Is there a way to write a generic GPIO code for MSP430?

Asked 4 years ago Modified 4 years ago Viewed 610 times

2

I'm currently working on an MSP430 board and I'm trying to create libraries for it so I can easily use them on my project. I'm starting with basic Digital I/O functions.

Say for example I need to set P1.0 ON. In that case, normally what I do is:

```
P1SEL &= (~BIT0);           // Set P1.0 SEL for GPIO
P1DIR |= BIT0;              // Set P1.0 as Output
P1OUT |= BIT0;              // Set P1.0 HIGH
```

But I need to write a function that will just take Port Num, Pin and Value and set the above registers. Something like this:

```
void setVal(int x, int y) {
    PxSEL &= (~BITy);
    PxDIR |= BITy;
    PxOUT |= BITy;
}
```

Where x is Port and Y is Pin. Is there a way I can implement such a function? OR has this been done before? If yes, please share the link for the same. I was thinking out maybe using a lookup table and selecting the Register via indexing. But I'm not sure if that's a good approach. Any help would be appreciated.

Thanks

c gpio msp430 library

Share Cite Follow

asked Sep 9, 2018 at 2:39

Shantanu Mhapankar
21 1

The bitmask part is easy: `uint16_t BITy = 1 << y;` . To get the Port registers, you'll probably have to look at the raw addresses and figure out how to calculate them from `y` . – The Photon Sep 9, 2018 at 2:50

Thanks @ThePhoton. Is there no better way to do it for the PORT number part? – Shantanu Mhapankar Sep 9, 2018 at 2:52

It'd be more portable to do some big case statement using the PxSEL/PxDIR/PxOUT macros. Hopefully your compiler will be smart enough to reduce that to something reasonably quick at runtime. – The Photon Sep 9, 2018 at 2:57

4 Answers

Sorted by: Highest score (default)

2

This can be done with a set of macros:

```
#define GPIO_MODE_GPIO(...)  GPIO_MODE_GPIO_SUB(__VA_ARGS__)
#define GPIO_MODE_GPIO_SUB(port, pin) (P##port##SEL &= ~(1<<pin)) // making an
assumption about register layout here

#define GPIO_OUT_SET(...)  GPIO_OUT_SET_SUB(__VA_ARGS__)
#define GPIO_OUT_SET_SUB(port, pin) (P##port##OUT &= ~(1<<pin))

//etc...
```

The `##` operator does string concatenation in the preprocessor. The two-level macros are used to allow a level of macro expansion to happen between the first and second macros, which allows you to do things like this:

```
#define LED_IO  1,5
GIPO_OUT_SET(LED_IO);
```

Without that extra indirection, the preprocessor would complain about not having enough arguments to the `GPIO_OUT_SET` macro.

This style of system has adapted quite well to every MCU I've used so far, from AVRs to ARMs, and since it's a preprocessor macro it compiles down into the smallest possible set of instructions.

Share Cite Follow

edited Sep 9, 2018 at 3:03

answered Sep 9, 2018 at 2:59

ajb

3,299 11 29

3 This only works if the port number is known at compile time. OP's skeleton code implied it will be passed in to a function at run time (though that might not be what they actually need). – The Photon Sep 9, 2018 at 3:01

That's what I was assuming Photon – Passerby Sep 9, 2018 at 3:09

Fair enough, that was a big assumption on my part. I can imagine some situations where you'd want to set an IO based on a variable, but I've never actually run into such an application myself (aside from Arduino and other heavily abstracted HALs, where the GPIO is runtime-abstracted for convenience rather than actual need or benefit). – ajb Sep 9, 2018 at 3:15

@ThePhoton You mean this only won't work if I want to set a port based on a variable I've received during run-time right? But will work if my LED_IO and other IO's are constant? – Shantanu Mhapankar Sep 11,

2018 at 2:12

Is there a way I can implement such a function?

1

Sure, this can be done. But the code will not be portable anymore.
You will have to validate the behavior when the part changes.

The GPIO peripheral registers are mapped on the bus in the following ranges:

```
MODULE       BASE  SIZE
Port P1, P2  0200h 0020h
Port P3, P4  0220h 0020h
Port P5, P6  0240h 0020h
```

This is a slightly untraditional implementation from TI. They've merged even and odd ports into one block. Usually all registers of one GPIO block are consecutive, what would make an arithmetic approach more feasible.
However, above layout gives the following registers:

```
P1OUT = 0200h + 02h
P2OUT = 0200h + 03h
P3OUT = 0220h + 02h
P4OUT = 0220h + 03h
P5OUT = 0240h + 02h
P6OUT = 0240h + 03h
```

With those you can make the following code, using above table:

```
void setVal(int x, int y) {
    const unsigned char *PxOUT = {0x202, 0x203, 0x222, 0x223, 0x242, 0x243};
    if(y)
        *PxOUT[x] |= (1<<y);
    else
        *PxOUT[x] &= ~(1<<y);
}
```

A table is fastest method available. At the costs of some ROM.
This can be changed to allow access to other registers as well.
For example PxOUT[] -2 accessed the PxIN register, and +2 is the PxDIR register.

I've taken my info from the datasheet of the msp430fr2153.
Table 6-33 and Table 6-41. This might not match your part, **please check**.

Share Cite Follow

answered Sep 10, 2018 at 8:44

Jeroen3
21k 33 72

My approach on MCUs that offer >8kB of memory is to pack GPIOs in structures:

```
//
// GPIO Single Pin Descriptor
//
struct GPIO_SPD      {      const char *    Description;
                            uint8_t         Port;
                            uint16_t        Pin;
                            bool            Initialzied;
};
```

Then you can define a GPIO inside your module:

```
struct GPIO_SPD GpioBlinkingLed= {"Simply blinking LED", GPIO_PORT_P1, GPIO_PIN0,
false};
```

And use this structure inside simple gpio functions similar to yours. This structure, for instance, can be directly used inside TI driverlib, as follows:

```
GPIO_setAsOutputPin(GpioBlinkingLed.Port, GpioBlinkingLed.Pin);
GPIO_setOutputHighOnPin(GpioBlinkingLed.Port, GpioBlinkingLed.Pin);
GPIO_toggleOutputOnPin(GpioBlinkingLed.Port, GpioBlinkingLed.Pin);
// etc.
```

These functions reside in so called gpio.c driver and, of course, are built upon a lookup table:

```
void GPIO_setOutputHighOnPin (uint8_t port, uint16_t pin)
{
    uint16_t portAddress = GPIO_PORT_ADDRESS_TABLE[port];

    (*((volatile uint16_t *)(portAddress + GPIO_OUT_REG_OFFSET))) |= pin;
}
```

I specifically simplified it, but you'll get the the idea. `GPIO_PORT_ADDRESS_TABLE` and `GPIO_OUT_REG_OFFSET` are predefined and values are taken from the MCU datasheet. Thankfully TI provides MSP430WARE that does those last bits for you.

Share Cite Follow

edited Sep 10, 2018 at 8:58

answered Sep 10, 2018 at 7:27

Andrejs Gasilovs
604 1 6 12

Hi @GAttuso, thanks for this! I'm unable to find GPIO_PORT_ADDRESS_TABLE and GPIO_OUT_REG_OFFSET for my board MSP430G2553 in MSP430WARE though? – Shantanu Mhapankar Sep 11, 2018 at 2:14

@Shantanu Mhapankar I have checked driverlib and haven't found header files for MSP430G2x series. So, probably, you'll need to make your own implementation. – Andrejs Gasilovs Sep 13, 2018 at 21:23

0

It be nice if microcontroller code could use substitution like Your typical computer scripting languages could, but that would require an extra layer of complexity that could only be done on the microcontroller, using up memory and code space and processing.

You could do this with if or case statements, and your compiler should be able to optimize it as needed. You could take the register value that the PxSel or whatever actually means, and use that instead. It would still require if or case or comparison statements. As photon mentions in the comments, you'd basically be building a lookup table. Thankfully these are fairly limited.

That said, these functions have been done to death, the wheel rewritten by countless devs multiple times. A Google search for gpio library would result in a couple. TI offers some one their code examples. And a fairly generic copy can be found on ***Energia***, the msp430 (and other TI microcontrollers) port of the Arduino code framework. Energia can be used directly, just for the gpio Pin library, or you can copy it out, following the license for it.

Share Cite Follow

answered Sep 9, 2018 at 3:07

Passerby

71.5k 7 86 196