Task list

Short-term

- [ ] Decide on whether to join SER and BT into one module, keeping in mind that:
    - One is made to work with a connection-oriented protocol and the other isn't.
    - Exception handling is different in both protocols.
- [ ] Decide if **GPIO** modules should return `E_MUTED` when blocked by a mutex or just wait for the mutex to be free.
- [ ] Decide if a get_last_error function should be implemented.

Long-term

- [ ] Idealize all the functionality for:
    - [ ] CFL
        - [ ] Global
        - [ ] HMI
        - [ ] DC
        - [ ] COM
    - [ ] HLHAL
        - [ ] Global
        - [ ] CRE
        - [x] GPIO
        - [ ] BT
        - [ ] SER

Index

CFL: *Core Functionality Layer*

Global

HMI: *Human-Machine Interface* [EMPTY]

DC: *Digital Control* [EMPTY]

COM: *Communications* [EMPTY]

HLHAL: *High-Level Hardware Abstraction Layer*

This layer of software is the only direct interface that the program has with the hardware and so it is meant to facilitate development in the layers that go above it and make it easier to port the code into a physical platform, since these are ideally the only places where it would need to be changed.

Global

Structures

> `HLHAL_State`

Structure of flags that make up the state of the High-Level Hardware Abstraction Layer peripherals.

- `created: boolean`
  Flag that signals whether or not the entity has been created.
- `configured: boolean`
  Flag that signals whether or not the entity has been configured.
- `running: boolean`
  Flag that signals whether or not the entity is executing its task.

CRE: *Communications Redundancy Engine* [EMPTY]

Enumerations

Structures

Attributes: Static

Attributes: Instance

Methods

Events

Typical Use Case

GPIO: *General Purpose Input/Output*

Enumerations

> `GPIO_Ports`

Hardware port enumeration. This exists mainly for use by the GPIO abstraction module.

```
PORT_A, PORT_B, PORT_C, PORT_D, PORT_E, PORT_F
```

> `GPIO_Pins`

Hardware pin enumeration. This, like **`GPIO_Ports`**, exists mainly for use by the `GPIO` abstraction module.

```
PIN_0, PIN_1, PIN_2, PIN_3, PIN_4, PIN_5, PIN_6, PIN_7,
PIN_8, PIN_9, PIN_10, PIN_11, PIN_12, PIN_13, PIN_14, PIN_15
```

> **`GPIO_UpdateModes`**

Conversion update mode, primarily for use by the `GPIO` abstraction module.

```
CONTINUOUS, SINGLE_SHOT
```

- `CONTINUOUS_CIRCULAR`: Signal for conversion is given once and all future conversions happen repeatedly, without user intervention, separated by a previously established sampling period;
- `CONTINUOUS_NON_CIRCULAR`: Behaves as in `CONTINUOUS_CIRCULAR` mode, but only makes enough conversions to fill `io_buffer`;
- `SINGLE_SHOT`: Only one conversion happens at the given signal.

For each mode the user should be able to prompt the module to keep the converted values in a list and generate events at the end of each conversion to be able to react promptly upon it.

> **`GPIO_PulseCountingPolarity`**

Defines the polarity for the `GPIO_OperationModes` in `INPUT_PULSE_COUNTING` mode.

```
RISING, FALLING, RISING_AND_FALLING
```

- `RISING`: Counts rising edge pulses;
- `FALLING`: Counts falling edge pulses;
- `RISING_AND_FALLING`: Counts rising and falling edge pulses.

> **`GPIO_OperationModes`**

Available operation modes for the `GPIO` module, based on direction and high-level functionality.

```
INPUT_MODE_PULSE_COUNTING, INPUT_MODE_ANALOG, INPUT_MODE_DIGITAL,
OUTPUT_MODE_PWM, OUTPUT_MODE_ANALOG, OUTPUT_MODE_DIGITAL,
```

- `INPUT_MODE_PULSE_COUNTING`: Pulse counting input;
- `INPUT_MODE_ANALOG`: Analog to digital conversion input;
- `INPUT_MODE_DIGITAL`: Digital (1 or 0) input;
- `OUTPUT_MODE_PWM`: PWM output;
- `OUTPUT_MODE_ANALOG`: Digital to analog conversion;
- `OUTPUT_MODE_DIGITAL`: Digital (1 or 0) output.

> **`GPIO_State`**

Structure for keeping all aspects related to the current state of a GPIO entity.

- `created: boolean`
  Flag that signals whether or not the entity has been created.

- `configured: boolean`
  Flag that signals whether or not the entity has been configured.

- `running: boolean`
  Flag that signals whether or not the entity is executing its task.

> `GPIO_Errors`

Error values to be set when the user requests an action that cannot be performed.

```
E_OK, E_MUTED, E_NOT_CONFIGURED, E_RUNNING, E_NOT_RUNNING,
E_WRONG_MODE, E_CNV_NOT_AVAILABLE, E_INVALID_INTERFACE_CONFIG,
E_BAD_PARAMETER
```

- `E_OK`: Operation was successful
- `E_MUTED`: The entity cannot tend to the user's request because the module is mutex-locking a static attribute.
- `E_NOT_CONFIGURED`: The entity is not yet configured to perform any task;
- `E_RUNNING`: The entity is still performing a task and thus cannot be reconfigured until it stops;
- `E_NOT_RUNNING`: The entity is not executing any task at the moment;
- `E_WRONG_MODE`: The user called for functionality not destined to run in the chosen operation mode;
- `E_CNV_NOT_AVAILABLE`: A certain conversion has not yet been made and thus its value is not available for retrieval;
- `E_INVALID_INTERFACE_CONFIG`: An invalid pin configuration has been chosen.
- `E_BAD_PARAMETER`: A parameter with an invalid value has been passed onto a method;
- `E_OUT_OF_BOUNDS`: The user request that an operation be performed to the input_output buffer in a position that is out of bounds.

Structures

> `GPIO_Config`

Structure for declaring and keeping all aspects related to the configuration of a GPIO entity.

Attributes:

- `port: GPIO_Ports`
  Selected port for this entity.

- `pin: GPIO_Pins`
  Selected pin for this entity.

- `mode: GPIO_OperationModes`
  Chosen operation mode for this entity.

- `update_mode: GPIO_UpdateModes`
  Chosen Conversion Mode for this entity.

- `pulse_counting_polarity: GPIO_PulseCountingPolarity`
  Relevant only for `GPIO_OperationModes.INPUT_MODE_PULSE_COUNTING` mode. Defines whether it counts rising edge pulses, falling edge pulses or both.

- `conversion_complete_callback: (uint32) -> void` A function to be called when a conversion or reading is completed. Should be left NULL if it is to be ignored.

- `sampling_period: uint32`
  Period elapsed between each conversion in Continuous Conversion Mode.

Attributes: Static

> `gpio_objects: GPIO*[]`

List of pointers to all currently existing objects of this type.

Objects are added to this list when created and removed when deleted. This exists to facilitate resource management between said objects.

> `gpio_mutex: OSMutex`

OS-specific mutex object that is meant to lock read or write access to sensitive module-wide static attributes.

Attributes: Instance

> `config: GPIO_Config`

Structure holding the current configuration values for its entity.

> `state: HLHAL_State`

Structure holding the current state values for its entity.

> `io_buffer: CircularBuffer`

Circular buffer that allows for storage and retrieval of past conversion values.

It is meant to be written to by the Sensor module exclusively and automatically upon finishing a conversion The retrieval of data from it should also be intermediated by this module.

It is most useful in Continuous Conversion Mode, but allowed for use in Single Shot Conversion Mode as well.

> `last_error: GPIO_Errors`

Error code for the last occurring error when accessing the entity's methods.

Methods

> `create() -> boolean`

- ***brief***

  Initialization various critical parameters to the *reset* state and addition of the current object to the `GPIO.gpio_objects` list.

- ***return***

  - `TRUE` and `last_error = E_OK` if the creation was successful;
  - `FALSE` and `last_error = E_RUNNING` if the entity is still performing a task and thus cannot be reconfigured until it stops. The solution is to call `stop()` on this entity to stop its execution, `delete()` to delete it and call `create()` again to try creating again;

> **`configure(`*`config: GPIO_Config*, io_buffer: CircularBuffer`*`) -> boolean`**

- ***brief***

  Entity configuration method. Every aspect of the configuration should be handled at once, from here.

- ***parameters***

  - **`config`**

    `GPIO_Config` structure that contains instructions on how the entity should be configured to operate.

  - **`io_buffer`**

    `CircularBuffer` that will hold the history of input values for the object. It must be configured in order to operate in `GPIO_UpdateModes.CONTINUOUS` mode but it can be left `NULL` in `GPIO_UpdateModes.SINGLE_SHOT`.

- ***return***

  - `TRUE` and `last_error = E_OK` if the configuration was successful;
  - `FALSE` and
    - `last_error = E_INVALID_INTERFACE_CONFIG` if the chosen port and pin combination is completely unavailable or cannot be assigned that specific task;
    - `last_error = E_MUTED` if the module is mutex-locking a static attribute that needs to be accessed or modified during configuration. It doesn't signal an error so to speak, but a brief restriction;
    - `last_error = E_RUNNING` if the entity is already performing a specific task. The solution is to call `stop()` on this entity to stop its execution and call `configure()` again to try configuring again;
    - `last_error = E_BAD_PARAMETER` if either `config` or `io_buffer` have a `NULL` value.

> `start() -> boolean`

- ***brief***

  Start executing the configured task.

- ***return***

  - `TRUE` and `last_error = E_OK` if the configuration was successful;
  - `FALSE` and `last_error = E_NOT_CONFIGURED` if the entity is not yet configured to perform any task;

> `fetch_value(index: uint32) -> uint32`

- ***brief***

  Fetch the conversion value `index` values behind the last one. This is executed whether or not `index` is less than or equal to the size of the buffer, but if it exceeds the size of the buffer, a `GPIO_Errors.E_OUT_OF_BOUNDS` error is placed in `last_error` so this should always be checked for when in doubt.

  Available for all `GPIO_OperationModes`.

- ***return***

  The conversion value `index` values *behind* the last one and:

  - `last_error = E_OK` if `index` is within bounds.
  - `last_error = E_OUT_OF_BOUNDS` if `index` is out of bounds. This should always be checked for when in doubt, as it signals an error.

> `push_value(index: uint32, value: uint32) -> boolean`

- ***brief***

  Push the conversion value `index` values *ahead* of the last one.

  It is important to note that in case `value` exceeds the maximum allowed value, it will be truncated to avoid unpredictable complications down the road.

  Only available for `GPIO_OperationModes` output modes, so using it while running in one of the input modes will generate an error.

- ***return***

  - `TRUE` and `last_error = E_OK` if is within bounds and the operation was successful;
  - `FALSE` and
    - `last_error = E_WRONG_MODE` if the entity is running in one of the `GPIO_OperationModes` input modes;
    - `last_error = E_OUT_OF_BOUNDS` if `index` is out of bounds.

> `stop() -> boolean`

- ***brief***

  Stop executing the running task.

- ***return***

  - `TRUE` and `last_error = E_OK` if the configuration was successful;
  - `FALSE` and `last_error = E_NOT_RUNNING` if the entity was not executing any task in the first place. This isn't an error as much as it is a warning.

> **`delete() -> boolean`**

- ***brief***

  Remove object from `gpio_objects`.

- ***return***

  - `TRUE` and `last_error = E_OK` if the deletion was successful;
  - `FALSE` and
    - `last_error = E_RUNNING` if the entity is performing a specific task. The solution is to call `stop()` on this entity to stop its execution and calling `delete()` again to try deleting again;
    - `last_error = E_MUTED` if the module is mutex-locking a static attribute that needs to be accessed or modified during deletion. It doesn't signal an error so to speak, but a brief restriction.

Events

> **`event_conversion_complete(value: uint32)`**

This event is associated with the `conversion_complete_callback` callback. It happens when an entity completes an output or input action conversion, may that be an ADC or DAC conversion or a simple pin toggle.

Note that `conversion_complete_callback` will only be called when it is configured.

Typical Use Case

```
CircularBuffer buffer = ...
GPIO rpm_sensor;
// Configure for using pin 0 from port A, in pulse counting mode on the
// rising edge, with a sampling period of 1 ms and no conversion complete
// callback
GPIO_Config rpm_sensor_config = {
    .port = PORT_A,
    .pin = PIN_0,
    .mode = INPUT_MODE_PULSE_COUNTING,
    .update_mode = CONTINUOUS_CIRCULAR,
    .pulse_counting_polarity = RISING_EDGE,
    .sampling_period = 1000,
    .conversion_complete_callback = NULL,
}
```

```
// Create the sensor
gpio_create(&rpm_sensor);
// Configure task parameters
gpio_configure(&rpm_sensor, &rpm_sensor_config, &buffer);
// Set running
gpio_start(&rpm_sensor);

...

// fetch 5th value from the last one back
gpio_fetch(&rpm_sensor, 5);

// Configure the sensor again without stopping it. This is not permitted.
if (gpio_configure(&rpm_sensor, &rpm_sensor_config) != E_OK) {
    // Identify the error and treat it
    if(rpm_sensor.last_error == E_RUNNING) {

        ...
    }
    ...
}
// Stop the sensor
gpio_stop(&rpm_sensor);
// Start the sensor again. This is permitted.
gpio_start(&rpm_sensor);
// Stop the sensor again. This is permitted.
gpio_stop(&rpm_sensor);
// Delete the sensor
gpio_delete(&rpm_sensor);
```

BT: ***Bluetooth*** [EMPTY]

> **NOTE**: *The design of the Bluetooth module is being briefly postponed due to uncertainty on whether the program will be running on a simulation of a microcontroller or on a normal computer program, as this is relevant information for the case.*

Enumerations

Structures

Attributes: Static

Attributes: Instance

Methods

Events

Typical Use Case

SER: ***Serial*** [EMPTY]

Enumerations

Structures

Attributes: Static

Attributes: Instance

Methods

Events

Typical Use Case