CET 241: Day 7

Dr. Noori KIM

Basic concepts of I/O

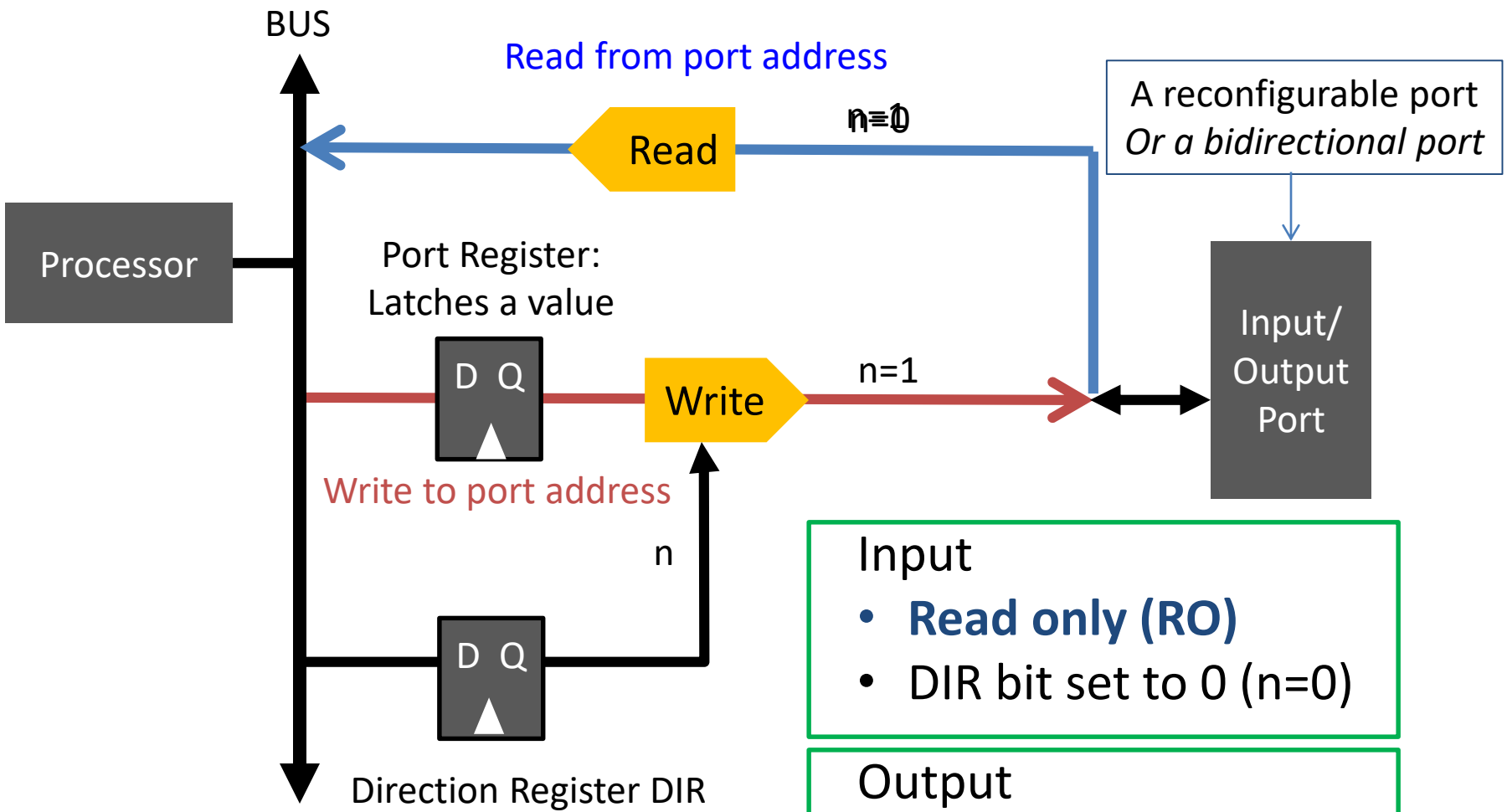
- Input signal mapping
  - Zero volt to logic level false (0)
  - 3.3 volt to logic level true (1)
- Output signal mapping: same
- Four types of I/O interfaces
  - Parallel: a bunch of bits at the same time
  - Serial: one bit at a time on a single line
  - Time: data are encoded as a period, freq, pulse width, or phase shift
  - Analog: data are decoded as an electrical voltage, current, or power
- We focus on **the Parallel interface**: GPIO ports

Let’s first look at GPIO datasheet of our MCU.

- How many registers are supporting GPIO?
- Understand that
  1. Different MCUs have their own features for each interface
  2. But certainly each interface has common features which are controlled by “peripheral registers”
- We will learn about GPIO initialization process focusing on important registers among all.
- All the other GPIO peripheral registers that we are not focusing → we are just using their default values.

Our GPIO ports are so called “Reconfigurable ports”
Can be used as input or output based on the configuration

Q: What kinds of tasks can be done by a port?

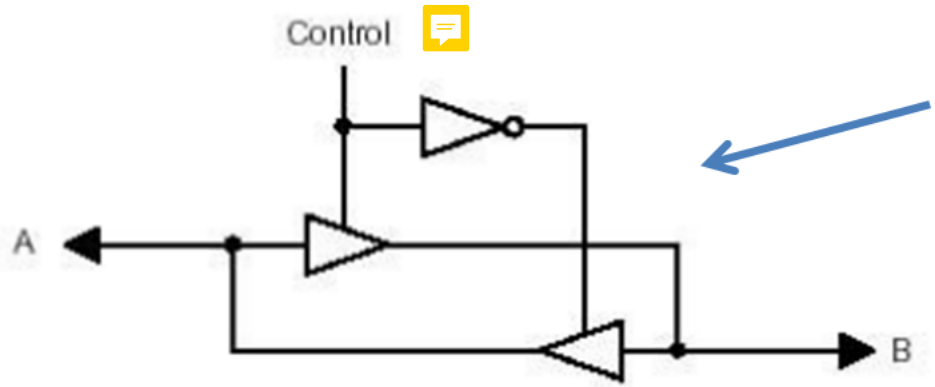


Input
- **Read only (RO)**
- DIR bit set to 0 (n=0)

Output
- **Read** and **write** (**RW**)
- DIR bit set to 1 (n=1)

Recap: Tri-state buffers

Tri-state Application 2: Bi-direction ports

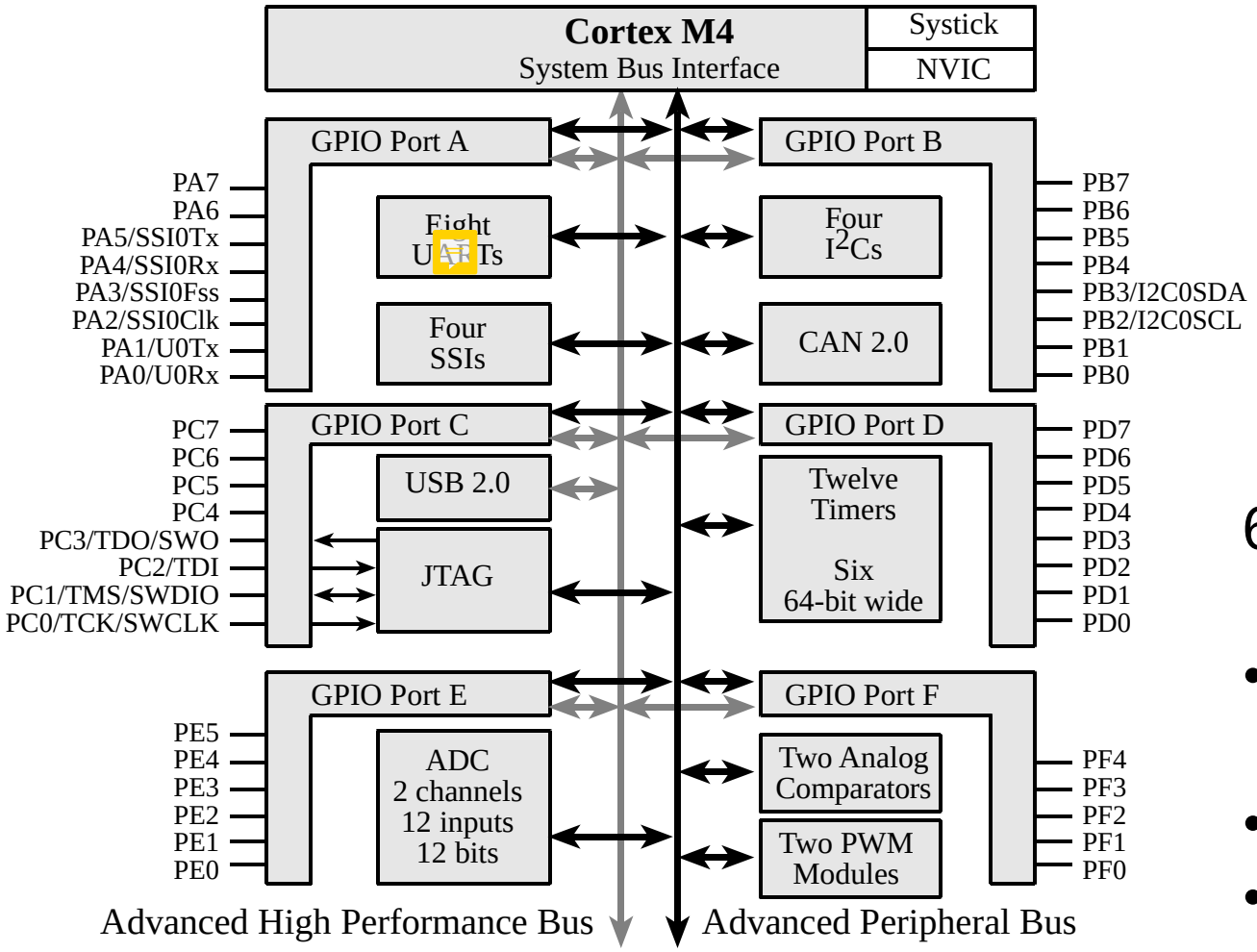


The Basic Idea

To the point

1. We can read **from an input port**
2. We can write or read **from an output port**
3. We control this operation with **the direction register (DIR)**
   - 0 means input
   - 1 means output.

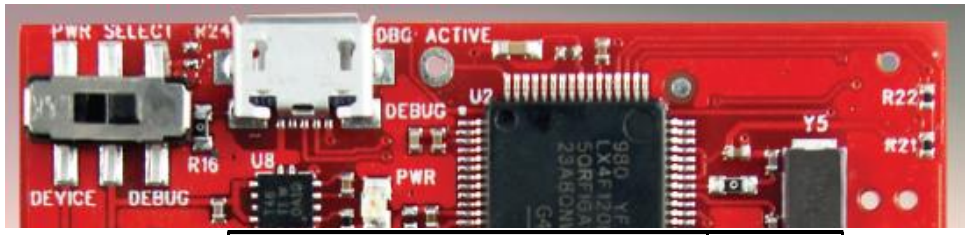
Input/Output: TM4C123

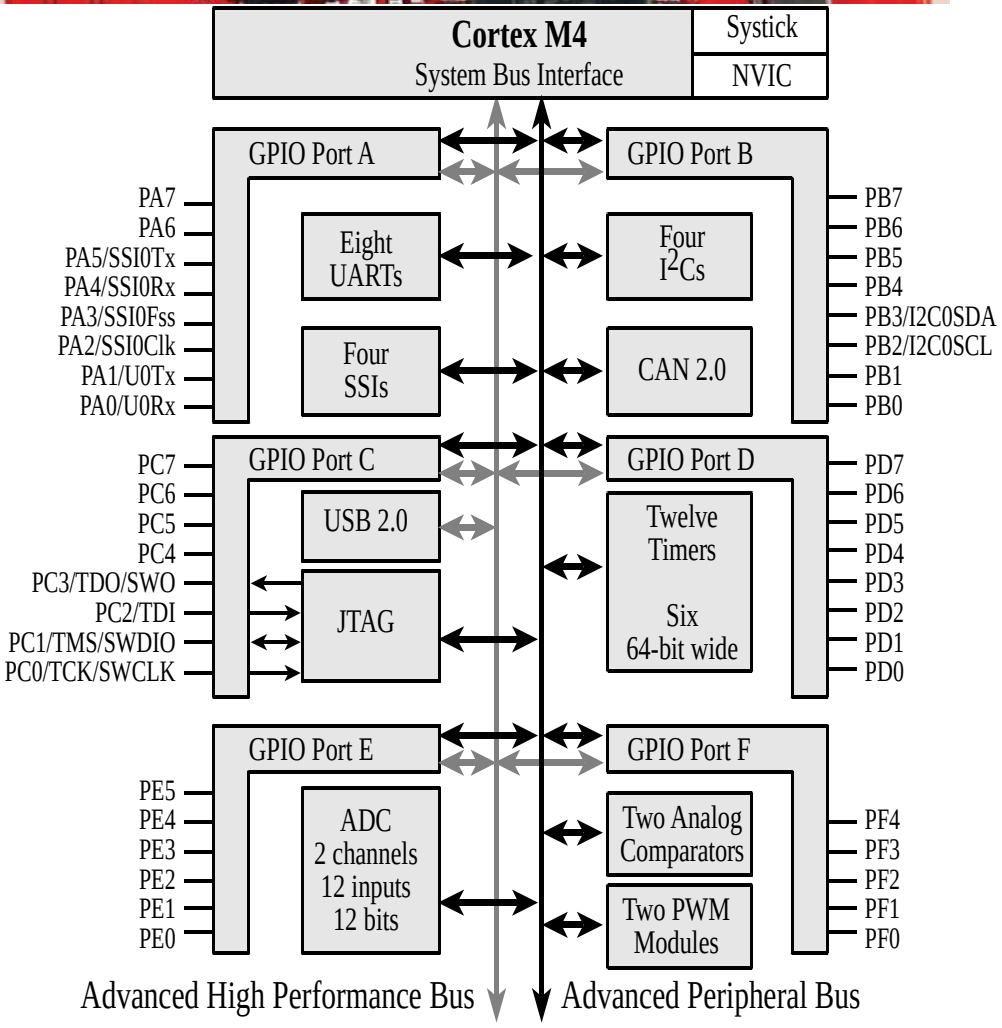


Notice that this is not one-to-one matching to the physical pins!! General M4 fam. Sharing structure

6 General-Purpose I/O (GPIO) ports:

- Four 8-bit ports (A, B, C, D)
- One 6-bit port (E)
- One 5-bit port (F)

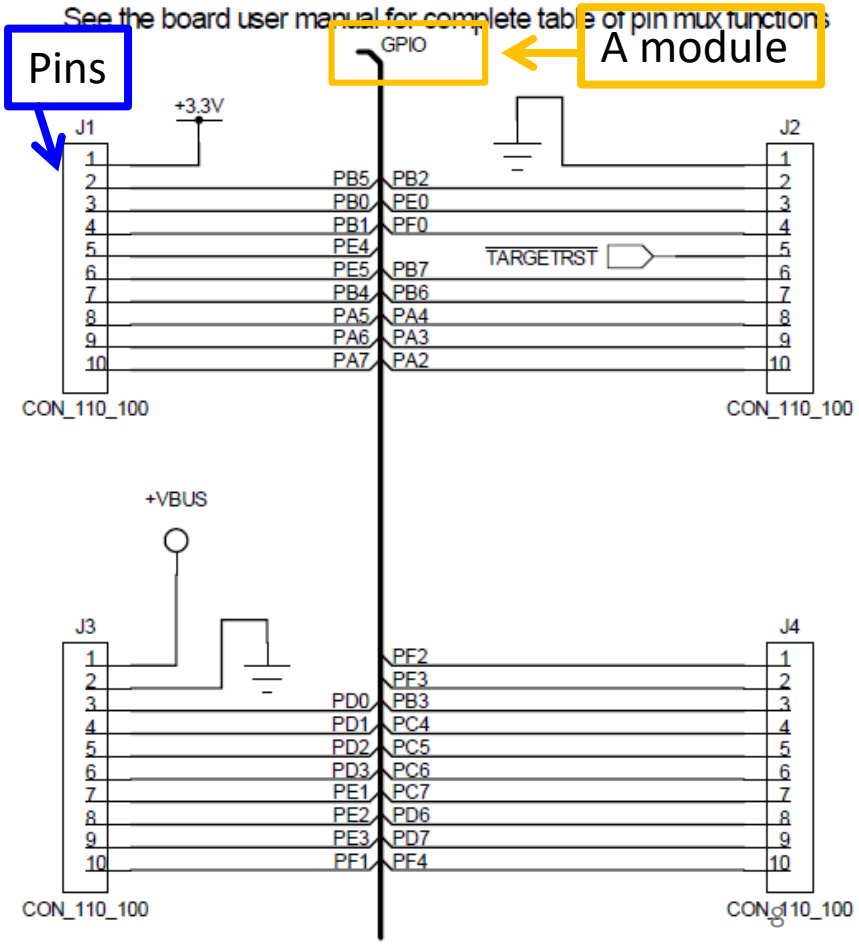




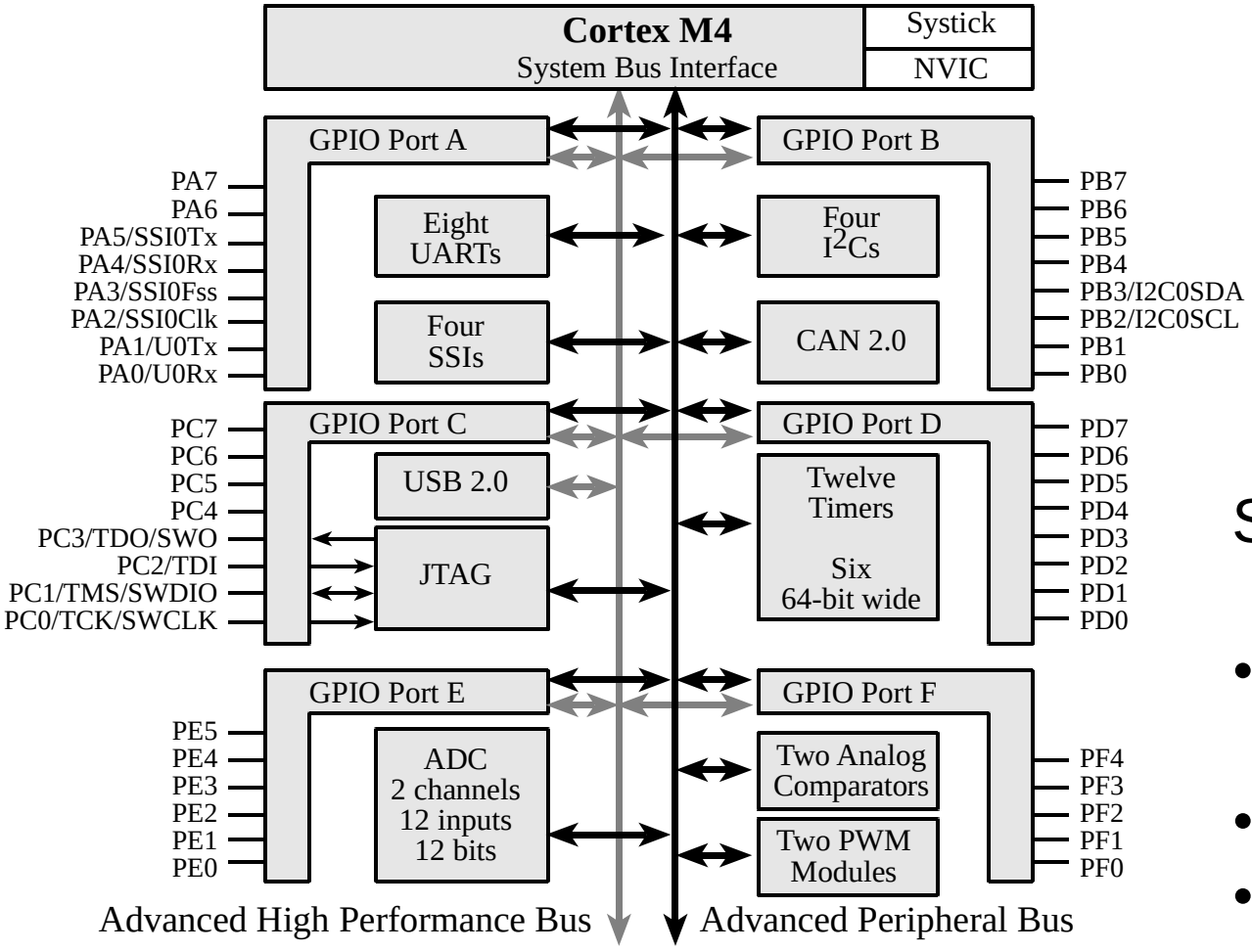
J1 and J2 provide compatability with Booster Packs designed for MSP430 Launchpad

J3 and J4 sit 100 mils inside J1 and J2 to provide extended functions specific to this board.

See the board user manual for complete table of pin mux functions



Input/Output: TM4C123



To use each port as other than GPIO: Control the AFSEL register Alternative Function SELection

Six General-Purpose I/O (GPIO) ports:

- Four 8-bit ports (A, B, C, D)
- One 6-bit port (E)
- One 5-bit port (F)

Check?

1. Set bits? Ans: Write 1 to the bits

   Register_name |= bit_position

2. Clear bits? Ans: Write 0 to the bits

   Register_name &= ~ (bit_position)

To set/clear/toggle

Friendly software modifies just the bits that need to be

The **or** operation to set bits 1 and 0 of a register.
The other six bits remain constant.

```
GPIO_PORTD_DIR_R |= 0x03; // PD1,PD0 outputs
```

The **and** operation to clear bits 1 and 0 of a register.
The other six bits remain constant.

```
GPIO_PORTD_DIR_R &= ~(0x03); // PD1,PD0 inputs
```

The **exclusive or** operation can also be used to toggle bits.

```
GPIO_PORTD_DATA_R ^= 0x80; /* toggle PD7 */
```

Let's go back to where we were at…

TM4C123 I/O Pins

- I/O Pin Characteristics
  - Can be employed as an n-bit *parallel* interface :GPIO port, default status mostly. Set AFSEL to 0
  - Pins also provide *alternative functions:*

UART	**Universal asynchronous receiver/transmitter**
SSI	**Synchronous serial interface**
I^2C	**Inter-integrated circuit**
Timer	**Periodic interrupts, input capture, and output compare**
PWM	**Pulse width modulation**
ADC	**Analog to digital converter, measure analog signals**
Analog	**Compare two analog signals comparator**
QEI	**Quadrature encoder interface**
USB	**Universal serial bus**
Ethernet	**High speed network**
CAN	**Controller area network**

Set AFSEL to 1

Analog and digital functions control

- Analog functions: AMSEL register
  - Analog Mode SELection
- Digital functions: DEN register
  - Digital ENable
- Digital function control: PCTL register
  - Port ConTroL
  - Does not specify “Digital,” as it is default (many)

Analog and digital functions control

- Analog functions: AMSEL register
  - Analog Mode SELection
- Digital functions: DEN register
  - Digital ENable
- Digital function control: PCTL register
  - Port ConTroL
  - Does not specify “Digital,” as it is default (many)

TM4C123 LaunchPad I/O Pins: 4 bits PCTL (Vol2:p74, Vol1:p142)

IO	Ain	0	1	2	3	4	5	6	7	8	9	14
PA2		Port		SSI0Clk								
PA3		Port		SSI0Fss								
PA4		Port		SSI0Rx								
PA5		Port		SSI0								
PA6		Port			I2C1SCL		M1PWM2					
PA7		Port			I2C1SDA		M1PWM3					
PB0		Port	U1Rx						T2CCP0			
PB1		Port	U1Tx						T2CCP1			
PB2		Port										
PB3		Port										
PB4	Ain10	Port				M0PWM2						
PB5	Ain11	Port				M0PWM3						
PB6		Port				M0PWM0						
PB7		Port		SSI2Tx		M0PWM1						
PC4	C1-	Port	U4Rx	U1Rx		M0PWM6		IDX1	WT0CCP0	U1RTS		
PC5	C1+	Port	U4Tx	U1Tx		M0PWM7		PhA1	WT0CCP1	U1CTS		
PC6	C0+	Port	U3Rx					Ph				
PC7	C0-	Port	U3Tx									
PD0	Ain7	Port	SSI3Clk	SSI1Clk	I2C3SCL	M0PWM6	M1PWM0					
PD1	Ain6	Port	SSI3Fss	SSI1Fss	I2C3SDA	M0PWM7	M1PWM1					
PD2	Ain5	Port	SSI3Rx	SSI1Rx		M0Fault0						
PD3	Ain4	Port	SSI3Tx	SSI1Tx				ID				
PD6		Port	U2Rx			M0Fault0		Ph				
PD7		Port	U2Tx					Ph				
PE0	Ain3	Port	U7Rx									
PE1	Ain2	Port	U7Tx									
PE2	Ain1	Port										
PE3	Ain0	Port										
PE4	Ain9	Port	U5Rx		I2C2SCL	M0PWM4	M1PWM2					
PE5	Ain8	Port	U5Tx		I2C2SDA	M0PWM5	M1PWM3					
PF0		Port	U1RTS	SSI1Rx	CAN0Rx		M1PWM4	Ph				
PF1		Port	U1CTS	SSI1Tx			M1PWM5	Ph				
PF2		Port		SSI1Clk		M0Fault0	M1PWM6					
PF3		Port		SSI1Fss	CAN0Tx		M1PWM7					
PF4		Port					M1Fault0	IDX0	T2CCP0	USB0epen		

Datasheet pp.650

Available Analog function

0 means Regular I/O ports

Column 5: Set 4-bit field in PCTL to 0101

Reserved pins:

- PA1-0: serial COM port linked through the debugger cable
- PC3-0: JTAG debugger, not used as regular I/Os
- PD5-4: USB device

The number on the first row is nothing to do with pins# or bits #.

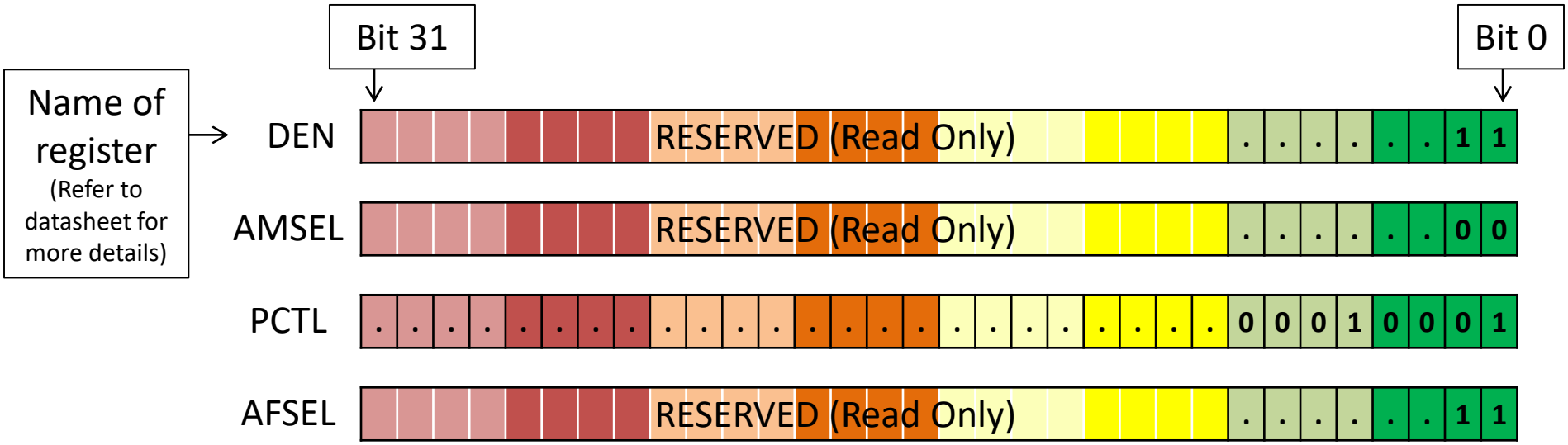
Just an encoding to use special digital functions of each pin.

Ex) If you want to use UART7 on pins PE0 and PE1

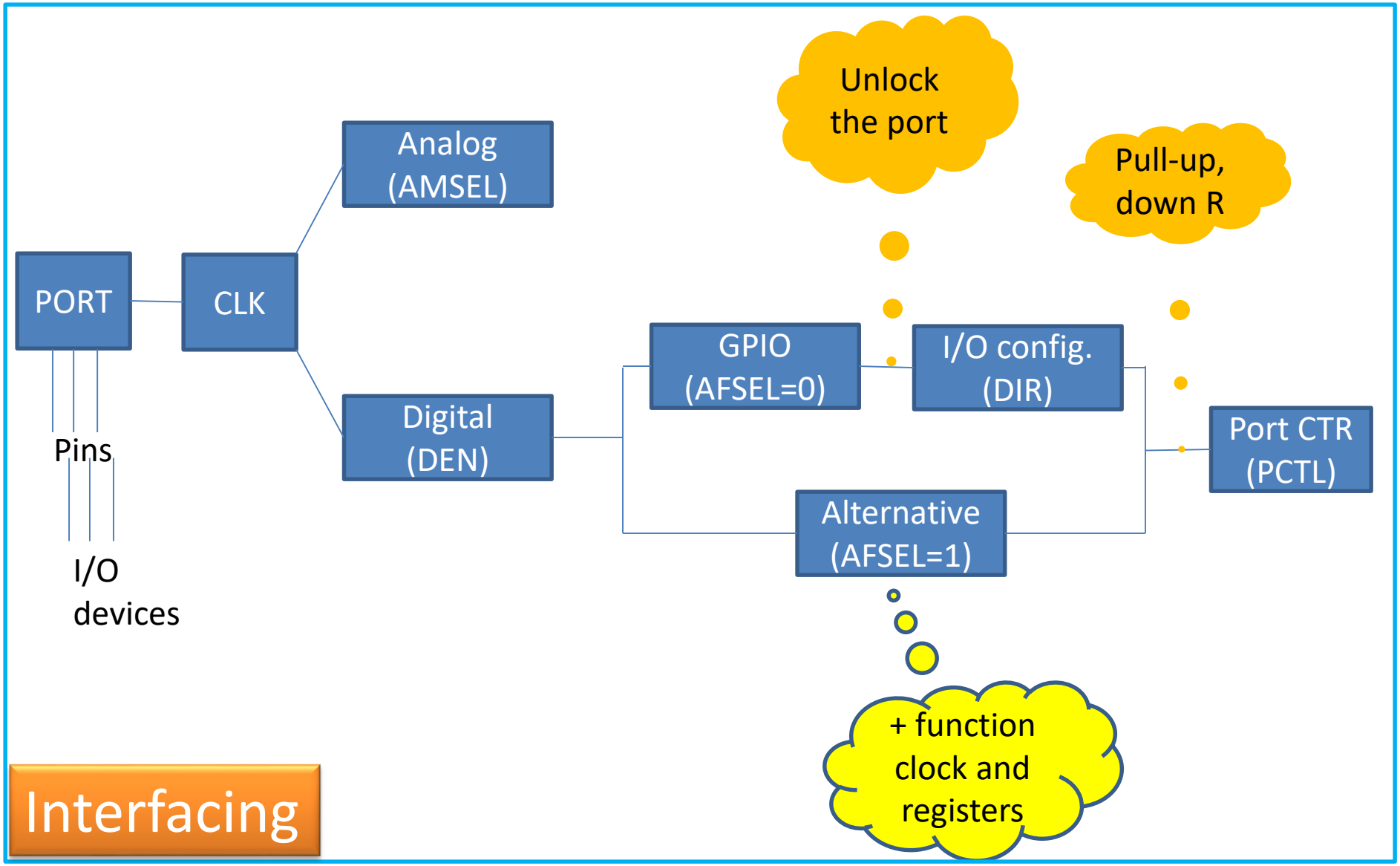
IO	Ain	0	1	2	3	4	5	6	7	8	9	14
PA2		Port		SSI0Clk								
PA3		Port		SSI0Fss								
PA4		Port		SSI0Rx								
PA5		Port		SSI0Tx								
PA6		Port			I_2C1SCL		M1PWM2					
PA7		Port			I_2C1SDA		M1PWM3					
PB0		Port	U1Rx						T2CCP0			
PB1		Port	U1Tx						T2CCP1			
PB2		Port			I_2C0SCL				T3CCP0			
PB3		Port			I_2C0SDA				T3CCP1			
PB4	Ain10	Port		SSI2Clk		M0PWM2			T1CCP0	CAN0Rx		
PB5	Ain11	Port		SSI2Fss		M0PWM3			T1CCP1	CAN0Tx		
PB6		Port		SSI2Rx		M0PWM0			T0CCP0			
PB7		Port		SSI2Tx		M0PWM1			T0CCP1			
PC4	C1-	Port	U4Rx	U1Rx		M0PWM6		IDX1	WT0CCP0	U1RTS		
PC5	C1+	Port	U4Tx	U1Tx		M0PWM7		PhA1	WT0CCP1	U1CTS		
PC6	C0+	Port	U3Rx					PhB1	WT1CCP0	USB0epen		
PC7	C0-	Port	U3Tx						WT1CCP1	USB0pflt		
PD0	Ain7	Port	SSI3Clk	SSI1Clk	I_2C3SCL	M0PWM6	M1PWM0		WT2CCP0			
PD1	Ain6	Port	SSI3Fss	SSI1Fss	I_2C3SDA	M0PWM7	M1PWM1		WT2CCP1			
PD2	Ain5	Port	SSI3Rx	SSI1Rx		M0Fault0			WT3CCP0	USB0epen		
PD3	Ain4	Port	SSI3Tx	SSI1Tx				IDX0	WT3CCP1	USB0pflt		
PD6		Port	U2Rx			M0Fault0		PhA0	WT5CCP0			
PD7		Port	U2Tx					PhB0	WT5CCP1	NMI		
PE0	Ain3	Port	U7Rx									
PE1	Ain2	Port	U7Tx									
PE2	Ain1	Port										
PE3	Ain0	Port										
PE4	Ain9	Port	U5Rx		I_2C2SCL	M0PWM4	M1PWM2			CAN0Rx		
PE5	Ain8	Port	U5Tx		I_2C2SDA	M0PWM5	M1PWM3			CAN0Tx		
PF0		Port	U1RTS	SSI1Rx	CAN0Rx		M1PWM4	PhA0	T0CCP0	NMI	C0o	
PF1		Port	U1CTS	SSI1Tx			M1PWM5	PhB0	T0CCP1		C1o	TRD1
PF2		Port		SSI1Clk		M0Fault0	M1PWM6		T1CCP0			TRD0
PF3		Port		SSI1Fss	CAN0Tx		M1PWM7		T1CCP1			TRCLK
PF4		Port					M1Fault0	IDX0	T2CCP0	USB0epen		

Ex) If you want to use UART7 on pins PE0 and PE1

1. Enable digital: **DEN** register
   - set bits # 0 and 1
2. Disable analog: **AMSEL** register
   - clear bits # 0 and 1
3. Enable UART7 functions: **PCTL** register
   - set bit 1 for the two ports: 0001,0001 (bits7-0 in the PCTL register)
4. Enable alternate functions: **AFSEL** register
   - set bits # 0 and 1



Then, let’s focus on our original objective, the GPIO ports!



Interfacing

Important: Datasheet pp.657

GPIO port initialization (7 steps)

1. Activate clock -> wait to be stabilized (using delay): RCGCGPIO or RCGC
   - Run mode Clock Gating Control: RCGCGPIO (datasheet p340)
     - Base 0x400F.E000, Offset 0x608

Register 60: General-Purpose Input/Output Run Mode Clock Gating Control (RCGCGPIO), offset 0x608

The **RCGCGPIO** register provides software the capability to enable and disable GPIO modules in Run mode. When enabled, a module is provided a clock and accesses to module registers are allowed. When disabled, the clock is disabled to save power and accesses to module registers generate a bus fault. This register provides the same capability as the legacy **Run Mode Clock Gating Control Register n RCGCn** registers specifically for the watchdog modules and has the same bit polarity as the corresponding **RCGCn** bits.

Important: This register should be used to control the clocking for the GPIO modules. To support legacy software, the **RCGC2** register is available. A write to the **RCGC2** register also writes the corresponding bit in this register. Any bits that are changed by writing to the **RCGC2** register can be read back correctly with a read of the **RCGC2** register. Software must use this register to support modules that are not present in the legacy registers. If software uses this register to write a legacy peripheral (such as GPIO A), the write causes proper operation, but the value of that bit is not reflected in the **RCGC2** register. If software uses both legacy and peripheral-specific register accesses, the peripheral-specific registers must be accessed by read-modify-write operations that affect only peripherals that are not present in the legacy registers. In this manner, both the peripheral-specific and legacy registers have coherent information.

General-Purpose Input/Output Run Mode Clock Gating Control (RCGCGPIO)

Base 0x400F.E000
Offset 0x608
Type RW, reset 0x0000.0000

Important: GPIO port initialization (7 steps)

1. Activate clock -> wait to be stabilized (using delay): RCGCGPIO or RCGC
   - Run mode Clock Gating Control: RCGCGPIO (datasheet p340)
     - Base 0x400F.E000, Offset 0x608
   - Or RCGC2 (p464)
     - Run Mode Clock Gating Control Register 2 (RCGC2)
     - Base 0x400F.E000, Offset 0x108

Register 136: Run Mode Clock Gating Control Register 2 (RCGC2), offset 0x108

This register controls the clock gating logic in normal Run mode. Each bit controls a clock enable for a given interface, function, or module. If set, the module receives a clock and functions. Otherwise, the module is unclocked and disabled (saving power). If the module is unclocked, reads or writes to the module generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional modules are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or modules to control. This configuration is implemented to assure reasonable code compatibility with other family and future parts. **RCGC2** is the clock configuration register for running operation, **SCGC2** for Sleep operation, and **DCGC2** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes. Note that there must be a delay of 3 system clocks after a module clock is enabled before any registers in that module are accessed.

Important: This register is provided for legacy software support only.

The peripheral-specific Run Mode Clock Gating Control registers (such as **RCGCDMA**) should be used to reset specific peripherals. A write to this legacy register also writes the corresponding bit in the peripheral-specific register. Any bits that are changed by writing to this register can be read back correctly with a read of this register. Software must use the peripheral-specific registers to support modules that are not present in the legacy registers. If software uses a peripheral-specific register to write a legacy peripheral (such as the µDMA), the write causes proper operation, but the value of that bit is not reflected in this register. If software uses both legacy and peripheral-specific register accesses, the peripheral-specific registers must be accessed by read-modify-write operations that affect only peripherals that are not present in the legacy registers. In this manner, both the peripheral-specific and legacy registers have coherent information.

Run Mode Clock Gating Control Register 2 (RCGC2)

Base 0x400F.E000
Offset 0x108
Type RO, reset 0x0000.0000

Important: GPIO port initialization (7 steps)

1. Activate clock -> wait to be stabilized (using delay): RCGCGPIO or RCGC
   - Run mode Clock Gating Control: RCGCGPIO (datasheet p340)
     - Base 0x400F.E000, Offset 0x608
   - Or RCGC2 (p464)
     - Run Mode Clock Gating Control Register 2 (RCGC2)
     - Base 0x400F.E000, Offset 0x108
   - Common Error: You will get a bus fault if you access a port without enabling its clock)

Important: GPIO port initialization (7 steps)

1. Activate clock -> wait to be stabilized (using delay): RCGCGPIO or RCGC
   - Run mode Clock Gating Control: RCGCGPIO (datasheet p340)
     - Base 0x400F.E000, Offset 0x608
   - Or RCGC2 (p464)
     - Run Mode Clock Gating Control Register 2 (RCGC2)
     - Base 0x400F.E000, Offset 0x108
2. Unlock the port (only for pins PC3-0, PD7 and PF0 on our board)

Table 10-5. GPIO Pins With Special Considerations

GPIO Pins	Default Reset State	GPIOAFSEL	GPIODEN	GPIOPDR	GPIOPUR	GPIOPCTL	GPIOCR
PA[1:0]	UART0	0	0	0	0	0x1	1
PA[5:2]	SSI0	0	0	0	0	0x2	1
PB[3:2]	$I^{21}C0$	0	0	0	0	0x3	1
PC[3:0]	JTAG/SWD	1	1	0	1	0x1	0
PD[7]	GPIO[a]	0	0	0	0	0x0	0
PF[0]	GPIO[a]	0	0	0	0	0x0	0

a. This pin is configured as a GPIO by default but is locked and can only be reprogrammed by unlocking the pin in the **GPIOLOCK** register and uncommitting it by setting the **GPIOCR** register.

The GPIO commit control registers provide a layer of protection against accidental programming of critical hardware signals including the GPIO pins that can function as JTAG/SWD signals and the `NMI` signal. The commit control process must be followed for these pins, even if they are programmed as alternate functions other than JTAG/SWD or NMI; see “Commit Control” on page 656.

Important:
GPIO port initialization (7 steps)

3. Disable analog mode function: AMSEL
4. Disable (digital) port control: PCTL
5. Set input/output bit direction: DIR
6. Disable alternate functions: AFSEL
7. Enable digital port: DEN

After initialization: Read/write GPIO_PORTF_DATA_R or access the bit directly (i.e., Input from switches, output to LED)

Table 10-3. GPIO Pad Configuration Examples

pp. 657

- We talked about general GPIO initializing process.
- Then let’s talk about address of each register that we need to initialize a port

We talked about I/O peripheral registers' initialization process so far such as

```
void PortF_Init_Lab0(void){
  volatile unsigned long delay;
      SYSCTL_RCGC2_R |= 0x00000020;         // 1) F clock
                delay  = SYSCTL_RCGC2_R;      //    delay
  // GPIO_PORTF_LOCK_R = 0x4C4F434B;          // 2) unlock PortF PF0 (see pp.688)
  // GPIO_PORTF_CR_R |= 0x01;                 //    In case to use SW2 (PF0) allowing changes
  GPIO_PORTF_AMSEL_R &= ~0x1E;              // 3) disable analog function (clear PF4-1)
   GPIO_PORTF_PCTL_R &= ~0xFFFFF0;             // 4) GPIO clear bit PCTL (clear PF4-1)
    GPIO_PORTF_DIR_R &= ~0x10;              // 5) PF4 input, SW1 (clear PF4)
    GPIO_PORTF_DIR_R |= 0x0E;               //    PF3,PF2,PF1 output, RBG LEDs (set PF3-1)
  GPIO_PORTF_AFSEL_R &= ~0x1E;              // 6) no alternate function (clear PF4-1)
    GPIO_PORTF_PUR_R |= 0x10;               //    enable pullup resistors on PF4 (set PF4)
    GPIO_PORTF_DEN_R |= 0x1E;               // 7) enable digital pins (set PF4-1)
}
```

Note that the code runs one time, it will never run again during your program run

Then let's talk about addresses of that (peripheral) registers such as

```
#define GPIO_PORTF_DATA_R       (*((unsigned long *)0x400250F8))
#define GPIO_PORTF_DIR_R        (*((unsigned long *)0x40025400))
#define GPIO_PORTF_AFSEL_R      (*((unsigned long *)0x40025420))
#define GPIO_PORTF_PUR_R        (*((unsigned long *)0x40025510))
#define GPIO_PORTF_DEN_R        (*((unsigned long *)0x4002551C))
#define GPIO_PORTF_LOCK_R       (*((unsigned long *)0x40025520))
#define GPIO_PORTF_CR_R         (*((unsigned long *)0x40025524))
#define GPIO_PORTF_AMSEL_R      (*((unsigned long *)0x40025528))
#define GPIO_PORTF_PCTL_R       (*((unsigned long *)0x4002552C))
#define SYSCTL_RCGC2_R          (*((unsigned long *)0x400FE108))
```

Port's base addr + offset for each bit field

Port's base addr + offset for each register

One address (to access each port, enable a bit field of the port)

TM4C123 I/O registers
GPIO base address p659

- GPIO Port A (APB): 0x4000.4000
- GPIO Port B (APB): 0x4000.5000
- GPIO Port C (APB): 0x4000.6000
- GPIO Port D (APB): 0x4000.7000
- GPIO Port E (APB): 0x4002.4000
- GPIO Port F (APB): 0x4002.5000

base+$000	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	GPIO_PORTx_DATA_R
base+$400	DIR	DIR	DIR	DIR	DIR	DIR	DIR	DIR	GPIO_PORTx_DIR_R
base+$420	AFSEL	AFSEL	AFSEL	AFSEL	AFSEL	AFSEL	AFSEL	AFSEL	GPIO_PORTx_AFSEL_R
base+$510	PUE	PUE	PUE	PUE	PUE	PUE	PUE	PUE	GPIO_PORTx_PUR_R
base+$51C	DEN	DEN	DEN	DEN	DEN	DEN	DEN	DEN	GPIO_PORTx_DEN_R
base+$524	CR	CR	CR	CR	CR	CR	CR	CR	GPIO_PORTx_CR_R
base+$528	AMSEL	AMSEL	AMSEL	AMSEL	AMSEL	AMSEL	AMSEL	AMSEL	GPIO_PORTx_AMSEL_R

Example address for GPIO PORTF

Address	7	6	5	4	3	2	1	0	Name
400F.E608	-	-	**GPIOF**	GPIOE	GPIOD	GPIOC	GPIOB	GPIOA	SYSCTL_RCGCGPIO_R
4002.5000	-	-	-	DATA	DATA	DATA	DATA	DATA	GPIO_PORTF_DATA_R
4002.5400	-	-	-	DIR	DIR	DIR	DIR	DIR	GPIO_PORTF_DIR_R
4002.5420	-	-	-	SEL	SEL	SEL	SEL	SEL	GPIO_PORTF_AFSEL_R
4002.551C	-	-	-	DEN	DEN	DEN	DEN	DEN	GPIO_PORTF_DEN_R

I/O Port Bit-Specific (vol1 p156, vol2 p107)

- I/O Port bit-specific addressing is used to access port data register
  - Define address offset as $4*2^b$, where b is the selected bit position
  - Add offsets for each bit selected to base address for the port

Students who want to understand the mechanism behind this method, Search for page 654 and 662

If we wish to access bit	*Constant*
7	0x0200
6	0x0100
5	0x0080
4	0x0040 → $4*2^4$
3	0x0020
2	0x0010
1	0x0008
0	0x0004 → $4*2^0$

- Example: PF4 and PF0

Port F = 0x4002.5000

0x4002.5000+0x0004+0x0040

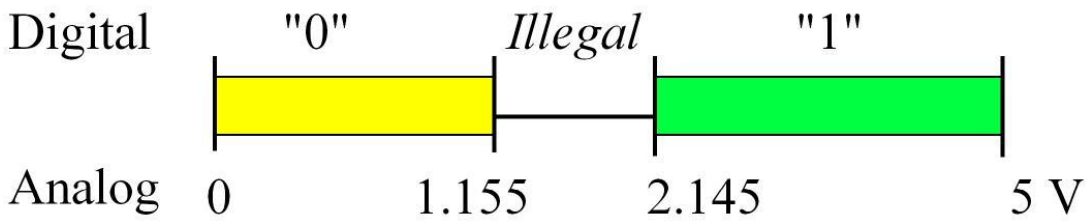
= 0x4002.5044

- What happens if we write to location GPIO Port A 0x4000.4000?
  - Nothing happens as specific bits are not selected. None of the port bits are affected.

In our practical session

- We will utilize port initialization knowledge
- We will build a simple alarm system
  - When a switch is pressed
  - An LED is blinking
- Need to learn an LED and a switch

Simple Digital Logic



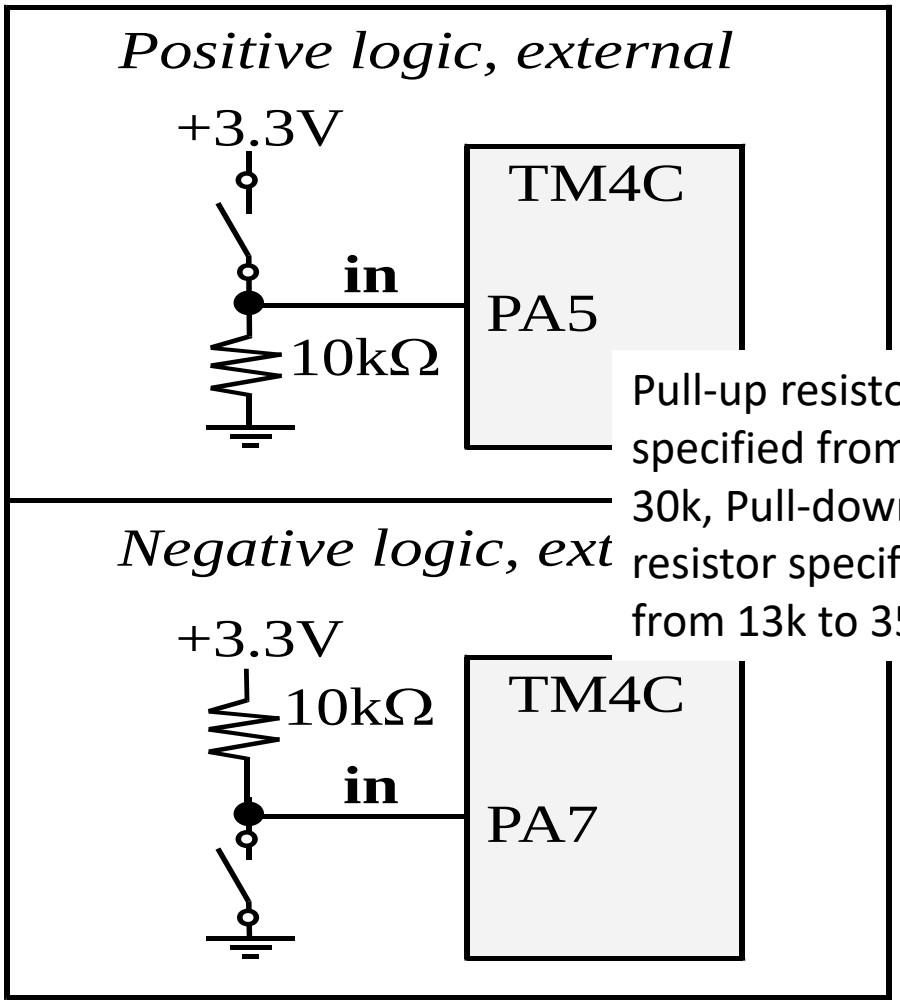
- Positive logic:
  True is higher voltage
  False is lower voltage
- Negative logic :
  True is lower voltage
  False is higher voltage

Have you encountered this problem?

https://www.youtube.com/watch?v=cdMJvFT-Afc&list=PLemM1Mp09_o1IHLzu9rAJghh4ZMNrwlvG

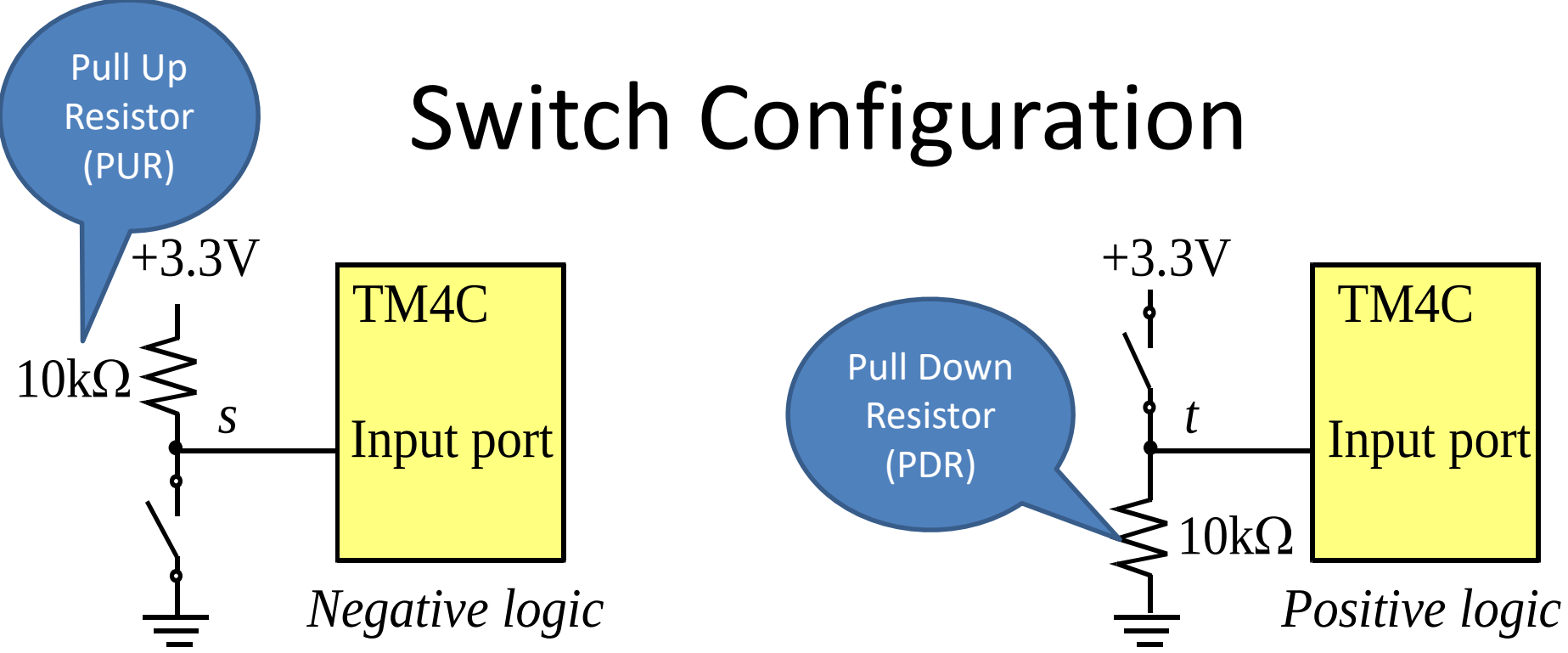
AND gate implementation
8:14---8:29 (switch problem)

Switch configuration (Analog / Digital)



Pull-up resistor is specified from 13k to 30k, Pull-down resistor specified from 13k to 35k

Switch Configuration



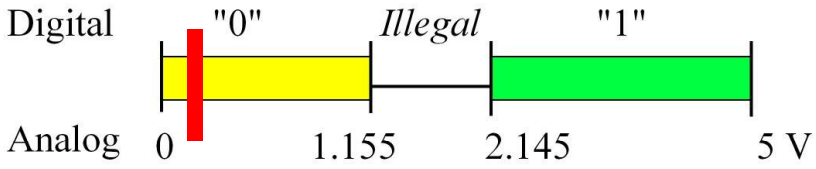
Positive or negative logics: refer to 'pressed/connected' status

Negative Logic *s*
- pressed, 0V, false
- not pressed, 3.3V, true

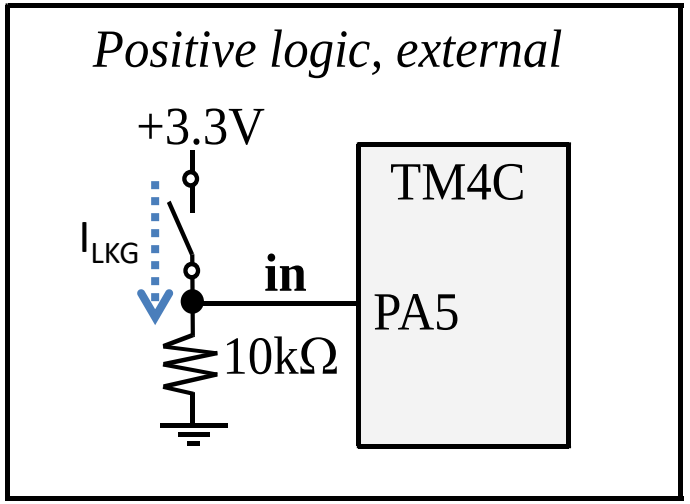
Positive Logic *t*
- pressed, 3.3V, true
- not pressed, 0V, false

Calculate the resistor value to interface TM4C switches

- (Datasheet p1385, leakage current) I_{LKG}=2µA
  - A lowest input current value that we keep with it to make a meaningful signal
  - (Ohm's law) 10kOhm*2µA=20mV

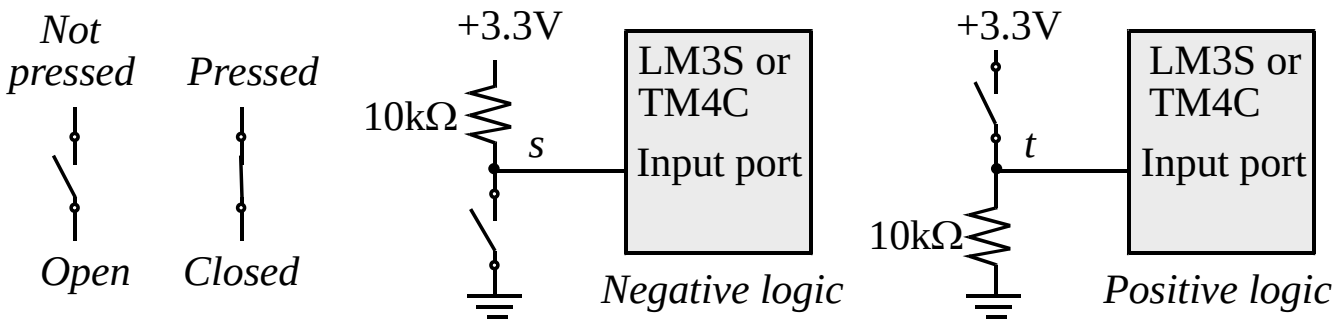


  - When the switch is opened



We will get 20mV which corresponds to a logic 0

Switch Interfacing



The **and** operation to extract, or *mask*, individual bits:

```
Pressed = GPIO_PORTA_DATA_R & 0x40;
```

Assembly:

```
LDR   R0,=GPIO_PORTA_DATA_R
LDR   R1,[R0]        ; read port A
AND   R1,R1,#0x40    ; clear all bits except bit 6
LDR   R0,=Pressed    ; update variable (pressed
STR   R1,[R0]        ; true if switch pressed
```

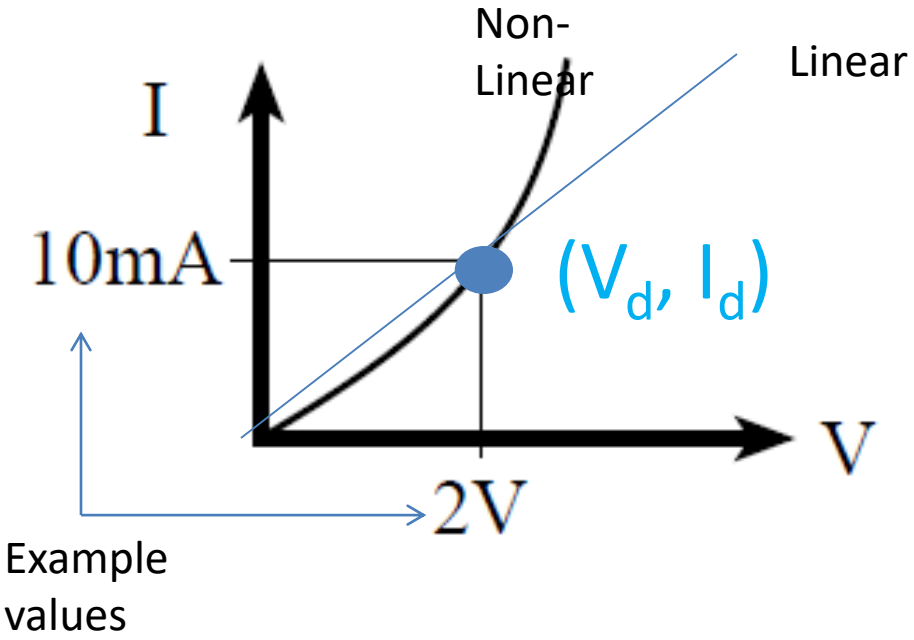
a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
0	1	0	0	0	0	0	0	value of **R1**
0	a_6	0	0	0	0	0	0	result of the **AND**

0x40 constant

Positive or negative logic or can't decide?

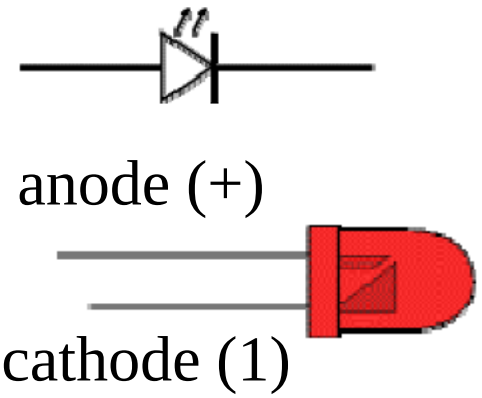
Assembly Vs. C

LED Interfacing



LED current vs. voltage

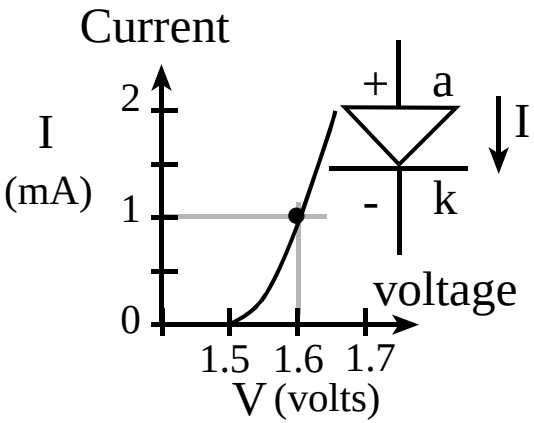
Brightness = power = V*I



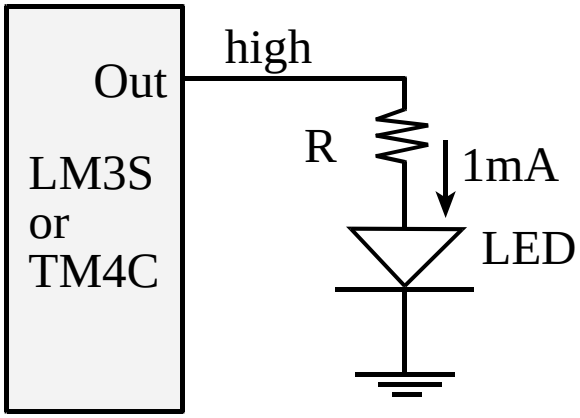
"big voltage connects to big pin"

(V_d, I_d): our target to use LED

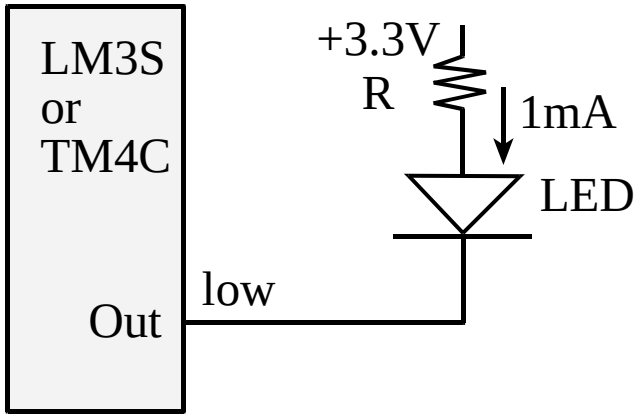
LED Configuration



(a) LED curve



(b) Positive logic interface



(c) Negative logic interface

$(V_d, I_d)=(1.6V,1mA)$

Look up the LED datasheet to search for VI characteristics

- LED turns on only when output is high

- LED turns on only when output is low to have enough voltage drop

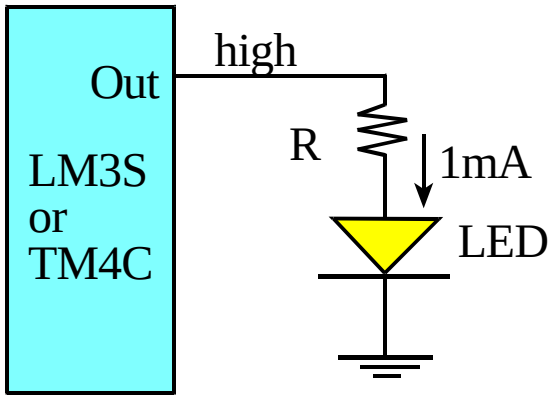
LED Interfacing

- **Positive logic: LED is on when the logic is high**

V_{dd}-R*I_d-V_d=0 // V_{dd}=R*I_d+V_d

R_{max} = (3.3V – 1.6V)/0.001A

= 1.7 kOhm



R_{min} = (3.3V – 1.6V)/0.008A

= 212.5Ohm

LED current < max current value i.e., 8 mA

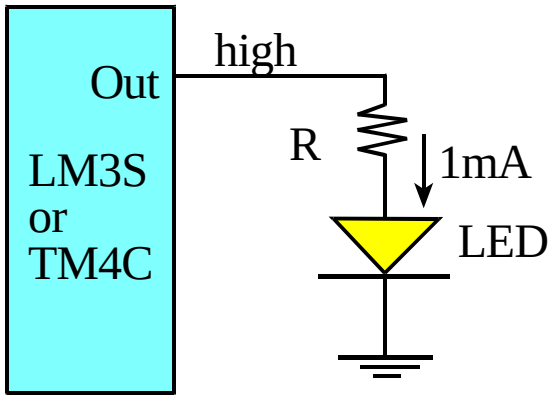
LED Interfacing

- **Positive logic: LED is on when the logic is high**

V_{dd}-R*I_d-V_d=0 // V_{dd}=R*I_d+V_d

R = (3.3V – 1.6V)/0.001A

= 1.7 kOhm



Ex) Our LED…

ELECTRICAL / OPTICAL CHARACTERISTICS (T_A =25°C)

Parameter	HER HLMP-1700/4700	YELLOW HLMP-1719/4719	GREEN HLMP-1790/4740 (MV2454)	Condition
Luminous Intensity (mcd)				I_F = 2mA
Minimum	1.0/1.2	1.0/1.2	1.0/1.2	
Typical	2.0/2.0	2.0/2.0	2.0/3.0	
Forward Voltage (V)				I_F = 2mA
Maximum	2.2	2.7	2.7	
Typical	1.8	1.9	1.9	
Peak Wavelength (nm)	635	585	565	I_F = 2mA
Spectral Line Half Width	45	35	30	I_F = 2mA
Viewing Angle (°)	50/35	50/35	50/35	I_F = 2mA

LED current < max current value i.e., 8 mA

Let’s talk about LaunchPad’s LEDs and Switches

- OK, I want to communicate with my LaunchPad board. For example,
  - Input: on-board (internal) SW1
  - Output: on-board (internal) LED
- What I need
  1. Hardware: Electrical circuit connection understanding
  2. Software: Proper programming

LaunchPad Switches and LEDs

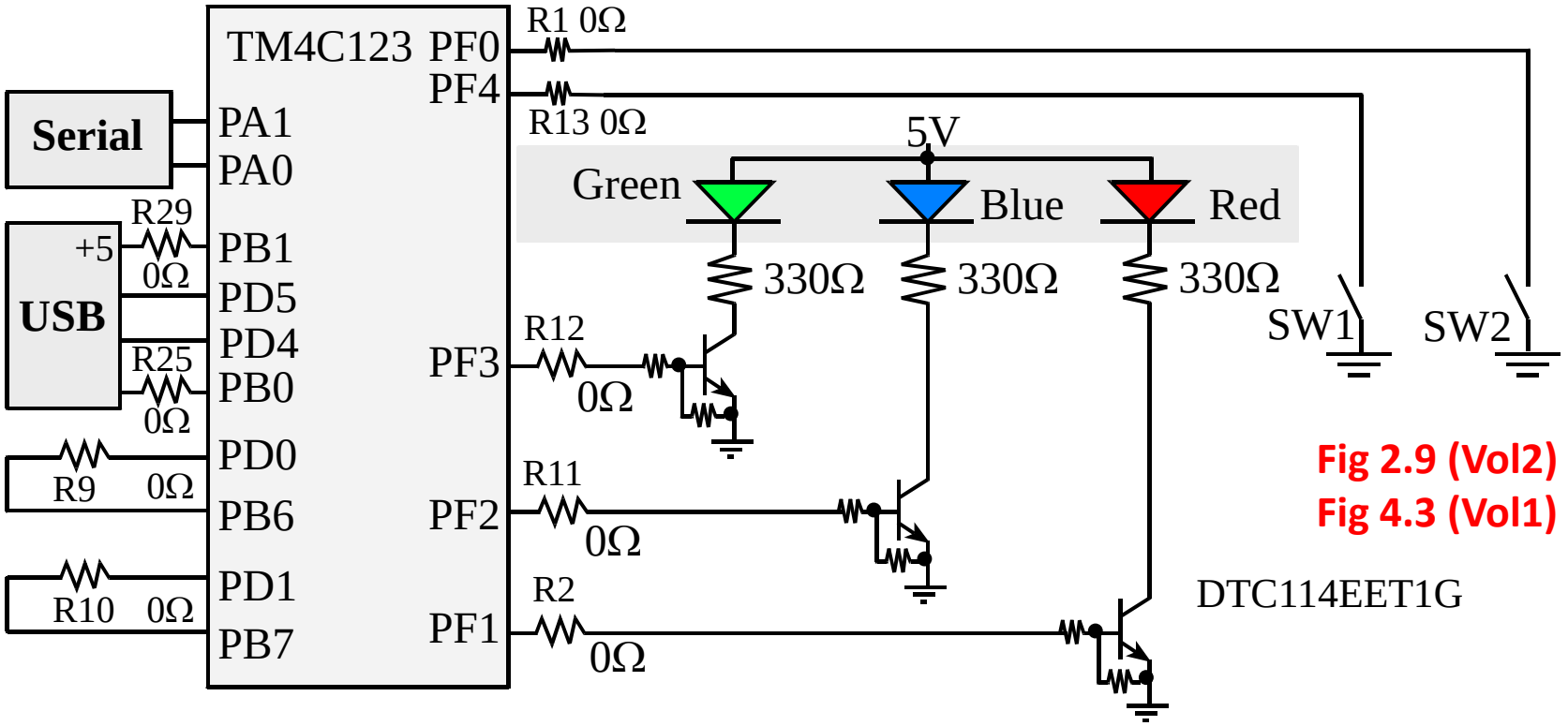
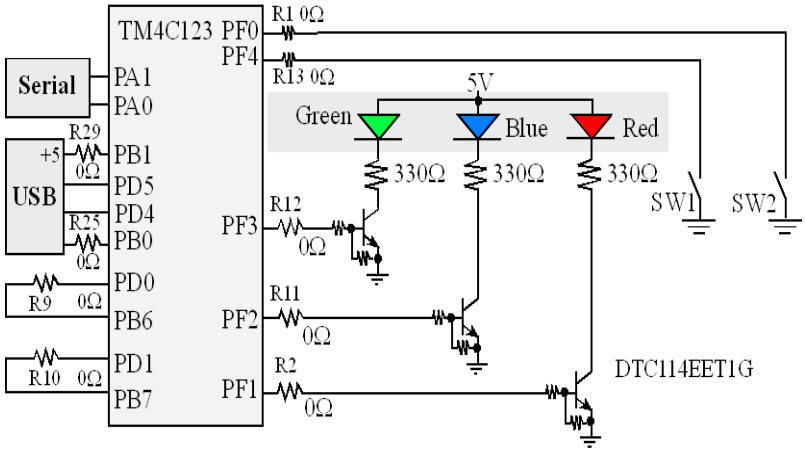


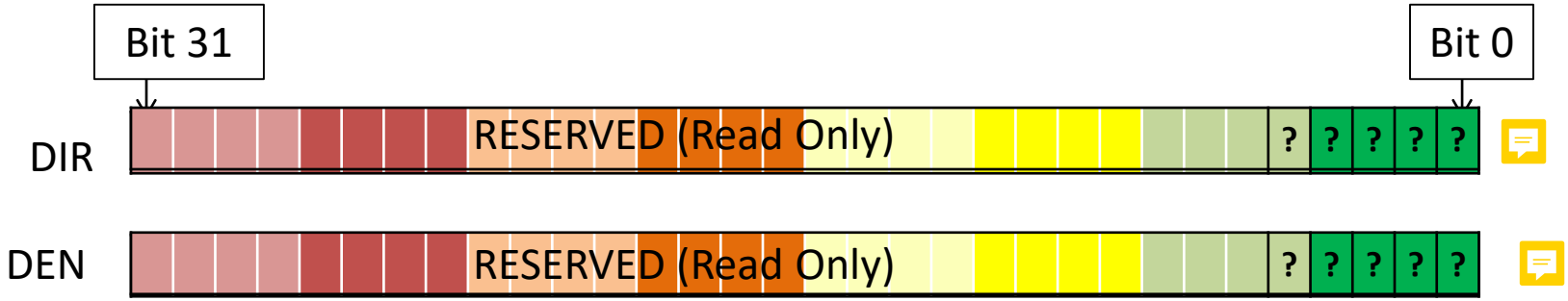
Fig 2.9 (Vol2)
Fig 4.3 (Vol1)

- **The switches on the LaunchPad**
  - **Negative logic**
  - **Require internal pull-up (set bits in the Pull-Up Register)**
- **The PF3-1 LEDs are positive logic**

PortF Register (5 pins -> bit 4-0)

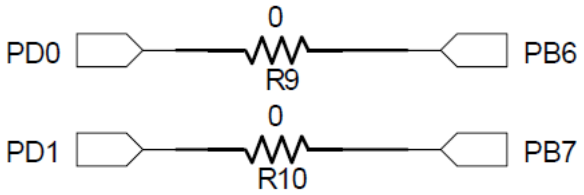
- Input: Q. Switches or LEDs?
  - Switches
  - PF0 and PF4
- Output: Q. Switches or LEDs?
  - LEDs
  - PF1-3



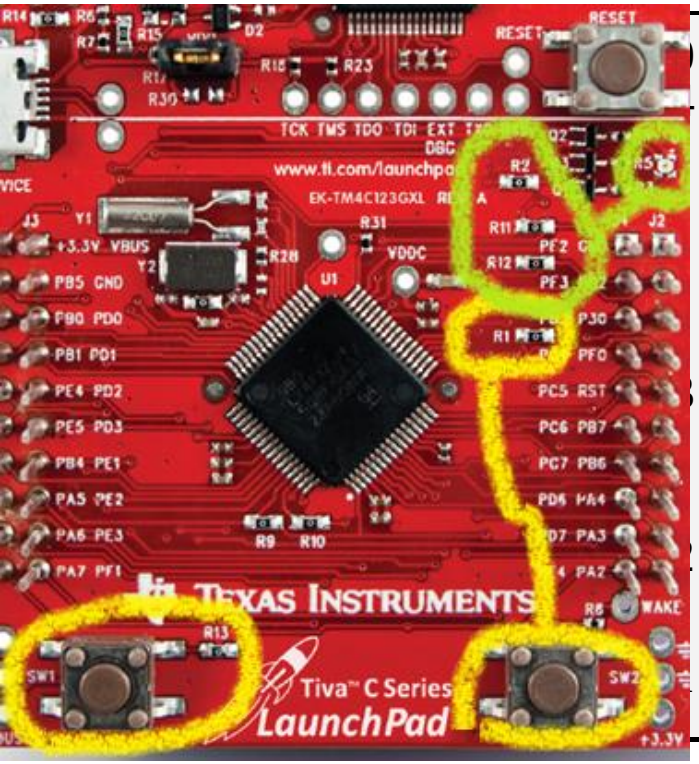


and so on and so forth.....

[A side bar] TILaunchPad_TM4C123GH6PM_Schematic.pdf



LaunchPad Switches and LEDs



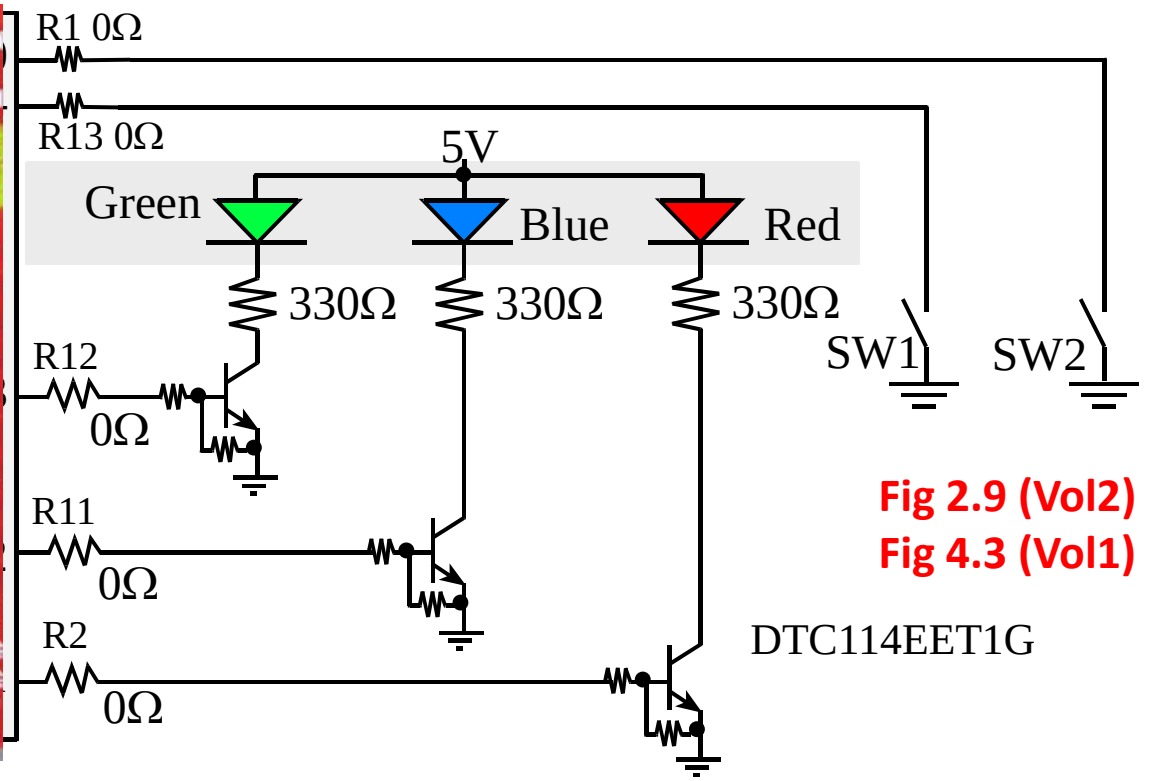
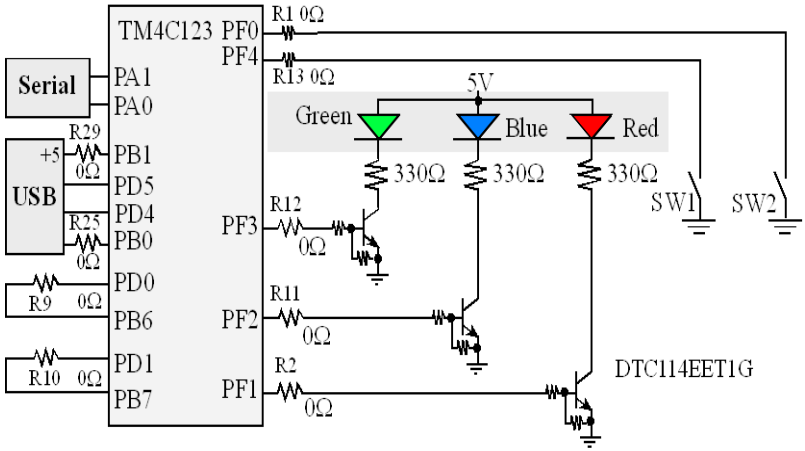


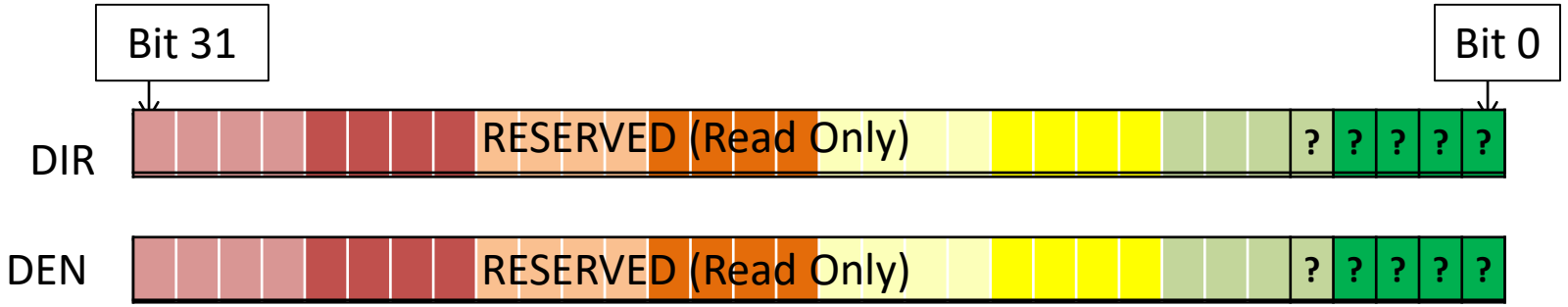
Fig 2.9 (Vol2)
Fig 4.3 (Vol1)

- **The switches on the LaunchPad**
  - **Negative logic**
  - **Require internal pull-up (set bits in the Pull-Up Register)**
- **The PF3-1 LEDs are positive logic**

PortF Register (5 pins -> bit 4-0)

- Input: Q. Switches or LEDs?
  - Switches
  - PF0 and PF4
- Output: Q. Switches or LEDs?
  - LEDs
  - PF1-3





and so on and so forth.....

Initialization Procedural: GPIO portF

- **Initialization (executed once at beginning)**
  1. Turn on Port F clock in **SYSCTL_RCGCGPIO_R**
     Wait two bus cycles (delay): stabilization
  2. Unlock PF0 (PD7 also needs unlocking)→CR

Table 10-10. GPIO Pins With Special Considerations

GPIO Pins	Default Reset State	GPIOAFSEL	GPIODEN	GPIOPDR	GPIOPUR	GPIOPCTL	GPIOCR
PA[1:0]	UART0	0	0	0	0	0x1	1
PA[5:2]	SSI0	0	0	0	0	0x2	1
PB[3:2]	$I^{21}C0$	0	0	0	0	0x3	1
PC[3:0]	JTAG/SWD	1	1	0	1	0x1	0
PD[7]	GPIO[a]	0	0	0	0	0x0	0
PF[0]	GPIO[a]	0	0	0	0	0x0	0

a. This pin is configured as a GPIO by default but is locked and can only be reprogrammed by unlocking the pin in the GPIOLOCK register and uncommitting it by setting the GPIOCR register.

The GPIO commit control registers provide a layer of protection against accidental programming of critical hardware peripherals.

Initialization Procedural: GPIO portF

- **Initialization (executed once at beginning)**
  1. Turn on Port F clock in **SYSCTL_RCGCGPIO_R**
     Wait two bus cycles (delay): stabilization
  2. Unlock PF0 (PD7 also needs unlocking)→CR
  3. Clear *AMSEL* to disable analog
  4. Clear *PCTL* to select GPIO
  5. Set *DIR* to 0 for input, 1 for output
  6. Clear *AFSEL* bits to 0 to select regular I/O

  Set *PUR* bits to 1 to enable internal pull-up
  7. Set *DEN* bits to 1 to enable data pins
- **Input from switches, output to LED:** Read/write **GPIO_PORTF_DATA_R**

TM4C123 I/O registers
GPIO base address p659

- GPIO Port A (APB): 0x4000.4000
- GPIO Port B (APB): 0x4000.5000
- GPIO Port C (APB): 0x4000.6000
- GPIO Port D (APB): 0x4000.7000
- GPIO Port E (APB): 0x4002.4000
- GPIO Port F (APB): 0x4002.5000

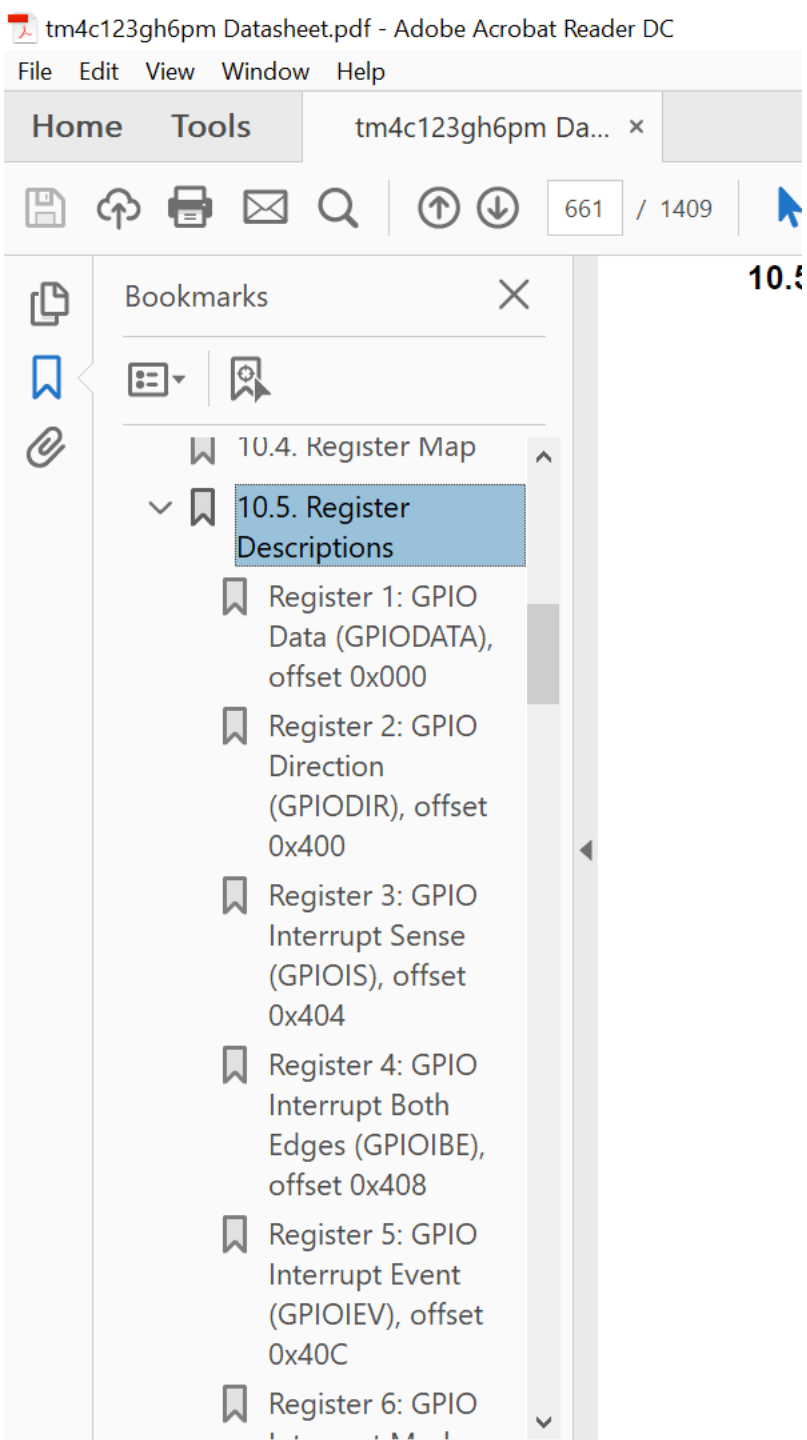
base+$000	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	GPIO_PORTx_DATA_R
base+$400	DIR	DIR	DIR	DIR	DIR	DIR	DIR	DIR	GPIO_PORTx_DIR_R
base+$420	AFSEL	AFSEL	AFSEL	AFSEL	AFSEL	AFSEL	AFSEL	AFSEL	GPIO_PORTx_AFSEL_R
base+$510	PUE	PUE	PUE	PUE	PUE	PUE	PUE	PUE	GPIO_PORTx_PUR_R
base+$51C	DEN	DEN	DEN	DEN	DEN	DEN	DEN	DEN	GPIO_PORTx_DEN_R
base+$524	CR	CR	CR	CR	CR	CR	CR	CR	GPIO_PORTx_CR_R
base+$528	AMSEL	AMSEL	AMSEL	AMSEL	AMSEL	AMSEL	AMSEL	AMSEL	GPIO_PORTx_AMSEL_R

Example address for GPIO PORTF

Address	7	6	5	4	3	2	1	0	Name
400F.E608	-	-	**GPIOF**	GPIOE	GPIOD	GPIOC	GPIOB	GPIOA	SYSCTL_RCGCGPIO_R
4002.5000	-	-	-	DATA	DATA	DATA	DATA	DATA	GPIO_PORTF_DATA_R
4002.5400	-	-	-	DIR	DIR	DIR	DIR	DIR	GPIO_PORTF_DIR_R
4002.5420	-	-	-	SEL	SEL	SEL	SEL	SEL	GPIO_PORTF_AFSEL_R
4002.551C	-	-	-	DEN	DEN	DEN	DEN	DEN	GPIO_PORTF_DEN_R

Something is weird..

Look at offsets for GPIO registers



I/O Port Bit-Specific (vol1 p156, vol2 p107)

- I/O Port bit-specific addressing is used to access port data register
  - Define address offset as $4*2^b$, where b is the selected bit position
  - Add offsets for each bit selected to base address for the port

Students who want to understand the mechanism behind this method, Search for page 654 and 662

If we wish to access bit	*Constant*
7	0x0200
6	0x0100
5	0x0080
4	0x0040 → $4*2^4$
3	0x0020
2	0x0010
1	0x0008
0	0x0004 → $4*2^0$

- Example: PF4 and PF0

Port F = 0x4002.5000

0x4002.5000+0x0004+0x0040

= 0x4002.5044

- What's the meaning of 0x3FC added on port data register's base??

#define GPIO_PORTE_DATA_R (*((volatile unsigned long *)0x4002.43FC))

662 / 1409 95% offset 0x000

Register 1: GPIO Data (GPIODATA), offset 0x000

The **GPIODATA** register is the data register. In software control mode, values written in the **GPIODATA** register are transferred onto the GPIO port pins if the respective pins have been configured as outputs through the **GPIO Direction (GPIODIR)** register (see page 663).

In order to write to **GPIODATA**, the corresponding bits in the mask, resulting from the address bus bits [9:2], must be set. Otherwise, the bit values remain unchanged by the write.

Similarly, the values read from this register are determined for each bit by the mask bit derived from the address used to access the data register, bits [9:2]. Bits that are set in the address mask cause the corresponding bits in **GPIODATA** to be read, and bits that are clear in the address mask cause the corresponding bits in **GPIODATA** to be read as 0, regardless of their value.

A read from **GPIODATA** returns the last bit value written if the respective pins are configured as outputs, or it returns the value on the corresponding input pin when these are configured as inputs. All bits are cleared by a reset.

GPIO Data (GPIODATA)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (APB) base: 0x4002.5000
GPIO Port F (AHB) base: 0x4005.D000
Offset 0x000
Type RW, reset 0x0000.0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								DATA							
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW

#define GPIO_PORTE_DATA_R (*((volatile unsigned long *)0x4002.43FC))

≠

#define GPIO_PORTE_DATA_R (*((volatile unsigned long *)0x4002.4000))

- What happens if we write to location GPIO Port E 0x4002.4000?
  - Nothing happens as specific bits are not selected. None of the port bits are affected.

If we wish to access bit	Constant
7	0x0200
6	0x0100
5	0x0080
4	0x0040
3	0x0020
2	0x0010
1	0x0008
0	0x0004

Table 1. Base Addresses for bit-specific addressing of ports A-F

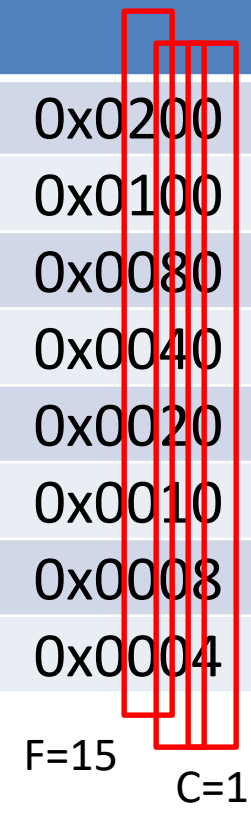
Port	Base address
PortA	0x4000.4000
PortB	0x4000.5000
PortC	0x4000.6000
PortD	0x4000.7000
PortE	0x4002.4000
PortF	0x4002.5000

Table 2. Address offsets used to specify individual data port bits

#define GPIO_PORTE_DATA_R (*((volatile unsigned long *)0x4002.43FC))

- To read and write all 8 bits of a port: sum all 8 offset constants,
  - The offset address of 0x3FC
  - 0011.1111.1100 in binary
- Port A-D: 8 pins (0-7 bit)
- Port E: 6 pins (0-5 bit)
- Port F 5 pins (0-4 bit)
  - We just use offset 0x3FC to access all bits of each port, but need to know the true meaning

If we wish to access bit	Constant
7	0x0200
6	0x0100
5	0x0080
4	0x0040
3	0x0020
2	0x0010
1	0x0008
0	0x0004



```
/* Design a security system using contact switches for the sensors and flashing LED
as the alarm.  We will first prototype it on a breadboard using a LaunchPad.
There are two contact switches that measure the status of your home.
Each contact switch is positioned on a window.
If the switch is closed or pressed the window is secure, but
if either switch is open it means the window is open and the home is unsecure.
There is also a toggle switch with which the user can use to activate or deactivate
the alarm. If the alarm is activated, the LED will flash at 5 Hz if either switch is not
pressed. */
// PE4 is an output to LED output, 5 Hz flashing means alarm
// PE2 is an input from toggle switch, 1 means activate, 0 means deactivate
// PE1 is an input from contact switch, 1 means secure, 0 means unsecure
// PE0 is an input from contact switch, 1 means secure, 0 means unsecure

#define SYSCTL_RCGC2_R        (*((volatile unsigned long *)0x400FE108))
#define GPIO_PORTE_DATA_R     (*((volatile unsigned long *)0x400243FC))
#define GPIO_PORTE_DIR_R      (*((volatile unsigned long *)0x40024400))
#define GPIO_PORTE_AFSEL_R    (*((volatile unsigned long *)0x40024420))
#define GPIO_PORTE_DEN_R      (*((volatile unsigned long *)0x4002451C))
#define GPIO_PORTE_AMSEL_R    (*((volatile unsigned long *)0x40024528))
#define GPIO_PORTE_PCTL_R     (*((volatile unsigned long *)0x4002452C))
```

If we wish to access bit	Constant
7	0x0200
6	0x0100
5	0x0080
4	0x0040
3	0x0020
2	0x0010
1	0x0008
0	0x0004

Port	Base address
PortE	0x4002.4000

- More proper way of address define: do not touch other bits
- 0x4002.4000+0x004+0x008+ 0x010+0x040 = 0x4002.405C

```
#define GPIO_PORTE_DATA_R
(*((volatile unsigned long *)  0x4002.405C))
```

```
#define PE0     (*((volatile unsigned long *)
0x4002.4004))
#define PE1     (*((volatile unsigned long *)
0x4002.4008))
#define PE2     (*((volatile unsigned long *)
0x4002.4010))
#define PE4     (*((volatile unsigned long *)
0x4002.4040))
```

We call this ***"bit-specific addressing"***

The GPIO Data Register

- Supports ***"bit-specific addressing"***
- Located at the offset address of from 0x000 to 0x3FF; total #256 (256 possibilities) data register address
  - 0-3FF: $2^8*2^2=256*4$ (more than 256, why?)
  - Last two bits (0 and 1) are masked resulting from the address bus (bits [9:2])
  - PortE address 0x4002.43FC is one case of 256

The 256 addresses are not just addresses but contain valid bit information

A sidebar to use “TM4C123GH6PM.h” and “tm4c123gh6pm.h”

```
/* Design a security system using contact switches for the sensors and flashing LED
as the alarm.  We will first prototype it on a breadboard using a LaunchPad.
There are two contact switches that measure the status of your home.
Each contact switch is positioned on a window.
If the switch is closed or pressed the window is secure, but
if either switch is open it means the window is open and the home is unsecure.
There is also a toggle switch with which the user can use to activate or deactivate
the alarm. If the alarm is activated, the LED will flash at 5 Hz if either switch is not
pressed. */
// PE4 is an output to LED output, 5 Hz flashing means alarm
// PE2 is an input from toggle switch, 1 means activate, 0 means deactivate
// PE1 is an input from contact switch, 1 means secure, 0 means unsecure
// PE0 is an input from contact switch, 1 means secure, 0 means unsecure

#include "TExaS.h"
#define SYSCTL_RCGC2_R        (*((volatile unsigned long *)0x400FE108))
#define GPIO_PORTE_DATA_R     (*((volatile unsigned long *)0x400243FC))
#define GPIO_PORTE_DIR_R      (*((volatile unsigned long *)0x40024400))
#define GPIO_PORTE_AFSEL_R    (*((volatile unsigned long *)0x40024420))
#define GPIO_PORTE_DEN_R      (*((volatile unsigned long *)0x4002451C))
#define GPIO_PORTE_AMSEL_R    (*((volatile unsigned long *)0x40024528))
#define GPIO_PORTE_PCTL_R     (*((volatile unsigned long *)0x4002452C))
```

If we wish to access bit	Constant
7	0x0200
6	0x0100
5	0x0080
4	0x0040
3	0x0020
2	0x0010
1	0x0008
0	0x0004

Port	Base address
PortE	0x4002.4000

- More proper way of address define: do not touch other bits
- 0x4002.4000+0x004+0x008+ 0x010+0x040 = 0x4002.45C0

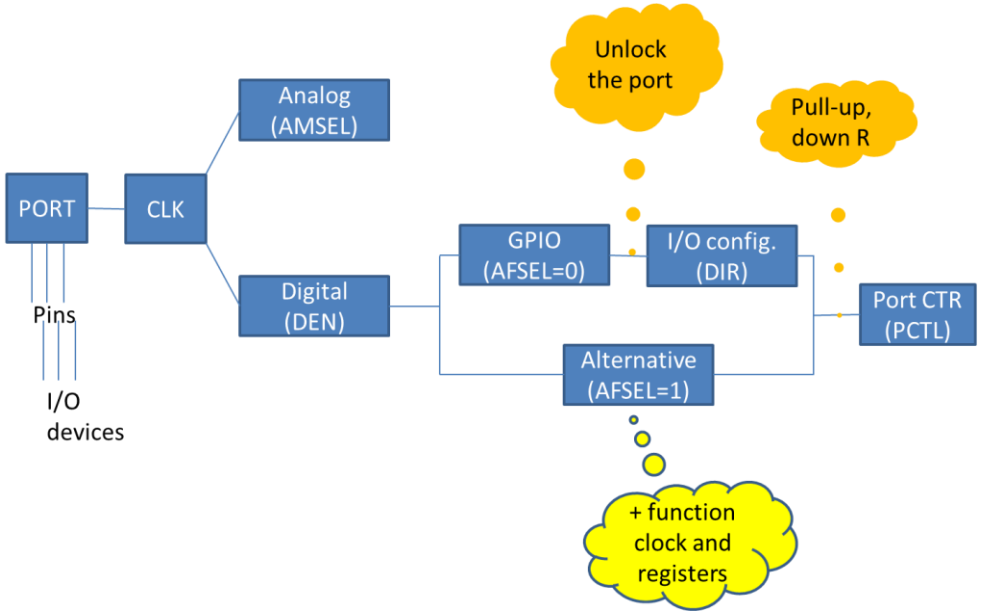
```
#define GPIO_PORTE_DATA_R
(*((volatile unsigned long *)  0x4002.45C0))
```

```
#define PE0     (*((volatile unsigned long *)
0x4002.4004))
#define PE1     (*((volatile unsigned long *)
0x4002.4008))
#define PE2     (*((volatile unsigned long *)
0x4002.4010))
#define PE4     (*((volatile unsigned long *)
0x4002.4040))
```

We call this ***"bit-specific addressing"***

- Use “tm4c123gh6pm.h” provided Valvano to keep the naming convention that we have used so far
- Another idea to define address for registers of each port? We several ports

➔ Solution: Base address + offset



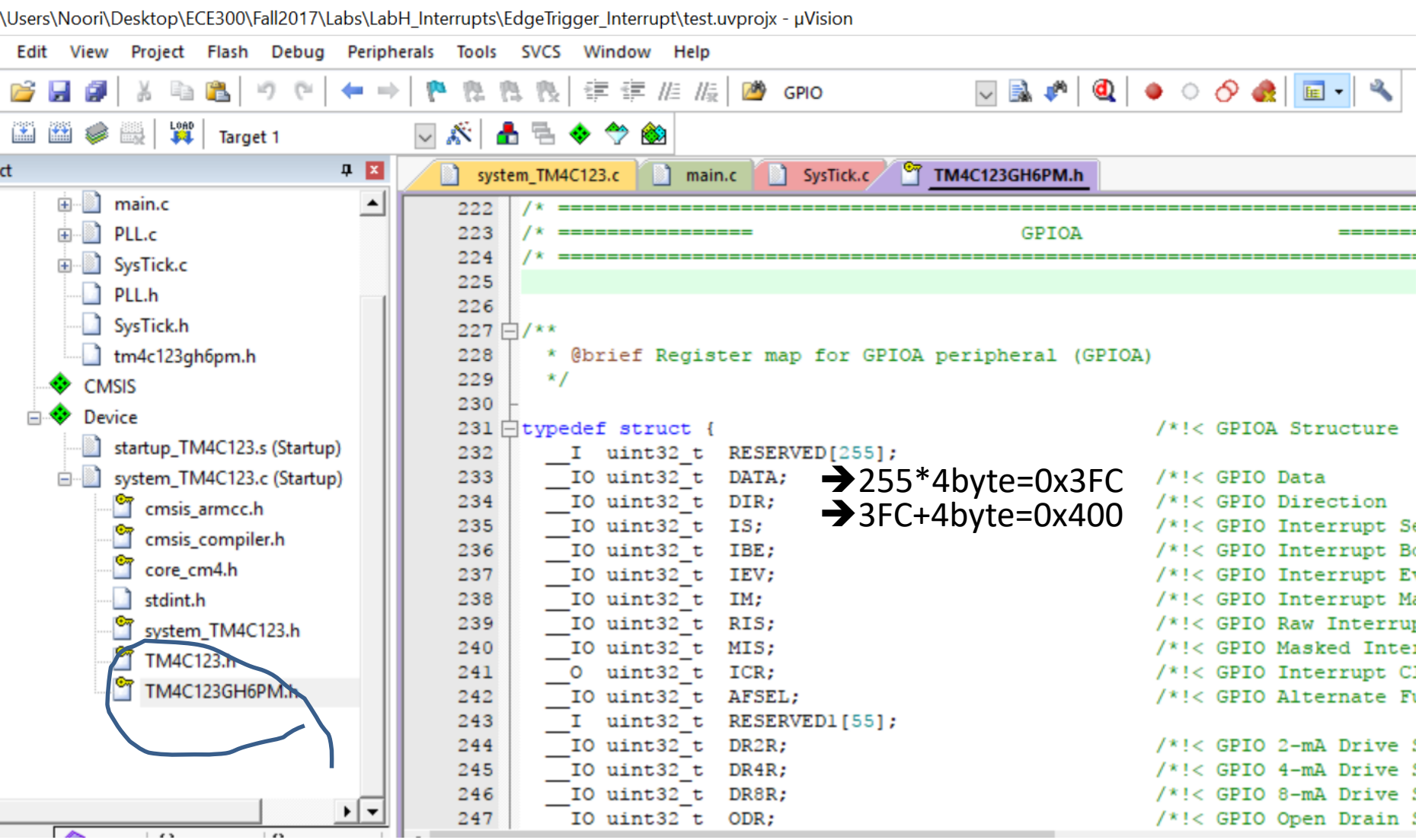
Address range (memory map)

Table 2-4. Memory Map

Start	End	Description	For details, see page ...
Memory			
0x0000.0000	0x0003.FFFF	On-chip Flash	540
0x0004.0000	0x1FFF.FFFF	Reserved	-
0x2000.0000	0x2000.7FFF	Bit-banded on-chip SRAM	525
0x2000.8000	0x21FF.FFFF	Reserved	-
0x2200.0000	0x220F.FFFF	Bit-band alias of bit-banded on-chip SRAM starting at 0x2000.0000	525
0x2210.0000	0x3FFF.FFFF	Reserved	-
Peripherals			
0x4000.0000	0x4000.0FFF	Watchdog timer 0	776
0x4000.1000	0x4000.1FFF	Watchdog timer 1	776
0x4000.2000	0x4000.3FFF	Reserved	-
0x4000.4000	0x4000.4FFF	GPIO Port A	658
0x4000.5000	0x4000.5FFF	GPIO Port B	658

……etc

TM4C123GH6PM.h (partial-GPIO)



#define GPIO_PORTE_DATA_R (*((volatile unsigned long *)0x4002.43FC))

- To read and write all 8 bits of a port: sum all 8 offset constants,
  - The offset address of 0x3FC
  - 0011.1111.1100 in binary
- Port A-D: 8 pins (0-7 bit)
- Port E: 6 pins (0-5 bit)
- Port F 5 pins (0-4 bit)
  - But for our convenient, we can just use offset 0x3FC to access all bits of each port

If we wish to access bit	Constant
7	0x0200
6	0x0100
5	0x0080
4	0x0040
3	0x0020
2	0x0010
1	0x0008
0	0x0004

3

F=15

C=12

Register 2: GPIO Direction (GPIODIR), offset 0x400

The **GPIODIR** register is the data direction register. Setting a bit in the **GPIODIR** register configures the corresponding pin to be an output, while clearing a bit configures the corresponding pin to be an input. All bits are cleared by a reset, meaning all GPIO pins are inputs by default.

GPIO Direction (GPIODIR)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000

- We can determine register address by looking at board header file such as TM4C123GH6PM.h
- Peripheral registers for specific port module can be determined by knowing
  - Port base address (i.e., port 0) and
  - Offset address for the specific register (i.e., DIR)

```
#define    GPIO_PORTA_DIR_R        (*((volatile unsigned long *) 0x40004400))
```

Another way to access contents of data

#define GPIO_PORTA_DIR_R (*((volatile unsigned long *) 0x40004400))

- Use a very well defined structure!!
  - i.e., GPIOA→DIR

```
typedef struct {                                /*!< GPIOA Structure
  __I  uint32_t  RESERVED[255];
  __IO uint32_t  DATA;                          /*!< GPIO Data
  __IO uint32_t  DIR;                           /*!< GPIO Direction
  __IO uint32_t  IS;                            /*!< GPIO Interrupt Sense
  __IO uint32_t  IBE;                           /*!< GPIO Interrupt Both Edges
  __IO uint32_t  IEV;                           /*!< GPIO Interrupt Event
  __IO uint32_t  IM;                            /*!< GPIO Interrupt Mask
  __IO uint32_t  RIS;                           /*!< GPIO Raw Interrupt Status
  __IO uint32_t  MIS;                           /*!< GPIO Masked Interrupt Status
  __O  uint32_t  ICR;                           /*!< GPIO Interrupt Clear
  __IO uint32_t  AFSEL;                         /*!< GPIO Alternate Function Select
  __I  uint32_t  RESERVED1[55];
  __IO uint32_t  DR2R;                          /*!< GPIO 2-mA Drive Select
  __IO uint32_t  DR4R;                          /*!< GPIO 4-mA Drive Select
  __IO uint32_t  DR8R;                          /*!< GPIO 8-mA Drive Select
  __IO uint32_t  ODR;                           /*!< GPIO Open Drain Select
  __IO uint32_t  PUR;                           /*!< GPIO Pull-Up Select
  __IO uint32_t  PDR;                           /*!< GPIO Pull-Down Select
  __IO uint32_t  SLR;                           /*!< GPIO Slew Rate Control Select
  __IO uint32_t  DEN;                           /*!< GPIO Digital Enable
  __IO uint32_t  LOCK;                          /*!< GPIO Lock
  __IO uint32_t  CR;                            /*!< GPIO Commit
  __IO uint32_t  AMSEL;                         /*!< GPIO Analog Mode Select
  __IO uint32_t  PCTL;                          /*!< GPIO Port Control
  __IO uint32_t  ADCCTL;                        /*!< GPIO ADC Control
  __IO uint32_t  DMACTL;                        /*!< GPIO DMA Control
} GPIOA_Type;
```

TM4C123GH6PM.h

```
1539  /* =========================================================================
1540  /* ================                  Peripheral memory map
1541  /* =========================================================================
1542
1543  #define WATCHDOG0_BASE                  0x40000000UL
1544  #define WATCHDOG1_BASE                  0x40001000UL
1545  #define GPIOA_BASE                      0x40004000UL
1546  #define GPIOB_BASE                      0x40005000UL
1547  #define GPIOC_BASE                      0x40006000UL
1548  #define GPIOD_BASE                      0x40007000UL
1549  #define SSI0_BASE                       0x40008000UL
 1601
 1602
 1603  /* ========================================================================== */
 1604  /* ================           Peripheral declaration           ================ */
 1605  /* ========================================================================== */
 1606
 1607  #define WATCHDOG0                       ((WATCHDOG0_Type       *) WATCHDOG0_BASE)
 1608  #define WATCHDOG1                       ((WATCHDOG0_Type       *) WATCHDOG1_BASE)
 1609  #define GPIOA                           ((GPIOA_Type           *) GPIOA_BASE)
 1610  #define GPIOB                           ((GPIOA_Type           *) GPIOB_BASE)
```

GPIOA_Type located at GPIOA_BASE

➔GPIOA defines a base address of GPIO port A

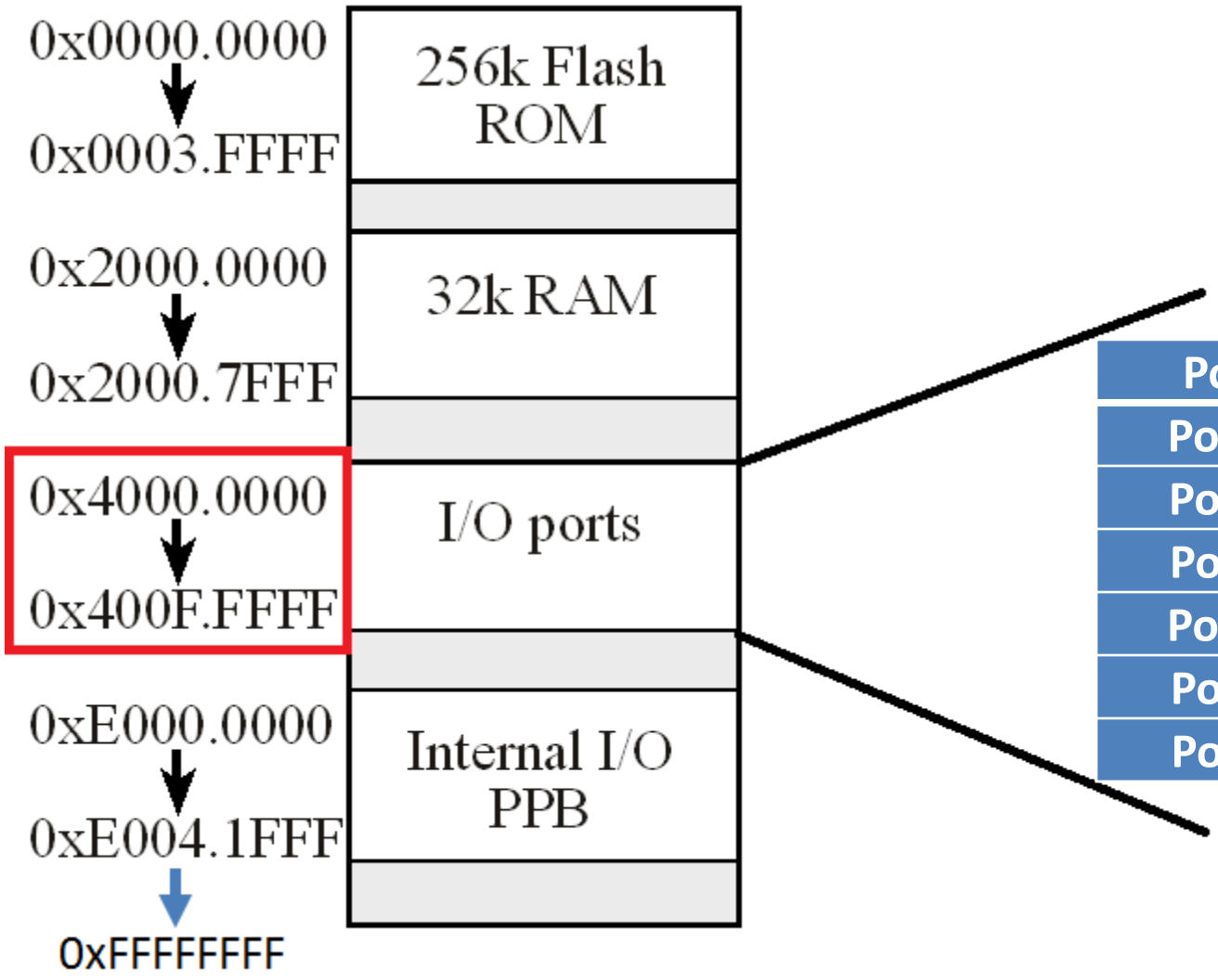
GPIOA_Type is a struct

- When it's aligned to that **base address**, each of the struct's fields will align with a **particular register** in the GPIO memory map

This allows us to write to the GPIO DIR Registers without using a raw memory location

- **GPIOA->DIR /* stuff you want to do */ ;**
  /* =, \=, &=~,…. */

i.e., GPIOA->DIR |= 0x03;



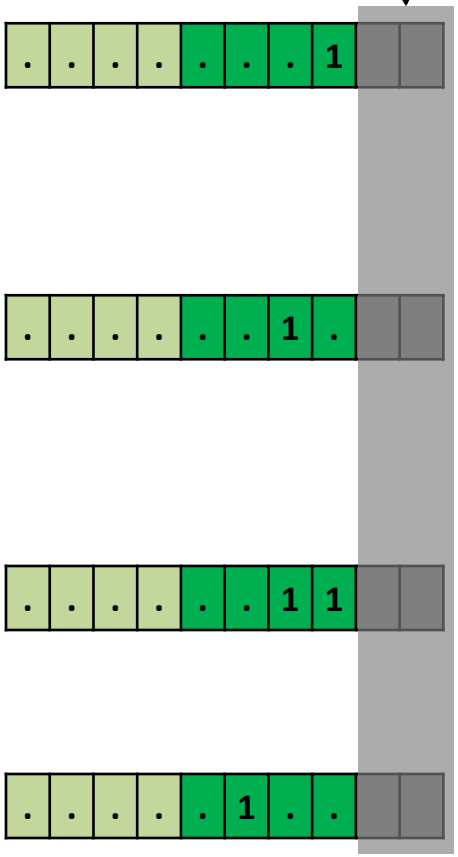
Port	Base address
PortA	0x4000.4000
PortB	0x4000.5000
PortC	0x4000.6000
PortD	0x4000.7000
PortE	0x4002.4000
PortF	0x4002.5000

- Datasheet page 654-62

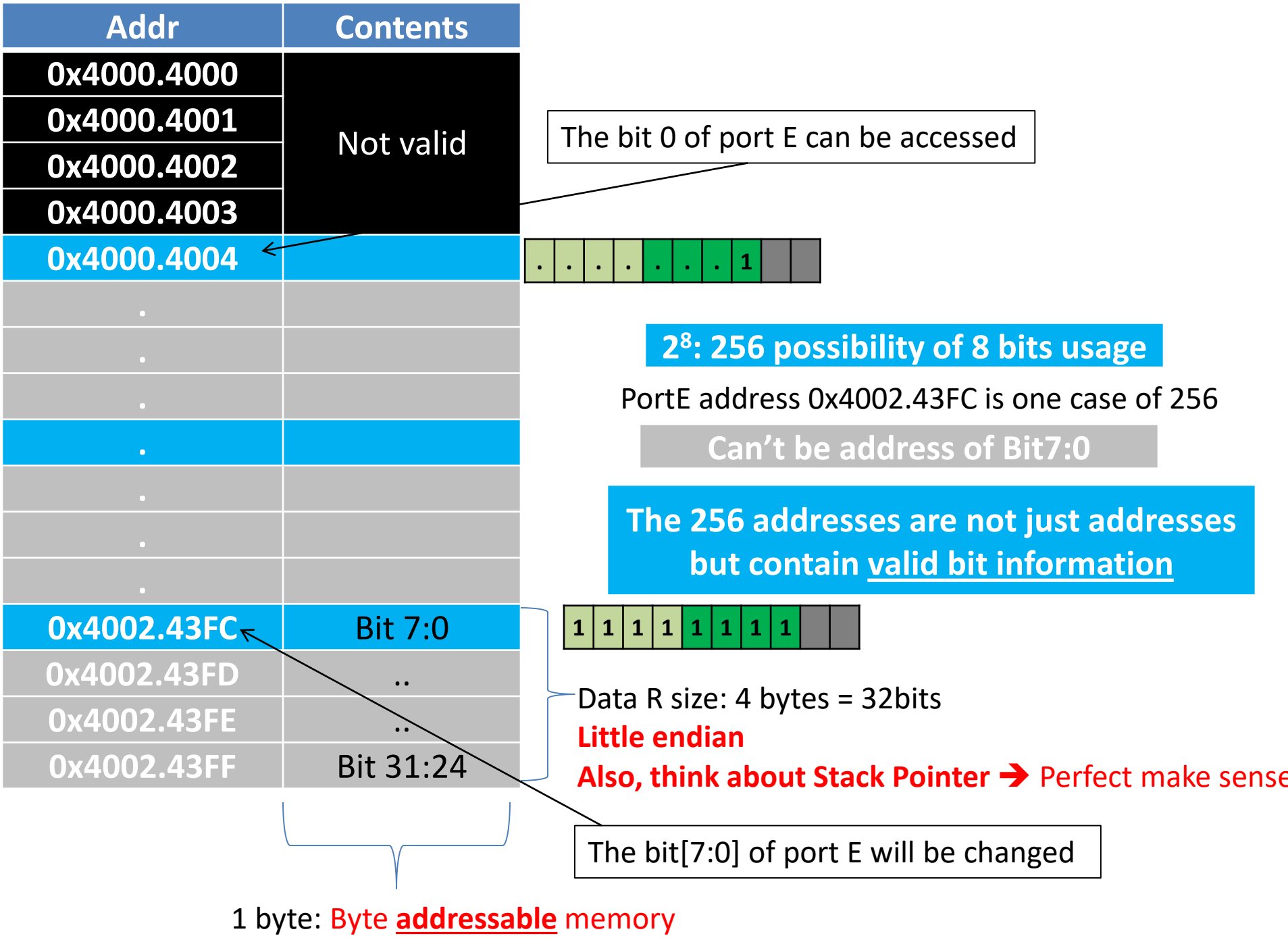
Port A base

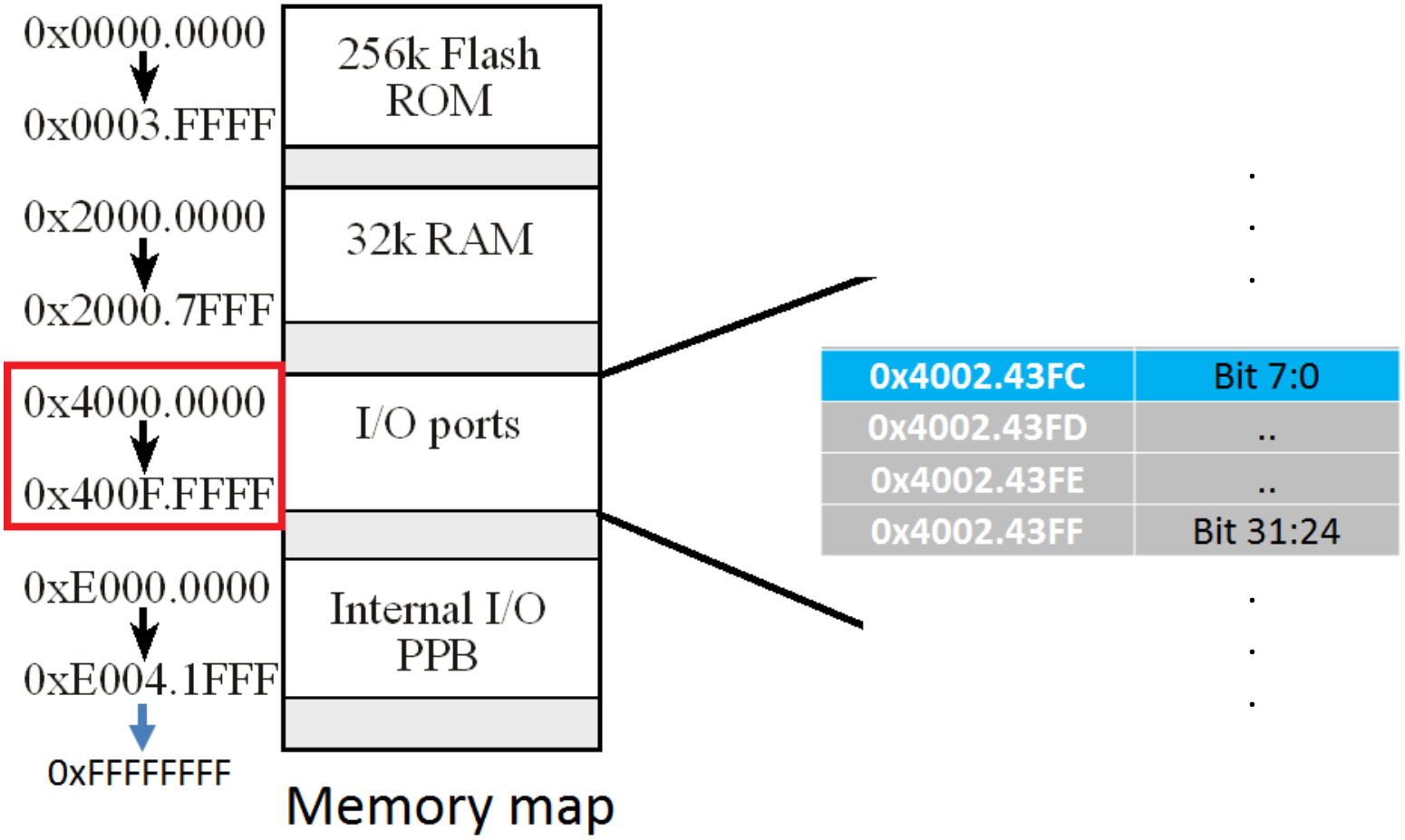
Addr	Contents
0x4000.4000	Not valid
0x4000.4001	
0x4000.4002	
0x4000.4003	
0x4000.4004	1 byte
0x4000.4005	.
0x4000.4006	.
0x4000.4007	.
0x4000.4008	
0x4000.4009	
0x4000.400A	
0x4000.400B	
0x4000.400C	
0x4000.400D	
0x4000.400F	
0x4000.4010	
0x4000.4011	

Acts and Looks like there are masked, hidden bits in the DATA register



If we wish to access bit	Constant
7	0x0200
6	0x0100
5	0x0080
4	0x0040
3	0x0020
2	0x0010
1	0x0008
0	0x0004

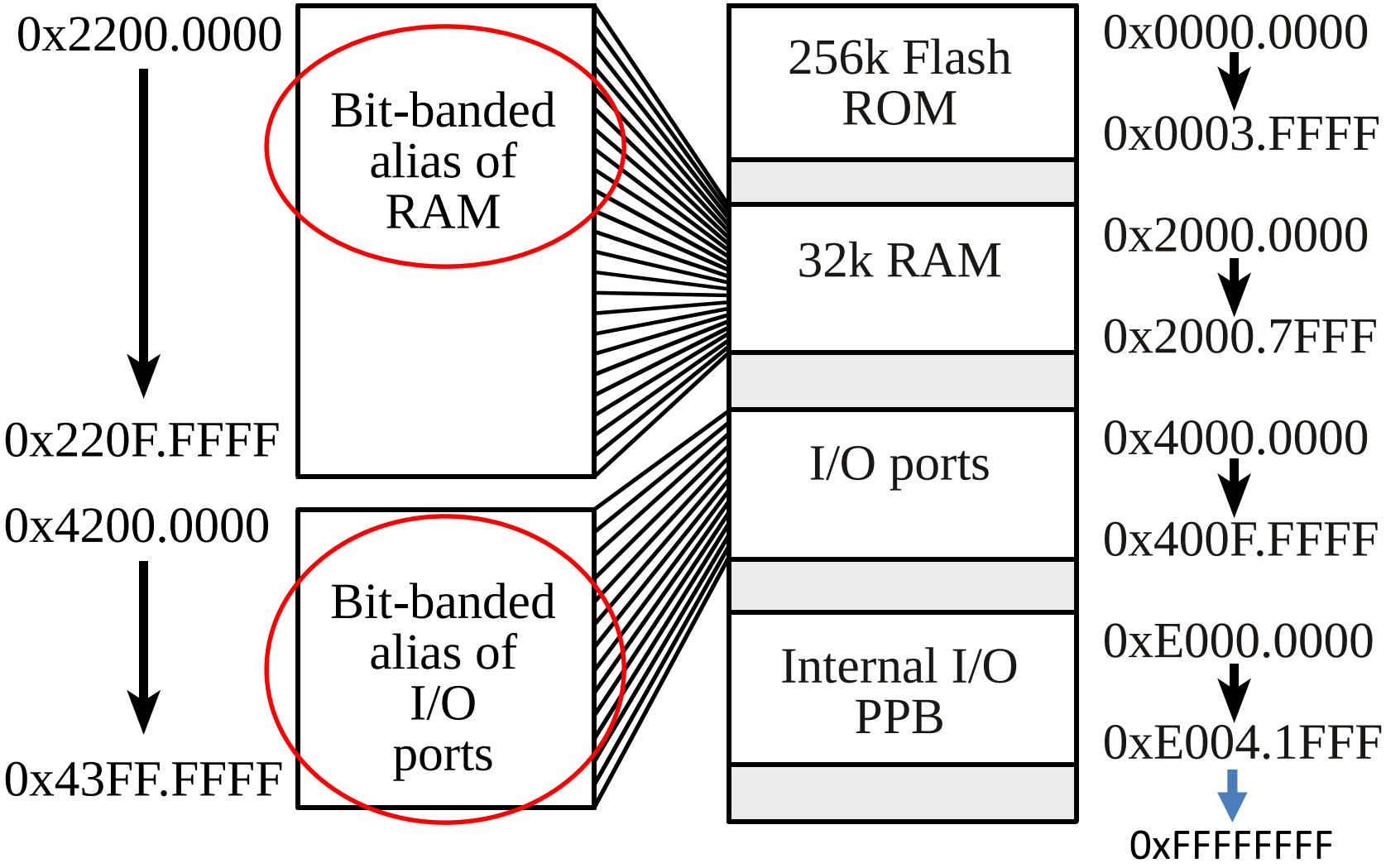

Addr
Contents
0x4000.4000
0x4000.4001
0x4000.4002
0x4000.4003
Not valid
0x4000.4004
0x4002.43FC
Bit 7:0
0x4002.43FD
..
0x4002.43FE
..
0x4002.43FF
Bit 31:24
The bit 0 of port E can be accessed
. . . . 1
2^8: 256 possibility of 8 bits usage
PortE address 0x4002.43FC is one case of 256
Can't be address of Bit7:0
The 256 addresses are not just addresses but contain valid bit information
1 1 1 1 1 1 1 1
Data R size: 4 bytes = 32bits
Little endian
Also, think about Stack Pointer ➔ Perfect make sense
The bit[7:0] of port E will be changed
1 byte: Byte addressable memory



In case of stack (somewhere on RAM area)

- **LIFO**
- **SP points the top of stack**

ARM Cortex uses **Bit-banding**



How does Bit-binding work?

- It allows to access individual bits in RAM and I/O
- Bit-banded alias region for RAM and I/O in the memory map (n: offset, b:bit)
  - RAM: 0x2200.0000+32*n+4*b
  - I/O: 0x4200.0000+32*n+4*b

Original:
RAM: 0x2000.0000
I/O: 0x4000.0000

Q. What Bit-binding address do you use to access bit 20 of the word at 0x2000.1000?

0x2200.0000+32*0x1000+4*20 = 0x2200.0000+0x20**x**0x1000+0x4**x**0x14
= 0x2200.0000 + 0x2.0000 + 0x50
= 0x2202.0050

How does Bit-binding work?

- It allows to access individual bits in RAM and I/O
- Bit-banded alias region for RAM and I/O in the memory map (n: offset, b:bit)
  - RAM: 0x2200.0000+32*n+4*b
  - I/O: 0x4200.0000+32*n+4*b

Q. What is a Bit-binding alias address to access bit 2 of the byte at 0x4000.0003?

0x4200.0000+32*0x0003+4*2 = 0x4200.0000+0x20**x**0x0003+0x4**x**0x2
= 0x4200.0000 + 0x60+ 0x8
= 0x4200.0068

More details: pg 97-99

- The alias word at maps to bit 0 of the bit-band byte at 0x200F.FFFF:

2.4.5 Bit-Banding

A bit-band region maps each word in a bit-band alias region to a single bit in the bit-band region. The bit-band regions occupy the lowest 1 MB of the SRAM and peripheral memory regions. Accesses to the 32-MB SRAM alias region map to the 1-MB SRAM bit-band region, as shown in Table 2-6 on page 97. Accesses to the 32-MB peripheral alias region map to the 1-MB peripheral bit-band region, as shown in Table 2-7 on page 98. For the specific address range of the bit-band regions, see Table 2-4 on page 92.

Note: A word access to the SRAM or the peripheral bit-band alias region maps to a single bit in the SRAM or peripheral bit-band region.

A word access to a bit band address results in a word access to the underlying memory, and similarly for halfword and byte accesses. This allows bit band accesses to match the access requirements of the underlying peripheral.

Table 2-6. SRAM Memory Bit-Banding Regions

Address Range		

- Why do we always have a “while(1){ }” loop inside of embedded C programs?
  - At the end of “int main(void)” functions, even in a simple program, an implementation is free to choose either
    - Reset (Restart) or
    - Hanging
  - We can enforce “Hanging” by the endless while loop