TERENCE BROADBENT

STAFFORDSHIRE UNIVERSITY

SOFTWARE DESIGN DOCUMENT

FIRE ALARM SYSTEM

Use of Micro-controller MC68HC11F1 with ported Visual Studio C computer programming language to construct a sophisticated fire alarm system.

PAPER CONTENTS

Use of Micro-controller MC68HC11F1 with ported Visual Studio C computer programming language to construct a sophisticated fire alarm system.

LIST OF ILLUSTRATIONS

Use of Micro-controller MC68HC11F1 with ported Visual Studio C computer programming language to construct a sophisticated fire alarm system.

LIST OF TABLES

Use of Micro-controller MC68HC11F1 with ported Visual Studio C computer programming language to construct a sophisticated fire alarm system.

PART A

INTRODUCTION

This student paper sets out to document the requirements for the fully structured design of a Visual studio ANSI C Program using a modular software design approach and full functionality testing.

It also seeks to document the methodology of porting the Visual C program to run on a MC68HC11F1 micro-controller and the additional required functionality testing.

Purpose

This software design document designates the system design and architecture for the public release of the 'Fire Alarm System' (release version 1.0); it fully exploits the (IEEE, 2003) Software Design Document (SDD) template in doing so.

Scope

The embedded system is to monitor 9 separate alarm circuits via the MC68HC11F1 parallel ports that can be split into three fire zones (3 trips per zone). Each zone must be capable of being enabled or disabled when the alarm is set via a menu driven interface on the system terminal. If the alarm is activated then the system should activate a single bit of port output and display the alarm status on the system terminal, until the password is entered. The program should log a limited number of set/alarm events (100 max) in memory and print these to the screen when required by the user (David Hodgkiss and James Mc Carren, 2015).

Overview

Software design is a process by which the software requirements are translated into a representation of software components, interfaces, and data necessary for the implementation phase. This SDD shows how the software system will be structured to satisfy the requirements. It is the primary reference for code development and therefore, it must contain all the information required by a programmer to write the code.

This SDD is performed in two stages. The first 'Part A' details the structured design of the Visual studio ANSI C program and the second 'PART B' details the porting of Visual C program to run on the MC68HC11F1 micro-controller - both sections include functionality testing.

Orientation Documentation

1	COCS50592 Advanced Programming Languages for Computer Systems.
2	C Cross Compiler User’s Guide for Motorola MC68HC11 version 4.1.
3	IEEE Recommended Practice for Software Design Descriptions.

Table 1- Orientation Documentation

Acronyms

IEEE	Institute of Electrical and Electronics Engineers.
SDD	Software Design Document

Table 2 - Acronyms

Definitions

Byte	A unit of digital information usually consisting of 8 bits.
Bit	A basic unit of digital information represented as a 1 or 0.
Serial port	A physical interface through which digital information transfers in or out one bit at a time.
Parallel port	A physical interface through which digital information transfers in or out 16 bits at a time.
I/O	Input / Output

Table 3 - Definitions

FIRE ALARM SYSTEM

The 'Fire Alarm System' (release version 1.0) has been purposefully designed to be as simple and intuitive as reasonable possible for progression by a user. For example, enabling or disabling any one of the three alarm zones is as simple as pressing keys 1, 2 or 3 on the 8 bit serial port keyboard as visually suggested by the program menu options detailed at the base of most display screens [A2].

Once a menu key has been pressed a new updated representation of the display is instantly sent to the screen as confirmation of the actions taken by the user. Further, the display screen is also automatically updated via the 16 bit parallel port. This port controls the status of the nine external sensors for the fire alarm system [A1].

For example as shown below in figure 1 - Zone 3 has been toggled (turned on) by the user via the keyboard and external sensors 8 and 9 have been remotely triggered, which in turn has flagged a fire notice within the associated zone to the user.

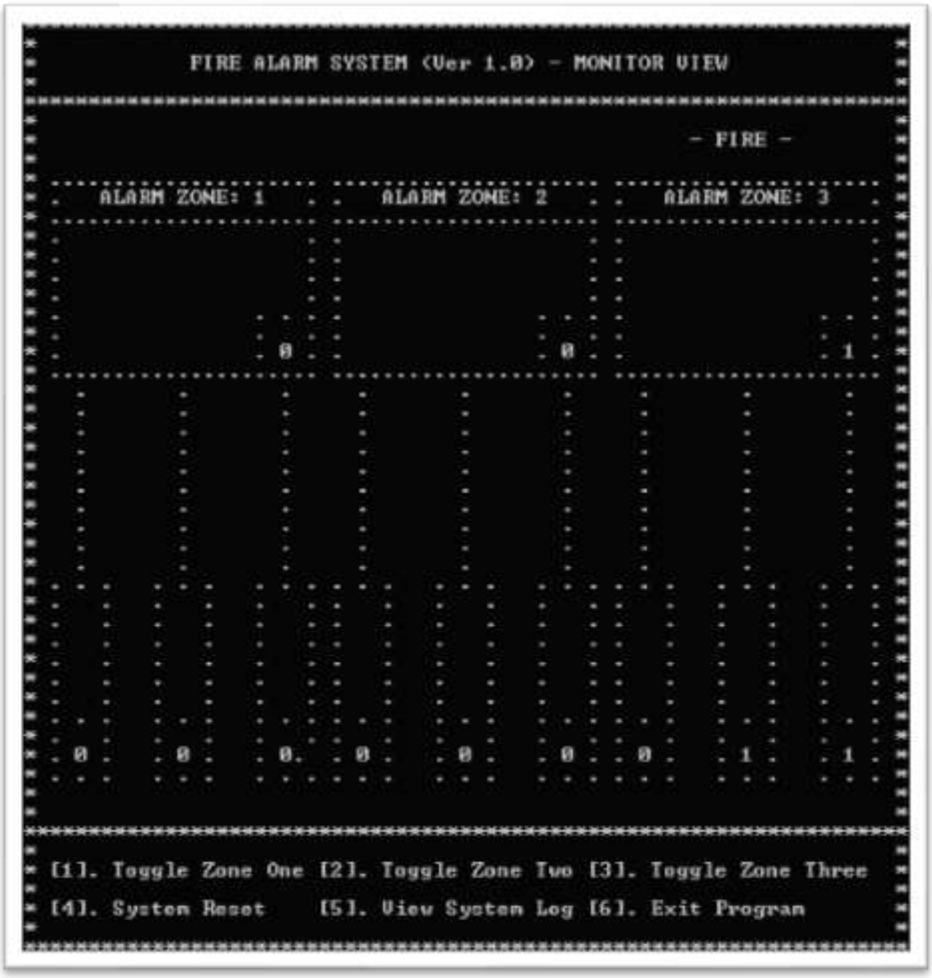


Figure 1 – Fire alarm system visual display (pre compiler)

Now, even if Zone 3 is accidently toggled again (turned off) by the user, following a validated zone sensor trip the screen will continually display that a fire has occurred within the associated zone until the user undertakes a complete system reset by pressing key 4.

Note: In order to successfully reset the system a unique case sensitive five character password is required to be correctly entered by the user [A3].

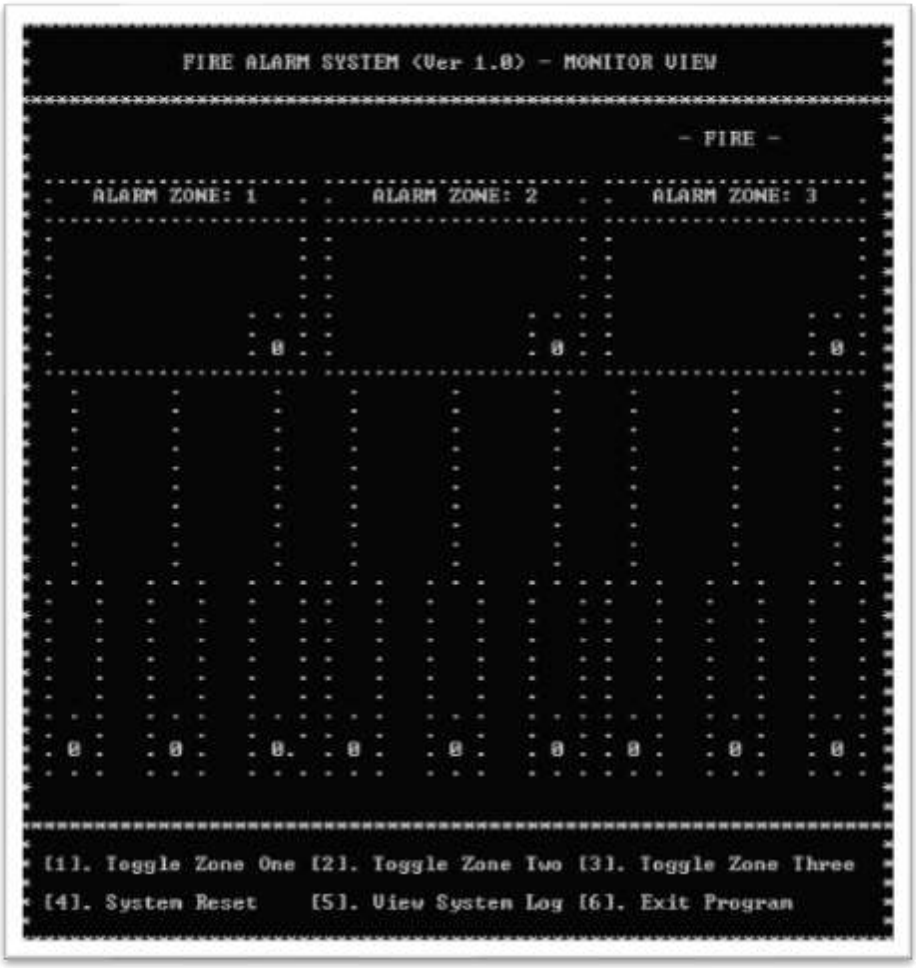


Figure 2 – Fire flag constantly displayed in zone 3 (pre compiler)

Finally, the zone sensors will only update the screen display via the 16 bit parallel port if the zone is actually switched on – for example in figure 1 sensor 1 may have been remotely triggered first but would not have updated the display as zone 1 was not switched on [A1].

System Architecture

To efficiently and effectively code the ‘Fire Alarm system’ using Visual studio ANSI C it was necessary to develop a modular program structure in order to manage and achieve complete functionality of the system [B1].

This high level overview forms the backbone structure for the responsibilities of the system and how and why they were partitioned and assigned to subsystems.

The following identifies each high level subsystem and the roles or responsibilities assigned to it. It describes how these subsystems collaborate with each other in order to achieve the desired overall functionality.

Design Rationale

[A1]. The system is to monitor 9 separate alarm circuits via the 68HC11 parallel ports that can be split into three fire zones (3 trips per zone).

[A2]. Each zone must be capable of being enabled or disabled when the alarm is set via a menu driven interface on the system terminal.

[A3]. If the alarm is activated then the system should activate a single bit of a port output and display the alarm status on the system terminal, until the password is entered.

[A4]. The program should log a limited number of set/alarm events (100 Max) in memory and print these to the screen when required by the user.

As a guide typical stages in development:-

[B1]. Write a C program to input and bit display the data from Port A.

[B2]. Implement a real time clock.

[B3]. Implement a routine to key scan the serial port i.e. ‘mygetchar’ without the need for carriage return.

[B4]. Extend the real rime clock to include a simple data logger of the zones and display the log via a screen.

[B5]. Combine all the above elements to form a working commercial system.

[B6]. Extend the program to include an additional 9 loopback circuits to enable the continuity of the trips to be tested.

[B7]. The program is required to be compiled and linked for use from the system RAM area.

The above elements were extracted from the initial briefing assignment.

(David Hodgkiss and James Mc Carren, 2015).

Architectural Design

Using this modular approach the 'Fire Alarm System' can be decomposed into the following functional program constructs based on the stated design rationale:

Figure 3	Initialise the program [B5].
Figure 4	Main program [B5].
Figure 5	Looping menu system [B5].
Figure 6	Switch menu system [B5].
Figure 7	Parallel port [B6].
Figure 8	Resetting the system [B5].
Figure 9	Start the system logging functionality [A4], [B4].
Figure 10	Display screen [B1], [B6].
Figure 12	Toggling the zones [A2].
Figure 14	Reset zones [B5].
Figure 16	Reset sensors [B5].
Figure 18	Reset fire status [A3].
Figure 20	Display the log book [A4], [B4].
Figure 21	The micro-controller interrupt clock [B2].
Figure 23	Write log data [A4], [B4].
Figure 25	Compare password strings [A3].

Table 4 – Architectural decomposition

Note: The above is not an exhaustive design list of program functions. There are a functions coded so small that a modular design approach is clearly not required.

These include: Serial port scanning and obtaining the keyboard key pressed [B3].

However, these functions have been detailed in 'Looping Menu System' and 'Parallel Port' design.

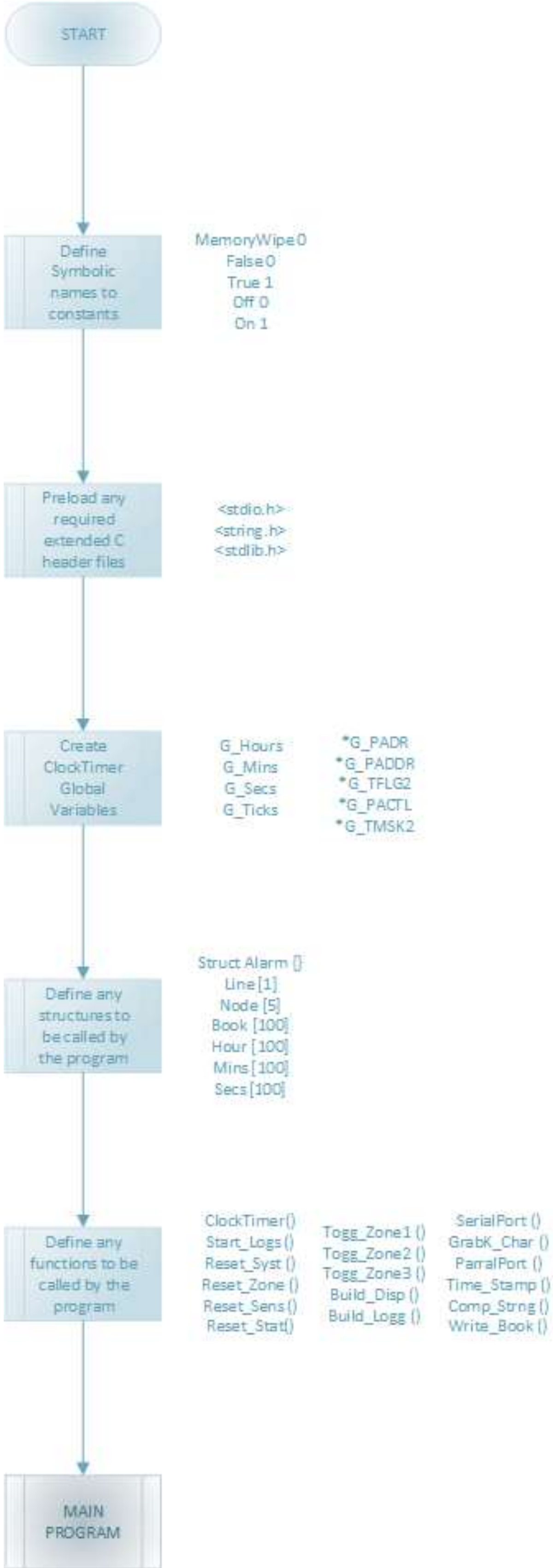


Figure 3 – Initialise the program

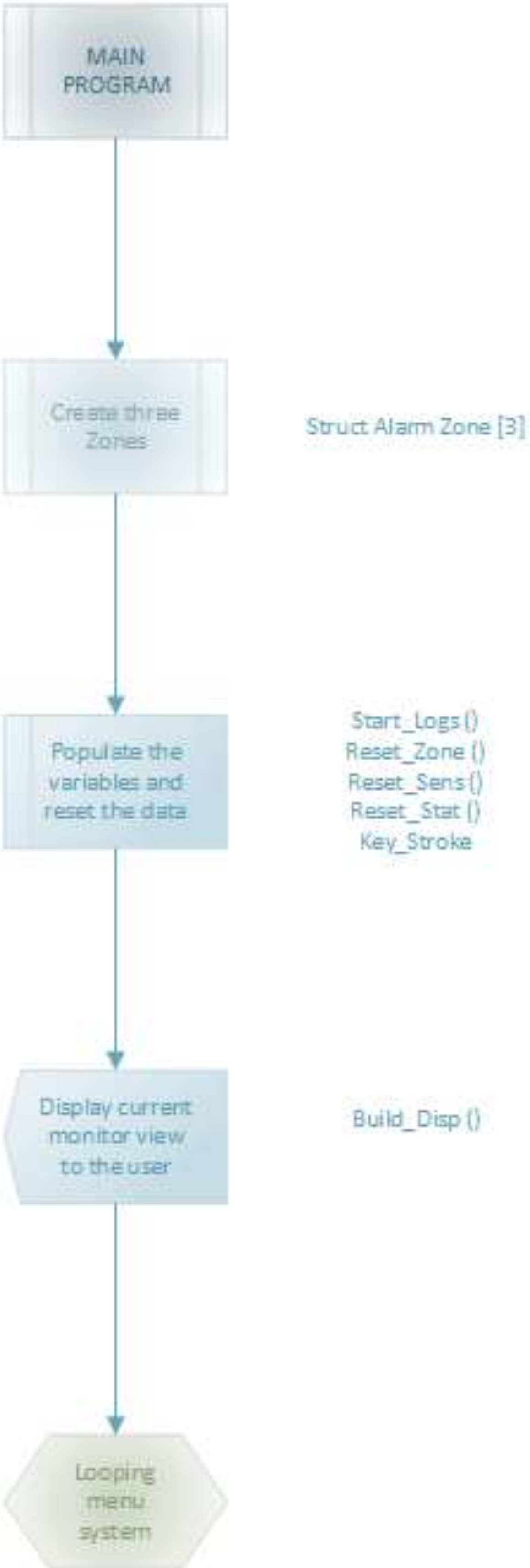


Figure 4 – Main program

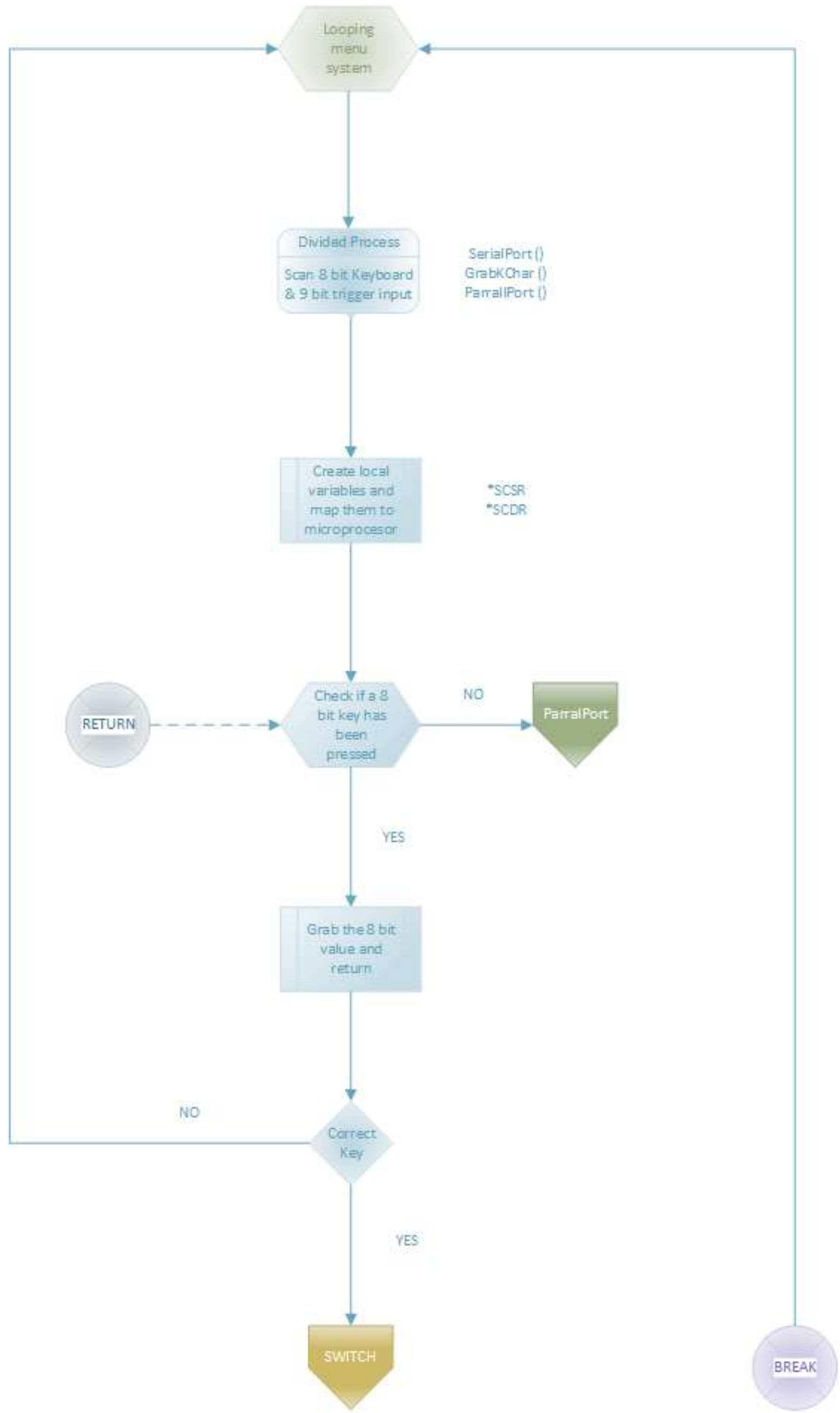


Figure 5 – Looping menu system

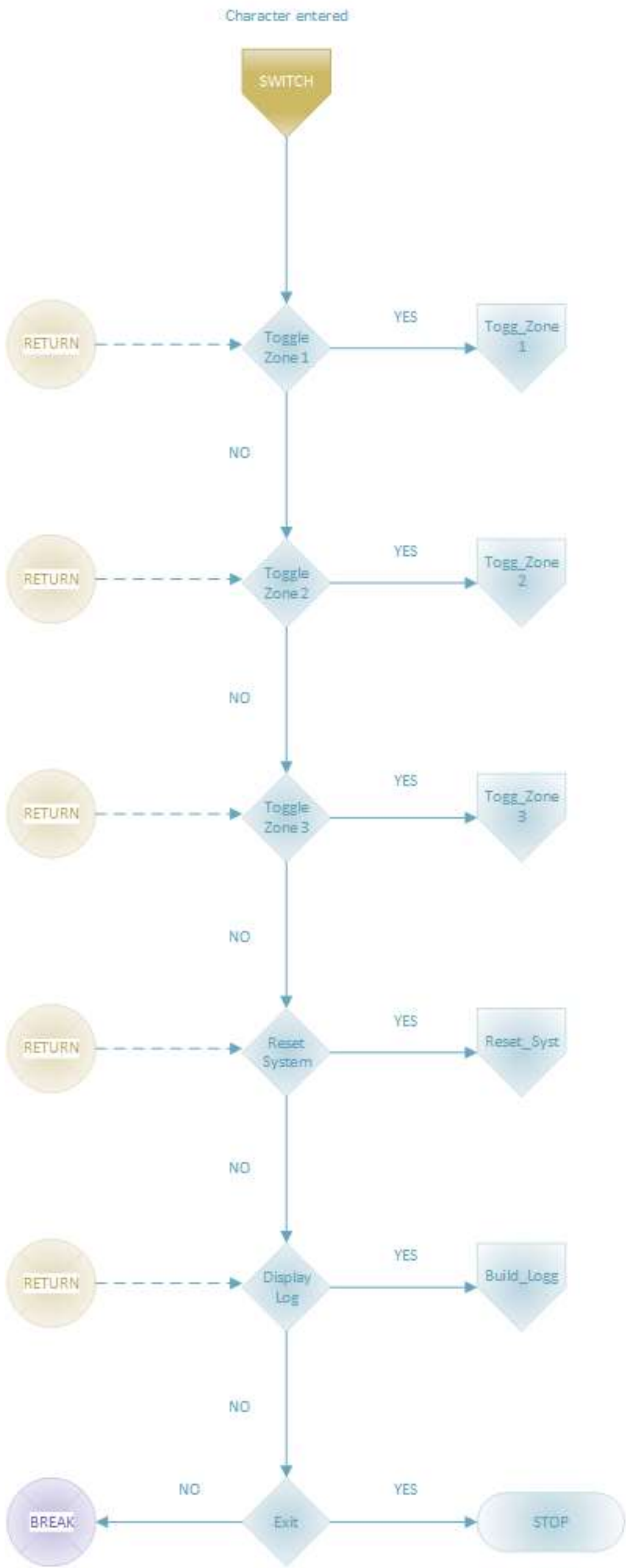


Figure 6 – Switch menu system

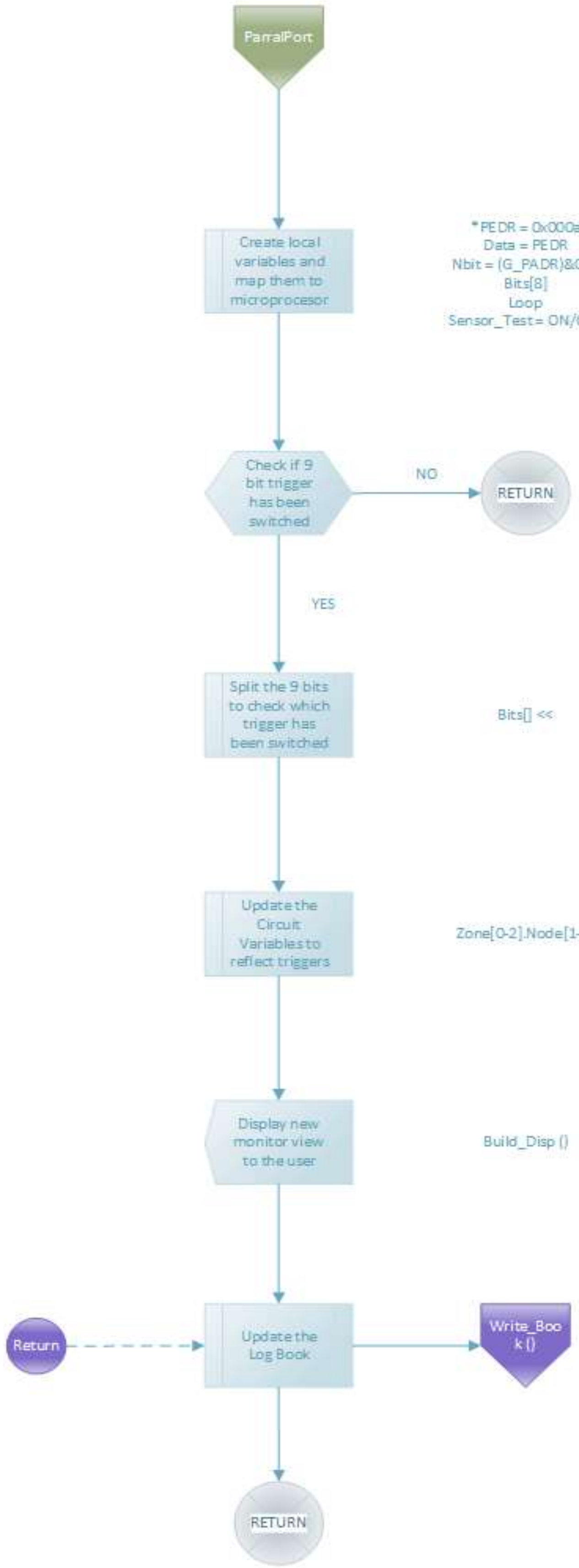


Figure 7 – Parallel port

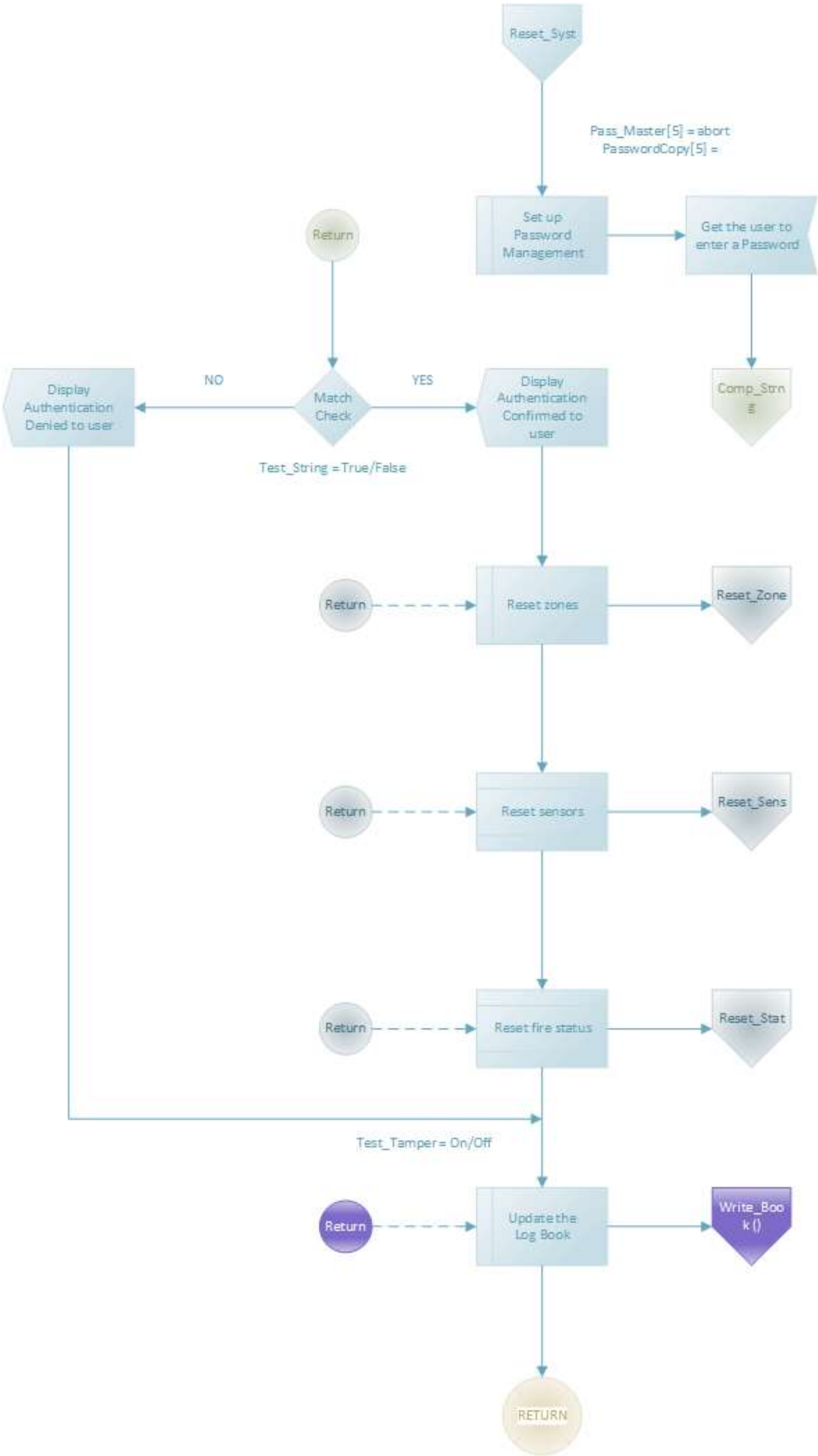


Figure 8 – Resetting the system

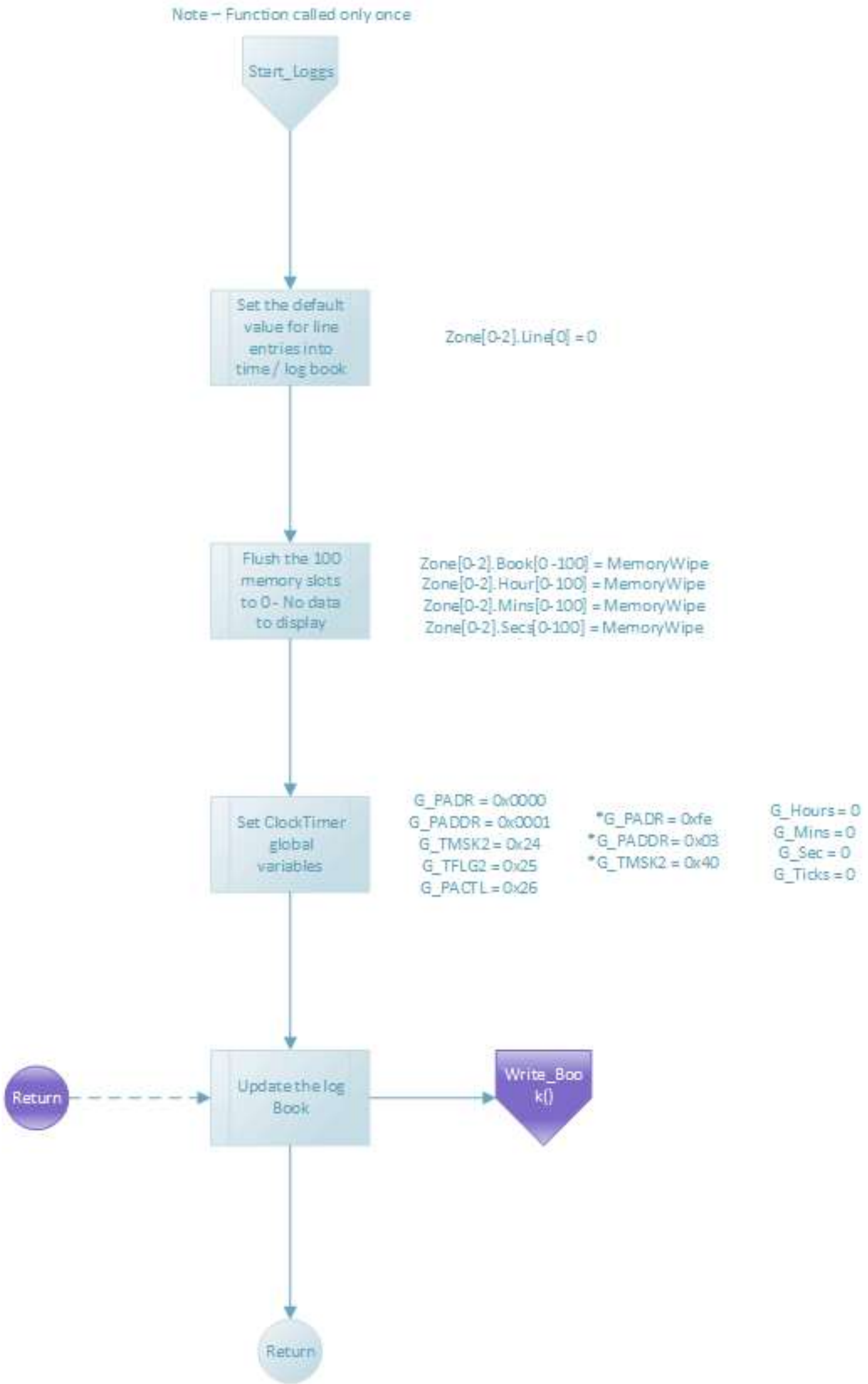


Figure 9 – Start the system logging functionality

PROGRAM TESTING I

The following modular function constructs were comprehensively tested within Visual Studio to confirm whether or not the corresponding program defined variables were configured correctly in order to display an accurate visual representations of the 'Fire Alarm System' data settings at any one time.

The coding tests were conducted and undertaken within their own defined classes and later incorporated into the working program. Consequently extra variables and error traps have been created and populated within the classes to simulate user inputs but discarded for the working program.

Finally, the program coding was also tested for robustness of usage in both data manipulation and simplification.

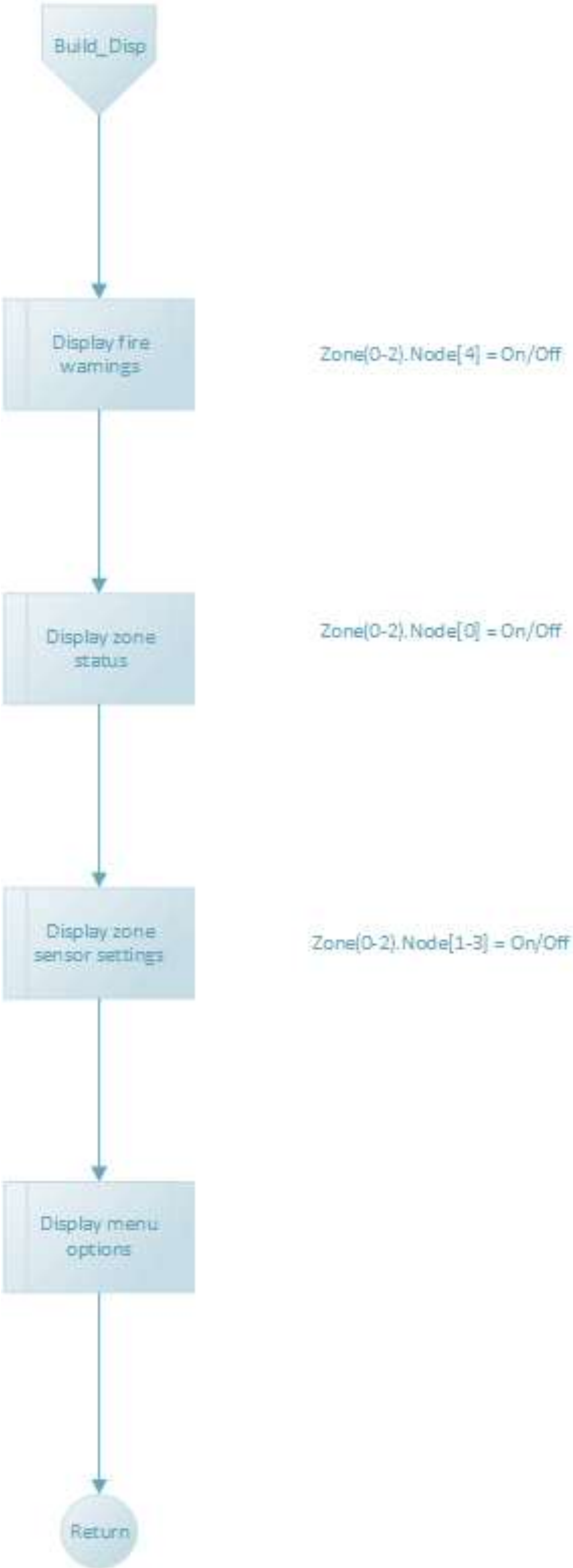


Figure 10 – Display screen

Build_Disp() Function Test

PRE COMPILER TEST ONE Build_Disp() function Zone[0-2].Node[0-4] Checked By: Terence Broadbent Date Checked: 07/01/2015

Table 5 – Pre compiler test one

This program function builds a visual representation of the three zone settings to the users display screen in order that the user can visually see any changes made to the 'Fire Alarm System' instantly.

Zone variables were pre-loaded with various default settings while testing and the function run several times to visually observe the output results to the screen.

Memory overflow was also tested by writing outside the defined Zone[0-2] stipulated array structure and observations made to the screen results.

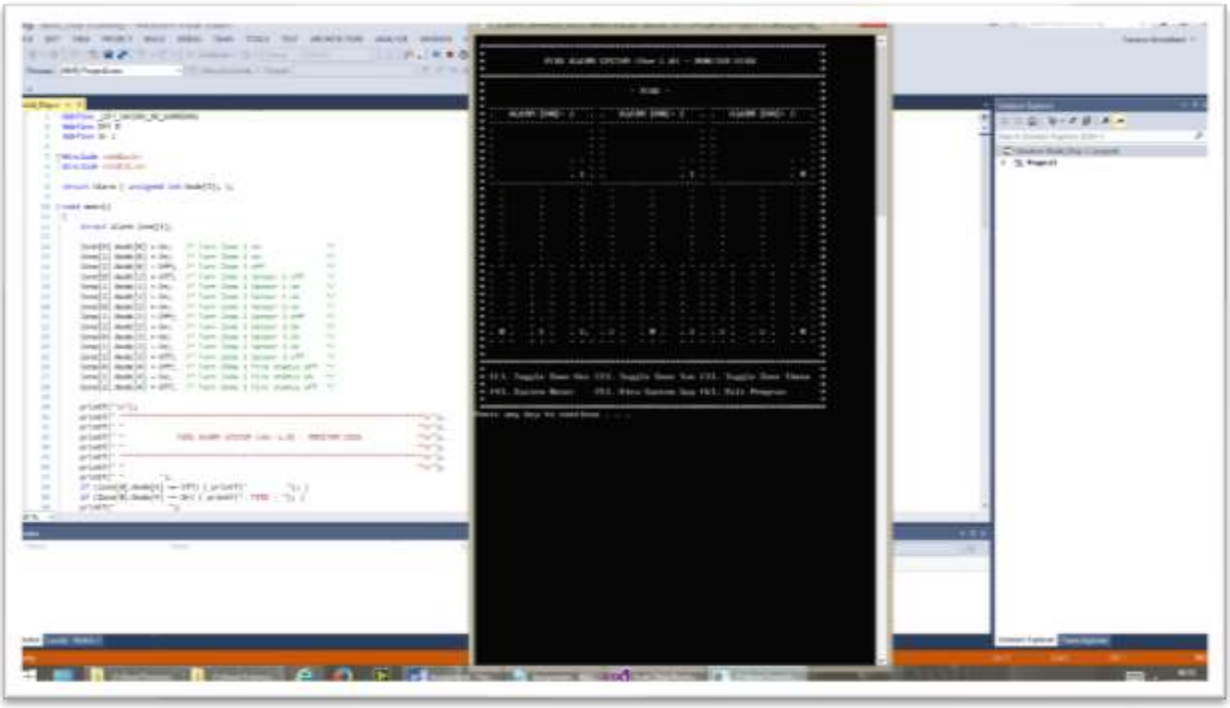


Figure 11 – Build_Disp test (pre compiler)

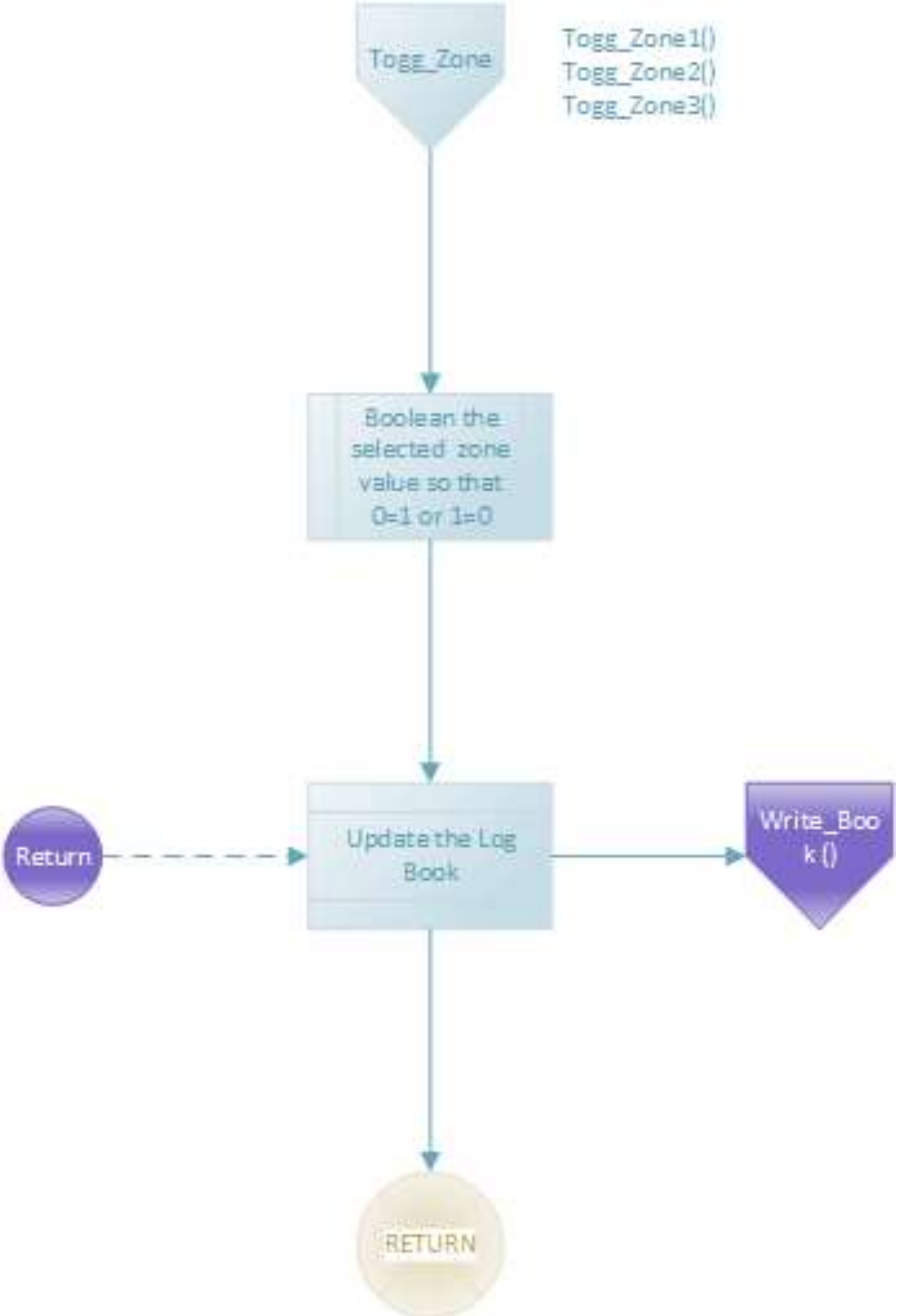


Figure 12 – Toggling the zones

Togg_Zone() Function Test

PRE COMPILER TEST TWO
Togg_Zone() function
Zone[0-2].Node[0] = !Zone[0-2].Node[0]
Checked: Terence Broadbent
Date Checked: 14/01/2015

Table 6 – Pre compiler test two

This program function toggles the three zones on or off based on simple Boolean logic [A2].

Zone variables were pre-loaded with various default settings while testing and the function run several times to visually observe the output results to the screen.

It was found using the above Boolean code was more efficient than running a short loop for the following code:

if Zone[0-2].Node[0] = Off { Zone[0-2].Node[0] = On };

if Zone[0-2].Node[0] = On { Zone[0-2.]Node[0] = Off };

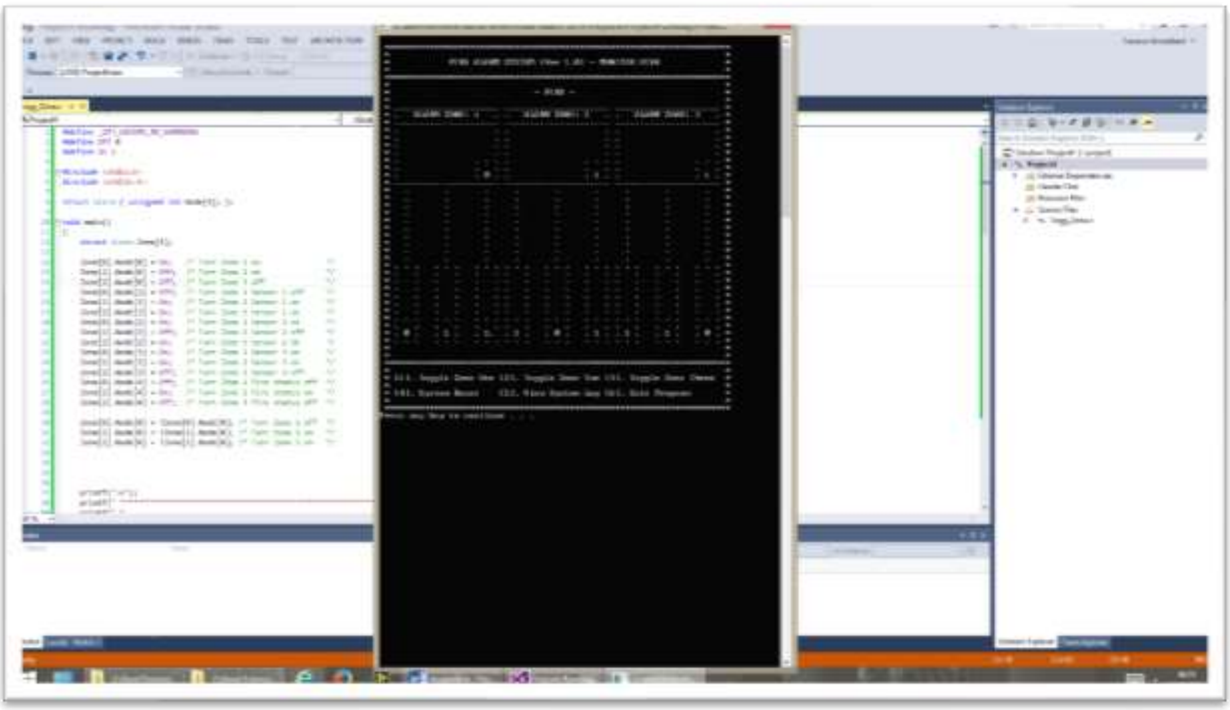


Figure 13 – Togg_Zone test (pre compiler)

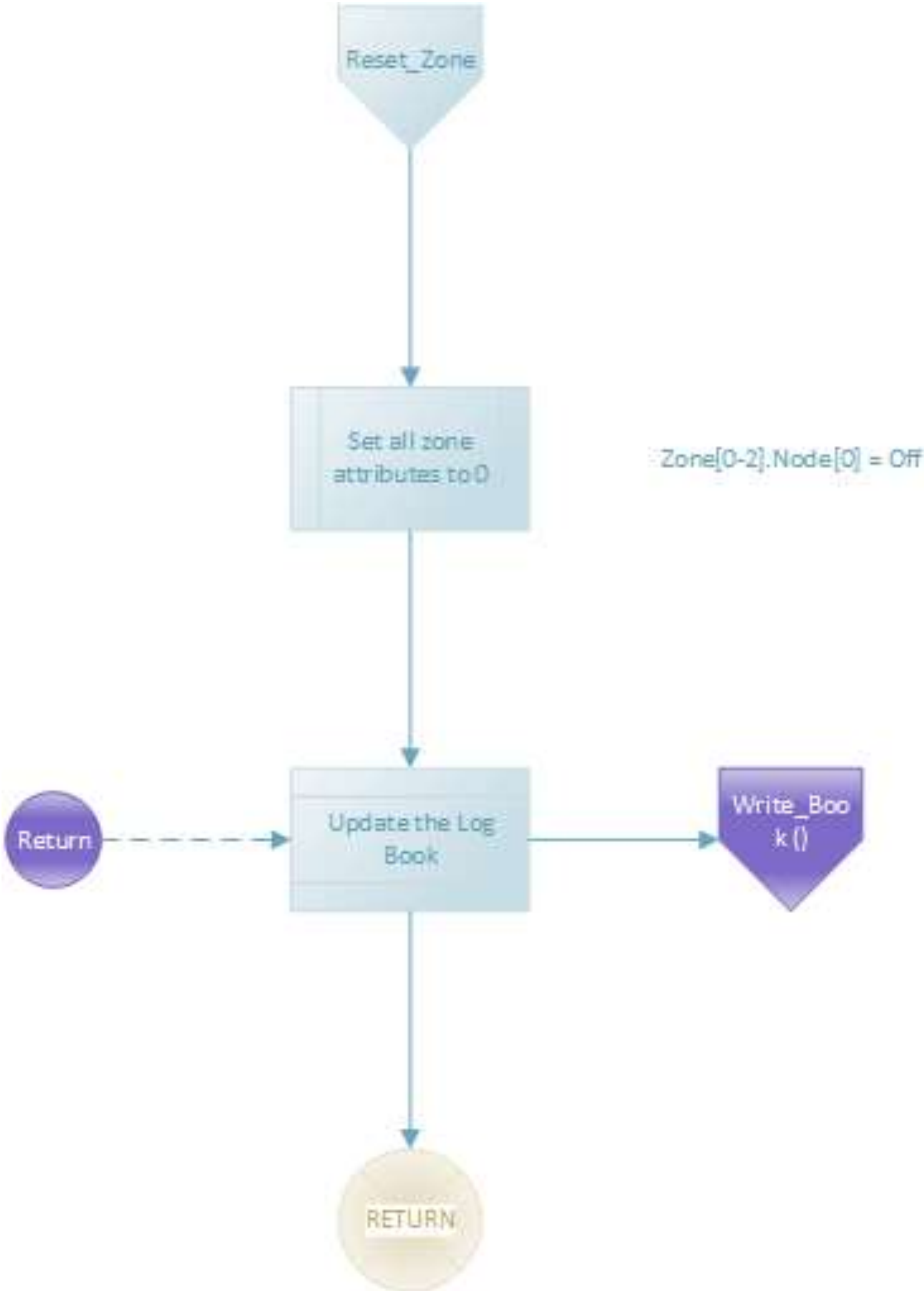


Figure 14 – Reset zone

Reset_Zone() Function Test

PRE COMPILER TEST THREE Reset_Zone() function Zone[0-2].Node[0] = Off Checked By: Terence Broadbent Date Checked: 21/01/2015

Table 7 – Pre compiler test three

This function system resets the zones on/off value to off.

Zone variables were pre-loaded with various default settings, then reset before testing and the function run several times to visually observe the output results to the screen.

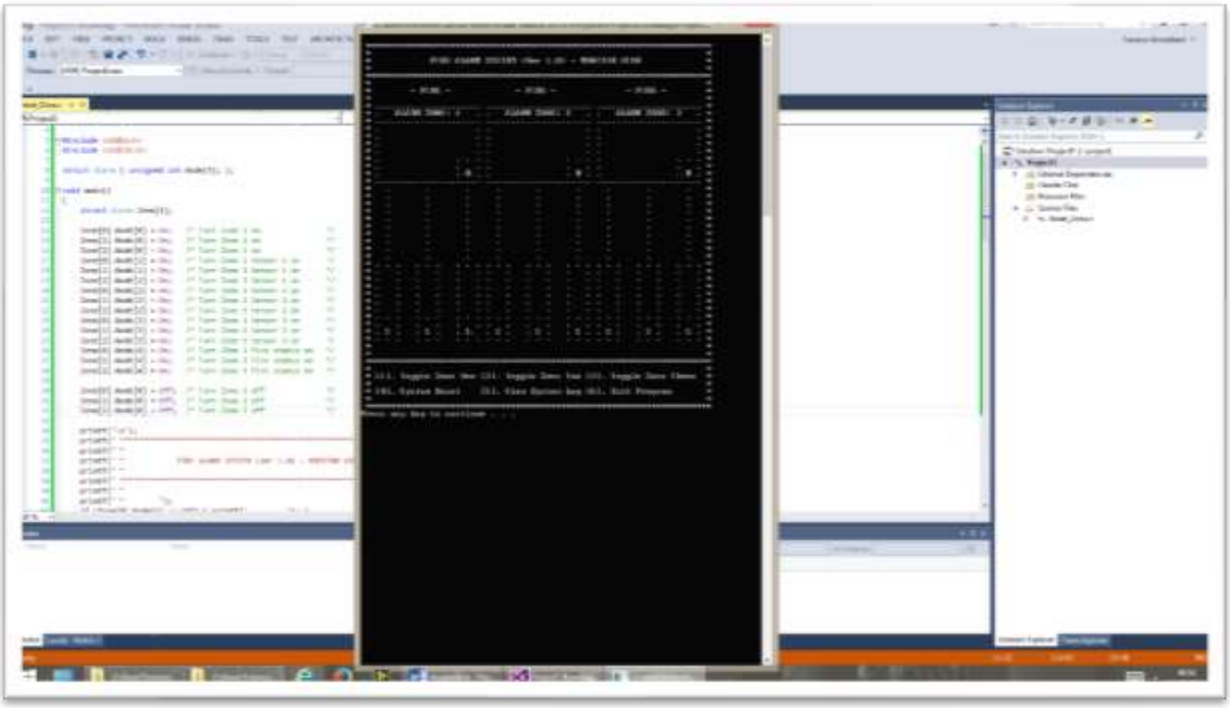


Figure 15 – Reset_Zone() test (pre compiler)

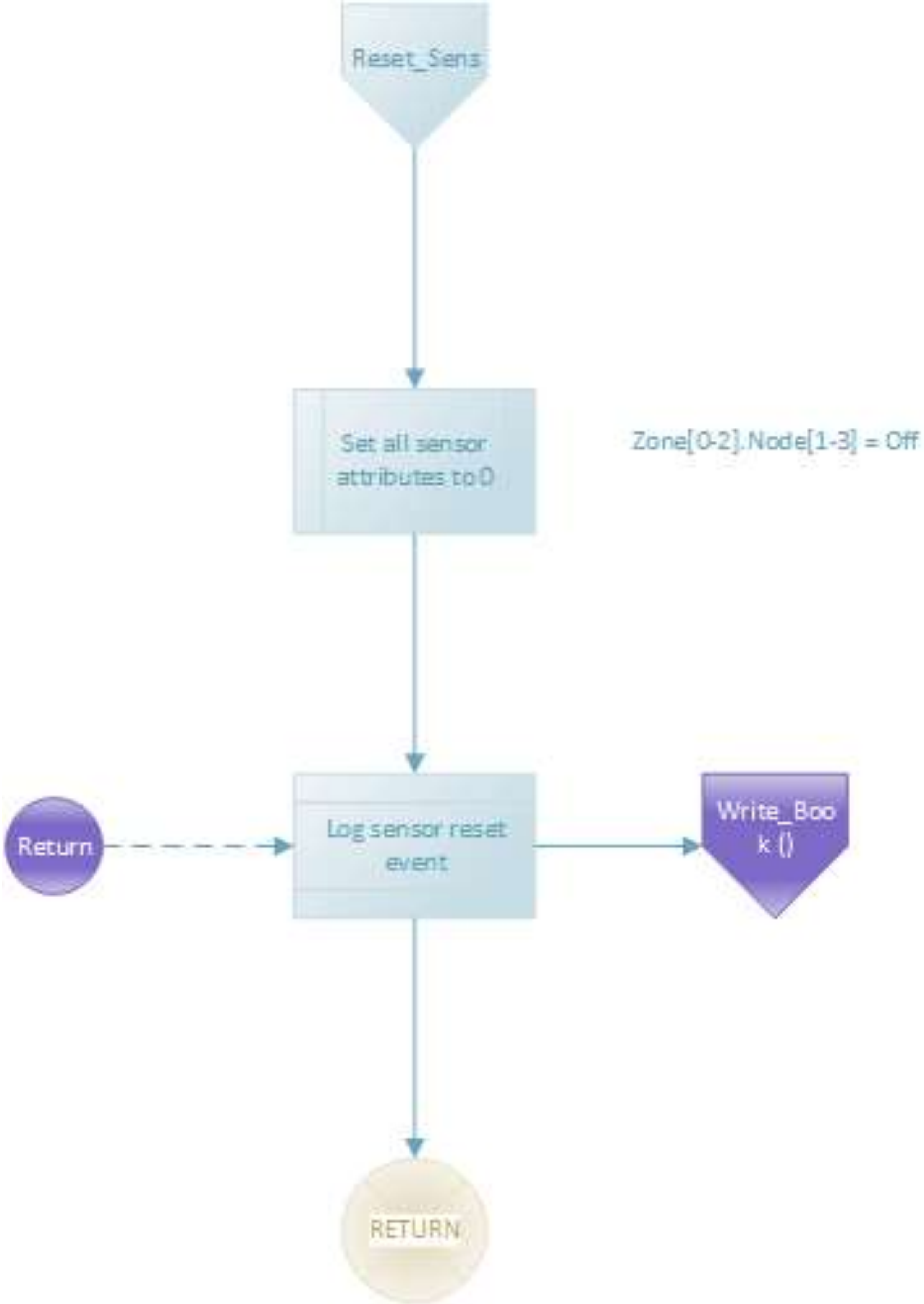


Figure 16 – Reset sensors

Reset_Sens() Function Test

PRE COMPILER TEST FOUR Reset_Sens() function Zone[0-2].Node[1-3] = Off Checked By: Terence Broadbent Date Checked: 04/02/2015

Table 8 – Pre compiler test four

This function system resets the sensors on/off value to off.

Zone variables were pre-loaded with various default settings then reset before testing and the function run several times to visually observe the output results to the screen.

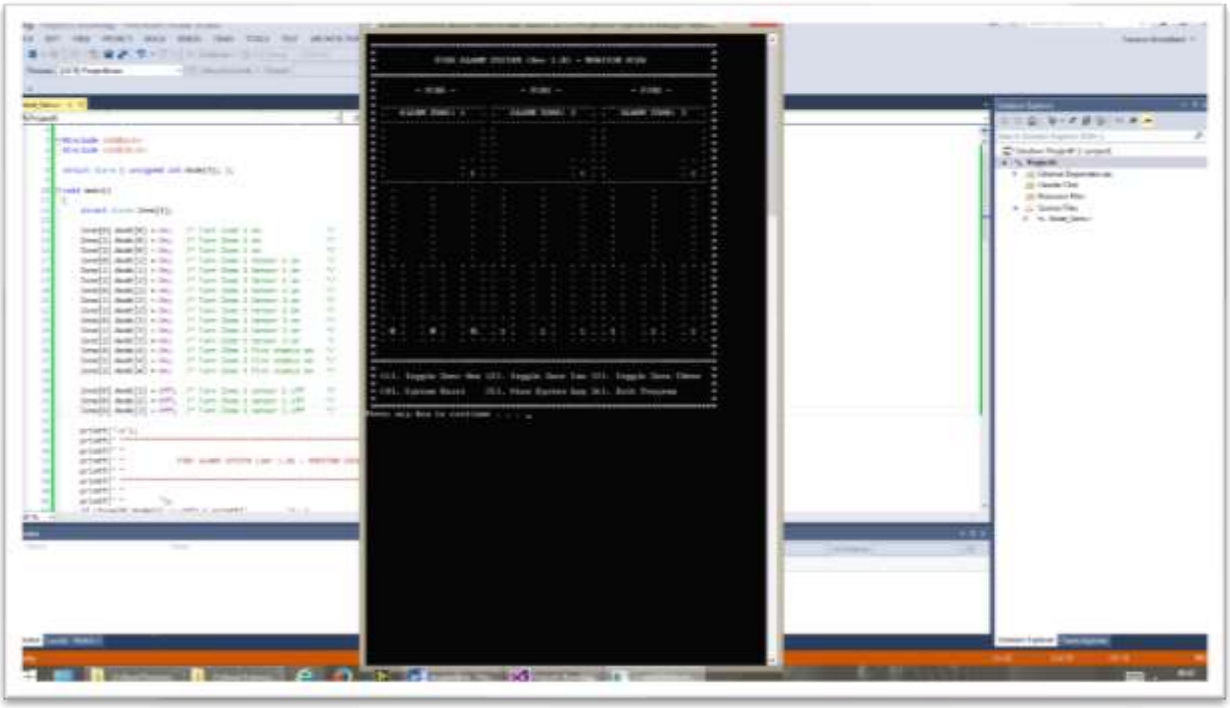


Figure 17 – Reset_Sens() test (pre compiler)

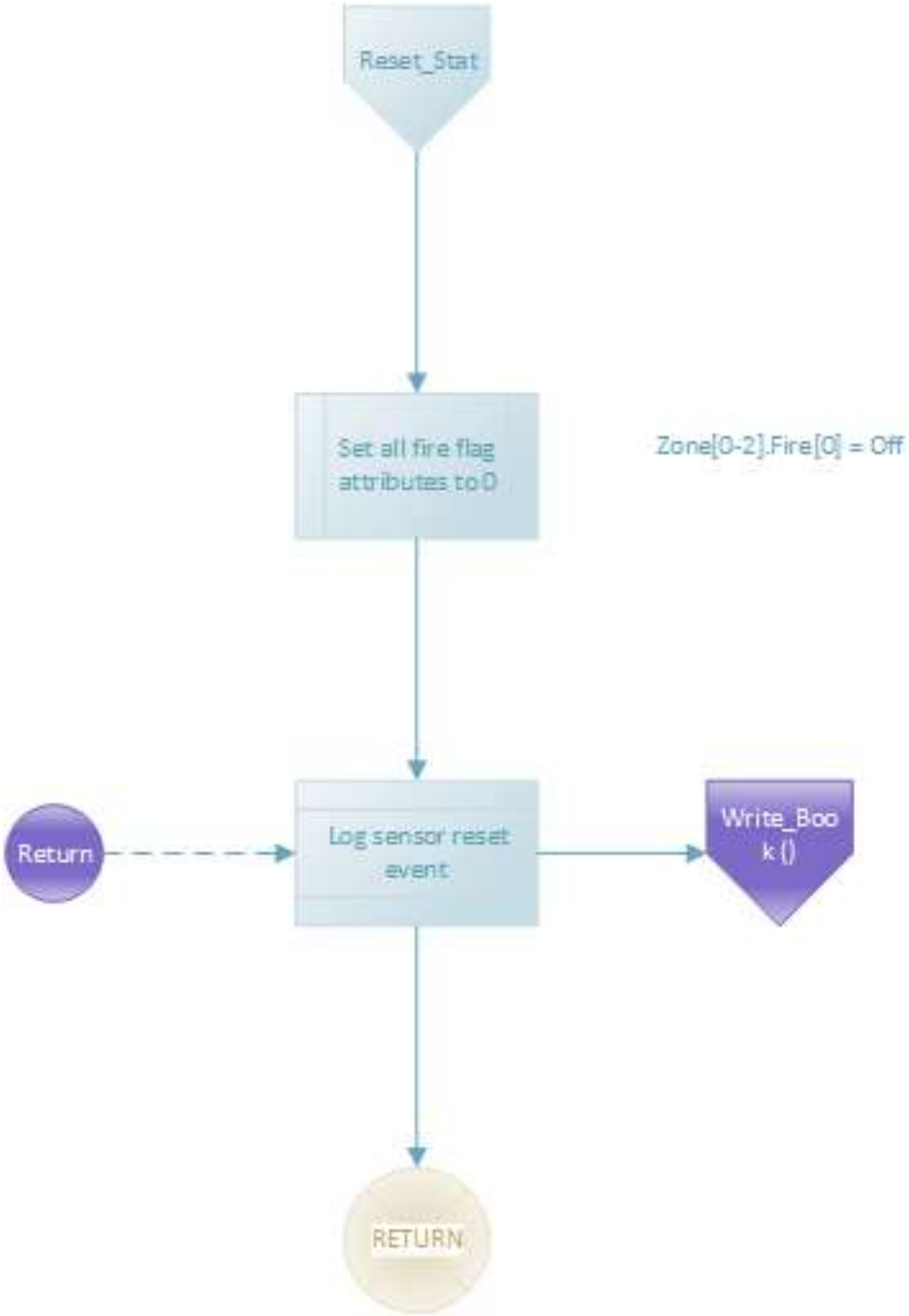


Figure 18 – Rest fire status

Reset_Stat() Function Test

PRE COMPILER TEST FIVE Reset_Stat() function Zone[0-2].Node[4] = Off Checked By: Terence Broadbent Date Checked:11/02/2015

Table 9 – Pre compiler test five

This function system resets the fire status settings within the zones.

Zone variables were pre-loaded with various default settings then reset before testing and the function run several times to visually observe the output results to the screen.

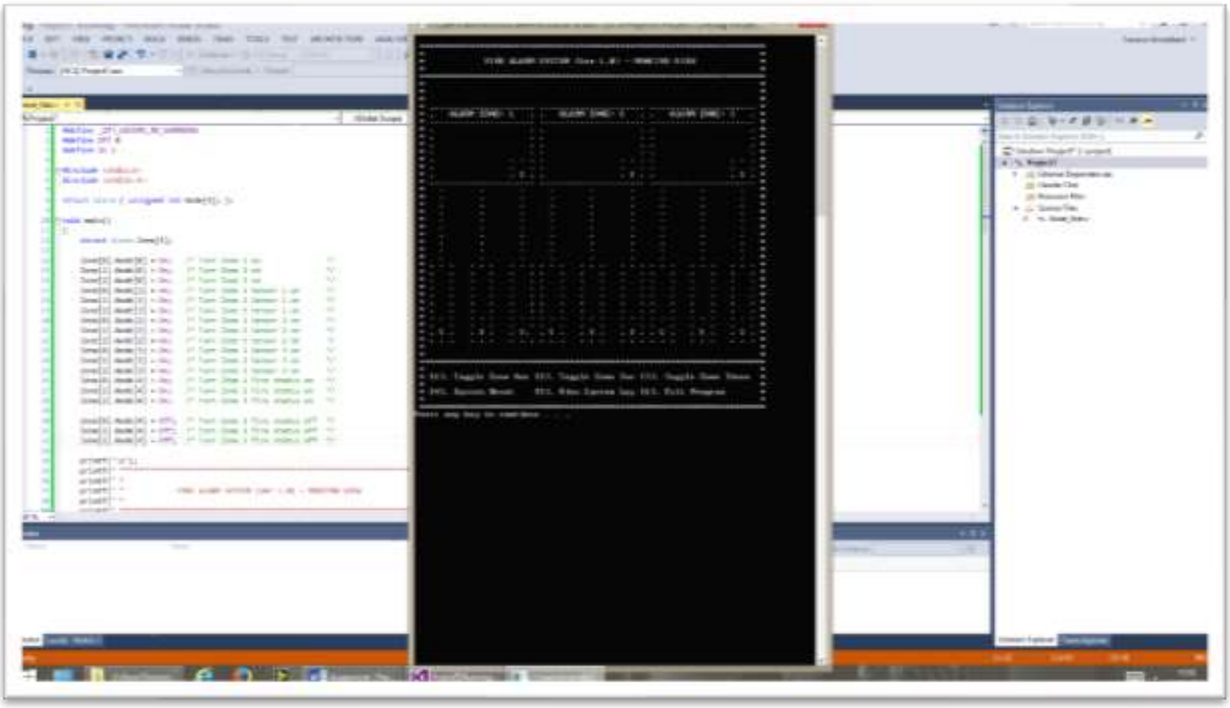


Figure 19 – Reset_Stat() test (pre compiler)

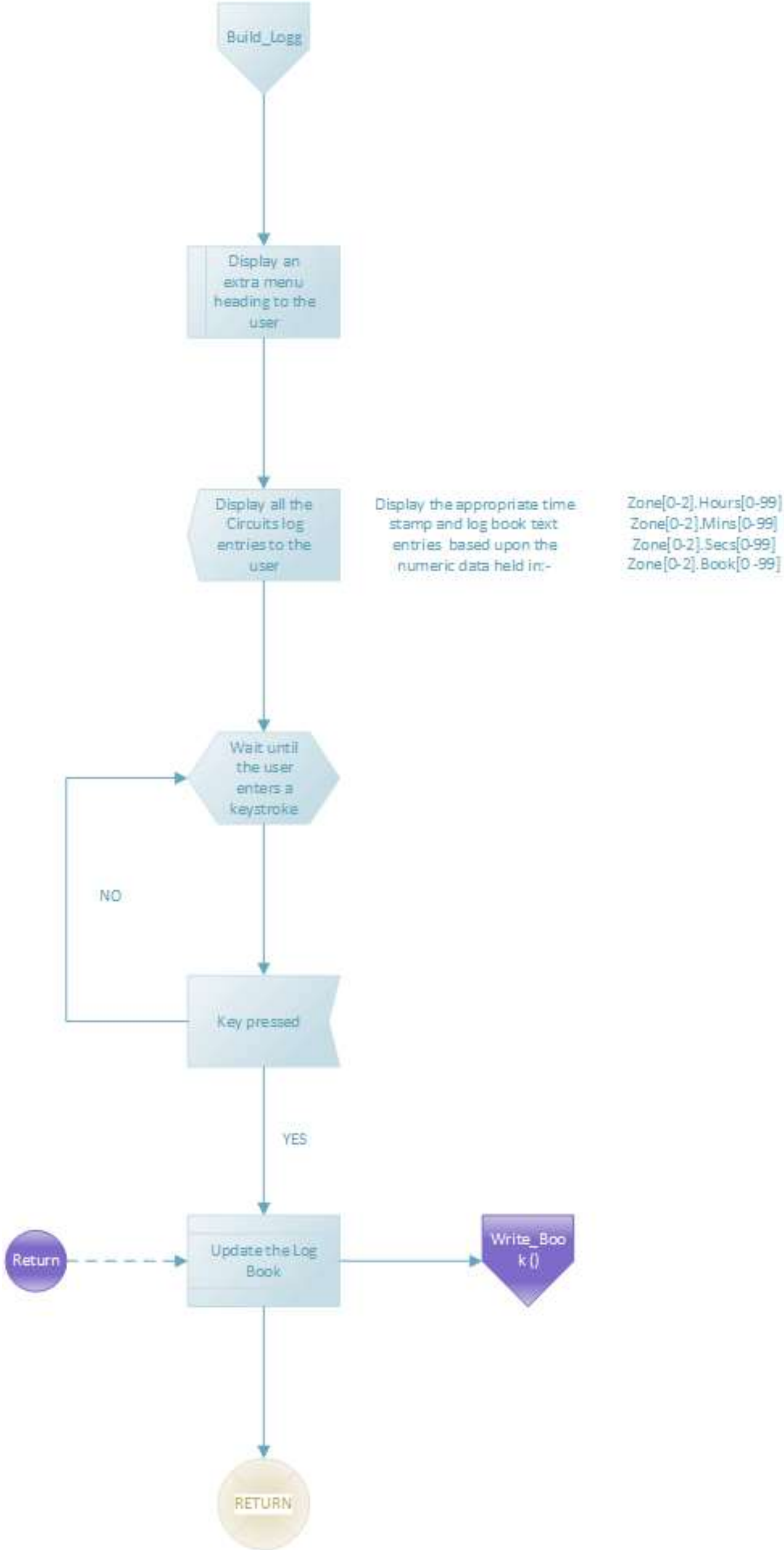


Figure 20 – Display the log book

Build_Logg() Function Test

PRE COMPILER TEST SIX
Build_Logg() function
Not defined
Checked By:
Date Checked:

Table 10 Pre compiler test six

This function displays a visual representation of the set/alarm events logged by the program to the user via the screen – up to a maximum of 100 entries are logged by the program, after which the logs will start over writing historic entries. It is also adhesively linked to the Write_Book() and ClockTimer() functions.

18/02/2015: After several unsuccessful or incomplete testing attempts it was decided to postpone this function test until post compilation i.e. actually running and interacting with the micro–controller.

This was mainly due to obtaining real time clock values and unfamiliarity at this stage on how the log book messages are to be coded and imprinted within the program variables.

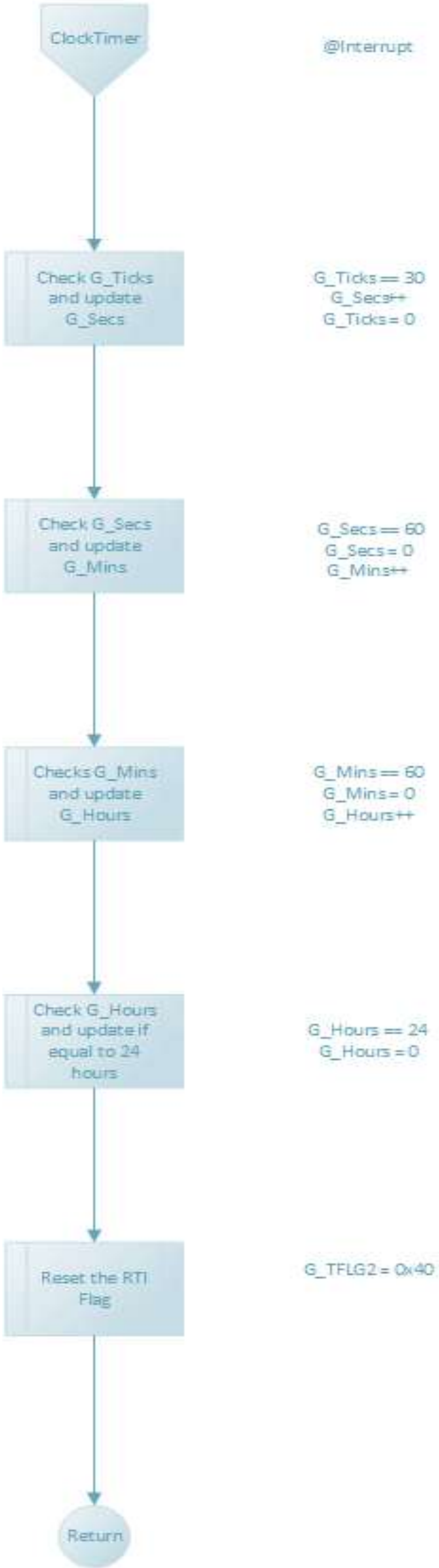


Figure 21 – The micro-controller interrupt clock

ClockTimer() Function Test

PRE COMPILER TEST SSEVEN ClockTimer() function G_Ticks, G_Secs, G_Mins, G_Hours Checked By: Terence Broadbent Date Checked: 18/02/2015

Table 11 – Pre compiler test seven

The micro-controller interrupts the operation of the 'Fire Alarm System' every 32.768 mS to run this particular function. In essence this sub-routine updates the global variable G_Ticks by 1 each time the interrupt occurs.

This variable is used by the program as an incremental counter - every 30 G_Ticks it updates another global variable G_Secs which in turn updates G_Mins which in turn updates G_Hours.

These global variables can then be used by the program to record the correct time entry of any log book messages.

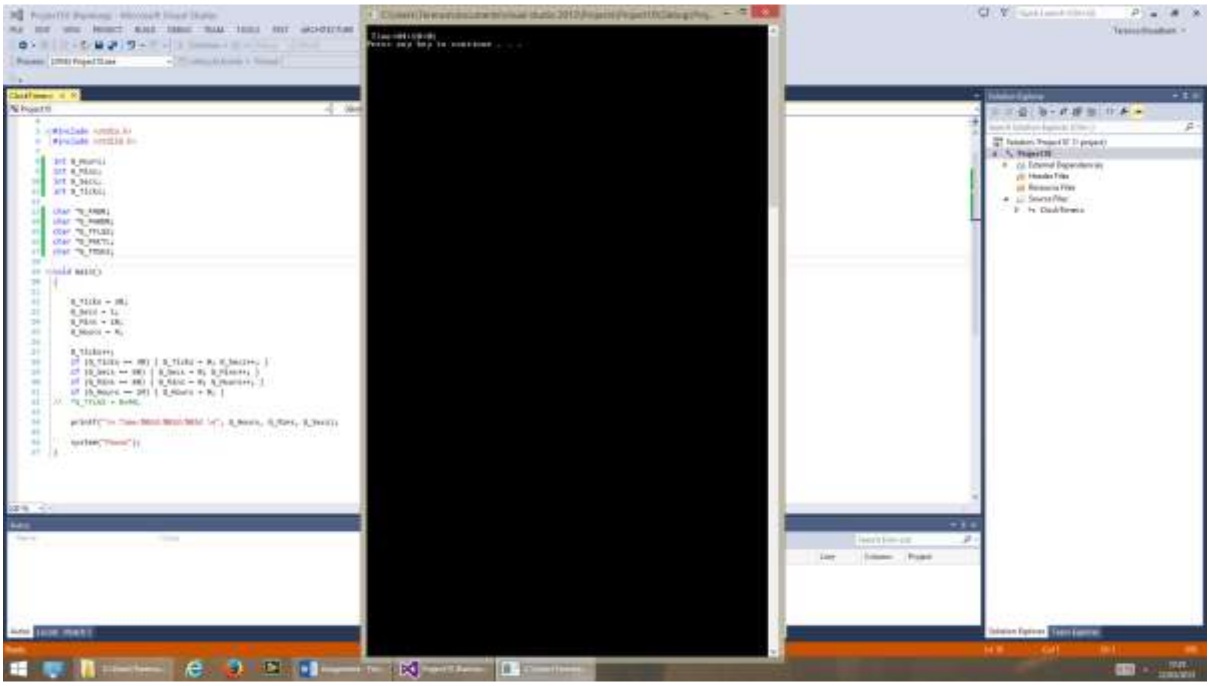


Figure 22 – ClockTimer() test (pre compiler)

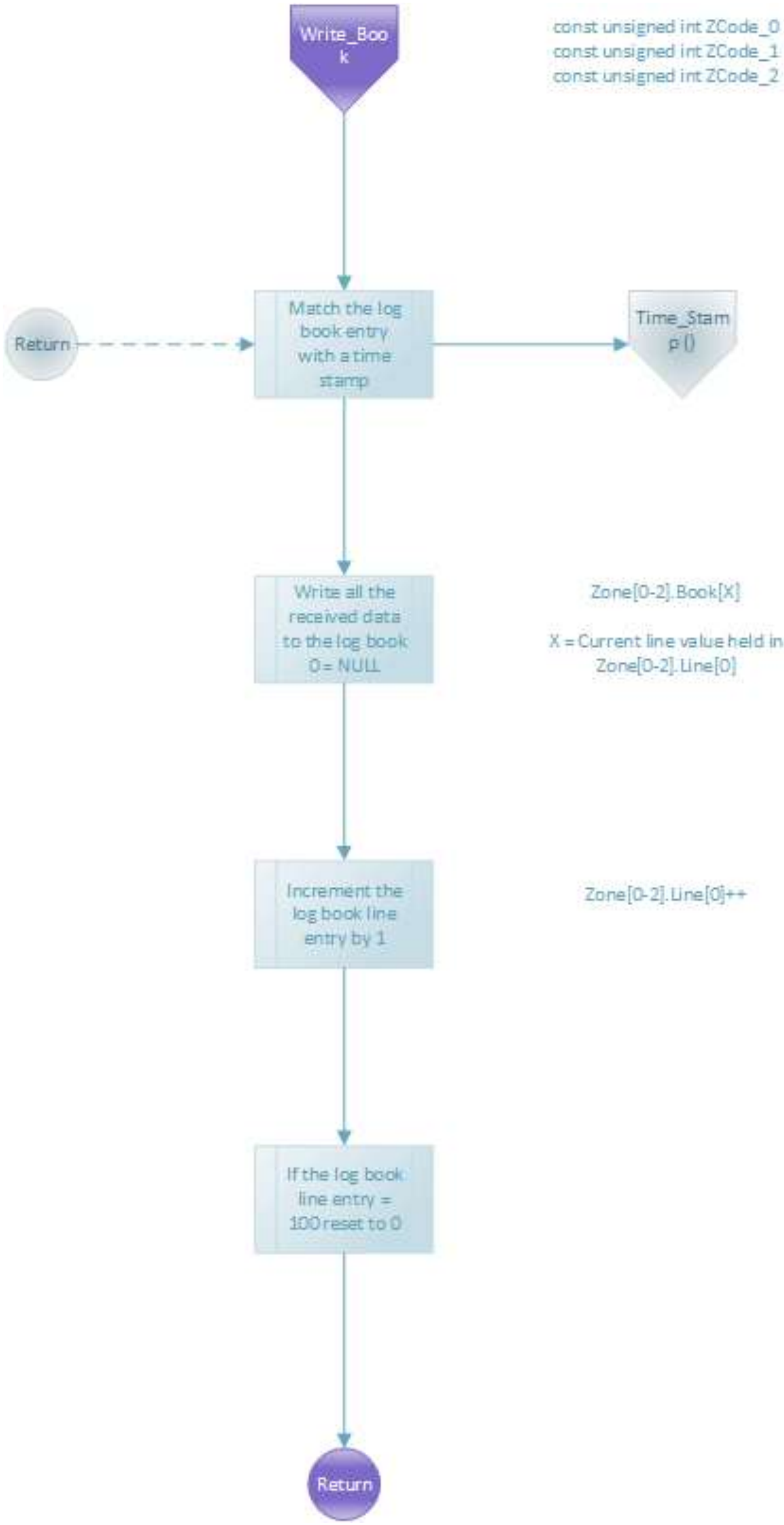


Figure 23 – Write log data

Write_Book() Function Test

PRE COMPILER TEST EIGHT
Write_Book() function
Not defined
Checked By:
Date Checked:

Table 12 – Pre compiler test eight

This function imprints a set/alarm event message in to the log book array [A4]. It is also adhesively linked to Build_Logg and Time_Stamp() function [B4].

18/02/2015: After several unsuccessful or incomplete testing attempts it was decided to postpone this function test until post compilation i.e. actually running and interacting with the micro–controller.

This was mainly due to obtaining real time clock values and unfamiliarity at this stage on how the log book messages are to be coded and imprinted within the program variables.

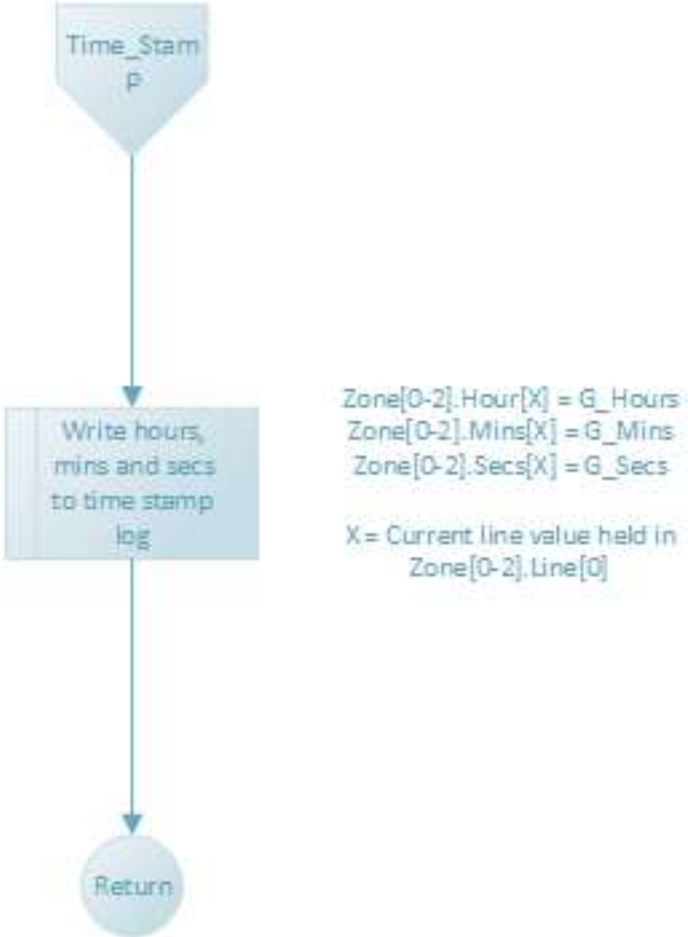


Figure 24 – Time stamp the log book entry

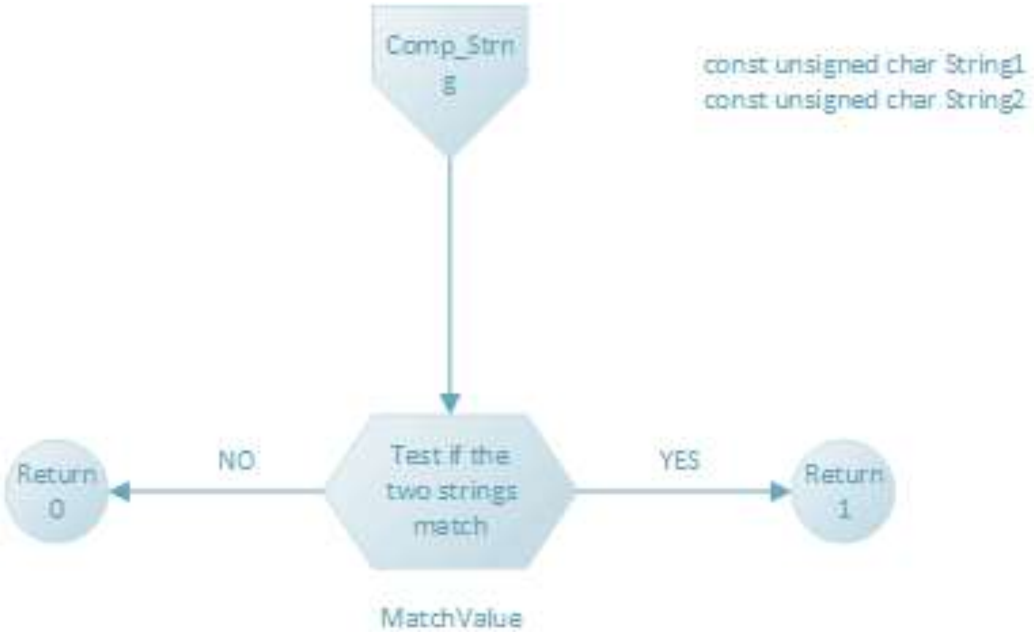


Figure 25 – Compare password strings

Time_Stamp() Function Test

PRE COMPILER TEST NINE Time_Stamp() function Checked By: Date Checked:

Table 13 – Pre compiler test nine

This function adds a time stamp to each log book entry [B4].

25/02/2015: The function was unable to be properly tested at the pre compile stage due to global variables originating from the micro-controller requiring updating every 32.768 mS.

However, due consideration was given for the use of the pre-set value from the Zone[0-2].Line[0] array – which increments by one each time a line is written to the log book. Thus displaying a numerical incremental output to the screen virtualising the clock counter system.

However, due to other testing restrictions – see Write_Book() and Build_Logg it was decided to postpone this function test until post compilation i.e. actually running and interacting with the micro–controller.

Comp_Strng() Function Test

PRE COMPILER TEST TEN
Comp_Strng() function
String1[5], String2[5], MatchValue
Checked By: Terence Broadbent
Date Checked: 23/02/2015

Table 14 – Pre compiler test ten

This function checks the individual characters of two strings that have been passed to the function. If every character matches the function returns a value ‘1’ else the function returns a value ‘0’ using the variable integer MatchValue.

Character strings ‘String1[0-4]’ and ‘String2[0-4]’ were preloaded with matching and incorrect text characters to ensure that the integer ‘MatchValue’ returns ‘1’ for True and ‘0’ for False.

This return value is used by the Reset_Sys() function as evidence of user authentication [A3].

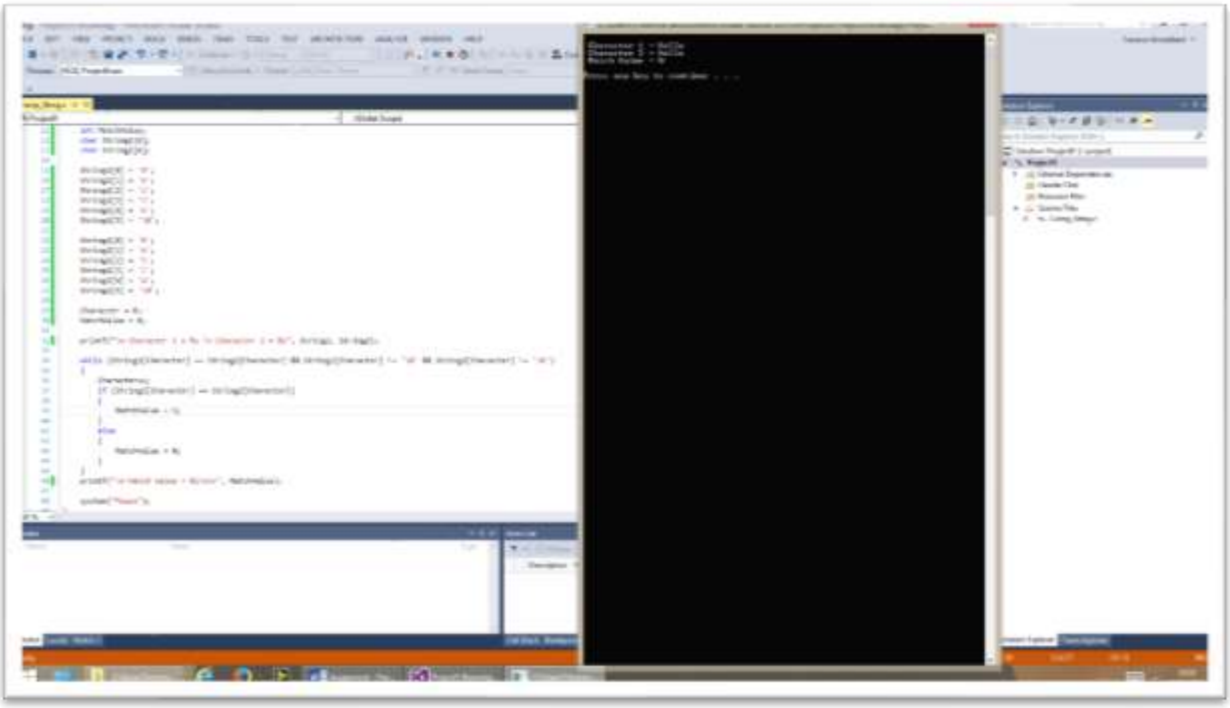


Figure 26 – Comp_Strng() test (pre compiler)

Decomposition Description

The ‘Fire Alarm System’ was progressively designed in tandem with the SDD modular program flowchart requirements for computer programmers.

A flowchart is a type of diagram that represents an algorithm, workflow or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows.

This approach is designed to emphasize the algorithm rather than the syntax of a specific programming language. The flowchart can be converted to several major programming languages if required.

The above diagrammatic representations illustrate one possible solution model to the given problem.

DATA DESIGN

To achieve the above stipulated design rationale the ANSI C code (See Appendix A) was constructed and coded to be as memory efficient and effective as possible by only storing the numeric values '0' (off) and '1' (on) wherever possible within the required program arrays.

Accordingly, all the zone property values are held within the single structure 'Alarm' with three alarm zones been created using the system command:

Struct Alarm Zone [3]

This means that there are three sets of individual and unique zone array variables to manipulate within the program - the main property array for any zone being '**Node**'.

Node [5]				
0	1	2	3	4
Zone On/Off	Sensor 1 On/Off	Sensor 2 On/Off	Sensor 3 On/Off	Fire Flag On/Off

Table 15 – Node [5]

Within each set all the zone activities can be switched on or off using the command:

Zone[0-2].Node[0-4] = On/Off

Once switched on via the menu options using key options [1], [2] or [3] the system will automatically update any activated sensors within the activated zone that may have been previously triggered but not displayed because the zone was switched off.

The program is continually polling the sensors to see if they have been triggered but will not report any activation unless the zone has actually been switched on by the user [B6].

Once a sensor within an activated zone has been triggered:

Zone[0-2].Node[1-3] = On

A visual representation of that activation will be displayed to the user and a 'Fire' confirmation flag will be displayed above the relevant zone [A3]:

Zone[0-2].Node[4] = On

Unlike the sensor which can be switched off after activation (simulating a burn out) or the zone toggled off the displayed 'Fire' flag requires a manual system reset from the main options using key [4].

Further, in order to successfully reset the system a unique five character password is required to be correctly entered by the user [A3].

The second set of property arrays for the zones are '**Book**' and '**Line**' [A4], [B4].

Book [100]
Currently holds values 0 – 17

Table 16 – Book [100]

The array 'Book' contains 100 memory slots for the following log codes (ZCode_0-2):

0	No zone activity to report.
1	Zone 1-3 time and log book created.
2	Zone 1-3 toggled.
3	Zone 1-3 successfully reset by the user.
4	An attempt was made to reset zone 1-3.
5	Zone 1-3 reset.
6	Zone 1-3 sensors reset.
7	Zone 1-3 fire flag reset.
8	Zone 1-3 system log displayed to the user.
9	Sensor 1 has detected a fire in zone 1.
10	Sensor 2 has detected a fire in zone 1.
11	Sensor 3 has detected a fire in zone 1.
12	Sensor 4 has detected a fire in zone 2
13	Sensor 5 has detected a fire in zone 2.
14	Sensor 6 has detected a fire in zone 2.
15	Sensor 7 has detected a fire in zone 3.
16	Sensor 8 has detected a fire in zone 3.
17	Sensor 9 has detected a fire in zone 3.

Table 17 – Log book reference table

This design functionality not only saves memory space but also aids redundancy issues by allowing more log book messages to be added if the program is expanded in the future. The system command that controls this within the program is:

Write_Book(Zone, 5, 0, 0)

The above the log book entry for example has recorded that Zone 1 has been reset (5) and that nothing (0) has occurred within Zones 2 and 3.

Of course each time an entry is recorded within the logbook a new line needs to be allocated.

The system command that manages this is:

Zone[0-2].Line[0] = 0-99

Line [1]
Holds values 0 – 99

Table 18 – Line [1]

Every time a log book entry is written, the above zone variable increments by one – thus indicating the next blank line number in the log book for the system to write too.

If however, 100 entries are found to have been entered into the log book the system resets the line values to '0' and continues by overwriting previously documented log book entries.

The log book also requires 100 matching memory slots for each of the following hours, minutes and seconds – hence the last property arrays within each zone are '**Hour**', '**Mins**' and '**Secs**'.

Hour [100]
Holds values 0 – 24

Table 19 – Hour [100]

Mins [100]
Holds values 0 – 60

Table 20 – Mins [100]

Secs [100]
0 – 60

Table 21 – Secs [100]

The above clock values are assigned the current micro-controller G_Hours, G_Mins and G_Secs from the global variables (G_) shown below before imprinting into their own specified zone arrays.

Zone[0-2].Hour[0-99] = G_Hours

Zone[0-2].Mins[0-99] = G_Mins

Zone [0-2].Secs[0-99] = G_Secs

The global variables are assigned their own values approximately every 32.768 mS using the micro-controller assigned interrupt function ClockTimer().

In short each time a log entry in created a matching time stamp is created – and when the user displays the log book to the screen using menu option key [5] both are displayed together to the screen to provide continuity for the entries – See page 74.

16 bit Sensor Triggers

Detailed below are the micro-controller setting required for the 16 bit sensor triggers - ParralPort() function [B6].

Variable Name	Memory Location	Value	Description
*G_PADR	0x0000	& 0x1	Reads single bit PA0 from the register using mask 0x1.
*PEDR	0x000a		Reads bits PE0-PE7 from the register as a char byte.

Table 22 - 16 bit sensor triggers

PADR – Port A data register.

Port A is an eight bit general purpose I/O port with a data register (PORTA) and a data direction register (DDRA) [See G_PADDR]. By using the mask '0x1' only the single bit value held in PA0 is read.

Bit 7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
PA7	PA6	PA5	PA4	PA3	PA2	Pa1	**PA0**
0/1	0/1	0/1	0/1	0/1	0/1	0/1	**0/1**

Figure 27 - PADR

PEDR – Port E data register.

Port E is an eight bit input only port with data register (PORTE) that is also used as the analog input port for the analog to digital converter. This port is read as one complete char byte – for example char byte '93' would equal 10010011 in binary.

Bit 7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

Figure 28 - PEDR

8 bit Keyboard Strokes

Detailed below are the micro-controller settings required for the 8 bit keyboard strokes - SerialPort() function [B3].

Variable Name	Memory Location	Value	Description
*SCDR	0x2f		The actual keystroke character
*SCSR	& 0x2e	0x20	A key has been pressed

Table 23 - 8 bit keyboard strokes

SCDR – Serial communications data register.

Reading SCDR retrieves the last byte received in the receive data buffer from the keyboard.

Bit 7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R7/T7	R6/T6	R5/T5	R4/T4	R3/R3	R2/T2	R1/T1	R0/T0
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

Figure 29 - SCDR

SCSR – SCI status register.

Clear the RDRF flag by reading SCSR with RDRF set then reading SCDR – ‘0x20’ sets the RDRF flag.

Bit 7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TDRE	TC	RDRF	IDLE	OR	NF	FE	0
0	0	1	0	0	0	0	0

Figure 30 - SCSR

Micro-controller Clock Time

Detailed below are the micro-controller setting required for the ClockTimer() function [B2].

Variable Name	Memory Location	Value	Description
*G_PADDR	0x0001	0xfe	Port A data direction
*G_TFLG2	0x25	0x40	Reset RTI flag
*G_PACTL	0x26	0x03	Set pulse period to 32.768 mS
*G_TMSK2	0x24	0x40	Enables the interrupt source

Table 24 – Micro-controller clock time

PADDR – Port A Data Direction Register

Port A is an eight bit general purpose I/O port with a data register (PORTA) and a data direction register (DDRA).

Bits in DDRA are cleared by writing a zero to the corresponding bit positions for example 'oxfe' sets DDA0 as input and all others as output.

Bit 7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
1	1	1	1	1	1	1	0

Figure 31 - DDRA

TFLG2 – Timer interrupt flag 2.

Bits within this register indicate when certain timer system events have occurred. Coupled with the four high-order bits of TMSK2, the bits of TFLG2 allow the timer subsystem to operate in either a polled or **interrupt** driven system. Each bit of TFLG2 corresponds to a bit in TMSK2 in the same position.

Bits in TFLG2 are cleared by writing a one to the corresponding bit positions – for example '0x40' resets the real time interrupt flag at a rate based on PACTL.

Bit 7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TOF	RTIF	PAOVF	0	0	0	0	0
0	1	0	0	0	0	0	0

Figure 32 – TFLG2

PACTL – Pulse accumulator control.

The pulse accumulator can be used either to count events or measure the duration of a particular event.

Bits in PACTL are cleared by writing a zero to the corresponding bit positions – for example ‘0x03’ sets the real time interrupt period to 32.768 mS.

Bit 7	6	5	4	3	2	1	0
0	PAEN	PAMOD	PEDGE	0	14/05	RTR1	RTR0
0	0	0	0	0	0	1	1

Figure 33 – PACTL

TMSK2 – Timer interrupt mask 2.

Bits in TMSK2 correspond bit for bit with flag bits in TFLG2. Setting any of this bits enables the corresponding interrupt source. TMSK2 can be written only once in the first 64 cycles out of reset in normal modes, or at any time in special modes.

Bit 7	6	5	4	3	2	1	0
TOI	RTII	PAOVI	PAII	0	0	PR1	PR0
0	1	0	0	0	0	0	0

Figure 34 – TMSK2

PART B

PORTING THE CODE

In order to run the Visual studio ANSI C code on the micro-controller it must be first converted to raw machine code [B7].

Cosmic C Cross Compiler

The first step to take in order to successfully port the C code for use on the MC68HC11F1 micro-controller is to cross compile it using the command Cram *filename.c* within a cmd.exe windows shell environment (Cosmic Software, 2002).

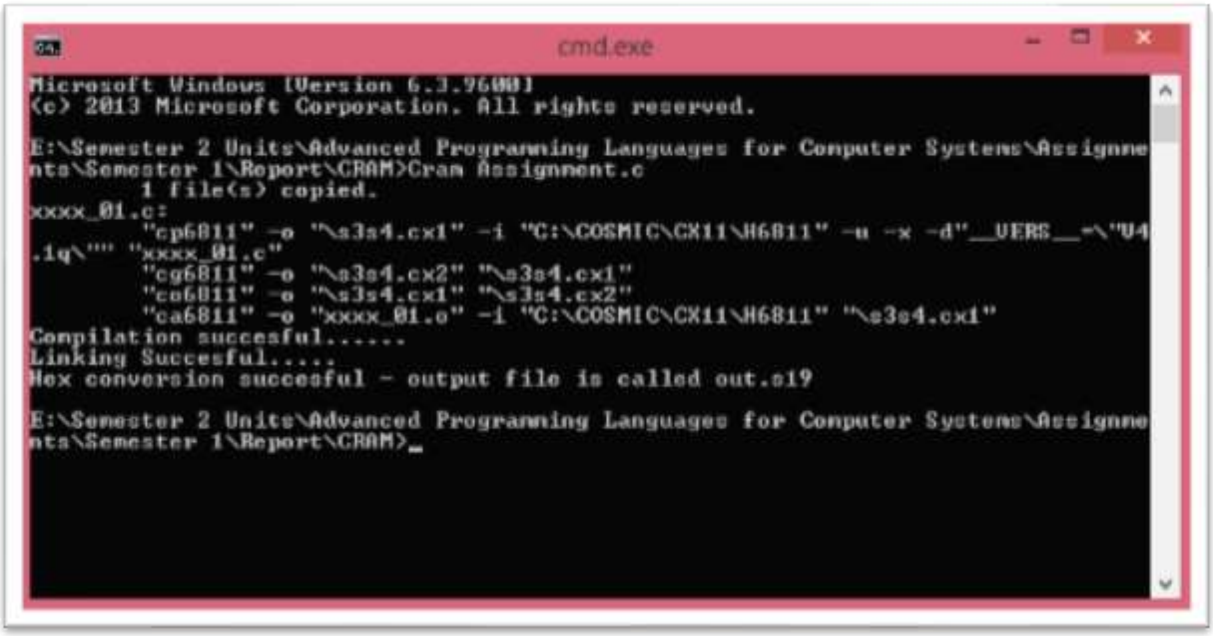


Figure 35 – Cross compiler screen

The cosmic compiler reads the C code file saved within the same directory and creates and writes a file in the same location called ‘outs19.txt’ which contains the raw machine code to be loaded into the micro-controller.

The next step in the process is to upload the raw machine code held within the text file into the microcontroller, this is achieved by using the communication software HyperTerminal.

First however, we need to take a few steps to set up the HyperTerminal – The first thing we need to do is create a new connection and give it a name.

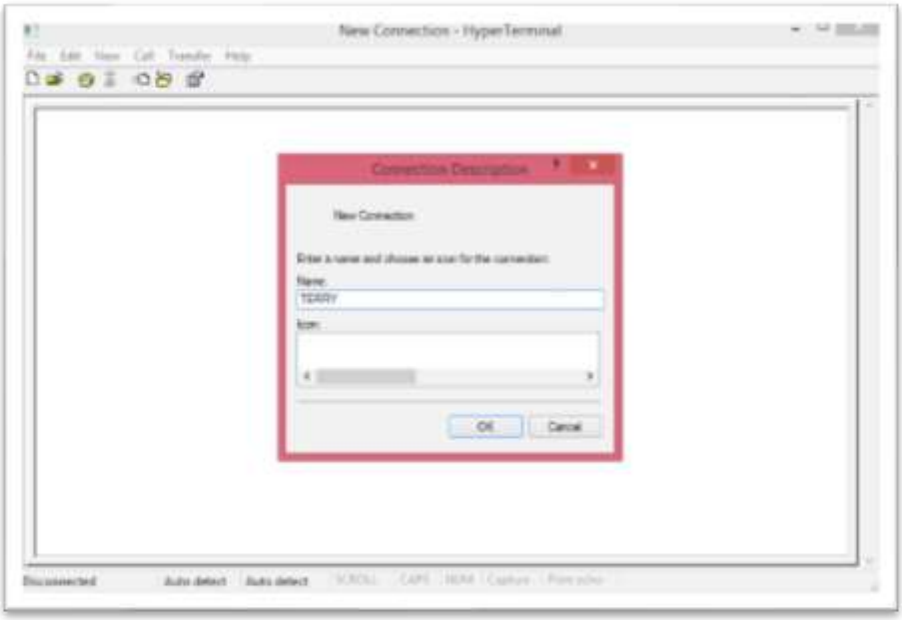


Figure 36 – Naming the HyperTerminal

Then connect to the micro-controller using parallel port COM1.

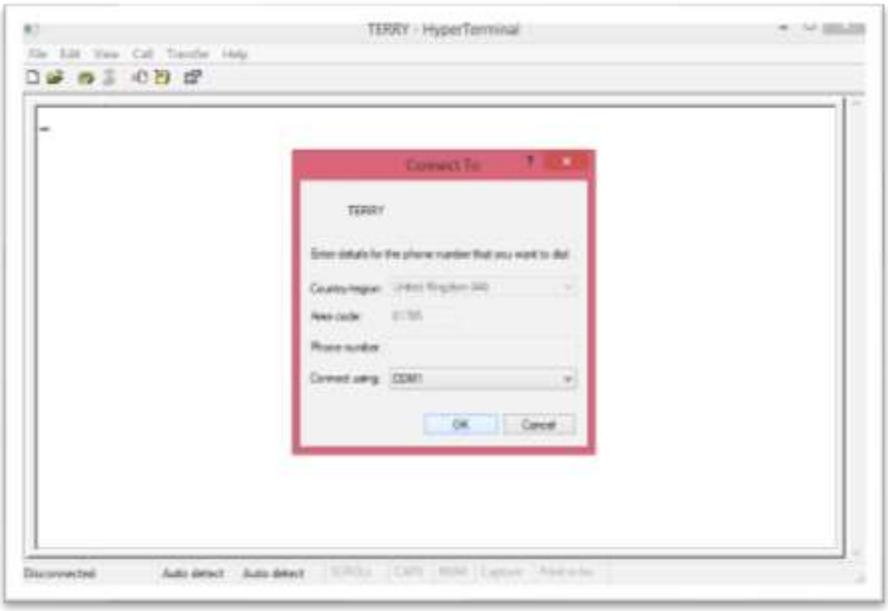


Figure 37 – Assigning the HyperTerminal port number

Next we need to set up the port settings – simple click 'Restore Defaults' to achieve this.

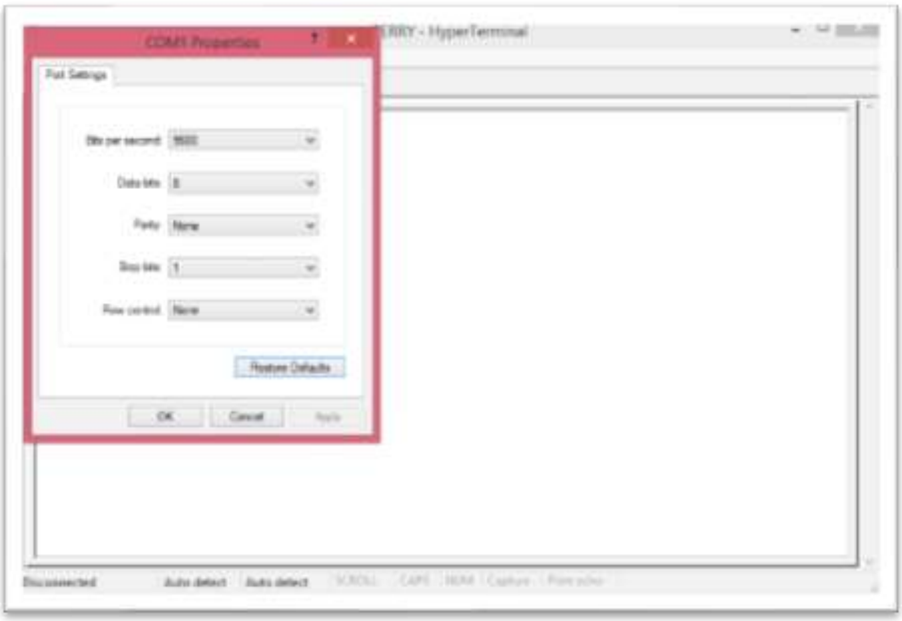
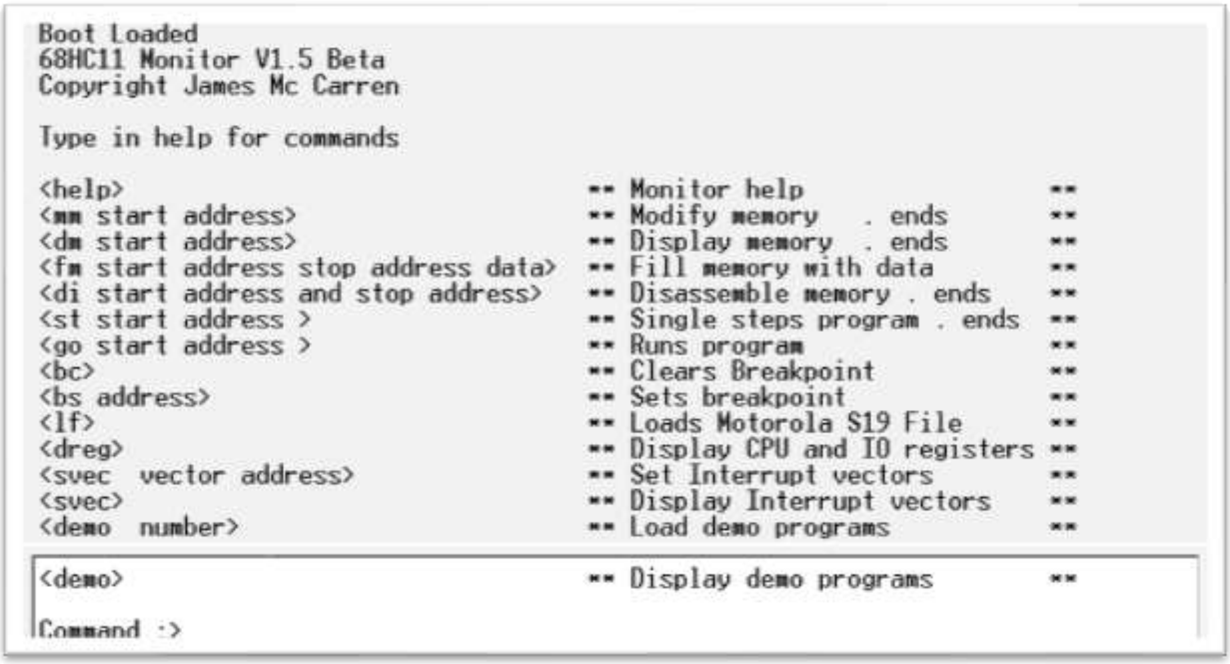


Figure 38 – Configuring the HyperTerminal communications settings

Once the reset button is pressed on the micro-controller a boot menu system will be appear within the HyperTerminal console screen.

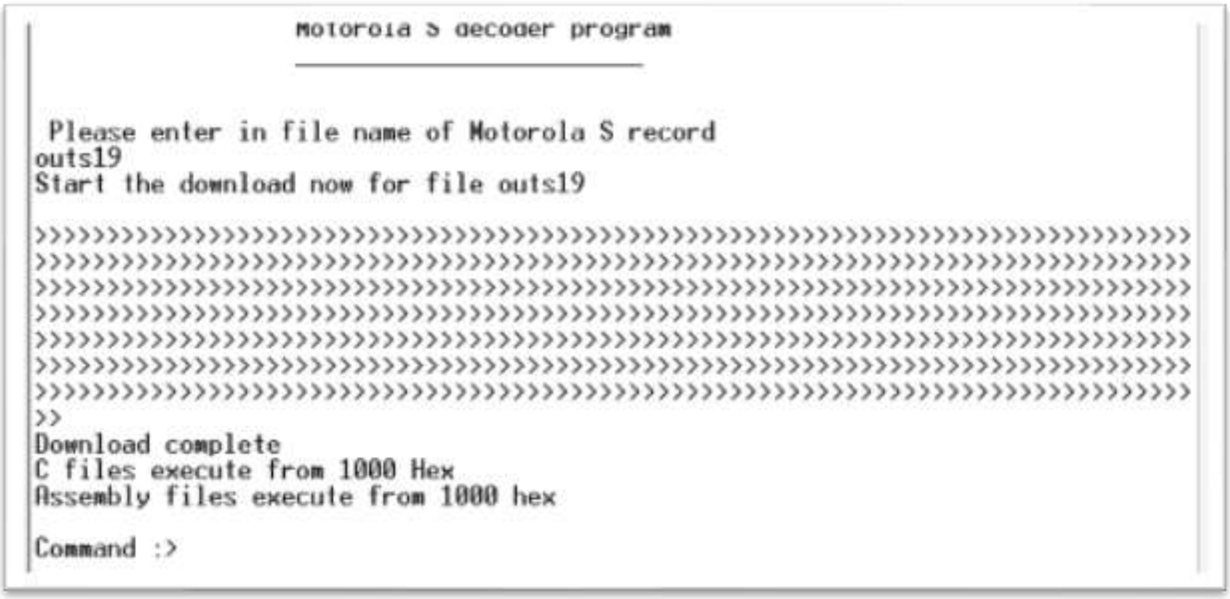


```
Boot Loaded
68HC11 Monitor V1.5 Beta
Copyright James Mc Carren

Type in help for commands

<help>                                    ** Monitor help                 **
<mm start address>                        ** Modify memory   . ends       **
<dm start address>                        ** Display memory  . ends       **
<fm start address stop address data>      ** Fill memory with data        **
<di start address and stop address>       ** Disassemble memory . ends    **
<st start address >                       ** Single steps program . ends  **
<go start address >                       ** Runs program                 **
<bc>                                      ** Clears Breakpoint            **
<bs address>                              ** Sets breakpoint              **
<lf>                                      ** Loads Motorola S19 File      **
<dreg>                                    ** Display CPU and IO registers **
<svec  vector address>                    ** Set Interrupt vectors        **
<svec>                                    ** Display Interrupt vectors    **
<demo  number>                            ** Load demo programs           **
<demo>                                    ** Display demo programs        **

Command :>
```

Figure 39 – Micro-controller boot menu

Now we are at the stage where we can actually upload the raw machine code into the micro-controller by typing in the system command 'lf' (load file).

The booted menu will then ask you for the file name – type in outs19 and press return. Next select Transfer and Send Text File from the HyperTerminal menu system.



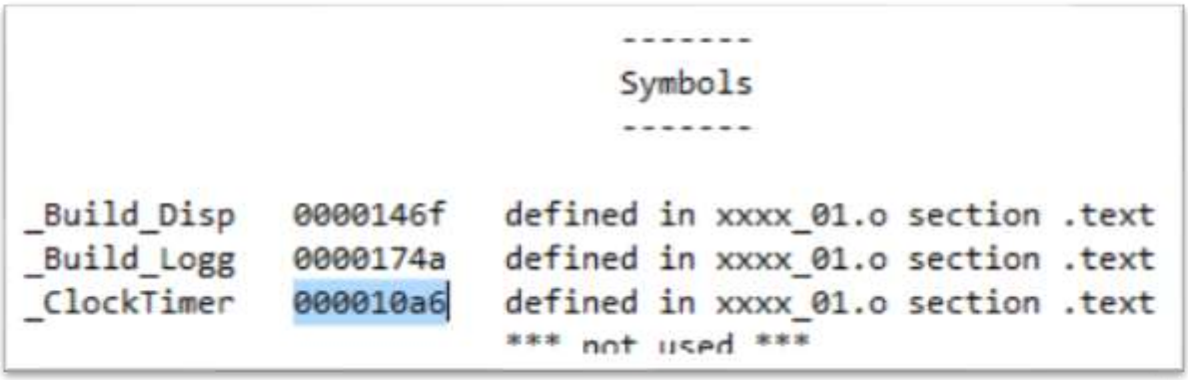
```
Motorola S decoder program

Please enter in file name of Motorola S record
outs19
Start the download now for file outs19

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>
Download complete
C files execute from 1000 Hex
Assembly files execute from 1000 hex

Command :>
```

Figure 40 – Outs19.txt file being loaded into the micro-controller.

Now, before proceeding any further we must set up the micro-controller interrupt vector table by looking at the linker listing file to obtain the memory address of the function ClockTimer().

The obtained memory address then needs to be remapped to another hardware address – the RTIF vector Svec 7.

Opening the file map.map we can obtain the _ClockTimer memory location – in this particular case '10a6'.



```
                          -------
                          Symbols
                          -------

_Build_Disp    0000146f   defined in xxxx_01.o section .text
_Build_Logg    0000174a   defined in xxxx_01.o section .text
_ClockTimer    000010a6   defined in xxxx_01.o section .text
                          *** not used ***
```

Figure 41 – Linker listing from map.map file

Now we need to type into the HyperTerminal command line ‘Svec 7 10a6’ to set the micro-controller real time interrupt to ClockTimer().

Finally typing ‘go 1000’ will execute the program - The HyperTerminal console screen should now look like this.

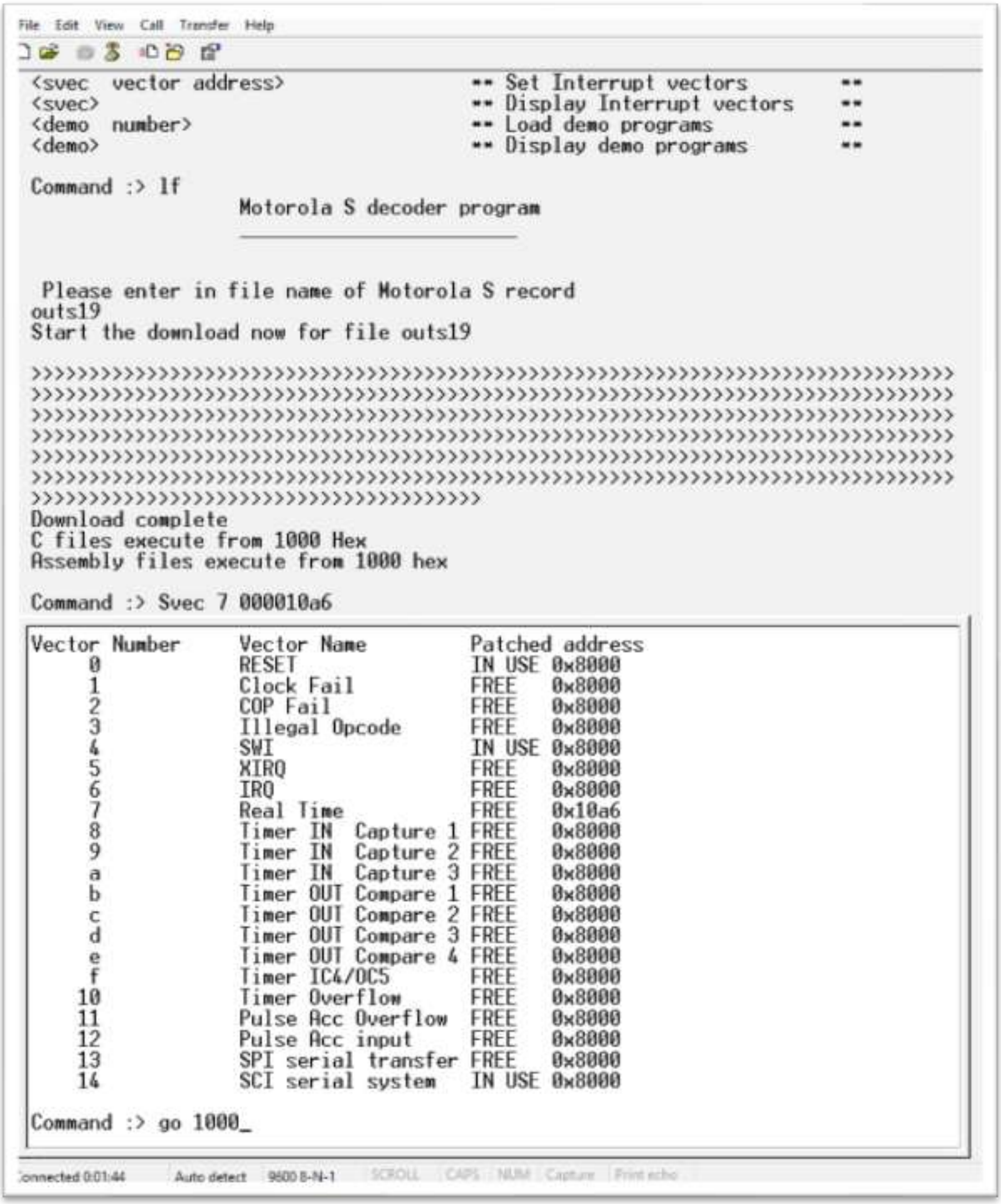


Figure 42 – HyperTerminal load file sequence

INTERFACE DESIGN

Human interface design was strongly considered when coding the 'Fire Alarm System' menu options. It was always intended from the start that it should be user friendly, quick and convenient – By stipulating that the user could only press 6 keyboard numbers (1 – 6) limited any errors that a user would or could make.

Logical design meant that number 1-3 activated zones 1-3 with 4-6 following naturally within the code and display options.

```
*******************************************************************
*                                                                 *
* [1]. Toggle Zone One [2]. Toggle Zone Two [3]. Toggle Zone Three *
*                                                                 *
* [4]. System Reset    [5]. View System Log [6]. Exit Program     *
*                                                                 *
*******************************************************************
```

Figure 43 – Menu options

Overview of the User Interface

Sub menus required for menu options 4 and 5 simply extended themselves onto the base of the exsiting menu system providing a continuous visual flow of information.

```
*******************************************************************
*                                                                 *
*              FIRE ALARM SYSTEM - SYSTEM RESET                   *
*                                                                 *
*******************************************************************

Please enter the five figure security Password > abort

Security password authenticated.
The fire alarm system has now been re-set to default values.

*******************************************************************

Please enter any key to continue....

Connected 0:03:31   Auto detect   9600 8-N-1   SCROLL   CAPS   NUM   Capture   Print echo
```

Figure 44 - Sub menus

Screen Images

The following seven screen shots are taken from the working program.

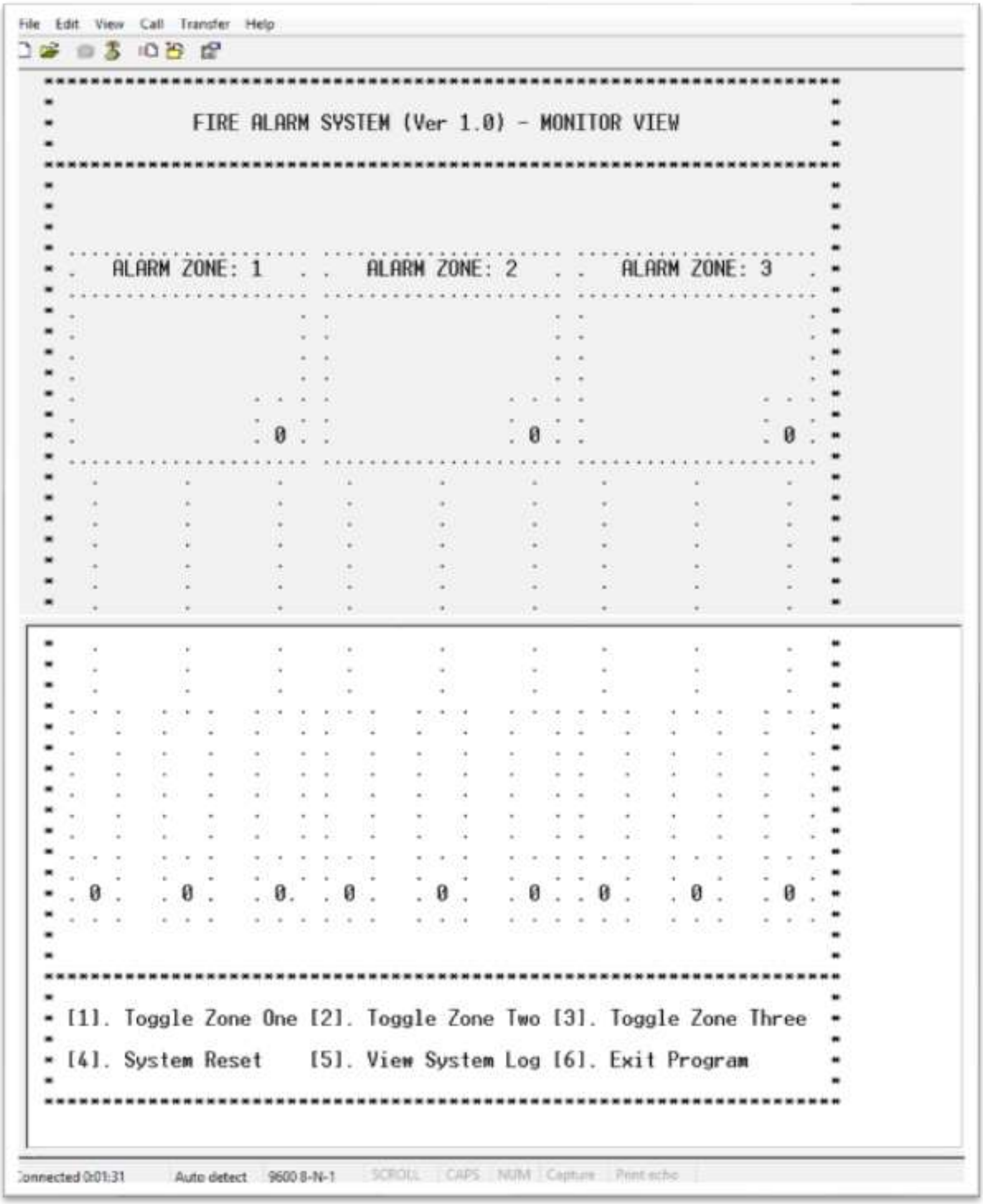


Figure 45 – Opening screen display and menu options (post compiler)

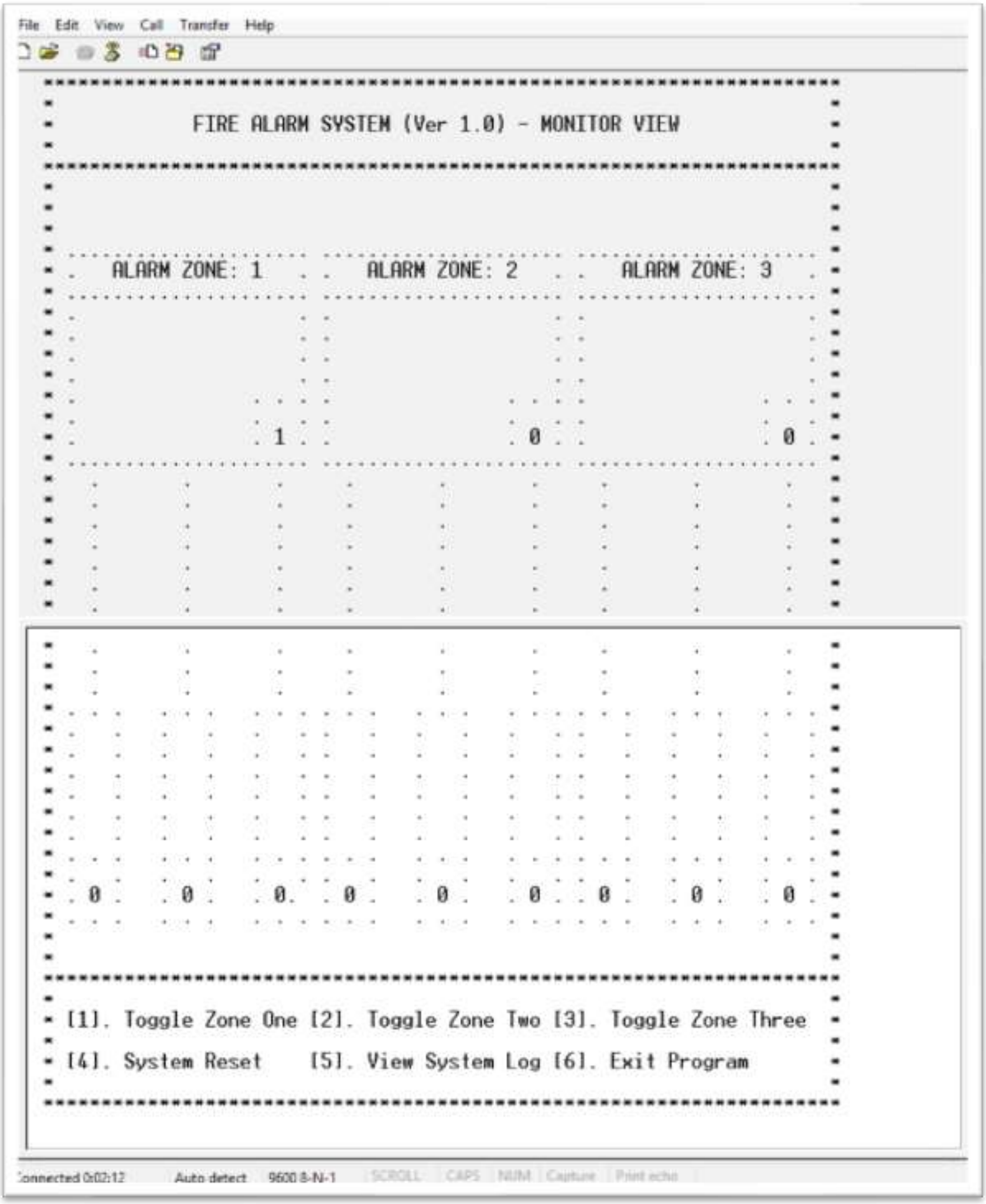


Figure 46 – Screen display with zone 1 activated (post compiler)

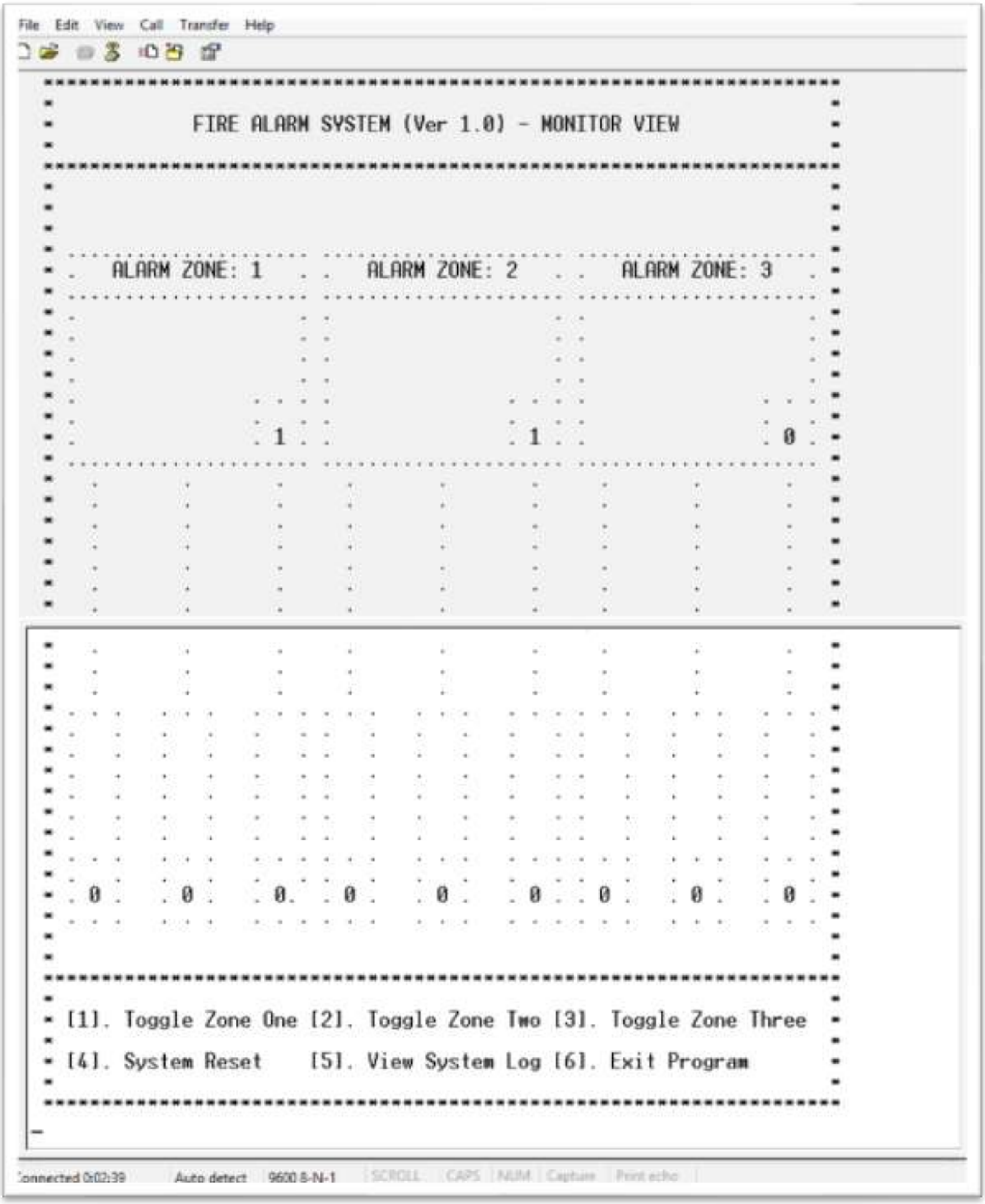


Figure 47 – Screen display with zone 1 and 2 activated (post compiler)

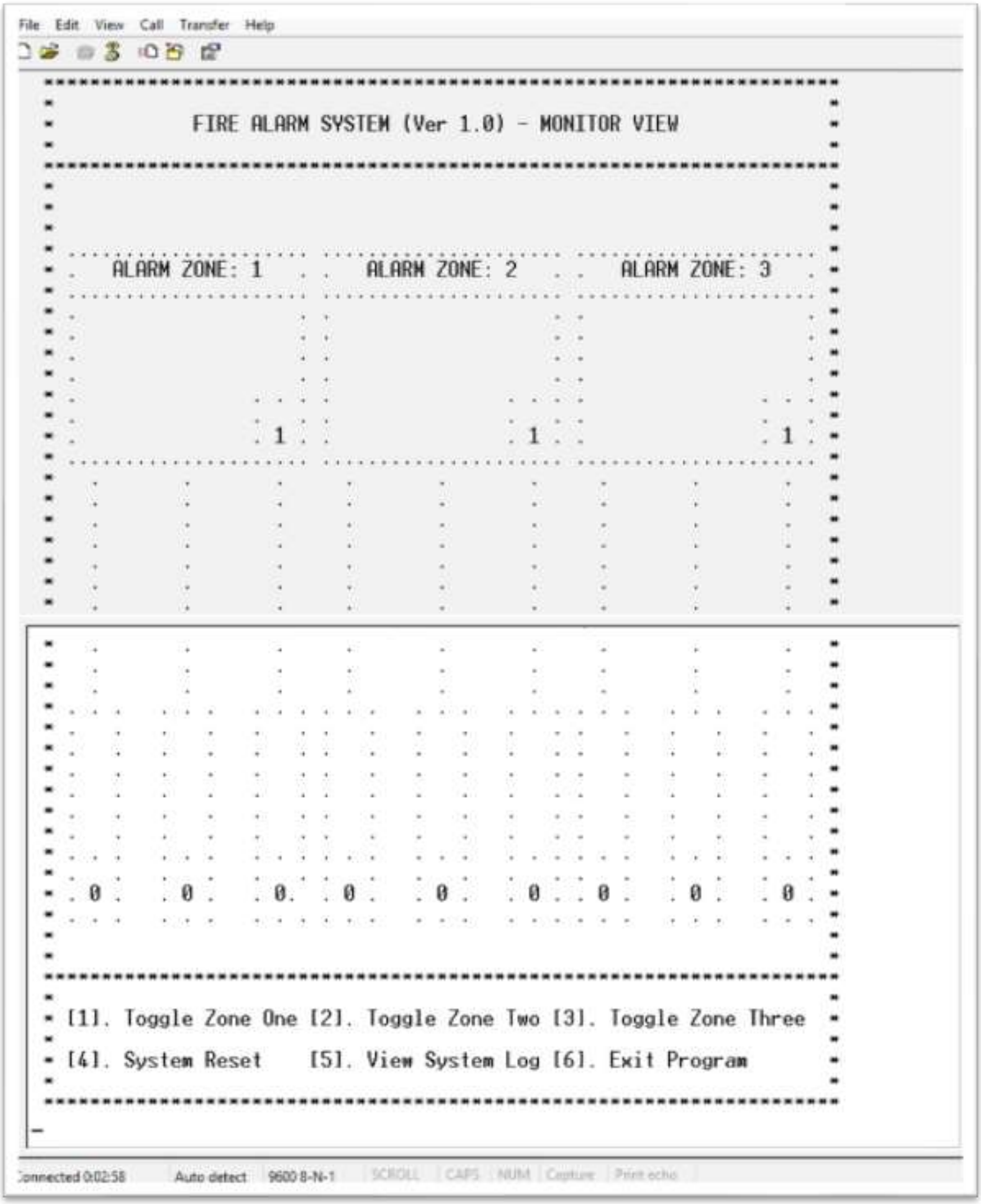


Figure 48 – Screen display with zone 1, 2 and 3 activated (post compiler)

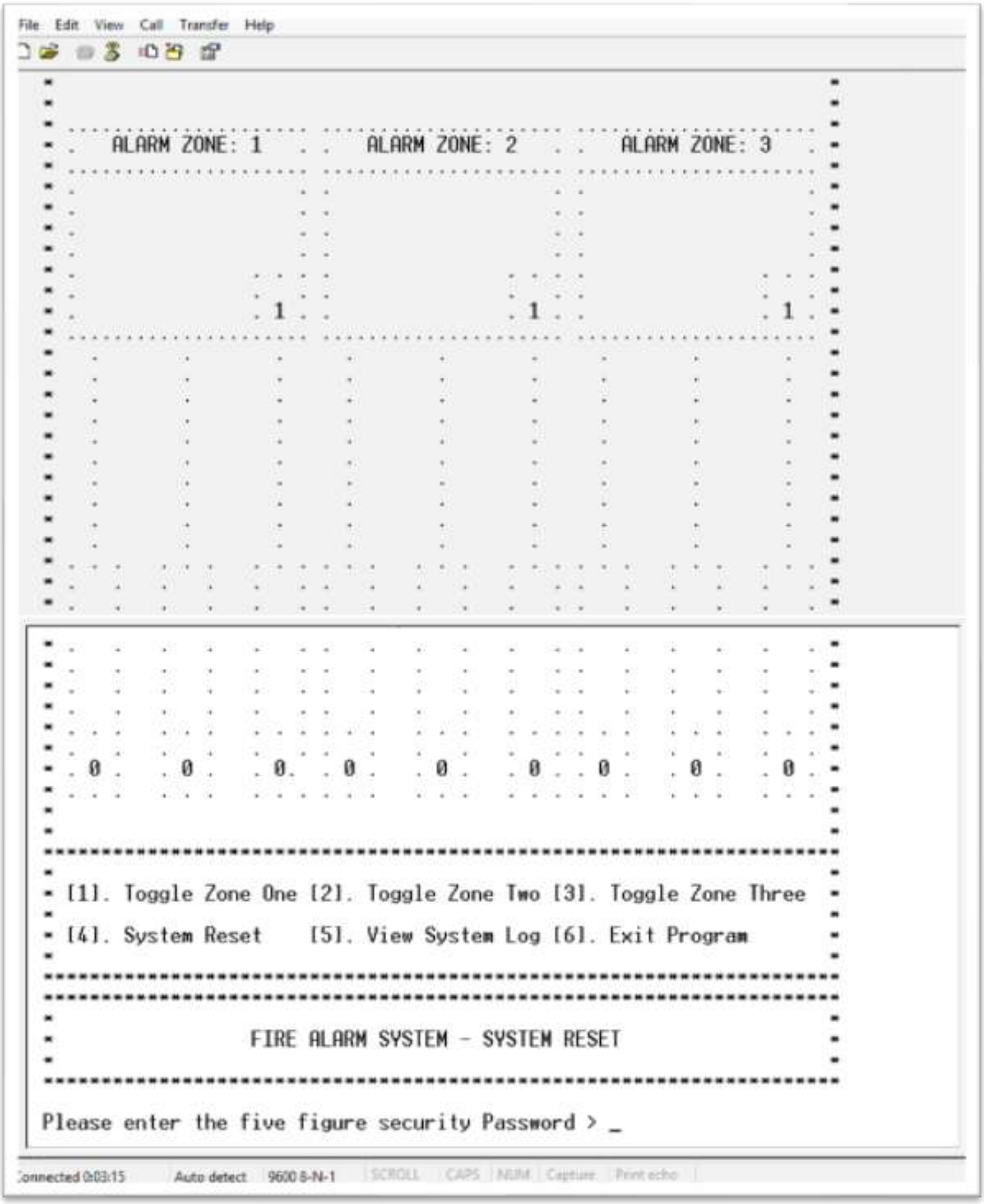


Figure 49 – System reset submenu display (post compiler)

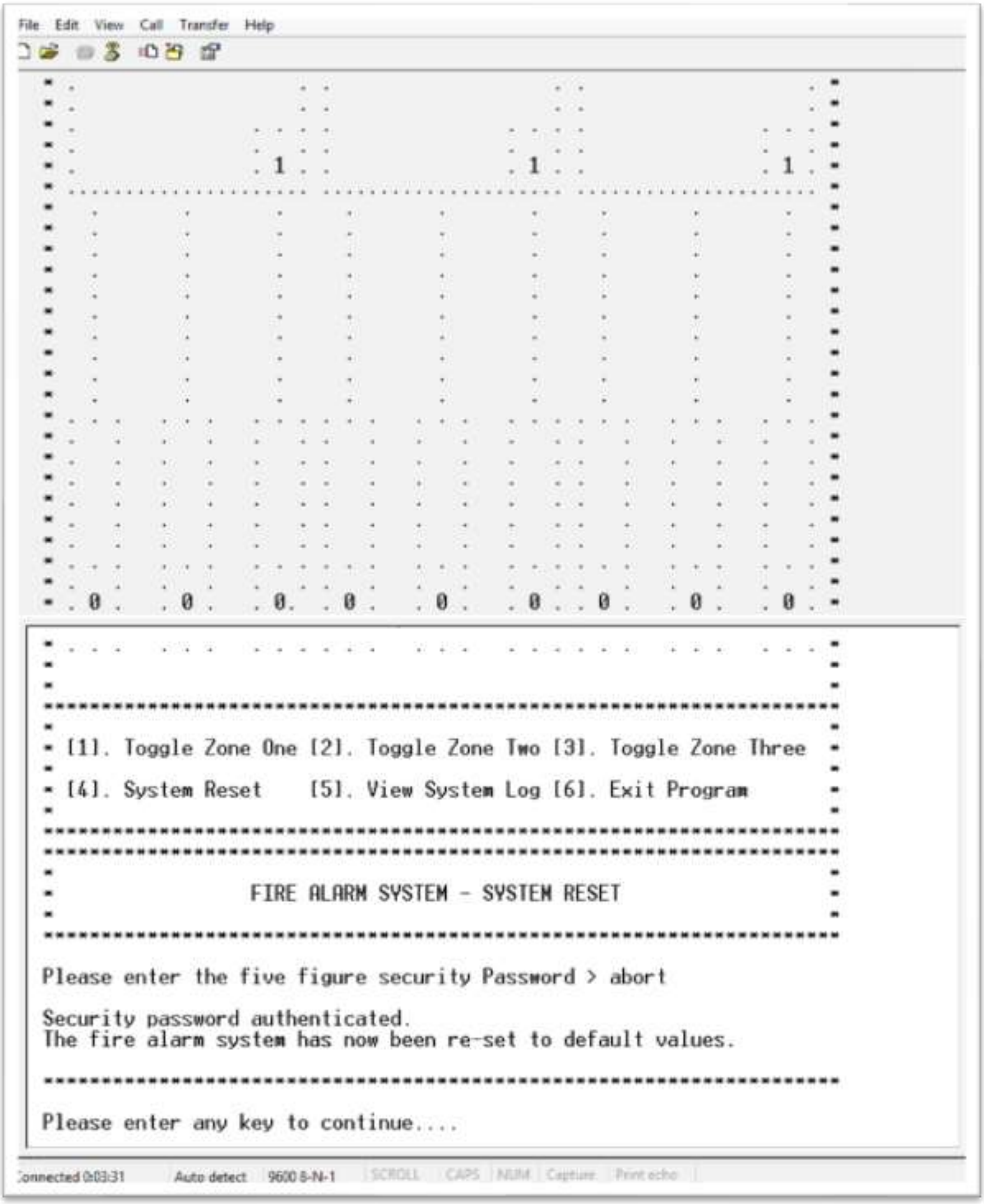


Figure 50 – Authenticated system reset (post compiler)

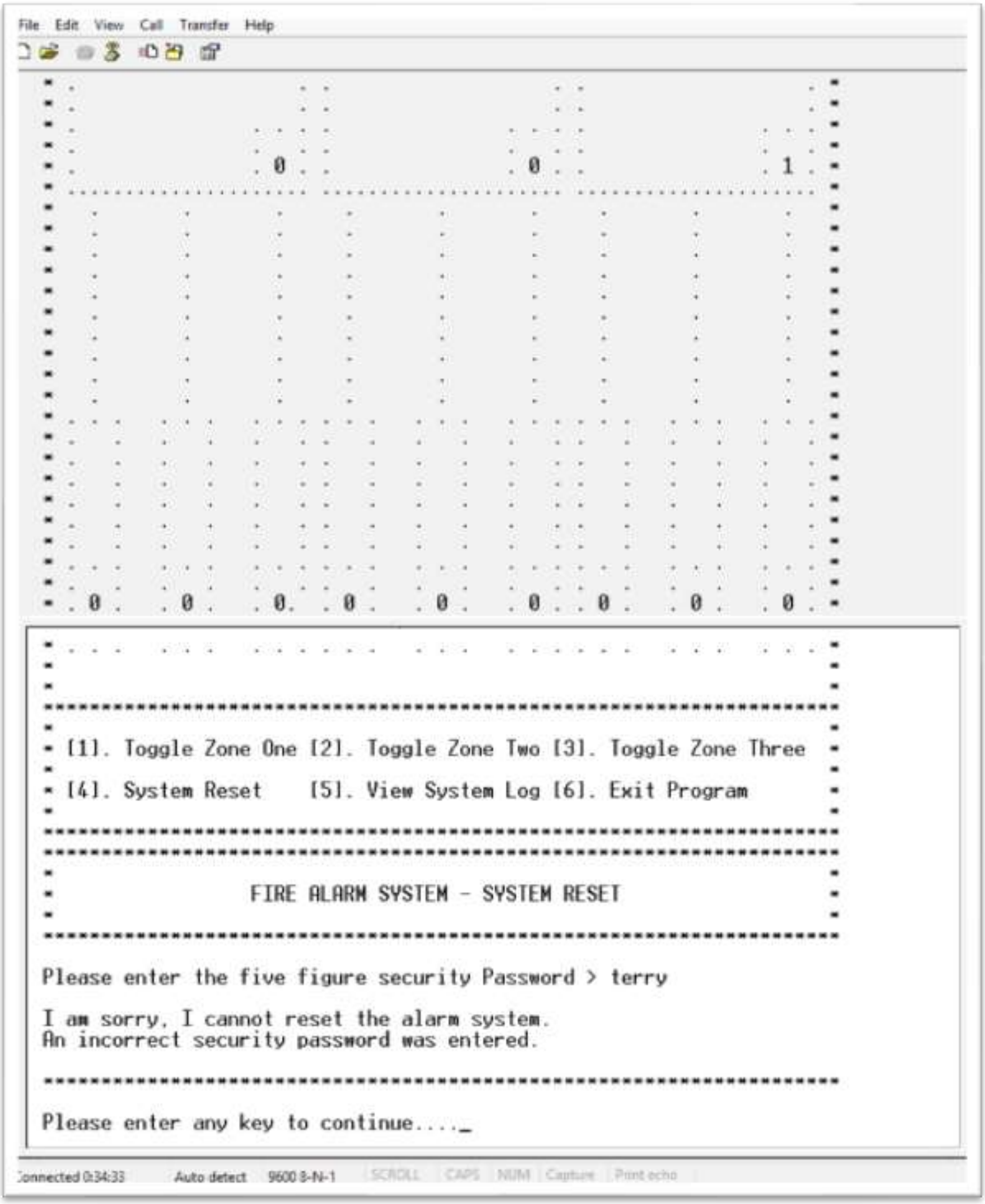


Figure 51 – Tamper system not reset (post compiler)

Screen Objects and External Actions

Shown below is the MC68HC11F1 micro-controller and the 9 external sensor triggers.

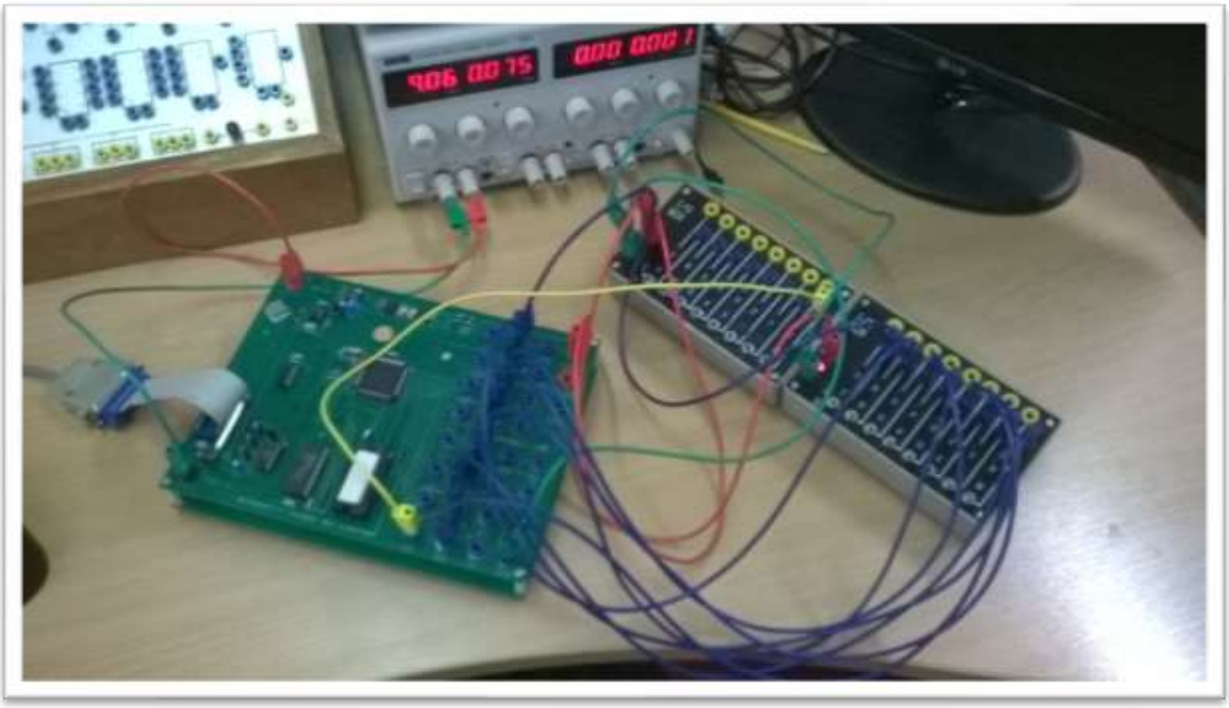


Figure 52 – The micro-controller and trigger switches

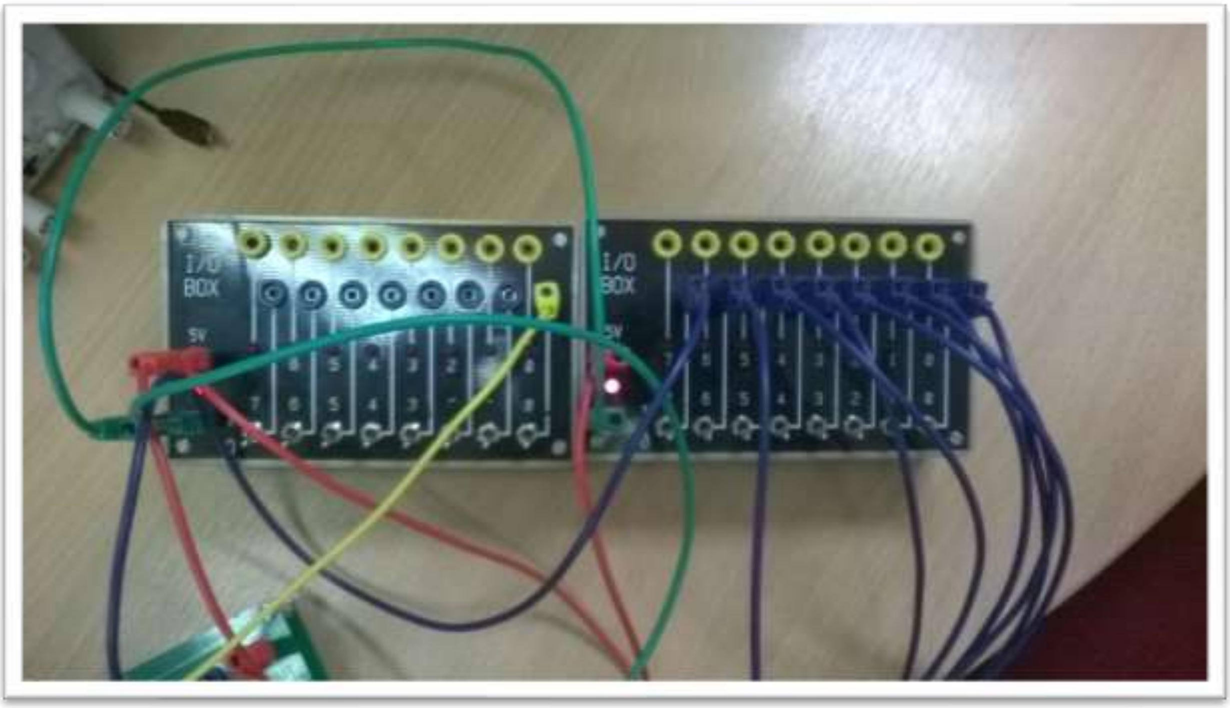


Figure 53 – 9 sensor trigger switches

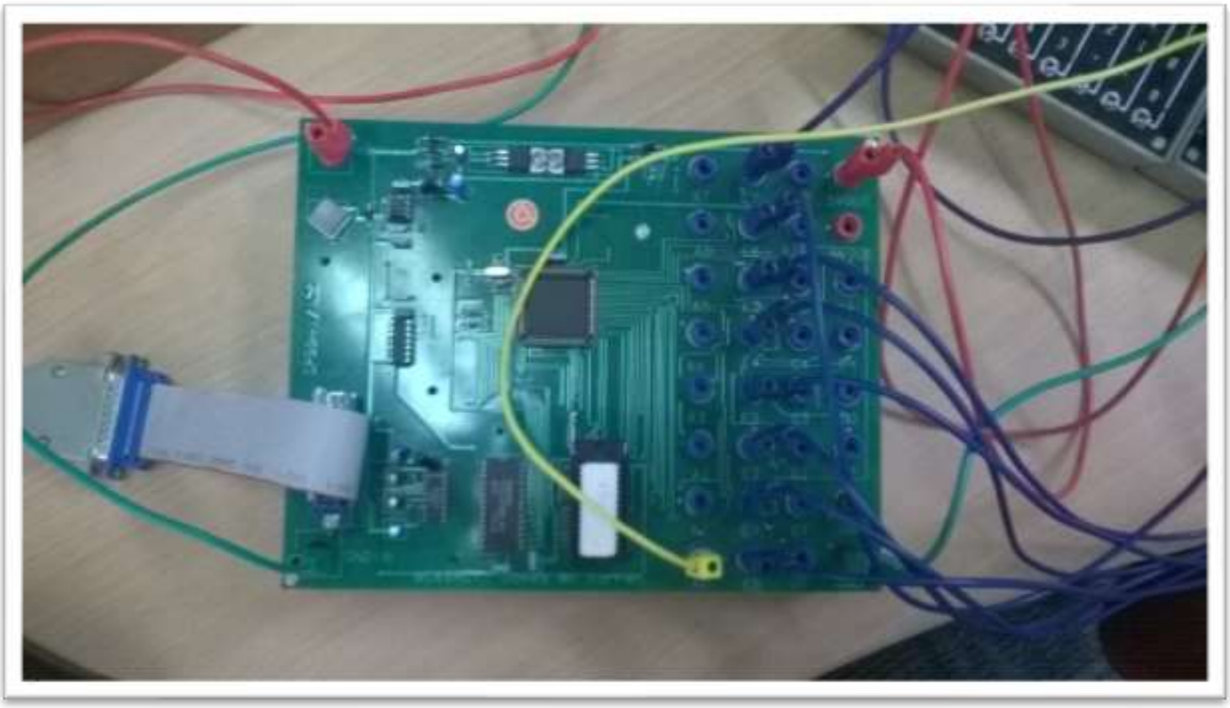


Figure 54 - MC68HC11F1 Micro-controller

The MC68HC11F1 micro-controller requires 9 volts to power on.

There is an inbuilt reset button (grey square) just below the red cable in the top left of the photograph above.

There are also two trigger switch boxes (see figure 53) both requiring 5 volts to operate – this is taken from the 5 volt output terminal of the MC68HC11F1 micro-controller (red cable top right of above photograph).

Trigger switches are physically connected to the micro-controller input terminals via blue connection leads using input ports E0 through to E7 and also A0*.

The micro-controller is now setup to output the 9 bits of trigger information required to interface with the program software via the grey 16 bit connection parallel cable.

*Note the single yellow cable connected to A0 representing the ninth bit sensor input to the micro-controller (sensor 3 within zone 3) from a separate switch box.

PROGRAM TESTING II

POST COMPILER TEST ONE		
Check zone 1 switches on/off correctly.	Check zone 2 switches on/off correctly.	Check zone 3 switches on/off correctly.
	Checked By: Terence Broadbent	
	Date Checked: 04/03/1015	

Table 25 – Post compiler test one

The above and following functionality tests were undertaken to confirm that the cross compiled program was correctly operating to read the 8 bit keyboard strokes and the 9 bit trigger switches [B3], [B6].

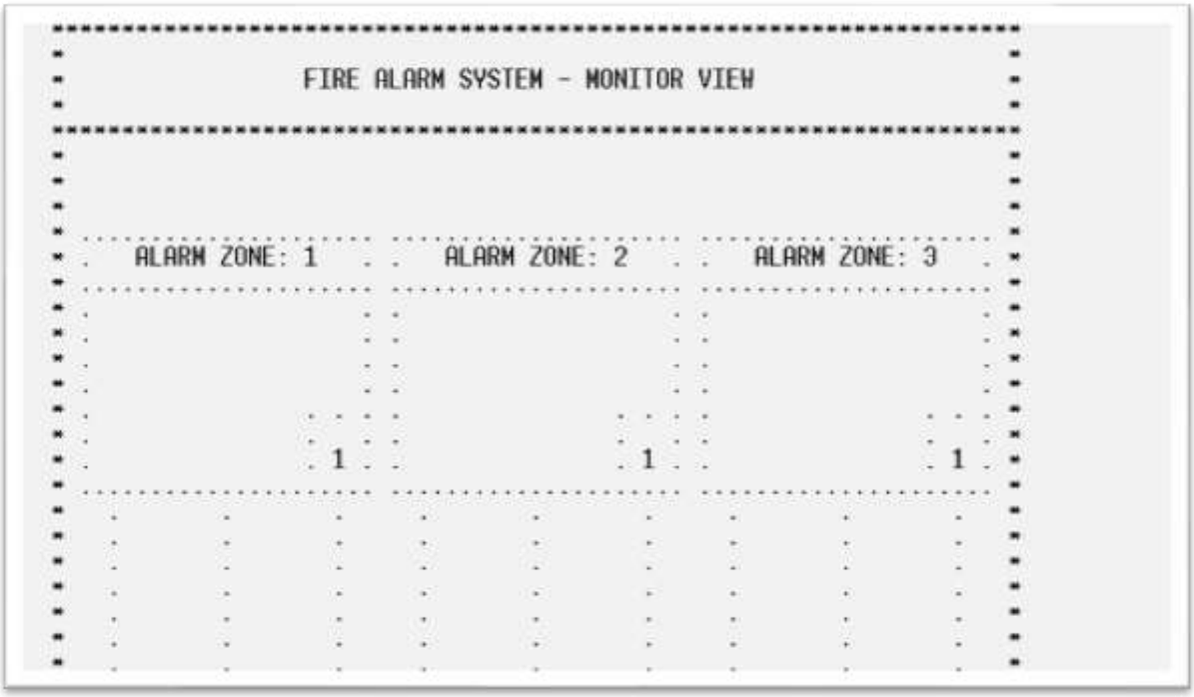


Figure 55 – Post compiler test one confirmation for zone 1, 2 and 3

POST COMPILER TEST TWO Zone 1-3		
Check sensor 1 switches on/off correctly.	Check sensor 2 switches on/off correctly.	Check sensor 3 switches on/off correctly.
Checked By: Terence Broadbent Date Checked: 04/03/2015		

Table 26 – Post Compiler test two

The same tests were also carried out on zone two and three successfully.

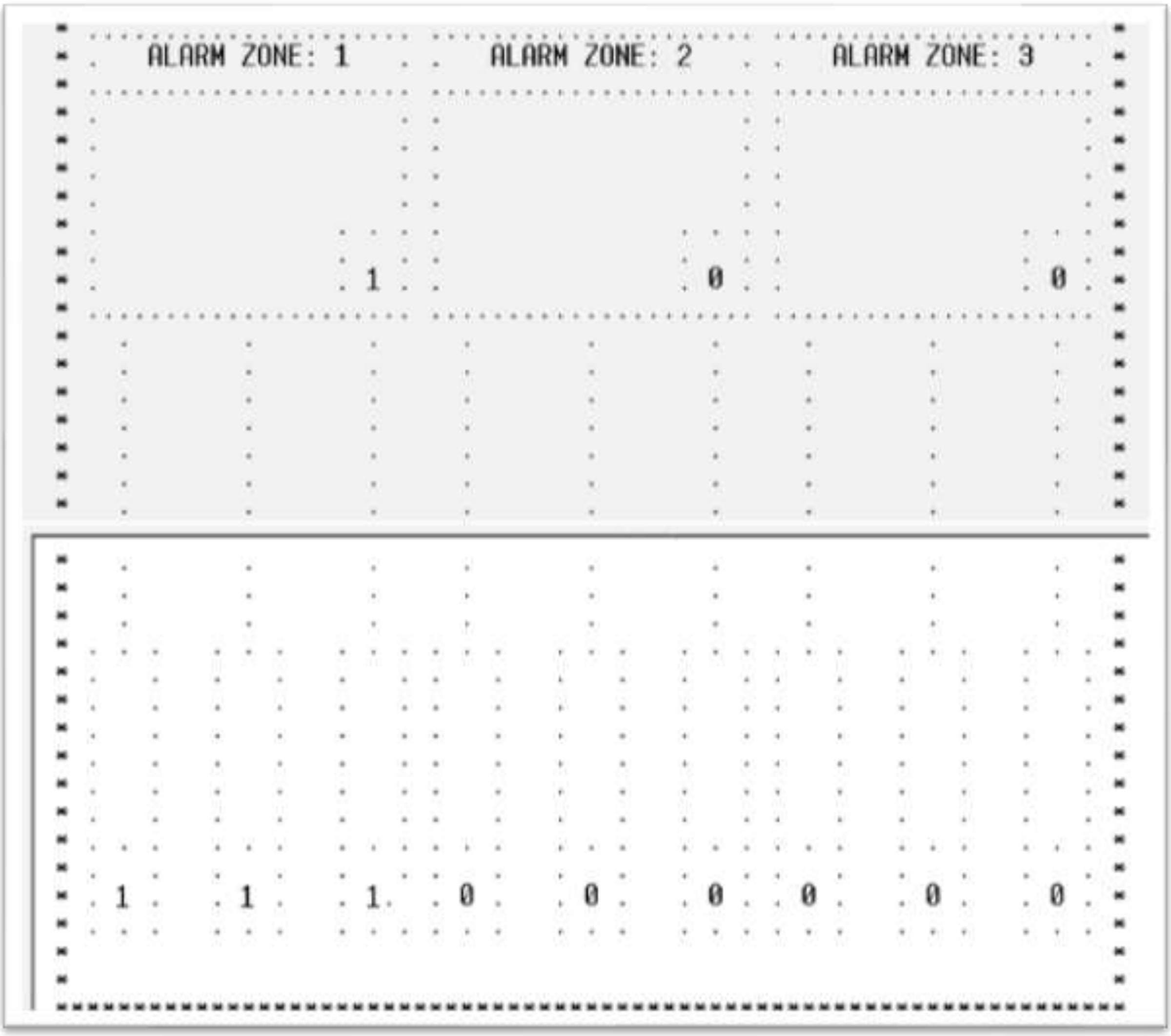


Figure 56 – Post compiler test two confirmation for zone 1 sensors

POST COMPILER TEST THREE Zone 1-3		
Check that the fire flag activates if zone 1 is on and any one of the sensors is activated.	Check that the fire flag activates if zone 2 is on and any one of the sensors is activated.	Check that the fire flag activates if zone 3 is on and any one of the sensors is activated.
Checked By: Terence Broadbent		
Date Checked: 04/03/2015		

Table 27 – Post compiler test three

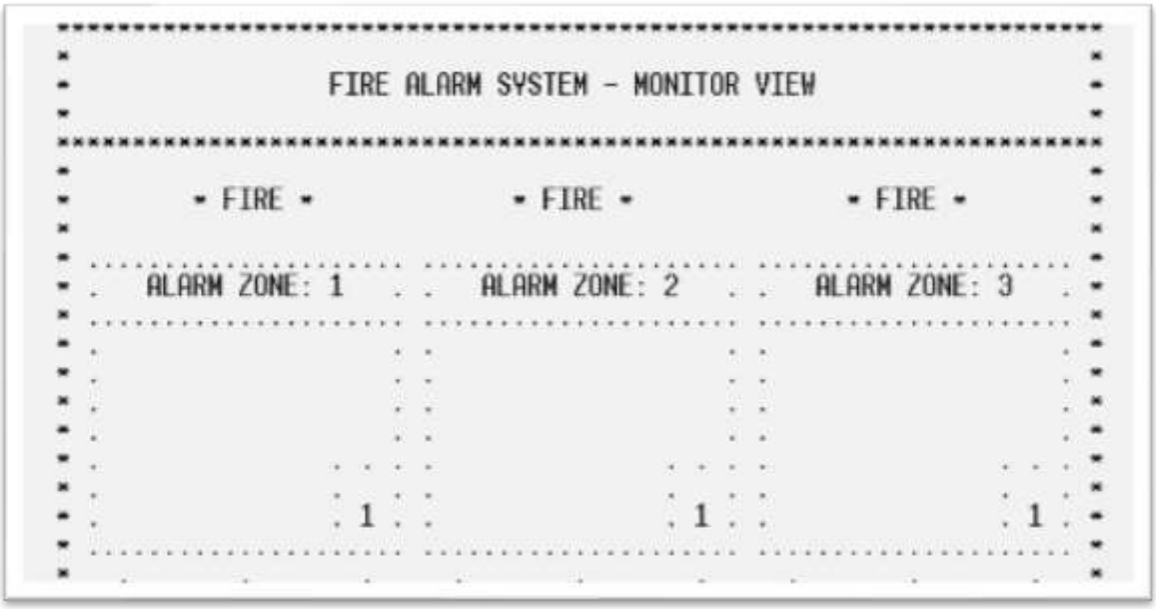


Figure 57 – Post compiler test three confirmation fire flags activate for zone 1, 2 and 3

POST COMPILER TEST FOUR Check that the system reset clears all the zone property values to the default '0'. Checked By: Terence Broadbent Date Checked: 04/03/2015

Table 28 – Post compiler test four

Figure 58 – Post compiler test four confirmation of system reset

This test also included testing the password requirements by inputting an incorrect password and pressing the return key before typing had finished.

Figure 59 – Post compiler test four confirmation of tamper

POST COMPILER TEST FIVE Check that the log book display shows the correct information and associated time stamp. Checked Terence Broadbent Date Checked: 04/03/2015

Figure 60 – Post compiler test five

Finally, checking the system log display – This also helped to check the correct system functionality was being called during the testing period.

Figure 61 – Post compiler test five confirmation of log book display

REQUIREMENTS MATRIX

The following table confirms that the design rationale have been met and complied with.

Component	Data Structures	Requirements	Page Numbers
Monitor nine alarm circuits via the MC68HC11F1.	ParrelPort()	A1	12,13
Enable or disable a zone when the zone fire flag is set.	Reset_Zone()	A2	12, 28
Set the zone fire flag until correct password entered.	Reset_Sys()	A3	13, 16, 43, 45, 46
Log 100 max set/alarm events in memory.	Build_Logg()	A4	16, 40, 46
Write C program to input & bit display data from port A.	ParrelPort()	B1	14
Implement a real time clock.	ClockTimer()	B2	16, 51
Implement a routine to key scan the serial port.	SerialPort()	B3	16, 50, 70
Extend the real time clock to include a simple data logger.	Write_Book()	B4	16, 40, 42, 46
Combine all the above elements to form a working commercial system	Appendix A.	B5	16
Extend the program to include 9 loopback circuits.	ParrelPort()	B6	16, 45, 49, 70
Compile the program to work in ram on the MC68HC11F1 micro-controller.	Part B	B7	54 - 59

Table 29 – Requirement matrix cross reference table

FUTURE ENHANCEMENT

The following are recommendations for bug fixes and future enhancements.

No.	Type	Comments
1	Bug Fix	There would seem to be a text display kink on the log book output screen. This is most likely linked to the ClockTimer() interrupt and screen print occurring at the same time. Possible solution linked to the tick counter. Requires investigating but as low priority task as this does not currently compromise program integrity.
2	Bug Fix	Once the 16 bit parallel port registers that an external sensor has been activated the screen will continually scroll (updating) while the trigger is turned on. Possibly by introducing a duplicate zone array of sensors for comparison within the ParrelPort() function a return could be made without the need for updating the screen.

Table 30 – Future bug fix

No.	Type	Comments
1	Enhancement	A function and menu option should be incorporated into program in order to change the default hard coded zone naming. Zone 1, 2 and 3 could be renamed by the user by adding a simple zone-name editing function within the program. A small amendment to the structure would be required – unsigned char Name [6] The program could then assign the typed name to the zones using the following command – Zone[0-2].Name[0-6] = User_Inputed_Name[0-6] Finally, another small update to the function Build_Disp() would be required to print the string value rather than the permanent coded names Zone 1-3. Medium priority – cosmetic only.
2	Enhancement	Add a printout option to the system log book display. Medium priority – cosmetic only.
3	Enhancement	A function and menu option should be incorporated into program in order to change the default hard coded master password. This task should be looked at as High priority as this is an important security point!!
4	Enhancement	A function and menu option should be incorporated into the program in order to change the default hard coded start time of the clock. This task should be looked at as High priority as this is an important functionality point.
5	Enhancement	Instead of exiting the program using exit(0) a micro-controller memory write reset would be a more preferred option as this would also remove the requirement for <stdlib.h> to be included within the program.

Table 31 – Future enhancements

REFERENCES

Cosmic Software. (2002). *C Cross Compiler User's Guide for Motorola MC68HC11 ver 4.1.* Cosmic Software.

David Hodgkiss and James Mc Carren. (2015). *COCS50592 Advanced Programming Languages for Computer Systems Assigngnent.* Stoke: Stafford University.

IEEE. (2003). *Software Design Document (SDD) Template.* Retrieved from www.cs.concordia.ca: http://www.cs.concordia.ca/~ormandj/comp354/2003/project/ieee-SDD.pdf

APPENDIX A

ANSI C Code

Detailed below is the ANSI code for release version 1.0 of the C program.

```
/* ******************************************************************************************************************************** */
/*                                              Assignment one - Staffordshire University 2015                                     */
/*                                                                                                                                  */
/*                  Microprocessor fire alarm system embedded within a 68hc11 Micro-controller - Release version 1.0               */
/*                                                                                                                                  */
/*                                                     by Terence Broadbent (B028035c)                                             */
/*                                                                                                                                  */
/* ******************************************************************************************************************************** */

/* ******************************************************************************************************************************** */
/* AUTHOR   : Terence Broadbent.                                                                                                    */
/* CONTRACT : Stafford University.                                                                                                  */
/* Version  : 1.0                                                                                                                   */
/* Details  : Ensure that no error messages appear plus system define name labels to make reading this source code a lot easier.  */
/* Modified : N/A                                                                                                                   */
/* ******************************************************************************************************************************** */

#define _CRT_SECURE_NO_WARNINGS
#define MemoryWipe 0
#define False 0
#define True 1
#define Off 0
#define On 1
```

```
/* ******************************************************************************************************************************** */
/* AUTHOR   : Terence Broadbent.                                                                                                    */
/* CONTRACT : Stafford University.                                                                                                  */
/* Version  : 1.0                                                                                                                   */
/* Details  : Preload any required extended C command header files.                                                                 */
/* Modified : N/A                                                                                                                   */
/* ******************************************************************************************************************************** */

#include <stdio.h>
#include <stdlib.h>

/* ******************************************************************************************************************************** */
/* AUTHOR   : Terence Broadbent.                                                                                                    */
/* CONTRACT : Stafford University.                                                                                                  */
/* Version  : 1.0                                                                                                                   */
/* Details  : Declare any global variables to be used by the system - The variables below are only used by the ClockTimer interrupt*/
/* Modified : N/A                                                                                                                   */
/* ******************************************************************************************************************************** */

unsigned int G_Hours;                                                        /* See ClockTimer() for explanation of global variables    */
unsigned int G_Mins;                                                         /*                         ""                              */
unsigned int G_Secs;                                                         /*                         ""                              */
unsigned int G_Ticks;                                                        /*                         ""                              */

unsigned char *G_PADR;                                                       /*                         ""                              */
unsigned char *G_PADDR;                                                      /*                         ""                              */
unsigned char *G_TFLG2;                                                      /*                         ""                              */
unsigned char *G_PACTL;                                                      /*                         ""                              */
unsigned char *G_TMSK2;                                                      /*                         ""                              */

/* ******************************************************************************************************************************** */
/* AUTHOR   : Terence Broadbent.                                                                                                    */
/* CONTRACT : Stafford University.                                                                                                  */
/* Version  : 1.0                                                                                                                   */
/* Details  : Define any structures to be called by the main program.                                                               */
/* Modified : N/A                                                                                                                   */
/* ******************************************************************************************************************************** */
```

```c
struct Alarm  {
              unsigned int Line [1];          /* Incrementing log entry counter for the individual zone           */
              unsigned int Node [5];          /* Nodes: 1 zone switch plus 3 sensors & 1 fire flag per zone       */
              unsigned int Book [100];        /* Stipulates the limit of any log book entries per zone            */
              unsigned int Hour [100];        /* Stipulates the limit of any time log entries per Zone            */
              unsigned int Mins [100];        /*                              ""                                  */
              unsigned int Secs [100];        /*                              ""                                  */
       };

/* ****************************************************************************************************** */
/* AUTHOR   : Terence Broadbent.                                                                          */
/* CONTRACT : Stafford University.                                                                        */
/* Version  : 1.0                                                                                         */
/* Details  : Define any functions to be called by the main program.                                      */
/* Modified : N/A                                                                                         */
/* ****************************************************************************************************** */


        void  ClockTimer (void);
unsigned int  Start_Logs (struct Alarm Zone []);
unsigned int  Reset_Syst (struct Alarm Zone []);
unsigned int  Reset_Zone (struct Alarm Zone []);
unsigned int  Reset_Sens (struct Alarm Zone []);
unsigned int  Reset_Stat (struct Alarm Zone []);
unsigned int  Togg_Zone1 (struct Alarm Zone []);
unsigned int  Togg_Zone2 (struct Alarm Zone []);
unsigned int  Togg_Zone3 (struct Alarm Zone []);
unsigned int  Build_Disp (struct Alarm Zone []);
unsigned int  Build_Logg (struct Alarm Zone []);
unsigned char SerialPort (struct Alarm Zone []);
unsigned char GrabK_Char (struct Alarm Zone []);
unsigned char ParralPort (struct Alarm Zone []);
unsigned int  Time_Stamp (struct Alarm Zone []);
unsigned int  Comp_Strng (const unsigned char String1[], const unsigned char String2[]);
unsigned int  Write_Book (struct Alarm Zone [], const unsigned int ZCode_0, const unsigned int ZCode_1, const unsigned int ZCode_2);

/* ****************************************************************************************************** */
```

```
/* AUTHOR    : Terence Broadbent.                                                                       */
/* CONTRACT : Stafford University.                                                                      */
/* Version  : 1.0                                                                                       */
/* Details  : Main program - Assignment 1 (Fire Alarm Embedded System) Version 1.0                      */
/* Libraries: <stdlib.h>                                                                                */
/* Modified : N/A                                                                                       */
/* ************************************************************************************************** */

void main()
{

/* ************************************************************************************************** */
/* Define and populate any local variables to be used by the main program                               */
/* ************************************************************************************************** */

       struct Alarm Zone[3]; /* Create 3 fire alarm zone circuits including switch, sensors and fire flag display plus log entries */
       unsigned char Key_Stroke;   /* Used to catch key strokes entered from the 8bit keyboard via the user                 */

       Key_Stroke = '0';                 /* Null the initial keystroke char value                                   */

       Start_Logs(Zone);                 /* Set up the time and book log entries with default values of '0'         */
       Reset_Zone(Zone);                 /* Set the zones to a default value of '0'                                 */
       Reset_Sens(Zone);                 /* Set the zones sensors to a defalt value of '0'                          */
       Reset_Stat(Zone);                 /* Set the zones fire status to a default value of '0'                     */
       Build_Disp(Zone);                 /* Show the fire alarm system to the user via the screen                   */

/* ************************************************************************************************** */
/* Human/Computer interface - Main menu system                                                          */
/* ************************************************************************************************** */

       for (;;)
              {
                     Key_Stroke = SerialPort(Zone);
                     switch (Key_Stroke)
                                   {
                                   case '1': { Togg_Zone1(Zone); break; }    /* Toggle zone one             */
                                   case '2': { Togg_Zone2(Zone); break; }    /* Toggle zone two             */
                                   case '3': { Togg_Zone3(Zone); break; }    /* Toggle zone three           */
```

```
                                        case '4': { Reset_Syst(Zone); break; }   /* Reset the alarm zones to default value '0'   */
                                        case '5': { Build_Logg(Zone); break; }   /* Show all log entries - upto maximum of 100   */
                                        case '6': { exit(0); }                   /* Exit the program                             */
                                        default : { break; }                     /* Ignore any incorrectly entered characters    */
                                        }
              }
}
/*End of main() */

/* ****************************************************************************************************************************** */
/* All the sub functions called by the main program are shown and detailed below:-                                                */
/* ****************************************************************************************************************************** */
```

```
/* ****************************************************************************************************************** */
/* AUTHOR    : Terence Broadbent.                                                                                     */
/* CONTRACT : Stafford University.                                                                                    */
/* Version  : 1.0                                                                                                     */
/* Function : ClockTimer()                                                                                            */
/* Details  : Interrupts the program every 32.768 ms to update the global clock variables then resets the real time interrupt flag.*/
/* Libraries: N/A
                                                                                               */
/* Variables: Global variables - G_Ticks, G_Secs, G_Mins, G_Mins and G_Hours.
                                                     */
/* Modified : Note Add @interrupt
                                                                                  */
/* ****************************************************************************************************************** */

@interrupt void ClockTimer(void)
{
       G_Ticks++;
       if (G_Ticks == 30) { G_Ticks = 0; G_Secs++; }   /* Executed every 30 seconds                                   */
       if (G_Secs  == 60) { G_Secs = 0; G_Mins++;  }   /* Executed every 60 seconds                                   */
       if (G_Mins  == 60) { G_Mins = 0; G_Hours++; }   /* Executed every 60 minutes                                   */
       if (G_Hours == 24) { G_Hours = 0; }             /* Executed every 24 hours                                     */
       *G_TFLG2 = 0x40;                                /* Reset RTI flag                                              */
}

/* ****************************************************************************************************************** */
/* AUTHOR    : Terence Broadbent.                                                                                     */
/* CONTRACT : Stafford University.                                                                                    */
/* Version  : 1.0                                                                                                     */
/* Function : Start_Logs()                                                                                            */
/* Details  : Sets the default line entry to '0' and clears the required memory locations for time and log book entries to '0'.    */
/*          : Allocates the memory locations for the global variables used by the clock timer and sets required controller settings*/
/*          : Only used once at the initiation of the program.                                                        */
/* Libraries: N/A                                                                                                     */
/* Variables: Zone[0-2].Line[0-99], G_PADR, *G_PADDR, *G_TMSK2, G_TFLG2, *G_PACTL, G_Ticks, G_Secs, G_Mins, G_Mins and G_Hours.    */
/* Modified : N/A                                                                                                     */
/* ****************************************************************************************************************** */

unsigned int Start_Logs(struct Alarm Zone[])
```

```
{
        auto unsigned int Loop;

        Zone[0].Line[0] = 0;                                  /* Start the log book (and time) line entries off at '0'      */
        Zone[1].Line[0] = 0;                                  /*                                 ""                         */
        Zone[2].Line[0] = 0;                                  /*                                 ""                         */

        for (Loop = 0; Loop < 100; Loop++)
        {
                Zone[0].Book[Loop] = MemoryWipe;              /* Clear the log book memory area for new data                */
                Zone[1].Book[Loop] = MemoryWipe;              /*                                 ""                         */
                Zone[2].Book[Loop] = MemoryWipe;              /*                                 ""                         */
                Zone[0].Hour[Loop] = MemoryWipe;              /* Clear the time log memory area for new data                */
                Zone[1].Mins[Loop] = MemoryWipe;              /*                                 ""                         */
                Zone[2].Secs[Loop] = MemoryWipe;              /*                                 ""                         */
        }

        G_PADR =  (unsigned char *)0x0000;                    /* Port A data register                                       */
        G_PADDR = (unsigned char *)0x0001;                    /* Port A data direction register                             */
        G_TMSK2 = (unsigned char *)0x24;                      /* Timer interrupt mask 2 - RTII flag                         */
        G_TFLG2 = (unsigned char *)0x25;                      /* Timer interrupt flag 2 - RTIF flag                         */
        G_PACTL = (unsigned char *)0x26;                      /* Pulse accumulator control                                  */

        *G_PADDR = 0xfe;                                      /* Sets Port A to output                                      */
        *G_PACTL = 0x03;                                      /* sets the real time interrupt period to 32.768 ms           */
        *G_TMSK2 = 0x40;                                      /* Enable RTI interrupt                                       */

        G_Hours = 0;                                          /* Start clock timer at 00:00:00                              */
        G_Mins  = 0;                                          /*                                 ""                         */
        G_Secs  = 0;                                          /*                                 ""                         */
        G_Ticks = 0;                                          /*                                 ""                         */

        Write_Book(Zone, 1, 1, 1);                            /* Update the log book with above function actions            */
        return (0);
}
```

```c
/* ************************************************************************************************************************************ */
/* AUTHOR    : Terence Broadbent.                                                                                                       */
/* CONTRACT : Stafford University.                                                                                                      */
/* Version  : 1.0                                                                                                                       */
/* Function : Reset_Syst()                                                                                                              */
/* Details  : Resets the zone and sensor data held within the alarm zones to a default value of '0' and clears fire status display.*/
/*          : Requires the correct user password to be entered to authenticate the action otherwise it is treated as a tamper event*/
/* Libraries: <stdio.h>                                                                                                                 */
/* Variables: Zone[0-2], *SCDR, *SCSR, Pass_Master[], PaswordCopy[], Test_String, Test_Tamper.                                          */
/* Modified : N/A                                                                                                                       */
/* ************************************************************************************************************************************ */

unsigned int Reset_Syst(struct Alarm Zone[])
{
       auto unsigned char *SCDR;                        /* Serial communications data register                                    */
       auto unsigned char *SCSR;                        /* SCI status register                                                    */
       auto unsigned char Pass_Master[5];               /* Master hard coded password                                             */
       auto unsigned char PaswordCopy[5];               /* User entered password                                                  */
       auto unsigned int  Loop;                         /*                                       ""                               */
       auto unsigned int  Test_String;                  /* Return value for testing matching Password strings                     */
       auto unsigned int  Test_Tamper;                  /* Authentication/Tamper flag                                             */

       SCSR = (unsigned char *)0x2e;                    /* Clears the RDRF flag                                                   */
       SCDR = (unsigned char *)0x2f;                    /* Retrieves last byte data buffer via the keyboard.                      */
       Test_String = False;                             /* Default that password strings do not match                             */
       Test_Tamper = Off;                               /* Default to tamper off                                                  */
       Loop = 0;

       printf(" **********************************************************************\n");/* Display sub-menu system on the screen */
       printf(" *                                                                    *\n");
       printf(" *                  FIRE ALARM SYSTEM - SYSTEM RESET                  *\n");
       printf(" *                                                                    *\n");
       printf(" **********************************************************************\n");
       printf("\n Please enter the five figure security Password > ");

/* ************************************************************************************************************************************ */
/* Grab the first 5 key characters entered by the user and compare it with the master password then process and display the outcome*/
/* ************************************************************************************************************************************ */
```

```
	for (Loop = 0; Loop < 5; Loop++)
	{
		while (((*SCSR) & 0x20) == 0x0) { ;; }			/* Wait until a 8 bit key stroke is pressed by the user	*/
		PaswordCopy[Loop] = (char) *SCDR;			/* Build the user entered password for authentication	*/
		if (PaswordCopy[Loop] == '\r') { ;; }			/* Check if the user has entered a return value		*/

			else
				{
					printf("%c", PaswordCopy[Loop]);	/* Display typed character to the user for confirmation	*/
				}
		if (PaswordCopy[Loop] == '\r') { Loop = 5; }		/* Break from loop once the enter key has been pressed	*/
	}

	Pass_Master[0] = 'a';						/* Build master password [Complexity can be changed !!!]	*/
	Pass_Master[1] = 'b';
	Pass_Master[2] = 'o';
	Pass_Master[3] = 'r';
	Pass_Master[4] = 't';

	Test_String = Comp_Strng(PaswordCopy, Pass_Master);		/* Test that the user password match's the master password	*/

	if (Test_String == True)	/* All good - Authenticated */
					{
						printf("\n\n Security password authenticated.\n The fire alarm system has now been 
re-set to default values.");
						Reset_Zone(Zone);
						Reset_Sens(Zone);
						Reset_Stat(Zone);
					}
				else		/* No good - Tampered */
					{
						printf("\n\n I am sorry, I cannot reset the alarm system.\n An incorrect security 
password was entered.");
						Test_Tamper = On;
					}

	printf("\n\n **********************************************************************\n");
```

```
        printf("\n Please enter any key to continue....");
        while (((*SCSR) & 0x20) == 0x0) { ;; }                  /* Wait until a 8 bit key stroke is pressed by the user       */
        Build_Disp(Zone);                                        /* Display a new screen build                                 */

        if (Test_Tamper == Off) { Write_Book(Zone, 3, 3, 3); }   /* Zone one, two and three have been reset                    */
        if (Test_Tamper == On)  { Write_Book(Zone, 4, 4, 4); }   /* Zone one, two and three have not been reset                */
        return (0);
}

/* ***************************************************************************************************************** */
/* AUTHOR   : Terence Broadbent.                                                                                      */
/* CONTRACT : Stafford University.                                                                                    */
/* Version  : 1.0                                                                                                     */
/* Function : Reset_Zone()                                                                                            */
/* Details  : System reset all the zone on/off settings to a default value '0' - ie Off.                             */
/* Libraries: N/A                                                                                                     */
/* Variables: Zone[0-2].Node[0].                                                                                      */
/* Modified : N/A                                                                                                     */
/* ***************************************************************************************************************** */

unsigned int Reset_Zone(struct Alarm Zone[])
{
        Zone[0].Node[0] = Off;
        Zone[1].Node[0] = Off;
        Zone[2].Node[0] = Off;
        Write_Book(Zone, 5, 5, 5);
        return (0);
}

/* ***************************************************************************************************************** */
/* AUTHOR   : Terence Broadbent.                                                                                      */
/* CONTRACT : Stafford University.                                                                                    */
/* Version  : 1.0                                                                                                     */
/* Function : Reset_Sens()                                                                                            */
/* Details  : System reset all the zone sensors on/off settings to a default value '0' - ie Off.                     */
/* Libraries: N/A                                                                                                     */
/* Variables: Zone[0-2].Node[1].                                                                                      */
/* Modified : N/A                                                                                                     */
```

```
/* ************************************************************************************************ */

unsigned int Reset_Sens(struct Alarm Zone[])
{
       Zone[0].Node[1] = Off;
       Zone[0].Node[2] = Off;
       Zone[0].Node[3] = Off;
       Zone[1].Node[1] = Off;
       Zone[1].Node[2] = Off;
       Zone[1].Node[3] = Off;
       Zone[2].Node[1] = Off;
       Zone[2].Node[2] = Off;
       Zone[2].Node[3] = Off;
       Write_Book(Zone, 6, 6, 6);
       return (0);
}

/* ************************************************************************************************ */
/* AUTHOR    : Terence Broadbent.                                                                   */
/* CONTRACT  : Stafford University.                                                                 */
/* Version   : 1.0                                                                                  */
/* Function  : Reset_Stat()                                                                         */
/* Details   : System reset all the zone fire status on/off settings to a default value '0' - ie Off.  */
/* Libraries: N/A                                                                                   */
/* Variables: Zone[0-2].Node[1].                                                                    */
/* Modified  : N/A                                                                                  */
/* ************************************************************************************************ */

unsigned int Reset_Stat(struct Alarm Zone[])
{
       Zone[0].Node[4] = Off;
       Zone[1].Node[4] = Off;
       Zone[2].Node[4] = Off;
       Write_Book(Zone, 7, 7, 7);
       return (0);
}
```

```
/* ***************************************************************************************************************************** */
/* AUTHOR   : Terence Broadbent.                                                                                                 */
/* CONTRACT : Stafford University.                                                                                               */
/* Version  : 1.0                                                                                                                */
/* Function : Togg_Zone1()                                                                                                       */
/* Details  : Toggle the zone on or off depending on the zones current setting using simple boolean logic.                      */
/* Libraries: N/A                                                                                                                */
/* Variables: Zone[0].Node[0].                                                                                                   */
/* Modified : N/A                                                                                                                */
/* ***************************************************************************************************************************** */

unsigned int Togg_Zone1(struct Alarm Zone[])
{
	Zone[0].Node[0] = !Zone[0].Node[0];
	Build_Disp(Zone);
	Write_Book(Zone, 2, 0, 0);
	return (0);
}

/* ***************************************************************************************************************************** */
/* AUTHOR   : Terence Broadbent.                                                                                                 */
/* CONTRACT : Stafford University.                                                                                               */
/* Version  : 1.0                                                                                                                */
/* Function : Togg_Zone2()                                                                                                       */
/* Details  : Toggle the zone on or off depending on the zones current setting using simple boolean logic.                      */
/* Libraries: N/A                                                                                                                */
/* Variables: Zone[1].Node[0].                                                                                                   */
/* Modified : N/A                                                                                                                */
/* ***************************************************************************************************************************** */

unsigned int Togg_Zone2(struct Alarm Zone[])
{
	Zone[1].Node[0] = !Zone[1].Node[0];
	Build_Disp(Zone);
	Write_Book(Zone, 0, 2, 0);
	return (0);
}
```

```
/* ************************************************************************************************************************************ */
/* AUTHOR   : Terence Broadbent.                                                                                                    */
/* CONTRACT : Stafford University.                                                                                                  */
/* Version  : 1.0                                                                                                                   */
/* Function : Togg_Zone3()                                                                                                          */
/* Details  : Toggle the zone on or off depending on the zones current setting using simple boolean logic.                          */
/* Libraries: N/A                                                                                                                   */
/* Variables: Zone[2].Node[0].                                                                                                      */
/* Modified : N/A                                                                                                                   */
/* ************************************************************************************************************************************ */

unsigned int Togg_Zone3(struct Alarm Zone[])
{
	Zone[2].Node[0] = !Zone[2].Node[0];
	Build_Disp(Zone);
	Write_Book(Zone, 0, 0, 2);
	return(0);
}

/* ************************************************************************************************************************************ */
/* AUTHOR   : Terence Broadbent.                                                                                                    */
/* CONTRACT : Stafford University.                                                                                                  */
/* Version  : 1.0                                                                                                                   */
/* Function : Build_Disp()                                                                                                          */
/* Details  : Build a display screen to the user of the current status of all the alarm zones, sensors and status.                  */
/*          : Build a display of the Human/Computer interface - menu system.                                                        */
/* Libraries: <stdio.h>                                                                                                             */
/* Variables: Zone[0-2].Node[0-4].                                                                                                  */
/* Modified : N/A                                                                                                                   */
/* ************************************************************************************************************************************ */

unsigned int Build_Disp(struct Alarm Zone[])
{
	printf("\n");
	printf(" ********************************************************************\n");
	printf(" *                                                                  *\n");
	printf(" *             FIRE ALARM SYSTEM (Ver 1.0) - MONITOR VIEW           *\n");
	printf(" *                                                                  *\n");
```

```
printf(" *********************************************************************\n");
printf(" *                                                                   *\n");
printf(" *        ");
if (Zone[0].Node[4] == Off) { printf("         "); }                 /* Display no status                                   */
if (Zone[0].Node[4] == On) { printf("- FIRE - "); }                  /* Display fire status                                 */
printf("            ");
if (Zone[1].Node[4] == Off) { printf("         "); }                 /* Display no status                                   */
if (Zone[1].Node[4] == On) { printf("- FIRE - "); }                  /* Display fire status                                 */
printf("              ");
if (Zone[2].Node[4] == Off) { printf("         "); }                 /* Display no status                                   */
if (Zone[2].Node[4] == On) { printf("- FIRE - "); }                  /* Display fire status                                 */
printf("       *\n");
printf(" *                                                                   *\n");
printf(" * ..................... ..................... ..................... *\n");
printf(" * .   ALARM ZONE: 1   . .   ALARM ZONE: 2   . .   ALARM ZONE: 3   . *\n");
printf(" * ..................... ..................... ..................... *\n");
printf(" * .                   . .                   . .                   . *\n");
printf(" * .                   . .                   . .                   . *\n");
printf(" * .                   . .                   . .                   . *\n");
printf(" * .                   . .                   . .                   . *\n");
printf(" * .               . . . .               . . . .               . . . *\n");
printf(" * .               .   . .               .   . .               .   . *\n");
printf(" * .               . ");
printf("%i", Zone[0].Node[0]);
       /* Display zone 1 status                           */
printf(" . .               . ");
printf("%i", Zone[1].Node[0]);
       /* Display zone 2 status                           */
printf(" . .               . ");
printf("%i", Zone[2].Node[0]);
       /* Display zone 3 status                           */
printf(" . *\n");
printf(" * ..................... ..................... ..................... *\n");
printf(" *   .       .       .     .       .       .     .       .       .   *\n");
printf(" *   .       .       .     .       .       .     .       .       .   *\n");
printf(" *   .       .       .     .       .       .     .       .       .   *\n");
printf(" *   .       .       .     .       .       .     .       .       .   *\n");
printf(" *   .       .       .     .       .       .     .       .       .   *\n");
```

```
printf(" *   .       .       .     .       .       .     .       .       .   *\n");
printf(" *   .       .       .     .       .       .     .       .       .   *\n");
printf(" *   .       .       .     .       .       .     .       .       .   *\n");
printf(" *   .       .       .     .       .       .     .       .       .   *\n");
printf(" *   .       .       .     .       .       .     .       .       .   *\n");
printf(" * . . .   . . .   . . . . . .   . . .   . . . . . .   . . .   . . . *\n");
printf(" * .   .   .   .   .   . .   .   .   .   .   . .   .   .   .   .   . *\n");
printf(" * .   .   .   .   .   . .   .   .   .   .   . .   .   .   .   .   . *\n");
printf(" * .   .   .   .   .   . .   .   .   .   .   . .   .   .   .   .   . *\n");
printf(" * .   .   .   .   .   . .   .   .   .   .   . .   .   .   .   .   . *\n");
printf(" * .   .   .   .   .   . .   .   .   .   .   . .   .   .   .   .   . *\n");
printf(" * .   .   .   .   .   . .   .   .   .   .   . .   .   .   .   .   . *\n");
printf(" * . . .   . . .   . . . . . .   . . .   . . . . . .   . . .   . . . *\n");
printf(" * .   .   .   .   .   . .   .   .   .   .   . .   .   .   .   .   . *\n");
printf(" * . ");
printf("%i", Zone[0].Node[1]);
       /* Display zone 1 sensor 1 status           */
printf(" .   . ");
printf("%i", Zone[0].Node[2]);
       /* Display zone 1 sensor 2 status           */
printf(" .   . ");
printf("%i", Zone[0].Node[3]);
       /* Display zone 1 sensor 3 status           */
printf(".  . ");
printf("%i", Zone[1].Node[1]);
       /* Display zone 2 sensor 1 status           */
printf(" .   . ");
printf("%i", Zone[1].Node[2]);
       /* Display zone 2 sensor 2 status           */
printf(" .   . ");
printf("%i", Zone[1].Node[3]);
       /* Display zone 2 sensor 3 status           */
printf(" . . ");
printf("%i", Zone[2].Node[1]);
       /* Display zone 3 sensor 1 status           */
printf(" .   . ");
printf("%i", Zone[2].Node[2]);
       /* Display zone 3 sensor 2 status           */
```

```
        printf(" .    . ");
        printf("%i", Zone[2].Node[3]);
               /* Display zone 3 sensor 3 status           */
        printf(" . *\n");
        printf(" * . . .   . . .   . . . . . .   . . .   . . . . . .   . . .   . . . *\n");
        printf(" *                                                                    *\n");
        printf(" *                                                                    *\n");
        printf(" *********************************************************************\n");
        printf(" *                                                                    *\n");
        printf(" * [1]. Toggle Zone One [2]. Toggle Zone Two [3]. Toggle Zone Three  *\n");  /* Human/Computer interface menu system*/
        printf(" *                                                                    *\n");
        printf(" * [4]. System Reset    [5]. View System Log [6]. Exit Program       *\n");
        printf(" *                                                                    *\n");
        printf(" *********************************************************************\n");
        return (0);
}

/* ***************************************************************************************************************************** */
/* AUTHOR   : Terence Broadbent.                                                                                                   */
/* CONTRACT : Stafford University.                                                                                                 */
/* Version  : 1.0                                                                                                                  */
/* Function : Build_Logg()                                                                                                         */
/* Details  : Display the last 100 user log entries per zone (if populated with data) to the screen for the user to read.          */
/*          : Default value '0' has no display output - i.e. no activity within a zone.                                            */
/* Libraries: <stdio.h>                                                                                                            */
/* Variables: Zone[0-2].Book[0-99], Zone[0-2].Hour[0-99], Zone[0-2].Mins[0-99], Zone[0-2].Secs[0-99], *SCDR, *SCDR.                */
/* Modified : N/A                                                                                                                  */
/* ***************************************************************************************************************************** */

unsigned int Build_Logg(struct Alarm Zone[])
{
        auto unsigned char *SCDR;                       /* Serial communications data register                                    */
        auto unsigned char *SCSR;                       /* SCI status register                                                    */
        auto unsigned int  Loop;

        SCSR = (unsigned char *)0x2e;                   /* Clears the RDRF flag                                                   */
        SCDR = (unsigned char *)0x2f;                   /* Retrieves last byte data buffer via the keyboard.                      */
```

```
printf(" ********************************************************************\n"); /* Display sub-menu system on the screen*/
printf(" *                                                                  *\n");
printf(" *                 FIRE ALARM SYSTEM - ENTRY LOGS                   *\n");
printf(" *                                                                  *\n");
printf(" ********************************************************************\n");

for (Loop = 0; Loop < 100; Loop++)
{
                                          /* ZCode */
if (Zone[0].Book[Loop] == 1)  { printf(" %02d:%02d:%02d Zone 1 time and log book created.\n",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[1].Book[Loop] == 1)  { printf(" %02d:%02d:%02d Zone 2 time and log book created.\n",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[2].Book[Loop] == 1)  { printf(" %02d:%02d:%02d Zone 3 time and log book created.\n",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[0].Book[Loop] == 2)  { printf(" %02d:%02d:%02d Zone 1 toggled.\n",
Zone[0].Hour[Loop],  Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[1].Book[Loop] == 2)  { printf(" %02d:%02d:%02d Zone 2 toggled.\n",
Zone[0].Hour[Loop],  Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[2].Book[Loop] == 2)  { printf(" %02d:%02d:%02d Zone 3 toggled.\n",
Zone[0].Hour[Loop],  Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[0].Book[Loop] == 3)  { printf(" %02d:%02d:%02d Zone 1 successfully reset by the user.\n",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[1].Book[Loop] == 3)  { printf(" %02d:%02d:%02d Zone 2 successfully reset by the user.\n",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[2].Book[Loop] == 3)  { printf(" %02d:%02d:%02d Zone 3 successfully reset by the user.\n",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[0].Book[Loop] == 4)  { printf(" %02d:%02d:%02d An attempt was made to reset zone 1.\n",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[1].Book[Loop] == 4)  { printf(" %02d:%02d:%02d An attempt was made to reset zone 2.\n",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[2].Book[Loop] == 4)  { printf(" %02d:%02d:%02d An attempt was made to reset zone 3.\n",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[0].Book[Loop] == 5)  { printf(" %02d:%02d:%02d Zone 1 reset.\n",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[1].Book[Loop] == 5)  { printf(" %02d:%02d:%02d Zone 2 reset.\n",
Zone[1].Hour[Loop], Zone[1].Mins[Loop], Zone[1].Secs[Loop]); }
```

```
if (Zone[2].Book[Loop] == 5)  { printf(" %02d:%02d:%02d Zone 3 reset.\n",
Zone[2].Hour[Loop], Zone[2].Mins[Loop], Zone[2].Secs[Loop]); }
if (Zone[0].Book[Loop] == 6)  { printf(" %02d:%02d:%02d Zone 1 sensors reset.\n",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[1].Book[Loop] == 6)  { printf(" %02d:%02d:%02d Zone 2 sensors reset.\n",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[2].Book[Loop] == 6)  { printf(" %02d:%02d:%02d Zone 3 sensors reset.\n",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[0].Book[Loop] == 7)  { printf(" %02d:%02d:%02d Zone 1 fire flag reset.\n",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[1].Book[Loop] == 7)  { printf(" %02d:%02d:%02d Zone 2 fire flag reset.\n",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[2].Book[Loop] == 7)  { printf(" %02d:%02d:%02d Zone 3 fire flag reset.\n",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[0].Book[Loop] == 8)  { printf(" %02d:%02d:%02d Zone 1 system log displayed to the user.\n ",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[1].Book[Loop] == 8)  { printf(" %02d:%02d:%02d Zone 2 system log displayed to the user.\n ",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[2].Book[Loop] == 8)  { printf(" %02d:%02d:%02d Zone 3 system log displayed to the user.\n ",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[0].Book[Loop] == 9)  { printf(" %02d:%02d:%02d Sensor 1 has detected a fire in zone 1.\n ",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[0].Book[Loop] == 10) { printf(" %02d:%02d:%02d Sensor 2 has detected a fire in zone 1.\n ",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[0].Book[Loop] == 11) { printf(" %02d:%02d:%02d Sensor 3 has detected a fire in zone 1.\n ",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[1].Book[Loop] == 12) { printf(" %02d:%02d:%02d Sensor 4 has detected a fire in zone 2.\n ",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[1].Book[Loop] == 13) { printf(" %02d:%02d:%02d Sensor 5 has detected a fire in zone 2.\n ",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[1].Book[Loop] == 14) { printf(" %02d:%02d:%02d Sensor 6 has detected a fire in zone 2.\n ",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[2].Book[Loop] == 15) { printf(" %02d:%02d:%02d Sensor 7 has detected a fire in zone 3.\n ",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[2].Book[Loop] == 16) { printf(" %02d:%02d:%02d Sensor 8 has detected a fire in zone 3.\n ",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
if (Zone[2].Book[Loop] == 17) { printf(" %02d:%02d:%02d Sensor 9 has detected a fire in zone 3.\n ",
Zone[0].Hour[Loop], Zone[0].Mins[Loop], Zone[0].Secs[Loop]); }
```

```
        }

        printf("\n ****************************************************************\n");
        printf("\n Press any key to continue...");
        while (((*SCSR) & 0x20) == 0x0) { ;; }
        Build_Disp(Zone);
        Write_Book(Zone, 8, 8, 8);
        return (0);
}

/* ************************************************************************************************************************ */
/* AUTHOR    : Terence Broadbent.                                                                                           */
/* CONTRACT  : Stafford University.                                                                                         */
/* Version   : 1.0                                                                                                          */
/* Function  : SerialPort()                                                                                                 */
/* Details   : Check the serial port and obtain the key character that the user has entered via the keyboard                */
/* Libraries: N/A                                                                                                           */
/* Variables: Zone[], Eight_Bit_Port_Val.                                                                                   */
/* Modified  : N/A                                                                                                          */
/* ************************************************************************************************************************ */

unsigned char SerialPort(struct Alarm Zone[])
{
        auto unsigned char Eight_Bit_Port_Val;                          /* Local variable used within this function             */

        for (;;)
        {
                Eight_Bit_Port_Val = GrabK_Char(Zone);                  /* Keep scanning until 8 bit keystroke entered by the user    */
                return (Eight_Bit_Port_Val);                            /* Return the actual keystroke entered by the user      */
        }

}

/* ************************************************************************************************************************ */
/* AUTHOR    : Terence Broadbent.                                                                                           */
/* CONTRACT  : Stafford University.                                                                                         */
/* Version   : 1.0                                                                                                          */
/* Function  : GrabK_Char()                                                                                                 */
```

```
/* Details  : Read SCSR register value if equal to '0' then no ascii character has been entered via the keyboard keep checking.   */
/*          : If SCSR register value is equal to '1' then ascii character has been entered via the keyboard so break from the loop.*/
/* Libraries: N/A                                                                                                                 */
/* Variables: Zone[], *SCSR, *SCDR.                                                                                               */
/* Modified : N/A                                                                                                                 */
/* ****************************************************************************************************************************** */

unsigned char GrabK_Char(struct Alarm Zone[])
{
            auto unsigned char *SCDR;                                      /* Serial communications data register         */
            auto unsigned char *SCSR;                                      /* SCI status register                         */

            SCSR = (unsigned char *)0x2e;                                  /* Clears the RDRF flag                        */
            SCDR = (unsigned char *)0x2f;                            /* Retrieves last byte data buffer via the keyboard.  */

            while (((*SCSR) & 0x20) == 0x0)
            {
                  ParralPort(Zone);                         /* Scan the parallel port for any activated (switched) triggers     */
            }
            return (*SCDR);                     /* Grab the Ascii character and return its char value back to the main program    */
}

/* ****************************************************************************************************************************** */
/* AUTHOR   : Terence Broadbent.                                                                                                  */
/* CONTRACT : Stafford University.                                                                                                */
/* Version  : 1.0                                                                                                                 */
/* Function : ParralPort()                                                                                                        */
/* Details  : Check the parallel port and obtain the trigger that have been switched by the user.                                 */
/* Libraries: N/A                                                                                                                 */
/* Variables: Zone[0-2].Node[1-4], *PEDR, Data, Nbit, Bits[0-7], Sensor_Test.                                                     */
/* Modified : N/A                                                                                                                 */
/* ****************************************************************************************************************************** */

unsigned char ParralPort(struct Alarm Zone[])
{
      auto unsigned char *PEDR;                                                  /* Port E data register               */
      auto unsigned char  Data;                                                  /* Byte containing switch bits 0 - 7  */
      auto unsigned char  Nbit;                                                  /* Bit 9                              */
```

```
	auto unsigned int   Bits[8];                                           /* Will become single bits of byte 0 - 7 */
	auto unsigned int   Loop;                                              /* Local variables used by this function */
	auto unsigned int   Sensor_Test;                                       /*                  ""                   */

	PEDR = (unsigned char *)0x000a;                                        /* Assign to port E                      */
	Data = *PEDR;                                                          /* Read triggers 0 - 7 as a char byte    */
	Nbit = (*G_PADR) & 0x1;                                                /* Read the ninth trigger                */
	Sensor_Test = Off;                                       /* Used to test if the screen display is up to date    */

/* ******************************************************************************************************** */
/* Return back to the serial port scan as quickly as possible if there is no visual bit trigger changes to be displayed to the user*/
/* ******************************************************************************************************** */

	for (Loop = 0; Loop < 3; Loop++)
		{
			if (Data != 0x00 || Nbit != 0x00 || Zone[Loop].Node[Loop + 1] == 1)  { Sensor_Test = On; }
			/* The screen still needs updating */
		}
	if (Sensor_Test == Off) { return (0); }
	/* Display is good - all sensors = 0 and no triggers switched since last display build      */

/* ******************************************************************************************************** */
/* New trigger switched or the last remaining trigger has been turned off but still needs to be removed from the display carry on  */
/* ******************************************************************************************************** */

	Bits[0] = (Data & (1 << 0)) != 0;  /* Split byte variable 'Data' into a 8 bits where '1' = triggered on & '0' = not triggered*/
	Bits[1] = (Data & (1 << 1)) != 0;
	Bits[2] = (Data & (1 << 2)) != 0;
	Bits[3] = (Data & (1 << 3)) != 0;
	Bits[4] = (Data & (1 << 4)) != 0;
	Bits[5] = (Data & (1 << 5)) != 0;
	Bits[6] = (Data & (1 << 6)) != 0;
	Bits[7] = (Data & (1 << 7)) != 0;

/* ******************************************************************************************************** */
/* Repopulate the zone sensors with 'bit' trigger values, so that the sensors display the current status and update FIRE flag     */
/* ******************************************************************************************************** */
```

```
if (Zone[0].Node[0] == On  && Bits[0] == On)  { Zone[0].Node[1] = On; }           /* Zone on trip on turn sensor on                */
if (Zone[0].Node[0] == On  && Bits[0] == On)  { Zone[0].Node[4] = On; }           /* Activate fire flag                            */
if (Zone[0].Node[0] == On  && Bits[0] == Off) { Zone[0].Node[1] = Off;}           /* Zone on Trip off turn sensor off              */
if (Zone[0].Node[0] == Off && Bits[0] == On)  { Zone[0].Node[1] = Off;}/* Zone off trip on turn sensor off if prev. set  reset     */
if (Zone[0].Node[0] == On  && Bits[1] == On)  { Zone[0].Node[2] = On; }           /* Zone on trip on turn sensor on                */
if (Zone[0].Node[0] == On  && Bits[1] == On)  { Zone[0].Node[4] = On; }           /* Activate fire flag                            */
if (Zone[0].Node[0] == On  && Bits[1] == Off) { Zone[0].Node[2] = Off;}           /* Zone on Trip off turn sensor off              */
if (Zone[0].Node[0] == Off && Bits[1] == On)  { Zone[0].Node[2] = Off;} /* Zone off trip on turn sensor off - if prev. set  re-set */

if (Zone[0].Node[0] == On  && Bits[2] == On)  { Zone[0].Node[3] = On; }           /* Zone on trip on turn sensor on                */
if (Zone[0].Node[0] == On  && Bits[2] == On)  { Zone[0].Node[4] = On; }           /* Activate fire flag                            */
if (Zone[0].Node[0] == On  && Bits[2] == Off) { Zone[0].Node[3] = Off;}           /* Zone on Trip off turn sensor off              */
if (Zone[0].Node[0] == Off && Bits[2] == On)  { Zone[0].Node[3] = Off;}/* Zone off trip on turn sensor off - if prev. set  re-set  */

if (Zone[1].Node[0] == On  && Bits[3] == On)  { Zone[1].Node[1] = On; }           /* Zone on trip on turn sensor on                */
if (Zone[1].Node[0] == On  && Bits[3] == On)  { Zone[1].Node[4] = On; }           /* Activate fire flag                            */
if (Zone[1].Node[0] == On  && Bits[3] == Off) { Zone[1].Node[1] = Off;}           /* Zone on Trip off turn sensor off              */
if (Zone[1].Node[0] == Off && Bits[3] == On)  { Zone[1].Node[1] = Off;} /* Zone off trip on turn sensor off - if prev. set  re-set */

if (Zone[1].Node[0] == On  && Bits[4] == On)  { Zone[1].Node[2] = On; }           /* Zone on trip on turn sensor on                */
if (Zone[1].Node[0] == On  && Bits[4] == On)  { Zone[1].Node[4] = On; }           /* Activate fire flag                            */
if (Zone[1].Node[0] == On  && Bits[4] == Off) { Zone[1].Node[2] = Off;}           /* Zone on Trip off turn sensor off              */
if (Zone[1].Node[0] == Off && Bits[4] == On)  { Zone[1].Node[2] = Off;} /* Zone off trip on turn sensor off - if prev. set re-set  */

if (Zone[1].Node[0] == On  && Bits[5] == On)  { Zone[1].Node[3] = On; }           /* Zone on trip on turn sensor on                */
if (Zone[1].Node[0] == On  && Bits[5] == On)  { Zone[1].Node[4] = On; }           /* Activate fire flag                            */
if (Zone[1].Node[0] == On  && Bits[5] == Off) { Zone[1].Node[3] = Off;}           /* Zone on Trip off turn sensor off              */
if (Zone[1].Node[0] == Off && Bits[5] == On)  { Zone[1].Node[3] = Off;} /* Zone off trip on turn sensor off - if prev. set  re-set */

if (Zone[2].Node[0] == On  && Bits[6] == On)  { Zone[2].Node[1] = On; }           /* Zone on trip on turn sensor on                */
if (Zone[2].Node[0] == On  && Bits[6] == On)  { Zone[2].Node[4] = On; }           /* Activate fire flag                            */
if (Zone[2].Node[0] == On  && Bits[6] == Off) { Zone[2].Node[1] = Off;}           /* Zone on Trip off turn sensor off              */
if (Zone[2].Node[0] == Off && Bits[6] == On)  { Zone[2].Node[1] = Off;} /* Zone off trip on turn sensor off - if prev. set  re-set */

if (Zone[2].Node[0] == On  && Bits[7] == On)  { Zone[2].Node[2] = On; }           /* Zone on trip on turn sensor on                */
if (Zone[2].Node[0] == On  && Bits[7] == On)  { Zone[2].Node[4] = On; }           /* Activate fire flag                            */
if (Zone[2].Node[0] == On  && Bits[7] == Off) { Zone[2].Node[2] = Off;}           /* Zone on Trip off turn sensor off              */
```

```
if (Zone[2].Node[0] == Off && Bits[7] == On)  { Zone[2].Node[2] = Off;}/* Zone off trip on turn sensor off - if prev. set  re-set  */

if (Zone[2].Node[0] == On  && Nbit == 0x01)   { Zone[2].Node[3] = On; }           /* Zone on trip on turn sensor on               */
if (Zone[2].Node[0] == On  && Nbit == 0x01)   { Zone[2].Node[4] = On; }           /* Activate fire flag                           */
if (Zone[2].Node[0] == On  && Nbit == 0x00)   { Zone[2].Node[3] = Off;}           /* Zone on Trip off turn sensor off             */
if (Zone[2].Node[0] == Off && Nbit == 0x01)   { Zone[2].Node[3] = Off;}/* Zone off trip on turn sensor off - if prev. set  re-set  */

       Build_Disp(Zone);

/* Write the above Zone data changes to the log book   for zones 1,2 and 3                                                        */

       if (Zone[0].Node[0] == On && Bits[0] == On) { Write_Book( Zone, 9, 0, 0);  }             /* Trigger 1 activated by the user */
       if (Zone[0].Node[0] == On && Bits[1] == On) { Write_Book( Zone, 10, 0, 0); }             /* Trigger 2 activated by the user */
       if (Zone[0].Node[0] == On && Bits[2] == On) { Write_Book( Zone, 11, 0, 0); }             /* Trigger 3 activated by the user */
       if (Zone[1].Node[0] == On && Bits[3] == On) { Write_Book( Zone, 0, 12, 0); }             /* Trigger 4 activated by the user */
       if (Zone[1].Node[0] == On && Bits[4] == On) { Write_Book( Zone, 0, 13, 0); }             /* Trigger 5 activated by the user */
       if (Zone[1].Node[0] == On && Bits[5] == On) { Write_Book( Zone, 0, 14, 0); }             /* Trigger 6 activated by the user */
       if (Zone[2].Node[0] == On && Bits[6] == On) { Write_Book( Zone, 0, 0, 15); }             /* Trigger 7 activated by the user */
       if (Zone[2].Node[0] == On && Bits[7] == On) { Write_Book( Zone, 0, 0, 16); }             /* Trigger 8 activated by the user */
       if (Zone[2].Node[0] == On && Nbit == 0x01 ) { Write_Book( Zone, 0, 0, 17); }             /* Trigger 9 activated by the user */
       return (0);
}

/* ******************************************************************************************************************************** */
/* AUTHOR    : Terence Broadbent.                                                                                                  */
/* CONTRACT  : Stafford University.                                                                                                */
/* Version   : 1.0                                                                                                                 */
/* Function  : Time_Stamp()                                                                                                        */
/* Details   : Set Hours:Minutes:Seconds for time and log book entries in each zone.                                               */
/* Libraries: N/A                                                                                                                  */
/* Variables: Zone[0-2].Hour[0-24], Zone[0-2].Mins[0-60], Zone[0-2].Secs[0-60], Zone[0-2].Line[0].                                 */
/* Modified  : N/A                                                                                                                 */
/* ******************************************************************************************************************************** */

unsigned int Time_Stamp(struct Alarm Zone[])
{
       Zone[0].Hour[Zone[0].Line[0]] = G_Hours;                              /* Write the hours into the time log for Zone 1        */
       Zone[1].Hour[Zone[1].Line[0]] = G_Hours;                              /*                         ""                 2        */
```

```
        Zone[2].Hour[Zone[2].Line[0]] = G_Hours;                            /*                    ""                3       */
        Zone[0].Mins[Zone[0].Line[0]] = G_Mins;                             /* Write the minutes into the time log for Zone 1       */
        Zone[1].Mins[Zone[1].Line[0]] = G_Mins;                             /*                    ""                  2     */
        Zone[2].Mins[Zone[2].Line[0]] = G_Mins;                             /*                    ""                  3     */
        Zone[0].Secs[Zone[0].Line[0]] = G_Secs;                             /* Write the seconds into the time log for Zone 1       */
        Zone[1].Secs[Zone[1].Line[0]] = G_Secs;                             /*                    ""                  2     */
        Zone[2].Secs[Zone[2].Line[0]] = G_Secs;                             /*                    ""                  3     */
        return (0);
}

/* ************************************************************************************************************************************ */
/* AUTHOR   : Terence Broadbent.                                                                                                        */
/* CONTRACT : Stafford University.                                                                                                      */
/* Version  : 1.0                                                                                                                       */
/* Function : Comp_Strng()                                                                                                              */
/* Details  : Check 2 strings the system password and the user entered password if they match return true else return false.          */
/* Libraries: N/A                                                                                                                       */
/* Variables: String1[], String2[], Charater, MatchValue.                                                                               */
/* Modified : N/A                                                                                                                       */
/* ************************************************************************************************************************************ */

unsigned int Comp_Strng(const unsigned char String1[], const unsigned char String2[])
{
        auto unsigned int Character;
        auto unsigned int MatchValue;

        Character = 0;
        MatchValue = False;

        while (String1[Character] == String2[Character] && String1[Character] != '\0' && String2[Character] != '\0')
                {
                        Character++;
                        if (String1[Character] == String2[Character]) { MatchValue = True;}
                                else
                                        { MatchValue = False;}
                }
                return (MatchValue);
}
```

```c
/* ************************************************************************************************************************************ */
/* AUTHOR   : Terence Broadbent.                                                                                                        */
/* CONTRACT : Stafford University.                                                                                                      */
/* Version  : 1.0                                                                                                                       */
/* Function : Write_Book()                                                                                                              */
/* Details  : Write to the log book the short logcodes (values) passed to this function ie what is happening in each function call.*/
/* Libraries: N/A                                                                                                                       */
/* Variables: Zone[0-2].Book[0-2]; Zone[0-2].Line[0], Zcode1, ZCode2, ZCode3.                                                          */
/* Modified : N/A                                                                                                                       */
/* ************************************************************************************************************************************ */

unsigned int Write_Book(struct Alarm Zone[], const unsigned int ZCode_0, const unsigned int ZCode_1, const unsigned int ZCode_2)
{
       Time_Stamp(Zone);                                            /* Time stamp the log entry                */
       Zone[0].Book[Zone[0].Line[0]] = ZCode_0;                     /* Write the data to the log book about Zone 1  */
       Zone[1].Book[Zone[1].Line[0]] = ZCode_1;                     /* Write the data to the log book about Zone 2  */
       Zone[2].Book[Zone[2].Line[0]] = ZCode_2;                     /* Write the data to the log book about Zone 3  */
       Zone[0].Line[0]++;                                           /* Increment Zone 1 log book entry by 1    */
       if (Zone[0].Line[0] == 100) { Zone[0].Line[0] = 0; }         /* Check 100 entries limit & reset if hit  */
       Zone[1].Line[0]++;                                           /* Increment Zone 2 log book entry by 1    */
       if (Zone[1].Line[0] == 100) { Zone[1].Line[0] = 0; }         /* Check 100 entries limit & reset if hit  */
       Zone[2].Line[0]++;                                           /* Increment Zone 3 log book entry by 1    */
       if (Zone[2].Line[0] == 100) { Zone[2].Line[0] = 0; }         /* Check 100 entries limit & reset if hit  */
       return (0);
}

/* ************************************************************************************************************************************ */
/* #Eof */
```

Figure 62 – Printout of ANSI C code