2017

Assignment 2

ENS2257: MICROPROCESSOR SYSTEMS

DAVID DUKE – 10374985
QJ STOUT-SPYKERS – 10223002

Table of Contents

Figures

Tables

Task 1

The task requires interfacing a 4x3 numeric keypad with the ATmega168 to take user input for controlling an array of three LEDs, which will act as a simplified proxy for a set of motors. The program design consists of eight LED modes, including the user defined mode, covered in Task 2, each activated by the buttons on the keypad. The program will stop at any point when the emergency stop button is pressed and set the program to an emergency state that must be cleared by pressing the * key on the keypad.

Circuit diagram

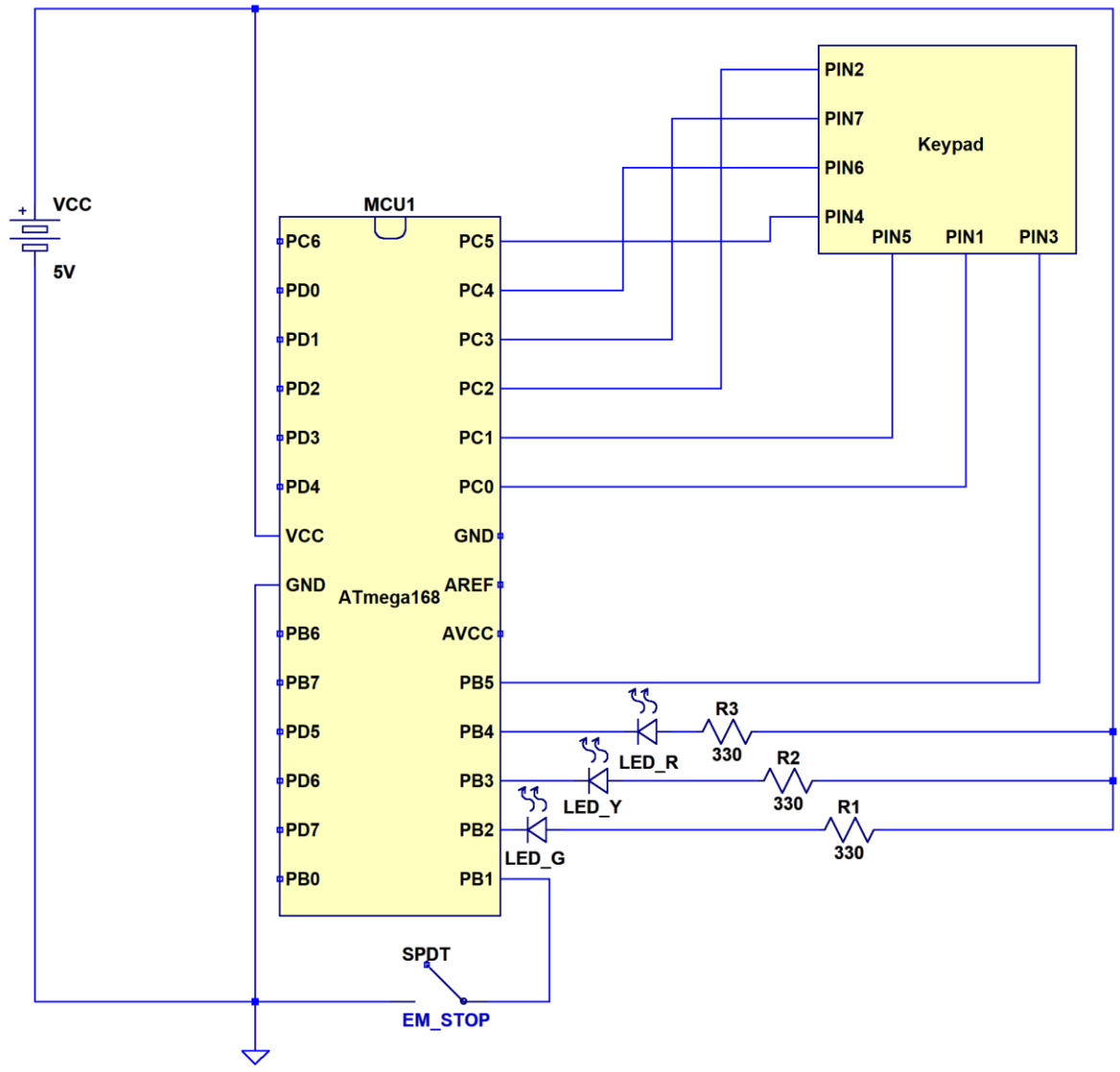


Figure 1: Circuit diagram for the keypad, LED and stop button interfacing

Interfacing requirements

The keypad package contains seven pins, 4 pins corresponding with the keypad rows and three pins corresponding to the keypad columns. When one of the keys is pressed, a connection is made between one of the rows and columns of the keypad.

Table 1: I/O register setup for interfacing with the keypad

MCU pin	Keypad Pin	Keypad row/column	DDRB/C settings		PORTB/C		Comments
PB5	Pin 3	first column	DDRB5	1	PORTB5	X	Set/clear in code
PC0	Pin 1	middle column	DDRC0	1	PORTC0	X	
PC1	Pin 5	third column	DDRC1	1	PORTC1	X	
PC2	Pin 2	first row	DDRC2	0	PORTC2	1	Pull up enable
PC3	Pin 7	second row	DDRC3	0	PORTC3	1	
PC4	Pin 6	third row	DDRC4	0	PORTC4	1	
PC5	Pin 4	fourth row	DDRC5	0	PORTC5	1	

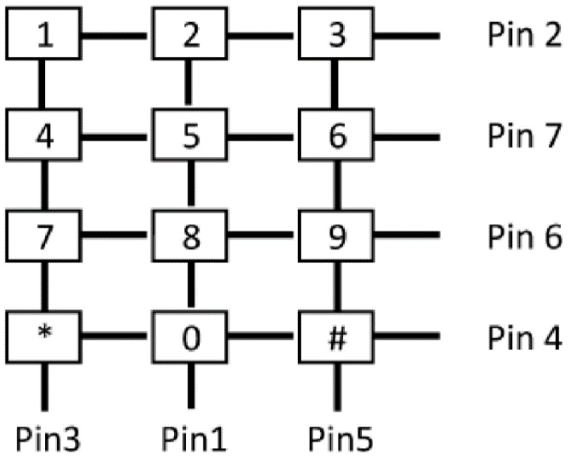


Figure 2: keypad layout from workshop 6

The colums are tied to the PORTC pins , 1 and 2 whilst the rows are tied to the PINC and PINB pins with the pullup resistors enabled. When one of the keys is depressed, a connection will be made between a row and a column on the keypad. The active low output on the column pins drives an active low signal to the corresponding input pins on any of the connected rows, which are then read through the PINB/C registers. The keypad detects button presses using the code supplied in workshop 6, modified to detect button presses from all four rows, therefore it was not necessary to design a flowchart for implementing the keypad functionality.

Table 2: LED and emergency stop button I/O configurations

MCU pin	Component	DDRB		PORTB		Comments
PB1	Emergency stop	DDRB1	0	PORTB1	1	Pull up enable
PB2	Green LED	DDRB2	1	PORTB2	X	Set/Clear in code
PB3	Yellow LED	DDRB3	1	PORTB3	X	
PB4	Red LED	DDRB4	1	PORTB4	X	

Principles of operation

The main program selection is completed with a continuous loop that handles two repeating tasks. First the keypad is checked for user input, then one of the eight programs is selected according to the most recent key press.

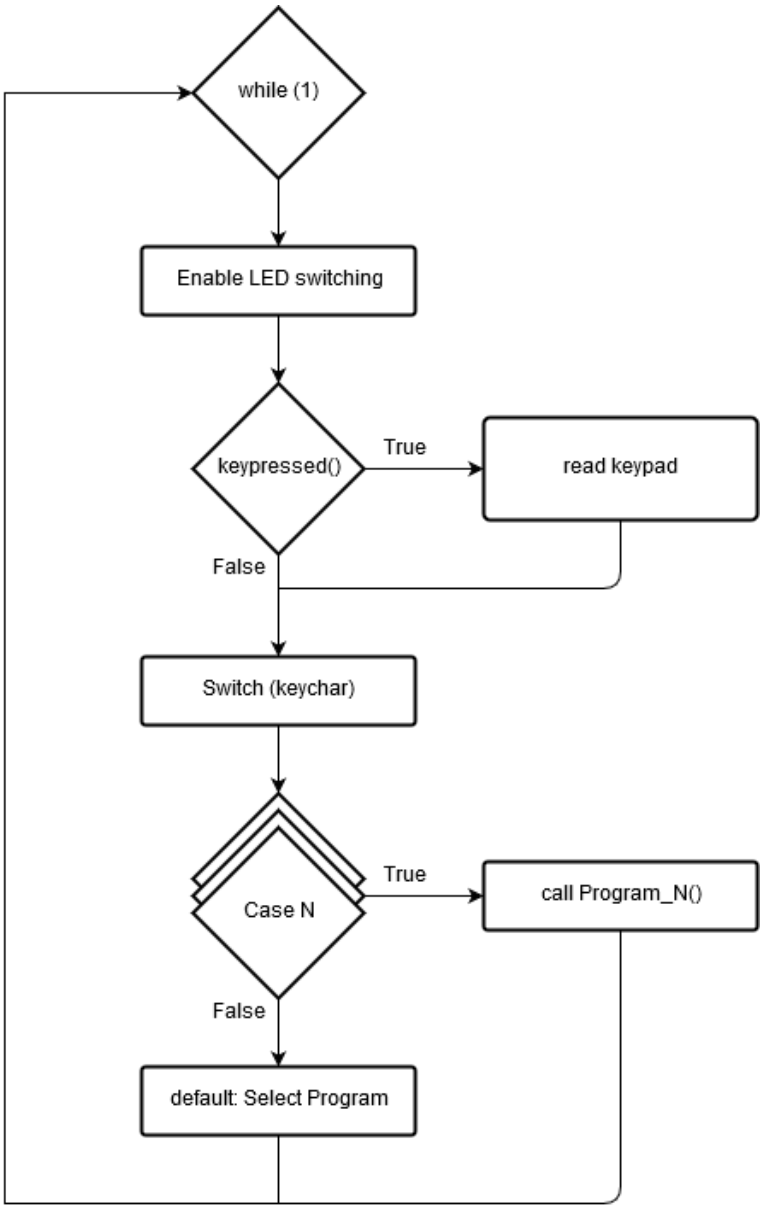


Figure 3: Flowchart of the main program loop

This method uses one main while loop and calls any one of the programs from the switch case only once per loop so that a new program can be selected after a key is pressed. This was chosen so that the code could be interrupted at any point by the emergency ISR without becoming stuck if the program flags/variables are initialized and the program counter returns to the location the ISR was called.

LEDs

The functionality of the LED.c code is to switch any one of the three coloured LEDs with the LED_ON function and to switch off the LEDs with the LED_OFF function. To switch on one of the coloured LEDs, a char specifying the LED colour is passed to the LED_ON function. The LED_ENABLED variable is checked in the LED_ON function allows the bypassing the LED selection code when the LEDs need to be temporarily disabled from running.

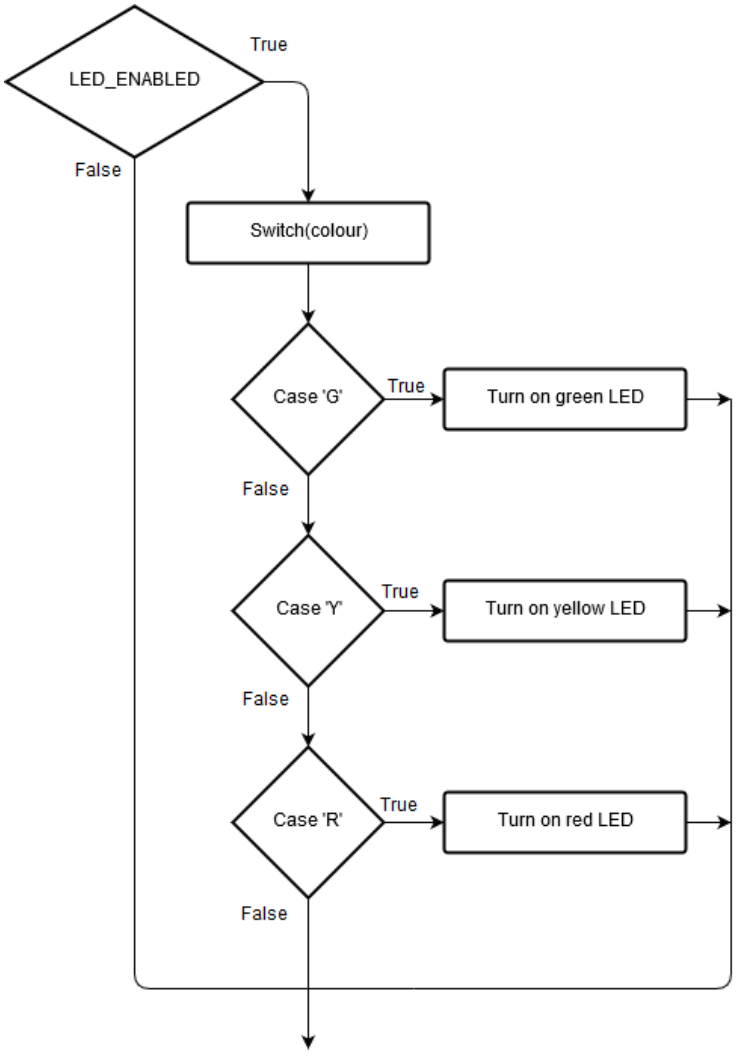


Figure 4: Flowchart of the LED_ON function

Functions defined in: LED.c

init_LEDS()

Sets DDRB and PORTB registers according to the register configuration to initialise the LEDs. The PORTB bits for the LEDs are initially set high (OFF).

LED_ON(char colour)

Flowchart shows behaviour of LED_ON function. Sets PORTB bits to active low (to turn LED on) depending on the LED colour argument passed into the function.

LED_DISABLE()/LED_ENABLE()

The two functions set or clear the LED_ENABLED variable to 0 or 1.

LED_OFF()

The LED_OFF function disables any of the LEDs by setting a high value to the PORTB bits controlling the LEDs.

Timer

Functions defined in: timer.c

The timer code used has been adapted from the timer task in assignment 1. When a timer is set, the TCNT1 register will count to the number specified in the OCR1A and trigger an interrupt every second. The interrupt service routine called by the timer then decrements the time variables. Once the minutes and seconds variables reach zero, the timer ISR code disables the timer interrupts.

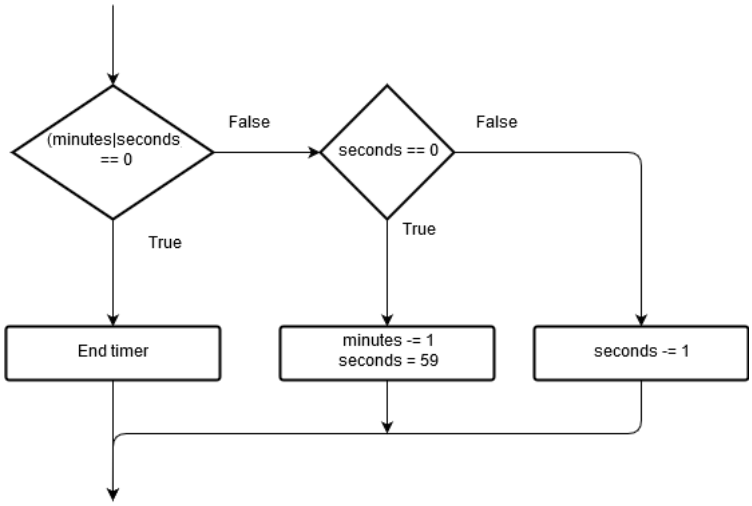


Figure 5: Flowchart for the timer ISR

init_timer():

Clears timer/counter control register A (TCCR1A).

Sets the timer/counter control register B (TCCR1B) bits WGM12, CS12, CS10 to enable CTC mode and clock pre-scale (clk/1024).

Sets OCR1A count to 14399 (one second count).

Zeros timer/counter 1 (TCNT1) register.

Enables timer/counter interrupt mask (TIMSK1) & global interrupts.

GET_MINUTES()/GET_SECONDS(): Returns the minutes and seconds variables declared in timer.c.

TIMER_IS_CLEAR(): Returns zero (false) if running and one (true) if timer is not running.

ADD_MINS()/SECS(): Adds extra time to the minutes and seconds variables in timer.c.

CLEAR_TIMER(): Sets timer minutes and seconds variables to zero, thereby ending the timer.

SET_TIMER(min, sec):

Sets MINUTES and SECONDS variables to the arguments passed into the function and then calls the init_timer() function to ready/run the timer.

PAUSE_TIMER()/RESUME_TIMER():

The two functions enable and disable the CS12/CS10 bits in the TCCR1B register to toggle the timer/counter clock source, essentially pausing the timer/counter.

ISR(TIMER1_COMPA_vect):

Decrements the MINUTES and SECONDS variables by the appropriate amount whenever the ISR is called (every second in this case) and disables the timer/counter once both variables reach zero.

Emergency stop ISR

Defined in: Assignment2.c

The emergency stop button is used to call the emergency stop ISR at any point during the program operation. Before waiting for the reset key to be pressed the emergency ISR turns off/disables the LEDs, clears timers, flags and interrupts so that the program will return to the menu select screen once the ISR is dismissed. When the ISR is running, the while loop locks the program into the ISR until '*' key is pressed, effectively preventing the microcontroller from doing anything whilst in the emergency mode

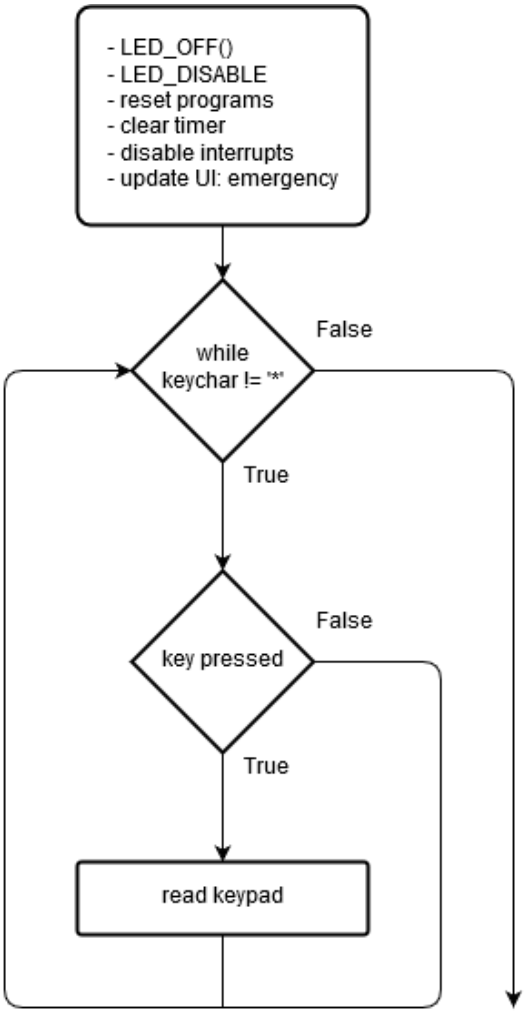


Figure 6: Flowchart of the emergency stop ISR

Program One & Two (sequence)

Defined in Assignment2.c

The two LED sequence programs are initiated with the ‘1’ and ‘2’ keys on the keypad and are composed of three main parts.

First the program is checked against the PREV_PROG and CURR_PROG variables to determine whether the program is already running or to be initialised.

The current stage in the LED timer sequence is selected with a switch case based on the persistent ‘currState’ variable, which is then updated to the next state, ready for the next program loop.

The final switch case in the program one flowchart is to update the user interface on the LED to reflect the current state of the program. Program two is based on the same design as with the switch cases adjusted accordingly.

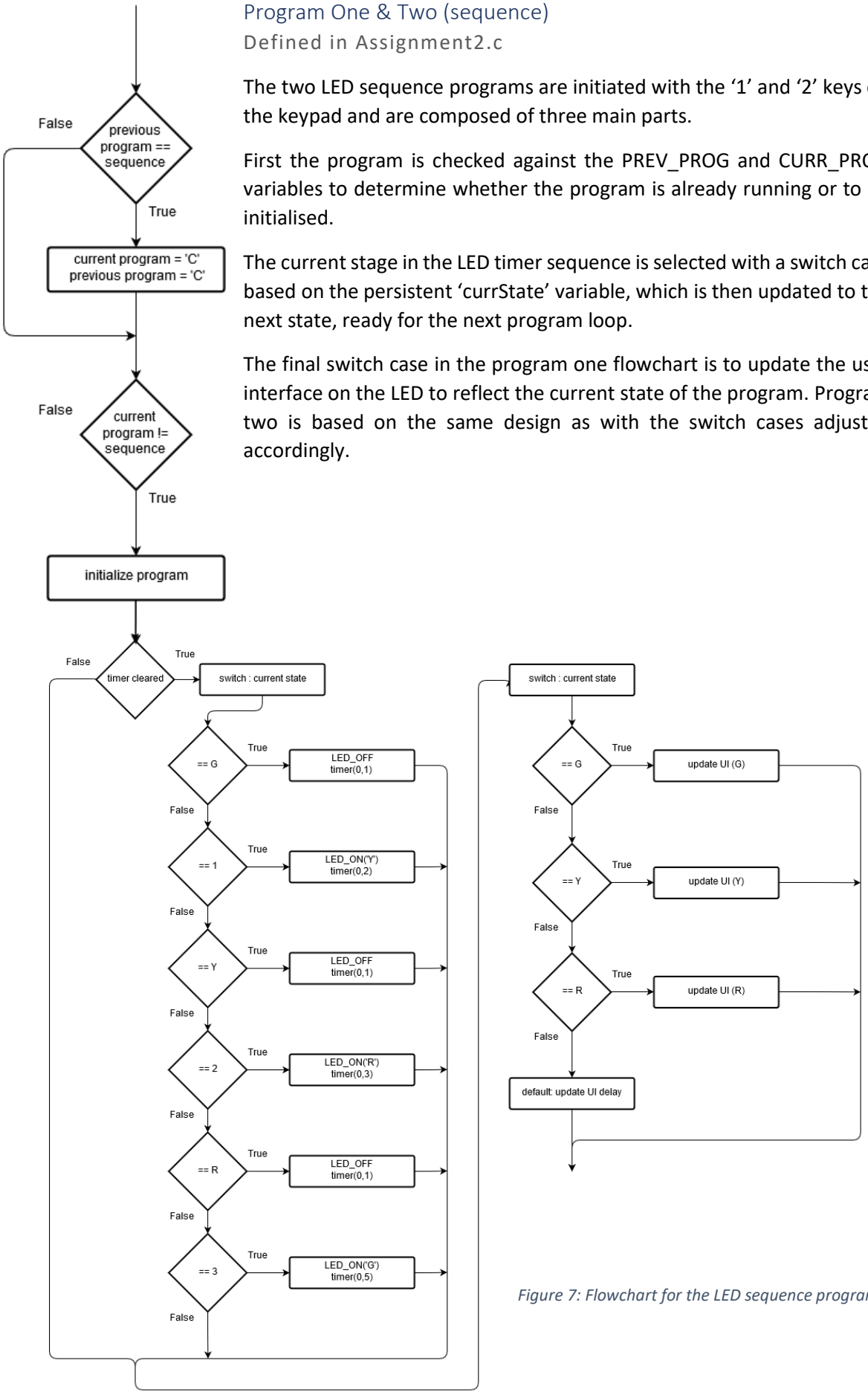


Figure 7: Flowchart for the LED sequence program one.

Program Four & Six (extend)

Defined in Assignment2.c

The two extend programs switch on the specified LED for five seconds and extends the timer by five additional seconds when the program key is pressed before the timer has run out.

The PREV_PROG variable is checked to determine whether the program is running. If the program is still running when the associated keypad key is pressed, then five extra seconds will be added to the timer.

After the initial program check, there is another branch that initialises the program when it is run for the first time.

Finally, the TIMER_IS_CLEAR () and the local 'isReady' flags are checked to set the LED, update the UI and turn the LED off one the timer runs out.

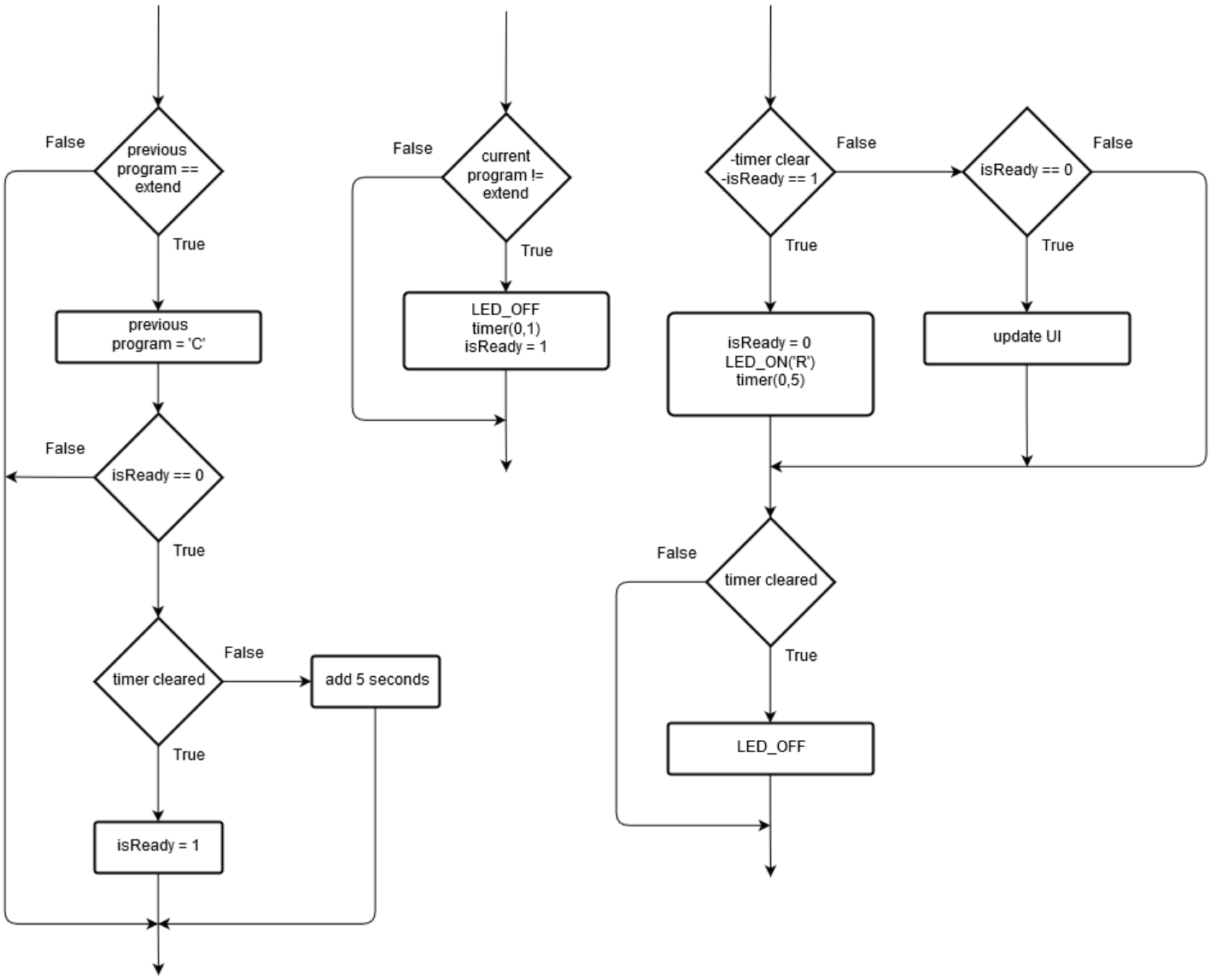


Figure 8: Flowchart of programs four and six (flows left to right)

Programs Seven, Eight & Nine (toggle)

The three programs initiated with the 7, 8 and 9 keys, each toggling one of the three coloured LEDs on an off when they key is repeatedly pressed.

First step in in the flowchart is checking whether the program has been run before. If the program has been run before, then a variable is simply toggled between zero and one, depending on the current state of the variable. If the program has not been run before, then the actions in the second flowchart block are taken to initialise the program for the first run. The last part of the flowchart checks to see if the LED is on the one second cooldown, then updates the LCD and switches the state of the LED before exiting the program for the next loop.

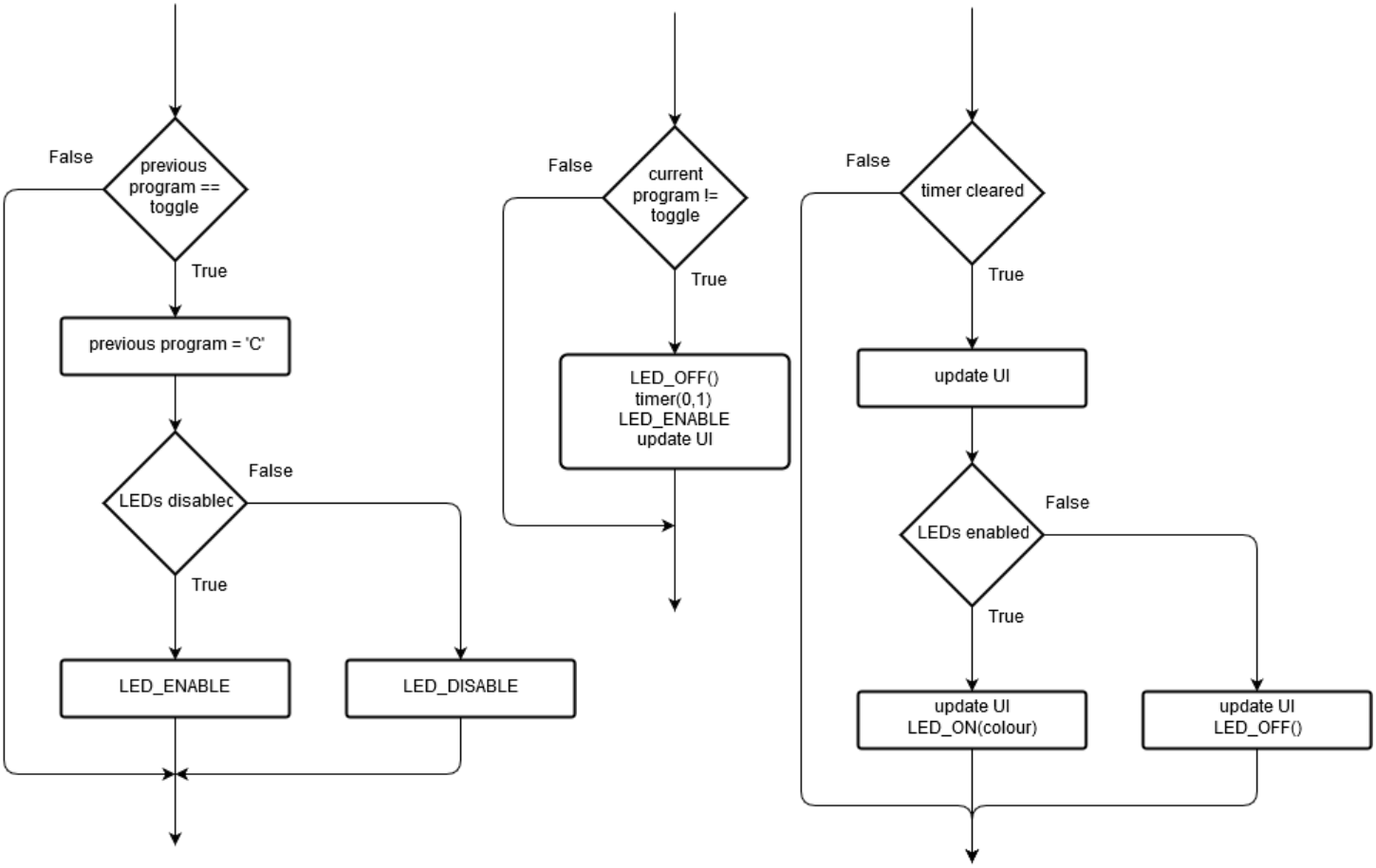


Figure 9: Flowchart of the design of programs 7, 8 and 9 (program flows from left to right)

Task 2 – User defined function

Defined in Assignment2.c

The user defined program in task 2 is required to accept user input for the timer, up to a maximum of 80 minutes and then turn the red LED on for that specified amount of time. Then, once the timer is finished counting down, the program must return to the main menu.

Principles of operation

The overall flowchart for the user defined program the following flowchart blocks correspond with the sub-flowcharts of the same name.

The user defined program must take user input for the timer without interfering with the program selection. A simple way to do this was to lock into the user defined program with a while loop that waits for the current program to be cleared, either by cancellation of the program by the user or automatically when the timer runs out.

There is a total of six states this program can be in, ranging from 0 to 5. The first two states, zero & one, are associated with taking the user input for the timer minutes digits, whilst states two & three are associated with the handling of the seconds digits. When a digit is successfully entered the timer, the program is shifted up to the next state.

Once all four digits have been successfully entered, the program moves to the fourth state for validating the user input and initialising the timer and the LED when a valid input has been detected.

State five is initiated when the timer and LED have been successfully set up and have started running, the main task for the running state is to update the timer on the UI and to detect a press of the # key to pause the program or a press of the * key to return the user to the main menu before the timer has expired. The last step in the running of the is to update the user interface on the LCD to match the state of the program.

User defined program design

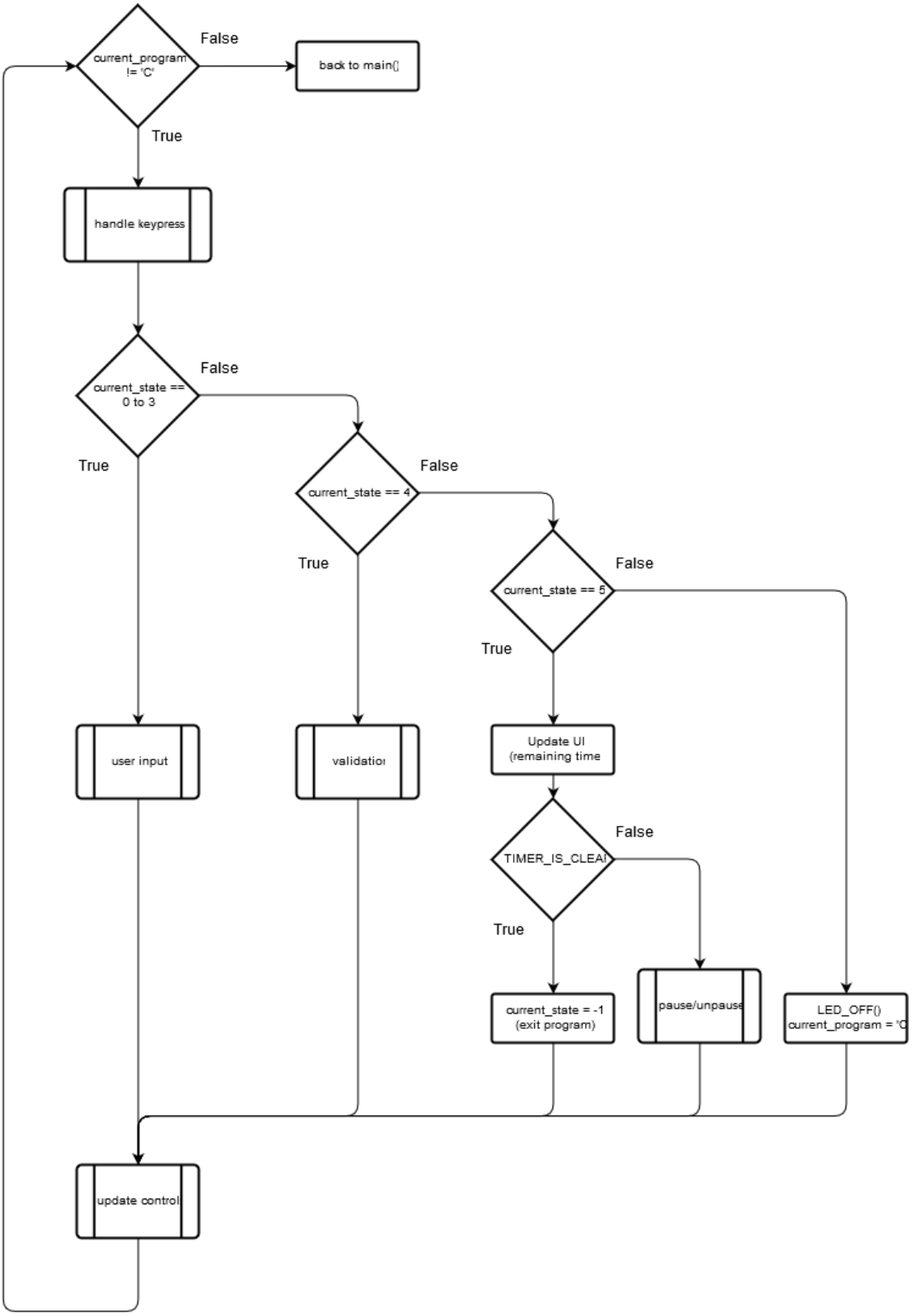
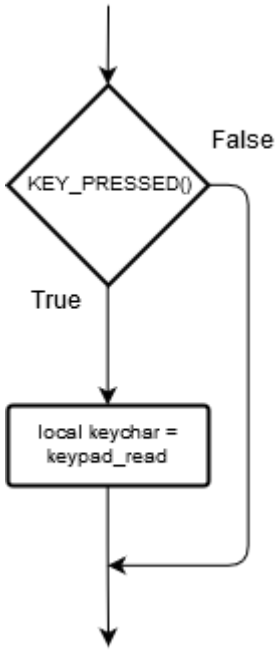
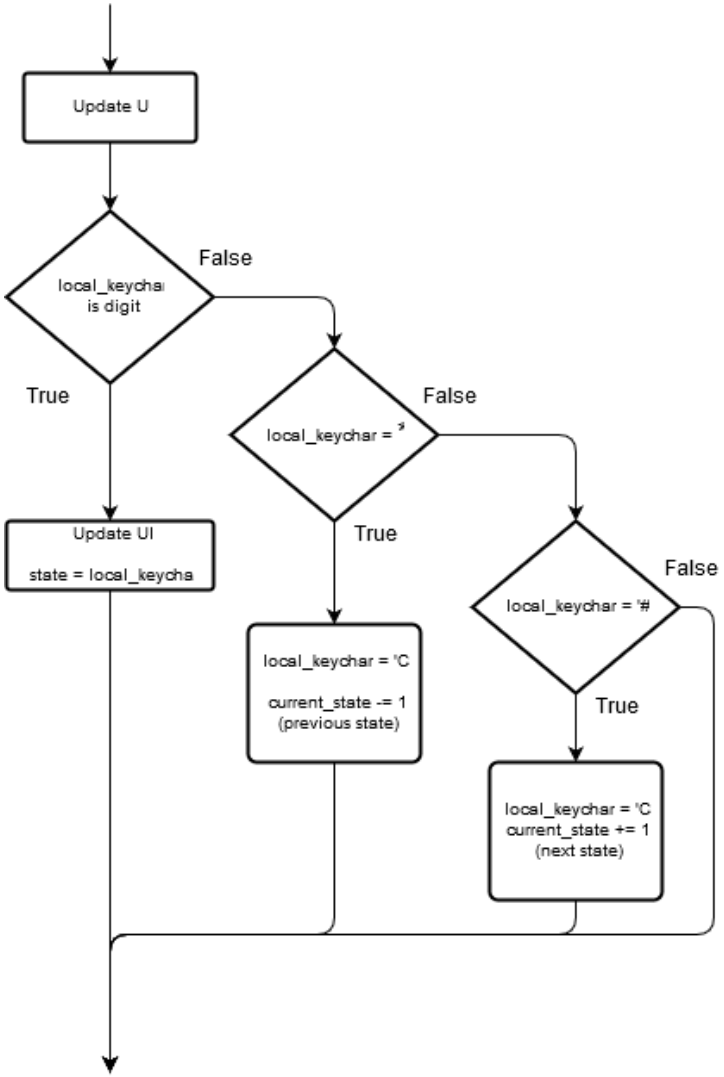
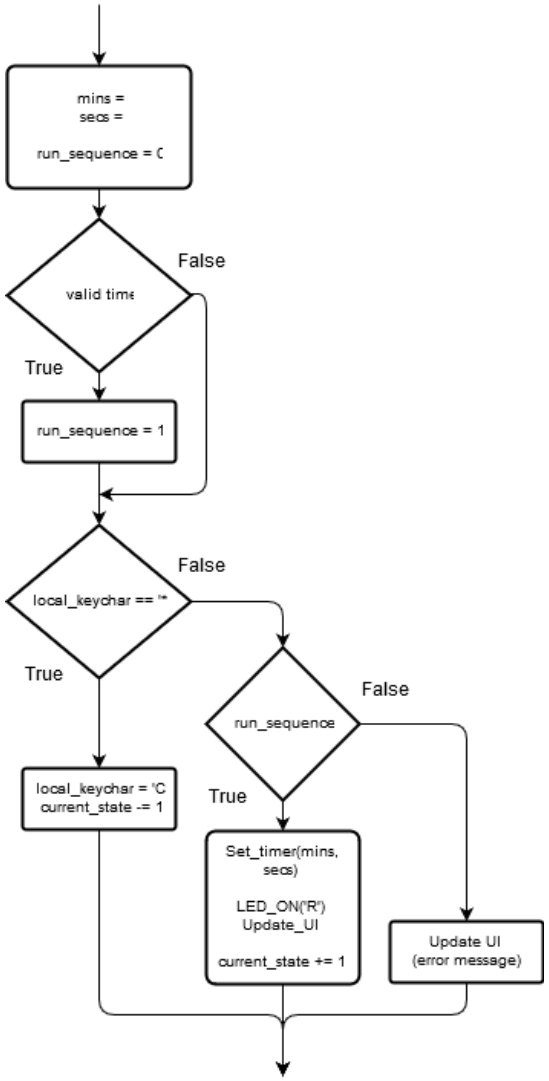
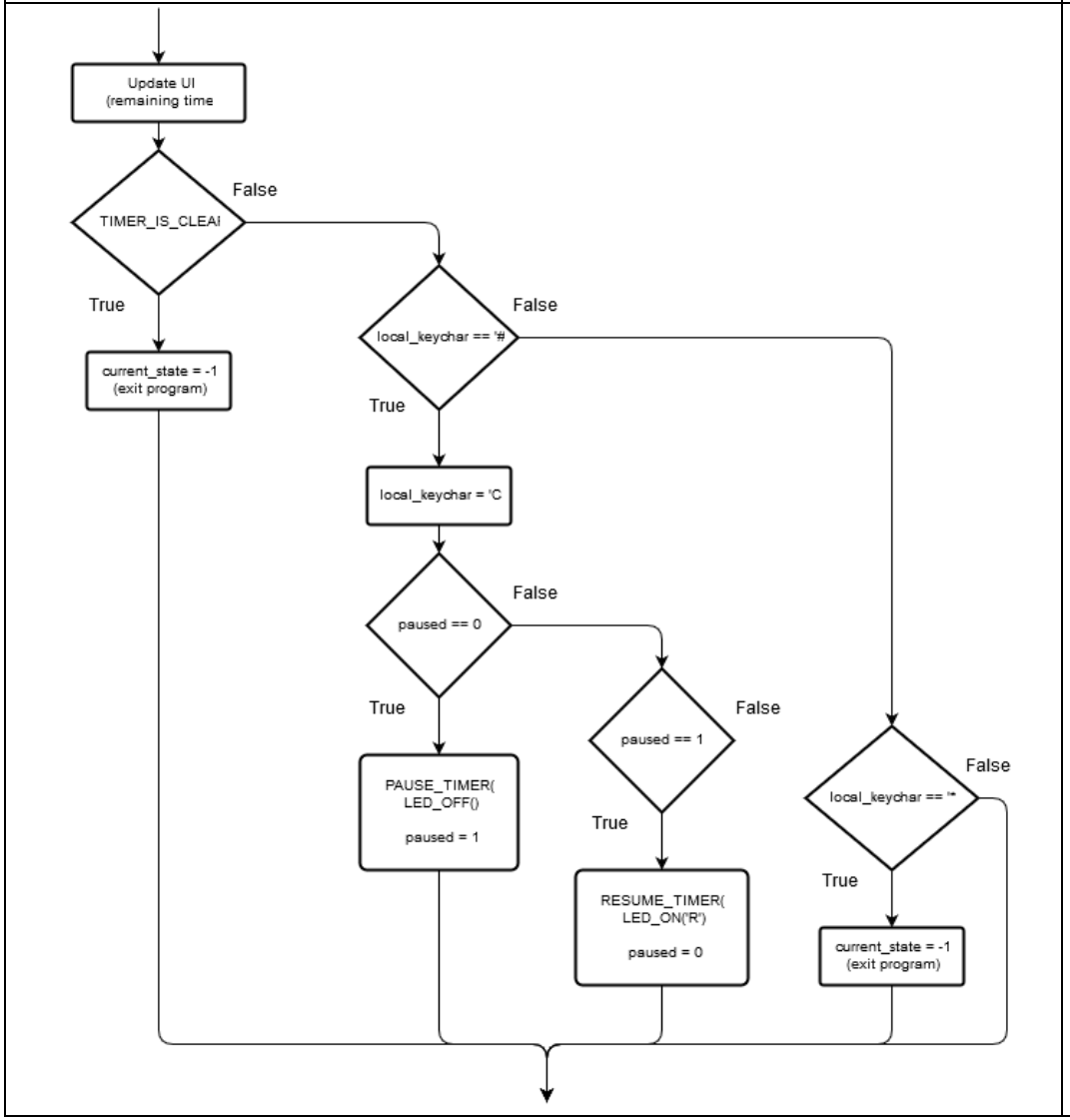
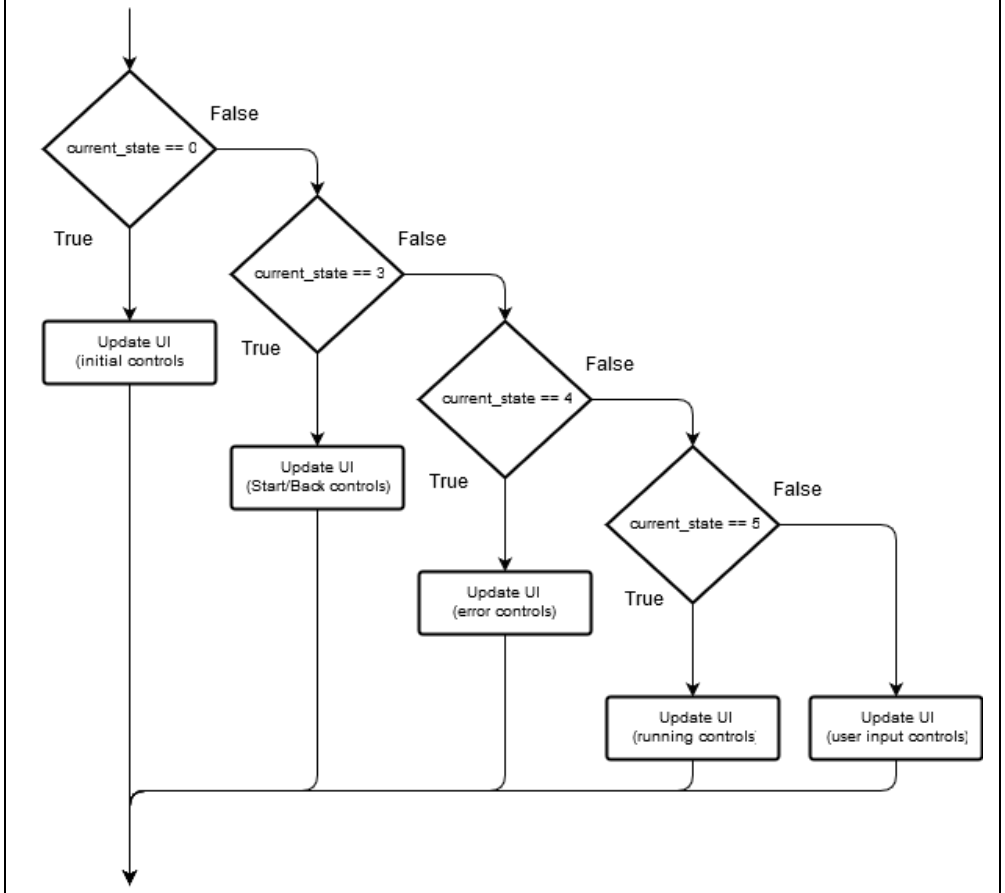


Figure 10: Flowchart for the main design of the user defined program zero

Handle keypress	User input state	Validate input
		

Running/Paused	Update controls UI
	

References

Atmel. (2007). *8-bit AVR Microcontroller with 8k bytes In-System Programmable Flash.*