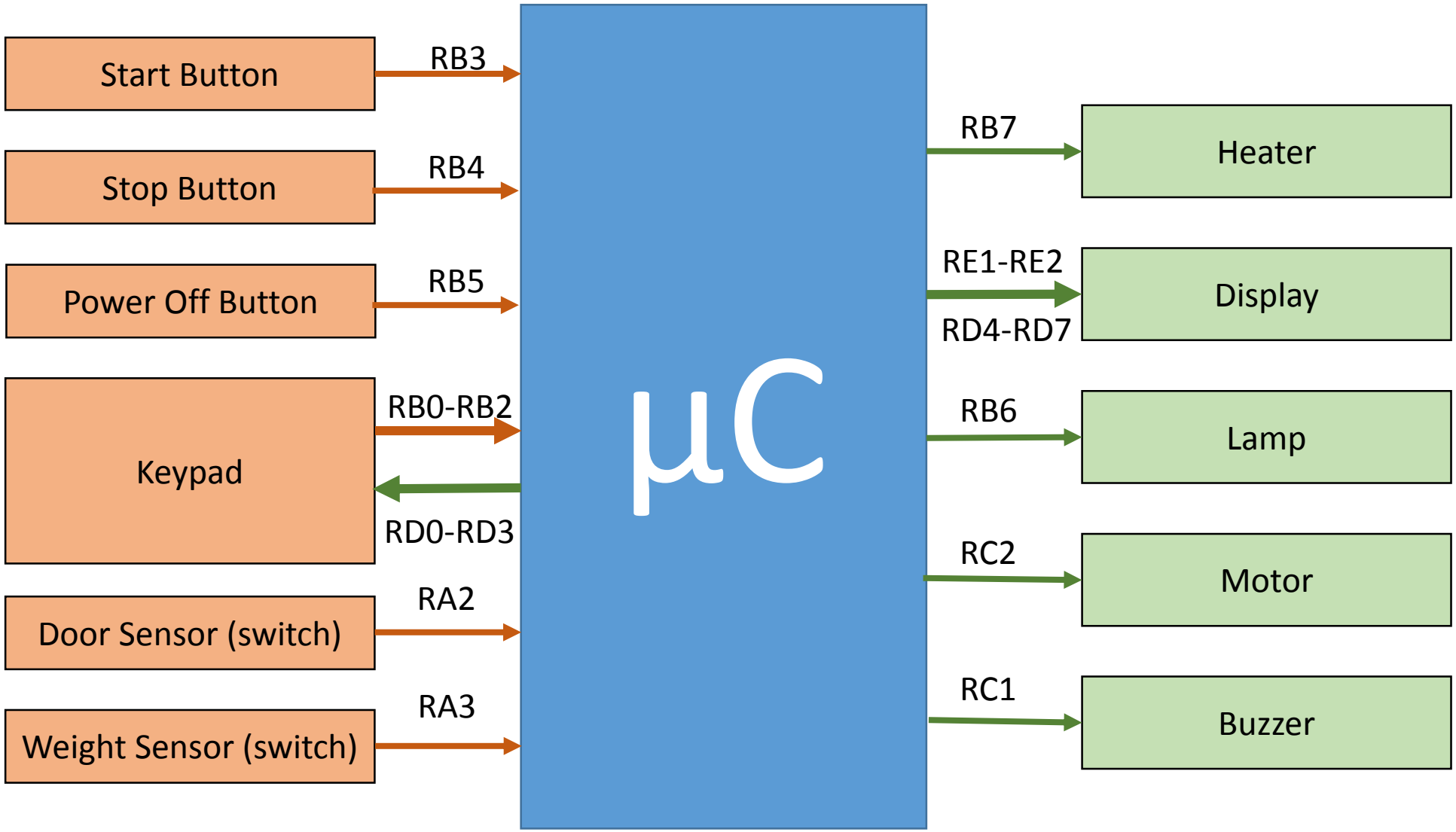
Microwave Project

By: Ahmed Wageh Abdelaziz Mohamed

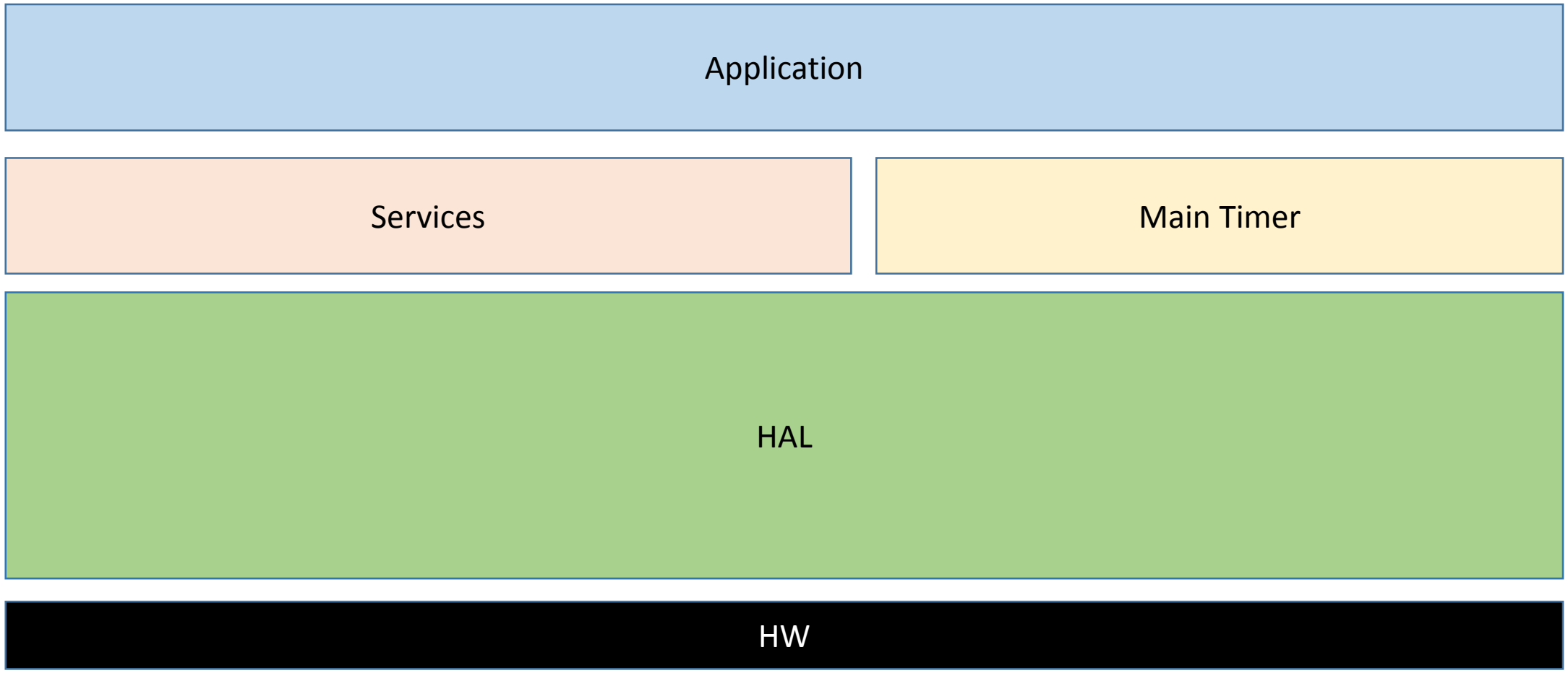
ahmedwageh463@gmail.com

+201143987151

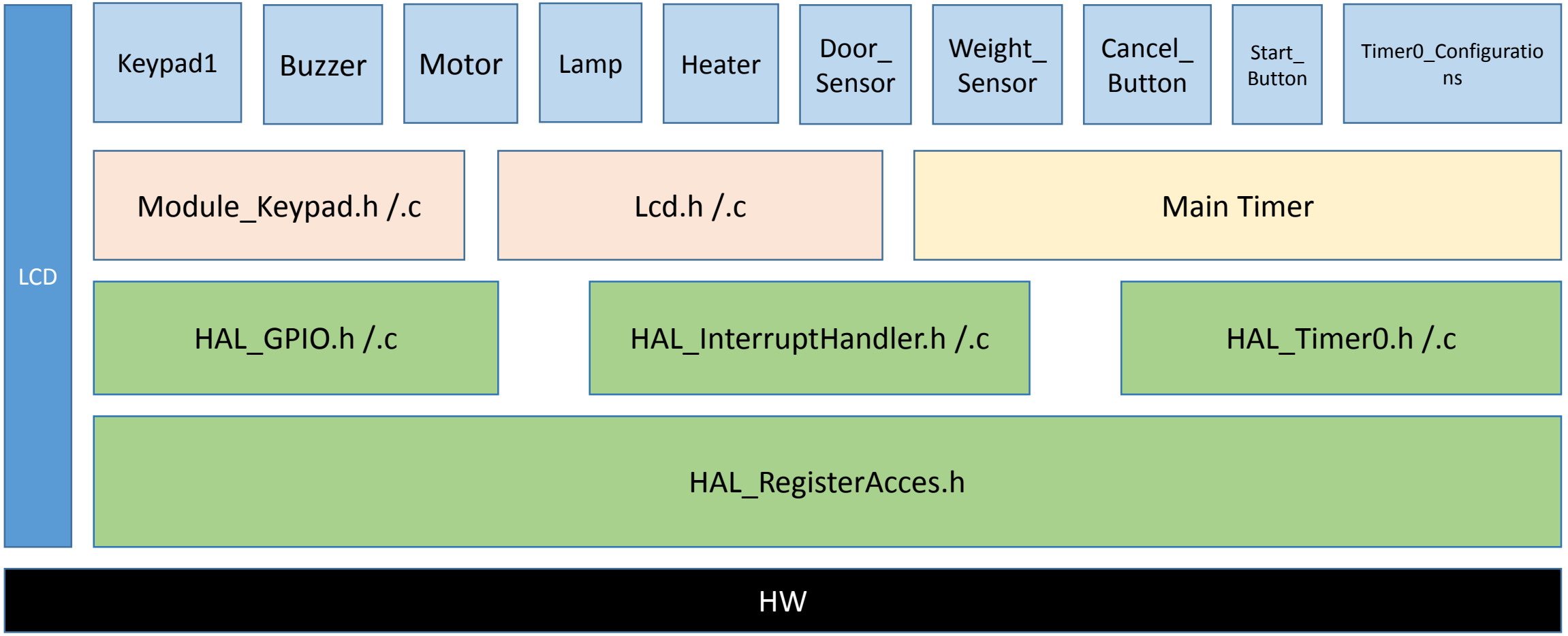
Block Diagram



Software Architecture



Detailed Software Architecture



Files Description

1. StdTypes
2. HAL
   a. HAL_RegisterAccess
   b. HAL_Timer0
   c. HAL_InterruptHandler
   d. HAL_GPIO
3. Service
   a. Module_Keypad
4. Other
   a. LCD
5. Application

1. StdTypes

- Files
  - StdTypes.h
- Dependencies on other Modules: None.
- Macros:

Macro	Description
NULL	Null definition
NULL_PTR	Null pointer definition
FALSE / False	False definition
TRUE / True	True definition

1. StdTypes

- Defined data types:

Data type	Type	Bits	Sign	Data type	Type	Bits	Sign
boolean	Typedef	1	No	float64	Typedef	64	Float
sint8	Typedef	8	Yes	vint8_t	Typedef	8	Yes - volatile
uint8	Typedef	8	No	vuint8_t	Typedef	8	No - volatile
sint16	Typedef	16	Yes	vint16_t	Typedef	16	Yes - volatile
uint16	Typedef	16	No	vuint16_t	Typedef	16	No - volatile
sint32	Typedef	32	Yes	vint32_t	Typedef	32	Yes - volatile
uint32	Typedef	32	No	vuint32_t	Typedef	32	No - volatile
sint64	Typedef	64	Yes	vint64_t	Typedef	64	Yes - volatile
uint64	Typedef	64	No	vuint64_t	Typedef	64	No - volatile
float32	Typedef	32	Float	Std_ErrorType	Enumeration	-	-

1. StdTypes

- Defined data types:

Std_ErrorType		
Error type		
STD_ERROR	0	Error happened
STD_OK	1	No Error

2.a. HAL_RegisterAccess

- Files
  - HAL_RegisterAccess.h
- Macros: following macros + macro function

Pins definition	
PIN_0	0
PIN_1	1
PIN_2	2
PIN_3	3
PIN_4	4
PIN_5	5
PIN_6	6
PIN_7	7

Bits definition	
BIT_0	0
BIT_1	1
BIT_2	2
BIT_3	3
BIT_4	4
BIT_5	5
BIT_6	6
BIT_7	7

2.a. HAL_RegisterAccess

- Dependencies on other Modules: Depend on StdTypes.h
- Defined data types: None.
- Functions: “Macro functions”

Function	Type
HAL_RegisterRead(REG_ADDRESS)	Global function
HAL_RegisterWrite(REG_ADDRESS,__VALUE)	Global function
HAL_RegisterSetBit(REG_ADDRESS,BIT_NO)	Global function
HAL_RegisterClearBit(REG_ADDRESS,BIT_NO)	Global function
HAL_RegisterAssignBit(REG_ADDRESS,BIT_NO,__VALUE)	Global function

2.a. HAL_RegisterAccess

HAL_RegisterRead(REG_ADDRESS) “Macro”	
Read the containing data of specific register.	
REG_ADDRESS	The address of the register
Replaced with	The containing data of the register

HAL_RegisterWrite(REG_ADDRESS,__VALUE) “Macro”	
Write value on specific register.	
REG_ADDRESS	The address of the register
__VALUE	The value to be written in the register
Replaced with	Don’t care

2.a. HAL_RegisterAccess

HAL_RegisterSetBit(REG_ADDRESS,BIT_NO) “Macro”	
Set specific bit of register (Pin= HIGH or 1)	
REG_ADDRESS	The address of the register
BIT_NO	The number of bit

HAL_RegisterClearBit(REG_ADDRESS,BIT_NO) “Macro”	
Clear specific bit of register (Pin= LOW or 0)	
REG_ADDRESS	The address of the register
BIT_NO	The number of bit

HAL_RegisterAssignBit(REG_ADDRESS,BIT_NO,__VALUE)“Macro”	
Write specific value on specific bit of register.	
REG_ADDRESS	The address of the register
BIT_NO	The number of bit
__VALUE	The value to be written in the register (HIGH or LOW)

2.b. HAL_Timer0

- Files
  - HAL_Timer0.h
  - HAL_Timer0.c
- Macros:Timer0L and Timer0H addresses.
- Dependencies on other Modules: Depend on:
  - StdTypes.h
  - HAL_RegisterAccess.h

2.b. HAL_Timer0

- Defined data types:

Data type	Type
HAL_Timer0_DataSizeType	Enumeration
HAL_Timer0_ModeType	Enumeration
HAL_Timer0_PrescalerType	Enumeration
HAL_Timer0_ConfigType	Structure

- Functions:

Functions	Type
Std_ErrorType HAL_Timer0_init(HAL_Timer0_ConfigType * Timer0_Config)	Initialization
void HAL_Timer0_start(void)	Global function
void HAL_Timer0_stop(void)	Global function
Std_ErrorType HAL_Timer0_updateConfig(HAL_Timer0_ConfigType * Timer0_Config)	Global function
static Std_ErrorType Timer0_ErrorCheck(HAL_Timer0_ConfigType * Timer0_Config)	Private function

2.b. HAL_Timer0 defined data types

HAL_Timer0_DataSizeType		
This is to indicate the size of Timer0 count register		
TIMER0_16_BITS	0x00	16 bits timer
TIMER0_8_Bits	0x01	8 bits timer

HAL_Timer0_ModeType		
This is to indicate the mode of Timer0		
TIMER0_TIMER	0x00	Timer0 increments with internal clock
TIMER0_COUNTER_RISING_EDGE	0x01	Timer0 increments with external clock connected to T0CKI pin with rising edge
TIMER0_COUNTER_FALLING_EDGE	0x02	Timer0 with external connected to T0CKI pin with falling edge

HAL_Timer0_PrescalerType		
This is to indicate the prescaler.		
TIMER0_PRESCALER_OFF	0x00	No prescaler
TIMER0_PRESCALER_2	0x01	Prescaler 2
TIMER0_PRESCALER_4	0x02	Prescaler 4
TIMER0_PRESCALER_8	0x03	Prescaler 8
TIMER0_PRESCALER_16	0x04	Prescaler 16
TIMER0_PRESCALER_32	0x05	Prescaler 32
TIMER0_PRESCALER_64	0x06	Prescaler 64
TIMER0_PRESCALER_128	0x06	Prescaler 132
TIMER0_PRESCALER_256	0x08	Prescaler 256

2.b. HAL_Timer0 defined data types

HAL_Timer0_ConfigType		
This is to define all needed configurations for Timer0.		
Timer0_Mode	HAL_Timer0_ModeType	The mode of the timer
Timer0_DataSize	HAL_Timer0_DataSizeType	16 bits or 8 bits data length
Timer0_Prescaler	HAL_Timer0_PrescalerType	Prescaler value
Timer0_Data	uint16	Data to be written in timer register

2.b. HAL_Timer0 functions

Std_ErrorType HAL_Timer0_init(HAL_Timer0_ConfigType * Timer0_Config)		
To initialize Timer0. **IT doesn't contain timer start**.		
Timer0_Config	HAL_Timer0_ConfigType *	The address of configurations struct
Return	STD_OK: When all configurations filled with correct data	E_NOT_OK: If there is data filled with wrong data

void HAL_Timer0_start(void)	
Start timer0	
void	
void	

Std_ErrorType HAL_Timer0_updateConfig(HAL_Timer0_ConfigType * Timer0_Config)		
It's initialize + start		
Timer0_Config	HAL_Timer0_ConfigType *	The address of configurations struct
Return	STD_OK: When all configurations filled with correct data	E_NOT_OK: If there is data filled with wrong data

void HAL_Timer0_stop(void)	
Stop timer0	
void	
void	

2.c. HAL_InterruptHandler

- Files
  - HAL_InterruptHandler.h
  - HAL_InterruptHandler.c
- Macros:

Macro	Description
DRIVER_HANDLE_ISR	To define if the driver handles the ISR or the user will handle it
NUMBER_OF_ACTIVE_INTERRUPTS	Number of active interrupts
INTERRUPT_<INTERRUPT_SOURCE>	To handle the interrupt source you want
INT0_PORT	Interrupt 0 pin port address
INT1_PORT	Interrupt 1 pin port address
INT2_PORT	Interrupt 2 pin port address
INT0_PIN	Interrupt 0 pin number
INT1_PIN	Interrupt 1 pin number
INT2_PIN	Interrupt 2 pin number

2.c. HAL_InterruptHandler

- Dependencies on other Modules: Depend on:
  - StdTypes.h
  - HAL_RegisterAccess.h
- Defined data types:

Data type	Type
InterruptSourceType	Enumeration
InterruptOcurrancyType	Enumeration

- Functions

Functions	Type
void InterruptHandler_EnableInterrupt(InterruptSourceType InterruptSource, InterruptOcurrancyType * flagPointer)	Global function
void InterruptHandler_DisableInterrupt(InterruptSourceType InterruptSource)	Global function
void InterruptHandler_DisableGlobalInterrupt(void)	Global function
void InterruptHandler_EnbleGlobalInterrupt(void)	Global function

2.c. HAL_InterruptHandler defined data types

InterruptOcurrancyType		
This is to indicate if the interrupt happened or not		
EXT_INT_NOT_HAPPENED	0x00	No Interrupt
EXT_INT_HAPPENED	0x01	Interrupt happened

InterruptSourceType			
This is to indicate the index of each interrupt source			
INT_EXT0	0x00	INT_TX	0x0A
INT_EXT1	0x01	INT_RC	0x0B
INT_EXT2	0x02	INT_SSP	0x0C
INT_RB	0x03	INT_AD	0x0D
INT_TMR0	0x04	INT_PSP	0x0E
INT_TMR1	0x05	INT_HLVD	0x0F
INT_TMR2	0x06	INT_BCL	0x10
INT_TMR3	0x06	INT_EE	0x11
INT_CCP1	0x08	INT_CM	0x12
INT_CCP2	0x09	INT_OSCF	0x13

2.c. HAL_InterruptHandler functions

void InterruptHandler_EnableInterrupt(InterruptSourceType InterruptSource, InterruptOcurrancyType * flagPointer)		
Enables specific interrupt source passed its identifier		
InterruptSource	InterruptSourceType	to know which interrupt source you want to enable
flagPointer <Optional if you defined DRIVER_HANDLE_ISR>	HAL_Timer0_DataSizeType	Pointer to flag to set if the interrupt happens
Return	None	

void InterruptHandler_DisableInterrupt(InterruptSourceType InterruptSource)		
Disables specific interrupt source passed its identifier		
InterruptSource	InterruptSourceType	to know which interrupt source you want to disable
Return	None	

2.c. HAL_InterruptHandler function

void InterruptHandler_DisableGlobalInterrupt(void)	
To disable global interrupt	
None	
Return	None

void InterruptHandler_EnbleGlobalInterrupt(void)	
To enable global interrupt	
None	
Return	None

2.d. HAL_GPIO

- Files
  - HAL_GPIO.h
  - HAL_GPIO.c
- Macros:

Macro	Description
PORT<X>_BASE_ADDRESS	The base address of ports, X is port letter (from A to E)
GPIO_OUTPUT_INITIAL_STATE	The initial state of output devices
PORT_DIRECTION_OFFSET	The difference of the addressees between PORTx and TRISx
PORT_LATCH_OFFSET	The difference of the addressees between PORTx and LATx
ADCON1_ADDRESS	ADCON1 address needed to disable analog function for portB

2.d. HAL_GPIO

- Dependencies on other Modules: Depend on:
  - StdTypes.h
  - HAL_RegisterAccess.h
- Defined data types:

Data type	Type
HAL_GPIO_DeviceDirectionType	Enumeration
HAL_GPIO_StatusType	Enumeration
HAL_GPIO_DeviceType	Structure

2.d. HAL_GPIO

- Functions:

Functions	Type
Std_ErrorType GPIO_DeviceInit(HAL_GPIO_DeviceType * GPIO_Device)	Global function
Std_ErrorType GPIO_DeviceSet(HAL_GPIO_DeviceType * GPIO_Device)	Global function
Std_ErrorType GPIO_DeviceClear(HAL_GPIO_DeviceType * GPIO_Device)	Global function
Std_ErrorType GPIO_DeviceToggle(HAL_GPIO_DeviceType * GPIO_Device)	Global function
Std_ErrorType GPIO_DeviceAssignStatus(HAL_GPIO_DeviceType * GPIO_Device, HAL_GPIO_StatusType GPIO_DeviceStatus)	Global function
Std_ErrorType GPIO_DeviceGetRead(HAL_GPIO_DeviceType * GPIO_Device, HAL_GPIO_StatusType * ReturnStatus)	Global function
static Std_ErrorType GPIO_CheckError(HAL_GPIO_DeviceType * GPIO_Device)	Private function

2.d. HAL_GPIO defined data types

HAL_GPIO_DeviceDirectionType		
This is to indicate the direction of the pin		
OUTPUT	0x00	Output pin
INPUT	0x01	Input pin

HAL_GPIO_StatusType		
This is to indicate the status of the pin		
LOW	0x00	Pin is cleared (0)
HIGH	0x01	Pin is set (1)

HAL_GPIO_DeviceType		
This is to define all needed configurations for GPIO		
devicePortBaseAddress	uint16	The base address of the port which the GPIO device connected to
devicePin	uint8	The pin number which the GPIO device connected to
deviceDirection	HAL_GPIO_DeviceDirectionType	The direction of the device

2.d. HAL_GPIO functions

Std_ErrorType GPIO_DeviceInit(HAL_GPIO_DeviceType * GPIO_Device)		
The passed GPIO device will be initialized with filled configurations in passed pointer to struct		
GPIO_Device	HAL_GPIO_DeviceType *	Address of configurations struct
Return	STD_OK: When all configurations filled with correct data	E_NOT_OK: If there is data filled with wrong data

Std_ErrorType GPIO_DeviceSet(HAL_GPIO_DeviceType * GPIO_Device)		
The passed GPIO device will be set (pin=1) with filled configurations in passed pointer to struct		
GPIO_Device	HAL_GPIO_DeviceType *	Address of configurations struct
Return	STD_OK: When all configurations filled with correct data	E_NOT_OK: If there is data filled with wrong data

2.d. HAL_GPIO functions

Std_ErrorType GPIO_DeviceClear(HAL_GPIO_DeviceType * GPIO_Device)		
The passed GPIO device will be cleared (pin=0) with filled configurations in passed pointer to struct		
GPIO_Device	HAL_GPIO_DeviceType *	Address of configurations struct
Return	STD_OK: When all configurations filled with correct data	E_NOT_OK: If there is data filled with wrong data

Std_ErrorType GPIO_DeviceToggle(HAL_GPIO_DeviceType * GPIO_Device)		
The passed GPIO device will be toggled (invert the status) with filled configurations in passed pointer to struct		
GPIO_Device	HAL_GPIO_DeviceType *	Address of configurations struct
Return	STD_OK: When all configurations filled with correct data	E_NOT_OK: If there is data filled with wrong data

2.d. HAL_GPIO functions

Std_ErrorType GPIO_DeviceAssignStatus(HAL_GPIO_DeviceType * GPIO_Device, HAL_GPIO_StatusType GPIO_DeviceStatus)		
The passed GPIO device will be assign with passed GPIO_DeviceStatus		
GPIO_Device	HAL_GPIO_DeviceType *	Address of configurations struct
GPIO_DeviceStatus	HAL_GPIO_StatusType	The status to be assigned
Return	STD_OK: When all configurations filled with correct data	E_NOT_OK: If there is data filled with wrong data

Std_ErrorType GPIO_DeviceGetRead(HAL_GPIO_DeviceType * GPIO_Device, HAL_GPIO_StatusType * ReturnStatus)		
The passed GPIO device pin status will be read and assigned in the passed buffer		
GPIO_Device	HAL_GPIO_DeviceType *	Address of configurations struct
ReturnStatus	HAL_GPIO_StatusType *	Pointer to buffer to assign data on it
Return	STD_OK: When all configurations filled with correct data	E_NOT_OK: If there is data filled with wrong data

3.a. Module_Keypad

- Files
  - Keypad_Config.h “contains keypad configurations”
  - Module_Keypad.h
  - Module_Keypad .c
- Macros:

Macro	Description
KEYPAD_MAX_COL_NUMBER	The maximum number of columns for all connected keypads
KEYPAD_MAX_ROW_NUMBER	The maximum number of rows for all connected keypads
KEYPAD_NOT_PRESSED	The value to be returned when no key is pressed

3.a. Module_Keypad

- Dependencies on other Modules: Depend on:
  - HAL_GPIO.h and all its dependencies
- Defined data types:

Data type	Type
keypad_returnDataType	Typedef
Keypad_ConfigType	Structure

- Functions:

Functions	Type
Std_ErrorType Keypad_init(Keypad_ConfigType * Keypad_Configuration)	Initialize function
Std_ErrorType Keypad_getReading(Keypad_ConfigType * Keypad_Configuration, keypad_returnDataType * keypad_returnData)	Global function
static Std_ErrorType Keypad_checkForError(Keypad_ConfigType * Keypad_Configuration)	Private function

3.a. Module_Keypad defined data types

keypad_returnDataType	
Define the type of return from keypad	
Data type	uint8

Keypad_ConfigType		
This is to define all needed configurations for Keypad		
rowsNumber	uint8	The number of ROWS
colsNumber	uint8	The number of COLS
rowConfiguration[KEYPAD_MAX_ROW_NUMBER]	HAL_GPIO_DeviceType	The configurations of row pins
colConfiguration[KEYPAD_MAX_COL_NUMBER]	HAL_GPIO_DeviceType	The configurations of col pins
returnDataArray[KEYPAD_MAX_ROW_NUMBER] [KEYPAD_MAX_COL_NUMBER]	keypad_returnDataType	2-D array to return data from keypad

3.a. Module_Keypad functions

Std_ErrorType Keypad_init(Keypad_ConfigType * Keypad_Configuration)		
The passed keypad will be initialized with filled configurations in passed pointer to struct		
Keypad_Configuration	Keypad_ConfigType *	Address of configurations struct
Return	STD_OK: When all configurations filled with correct data	E_NOT_OK: If there is data filled with wrong data

Std_ErrorType Keypad_getReading(Keypad_ConfigType * Keypad_Configuration, keypad_returnDataType * keypad_returnData)		
The passed Keypad will be read and assigned in the passed buffer		
Keypad_Configuration	Keypad_ConfigType *	Address of configurations struct
keypad_returnData	keypad_returnDataType *	Pointer to buffer to assign the data
Return	STD_OK: When all configurations filled with correct data	E_NOT_OK: If there is data filled with wrong data

4.a. LCD

- Files
  - LCD_Config.h “contains LCD configurations”
- Note: for LCD I used LCD built-in library in MiKroC pro for PIC IDE, I just configured it in LCD_Config.h .

5. Application

- Files
  - App.h “contains the declaration and definition of all used modules from all layers”
  - App_Functions.h ”update time function and its independencies”
  - App_Functions.c
  - MicroWave.c “The application itself”
- Macros:

Macro	Description
Sleep()	Sleep the controller

- Dependencies on other Modules: on all previous modules.

5. Application

- Defined data types:

Data type	Type	File
APP_stateType	Enumeration	App.h
Time_DataType	Structure	App_Functions.h

- Functions:

Functions	Type
void APP_Init(void)	Initialize function
void APP_Timeupdate (Time_DataType * Time_Data)	Global function
void APP_Off_Mode(void)	Global function
void APP_WakeUp_Mode(void)	Global function
void APP_Edit_Mode(void)	Global function
void APP_Run_Mode(void)	Global function
void APP_Notification_Mode(void)	Global function

5. Application defined data types

APP_stateType		
This is to indicate if the state of the program		
APP_OFF_STATE	0x00	LCD cleared and Microwave off
APP_WAKE_UP_STATE	0x01	Transition state after wake up just to initialize all disabled peripherals when sleeping
APP_EDIT_STATE	0x02	LCD on and user edit time
APP_RUNNING_STATE	0x03	Microwave is on and LCD on
APP_NOTIFICATION_STATE	0x04	Microwave done and buzzer on

Time_DataType	
This is to define Time data	
seconds	sint8
minutes	sint8
hours	sint8

5. Application functions

void APP_Init(void)	
This function to initialize all used modules in the application	
Parameters	None
Return	None

void APP_Off_Mode(void)	
This function contains all needed processes for Wakeup Mode	
Parameters	None
Return	None

void APP_Timeupdate (Time_DataType * Time_Data)		
This function to update Time on Display (Here we use LCD)		
Time_Data	Time_DataType *	Address of configurations struct
Return	None	

void APP_Init(void)	
This function to initialize all used modules in the application	
Parameters	None
Return	None

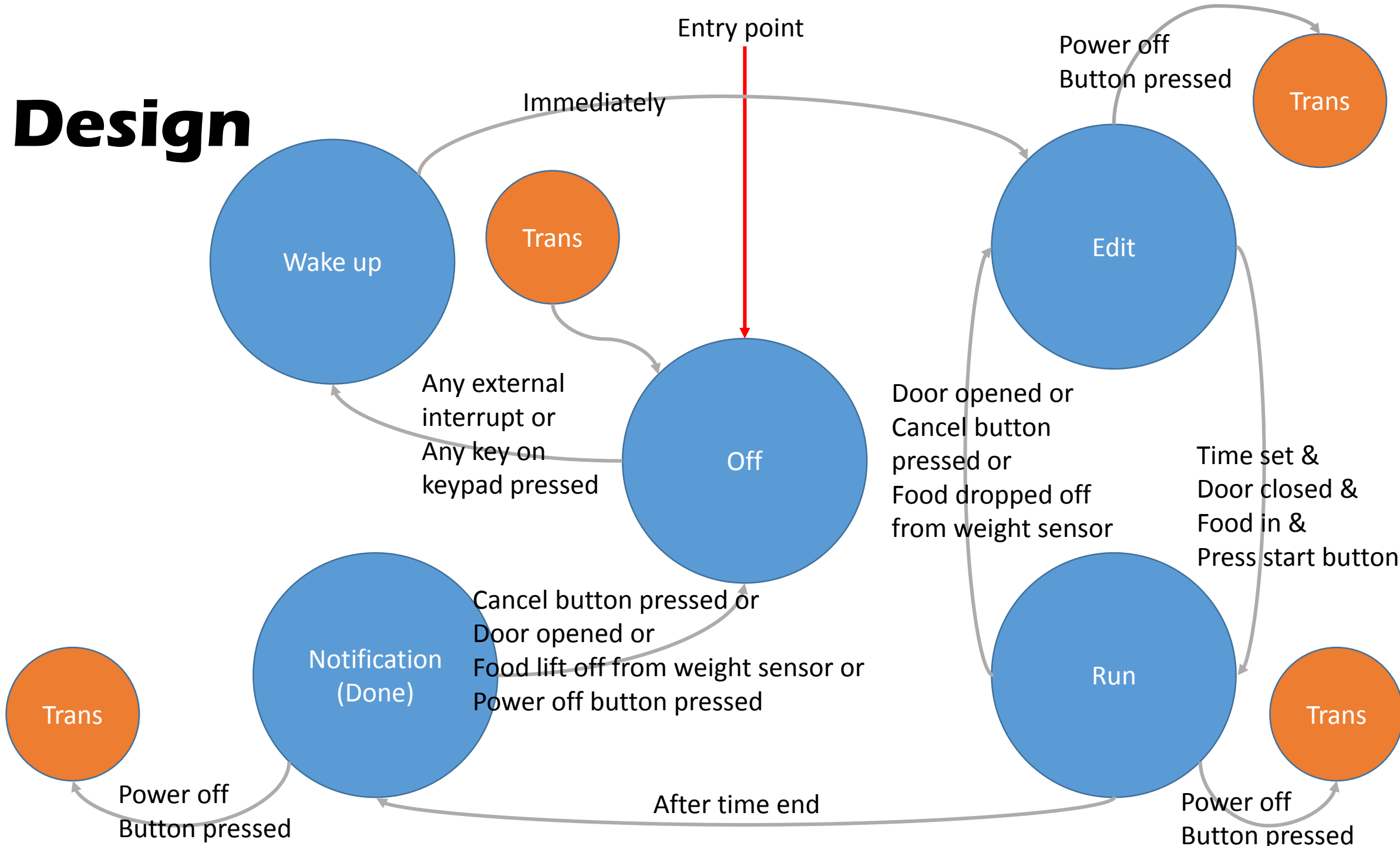
5. Application functions

void APP_WakeUp_Mode(void)	
This function contains all needed processes for Wakeup Mode	
Parameters	None
Return	None

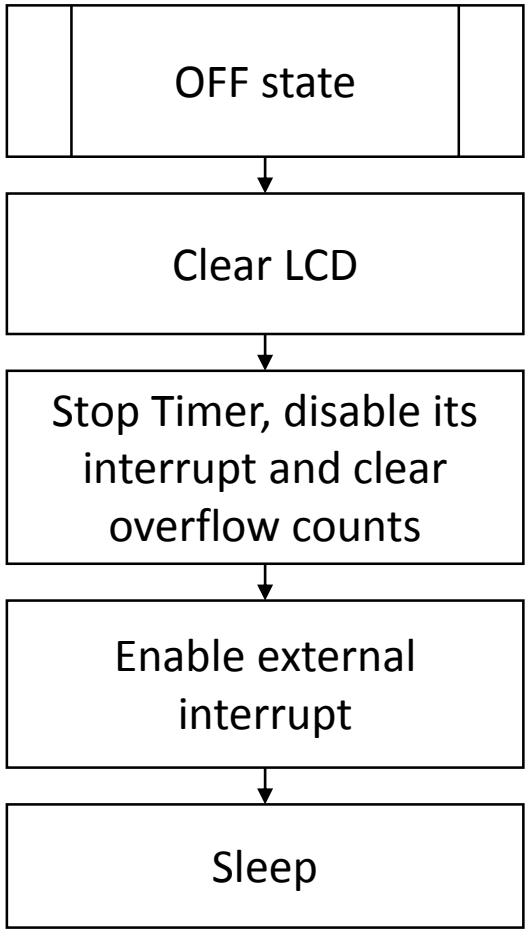
void APP_Notification_Mode(void)	
This function contains all needed processes for Notification Mode	
Parameters	None
Return	None

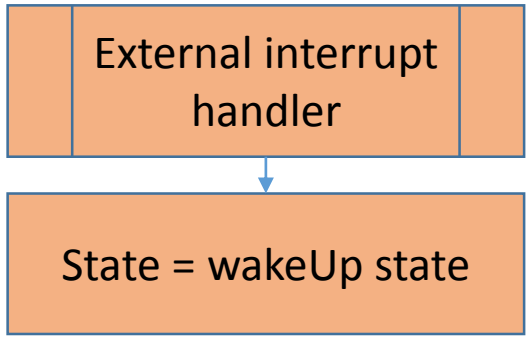
void APP_Edit_Mode(void)	
This function contains all needed processes for Run Mode	
Parameters	None
Return	None

Design

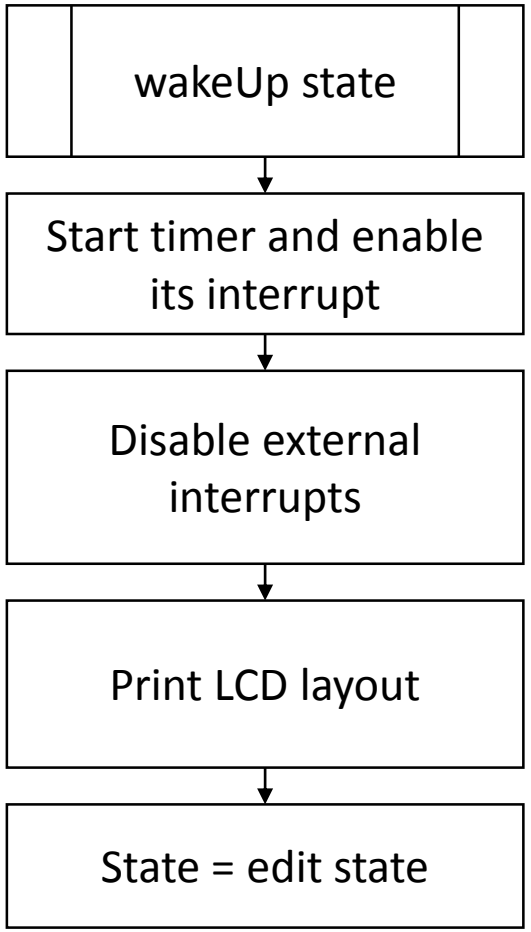

Entry point
Immediately
Power off
Button pressed
Trans
Wake up
Trans
Edit
Any external
interrupt or
Any key on
keypad pressed
Off
Door opened or
Cancel button
pressed or
Food dropped off
from weight sensor
Time set &
Door closed &
Food in &
Press start button
Cancel button pressed or
Door opened or
Food lift off from weight sensor or
Power off button pressed
Notification
(Done)
Run
Trans
Trans
Power off
Button pressed
After time end
Power off
Button pressed

States flowchart (OFF state) Run Once

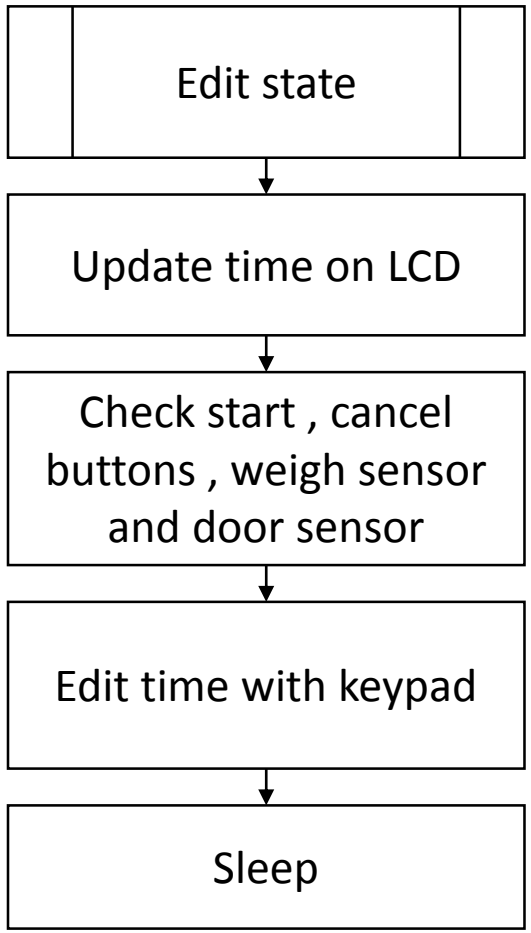




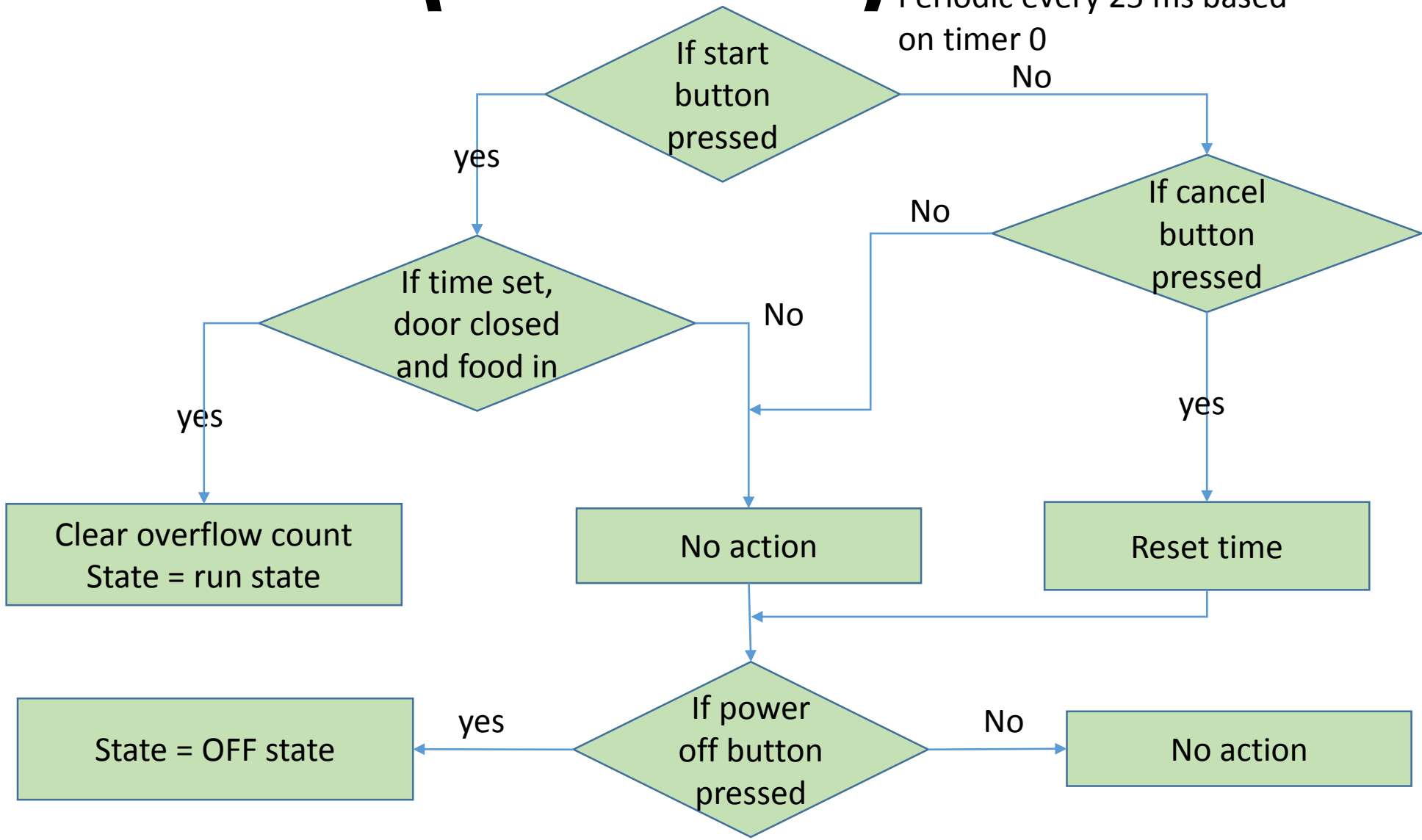
States flowchart (wakeUp state) Run Once



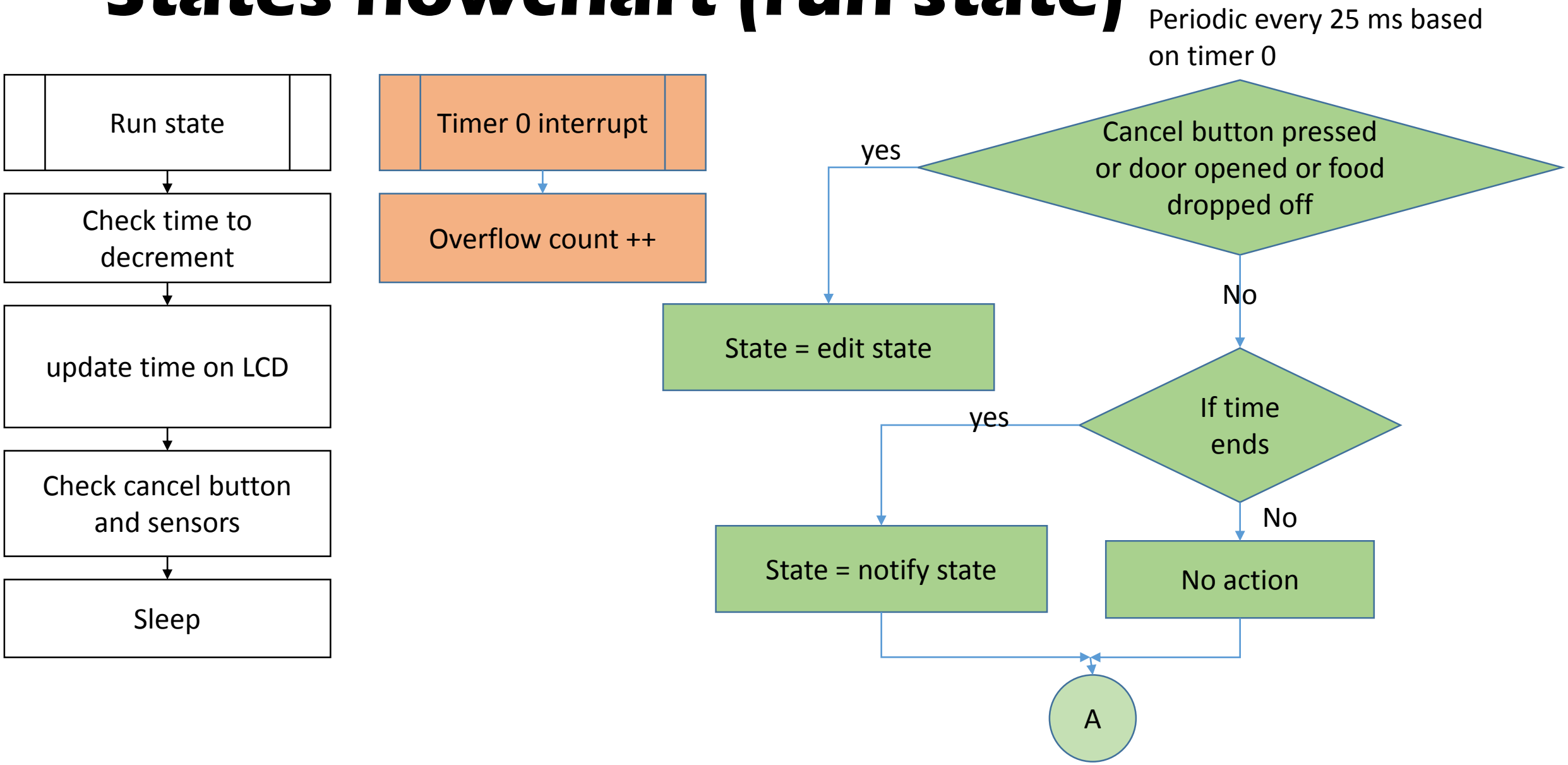
States flowchart (edit state)



Periodic every 25 ms based on timer 0

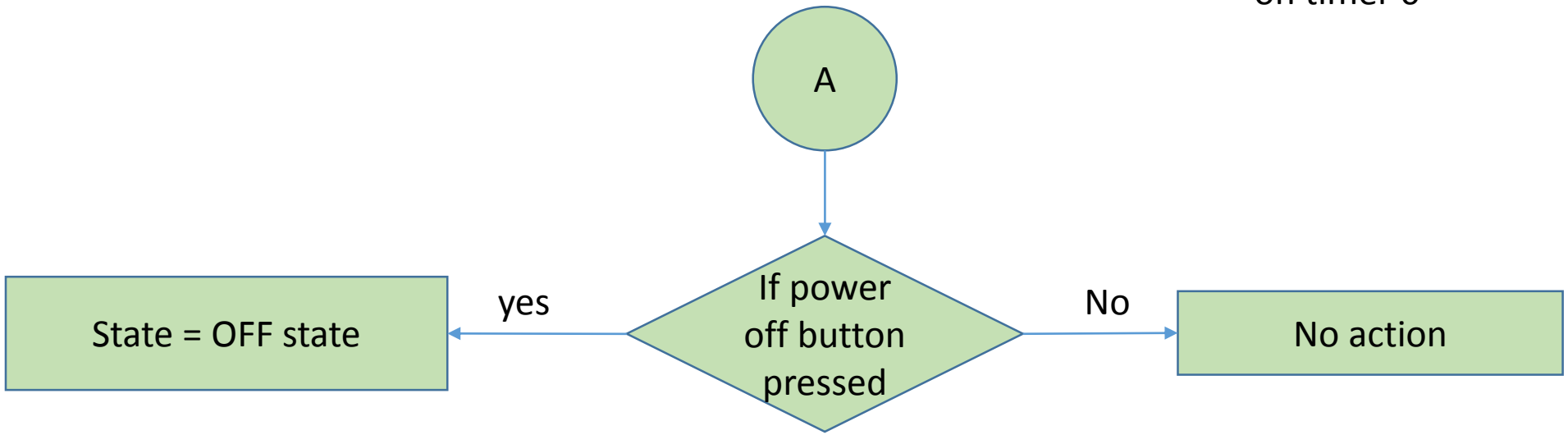


States flowchart (run state)

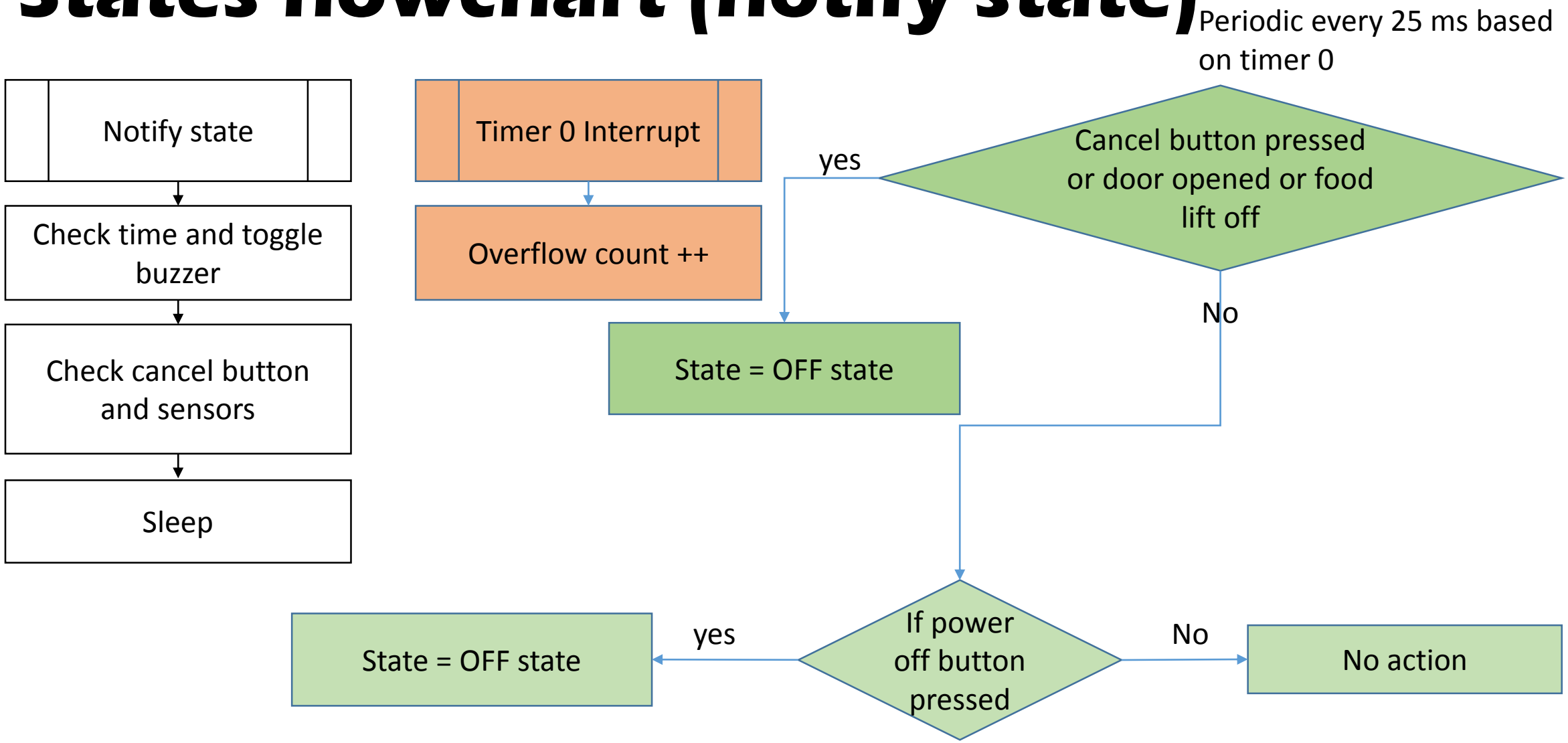


States flowchart (run state) .cont

Periodic every 25 ms based on timer 0



States flowchart (notify state)



Assumptions

- Frequency is 8 MHz
- Microcontroller is PIC18f4620
- LCD size is 4*16
- Power off button power down the microwave in any status at any time.
- Weight sensor and door sensor are digital sensors.
- Program has 5 states:
  - APP_OFF_STATE
  - APP_WAKE_UP_STATE
  - APP_EDIT_STATE
  - APP_RUNNING_STATE
  - APP_NOTIFICATION_STATE
- Initial state is APP_OFF_STATE
- APP_OFF_STATE executed once then sleep until external interrupt happened.

Assumptions

- APP_WAKE_UP_STATE executed once to reinitialize all needed after device is on then go to APP_EDIT_STATE
- APP_EDIT_STATE is periodic based on timer 0 (every 25 minutes)
- At first you edit the second digit of hours you can switch digits with (‘*’ and ‘#’) from keypad
- ‘*’ goes left and if you are at the most left(second digit in hours) and pressed ‘*’ you go to the most right and vice versa for ‘#’.
- Hours has no limitations
- Seconds and minutes limited with 59 maximum so maximum time is 99:59:59

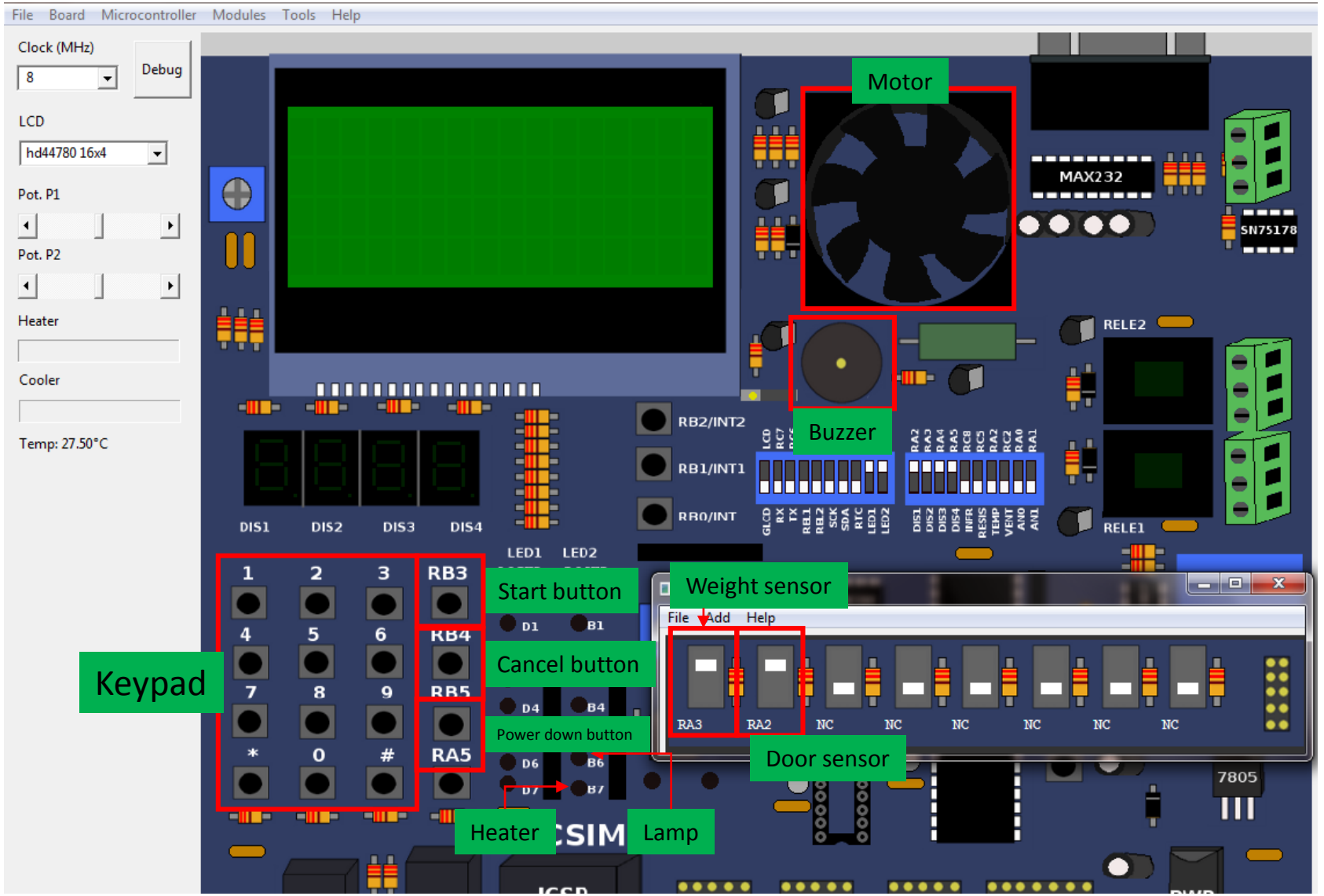
Assumptions

- If you try to enter number >= 6 in second digit of minutes or seconds your entry won’t be accepted
- If you press start button and time is set and door is closed and food is in you enter APP_RUNNING_STATE
- APP_RUNNING_STATE is periodic every (25 ms) so it decrements seconds every 40 overflow counts (= 1 second)
- It checks cancel button and sensors and if any of them changes we enter APP_EDIT_STATE mode again
- After the set time ends we enter APP_NOTIFICATION_STATE

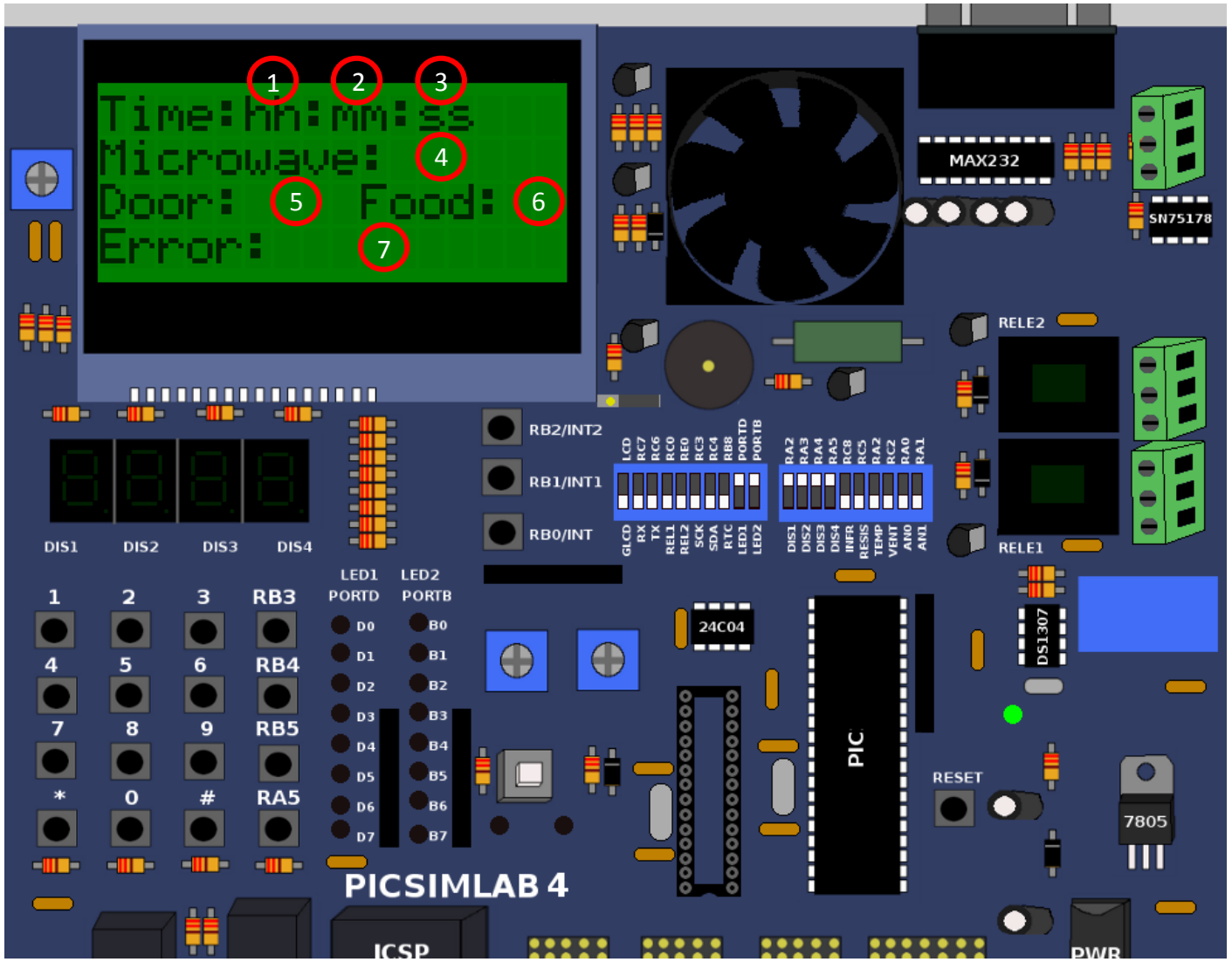
Assumptions

- APP_NOTIFICATION_STATE is periodic (every 25 ms) it toggles the buzzer every .5 second (20 overflow count) until cancel button or any other sensor change its status.
- After that it enters APP_OFF_STATE and so on.

Simulation

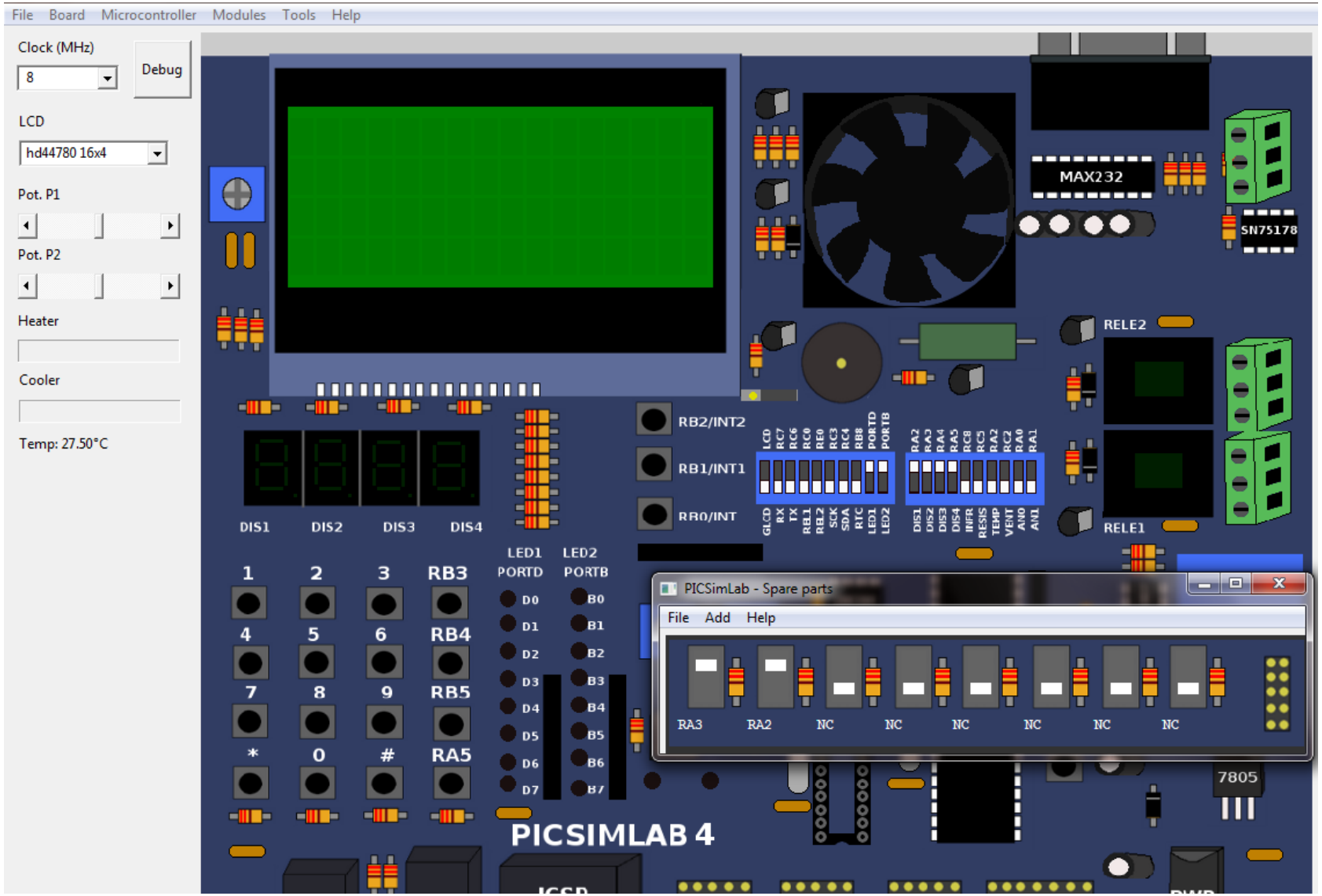

File Board Microcontroller Modules Tools Help
Clock (MHz)
8
Debug
LCD
hd44780 16x4
Pot. P1
Pot. P2
Heater
Cooler
Temp: 27.50°C
Motor
MAX232
SN75178
RELE2
RELE1
Buzzer
RB2/INT2
RB1/INT1
RB0/INT
DIS1 DIS2 DIS3 DIS4
LED1 LED2
1 2 3 RB3
4 5 6 RB4
7 8 9 RB5
* 0 # RA5
Keypad
Start button
Cancel button
Power down button
Heater
Lamp
Weight sensor
Door sensor
File Add Help
RA3 RA2 NC NC NC NC NC NC
7805
CSIM

Simulation

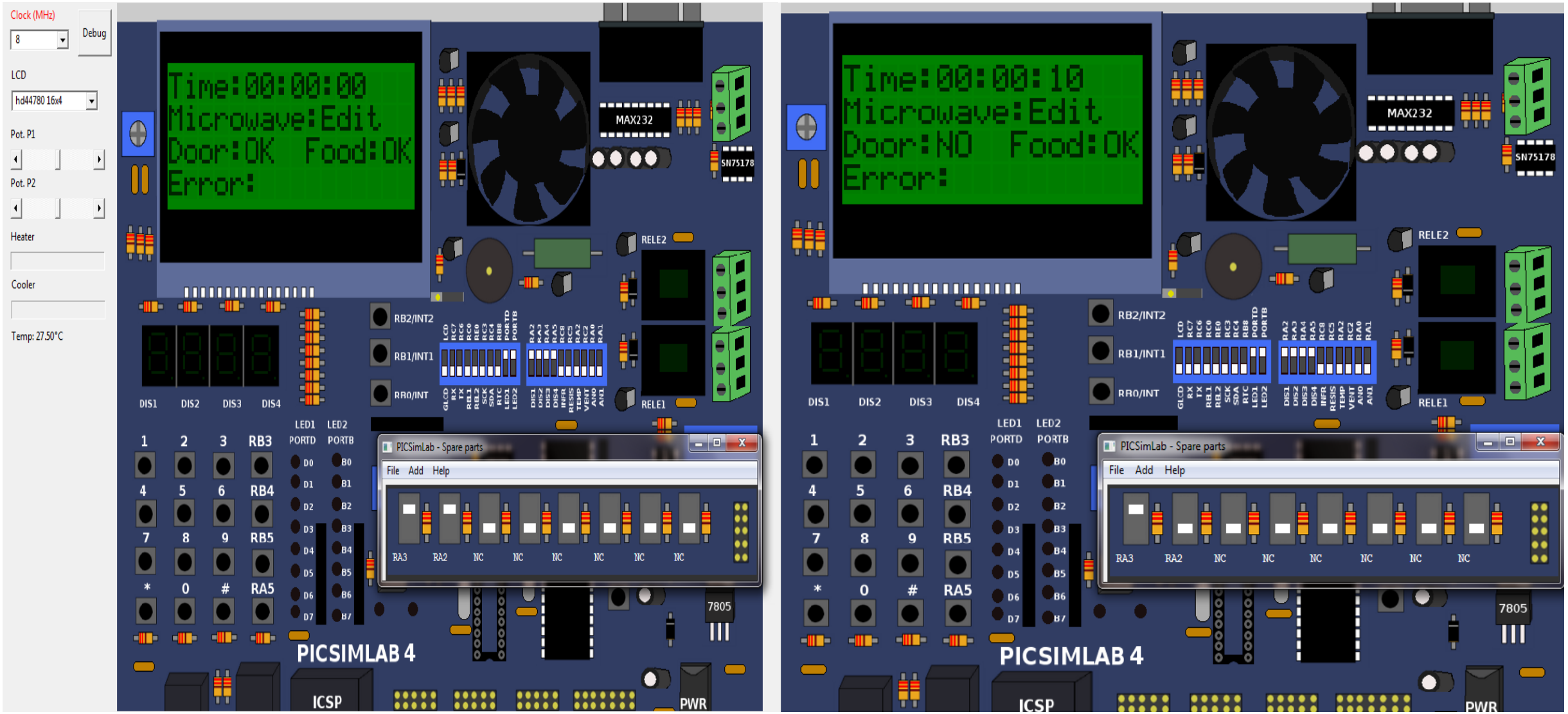


1. Display remaining / entered hours
2. Display remaining / entered minutes
3. Display remaining / entered seconds
4. Program status:
   - Edit
   - Run
   - Done
5. Door status:
   - OK => door is closed
   - NO => door is opened
6. Food status:
   - OK => food is in
   - NO => microwave is empty
7. Error message:
   - TimeNotSet
   - PutFoodIn
   - Close Door

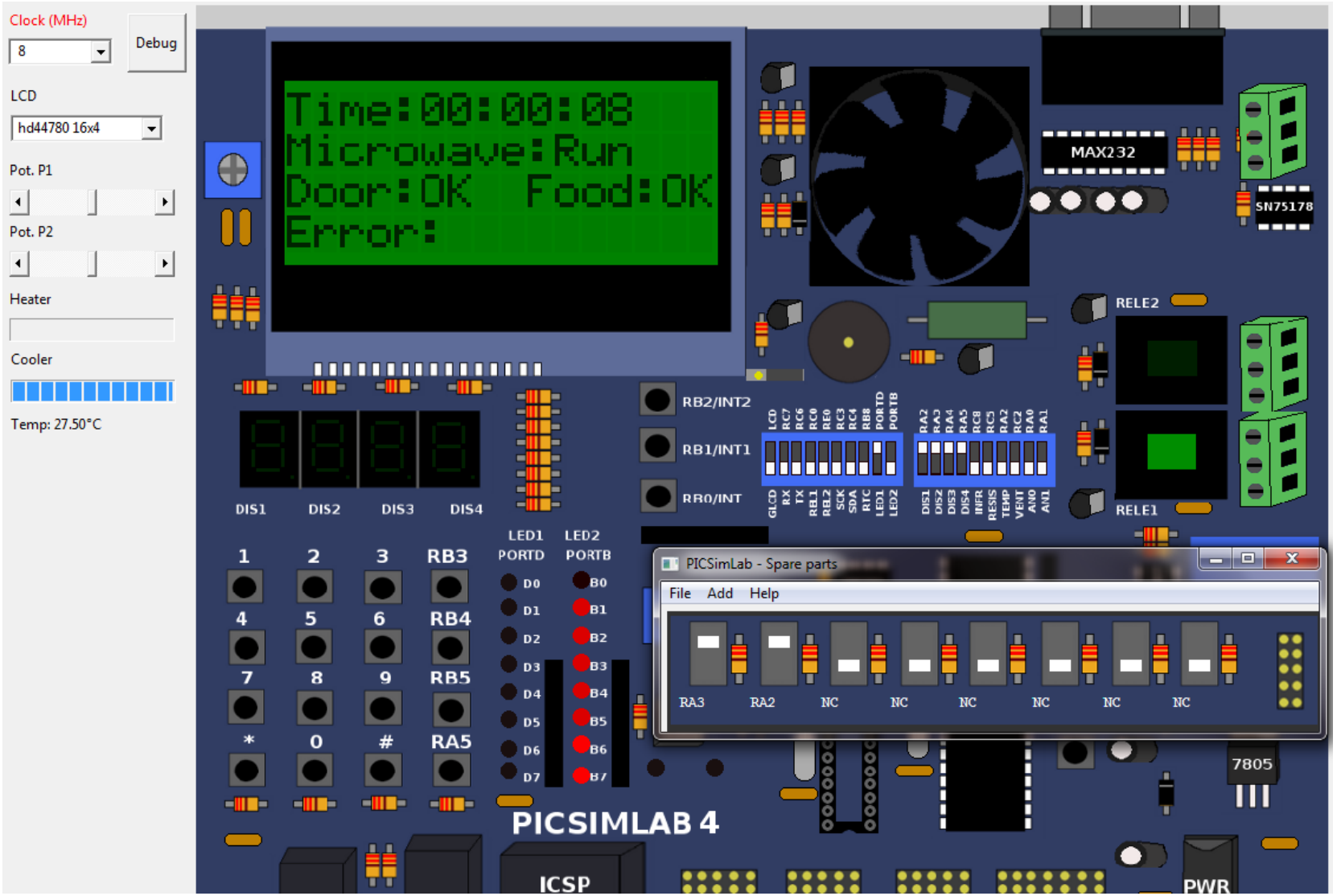
Simulation OFF Mode



Simulation Edit Mode



Simulation Run Mode



Simulation Notify Mode