Sensor Motor

Developer Manual

Jenna-Luz Pura

Data Structures and Handlers

Enumerated Types

display_pinout

display_config

display_setting

system_code

Semaphores and Queues

SemaphoreHandle_t *display_semaphore*	*binary semaphore*
QueueHandle_t *display_queue*	type *system_code*
QueueHandle_t *left_display_queue*	type *int*
QueueHandle_t *right_display_queue*	type *int*
QueueHandle_t *sensor_base_queue*	type *display_setting*
QueueHandle_t *motor_queue*	type *system_code*
QueueHandle_t *motor_direction_queue*	type *system_code*
QueueHandle_t *temperature_queue*	type *int*
QueueHandle_t *humidity_queue*	type *int*
QueueHandle_t *pixel_queue*	type *system_code*

Tasks

button1_handler	*priority 4*
button2_handler	*priority 4*
button3_handler	*priority 4*
display_handler	*priority 3*
left_display_handler	*priority 3*
right_display_handler	*priority 3*
motor_handler	*priority 3*
sensor_handler	*priority 3*
pixel_handler	*priority 3*

Task and Driver Functionality

Buttons

An interrupt handler waits for all of the three button events and calls a callback function when an event occurs. There is one semaphore and one task for each button. The callback function gives the appropriate semaphore and the particular button task is now unblocked.

Once a button task is unblocked, it debounces each event in a two second interval by delaying for 200 milliseconds. It stores the number of bounces in a variable and uses a switch statement to determine what to do.

button1_handler

> ***1 Push***: sends the *system_code* MOTOR_TEMPERATURE to the *motor_queue*.
>
> ***2 Pushes:*** sends the *system_code* MOTOR_HUMIDITY to the *motor_queue*.
>
> ***3 Pushes:*** if the base_code is currently equal to *display_setting* SET_DECIMAL, SET_HUMIDITY is sent to the *sensor_base_queue*; otherwise, SET_DECIMAL is sent to the *sensor_base_queue*.
>
> ***4 Pushes:*** calls API function *system_error()* with parameter *system_code* ERROR_EMERGENCY_STOP. *system_code* MOTOR_HALT and ERROR_EMERGENCY_STOP is overwritten on the *motor_queue* and *pixel_queue*, respectively.
>
> ***Default:*** calls API function *system_error(*ERROR_UNKNOWN_INPUT*)*.

button2_handler

> ***1 Push:*** sends *system_code* MOTOR_CLOCKWISE to the *motor_queue*.
>
> ***2 Pushes:*** sends *system_code* MOTOR_COUNTERCLOCKWISE to the *motor_queue*.
>
> ***3 Pushes:*** sends *system_code* MOTOR_ALTERNATE to the *motor_queue*.
>
> ***Default:*** calls API function *system_error(*ERROR_UNKNOWN_INPUT*)*.

button3_handler

> ***1 Push:*** sends *display_code* DISPLAY_TEMPERATURE to the *display_queue* and the *pixel_queue*.

2 Pushes: sends *display_code* DISPLAY_HUMIDITY to the *display_queue* and the *pixel_queue*.

3 Pushes: sends *display_code* MOTOR_STATUS to the *display_queue* and the *pixel_queue*.

Default: calls API function *system_error(*ERROR_UNKNOWN_INPUT*)*.

In each case, it checks if the *display_queue* is full. If it is, it calls API function *system_error(*ERROR_OVERFLOW*)*.

HDC1080

sensor_handler

This task uses the API function *sensor_read_tmp()* to get the current temperature and *sensor_read_hmd()* to get the current relative humidity. The temperature is sent to the *temperature_queue* and the relative humidity is sent to the *humidity_queue*.

Stepper Motor

motor_handler

This task receives from *motor_queue* and stores the value in *motor_code*. It switches functionality based on this variable.

MOTOR_CLOCKWISE, MOTOR_COUNTERCLOCKWISE, or MOTOR_HALT is overwritten to *motor_direction_queue* when applicable.

MOTOR_RESET

Sets the *motor_code* equal to *motor_status*, restoring the previous functionality. Used when turning off the emergency stop mechanism.

MOTOR_CLOCKWISE & MOTOR_COUNTERCLOCKWISE

Calls API functions *motor_clockwise()* and *motor_counterclockwise* respectively.

MOTOR_ALTERNATE

Calls API function *motor_clockwise()* for a certain amount of steps, then *motor_counterclockwise()* for the same amount of steps.

MOTOR_TEMPERATURE & MOTOR_HUMIDITY

Receives from *temperature_queue* and *humidity_queue* respectively. If the previously stored value is equal to the current value, the API function *motor_halt()* is called. If the current value is greater, API function *motor_increment()* is called; otherwise, *motor_decrement()* is called.

MOTOR_HALT

Calls API function *motor_halt()*.

If there is any unknown input, API function *system_error(*ERROR_UNKNOWN_INPUT*)* is called.

7 Segment Display

display_handler

This task receives from *display_queue* and stores the value in *display_code*.

DISPLAY_TEMPERATURE & DISPLAY_HUMIDITY

Receives from the *sensor_base_queue* and sends *display_config* DISPLAY_H or DISPLAY_D to the *left_display_queue* and *right_display_queue* when necessary and delays for three seconds.

Receives from the *temperature_queue* and *humidity_queue* respectively. Calculates the left and right digits based on *base_code* and sends to the *left_display_queue* and *right_display_queue* respectively.

MOTOR STATUS

Receives from *motor_direction_queue* and stores the value in *motor_dir*.

MOTOR_CLOCKWISE: *display_config* DISPLAY_C is sent to the *left_display_queue* and *right_display_queue*.

MOTOR_COUNTERCLOCKWISE: *display_config* DISPLAY_CC is sent to the *left_display_queue* and *right_display_queue*.

MOTOR_HALT: *display_config* DISPLAY_O is sent to the *left_display_queue* and *right_display_queue*.

ERROR_UNKNOWN_INPUT

display_config DISPLAY_X is sent to the *left_display_queue* and *right_display_queue*. Then, it is delayed for five seconds.

If the emergency stop is currently active, *display_code* is set to ERROR_EMERGENCY_STOP.

ERROR_OVERFLOW

display_config DISPLAY_O is sent to *left_display_queue* and *display_config* DIPLAY_F is sent to *right_display_queue*. Then, it is delayed for five seconds.

ERROR_EMERGENCY_STOP

display_config DISPLAY_E is sent to the *left_display_queue* and *right_display_queue*. *display_code* is set to DISPLAY_REPEAT.

If the emergency stop is currently active, *system_code* MOTOR_RESET is sent to the *motor_queue*. d*isplay_mode* is set to *display_status* and overwritten to the *pixel_queue*, restoring the previous functionality.

DISPLAY_REPEAT

This allows for the most recently sent value sent to the *left_display_queue* and *right_display_queue* to repeatedly show on the display.

left_display_handler & right_display_handler

Both tasks block on the *display_semaphore* to synchronize the left and right displays. Once unblocked, these tasks receive from the *left_display_queue* and *right_display_queue*, respectively, and stores the value in *pin_config*. API function *display_value(*SET_LEFT, pin_config*)* and *display_value(*SET_RIGHT, pin_config*)* is called, respectively. The *display_semaphore* is then given.

Neopixels

Each neopixel represents the state of a different peripheral device.

Neopixel 1 (top-most): reflects temperature

Bright red when temperature is increasing
Dim red when temperature is decreasing

Neopixel 2: reflects relative humidity

Bright yellow when relative humidity is increasing

Dim yellow when relative humidity is decreasing

Neopixel 3: reflects the status of the 7 Segment Display

Red when displaying temperature

Yellow when displaying relative humidity

Purple when displaying motor status

Neopixel 4 (bottom-most): reflects the status of the Stepper Motor

Green when moving clockwise

Blue when moving counterclockwise

Orange when not moving

pixel_handler

This task initializes the neopixels and sets their color values according to the default settings of the peripheral devices.

On every iteration, it peeks at the *temperature_queue* and *humidity_queue* to compare the current corresponding sensor value to a previously recorded value. If the new value is greater, then the appropriate pixel is brightened; if it is less, the appropriate pixel is dimmed.

It then peeks at the *motor_queue* and stores it in *system_code motor_code*, then switches based on that value. The third pixel is changed accordingly.

It also receives from the *pixel_queue* and stores it in *system_code pixel_code*. Based on the value of *pixel_code*, the fourth pixel will change accordingly.

In the event that ERROR_EMERGENCY_STOP is sent over the *pixel_queue*, the four pixels will perform a rainbow effect. Once the emergency stop mechanism is deactivated, the four pixels will proceed to perform their individual functionality.

System Errors

API function *system_error()* is called with a *system_code* parameter. When ERROR_UNKNOWN_INPUT, ERROR_OVERFLOW, or ERROR_EMERGENCY_STOP is sent as a parameter, their respective *system_code* values are sent to the *display_queue*.

If ERROR_OVERFLOW is sent as a parameter, the *display_queue* is cleared before its *system_code* value is sent to the *display_queue*.