Configuration and Programming of Embedded Systems:

Assessment no* 1

[15]

Student no: 18026826 Name: Benjamin Samuel Edwards

Index:

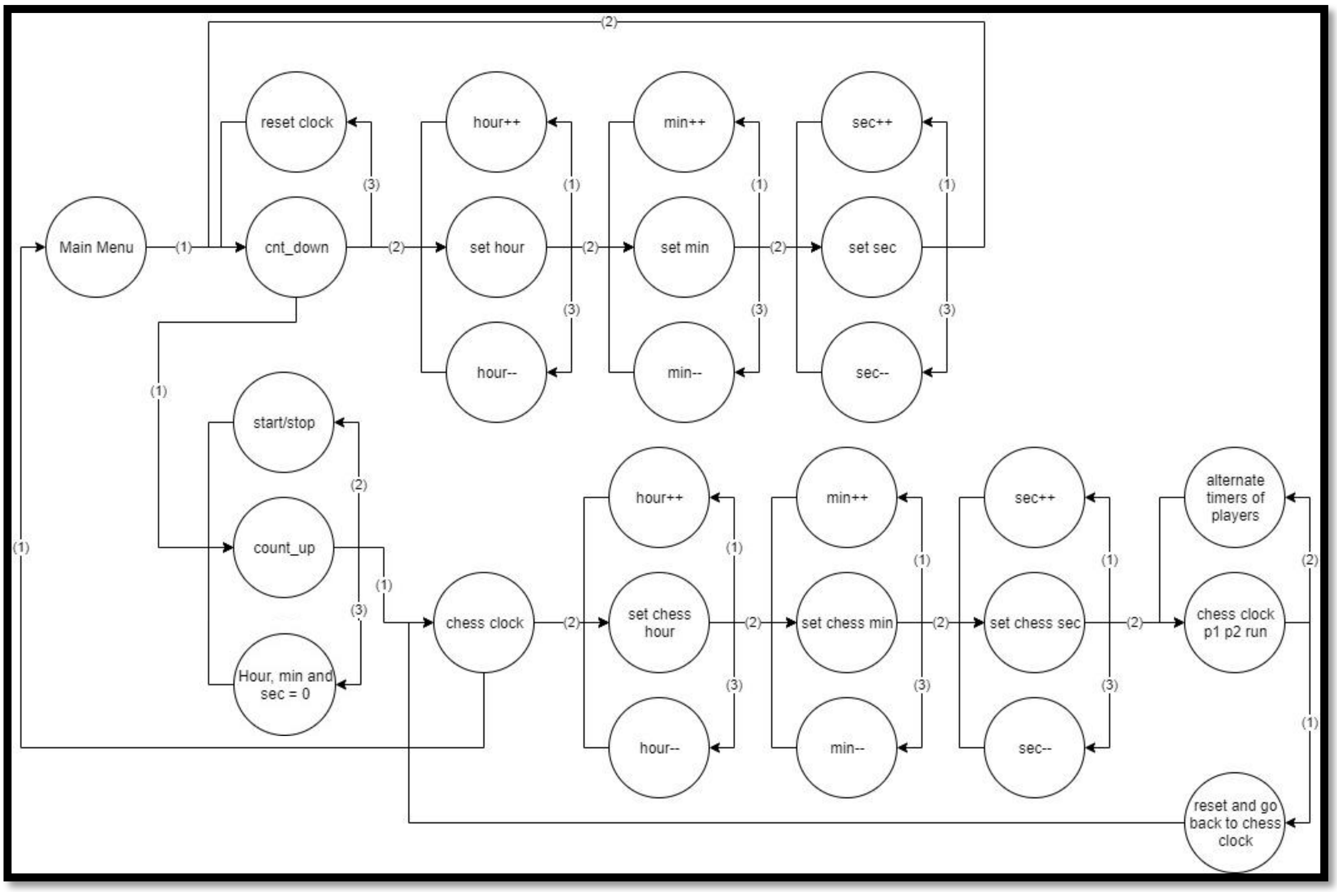
Introduction:

For this assignment I have been tasked with programming a Renesas RX63N board using a FAT development board that has a port for the RX63N programmable microcontroller. The task given is to program a Digital clock functions using a menu system to navigate through the application with. A stopwatch, a count-down timer and a two-player chess clock are the specific applications that are required for this assignment. For this the RX63N was chosen for this task was due to its two flash memory blocks for the program memory and the data memory. This allowed for faster processing time without needing to process the program and data memory one after the other, instead just processing them simultaneously. The RX63N also has a built in ADC that can be used to convert an analogue signal into a digital format. This is set to a pin using a peripheral instruction that will active the voltage comparator before being cleared automatically after the completion of the conversion.

This task includes the specific task of programming a series of time keeping functions that the user can switch between using a series of three buttons on a keypad. The functions are listed as…

- a countdown timer, or a stopwatch, allowing the user to…
  - press button 1 to cycle to the next timer function in the queue.
  - Button 2 will start the time set function allowing the user to add and take away a unit to the hour value using the 1st and 3rd buttons, using the 2nd button to move onto the min and sec, repeating the process respectively.
  - The third button will also reset the counter back to zero.
  - There must also be an alarm for when the countdown reaches zero, which can be done by having an LED flash for a period of time.
- A count-up timer allows the user to…
  - press button 1 to cycle to the next timer function in the queue.
  - Press button 2 to start and stop the count-up sequence
  - Press button 3 to reset the counter back to zero
- A chess clock will allow the user to
  - press button 1 to cycle to the next timer function in the queue.
  - Press button 2 to set the countdown time similarly to the first function
  - Press button 3 to stop the one timer and start the other simultaneously, setting an alarm off when one of the timers reaches zero.

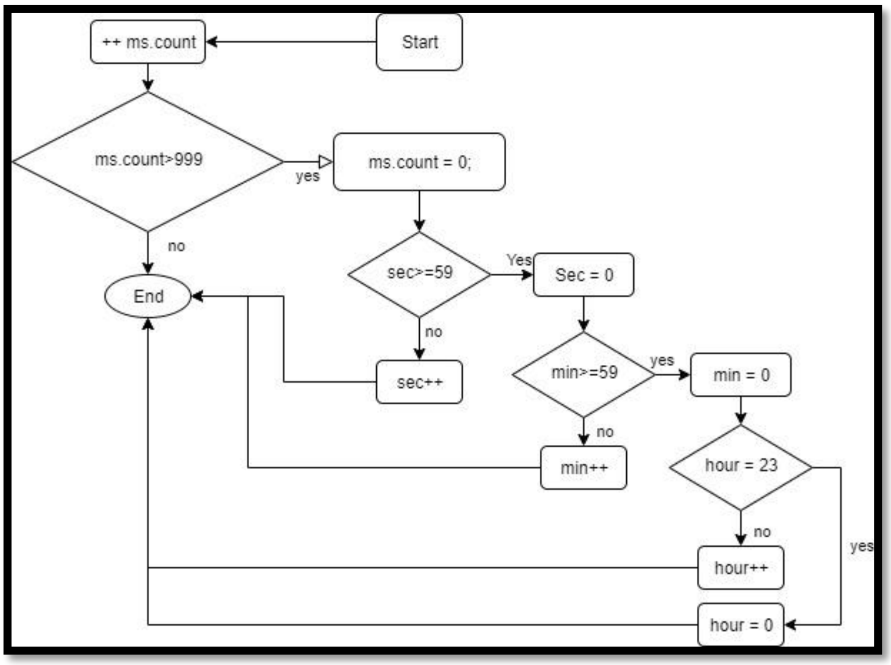
State Machine:



Looking at this flow chart, it is shown that the program will start with the main menu window. The user can move to the count-down function by pressing button 1 on the keypad. Now the user will be able to decide on the time that is wanted to be counted down from by pressing button 2 to cycle between hour, minutes and seconds, using buttons 1 and 3 to increment or decrement a unit. The way out of the count-down function is to press button 1 in the count-down menu, leading the user to the count-up function now instead. By using button 2 the user could start and stop the count-up of the stopwatch using an interrupt flag and buy using the third button the user could then clear the count-up time to reset the clock. Pressing button 1 would take the user to the chess clock menu function. Button 2 will allow the user to set the time similarly to the count-down function using the same buttons. After setting the time the clocks begin and on players clock starts while the other stays the same until the player presses button 2 to change the turn and start the opponent's clock. Button 1 will reset the clocks and return to the chess clock menu. From the chess clock menu, the user can press button 1 to return to the main menu to complete the menu cycle.

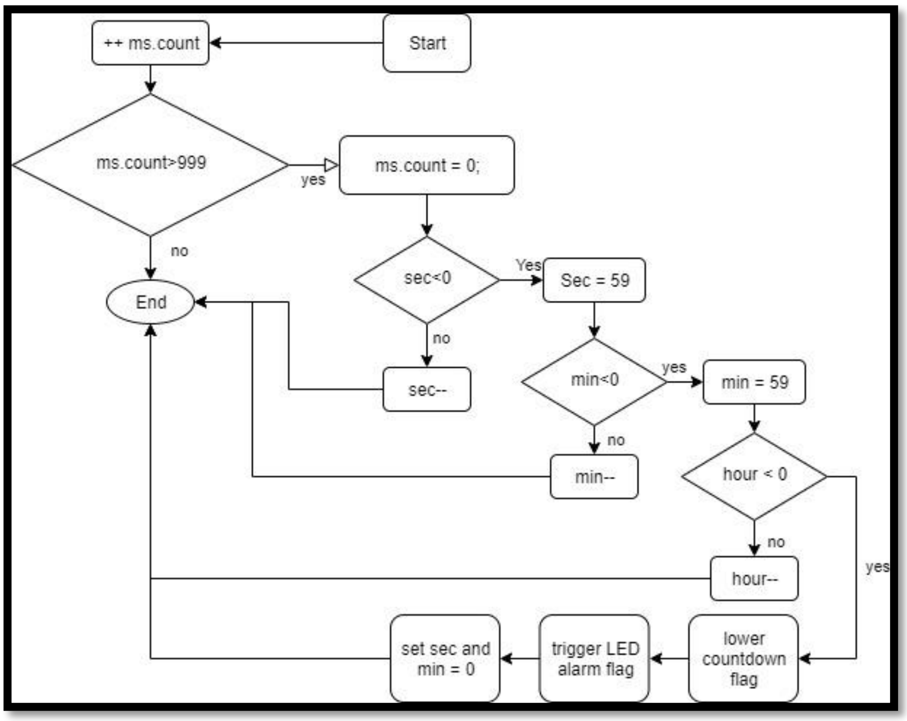
Flow Charts:

Count-up timer flow chart:



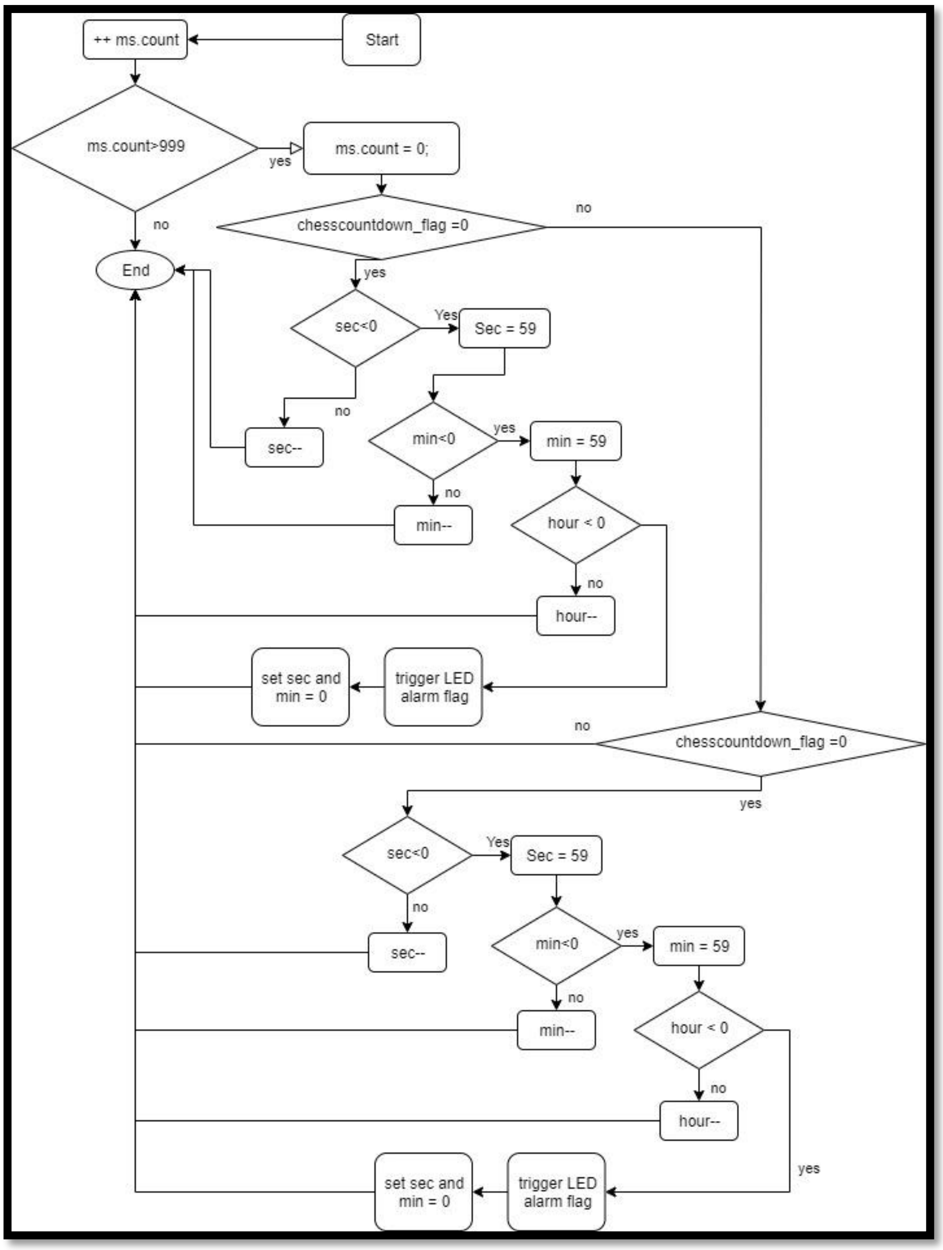
Using this simple count-up timer program, the millisecond count will increment 1000 times before calculating a second and executing the following code. The second would increment to above 59 and would then be checked to see if it was over that specified value of 59. If yes then second would equal to zero and would move onto the minute step, else second increments by 1 each cycle. This would continue until hour reaches over 23 and would then be reset to 0 again.

Count-down timer flow chart:



The count down works similarly to the count-up flow chart. However, instead of checking to see if its over 59 it checks to see if the number if under 0. If no then an increment occurs but if yes, then second gets set to equal 59 before moving on to the minute step. When the count-down has been completed then the alarm flag will be raised, and the countdown flag will be lowered. This will trigger a flashing LED to signify the end of the countdown. The seconds and minutes values will also be reset to zero to stop them from starting from 59:59 again.

Chess clock timer flow chart:



The chess clock works exactly like the countdown function but instead has a countdown segment for each player clock. There will be a flag to signify which player whose current turn it is, which will signal one of the players clocks to start counting down. When an interrupt has been flagged then the countdown of the clocks will alternate between the players until one of the clocks reaches 0 which will trigger an alarm flag and a message will be displayed on the LCD screen saying which player had won.

Explanation of software and hardware:

Initialisation: -

For the initialisation stage, I had to create a series of strings that would display information to the user that would include instruction, the counter and information about what's currently on screen. For this I used the char array to hold all the characters to create a string, using only the memory necessary to hold this information. Not wasting any memory in the process by using more than what was needed since.

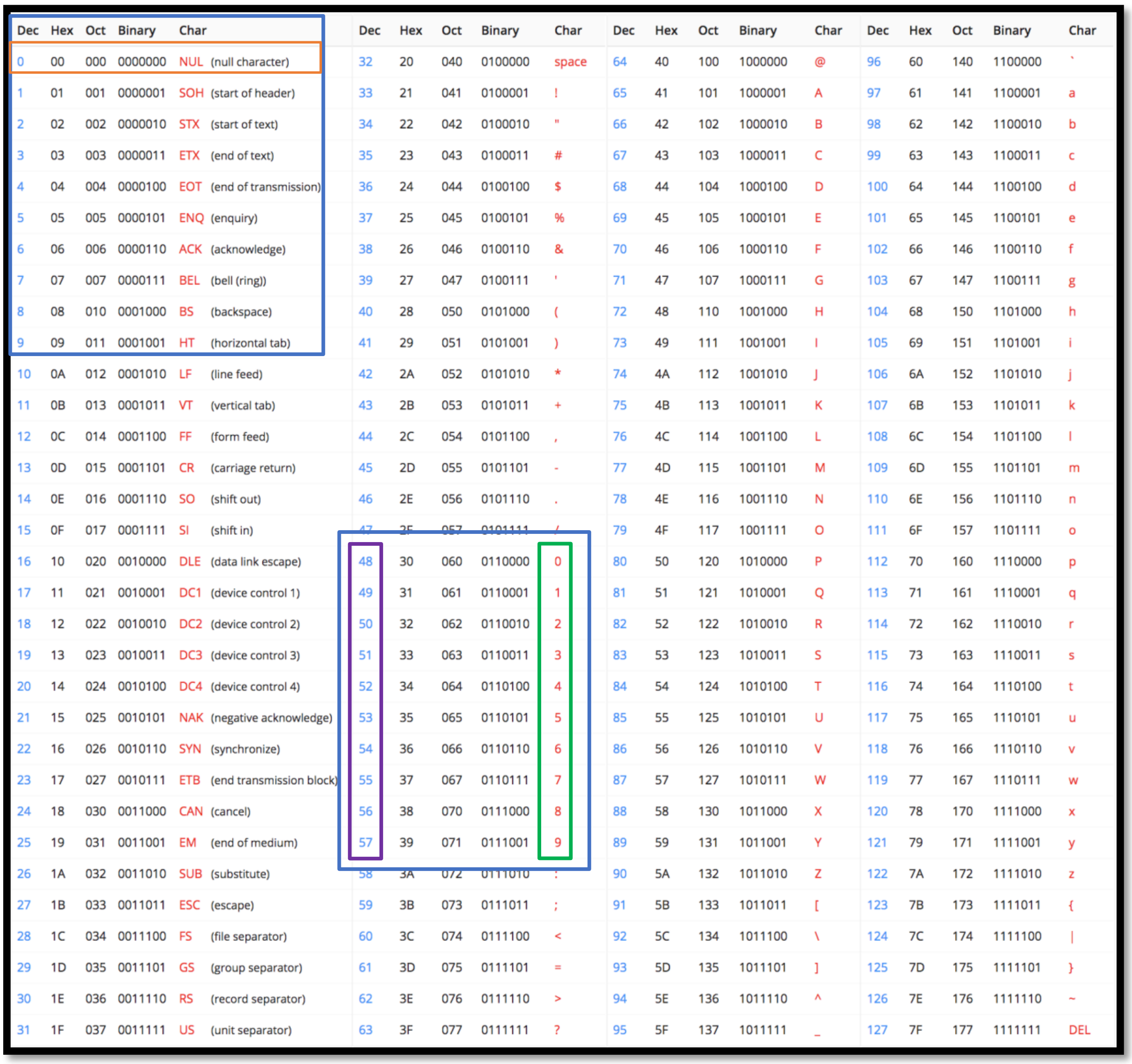
A series of flags was also needed to signify when an event should occur. To do this I used an ISR (interrupt service routine) over a polling method due to its processing efficiency compared to the simpler, but less efficient, polling loop statements. The difference between the two is quite simple when using the boiling kettle example. Using the polling method, the user would put the kettle to boil and continue to stare at it for 5 minutes while also doing nothing else besides checking if the water has boiled yet. With an interrupt, the user can put the kettle on and go about completing another task in the meantime. When the kettle has boiled completely, a whistling sound will be emitted. Ironically, given the subject, simulating the flag being raised, signalling the user to stop what they are doing and to take the kettle off the boil and pour the hot water in a coffee filter or a mug with a tea bag. After completing this interrupt task, the user will then resume with the previous task where they left off.

Switch Statements: -

After implemented these interrupts into the program, a series of Switch statements would be used, which would be called when a key on the built-in keypad is pressed. Depending on the current case the program is on and the key number pressed, a certain action will be taken and often would task the program to switch to a new specified case. Using this Switch statement method, it will be able to use only three buttons for the whole program, while also using a single key for preforming multiple actions by using them in multiple cases.

ASCII code: -

To display the clock on the LCD display it had to first create a char array containing the string "00:00:00". This will be displayed on the LCD screen. In the 'd_stop watch.h' file the structure 'clock_t' is created to store the hour, min and sec variables as an unsigned char. To initialise the clock string to the 'clock_t' structure, using the line 'clock_t countup_tim;' to create a structure under that name. the hour, min and sec values are initialised to zero at the setup stage by typing the structure name and then 'dot' the variable name, (countup_tim.hour = 0;) ect.
Next, the program will need to convert the characters of the '00:00:00' string to an ASCII format so they can be used to display the time in a numerical form. The way this works is that by using MOD to find the remainder of the 'ones' unit of the 'countup_tim.sec' structure variable, then add 48 to that result. That character in that array will display the number that is associated with that ASCII number. Using the character 3 as an example for this, dividing the number 3 with 10 would give the remainder of 3. The 3 being an ASCII number. Since the ASCII of 3 is ETX (end of text) then it will display a corrupted looking character on the display. To change this the ASCII code for the number 3 as a character, which is 51. So, by adding the ASCII code for 0, which is 48, to the remainder value that was given before, then the display will show the number '3' on the display in the clock_dsp[7] array char.



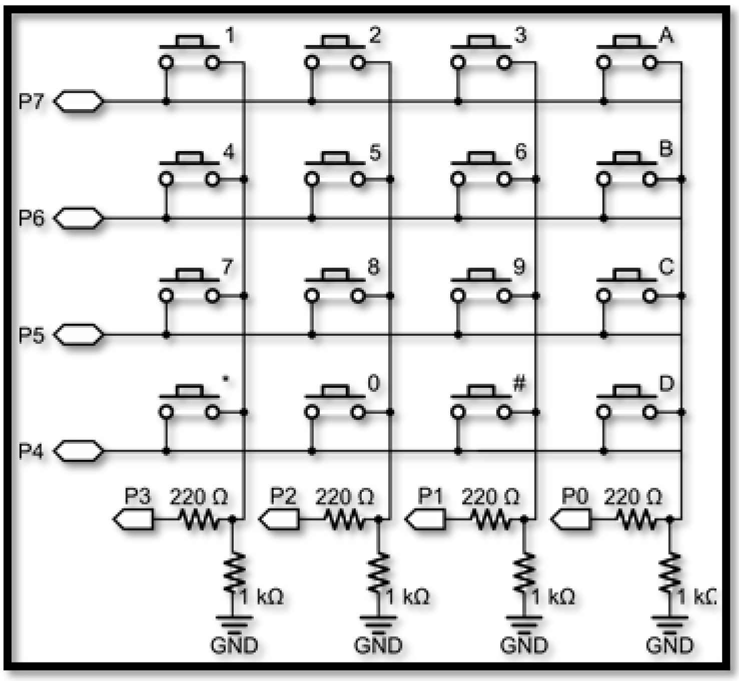
Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char
0	00	000	0000000	NUL (null character)	32	20	040	0100000	space	64	40	100	1000000	@	96	60	140	1100000	`
1	01	001	0000001	SOH (start of header)	33	21	041	0100001	!	65	41	101	1000001	A	97	61	141	1100001	a
2	02	002	0000010	STX (start of text)	34	22	042	0100010	"	66	42	102	1000010	B	98	62	142	1100010	b
3	03	003	0000011	ETX (end of text)	35	23	043	0100011	#	67	43	103	1000011	C	99	63	143	1100011	c
4	04	004	0000100	EOT (end of transmission)	36	24	044	0100100	$	68	44	104	1000100	D	100	64	144	1100100	d
5	05	005	0000101	ENQ (enquiry)	37	25	045	0100101	%	69	45	105	1000101	E	101	65	145	1100101	e
6	06	006	0000110	ACK (acknowledge)	38	26	046	0100110	&	70	46	106	1000110	F	102	66	146	1100110	f
7	07	007	0000111	BEL (bell (ring))	39	27	047	0100111	'	71	47	107	1000111	G	103	67	147	1100111	g
8	08	010	0001000	BS (backspace)	40	28	050	0101000	(	72	48	110	1001000	H	104	68	150	1101000	h
9	09	011	0001001	HT (horizontal tab)	41	29	051	0101001	)	73	49	111	1001001	I	105	69	151	1101001	i
10	0A	012	0001010	LF (line feed)	42	2A	052	0101010	*	74	4A	112	1001010	J	106	6A	152	1101010	j
11	0B	013	0001011	VT (vertical tab)	43	2B	053	0101011	+	75	4B	113	1001011	K	107	6B	153	1101011	k
12	0C	014	0001100	FF (form feed)	44	2C	054	0101100	,	76	4C	114	1001100	L	108	6C	154	1101100	l
13	0D	015	0001101	CR (carriage return)	45	2D	055	0101101	-	77	4D	115	1001101	M	109	6D	155	1101101	m
14	0E	016	0001110	SO (shift out)	46	2E	056	0101110	.	78	4E	116	1001110	N	110	6E	156	1101110	n
15	0F	017	0001111	SI (shift in)	47	2F	057	0101111	/	79	4F	117	1001111	O	111	6F	157	1101111	o
16	10	020	0010000	DLE (data link escape)	48	30	060	0110000	0	80	50	120	1010000	P	112	70	160	1110000	p
17	11	021	0010001	DC1 (device control 1)	49	31	061	0110001	1	81	51	121	1010001	Q	113	71	161	1110001	q
18	12	022	0010010	DC2 (device control 2)	50	32	062	0110010	2	82	52	122	1010010	R	114	72	162	1110010	r
19	13	023	0010011	DC3 (device control 3)	51	33	063	0110011	3	83	53	123	1010011	S	115	73	163	1110011	s
20	14	024	0010100	DC4 (device control 4)	52	34	064	0110100	4	84	54	124	1010100	T	116	74	164	1110100	t
21	15	025	0010101	NAK (negative acknowledge)	53	35	065	0110101	5	85	55	125	1010101	U	117	75	165	1110101	u
22	16	026	0010110	SYN (synchronize)	54	36	066	0110110	6	86	56	126	1010110	V	118	76	166	1110110	v
23	17	027	0010111	ETB (end transmission block)	55	37	067	0110111	7	87	57	127	1010111	W	119	77	167	1110111	w
24	18	030	0011000	CAN (cancel)	56	38	070	0111000	8	88	58	130	1011000	X	120	78	170	1111000	x
25	19	031	0011001	EM (end of medium)	57	39	071	0111001	9	89	59	131	1011001	Y	121	79	171	1111001	y
26	1A	032	0011010	SUB (substitute)	58	3A	072	0111010	:	90	5A	132	1011010	Z	122	7A	172	1111010	z
27	1B	033	0011011	ESC (escape)	59	3B	073	0111011	;	91	5B	133	1011011	[	123	7B	173	1111011	{
28	1C	034	0011100	FS (file separator)	60	3C	074	0111100	<	92	5C	134	1011100	\	124	7C	174	1111100	|
29	1D	035	0011101	GS (group separator)	61	3D	075	0111101	=	93	5D	135	1011101	]	125	7D	175	1111101	}
30	1E	036	0011110	RS (record separator)	62	3E	076	0111110	>	94	5E	136	1011110	^	126	7E	176	1111110	~
31	1F	037	0011111	US (unit separator)	63	3F	077	0111111	?	95	5F	137	1011111	_	127	7F	177	1111111	DEL

[1]

A similar method is used for the ‘tens’ unit of the array. However, instead of using MOD, a division is used since the unit value is irrelevant when calculating the tens value. Once the result is given and then the number 48 is added, the number displayed on the screen should then be equal to the value of the division performed. Another thing to note is that the ‘tens’ and ‘ones’ units are divided and MODed respectively by the value of the ‘countup_tim.sec’ value.

<u>4x4 Keypad: -</u>

The keypad used to navigate the program and to trigger flags within the program is arranged in a matrix format. By using this method the number of pins need to identify a button press has been greatly reduced in relation to the number of buttons available on the device. This method is used by having four columns and rows shown in the diagram below.

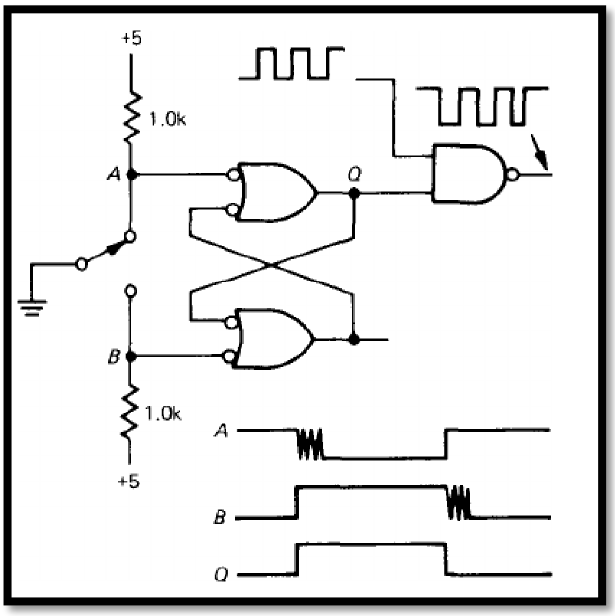


[2]

The way this work is while the button is being pressed down the processor is cycling through and setting HIGH each row while also checking the columns P0 to P3. This is to check if any of these columns have changed from a LOW input to a HIGH input. When the program picks up on one of these columns having a HIGH input it will then be able to detect which row and which column and deduct the button that had been pressed.
The "rx_keyscan_4x4.h" library which has been included in the program should be able to preform this task without having to write a series of lines of code dedicated to just the keypad. Instead the 'keyscan_4x4_init()' line will be all that is needed for the initialization step. The line 'key = getcommand()' is used to make the key variable equal to the returned value of the 'getcommand()' function. Following that, the 'key' variable has been used as an argument for the switch statement to return the value of the key pressed.

Button bouncing: -

The issue of using buttons in conjunction with a programable CPU is that there is the inevitability of button bounce. Button bounce can occur from 10 to 100 times for around 1ms **[3]**. This means the hardware will register multiple button presses, due to the reason that the hardware works faster than the number of bounces being generated. There is a software and a hardware method to fix this issue, however. The hardware method involves using a low pass filter for each button input. The low pass filter will remove the high frequency, which is the button bounce, and replace it with a quick discharge roll off time. Another hardware method can include the use of an SR latch circuit which will stay at the changed logic state without any button bouncing at the Q output. A diagram of this circuit from the 'Art of Electronics second edition' can be seen below.



[4]

However, if there are no hardware methods that have been applied to the circuit, then a software method can be used instead to save money resources and PCB space. This is a simple solution that only requires there to be a small 50ms delay forcing the program to wait for the bouncing to stop before resuming with the program. This method is considered to be very inefficient as it forces the CPU to waste processing time, just waiting for the condition to return as true to allow the program to resume [5]. So, by looking at these methods, it is simple to assume that the hardware method rules supreme over the resource wasting software method in terms of processing time. However, the software can save money on components and space at the expense of processing time. Either choice can be useful depending on how crucial the processing time is for its application and the monetary cost involved. Either choice is available at the user's discretion.

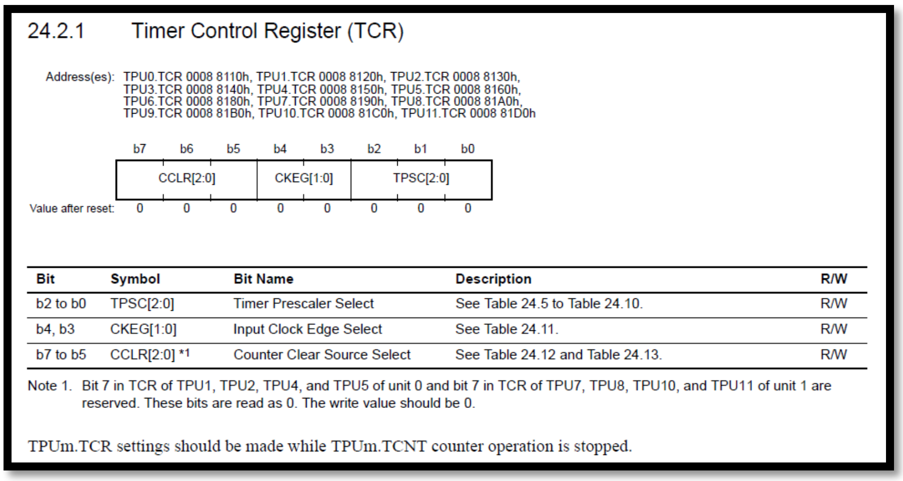
128x64 pixels LCD screen: -

The LCD screen has a 128x64 pixel screen that will allow the user to display any message of image on that screen using the 8 data pins to transmit data along a data bus. The RS pin allows the device to dictate the slave select (SS) state. An active high will disconnect the device from the SPI bus and when the line is active low the device is connected to the SPI bus again before data gets transmitted **[6][7]**. A simpler way of describing this would be that when the pin is high, the slave ignores the master device and when its low, it talks to the master device.
The R/W pin set to high will be used to decide if the data in the device should be read by the processor, or if it needs to have data written to the device it will write to the device when the pin is set to low **[8][9]**. The pin will be set to low often due to the general purpose of the device needing to display information written to in on its screen. Depending on the application will the read function rarely ever need to be used on average.
The E enable pin will allow the LCD module to capture the data presented to the register when the E pin transitions from high to low. Essentially meaning that it triggers on the falling edge from high to low. An extra voltage push is required to execute the specified instruction in an application, so this is the purpose of the of the enable pin **[10][11]**.
CS1 and CS2 are used to select the two drivers within the LCD screen which accommodates for a group of segments between them. For example, driver 1 drives the segments 0-63 on the LCD panel, while the second driver, driver 2, will drive the segments of 64-127 **[12]**.
The LED+/- pins are used as the anode and cathode respectively to activate and power the backlight.
Adjusting the resistance connected to the V0 pin to alter the darkness of the activated pixels against the backlight to display a sharper character on the screen.

Interupt service routine (ISR):

When initialising the ISR, bits 0 to 3 of the TMDR register are set to 0000 or 0x00 for enabling the normal mode operations setting. The normal mode setting allows the clock count to be internally generated.

Looking at the timer control register table in the datasheet **[13]**, a setting is required that will allow the peripheral clock to be reduced so that the timer can count in milliseconds. Since the peripheral clock needs to be reduced because of its high speed, the prescaler can be used to reduce the clock speed by creating a delay cycle of 64. This means that the time will increment by 1 whenever 64 cycles has occurred. To set the clock edge select for the timer, the function TCR is used with the with the hex number of 0x2B. this sets the clock division to 64 with bits 0-2, the clock edge select to count at the rising edge with pins 3-4 and set the counter clear source to be cleared by the TGRA input capture. Together this set of of instructions come to a binary number of ‘b00101011’.



24.2.1 Timer Control Register (TCR)

Address(es): TPU0.TCR 0008 8110h, TPU1.TCR 0008 8120h, TPU2.TCR 0008 8130h, TPU3.TCR 0008 8140h, TPU4.TCR 0008 8150h, TPU5.TCR 0008 8160h, TPU6.TCR 0008 8180h, TPU7.TCR 0008 8190h, TPU8.TCR 0008 81A0h, TPU9.TCR 0008 81B0h, TPU10.TCR 0008 81C0h, TPU11.TCR 0008 81D0h

	b7	b6	b5	b4	b3	b2	b1	b0
	CCLR[2:0]			CKEG[1:0]		TPSC[2:0]		
Value after reset:	0	0	0	0	0	0	0	0

Bit	Symbol	Bit Name	Description	R/W
b2 to b0	TPSC[2:0]	Timer Prescaler Select	See Table 24.5 to Table 24.10.	R/W
b4, b3	CKEG[1:0]	Input Clock Edge Select	See Table 24.11.	R/W
b7 to b5	CCLR[2:0] *1	Counter Clear Source Select	See Table 24.12 and Table 24.13.	R/W

Note 1. Bit 7 in TCR of TPU1, TPU2, TPU4, and TPU5 of unit 0 and bit 7 in TCR of TPU7, TPU8, TPU10, and TPU11 of unit 1 are reserved. These bits are read as 0. The write value should be 0.

TPUm.TCR settings should be made while TPUm.TCNT counter operation is stopped.

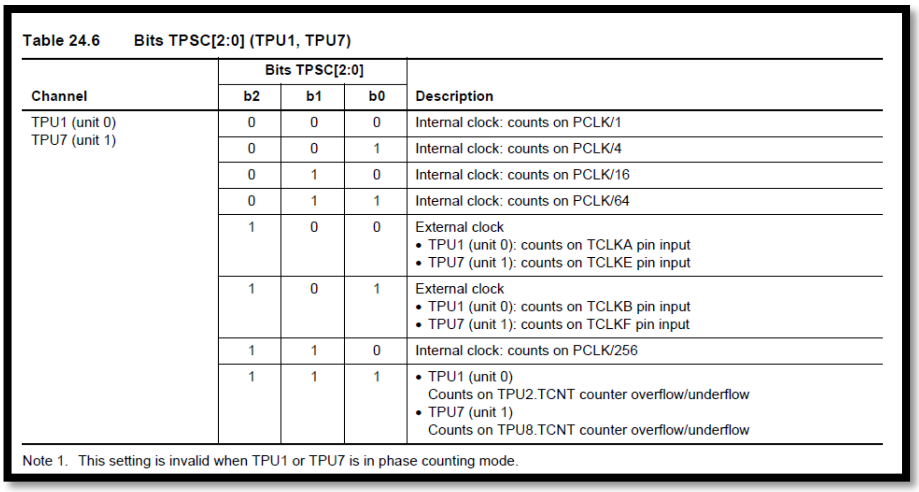


Table 24.6 Bits TPSC[2:0] (TPU1, TPU7)

	Bits TPSC[2:0]			
Channel	b2	b1	b0	Description
TPU1 (unit 0) TPU7 (unit 1)	0	0	0	Internal clock: counts on PCLK/1
	0	0	1	Internal clock: counts on PCLK/4
	0	1	0	Internal clock: counts on PCLK/16
	0	1	1	Internal clock: counts on PCLK/64
	1	0	0	External clock • TPU1 (unit 0): counts on TCLKA pin input • TPU7 (unit 1): counts on TCLKE pin input
	1	0	1	External clock • TPU1 (unit 0): counts on TCLKB pin input • TPU7 (unit 1): counts on TCLKF pin input
	1	1	0	Internal clock: counts on PCLK/256
	1	1	1	• TPU1 (unit 0) Counts on TPU2.TCNT counter overflow/underflow • TPU7 (unit 1) Counts on TPU8.TCNT counter overflow/underflow

Note 1. This setting is invalid when TPU1 or TPU7 is in phase counting mode.

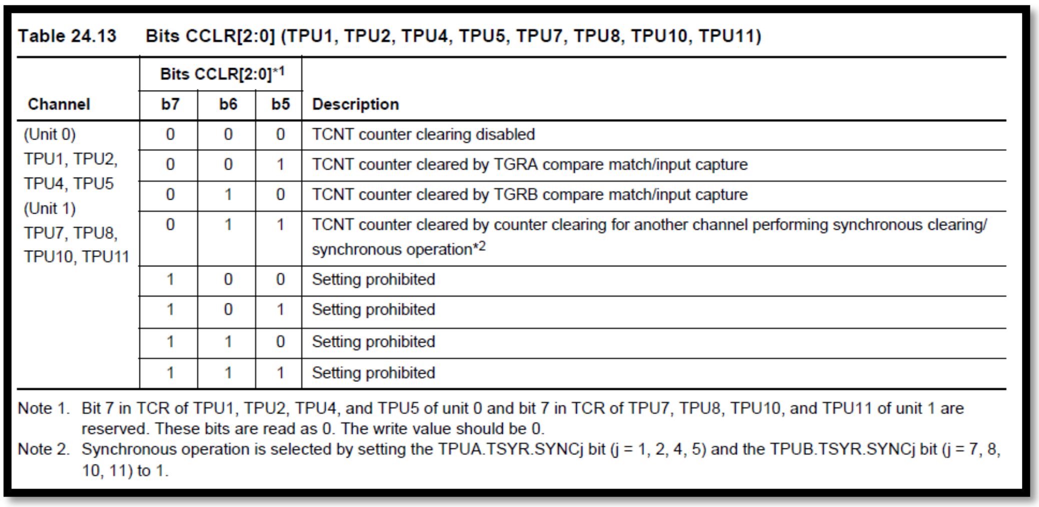


Table 24.13 Bits CCLR[2:0] (TPU1, TPU2, TPU4, TPU5, TPU7, TPU8, TPU10, TPU11)

	Bits CCLR[2:0]*1			
Channel	b7	b6	b5	Description
(Unit 0) TPU1, TPU2, TPU4, TPU5 (Unit 1) TPU7, TPU8, TPU10, TPU11	0	0	0	TCNT counter clearing disabled
	0	0	1	TCNT counter cleared by TGRA compare match/input capture
	0	1	0	TCNT counter cleared by TGRB compare match/input capture
	0	1	1	TCNT counter cleared by counter clearing for another channel performing synchronous clearing/ synchronous operation*2
	1	0	0	Setting prohibited
	1	0	1	Setting prohibited
	1	1	0	Setting prohibited
	1	1	1	Setting prohibited

Note 1. Bit 7 in TCR of TPU1, TPU2, TPU4, and TPU5 of unit 0 and bit 7 in TCR of TPU7, TPU8, TPU10, and TPU11 of unit 1 are reserved. These bits are read as 0. The write value should be 0.
Note 2. Synchronous operation is selected by setting the TPUA.TSYR.SYNCj bit (j = 1, 2, 4, 5) and the TPUB.TSYR.SYNCj bit (j = 7, 8, 10, 11) to 1.

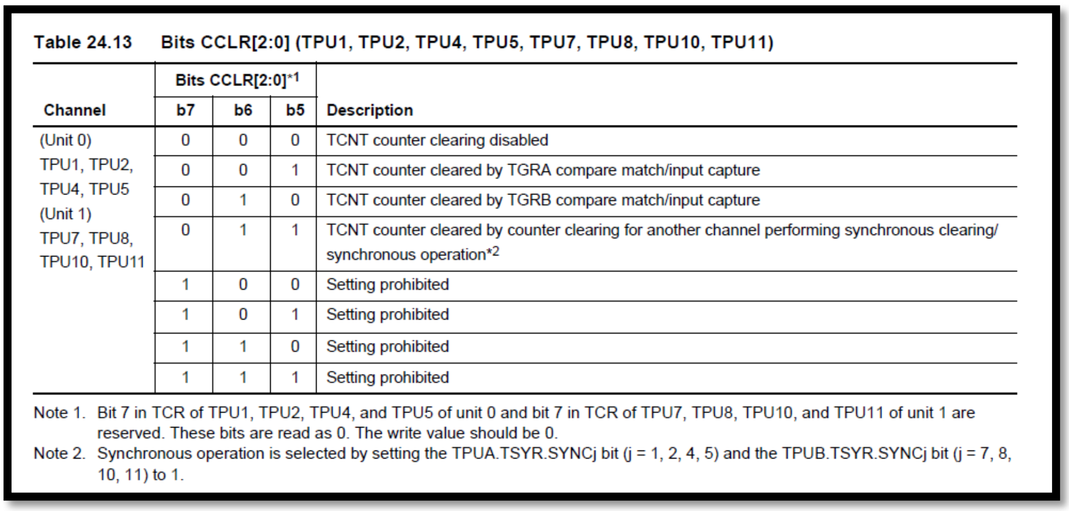


Table 24.13 Bits CCLR[2:0] (TPU1, TPU2, TPU4, TPU5, TPU7, TPU8, TPU10, TPU11)

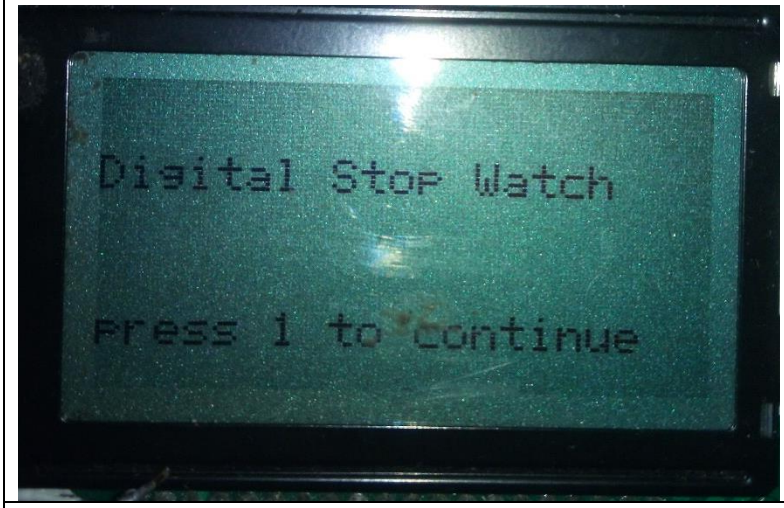
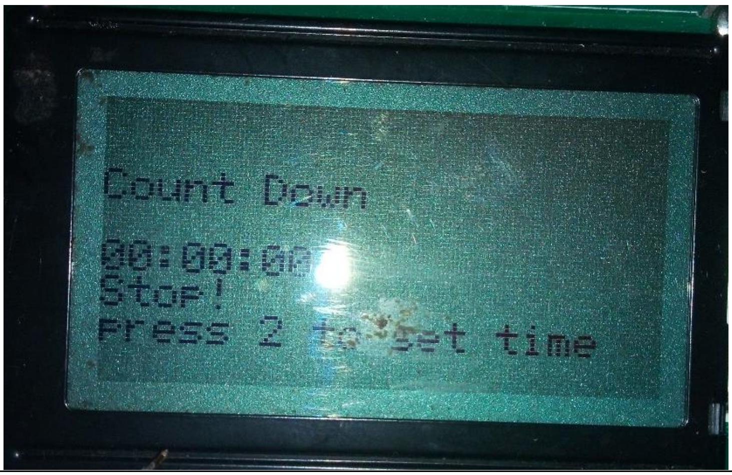
Channel	Bits CCLR[2:0]*1			
	b7	b6	b5	Description
(Unit 0) TPU1, TPU2, TPU4, TPU5 (Unit 1) TPU7, TPU8, TPU10, TPU11	0	0	0	TCNT counter clearing disabled
	0	0	1	TCNT counter cleared by TGRA compare match/input capture
	0	1	0	TCNT counter cleared by TGRB compare match/input capture
	0	1	1	TCNT counter cleared by counter clearing for another channel performing synchronous clearing/ synchronous operation*2
	1	0	0	Setting prohibited
	1	0	1	Setting prohibited
	1	1	0	Setting prohibited
	1	1	1	Setting prohibited

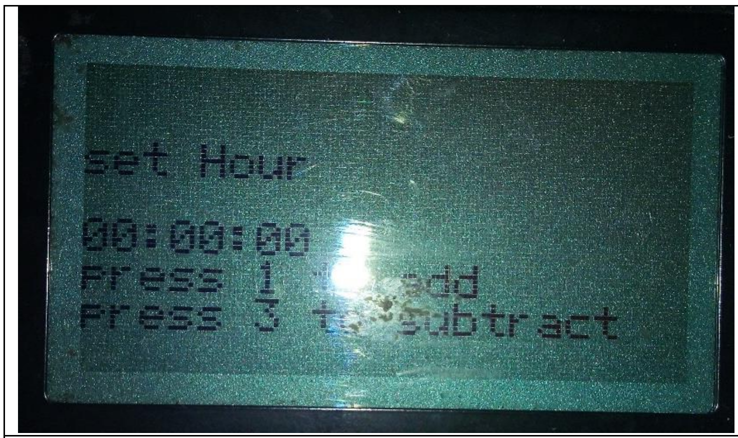
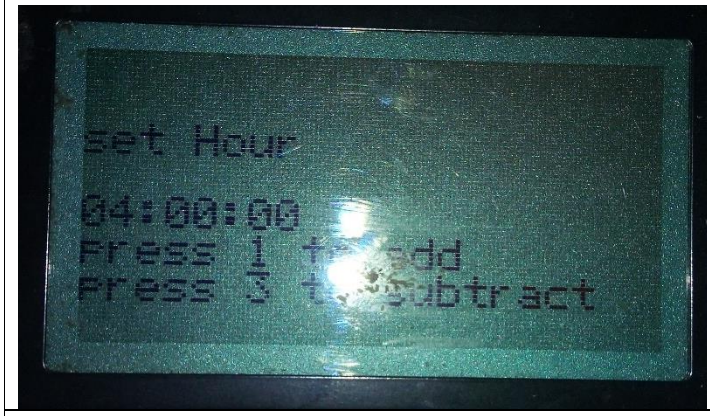
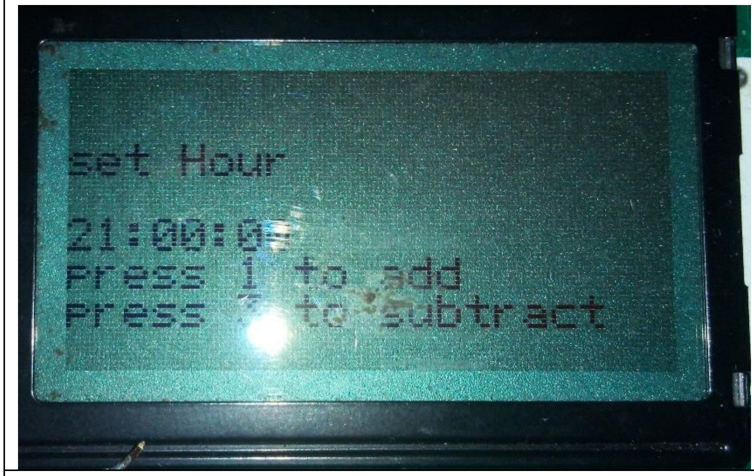
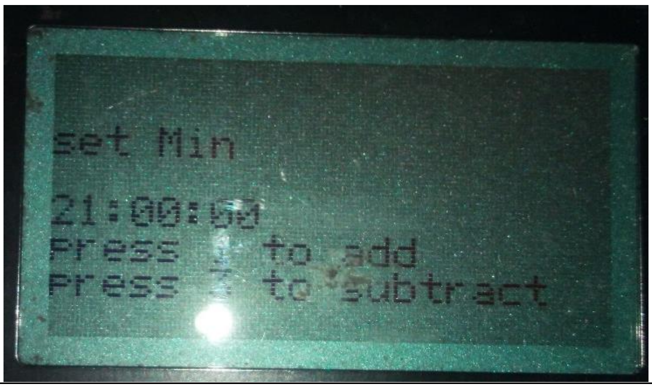
Note 1. Bit 7 in TCR of TPU1, TPU2, TPU4, and TPU5 of unit 0 and bit 7 in TCR of TPU7, TPU8, TPU10, and TPU11 of unit 1 are reserved. These bits are read as 0. The write value should be 0.
Note 2. Synchronous operation is selected by setting the TPUA.TSYR.SYNCj bit (j = 1, 2, 4, 5) and the TPUB.TSYR.SYNCj bit (j = 7, 8, 10, 11) to 1.

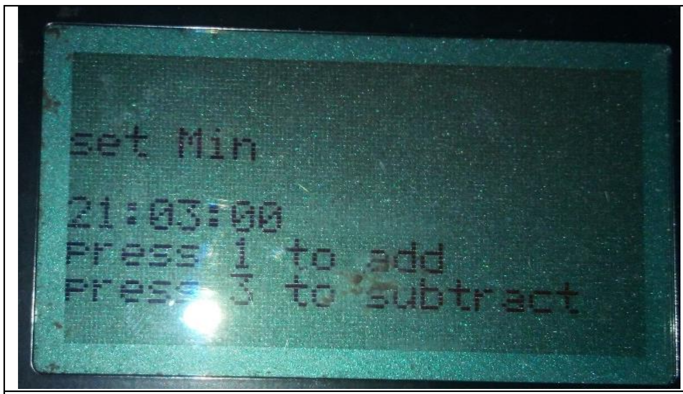
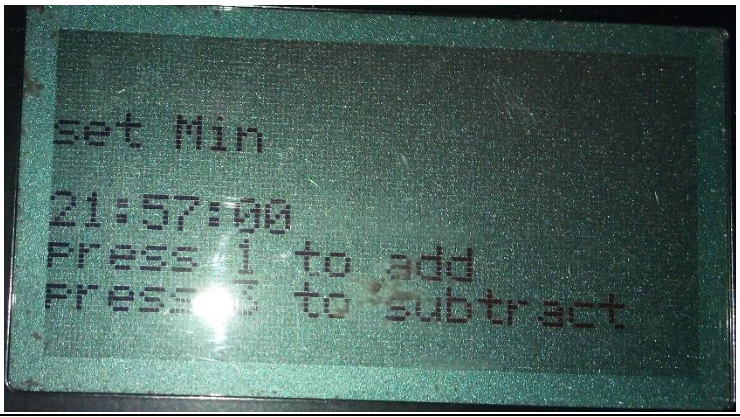
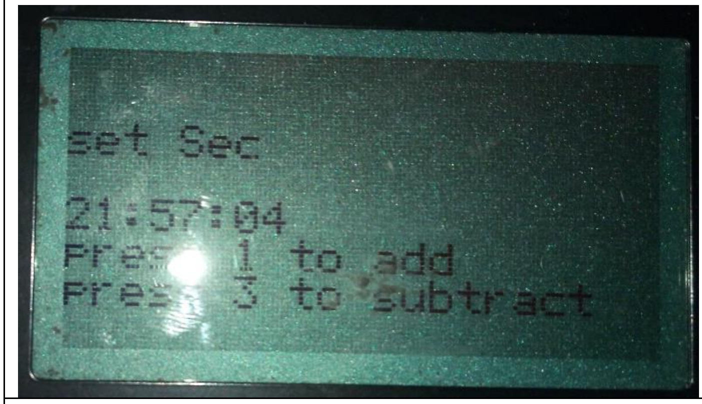
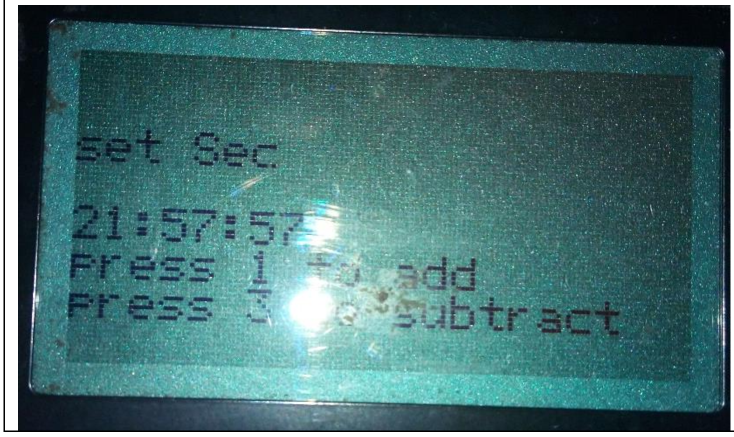
[14]

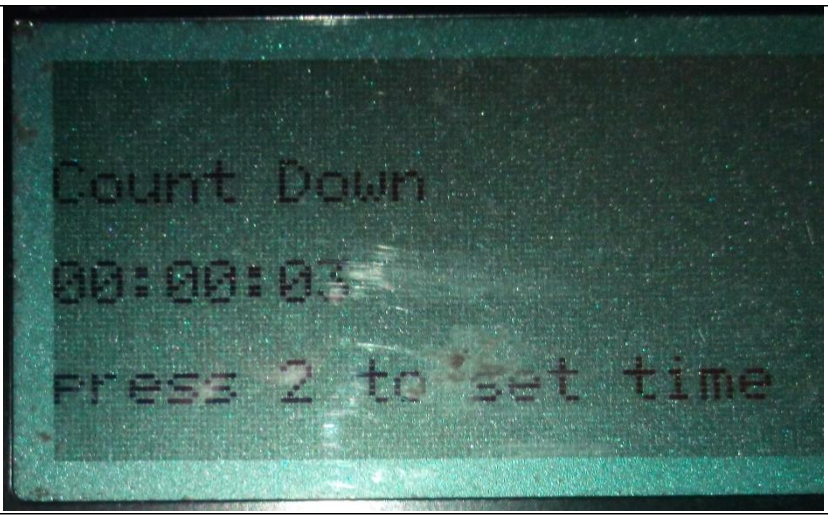
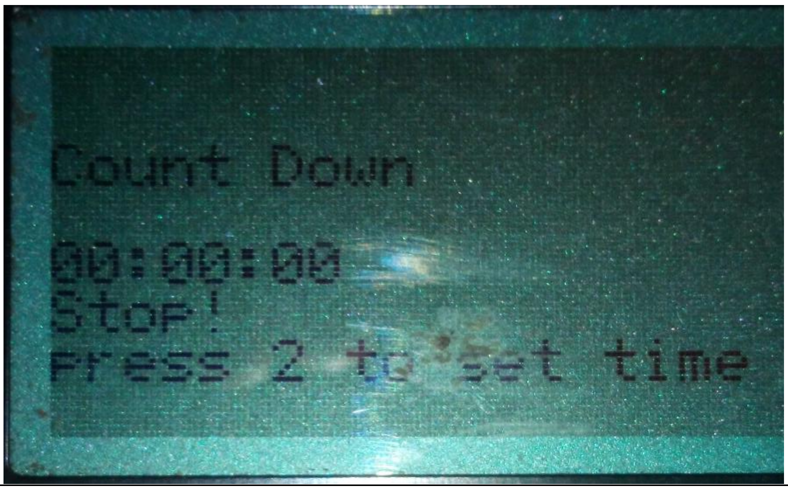
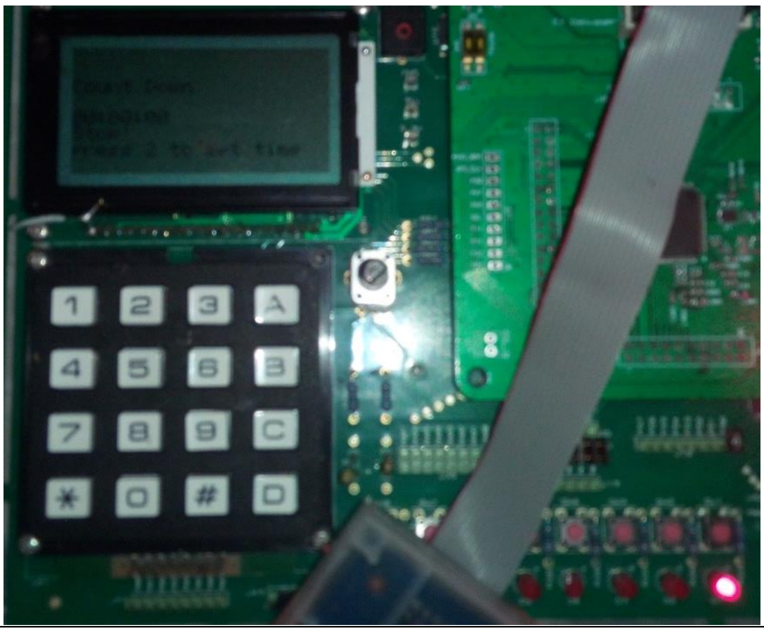
These figures were gathered from looking within the hardware manual for the RX63N microcontroller in chapter 24.2.1.

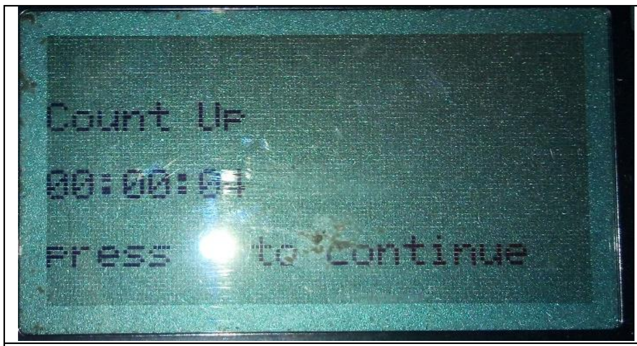
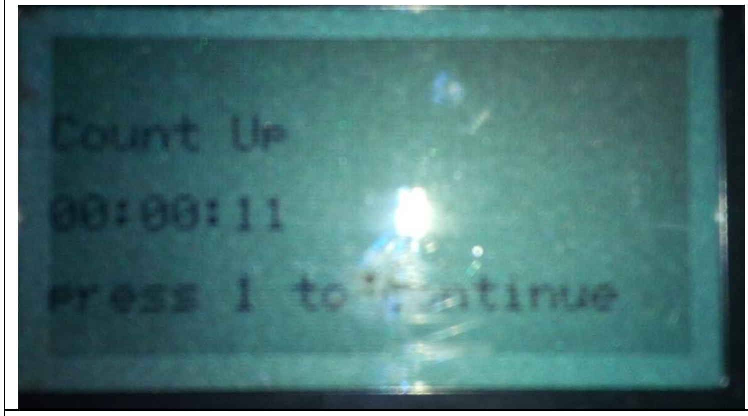
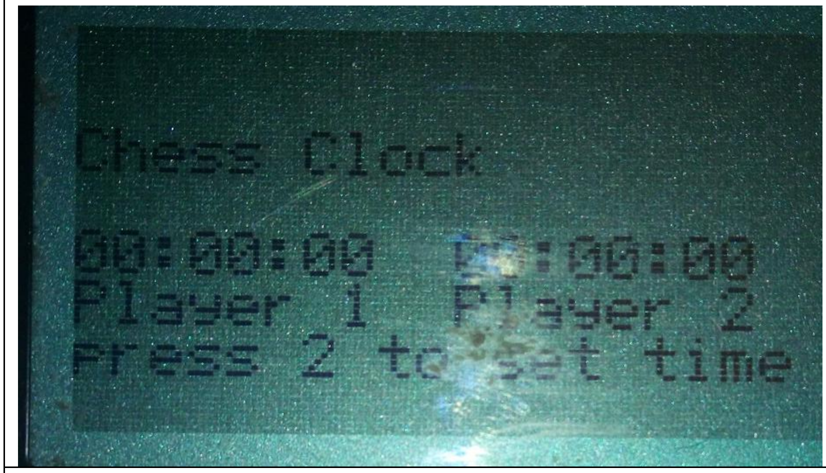
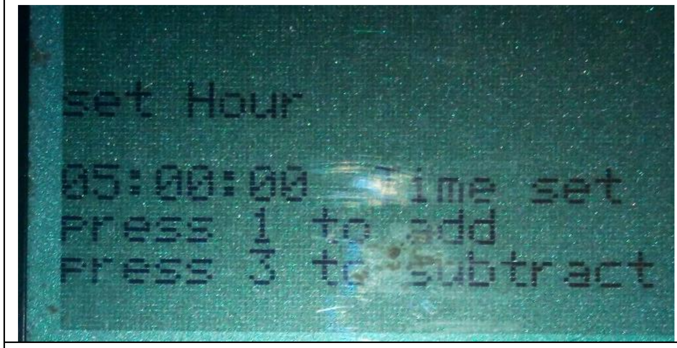
Testing:

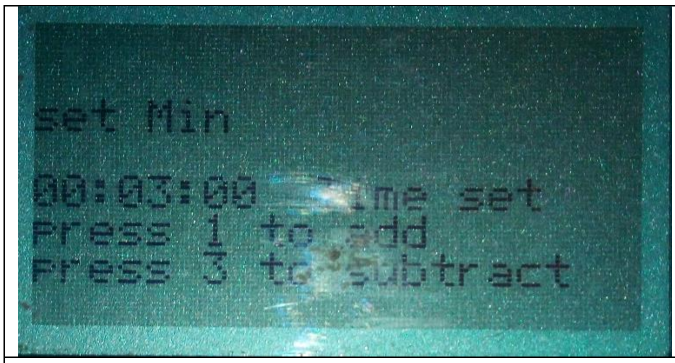
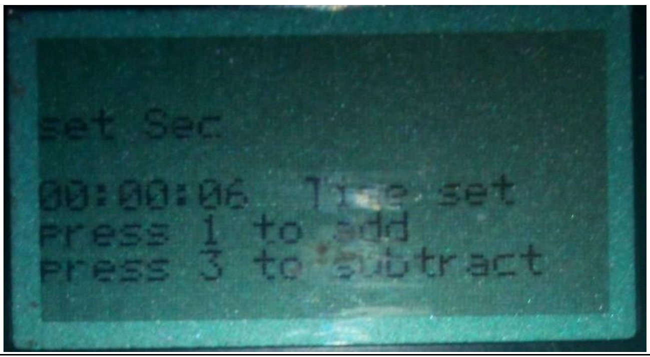
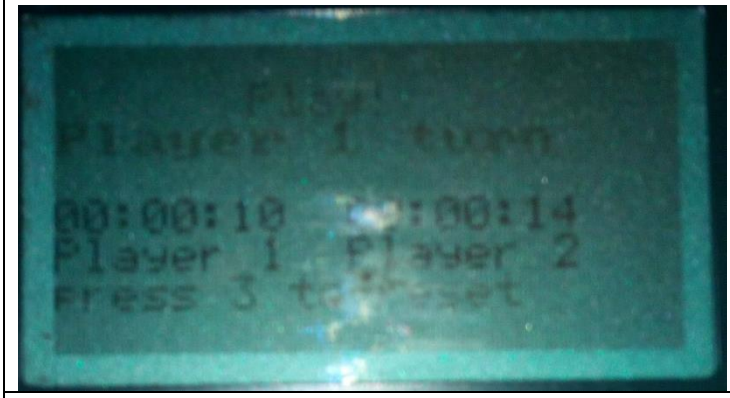
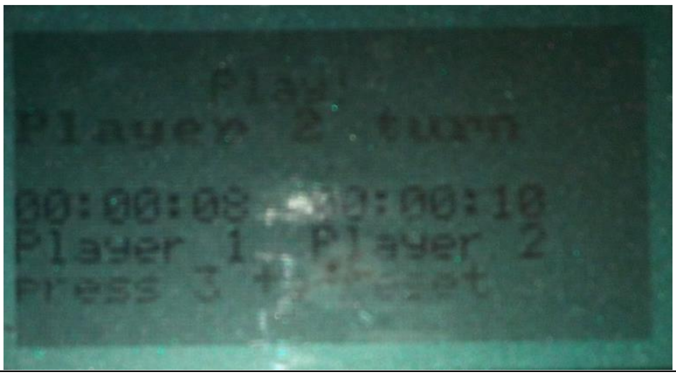
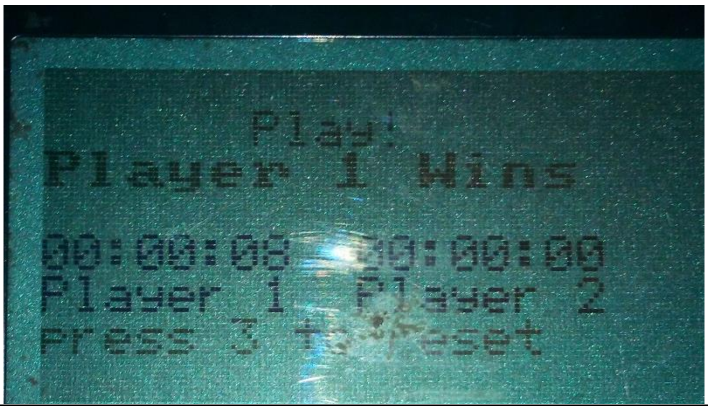
 Digital Stop Watch press 1 to continue	Here is the main menu when the program starts
 Count Down 00:00:00 Stop! press 2 to set time	Here is the count down menu after pressing button 1

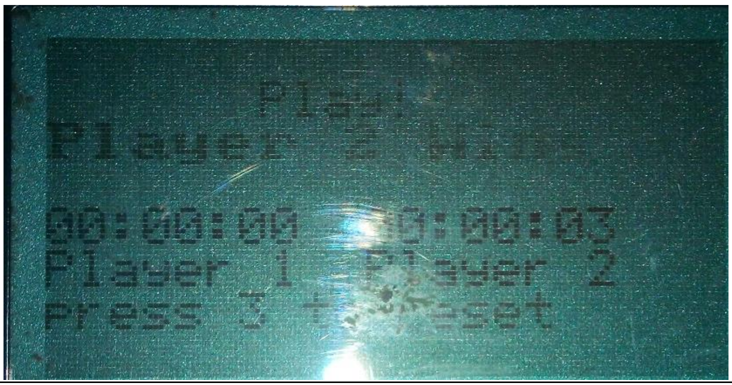
 set Hour 00:00:00 Press 1 to add Press 3 to subtract	Here is the set hour screen after pressing 2
 set Hour 04:00:00 Press 1 to add Press 3 to subtract	Pressing 1 increases the hour time by one unit per press
 set Hour 21:00:00 Press 1 to add Press 3 to subtract	Pressing 3 will decrement the hours by 1 each press.
 set Min 21:00:00 Press 1 to add Press 3 to subtract	This is the set minutes screen.

 set Min 21:03:00 Press 1 to add Press 3 to subtract	Pressing 1 increases the minute time by one unit per press
 set Min 21:57:00 Press 1 to add Press 3 to subtract	Pressing 3 will decrement the minute by 1 each press.
 set Sec 21:57:04 Press 1 to add Press 3 to subtract	Pressing 1 increases the second time by one unit per press
 set Sec 21:57:57 Press 1 to add Press 3 to subtract	Pressing 3 will decrement the second by 1 each press.

 Count Down 00:00:03 Press 2 to set time	This is after pressing 2 again to begin the countdown with that time set
 Count Down 00:00:00 Stop! Press 2 to set time	This is the message “Stop!” displayed when the countdown has ended.
	This is the flashing alarm LED that will flash when the countdown has ended
	This is the count up menu after pressing 1. Pressing 2 starts the count up time.

 Count Up 00:00:04 Press [illegible] to Continue	
 Count Up 00:00:11 Press 1 to Continue	This is the count up time paused after pressing button 2
 Chess Clock 00:00:00 00:00:00 Player 1 Player 2 Press 2 to set time	This is the chess clock menu after pressing button 1
 Set Hour 05:00:00 Time set Press 1 to add Press 3 to subtract	This is the set hour screen after pressing button 2. Using button 1 and 3 to increment and decrement.
	This is the minute screen after pressing button 2. Using button 1 and 3 to increment and decrement.

 set Min 00:03:00 Time set Press 1 to add Press 3 to subtract	
 set Sec 00:00:06 Time set Press 1 to add Press 3 to subtract	This is the second screen after pressing button 2. Using button 1 and 3 to increment and decrement.
 Play! Player 1 turn 00:00:10 00:00:14 Player 1 Player 2 Press 3 to reset	This is the chess clock running after pressing 2 and confirming the time. Player 1 is running and player 2 is paused. The message "player 1 turn" is being shown. some of the messages were flashing
 Play! Player 2 turn 00:00:08 00:00:10 Player 1 Player 2 Press 3 to reset	Player 2 is running and player 1 is paused. The message "player 2 turn" is being shown. This is after pressing button 2 to alternate the clocks. some of the messages were flashing
 Play! Player 1 Wins 00:00:08 00:00:00 Player 1 Player 2 Press 3 to reset	This is the message displayed when player 2's timer has reached zero. However, some of the messages were flashing


`Play!`
`Player 2 Wins`
`00:00:00 00:00:03`
`Player 1 Player 2`
`Press 3 to reset`
	This is the message displayed when player 1’s timer has reached zero. some of the messages were flashing

Conclusion:

In conclusion, it is shown that there was an issue with the displaying of certain messages in the chess clock function due to its flickering effect whenever it was prompted to be shown on screen. A reason I believe for this, is to do with the clock speed of the RX63N not being able to efficiently display these messages using if statements to check the status of the clocks and flag. A more efficient way of doing this would have been to use case statements in conjunction with an interrupt to give the required results when the required value has been achieved.

The functions however were able to successfully run without any issues, counting-up and down with each timer or stopwatch as expected. The alarm was also able to be utilised for this assignment by being set to flash on and off when the countdown timer and one of the players in the chess timer had counted down to 0.

Looking at the program, it is noticeable that there is no way to reset the alarm LED after it has been triggered. In the future I would have added a line of code to make the flag == 0 and to set the variable value of toggle_LED == 0 to turn the LED off after a reset button has been pressed or when a new timer function menu has been selected.

For any hardware requirements, I would include an external clock that would exceed the speed of the current external clock to possibly resolve this flickering issue the device has been having. Although, I don’t think it is necessary to do so due to the inefficient coding that has been implemented. A buzzer should have also been included in this project to solve the alarm issue that has been present throughout this assignment.

Sources:

[1] Asciitablexyz/, 2020. Asciitablexyz/. [Online].[23 January 2020]. Available from: https://www.asciitable.xyz/

[2] Parallax Inc, 2020. LearnParallaxcom. [Online].[24 January 2020]. Available from: http://learn.parallax.com/tutorials/language/propeller-c/propeller-c-simple-devices/read-4x4-matrix-keypad

[3] "The Art of Electronics", Horowitz & Hill, Second edition, pg 506, under heading of 'Switch debouncing'

[4] "The Art of Electronics", Horowitz & Hill, Second edition, pg 507, under heading of 'Switch debouncing'

[5] Jens Christoffersen, J.C. 2020. Allaboutcircuitscom/technical-articles/switch-bounce-how-to-deal-with-it/. [Online].[25 January 2020]. Available from: https://www.allaboutcircuits.com/technical-articles/switch-bounce-how-to-deal-with-it/

[6] Arduino, 2020. Arduinocc/en/reference/SPI. [Online].[26 January 2020]. Available from: https://www.arduino.cc/en/reference/SPI

[7] MIKEGRUSIN, 2020. Sparkfuncom/tutorials/serial-peripheral-interface-spi/slave-select-ss. [Online].[26 January 2020]. Available from: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/slave-select-ss

[8] Learningaboutelectronicscom, 2020. Learningaboutelectronicscom/Articles/HD44780-LCD-read-write-RW-pin. [Online].[26 January 2020]. Available from: http://www.learningaboutelectronics.com/Articles/HD44780-LCD-read-write-RW-pin

[9] I-lcdcom/PDFs/TMBG12864EPT-08-SPA00pdf, 2020. I-lcdcom/PDFs/TMBG12864EPT-08-SPA00pdf. [Online].[26 January 2020]. Available from: http://www.i-lcd.com/PDFs/TMBG12864EPT-08-SPA00.pdf pg5

[10] Electronicsforucom, 2020. Electronicsforucom/resources/learn-electronics/16x2-lcd-pinout-diagram. [Online].[26 January 2020]. Available from: https://www.electronicsforu.com/resources/learn-electronics/16x2-lcd-pinout-diagram

[11] Sparkfuncom, 2020. Sparkfuncom/datasheets/LCD/HD44780pdf. [Online].[26 January 2020]. Available from: https://www.sparkfun.com/datasheets/LCD/HD44780.pdf pg58 fig 26

[12] I-lcdcom/PDFs/TMBG12864EPT-08-SPA00pdf, 2020. I-lcdcom/PDFs/TMBG12864EPT-08-SPA00pdf. [Online].[26 January 2020]. Available from: http://www.i-lcd.com/PDFs/TMBG12864EPT-08-SPA00.pdf pg2 under Block Diagram.

[13] hardware manual for the RX63N microcontroller.Renesas.[27 January 2020].chapter24.2

[14] hardware manual for the RX63N microcontroller.Renesas.[27 January 2020].chapter24.2

[15] https://picclick.com/Renesas-YRDKRX63N-RX63N-Development-Kit-153731072497.html

Appendices:

```
/////////////////////////////////////////////////////////////////////////////////

MAIN

/////////////////////////////////////////////////////////////////////////////////

/******************* (C) COPYRIGHT 2018-2019 University of South Wales
******************************

* File Name         : main.c

* Author           : SS

* Version          : V1.0.1

* Date             : 21-March-2019

* Description       : main program

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

* THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING STUDENTS

* WITH CODING INFORMATION REGARDING THEIR ASSIGNMENT IN ORDER FOR THEM DELIVER TO THE
STANDARD.

* AS A RESULT, UNIVERSITY OF SOUTH WALES SHALL NOT BE HELD LIABLE FOR ANY DIRECT,

* INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE

* CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY STUDENTS OF THE CODING

* INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS,PROJECT OR ASSIGNMENTS.

*************************************************************************************
***********/


/*********** System Clock Configuration ********************************************

  Clock Description          Frequency

  ---------------------------------------

  Input Clock Frequency...........  12 MHz

  PLL frequency (x16)............. 192 MHz

  Internal Clock Frequency.........  96 MHz

  Peripheral Clock Frequency.......  48 MHz

  USB Clock Frequency.............  48 MHz

  External Bus Clock Frequency.....  24 MHz

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

*/
```

```
#include "iodefine.h"

#include "typedefine.h"

#include "d_stop_watch.h"


/* Private Defines -----------------------------------------------------------*/


/* Extern variables ----------------------------------------------------------*/

//extern FONT_DEF Font_System3x6;     /*uncomment this if needed */

//extern FONT_DEF Font_System5x8;

//extern FONT_DEF Font_System7x8;


/* Private variables ---------------------------------------------------------*/




/* Private function prototypes -----------------------------------------------*/

void main(void);

#ifdef __cplusplus

extern "C" {

void abort(void);

}

#endif




void main(void)

{


/* System Clock has already been configured at this point, Please Refer to the System Clock Configuration for
more Details */

 d_stop_watch_Init();                  // initialise Stop watch application
```

```
        while(1)        // Beginning of super loop

        {

                /* add your code here */


                d_stop_watch_process();                 // begin stop watch process task


        } // end of super loop


}       // end of main program


#ifdef __cplusplus

void abort(void)

{


}

#endif
```

///

D_stop_watch

///

```
/******************** (C) COPYRIGHT 2018-2019 University of South Wales
********************************

* File Name      : d_stop_watch.c

* Author         : SS

* Version        : V1.0.1

* Date           : 18-September-2019

* Description    : NG2S901 Assignment Template (Digital Stop Watch)

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

* THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING STUDENTS

* WITH CODING INFORMATION REGARDING THEIR ASSIGNMENT IN ORDER FOR THEM DELIVER TO THE
STANDARD.

* AS A RESULT, UNIVERSITY OF SOUTH WALES SHALL NOT BE HELD LIABLE FOR ANY DIRECT,

* INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE
* CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY STUDENTS OF THE CODING
* INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS,PROJECT OR ASSIGNMENTS.
*******************************************************************************************
***********/

#include "rx_glcd.h"       // include GLCD
#include "font.h"
#include "rx_keyscan_4x4.h"
#include "d_stop_watch.h"



extern FONT_DEF Font_System3x6;
extern FONT_DEF Font_System5x8;
extern FONT_DEF Font_System7x8;





estate state;                    // state variable deceleration

clock_t tmp_tim;                                   // temp timer structure
clock_t countup_tim;                               // temp timer structure
clock_t countdown_tim;                                     //temp timer structure count down
clock_t chess_P_One_tim;                                          //temp chess timer for player one
clock_t chess_P_two_tim;                                          //temp chess timer for player two

//**************************a collection of strings that can be displayed on the lcd
screen******************************************
unsigned char main_menu_1[]="Digital Stop Watch";      // create and initialise array of string
unsigned char main_menu_2[]="press 1 to continue";     // instruction for the user
unsigned char adding[]="press 1 to add";
unsigned char subbing[]="press 3 to subtract";
unsigned char settimeing[]="press 2 to set time";
unsigned char play[]="Play!";

unsigned char time[]="Time set";

unsigned char resetingbutton[]="press 3 to reset";


//*****************Titles of Functions*******************

unsigned char count_down_menu[]="Count Down";

unsigned char count_up_menu[]="Count Up";

unsigned char ches_clk_menu[]="Chess Clock";

//*****************************************************

//******************the time display and the display of the time set
functions**********************

unsigned char clock_dsp[] = "00:00:00";

unsigned char clock_dsptwo[] ="00:00:00";


unsigned char count_down_setH[]="set Hour";

unsigned char count_down_setM[]="set Min";

unsigned char count_down_setS[]="set Sec";


//************************************************************************************
*************

//******************The start and stop messages*******************

unsigned char start_message[] = "Start!";

unsigned char stop_message[] = "Stop!";

extern unsigned char Ptwo_win[] = "Player 2 Wins";

extern unsigned char Pone_win[] = "Player 1 Wins";

unsigned char Ptwo_turn[] = "Player 2 turn";

unsigned char Pone_turn[] = "Player 1 turn";

//**************************************************************


unsigned char Player_One[]="Player 1";

unsigned char Player_Two[]="Player 2";

unsigned char Player_Num[]="Player number displayed here";


extern unsigned char countup_flag;                    // a flag to start the count up time
extern unsigned char countdown_flag;            // a flag to start the count down time
extern unsigned char chesscountdown_flag;       //a flag to start the count down on the chess clock
extern unsigned char alarm_flag;                            // a flag for when the count down alarm goes off


//*********************flags trigger certain code.************************
//char cnt_down_timer_flag = 0;
//char cnt_up_timer_flag = 0;
//char chess_clock_timer_flag = 0;

char key;          // used to hold the button number pressed.




/* Digital stop watch initialisation */

void d_stop_watch_Init (void)
{

        /* Initialise Graphic Display */

        GLCD_Initialise();                  // Hardware configuration for GLCD
        GLCD_ClearScreen();     // clear the GLCD

        keyscan_4x4_init();      // initialise key pad

//      GLCD_GoTo(0,2);
//      GLCD_WriteString(main_menu_1,Font_System5x8);                // display title
//
//      GLCD_GoTo(0,4);
//      GLCD_WriteString(clock_dsp,Font_System5x8);              // user instruction
//
//      GLCD_GoTo(0,6);
//      GLCD_WriteString(main_menu_2,Font_System5x8);                // user instruction
```

```
/* state initialisation */
state = menu;

/* Timer Variable Init */
//******COUNT UP TIME******
countup_tim.hour = 0;
countup_tim.min = 0;
countup_tim.sec = 0;

//******COUNT DOWN TIME*******
countdown_tim.hour = 0;
countdown_tim.min = 0;
countdown_tim.sec = 0;

//******TEMPORARY TIME*******
tmp_tim.hour = 0;
tmp_tim.min = 0;
tmp_tim.sec = 0;

//*******PLAYER 1 TIME*******
chess_P_One_tim.hour = 0;
chess_P_One_tim.min = 0;
chess_P_One_tim.sec = 0;

//*******PLAYER 2 TIME*******
chess_P_two_tim.hour = 0;
chess_P_two_tim.min = 0;
chess_P_two_tim.sec = 0;
//##############################################################

/* IO port configuration */
//INITIALISE THE ALARM LED BIT TO AN OUTPUT.
PORTE.PDR.BYTE = 0xFF;
```

```
//#############################################################

/* Timer Configurations */
SYSTEM.PRCR.WORD = 0xA50B; /* Protect off */
MPC.PWPR.BIT.B0WI = 0 ;                    /* Unlock protection register */
MPC.PWPR.BIT.PFSWE = 1 ;                   /* Unlock MPC registers */

MSTP(TPU1) = 0;     /* Cancel TPU peripheral stop state. */

/* Timer Control Register (TCR)
b7:b5   CCLR: Counter Clear Source 1 = TCNT cleared by TGRA
b4:b3   CKEG: Clock Edge          2 = count at rising edge
b2:b0   TPSC: Time Prescaler      3 = PCLK/64 */
TPU1.TCR.BYTE=0x2B;   // PCLK/64,rising edge,count cleared by TGRA looked it up in the manual

            /* Timer Mode Register (TMDR)
b6      BFE - TPUm.TGRE operates normally
b5      BFB - TPUm.TGRB operates normally
b4      BFA - TPUm.TGRA operates normally
b3:b0   MD - Normal operation                         */
TPU1.TMDR.BYTE=0x00;            // normal operation

            /* Timer I/O Control Register (TIOR)
b7:b4   IOB - Output compare, output disabled
b3:b0   IOA - Output compare, output disabled
*/
TPU1.TIOR.BYTE=0x00;  // Output disable

/* Timer Interrupt Enable Register*/
TPU1.TIER.BYTE=0x01;   // TGRA Interrupt enabled

/*Prescaler Reload for TGRA */
TPU1.TGRA= 749;                     // generate 1ms interrupt
```

```
        TPU1.TCNT=0x0000;


        IR (TPU1, TGI1A);    /* interrupt reset */
        IPR(TPU1, TGI1A) = 3; /* interrupt priority set */
        IEN(TPU1, TGI1A) = 1; /* interrupt enable */


        //starts the tpu1 timer
        TPUA.TSTR.BIT.CST1 = 1;                 // start TPU1 timer


        MPC.PWPR.BIT.B0WI = 1 ;                 /* lock protection register */
        MPC.PWPR.BIT.PFSWE = 0 ;                /* lock MPC registers */
        SYSTEM.PRCR.WORD = 0xA500; /* Protect on */


}


//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
//**********************************************MAIN BLOCK OF
CODE*****************************************************
//-------------------------------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------
void d_stop_watch_process (void)
{




                switch(state)
                {
                        case menu:                      // main menu
                                clock_dsp[7] = countup_tim.sec%10+48;           // convert second to
ASCII(ones)
                                clock_dsp[6] = countup_tim.sec/10+48;   // convert second to ASCII (tens)
```

```
				clock_dsp[4] = countup_tim.min%10+48;		// convert min to ASCII(ones)
				clock_dsp[3] = countup_tim.min/10+48;	// convert min to ASCII (tens)

				clock_dsp[1] = countup_tim.hour%10+48;		// convert hour to 
ASCII(ones)
				clock_dsp[0] = countup_tim.hour/10+48;	// convert hour to ASCII (tens)


				/* display String 'clock_dsp'*/
				GLCD_GoTo(0,2);
				GLCD_WriteString(main_menu_1,Font_System5x8);			// display 
title

				GLCD_GoTo(0,6);
				GLCD_WriteString(main_menu_2,Font_System5x8);			// user 
instruction

				/* get key commands from the key pad */
				key  = getcommand();
				switch (key)
				{
					case '1':// '1' is pressed
					state = cnt_dwn;
					GLCD_ClearScreen();	// clear the GLCD
					GLCD_GoTo(0,2);
					GLCD_WriteString(count_down_menu,Font_System5x8);	
	// display Count Down Menu
					GLCD_GoTo(0,6);
					GLCD_WriteString(main_menu_2,Font_System5x8);			// 
user instruction

						break;
					case '2':// '2' is pressed
						break;
					case '3':// '3' is pressed
						break;
```

```
                    default:

                            break;

                }



        break;



//**************************************************************************

        //********************Count down
Timer*************************************

//**************************************************************************

        case cnt_dwn:           // count down timer

                clock_dsp[7] = countdown_tim.sec%10+48;     // convert second to
ASCII(ones)

                clock_dsp[6] = countdown_tim.sec/10+48;   // convert second to ASCII (tens)


                clock_dsp[4] = countdown_tim.min%10+48;     // convert second to
ASCII(ones)

                clock_dsp[3] = countdown_tim.min/10+48;   // convert second to ASCII (tens)


                clock_dsp[1] = countdown_tim.hour%10+48;     // convert second to
ASCII(ones)

                clock_dsp[0] = countdown_tim.hour/10+48;   // convert second to ASCII
(tens)


                if (countdown_flag == 0)        //checks if the count down flag is equal to 0

                {


                        GLCD_GoTo(0,5);                         //goes to the 0,6
position on the LCD

                        GLCD_WriteString(stop_message,Font_System5x8);          //
displays a stop message


                }
```

```
                    GLCD_GoTo(0,2);
                    GLCD_WriteString(count_down_menu,Font_System5x8);                   //
display Count Down Menu

                    GLCD_GoTo(0,4);
                    GLCD_WriteString(clock_dsp,Font_System5x8);            //displays the clock
on line 4

                    GLCD_GoTo(0,6);
                    GLCD_WriteString(settimeing,Font_System5x8);           // user instruction

                    key  = getcommand();


                    switch (key)
                    {
                        case '1':
                        state = cnt_up;
                        GLCD_ClearScreen();     // clear the GLCD
                        GLCD_GoTo(0,2);
                        GLCD_WriteString(count_up_menu,Font_System5x8);             //
display Count Down Menu
                        GLCD_GoTo(0,6);
                        GLCD_WriteString(main_menu_2,Font_System5x8);               //
user instruction
                            break;
                        case '2':
                            state = cnt_dwn_setH;
                            GLCD_ClearScreen();     // clear the GLCD
```

```
                                        break;
                                case '3':
                                        countdown_tim.sec=0;  //reset count up timer
                                        countdown_tim.min=0;
                                        countdown_tim.hour=0;
                                        break;

                                default:
                                        break;
                        }

                break;
        //***************************************************************************
                //******************************Stop
watch***********************************

        //***************************************************************************
                case cnt_dwn_setH:      // count down set hour
                        clock_dsp[7] = countdown_tim.sec%10+48;        // convert second to
ASCII(ones)
                        clock_dsp[6] = countdown_tim.sec/10+48;   // convert second to ASCII (tens)

                        clock_dsp[4] = countdown_tim.min%10+48;       // convert second to
ASCII(ones)
                        clock_dsp[3] = countdown_tim.min/10+48;   // convert second to ASCII (tens)

                        clock_dsp[1] = countdown_tim.hour%10+48;      // convert second to
ASCII(ones)
                        clock_dsp[0] = countdown_tim.hour/10+48;   // convert second to ASCII
(tens)

                        GLCD_GoTo(0,2);
                        GLCD_WriteString(count_down_setH,Font_System5x8);             // display
Count Down Menu
```

```
GLCD_GoTo(0,4);
GLCD_WriteString(clock_dsp,Font_System5x8);


GLCD_GoTo(0,5);
GLCD_WriteString(adding,Font_System5x8);
GLCD_GoTo(0,6);
GLCD_WriteString(subbing,Font_System5x8);


key  = getcommand();
switch (key)
{
        case '1':
                countdown_tim.hour++;
                if (countdown_tim.hour>23)
                {
                        countdown_tim.hour=0;
                }


                break;
        case '2':
                state = cnt_dwn_setM;
                GLCD_ClearScreen();     // clear the GLCD


                break;
        case '3':
                countdown_tim.hour--;           //takes away 1 from the
hour value
                if (countdown_tim.hour <= 0)
                {
                        countdown_tim.hour = 23;
                }


                break;
}
```

```
                        break;

          //***************************************************************************
                        case cnt_dwn_setM:      // count down set minute
                                  clock_dsp[7] = countup_tim.sec%10+48;             // convert second to
ASCII(ones)
                                                                      clock_dsp[6] = countdown_tim.sec/10+48;
// convert second to ASCII (tens)

                                                                      clock_dsp[4] = countdown_tim.min%10+48;
          // convert second to ASCII(ones)
                                                                      clock_dsp[3] = countdown_tim.min/10+48;
// convert second to ASCII (tens)

                                                                      clock_dsp[1] = countdown_tim.hour%10+48;
          // convert second to ASCII(ones)
                                                                      clock_dsp[0] = countdown_tim.hour/10+48;
// convert second to ASCII (tens)

                                                                      GLCD_GoTo(0,2);

          GLCD_WriteString(count_down_setM,Font_System5x8);                        // display Count Down Menu

                                                                      GLCD_GoTo(0,4);

          GLCD_WriteString(clock_dsp,Font_System5x8);

                                                                      GLCD_GoTo(0,5);
                                                                      GLCD_WriteString(adding,Font_System5x8);
                                                                      GLCD_GoTo(0,6);

          GLCD_WriteString(subbing,Font_System5x8);

                                                                      key  = getcommand();

                                                                      switch (key)
```

```
                                                        {
                                                             case '1':
                                                                   // adds 1 to the minute
value for countdown_tim and resets it back to 0 if the number exceeds 59
                                                                   countdown_tim.min++;
                                                                               if
(countdown_tim.min>59)
                                                                               {

      countdown_tim.min=0;
                                                                               }
                                                                   break;
                                                             case '2':
                                                                   state = cnt_dwn_setS;
                                                                   GLCD_ClearScreen();     //
clear the GLCD

                                                                   break;
                                                             case '3':
                                                                   countdown_tim.min--;
      //takes away 1 from the min value
                                                                   if (countdown_tim.min <= 0)
                                                                   {
                                                                         countdown_tim.min
= 59;
                                                                   }

                                                                   break;
                                                        }

                  break;

      //*********************************************************************
                  case cnt_dwn_setS:      // count down set second
                        clock_dsp[7] = countdown_tim.sec%10+48;       // convert second to
ASCII(ones)
                        clock_dsp[6] = countdown_tim.sec/10+48;   // convert second to ASCII (tens)
```

```
                clock_dsp[4] = countdown_tim.min%10+48;       // convert second to
ASCII(ones)
                clock_dsp[3] = countdown_tim.min/10+48;   // convert second to ASCII (tens)

                clock_dsp[1] = countdown_tim.hour%10+48;      // convert second to
ASCII(ones)
                clock_dsp[0] = countdown_tim.hour/10+48;   // convert second to ASCII
(tens)

                GLCD_GoTo(0,2);
                GLCD_WriteString(count_down_setS,Font_System5x8);              // display
Count Down Menu

                GLCD_GoTo(0,4);
                GLCD_WriteString(clock_dsp,Font_System5x8);

                GLCD_GoTo(0,5);
                GLCD_WriteString(adding,Font_System5x8);
                GLCD_GoTo(0,6);
                GLCD_WriteString(subbing,Font_System5x8);

                key  = getcommand();

                switch (key)
                {
                        case '1':
                                countdown_tim.sec++;
                                        if (countdown_tim.sec>59)
                                        {
                                                countdown_tim.sec=0;
                                        }
                                break;
```

```
                                        case '2':
                                              GLCD_ClearScreen();     // clear the GLCD
                                              state = cnt_dwn;
                                              GLCD_GoTo(0,6);
                                              GLCD_WriteString(start_message,Font_System5x8);
      // user instruction
                                              countdown_flag =! countdown_flag;      //triggers flag



                                              break;
                                        case '3':
                                              countdown_tim.sec--;               //takes away 1 from the
second value
                                              if (countdown_tim.sec <= 0)
                                              {
                                                    countdown_tim.sec = 59;
                                              }

                                              break;
                                  }

                            break;




      //***************************************************************************
                            //************************count up
timer*************************************

      //***************************************************************************
                            case cnt_up:      // count up timer
                                  clock_dsp[7] = countup_tim.sec%10+48;            // convert second to
ASCII(ones)
                                  clock_dsp[6] = countup_tim.sec/10+48;   // convert second to ASCII (tens)
```

```
                clock_dsp[4] = countup_tim.min%10+48;           // convert second to
ASCII(ones)

                clock_dsp[3] = countup_tim.min/10+48;   // convert second to ASCII (tens)


                clock_dsp[1] = countup_tim.hour%10+48;          // convert second to
ASCII(ones)

                clock_dsp[0] = countup_tim.hour/10+48;   // convert second to ASCII (tens)


                /* display String 'clock_dsp'*/
                GLCD_GoTo(0,4);
                GLCD_WriteString(clock_dsp,Font_System5x8);             // user instruction


                key  = getcommand();
                switch (key)
                {
                        case '1':
                        state = ches_clk;
                        GLCD_ClearScreen();      // clear the GLCD
                        GLCD_GoTo(0,2);
                        GLCD_WriteString(ches_clk_menu,Font_System5x8);                 //
display Count Down Menu

                        GLCD_GoTo(0,6);
                        GLCD_WriteString(main_menu_2,Font_System5x8);                   //
user instruction

                                break;
                        case '2':
                                countup_flag =!countup_flag;


                                break;


                        case '3':
                                countup_tim.sec=0;       //reset counter up timer
                                countup_tim.min=0;
                                countup_tim.hour=0;
```

```
                                        break;

                                default:
                                        break;
                        }

                break;

        //***************************************************************************
                //**************************Chess
clock**************************************

        //***************************************************************************
                case ches_clk:          // chess clock
                        clock_dsp[7] = chess_P_One_tim.sec%10+48;     // convert second to
ASCII(ones)
                        clock_dsp[6] = chess_P_One_tim.sec/10+48;   // convert second to ASCII
(tens)

                        clock_dsp[4] = chess_P_One_tim.min%10+48;    // convert second to
ASCII(ones)
                        clock_dsp[3] = chess_P_One_tim.min/10+48;   // convert second to ASCII
(tens)

                        clock_dsp[1] = chess_P_One_tim.hour%10+48;   // convert second to
ASCII(ones)
                        clock_dsp[0] = chess_P_One_tim.hour/10+48;   // convert second to ASCII
(tens)

                        clock_dsptwo[7] = chess_P_two_tim.sec%10+48;            // convert second to
ASCII(ones)
                        clock_dsptwo[6] = chess_P_two_tim.sec/10+48;   // convert second to ASCII
(tens)

                        clock_dsptwo[4] = chess_P_two_tim.min%10+48;           // convert second to
ASCII(ones)
```

```
                    clock_dsptwo[3] = chess_P_two_tim.min/10+48;   // convert second to ASCII
(tens)


                    clock_dsptwo[1] = chess_P_two_tim.hour%10+48;         // convert second to
ASCII(ones)
                    clock_dsptwo[0] = chess_P_two_tim.hour/10+48;   // convert second to ASCII
(tens)



                    GLCD_GoTo(0,4);
                    GLCD_WriteString(clock_dsp,Font_System5x8);              // clock display
                    GLCD_GoTo(60,4);
                    GLCD_WriteString(clock_dsptwo,Font_System5x8);                    // clock
display
                    GLCD_GoTo(0,5);
                    GLCD_WriteString(Player_One,Font_System5x8);                      // player
number display
                    GLCD_GoTo(60,5);
                    GLCD_WriteString(Player_Two,Font_System5x8);                      // player
number display


                    GLCD_GoTo(0,6);
                    GLCD_WriteString(settimeing,Font_System5x8);             // user instruction

                    key  = getcommand();
                    switch (key)
                    {
                         case '1':
                         state = menu;
                         GLCD_ClearScreen();      // clear the GLCD
                         GLCD_GoTo(0,2);
                         GLCD_WriteString(main_menu_1,Font_System5x8);                  //
display Count Down Menu
                         GLCD_GoTo(0,6);
                         GLCD_WriteString(main_menu_2,Font_System5x8);                  //
user instruction
                              break;
```

```
                                        case '2':

                                                state = ches_clk_setH;

                                                GLCD_ClearScreen();      // clear the GLCD

                                                GLCD_GoTo(0,2);

                                                GLCD_WriteString(count_down_setH,Font_System5x8);
        // display Count Down Menu




                                                break;

                                        case '3':


                                                break;



                                        default:

                                                break;

                                }




                        break;



                        case ches_clk_setH:       // chess clock set hour

                                clock_dsp[7] = chess_P_One_tim.sec%10+48;     // convert second to
ASCII(ones)

                                clock_dsp[6] = chess_P_One_tim.sec/10+48;   // convert second to ASCII
(tens)



                                clock_dsp[4] = chess_P_One_tim.min%10+48;    // convert second to
ASCII(ones)

                                clock_dsp[3] = chess_P_One_tim.min/10+48;   // convert second to ASCII
(tens)



                                clock_dsp[1] = chess_P_One_tim.hour%10+48;   // convert second to
ASCII(ones)

                                clock_dsp[0] = chess_P_One_tim.hour/10+48;   // convert second to ASCII
(tens)
```

```
                        clock_dsptwo[7] = chess_P_two_tim.sec%10+48;        // convert second to
ASCII(ones)
                        clock_dsptwo[6] = chess_P_two_tim.sec/10+48;   // convert second to ASCII
(tens)

                        clock_dsptwo[4] = chess_P_two_tim.min%10+48;        // convert second to
ASCII(ones)
                        clock_dsptwo[3] = chess_P_two_tim.min/10+48;   // convert second to ASCII
(tens)

                        clock_dsptwo[1] = chess_P_two_tim.hour%10+48;       // convert second to
ASCII(ones)
                        clock_dsptwo[0] = chess_P_two_tim.hour/10+48;   // convert second to ASCII
(tens)

                        GLCD_GoTo(0,2);
                        GLCD_WriteString(count_down_setH,Font_System5x8);

    //#####################################################################################
######
                        GLCD_GoTo(0,4);
                        GLCD_WriteString(clock_dsp,Font_System5x8);         // clock display
player 1
                        GLCD_GoTo(60,4);
                        GLCD_WriteString(time,Font_System5x8);              // clock display
player 2
                        GLCD_GoTo(0,5);
                        GLCD_WriteString(adding,Font_System5x8);
                        GLCD_GoTo(0,6);
                        GLCD_WriteString(subbing,Font_System5x8);

    //#####################################################################################
######
                        key  = getcommand();
                        switch (key)
                        {
                            case '1':
```

```
                        chess_P_One_tim.hour++;

                        chess_P_two_tim.hour++;

                        if (chess_P_One_tim.hour>23)

                        {

                                chess_P_One_tim.hour=0;

                                chess_P_two_tim.hour=0;

                        }


                        break;

                case '2':

                        state = ches_clk_setM;

                        GLCD_ClearScreen();     // clear the GLCD


                        break;

                case '3':


                        //takes away 1 from the hour value

                        chess_P_One_tim.hour--;

                        chess_P_two_tim.hour--;


                        if (chess_P_One_tim.hour <= 0)

                        {

                                chess_P_One_tim.hour = 23;

                                chess_P_two_tim.hour = 23;

                        }


                        break;

        }

break;


case ches_clk_setM:     // chess clock set minute


        clock_dsp[7] = chess_P_One_tim.sec%10+48;     // convert second to
ASCII(ones)
```

```
                    clock_dsp[6] = chess_P_One_tim.sec/10+48;   // convert second to ASCII
(tens)


                    clock_dsp[4] = chess_P_One_tim.min%10+48;    // convert second to
ASCII(ones)

                    clock_dsp[3] = chess_P_One_tim.min/10+48;   // convert second to ASCII
(tens)


                    clock_dsp[1] = chess_P_One_tim.hour%10+48;   // convert second to
ASCII(ones)

                    clock_dsp[0] = chess_P_One_tim.hour/10+48;   // convert second to ASCII
(tens)


                    clock_dsptwo[7] = chess_P_two_tim.sec%10+48;           // convert second to
ASCII(ones)

                    clock_dsptwo[6] = chess_P_two_tim.sec/10+48;   // convert second to ASCII
(tens)


                    clock_dsptwo[4] = chess_P_two_tim.min%10+48;           // convert second to
ASCII(ones)

                    clock_dsptwo[3] = chess_P_two_tim.min/10+48;   // convert second to ASCII
(tens)


                    clock_dsptwo[1] = chess_P_two_tim.hour%10+48;          // convert second to
ASCII(ones)

                    clock_dsptwo[0] = chess_P_two_tim.hour/10+48;   // convert second to ASCII
(tens)


                    GLCD_GoTo(0,2);

                    GLCD_WriteString(count_down_setM,Font_System5x8);

     //##########################################################################################
######

                    GLCD_GoTo(0,4);

                    GLCD_WriteString(clock_dsp,Font_System5x8);             // clock display
player 1

                    GLCD_GoTo(60,4);

                    GLCD_WriteString(time,Font_System5x8);                  // clock display
player 2
```

```
                                        GLCD_GoTo(0,5);
                                        GLCD_WriteString(adding,Font_System5x8);
                                        GLCD_GoTo(0,6);
                                        GLCD_WriteString(subbing,Font_System5x8);

        //####################################################################################
######
                                        key  = getcommand();
                                        switch (key)
                                        {
                                                case '1':
                                                        chess_P_One_tim.min++;
                                                        chess_P_two_tim.min++;
                                                        if (chess_P_One_tim.min>59)
                                                        {
                                                                chess_P_One_tim.min=0;
                                                                chess_P_two_tim.min=0;
                                                        }

                                                        break;
                                                case '2':
                                                        state = ches_clk_setS;
                                                        GLCD_ClearScreen();     // clear the GLCD

                                                        break;
                                                case '3':
                                                        //takes away 1 from the hour value
                                                        chess_P_One_tim.min--;
                                                        chess_P_two_tim.min--;

                                                        if (chess_P_One_tim.min <= 0)
                                                        {
                                                                chess_P_One_tim.min = 59;
                                                                chess_P_two_tim.min = 59;
```

```
                                }

                        }
                                break;

                break;


                case ches_clk_setS:     // chess clock set second

                        clock_dsp[7] = chess_P_One_tim.sec%10+48;     // convert second to
ASCII(ones)
                        clock_dsp[6] = chess_P_One_tim.sec/10+48;   // convert second to ASCII
(tens)

                        clock_dsp[4] = chess_P_One_tim.min%10+48;    // convert second to
ASCII(ones)
                        clock_dsp[3] = chess_P_One_tim.min/10+48;   // convert second to ASCII
(tens)

                        clock_dsp[1] = chess_P_One_tim.hour%10+48;   // convert second to
ASCII(ones)
                        clock_dsp[0] = chess_P_One_tim.hour/10+48;   // convert second to ASCII
(tens)

                        clock_dsptwo[7] = chess_P_two_tim.sec%10+48;            // convert second to
ASCII(ones)
                        clock_dsptwo[6] = chess_P_two_tim.sec/10+48;   // convert second to ASCII
(tens)

                        clock_dsptwo[4] = chess_P_two_tim.min%10+48;           // convert second to
ASCII(ones)
                        clock_dsptwo[3] = chess_P_two_tim.min/10+48;   // convert second to ASCII
(tens)

                        clock_dsptwo[1] = chess_P_two_tim.hour%10+48;         // convert second to
ASCII(ones)
```

```
                                        clock_dsptwo[0] = chess_P_two_tim.hour/10+48;   // convert second to ASCII
(tens)


                                        GLCD_GoTo(0,2);
                                        GLCD_WriteString(count_down_setS,Font_System5x8);

        //##########################################################################################
######
                                        GLCD_GoTo(0,4);
                                        GLCD_WriteString(clock_dsp,Font_System5x8);             // clock display
player 1
                                        GLCD_GoTo(60,4);
                                        GLCD_WriteString(time,Font_System5x8);                  // clock display
player 2
                                        GLCD_GoTo(0,5);
                                        GLCD_WriteString(adding,Font_System5x8);
                                        GLCD_GoTo(0,6);
                                        GLCD_WriteString(subbing,Font_System5x8);

        //##########################################################################################
######
                                        key  = getcommand();
                                        switch (key)
                                        {
                                                case '1':
                                                        chess_P_One_tim.sec++;
                                                        chess_P_two_tim.sec++;
                                                        if (chess_P_One_tim.sec>59)
                                                        {
                                                                chess_P_One_tim.sec=0;
                                                                chess_P_two_tim.sec=0;
                                                        }

                                                        break;
                                                case '2':
                                                        state = ches_clk_p1p2_run;
```

```
                    GLCD_ClearScreen();     // clear the GLCD
                    chesscountdown_flag = 0;

                    break;
                case '3':
                    //takes away 1 from the hour value
                    chess_P_One_tim.sec--;
                    chess_P_two_tim.sec--;

                    if (chess_P_One_tim.sec <= 0)
                    {
                        chess_P_One_tim.sec = 59;
                        chess_P_two_tim.sec = 59;
                    }

            }
                    break;

        break;

        case ches_clk_p1p2_run:         // chess clock switch player

            clock_dsp[7] = chess_P_One_tim.sec%10+48;    // convert second to
ASCII(ones)
            clock_dsp[6] = chess_P_One_tim.sec/10+48;  // convert second to ASCII
(tens)

            clock_dsp[4] = chess_P_One_tim.min%10+48;    // convert second to
ASCII(ones)
            clock_dsp[3] = chess_P_One_tim.min/10+48;  // convert second to ASCII
(tens)

            clock_dsp[1] = chess_P_One_tim.hour%10+48;  // convert second to
ASCII(ones)
            clock_dsp[0] = chess_P_One_tim.hour/10+48;  // convert second to ASCII
(tens)
```

```
                              clock_dsptwo[7] = chess_P_two_tim.sec%10+48;          // convert second to
ASCII(ones)
                              clock_dsptwo[6] = chess_P_two_tim.sec/10+48;  // convert second to ASCII
(tens)

                              clock_dsptwo[4] = chess_P_two_tim.min%10+48;          // convert second to
ASCII(ones)
                              clock_dsptwo[3] = chess_P_two_tim.min/10+48;  // convert second to ASCII
(tens)

                              clock_dsptwo[1] = chess_P_two_tim.hour%10+48;         // convert second to
ASCII(ones)
                              clock_dsptwo[0] = chess_P_two_tim.hour/10+48;  // convert second to ASCII
(tens)


       //##########################################################################################
######
                              GLCD_ClearScreen();     // clear the GLCD
                              GLCD_GoTo(0,4);
                              GLCD_WriteString(clock_dsp,Font_System5x8);           // clock display
player 1
                              GLCD_GoTo(60,4);
                              GLCD_WriteString(clock_dsptwo,Font_System5x8);               // clock
display player 2
                              GLCD_GoTo(0,5);
                              GLCD_WriteString(Player_One,Font_System5x8);                 // player 1
name tag
                              GLCD_GoTo(60,5);
                              GLCD_WriteString(Player_Two,Font_System5x8);                 // player 2
name tag
                              GLCD_GoTo(40,1);
                              GLCD_WriteString(play,Font_System5x8);              // player 2 name tag

       //##########################################################################################
######
                              //checks if the time is more than zero and if the flag is 0
```

```
                    if(chesscountdown_flag == 0 && (chess_P_One_tim.sec > 0 || 
chess_P_One_tim.min > 0 || chess_P_One_tim.hour || 0))
                    {
                         GLCD_GoTo(0,2);
                         GLCD_WriteString(Pone_turn,Font_System7x8);          // displays 
player one turn
                    }
                    //checks if the time is more than zero and if the flag is 1
                    if(chesscountdown_flag == 1 && (chess_P_two_tim.sec > 0 || 
chess_P_two_tim.min > 0 || chess_P_two_tim.hour || 0))
                    {
                         GLCD_GoTo(0,2);
                         GLCD_WriteString(Ptwo_turn,Font_System7x8);          // displays 
player two turn
                    }

                    //checks if the time is zero and if the flag is 3
                    if (chesscountdown_flag == 3 && chess_P_One_tim.sec == 0 && 
chess_P_One_tim.min == 0 && chess_P_One_tim.hour == 0)      //checks if the count down flag is equal to 0
                    {

                         GLCD_GoTo(0,2);                                 //goes to the 0,6 
position on the LCD
                         GLCD_WriteString(Ptwo_win,Font_System7x8);           // displays 
player two win

                    }
                    //checks if the time is zero and if the flag is 3
                    if (chesscountdown_flag == 3 && chess_P_two_tim.sec == 0 && 
chess_P_two_tim.min == 0 && chess_P_two_tim.hour == 0)      //checks if the count down flag is equal to 0
                    {

                         GLCD_GoTo(0,2);                                 //goes to the 0,6 
position on the LCD
                         GLCD_WriteString(Pone_win,Font_System7x8);           // displays 
player one win
```

```
                        }

                        GLCD_GoTo(0,6);                         //goes to the 0,6 position on
the LCD
                        GLCD_WriteString(resetingbutton,Font_System5x8);            // displays
player one win

                        key  = getcommand();
                        switch (key)
                        {
                                case '1':

                                        break;
                                case '2':
                                        chesscountdown_flag =! chesscountdown_flag;

                                        break;
                                case '3':

                                        chess_P_One_tim.sec=0;          //reset player 1 timer
                                        chess_P_One_tim.min=0;
                                        chess_P_One_tim.hour=0;
                                        chess_P_two_tim.sec=0;          //reset player 2 timer
                                        chess_P_two_tim.min=0;
                                        chess_P_two_tim.hour=0;

                                        state = ches_clk;
                                        break;
                        }

                break;
```

```
                        default:

                        break;


            }


}

////////////////////////////////////////////////////////////////////////////////////

ISR

////////////////////////////////////////////////////////////////////////////////////

/******************* (C) COPYRIGHT 2018-2019 University of South Wales
********************************

* File Name        : ISR.c

* Author           : SS

* Version          : V1.0.2

* Date             : 21-August-2019

* Description      : (User) Interrupt Service Routine Definitions

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

* THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING STUDENTS

* WITH CODING INFORMATION REGARDING THEIR ASSIGNMENT IN ORDER FOR THEM DELIVER TO THE
STANDARD.

* AS A RESULT, UNIVERSITY OF SOUTH WALES SHALL NOT BE HELD LIABLE FOR ANY DIRECT,

* INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE

* CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY STUDENTS OF THE CODING

* INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS,PROJECT OR ASSIGNMENTS.

******************************************************************************************
***********/


#include "iodefine.h"

#include "typedefine.h"

#include "d_stop_watch.h"
```

```
extern clock_t countup_tim;                    // temp timer structures for countup clock
extern clock_t countdown_tim;          // count down clock
extern clock_t chess_P_One_tim;                // chess player 1 clock
extern clock_t chess_P_two_tim;                // chess player 2 clock

unsigned int ms_count = 0;



unsigned char toggle_led = 0;




//****************************FLAGS*******************************************
unsigned char countup_flag = 0;          //initialise flag
//unsigned char Chess_Player_One = 0;  //initialise flag
//unsigned char Chess_Player_Two = 0;  //initialise flag
unsigned char countdown_flag = 0;
unsigned char alarm_flag = 0;
unsigned char chesscountdown_flag = 2;
//**************************************************************************



/* External Interrupts */
#pragma interrupt (sw1_isr (vect = VECT(ICU, IRQ4)))      // Vector = 68
void sw1_isr (void)
{


}


#pragma interrupt (sw2_isr (vect = VECT(ICU, IRQ5)))    // Vector = 69
void sw2_isr (void)
{
}


#pragma interrupt (sw3_isr (vect = VECT(ICU, IRQ7)))    // Vector = 71

void sw3_isr (void)

{



}



/* TPU Timer Interrupts  */

#pragma interrupt (tpu0_isr (vect = VECT(TPU0, TGI0A)))   // Vector = 126

void tpu0_isr (void)

{



}


#pragma interrupt (tpu1_isr (vect = VECT(TPU1, TGI1A)))  // Vector = 130

void tpu1_isr (void)

{

            if (++ms_count>999)                   //a count to create a delay that will count for 1 second and then
continue the code

            {

                        ms_count = 0;    // resets the count


                        if (alarm_flag==1)           // will make the led alarm start to flash when the flag has been
raised.

                        {

                                    toggle_led =! toggle_led;

                                    PORTE.PODR.BYTE = toggle_led;

                        }
```

```
                    //*********************************COUNT DOWN
CODE*****************************************************
                    if(countdown_flag==1)
                                        {
                                         if(countdown_tim.sec==0)
                                         {
                                                   countdown_tim.sec = 59;
                                                   if(countdown_tim.min==0)
                                                   {
                                                             countdown_tim.min = 59;
                                                             if(countdown_tim.hour==0)
                                                             {

                                                                       //******************dont use
this*********************
                                                                       //countdown_tim.hour = 23;

//******************************************************

                                                                       //YADA YADA FLASH ALARM CODE
PUT HERE
                                                                       PORTE.PODR.BYTE = toggle_led = 0;
                                                                       countdown_flag =! countdown_flag;
                    //sets flag to logic 0 and stops the countdown
                                                                       countdown_tim.min = 0;
                                                                       countdown_tim.sec = 0;
                                                                       alarm_flag =! alarm_flag;


                                                             }
                                                             else
                                                             {
                                                                       countdown_tim.hour =
countdown_tim.hour - 1;
                                                             }
                                                   }
                                        else
                                        {
                                                countdown_tim.min = countdown_tim.min -
1;
                                        }
                                }
                                else
                                {
                                        countdown_tim.sec = countdown_tim.sec - 1;
                                }
                               }




//*************************************COUNT UP
CODE*****************************************************

        if(countup_flag==1)                 // checks if flag has been raised
        {
         if(countup_tim.sec>=59)           //checks if the time is 60 seconds
         {
                countup_tim.sec = 0;
                if(countup_tim.min>=59)           //checks if the time is 60 minutes
                {
                        countup_tim.min = 0;
                        if(countup_tim.hour>=23)         //checks if the time is 24 hours
                        {
                                countup_tim.hour = 0;
                        }
                        else
                        {
                                countup_tim.hour = countup_tim.hour + 1;
                        }
                }
```

```
                    else
                    {
                            countup_tim.min = countup_tim.min + 1;
                    }
            }
            else
            {
                    countup_tim.sec = countup_tim.sec+1;
            }
           }


    //****************************************CHESS COUNT DOWN
CODE**********************************************


            //PLAYER ONE SECTION
                    if(chesscountdown_flag == 0 )    //checks if the flag is equal to 0 which is player 1
                    {
                            if(chess_P_One_tim.sec==0)
                            {
                                    chess_P_One_tim.sec = 59;
                                    if(chess_P_One_tim.min==0)
                                    {
                                            chess_P_One_tim.min = 59;
                                            if(chess_P_One_tim.hour==0)
                                            {

                                                    //FLASH ALARM CODE
                                                    chess_P_One_tim.min = 0;                //resets the
time to zero to stop 00:59:59 issue
                                                    chess_P_One_tim.sec = 0;
                                                    alarm_flag =! alarm_flag;               // triggers
alarm flag
                                                    chesscountdown_flag = 3;                //SETS THE
FLAG TO A VALUE THAT WILL NOT TRIGGER EITHER PLAYERS
```

```
                                                }
                                                else
                                                {
                                                        chess_P_One_tim.hour = chess_P_One_tim.hour - 1;
                                                }
                                        }
                                        else
                                        {
                                                chess_P_One_tim.min = chess_P_One_tim.min - 1;
                                        }
                                }
                                else
                                {
                                        chess_P_One_tim.sec = chess_P_One_tim.sec - 1;
                                }
                        }

                        //PLAYER 2 SECTION
                        else if(chesscountdown_flag == 1 )        //checks if the flag is equal to 0 which is
player 1
                        {
                                if(chess_P_two_tim.sec==0)
                                {
                                        chess_P_two_tim.sec = 59;
                                        if(chess_P_two_tim.min==0)
                                        {
                                                chess_P_two_tim.min = 59;
                                                if(chess_P_two_tim.hour==0)
                                                {
                                                        //FLASH ALARM CODE
                                                        chess_P_two_tim.min = 0;              //resets the
time to zero to stop 00:59:59 issue

                                                        chess_P_two_tim.sec = 0;
```

```
                            alarm_flag =! alarm_flag;            // triggers
alarm flag
                            chesscountdown_flag = 3;             //SETS THE
FLAG TO A VALUE THAT WILL NOT TRIGGER EITHER PLAYERS
                        }
                        else
                        {
                            chess_P_two_tim.hour = chess_P_two_tim.hour - 1;
                        }
                    }
                    else
                    {
                        chess_P_two_tim.min = chess_P_two_tim.min - 1;
                    }
                }
                else
                {
                    chess_P_two_tim.sec = chess_P_two_tim.sec - 1;
                }
            }
    }
}

#pragma interrupt (tpu2_isr (vect = VECT(TPU2, TGI2A)))  // Vector = 132
void tpu2_isr (void)
{

}

#pragma interrupt (tpu3_isr (vect = VECT(TPU3, TGI3A)))  // Vector = 134
void tpu3_isr (void)
{
}



#pragma interrupt (tpu4_isr (vect = VECT(TPU4, TGI4A)))  // Vector = 138
void tpu4_isr (void)
{



}



#pragma interrupt (tpu5_isr (vect = VECT(TPU5, TGI5A)))  // Vector = 140
void tpu5_isr (void)
{



}

/*ADC Interrupts */

/* 12 bit ADC */
#pragma interrupt (sa12ad0_isr (vect = VECT(S12AD0, S12ADI0)))  // Vector = 102
void sa12ad0_isr (void)
{



}



#pragma interrupt (adi0_isr (vect = VECT(AD0, ADI0)))   // Vector = 98
void adi0_isr (void)
{
}
```