



Electronic Engineering Department

EBU5475 Microprocessor Systems Design
Introduction to the MCS-51 Development Board.

Laboratory Session 3

1. Overview

The purpose of these experiments is to familiarise you with the 8051 microcontroller family. With this intention, you will use the MCS-51 development board developed at BUPT with the READS51 software. In this lab you will use relative assembly techniques to build programs using reusable code modules in order to control the display and keypad on the lab development board.

NOTE: All the information you need to prepare the code for this lab is contained in the lab script and its appendices – you do not need to have the development board in order to plan, design and code the software as long as you read all the information carefully.

2. The MCS-51 Development Board

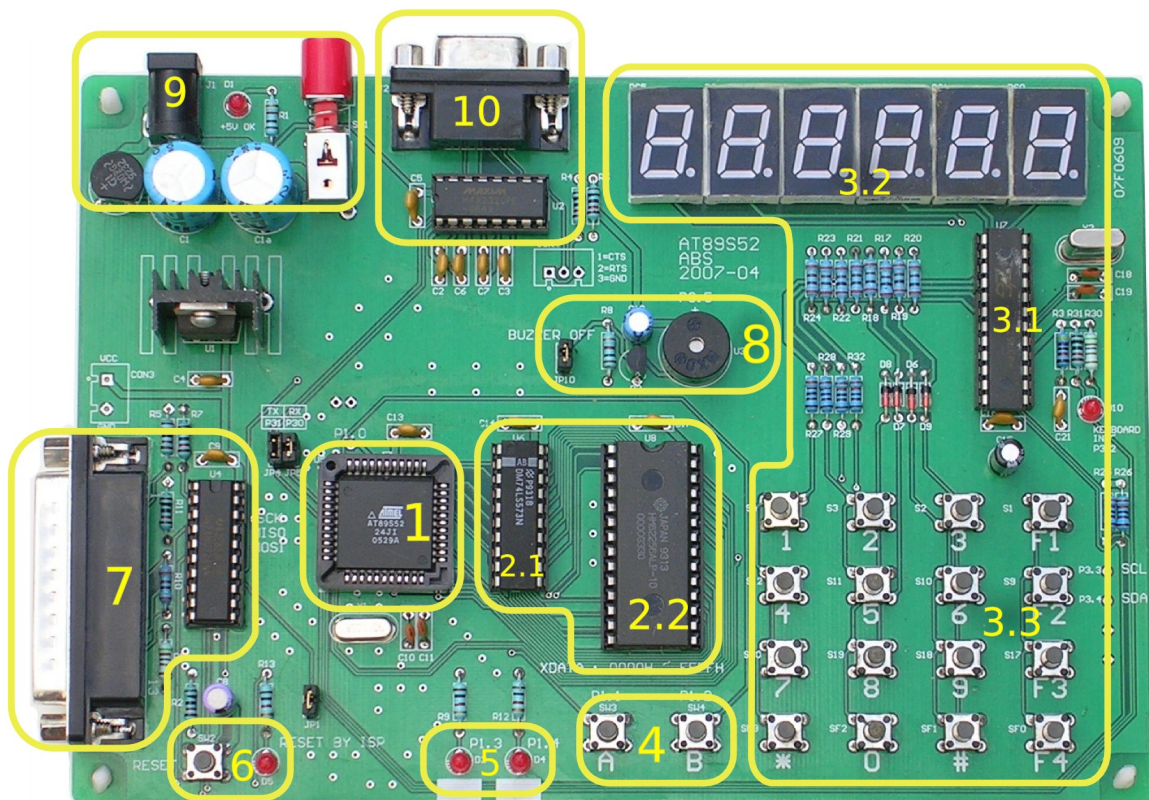


Figure 1: MCS-51 Development Board – see Table 1 for key

Figure 1 shows the board you will use for these labs. It is built around an Atmel AT89S52 microcontroller chip, which is basically an 8052 microcontroller with Flash memory for its internal 8K program ROM. The MCU is clocked using a 11.0592MHz crystal oscillator and 32K external RAM is provided for data storage.

The board has an RS232 serial port, a 6-digit display and 16-key keypad plus two general-purpose switches and two general-purpose LED indicators. The display and keypad are controlled by a ZLG7290 driver chip that is connected to the MCU via an I²C bus interface.

The Atmel MCU is In-System-Programmable (ISP) which means that it can be reprogrammed while on the circuit board instead of having to use a dedicated chip programming device. This makes programming the board quick and easy and is therefore very useful for prototyping designs. The ISP connection is made using a parallel port cable to interface to a PC.

No.	Description
1	Atmel AT89S52 Microcontroller Chip
2.1	Address Latch (74LS573)
2.2	32K External RAM (HM62256)
3.1	I ² C Display and Keypad Driver Chip (ZLG7290) – Interfaces the keypad and display to the MCU
3.2	6-digit, 7-segment (plus decimal points) display
3.3	Hex Keypad
4	Two general purpose switches (labelled A and B) connected to P1.1 and P1.2
5	Two LED indicators (labelled D3 and D4) connected to P1.3 and P1.4
6	RESET switch
7	Parallel port connector and buffer chip (74HC244) for programming the MCU
8	Buzzer connected to P3.5 – disabled by jumper JP10
9	Power Connector and On/Off switch
10	RS232 Serial Port (using a MAX232 driver chip)

Table 1: Key to the MCS-51 development board components shown in Figure 1

Figure 2 shows a system diagram for the board. Ports 0 and 2 are used for interfacing to the external RAM while ports 1 and 3 are used to interface with the other components on the board. Most of the components connected to the pins on ports 1 and 3 can be disconnected from the port by using the jumpers on the board (see table 2).

Details of all the ICs on the development board are available in data sheets available from the course website. You should read the data sheets to make sure you are familiar with the components that are used.

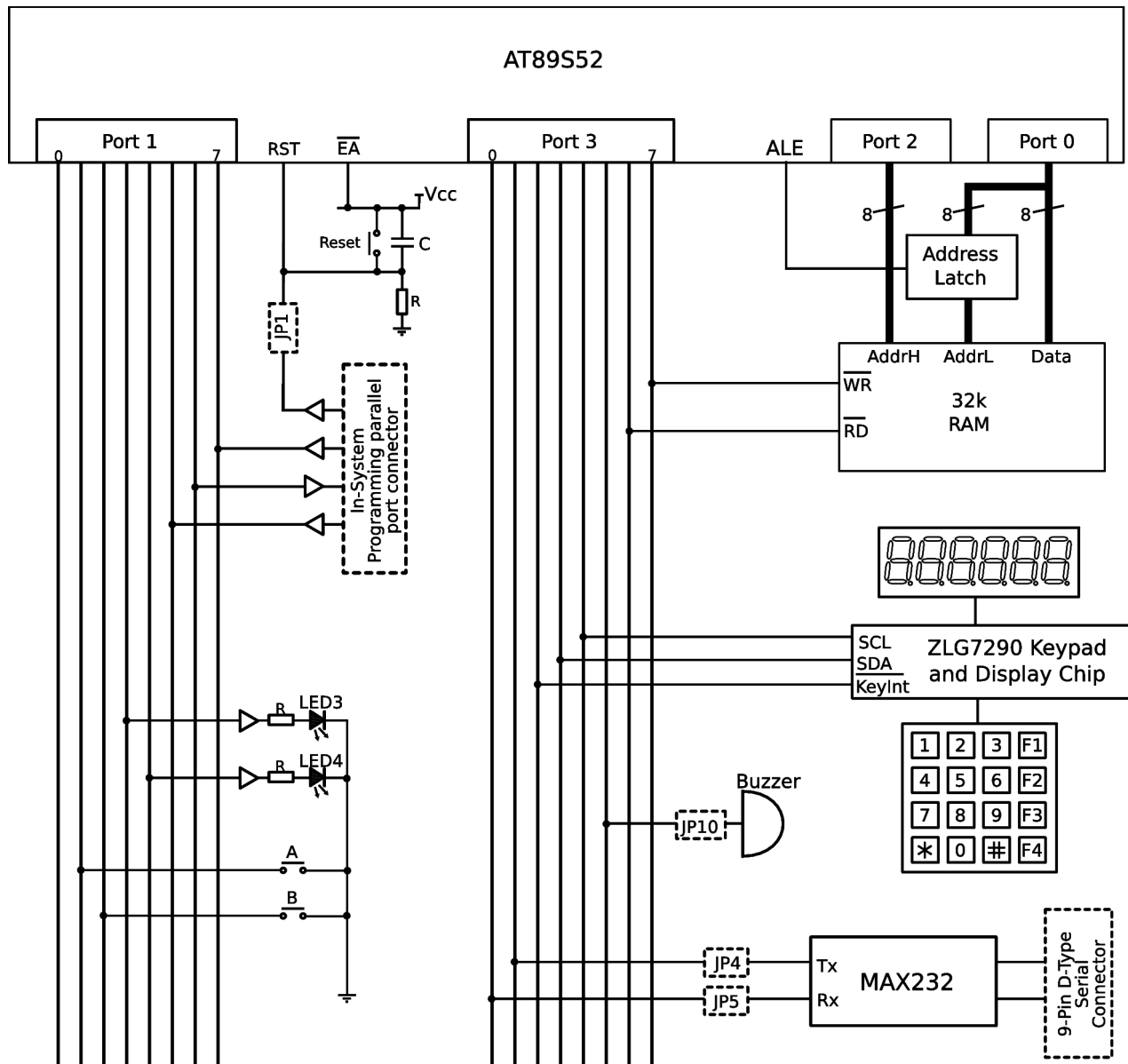


Figure 2: System Diagram for the MCS-51 Development Board

JP1	Disconnects ISP reset input from RST pin	JP4	Disconnects MAX232 Tx input from P3.1
JP10	Disables the Buzzer (P3.5)	JP5	Disconnects MAX232 Rx output from P3.0

Table 2: Functions of the jumpers on the development board

3. Preparatory Work: Writing code for the development board

You are given the following files:

- **boardtest10.hex** – this is an assembled hex file ready to download to the board in the lab
- **boardtestmain_10.asm** – this is a module containing the test program. It uses subroutines from the ZLG7290_driver_10.asm driver module.
- **ZLG7290_driver_10.asm** – this is a module containing driver code for the display and keypad chip. It uses subroutines from the I2C_driver_10.asm driver module.
- **I2C_driver_10.asm** – this is a module containing driver code for I²C bus connection to the display chip.
- **ZLG7290_dummy_10.asm** – this is a dummy version of the driver module for use when debugging your code in Reads51.

Tasks

Now you have read how the board is laid out your groups should prepare some programs in Reads51 to download to the MCU during the lab. The following simple tasks can be done in single .asm files:

- 1) Using the information from section 2 of the lab script, write a simple program that lights LED3 when switch A is pressed and LED4 when switch B is pressed (note that the switches pull the port pins low).
- 2) Write a simple program that will sound the buzzer when switches A and B are pressed at the same time.
- 3) Write a program that will flash LED3 on and off every 0.25 seconds using nested delay loops as seen in week 2 lectures.

Read through the information on **Relative Assembly**, the **ZLG7290 driver** and the **example test program** in the appendices. Create a Reads51 project including the three .asm code modules provided. Using the information provided, prepare a program that will perform the following tasks:

- 4) Using the source file for the test program “boardtestmain_10.asm” as a starting point, write code that will put the word “HONG FU” on the display when Key A is pressed (Hint, you can do this in the same way as messages “NI HAO” and “HELLO” are put on the display in the example program).
- 5) Alter the code so that the typing on the keypad inserts characters in from the left side, shifting the rest of the display to the right.
- 6) Alter the code so that pressing Key B once will make the display flash and pressing it again stops the flashing.
- 7) Alter the code so that instead of displaying hex values, pressing the keys on the keypad will show the values that appear marked in white under the keys as shown in Figure 3. When an “Fn” key is pressed it should insert the character “F” and the number n. When the * is pressed insert “A” and when # is pressed insert “C”. (Hint: you should use a “case” program structure to implement the decoding).

1	2	3	F1
4	5	6	F2
7	8	9	F3
*	0	#	F4

Figure 3: Keypad labels

4. Preparation Final Challenge: An electronic “Safe”

If your group managed to do the rest of the exercises then here is a final challenge – programming an electronic version of a combination lock.



Figure 4: A safe with an electronic lock

The Safe has two modes: Open and Closed

Open Mode:

In this mode the safe is open so your program should allow the user to set the code and lock it closed.

- The user can enter a 6 digit “key code” on the labelled keypad number keys.
- Keypad “F1” should clear the display
- Keypad “F2” should save the new code into the internal RAM
- Keypad “F3” should lock the Safe and display the word “CLOSEd” then change to Closed Mode.
- Keypad keys *, # and F4 should be ignored.
- Keys A and B should be ignored. LED3 should be lit. LED4 should be off.

Closed Mode:

In this mode your program should allow the user to try to open the safe with a key code.

- The user can enter a 6 digit “key code” on the labelled keypad number keys.
- Keypad “F1” should clear the display.
- Keypad “F3” should attempt to unlock and open the safe with the entered key code:
 - ◆ If the entered code correctly matches the saved code in the RAM then display the word “OPEN” and change to Open mode.
 - ◆ If the entered code is wrong then display the words “BU HAO” and stay in Closed mode.
- Keypad keys *, #, F2 and F4 should be ignored.
- Keys A and B should be ignored. LED3 should be off. LED4 should be lit.

Tips for successful preparation:

- You should use the structured programming techniques discussed in week 2 lectures in order to guide your code writing.
- After designing and writing your code, use the dummy driver module for testing that you call the subroutines correctly and handle return values as expected. Remember you can alter any register or memory value in the debugger when you want to simulate the correct return values.

Deliverables: Preparatory Work

Your code should be prepared before the session because there will be no time to write code during the lab. You will be expected to demonstrate your programs to the teaching assistants during the lab session.

Part 1: Testing the board

Download the boardtest10.hex file from the web page if you have not already done so. Following the instructions in Appendix 1 on how to download programs to the board using the Atmel ISP software. Make sure that you can correctly program a hex file to the board.

Part 2: Downloading and testing your own programs

Once you have successfully programmed the board with the example hex file, try assembling and downloading the single file example programs from preparatory work tasks 1, 2 and 3. (note: Reads51 produces a hex file every time you build your code or toggle into Run/Debug mode). Do the programs function as you expected?

Now try assembling the board test program using the code modules boardtestmain_10.asm, ZLG7290_driver_10.asm and I2C_driver_10.asm in a Reads51 project. Make sure that you can program the board with your newly assembled hex file (remember to use the driver module and exclude the dummy version from the build...).

For the remaining time in the lab session, download the program you have written for preparatory tasks 4,5,6 and 7 to the boards and test the functionality for each task. Demonstrate to the Teaching Assistants that you have correctly programmed these tasks.

If you have managed to complete the final challenge download this to the board and test it also. Demonstrate this to the Teaching Assistants if you complete it.

Deliverables: Lab 3

All parts of the lab preparatory work (including the final challenge) carry marks and you will also be awarded marks for demonstrating your work to the Teaching Assistants during the lab session.

Your group should hand in the following in electronic form:

- Code for all the tasks in the lab arranged in sensible folders in a zip file.

Appendix 1: Programming the board

The first thing to do is to establish that your development board is working properly by downloading a test program to it. The program you will download (boardtest10.hex) should test all of the components on the board. Follow these steps:

- 1) Plug the power supply into the board then connect to the PC using the parallel cable.
- 2) Run the Atmel ISP software. At the top of the window you will see a menu like this:

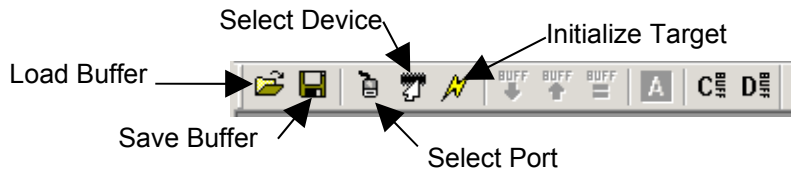


Figure 5: Key to the Atmel ISP software menu setup buttons

- 3) Choose "Select Device" and you should get a menu that looks like the picture below. You want to choose device "AT89S52" in "Byte Mode" with XTAL at 12MHz then press OK.

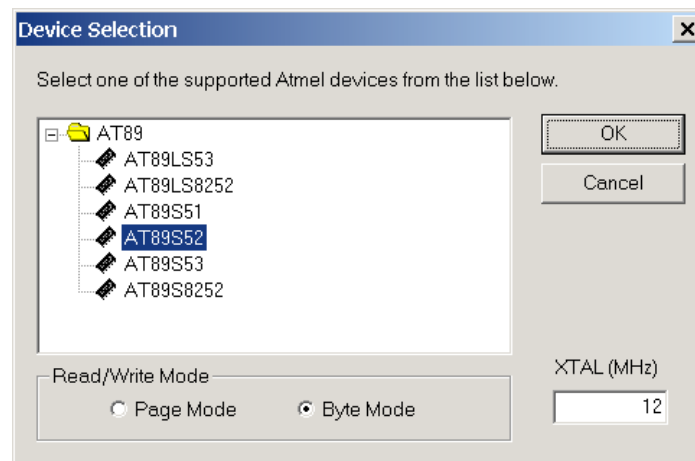


Figure 6: Selecting the AT89S52 device

- 4) If the board is correctly connected, the software should sense it and a blank buffer screen containing FF in every byte position should appear (see below) showing that the chip is ready to be programmed. If this does not happen, try pressing "Initialize Target" and if that doesn't help, call a demonstrator.

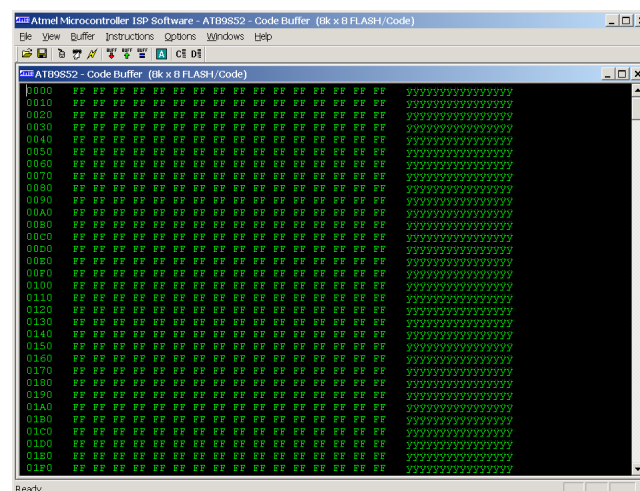


Figure 7: AT89S52 buffer ready to program

- 5) Now you should be ready to program the chip so you need to load a hex file into the programming buffer. Choose "Load Buffer" and select the test file "boardtest10.hex". This should change the contents of the buffer on the screen.
- 6) Now that the chip is ready to program several new buttons should have become active on the toolbar (see below). Select "Autoprogram" to download the program code to the chip.

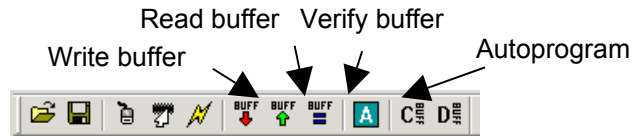


Figure 8: Key to the Atmel ISP software programming buttons

- 7) If the download is successful you will then be prompted by the menu shown below. Select "Lock0" and press OK.

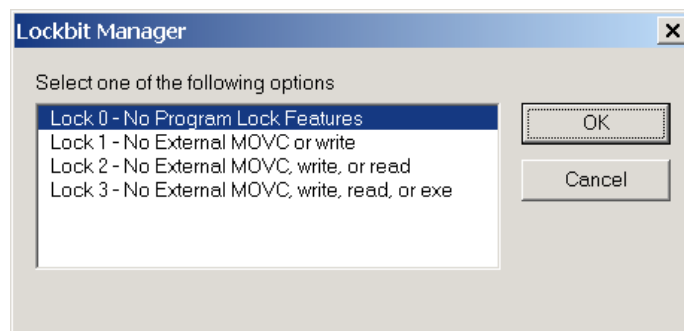


Figure 9: Setting the lock mode on the microcontroller (choose Lock 0)

- 8) At this point the download should be complete and you should see this window:

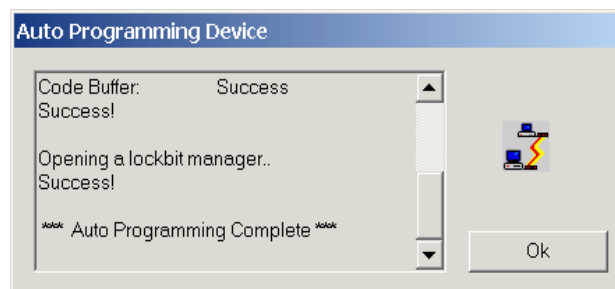


Figure 10: Successful programming

- 9) The download has been successful so press OK. Now it is time to see what the code does by running it on the board. To take the board out of programming mode select "Instructions -> Run Target" from the menu. This will put the board into run mode and show the following message:



Figure 11: Run target

- 10) When you are ready to program the board again, you can select "Initialize Target" to put the board back into programming mode then start the process again from step 4.

Take a couple of minutes to try out the test program with the board to see what it does. All of the buttons, keys, LEDs and displays can be tested with this program. By carefully removing jumper JP10 you will be able to test the buzzer as well.

Appendix 2: Relative assembly in Reads51

In this lab you will use relative assembly techniques to organise your code into modules as we discussed in lectures. To use relative assembly in Reads51 you must open a project. A Reads51 project file (.rpj file) stores information about all the different code files that make up your program (This is analogous to the role of the .ise file in your Xilinx VHDL projects from last term.).

Select **Project→New Project** and choose a name and location for your project files.

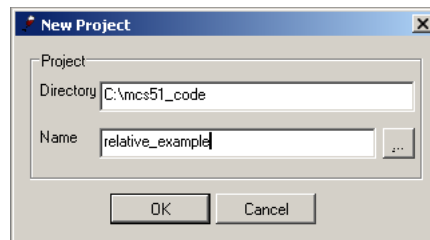


Figure 12: Creating a new project

Once your project has been created you should see a project window appear on the left side of the screen. You now need to add some code modules to the project. Select **Module→Create Module** to add start a new code file:

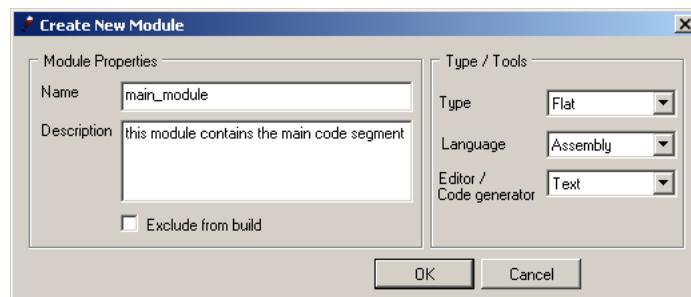


Figure 13: Creating a new code module

Give your module a name and select “Assembly” as the language. this will create a new .asm file and it will appear as a module in the project window.

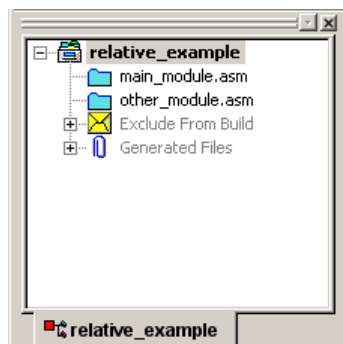


Figure 14: Project files window

As a simple example of relative assembly try entering the following two sections of code in two different modules:

Main module

```
; external subroutines used by this module
extern code my_subroutine

; main code segment starts at 0000h
cseg at 0

lcall my_subroutine ; main code just calls the external subroutine

end
```

Other module

```
;declare the code segments in this module
Module_2_Routines segment code

; Declare public subroutines available from this module
public my_subroutine

; relative code segment
rseg Module_2_Routines

my_subroutine:
    mov A, #CCh ; subroutine sets A to CCh
    ret         ; return to calling code

end
```

If you run this code in RUN/DEBUG mode you will see that you start in the main code segment then jump to the external subroutine in the second module before returning to the main module (remember to set your Project Build options to generate debug information for this to work properly).

Dealing with multiple modules and testing

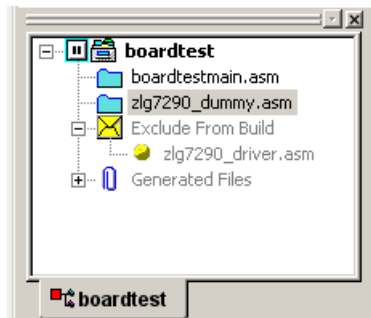


Figure 15: ZLG7290_dummy_10.asm is used for testing instead of ZLG7290_driver_10.asm

Sometimes it is necessary to test code which will eventually rely on the functionality of another module which may not yet be written or may be awkward to step through in the debugger. In these instances it is helpful to write test modules that expose the same interfaces yet do nothing themselves. An example of this kind of file “ZLG7290_dummy_10.asm” is provided for testing your preparation work. The same subroutines are exposed in the API as those of the driver but none contain any code. When the debugger jumps to these routines you can check whether the registers and RAM contain the correct values from your code and then manually set the return values. As you can see in Figure 15, it is possible to drag files that you do not wish to use to build your assembled code into the “Exclude From Build” area. In this case we want both the driver and the dummy to be part of the project but we do not want to try building with both modules together as they will conflict with each other. Therefore we can choose to exclude the driver file and use the dummy while testing but use the driver and exclude the dummy when we do a final build.

Appendix 3: The ZLG7290 and I²C Driver Modules

The files ZLG7290_driver_10.asm and I2C_driver_10.asm contain code that allows you to interface easily with the display and keypad on the board. The drivers are made up of a set of MCS-51 assembly code subroutines in a relative code segment. The ZLG7290 driver abstracts the functionality of the hardware exposing an application programmers interface (API) of eight simple functions which can be used to control the display and keypad. The user software (i.e. your program code) can call these functions to interface easily with the hardware. This arrangement can be described graphically as shown in Figure 16.

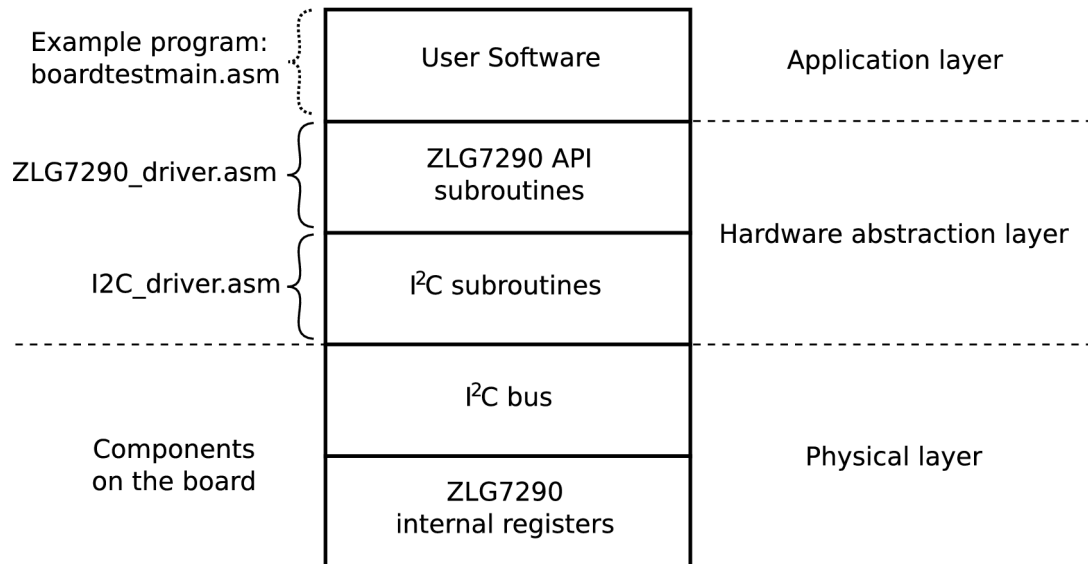


Figure 16: Arrangement of software and hardware for using ZLG7290 driver

The API exposed by the ZLG7290 driver provides the following eight subroutines:

<code>_ZLG_Init</code>	Initialise the ZLG routine
<code>_ZLG_DisplayMessage</code>	Display a message by loading data directly to the display
<code>_ZLG_DisplayHexChar</code>	Display a hexadecimal value on a specific digit
<code>_ZLG_ShiftRight</code>	Shift the display right by one digit
<code>_ZLG_ShiftLeft</code>	Shift the display left by one digit
<code>_ZLG_DisplayFlashOn</code>	Flash all digits on the display
<code>_ZLG_DisplayFlashOff</code>	Disable flash on all digits on the display
<code>_ZLG_ReadHexValue</code>	Read a hexadecimal value decoded from the lab board keypad

NOTE: The underscore “_” at the beginning of each subroutine name helps to distinguish them from subroutine names in your main code.

To use these subroutines you need to set up a **data structure** in the internal RAM that contains the information necessary to perform the desired operation. It is then possible to pass the address of this structure as an argument to the subroutine in one of the registers.

ZLG_Init

Description:

Subroutine `_ZLG_Init` is used to initialise a data structure that will be used with the driver. This structure is simply a 10-byte space in 8051 internal RAM that the programmer reserves for use when communicating with the ZLG7290 driver. The user passes the address of this data structure to this subroutine and the correct slave address for the ZLG7290 will be loaded into the first byte of the structure.

This subroutine must be run to initialise the data structure before using it with the other subroutines.

Inputs:

The base address of the data structure should be passed to the subroutine as a pointer in R3.

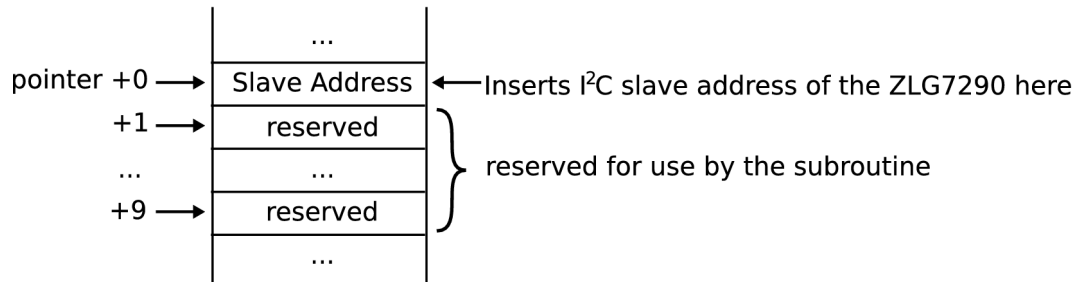


Figure 17: ZLG structure initialised in internal RAM

DisplayMessage

Description:

Subroutine `_ZLG_DisplayMessage` changes the current message shown on the lab board 6 digit display. It returns a 0 in the carry flag if successful.

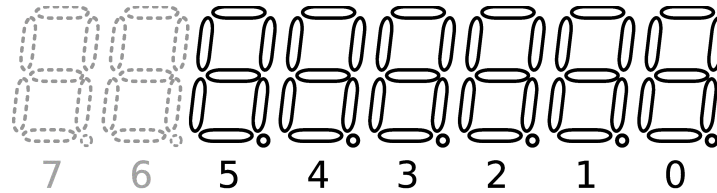


Figure 18: Arrangement of digit displays on the board

The ZLG7290 chip can actually control up to 8 seven-segment displays but the development boards we will use in the labs have only six (the arrangement of which is shown in Figure 18).

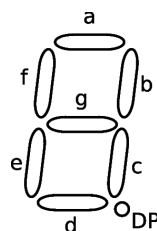


Figure 19: Seven-segment display segment arrangement

The DisplayMessage subroutine allows you to write directly to the display digits using a bit pattern to control which segments of the digit are lit. The segments are labelled a-g and DP for the decimal point and are arranged in a data byte in the bit order shown below:

D7	D6	D5	D4	D3	D2	D1	D0
a	b	c	d	e	f	g	DP

Hence to show the letter "E" you would use the binary bit pattern 10011110 or hex value 0x9E.

Inputs:

The base address of the data structure should be passed to the subroutine as a pointer in R3.

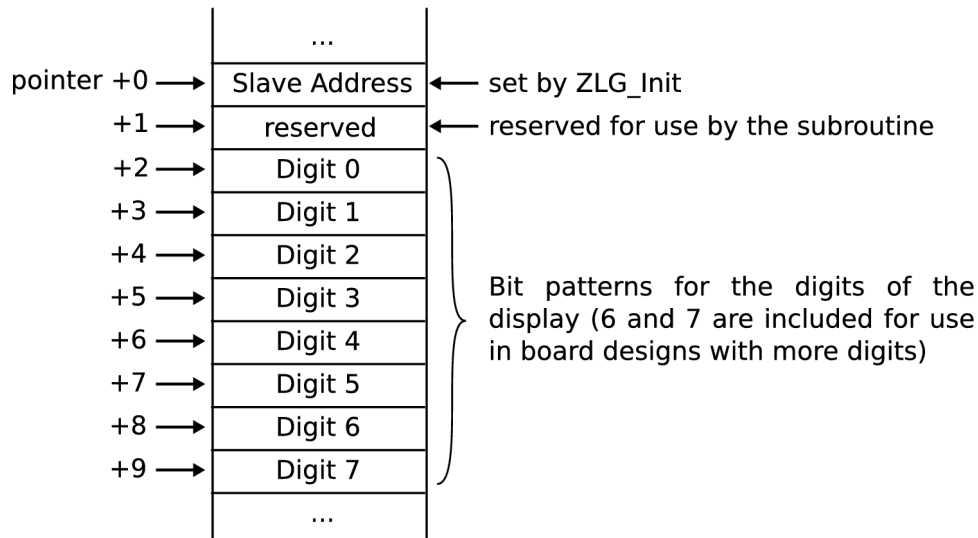
Data Structure arrangement:

Figure 20: Data structure for DisplayMessage subroutine in internal RAM

To use DisplayMessage you must set up a data structure in internal RAM as shown in Figure 20. The structure must contain data bytes holding the bit patterns for each digit in the display.

DisplayHexChar**Description:**

Subroutine `_ZLG_DisplayHexChar` puts a hexadecimal character on a specific digit on the display.

Inputs:

The base address of the data structure should be passed to the subroutine as a pointer in R3.

The address of the digit to change on the display (see Figure 18 for digit addresses) should be passed to the subroutine in the accumulator.

Outputs:

Returns a 0 in the carry flag if successful.

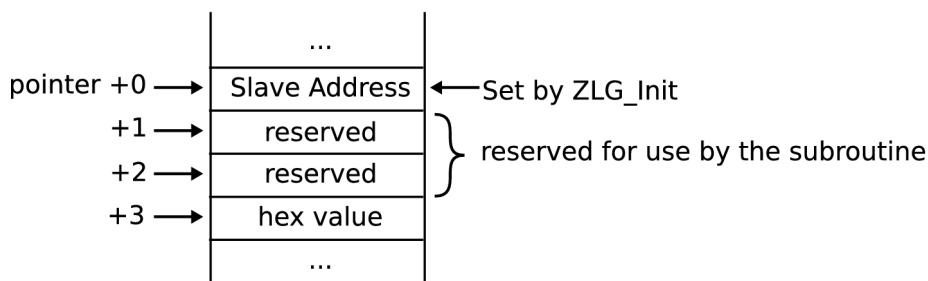
Data Structure arrangement:

Figure 21: Data structure for DisplayHexChar

The hexadecimal value to be displayed should be placed in RAM 3 bytes above the base address of the structure (i.e. pointer +3).

ShiftRight and ShiftLeft

Description:

Subroutines `_ZLG_ShiftRight` and `_ZLG_ShiftLeft` shift the digits shown on the display right or left by one space respectively.

Inputs:

The base address of the data structure should be passed to the subroutine as a pointer in R3.

Outputs:

Both return 0 in the carry flag if successful.

Data Structure arrangement:

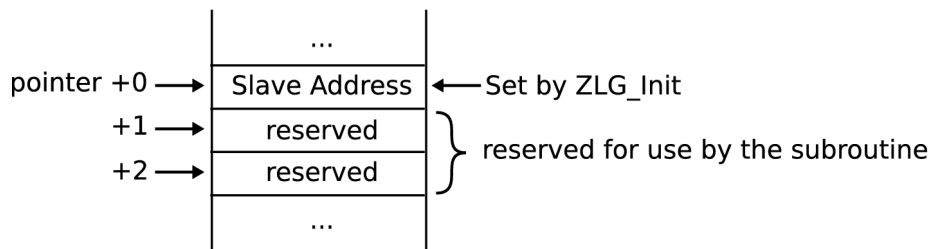


Figure 22: Data structure for DisplayFlashOn and DisplayFlashOff

DisplayFlashOn and DisplayFlashOff

Description:

The ZLG driver subroutine `_ZLG_DisplayFlashOn` makes all digits on the display start to flash; `_ZLG_DisplayFlashOff` stops all digits flashing.

Inputs:

The base address of the data structure should be passed to the subroutine as a pointer in R3.

Outputs:

Both return 0 in the carry flag if successful.

Data Structure arrangement:

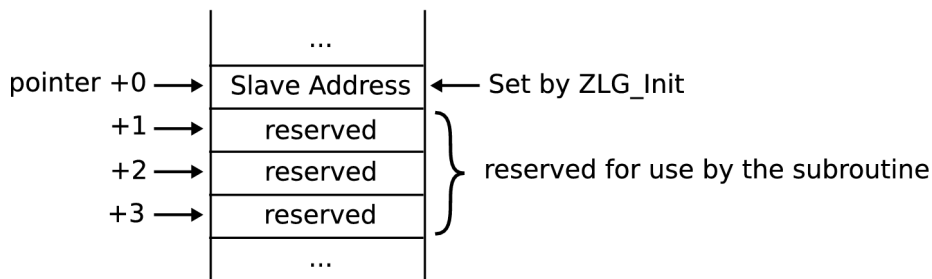


Figure 23: Data structure for DisplayFlashOn and DisplayFlashOff

ReadHexValue

Description:

Subroutine `_ZLG_ReadHexValue` reads which key has been pressed on the keypad and returns a hexadecimal value corresponding to the mapping (shown in Figure 24) in the accumulator. This subroutine would usually be used after the KeyInt (keypad interrupt) line from the ZLG7290 goes low indicating a keypad event.

0	1	2	3
4	5	6	7
8	9	A	B
C	D	E	F

Figure 24: Keypad mapping for ReadHexValue

Inputs:

The base address of the data structure should be passed to the subroutine as a pointer in R3.

Outputs:

The hexadecimal value corresponding to the key that was pressed is returned in the accumulator.

The subroutine also returns 0 in the carry flag if a key was correctly read or 1 if no key was pressed.

Data Structure arrangement:

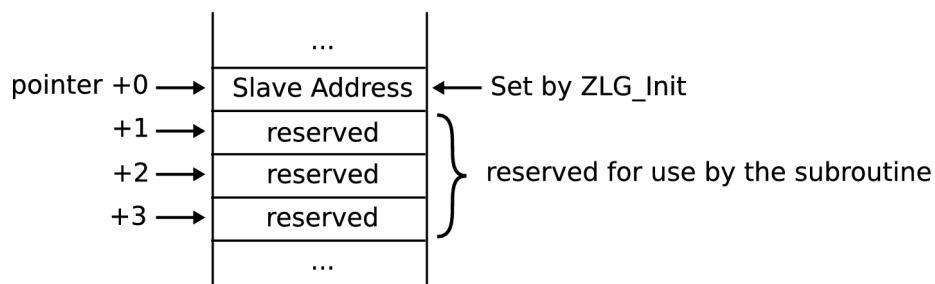


Figure 25: Data structure for ReadHexValue

Appendix 4: The design of the board test program

The lab board test program is made up of three .asm files: boardtestmain_10.asm, ZLG7290_driver_10.asm and I2C_driver_10.asm. These can be put together in a Reads51 project and assembled to produce the downloadable file “boardtest10.hex”.

This program demonstrates the functions of the board in the following ways:

- Press Key A: LED3 lights, buzzer sounds, display shows “HELLO” as shown in Figure 26.

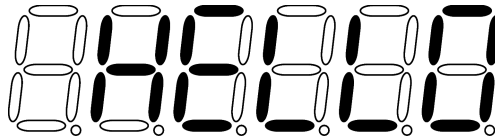


Figure 26: Result of SayHello code

- Press Key B: LED4 lights, buzzer sounds, display shows “Ni HAO” as shown in Figure 27.

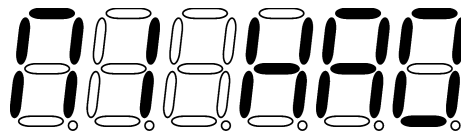


Figure 27: Result of SayNiHao code

- Press a key on the keypad: A hex character (keys corresponding to mapping shown in Figure 24) is inserted in the rightmost digit (all other digits shift along to the left).

Flow Diagrams:

The code to enable this behaviour can be described by the following flow diagrams:

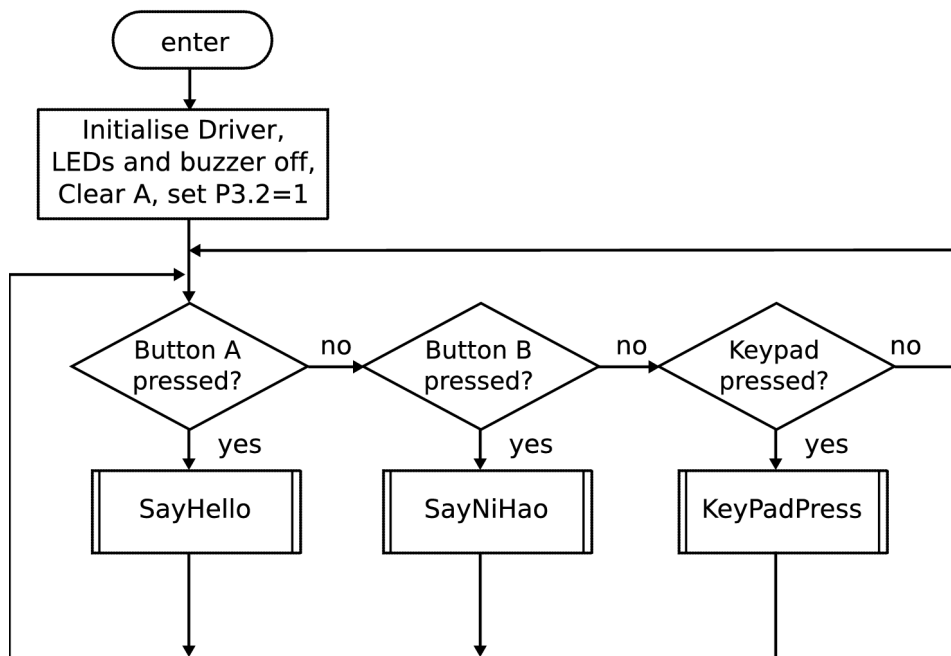


Figure 28: Top level of the program – endless loop so no exit necessary.

The top level of the code has a short initialisation step to set things up then goes into a simple loop which checks to see if key A or B or any key on the keypad has been pressed.

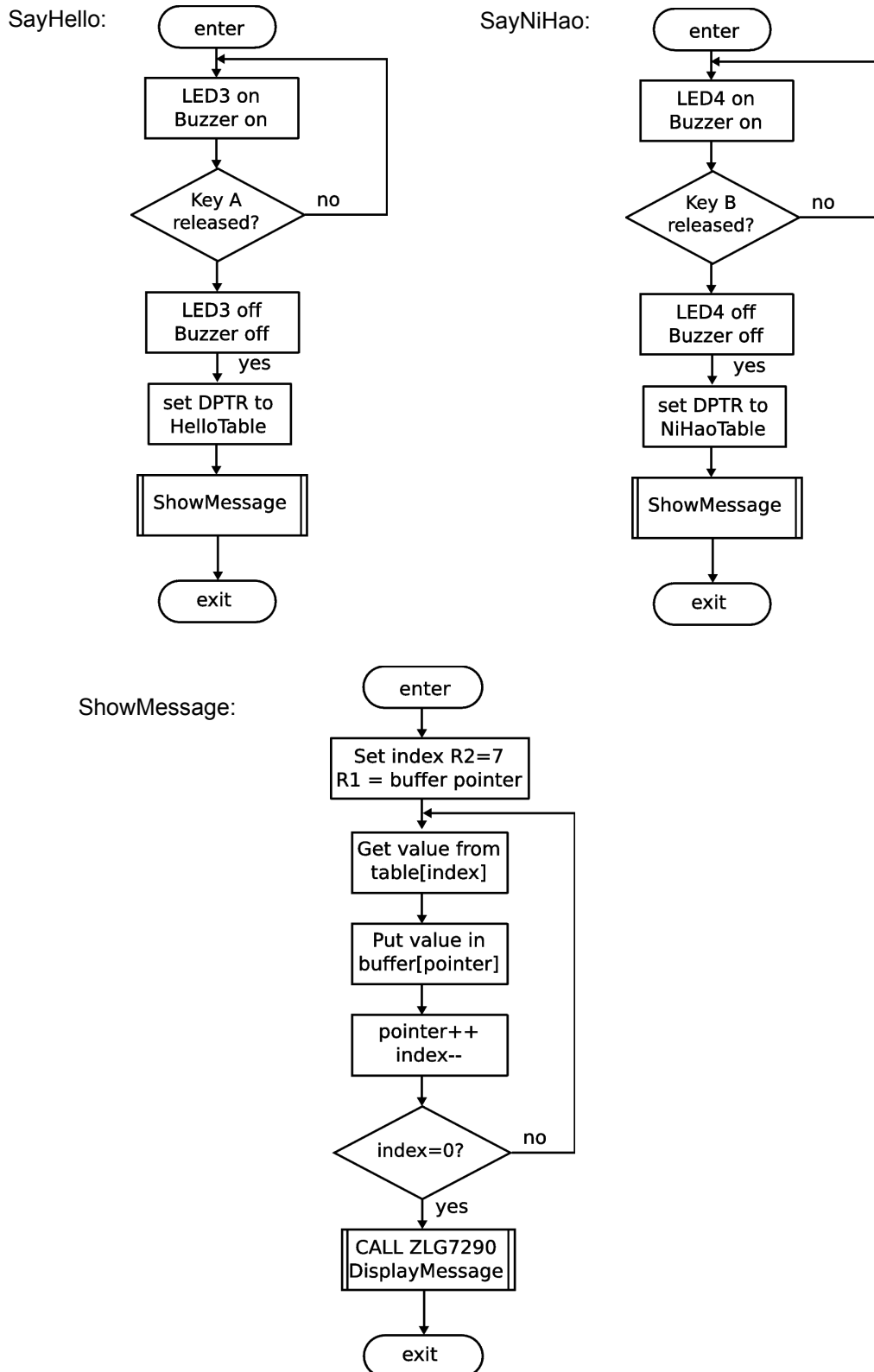


Figure 29: SayHello (top left) and SayNiHao (top right) both call Show Message (bottom)

The SayHello and SayNiHao functions use the ZLG7290_driver_10.asm module _ZLG_DisplayMessage subroutine to output messages on the display via the code at ShowMessage.

The KeyPadPress function reads a hex value from the keypad and inserts it in the rightmost digit on the display, shifting all the other characters to the left.

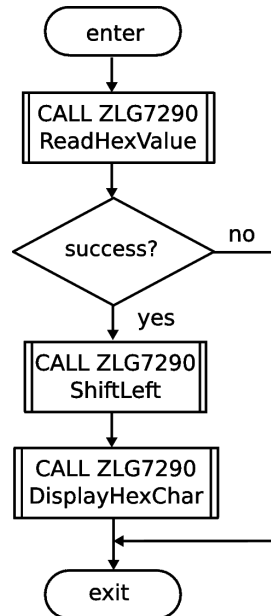


Figure 30: KeyPadPress reads the keypad and outputs the hex character on the display

For further explanation of how the code works, please refer to the comments in the code modules themselves.

History:

Chris Harte 5th May 2007

1st Revision: updated for new board details - Chris Harte 8th May 2007

2nd Revision: updated for new driver details – Chris Harte 30th April 2008

3rd Revision: updated for new driver details – Chris Harte 5th April 2009

4th Revision: updated for EBU5475 – Chris Harte 14th May 2010