

# Driver for Fingerprint Scanner

Nelson Talukder  
CID 01055592

Blackett Laboratory, Imperial College London  
Word Count: 3968

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. 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.

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].

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 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 or scattered at the boundary between the ridge and the glass. This meant reflected light would show the pattern of ridges as dark lines.

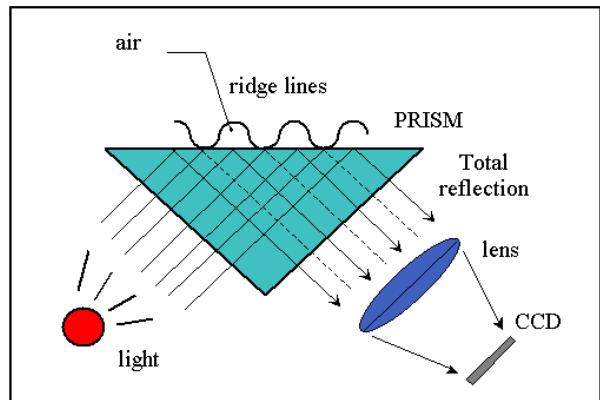


Figure 1: The reflection of light from the of a finger ridge compared to a groove.

The Scanner analysed certain features, or minutiae [2], in the image of the finger. Minutiae include places where the ridges meet or end. 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

## 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 function was removed when the LCD display was initialised. Constant reference to the atmega128 datasheet helped to ensure that every step could be

suggested as a reason for limits reliability in matching.

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.

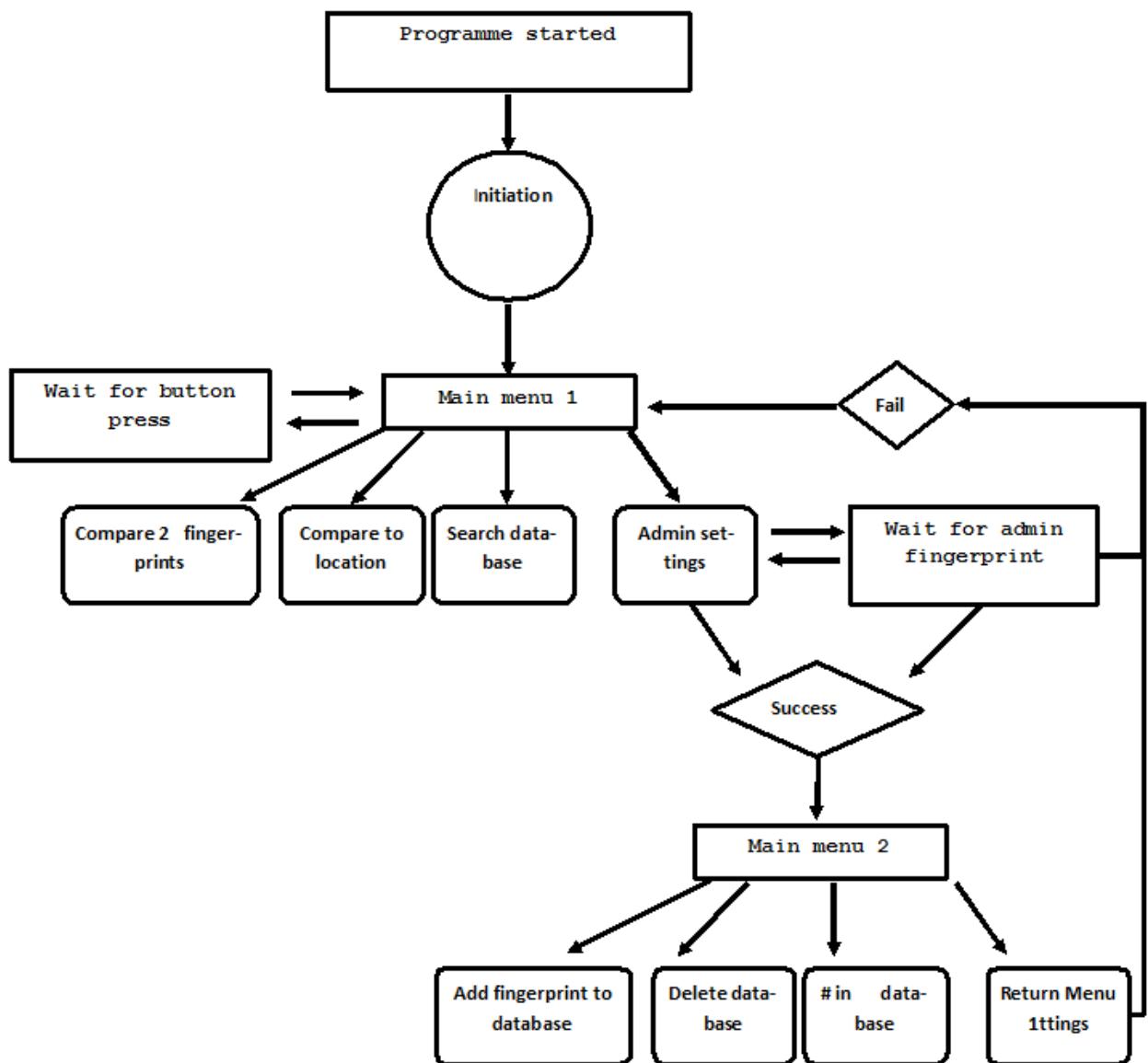


Figure 2: Flow map of the entire operation of the device.

## Overall Diagram

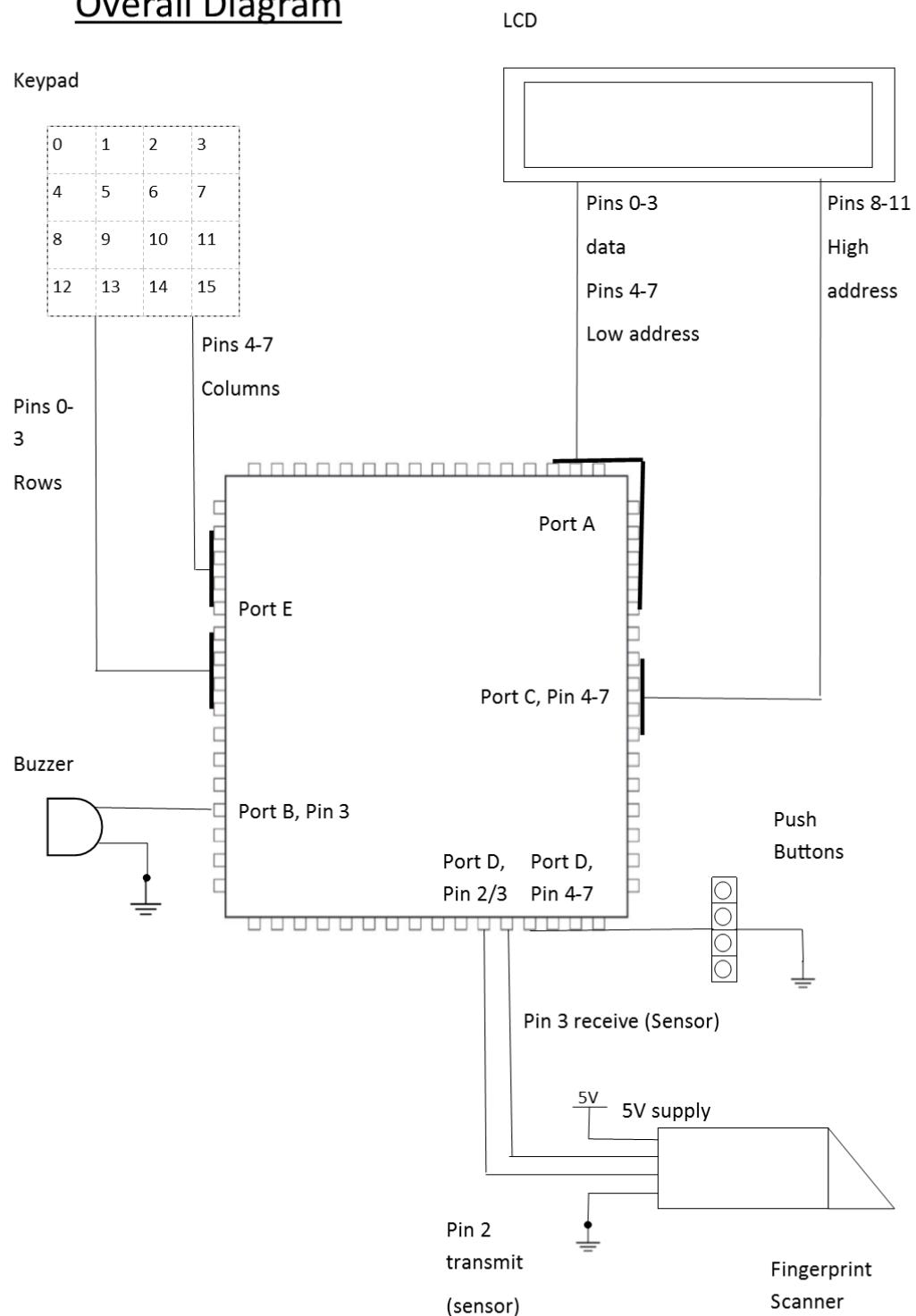


Figure 3: Diagram of overall setup. The full operation of each pin can be viewed in figure 4.

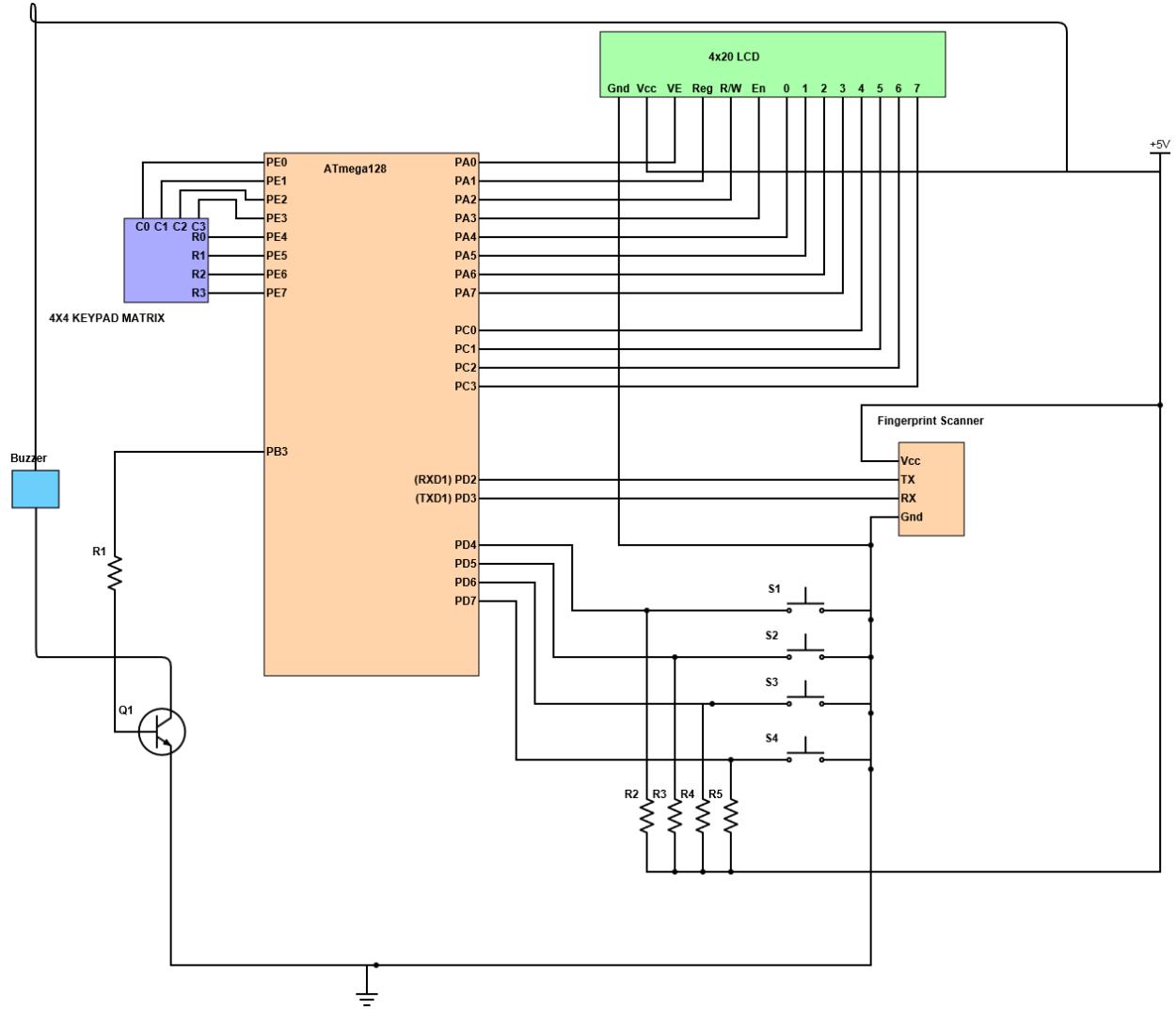


Figure 4: Schematic of the hardware used.

## 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 two values were added together by software and saved.

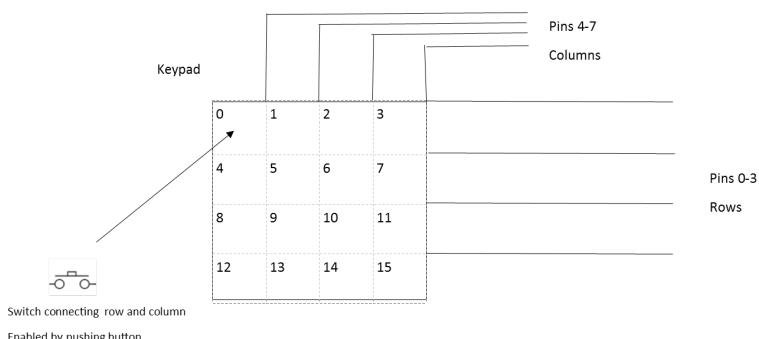


Figure 5: Diagram of the Keypad used.

### 2.2.2 Buzzer

The relay circuit for the buzzer was connected to pin 3, Port B. Software controlled whether it received a DC voltage signal, enabling the buzzer 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

A Hitachi LCD with a  $20 \times 4$  display was used. 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.

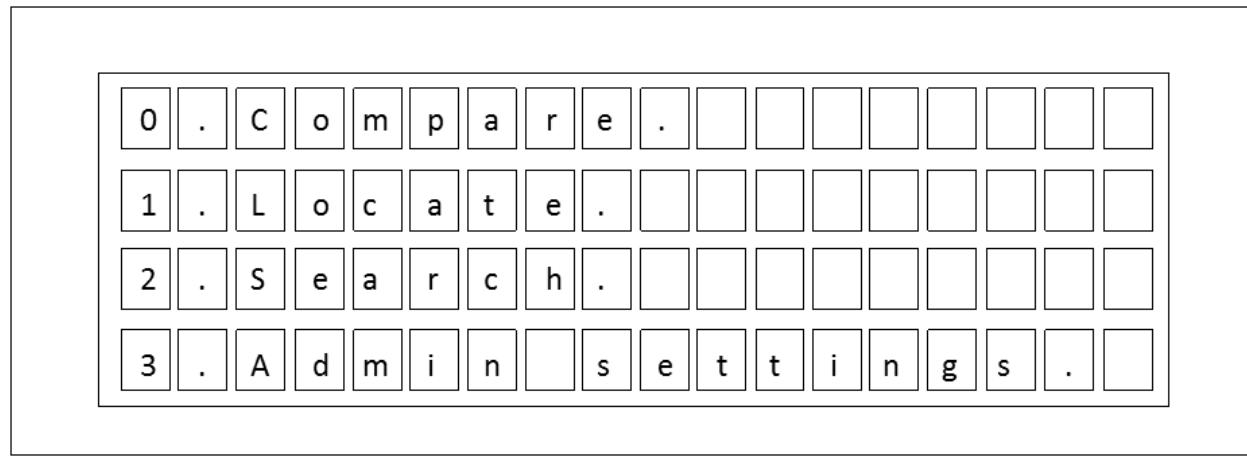


Figure 6: The layout of the options on the LCD.

### 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 input for the whole of 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.

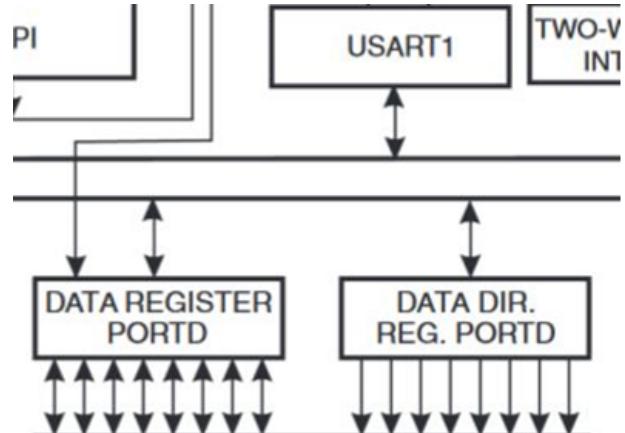


Figure 7: USART1 communication with Port D.

### 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.

### 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 completed a requested procedure on command.

## 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 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 where the Atmega128 board had to be prepared at the correct 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 automatically 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 frequency 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.

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

## 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.

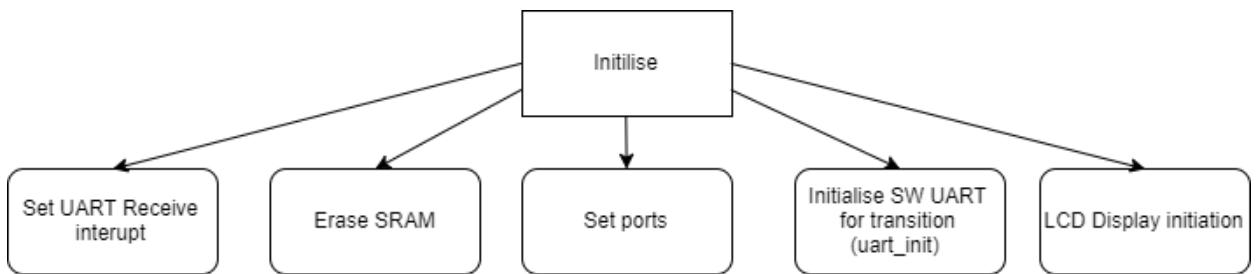


Figure 8: Sketch of the initialisation

### 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 sep-

arate .include file. Actions 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 acknowledgement 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: Action 5, 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
    MOV r17, r0
    mov r23,r17
    rcall sendUART
    rcall delaycycles
    |
    |
    DEC r18
    cpi r18, $0
    breq Mess1End
    ADIW ZL, $01
    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)
    sts UCSR1B, r16
;
    ldi r16, (1<<USBS1) | (3<<UCSZ10)
    sts UCSR1C, r16
;
    push r16
    push r17
    ldi r16, $08
    ldi r17, $00
    sts UBRR1H, r17
    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 memory 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.

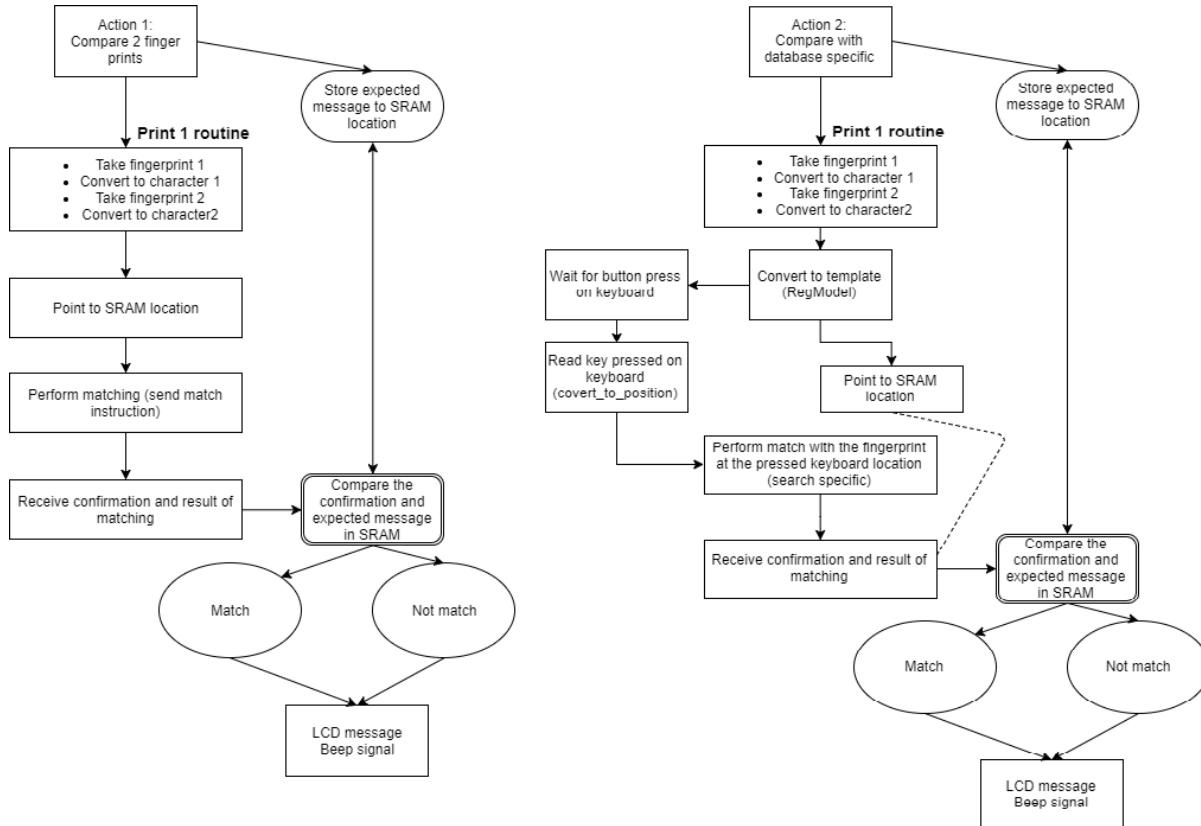


Figure 15: Flow Charts for the Compare and Locate functions.

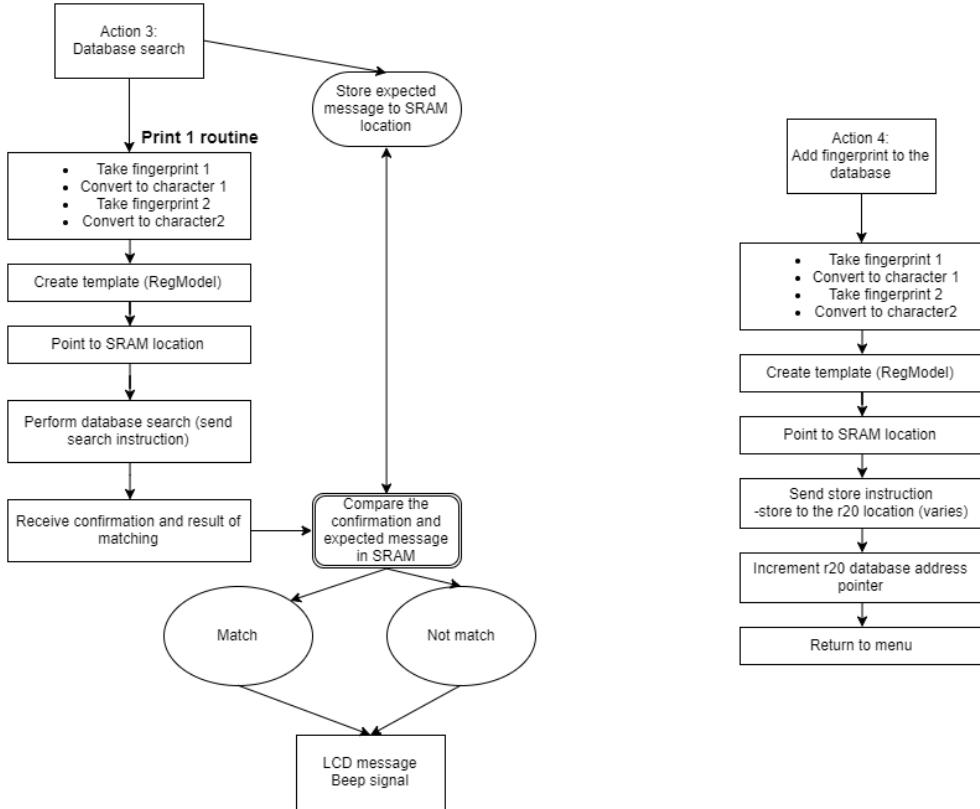


Figure 16: Flow Charts for the Search and Store functions.

### 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 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.

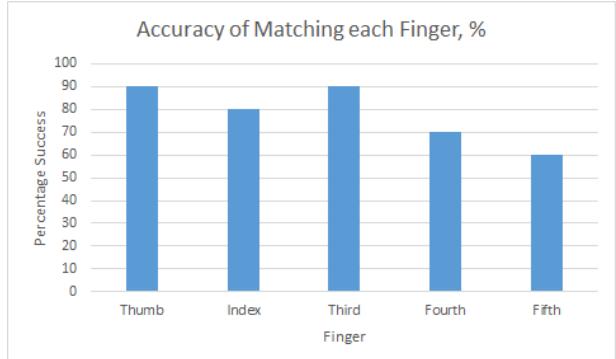


Figure 17: The varying reliability of matching different fingerprints.

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 <sup>1</sup>. 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 6 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 6. 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 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

<sup>1</sup>See 4.2 Modifications.

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 outcome 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, more reliable 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

## Specification Table

Number of keypad accessible prints	15
Maximum Capacity	120 [6]
<b>Function</b>	<b>Times</b>
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 (Attached to Scanner)

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 successfully implemented and tested. The reliability was investigated and clear methods for improvement identified.

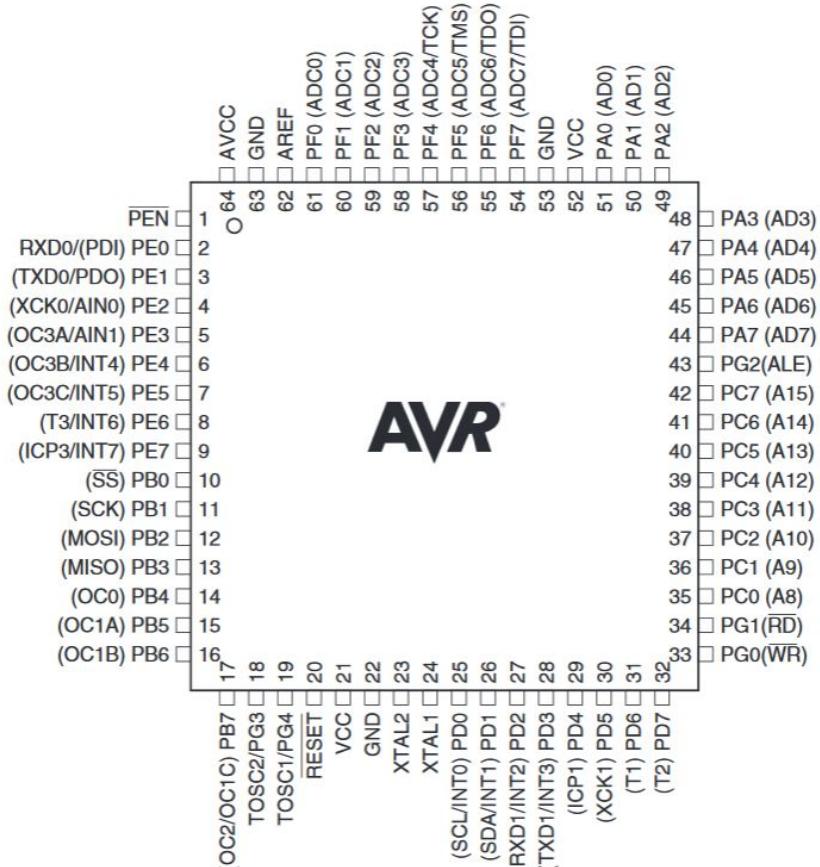
## References

- [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](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/](http://wiki.seeed.cc/Grove-Fingerprint_Sensor/).

# Appendices

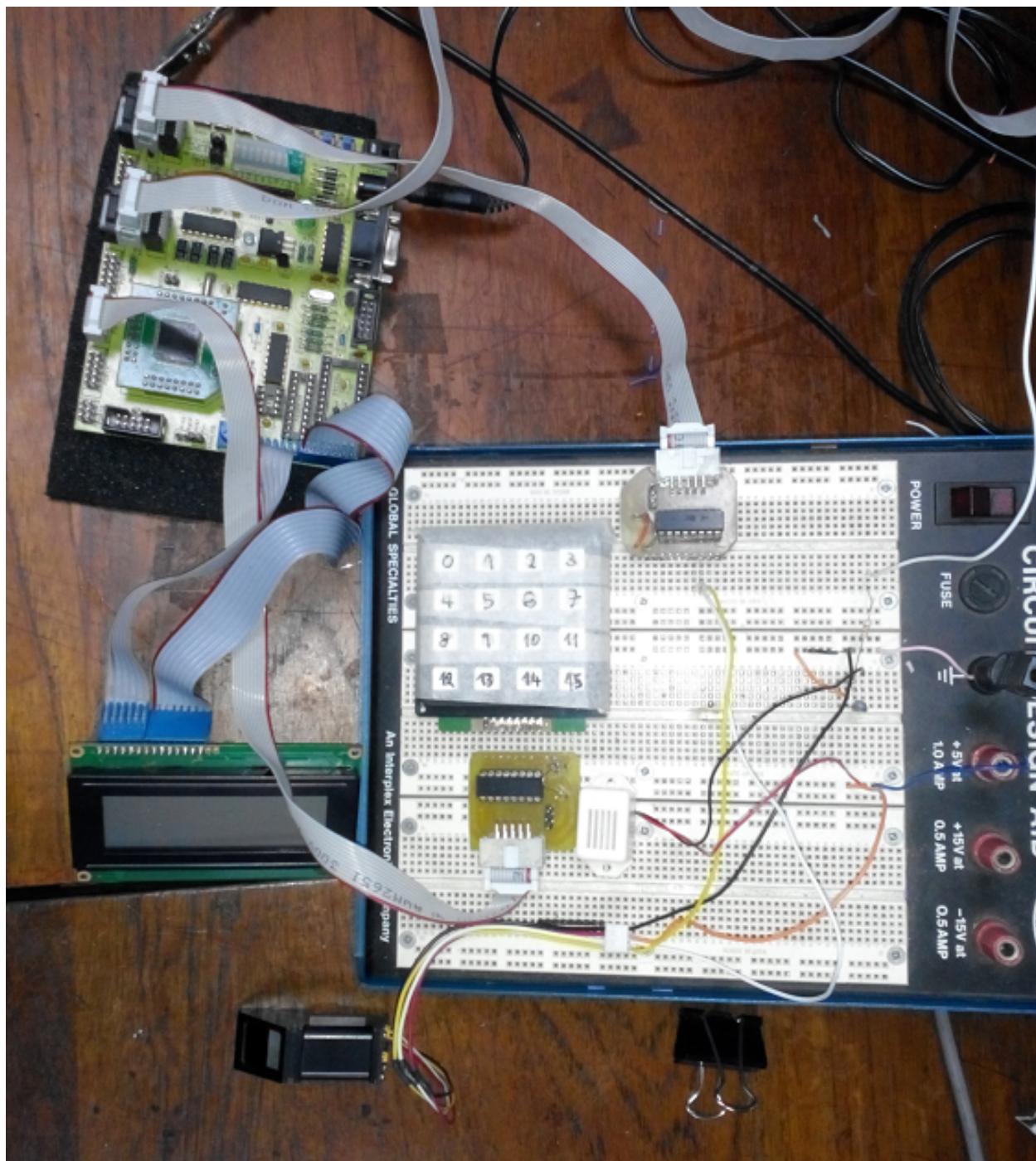
[h!]

## A Complete Representation of the Atmega128 by Pin [5]



[h!]

## B Photograph of the Entire configuration



## C Screen Shots of the various Load Screens



[h!]

## D Actions code, Comments formatted in C as Assembler not recognised by LaTeX

```

69  push r19
70  push r23
71  ldi r23, 0
72  ldi r19, 2
73  ldi r18, 11
74  LDI ZH, HIGH(search2*2)      //Sends standard first 8 bytes
75  LDI ZL, LOW(search2*2)
76  rcall Mess1Out
77  lds r25, 0x0808           //R25 has the ID ($xx) of the memory location
78 repeat:
79  mov buffer,r23            //send $00 $xx twice
80  rcall sendUART           //Send $00 (Byte 9), to indicate Buffer1
81  rcall UART_delay
82  mov buffer,r25
83  rcall sendUART           //Send ID (= first memory location to be searched)
84  rcall UART_delay
85  dec r19
86  brne repeat              //Send ID (= Last memory location to be searched)
87 norrepeat:
88  mov buffer,r23
89  rcall sendUART           //Send $00 $xx (Checksum)
90  lds r25, 0x0809           //Send correct calculated value
91  mov buffer,r25
92  rcall sendUART           //Second byte of checksum
93  rcall UART_delay
94  pop r23
95  pop r19
96  pop r25
97  ret
98
99
100 convert_to_position:     //Convert KEypad Signal to position (0-15)
101   //r21 signal from Keypad
102  push r23
103  Push r24
104  push r21
105  //column
106  sbrs r21,0               //Standard, Check each bit in signal
107  ldi r23,$0
108  sbrs r21,1               //Depending on Column, add (0,1,2,3)to base-
109  ldi r23,$1
110  sbrs r21,2
111  ldi r23,$2
112  sbrs r21,3
113  ldi r23,$3
114
115  //row
116  sbrs r21,4
117  ldi r24,$0
118  sbrs r21,5
119  ldi r24,$4
120  sbrs r21,6
121  ldi r24,$8
122  sbrs r21,7
123  ldi r24,$C
124
125  //add
126  add r23,r24
127  mov r21,r23
128  sts 0x808, r21          //R21 is the hex position between $00-$0F-
129  pop r21
130  pop r23
131  pop r24
132  ret
133
134 waitforbuttonpress2:
135  ldi r21, $00             //high 4-bit
136  ldi r18, $00
137
138           // set keyboard out in
139  ldi r16, $F0             // load r16 with 0-3 input
140  out DDRE, r16            // Direction register
141  ldi r16, $0F             // 4-7 output
142  out PORTE, r16
143  rcall DEL49ms

```





```

294 rcall send_Store4          //Stores the fingerprint in the next location
295 rcall BigDEL
296
297 ret
298
299
300 Print1:
301 ldi YH, high(testregister9) //Generate image
302 ldi YL, low(testregister9)
303 rcall send_genimg
304 rcall BigDEL
305
306 ldi YH, high(testregister) //360
307 ldi YL, low(testregister)
308 rcall send_img2tBuff1      //Convert to Character file , put in Buffer 1
309 rcall BigDEL
310
311 ldi YH, high(testregister9)
312 ldi YL, low(testregister9)
313 rcall send_genimg
314 rcall BigDEL
315
316 ldi YH, high(testregister1) //380
317 ldi YL, low(testregister1)
318 rcall send_img2tBuff2      //Convert to Character file , put in Buffer 2
319 rcall BigDEL
320 ret
321
322 //~~~~~ Action 5 ~~~~~
323 //deletes database , memory 1-15 addresses
324 action5:
325 ldi YH, high(testregister9) //3E0 //Point to location to save acknowledgement package
326 ldi YL, low(testregister9) //This meant the USART interrupt knew where to store
327 rcall send_DeleteChar //Deletes values between ID values 0 and 15
328 ldi r25, 2 //Reload the place from where to start counting the ID
329 rcall bigdel
330 ret
331
332
333 //aaaaaaaaaaaaaaaaaaaaaaa Action 6 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
334 //returns number of existing stored fingerprints in the memory of scanner
335 action6:
336 ldi YH, high(testregister6) //400
337 ldi YL, low(testregister6)
338 rcall send_TempplateNum //Gets the number of stored fingerprints
339 rcall bigdel
340 push r16
341
342 lds r16, 0x040B
343 sts 0x03EB, r16
344 rcall binary_to_decimal //Display the number
345 rcall bigdel
346 rcall clrdis
347 pop r16
348 ret

```

[h!]

## E Main code

```
1 // AssemblerApplication3.asm
2 //
3 // Created: 03/11/2017 16:31:37
4 // Authors : Nelson Talukder and Tomas Jirman
5 //
6
7
8 .DEVICE ATmega128
9 .include "m128def.inc"
10
11
12 //#####
13 jmp Init
14 // INTERRUPT:
15 .org $003c
16 rjmp save_data_start352      // UART Receive Vector
17 .org $0080      // start address above interrupt table
18
19 //#####
20 .include "action4include.asm"
21 init:           // system init routine
22 .dseg
23 delaysaved: .byte 1      //assign memory for clock speed value
24 .cseg
25
26 //Locations in SRAM that are used in the programme
27 .equ compare_address = 0x0300 //DATABASE to SRAM, saves to here
28 .equ savedregister = 0x0320
29 .equ testregister = 0x0340      // all used in interrupt
30 .equ testregister1 = 0x0360
31 .equ testregister2 = 0x0380
32 .equ testregister3 = 0x03A0
33 .equ testregister4 = 0x03C0
34 .equ testregister5 = 0x03E0
35 .equ testregister6 = 0x0400
36 .equ testregister7 = 0x0420
37 .equ testregister8 = 0x0440
38 .equ testregister9 = 0x0460
39
40 .equ r = 18          //Setting the no. of delay cycles
41
42
43 // ***** MEMORY ERASER *****
44 // It is crucial for the operation of device to start with clear memory where
45 // data is stored as response from the finger print scanner.
46
47           // erases memory from 0x0300 to 0x0900
48 eraser:
49   ldi XH, high(compare_address)
50   ldi XL, low(compare_address)
51 loop:
52   push r20
53   ldi r20, $01
54   st X+, r20
55   cpi XH, $09
56   brne loop
57   pop r20
58
59 //##### PORT SET UPS #####
60
61
62 ldi r16, $FF          //Set-up Port B as Output
63 out DDRB, r16
64 ldi r16, 00
65 out PORTB, r16
66
67
68 ldi r16, $0F          // Stack Pointer Setup
69 out SPH, r16          // Stack Pointer High Byte
70 ldi r16, $FF          // Stack Pointer Setup
```

```

71    out SPL, r16           // Stack Pointer Low Byte
72
73 // ***** RAMPZ Setup Code **** lower memory page arithmetic
74 ldi r16, $00             // 1 = EPLM acts on upper 64K
75 out RAMPZ, r16          // 0 = EPLM acts on lower 64K
76
77
78 // ***** Enable External SRAM i.e the LCD *****
79 ldi r16, $C0              // Set 11000000, i.e enable read/write
80 out MCUCR, r16           // External SRAM Enable Wait State Enabled
81
82 // ***** Comparator Setup Code ****
83 ldi r16, $80              // Comparator Disabled, Input Capture Disabled
84 out ACSR, r16             // Comparator Settings
85
86
87 //////////////// USART SET UP /////////////////////////////////
88 ldi r16, (1<<RXEN1)|(1<<RXCIE1) //enables interrupt
89 sts UCSR1B, r16
90 // Set frame format: 8data, 2stop bit
91 ldi r16, (1<<USBS1)|(3<<UCSZ10)
92 sts UCSR1C, r16
93 push r16
94 push r17
95 ldi r16, $08
96 ldi r17, $00
97 sts UBRR1H, r17
98 sts UBRR1L, r16
99 pop r17
100 pop r16
101
102 ldi YH, high(testregister)
103 ldi YL, low(testregister)
104 sei
105
106 //((((((((((((((((((((( )))))))))))))))))))))))))
107 rjmp main
108 //((((((((((((((((((((( )))))))))))))))))))))))))))))
109
110
111 // ----- DATABASE OF MESSAGES AND COMMANDS TO THE SCANNER-----
112 ReadSysPara:
113 .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$03,$0f, $00, $13
114 //===== Admin Settings =====
115 TempalteNum:
116 .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$03,$1d, $00, $21 //no of templates
117 DeleteChar:
118 .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$07,$0C, $00, $00, $00, $0F, $00, $23
119 //=====
120
121 genimg:
122 .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$03,$01, $00, $05,$08,$00
123
124 img2tBuff1:
125 .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$04,$02,$1,$00,$08,$0
126
127 img2tBuff2:
128 .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$04,$02,$2,$00,$09,$0
129
130 regmodel:
131 .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$03,$05,$00,$09
132
133 Store4:
134 .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$06,$06,$01, $00
135
136 PreciseMatch:
137 .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$03,$03, $00, $07,$08,$00
138
139 Search:
140 .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$08,$04, $01, $00,$00 ,$00, $46, $00, $54, $00
141
142 search2:
143 .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$08,$04,$01
144
145

```

```

146
147
148 //&&&&&&&&&&&&&&&&&&&&&&Compare DATABASE&&&&&&&&&&&&&&&&&&
149
150 compare_database:
151     .db $EF, $01, $FF, $FF, $FF, $FF, $07,$00, $05, $00, $00, $0A //Used in database_to_SRAM
152
153 compare_database2:
154     .db $EF, $01, $FF, $FF, $FF, $FF, $07,$00, $07, $00, $00, $0A //Used in database_to_SRAM4
155
156
157 // LCD DATABASE

158
159 Match:
160     .db ' ', 'M', 'a', 't', 'c', 'h'
161
162 Different:
163     .db ' ', 'D', 'i', 'f', 'f', 'e', 'r', 'e', 'n', 't'
164
165 Tomas:
166     .db " Tomas"
167
168 Nelson:
169     .db " Nelson"
170
171 George:
172     .db " George"
173
174 Welcome:
175     .db " Welcome! Press Pins 0 to 3 to start."
176
177 Welcomemessage1:
178     .db " 0. Compare.           2. Search database. 1. Locate.           3. Admin settings. "
179
180 Welcomemessage2:
181     .db " 4. Store finger.      6. Full templates. 5. Delete.           7. Return.          "
182
183 ready:
184     .db " Enter ID number: "
185
186
187 ID_recognised:
188     .db " Access granted.   "
189
190 //
191 //set up assuming clock =8 MHZ, BAUD wanted is 57600,
192 //totals 30 cycles + 108 dlay cycles
193
194 //***** CYCLES FOR DELAY *****
195 delaycycles:
196     lds r16, delaysaved
197     rcall delaycycles1
198     ret
199
200 delaycycles1:
201     dec r16
202     brne delaycycles1
203     nop
204     ret
205
206 uart_init:           // initiation of SW UART
207     in r16, DDRD
208     sbr r16, (1<<3)      //set pin 3 as output
209     out DDRD, r16
210     sbi PORTD, 3
211     nop
212     nop
213     ldi r16, r           //really important helps get
214     sts delaysaved, r16  // correct No. of Cycles for frequency in specific location
215     ret
216
217     ldi r16, $01
218     ret
219 //*****

```



```

295
296 Welcome5:           //Second Screen with Admin Options
297   rcall clrdis
298   LDI ZH, HIGH(2*Welcomemessage2)
299   LDI ZL, LOW(2*Welcomemessage2)
300   LDI r18, 80
301   rcall Mess1More22
302   rcall bigdel
303   ret
304
305 READY2:             // Asking for Keypad Input
306   rcall clrdis
307   LDI ZH, HIGH(2*ready)
308   LDI ZL, LOW(2*ready)
309   LDI r18, 19
310   rcall Mess1More22
311   rcall bigdel
312   ret
313
314
315 message_ID_recognised:      // 'Access Granted'
316   rcall clrdis
317   LDI ZH, HIGH(2*ID_recognised)
318   LDI ZL, LOW(2*ID_recognised)
319   LDI r18, 19
320   rcall Mess1More22
321   rcall bigdel
322   ret
323
324 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX Beginning OF RETURN XXXXXXXXXXXXXXXXXXXXXXXX
325 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX Beginning OF RETURN XXXXXXXXXXXXXXXXXXXXXXXX
326
327
328 //*****COMPARING2*****
329
330 // in this routine we are going to compare 2 sram databases
331 // this is crucial routine for decitions about fullfillment of conditions and sucesses
332 // of finger print detections
333 compare_basic2:
334
335   push r17
336   push r20
337   push r21
338   push r18
339   ldi r18, 0          //counter
340   rcall Comp1Out2Z
341   rcall Comp1Out2Y
342   rjmp Comp1More2
343
344
345 Comp1Out2Z:           //the one we compare to
346
347   LDI ZH, HIGH(compare_address) //address for comparison
348   LDI ZL, LOW(compare_address)
349   ret
350 Comp1Out2Y:
351   ldi YL, low(testregister5) //saved Acknowledgement Package
352   ldi YH, high(testregister5)
353   ret
354
355
356 Comp1More2:
357   ld r20, Z+
358   ld r17, Y+
359
360   inc r18
361   cp r20, r17          // compare the two X, Z
362   brne statefalse2      //go to false message if not equal
363
364
365
366
367   cpi r18, 10          // compare when end the cycle
368   brne Comp1More2
369

```

```

370 Comp1End2:           //True , 1 beep + 'Different'
371
372    rcall matching
373    rcall beepon
374    rcall megadel
375    rcall CLRDIS
376    rcall bigdel2
377    pop r18
378    pop r21
379    pop r20
380    pop r17
381    ret
382
383 statefalse2:          //False Branch , 3 beeps + 'Match'
384    rcall differently
385    rcall megadel
386    rcall beepon
387    rcall bigdel2
388    rcall beepon
389    rcall bigdel2
390    rcall beepon
391    pop r18
392    pop r21
393    pop r20
394    pop r17
395    rjmp main2
396
397
398 //XXXXXXXXXXXXXXXXXXXXXXXXXXXX END OF RETURN      XXXXXXXXXXXXXXXXXXXXXXX
399 //XXXXXXXXXXXXXXXXXXXXXXXXXXXX END OF RETURN      XXXXXXXXXXXXXXXXXXXXXXX
400
401
402 //&&&&&&&&&&&&&&&&&&&&&& SENDING PACKAGES &&&&&&&&&&&&&&&&&&&
403 // the sending of packages works that the address to PM is stored in Z and then routine
   sends
404 // the message to the scanner using UART protocol
405
406 send_ReadSysPara:        //Subroutine to send CP asking for System Parameters
407    ldi r18, 12
408    LDI ZH, HIGH(ReadSysPara*2)
409    LDI ZL, LOW(ReadSysPara*2)
410    rcall Mess1Out
411    ret
412
413 send_TempleteNum:        //Send CP for template number
414    ldi r18, 12           //Load r18 with the number of bytes
415    LDI ZH, HIGH(TemplateNum*2) //Point the Z register to the correct database
416    LDI ZL, LOW(TemplateNum*2)
417    rcall Mess1Out         //call send routines
418    ret
419
420 send_DeleteChar:         //Send CP for Clear Memory
421    ldi r18, 16
422    LDI ZH, HIGH>DeleteChar*2)
423    LDI ZL, LOW>DeleteChar*2)
424    rcall Mess1Out
425    ret
426
427 send_genimg:             //Send CP for generate image
428    ldi r18, 12
429    LDI ZH, HIGH(genimg*2)
430    LDI ZL, LOW(genimg*2)
431    rcall Mess1Out
432    ret
433
434 send_img2tBuff1:          //Send CP for create character file , load to Buffer1
435    ldi r18,13
436    LDI ZH, HIGH(img2tBuff1*2)
437    LDI ZL, LOW(img2tBuff1*2)
438    rcall Mess1Out
439    ret
440
441 send_img2tBuff2:          //Send CP for create character file , load to Buffer2
442    ldi r18,13
443    LDI ZH, HIGH(img2tBuff2*2)

```

```

444 LDI ZL, LOW(img2tBuff2*2)
445 rcall Mess1Out
446 ret
447
448 send_Regmodel:           //Send CP for create a template from Character files
449 ldi r18,12
450 LDI ZH, HIGH(regmodel*2)
451 LDI ZL, LOW(regmodel*2)
452 rcall Mess1Out
453 ret
454
455 send_Store4:           // this one has to split the message and add variable (register)
   pointing
456 ldi r18,12           // to the variable memory address in the device -> to prevent
   overwriting
457 LDI ZH, HIGH(2*Store4) // THis adds fingerprint to the database of the scanner
458 LDI ZL, LOW(2*Store4)
459 rcall Mess1Out
460 sts 0x0952, r25
461           // now fine for 12
462           // send variable location
463 mov r23,r25
464 rcall sendUART
465 rcall delaycycles
466           //send r 25
467 push r25
468 ldi r25, 00
469 mov r23,r25
470 rcall sendUART
471 rcall delaycycles
472 pop r25
473
474 push r21           //contains the sum
475 mov r21,r25
476 sts 0x0954, r21      // this is checksum
477 subi r21,$F2          //add 14 to
478 sts 0x0954, r21      // this is checksum
479 mov r23,r21
480 rcall sendUART
481           //rcall delaycycles
482 sts 0x0956, r21
483 pop r21           //out portb, r25
484 inc r25
485 rcall bigdel
486 ret
487
488
489
490 send_PreciseMatch:
491 ldi r18,12
492 LDI ZH, HIGH(PreciseMatch*2)
493 LDI ZL, LOW(PreciseMatch*2)
494 rcall Mess1Out
495 ret
496
497 send_Search:
498 ldi r18, $11
499 LDI ZH, HIGH(Search*2)
500 LDI ZL, LOW(Search*2)
501 rcall Mess1Out
502 ret
503
504 //*****SEND BYTE BY BYTE, Transmission *****
505 //this routine sends byte by byte given message
506 //r23 used as buffer
507 Mess1Out:
508 push r17           //push to Stack
509
510 Mess1More:
511 LPM                 //Load the CP from the SRAM location
512 MOV r17, r0          //R0 loaded with first byte in CP by def
513 mov r23,r17          //The Manual UART buffer loaded with the value
514 rcall sendUART       //call routine to send byte
515 rcall delaycycles
516

```

```

517 DEC r18           //r18 loaded in previous code with no. of bytes
518 cpi r18, $0        //deincremented until no bytes left to send
519 breq Mess1End     //End when all bytes are sent
520 ADIW ZL, $01       //Move onto next byte in CP
521 rjmp Mess1More
522 Mess1End:
523 pop r17           //return previous stack value
524 ret
525
526 //***** bit by bit *****
527 // This sends one byte using SW UART
528 //r23 used as buffer
529
530 sendUART:
531 rcall start
532
533 next:
534 brcc beginsend    //goes to send a 1, if 0, due to com
535 cbi PORTD, 3       //clears bit for sending
536 rjmp uart_wait
537 rcall beginsend
538
539 uart_wait:
540 rcall delaying
541 nop
542 nop
543 nop
544 lsr r23           //move onto next bit to send
545 cpi r24, 0          //reduce counter
546 brne next          //branches if not counted to 0
547 last:
548 sbi PORTD, 3       //send final stop bit
549 ret                // return to send next byte
550
551 //+++++ //delay routine
552 delaying:
553 rcall delaycycles
554 rcall delaycycles
555 ret
556
557 beginsend:
558 sbi PORTD, 3
559 nop
560 ret
561
562 start:
563 ldi r24, 10         //counter that has 2 extra bits for the start and end of each
      byte
564 com r23             //com of signal to be sent, set in Mess1out
565 sec                 //sets carry flag in SREG, to allow transmission
566 ret
567 //***** LCD Routines *****
568 //This Routine is copied from lectures:
569 //Display Initialization routine
570
571 Idisp:
572 RCALL DEL15ms        // wait 15ms for things to relax after power up
573 ldi r16, $30          // Hitachi says do it...
574 sts $8000,r16
575 RCALL DEL4P1ms        // Hitachi says wait 4.1 msec
576 sts $8000,r16          // and again I do what I'm told
577 rcall Del49ms
578 sts $8000,r16          // here we go again folks
579 rcall busylcd
580 ldi r16, $3F          // Function Set : 2 lines + 5x7 Font
581 sts $8000,r16
582 rcall busylcd
583 ldi r16, $08          //display off
584 sts $8000, r16
585 rcall busylcd
586 ldi r16, $01          //display on
587 sts $8000, r16
588 rcall busylcd
589 ldi r16, $38          //function set
590 sts $8000, r16

```

```

591 rcall busylcd
592 ldi r16, $0E           //display on
593 sts $8000, r16
594 rcall busylcd
595 ldi r16, $06           //entry mode set increment no shift
596 sts $8000, r16
597 rcall busylcd
598 clr r16
599 ret
600
601 //*****
602 // This clears the display so we can start all over again
603 // given in lectures:
604 CLRDIS:
605 ldi r16,$01           // Clear Display send cursor
606 sts $8000,r16         // to the most left position
607 ret
608 //*****
609 // A routine the probes the display BUSY bit
610 // given in lectures
611
612 busylcd:
613 lds r23, $8000         //access
614 sbrc r23, 7            //check busy bit 7
615 rjmp busylcd
616 ret                     //return if clear
617
618 //*****
619
620 Matching:              //Called by comparebasic2, Output 'Match' to LCD
621 LDI ZH, HIGH(2*match)
622 LDI ZL, LOW(2*match)
623 LDI r18, 6              //number of characters or bytes
624 Mess1More22:           //Standard write to LCD code
625 LPM
626 MOV r17, r0
627 sts $C000, r17
628 rcall busylcd
629 DEC r18
630 BREQ Mess1End22
631 ADIW ZL, $01
632 RJMP Mess1More22
633 Mess1End22:
634 ret
635
636
637 differently:           //Called by comparebasic2, Output 'Different' to LCD
638 LDI ZH, HIGH(2*different)
639 LDI ZL, LOW(2*different)
640 LDI r18, 10             //number of characters or bytes
641 rcall Mess1More22
642 ret
643
644
645
646
647 //***** DELAY ROUTINES *****
648 BigDEL2:
649 rcall Del49ms
650 rcall Del49ms
651 rcall Del49ms
652 rcall Del49ms
653 rcall Del49ms
654 rcall Del49ms
655 rcall Del49ms
656 rcall Del49ms
657 rcall Del49ms
658 rcall Del49ms
659 rcall Del49ms
660 rcall Del49ms
661 rcall Del49ms
662 rcall Del49ms
663 rcall Del49ms
664 rcall Del49ms
665 rcall Del49ms

```

```

666    r call Del49ms
667    ret
668
669 BigDEL:
670    r call BigDEL2
671    r call BigDEL2
672    ret
673
674 BigDEL3:
675    r call Del49ms
676    r call Del49ms
677    r call Del49ms
678    r call Del49ms
679    r call Del49ms
680    ret
681
682 megadel:
683    r call BigDEL2
684    r call BigDEL2
685    r call BigDEL2
686    r call BigDEL2
687    r call BigDEL2
688    ret
689
690 DEL15ms:
691    LDI XH, HIGH(19997)
692    LDI XL, LOW (19997)
693 COUNT:
694    SBIW XL, 1
695    BRNE COUNT
696    RET
697
698 DEL4P1ms:
699    LDI XH, HIGH(5464)
700    LDI XL, LOW (5464)
701 COUNT1:
702    SBIW XL, 1
703    BRNE COUNT1
704    RET
705
706 DEL100mus:
707    LDI XH, HIGH(131)
708    LDI XL, LOW (131)
709 COUNT2:
710    SBIW XL, 1
711    BRNE COUNT2
712    RET
713
714 DEL49ms:
715    LDI XH, HIGH(65535)
716    LDI XL, LOW (65535)
717 COUNT3:
718    SBIW XL, 1
719    BRNE COUNT3
720    RET
721
722
723
724
725 //%%%%%%%%%%%%% Interrupt %%%%%%%%%%%%%%
726
727 save_data_start352:
728    push r17
729    lds r17, UDR1
730    st Y+, r17           // store byte received in USART1 to memory location
731    pop r17
732    reti
733
734
735 //%%%%%%%%%%%%% Calling Actions %%%%%%%%%%%%%%
736
737
738 // ~~~~~ WAIT FOR BUTTION PRESS AND MOVE SOMWEHRE ~~~~~
739
740 waitforbuttonpress:

```

```

741 rcall waitforbuttonpresscycle
742 rcall FirstScreen
743 ret
744 waitforbuttonpresscycle: //Subroutine to wait for Pin input
745 in r21, PIND
746 cpi r21, $FF
747 breq waitforbuttonpresscycle
748 rcall clrdis
749 ret
750
751
752
753 FirstScreen:
754                 //here have list of conditions
755 cpi r21,$7F          // pin 1
756 breq action1call      // ACTION 1
757
758 cpi r21, $BF          // PIN2
759 breq action2call      // ACTION 2
760
761 cpi r21, $DF          // PIN3
762 breq action3call      //Action 3
763
764 cpi r21, $EF          // PIN4
765 breq SecondScreen     //Action 4
766
767 ret
768
769 SecondScreen:
770 rcall verification     //Required: Verify ID for Admin Access
771 correct:
772 rcall Welcome5
773 rcall waitforbuttonpresscycle
774                 //here have list of conditions
775 cpi r21,$7F          // pin 1
776 breq action4call      // ACTION 1
777
778 cpi r21, $BF          // PIN2
779 breq action5call      // ACTION 2
780
781 cpi r21, $DF          // PIN3
782 breq action6call      //Action 3
783
784 cpi r21, $EF          // PIN4
785 breq action7call      //Action 4
786 rjmp main2
787
788
789 actions:
790
791 action1call:
792 rcall action1
793 rjmp main2
794 action2call:
795 rcall action2
796 rjmp main2
797 action3call:
798 rcall action3
799 rjmp main2
800 action4call:
801 rcall action4          //included
802 rjmp main2
803 action5call:
804 rcall action5
805 rjmp main2
806 action6call:
807 rcall action6
808 rjmp main2
809 action7call:
810 rjmp main2
811
812 /%%%%%%%%%%%%% Verification %%%%%%%%%%%%%%
813 Verification:          //Checks 0x0750 to see if Password subroutine-
814 rcall action3          //was reached
815 lds r16, 0x0750

```

```

816    cpi r16, 22
817    breq continue
818    rjmp main2           //Jumps to Screen1 if not
819    ret
820
821 continue:           //Called if 0x0750 has been changed by Password
822    rcall message_ID_recognised
823    rcall clear750
824    rcall bigdel
825    rjmp correct
826
827
828
829
830 //^^^^^^^^^^^^^^^^^^^^^^^^^DATABASE TO SRAM MATCH ^^^^^^^^^^^^^^^^^^^^^^
831
832 DATABASE_TO_SRAM:          //saves to 0x0300
833    push r20
834
835 makingZolocation:
836    ldi ZH, HIGH(compare_database*2) //leave as 16 bit
837    ldi ZL, LOW(compare_database*2)
838    rcall step_by_step_LOADING
839    pop r20
840    ret
841
842 //^^^^^^^^^^^^^^^^^^^^^^^^^DATABASE TO SRAM SEARCH ^^^^^^^^^^^^^^^^^^^^^^
843
844
845 DATABASE_TO_SRAM4:         //saves to 0x0300
846    push r20
847
848 makingZolocation4:
849    ldi ZH, HIGH(compare_database2*2)//leave as 16 bit
850    ldi ZL, LOW(compare_database2*2)
851    rcall step_by_step_LOADING
852    pop r20
853    ret
854
855 //<<<<<<<<<< Subroutine for databases >>>>>>>>>>>>>>>>>>>
856 step_by_step_LOADING:      // this stores given message from PM (Z) to the SRAM
location (X)
857    ldi r20, 12            //counter
858    ldi XH, high(compare_address) //create 16 bit address? //0x0300
859    ldi XL, low(compare_address) //Leave as 8 bit
860
861 program_to_memory:
862    lpm
863    ADIW ZL, $01
864    dec r20
865
866    st X+, r0              //don't need sts, if using X
867
868    cpi r20, $0             //end if counted
869    brne program_to_memory
870    ret
871
872 //?????????????????????????????????????????????????????????????????????????
873                                //rings once for correct finger
874                                //three times incorrect (see Statefalse)
875
876 beepon:                  // turns on beep alarm - sound generator
877    ldi r17, $FF
878    out portB, r17          //Port B used to output signals
879    rcall del49ms
880    rcall del49ms
881    rcall del49ms
882    rcall del49ms
883    rcall del49ms
884    rcall del49ms
885    ldi r17, $00
886    out portB, r17
887    ret

```