Introduction:

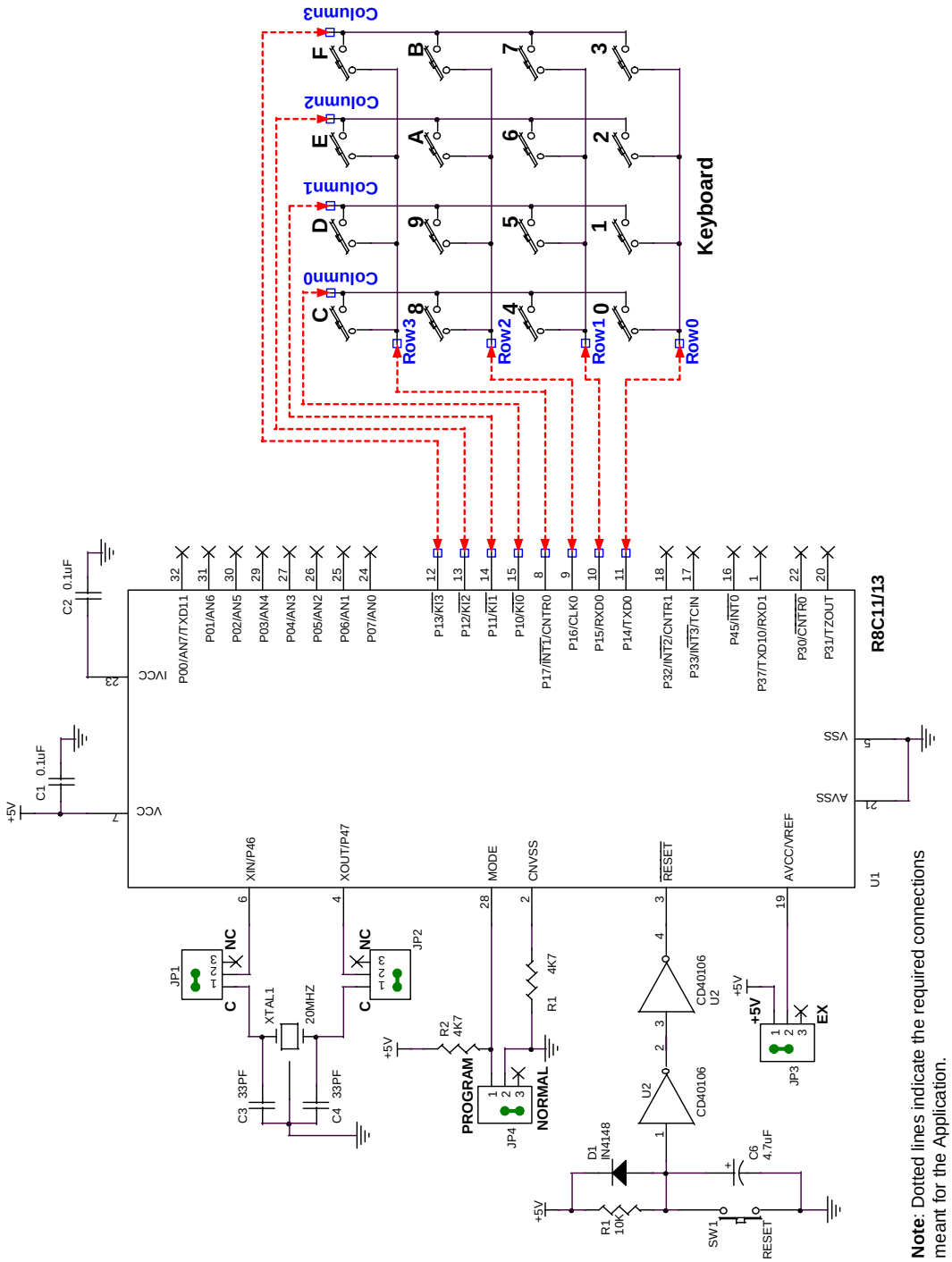
Sixteen keys are connected in 4 X 4 matrix format. A One Millisecond interrupt is generated using Timer X and the keyboard is scanned in the interrupt and the pressed key is identified. The pressed key is displayed in 2 digit 7-segment display.

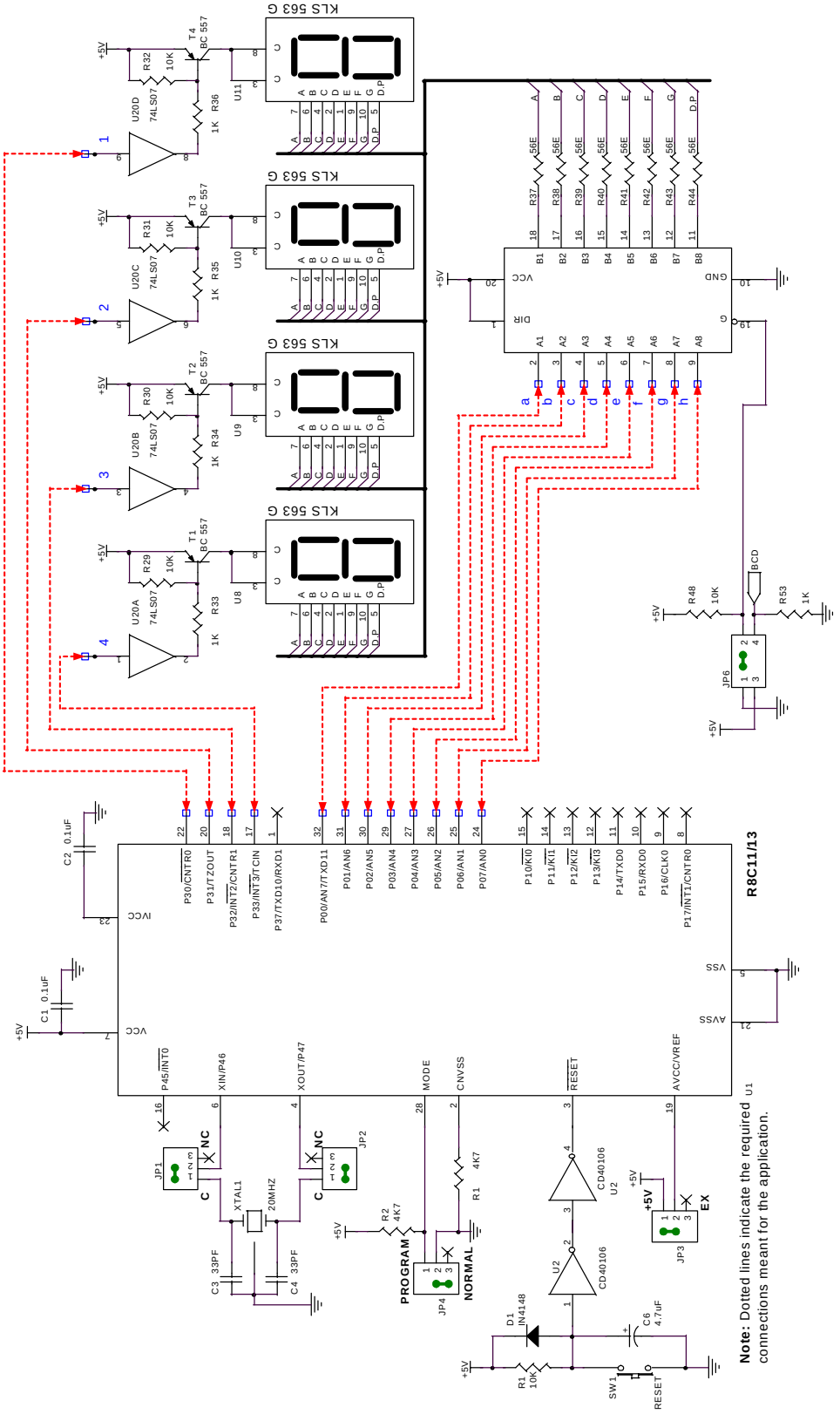
Demo Hardware:

In this demo, the row lines of the keyboard are connected to the port lines P14 to P17 and the column lines are interfaced with the port lines P10 to P13. Hence row lines can be used as output lines and column lines are defined as input lines.

2 digits of seven segment LED displays are connected in multiplexed mode with seven segment data input. The seven segments are connected to the port P0 sequentially and the Port lines P30 and P31 are conneced to the digit selection lines.

Circuit Connection:




R8C11/13
U1
Note: Dotted lines indicate the required connections meant for the application.
PROGRAM
NORMAL
XTAL1
20MHZ
CD40106
KLS 563 G
BCD

Verify the Connections:

- **For Keyboard**

Row0 -> P17
Row1 -> P16
Row2 -> P15
Row3 -> P14

Column0 -> P10
Column1 -> P11
Column2 -> P12
Column3 -> P13

- **For 7-segment Display**

Port lines	**Display Lines**
P00	Segment a
P01	Segment b
P02	Segment c
P03	Segment d
P04	Segment e
P05	Segment f
P06	Segment g
P07	Segment dp
P30	Digit Selection Control for digit 1
P31	Digit Selection Control for digit 2

Functional Description:

The keys are connected with the microcontroller in many ways. The most simple way is to assign a port line to each of these switches to sense the closure of the key. Because of this arrangement, this interface requires as many port lines as the keys. When more keys are required for the application, this interface arrangement demands more port lines.

To manage this resource drain, the keys can be arranged in a matrix form, where each of these keys is connected at the intersection of row and column lines of the matrix.

These row and column lines of the key matrix are connected with the port lines. When a key is pressed, it connects one row with a column and this connection is sensed by the controller. In this way, the port line requirement is kept as the lowest level. For a 16 key matrix arranged in a 4X 4 matrix requires only 8 port lines for usage.

Similarly, for a key assembly of 64 keys arranged in a 8 X 8 matrix, only 16 port lines are required during interfacing.

In this application, an interrupt is used to scan the keyboard for the key closure. A timer is used to generate a periodical time delay of 1 millisecond to find out the key closure. When servicing the interrupt routine, the key scanning task is taking place.

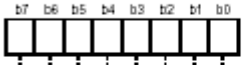
Registers used:

PD1 - Port 1 Direction Register
P1 - Port 1 Data Register
TZPR - 8-bit Timer Y Primary Register
TYZMR - Timer Y,Z Mode Register
TCSS - Timer Count Source Setting Register
TXIC - Timer X Interrupt control Register.

Port 1 Direction Register

Port Pi direction register (i=0, 1, 3, 4)[1, 2, 3]

Symbol	Address	After reset
PD0	00E2₁₆	00₁₆
PD1	00E3₁₆	00₁₆
PD3	00E7₁₆	00₁₆
PD4	00EA₁₆	00₁₆



Bit symbol	Bit name	Function	RW
PDi_0	Port Pi0 direction bit	0 : Input mode (Functions as an input port) 1 : Output mode (Functions as an output port)	RW
PDi_1	Port Pi1 direction bit		RW
PDi_2	Port Pi2 direction bit		RW
PDi_3	Port Pi3 direction bit		RW
PDi_4	Port Pi4 direction bit		RW
PDi_5	Port Pi5 direction bit		RW
PDi_6	Port Pi6 direction bit		RW
PDi_7	Port Pi7 direction bit		RW

Notes:
1. The PD0 register must be written to by the next instruction after setting the PRC2 bit in the PRCR register to "1" (write enabled).
2. Nothing is assigned to the PD3_4 to PD3_6 bits in the PD3 register. When writing to the PD3_4 to PD3_6 bits, write "0" (input mode). When read, its content is indeterminate.
3. Nothing is assigned to the PD4_0 to PD4_4, PD4_6 and PD4_7 bits in the PD4 register. When writing to the PD4_0 to PD4_4, PD4_6 and PD4_7 bits, write "0" (input mode). When read, its content is indeterminate.

To set a port line as output line, load the corresponding bit of the direction register with a value of 1 level and for the input line the value should be 0 level. After the reset, all the port lines are set to input mode, which indicates that all direction registers are already set with a value 0.

Port 1 Data Register

Port Pi register (i=0, 1, 3, 4)[1, 2]

Symbol	Address	After reset
P0	00E0₁₆	Indeterminate
P1	00E1₁₆	Indeterminate
P3	00E5₁₆	Indeterminate
P4	00E8₁₆	Indeterminate

Bit symbol	Bit name	Function	RW
Pi_0	Port Pi0 bit	The pin level on any I/O port which is set for input mode can be read by reading the corresponding bit in this register. The pin level on any I/O port which is set for output mode can be controlled by writing to the corresponding bit in this register 0 : "L" level 1 : "H" level (i = 0, 1, 3, 4)	RW
Pi_1	Port Pi1 bit		RW
Pi_2	Port Pi2 bit		RW
Pi_3	Port Pi3 bit		RW
Pi_4	Port Pi4 bit		RW
Pi_5	Port Pi5 bit		RW
Pi_6	Port Pi6 bit		RW
Pi_7	Port Pi7 bit		RW

Notes:
1. Nothing is assigned to the P3_4 to P3_6 bits in the P3 register. When writing to the P3_4 to P3_6 bits, write "0" ("L" level). When read, its content is indeterminate.
2. Nothing is assigned to the P4_0 to P4_4 bits in the P4 register. When writing to the P4_0 to P4_4 bits, write "0" ("L" level). When read, its content is indeterminate.

The Port data register keeps data meant for both input and output operations. During output operation, the corresponding output data should be written in this register. Similarly, during input operations, reading this register is sufficient to get the data from that particular port.

Timer X Mode Register

Timer X mode register

b7	b6	b5	b4	b3	b2	b1	b0

Symbol: TXMR — Address: 008B16 — After reset: 0016

Bit symbol	Bit name	Function	RW
TXMOD0	Operation mode select bit 0, 1	b1 b0 0 0 : Timer mode or pulse period measurement mode 0 1 : Pulse output mode 1 0 : Event counter mode 1 1 : Pulse width measurement mode	RW
TXMOD1			RW
R0EDG	INT1/CNTR0 polarity switching bit[1]	Function varies with each operation mode	RW
TXS	Timer X count start flag	0 : Stops counting 1 : Starts counting	RW
TXOCNT	P30/CNTR0 select bit	Function varies with each operation mode	RW
TXMOD2	Operation mode select bit 2	0 : Except in pulse period measurement mode 1 : Pulse period measurement mode	RW
TXEDG	Active edge reception flag	Function varies depending on each operation mode.	RW
TXUND	Timer X under flow flag	Function varies depending on each operation mode.	RW

Notes:
1. The IR bit in the INT1IC register may be set to "1" (interrupt requested) when the R0EDG bit is rewritten. Refer to the paragraph 19.2.5 "Changing Interrupt Factor" in the Usage Notes Reference Book.

Set a value of 0x00 in the timer X mode register to select timer mode operation for Timer X.

Timer Count Source Setting Register

Timer count source setting register

b7	b6	b5	b4	b3	b2	b1	b0
0	0						

Symbol: TCSS — Address: 008E16 — After reset: 0016

Bit symbol	Bit name	Function	RW
TXCK0	Timer X count source select bit[1]	b1 b0 0 0 : f1 0 1 : f8 1 0 : f32 1 1 : f2	RW
TXCK1			RW
TYCK0	Timer Y count source select bit[1]	b3 b2 0 0 : f1 0 1 : f8 1 0 : fRING 1 1 : Selects input from CNTR1 pin	RW
TYCK1			RW
TZCK0	Timer Z count source select bit[1]	b5 b4 0 0 : f1 0 1 : f8 1 0 : Selects Timer Y underflow 1 1 : f2	RW
TZCK1			RW
— (b7-b6)	Reserved bit	Must be set to "0"	RW

Clock for Timer X is selected using TCSS register.
Set TXCK0 and TXCK1 to 0 to select f1 for Timer X

Timer X Interrupt Control Register

Bit symbol	Bit name	Function	RW
ILVL0	Interrupt priority level select bit	b2 b1 b0 0 0 0 : Level 0 (Interrupt disabled) 0 0 1 : Level 1 0 1 0 : Level 2 0 1 1 : Level 3 1 0 0 : Level 4 1 0 1 : Level 5 1 1 0 : Level 6 1 1 1 : Level 7	RW
ILVL1			RW
ILVL2			RW
IR	Interrupt request bit	0 : Interrupt not requested 1 : Interrupt requested	RW1
— (b7-b4)	Nothing is assigned. When write, set to "0". When read, its content is indeterminate.		—

Set the priority level 2 for Timer X.

Software Description:

Timer X is used to generate a one millisecond delay. The clock frequency for the Timer X is fosc.

The timer is initialized to generate an interrupt for every millisecond. The PREX register is loaded with 99. The timer TX is loaded with a value 199.

The Time delay generated is calculated as

$$= \frac{(n+1)\ (m+1)}{Fosc}$$

n - Prescalar value

m - Primary register

$$= \frac{(100) * (200)}{20000000}$$

= 1 millisecond.

Timer X overflows for every millisecond and an interrupt is generated and also the timer registers are reloaded with the initial set values. For every millisecond lapsed, the interrupt is serviced and the keyboard is scanned. If any key is pressed, the key value will be stored in a variable and the control returns back to main program.

Timer X Interrupt Service Routine:

The column lines are provided with individual pull-up resistors. When they are read, column data may contain high level from all the columns.

The rows are activated one by one and the selected row is supplied with a low level. Then the column data is read. If any key is pressed then that particular column will be at low level. Thus the pressed key is identified and **KeyboardFlag** is set. The key code of the pressed key is stored in the **KeyboardCode** variable.

If no key has been pressed in that row, the next row is activated and checked for any keypress. This process is repeated for the remaining rows.

Using the routine

The keyboard scanning routine provides two variables for processing. The keyboard scanning is done in the Timer X Interrupt service routine. The user application checks for level 1 in the variable **KeyboardFlag**. If this flag is at 1 level, then the keycode is read from the variable **KeyboardCode** after clearing **KeyboardFlag** variable.

Another routine, **"WaitForKeypress",** when used, can wait for a key press and return the key value when the key press is sensed.

One more option is to read the status of the keyboard (Whether any key is pressed or not) using the routine **"ReadKeyboardStatus"**. After this routine, the pressed key value is read using **"ReadKeyCode"**.

The files used in this module are listed below:

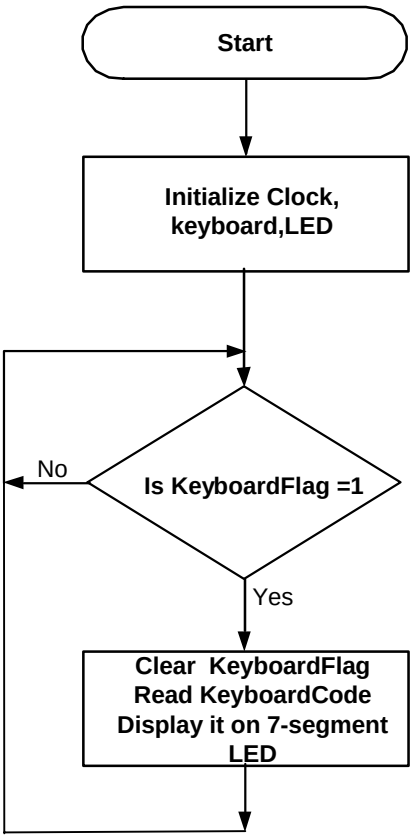
Files	*Description*
Demo4.C	Keyboard routines to initialize Keyboard, WaitForKeyPress, ReadKeyboardStatus, Display routines for 7-segment display etc.
Keyboard.H	Declarations of functions in keybaord
LEDDisplay.H	Declarations of functions in 7-segement display

The main file Demo4.C has the following routines.

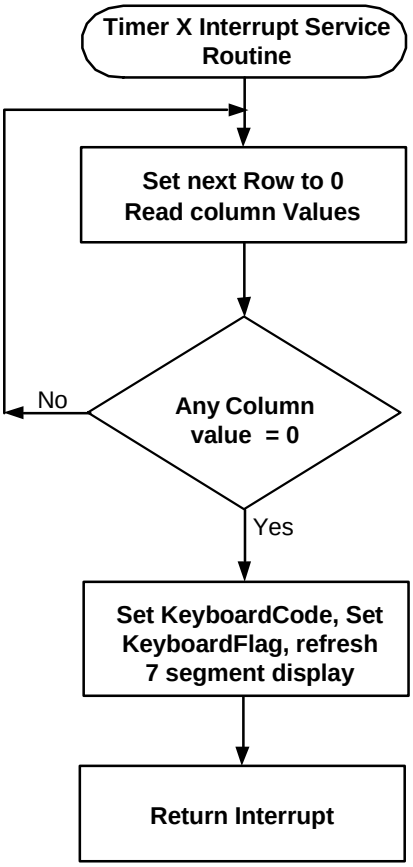
Files	*Description*
Main	Reads the keyboard and displays it on the seven segment LED.
WaitForKeyPress	Waits for a key press and returns the key code of the pressed key. **Input**: None. **Output** : Key Code.
ReadKeyCode	Reads and returns the last pressed key code without waiting for a key press. **Input**: None. **Output** : Key Code.
InitializeKeyboard	Initializes the I/O lines used by keyboard and Initializes Timer X Interrupt
LEDInterrupt	Interrupt service routine Timer X interrupt. Refreshing the 7 segment display and scanning the keyboard.

Program Flow.

Flow for WaitForKeypress Routine:



Flow for Key Input Interrupt Service Routine:



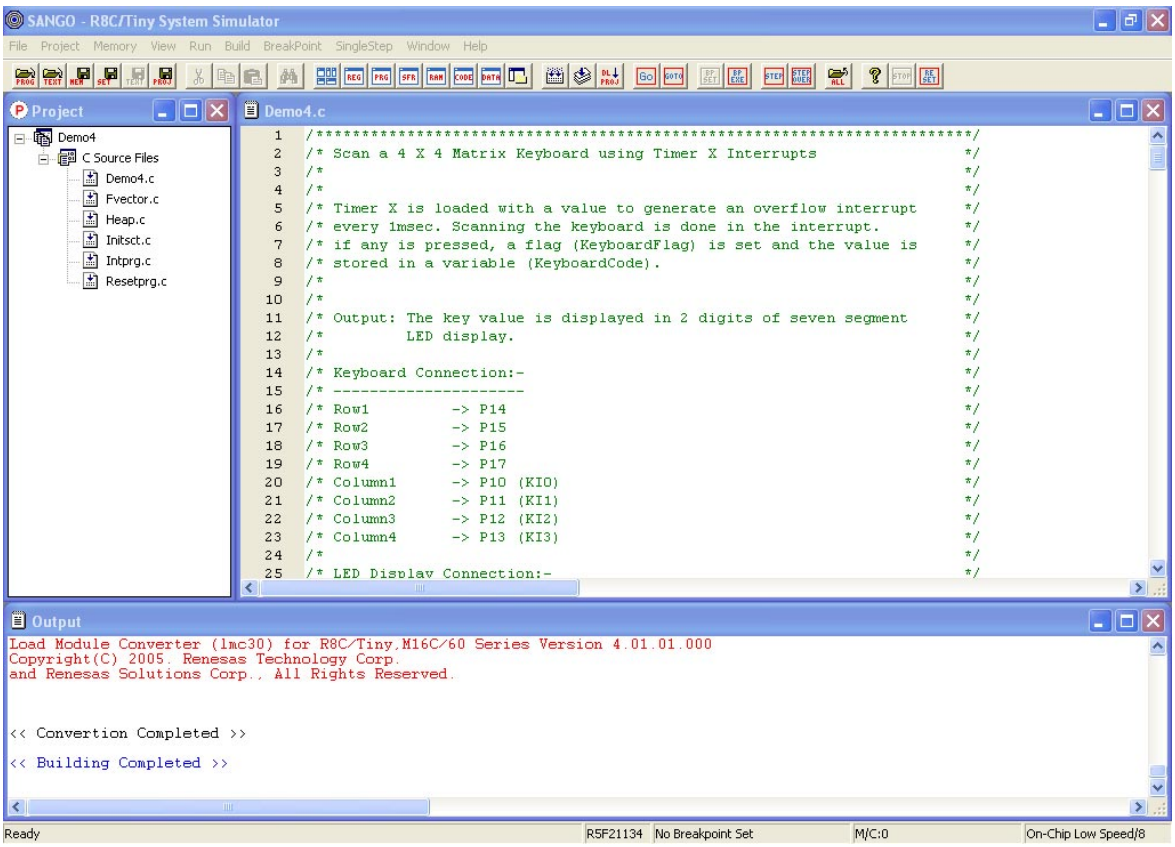
Execute Application:

The pressed keyvalue is displayed on the 7-segment display.

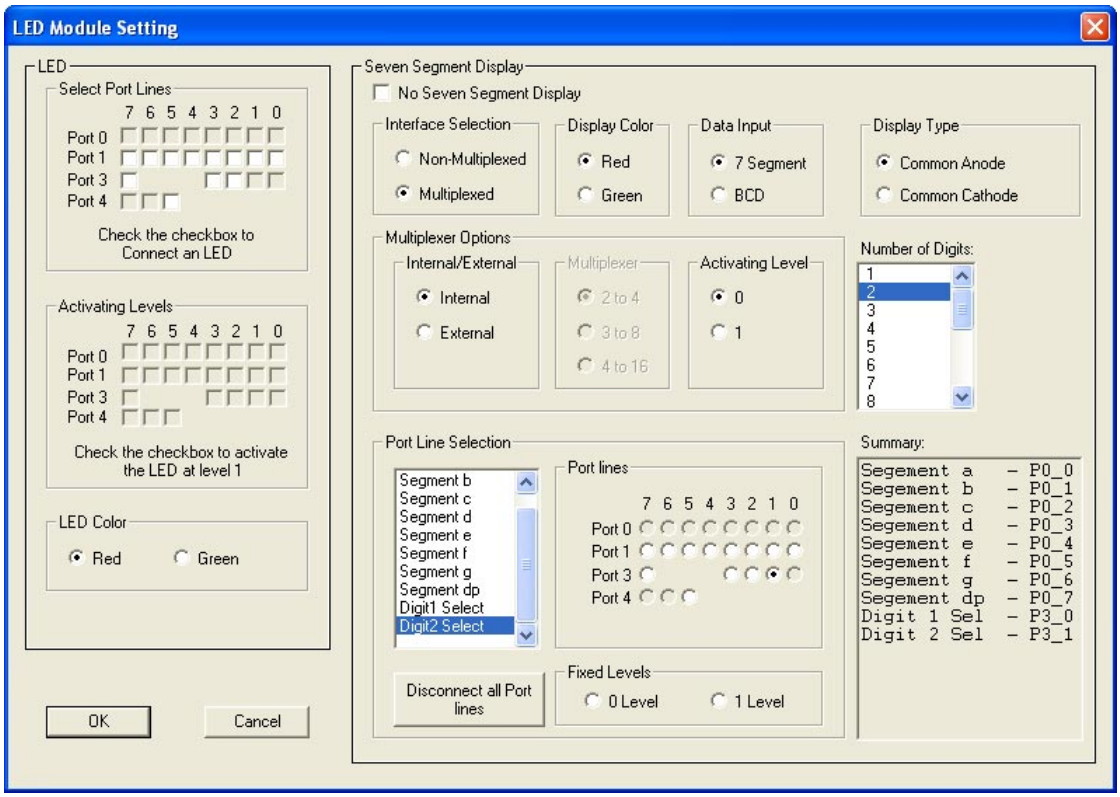
Use Topview Simulator to Verify the Design.

Open the project Demo4 in the R8C/Tiny System Simulator using **Open Project** option from **Project menu**. The project window opens up along with the Demo4.c file. Use **Build** option from **Build** menu to compile the project. An output window captures the compiler ouput.

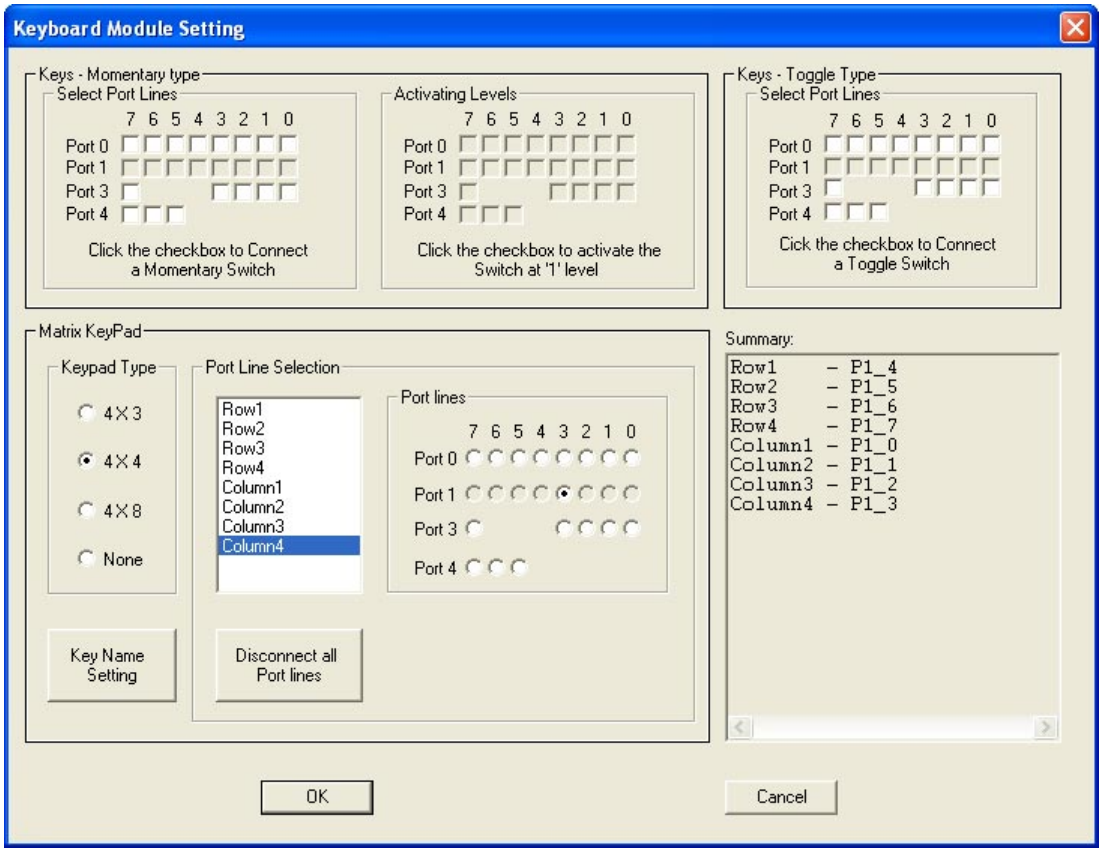
Use **Project** -> **Download Project** from main menu to download the Demo4.mot file into the simulator's memory for simulation.



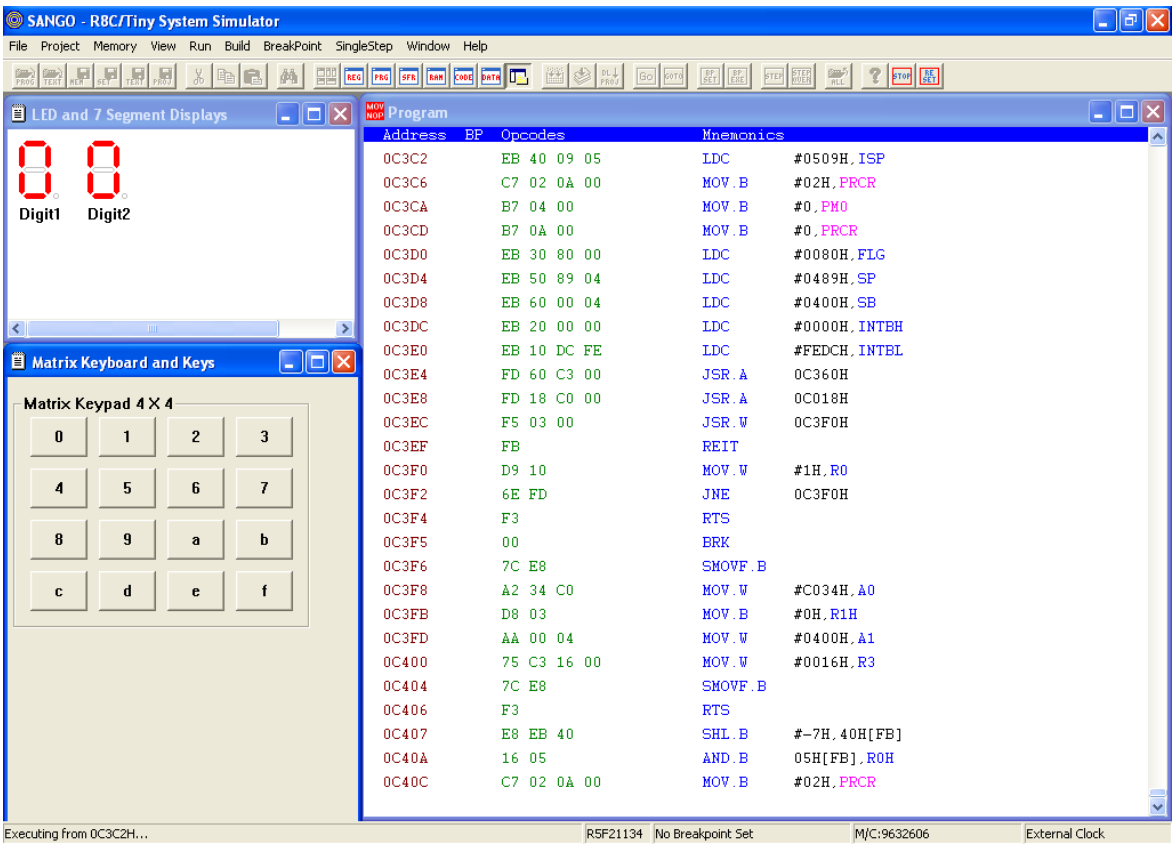
Open the LED Module settings window and do the settings to the 7-segment LED module as shown. Connect 7 segments of the display to the port lines P00 to P07 and the 2 digit selection lines to P30 and P31 respectively using radio buttons.



Open the Keybaord Module settings window and do the settings. Connect the rows to P14 to P17 and columns to P10 to P13.



Then open the **LED window** usign the option **View** -> **External Modules** -> **LED** as shown below and the Program Window. Open the **Keybaord window** usign the option **View** -> **External Modules** -> **Keyboard** as shown below and the Program Window.



Run the program using **Go** from the **Run** menu. The program will display the pressed keyvalue on the 7-segment display.