Documentation presentation for SwiftAct(Microwave)

Ahmed Abdallah Eldakhly

Assumption:

The microwave has three states:

1- The Idle state: in this state, the user can open the door, put the food in case the door is opened, set the heating time, clear the screen to reinsert the time again and set the temperature of the heating.
The LCD displays the inserted time, temperature, door state and weight state inside the microwave.
The user can start heating process if the time more than zero, the door is closed and something exists inside the microwave.
Note: the Led, the heating process, and the buzzer are off in this state.

2- The heating state: in this state, the heating process will start with the specified temperature(from 27 to 77 c) and the LED is ON with the decrement of timer every one second.
he LCD displays the reminding time, temperature, and how to pause or stop the heating.
Note: during the heating process, the user can only change the temperature and pause the heating process.
if the user pauses the heating, the user can open the door and see the food.
If the heating is paused, the user can stop the heating process and return to idle state for new insertion or can continue the reminding time if the door is closed and something exists inside the microwave.

3- the end of heating state: in this state the LED is off and the buzzer blanks to alarm the user for this state and the LCD blanks the end message.

Note: this state will finish when the user open the door or press on the key in the end message.

The tasks in this system:

1- LCD task.	(repeated every 30 ms)	(Need to 18.2 ms to finish)
2- Keypad task.	(repeated every 30 ms)	(Need to 2.1 ms to finish)
3- Door sensor task.	(repeated every 30 ms)	(Need to 0.1 ms to finish)
4- weight sensor task.	(repeated every 30 ms)	(Need to 0.1 ms to finish)
5- temperature task.	(repeated every 30 ms)	(Need to 0.7 ms to finish)
6- LED_Buzzer task.	(repeated every 30 ms)	(Need to 0.03ms to finish)
7- welcome message task.	(occur one time)	
8- Home screen task.	(occur one time)	

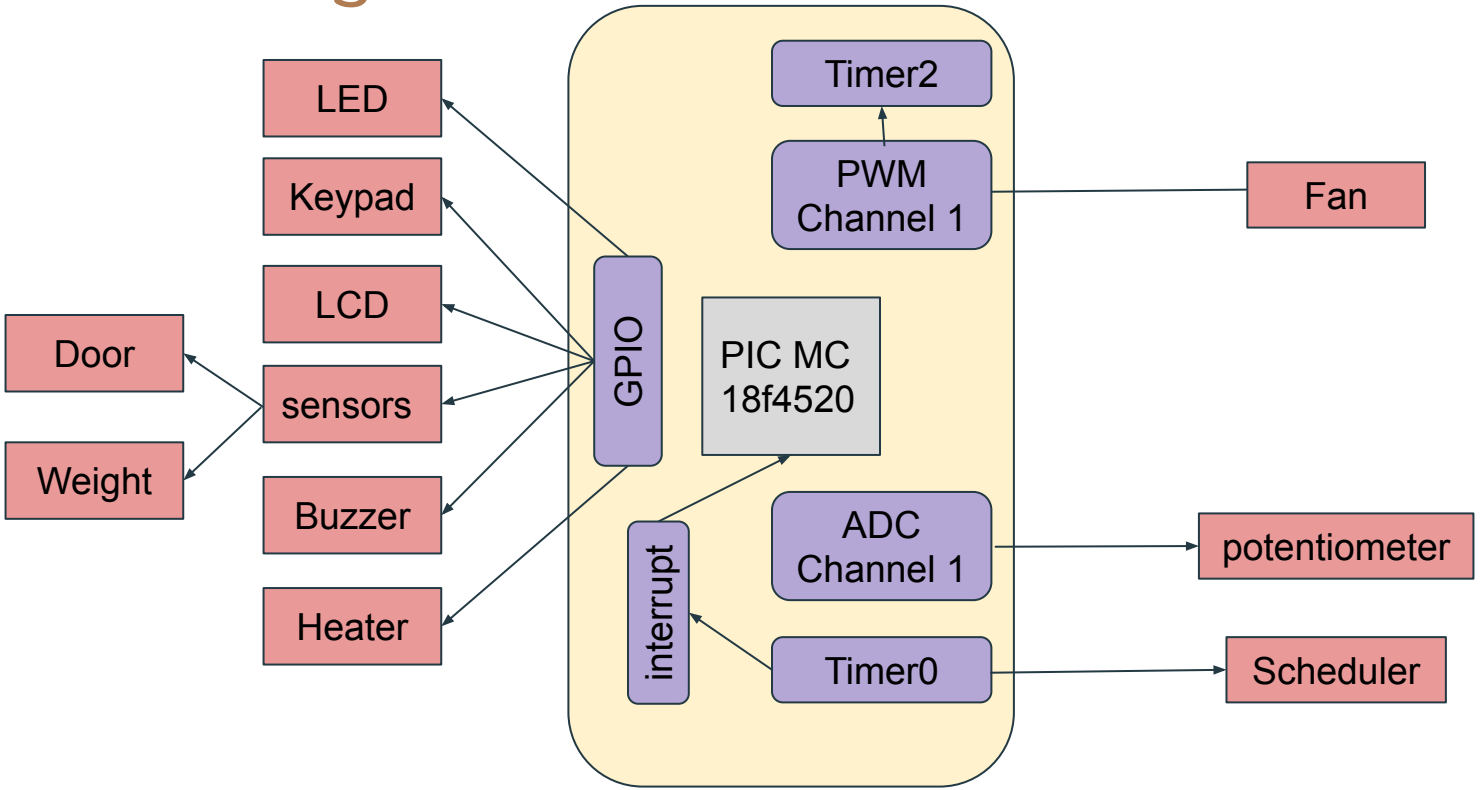
Notes: if the design make only one task execute each tick time, so the total time for all of those tasks is 26 tick

So all of tasks will repeat every 30 tick or 30 ms (i use 1 ms for the tick time in this application).

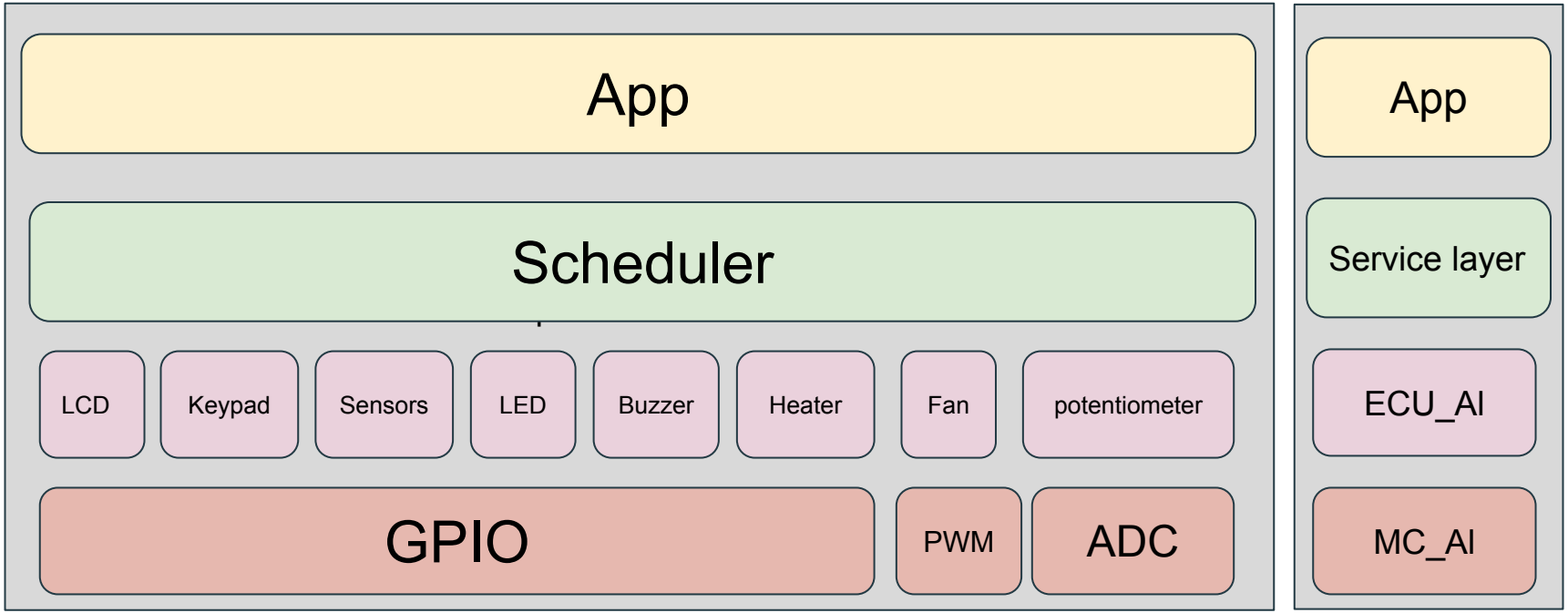
We can sleep the cpu when no task to execute to save power consumption.

When the microwave change its state, the LCD need to more than 50 ms to display new screen but that occurs few times and the app isn't critical to design this system with this case.

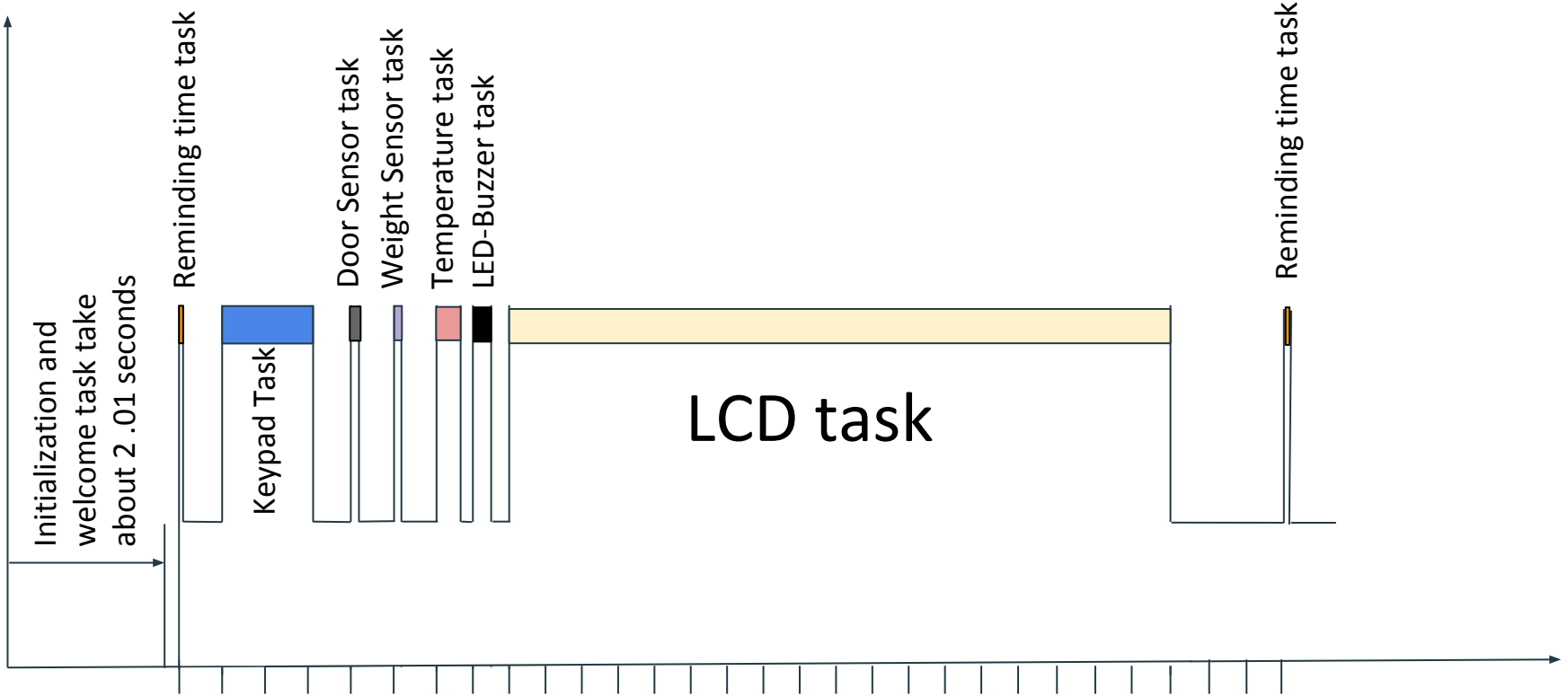
Block Diagram:



Static Design



Dynamic Design/Timeline



ADC Module:

Function	Type
void ADC_Initialization (EnumADC_channels_t)	Initialization
void ADC_Select_acquisition_time(EnumADC_acquisition_Time_t)	Global function
void ADC_Select_prescaler(EnumADC_prescaler_t)	Global function
uint16 ADC_Read_value(void)	Global function
void ADC_Start_conversion(void)	Global function

void ADC_Initialization (EnumADC_channels_t)	
initialize ADC module with specific channel that is selected by the user.	
EnumADC_channels_t	typedef for all ADC channels that can be selected

void ADC_Select_acquisition_time(EnumADC_acquisition_Time_t)	
this function allow the user to select the acquisition time to invoke automatically or set by user calculated delay function.	
EnumADC_acquisition_Time_t	typedef for all available ADC acquisition values

void ADC_Select_prescaler(EnumADC_prescaler_t)	
this function allow the user to select the clock frequency for ADC module by division system clock on selected pre_scaler value.	
EnumADC_prescaler_t	typedef for all available ADC pre_scaler values

uint16 ADC_Read_value(void)	
this function return the value of signal that is converted by ADC.	
Return	measured value will be in 10 bit

void ADC_Start_conversion(void)
start the ADC sampling process on initialized channel.

GPIO Module:

Function	Type
EnumGPIO_Status_t GPIO_SetPinDirection(uint8 a_u8port , uint8 a_u8pin , uint8 a_u8direction)	Global function
EnumGPIO_Status_t GPIO_WriteOnPin(uint8 a_u8port , uint8 a_u8pin , uint8 a_u8value)	Global function
uint8 GPIO_ReadFromPin(uint8 a_u8port , uint8 a_u8pin)	Global function
void GPIO_Disable_Comparator_On_PORTA(void)	Global function
EnumGPIO_Status_t GPIO_Disable_ADC_On_Pins(uint8 a_u8port , uint8 a_u8pin)	Global function
void GPIO_Enable_Pull_Up_On_PortB(void)	Global function

EnumGPIO_Status_t GPIO_SetPinDirection(uint8 a_u8port , uint8 a_u8pin , uint8 a_u8direction)	
Select direction of GPIO pin	
uint8 a_u8port	Select port
uint8 a_u8pin	Select pin
uint8 a_u8direction	Select pin direction (INPUT-OUTPUT)
Return EnumGPIO_Status_t	Status to check if the function execute correctly

void GPIO_Enable_Pull_Up_On_PortB(void)
Disable all weak pull up resistors on PortB's Pins.

EnumGPIO_Status_t GPIO_WriteOnPin(uint8 a_u8port , uint8 a_u8pin , uint8 a_u8value)	
Write value on GPIO pin.	
uint8 a_u8port	Select port
uint8 a_u8pin	Select pin
uint8 a_u8value	Value out on the selected pin (HIGH-LOW)
Return EnumGPIO_Status_t	Status to check if the function execute correctly

void GPIO_Disable_Comparator_On_PORTA(void)
Disable Comparator peripheral on PortA pins.

uint8 GPIO_ReadFromPin(uint8 a_u8port , uint8 a_u8pin)	
Read value from GPIO pin.	
uint8 a_u8port	Select port
uint8 a_u8pin	Select pin
Return uint8	Input value on the pin(HIGH-LOW)

EnumGPIO_Status_t GPIO_Disable_ADC_On_Pins(uint8 a_u8port , uint8 a_u8pin)	
Disable Comparator peripherals on selected pins from PortA, PortB and PortE.	
uint8 a_u8port	Select port
uint8 a_u8pin	Select pin
Return uint8	Status to check if the function execute correctly

Keypad Module:

Function	Type
void KeyPad_Initialization(void)	Initialization
uint8 KeyPad_get_PressedKey_with_debouncing(void)	Global function
uint8 KeyPad_getPressedKey(void);	Private function
uint8 KeyPad_switch(uint8 a_u8row , uint8 a_u8col)	Private function
void KeyPad_Task(void)	Periodic task

void KeyPad_Initialization(void)	
Initialize KeyPad with Configured mode	

uint8 KeyPad_get_PressedKey_with_debouncing(void)	
get pressed key from keypad with avoid bouncing effect.	
Return	Value of pressed key

uint8 KeyPad_getPressedKey(void);	
Return value of pressed key	
Return	Value of pressed key

uint8 KeyPad_switch(uint8 a_u8row , uint8 a_u8col)	
Static function to return value of Pressed Key from user definition values.	
uint8 a_u8row	Row of the pressed key.
uint8 a_u8col	Column of the pressed key.
Return	Value of pressed key

void KeyPad_Task(void)
Read time from the user and start the heating process or cancel it by specified keys.

LCD Module:

Function	Type
void LCD_Initialization (void)	Initialization
void LCD_SendCommand(uint8 a_u8command)	Global function
void LCD_SendDataByte(uint8 a_u8data)	Global function
void LCD_SendDataString(uint8* a_u8data_ptr)	Global function
void LCD_SetDisplayPosition(uint8 a_u8position_X , uint8 a_u8position_Y)	Global function
void LCD_DisplaNumber(sint32 a_s32number)	Global function
void LCD_ClearScreen(void)	Global function

Function	Type
void LCD_Task(void)	Periodic task
void Welcome_screen_Task(void)	Aperiodic task
void Home_screen_display(void)	Aperiodic task

void LCD_Initialization (void)
Initialize LCD by set directions of LCD pins

void LCD_SendCommand(uint8 a_u8command)	
send commands to LCD controller.	
uint8 a_u8command	The command (one of available command in data sheet).

void LCD_SendDataByte(uint8 a_u8data)	
send Data to LCD controller to display on screen (only one byte).	
uint8 a_u8data	The character that will be displayed on LCD.

LCD_SendDataString(uint8* a_u8data_ptr)	
send Data to LCD controller to display on screen (more than one byte).	
uint8* a_u8data_ptr	The string that will be displayed on LCD.

void LCD_DisplaNumber(sint32 a_s32number)	
send a number to LCD controller to display on screen.	
sint32 a_s32number	The number that will be displayed on LCD.

void LCD_SetDisplayPosition(uint8 a_u8position_X , uint8 a_u8position_Y)	
set the position for next character which will be displayed on LCD.	
uint8 a_u8position_X	The column position (from 0 to 15)
uint8 a_u8position_Y	The row position (from 1 to 4)

void LCD_ClearScreen(void)
clean LCD screen and go to the first position on LCD screen to display the next character.

void LCD_Task(void)
Display on LCD screen at any state of microwave operations.

void Home_screen_display(void)
Display home screen or the state of door and weight and time and temperature insertion before start of the heating process.

void Welcome_screen_Task(void)
Display the welcome message.

PWM Module:

Function	Type
void PWM_Initialization(EnumPWM_channel_t a_PWM_channel_t)	Initialization
void PWM_Start(EnumPWM_channel_t a_PWM_channel_t)	Global function
void PWM_Stop(EnumPWM_channel_t a_PWM_channel_t)	Global function
void PWM_Frequency(uint32 a_u32PWM_frequency)	Global function
void PWM_DutyCycle(EnumPWM_channel_t a_PWM_channel_t , uint8 a_u16PWM_duty_cycle)	Global function

void PWM_Initialization(EnumPWM_channel_t a_PWM_channel_t)	
Initialize PWM by configure timer 2 and set direction of selected channel as output pin.	
EnumPWM_channel_t	typedef for available channels in this micro_controller.

void PWM_Start(EnumPWM_channel_t a_PWM_channel_t)	
make PWM start working.	
EnumPWM_channel_t	typedef for available channels in this micro_controller.

void PWM_Stop(EnumPWM_channel_t a_PWM_channel_t)	
Stop PWM module.	
EnumPWM_channel_t	typedef for available channels in this micro_controller.

void PWM_Frequency(uint32 a_u32PWM_frequency)	
set the frequency of PWM signal.	
uint32 a_u32PWM_frequency	Frequency of PWM signal.

void PWM_DutyCycle(EnumPWM_channel_t a_PWM_channel_t , uint8 a_u16PWM_duty_cycle)	
make PWM pin for specified channel work with specific duty cycle.	
EnumPWM_channel_t	typedef for available channels in this micro_controller.
uint8 a_u16PWM_duty_cycle	Duty cycle value (from 0 to 100).

Timer 0 Module:

Function	**Type**
void Timer0_Initialization(void)	Initialization
void Timer0_enable(void)	Global function
void Timer0_write_counter(uint16 a_u16init_value_for_timer)	Global function

void Timer0_Initialization(void)
Initialize timer 0 with specified configuration in configuration structure.Initialize timer 0 with specified configuration in configuration structure.

void Timer0_enable(void)
enable timer 0 to start counting.

void Timer0_write_counter(uint16 a_u16init_value_for_timer)	
write user value in timer 0 counter to start count from this number.	
uint16 a_u16init_value_for_timer	user number that will be initialize timer counter.

Timer 2 Module:

Function	**Type**
void Timer2_Initialization(void)	Initialization
void Timer2_enable(void)	Global function
void Timer2_write_period_register(uint8 a_u8init_value_for_period_register)	Global function

void Timer2_Initialization(void)
Initialize timer 2 with specified configuration in configuration structure.

void Timer2_enable(void)
enable timer 2 to start counting.

void Timer2_write_period_register(uint8 a_u8init_value_for_period_register)	
write the match value that the timer 2 counter will reset when matching with it.	
uint8 a_u8init_value_for_period_register	match value to reset the timer 2 counter

Scheduler Module:

Function	Type
EnumScheduler_Error_t Scheduler_Initialization(TickTime a_u8tickTme)	Initialization
EnumScheduler_Error_t Scheduler_StartFunction(void (*task_function_ptr) (void), uint8 a_u8task_ID, uint16 a_u16task_periodicity , uint16 a_u16task_start_time , EnumScheduler_RepeatedFunction_t a_RepeatType_t)	Global function
EnumScheduler_Error_t Scheduler_StopFunction(uint8 a_u8_task_ID)	Global function
void Scheduler_Dispatcher(void)	Global function
void Scheduler_Core(void)	Private function
EnumScheduler_Error_t Scheduler_Tick_time_calculation(TickTime a_u8tickTme)	Private function

EnumScheduler_Error_t Scheduler_Initialization(TickTime a_u8tickTme)	
Initialize the Scheduler Driver with specific tick time.	
TickTime a_u8tickTme	typedef from u8 with specific values of tick time
Return EnumScheduler_Error_t	Return error if the specified tick time isn’t suitable with limitation of clock source and timer0 .

EnumScheduler_Error_t Scheduler_StopFunction(uint8 a_u8_task_ID)	
Remove Function from Scheduler Queue	
uint8 a_u8_task_ID	Function's ID that will stop to run again.
Return EnumScheduler_Error_t	Return error if the task isn’t exist in the queue.

EnumScheduler_Error_t Scheduler_StartFunction(void (*task_function_ptr) (void), uint8 a_u8task_ID , uint16 a_u16task_periodicity , uint16 a_u16task_start_time , EnumScheduler_RepeatedFunction_t a_RepeatType_t)	
Add Function to Scheduler Queue	
void (*task_function_ptr) (void)	Pointer to function that will add to queue.
uint8 a_u8_task_ID	Function's ID that will start to run again.
uint16 a_u16task_periodicity	Periodicity of the function to execute again.
uint16 a_u16task_start_time	Time for the first execution for this function.
EnumScheduler_RepeatedFunction_t	Type of this function (REPEATED- ONCE).
Return EnumScheduler_Error_t	Return error if the queue is full.

void Scheduler_Dispatcher(void)
Run the ready functions.

void Scheduler_Core(void)
Call Back Function for Timer ISR to check which task get the time to execute.

EnumScheduler_Error_t Scheduler_Tick_time_calculation(TickTime a_u8tickTme)	
select suitable initialization of timer to get the specified tick time.	
TickTime a_u8tickTme	typedef from u8 with specific values of tick time
Return EnumScheduler_Error_t	Return error if the specified tick time isn’t suitable with limitation of clock source and timer0 .

Tasks without specific Module:

Function	Type
void initialization_Tasks(void)	Aperiodic task
void Temperature_Task (void)	Periodic task
void Door_Sensor_Task(void)	Periodic task
void Weight_Sensor_Task(void)	Periodic task
void LED_Buzzer_Task(void)	Periodic task
void Reminding_time_Task(void)	Periodic task

void initialization_Tasks(void)
initialize all tasks to put them in scheduler queue.

void Temperature_Task (void)
set and update microwave temperature using ADC, PWM and GPIO modules.

void Door_Sensor_Task(void)
get microwave door sensor state.

void Weight_Sensor_Task(void)
get microwave weight sensor state.

void LED_Buzzer_Task(void)
set and update microwave LED & Buzzer.

void Reminding_time_Task(void)
update the reminding time for heating operation.

Comments:

1- this program based on time trigger design but i worked on event trigger to see the different and i got the same result.
The only difference is that, we can make CPU sleep in the time trigger code when the CPU doesn't execute any task and that reduce power consumption but with event trigger we can't make CPU sleep because we should be in while one forever.
Note: i submited the event trigger program too.

2- i think the PICSimLab has some problems with Keypad in PICGenios board simulation:
The code should prevent multi reading for long press but that happened with column 3 only but column 1 and 2 didn't work correctly with long press.
And, only column 1 sometimes doesn't respond, so i think it is an issue with the program because all of this issues doesn't exist with proteus simulation or hardware simulation.
I think you can see that problem with the led that is connected with PB0 (the same of column 1) doesn't work like the others LEDs (the LED is not working with my code or with the test code of the program itself).