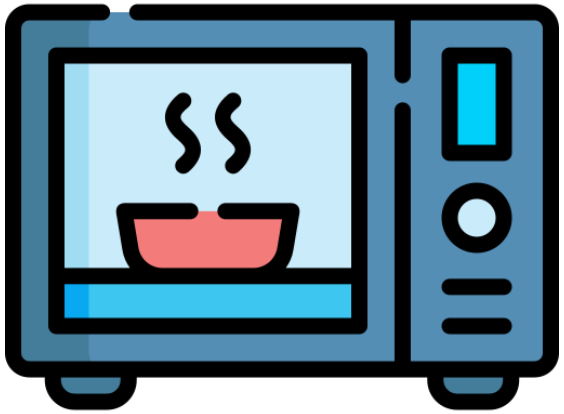
Microwave Oven System Task

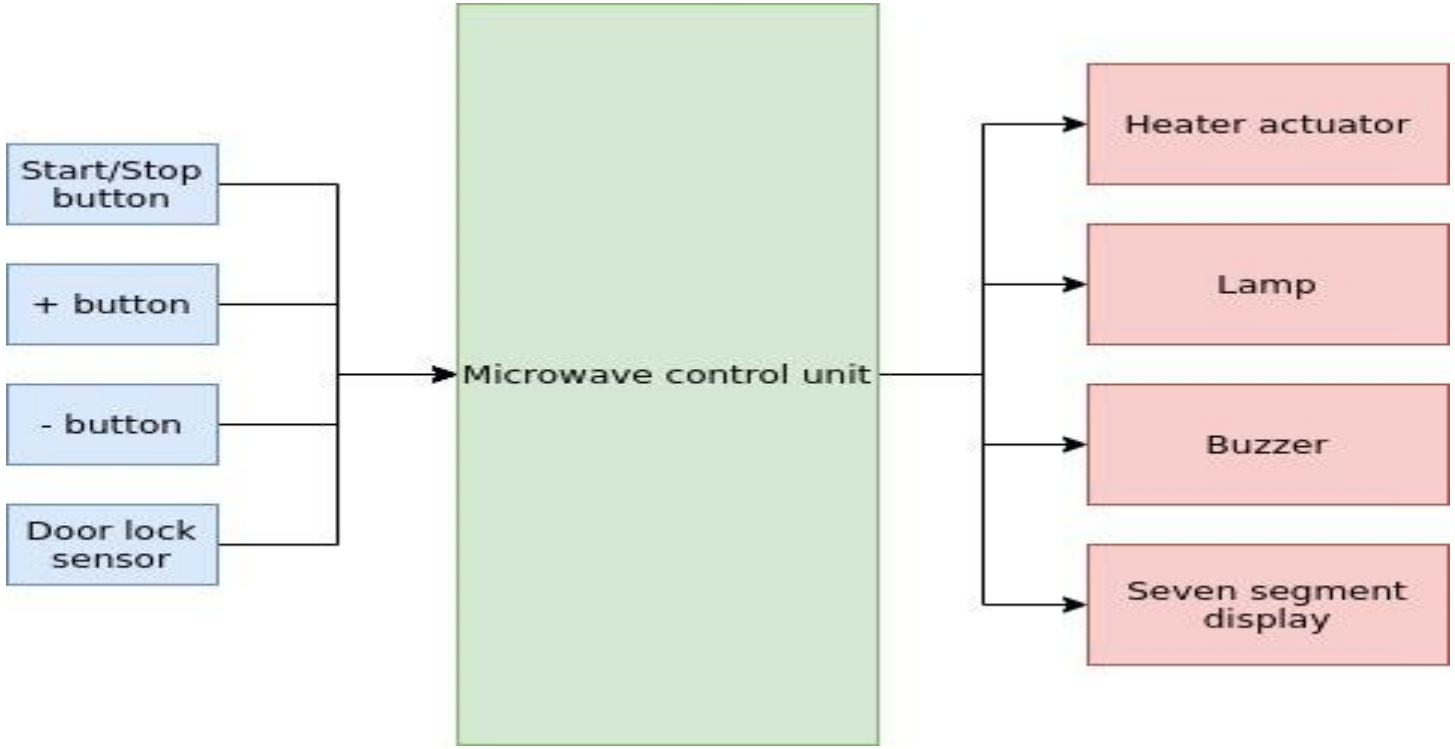
GARMENT IO

Smart manufacturing made simple.



- By: Mahmoud Saad Abd-Elhares Mahmoud
- E-Mail: Mahmoud.S.AbdElhares@gmail.com
- LinkedIn: linkedin.com/in/mahmoudsaad96
- Git-Hub: https://github.com/ZeroX96

Task: Design and implement an embedded software using FreeRTOS for a microwave oven.



System Requirements:

1. **Buzzer beeps 100ms at each button press.**
2. **The heater actuator requires 60% ,50ms period PWM signal in order to be turned on.**

1) When it's powered on :

a. Buzzer beeps 100ms

b. Seven segment display is initialized with zeros

2) When heating is off :

a. + and - buttons are used to set the required heating time with 5 second step

b. Lamp is: i. on when door is open
ii. off when door is closed

c. Seven segment displays the time setting while blinking 300ms on and 300ms off after the first change from zeros

d. Start/stop button start heating only if door is closed

4) When heating is done:

a. lamp is off

b. Heater is off

c. Buzzer beeps two times 100ms on and 100ms off

3) When heating is on:

a. + button is used to increment the required heating time with 5 second step

b. Seven segment displays time remaining (no blinking)

c. lamp is on

d. Heater is on

e. Start/stop button stop heating

System Assumptions:

1) **The System-Tick is set to be 5ms each.**
   a. To Achieve the illusion to the human eyes displaying the time settings on the SSD while Multiplexing
   b. The buzzer beeps 100ms, The system 7-Segments are Blinked ever 300ms, and, The usual user button pressing time is about 100ms, and all of them are divisible by the SysTick Value of 5.
   c. So, to reduce the kernel invocation overhead on the system, the Systick value is set to be 5ms.

2) **The number of Seven-Segments Displays is set to be 4.**
   a. To Give the user a good range from 0-Seconds up to N-Seconds, N is defined in the header-file.

3) **The PWM signal that will be driving the heater, will be working at a frequency of 20-hz.**
   a. Given the required period to enable the Heater to be 50-ms.

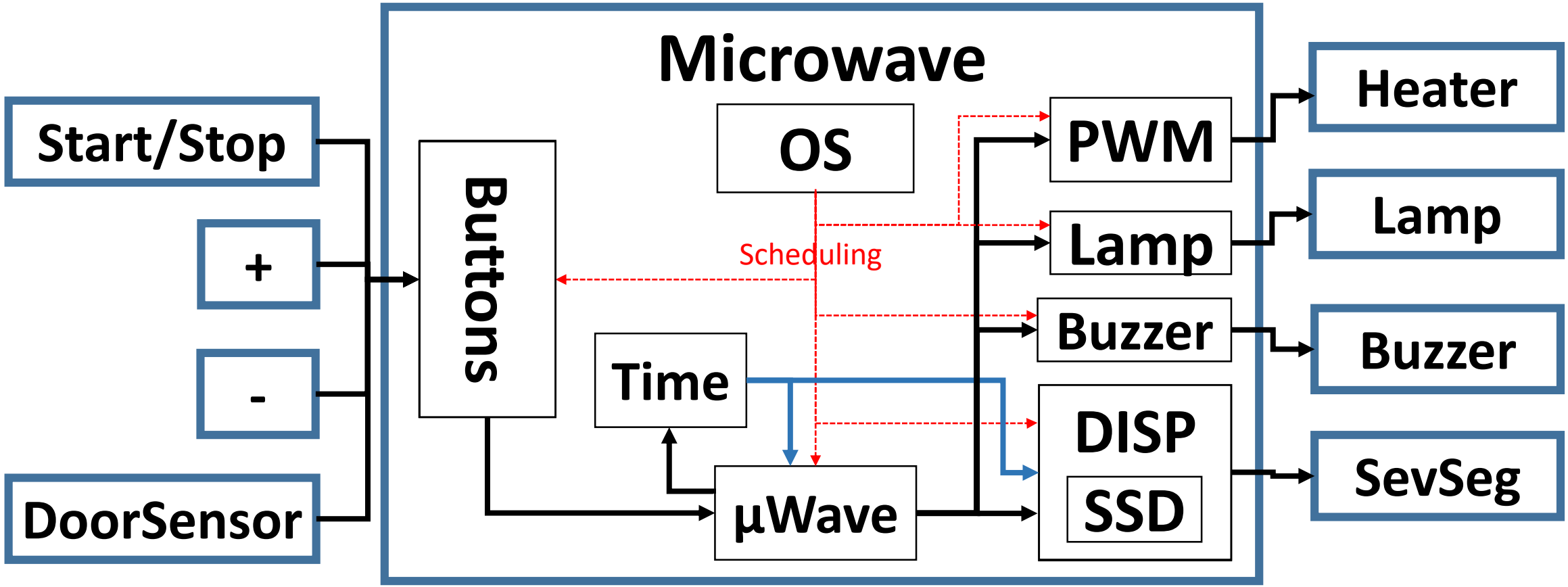
4) **The Buttons Scanning Rate is set to be once every 20ms. And, 2-Samples are got to ensure a user press.**
   a. Given the usual user press time is about 100ms, scanning every 20ms is enough and to reduce the overhead.

5) **The Time Counter is based on seconds and don't follow the hour/minute/second scheme**
   a. For example 12:34 not equal 12 minutes but, 1234 Seconds

6) **If the door gets opened while heating is ON, the Heater is Turned-Off, Lamp is Turned-On and System is PAUSED**
   a. If Door is Open while Heating is OFF, Lamp is Turned-On (as Given in the document)
   b. If Door is Open while Heating is ON, Heater is Turned-Off and Lamp is Turned-On (the Assumption)
   c. While the system is paused, if the door is closed, system continues.
   d. While the system is paused, If start button is pressed, the system is turned-off.

Microwave: Static Architecture

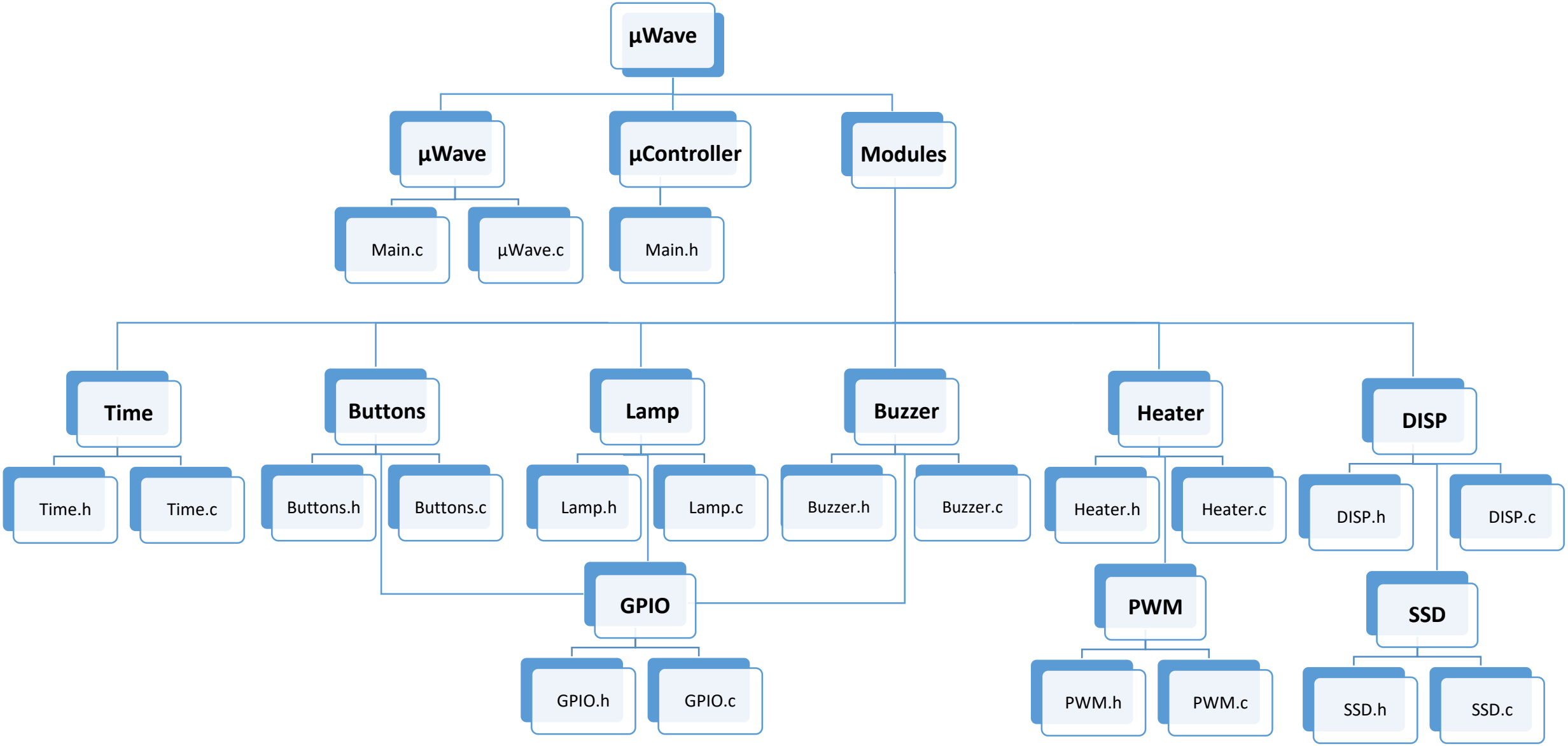
"Assuming the Door Sensor is just another button in the system"



Microwave: Detailed Static Design

- OS
  - xTaskCreate
  - vTaskStartScheduler
  - xTaskGetTickCount
  - vTaskDelayUntil
  - vTaskDelete
- Buzzer
  - BuzzInit
  - BuzzSetState
  - BuzzUpdate
- Microwave
  - MicroWaveInitState
  - MicroWaveSetState
  - MicroWaveGetState
- TIME
  - TimeInit
  - TimeIncTime / TimeDecTime
  - TimeSetTime / TimeGetTime
  - TimePauseTime / TimeResumeTime
  - TimeUpdate
- Heater
  - HeaterInit
  - HeaterSetState
  - HeaterGetState
  - HeaterUpdate
- PWM
  - PwmInit
  - PwmRun
  - PwmStop
  - PwmDeInit
- Buttons & Door
  - BtnInit
  - DoorSensInit
  - BtnGetState
  - DoorSensGetState
  - BtnUpdate
  - DoorSensUpdate
- DISP
  - DispInit
  - DispSetVal
  - DispUpdate
- SSD
  - DisplayInit
  - DisplayWrite
  - DisplayReset
  - DisplayDeInit
  - DisplayMuxInit
  - DisplayMuxWrite
  - DisplayMuxReset
  - DisplayMuxDeInit
- Lamp
  - LampInit
  - LampSetState
  - LampGetState
  - LampUpdate

Needed Files for The Project



Microwave: Project Partitioning

- Following the Divide and Conquer Algorithm.
- Dividing and Conquering is based on:
  - Functions Periodicity and Time-Cohesion.
  - Functions Nature (IO Bound – CPU Bound).

Divide and Conquer				
Modules	**Nature**		**Periodicity**	**Task/Group**
Buttons	**IO Bound**	**Input**	**20 ms**	**INPUTS TSK**
Time	**CPU Bound**	**Timing**	**1000 ms**	**TIME TASK**
Heater – Lamp - Buzzer	**IO Bound**	**Output**	**10 ms**	**OUTPUTS TASK**
Display	**IO Bound**	**Output**	**5 ms**	**DISPLAY TASK**

Microwave:
Dynamic Design and Schedulability Check

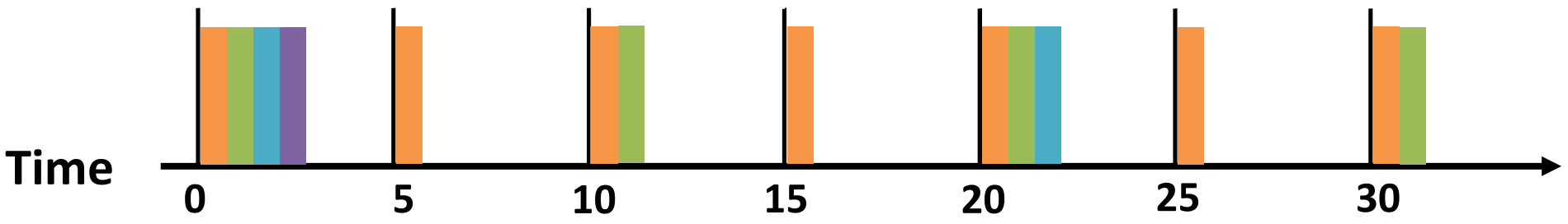
DISP Task @ 5 ms

TIME Task @ 1000 ms

INPUTS Task @ 20 ms

OUTPUTS Task @ 10 ms

<u>Major Cycle:</u> 1000 ms.



Microwave: Dynamic Design and Priorities Assignment

Rate Monotonic Scheduling, RMS:

The tasks with the highest frequencies, are given the highest priorities.

Following RMS Scheduling in assigning Task Priorities		
Rate (Once Every)	Priority	Task
1000 (ms)	Lowest	TIME
20 (ms)	Lower	INPUTS
10 (ms)	Higher	OUTPUTS
5 (ms)	Highest	DISP

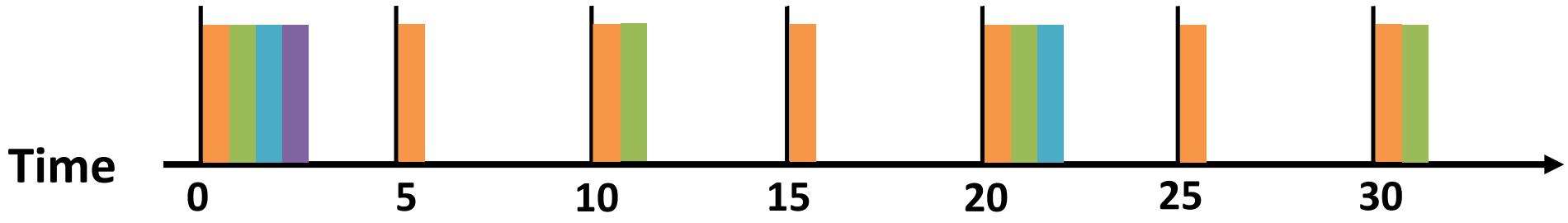
DISP Task @ 5 ms

TIME Task @ 1000 ms

INPUTS Task @ 20 ms

OUTPUTS Task @ 10 ms

Major Cycle: 1000 ms.

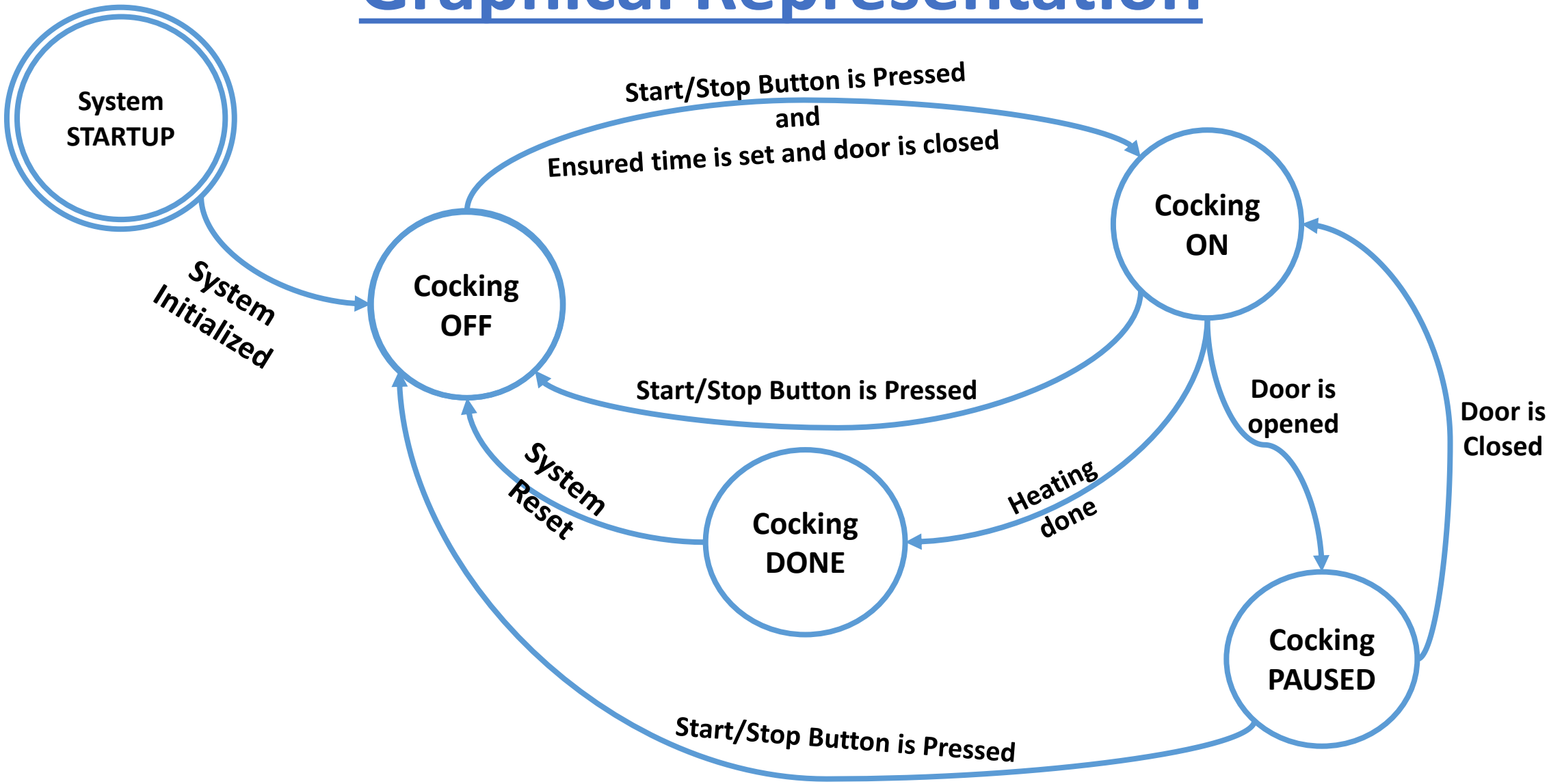


Microwave : Timing Analysis

0- The System is running at 8-MHz with nearly all the instructions takes one or two clock-cycles.
1- I Have No Delays inside all the Tasks and the Modules in the System hence, Nearly all tasks Execution Time = 0 ms.

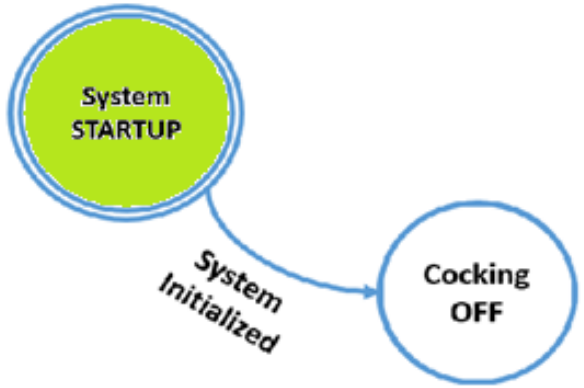
Task	Actions	BCET (ms)	WCET(ms)	Period of Action(ms)	Period of task (ms)
Time	Update Time Update System state	~0 ~0	~0 ~0	1000 1000	1000
Inputs	Update Switches States Update System State	~0 ~0	~0 ~0	20 20	20
Outputs	Update Outputs	~0	~0	10	10
Display	Update SSD	~0	~0	5	5
				Tick (ms)	5
				Major Cycle (ms)	1000

System State Machine Graphical Representation



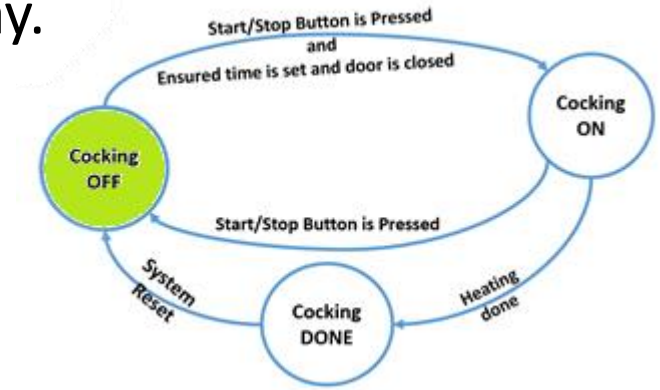
State: System Startup

- The initial state that the system wakes-up to.
- The system shows all zeros on the display
- The system buzzer beebs for 10ms.
- The heater is turned-off.
- The lamp is turned-off.



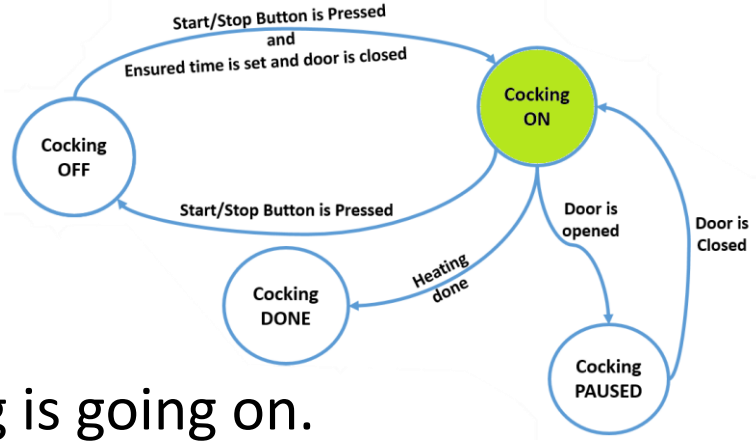
State: System Cocking-OFF

- The default state that the system stays in after startup, after finishing the cocking operation, or after a Start/Stop Button Pressed while the system was running.
- The system shows the current time settings value on the display.
- The system buzzer beebs for 10ms.
- The heater is turned-off.
- The lamp is turned-off.



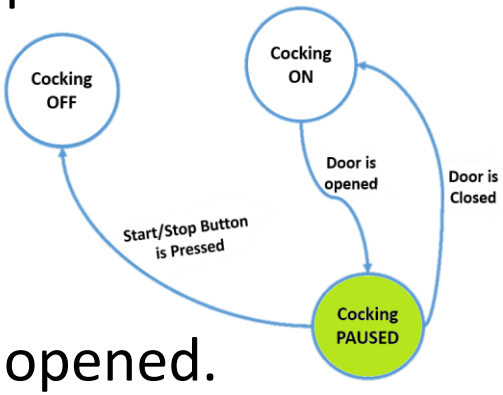
State: System Cocking-ON

- The state that the system performs its main task during it.
- The system shows the remaining-time for the cocking to be done on the display
- The system responds only to the + Button and neglects the - Button.
- The heater is turned-on.
- The lamp is turned-on.



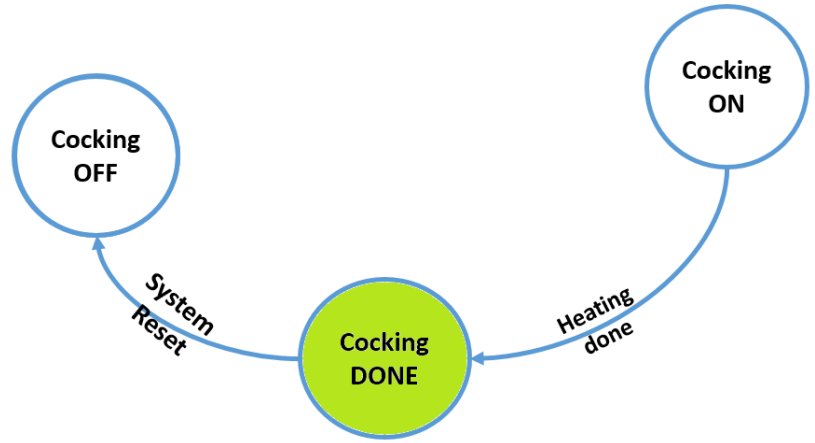
State: System Cocking-Paused

- The state of the system if the door is opened while cocking is going on.
- It stays hear as long as the door is opened or the Start/Stop button is pressed.
- The system shows the current time value on the display.
- The system timing is paused
- The heater is turned-off.
- The lamp is turned-off If the door is closed or it’s turned-on if door is opened.

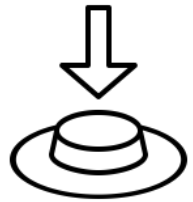


State: System Cocking-Done

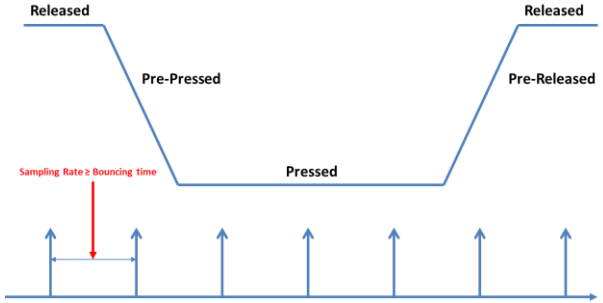
- The state that the system goes through after finishing the cocking operation.
- The system shows the current time value on the display.
- The system buzzer beebs two times for 100ms ON and 100ms OFF.
- The heater is turned-off.
- The lamp is turned-off.
- The system is reset and goes back to the cocking-off state.



System Modules and Functions: Buttons

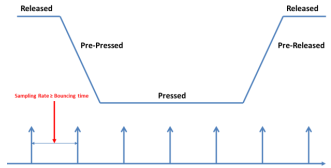


- Handles the system buttons and the resulting states.



Module Buttons	
Function	**Type**
`void BtnInit(BtnSelectT BtnSelect, BtnStateT BtnInitState);`	Initialization
`BtnStateT BtnGetState(BtnSelectT BtnSelect);`	Global Function
`void BtnUpdate(void );`	Periodic Function

System Modules and Functions: Buttons



`void BtnInit(BtnSelectT BtnSelect, BtnStateT BtnInitState);`	
Function initializes the hardware for the system buttons	
`BtnSelect`	Defines which button to initialize
`BtnInitState`	Defines the initial state for the targeted button
Return	NON, Void

`BtnStateT BtnGetState(BtnSelectT BtnSelect);`	
Function gets the current state of the targeted button	
`BtnSelect`	Defines which button to get its state
Return	returns the state of the targeted button

`void BtnUpdate(void );`	
Function updates the current state of all the buttons	
Return	NON, Void

System Modules and Functions: Time

- Handles the system timer set by user to cock the food.

Module Time	
Function	**Type**
`void TimeInit(void);`	Initialization
`void TimeIncTime(void);`	Global Function
`void TimeDecTime(void);`	Global Function
`void TimeSetTime(uint16_t TimeVal);`	Global Function
`void TimeResumeTime(void);`	Global Function
`void TimePauseTime(void);`	Global Function
`uint16_t TimeGetTime(void);`	Global Function
`void TimeUpdate(void);`	Periodic Function

System Modules and Functions: Time

`void TimeInit(void);`	
Function initializes the software for the system timer	
Return	NON, Void

`void TimeIncTime(void);`	
Function increments the system timer with step of 5 s	
Return	NON, Void

`void TimeDecTime(void);`	
Function decrements the timer variable with step of 5 s	
Return	NON, Void

`void TimeSetTime(uint16_t TimeVal);`	
Function updates the timer value as needed by TimeVal	
`TimeVal`	the required value to be set as the system time
Return	NON, Void

System Modules and Functions: Time

`uint16_t TimeGetTime(void);`	
Function returns the system timer variable current value	
Return	the value of the current system time variable

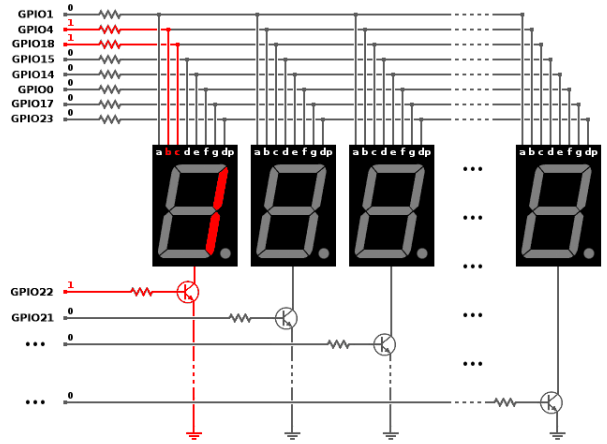
`void TimeResumeTime(void);`	
Function resumes the timer variable and module operation	
Return	NON, Void

`void TimePauseTime(void);`	
Function pauses the timer variable and module operation	
Return	NON, Void

`void TimeUpdate(void);`	
Function updates the timer variable current value and handles the system mode of cocking when cocking is done	
Return	NON, Void

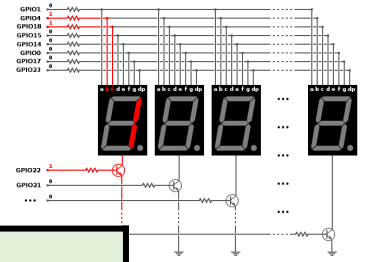
System Modules and Functions: DISPLAY

- Handles the Multiplexed system Seven-Segment Display



Module Display	
Function	**Type**
`void DispInit(void);`	Initialization
`void DispSetVal(uint16_t DataVal);`	Global Function
`void DispUpdate(uint8_t Id);`	Periodic Function

System Modules and Functions: DISPLAY

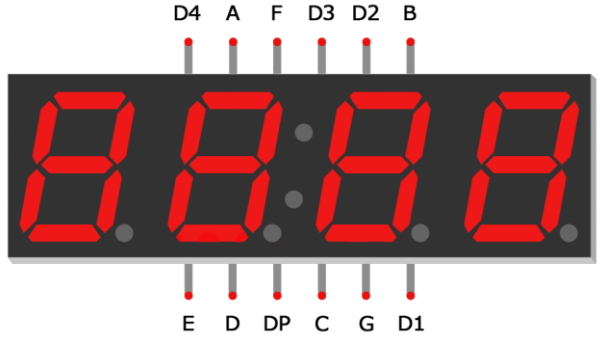


`void DispInit(void);`	
Function initializes the hardware for the system Display	
Return	NON, Void

`void DispSetVal(uint16_t DataVal);`	
Function updates the Display value as needed by DataVal	
`DataVal`	Defines the value to be presented on the display
Return	NON, Void

`void DispUpdate(uint8_t Id);`	
Function updates the display data current shown on the segments	
`Id`	defines which seven segment to turn on and show the data on
Return	NON, Void

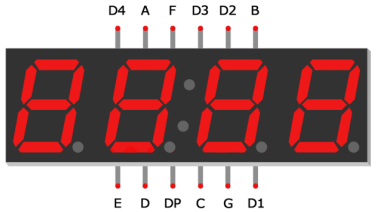
System Modules and Functions: SSD



- Handles the system Seven Segment Displays

Module SSD	
Function	**Type**
`SevenSegRetT DisplayInit(SevenSegT* Display, GpioBaseT DispPort,SevenSegTypeT DispType);`	Initialization
`SevenSegRetT DisplayWrite(SevenSegT* Display,uint8_t OutVal);`	Global Function
`SevenSegRetT DisplayReset(SevenSegT* Display);`	Global Function
`SevenSegRetT DisplayDeInit(SevenSegT* Display);`	Global Function
`SevenSegRetT DisplayMuxInit(SevenSegT* Display,GpioBaseT DataPort,GpioBaseT CtrlPort,SevenSegTypeT DispType);`	Global Function
`SevenSegRetT DisplayMuxWrite(SevenSegT* Display, uint8_t OutDataVal[], uint8_t DispId);`	Global Function
`SevenSegRetT DisplayMuxReset(SevenSegT* Display);`	Global Function
`SevenSegRetT DisplayMuxDeInit(SevenSegT* Display);`	Global Function

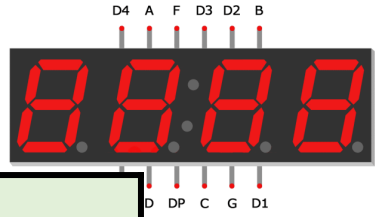
System Modules and Functions: SSD



`SevenSegRetT DisplayInit(SevenSegT* Display, GpioBaseT DispPort,SevenSegTypeT DispType);`	
Function initializes the hardware for the system display	
`Display`	Pointer to the display handler that leads to the display
`DispPort`	Which port is the display connected to on the controller
`DispType`	Defines which type is the display, Common Anode-Cathode
Return	State of the initialization process

`SevenSegRetT DisplayWrite(SevenSegT* Display,uint8_t OutVal);`	
Function updates the currently presented data on the display by the OutVal	
`Display`	Pointer to the display handler that leads to the display
`OutVal`	The value that will be presented on the display
Return	State of the writing process

System Modules and Functions: SSD

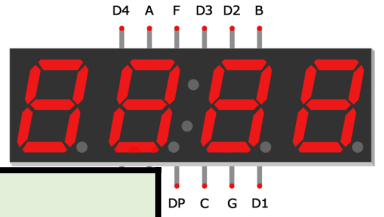


SevenSegRetT DisplayReset(SevenSegT* Display);	
Function Resets the currently presented value to zero	
Display	Pointer to the display handler that leads to the display
Return	The state of the reseting process

SevenSegRetT DisplayDeInit(SevenSegT* Display);	
Function un-initializes the display Module	
Display	Pointer to the display handler that leads to the display
Return	The state of de-initialization process

SevenSegRetT DisplayMuxInit(SevenSegT* Display,GpioBaseT DataPort,GpioBaseT CtrlPort,SevenSegTypeT DispType);	
Function initializes the hardware for the Multiplexed system display	
Display	Pointer to the display handler that leads to the display
DataPort	Which port is the display connected to on the controller
CtrlPort	Which port is the display connected to on the controller
DispType	Defines which type is the display, Common Anode-Cathode
Return	State of the initialization process

System Modules and Functions: SSD



`SevenSegRetT DisplayMuxWrite(SevenSegT* Display, uint8_t OutDataVal[], uint8_t DispId);`	
Function updates the currently presented data on the display by the OutDataVal	
`Display`	Pointer to the display handler that leads to the display
`OutDataVal`	The value that will be presented on the display
`DispId`	defines which seven segment to turn on and show the data on
Return	State of the writing process

`SevenSegRetT DisplayMuxReset(SevenSegT* Display);`	
Function Resets the currently presented value to zero	
`Display`	Pointer to the display handler that leads to the display
Return	State of the initialization process

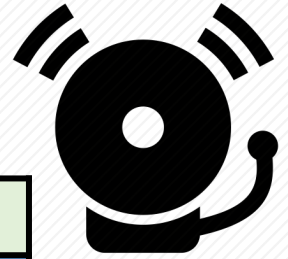
`SevenSegRetT DisplayMuxDeInit(SevenSegT* Display);`	
Function un-initializes the display Module	
`Display`	Pointer to the display handler that leads to the display
Return	State of the initialization process

System Modules and Functions: Buzzer

- Handles the system Buzzer that's used to notify the user.

Module Buzzer	
Function	**Type**
`void BuzzInit(BuzzSelectT BuzzSelect, BuzzStateT BuzzInitState);`	Initialization
`void BuzzSetState(BuzzSelectT BuzzSelect, BuzzStateT BuzzState, int16_t Duration);`	Global Function
`void BuzzUpdate(void );`	Periodic Function

System Modules and Functions: Buzzer



`void BuzzInit(BuzzSelectT BuzzSelect, BuzzStateT BuzzInitState);`	
Function initializes the hardware for the system buzzer	
`BuzzSelect`	Defines which Buzzer to initialize
`BuzzInitState`	Defines the initial state for the targeted Buzzer
Return	NON, Void

`void BuzzSetState(BuzzSelectT BuzzSelect, BuzzStateT BuzzState, int16_t Duration);`	
Function sets the state of the targeted buzzer for a specified duration	
`BuzzSelect`	Defines which button to set its state
`BuzzState`	Defines the state for the targeted Buzzer
`Duration`	Defines the duration needed to put the buzzer in the new state
Return	NON, Void

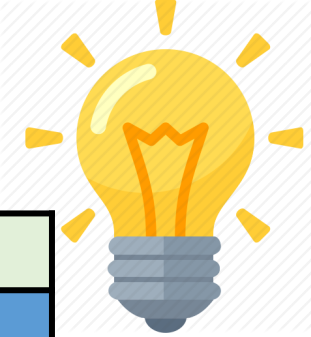
`void BuzzUpdate(void );`	
Function updates the current state of all the buttons	
Return	NON, Void

System Modules and Functions: Lamp

- Handles the system Lamp that's used to Light up the system.

Module Lamp	
Function	**Type**
`void LampInit(LampSelectT LampSelect, LampStateT LampInitState);`	Initialization
`void LampSetState(LampSelectT LampSelect, LampStateT LampState);`	Global Function
`BtnStateT LampGetState(LampSelectT LampSelect);`	Global Function
`void LampUpdate(void );`	Periodic Function

System Modules and Functions: Lamp



`void LampInit(LampSelectT LampSelect, LampStateT LampInitState);`	
Function initializes the hardware for the system Lamps	
`LampSelect`	Defines which Lamp to initialize
`LampInitState`	Defines the initial state for the targeted Lamp
Return	NON, Void

`void LampSetState(LampSelectT LampSelect, LampStateT LampState);`	
Function sets the state of the targeted Lamp	
`LampSelect`	Defines which Lamp to set its state
`LampState`	Defines the state for the targeted Lamp
Return	NON, Void

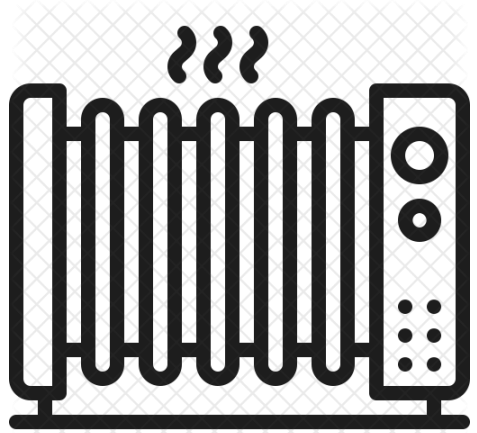
`BtnStateT LampGetState(LampSelectT LampSelect);`	
Function sets the state of the targeted Lamp	
`LampSelect`	Defines which Lamp to get its state
Return	Returns the current state for the targeted Lamp

System Modules and Functions: Lamp

`void LampUpdate(void );`	
Function updates the current state of all the Lamps in the system	
Return	NON, Void

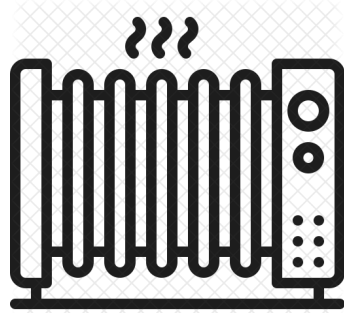
System Modules and Functions: Heater

- Handles the system Heaters.



Module Heater	
Function	**Type**
`void HeaterInit(HeaterSelectT HeaterSelect, HeaterStateT HeaterState);`	Initialization
`void HeaterSetState(HeaterSelectT HeaterSelect, HeaterStateT HeaterState);`	Global Function
`HeaterStateT HeaterGetState(HeaterSelectT HeaterSelect);`	Global Function
`void HeaterUpdate(void );`	Periodic Function

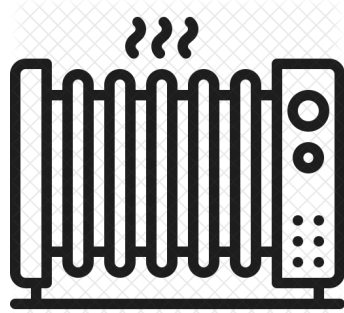
System Modules and Functions: Heater



`void HeaterInit(HeaterSelectT HeaterSelect, HeaterStateT HeaterState);`	
Function initializes the hardware for the system heater	
`HeaterSelect`	Defines which heater to initialize
`HeaterState`	Defines the initial state for the targeted heater
Return	NON, Void

`void HeaterSetState(HeaterSelectT HeaterSelect, HeaterStateT HeaterState);`	
Function sets the state of the targeted heater	
`LampSelect`	Defines which heater to set its state
`LampState`	Defines the state for the targeted heater
Return	NON, Void

System Modules and Functions: Heater



`HeaterStateT HeaterGetState(HeaterSelectT HeaterSelect);`	
Function sets the state of the targeted heater	
`LampSelect`	Defines which heater to get its state
Return	Returns the current state for the targeted heater

`void HeaterUpdate(void );`	
Function updates the current state of all the Heaters in the system	
Return	NON, Void

System Modules and Functions: PWM

- Handles the system PWM Driver used to handle the System Heaters.

Module PWM	
Function	**Type**
`PwmRetT PwmInit(PwmT* Pwm, PwmBaseT PwmBase, PwmOutputModeT PwmOutputMode, PWmDutyCycleT PWmDutyCycle, PwmOperatingModeT PwmOperatingMode, uint32_t PwmFrequency);`	Initialization
`PwmRetT PwmRun(PwmT* Pwm);`	Global Function
`PwmRetT PwmStop(PwmT* Pwm);`	Global Function
`PwmRetT PwmDeInit(PwmT* Pwm);`	Global Function

System Modules and Functions: PWM


```
PwmRetT PwmInit(PwmT* Pwm, PwmBaseT PwmBase, PwmOutputModeT PwmOutputMode,
                PWmDutyCycleT PWmDutyCycle, PwmOperatingModeT PwmOperatingMode,
                uint32_t PwmFrequency);
```

Function initializes the hardware for the system PWM	
Pwm	Pointer to the PWM Module handler that controls the heater
PwmBase	Which PWM source is the heater connected to on the controller
PwmOutputMode	Defines the output mode of the Signal if Inverted or not
PWmDutyCycle	Defines the duty cycle needed from the pwm driver on the source
PwmOperatingMode	Defines the operating mode if it’s Phase-correct or Fast PWM
PwmFrequency	The frequency needed that the output signal to be operating on
Return	State of the initialization process

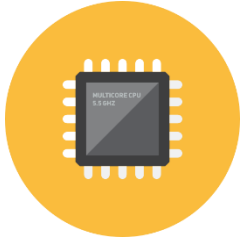
System Modules and Functions: PWM

PwmRetT PwmRun(PwmT* Pwm);	
Function enables the signal to be output on the signal pin and connected to the heater	
Pwm	Pointer to the PWM Module handler that controls the heater
Return	State of the Runing process

PwmRetT PwmStop(PwmT* Pwm);	
Function disables the signal to be output on the signal pin and connected to the heater	
Pwm	Pointer to the PWM Module handler that controls the heater
Return	State of the Stopping process

PwmRetT PwmDeInit(PwmT* Pwm);	
Function un-initializes the hardware module that generates the pwm signal connected to the heater	
Pwm	Pointer to the PWM Module handler that controls the heater
Return	State of the un-initialization process

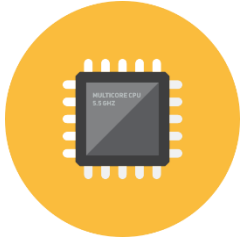
System Modules and Functions: GPIO



- Handles the system GPIO Driver used to handle the System Inputs & Outputs.

Module GPIO	
Function	**Type**
`GpioRetT GpioInitPort(GpioBaseT Base,GpioStateT State);`	Initialization
`GpioRetT GpioInitPin(GpioBaseT Base,GpioPinT Pin,GpioStateT State);`	Global Function
`GpioRetT GpioWritePort(GpioBaseT Base,uint8_t Val);`	Global Function
`GpioRetT GpioWriteGroup(GpioBaseT Base,uint8_t Val,uint8_t Len,uint8_t FirstBit);`	Global Function
`GpioRetT GpioWritePin(GpioBaseT Base,GpioPinT Pin,GpioStateT State);`	Global Function
`uint8_t  GpioReadPort(GpioBaseT Base);`	Global Function
`uint8_t GpioReadGroup(GpioBaseT Base, uint8_t Len, uint8_t FirstBit);`	Global Function
`uint8_t  GpioReadPin(GpioBaseT Base,GpioPinT Pin);`	Global Function

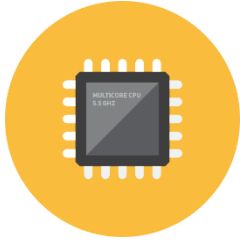
System Modules and Functions: GPIO



`GpioRetT GpioInitPort(GpioBaseT Base,GpioStateT State);`	
Function initializes the hardware for the system Ports	
`Base`	Defines which gpio port to initialize
`State`	State of the Targeted port
Return	State of the initialization process

`GpioRetT GpioInitPin(GpioBaseT Base,GpioPinT Pin,GpioStateT State);`	
Function initializes the hardware for the system Pins	
`Base`	Defines which gpio port to initialize
`Pin`	Defines which pin on that port to set its state
`State`	State of the Targeted pin to be set
Return	State of the initialization process

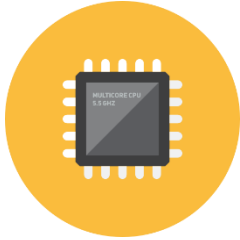
System Modules and Functions: GPIO



`GpioRetT GpioWritePort(GpioBaseT Base,uint8_t Val);`	
Function initializes the hardware for the system Pins and Ports	
`Base`	Defines which gpio port to write to
`Val`	Value to be written on that port
Return	State of the writing process

`GpioRetT GpioWriteGroup(GpioBaseT Base,uint8_t Val,uint8_t Len,uint8_t FirstBit);`	
Function initializes the hardware for the system Pins	
`Base`	Defines which gpio port to write to
`Val`	Defines the value to be writen
`Len`	Defines the length of the targeted group
`FirstBit`	defines the starting bit of the group needed to be writen
Return	State of the writing process

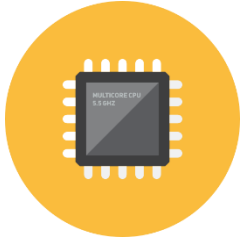
System Modules and Functions: GPIO



`GpioRetT GpioWritePin(GpioBaseT Base,GpioPinT Pin,GpioStateT State);`	
Function initializes the hardware for the system Pins and Ports	
`Base`	Defines which gpio port to write to
`Pin`	The targeted pin to be writen
`State`	State of the Targeted pin to be set
Return	State of the initialization process

`uint8_t GpioReadPort(GpioBaseT Base);`	
Function initializes the hardware for the system Pins	
`Base`	Defines which gpio port to read from
Return	State of the Targeted port

System Modules and Functions: GPIO



`uint8_t GpioReadGroup(GpioBaseT Base, uint8_t Len, uint8_t FirstBit);`	
Function initializes the hardware for the system Pins and Ports	
`Base`	Defines which gpio port to read from
`Len`	Defines the length of the targeted group
`FirstBit`	defines the starting bit of the group needed to be read
Return	State of the Targeted group in a byte to be extracted later

`uint8_t  GpioReadPin(GpioBaseT Base,GpioPinT Pin);`	
Function reads the hardware state of the targeted pin	
`Base`	Defines which gpio port to read from
`Pin`	Defines which pin on that port to get its state
Return	State of the Targeted Pin

Done

- Mahmoud Saad Abd-Elhares
- Mahmoud.S.AbdElhares@gmail.com
- Linkedin.com\in\MahmoudSaad96
- Github.com\ZeroX96