B. Action 1 worked successfully. It also allowed the program to continue the fingerprints were not presented in the time. In which case it 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. The general search of the Scanner's database was only limited by the speed of searching. According to the datasheet, the fingerprint scanner could take up to a second to search the database. This caused a significant delay in receiving a result. This would be difficult to resolve however without resorting to using another Scanner. Actions 4, 5 and 6 were successfully set up as administrative options. They required the User 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 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 wold be incremented, regardless of whether a fingerprint was taken or not. This could be resolved with modification. 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 cleared recently of non-permanent identities. The device was capable of clearing memory locations. Although the user could enter more fingerprints not accessible by the Keyboard i.e. beyond location 15. This function allowed these also to be removed. 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'.

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 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 uses a small area scanner that does not scan the whole fingerprint. This requires the same area of a finger to be presented every time. Previous investigation [3] has suggested that the area of the sensor is the most likely factor in

misidentifying a correct finger. As this was a small area Scanner, its was likely affected by this.

3.3.1 Effect of Moisture and dust

The most productive investigation would likely be a check the effect of a dry and wet print to determine if there was a noticeable difference. Previous study [1] has indicated that it may be a significant detractor form the reliability of the Scanner.

4 Updates, Modifications and improvements

4.1 Updates

Additional functions that can be added to the Product could involve a third screen to get a few more options. One that would be relatively easy to implement would be toggling the alarm. This would be the equivalent of a feature that disabled the security. This would just require some coding that would only call the setup of Port B as an output port in certain situations. Another feature would be to add a pass-code override for the admin settings when a fingerprint is not recognized. 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. One possible area of improvement could be the time between the User selecting an action and an output being displayed in the screen. Some of this can't be completely avoided as it takes roughly one second, according to the datasheet, to carry out full search. 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 load screen where a set of blocks would incrementally be displayed on the screen according to an estimate of progress.

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 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. A method could be that someone would have to take multiple prints for a verification. This would involve a loop that kept taking prints and comparing them. 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 reduce the chance of false rejection to 16% from 40% for the little finger, assuming 2 cycles could be implemented in the time.

5 Conclusion

We managed to complete the project to the point where the device could alert of a possible intruder via a buzzer as was set out. A slight variation was that we made it more of a user experience where they could interact with the device in several ways however it could also strictly be used as a security device.

References

[1] Lee S Son G Back S, Lee Y. Moisture-insensitive optical fingerprint scanner based on polarization resolved in-finger scattered light. *Optics Express*, 24(17), 2016.

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

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

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

Specification

Number of keypad accessible prints	15
Maximum Capacity	162
Function	**Times**
Compare	1s
locate	<2s from Keypad Input
Search Database	<3s
Store	<2s
Template Number 5	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	82.6
Image Capture	CCD
Storage	Flash

Appendices

A Complete Representation of the Atmega128 by Pin