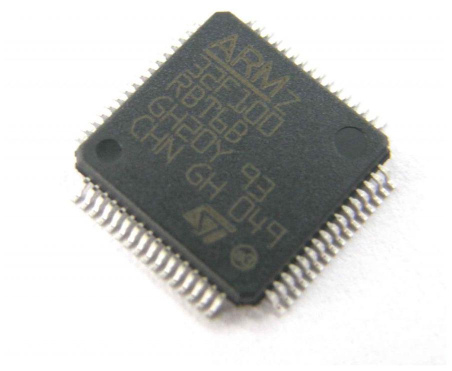
MCU pin configuration, GPIOs and a word on software architecture

january 10, 2013 by rtos.be leave a comment

Rating: 0.0/**10** (0 votes cast)

Basic peripheral setup: pin configuration

All MCUs have pins. They might come in different package types such as QFP (Quad Flat Package – Figure left) or BGA (Ball Grid Array – Figure right). But for us, embedded software engineers, pins or balls, we don't really mind: MCUs have pins and we need to configure them.



A MCU pin has:

- a *main function* which is typically GPIO, and
- *alternate function(s)* which can be anything e.g. Pins for SPI, CAN, I2C, ADC etc.

After a processor reset pins are normally configured as gpio input which is a safe state.

Each MCU vendor (ST, NXP, FreeScale…) implements the pin configuration block in a (slightly) different way. Given our goal of Simple Platform Abstraction, this is our try to define a generic 'one-fits-all' api:

```
enum pin_mode {
    /* mcu-family specific */
};
```

```
/*! Initialize the pin configuration module with a lock. */
BOOLEAN pinconfig_init(LOCK_ObjectHandle lock);

/*! Set the mode of a pin. */
BOOLEAN pin_set_mode(PIN pin, enum pin_mode mode);

/*! Get the mode of a pin. */
enum pin_mode pin_get_mode(PIN pin);

/*! Is this a valid pin (number)? */
BOOLEAN pin_is_valid(PIN pin);
```

PIN is an abstract representation of a pin and it is up to the developer to define it in such a way that it easily maps on the underlying hardware.

The enum is MCU-family specific so you probably want to move them to an MCU-family specific header file.

In our energy harvester project (STM32F100 mcu) we use the following enum definition:

```
enum pin_mode {
    pin_mode_error=-1,
    \
    input_analog,
    input_floating,
    input_pullupdown,
    \
    output_gpio_pushpull_max_10MHz,
    output_gpio_opendrain_max_10MHz,
    output_alternate_pushpull_max_10MHz,
    output_alternate_opendrain_max_10MHz,
    \
    output_gpio_pushpull_max_2MHz,
    output_gpio_opendrain_max_2MHz,
    output_alternate_pushpull_max_2MHz,
    output_alternate_opendrain_max_2MHz,
    \
    output_gpio_pushpull_max_50MHz,
    output_gpio_opendrain_max_50MHz,
    output_alternate_pushpull_max_50MHz,
    output_alternate_opendrain_max_50MHz
};
```

In order to configure the pins of an NXP LPC2000 MCU, one might use:

```
enum pin_mode {
    pin_mode_error=-1,
    main,
    alternate1,
    alternate2,
    alternate3
};
```

GPIO

Introduction

Being able to control GPIOs (General Purpose I/O's) is probably the most basic feature of the low-level embedded software engineer toolbox. GPIOs are discrete (OFF or ON) I/O's and can be used to:

control LEDs,
perform a chip select (on buses such as SPI),
read a switch input (button),
latch ICs,
debug software,
and so on and so on.

GPIO's are grouped together in ports e.g. one port can have 32 GPIOs in a specific platform.

Api

The functionality of a GPIO is quite basic and is shown here by our generic api:

```
/*! Initialize the gpio module with a lock. */
BOOLEAN gpio_init(LOCK_ObjectHandle lock);

/*! Configure pin as default (pullpush) output gpio. */
BOOLEAN gpio_set_dir_output(PIN pin);

/* Configure pin as default (pullupdown) input gpio. */
BOOLEAN gpio_set_dir_input(PIN pin);

/*! Is this pin configured as GPIO input? */
BOOLEAN gpio_is_input(PIN pin);

/*! Is this pin configured as GPIO output? */
BOOLEAN gpio_is_output(PIN pin);

/*! Set gpio high. */
```

```
BOOLEAN gpio_set_high(PIN pin);

/*! Set gpio low. */
BOOLEAN gpio_set_low(PIN pin);

/*! Toggle gpio. */
BOOLEAN gpio_toggle(PIN pin);

/*! Is this gpio (input OR output) high? */
BOOLEAN gpio_is_high(PIN pin);

/*! Is this gpio (input OR output) low? */
BOOLEAN gpio_is_low(PIN pin);
```

In the energy harvesting project we use GPIOs to control LEDs, relay's, mosfets etc.

A word on architecture

Controlling a GPIO is never a goal on itself, it is always a means of achieving something else. For that reason, it is important to note that you don't want to control GPIOs in a direct way in your application code, rather, you want to build higher-level abstractions that map on your application domain. For example, compare next two code snippets:

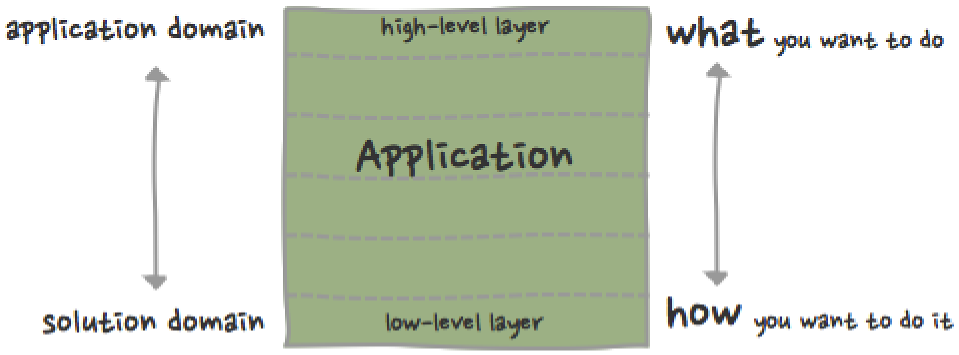
Code snippet 1 (you're lucky to have a reasonable comment):

```
/* pin3 controls the 'dump load' relay via a mosfet */
gpio_set_low(PIN3);
```

Code snippet 2:

```
divert_wind_energy_to_dump_load_output();
```

The high-level application code shouldn't talk about items from the solution space (gpio's, relay's…) but about concepts from the application domain. Code snippet 1 is wrong because GPIOs are not a concept from the high-level application domain. See Figure.



Share

Rating: 0.0/**10** (0 votes cast)

Related Posts

- Coming Soon
- The embedded hierarchy — Part 1
- Reverse engineering the bug of my Opel Zafira’s volume knob (with movie!)
- Debounced Buttons
- Driving LEDs by GPIO: finally resolved!

Share / Save …

filed under: energy harvester project, interfaces, mcu peripherals, software design and architecture tagged with: gpio, pin configuration

About rtos.be

rtos.be is a joined initiative from Gert Boddaert and Pieter Beyens.

More info: about us, our mission, contact us.