Macros in TivaWare

If you ever tried to do direct register programming with the method in the "blinky" example (see **How to "find" registers - 1**) and also the TivaWare driverlib, you probably faced problems. Allot of warnings of re-definitions.

These are due to the first method adding the include #include "inc/tm4c123gh6pm.h"which conflicts with some other headers in the driverlib.

Luckily TivaWare has defined macros to solve this problem. You can find these in the "hw_types.h" inside the "inc" folder.

These macros are also useful to help just access a exact bit.

Here are the macros taken from that said source file from TivaWare:

```
//*****************************************************************************
//
// Macros for hardware access, both direct and via the bit-band region.
//
//*****************************************************************************
#define HWREG(x) \
        (*((volatile uint32_t *)(x)))
#define HWREGH(x) \
        (*((volatile uint16_t *)(x)))
#define HWREGB(x) \
        (*((volatile uint8_t *)(x)))
#define HWREGBITW(x, b) \
        HWREG(((uint32_t)(x) & 0xF0000000) | 0x02000000 | \
              (((uint32_t)(x) & 0x000FFFFF) << 5) | ((b) << 2))
#define HWREGBITH(x, b) \
        HWREGH(((uint32_t)(x) & 0xF0000000) | 0x02000000 | \
               (((uint32_t)(x) & 0x000FFFFF) << 5) | ((b) << 2))
#define HWREGBITB(x, b) \
        HWREGB(((uint32_t)(x) & 0xF0000000) | 0x02000000 | \
               (((uint32_t)(x) & 0x000FFFFF) << 5) | ((b) << 2))
//*****************************************************************************
```

The first 3 macros simply receive a 32, 16 , 8 bit pointer, personally i just use the first one of 32bits pointer, since the registers addresses are of 32 bits.

The last 3 macros allow you to access registers with bit banding.

You should just need HWREGfor access registers. But the rest is useful if you just want to save memory when using a 16bit or 8bit word, using the 32bit macro would be unnecessarily.

Examples

If you usually program with TivaWare (which i advise unless it's necessary to use direct register) you should know the macros for the peripherals.

First let's see a simple example. To set a output to 1. Let's say PF1. Here is the TivaWare driverlib example:

```
GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_1, GPIO_PIN_1);
```

Now the direct register access equivalent:

```
HWREG(GPIO_PORTF_BASE+ (GPIO_O_DATA + (GPIO_PIN_1<< 2))) = GPIO_PIN_1;
```

IMPORTANT

The line of code is not completely equivalent to the TivaWare API function call. Although it will do the same, turn the LED on and it is what you should do, the functions from the API also have an assert to check the base you inserted as parameter.

Now let's see a example for HWREGBITW. Let's use enabling a peripheral clock for the GPIOF. You will see some diferences here from "blinky" example.

First the TivaWare API example:

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
```

Now with the Macros:

```
HWREGBITW(SYSCTL_RCGCBASE + ((SYSCTL_PERIPH_GPIOF & 0xff00) >> 8), SYSCTL_PERIPH_GPIOF &
```

You should just need to use this and change the peripheral (here SYSCTL_PERIPH_GPIOF) to enable the clock.

As for the HWREGBITW. It just takes the as a first parameter a address but then it just changes 1 bit, specified in the 2nd parameter. In this case it starts in system control register base address + 0x608 (1st parameter) and changes bit5 (2nd parameter) to 1 ("= 1").

The calculations for this line of code:

SYSCTL_PERIPH_GPIOF =0xf0000805, so 0xf0000805 & 0xff00
= 0x00000800,0x00000800 >> 8
= 0x00000008.
With SYSCTL_RCGCBASE having a offset 0x600 (from the system control base address) ,SYSCTL_RCGCBASE +0x00000008 = 0x608.

0xf0000805 & 0xFF = 5 -> that's bit 5

I hope you understand now better some of the macros from TivaWare when using direct register access. Next i will explain how to get a TivaWare function equivalent in direct register access easily. I hope you liked this tutorials and i leave you with the blinky code with these macros.

Now let's see blinky code

```
//*****************************************************************************
//
// blinky.c - Simple example to blink the on-board LED.
//
// Copyright (c) 2012-2014 Texas Instruments Incorporated.  All rights reserved.
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
// This is part of revision 2.1.0.12573 of the EK-TM4C123GXL Firmware Package.
//
//*****************************************************************************

/*
 * This is the example blinky with TivaWare macros like HWREG
 *      It was added the files:
                #include <stdint.h>
                #include <stdbool.h>
                #include "inc/hw_memmap.h"
                #include "inc/hw_types.h"
                #include "inc/hw_gpio.h"
                #include "inc/hw_sysctl.h"
                #include "driverlib/sysctl.h"
                #include "driverlib/sysctl.c"
                #include "driverlib/gpio.h"

                The somewhat odd one, "driverlib/sysctl.c" was added because there are u


 */



#include <stdint.h>
#include <stdbool.h>
#include "stdlib.h"
// don't use #include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "inc/hw_sysctl.h"
#include "driverlib/sysctl.h"
#include "driverlib/sysctl.c"
#include "driverlib/gpio.h"

//*****************************************************************************
//
//! \addtogroup example_list
//! <h1>Blinky (blinky)</h1>
//!
//! A very simple example that blinks the on-board LED using direct register
//! access.
//
//*****************************************************************************


//*****************************************************************************
//
```

```
// Blink the on-board LED.
//
//*****************************************************************************
int
main(void)
{
        volatile uint32_t ui32Loop;

    //
    // Enable the GPIO port that is used for the on-board LED.
    //

    //Before: SYSCTL_RCGC2_R = SYSCTL_RCGC2_GPIOF;
        //Note, this, unlike in the blinky example, doesn't use a legacy register
    HWREGBITW(SYSCTL_RCGCBASE + ((SYSCTL_PERIPH_GPIOF & 0xff00) >> 8),SYSCTL_PERIPH_GPIOF
    /*
     * alternative:
     * HWREG(SYSCTL_RCGCGPIO + 0x05) |= 1;
     * I prefer this one since it uses address defined in "hw_sysctl.h"
     * I like to use the address and offsets in the "hw" files.
     * Notice that SYSCTL_RCGCGPIO is actually absolute address instead of a offset
     */


    //
    // Do a dummy read to insert a few cycles after enabling the peripheral.
    //
    //before: ui32Loop = SYSCTL_RCGC2_R; <- this macro, SYSCTL_RCGC2_R now doesn't exist
    ui32Loop = SYSCTL_RCGCBASE;
    //
    // Enable the GPIO pin for the LED (PF3).  Set the direction as output, and
    // enable the GPIO pin for digital function.
    //

    //beforeGPIO_PORTF_DIR_R = 0x08;
    HWREG(GPIO_PORTF_BASE + GPIO_O_DIR) = GPIO_PIN_3;

    //before :GPIO_PORTF_DEN_R = 0x08;
    HWREG(GPIO_PORTF_BASE + GPIO_O_AFSEL) = GPIO_PIN_3;

    //
    // Loop forever.
    //
    while(1)
    {
        //
        // Turn on the LED.
        //

        //before: GPIO_PORTF_DATA_R |= 0x08;
        HWREG(GPIO_PIN_3 + (GPIO_O_DATA + (GPIO_PIN_3 << 2))) = GPIO_PIN_3;

        //
        // Delay for a bit.
        //
        for(ui32Loop = 0; ui32Loop < 200000; ui32Loop++)
        {
        }

        //
        // Turn off the LED.
        //

        //before: GPIO_PORTF_DATA_R &= ~(0x08);
        HWREG(GPIO_PIN_3 + (GPIO_O_DATA + (GPIO_PIN_3 << 2))) = ~GPIO_PIN_3;

        //
        // Delay for a bit.
        //
```

```
            for(ui32Loop = 0; ui32Loop < 200000; ui32Loop++)
            {
            }
        }
    }
```