University of Texas at El Paso
Electrical and Computer Engineering Department

EE 4178/5190 – Laboratory for Microprocessors II

LAB 2

Microwave – Interrupt System

Objective:

- Given the program template shown in listing 1, edit it so that the code so that you replicate the functions of a microwave.
- The code is expected to have a total of 6 buttons which will simulate the functions of the microwave.
  - The first and second button should control a counter that goes up or down.
  - The next two buttons should act as a start or reset counter.
  - The last 2 buttons are your interrupts which simulates if the door open or closes. Since the Lab is a microwave, the timer should stop in the event that the door open button is been pressed the timer should stop and cannot start again until the close door button has been pressed
- Print the counter time as you add or subtract the total amount of time the microwave will run. Then when the timer start button is pressed the counter should display in the screen as it goes down every 500 ms.

Bonus:

- For EE4178 is just a bonus and for EE5190 is mandatory
  - Modify the code by using the **gpio_config_t** to create an output LED which will display when the door is open or close.
- Bonus for EE5190
  - Create another interrupt button which do the pause option in the microwave.

Pre-Lab:

- What is an interrupt?
- What is the ISR function that enables the pin to be an interrupt?

By: Erick Baca and Jesus Minjares

C helpful functions

For this lab, there are couple additional functions from **ESPRESSIF** that are important for using inputs. In the previous lab we use the function **gpio_set_direction(gpio_num_t gpio_num, gpio_mode_t mode)** in order to set our inputs and outputs. In this lab, we will use the **gpio_config_t** which will help us setup the interrupt function.

- **typedef struct** {
  uint64_t pin_bit_mask;   /*!< GPIO pin: set with bit mask, each bit maps to a GPIO */
  gpio_mode_t mode;   /*!< GPIO mode: set input/output mode*/
  gpio_pullup_t pull_up_en;   /*!< GPIO pull-up*/
  gpio_pulldown_t pull_down_en;   /*!< GPIO pull-down*/
  gpio_int_type_t intr_type;   /*!< GPIO interrupt type*/
  } **gpio_config_t;**

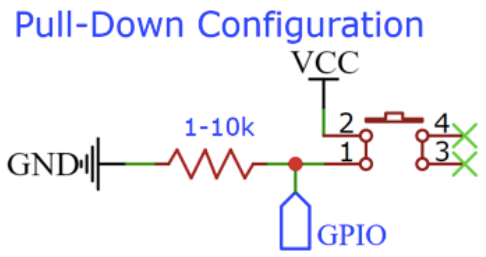
To fill in the structure **gpio_config_t**, each variable is reference to another structure. Inside the uint64_t pin_bin_mask it need a to bit shift the pin number that converts from an int to an unsigned long long by using **1ULL << BUTTON**. After knowing the next variables have different function structures. For the **gpio_mode_t mode**, use the listing below to set what your button will do as well if you want to set it up as an output. If you are using the output mode, just remember that in order o call an output you need to use GPIO.out = BIT#.

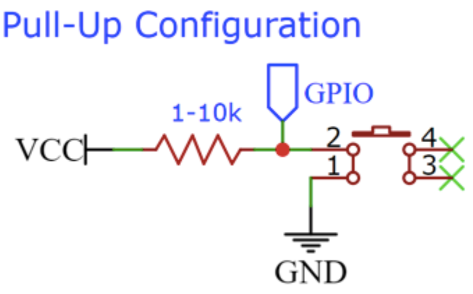
- typedef enum {
  GPIO_MODE_DISABLE = GPIO_MODE_DEF_DISABLE,
  /*!< GPIO mode : disable input and output   */
  GPIO_MODE_INPUT = GPIO_MODE_DEF_INPUT,
  /*!< GPIO mode : input only   */
  GPIO_MODE_OUTPUT = GPIO_MODE_DEF_OUTPUT,
  /*!< GPIO mode : output only mode   */
  GPIO_MODE_OUTPUT_OD = ((GPIO_MODE_DEF_OUTPUT) |(GPIO_MODE_DEF_OD)),   /*!< GPIO mode : output only with open-drain mode   */
  GPIO_MODE_INPUT_OUTPUT_OD = ((GPIO_MODE_DEF_INPUT) | (GPIO_MODE_DEF_OUTPUT) | (GPIO_MODE_DEF_OD)), /*!< GPIO mode : output and input with open-drain mode*/
  GPIO_MODE_INPUT_OUTPUT = ((GPIO_MODE_DEF_INPUT)|(GPIO_MODE_DEF_OUTPUT)),   /*!< GPIO mode : output and input mode   */
  } **gpio_mode_t;**

Then in the next two points of the **gpio_config_t** which are **gpio_pulldown_t** or **gpio_pullup_t** is where the button is enable as a pulldown or as a pullup. Use the picture below to guide yourself into how tosetup your hardware for the pullup or pulldown. Finally, the last configuration **gpio_int_type_t**, it decides what type of edge will the button read, and the configuration is shown below.

- **typedef enum** {

```
    GPIO_INTR_DISABLE = 0,     /*!< Disable GPIO interrupt                          */
    GPIO_INTR_POSEDGE = 1,     /*!< GPIO interrupt type : rising edge               */
    GPIO_INTR_NEGEDGE = 2,     /*!< GPIO interrupt type : falling edge              */
    GPIO_INTR_ANYEDGE = 3,     /*!< GPIO interrupt: both rising and falling edge */
    GPIO_INTR_LOW_LEVEL = 4,   /*!< GPIO interrupt type: input low level trigger */
   GPIO_INTR_HIGH_LEVEL = 5,  /*!< GPIO interrupt type : input high level trigger */
   GPIO_INTR_MAX,
} gpio_int_type_t;
```





Next, you will need to set the interrupt flag into 0 which uses the function: **gpio_install_isr_service(int intr_alloc_flags)**

- **esp_err_t gpio_install_isr_service(int intr_alloc_flags)**

Lastly, there is a function that allows the interrupt button to enter the static **void IRAM_ATTR gpio_isr_handler(void* arg)**. In order to enter this function you need to declare using the function **gpio_isr_handler_add(gpio_num_t gpio_num, gpio_isr_t isr_handler, void *args).**

- **esp_err_t gpio_isr_handler_add(gpio_num_t gpio_num, gpio_isr_t isr_handler, void *args)**

By: Erick Baca and Jesus Minjares