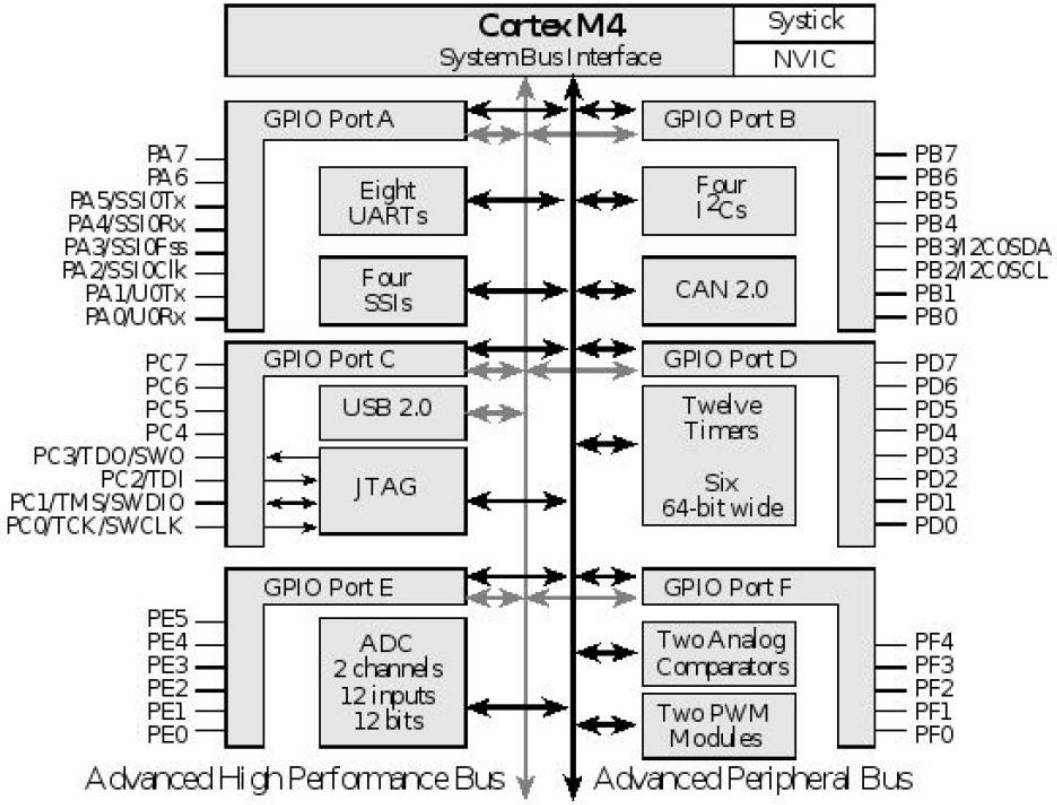
General Purpose Input-Output (GPIO)

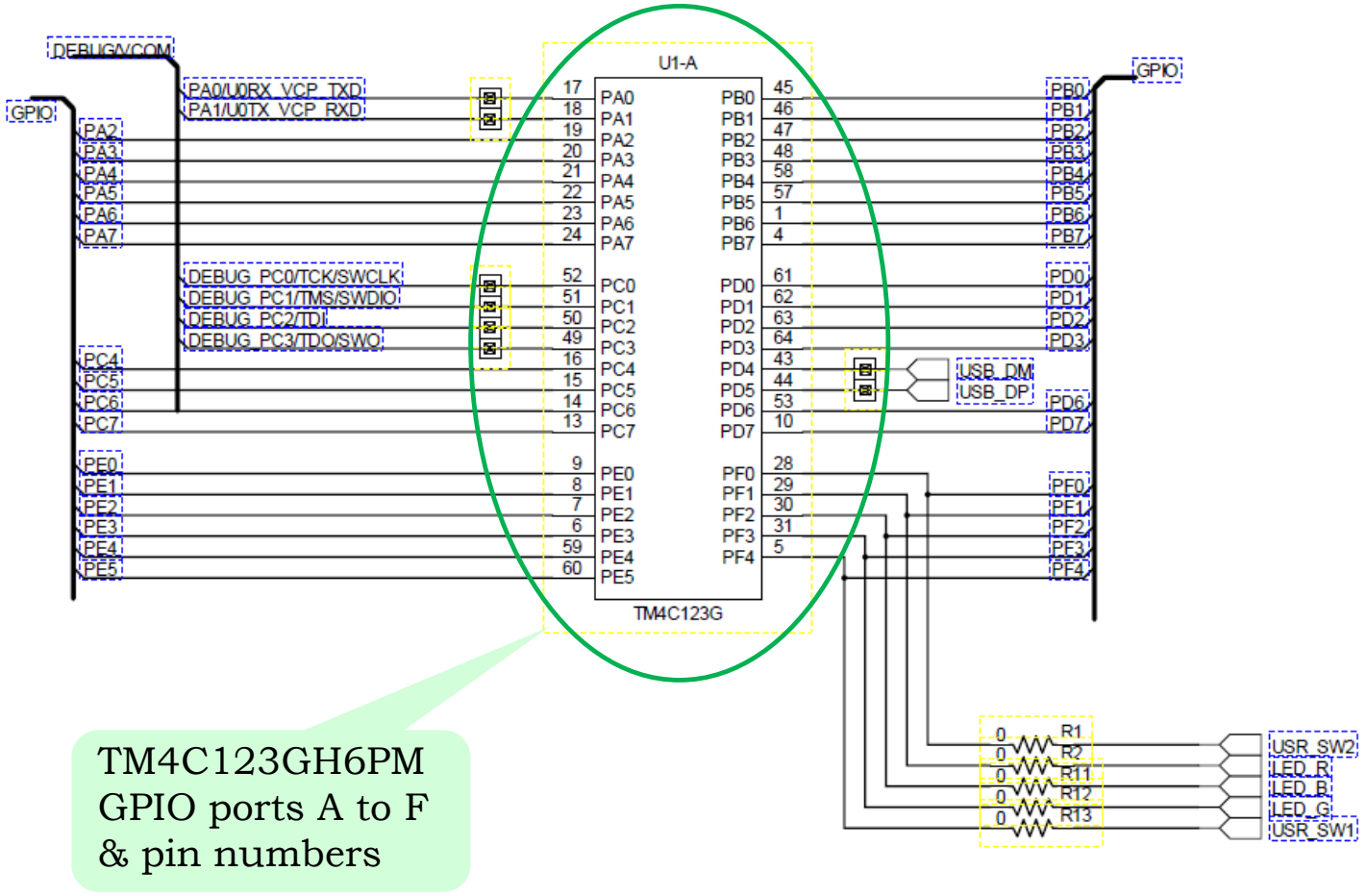
Tiva LaunchPad

LaunchPad GPIOs (Block Diagram)

- GPIO ports A to F.
- Total of 43 GPIO pins, 35 available on Tiva connectors.
- GPIO ports configurable either connected to AHB or APB bus.



LaunchPad GPIO



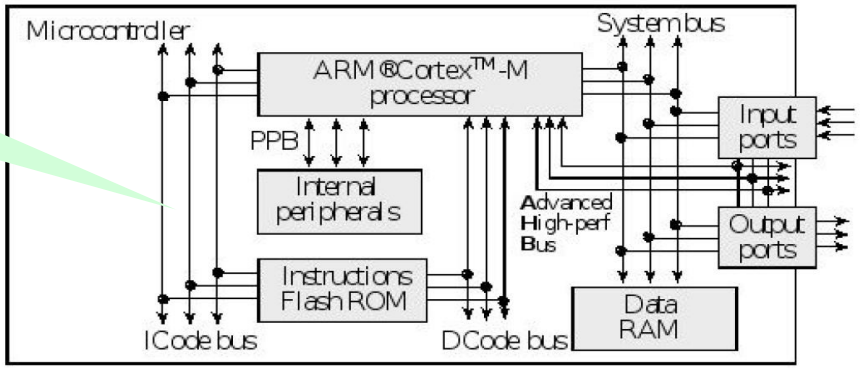
TM4C123GH6PM
GPIO ports A to F
& pin numbers

[from Tiva LaunchPad Schematics]

Multiple Buses

Separate I- & D-buses.
Q: What architecture is this?

Harvard Architecture
(separate program & data buses)



Cortex-M4 implements **parallel buses**:

- **ICode Bus**: Fetch opcode from ROM/Flash.
- **DCode Bus**: Read data from ROM.
- **System Bus**: Read/Write from RAM or I/O, fetch opcode from RAM.
- **PPB** (Private Peripheral Bus): Read/Write from internal registers, e.g. NVIC, SysTick timer.
- **AHB, APB** buses.

Advantage of use of multiple buses:

Multiple processes can be performed in same bus cycle.

E.g. I/O bus cycles can happen in parallel with Code Instruction fetches from Flash.

=> **Results in improved overall throughput.**

GPIO Pins – Features Summary

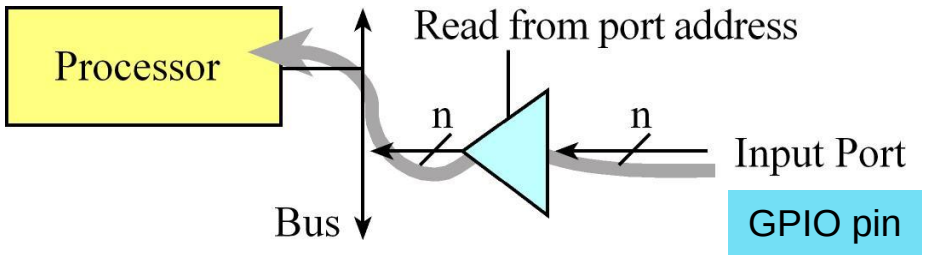
- Any GPIO pin can be a digital pin.
- Any GPIO pin can be an **Interrupt** source.
  - **Edge interrupt** on rising, falling or both edges.
  - **Level sensitive** on high or low logic levels.
- Some GPIO pins can be configured as an **external trigger** for the **ADC**.
- **5V-tolerant** input configuration – except for PD4, PD5, PB0 & PB1 which is limited to 3.6V.
- **Programmable drive strength** of 2mA (default), 4mA, or 8mA with slew rate control.
- Programmable weak pull-up, pull-down, or open-collector.
- Pin state can be retained during Hibernation mode => GPIO pin states kept intact when in power-down mode.

Digital I/O Types

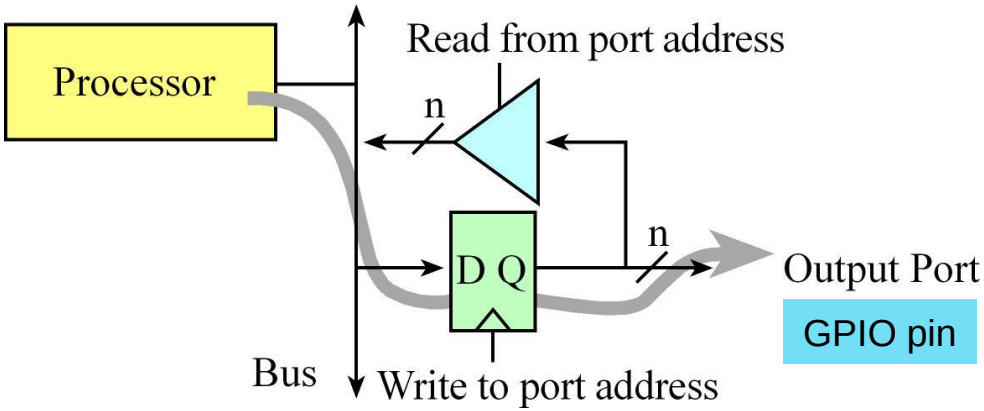
GPIODATA, GPIODIR registers

Input & Output I/O

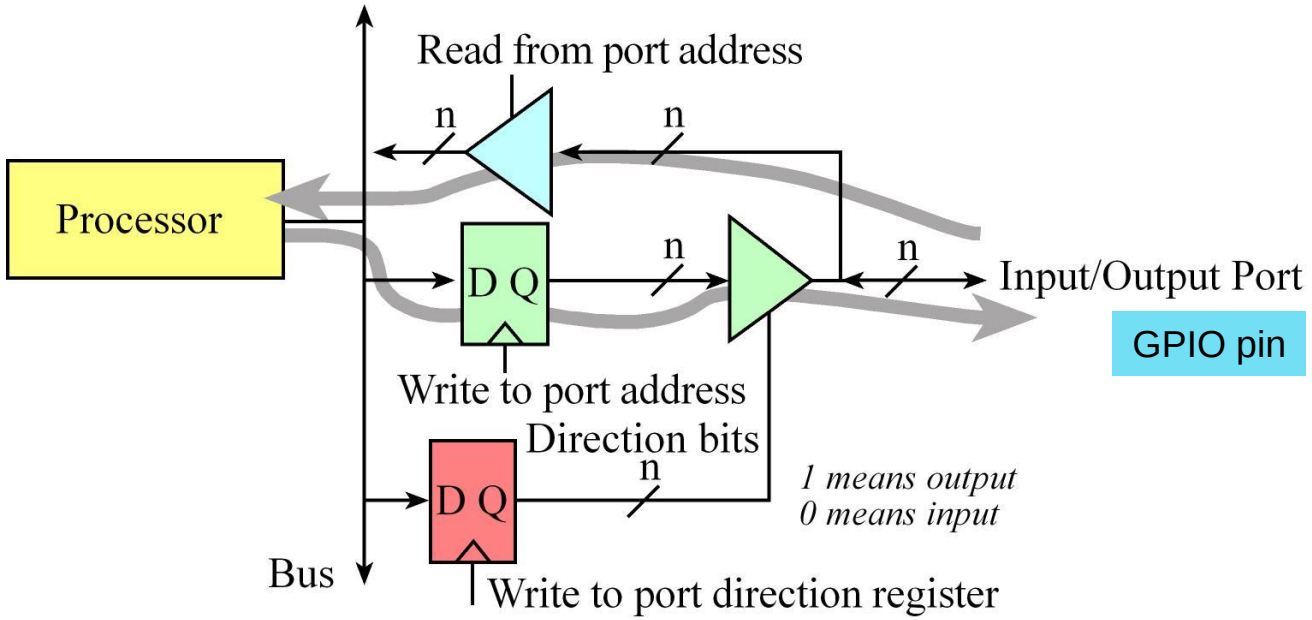
- **Input port** (Read-only):
  - Allows software to read external digital signals.



- **Readable Output port**:
  - Write data is clocked through D-FF.
  - Read cycle access returns current port pin values.
- **Note:** There is no output-only ports in the TM4C family.

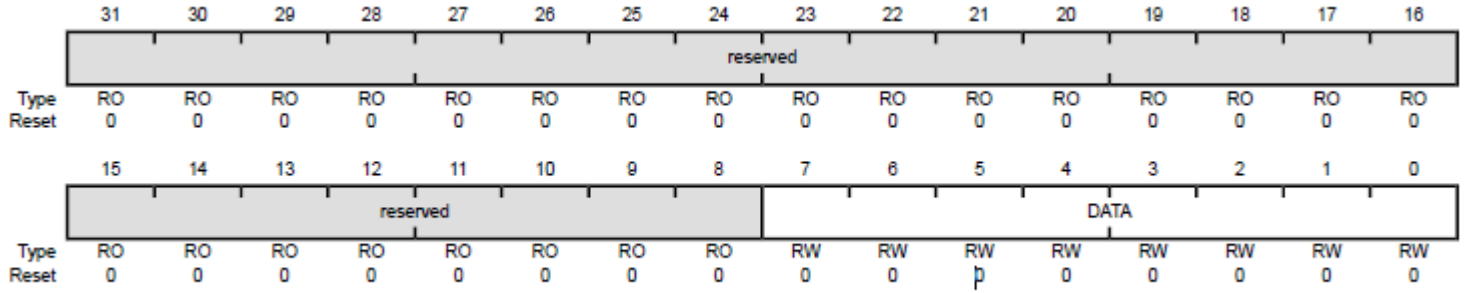


Bi-directional I/O



- **Direction register** to determine if a pin is Input or Output.
  - *Register bit: 0 = input; 1 = output.*
- Note that TM4C123 does not implement a true bi-directional GPIO port pin.

GPIO Data Register (GPIODATA)



GPIODATA register

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

GPIO Ports A to F.
Each port can be connected to either AHB or APB bus with different base address.

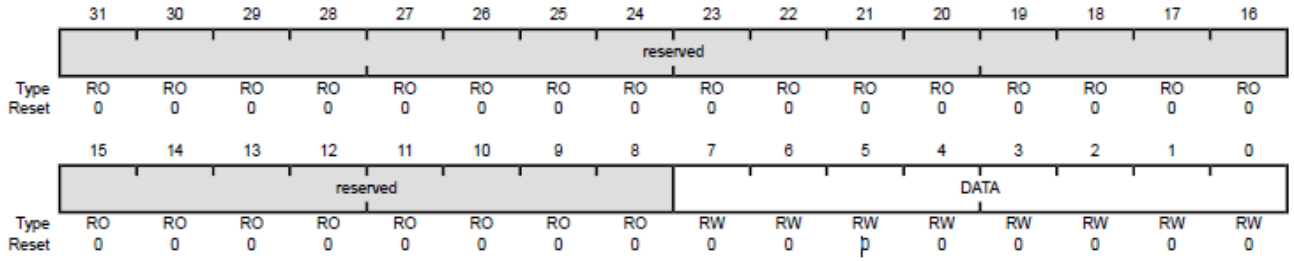
Read-Write register,
Default Value = 0 (upon reset).

*More details of **GPIODATA** register in next slide ….*

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p662)

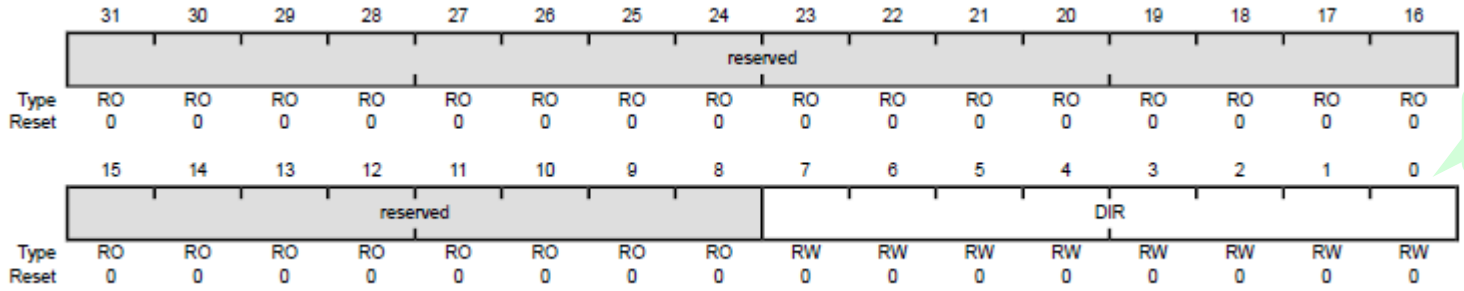
GPIO Write/Read

- Write to or Read from a GPIO port is done through **GPIODATA** register.
  - Every GPIO port has its own **GPIODATA** register.
  - Each GPIODATA register consist of **8 bits** to cater for the 8 bits in the GPIO port.
  - Values written to the Data Register is transferred to the GPIO pins if the pins are configured as outputs in the **GPIODIR** register.
  - A read from the Data Register returns the last bit value written if the pin is configured as output, or returns the value of the input pin if configured as input.
  - Upon reset, all bits in the Data Register is cleared to 0.



GPIODATA register

GPIO Port Direction (GPIODIR)



GPIO port pin number

GPIODIR register, RW, Reset = 0x0000.0000

- A GPIO port can be set to be INPUT or OUTPUT.
- The GPIO Direction (**GPIODIR**) Register configures a GPIO pin to be either input or output pin.
- **DIR:** '0' = Input; '1' = Output

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
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 0x400
Type RW, reset 0x0000.0000

Read-Write register,
Default Value = 0
=> Pins are INPUTs upon reset.

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p663)

GPIO Alternate Functions

GPIODEN, GPIOAFSEL, GPIOPCTL, GPIOAMSEL registers

GPIO Pins – Alternate Functions

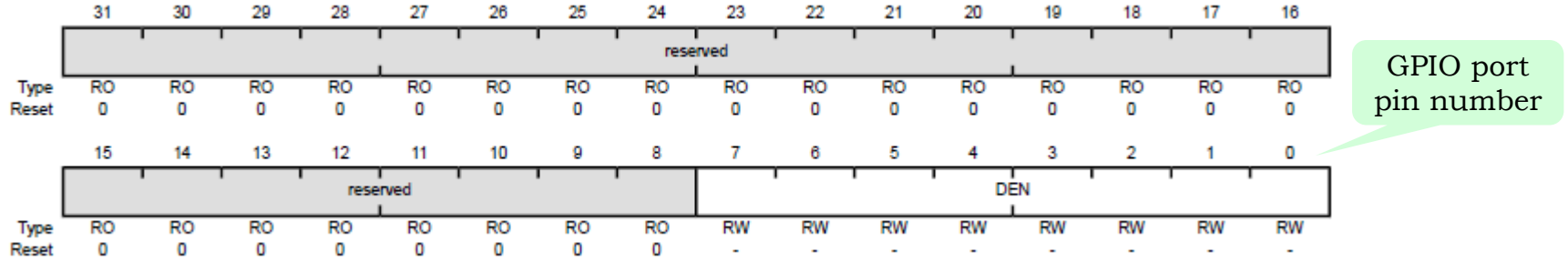
- Many of the GPIO pins can be multiplexed & programmed to be of another function.
- We called it the **Alternate Function**.
- Alternate hardware functions are enabled through the following steps:
  - Enabling alternate function in the GPIO Alternate function (**GPIOAFSEL**) register.
  - Enabling digital function in the GPIO Digital Enable (**GPIODEN**) registers,
  - Configuring the **PMCx** bits in the GPIO Port control (**GPIOPCTL**) register to it's numeric coding
    (*see table next slide*).

Note:

Alternate function is a Digital function *(versus an Analog function)*.

GPIODEN register bit needs to be set.

GPIO Digital Enable Register (GPIODEN)

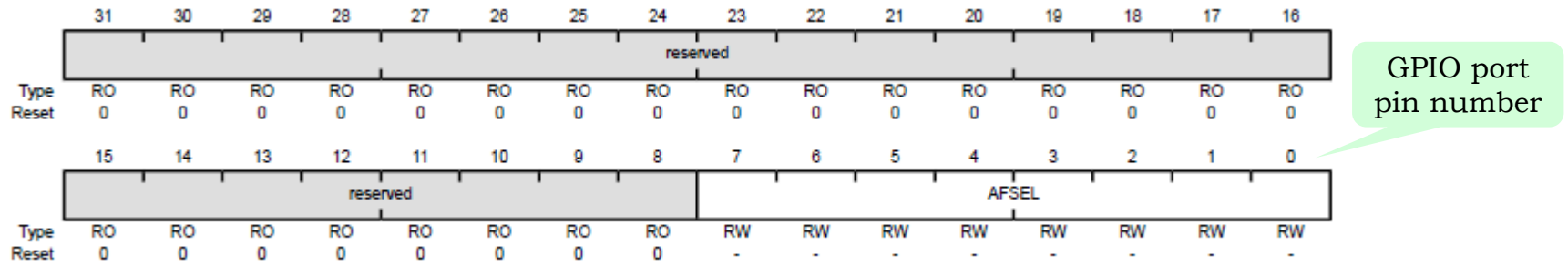


GPIODEN register. R/W; Reset = 0x0000.0000 (*except for pins with special considerations*)

- Setting the **DEN** bit to '0' disables the digital function.
- **To use a GPIO pin as digital input/output (either GPIO or alternate function), the corresponding GPIODEN pin must be set to '1'.**
- Reset value for GPIO pin = 0 (*for ports not listed with pins with Special Functions*) => **digital function not enabled upon reset**.
  – *see also pg 650 of datasheet & lecture slide 23.*

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p682)

GPIO Alternate Function Select Register (GPIOAFSEL)

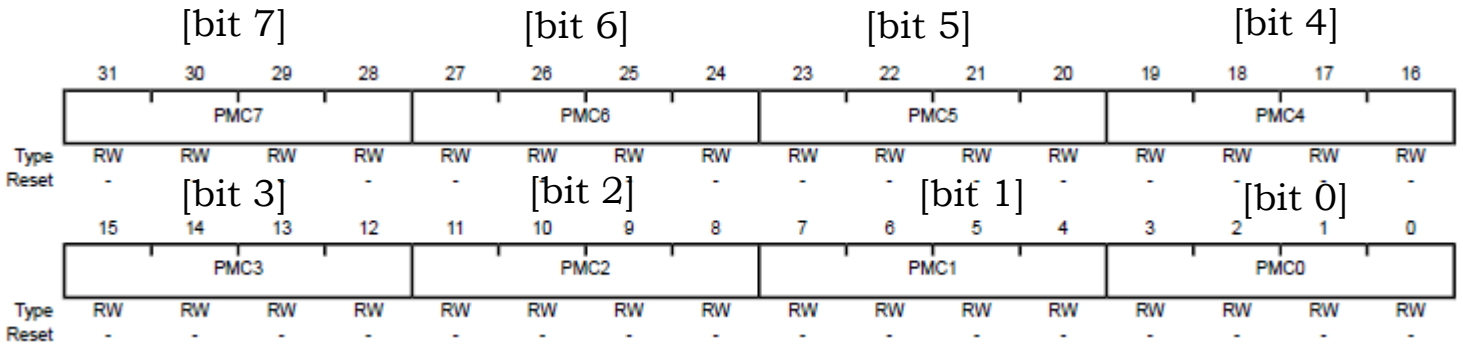


GPIOAFSEL register. R/W; Reset = 0x0000.0000 (*except for pins with special considerations*)

- The Alternate Function Select (**GPIOAFSEL**) register is used to select the mode to be used for the GPIO pin.
- **AFSEL[7:0]**:
  - '0' = GPIO pin function;
  - '1' = Peripheral function.
- Reset value= 0x0000.0000 (default to GPIO function for all 8 pins).
- Type of Alternate function to map to is selected through the **GPIOPCTL** register.

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p671)

GPIO Port Control Register (GPIOPCTL)



GPIOPCTL register. R/W; Reset = 0x0000.0000 (*except for pins with special considerations*)

- Port Control (**GPIOPCTL**)Register is used together with the GPIOAFSEL register to select the specific **peripheral** assigned for each GPIO pin.
- **GPIOPCTL** selects one of the set of peripheral functions assigned to the GPIO pin.
  - **PMC*x*** = Port Mux control for GPIO pin *x*; where *x* = 7 to 0.
- Reset value for GPIO pin = 0 (*for ports not listed with pins with Special Functions*) => **digital function not enabled upon reset**.
  – see also pg 650 of datasheet & lecture slide 23.

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p689)

TM4C123G GPIO Ports A, B

GPIO port pins

Values 10-13 not used

GPIOPCTL PMCx bit values

IO	Pin	Analog Function	Digital Function (GPIOPCTL PMCx Bit Field Encoding)[a]										
			1	2	3	4	5	6	7	8	9	14	15
PA0	17	-	U0Rx	-	-	-	-	-	-	CAN1Rx	-	-	
PA1	18	-	U0Tx	-	-	-	-	-	-	CAN1Tx	-	-	
PA2	19	-	-	SSI0Clk	-	-	-	-	-	-	-	-	
PA3	20	-	-	SSI0Fss	-	-	-	-	-	-	-	-	-
PA4	21	-	-	SSI0Rx	-	-	-	-	-	-	-	-	-
PA5	22	-	-	SSI0Tx	-	-	-	-	-	-	-	-	-
PA6	23	-	-	-	I2C1SCL	-	M1PWM2	-	-	-	-	-	-
PA7	24	-	-	-	I2C1SDA	-	M1PWM3	-	-	-	-	-	-
PB0	45	USB0ID	U1Rx	-	-	-	-	-	T2CCP0	-	-	-	-
PB1	46	USB0VBUS	U1Tx	-	-	-	-	-	T2CCP1	-	-	-	-
PB2	47	-	-	-	I2C0SCL	-	-	-	T3CCP0	-	-	-	-
PB3	48	-	-	-	I2C0SDA	-	-	-	T3CCP1	-	-	-	-
PB4	58	AIN10	-	SSI2Clk	-	M0PWM2	-	-	T1CCP0	CAN0Rx	-	-	-
PB5	57	AIN11	-	SSI2Fss	-	M0PWM3	-	-	T1CCP1	CAN0Tx	-	-	-
PB6	1	-	-	SSI2Rx	-	M0PWM0	-	-	T0CCP0	-	-	-	-
PB7	4	-	-	SSI2Tx	-	M0PWM1	-	-	T0CCP1	-	-	-	-

For most GPIO pins, PMCx = 0 upon power on => GPIO pin is a digital pin.
Exceptions are those pins with entries **shaded Grey** where it defaults to the shaded function.

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p650)

TM4C123G GPIO Ports C, D

GPIO port pins

10-13 not used

IO	Pin	Analog Function	Digital Function (GPIOPCTL PMCx Bit Field Encoding)[a]										
			1	2	3	4	5	6	7	8	9	14	15
PC0	52	-	TCK SWCLK	-	-	-	-	-	T4CCP0	-	-	-	-
PC1	51	-	TMS SWDIO	-	-	-	-	-	T4CCP1	-	-	-	-
PC2	50	-	TDI	-	-	-	-	-	T5CCP0	-	-	-	-
PC3	49	-	TDO SWO	-	-	-	-	-	T5CCP1	-	-	-	-
PC4	16	C1-	U4Rx	U1Rx	-	M0PWM6	-	IDX1	WT0CCP0	U1RTS	-	-	-
PC5	15	C1+	U4Tx	U1Tx	-	M0PWM7	-	PhA1	WT0CCP1	U1CTS	-	-	-
PC6	14	C0+	U3Rx	-	-	-	-	PhB1	WT1CCP0	USB0EPEN	-	-	-
PC7	13	C0-	U3Tx	-	-	-	-	-	WT1CCP1	USB0PFLT	-	-	-
PD0	61	AIN7	SSI3Clk	SSI1Clk	I2C3SCL	M0PWM6	M1PWM0	-	WT2CCP0	-	-	-	-
PD1	62	AIN6	SSI3Fss	SSI1Fss	I2C3SDA	M0PWM7	M1PWM1	-	WT2CCP1	-	-	-	-
PD2	63	AIN5	SSI3Rx	SSI1Rx	-	M0FAULT0	-	-	WT3CCP0	USB0EPEN	-	-	-
PD3	64	AIN4	SSI3Tx	SSI1Tx	-	-	-	IDX0	WT3CCP1	USB0PFLT	-	-	-
PD4	43	USB0DM	U6Rx	-	-	-	-	-	WT4CCP0	-	-	-	-
PD5	44	USB0DP	U6Tx	-	-	-	-	-	WT4CCP1	-	-	-	-
PD6	53	-	U2Rx	-	-	M0FAULT0	-	PhA0	WT5CCP0	-	-	-	-
PD7	10	-	U2Tx	-	-	-	-	PhB0	WT5CCP1	NMI	-	-	-

GPIOPCTL PMCx bit values

Locked pins

For most GPIO pins, PMCx = 0 upon power on => GPIO pin is a digital pin.
Exceptions are those pins with entries **shaded Grey** where it defaults to the shaded function.

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p651)

TM4C123G GPIO Ports E, F

Bits 10-13 not used

IO	Pin	Analog Function	Digital Function (GPIOPCTL PMCx Bit Field Encoding)[a]										
			1	2	3	4	5	6	7	8	9	14	15
PE0	9	AIN3	U7Rx	-	-	-	-	-	-	-	-	-	-
PE1	8	AIN2	U7Tx	-	-	-	-	-	-	-	-	-	-
PE2	7	AIN1	-	-	-	-	-	-	-	-	-	-	-
PE3	6	AIN0	-	-	-	-	-	-	-	-	-	-	-
PE4	59	AIN9	U5Rx	-	I2C2SCL	M0PWM4	M1PWM2	-	-	CAN0Rx	-	-	-
PE5	60	AIN8	U5Tx	-	I2C2SDA	M0PWM5	M1PWM3	-	-	CAN0Tx	-	-	-
PF0	28	-	U1RTS	SSI1Rx	CAN0Rx	-	M1PWM4	PhA0	T0CCP0	NMI	C0o	-	-
PF1	29	-	U1CTS	SSI1Tx	-	-	M1PWM5	PhB0	T0CCP1	-	C1o	TRD1	-
PF2	30	-	-	SSI1Clk	-	M0FAULT0	M1PWM6	-	T1CCP0	-	-	TRD0	-
PF3	31	-	-	SSI1Fss	CAN0Tx	-	M1PWM7	-	T1CCP1	-	-	TRCLK	-
PF4	5	-	-	-	-	-	M1FAULT0	IDX0	T2CCP0	USB0EPEN	-	-	-

GPIOPCTL PMCx bit values

Locked pins

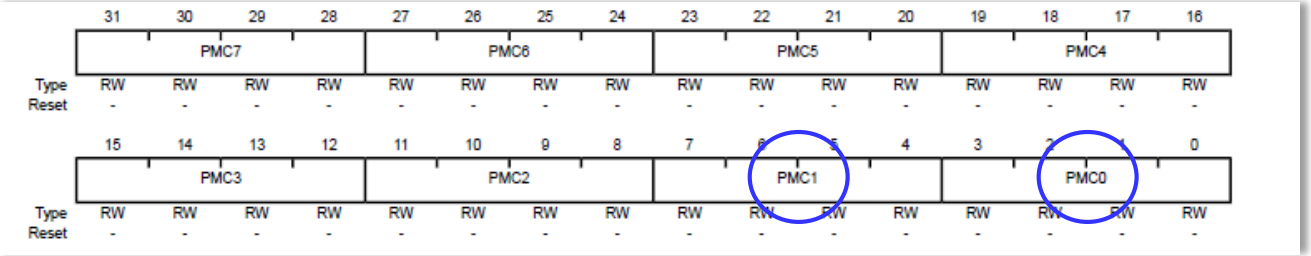
GPIO port pins

Example: Alternate Functions

- Example:
  - To select UART0 in GPIO Port A (U0Rx & U0Tx), we write:
    - 0x03 to Port A GPIODEN register.
    - 0x03 to Port A GPIOAFSEL register.
    - 0x0000.0011 to GPIOPCTL register.
      - 0x01 to PMC0 (U0Rx);
      - 0x01 to PMC1 (U0Tx).

PMC1 PMC0

IO	Pin	Analog Function	Digital Function (GPIOPCTL PMCx Bit Field Encoding)[a]										
			1	2	3	4	5	6	7	8	9	14	15
PA0	17	-	U0Rx	-	-	-	-	-	-	CAN1Rx	-	-	-
PA1	18	-	U0Tx	-	-	-	-	-	-	CAN1Tx	-	-	-



[GPIOPCTL register]

GPIOs with Special Considerations

Special Consideration:

These are the GPIO pins that are greyed out (in slides 17-19).

Table 10-1. 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^2C0	0	0	0	0	0x3	1
PC[3:0]	JTAG/SWD	1	1	0	1	0x1	0

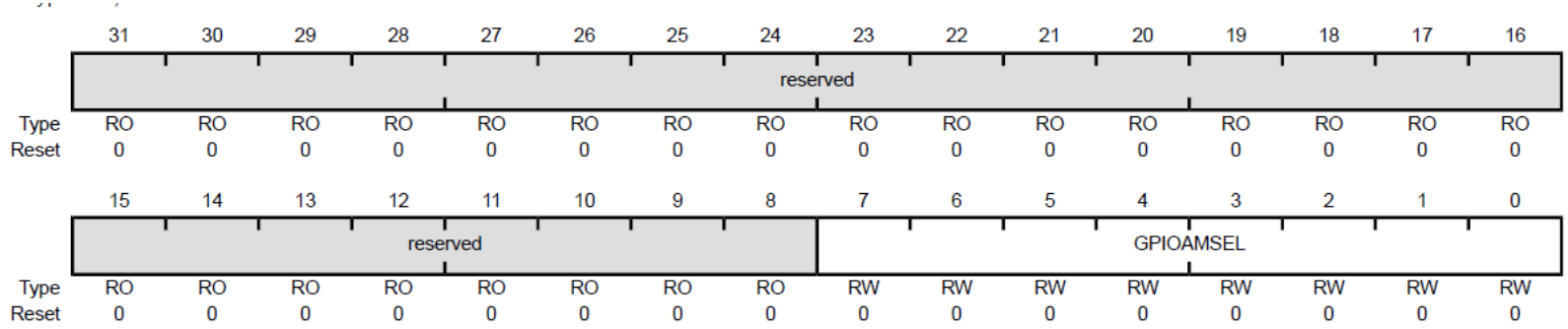
- Most GPIO pins are configured as digital pins & tri-stated by default (**GPIOAFSEL**=0, **GPIODEN**=0, **GPIOPDR**=0, **GPIOPUR**=0 & **GPIOPCTL**=0: registers settings for tri-state).
- The above pins DO NOT default to digital functions upon Reset. Instead, they default to one of the alternate functions instead (**GPIOPCTL** ≠ 0 by default).
- *Pay attention to them in your programs if you are using these GPIOs.*

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p688)

GPIO Pins – Analog Functions

- Analog hardware functions are enabled through:
  - Clearing corresponding bit in GPIO Digital Enable (**GPIODEN**) register.
  - Setting the corresponding bits in the GPIO Analog Mode Select (**GPIOAMSEL**) register.

GPIO Analog Mode Select Register (GPIOAMSEL)



GPIOAMSEL register. R/W; Reset = 0x0000.0000

Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0x0000.00	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7:0	GPIOAMSEL	RW	0x00	GPIO Analog Mode Select

Value	Description
0	The analog function of the pin is disabled, the isolation is enabled, and the pin is capable of digital functions as specified by the other GPIO configuration registers.
1	The analog function of the pin is enabled, the isolation is disabled, and the pin is capable of analog functions.

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p687)

More GPIO Registers ….

GPIODR2R, GPIODR4R, GPIODR8R, GPIOPUR, GPIOPDR registers

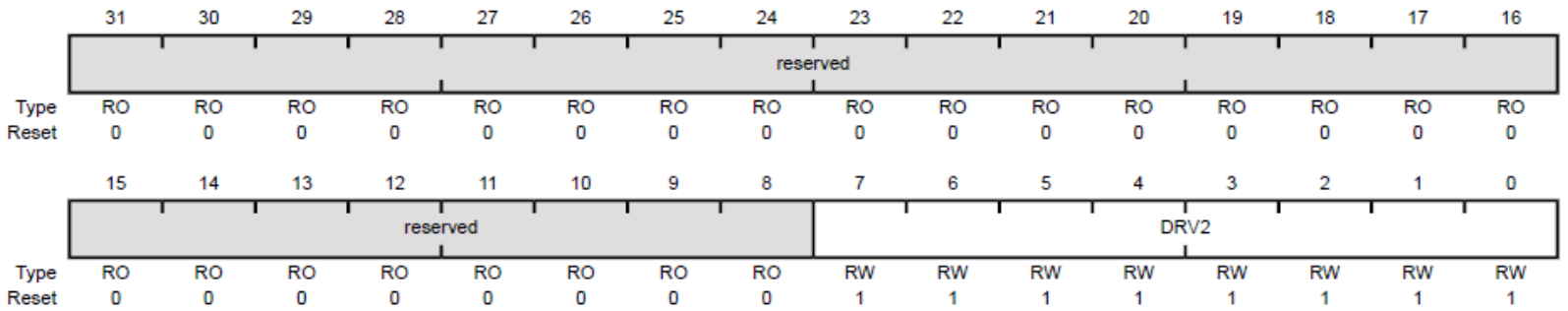
GPIO Registers

- GPIO registers needed during initialization *(not complete list – interrupt register not listed)*:
  - Data Register (GPIODATA)
  - Direction Register (GPIODIR)
  - Alternate Function Select (GPIOAFSEL)
  - GPIO Drive Strength:
    - 2mA Drive (GPIODR2R)
    - 4mA Drive (GPIODR4R)
    - 8mA Drive (GPIODR8R)
  - Pull UP/Down:
    - Pull-up (GPIOPUR)
    - Pull-down (GPIOPDR)
  - Digital Function Enable (GPIODEN)
  - Lock register (GPIOLOCK)
  - Commit Register (GPIOCR)
  - Analog Mode Select (GPIOAMSEL)
  - Port Control (GPIOPCTL)

Blue indicates the register has been covered in an earlier slide.

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p650)

GPIO 2mA Drive Strength Register (GPIODR2R)



GPIODR2R register. R/W; Reset = 0x0000.00FF

- Sets the drive strengths for the GPIO pins.
- **DRV2** bits:
  - '0' = Drive is controlled by GPIODR4R or GPIODR8R registers;
  - '1' = GPIO pin drive is 2mA.

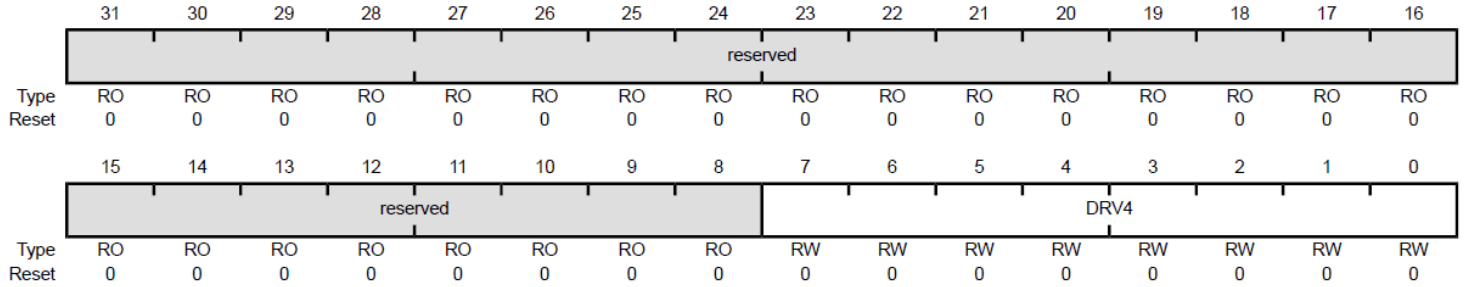
When DRV2 bit is set, the corresponding DRV4 bit in GPIODR4R & DRV8 bit in GPIODR8R are automatically cleared.

GPIO 2-mA Drive Select (GPIODR2R)

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 0x500
Type RW, reset 0x0000.00FF

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p673)

GPIO 4mA Drive Strength Register (GPIODR4R)



	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								DRV4							
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

GPIODR4R register. R/W; Reset = 0x0000.0000

- Sets the drive strengths for the GPIO pins.
- **DRV4** bits:
  - '0' = Drive is controlled by GPIODR2R or GPIODR8R registers;
  - '1' = GPIO pin drive is 4mA.

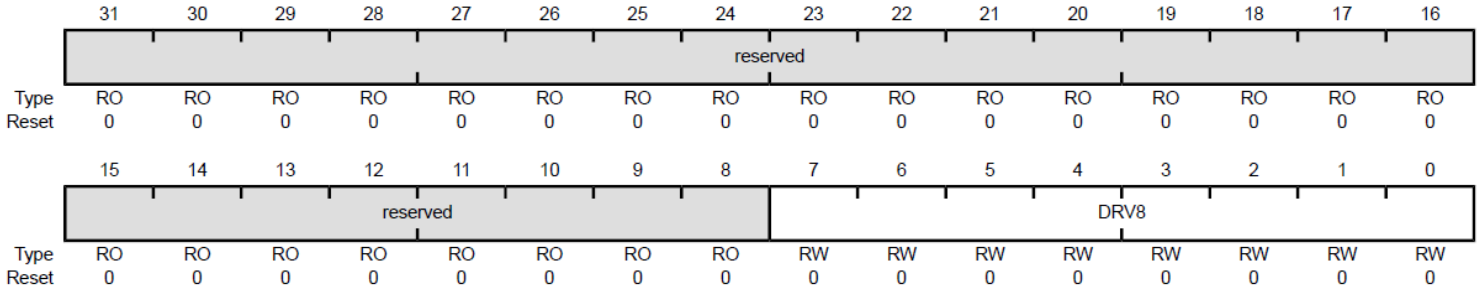
When DRV4 bit is set, the corresponding DRV2 bit in GPIODR2R & DRV8 bit in GPIODR8R are automatically cleared.

GPIO 4-mA Drive Select (GPIODR4R)

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 0x504
Type RW, reset 0x0000.0000

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p674)

GPIO 8mA Drive Strength Register (GPIODR8R)



	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								DRV8							
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

GPIODR8R register. R/W; Reset = 0x0000.0000

- Sets the drive strengths for the GPIO pins.
- **DRV8** bits:
  - '0' = Drive is controlled by GPIODR2R or GPIODR4R registers;
  - '1' = GPIO pin drive is 8mA.

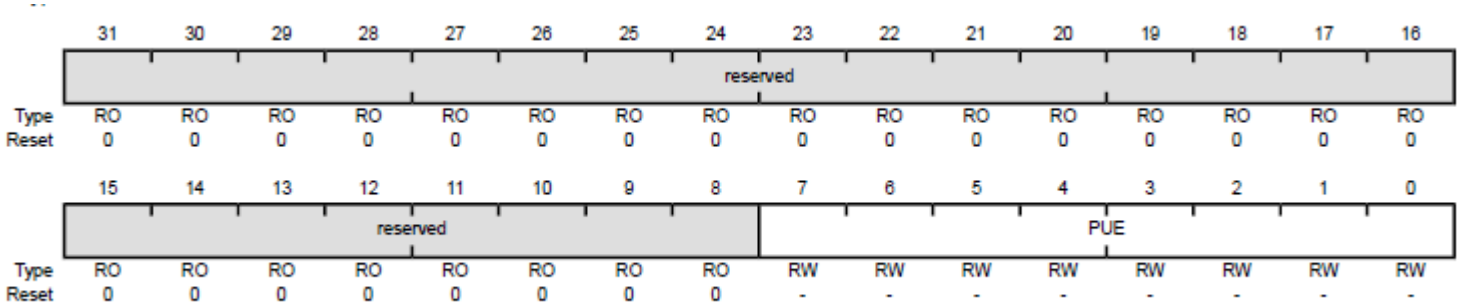
When DRV8 bit is set, the corresponding DRV2 bit in GPIODR2R & DRV4 bit in GPIODR4R are automatically cleared.

GPIO 8-mA Drive Select (GPIODR8R)

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 0x508
Type RW, reset 0x0000.0000

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p675)

GPIO Pull-Up Select Register (GPIOPUR)

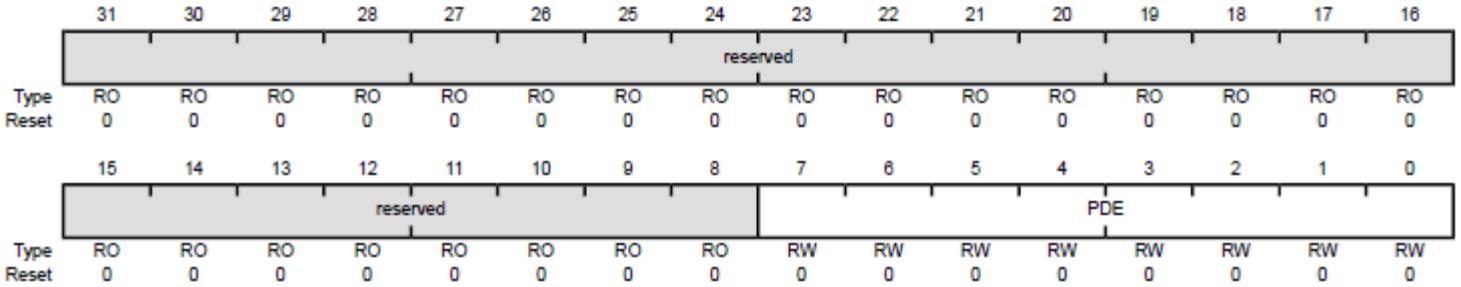


GPIOPUR register. R/W; Reset = 0x0000.0000 (except for pins with special considerations)

- When the **PUE** bit is set, a weak pull-up resistor on the GPIO pin is enabled.
- Used with the Drive Strength registers (DR2R, DR4R, DR8R).
- Setting a bit in the **GPIOPUR** (pull-up) automatically clears the corresponding bit in the **GPIOPDR** (pull-down).
- Note:
  - Write-access is controlled by the **GPIOCR** register. If a bit in the GPIOCR register is cleared, write to the corresponding GPIOPUR bit is disabled.

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p678)

GPIO Pull-Down Select Register (GPIOPDR)



GPIOPDR register. R/W; Reset = 0x0000.0000

- When the **PDE** bit is set, a weak pull-down resistor on the GPIO pin is enabled.
- Used with the Drive Strength registers.
- Setting a bit in the GPIOPUR automatically clears the corresponding bit in the GPIOPDR.
- Note:
  - Write-access is controlled by the GPIOCR register. If a bit in the GPIOCR register is cleared, write to the corresponding GPIOPUR bit is disabled.

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p679)

GPIOs with Lock & Commit

GPIOLOCK, GPIOCR registers

GPIOs with Write-Protection

- Certain GPIO pins are protected against accidental programming changes.
- Applies to certain critical hardware peripherals:
  - JTAG/SWD (4 pins) : PC[3:0] *(these pins are used for debug)*
  - NMI (2 pins) : PF0, PD7
- In order to write to certain registers related to the above pins, we need to unlock the pins through writing a special code to the GPIO Lock (**GPIOLOCK)** register and then set the appropriate bits in the GPIO Commit (**GPIOCR**) register.

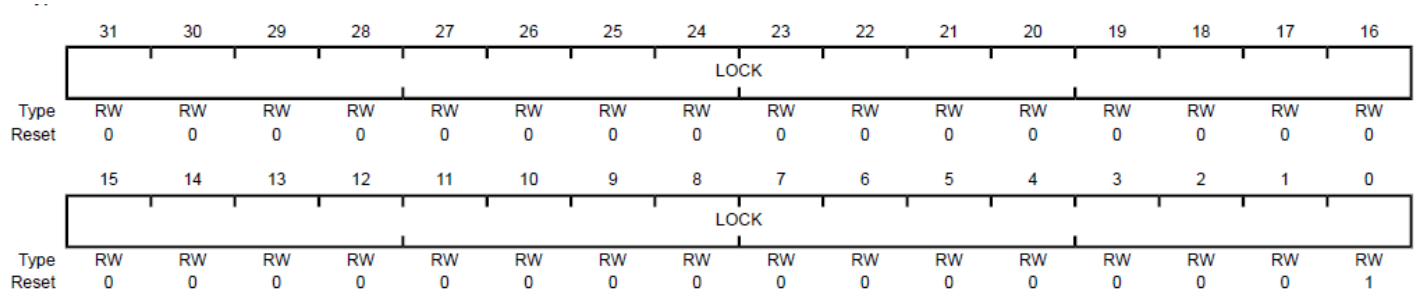
Table 10-1. GPIO Pins With Special Considerations

	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
JTAG:	PC[3:0]	JTAG/SWD	1	1	0	1	0x1	0
NMI:	PD[7]	GPIO[a]	0	0	0	0	0x0	0
NMI:	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.

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p650)

GPIO Lock Register (GPIOLOCK)



GPIOLOCK register. R/W; Reset = 0x0000.0001

WRITE:

- Writing special bit pattern of **0x4C4F.434B** unlocks the GPIOLOCK register.
- Writing any other value re-enables the locked state.

READ:

- Reading the GPIOLOCK register returns the lock status:
  - 0x01 (locked); 0x00 (not locked).

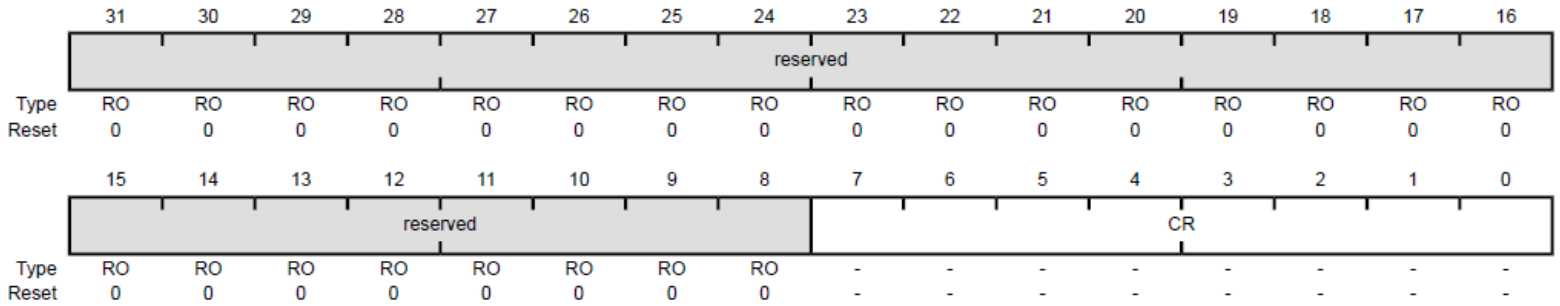
Only PC[3:0], PD7 & PF0 need to unlock & commit.
PF0 is used for SW2 on LaunchPad.

GPIO Lock (GPIOLOCK)

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 0x520
Type RW, reset 0x0000.0001

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p684)

GPIO Commit Register
(GPIOCR)



- GPIOCR provides a layer of protection against accidental programming of critical signals.
- Applies to the following GPIO registers for the pins involved: **GPIOAFSEL**, **GPIOPUR**, **GPIOPDR** & **GPIODEN**.
- Register determines which bits [7:0] of the GPIO port are committed when a write to the register is performed.
  - bit = 0 : GPIO port bit cannot be written.
  - bit = 1 : GPIO port pin can be committed.

GPIO Commit (GPIOCR)

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 0x524
Type -, reset -

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p685)

GPIO Commit Register
(GPIOCR)

Commit Register: (Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p685))

Important: This register is designed to prevent accidental programming of the registers that control connectivity to the NMI and JTAG/SWD debug hardware. By initializing the bits of the **GPIOCR** register to 0 for the NMI and JTAG/SWD pins (see "Signal Tables" on page 1329 for pin numbers), the NMI and JTAG/SWD debug port can only be converted to GPIOs through a deliberate set of writes to the **GPIOLOCK**, **GPIOCR**, and the corresponding registers.

Because this protection is currently only implemented on the NMI and JTAG/SWD pins (see "Signal Tables" on page 1329 for pin numbers), all of the other bits in the **GPIOCR** registers cannot be written with 0x0. These bits are hardwired to 0x1, ensuring that it is always possible to commit new values to the **GPIOAFSEL**, **GPIOPUR**, **GPIOPDR**, or **GPIODEN** register bits of these other pins.

It means, only pins **PC[3:0], PD[7], PF[0]** requires to be unlocked to write to the GPIOAFSEL, GPIOPUR, GPIOPDR & GPIODEN registers. Not needed for the rest.

GPIOs with Write-Protection

- Example: Programming sequence to unlock PF0

```
#define GPIO_LOCK_KEY  0x4C4F434B

#define BIT( x ) (1U<<(x))

#define PF_0 0U

GPIOF->LOCK = GPIO_LOCK_KEY; /* unlock Port F */
GPIOF->CR |= BIT(PF_0);
```

- Note: Reading the GPIOLOCK register returns it to its LOCK state.

System Control Registers for GPIO Ports

RCGCGPIO, PRGPIO registers

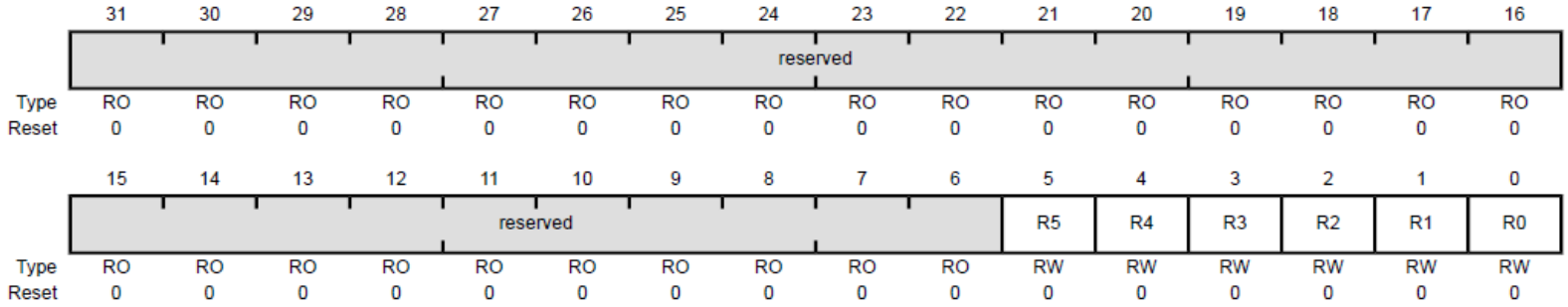
System Control Registers (GPIO Ports)

- Two System Control registers are used to set up a GPIO port.
  - **RCGCGPIO**: **R**un-mode **C**lock **G**ating **C**ontrol register that enables clock source to a GPIO port.
  - **PRGPIO**: **P**eripheral **R**eady register indicates if GPIO port is ready to be accessed.

Offset	Name	Type	Reset	Description	See page
0x608	RCGCGPIO	RW	0x0000.0000	General-Purpose Input/Output Run Mode Clock Gating Control	340
0xA08	PRGPIO	RO	0x0000.0000	General-Purpose Input/Output Peripheral Ready	406

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p340, 406)

GPIO Clock Gating Control Register (RCGCGPIO)



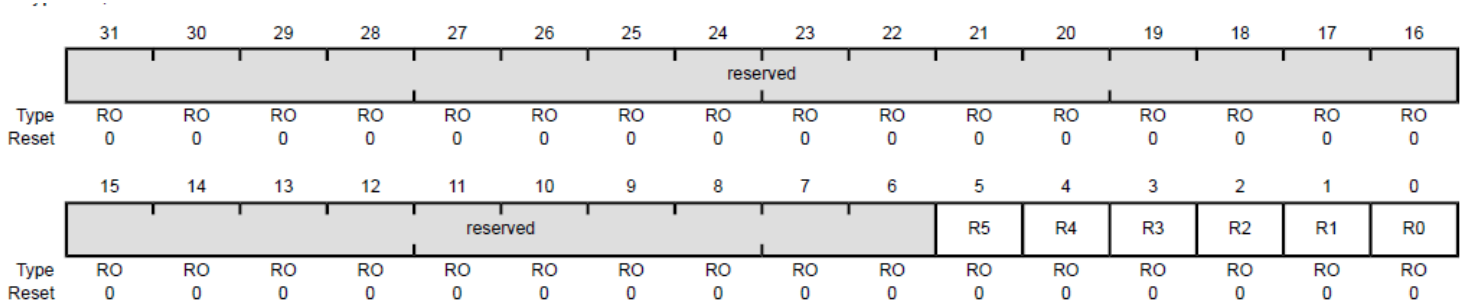
RCGCGPIO register. R/W; Reset =0x0000.0000

- Register to **enable clock** sources & access to GPIO ports & registers.
- If bit is 1, clock to GPIO port is enabled & access to the GPIO registers is allowed.
- If bit is 0, clock to GPIO port is disabled & access to GPIO registers results in a bus fault.

R0 bit denotes Port A.
R1 bit denotes Port B.
R2 bit denotes Port C.
R3 bit denotes Port D.
R4 bit denotes Port E.
R5 bit denotes Port F.

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p406)

GPIO Peripheral Ready Register (PRGPIO)



PRGPIO register. RO; Reset =0x0000.0000

- Register to indicate **if a GPIO module is ready** to be accessed.
- If bit read is '1', GPIO port is ready.
- If bit read is '0', GPIO port is not ready to be accessed.

R0 bit denotes Port A.
R1 bit denotes Port B.
R2 bit denotes Port C.
R3 bit denotes Port D.
R4 bit denotes Port E.
R5 bit denotes Port F.

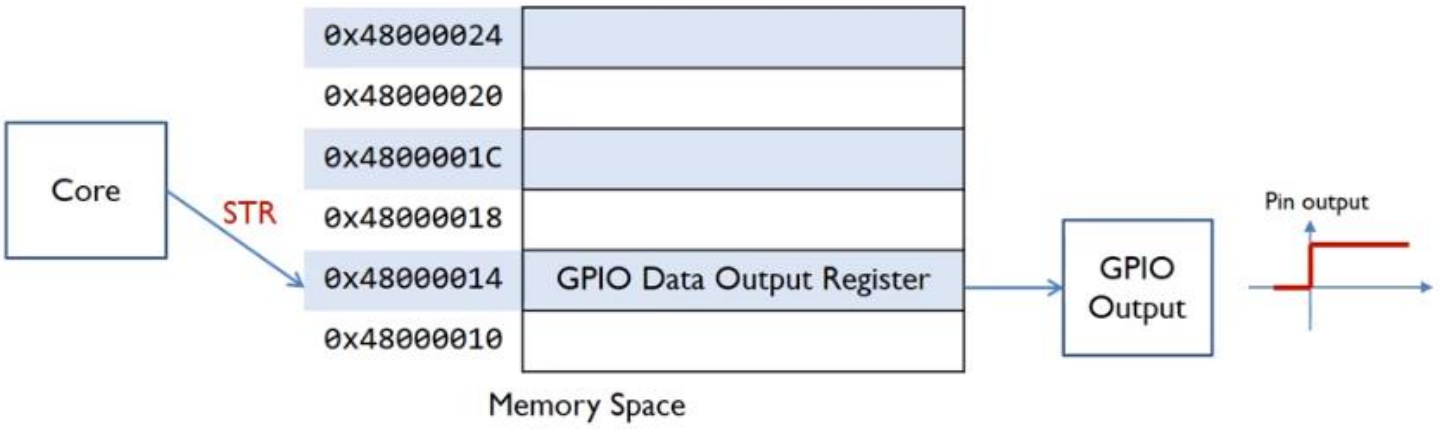
Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p406)

Memory Mapped I/Os & Data Structure

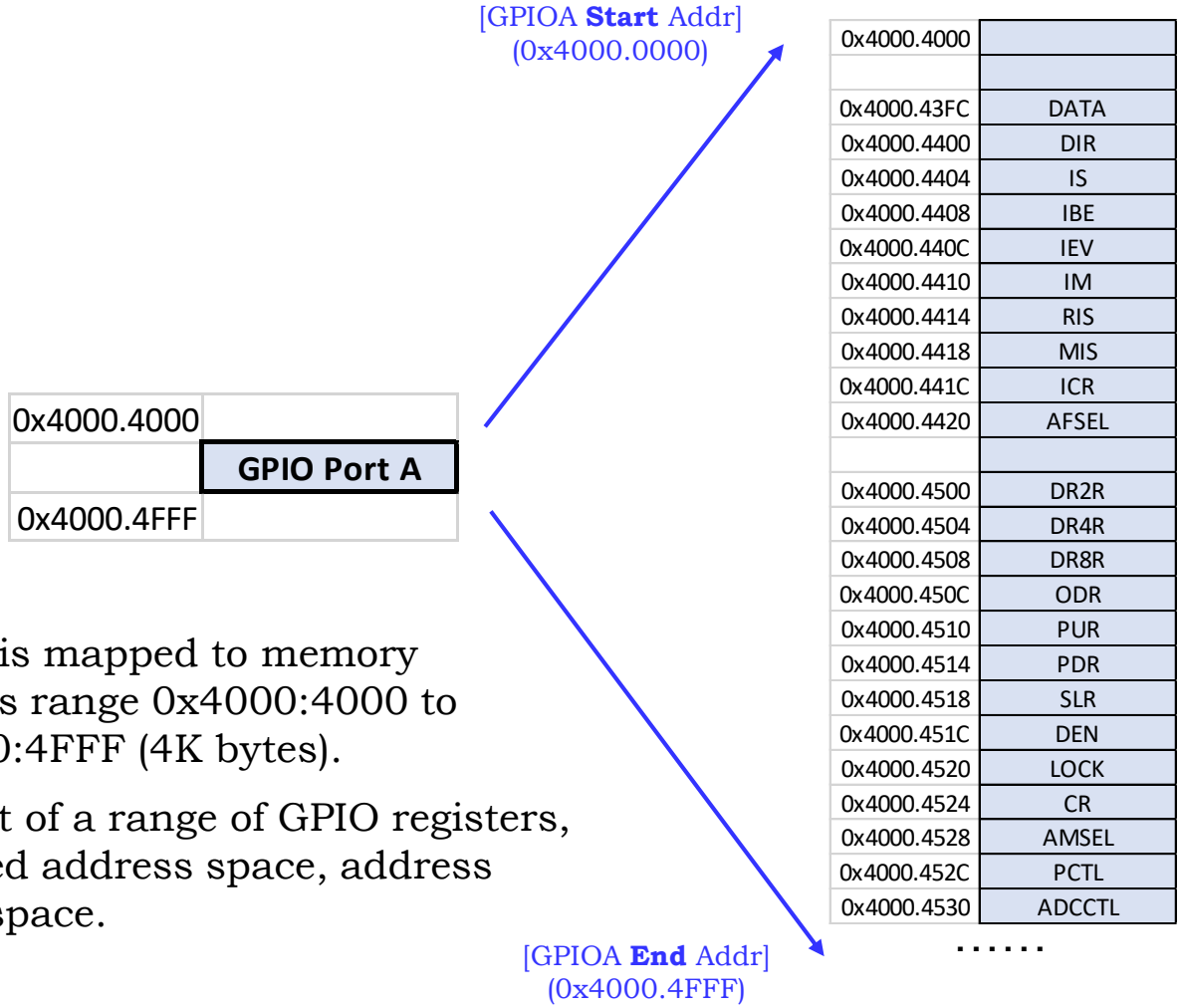
TM4C123GH6PM7

Memory Mapped I/O

- ARM peripherals are mapped to the memory addresses.
- SW access an I/O port through reading from or writing to a memory address.
- Each peripheral I/O device register is assigned to a memory address in the address space of the ARM processor.
- Makes use of native **`LDR / STR Reg, [Reg, #imm]`** instructions to access I/O.



Memory Mapped I/O: GPIO Port A



- Port A is mapped to memory address range 0x4000:4000 to 0x4000:4FFF (4K bytes).
- Consist of a range of GPIO registers, reserved address space, address mask space.

GPIO Data Structure

- For our programs, we define a data structure for GPIO registers.
- Makes access to the registers more systematic & code more readable.
- Registers are accessed through pointers to a C **`struct`** construct.

GPIO Data Structure

GPIO Port A

0x4000.4000	
0x4000.43FC	DATA
0x4000.4400	DIR
0x4000.4404	IS
0x4000.4408	IBE
0x4000.440C	IEV
0x4000.4410	IM
0x4000.4414	RIS
0x4000.4418	MIS
0x4000.441C	ICR
0x4000.4420	AFSEL
0x4000.4500	DR2R
0x4000.4504	DR4R
0x4000.4508	DR8R
0x4000.450C	ODR
0x4000.4510	PUR
0x4000.4514	PDR
0x4000.4518	SLR
0x4000.451C	DEN
0x4000.4520	LOCK
0x4000.4524	CR
0x4000.4528	AMSEL
0x4000.452C	PCTL
0x4000.4530	ADCCTL

file: TM4C123GH6PM7.h

```
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;
```

Accessing the GPIO Data Structure

file: TM4C123GH6PM7.h

```
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         */
  /** some lines omitted - see previous slide  **/
  __IO uint32_t  ADCCTL;            /* GPIO ADC Control    */
  __IO uint32_t  DMACTL;            /* GPIO DMA Control    */
} GPIOA_Type;
```

Macros (used in CMSIS header files):

__IO: Read/Write (volatile)

__I : Read-only (volatile)

__O: Write-only

CMSIS: Cortex Microcontroller Software Interface Standard

```
/* GPIO port base addresses                                    */
#define GPIOA_BASE    0x40004000UL
#define GPIOB_BASE    0x40005000UL
#define GPIOC_BASE    0x40006000UL
#define GPIOD_BASE    0x40007000UL
#define GPIOE_BASE    0x40024000UL
#define GPIOF_BASE    0x40025000UL

/* GPIOA - GPIOF defined as pointers to the data structure   */
#define GPIOA     ((GPIOA_Type *) GPIOA_BASE)
#define GPIOB     ((GPIOA_Type *) GPIOB_BASE)
#define GPIOC     ((GPIOA_Type *) GPIOC_BASE)
#define GPIOD     ((GPIOA_Type *) GPIOD_BASE)
#define GPIOE     ((GPIOA_Type *) GPIOE_BASE)
#define GPIOF     ((GPIOA_Type *) GPIOF_BASE)
```

Accessing the GPIO Data Structure

- With the structure declared & definitions in previous slides, we can access the GPIO registers in the following manner:

```
/* accessing GPIO registers  */
GPIOA->DATA |= 1UL <<7;     /* set bit 7                      */
GPIOA->DATA &= ~(1UL <<2);  /* reset bit 2                    */
GPIOA->DIR = 0x03;          /* assign 0x03 to GPIOA DIR reg */
```

```
/* accessing GPIO registers   */
(*GPIOA).DATA |= 1UL <<7;    /* set bit 7                      */
(*GPIOA).DATA &= ~(1UL <<2); /* reset bit 2                    */
(*GPIOA).DIR = 0x03;         /* assign 0x03 to GPIOA DIR reg */
```

Bit Operations

TM4C123GH6PM7

Common Bit Operations

- **Bit-shifts**
  - **Right shift**: `num >> shift`, e.g. `0b1001 >> 2` → `0b0010`
  - **Left shift**: `num << shift`, e.g. `0b0011 << 2` → `0b1100`
- **Masking**
  - **Bit-or**: `num | mask`, e.g. `0b0001 | 0b1000` → `0b1001` *(set bit 3, keep all others unchanged)*
  - **Bit-and**: `num & mask`, e.g. `0b1001 & 0b1000` → `0b1000` *(keep bit 3, reset all others)*
- **Complement**
  - **Not**: `~ num`, e.g. `~ 0b0101` → `0b1010` *(invert all bits)*
- **Set bits**
  - set bit 4: `x |= (1 << 4);`
    (same as `x = x | (1 << 4);`)
- **Clear bits**
  - clear bit 4 and 5: `x &= ~(0b11 << 4);`
    (same as, `x = x & ~(0b11 << 4);`)

Note that KEIL µVison do not support binary types. This is for illustration.

Examples of Bit Operations …

```
GPIOA->DATA = 0x30CC; /* write 0b0011.0000.1100.1100 to reg */

/* Set only bit 1; do not alter other bits                  */
GPIOA->DATA |= 0x02; /* 0b0000.0000.0000.0010               */

/* Set bits 4,10,14; do not alter other bits                */
GPIOA->DATA |= 0x4410; /* 0b0100.0100.0001.0000             */

/* Reset bits 2, 7, 13; do not alter other bits             */
GPIOA->DATA &= ~0x2084; /* 0b0010.0000.1000.0100            */
```

```
in = GPIOB->DATA; /* read register value                    */
in |= 0x20;       /* set bit 5  (0x20 = 0b0010.0000        */
in &= ~0x08;      /* clear bit 3 (0x08 = 0b1000)           */
GPIOB->DATA = in; /* write updates back to register        */
                  /* what is the value of GPIOB->DATA?     */
```

Bit Operations (SET Bits)

```
GPIOA->DATA = 0xD455; /* write 0b1101.0100.0101.0101 to reg    */

/* Set bits 2, 7, 13; do not alter other bits                  */
GPIOA->DATA |= 0x2084; /* 0b0010.0000.1000.0100                */
/* above statement same as:                                    */
/*   GPIOA->DATA = GPIOA->DATA | 0x2084;                       */
```

Bit Position:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remarks
0x			D				4				5				5		
GPIO->DATA=0xD455	1	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	**(A)** Write 0xD455 to DATA reg.
0x			2				0				8				4		
0x2084	0	0	**1**	0	0	0	0	0	**1**	0	0	0	0	**1**	0	0	**(B)** We want to SET bits at 13,7,2
			2				0				8				4		
GPIO->DATA & | 0x2084	1	1	**1**	1	0	1	0	0	**1**	1	0	1	0	**1**	0	1	**(C)** BIT-OR of **(A)** and **(B)**. Result will be written back to GPIOA->DATA

Only bits 13,7,2 in original data in DATA register are SET. All other bits are not affected.

Bit Operations (RESET Bits)

```
GPIOA->DATA = 0x30CC; /* write 0b0011.0000.1100.1100 to reg    */

/* Reset bits 2, 7, 13; do not alter other bits                */
GPIOA->DATA &= ~0x2084; /* 0b0010.0000.1000.0100               */
/* above statement same as:                                    */
/*   GPIOA->DATA = GPIOA->DATA & ~0x2084;                      */
```

Bit Position:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remarks
0x	3				0				C				C				
GPIO->DATA = 0x30CC	0	0	1	1	0	0	0	0	1	1	0	0	1	1	0	0	**(A)** Write 0x30CC to DATA reg.
0x	2				0				8				4				
0x2084	0	0	**1**	0	0	0	0	0	**1**	0	0	0	0	**1**	0	0	**(B)** We want to RESET bits at 13,7,2
~0x2084	1	1	**0**	1	1	1	1	1	**0**	1	1	1	1	**0**	1	1	**(C)** Bit Invert bit pattern in **(B)**
0x	2				0				8				4				
GPIO->DATA & ~0x2084	0	0	**0**	1	0	0	0	0	**0**	1	0	0	1	**0**	0	0	**(D)** Result of BIT-AND of **(A)** and **(C)** will be written back to GPIOA->DATA

Only bits 13,7,2 in original data in DATA register are RESET. All other bits were not affected.

Type Qualifier: `volatile`

- Variables that tend to be re-used repeatedly in different parts of the code are often identified by compilers for optimization of program performance.
  - They are often stored in an internal register and read- from/written-to whenever the variable is accessed in the code.
- For embedded systems, some memory values may change through hardware acquisition:
  - Example: Hardware switch is mapped to memory-mapped location & location is read and saved into a general-purpose register
  - A program may erroneously keep reading the old value, even if switch status had changed
- A **volatile** qualifier ensure that the data is loaded from or stored to memory each time it is referenced.
  - This way, the data value is **ALWAYS updated** when accessed.
- Example: *(as defined in the labs)*

```
#define TRUE  1
#define FALSE 0
typedef int BOOL;
volatile BOOL switch_A = FALSE;
```

Type Qualifier: `static`

- Variables local to function no longer exist when the function returns a value and exits (they have *local scope*)
- A variable is defined as **static** so it retains its value between function calls.
  - Example: a function that that records the change in a rotating motor shaft encoder. It needs to know the previous reading in order to compute the difference.
- A **static** variable in a function is stored in the "heap" (*dedicated memory space*) instead of function "stack" (*temporary storage*). **It retains its value between function calls**.
- Example: `static int reading = 23;`
- When used with global variables and functions, static limits the scope of the variable to its file:
  - Static global variables & functions are not visible outside of the C file they are defined in.

Address Masking

TM4C123GH6PM7

GPIO Address Masking

- Each GPIO port data register has a base-address. For example,
  - PORTA (APB) DATA has base address at 0x4000.4000.
  - PORTB (APB) DATA has base address at 0x4000.5000.
  - PORTC (APB) DATA has base address at 0x4000.6000, and so on.
- We can write an 8-bit value directly to the GPIODATA register. This would modify ALL 8 bits.
- However, in many embedded applications we want to be able to write to or read from only selected bits. We therefore wants an efficient manner to be able to **set or clear individual selected bits**.
- In order to do so, we make use of a **bit mask** to indicate which bits are to be modified.
- Bit-mask implementation on Tiva LaunchPad:
  - **each GPIO port is mapped to 256 addresses**.
  - **Bits 9 to 2** of the address bus are used as the **bit mask**.

Address Masking for GPIO PORTA

- PORTA Data register base address = 0x4000.4000.
- PORTA covers a range of 256 (2^8) memory locations, from offset address 0x0000 to 0x03FC. This range of addresses are used for Address Masking.
- Address Mask bits define which bits can be modified. 256 locations would cover every possible combination for an 8-bit GPIO Port. *[See table below]*

256 locations of 32-bit length

Offset Addr:	Address Mask Bits 9	8	7	6	5	4	3	2	1	0	PORTA Addr:	
0x00	0	0	0	0	0	0	0	0	0	0	0x4000.4000	**GPIO PORTA base address**. No bits can be modified.
0x04	0	0	0	0	0	0	0	1	0	0	0x4000.4004	Write to offset 0x04 => GPIOA bit 0 can be modified.
0x08	0	0	0	0	0	0	1	0	0	0	0x4000.4008	Write to offset 0x08 => GPIOA bit 1 can be modified.
0x0C	0	0	0	0	0	0	1	1	0	0	0x4000.400C	Write to offset 0x0C => GPIOA bits 0 & 1 can be modified.
0x10	0	0	0	0	0	1	0	0	0	0	0x4000.4010	Write to offset 0x10 => GPIOA bit 2 can be modified.
0x14	0	0	0	0	0	1	0	1	0	0	0x4000.4014	Write to offset 0x14 => GPIOA bits 0 & 2 can be modified.
0x18	0	0	0	0	0	1	1	0	0	0	0x4000.4018	Write to offset 0x18 => GPIOA bits 1 & 2 can be modified.
									0	0		
.	.	.	.	.	.	.	.	.	0	0	.	
0x10	0	0	0	1	0	0	0	0	0	0	0x4000.4040	Write to offset 0x40 => GPIOA bit 4 can be modified.
.	.	.	.	.	.	.	.	.	0	0	.	
									0	0		
0x3F4	1	1	1	1	1	1	0	1	0	0	0x4000.43F4	Write to offset 0x3F4 => GPIOA bits 7,6,5,4,3,2,0 can be modified.
0x3F8	1	1	1	1	1	1	1	0	0	0	0x4000.43F8	Write to offset 0x3F8 => GPIOA bits 7,6,5,4,3,2,1 can be modified.
0x3FC	1	1	1	1	1	1	1	1	0	0	0x4000.43FC	**PORTA->DATA register**. At this addr, all 8 bits can be modified.

Address Masking for GPIO PORTA

		Address Mask Bits										
	Offset Addr:	9	8	7	6	5	4	3	2	1	0	PORTA Addr:
256 locations of 32-bit	0x00	0	0	0	0	0	0	0	0	0	0	0x4000.4000
	0x04	0	0	0	0	0	0	0	1	0	0	0x4000.4004
	.	.	.	.	.	.	.	.	.	0	0	.
										0	0	
	0x3F4	1	1	1	1	1	1	0	1	0	0	0x4000.43F4
	0x3F8	1	1	1	1	1	1	1	0	0	0	0x4000.43F8
	0x3FC	1	1	1	1	1	1	1	1	0	0	0x4000.43FC

GPIO PORTA base address. No bits can be modified.

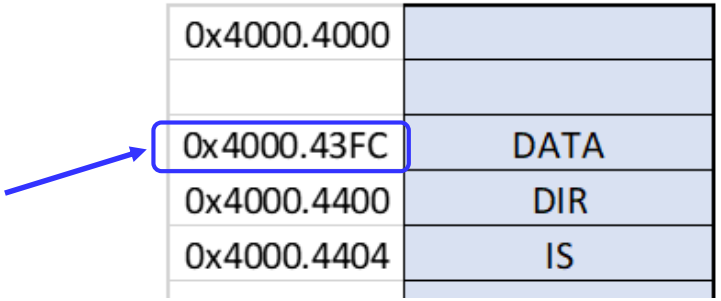
Write to offset 0x04 => GPIOA bit 0 can be modified.

Write to offset 0x3F4 => GPIOA bits 7,6,5,4,3,2,0 can be modified.

Write to offset 0x3F8 => GPIOA bits 7,6,5,4,3,2,1 can be modified.

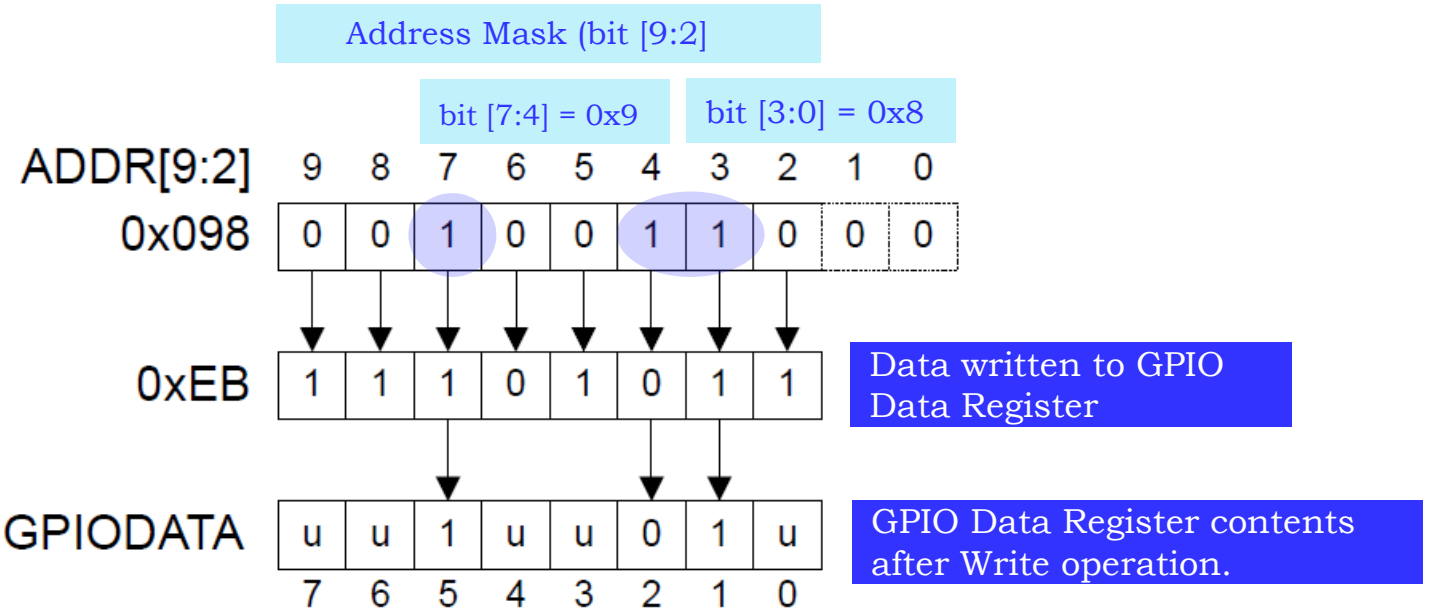
PORTA->DATA register. At this addr, all 8 bits can be modified.

- At the last offset of 0x3FC, all bits in Address Mask are set which means we are writing to ALL 8-bits of GPIO port.
- This is the address of the **GPIOA->DATA** register (*check this against the data sheet*). Therefore, writing to **GPIOA->DATA** writes to ALL 8 bits.



GPIO Port A Registers

GPIO Address Masking (**WRITE**)



- Bits [9:2] are used as the address mask.
  - To enable a bit to be modified, we set the corresponding bit to '1' in the address mask. A '0' would mean a bit would not be affected.
- In above example, to change only bits 5, 2 and 1, we write to address: [GPIO Data Register base address + 0x98].

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p654)

Ex 1: GPIO Address Masking *(Write)*

- I want to write to only bits 6, 4, 1 of GPIO->DATA register and not change rest of the other bits.
- Address Mask = 0b01.0100.1000 (0x148).
- We write to address: [GPIO->DATA base address + 0x148].

Bit:	9	8	7	6	5	4	3	2	1	0	
ADDR MASK (0b)	0	1	0	1	0	0	1	0	0	0	
ADDR MASK (0x)	1			4			8				
Bit:	7	6	5	4	3	2	1	0			
DATA WRITTEN	1	1	0	0	1	0	1	1	Addr: 0x148		
Bit:	7	6	5	4	3	2	1	0			
GPIO->DATA	1	1	1	0	1	0	0	1	**[CONTENTS BEFORE WRITE]**		
Bit:	7	6	5	4	3	2	1	0			
GPIO->DATA	1	**1**	1	**0**	**1**	0	**1**	1	**[CONTENTS AFTER WRITE]**		

Ex 2: GPIO Address Masking

- I want to be able to write to only bits 7, 3, 0 of GPIO->DATA register.
- Address Mask = ?
- What address should I write to? : [GPIO->DATA base addr + offset] = ?

Bit:	9	8	7	6	5	4	3	2	1	0	
ADDR MASK (0b)											
ADDR MASK (0x)											

Bit:	7	6	5	4	3	2	1	0	
DATA WRITTEN	1	1	0	0	1	0	1	1	[DATA WRITTEN TO GPIO->DATA]

Bit:	7	6	5	4	3	2	1	0	
GPIO->DATA	1	1	1	0	1	0	0	1	[CONTENTS BEFORE WRITE]

Bit:	7	6	5	4	3	2	1	0	
GPIO->DATA									[CONTENTS AFTER WRITE]

Ex 2: GPIO Address Masking *(Answer)*

- Write to bits 7, 3, 0 of GPIO->DATA register.
- Address Mask = **0x0224**
- Address to write to: [GPIO->DATA base + offset] = **[GPIO->DATA base+ 0x0224]**.

Bit:	9	8	7	6	5	4	3	2	1	0
ADDR MASK (0b)	1	0	0	0	1	0	0	1	0	0
ADDR MASK (0x)	2		2				4			

Bit:	7	6	5	4	3	2	1	0	
DATA WRITTEN	1	1	0	0	1	0	1	0	

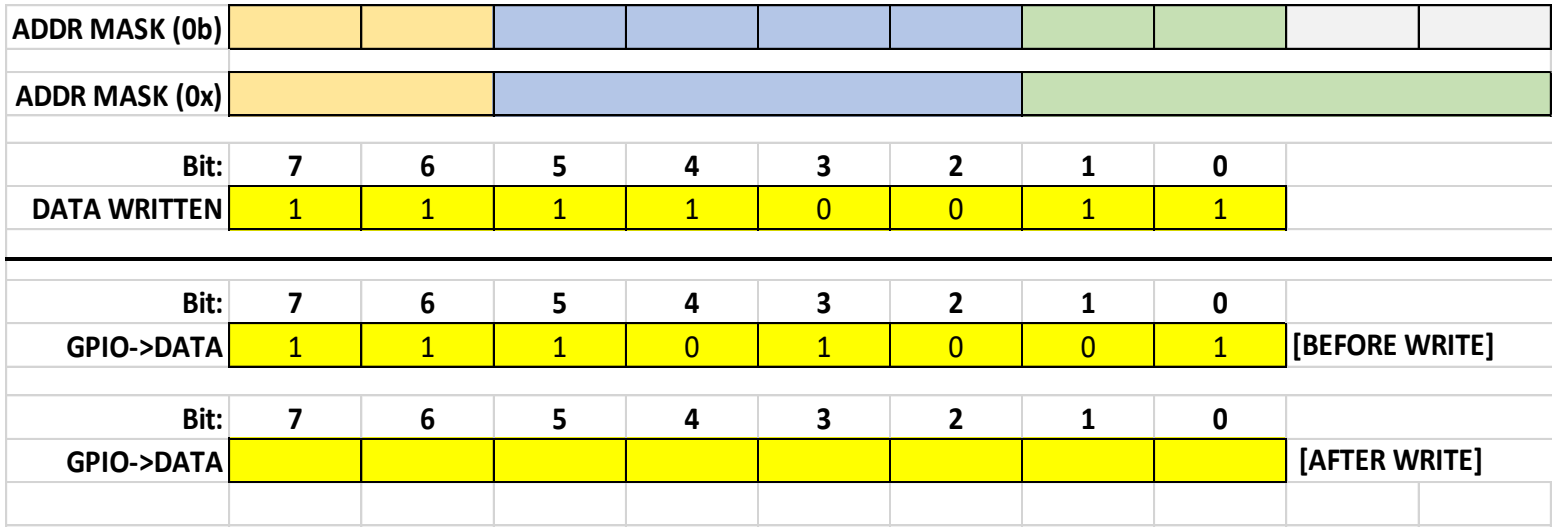
Bit:	7	6	5	4	3	2	1	0	
GPIO->DATA	1	1	1	0	1	0	0	1	[BEFORE WRITE]

Bit:	7	6	5	4	3	2	1	0	
GPIO->DATA	1	1	1	0	1	0	0	0	[AFTER WRITE]

Ex 3: GPIO Address Masking

- I want to Write to bits 7, 6, 5, 3, 1 of GPIO->DATA register.

- Data written = 0b1111.0011
- Address Mask = ?
- What address should I write to? : [GPIO->DATA base addr + offset] = ?



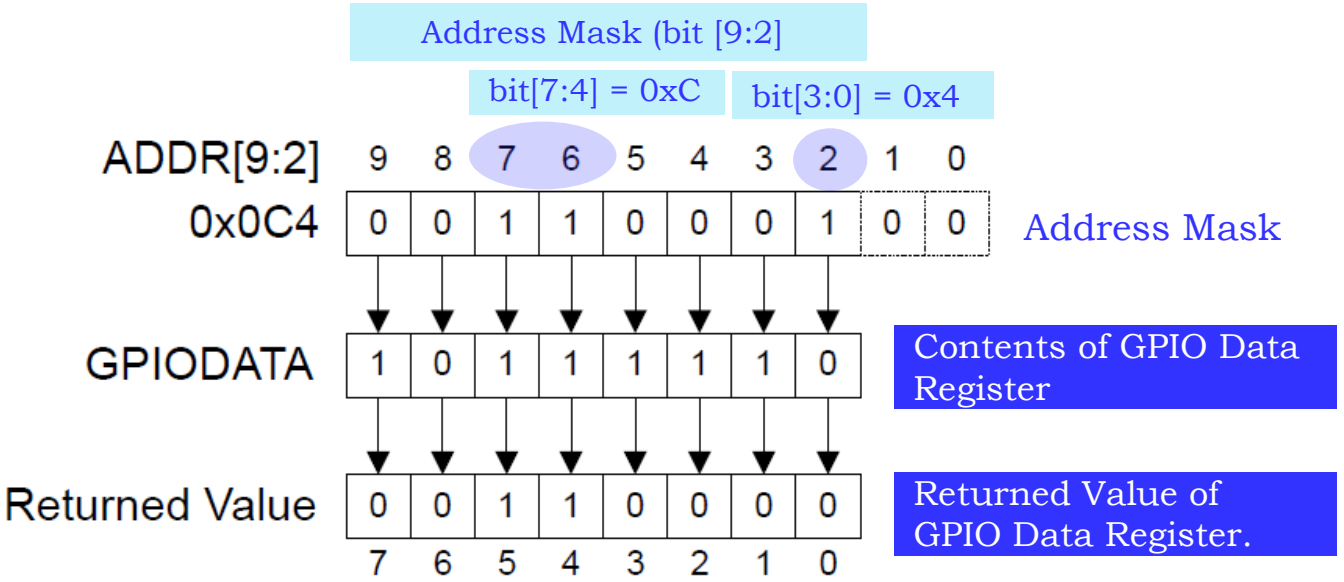
Ex 3: GPIO Address Masking *(Answer)*

- I want to Write to bits 7, 6, 5, 3, 1 of GPIO->DATA register.

- Data written = 0b1111.0011
- Address Mask = **0x3A8**
- Address to write to: [GPIO->DATA base + offset] = **[GPIO->DATA base + 0x3A8]**

Bit:	9	8	7	6	5	4	3	2	1	0	
ADDR MASK (0b)	1	1	1	0	1	0	1	0	0	0	
ADDR MASK (0x)	3		A				8				
Bit:	7	6	5	4	3	2	1	0			
DATA WRITTEN	1	1	1	1	0	0	1	1			
Bit:	7	6	5	4	3	2	1	0			
GPIO->DATA	1	1	1	0	1	0	0	1	[BEFORE WRITE]		
Bit:	7	6	5	4	3	2	1	0			
GPIO->DATA	1	1	1	0	0	0	1	1	[AFTER WRITE]		

GPIO Address Masking (**READ**)



- Bits [9:2] are used as the address mask.
  - To enable a bit to be read, we set the corresponding bit to '1'.
- In above example, to read only bits 5,4 & 0, we read from address: [GPIO Data Register base addr + 0xC4].
- **Bits not set in address mask are returned (read) as '0'.**

Source: Tiva TM4C123GH6PM Microcontroller Data Sheet (spmu376e.pdf, p654)

Ex 4: GPIO Address Masking

- I want to Read bits 7, 5, 2 of GPIO->DATA register.

- Address Mask = ?
- To do so, we perform a Read at address: [GPIO->DATA base addr + offset].

Bit:	9	8	7	6	5	4	3	2	1	0
ADDR MASK (0b)									0	0
ADDR MASK (0x)										

Bit:	7	6	5	4	3	2	1	0	
GPIO->DATA	1	1	1	0	1	0	0	1	CURRENT DATA

Bit:	7	6	5	4	3	2	1	0
RETURNED VALUE								

Ex 4: GPIO Address Masking *(Answer)*

- I want to Read bits 7, 5, 2 of GPIO->DATA register.

- Address Mask = **0x290**
- We perform a Read at address [GPIO->DATA base addr + offset] = **[GPIO->DATA base addr + 0x290]**.
- Returned Value of GPIO->DATA = 0xA0.
- **Bits not set in address mask are read as 0, regardless of their value.**

ADDR MASK (0b)	1	0	1	0	0	1	0	0	0	0
ADDR MASK (0x)	2		9				0			

Bit:	7	6	5	4	3	2	1	0	
GPIO->DATA	1	1	1	0	1	0	0	1	CURRENT DATA

Bit:	7	6	5	4	3	2	1	0
RETURNED VALUE	**1**	0	**1**	0	0	**0**	0	0

GPIO Address Masking

- Why do we want to use **Address Masking**?
  - To allow for modification of individual bits in the GPIO Data registers. Needed in many embedded applications for bit operations.
  - Address masking is efficient as it allows for modifying individual or combination of GPIO pins using a single instruction.
  - Tiva LaunchPad makes use of address bits [9:2] as address mask.

- Disadvantage of Address Masking:
  - Each GPIO port's data register is mapped to 256 memory locations in the memory map. Thus, more memory needed.

Ex: GPIO Address Masking (GPIOD)

- **Port D** base addr = **0x4000.7000**.
- Macro to access only bit **PD1**:

	Address Mask Bits											
Offset	9	8	7	6	5	4	3	2	1	0	PORTD Addr:	
0x08	0	0	0	0	0	0	1	0	0	0	0x4000.7008	PD1

```
#define GPIOD_BASE 0x40007000UL
/** remember that addr mask goes from bit 2 to 9   **/
#define PD1(x) ( *((volatile uint32_t *) (GPIOD_BASE + 0x08)) = x << 1)
PD1(1); /* make PD1 high; other pins not affected */
PD1(0); /* make PD1 low; other pins not affected  */
```

- Macro to access only bit **PD2**:

	Address Mask Bits											
Offset	9	8	7	6	5	4	3	2	1	0	PORTD Addr:	
0x10	0	0	0	0	0	1	0	0	0	0	0x4000.7010	PD2

```
#define PD2(x) ( *((volatile uint32_t *) (GPIOF_BASE + 0x10)) = x << 2)
PD2(1); /* make PD2 high; other pins not affected */
PD2(0); /* make PD2 low; other pins not affected  */
```

- Alternatively, we can write:

```
#define PD1(x) ( *((volatile uint32_t *) (GPIOD_BASE + (0x02<<2))) = x << 1)
/* 0x02 = 0b0010 = 2nd bit  */
#define PD2(x) ( *((volatile uint32_t *) (GPIOD_BASE + (0x04<<2))) = x << 2)
/* 0x04 = 0b0100 = 3rd bit  */
```

Ex: GPIO Address Masking (GPIOB)

- Base address for **Port B** = **`0x4000.5000`**.
- To access only bit **PB4**:

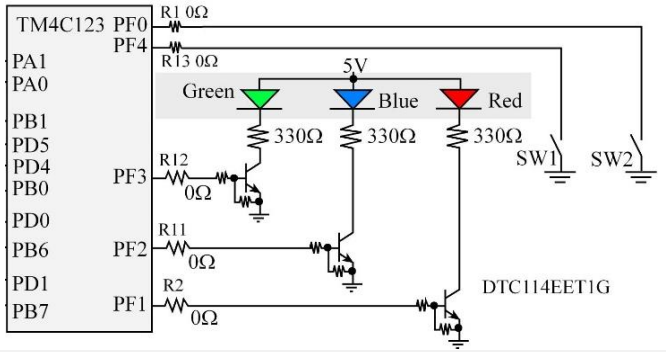
```
#define GPIOB_BASE 0x40005000UL

/** remember that addr mask goes from bit 2 to 9   **/
#define PB4(x) ( *((volatile uint32_t *) (GPIOD_BASE + 0x40)) = x << 4)

PB4(1); /* make PB4 high; other pins not affected */
PB4(0); /* make PB4 low; other pins not affected  */
```

	Address Mask Bits											
Offset	9	8	7	6	5	4	3	2	1	0	PORTD Addr:	
0x40	0	0	0	1	0	0	0	0	0	0	0x4000.5040	PB4

Bit PB4

Lab Macro (GPIOF) – `LED.h`

- Base address for **Port F** = 0x4002.5000.
- Color LED: Green (PF3), Blue (PF2), Red (PF1).
- To access colour LEDs on Tiva LaunchPad:




```
#define GPIOF_BASE 0x40025000UL
/** macro defined in LED.h      **/
#define LED_RGB_SET(x) ((*((volatile uint32_t *)(GPIOF_BASE+(0x0E<<2))))=(x)%0xFF)
```

```
#define RGB_OFF                 0x00
#define RGB_RED                 0x02
#define RGB_GREEN               0x08
#define RGB_BLUE                0x04
#define RGB_MAGENTA             0x06
#define RGB_YELLOW              0x0A
#define RGB_CYAN                0x0C
#define RGB_WHITE               0x0E

LED_RGB_SET(RGB_RED);           /* turn on red LED */
LED_RGB_SET(RGB_BLUE);          /* turn on blue LED */
```

Lab Macro (GPIOF)

```
#define GPIOF_BASE 0x40025000UL
#define LED_RGB_SET(x) ((*((volatile uint32_t *)(GPIOF_BASE+(0x0E<<2))))=(x)%0xFF)
```

0x0E defines only bits PF3, PF2, PF1 will be modified.

(shift right by 2)

Addr Bit:	9	8	7	6	5	4	3	2	1	0
ADDR MASK (0b)	0	0	0	0	1	1	1	0	0	0
ADDR MASK (0x)	0		3				8			

```
/* to turn on Red LED, x = 0x02  */
LED_RGB_SET(0x02);
```

Data Bit:	7	6	5	4	3	2	1	0	
GPIOF->DATA	0	0	0	0	0	0	1	0	**(=0x02)**

Write 0x02 through addr mask; only bits 3,2,1 are affected.

Since bit 1 is a '1', Red LED is turned ON.

GPIO Initialization

Let's put everything together ….

GPIO Port/Pins Initialization

To initialize a GPIO port/pins, we perform the following steps:

1. Activate clock for the port.
2. Check that port is ready for access.
3. Unlock the port & pins- only needed for pins PC[3:0], PD7 & PF0.
4. Disable analog function.
5. Clear bits in the PCTL register to select regular digital function.
6. Set pin direction in the direction register (DIR).
   - Input = ‘0’; Output = ‘1’.
7. Clear bits in the alternate function register.
8. Enable the port pin.

Example: Port A Initialization

```
 #define BIT( x )    (1U<<(x))   /* defined in common.h   */

 #define PA_0   0U   /* defined in hal.h    */
 #define PA_1   1U
 #define PA_2   2U
 #define PA_3   3U
 #define PA_4   4U
 #define PA_5   5U
 #define PA_6   6U
 #define PA_7   7U
```

```
/* set GPIO Port pins 7-4 as inputs, 3 to 0 as outputs */
void Port_Init( void )  /* defined in hal.c    */
{
   SYSCTL->RCGCGPIO |= SYSCTL_RCGCGPIO_R0; /* enable clock to GPIOA */

   /* Wait for GPIOA to be ready */
   while( 0 == (SYSCTL->PRGPIO & SYSCTL_PRGPIO_R0)){};

   GPIOA->DIR |= BIT(PA_3) | BIT(PA_2) | BIT(PA_1) | BIT(PA_0); /* outputs  */
   GPIOA->DIR &= ~( BIT(PA_4) | BIT(PA_5) | BIT(PA_6) | BIT(PA_7)); /* inputs  */
   GPIOA->PUR |= BIT(PA_4) | BIT(PA_5) | BIT(PA_6) | BIT(PA_7); /* pull-up inputs */
   GPIOA->DEN |= BIT(PA_7) | BIT(PA_6) | BIT(PA_5) | BIT(PA_4) |
                 BIT(PA_3) | BIT(PA_2) | BIT(PA_1) | BIT(PA_0);
}
```

Example: Port F Initialization

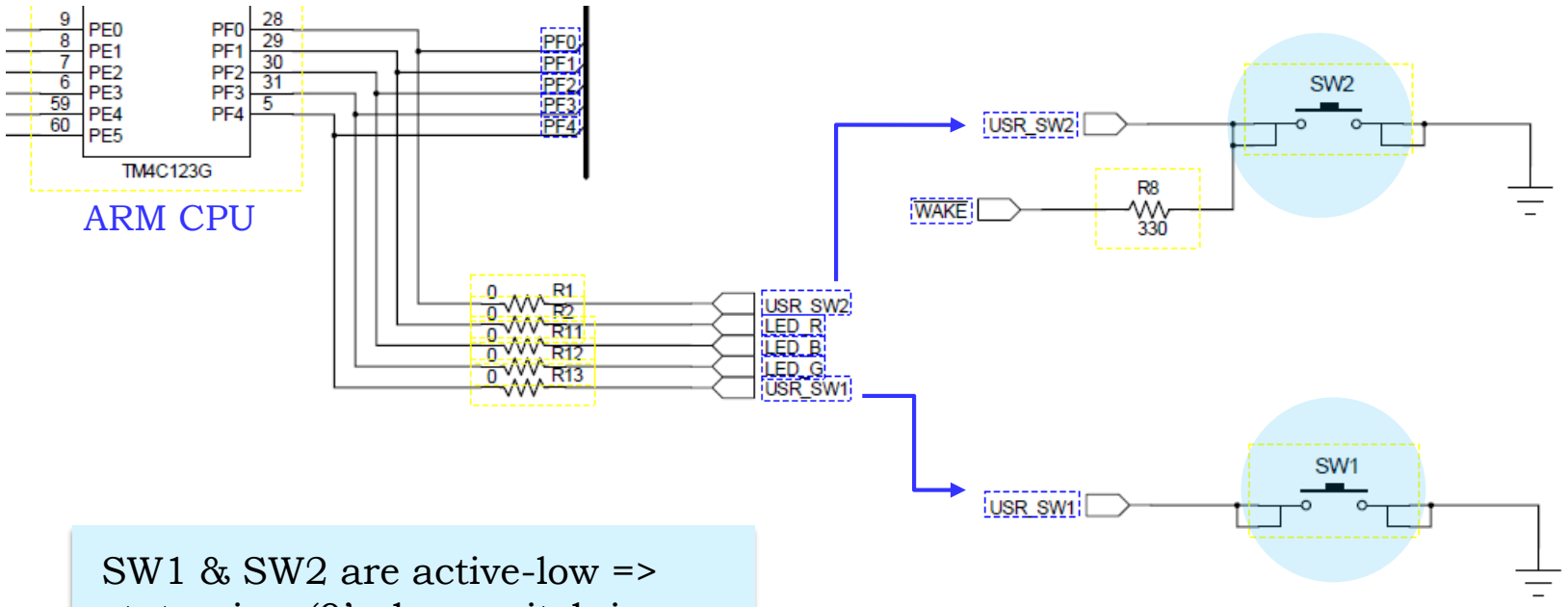
```
#define GPIO_LOCK_KEY  0x4C4F434B  // Unlocks GPIO_CR register

#define BIT( x )   (1U<<(x))   /* defined in common.h   */
#define PF_SW2               0U
#define PF_LED_RED           1U
#define PF_LED_BLUE          2U
#define PF_LED_GREEN         3U
#define PF_SW1               4U
```

```
void Port_Init( void )
{
   SYSCTL->RCGCGPIO |= SYSCTL_RCGCGPIO_R5; /* enable clock to Port F */

   /* Wait for GPIOF to be ready */
   while( 0 == (SYSCTL->PRGPIO & SYSCTL_PRGPIO_R5)){};

   GPIOF->LOCK = GPIO_LOCK_KEY;        /* enable lock */
   GPIOF->CR |= BIT(PF_SW2);
   GPIOF->DIR |= BIT(PF_LED_RED) | BIT(PF_LED_BLUE) | BIT(PF_LED_GREEN);
   GPIOF->DIR &= ~( BIT(PF_SW1) | BIT(PF_SW2) );
   GPIOF->PUR |= BIT(PF_SW1) | BIT(PF_SW2);
   GPIOF->DEN |= BIT(PF_LED_RED) | BIT(PF_LED_BLUE) | BIT(PF_LED_GREEN) |
                 BIT(PF_SW1) | BIT(PF_SW2);
}
```

Reading GPIO Port Pins

Macros

Tiva LaunchPad – SW1, SW2 Schematics



SW1 & SW2 are active-low => status is a '0' when switch is pressed.

SW1 at PF4; SW2 at PF0.

Source: Tiva C Series TM4C123G LaunchPad Evaluation Board – User Guide (spmu296.pdf)

Reading GPIOF->PF4 (SW1)

```
#define SW1_STATUS (~GPIOF->DATA>>4 & 0x0001? 1 : 0 )
```

	PF7	PF6	PF5	PF4	PF3	PF2	PF1	PF0
1) GPIOF->DATA (assume SW1 pressed)	X	X	X	**0**	X	X	X	**1**
				(**SW1**)				(**SW2**)

Data is read from GPIOF->DATA register. Assume **SW1 is pressed**, SW2 not pressed => PF4 is '0', PF0 is '1". Buttons are active-low. 'X' = don't care.

2) ~GPIOF->DATA	X	X	X	**1**	X	X	X	**0**

Bit inversion of GPIOF->DATA register to reverse the logic.

3) ~GPIOF->DATA>>4	0	0	0	0	X	X	X	**1**

Shift PF1 bits to LSB (bit 0).

4) & 0x0001	0	0	0	0	0	0	0	**1**

Bit-AND with bit-pattern '00000001' to zero-out all bits except the bit related to SW1.
Macro returns a '1' if SW1 is pressed & '0' if SW1 is not pressed.

Reading GPIOF->PF0 (SW2)

```
#define SW2_STATUS    (~GPIOF->DATA & 0x0001? 1 : 0 )
```

1) GPIOF->DATA
(assume SW2 pressed)

PF7	PF6	PF5	PF4	PF3	PF2	PF1	PF0
X	X	X	**1**	X	X	X	**0**
			(**SW1**)				(**SW2**)

Data is read from GPIOF->DATA register. Assume **SW2 is pressed**, SW1 not pressed => PF4 is '1', PF0 is '0".
'X' = don't care.

2) ~GPIOF->DATA

X	X	X	**0**	X	X	X	**1**

Bit inversion of GPIOF->DATA register to reverse the logic. Since PF0 is at LSB position, there is no need to perform a shift operation.

3) & 0x0001

0	0	0	0	0	0	0	**1**

Bit-AND with bit-pattern '00000001' to zero-out all bits except the bit related to SW2.
Macro returns a '1' if SW2 is pressed & '0' if SW2 is not pressed.

Review Questions

1. Determine if the following statements are TRUE or FALSE:
   a) Each GPIO port can have up to 16 port pins.
   b) GPIO port pins can be individually configured to be Input or Output pins.
   c) The maximum toggle speed of a GPIO pin when connected to the APB bus is the system bus clock speed.
   d) The default drive strength of a GPIO port pin is 2mA.
   e) The GPIO DEN register needs to be set to '1' to configure a port pin to be an Analog function.
   f) Setting the corresponding bit in the GPIOAMSEL register enables the analog function for that pin (if there is a valid a mapping).
   g) Analog function is an Alternate function.
   h) GPIO port pins with 'Special Considerations' are set to their Alternate functions upon power up.
2. I would like to map port PE0 to function as an alternate function as UART7 Rx. What value should be written to the GPIOPCTL register?
3. Why are port pins PC0 to PC3 not available on the Tiva LaunchPad?
4. What is the GPIODEN register used for? What is its default function upon power up?

Review Questions

5. GPIO pin drive strengths can be set through the GPIODR2R (for 2mA), GPIODR4R or GPIODR8R. What is the drive strength for a GPIO pin if we write to both GPIODR4R and GPIOR8R consecutively during initialization?
6. How would you check if a GPIO port is ready to be accessed after initialization?
7. Write a C function **Port_Init()** to initialize Port E, bit 4 as Input & Port B, pin 0 as Output.
8. List the GPIO port pins that do not default to be GPIO pins?
9. Which are the GPIO pins that requires us to unlock and commit before it's configuration can be change?
10. List the GPIO port registers that requires an 'unlock-commit' sequence before access.
11. I would like to use UART7 TX and RX in my design. How should I configure the port pins in my program? Consider the DEN, AMSEL, PCTL,, AFSEL, … registers.
12. I would like to write to only bits 6, 3 and 1 of GPIOB register. Using address mask, which offset address should I write to?

Review Questions

13. I want to read only bits 5 and 4 of a GPIODATA register through a Read address mask. The current GPIODATA is 0xAA. What is the data read using an address mask?
14. What is the GPIOLOCK & GPIOCR used for?
15. A GPIO port can be accessed through either the APB and AHB bus. Which register is used to select the bus connection?