How to "find" registers - 1

This is if you want to start using, or just understanding better, how to program in direct register. Remeber this is for the TM4C123GH6PM MCU and all datasheet references are from that MCU

In this first tutorial let's see the example from TivaWare "Blinky", here it is

```
//*****************************************************************************
//
// 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.
//
//*****************************************************************************

#include <stdint.h>
#include "inc/tm4c123gh6pm.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.
    //
    SYSCTL_RCGC2_R = SYSCTL_RCGC2_GPIOF;
```

```
    //
    // Do a dummy read to insert a few cycles after enabling the peripheral.
    //
    ui32Loop = SYSCTL_RCGC2_R;

    //
    // Enable the GPIO pin for the LED (PF3).  Set the direction as output, and
    // enable the GPIO pin for digital function.
    //
    GPIO_PORTF_DIR_R = 0x08;
    GPIO_PORTF_DEN_R = 0x08;

    //
    // Loop forever.
    //
    while(1)
    {
        //
        // Turn on the LED.
        //
        GPIO_PORTF_DATA_R |= 0x08;

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

        //
        // Turn off the LED.
        //
        GPIO_PORTF_DATA_R &= ~(0x08);

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

Notice that this example doesn't use TivaWare APIs. It just uses Macros for register addresses that come with the tm4c123gh6pm.h Header that is included in the TivaWare.Depending on the part you are using you need to include the right header file.

Let's try to understand the code:

First we have SYSCTL_RCGC2_R = SYSCTL_RCGC2_GPIOF;

This enables the clock for the GPIOF. How do we find these registers in the datasheet? First the see the first "word", **SYSCTL**. This means that the register belongs to System Control. So let's go to the datasheet and select marker 5, "5 System Control". We want the registers so let's go to **5.4** marker which is "**Register Map**". Next let's search (CTRL+F) for **RCGC2**. You can click the page number the description is in, indicated on the table, which should be 464, and that will take you to the **RCGC2**description.

There it will explain that the **RCGC2** is one of the registers that controls which modules have the clock enable to them. You see GPIOF in bit5. If you check the headers you will se that SYSCTL_RCGC2_GPIOFcorresponds to 32 (decimal) or 0x20.

IMPORTANT

In this example the legacy registers are used to enable the GPIOF clock. Normaly you should not use legacy registers. Use RCGCGPIO , offset 0x608, to enable the GPIO clocks

ui32Loop = SYSCTL_RCGC2_R;

After enabling the clock you need to wait at least 3 clock cycles before accessing GPIOF registers, that's what ui32Loop =SYSCTL_RCGC2_R; is for. If you check some of the TivaWare examples you will see the use of SysCtlDelay(), but here we don't have any of that, that's why there's that dummy attribution.

```
GPIO_PORTF_DIR_R = 0x08;

GPIO_PORTF_DEN_R = 0x08;
```

Now with GPIO_PORTF_DIR_R =0x08;

GPIO_PORTF_DEN_R = 0x08;not only it sets the pin as a output, but also sets it as a digital pin. Let's see better these registers in the datasheet. They start with the GPIO, so we know we have to search the register map from the GPIO, marker 10.4.

Now here you **don't search for PORTF**. Instead search forDIR in Table 10-6 GPIO Register Map. You will find a register named **GPIODIR not just DIR**. This is the one you want. **All GPIO registers are like that if you notice the rest of the table, they start with GPIO**. To see the register description press the page on the table corresponding to that register.

Now do the same for GPIO_PORTF_DEN_R to see the description. The only thing that this does is set the pin as digital or analog, check Table 10-3. GPIO Pad Configuration Examples for more info.

In **GPIODIR**, when a bit is set (to 1) the corresponding pin bit changes to a output (value 0 means input). So 0x08 sets bit3 to 1, so PF3 now is a output and all the other input. Note that since a equal is used then you set all the bits in the register. To simply change the bit3 to 1, while not changing the others just use a OR: GPIO_PORTF_DIR_R |= 0x08;

For **GPIODEN**, as you can read in the description, 1 means the pin is digital, 0 means it's analog.

Now first a bit of clarification about the GPIO registers.

Each GPIO has **a different base address**.

There are multiple GPIO right? They all have exactly the same registers. Meaning that the direction register for the GPIOF has exactly the same offset from the base address.

Ex:

- **GPIOF** has a base address of **0x4005.D000. GPIODIR**has a offset of **0x400**. This means **GPIODIR** has a absolute addressof 0x4005.D000+0x400. But for **GPIOD** with a address of **0x4005.B000**, **GPIODIR**for it has a absolute addressof **0x4005.B000+0x400**.

This makes it easier to always have the same offset for the registers. The difference from configuring GPIOF or GPIOD is just using a different base address.

GPIO_PORTF_DATA_R

Now try to do the same for GPIO_PORTF_DATA_R. Search on the datasheet for the description and how to use it. If you have any questions feel free to do it here [Discussion/Questions](). Tip(check out 10.2.1.2 Data Register Operation and how big is the difference between GPIODATA and GPIODIR address and compare with the others. Notice anything different?)

Next i will explain some macros that help using direct register access with TivaWare.

Quick note on logic operations for register access:

REG|= 0x08;
REG &= ~(0x08);.

- The first operation with a OR, will do REG = REG | 0x08. This will set the bit3 (0x08 in hexa= 1000 binary) to 1 without changing the other bits. Handy to set to 1 just the bits you want.

- Second will do, REG = REG & ~(0x08). This is REG = REG & ~(0000 1000), REG = REG & (1111 0111). This will set the bit3 to 0 and leaves all the other bits untouched. Pretty handy to clear just the bits you want.