PINS library

Interface to hardware specific IO.

16.9.2019/Pekka Lehtikoski

Table of Contents

1. Introduction

190916, updated 16.9.2019/pekka

The “pins” library separates IO application from hardware level IO functions, so the same IO application can be run on different micro-controllers or as Linux/Windows simulation. This library is optional and not integal part of iocom library: Any other IO library can be used instead. Typically IO hardware accessed tough the “pins” interface are GPIO, ADC/DAC, PWM, timers, etc.

PINS LIBRARY IS IN DRAFTING/PLANNING STAGE, NOT USEFUL.

1.1 About “pins” library

Function of the "pins" library is to:

- Provide an application with hardware independent access to GPIO, analogs, PWM, timers, etc.
- Provide subs to call forward hardware dependent IO library, like pigpio on Raspberry PI, HAL on STM32, IO simulation on PC and so forth.
- Define hardware pin configuration in such way that it needs to be written only once, and is then available from C code, memory maps and in documentation.

Some taughts

- Hardware specific IO headers are not included in application (except when there is need to bypass "pins").
- Hardware addresses, etc. need to be mapped to application at some point. This could be simply written as one C file. But we write it as JSON and then generate the C file automatically by Python script. This enable using the PIN information to automate documentation, IOCOM memory mapping, etc.

1.2 JSON configuration

190917, updated 17.9.2019/pekka

We specify hardware configuration as JSON and then generate the C file automatically by Python. Simple case: WORK_LIGHT binary output turns on the lights. The example IO application is called “jane” and example hardware it runs on is named “carol” (ESP32 development board).

- The WORK_LIGHT output is mapped to specific IO pin by gpio pin number, or bank number and pin number within bank. We may also have some attributes for IO pin. For example LIGHT_SWITCH input could have sensing speed to filter out very short pulses, etc. For analog input, PWMs, etc, there can be a lot of parameters.
- So one pin (or more generally IO item since we can include timers etc), is configured by set of attribytes, name, type, address and various parameters. For example a digital output WORK_LIGHT is at address 3 of bank 1, or digital input LIGHT_SWITCH is address 2 of bank 4 using filtering speed 3.
- An IO pin is “input”, “output”, “analoginput”...
- Pin attributes are “bank”, “addr”, “speed”...
- Attribute values excluding group are always integers.
- Some parameters can be changed while the code runs, like speed.

As JSON this pin setup would look something like:

```
{
  "pins": {
    "name": "jane",
    "title": "IO pin setup for 'jane' application on 'carol' hardware",
    "inputs": {
          "LIGHT_SWITCH": {"bank": 4, "addr": 2, "speed": 3},
          "GARAGE_DOOR_SWITCH": {"bank": 1, "addr": 5}
    },
    "outputs": {
          "WORK_LIGHT": {"bank": 1, "addr": 3, "group": "MYLIGHTS"},
```

```
        "NIGHT_LIGHT": {"bank": 1, "addr": 5, "group": "MYLIGHTS"}
    },
    "analog_inputs": {
        "TEMPERATURE_SENSOR": {"bank": 2, "addr": 2, "speed": 3, "delay": 11}
    }
  }
}
```

1.3 Generating C source and header files from JSON

190917, updated 17.9.2019/pekka

From this JSON file, let's call this alice.json, we can automatically generate a C header and code files, which contain this info, for example alice.h:

extern const Pin alice_WORK_LIGHT;
extern const Pin alice_LIGHT_SWITCH;
extern const Pin alice_GARAGE_DOOR_SWITCH;

And in C file

/* Parameters for pins, do not use const, these can be modified.
 */
static os_short alice_prm_LIGHT_SWITCH[] = {PIN_SPEED, 3};
static os_short alice_prm_GARAGE_DOOR_SWITCH[] = {PIN_SPEED, 5, PIN_DELAY, 1};

/* Parameters for pins, do not use const, these can be modified.
 */
const Pin alice_WORK_LIGHT = {"WORK_LIGHT", PIN_BOUT, 5|(3 << 8), OS_NULL, 0);
const Pin alice_LIGHT_SWITCH = {"LIGHT_SWITCH", PIN_BIN, 4|(2 << 8), alice_prm_LIGHT_SWITCH, sizeof(alice_prm_LIGHT_SWITCH)/sizeof(os_short));
const Pin alice_GARAGE_DOOR_SWITCH = {"GARAGE_DOOR_SWITCH", PIN_BIN, 4|(7 << 2), alice_GARAGE_DOOR_SWITCH, sizeof(alice_prm_GARAGE_DOOR_SWITCH)/sizeof(os_short));

- Bank and pin address are encoded as sigle integer. Bank number 0 means that bank is not used, bank 1 translates to as text to "BANK A", bank 2 to "BANK B"...
- I prefix everything with some version of "pin" or board name, this is to avoid using global symbols which have been already used by some other library, etc, in the same binary.

Then we needs a normal header file in pins library, like "pin_defs.h":
Enumeration of pin types

```
typedef enum
{
    PIN_INPUT,
    PIN_OUTPUT,
    PIN_ANALOG_INPUT,
    PIN_ANALOG_OUTPUT,
    PIN_PWM,
    PIN_TIMER
}
pinType;
```

Enumeration of pin parametes

```
typedef enum
{
    PIN_SPEED,
    PIN_DELAY
}
pinPrm;
```

Pin definition structure

```
typedef struct Pin
{
    /* Pointer to pin name string.
     */
    os_char *name;

    /* Pint type, like PIN_INPUT, PIN_OUTPUT... See pinType enumeration.
     */
    pinType type;

    /* Hardware bank number for the pin, if applies.
     */
    os_short bank;

    /* Hardware address for the pin.
     */
    os_short addr;

    /* Pointer to parameter array, OS_NULL if none.
     */
    os_short *prm;

    /* Number of items in parameter array.
     */
    os_char prm_n;

    /* Next pin in linked list of pins belonging to same group as this one.
     */
    const struct Pin *congroup_next;

    /* Next pin in linked list of io configuration.
     */
    const struct Pin *board_next;
}
Pin;
```

There is group_next and board_next. Often is handy to loop trough all pins, like when making memory map for IO com. Or reading group of inputs with one command. To facilitate this we can group pins together. Pins with same PIN_GROUP number set go generate linked list and all pins of IO board a second linked list.

These appear in automatically generated .h file as:

extern const Pin *alice_pins; Pointer to first pin of alice board. We can loop trough pins by following board_next pointers.
extern const Pin *alice_group_1_pins; Pointer to first pin of group 1 on alice board. We can loop trough pins by following group_next pointers.

WARNING: Python 3.6 or newer is needed to preserve dictionary order.
If run with older versions, IO pins in C are not in same order as in this C file,
which can cause problems down the road since linked list order is may be used to
give pins addresses is iocom memory block.

1.4 Pins library functions

190917, updated 17.9.2019/pekka

- pin_set(Pin *pin, os_int state); Set pin state
- os_int pin_bin_get(Pin *pin); Get pin state

- pin_set_prm(Pin *pin, pinPrmEnum prm, os_int value); Modify IO pin parameter
- os_int pin_get_prm(Pin *pin, pinPrmEnum prm); Get value of IO pin parmeter
- pin_get_info(Pin *pin, os_int flags, os_char *buf, os_memsz buf_sz); Get info in printable for according to flags

1.5 MIT license

190625, updated 27.6.2019/pekka