LAB 4

GUIDED WALK THROUGH OF RESOURCES

The following files have been collected in this document to support a guided walk through of some resources introduced and used in Lab 4. In Lab 4, you will do your first lower level programming of an input/output interface on the microcontroller, the GPIO module (General Purpose Input/Output), using it to read pushbutton switches on the CyBot. It is lower level programming because you are not using a library of functions that hide the GPIO details. You are writing code that directly accesses the registers for the GPIO port connected to the pushbuttons (called DRA, or direct register access). Coding at this level requires not only an understanding of C, such as bitwise operations, but also of the GPIO hardware unit in the microcontroller. This guided walk through of resources will help you navigate the documents with important information.

Think of this as a quick start guide. To make the walk through easier, these documents have been put in one file. However, you have all of these documents as separate files, they would typically be found and used as separate files (because that's how you would find them in the workplace), and you will continue to use them as separate files as we go through the labs. Each lab will re-use some resources and add other resources specific to the input/output modules being used on the microcontroller.

Here are the files in this document:

1. Lab4-walkthrough-page1.pdf (this page)
2. Lab4-Prelab.pdf
3. mtg08-notes-concepts-GPIO.pdf
4. Cybot-Baseboard-LCD-Schematic.pdf
5. button.c
6. button.h
7. TI Tiva TM4C123G Microcontroller Datasheet (not included entirely, access separately)
8. tm4c123gh6pm.h (not included entirely, access separately)
9. GPIO-registers.pdf
10. Lab4-Manual.pdf (Part 1)
11. mtg08-slides-GPIO.pdf
12. reading-guide-GPIO.pdf

Let me repeat advice from the lab manual: **Your goal is to be aware of information so that you can find and read it in more detail if and when needed. You are not expected to read or know everything at once.** Take it one step at a time. Be patient but persistent and purposeful. What do you need to know? What do you know that you can build on? What don't you know that might take more effort? Where or how might you get started?
(Arthur Ashe: "Start where you are. Use what you have. Do what you can.")

Embedded Systems International

Lab 4 Prelab

Name:

Lab Partner Name (if you worked together and are submitting the same document or mostly the same answers):

Lab Section:

Submit your prelab document as a PDF file in Canvas under the corresponding prelab assignment. Every student submits their own prelab. Lab partners are allowed to work on the prelab together and submit the same document (if there is actual collaboration on the document). For full credit, the prelab must be submitted prior to the start of lab. Text responses should be typed or printed neatly. You can draw a sketch by hand, or you can use a drawing tool. Try to have started a rough draft of the prelab when you come to class on Tuesday.

1. System sketch

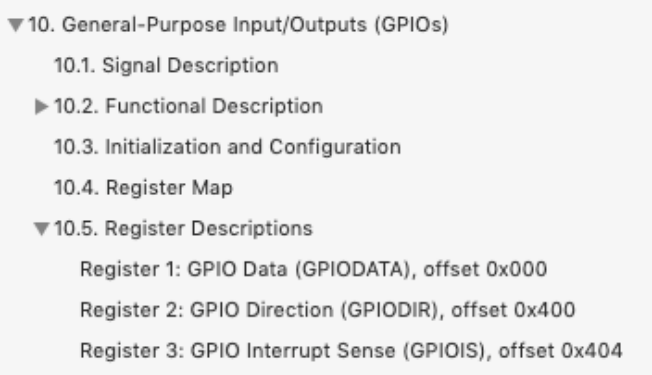
Similar to the system sketch you did in Lab 1, sketch a diagram that shows how the four push buttons connect to the microcontroller. Your diagram should show the port and the pin (or bit) number of the port that each button is connected to. Give the port name and pin numbers used in the microcontroller. The microcontroller package pin numbers (1-64) are different than the port pin names (e.g, Port A pin 0, PA0). Use the following resources as needed: (1) in-class notes, (2) CyBot board schematics and GPIO pin usage, (3) button.c code, and (4) Tiva datasheet. As part of your sketch, indicate the port and bit of the port that each button is connected to.

2. Data register for a GPIO port

The GPIO port connected to the push buttons is associated with a specific GPIO data register. Answer these questions for the actual port used with the push buttons.

a) What is the specific name of the data register as given in the Tiva TM4C Datasheet (shown in all capital letters)?

Hint: see the List of Registers in the Table of Contents, or open the Table of Contents in the sidebar. Excerpts shown below.



b) What is the specific name of the data register as given by the #define macro in the header file, tm4c123gh6pm.h (TM4C123GH6PM Register Definitions)?

Hint: search for “GPIO registers” in the file, ignore the AHB version (if you find it), ignore the “DATA_BITS_R” name, the 32-bit memory address should start with “0x4…”

c) What is the 32-bit memory address of the data register (in hex)?

mtg notes concepts GPIO

CONCEPTS

Previous concepts to be expanded on:
- I/O
- ports
- registers
- memory-mapped I/O
- pointers
- bit wise operations

Hardware

GPIO (General Purpose IO)
- parallel port
- digital input
- digital output
- data register for port
- register descriptions in MCU data sheet
  - memory-mapped registers

/1

1

CONCEPTS (continued)

Software

- port initialization (configuration)
- reading a port (or register)
- writing a port (or register)
- volatile keyword in C

HW/SW

- register definitions in C
- polling method

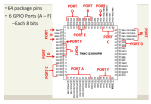
/2

2

CyBot Push Buttons

Push buttons on LCD board are digital inputs connected to microcontroller GPIO PORTE, bits 0-3:

SW1	SW2	SW3	SW4
Bit 0	Bit 1	Bit 2	Bit 3



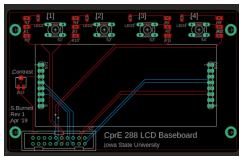
Read the port's data register:

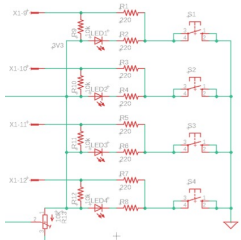
GPIO_PORTE_DATA_R

Then, how does the code know which button is being pushed?

3

Push Buttons on LCD Board (schematics file)

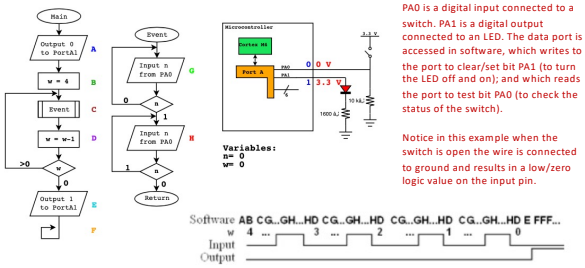




Notice that pressing a button makes contact with ground and results in a low or zero logic value on the connector wire. This is called an active low signal. The digital input is "0" when a button is pressed.

4

Software Example (VY ES book, Ch. 5, Interactive Tool 5.1)



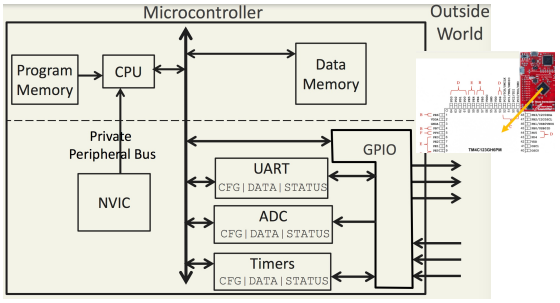
PA0 is a digital input connected to a switch. PA1 is a digital output connected to an LED. The data port is accessed in software, which writes to the port to clear/set bit PA1 (to turn the LED off and on); and which reads the port to test bit PA0 (to check the status of the switch).

Notice in this example when the switch is open the wire is connected to ground and results in a low/zero logic value on the input pin.

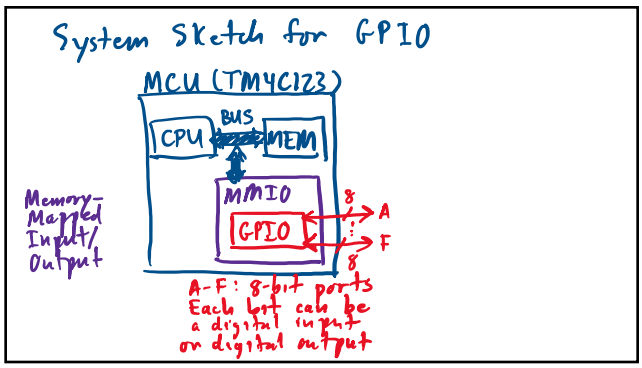
http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C5_IntroductionToC.htm

5

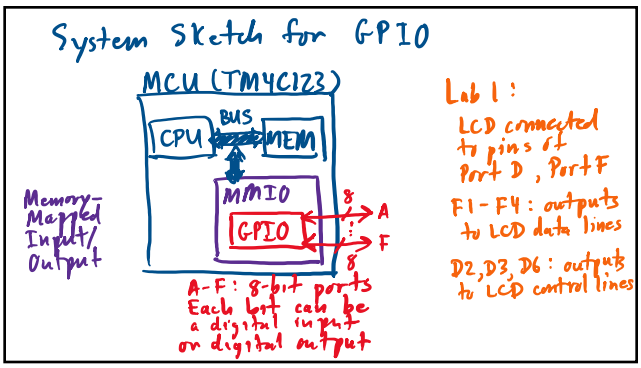
Microcontroller (TM4C123)



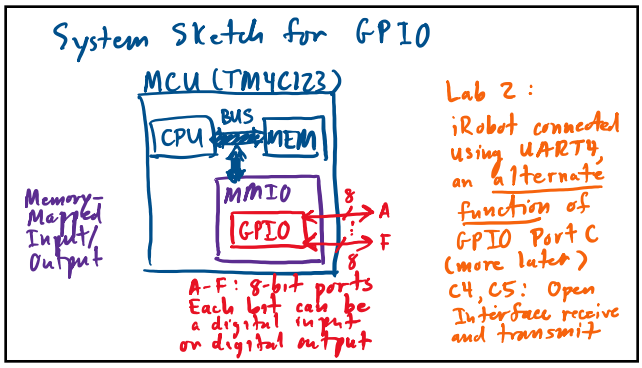
6


System Sketch for GPIO
MCU (TM4C123)
CPU
BUS
MEM
MMIO
GPIO
8
A
F
Memory-Mapped Input/Output
A-F: 8-bit ports
Each bit can be a digital input or digital output

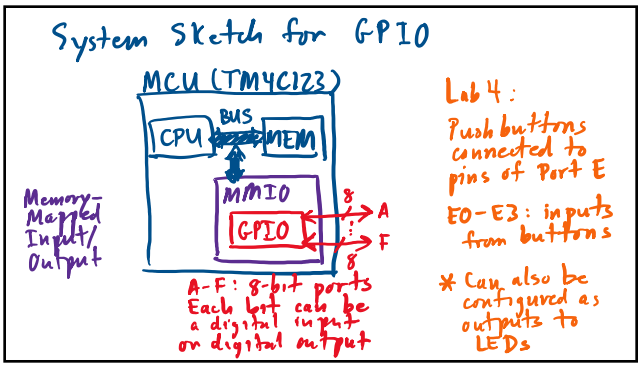
7


System Sketch for GPIO
MCU (TM4C123)
CPU
BUS
MEM
MMIO
GPIO
8
A
F
Memory-Mapped Input/Output
A-F: 8-bit ports
Each bit can be a digital input or digital output
Lab 1:
LCD connected to pins of Port D, Port F
F1-F4: outputs to LCD data lines
D2, D3, D6: outputs to LCD control lines

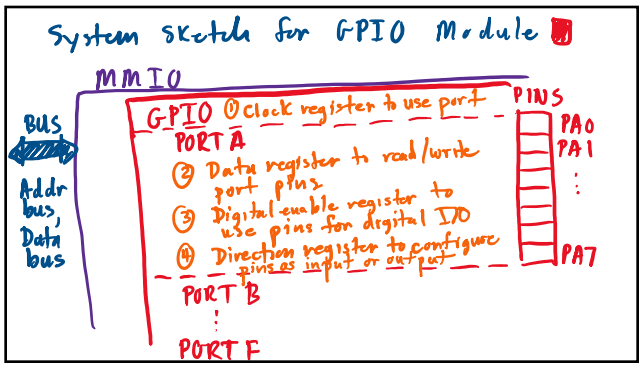
8


System Sketch for GPIO
MCU (TM4C123)
CPU
BUS
MEM
MMIO
GPIO
8
A
F
Memory-Mapped Input/Output
A-F: 8-bit ports
Each bit can be a digital input or digital output
Lab 2:
iRobot connected using UART4, an alternate function of GPIO Port C (more later)
C4, C5: Open Interfaces receive and transmit

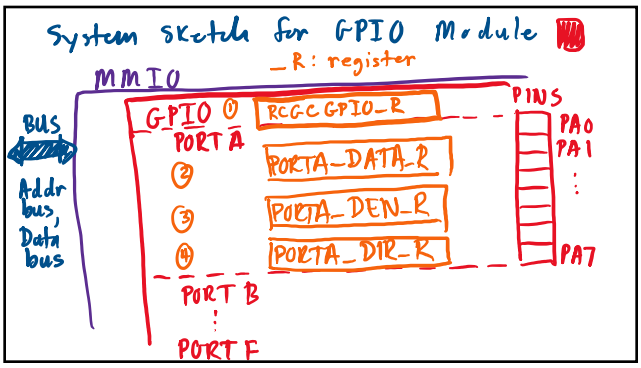
9


System Sketch for GPIO
MCU (TM4C123)
CPU
BUS
MEM
MMIO
GPIO
8
A
F
Memory-Mapped Input/Output
A-F: 8-bit ports
Each bit can be a digital input or digital output
Lab 4:
Push buttons connected to pins of Port E
E0-E3: inputs from buttons
* Can also be configured as outputs to LEDs

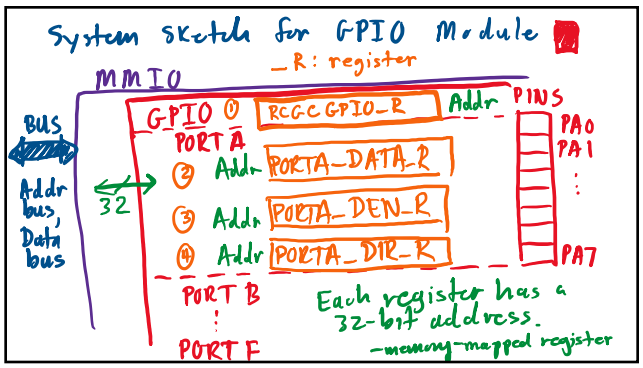
10


System Sketch for GPIO Module
MMIO
BUS
Addr bus, Data bus
GPIO
PORT A
1 Clock register to use port
2 Data register to read/write port pins
3 Digital enable register to use pins for digital I/O
4 Direction register to configure pins as input or output
PORT B
PORT F
PINS
PA0
PA1
PA7

11


System Sketch for GPIO Module
_R: register
MMIO
BUS
Addr bus, Data bus
GPIO
PORT A
1 RCGCGPIO_R
2 PORTA_DATA_R
3 PORTA_DEN_R
4 PORTA_DIR_R
PORT B
PORT F
PINS
PA0
PA1
PA7

12



13

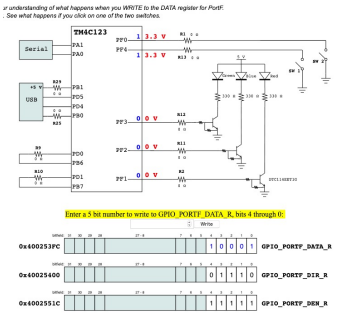
GPIO Port and Registers Example (VY ES book, Ch. 6, Interactive Tool 6.1) [may require Firefox browser]

Notice the GPIO port register names. *

Notice the register addresses. *

* Stay tuned to learn more!

In this example when the switch is closed the wire is connected to ground and results in a low/zero logic value on the input pin.



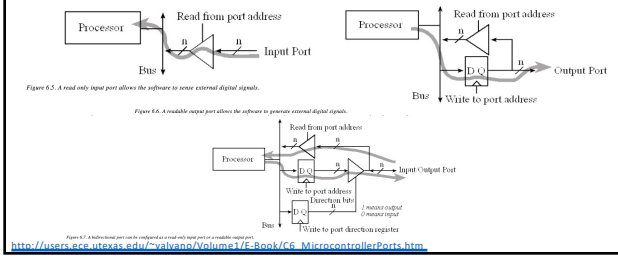
http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C6_MicrocontrollerPorts.htm

14

More about GPIO Port and Registers (VY ES book, Ch. 6)

Common Error: You will get a bus fault if you access a port without enabling its clock.

Section 6.2 Basic Concepts of Input and Output Ports



http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C6_MicrocontrollerPorts.htm

15

More about GPIO Port and Registers (VY ES book, Ch. 6)

Video 6.6. Writing friendly code, https://www.youtube.com/watch?v=0LLD1kLFN-U&feature=youtu.be
Examples of bitwise operations on GPIO registers

6.3 Which line of C code is a friendly way to set Port B bit 2 assuming this pin has already been initialized as an output

```
GPIO_PORTB_DATA_R = 0x00;
GPIO_PORTB_DATA_R = 0x02;
GPIO_PORTB_DATA_R = 0x04;
GPIO_PORTB_DATA_R |= 0x02;
GPIO_PORTB_DATA_R |= 0x04;
GPIO_PORTB_DATA_R &= 0x02;
GPIO_PORTB_DATA_R &= 0x04;
GPIO_PORTB_DATA_R &= ~0x02;
GPIO_PORTB_DATA_R &= ~0x04;
```

6.4 Which line of C code is a friendly way to clear Port B bit 2 assuming this pin has already been initialized as an output

```
GPIO_PORTB_DATA_R = 0x00;
GPIO_PORTB_DATA_R = 0x02;
GPIO_PORTB_DATA_R = 0x04;
GPIO_PORTB_DATA_R |= 0x02;
GPIO_PORTB_DATA_R |= 0x04;
GPIO_PORTB_DATA_R &= 0x02;
GPIO_PORTB_DATA_R &= 0x04;
GPIO_PORTB_DATA_R &= ~0x02;
GPIO_PORTB_DATA_R &= ~0x04;
```

http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C6_MicrocontrollerPorts.htm

16

Resources about GPIO Registers (Keep Calm and …)

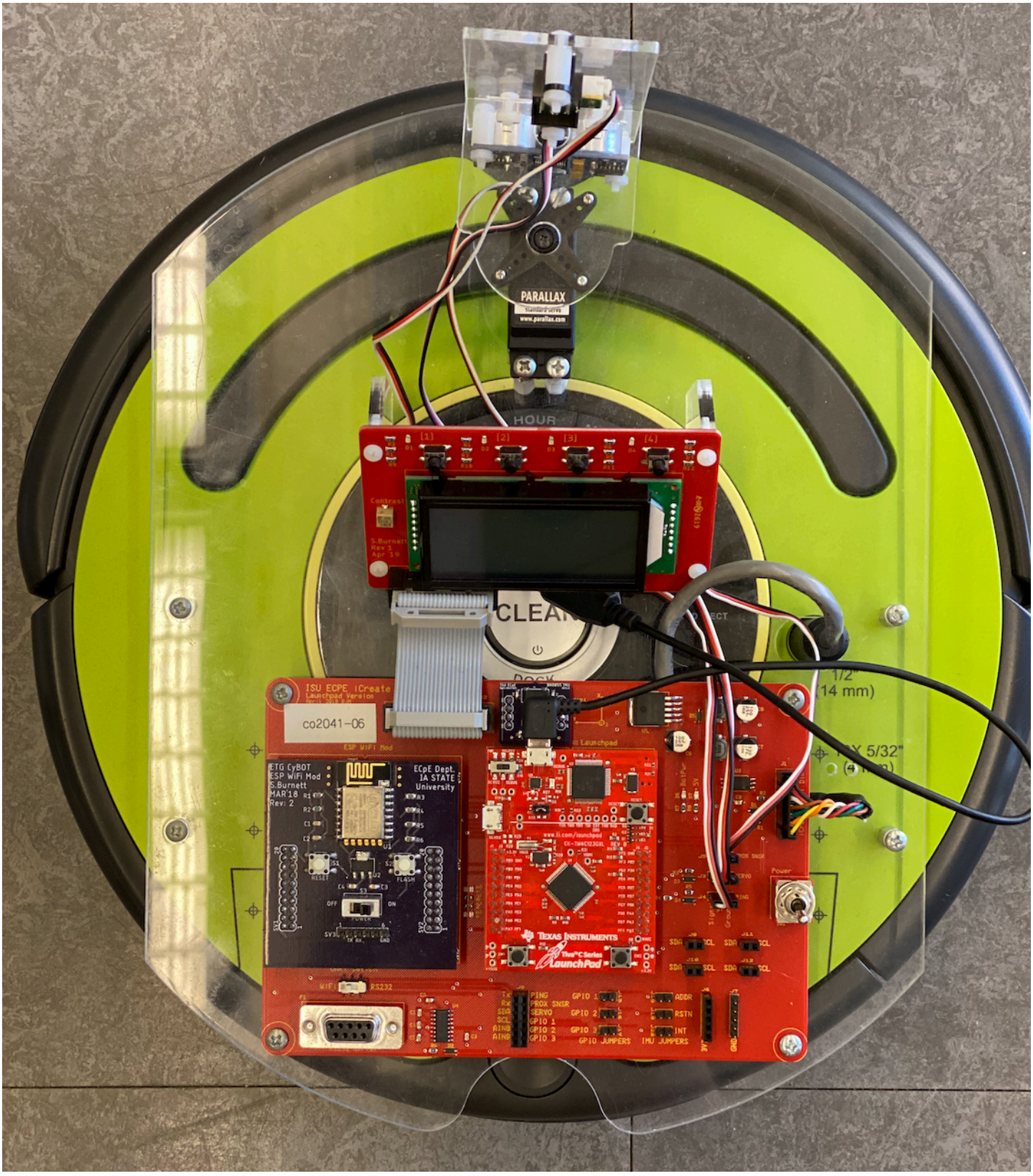
Tiva datasheet: Tiva_TM4C123GH6PM_datasheet.pdf (Lab Resources page etc. in Canvas)
We will learn how to efficiently read/use the datasheet.
VY ES book, Video D6-0: Using the GPIO register datasheet, https://youtu.be/VI2vrRw2Wio
The register names used in your code will be similar but not identical to the datasheet acronyms. Register definition macros for C code are in the tm4c123gh6pm.h header file.

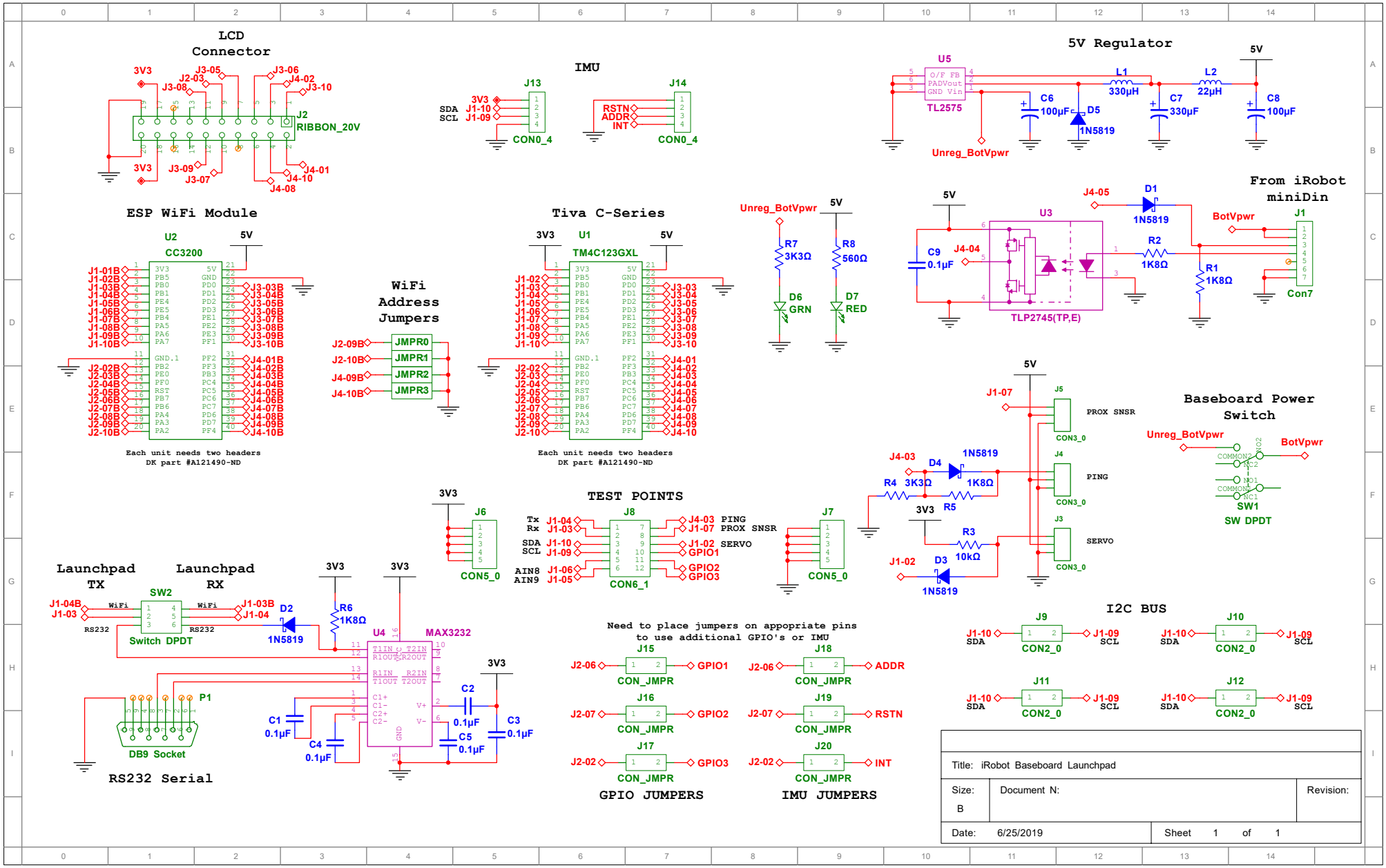
Reading guide for GPIO: reading-guide-GPIO.pdf (Readings page etc. in Canvas)
Refers to specific materials and information in relation to concepts and programming.
Browse first, then select items to peruse more carefully.
There is a section on how to search for registers in the datasheet. It takes practice, and we'll get practice in class and lab.
The Bai book has sections and slides that elaborate on datasheet information.

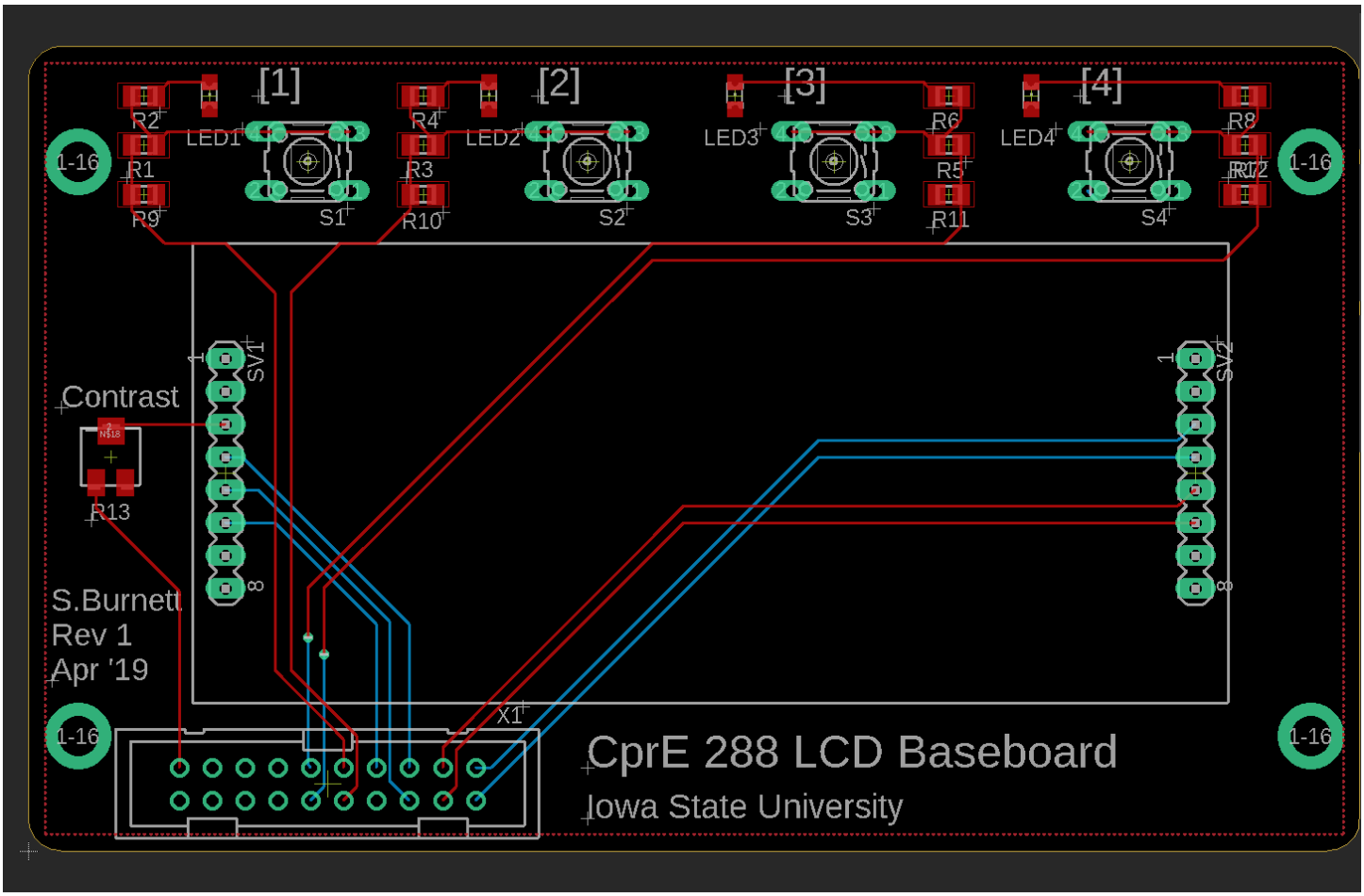
Class meeting slides: GPIO in the filename
Combination of instructor slides and Bai book slides.

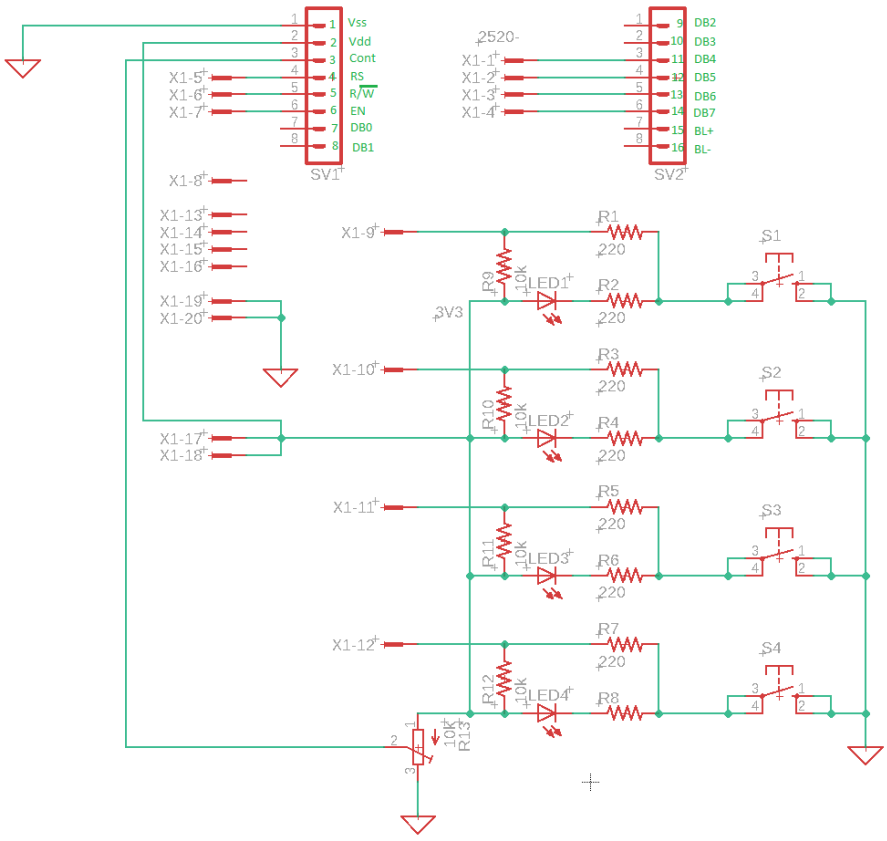
17

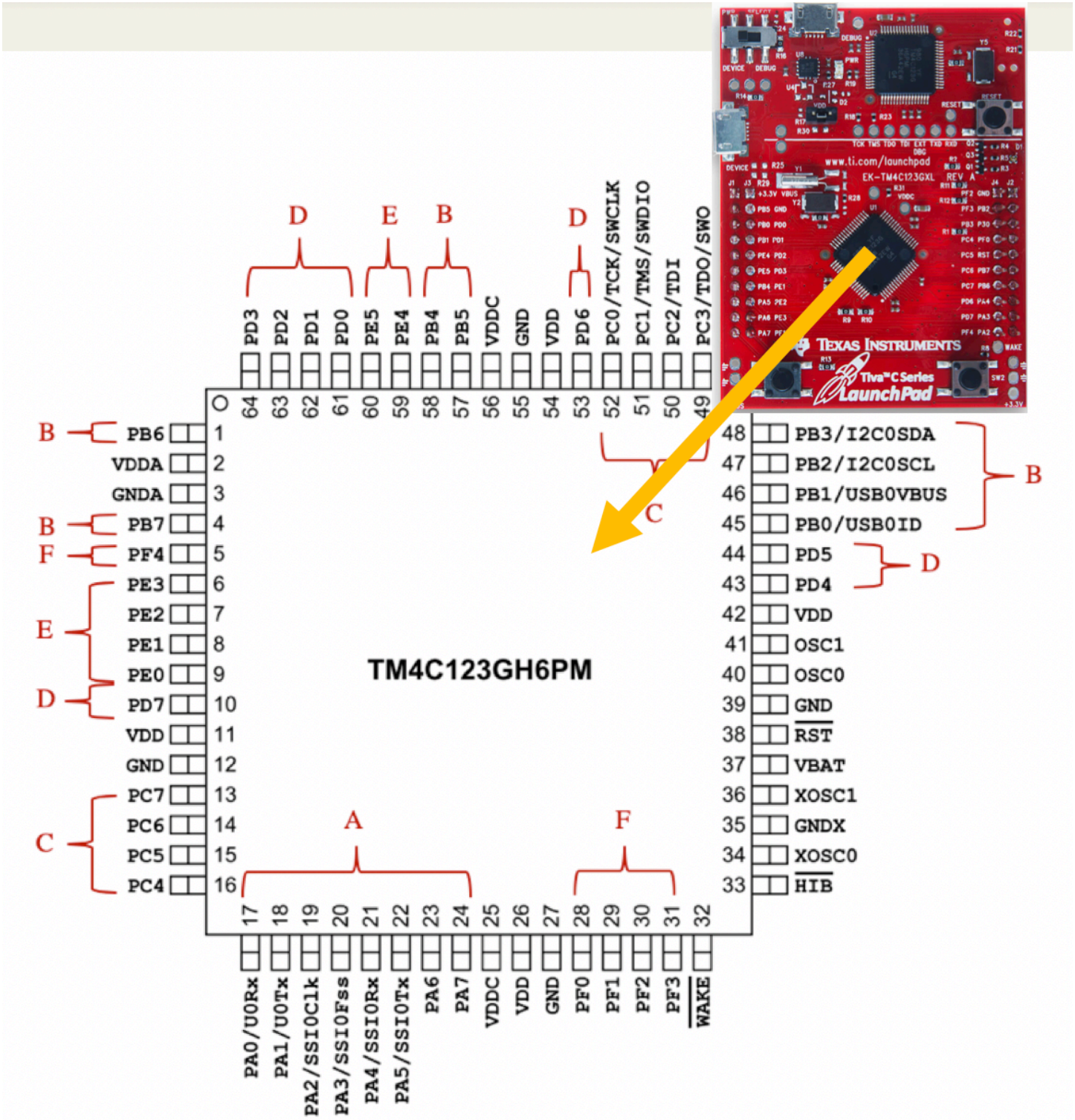
CyBot schematic




LCD Connector
IMU
5V Regulator
ESP WiFi Module
Tiva C-Series
From iRobot miniDin
WiFi Address Jumpers
TEST POINTS
Baseboard Power Switch
Launchpad TX
Launchpad RX
RS232 Serial
I2C BUS
GPIO JUMPERS
IMU JUMPERS
Need to place jumpers on appopriate pins to use additional GPIO's or IMU
Each unit needs two headers
DK part #A121490-ND
Title: iRobot Baseboard Launchpad
Size: B
Document N:
Revision:
Date: 6/25/2019
Sheet 1 of 1


[1]
[2]
[3]
[4]
R2
LED1
R1
R9
S1
R4
LED2
R3
R10
S2
R6
LED3
R5
R11
S3
R8
LED4
R12
S4
1-16
1-16
SV1
SV2
Contrast
R13
S.Burnett
Rev 1
Apr '19
X1
CprE 288 LCD Baseboard
Iowa State University


SV1
Vss
Vdd
Cont
RS
R/W
EN
DB0
DB1
X1-5
X1-6
X1-7
X1-8
2520-
X1-1
X1-2
X1-3
X1-4
SV2
DB2
DB3
DB4
DB5
DB6
DB7
BL+
BL-
X1-13
X1-14
X1-15
X1-16
X1-19
X1-20
X1-9
X1-10
X1-11
X1-12
X1-17
X1-18
3V3
R1
220
R2
220
R3
220
R4
220
R5
220
R6
220
R7
220
R8
220
R9
10k
R10
10k
R11
10k
R12
10k
R13
10k
LED1
LED2
LED3
LED4
S1
S2
S3
S4


TM4C123GH6PM
PB6
VDDA
GNDA
PB7
PF4
PE3
PE2
PE1
PE0
PD7
VDD
GND
PC7
PC6
PC5
PC4
PD3
PD2
PD1
PD0
PE5
PE4
PB4
PB5
VDDC
GND
VDD
PD6
PC0/TCK/SWCLK
PC1/TMS/SWDIO
PC2/TDI
PC3/TDO/SWO
PB3/I2C0SDA
PB2/I2C0SCL
PB1/USB0VBUS
PB0/USB0ID
PD5
PD4
VDD
OSC1
OSC0
GND
RST
VBAT
XOSC1
GNDX
XOSC0
HIB
PA0/U0Rx
PA1/U0Tx
PA2/SSI0Clk
PA3/SSI0Fss
PA4/SSI0Rx
PA5/SSI0Tx
PA6
PA7
VDDC
VDD
GND
PF0
PF1
PF2
PF3
WAKE
A
B
C
D
E
F
www.ti.com/launchpad
EK-TM4C123GXL
Texas Instruments
Tiva™C Series
LaunchPad

GPIO Port A	Function	288 Inputs/Outputs	Stepper Motor	WiFi	Color Key
					PINS W/ RESERVED FUNCTIONS
Pin 7:	LP_I2C1_SCL / IMU SCL				PIN MAPPED TO IMU
Pin 6:	LP_I2C1_SDA / IMU SDA				PIN MAPPED TO TEST POINT
Pin 5:	SSIO CLK				PIN MAPPED TO LP FOR 288
Pin 4:	SSIO Fss				
Pin 3:	SSIO Rx				
Pin 2:	SSIO Tx				
Pin 1:	UART0 Tx				
Pin 0:	UART0 Rx				
GPIO Port B					
Pin 7:	IMU I2C ADDR / GPIO 1	T0CCP1	Motor Coil 4/ Add Parallel LED		
Pin 6:	IMU I2C RSTN / GPIO 2	MOPWM0	Motor Coil 3/ LED		
Pin 5:	SERVO_2		Motor Coil 2/ LED		
Pin 4:	IR1	AIN10	Motor Coil 1/ LED		
Pin 3:	PING_CAP_1	T3CCP1			
Pin 2:	IMU I2C INT / GPIO 3				
Pin 1:	LP_UART1_Tx		Shaft Encoder Channel A		
Pin 0:	LP_UART1_Rx		Shaft Encoder Channel B		
GPIO Port C					
Pin 7:					
Pin 6:					
Pin 5:	OI_UART4_Tx				
Pin 4:	OI_UART4_Rx				
Pin 3:	JTAG TCK/SWCLK				
Pin 2:	JTAG TMS/SWDIO				
Pin 1:	JTAG TDI				
Pin 0:	JTAG TDO/SWO				
GPIO Port D					
Pin 7:	ADDR_JMP_4				
Pin 6:	LCD_RW		LCD_RW		
Pin 3:	LCD_RS		LCD_RS		
Pin 2:	LCD_EN		LCD_EN		
Pin 1:	Connected to other ports through R9 and R10				
Pin 0:					
GPIO Port E	PUSH BUTTONS				
Pin 5:	AIN8		PB_SW6		
Pin 4:	AIN9		PB_SW5		
Pin 3:	PB_SW4		I2C0 SCL / PB_SW4		
Pin 2:	PB_SW3		I2C0 SDA / PB_SW3		
Pin 1:	PB_SW2		PB_SW2		
Pin 0:	PB_SW1		PB_SW1		
GPIO Port F					
Pin 4:	LCD DATA7		LCD DATA7		
Pin 3:	LCD DATA6		LCD DATA6		
Pin 2:	LCD DATA5		LCD DATA5		
Pin 1:	LCD DATA4		LCD DATA4		
Pin 0:	USR_SW2	OI shutoff			

button.c and button.h code

```
/*
 * button.c
 *
 *  Created on: Jul 18, 2016
 *      Author: Eric Middleton, Zhao Zhang, Chad Nelson, & Zachary Glanz.
 *
 *  @edit: Lindsey Sleeth and Sam Stifter on 02/04/2019
 *  @edit: Phillip Jones 05/30/2019: Merged Spring 2019 version with Fall 2018
 *  @edit: Diane Rover 02/01/20: Corrected comments about ordering of switches for
new LCD board and added busy-wait on PRGPIO
 */


// The buttons are on PORTE 3:0
// GPIO_PORTE_DATA_R -- Name of the memory mapped register for GPIO Port E,
// which is connected to the push buttons

#include "button.h"

/**
 * Initialize PORTE and configure bits 0-3 to be used as inputs for the buttons.
 */
void button_init() {
      static uint8_t initialized = 0;

      //Check if already initialized
      if(initialized){
            return;
      }

      // delete warning after implementing
      #warning "Unimplemented function: void button_init()"

      // Reading: To initialize and configure GPIO PORTE, visit pg. 656 in the
      // Tiva datasheet.

      // Follow steps in 10.3 for initialization and configuration. Some steps
      // have been outlined below.

      // Ignore all other steps in initialization and configuration that are not
      // listed below. You will learn more about additional steps in a later lab.

      // 1) Turn on PORTE system clock, do not modify other clock enables
      // SYSCTL_RCGCGPIO_R |=
      // You may need to add a delay here of several clock cycles for the clock to
start, e.g., execute a simple dummy assignment statement, such as "long delay =
SYSCTL_RCGCGPIO_R".
      // Instead, use the PRGPIO register and busy-wait on the peripheral ready bit
for PORTE.
      // while ((SYSCTL_PRGPIO_R & 0x??) == 0) {};
      // 2) Set the buttons as inputs, do not modify other PORTE wires
      // GPIO_PORTE_DIR_R &=

      // 3) Enable digital functionality for button inputs,
      //    do not modify other PORTE enables
      // GPIO_PORTE_DEN_R |=
```

```
	initialized = 1;
}



/**
 * Returns the position of the rightmost button being pushed.
 * @return the position of the rightmost button being pushed.
 * 1 is the leftmost button, 4 is the rightmost button.
 * 0 indicates no button being pressed
 */
uint8_t button_getButton() {

	#warning "Unimplemented function: uint8_t button_getButton(void)" // delete
warning after implementing

	//
	// DELETE ME - How bitmasking works
	// ----------------------------------------
// In embedded programming, often we only care about one or a few bits in
// data.  There are several bitwise operators that we can apply to data in order
// to "mask" the bits that we don't care about.
//
//    | = bitwise OR          & = bitwise AND
//    ^ = bitwise XOR         ~ = bitwise NOT
//    << x = shift left by x bits          >> x = shift right by x bits
//
// Let's say we want to know if push button 3 (Sw3) is pushed.
// Sw3 is bit 2 of GPIO_PORTE_DATA_R.
// Since push buttons are high (1) initially, and low (0) if pushed, PORTE should
// look like:
	// GPIO_PORTE_DATA_R == 0b???? ?0?? if Sw3 is pushed
	// GPIO_PORTE_DATA_R == 0b???? ?1?? if Sw3 is not pushed
	//
	// We want to only look at bit 2 and mask the other 7 bits:
	//
	// Bitwise AND:
	// (GPIO_PORTE_DATA_R & 0b0000 0100) => 0b0000 0000 if Sw3 is pushed
	// (GPIO_PORTE_DATA_R & 0b0000 0100) => 0b0000 0100 if Sw3 is not pushed
	//
	// Bitwise OR:
	// (GPIO_PORTE_DATA_R | 0b1111 1011) => 0b1111 1011 if Sw3 is pushed
	// (GPIO_PORTE_DATA_R | 0b1111 1011) => 0b1111 1111 if Sw3 is not pushed
	//
	// Other techniques (Shifting and bitwise AND)
	// ((GPIO_PORTE_DATA_R >> 2) & 1) => 0 if Sw3 is pushed
	// ((GPIO_PORTE_DATA_R >> 2) & 1) => 1 if Sw3 is not pushed

	// TODO: Write code below -- Return the rightmost button position pressed

	// INSERT CODE HERE!

	return 0; // EDIT ME
}
```

```
/*
 * button.h
 *
 *  Created on: Jul 18, 2016
 *      Author: Eric Middleton
 *
 * @edit: Phillip Jones 05/30/2019 : Removed uneeded helper functions
 */

#ifndef BUTTON_H_
#define BUTTON_H_

#include <stdint.h>
#include <inc/tm4c123gh6pm.h>

//initialize the push buttons
void button_init();


///Non-blocking call
///Returns highest value button being pressed, 0 if no button pressed
uint8_t button_getButton();


#endif /* BUTTON_H_ */
```

Tiva™ TM4C123GH6PM Microcontroller

DATA SHEET

DS-TM4C123GH6PM-15842.2741
SPMS376E

Copyright © 2007-2014
Texas Instruments Incorporated

EXCERPTS FROM THE HEADER FILE tm4c123gh6pm.h

```
/*****************************************************************
**********************
//
// tm4c123gh6pm.h - TM4C123GH6PM Register Definitions
//
// Copyright (c) 2013 Texas Instruments Incorporated.  All rights reserved.
// Software License Agreement
//
…
//*****************************************************************************
//
// GPIO registers (PORTA)
//
//*****************************************************************************
#define GPIO_PORTA_DATA_BITS_R  ((volatile unsigned long *)0x40004000)
#define GPIO_PORTA_DATA_R       (*((volatile unsigned long *)0x400043FC))
#define GPIO_PORTA_DIR_R        (*((volatile unsigned long *)0x40004400))
#define GPIO_PORTA_IS_R         (*((volatile unsigned long *)0x40004404))
#define GPIO_PORTA_IBE_R        (*((volatile unsigned long *)0x40004408))
#define GPIO_PORTA_IEV_R        (*((volatile unsigned long *)0x4000440C))
#define GPIO_PORTA_IM_R         (*((volatile unsigned long *)0x40004410))
#define GPIO_PORTA_RIS_R        (*((volatile unsigned long *)0x40004414))
#define GPIO_PORTA_MIS_R        (*((volatile unsigned long *)0x40004418))
#define GPIO_PORTA_ICR_R        (*((volatile unsigned long *)0x4000441C))
#define GPIO_PORTA_AFSEL_R      (*((volatile unsigned long *)0x40004420))
#define GPIO_PORTA_DR2R_R       (*((volatile unsigned long *)0x40004500))
#define GPIO_PORTA_DR4R_R       (*((volatile unsigned long *)0x40004504))
#define GPIO_PORTA_DR8R_R       (*((volatile unsigned long *)0x40004508))
#define GPIO_PORTA_ODR_R        (*((volatile unsigned long *)0x4000450C))
#define GPIO_PORTA_PUR_R        (*((volatile unsigned long *)0x40004510))
#define GPIO_PORTA_PDR_R        (*((volatile unsigned long *)0x40004514))
#define GPIO_PORTA_SLR_R        (*((volatile unsigned long *)0x40004518))
#define GPIO_PORTA_DEN_R        (*((volatile unsigned long *)0x4000451C))
#define GPIO_PORTA_LOCK_R       (*((volatile unsigned long *)0x40004520))
#define GPIO_PORTA_CR_R         (*((volatile unsigned long *)0x40004524))
#define GPIO_PORTA_AMSEL_R      (*((volatile unsigned long *)0x40004528))
#define GPIO_PORTA_PCTL_R       (*((volatile unsigned long *)0x4000452C))
#define GPIO_PORTA_ADCCTL_R     (*((volatile unsigned long *)0x40004530))
#define GPIO_PORTA_DMACTL_R     (*((volatile unsigned long *)0x40004534))

…

//*****************************************************************************
//
// System Control registers (SYSCTL)
//
//*****************************************************************************
…
#define SYSCTL_RCGCGPIO_R       (*((volatile unsigned long *)0x400FE608))

…

#define SYSCTL_PRGPIO_R         (*((volatile unsigned long *)0x400FEA08))

…
```

```
//*****************************************************************************
//
// The following are defines for the bit fields in the SYSCTL_RCGCGPIO
// register.
//
//*****************************************************************************
#define SYSCTL_RCGCGPIO_R5      0x00000020  // GPIO Port F Run Mode Clock
                                            // Gating Control
#define SYSCTL_RCGCGPIO_R4      0x00000010  // GPIO Port E Run Mode Clock
                                            // Gating Control
#define SYSCTL_RCGCGPIO_R3      0x00000008  // GPIO Port D Run Mode Clock
                                            // Gating Control
#define SYSCTL_RCGCGPIO_R2      0x00000004  // GPIO Port C Run Mode Clock
                                            // Gating Control
#define SYSCTL_RCGCGPIO_R1      0x00000002  // GPIO Port B Run Mode Clock
                                            // Gating Control
#define SYSCTL_RCGCGPIO_R0      0x00000001  // GPIO Port A Run Mode Clock
                                            // Gating Control

…

//*****************************************************************************
//
// The following are defines for the bit fields in the SYSCTL_PRGPIO register.
//
//*****************************************************************************
#define SYSCTL_PRGPIO_R5        0x00000020  // GPIO Port F Peripheral Ready
#define SYSCTL_PRGPIO_R4        0x00000010  // GPIO Port E Peripheral Ready
#define SYSCTL_PRGPIO_R3        0x00000008  // GPIO Port D Peripheral Ready
#define SYSCTL_PRGPIO_R2        0x00000004  // GPIO Port C Peripheral Ready
#define SYSCTL_PRGPIO_R1        0x00000002  // GPIO Port B Peripheral Ready
#define SYSCTL_PRGPIO_R0        0x00000001  // GPIO Port A Peripheral Ready

…
```

GPIO-registers.pdf

GPIO Registers

List of Several TM4C Registers by Name (DATASHEET ACRONYM): C MACRO NAME

See also **tm4c123gh6m.h system header file**

GPIO Data (GPIODATA): GPIO_PORTx_DATA_R
GPIO Direction (GPIODIR): GPIO_PORTx_DIR_R
GPIO Digital Enable (GPIODEN): GPIO_PORTx_DEN_R
GPIO Alternate Function Select (GPIOAFSEL): GPIO_PORTx_AFSEL_R
GPIO Port Control (GPIOPCTL): GPIO_PORTx_PCTL_R
GPIO Analog Mode Select (GPIOAMSEL): GPIO_PORTx_AMSEL_R
GPIO Run Mode Clock Gating Control (RCGCGPIO): SYSCTL_RCGCGPIO_R
GPIO Peripheral Ready (PRGPIO): SYSCTL_PRGPIO_R

GPIOIS Interrupt Sense (GPIOIS): GPIO_PORTx_IS_R
GPIOIBE Interrupt Both Edges (GPIOIBE): GPIO_PORTx_IBE_R
GPIOIEV Interrupt Event (GPIOIEV): GPIO_PORTx_IEV_R
GPIOIM Interrupt Mask (GPIOIM): GPIO_PORTx_IM_R
GPIORIS Raw Interrupt Status (GPIORIS): GPIO_PORTx_RIS_R
GPIOMIS Masked Interrupt Status (GPIOMIS): GPIO_PORTx_MIS_R
GPIOICR Interrupt Clear (GPIOICR): GPIO_PORTx_ICR_R

LAB 4 Lab Manual (Part 1)

PUSH BUTTON DIGITAL INPUTS AND CYBOT-PC COMMUNICATIONS

INTRODUCTION

This week you will program push button messages using general-purpose input/output pins (GPIO), bitwise operations, and polling. The GPIO pins are grouped into sets of 8 pins called ports. A port is accessed with a data register, and the data register is accessed using a memory address – this is called memory-mapped input/output. A programmer must initialize a port before using it for input/output. Initialization is done using a set of configuration registers associated with a GPIO port. You will use relevant concepts and skills to develop functionality that allows a user to interact with a running program on the CyBot through push buttons, the LCD screen, and other communications.

Read each step in this manual completely before attempting it so that you do not miss out on important details.

REFERENCE FILES

The following reference files will be used in this lab:

- button.c, contains functions that you will implement for this lab and use in future labs
- button.h, header file for button.c
- lab4_template.c, contains a main function template that you will implement for this lab
- cyBot_uart.h, header file for pre-compiled CyBot-PC UART communication library
- libcybotUART.lib: pre-compiled library for CyBot-PC UART communication (note: must change extension of file from .txt to .lib after copying)
- lcd.c, program file containing various LCD functions
- lcd.h, header file for lcd.c
- timer.c, program file containing various wait commands
- timer.h, header file for timer.c
- open_interface.c, API functions for basic Open Interface functions
- open_interface.h, header file for open_interface.c
- TI Tiva TM4C123G Microcontroller Datasheet
- TI TM4C123G Register Definitions C header file: tm4c123gh6pm.h
- Cybot baseboard and LCD schematics: Cybot-Baseboard-LCD-Schematic.pdf
- GPIO register list: GPIO-registers.pdf
- GPIO Reading Guide: reading-guide-GPIO.pdf **← BROWSE THROUGH THIS**

The code files are available to download.

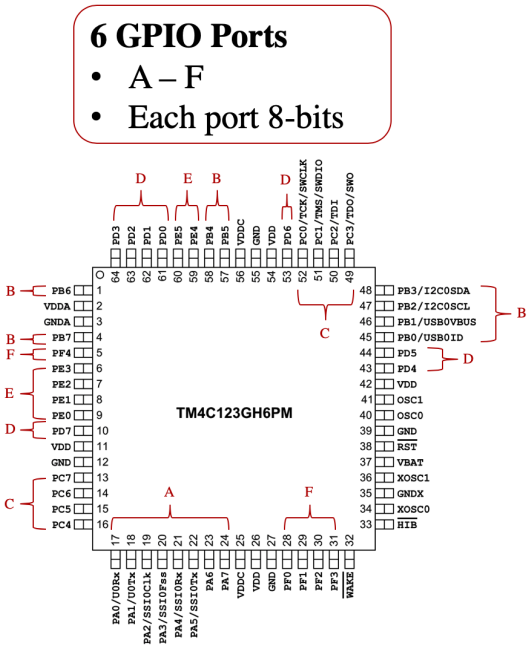
Being familiar with the GPIO reading guide may help you find specific information in various resources. This lab manual also cites specific resources and contents. **Your goal is to be aware of information so that you can find and read it in more detail if and when needed. You are not expected to read or know everything at once.** Take it one step at a time. Be patient but persistent and purposeful. What do you need to know? What do you know that you can build on? What don't you know that might take more effort? Where or how might you get started? (Arthur Ashe: "Start where you are. Use what you have. Do what you can.")

PRELAB

See the prelab assignment in Canvas and submit it prior to the start of lab.

STRUCTURED PAIRING

You are expected to continue to use structured pairing in this lab and in future labs. It was introduced in Lab 2.



PART 1: READING PUSH BUTTONS

Write a short program in main using a polling method (continuously checking the status of a device in software using a loop) that repeatedly calls the function **button_getButton()** to check if a button has been pushed and displays the result on the LCD screen.

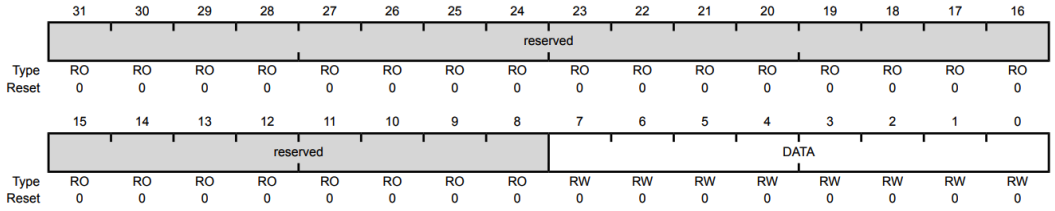
To begin, you will need to complete an API function for reading the push buttons on the LCD board by initializing the GPIO port in the function **button_init()** and finishing the **button_getButton()** function in button.c. Some template code has been provided for you in the button.c file.

Chapter 10 (pg. 649) in the Tiva datasheet provides details on using GPIO ports on the microcontroller. Remember that the microcontroller contains 6 GPIO ports (A-F), which each contain 8 bits. Each port has a set of registers associated with it, and a full list of registers can be seen in table **10-6. GPIO Register Map** on pg. 660.

You will be using GPIO Port E to program the buttons on the LCD board. You will need to use your knowledge of bitwise operations to mask bits in the data register and detect which button has been pressed.

You will find section **10.3 Initialization and Configuration** in the datasheet helpful for initializing and configuring the GPIO port pins. Initialization, or configuration, of the GPIO pins is necessary in order to use the GPIO port and to read the status of a push button, i.e., pressed or not (note: buttons are active low, meaning that pressing a button results in a low or logic 0 signal on the digital input).

Page 662 in the Tiva datasheet corresponds to the **GPIODATA** register. It is recommended that you read the register description as you will need to read the GPIODATA register in order to determine which button has been pressed. **Pins 0 – 3 of GPIO Port E correspond to buttons 1 – 4 on the LCD board.**



mtg slides GPIO

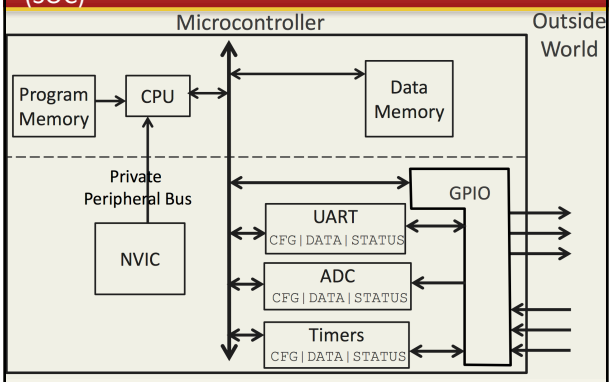
CprE 288 – Introduction to Embedded Systems
(memory-mapped I/O, I/O ports, GPIO)

Instructor:
Dr. Diane Rover

1

1

Microcontroller (TM4C123) – System-on-Chip (SOC)



2

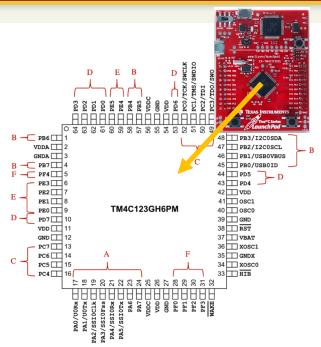
2

Microcontroller (TM4C123) – Chip Package

64 package pins allow software to access the world outside of the chip

6 GPIO Ports
- A – F
- Each port 8-bits

Many GPIOs have alternate functions



3

3

Memory Mapped I/O

4

4

Memory-Mapped General Purpose I/O (GPIO)

TM4C123GH6PM
- 6 general purpose I/O ports: Ports A, B, C, D, E, F
- Processor communicates with GPIO ports through memory-mapped registers.
- Set of data and control/status registers associated with each port.
  - Configuration registers
  - Data registers
  - Status registers

5

5

GPIO Module and Ports

Bai book

GPIO Module Architecture and GPIO Port Configuration

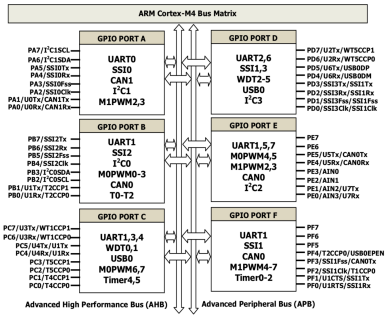


Figure 7.1 GPIO Port and pin configuration.

6

6

Memory-Mapped General Purpose I/O (GPIO)

Each GPIO port has several registers, we will focus on 5.

SYSCTL_RCGCGPIO_R – Clock control register - enables the port's system clock

SYSCTL_PRGPIO_R – Peripheral ready status register - indicates if the port is ready (after enabling)

GPIO_PORTx_DATA_R – 8-bit data register (read/write)

GPIO_PORTx_DIR_R – Data direction register

GPIO_PORTx_DEN_R – Digital enable register

7

7

TM4C123 Memory Map (Bai book)

2.6.2 TM4C123GH6PM Microcontroller On-Chip Memory Map

Table 2.5 Memory maps for TM4C123GH6PM MCU.

Start Address	End Address	Descriptions
256 KB On-Chip Flash ROM		
0x0000.0000	0x0000.003C	Exceptions Vector Table
0x0000.0040	0x0000.00AB	Interrupts Vector Table
0x0000.00B0	0x0000.02AB	System Controls
0x0002.0000	0x0003.FFFF	User Codes
Internal ROM		
0x0100.0000	0x01FF.FFFF	Tiva Driver Libraries, Boot Loader, AE
32KB SRAM		
0x2000.0000	0x2000.7FFF	User Data
0x2200.0000	0x23FF.FFFF	Bit-band alias of bit-banded on-chip S
Peripherals		
0x4000.0000	0x4000.0FFF	Watchdog Timer 0
0x4000.1000	0x4000.1FFF	Watchdog Timer 1
0x4000.4000	0x4000.4FFF	GPIO Port A (APB Aperture)
0x4000.5000	0x4000.5FFF	GPIO Port B (APB Aperture)
0x4000.6000	0x4000.6FFF	GPIO Port C (APB Aperture)
0x4000.7000	0x4000.7FFF	GPIO Port D (APB Aperture)
0x4000.8000	0x4000.8FFF	SSI0 ~ SSI3
0x4000.C000	0x4001.3FFF	UART0 ~ UART7
0x4002.0000	0x4002.3FFF	I²C0 ~ I²C3
0x4002.4000	0x4002.4FFF	GPIO Port E (APB Aperture)
0x4002.5000	0x4002.5FFF	GPIO Port F (APB Aperture)
0x4002.8000	0x4002.9FFF	PWM0 ~ PWM1
0x4002.C000	0x4002.DFFF	QEI0 ~ QEI1
0x4003.0000	0x4003.5FFF	16/32-Bit Timer 0 ~ 16/32-Bit Timer 5
0x4003.6000	0x4003.7FFF	32/64-Bit Timer 0 ~ 32/64-Bit Timer 1
0x4003.8000	0x4003.8FFF	ADC0
0x4003.9000	0x4003.9FFF	ADC1
0x4003.C000	0x4003.CFFF	Analog Comparators
0x4004.0000	0x4004.1FFF	CAN0 ~ CAN1 Controllers
0x4004.C000	0x4005.1FFF	32/64-Bit Timer 2 ~ 32/64-Bit Timer 5
0x4005.0000	0x4005.0FFF	USB
0x4005.8000	0x4005.DFFF	GPIO Port A ~ Port F (AHB Aperture)
0x400A.F000	0x400A.FFFF	2KB EEPROM and Key locker
0x400F.9000	0x400F.9FFF	System Exception Module
0x400F.C000	0x400F.C7FF	Hibernation Module
0x400F.D000	0x400F.DFFF	Flash Memory Control
0x400F.E000	0x400F.EFFF	System Control
0x400F.F000	0x400F.FFFF	µDMA
0x4200.0000	0x43FF.FFFF	Bit-banded alias of 0x4000.0000 through 0x400F.FFFF

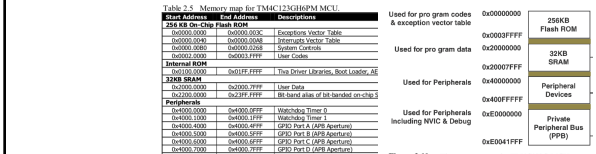


Figure 2.10. The memory map for TM4C123G MCU.

8

8

GPIO Port Registers (Bai book)

2.6.3.4 The Initialization and Configuration of GPIO Ports

- Each GPIO Port (Ports A ~ F) contains a set of all GPIO Registers discussed above and each register can be accessed by using an offset address combining (plus) with a base address.
- For example, the Port A in the APB bus aperture has a base address 0x4000.4000. Its corresponding registers have different associate offset addresses. Table 2.7 shows a relationship between the base address, offset address and full address for some Port A registers.

Table 2.7 Some popular GPIO Port A registers in APB bus aperture.

GPIO Register	Base Address	Offset	Full Address	SW Symbolic Definition
GPIO Port A Data Register	0x4000.4000	0x000	0x4000.4000	GPIO_PORTA_DATA_R
GPIO Port A Direction Register	0x4000.4000	0x400	0x4000.4400	GPIO_PORTA_DIR_R
GPIO Port A AFSEL Register	0x4000.4000	0x420	0x4000.4420	GPIO_PORTA_AFSEL_R
GPIO Port A IS Register	0x4000.4000	0x404	0x4000.4404	GPIO_PORTA_IS_R
GPIO Port A ODR Register	0x4000.4000	0x50C	0x4000.450C	GPIO_PORTA_ODR_R
GPIO Port A PUR Register	0x4000.4000	0x510	0x4000.4510	GPIO_PORTA_PUR_R
GPIO Port A PDR Register	0x4000.4000	0x514	0x4000.4514	GPIO_PORTA_PDR_R
GPIO Port A DEN Register	0x4000.4000	0x51C	0x4000.451C	GPIO_PORTA_DEN_R
GPIO Port A LOCK Register	0x4000.4000	0x520	0x4000.4520	GPIO_PORTA_LOCK_R
GPIO Port A CR Register	0x4000.4000	0x524	0x4000.4524	GPIO_PORTA_CR_R

9

9

GPIO Register Symbolic Definitions

```
// tm4c123gh6pm.h
//*****************************************************************************
// GPIO registers (PORTA)
//*****************************************************************************
#define GPIO_PORTA_DATA_BITS_R  ((volatile unsigned long *)0x40004000)
#define GPIO_PORTA_DATA_R       (*((volatile unsigned long *)0x400043FC))
#define GPIO_PORTA_DIR_R        (*((volatile unsigned long *)0x40004400))
#define GPIO_PORTA_IS_R         (*((volatile unsigned long *)0x40004404))
#define GPIO_PORTA_IBE_R        (*((volatile unsigned long *)0x40004408))
#define GPIO_PORTA_IEV_R        (*((volatile unsigned long *)0x4000440C))
#define GPIO_PORTA_IM_R         (*((volatile unsigned long *)0x40004410))
#define GPIO_PORTA_RIS_R        (*((volatile unsigned long *)0x40004414))
#define GPIO_PORTA_MIS_R        (*((volatile unsigned long *)0x40004418))
#define GPIO_PORTA_ICR_R        (*((volatile unsigned long *)0x4000441C))
#define GPIO_PORTA_AFSEL_R      (*((volatile unsigned long *)0x40004420))
#define GPIO_PORTA_DR2R_R       (*((volatile unsigned long *)0x40004500))
#define GPIO_PORTA_DR4R_R       (*((volatile unsigned long *)0x40004504))
#define GPIO_PORTA_DR8R_R       (*((volatile unsigned long *)0x40004508))
#define GPIO_PORTA_ODR_R        (*((volatile unsigned long *)0x4000450C))
#define GPIO_PORTA_PUR_R        (*((volatile unsigned long *)0x40004510))
#define GPIO_PORTA_PDR_R        (*((volatile unsigned long *)0x40004514))
#define GPIO_PORTA_SLR_R        (*((volatile unsigned long *)0x40004518))
#define GPIO_PORTA_DEN_R        (*((volatile unsigned long *)0x4000451C))
#define GPIO_PORTA_LOCK_R       (*((volatile unsigned long *)0x40004520))
#define GPIO_PORTA_CR_R         (*((volatile unsigned long *)0x40004524))
#define GPIO_PORTA_AMSEL_R      (*((volatile unsigned long *)0x40004528))
#define GPIO_PORTA_PCTL_R       (*((volatile unsigned long *)0x4000452C))
#define GPIO_PORTA_ADCCTL_R     (*((volatile unsigned long *)0x40004530))
#define GPIO_PORTA_DMACTL_R     (*((volatile unsigned long *)0x40004534))
```

10

10

Register Definition and Volatile Keyword (Bai book)

4.5.6.5 Naming Convention and Definition

- To use these registers, you should declare them by using the **#define** statement. In fact, you need to define each register in your C codes based on its address.
- Since the address in C code can be mapped a pointer, you can define each register by using a pointer points to this register.
- One easy way to do this definition is to use **#define** statement to define each register one by one. A better way is to define a structure for a port that contains all registers. An example of this definition for Port A Data register is:
  - **#define GPIO_PORTA_DATA_R (* ((volatile uint32_t *) 0x40004000))**
- It looks like that this is a double pointer definition. The **(volatile uint32_t *)** indicates that this is a pointer or address with the volatile **uint32_t** data type, and the keyword **volatile** means that the data written into this address can be modified, which is opposite to the keyword **const**.

11

11

GPIO Port Initialization (Bai book)

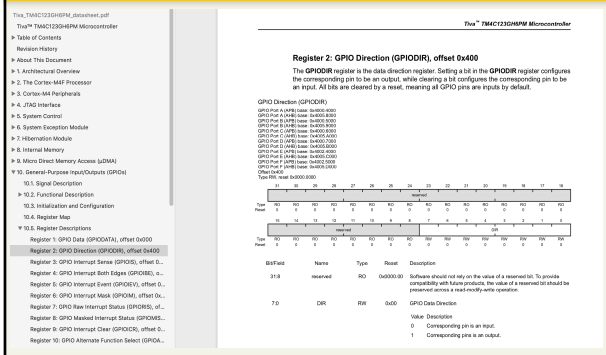
2.6.3.4 The Initialization and Configuration of GPIO Ports

- Perform the following operations to initialize a specific GPIO Port:
  - Setup the clock for the desired GPIO Port with **RCGCGPIO** Register.
  - Setup the direction for each pin on the GPIO port by programming the **GPIODIR** register (0: Input, 1: Output).
  - Optionally you can configure the **GPIOAFSEL** register to program each bit as a GPIO Mode or Alternate Mode. If an alternate pin is chosen for a bit, then the PMCx field must be programmed in the GPIOPCTL register for the specific peripheral function.
  - Enable GPIO pins as digital I/Os by setting the appropriate DEN bit in the **GPIODEN** register. Since all GPIO pins are tri-state status after a reset, this step is necessary. To enable GPIO pins to their analog function, set the GPIOAMSEL bit in the **GPIOAMSEL** register.

12

12

Tiva Datasheet Register Descriptions



13

13

Tiva Datasheet Register Descriptions

In section 10.5, Register Descriptions, only read about the following important registers. The register macro names shown below are defined for our platform in the tm4c123gh6pm.h header file, where X denotes the ports A-F. Keep in mind our platform uses the APB bus addresses.

i) Register 1: GPIO Data (GPIO_PORTX_DATA_R)
ii) Register 2: GPIO Direction (GPIO_PORTX_DIR_R)
iii) Register 10: GPIO Alternate Function Select (GPIO_PORTX_AFSEL_R)
iv) Register 18: GPIO Digital Enable (GPIO_PORTX_DEN_R)
v) Register 21: GPIO Analog Mode Select (GPIO_PORTX_AMSEL_R)
vi) Register 22: GPIO Port Control (GPIO_PORTA_PCTL_R)

In section 5.5, Registers Descriptions

i) Register 60: General-Purpose Input/Output Run Mode Clock Gating Control (SYSCTL_RCGCGPIO_R)

Plus GPIO Peripheral Ready (SYSCTL_PRGPIO_R) …

14

14

Tiva Datasheet

Register 108: General-Purpose Input/Output Peripheral Ready (PRGPIO), offset 0xA08

The **PRGPIO** register indicates whether the GPIO modules are ready to be accessed by software following a change in status of power, Run mode clocking, or reset. A Run mode clocking change is initiated if the corresponding **RCGCGPIO** bit is changed. A reset change is initiated if the

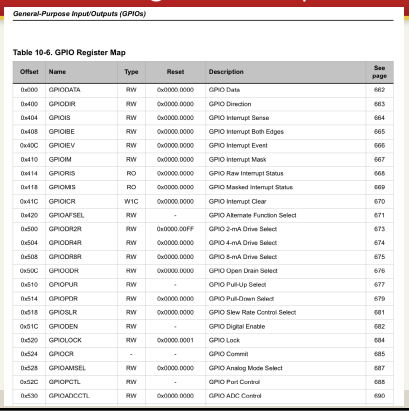
5.2.6 System Control

For power-savings purposes, the peripheral-specific **RCGCx**, **SCGCx**, and **DCGCx** registers (for example, **RCGCWD**) control the clock gating logic for that peripheral or block in the system while the microcontroller is in Run, Sleep, and Deep-Sleep mode, respectively. These registers are located in the System Control register map starting at offsets 0x600, 0x700, and 0x800, respectively. There must be a delay of 3 system clocks after a peripheral module clock is enabled in the **RCGC** register before any module registers are accessed.

15

15

Tiva Datasheet Register Descriptions



General-Purpose Input/Outputs (GPIOs)

Table 10-6. GPIO Register Map

Offset	Name	Type	Reset	Description	See page
0x000	GPIODATA	RW	0x0000.0000	GPIO Data	662
0x400	GPIODIR	RW	0x0000.0000	GPIO Direction	663
0x404	GPIOIS	RW	0x0000.0000	GPIO Interrupt Sense	664
0x408	GPIOIBE	RW	0x0000.0000	GPIO Interrupt Both Edges	665
0x40C	GPIOIEV	RW	0x0000.0000	GPIO Interrupt Event	666
0x410	GPIOIM	RW	0x0000.0000	GPIO Interrupt Mask	667
0x414	GPIORIS	RO	0x0000.0000	GPIO Raw Interrupt Status	668
0x418	GPIOMIS	RO	0x0000.0000	GPIO Masked Interrupt Status	669
0x41C	GPIOICR	W1C	0x0000.0000	GPIO Interrupt Clear	670
0x420	GPIOAFSEL	RW	-	GPIO Alternate Function Select	671
0x500	GPIODR2R	RW	0x0000.00FF	GPIO 2-mA Drive Select	673
0x504	GPIODR4R	RW	0x0000.0000	GPIO 4-mA Drive Select	674
0x508	GPIODR8R	RW	0x0000.0000	GPIO 8-mA Drive Select	675
0x50C	GPIOODR	RW	0x0000.0000	GPIO Open Drain Select	676
0x510	GPIOPUR	RW	-	GPIO Pull-Up Select	677
0x514	GPIOPDR	RW	0x0000.0000	GPIO Pull-Down Select	679
0x518	GPIOSLR	RW	0x0000.0000	GPIO Slew Rate Control Select	681
0x51C	GPIODEN	RW	-	GPIO Digital Enable	682
0x520	GPIOLOCK	RW	0x0000.0001	GPIO Lock	684
0x524	GPIOCR	-	-	GPIO Commit	685
0x528	GPIOAMSEL	RW	0x0000.0000	GPIO Analog Mode Select	687
0x52C	GPIOPCTL	RW	-	GPIO Port Control	688
0x530	GPIOADCCTL	RW	0x0000.0000	GPIO ADC Control	690

16

16

Memory-Mapped General Purpose I/O (GPIO)

Each GPIO port has several registers, we will focus on 5.

SYSCTL_RCGCGPIO_R – Clock control register - enables the port's system clock

SYSCTL_PRGPIO_R – Peripheral ready status register - indicates if the port is ready (after enabling)

GPIO_PORTx_DATA_R – 8-bit data register (read/write)

GPIO_PORTx_DIR_R – Data direction register

GPIO_PORTx_DEN_R – Digital enable register

17

17

GPIO RCGC and Peripheral Ready

SYSCTL_RCGCGPIO_R (GPIO Run Mode Clock Gating Control)

- A GPIO port's clock must be enabled to use its registers.
- bits 0-5 in SYSCTL_RCGCGPIO_R represent ports A-F.

Your program will crash if you attempt to update GPIO port registers without enabling its clock.

Example:

```
//Enable clock for Port F
SYSCTL_RCGCGPIO_R |= 0b100000;
//Wait for Port F to be ready
while ((SYSCTL_PRGPIO_R & 0x20) == 0) {};
```

18

18

GPIO Direction Register

GPIO_PORTn_DIR_R (GPIO Direction Register)
- Writing '0' to a bit programs the corresponding pin of the port as input
- Writing '1' to a bit programs the corresponding pin of the port as output

Example:

```
//All bits of port A used for input except bit0
GPIO_PORTA_DIR_R = 0b00000001;
```

19

19

GPIO Digital Enable Register

GPIO_PORTn_DEN_R (GPIO Digital Enable Register)
- Writing '0' to a bit disables the digital input/output
- Writing '1' to a bit enables digital functionality

Example:

```
//All bits of port A enabled as digital pins
GPIO_PORTA_DEN_R = 0b11111111;
```

20

20

GPIO Data Register

GPIO_PORTn_DATA_R (GPIO Data Register)

For output configured port: If a bit in in the port's DATA register is written when the corresponding pin is configured as an output, then the port's pin will be driven with this value.

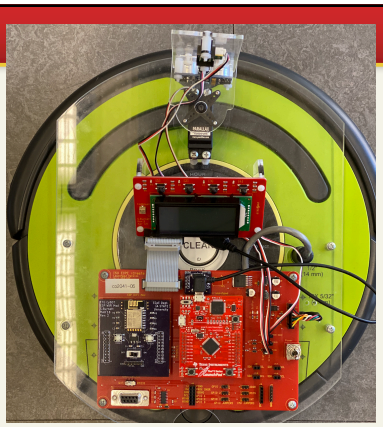
Write to a port using its DATA register.

Example:

```
SYSCTL_RCGCGPIO_R |= 0b000001; //enable Port A clock
while ((SYSCTL_PRGPIO_R & 0x01) == 0) {}; // wait
GPIO_PORTA_DIR_R = 0xFF; //set Port A dir to output
GPIO_PORTA_DEN_R = 0xFF; //enable pins 0-7
GPIO_PORTA_DATA_R = my_char;//write my_char to port
```

21

21



22

22

Example: Initialize Push Buttons

Push button port connection
- Port E, pin 0 to pin 3 (button SW1 to SW4)
- Configure push button connections to be inputs

```
/// Initialize PORTE to set up 4 push buttons as inputs
void init_push_buttons(void) {
  SYSCTL_RCGCGPIO_R |= 0b010000; //Enable port E clock
  while ((SYSCTL_PRGPIO_R & 0x10) == 0) {}; // Ready?
  GPIO_PORTE_DIR_R &= 0xF0; //Set up PE0-PE3 as inputs
  GPIO_PORTE_DEN_R |= 0x0F; //Enable pins 0-3 for digital
}
```

23

23

reading-guide-GPIO.pdf

Datasheet and Textbook Reading Guide for CprE 288
General-Purpose Input/Output (GPIO) on the TM4C Microcontroller

Chapter 10 of the Tiva TM4C microcontroller datasheet, General-Purpose Input/Outputs (GPIOs), covers the details of how GPIO works on our platform. Additionally, chapters 2, 4, and 7 of the Bai textbook have sections that help explain important aspects of this part of the datasheet. To help you work through this information, here is a reading guide to reduce your reading from over 50 pages of the datasheet to about 15 pages (i.e., for the GPIO Module). You may want to bookmark pages, sections, figures and tables for easy reference.

Readings about the TM4C Memory Map

1) Datasheet Section 2.4 Memory Model and Table 2-4 Memory Map
2) Bai textbook Chapters 2 and 6. Sections in these chapters describe the memory map used for the TM4C123G microcontroller. GPIO registers, along with other memory-mapped peripheral devices, occupy a specific range in the memory space. Each register has a unique address.
   a) Section 2.3.1 (Figure 2.10), section 2.6.2 (Table 2.5), section 6.3 (Figure 6.14)
3) tm4c123gh6pm.h header file: See the addresses used in the Register Definition macros. Compare these addresses with the peripheral device range in the memory map.
4) Valvano and Yerraballi Embedded Systems book, section 2.7, Address Space, http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C2_FundamentalConcepts.htm

Readings about the GPIO Module

1) Datasheet Chapter 10 (~15 pages)
   a) Skip: Sections 10.2.4 Commit Control, 10.2.6 Identification, Table 10-5
   b) In section 10.5, Register Descriptions, only read about the following important registers. The register macro names shown below are defined for our platform in the tm4c123gh6pm.h header file, where X denotes the ports A-F. Keep in mind our platform uses the APB bus addresses.
      i) Register 1: GPIO Data (GPIO_PORTX_DATA_R)
      ii) Register 2: GPIO Direction (GPIO_PORTX_DIR_R)
      iii) Register 10: GPIO Alternate Function Select (GPIO_PORTX_AFSEL_R)
      iv) Register 18: GPIO Digital Enable (GPIO_PORTX_DEN_R)
      v) Register 21: GPIO Analog Mode Select (GPIO_PORTX_AMSEL_R)
      vi) Register 22: GPIO Port Control (GPIO_PORTA_PCTL_R)
   c) In section 5.5, Registers Descriptions
      i) Register 60: General-Purpose Input/Output Run Mode Clock Gating Control (SYSCTL_RCGCGPIO_R)
   d) In section 23.4, see Table 23-5: GPIO Pins and Alternate Functions. This is the same table as in the Bai textbook: Table 7.1 and Table 2.6

2) Bai textbook Chapters 2, 4, and 7. Sections in these chapters elaborate on the content of datasheet Chapter 10. Read though these as needed to help you understand the datasheet.
   a) Sections 2.6.1, 2.6.2.3, 2.6.3: GPIO (~6.5 pages). Read sections except as noted below:
      i) Light read of section 2.6.3.1, The System Clock
      ii) Skip sections 2.6.3.3.4 – 5, Commit and Interrupt Control Registers
      iii) In section 2.6.3.3.6, Pad Control Registers, skip GPIODR2R – GPIOSLR (start with GPIODEN)
      iv) Skip section 2.6.3.3.7, Identification Registers
   b) Sections 4.5.7.1 – 4.5.7.2.4 (notice the register information in the figures), and section 4.5.7.2.5.3 (Figure 4.33: DRAModel.c sample code) (~6.5 pages).
      Section 4.5.6.5, Naming Convention and Definition, is helpful and explains the C keyword *volatile*. For more information, consult C programming resources, or check out these articles:
      "Introduction to the Volatile Keyword," by Nigel Jones, Embedded Systems Programming, July 2001: http://www.embedded.com/electronics-blogs/beginner-s-corner/4023801/Introduction-to-the-Volatile-Keyword
      Wikipedia Article: https://en.wikipedia.org/wiki/Volatile_(computer_programming)
   c) Sections 7.1, 7.2, 7.3, programming example in section 7.4 (Figure 7.7: DRAKeyPadPoll.c sample code) (~13 pages)
      Note that the section 7.4 programming example uses a different version of the header file to reference the registers. We use a Register Definition header file like the one shown in Figure 5.10 that provides macros to access the registers. The other header file – which we don't use in lab – defines each peripheral with a structure pointer, and registers are accessed using the pointer operator (->). The structure version is used in this example and several others in the textbook (read it, but don't code with it).
3) tm4c123gh6pm.h header file: See the Register Definition macros for the GPIO port registers. Compare their addresses with the address information given in the datasheet register descriptions.

How To Search for Registers in the Datasheet

There are various ways to find information in the datasheet. Use the table of contents. Search the PDF document. Use the sidebar in a PDF viewer. This short video from the Valvano and Yerraballi supplemental text demonstrates this:

Valvano and Yerraballi, Video D6-0: Using the GPIO register datasheet, https://youtu.be/VI2vrRwzWio (2 minutes)

Note: Refer to your own PDF copy of the Tiva TM4C123 datasheet -- open it separately in a PDF viewer to see the table of contents sidebar.

Ultimately, you need to have some part of the register name to look up or search for. You might find that in the Bai textbook, V&Y ES book, or the tm4c123gh6pm.h header file. Often, it works best to use just part of the register name. For example, I can search for "RCGC" in the header

file. That gives me the macro definition for SYSCTL_RCGCPGIO_R. Likewise in the datasheet, you'll get several matches when searching for "RCGC" and can browse through them to see what looks most useful for your purposes. Often, even if you don't get to the register description on the first match, there are hyperlinks in the datasheet that will take you there. Plus, after you find things a few times, you will begin to know where to look.

Supplemental Readings

Various sections from the book, Valvano and Yerraballi, Embedded Systems - Shape the World, http://users.ece.utexas.edu/~valvano/Volume1/E-Book/ , cover information in a more tutorial and step-by-step manner. You are encouraged to browse the V&Y book and choose what might be helpful to spend time on.

1) Valvano and Yerraballi supplemental text:
   a) Section 2.3: Introduction to Computers (Figures 2.8, 2.10), http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C2_FundamentalConcepts.htm
   b) Section 2.7: Address Space, http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C2_FundamentalConcepts.htm
   c) Chapter 6: Parallel I/O Ports, Embedded Systems - Shape the World, Sections 6.0 - 6.3 (through Interactive Tool 6.1), http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C6_MicrocontrollerPorts.htm
   d) Interactive exercise 6.1: http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C6_Interactives.htm
   e) Short videos available at Chapter 6 link or from YouTube, https://www.youtube.com/playlist?list=PLyg2vmIzGxXH48Rq5HsilGdQxbAh0w4Di (e.g., C6-1a, C6-1c, D6-0)
   f) Video C6-2b: Friendly Code, via Chapter 6 link or from YouTube, https://youtu.be/0LLD1kLFN-U (explains setting and clearing specific I/O port bits using bitwise operations)
   g) Section 10.3: Structures, http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C10_FiniteStateMachines.htm