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.

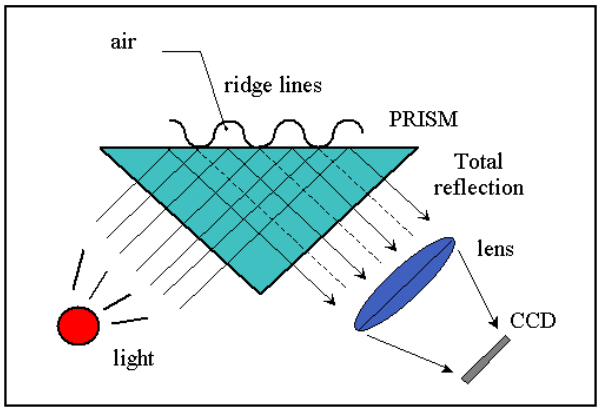


Figure 1: The reflection of light from the of a finger ridge compared to a groove.

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.

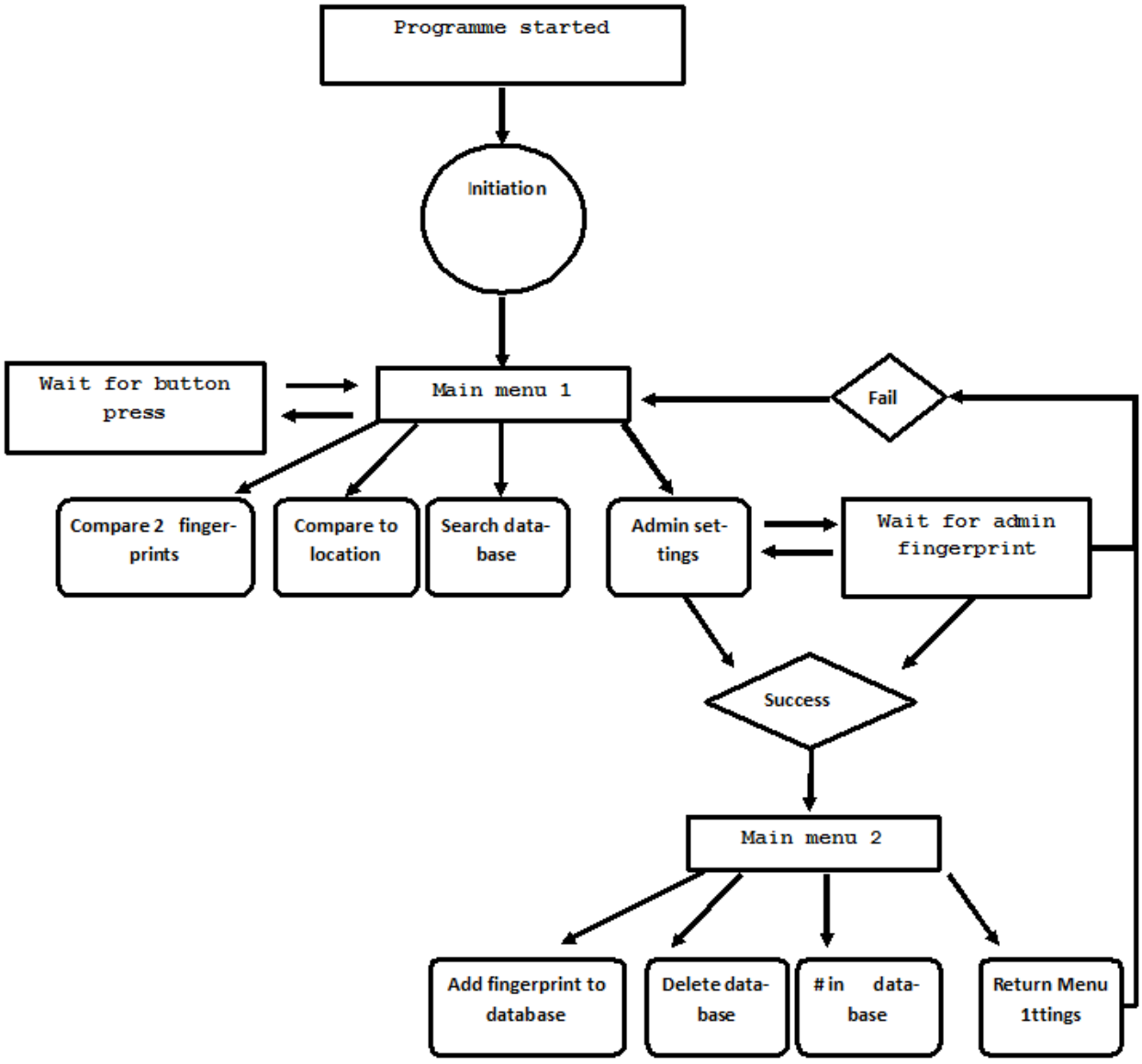


Figure 2: Flow map of the entire operation of the device.

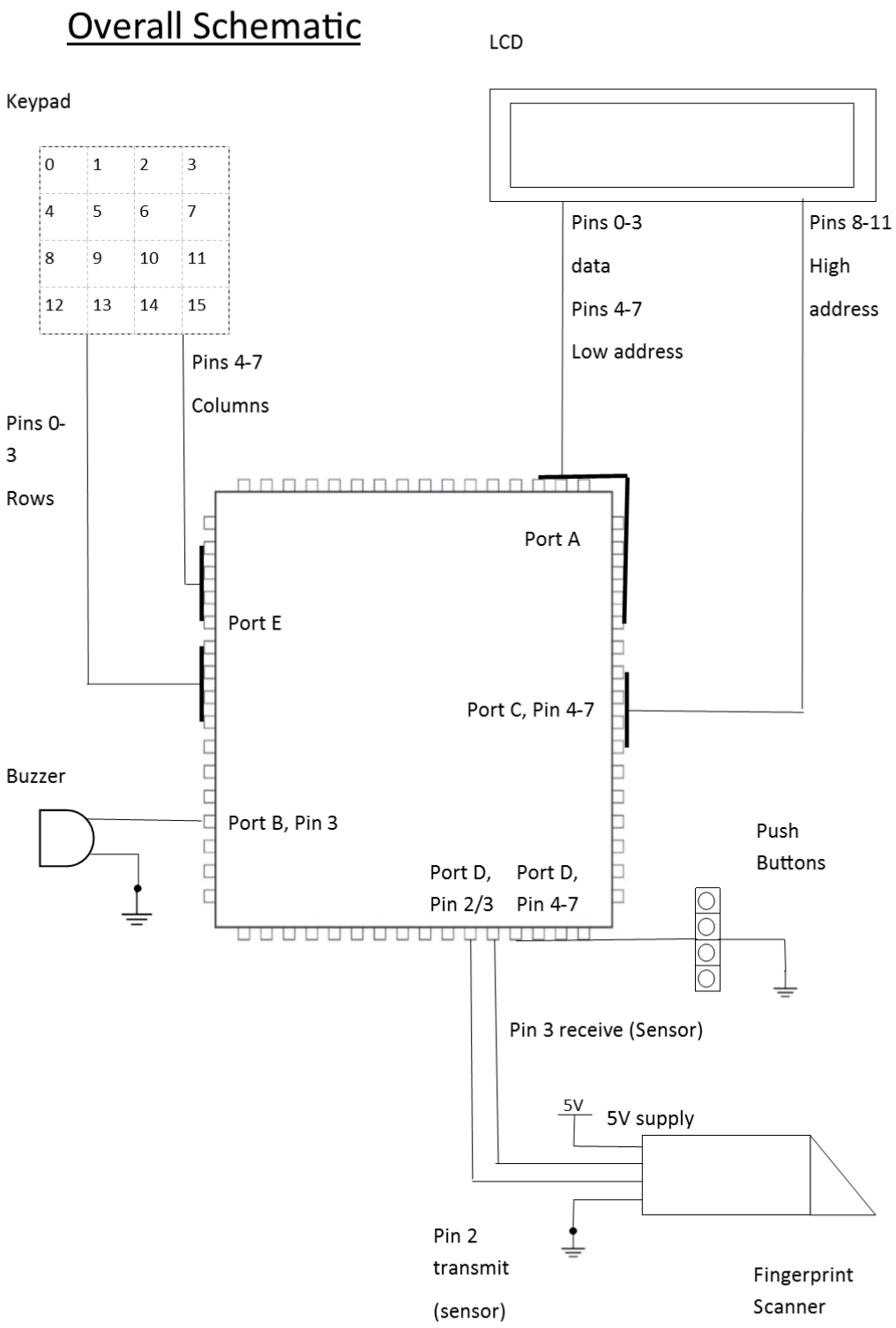


Figure 3: Diagram of overall setup. The full operation of each pin can be viewed in figure 4.

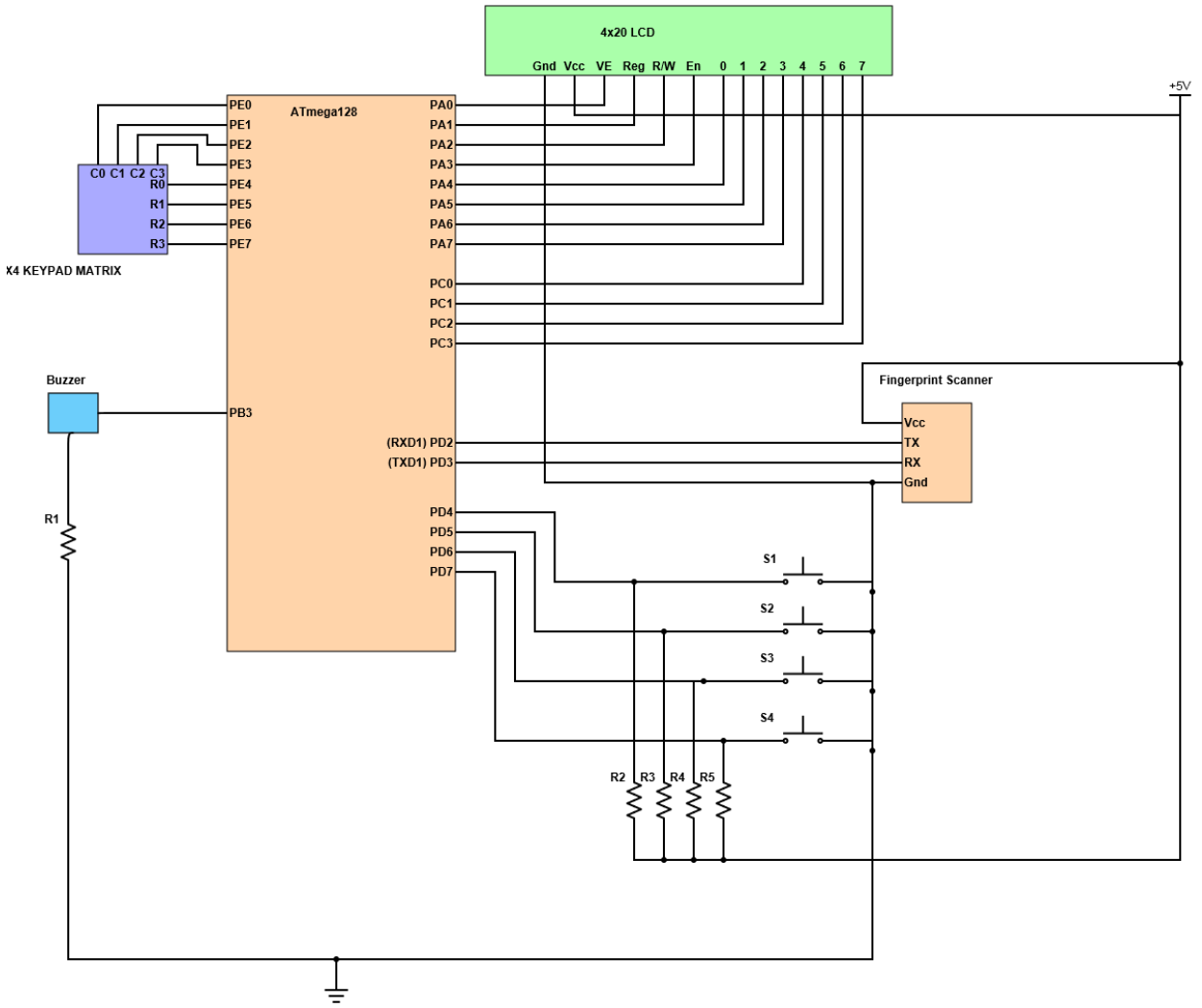


Figure 4: Schematic of the Hardware used.

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.

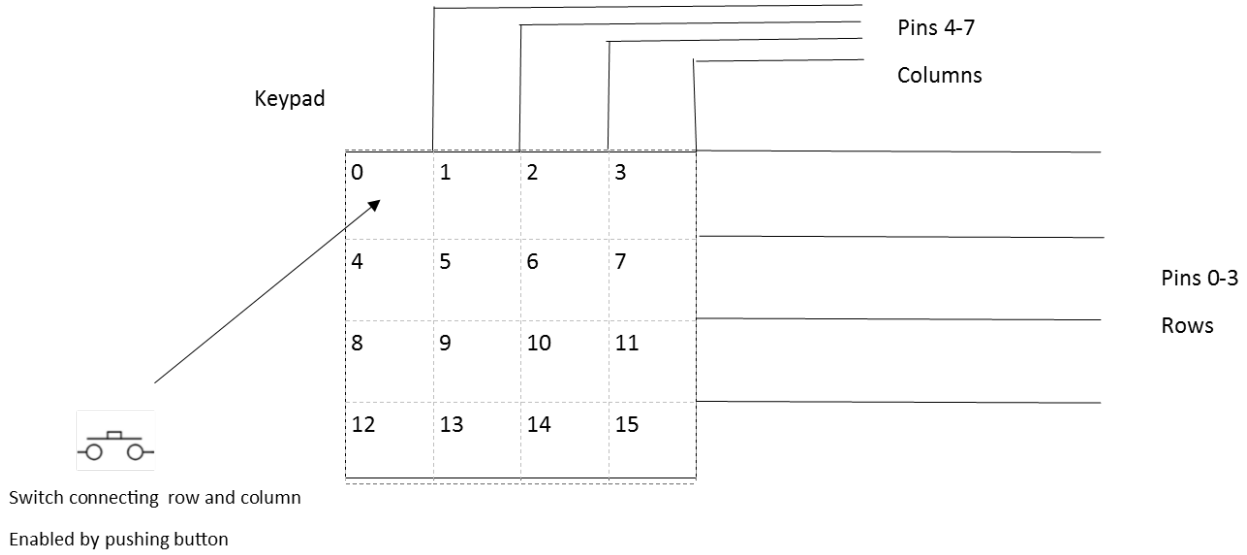


Figure 5: Diagram of the Keypad used.

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.

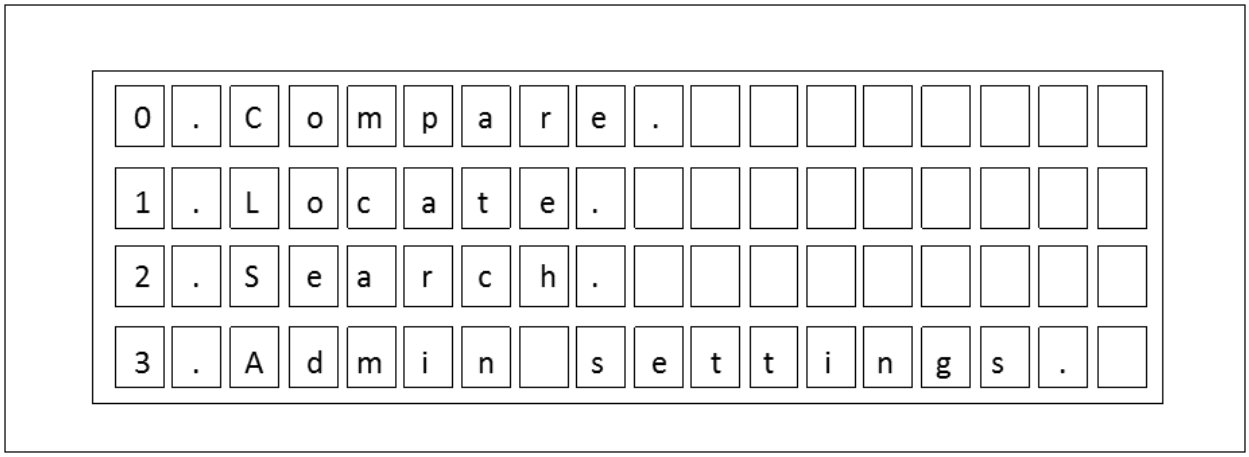


Figure 6: The layout of the options on the LCD.

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.

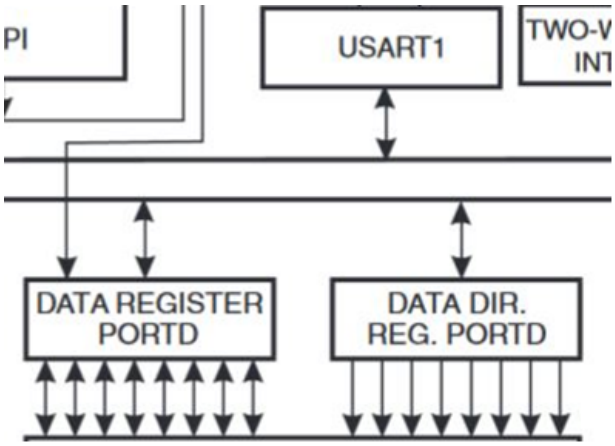


Figure 7: USART1 communication with Port D.

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.

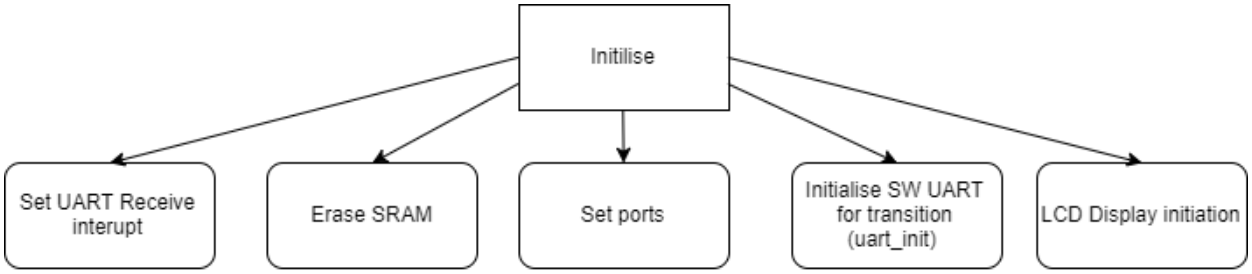


Figure 8: Mid level sketch of the initialisation

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.

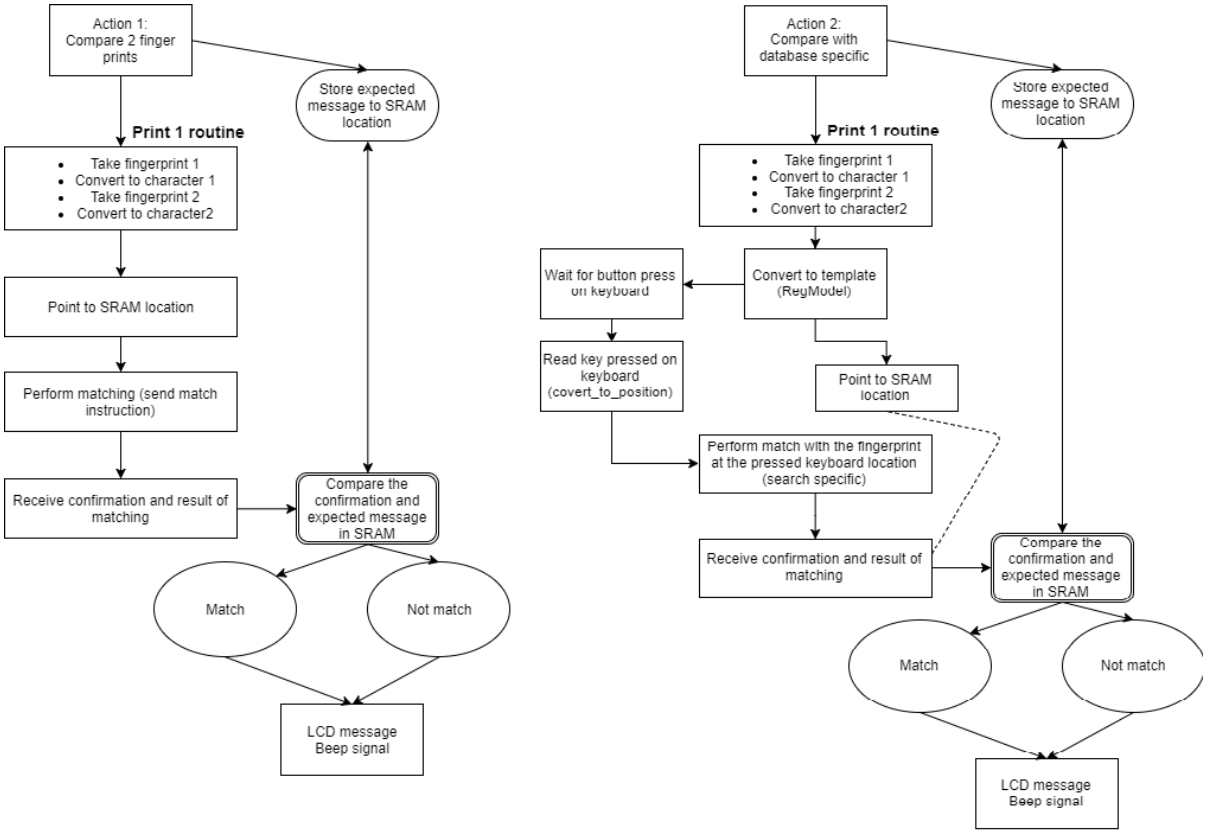


Figure 15: Flow Charts for the Compare and Locate functions.

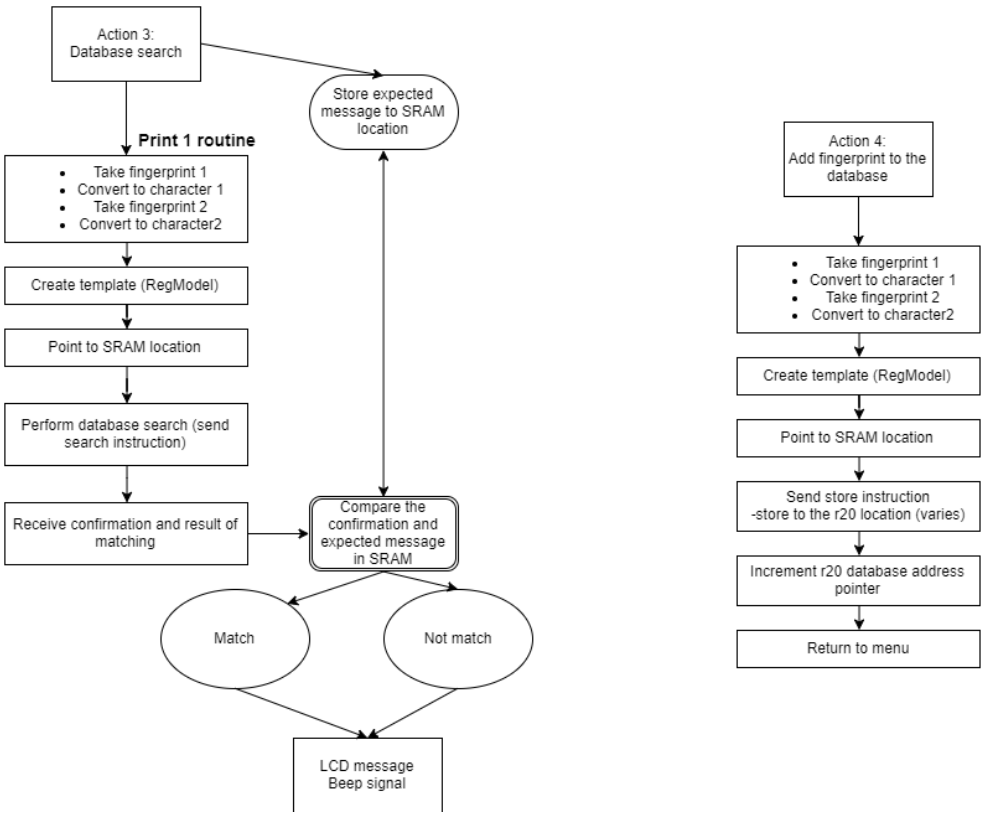


Figure 16: Flow Charts for the Search and Store functions.

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.

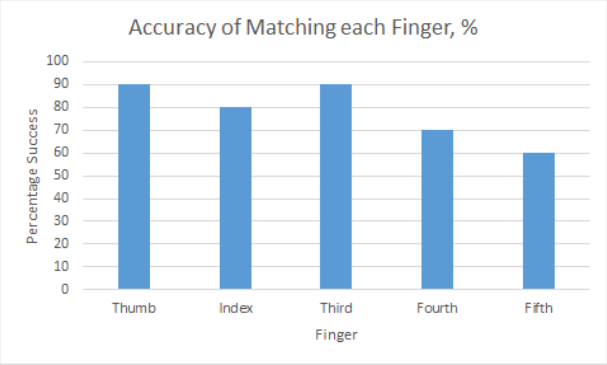


Figure 17: The varying reliability of matching different fingerprints.

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.

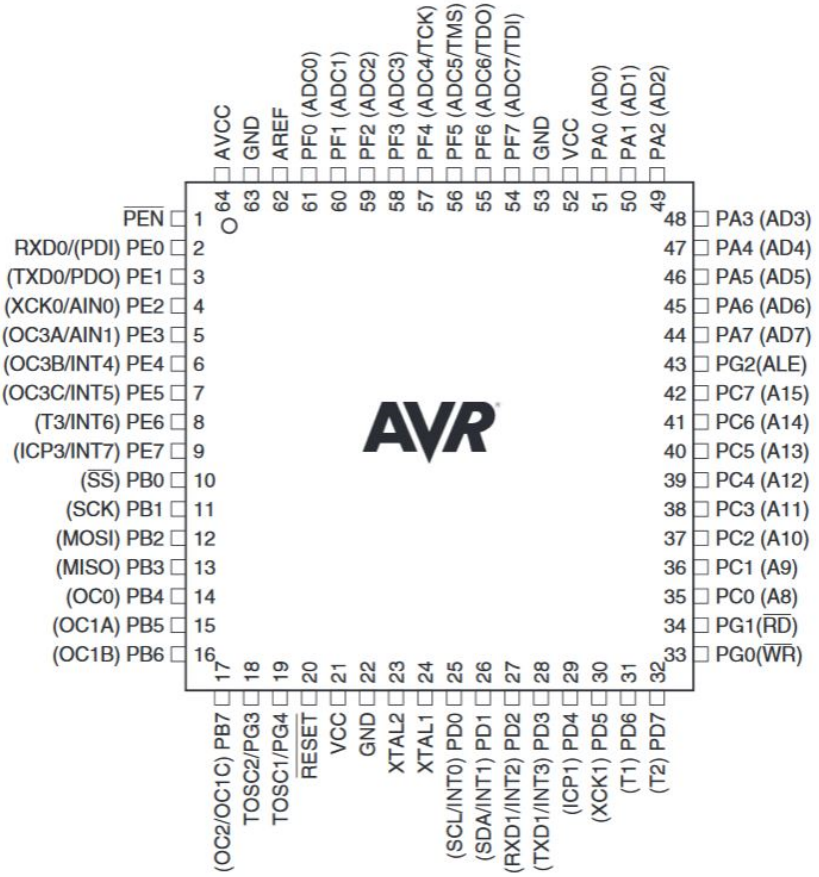
[4] Protostack. Hd44780 character lcd displays - part 1 - protostack, 2017. https://protostack.com.au/2010/03/character-lcd-displays-part-1/.

Specification

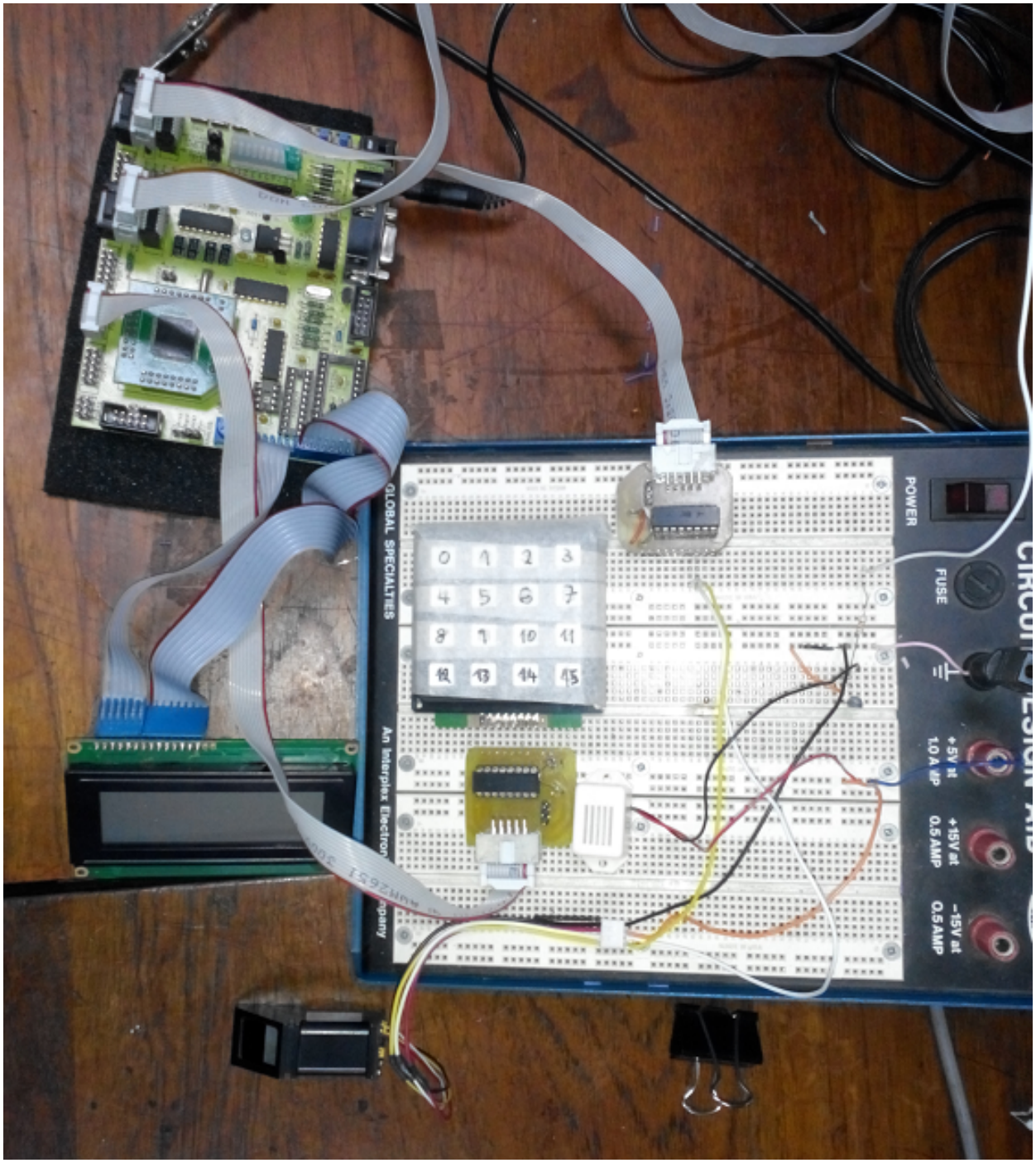
Number of keypad accessible prints	15
Maximum Capacity	162
Function	**Times**
Compare	1s
locate	<2s from Keypad Input
Search Database	<3s
Store	<2s
Template Number 5	1s
Delete	0.5s
Power	DC, 5V
False Read Rate (FRR)	10-20% Thumb, Index and Middle finger
FRR	30-40% Fourth and Fifth fingers
DPI	82.6
Image Capture	CCD
Storage	Flash

Appendices

A Complete Representation of the Atmega128 by Pin



B Photograph of the Entire configuration



C Screen Shots of the various Load Screens

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

```
1  //* AsmFile1.asm
2  //*
3  //*  Created: 14/11/2017 15:41:18
4  //*   Authors: Nelson Talukder and Tomas Jirman
5   //Includes All action routines
6
7
8
9  //1111111111111111111111 Action 1 1111111111111111111111111111111111111111111
10 //Compares 2 presented fingerprints to see if they match, useful for testing
11 //if a fake works
12 action1:
13   rcall DATABASE_TO_SRAM      //Load required comparison Database inot 0x0300
14   rcall Print1          //Routine to Load a fingerprint,1 into each buffer
15                  //Also converts to Character file
16   ldi YH, high(testregister5)   //Point to the SRAM location to save confirmation
17   ldi YL, low(testregister5)
18   rcall send_precisematch     //Compare character files in Buffer1 and Buffer2
19   rcall Bigdel
20   rcall compare_basic2       //Output on LCD and Buzzer the result of Matching
21   ret
22
23
24
25 //333333333333333333333 Action 2 333333333333333333333333333333333333333333
26 //Compares a given Fingerprint to a memoory location selected by the user
27
28 action2:
29   rcall DATABASE_TO_SRAM4     //Load the Search compare database into 0x0300
30   rcall print1          //Collect FIngerprint to be compared
31
32   ldi YH, high(testregister4)   //3C0
33   ldi YL, low(testregister4)
34   rcall send_regmodel       //converts CHaracter files in Buffers-
35                  //-to searchable template
36   rcall BigDEL
37   rcall DEL15ms
38
39 get_input:
40   rcall ready2          //Loadscreen to ask for memory location to check-
41   rcall waitforbuttonpress2   //-Polling, waiting for the result
42   rcall convert_to_position   //Get a hex number of position from 8-bit signal
43   push r18
44   push r21
45   push r22
46   lds r21,  0x0808               //Save ID in SRAM, needed for  instruction-
47   mov r22,r21           //-code for Command Package (CP)
48   lsl r22
49   subi r22,$F2
50   sts 0x0809, r22         //Store Checksum, for CP
51   sts 0x03EB, r21         //Store Hex number
52   rcall binary_to_decimal
53
54   pop r22
55   pop r21
56   pop r18
57
58   ldi YH, high(testregister5)   //3E0
59   ldi YL, low(testregister5)
60   rcall send_search2        //Tailored Search Function to search only 1 location
61   rcall BigDEL
62   rcall compare_basic2
63   rcall BigDEL
64   ret
65
66
67 send_search2:           //Search Function modified by this Routine-
68   push r25           //Breaks up the CP into parts
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 correcttl 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             //Depending on Column, add (0,1 ,2 3)to base-
108   sbrs r21,1             //-values given by rows (e.g. 0, 4, 8 or 12)
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              //Saved to give ID for CP
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 keayboard 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
144   in r21, PINE           //getting value from pinE
```

```
145
146
147                     //inverse the pin output input
148   ldi r16, $0F          // load r16 with 4-7 input, reverse input and ouput
149   out DDRE , r16          // Direction register
150   ldi r16, $F0          // 0-3 output
151   out PORTE, r16
152   rcall DEL49ms
153   in r18, PINE          //getting value from pinE, red PINE again
154
155
156                     //R21 is the result of reading the key
157   add r21, r18          //add 2 results
158
159   cpi r21, $FF
160   breq waitforbuttonpress2    //leaves waiting for button, if value input is not $FF
161   ret
162
163
164
165
166 //444444444444444444444444  Action 3 4444444444444444444444444444444444444444
167 //general database search for match
168 action3:
169   rcall clrdis
170   rcall DATABASE_TO_SRAM4     //Load the Search compare database into 0x0300
171
172
173   rcall print1          //Collect FIngerprint to be compared
174
175   ldi YH, high(testregister4)   //3C0
176   ldi YL, low(testregister4)
177   rcall send_regmodel       //converts CHaracter files in Buffers-
178                     //-to searchable template
179   rcall BigDEL
180   rcall DEL15ms
181
182
183   ldi YH, high(testregister5)   //3E0
184   ldi YL, low(testregister5)
185   rcall send_search       //General Search of Entire Database
186   rcall BigDEL
187   rcall compare_basic2      //Output on LCD and Buzzer the result
188   rcall comparepeople       //Check to See if a name is attached to the ID
189   rcall BigDEL
190   rcall clrdis
191   rcall send_This_is_ID     //LCD output of ID/ memory location
192   rcall Password
193   rcall bigdel
194 finalaction3:
195   ret
196
197
198 /*
199 recordspecific:           //Give Someone a unique address above th 0-15
200   push r25            //Set value for Storage function
201   ldi r25, 62           //Call Storage Function
202   rcall action4
203   pop r25
204 */
205
206 comparepeople:            //Check for address to see if it has a name
207   push r16
208   lds r16, 0x03EB
209   cpi r16, 60
210   breq SayTomas
211   cpi r16, 61
212   breq SayNelson
213   pop r16
214   ret
215
216 SayTomas:             //Load 'Tomas' to LCD
217   rcall clrdis
218   LDI ZH, HIGH(2*Tomas)
219     LDI ZL, LOW(2*Tomas)
```

```
220     LDI r18, 7            //number of characters or bytes
221   rcall Mess1More22
222   rcall password
223   rcall bigdel
224   rjmp finalaction3
225
226 SayNelson:               //Load 'Nelson' to LCD
227   rcall clrdis
228   LDI ZH, HIGH(2*Nelson)
229     LDI ZL, LOW(2*Nelson)
230     LDI r18, 8
231   rcall Mess1More22
232   rcall password
233   rcall bigdel
234   rjmp finalaction3
235
236
237 Password:              //Load SRAM with value if verified as Admin
238   push r16
239   ldi r16, 22           //Checks later to see if 0x0750 = 0 for-
240   sts 0x0750, r16         //admin access
241   pop r16
242   ret
243
244
245 This_is_ID:             //Indicates to the User their ID will be displayed
246   .db " This is the ID: "
247 send_This_is_ID:
248   rcall clrdis
249   LDI ZH, HIGH(2*This_is_ID)
250     LDI ZL, LOW(2*This_is_ID)
251     LDI r18, 16           //number of characters or bytes
252   rcall Mess1More22
253   rcall binary_to_decimal     //COnvert Hex address to Decimal for LCD Screen
254   rcall bigdel
255   rcall clrdis
256   ret
257
258  //^^^^^^^^^^^^^^^^^^^^^^^^^^Binary to decimal subroutine  ^^^^^^^^^^^^^^^^^^^^^^^^^^
259  //Capable of convertng any binary number between 0 and 15 to decimal
260
261  binary_to_decimal:
262   push r16
263   push r23
264   lds r16, 0x03EB         //Load the Hex value from SRAM
265   cpi r16, 10
266   brge greaterthan10
267 seconddigit:             //Converts the remainder to Hex by getting ascii-
268   subi r16, $D0         //-value which is Hex value + $30
269   rcall del49ms
270   sts $C000, r16          //displays remainder as decimal
271   rcall bigdel
272   pop r16
273   pop r23
274   ret
275
276 greaterthan10:            //Dsplays a 1 if Hex value is greater than 10
277   ldi r23, $31
278   sts $C000, r23
279   subi r16, $0A         //get remainder
280   rjmp seconddigit
281
282 //22222222222222222222 Action 4 22222222222222222222222222222222222222222222
283 //Adds new fingerprint to database
284 action4:
285   rcall print1          //Collect Fingerprint to be compared
286   ldi YH, high(testregister2)   //380
287   ldi YL, low(testregister2)
288   rcall send_regmodel       //converts CHaracter files in Buffers-
289                 //-to searchable template//
290   rcall BigDEL
291
292   ldi YH, high(testregister3)   //3A0
293   ldi YL, low(testregister3)
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 acknolegement 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 //@@@@@@@@@@@@@@@@@@@@@@@@@@@@ Action 6 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
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_TempleteNum      //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
```

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   // INTERUPT:
15   .org    $003c
```

```
  rjmp save_data_start352        // UART Receive  Vector
  .org    $0080          // start address above interrupt table

//”””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””
.include ”action4include.asm”
init:                 // system init routine
  .dseg
  delaysaved: .byte 1       //assign memory for clock speed value
  .cseg

  //Locations in SRAM that are used in the programme
  .equ  compare_address = 0x0300  //DATABASE to SRAM, saves to here
  .equ  savedregister = 0x0320
  .equ  testregister  = 0x0340           //all used in interrupt
  .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

  .equ  r = 18            //Setting the no. of delay cycles


// ************************* MEMORY ERRASER ************************************
// It is crucial for the operation of device to start with clear memory where
//data is stored as responce from the finger print scanner.

                  // erases memory from 0x0300 to 0x0900
eraser:
  ldi XH, high(compare_address)
  ldi XL, low(compare_address)
loop:
  push r20
  ldi r20, $01
  st X+, r20
  cpi XH, $09
  brne loop
  pop r20

//########################## PORT SET UPS #########################################


  ldi r16, $FF          //Set-up Port B as Output
  out DDRB , r16
  ldi r16, 00
  out PORTB, r16


  ldi r16, $0F          // Stack Pointer Setup
  out SPH,r16           // Stack Pointer High Byte
  ldi r16, $FF          // Stack Pointer Setup
  out SPL,r16           // Stack Pointer Low Byte

  // ******* RAMPZ Setup Code ****  lower memory page arithmetic
  ldi  r16, $00         // 1 = EPLM acts on upper 64K
  out RAMPZ, r16          // 0 = EPLM acts on lower 64K


  // ******* Enable External SRAM i.e the LCD  *******
  ldi r16, $C0          // Set 11000000, i.e enable read/write
  out MCUCR, r16          // External SRAM Enable Wait State Enabled

  // ******* Comparator Setup Code ****
  ldi r16,$80           // Comparator Disabled, Input Capture Disabled
  out ACSR, r16         // Comparator Settings


//########################## USART SET UP #########################################
  ldi r16, (1<<RXEN1)|(1<<RXCIE1)   //enables interrupt
  sts UCSR1B,r16
// Set frame format: 8data, 2stop bit
```

```
  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

  ldi YH, high(testregister)
  ldi YL, low(testregister)
  sei

//((((((((((((((((((((((((((((((((()))))))))))))))))))))))))))))))))))))))))))))))))))))))
  rjmp main
//((((((((((((((((((((((((((((((((()))))))))))))))))))))))))))))))))))))))))))))))))))))))


//--------------------- DATABASE OF MESSAGES AND COMMANDS TO THE SCANNER-----------
ReadSysPara:
  .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$03,$0f, $00, $13
//========================== Admin Settings ===================================
TempleteNum:
  .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$03,$1d, $00, $21     //no of templates
DeleteChar:
  .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$07,$0C, $00, $00, $00, $0F, $00, $23
//==========================================================================================

genimg:
  .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$03,$01, $00, $05,$08,$00

img2tBuff1:
  .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$04,$02,$1,$00,$08,$0

img2tBuff2:
  .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$04,$02,$2,$00,$09,$0

regmodel:
  .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$03,$05,$00,$09

Store4:
  .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$06,$06,$01, $00

PreciseMatch:
  .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$03,$03, $00, $07,$08,$00

Search:
  .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$08,$04, $01, $00,$00 ,$00, $46, $00, $54, $00

search2:
  .db $EF,$01,$FF,$FF,$FF,$FF,$01,$00,$08,$04,$01




//&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&Compare DATABASE&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

compare_database:

  .db $EF, $01, $FF, $FF, $FF, $FF, $07,$00, $05, $00, $00, $0A  //Used in datbase_to_SRAM

compare_database2:
  .db $EF, $01, $FF, $FF, $FF, $FF, $07,$00, $07, $00, $00, $0A  //Used in datbase_to_SRAM4

//                                                   LCD DATABASE

Match:
  .db ' ','M', 'a', 't' ,'c', 'h'

Different:
  .db ' ', 'D','i','f','f','e','r','e','n','t'

```

```
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 //*****************************************************************************
220 //*****************************************************************************
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235 //*************************  MAIN ***********************************************
236 //*****************************************************************************
237 //*****************************************************************************
238
239
```

```
240 main:               //RUNS ONCE HERE - THis is basically init2 which runs just once
241     rcall Idisp
242   rcall uart_init
243   rcall CLRDIS
244   rcall beepon
245
246   rcall Welcome2          // first welcoming message
247   rcall bigdel
248   rcall bigdel
249   rcall bigdel
250   rcall CLRDIS
251
252 LDI r25, $02            // start dsavinbg 03 -> this points to starting address in scanner
253
254
255 main2:                // this is the main loop -> menu 1 with normal user functions
256   rcall clear750          //clears SRAM location, used to store Admin Verification result
257   rcall Welcome3
258   rcall waitforbuttonpress
259   rcall CLRDIS
260   rjmp main2
261
262
263 //*****************************************************************************
264 //*****************************************************************************
265 //*****************************************************************************
266
267
268 clear750:               // clears SRAM locaiton used for tests
269   push r16
270   ldi r16, 0
271   sts 0x0750, r16
272   pop r16
273   ret
274
275
276 //GGGGGGGGGGGGGGGGGGGGGGG Welcome Messages GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
277 // the logic of the messages is to load SRAM address of the welcome message to Z and
278 //then start displaying one by one characted in LCD routines => Mess1More2
279
280 Welcome2:               //First Screen wth instructions selecting options
281   LDI ZH, HIGH(2*Welcome)
282     LDI ZL, LOW(2*Welcome)
283     LDI r18, 37           //number of characters or bytes
284   rcall Mess1More22
285   ret
286
287 Welcome3:               //First Screen with non Admin Options
288   rcall clrdis
289   LDI ZH, HIGH(2*Welcomemessage1)
290     LDI ZL, LOW(2*Welcomemessage1)
291     LDI r18, 80
292   rcall Mess1More22
293   rcall bigdel
294   ret
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
```

```
message_ID_recognised:          //'Access Granted'
  rcall clrdis
  LDI ZH, HIGH(2*ID_recognised)
    LDI ZL, LOW(2*ID_recognised)
    LDI r18, 19
  rcall Mess1More22
  rcall bigdel
  ret

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  Beginning OF RETURN    XXXXXXXXXXXXXXXXXXXXXXXX
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  Beginning OF RETURN    XXXXXXXXXXXXXXXXXXXXXXXX


 //*******************************COMPARING2***********************************

// in this routine we are going to compare 2 sram databases
// this is crucial routine for decitions about fullfillment of conditions and sucesses
// of finger print detections
compare_basic2:

  push r17
  push r20
  push r21
  push r18
  ldi r18,  0           //counter
  rcall Comp1Out2Z
  rcall Comp1Out2Y
  rjmp Comp1More2


Comp1Out2Z:             //the one we compare to

  LDI ZH, HIGH(compare_address)   //address for comparison
    LDI ZL, LOW(compare_address)
  ret
Comp1Out2Y:
  ldi YL, low(testregister5)      //saved Acknowledgement Package
  ldi YH, high(testregister5)
  ret


Comp1More2:
  ld r20, Z+
  ld r17, Y+

  inc r18
  cp r20, r17           // compare the two X, Z
  brne statefalse2        //go to false message if not equal




  cpi r18, 10           // compare when end the cycle
  brne Comp1More2

Comp1End2:                //True, 1 beep + 'Different'

  rcall matching
  rcall beepon
    rcall megadel
    rcall CLRDIS
  rcall bigdel2
  pop r18
  pop r21
  pop r20
  pop r17
  ret

statefalse2:            //False Branch, 3 beeps + 'Match'
  rcall differently
  rcall megadel
  rcall beepon
  rcall bigdel2
  rcall beepon
  rcall bigdel2
```

```
390   rcall beepon
391   pop r18
392   pop r21
393   pop r20
394   pop r17
395   rjmp main2
396
397
398
399 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  END OF RETURN    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
400 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  END OF RETURN    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
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 thescanner 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(TempleteNum*2)    //Point the Z register to the correct database
416   LDI ZL, LOW(TempleteNum*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
484                     //out portb, r25
485   inc r25
486   rcall bigdel
487   ret
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                 // retunr to send next byte
550
551 //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
552 delaying:              //delay routine
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 Messlout
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   rcall Del49ms
667   ret
668
669 BigDEL:
670   rcall BigDEL2
671   rcall BigDEL2
672   ret
673
674 BigDEL3:
675   rcall Del49ms
676   rcall Del49ms
677   rcall Del49ms
678   rcall Del49ms
679   rcall Del49ms
680   ret
681
682 megadel:
683   rcall BigDEL2
684   rcall BigDEL2
685   rcall BigDEL2
```

```
  rcall BigDEL2
  rcall BigDEL2
  ret

DEL15ms:
  LDI XH, HIGH(19997)
  LDI XL, LOW (19997)
COUNT:
  SBIW XL, 1
  BRNE COUNT
  RET

DEL4P1ms:
  LDI XH, HIGH(5464)
  LDI XL, LOW (5464)
COUNT1:
  SBIW XL, 1
  BRNE COUNT1
  RET

DEL100mus:
  LDI XH, HIGH(131)
  LDI XL, LOW (131)
COUNT2:
  SBIW XL, 1
  BRNE COUNT2
  RET

DEL49ms:
  LDI XH, HIGH(65535)
  LDI XL, LOW (65535)
COUNT3:
  SBIW XL, 1
  BRNE COUNT3
  RET




//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%    Interrupt   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

save_data_start352:
  push r17
  lds r17, UDR1
  st Y+, r17            // store byte received in USART1 to memory location
  pop r17
  reti


//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  Calling Actions  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


// ~~~~~~~~~~~~~~~~~~~~~~~~ WAIT FOR  BUTTION PRESS AND MOVE SOMWEHRE ~~~~~~~~~~

waitforbuttonpress:
  rcall waitforbuttonpresscycle
  rcall FirstScreen
  ret
waitforbuttonpresscycle:      //Subroutine to wait for Pin input
  in r21, PIND
  cpi r21, $FF
  breq waitforbuttonpresscycle
  rcall clrdis
  ret



FirstScreen:
                  //here have list of conditions
  cpi r21,$7F           // pin 1
  breq action1call        // ACTION 1

  cpi r21, $BF          // PIN2
  breq action2call        // ACTION 2

```

```
  cpi r21, $DF          // PIN3
  breq action3call        //Action 3

  cpi r21, $EF          // PIN4
  breq SecondScreen       //Action 4

  ret

SecondScreen:
  rcall verification        //Required: Verify ID for Admin Access
correct:
  rcall Welcome5
  rcall waitforbuttonpresscycle
                  //here have list of conditions
  cpi r21,$7F           // pin 1
  breq action4call        // ACTION 1

  cpi r21, $BF          // PIN2
  breq action5call        // ACTION 2

  cpi r21, $DF          // PIN3
  breq action6call        //Action 3

  cpi r21, $EF          // PIN4
  breq action7call        //Action 4
  rjmp main2


actions:

action1call:
  rcall action1
  rjmp main2
action2call:
  rcall action2
  rjmp main2
action3call:
  rcall action3
  rjmp main2
action4call:
  rcall action4         //included
  rjmp main2
action5call:
  rcall action5
  rjmp main2
action6call:
  rcall action6
  rjmp main2
action7call:
  rjmp main2

//%%%%%%%%%%%%%%%%%%%%%%%%      Verification       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Verification:           //Checks 0x0750 to see if Password subroutine-
  rcall action3         //-was reached
  lds r16, 0x0750
  cpi r16, 22
  breq continue
  rjmp main2            //Jumps to Screen1 if not
  ret

continue:             //Called if 0x0750 has been changed by Password
  rcall message_ID_recognised
  rcall clear750
  rcall bigdel
  rjmp correct




//^^^^^^^^^^^^^^^^^^^^^^^^^^DATABASE TO SRAM  MATCH   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

DATABASE_TO_SRAM:           //saves to 0x0300
  push r20

makingZalocation:
```

```
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 makingZalocation4:
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 //??????????????????????????? Alarm    ?????????????????????????????????????????
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
```