Maintainers Manual: STM32F407 – P24 with Clock Interface and other Peripherals

Emanuelle Crespi – Spring 2016

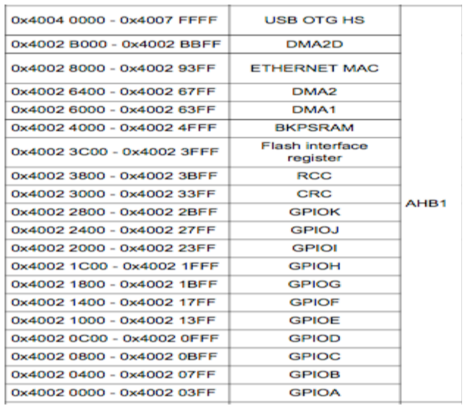
Table of Contents

Modules/Component Interface

NOTE: *For all of the following components, please refer to the documented source code for explanations of the pin initializations and configurations. These can be found in the project directory in the files: P24v04r16A_GPIOinit.asm, display_led_switch_drivers.asm, and display_led_switch_functions.c. Also, the P24v04r16A.brd file can be useful for observing PIN layouts.*

A. GPIO Explained

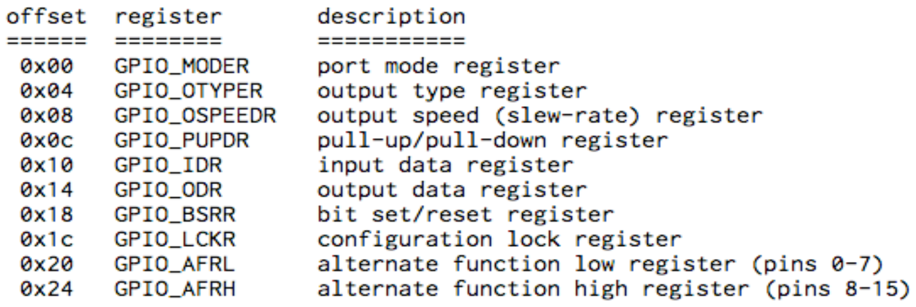
This Project uses GPIOA-D to activate the different components of the P24 board explained in the sections below. Each of these GPIOs have 0-16 pins identified with its letter followed by the pin number. For example, PA1 corresponds to pin one of GPIOA. GPIO are located in the AHB1 peripherals base. The GPIO memory locations can be found in the RM0090.pdf manual on page 65 with corresponding links to the register maps of these pins. (Figure 2)



0x4004 0000 - 0x4007 FFFF	USB OTG HS	
0x4002 B000 - 0x4002 BBFF	DMA2D	
0x4002 8000 - 0x4002 93FF	ETHERNET MAC	
0x4002 6400 - 0x4002 67FF	DMA2	
0x4002 6000 - 0x4002 63FF	DMA1	
0x4002 4000 - 0x4002 4FFF	BKPSRAM	
0x4002 3C00 - 0x4002 3FFF	Flash interface register	
0x4002 3800 - 0x4002 3BFF	RCC	
0x4002 3000 - 0x4002 33FF	CRC	
0x4002 2800 - 0x4002 2BFF	GPIOK	AHB1
0x4002 2400 - 0x4002 27FF	GPIOJ	
0x4002 2000 - 0x4002 23FF	GPIOI	
0x4002 1C00 - 0x4002 1FFF	GPIOH	
0x4002 1800 - 0x4002 1BFF	GPIOG	
0x4002 1400 - 0x4002 17FF	GPIOF	
0x4002 1000 - 0x4002 13FF	GPIOE	
0x4002 0C00 - 0x4002 0FFF	GPIOD	
0x4002 0800 - 0x4002 0BFF	GPIOC	
0x4002 0400 - 0x4002 07FF	GPIOB	
0x4002 0000 - 0x4002 03FF	GPIOA	

Figure 2: GPIO Locations in Memory Map

Every GPIO has its own dedicated set of registers in the STM32 so that the pins can be configured to their desired mode [input/output, pull up/pull down, etc]. These modes are shown in Figure 2. The drivers make initializations of these pins by setting a particular set of bits in these registers (Figure 3). (Initializations of these pins can be found in *P24v04r16A_GPIOinit.asm)*



```
offset  register       description
======  ========       ===========
 0x00   GPIO_MODER     port mode register
 0x04   GPIO_OTYPER    output type register
 0x08   GPIO_OSPEEDR   output speed (slew-rate) register
 0x0c   GPIO_PUPDR     pull-up/pull-down register
 0x10   GPIO_IDR       input data register
 0x14   GPIO_ODR       output data register
 0x18   GPIO_BSRR      bit set/reset register
 0x1c   GPIO_LCKR      configuration lock register
 0x20   GPIO_AFRL      alternate function low register (pins 0-7)
 0x24   GPIO_AFRH      alternate function high register (pins 8-15)
```

Figure 3: GPIO Registers

B. 7-Segment Display

The 7- Segment Display is controlled through the U_CA and U_AN latches on the P24. The GPIO are configured as output pins and send signals to these flip-flops to enable/disable and set various cathodes (for segments A-G) and anodes (for digits, colon, and decimal) patterns for the display (Figure 3).

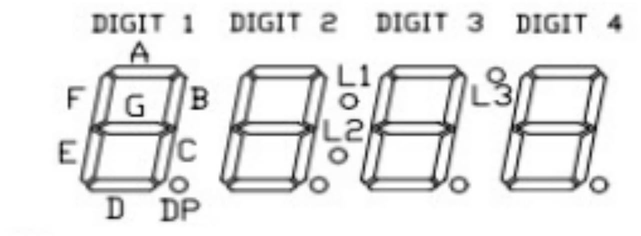


Figure 3: 7-Segment Display Layout

NOTE: The latches are active low on input! It is important to disable then clock the latch after setting the correct anode/cathode pattern. For example, setting an output pin to 1 will set the corresponding input pins to a 0. [This also applies to the rest of the components below that rely on the U_CA and U_AN latches]

The pattern will display on every digit after the changes have been clocked, so it is necessary to use time multiplexing on each digit if you want to show different patterns on each digit. A 250Hz refresh on each digit proves sufficient for an acceptable intensity. (Refer to the drivers: *display_led_switch_drivers.asm* and especially *display_led_switch_functions.c* for explanations in the commented code on how to use the display)

C. LEDs

LEDs 1-6 are above the buttons, between the 7-segment display and the rotary encoder. The LEDs share some of the cathode pins with the 7seg display, so it is important to pay special attention when clocking any changes into these components. The cathodes can be used to toggle the LED between on and off and the anodes allow two possible colors, red and green. Pulsing both anodes at the same time has the LEDs appear orange. The drivers written for these components are also found in *display_led_switch_drivers.asm* and especially *display_led_switch_functions.c* with well-commented explanation of their functionality.

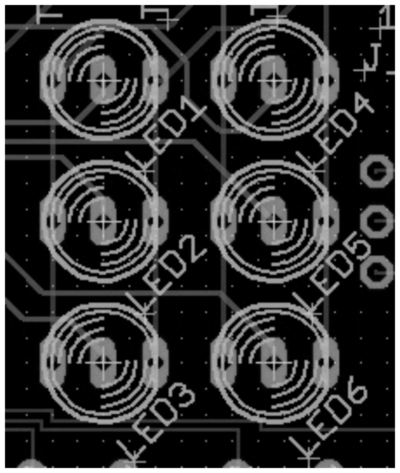


Figure 4: LEDs 1-6 on P24 Board

D. Buttons

Buttons 1-13 are labeled S1-S13 at the very bottom of the P24 board (Figure 5). Their layout is particular in that the columns share cathode outputs on the particular pins and each row has a set of buttons sharing the same output pin (Figure x). To single out a button press, a function pulses the columns 1-7 for corresponding buttons 1-2, 3-4, etc.., then reads for a 0 in PA15 (odd button) or PC8 (even button) to verify the button. There is also a useful function called 'button_state_machine' that accounts for any de-bouncing issues on a button press. To modify the confirmation of a button press the static wait value can be changed accordingly.

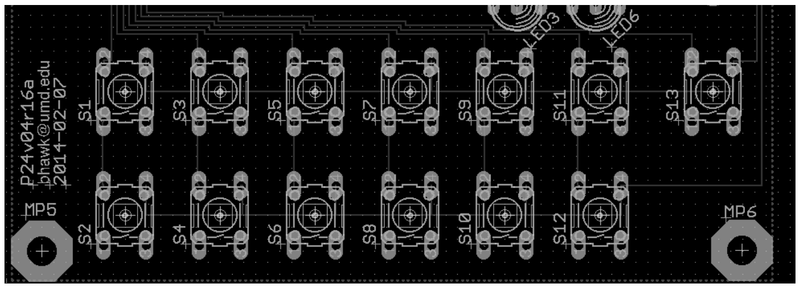


Figure 5: Buttons 1-13 on P24 Board

E. Rotary Encoder

Processing the rotary encoder is done by enabling it's anode and reading from pins PA15 and PC8 to observe corresponding states A and B. Again, please refer to *display_led_switch_drivers.asm* and especially *display_led_switch_functions for a better explanation on how to process turns.*

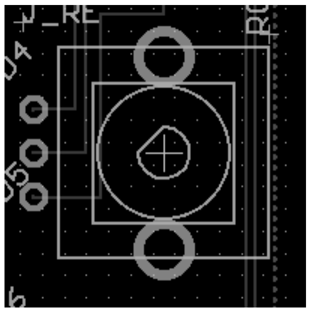


Figure 6: Rotary Encoder (to the right of LEDs 1-6) on P24 Board

---

System Level (States) **[Please refer to file not_so_trivial_main.c to view implemented source code]**

F. Power Up/Init Mode – (RTC Setup/Calibration)

The 7-Segment display shows all zeros on the display in standard 24-hr format allowing for the user to calibrate the time using the buttons and the rotary encoder. Initializations on the hardware include Systick calibrated at 1ms interrupts, RTC initializations for setup and also startup and calibration of the TIM5 interrupt.

***NOTE**: It is important to first perform a full chip erase, followed by programming the board with the USBvcp13.hex file before programming and running the Blinky.hex compiled using makeBlinky.*

Rotary Encoder: The rotary encoder is available to provide an easier option for calibrating the time by the minute.

Buttons 1-4: Immediately after power up, the system will be in it's Initialization Mode. This mode is for calibrating the general purpose Real Time Clock peripheral. Corresponding buttons 1-2 handle increase and decrease on the hours, while buttons 3-4 are available for increase and decrease on the minutes. Buttons 5-12 will remain unavailable in this state.

Button 13: When the user has finished calibrating the time, button 13 will set the clock to the desired hours and minutes, starting at 0 seconds. This is followed by a transition to the Home Menu where a link to an available COM Port via TIM2 becomes available. You may now power the other end of the STM32 to communicate to the device using PUTTY.

There is no getting back to this state after pressing button 13, unless a reset is performed. To reset, either remove and reestablish the power connection to the board or press one of the available reset buttons [SWB above the anode latch on the P24 or the blue button on the STM32]

G. Home Menu

- Systick→SVC→PendSV Implemented Tasks

Immediately after button 13 is pressed in Init Mode –

1. TIM2 is initialized and set up the VCP functions for an available COM link. The hardware may now be controlled via remote control on PUTTY [Refer to Final_Proj_User_Manual.pdf for more info on setting this up]

2. SVC is set with the highest priority of the interrupts to allow for Systick scheduled task switching

***NOTE*: The SysTick→SVC→PendSV interrupt routine has been set up so that buttons 7-13 invoke corresponding svc interrupts 0-6 with SysTick polling the global SVC flags set on a button press. The functionality of a task switch has yet to be implemented due to time constraints on this project, but the setup is there. For now, a button press on 7-13 will print to display which SVC interrupt has been invoked. For example, pressing button 7 will print "SVC 0 Task" to indicate that a task switch can be implemented at this state** [*refer to SVC_Handler(void) function in TIMx_PSV_SVC_Handlers.c*]

- Toggling LEDs (GRN→RED→OFF)

Buttons 1-6 will toggle corresponding LEDs 1-6 between states GREEN→RED→OFF

This could be used to represent more modes such as *STARTED/STOPPED/TERMINATED* since their states ON/OFF and color GRN/RED are global. For now, they are what I like to call the LED game, since they serve no particular, but the can be useful in indicating a peripheral (such as ADC or DMA) being toggled in the background

H. Remote Control (ON/OFF)

- Shell Interface via USB/VCP Commands – (TIM2)

[Again, please reference the user manual for instructions on accessing the remote control interface using PUTTY] TIM2 is constantly polling for characters using the VCP commands described in the *TIMx_PSV_SVC_Handlers.c* source code. When the controller is ON the user may now access the board using the keyboard to toggle LEDs, turn ON/OFF the ADC and calibrate the ADC.

Pressing 'Enter' will turn the remote ON/OFF. **In the ON state, the user will not have access to the buttons on the hardware**, but will now be able to play 'the LED game' described in part G with the corresponding keys 1-6. For example, typing '1' will generate a button press on button 1. This will work for corresponding buttons 1-6 on keys 1-6. Also not that buttons 7-13 are not available [This has yet to be implemented].

NOTE: ***When remote control is OFF, all buttons 1-13 are once again available to the user on the hardware. Pressing buttons 1-6 will toggle LEDs 1-6 and buttons 7-13 interrupt on svc 0-6.***

→Toggling LEDs [typing '1'-'6' toggles the corresponding LEDs]

→ADC ON/OFF [typing 'adcon' or 'adcoff' followed by pressing the space bar will toggle the global ADCON flag [the ADC has yet to be implemented]

→ADC Calibration via Rotary Encoder on TIM5

[typing 'adccal' or 'done' followed by pressing the space bar will enter/exit this mode] The ADC can be calibrated visually by use of the rotary encoder on TIM5. **Entering this mode will turn off the 7-segment display and pulse LEDs 1-6 to be red**. Turning right will toggle global LED_color 1-6 to be GREEN in that order. Turning left will toggle the LED_color in reverse order. This allows a visual on the calibration scheme for the ADC.

NOTE: The ADC has yet to be implemented, however these routines are available for the future design

→Make the P24 dance [typing 'DANCE' followed by the space bar will make the P24 happy and toggle LEDs1-6 through states GREEN→RED→ORANGE→OFF randomly by reading from the TIM5_CNT register and toggling LEDcurrent_timer_5_count%6+1]

[typing 'dance' followed by a space will make the P24 unhappy setting the LEDs back to their previous state before being told to 'DANCE']