香港中文大學
The Chinese University of Hong Kong

CENG2400 Embedded System Design

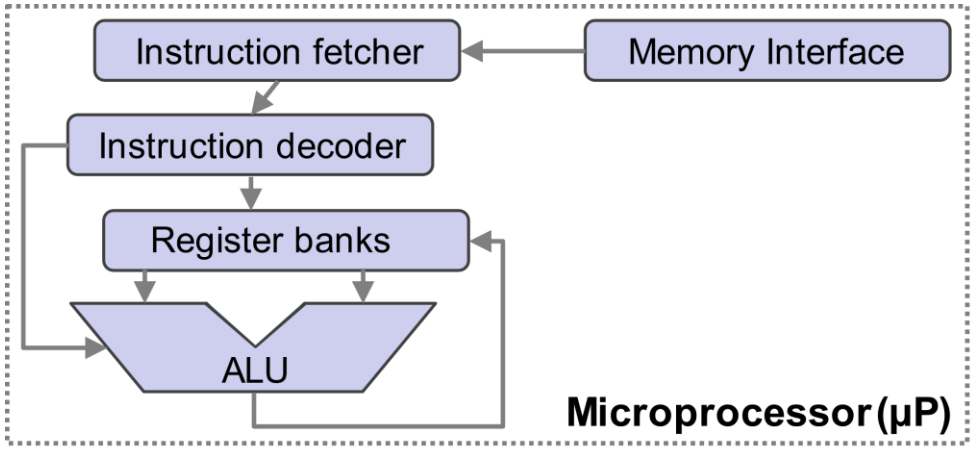
Lecture 02: General Purpose Input/Output

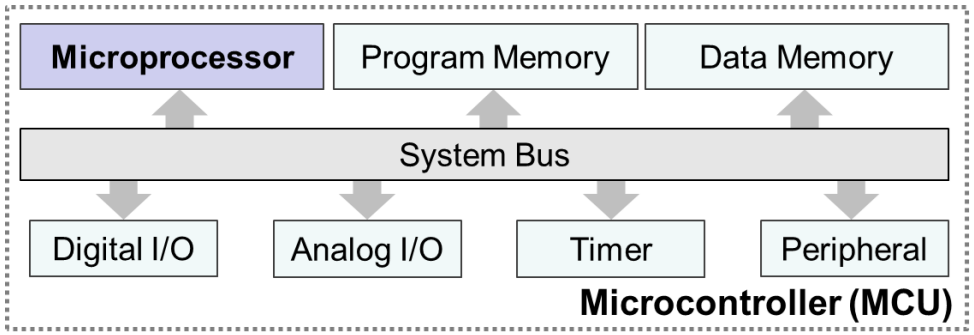
Ming-Chang YANG

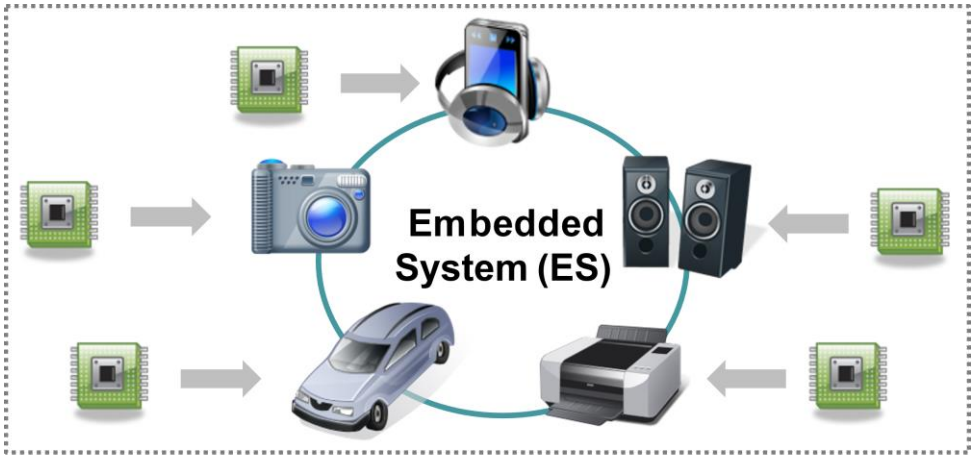
mcyang@cse.cuhk.edu.hk

Thanks to Prof. Q. Xu and Drs. K. H. Wong, Philip Leong, Y.S. Moon, O. Mencer, N. Dulay, P. Cheung for some of the slides used in this course!

Recall: µP → MCU → ES

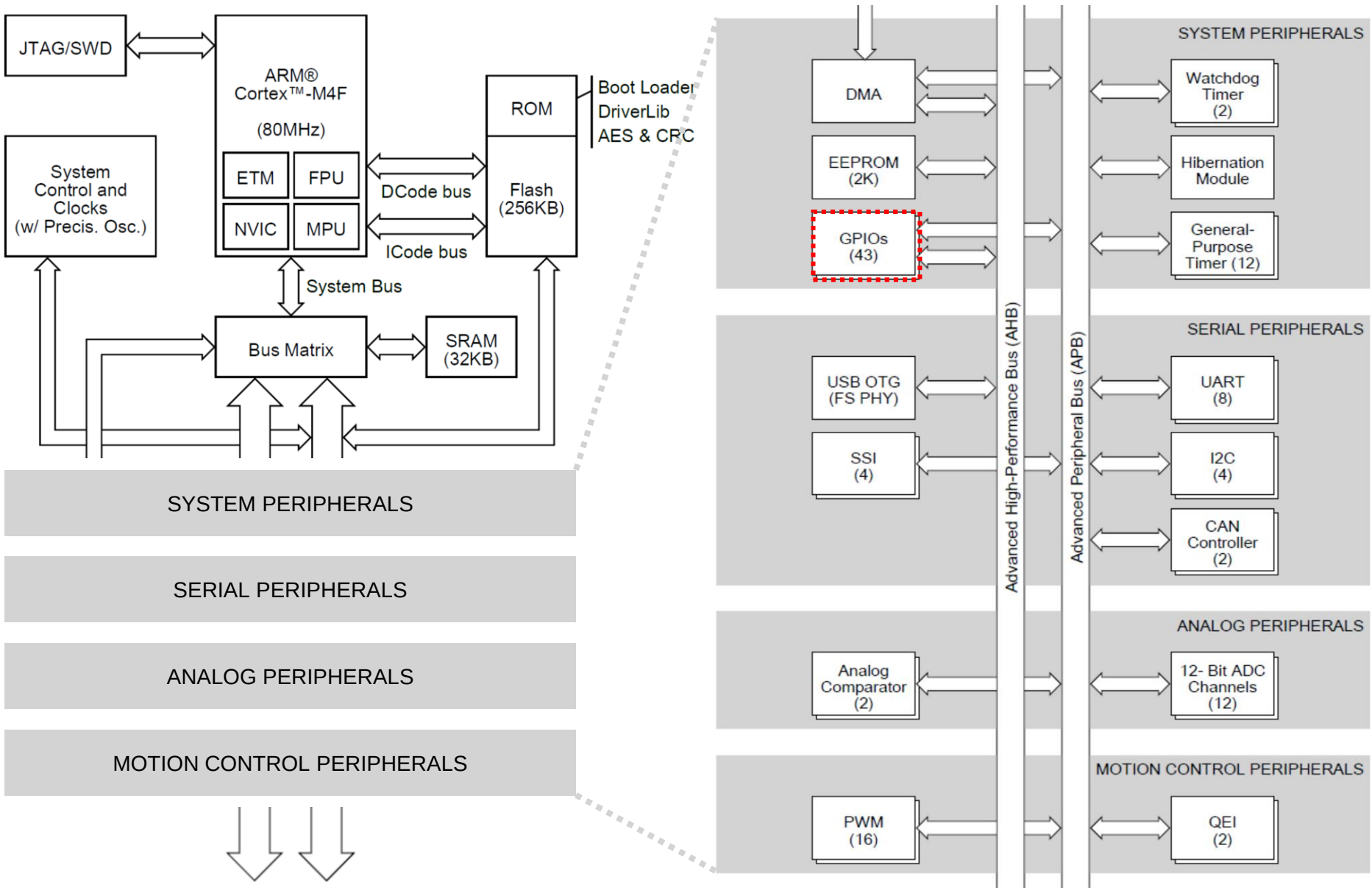






- **Microprocessor (µP)**
  - As a compact form of CPU implemented on an IC.
- **Microcontroller (MCU)**
  - Has the microprocessor;
  - Integrated with other components including memory, digital/analog IOs, and other peripherals.
- **Embedded System (ES)**
  - Typically implemented using MCUs;
  - Often integrated into a larger mechanical or electrical system.

Recall: MCU: Tiva™ TM4C123GH6PM

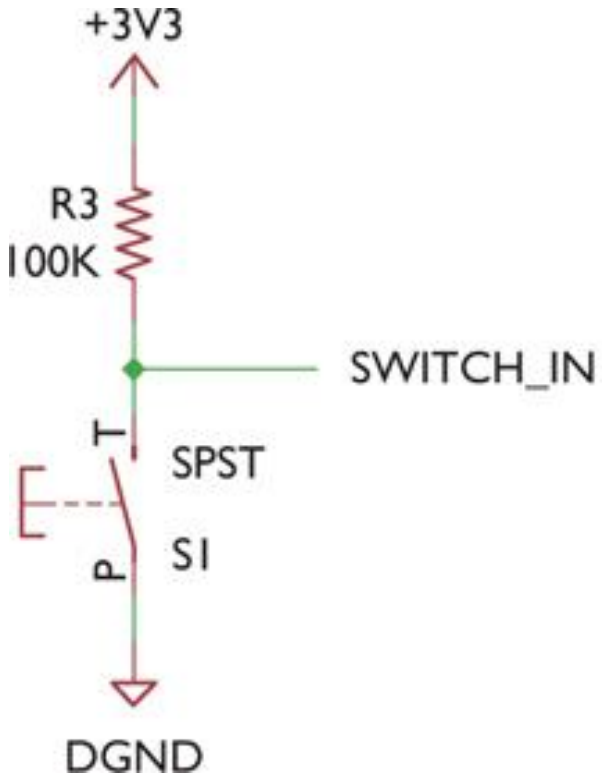


Outline

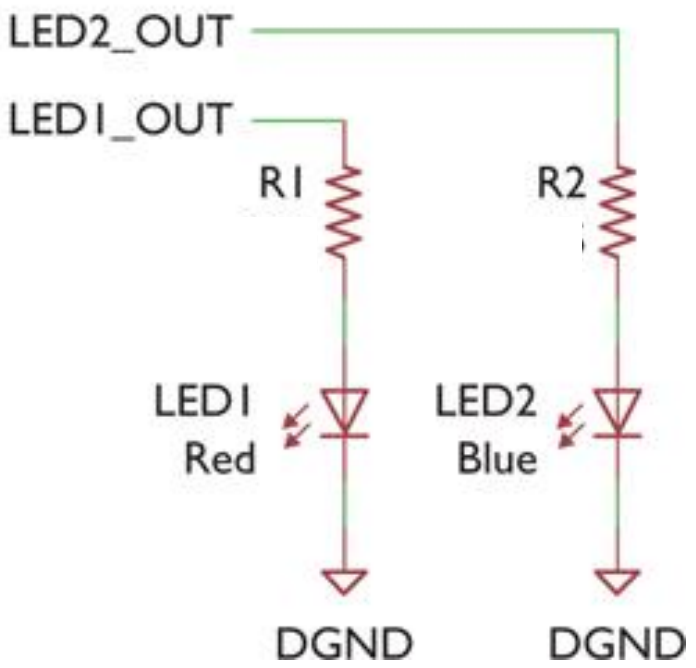
- **GPIO Basics**
  - What's a one? A zero?
  - How does it work?
- **GPIO Module on Tiva™ TM4C123GH6PM**
  - Typical Structure of a GPIO Pin
  - Memory-mapped I/O
  - Control Registers
- **TivaWare™ Peripheral Driver Library**
  - Direct Register Access Model
  - Software Driver Model
  - Programming Example: Toggling LEDs

General Purpose Input/Output (GPIO)

- **GPIO** are peripherals with digital input and output bits:
  - An input port enables us to read a 1 or 0 from a GPIO pin.
  - An output port enables us to set a GPIO pin to a 1 or 0.

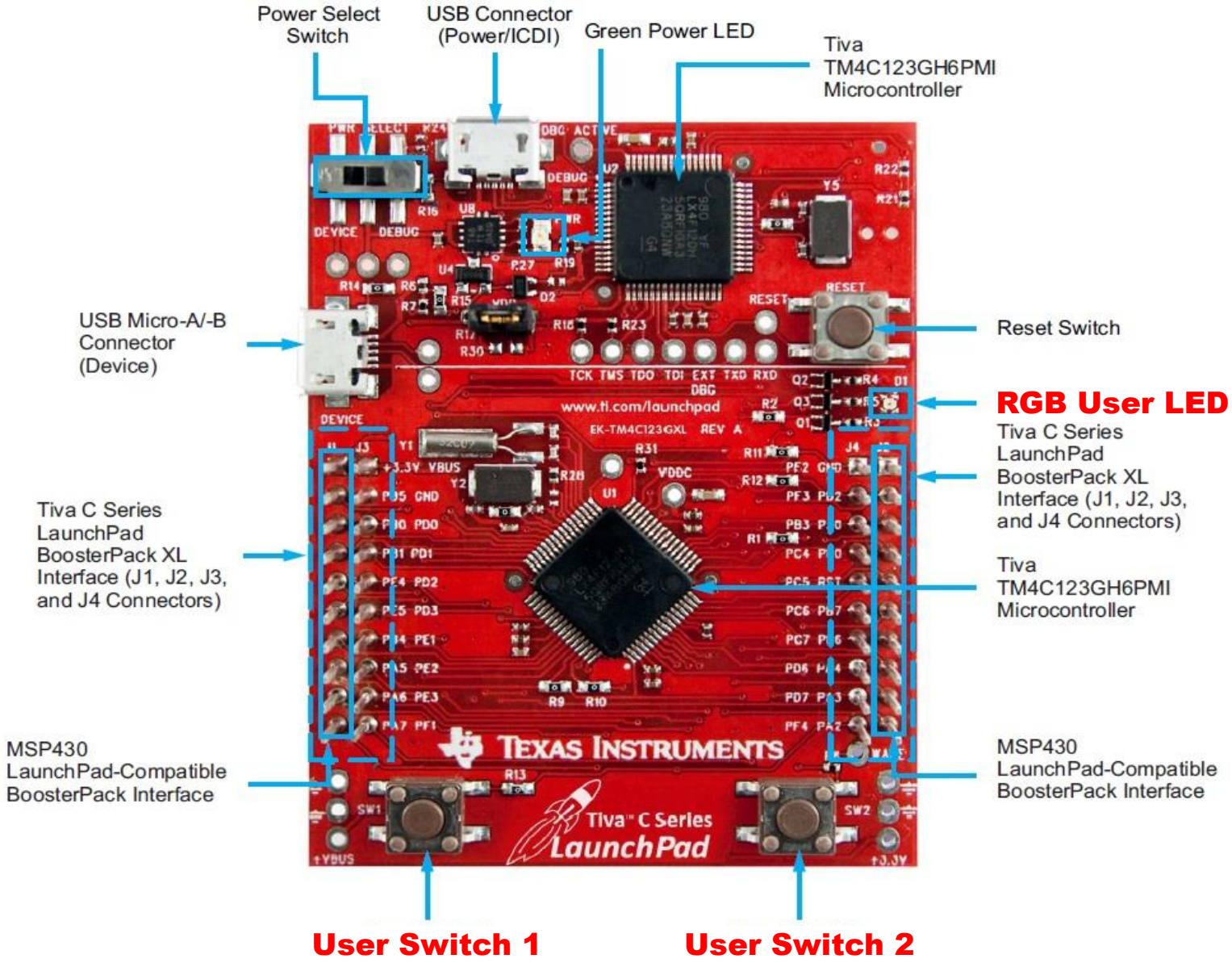


Example: Switch (Input)



Example: LED (Output)

Switch/LED on Tiva™ LaunchPad

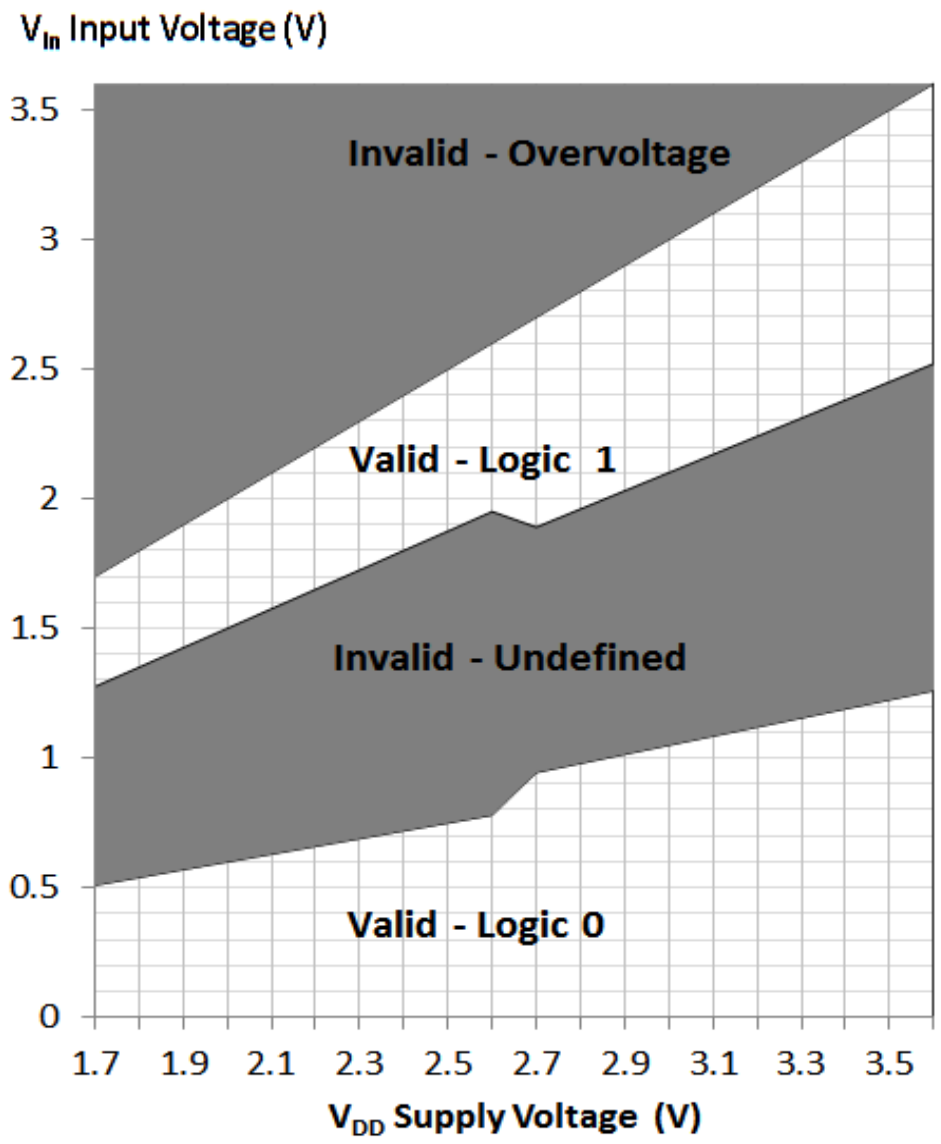


Inputs: What's a one? A zero?

- **Digital inputs** are interpreted based on voltage levels.
  - The threshold voltages depend on the supply voltage V_{DD}, e.g.:

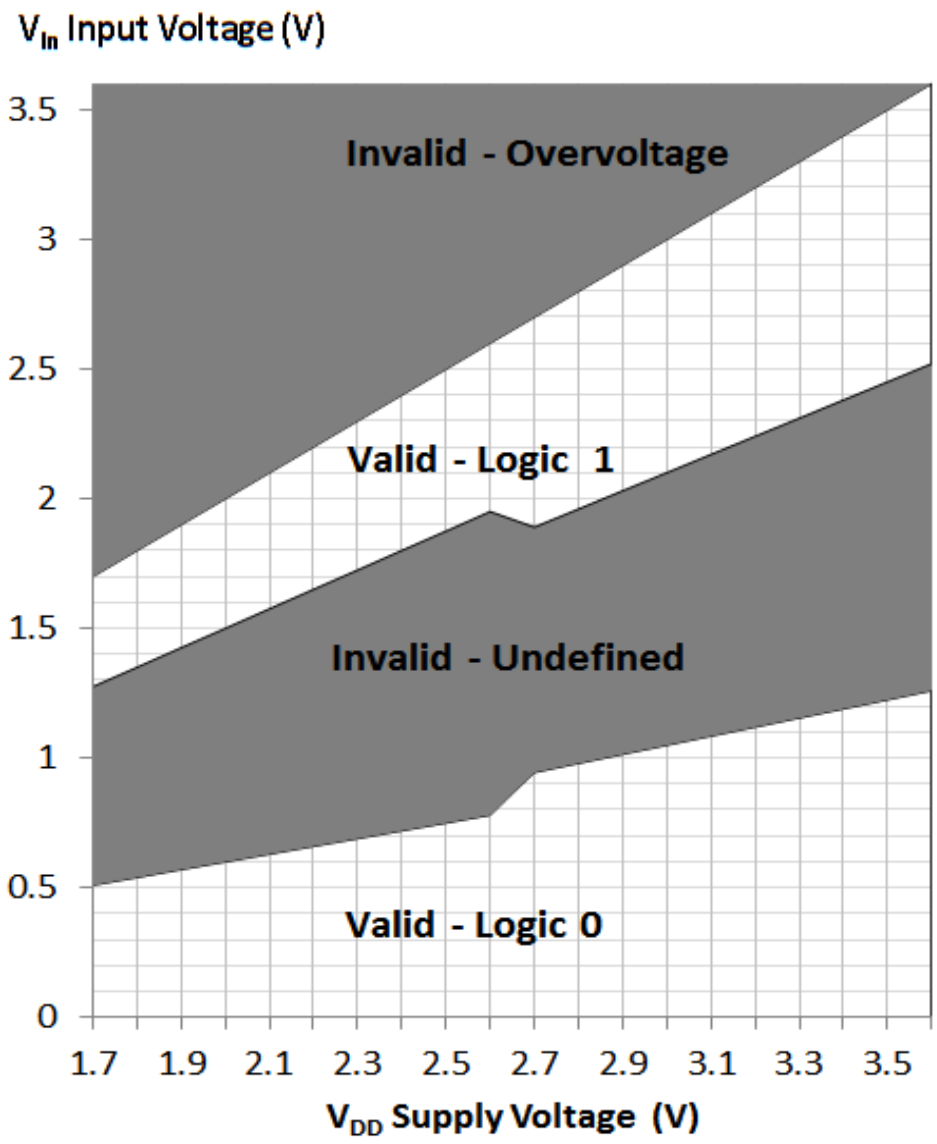
$$Logic\ One: 0.7 * V_{DD} \sim V_{DD}$$
$$Logic\ Zero: 0\ V \sim 0.35 * V_{DD}$$

  - Note: Voltages exceeding V_{DD} or falling GND may damage the chip.



Class Exercise 2.1

- If V_{DD} is set to 3.3 V, what levels of input voltages will be interpreted as a logic zero and a logic one, respectively?



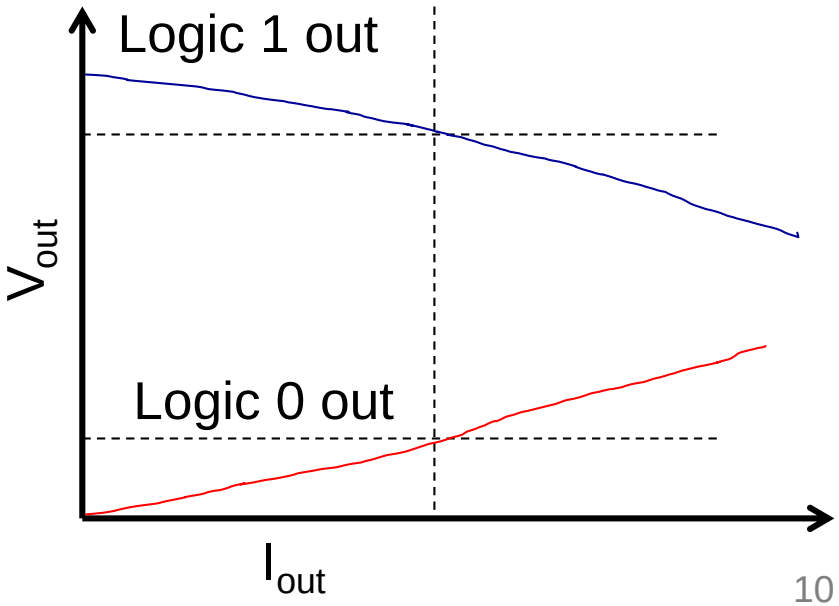
Outputs: What's a one? A zero?

- **Output voltages** for many MCUs are defined as:

$$Logic\ One\ or\ V_{OH\ (out\ high)}: 1 * V_{DD} - 0.5 * V \sim 1 * V_{DD}$$
$$Logic\ Zero\ or\ V_{OL\ (out\ low)}: 0\ V \sim 0.5\ V$$

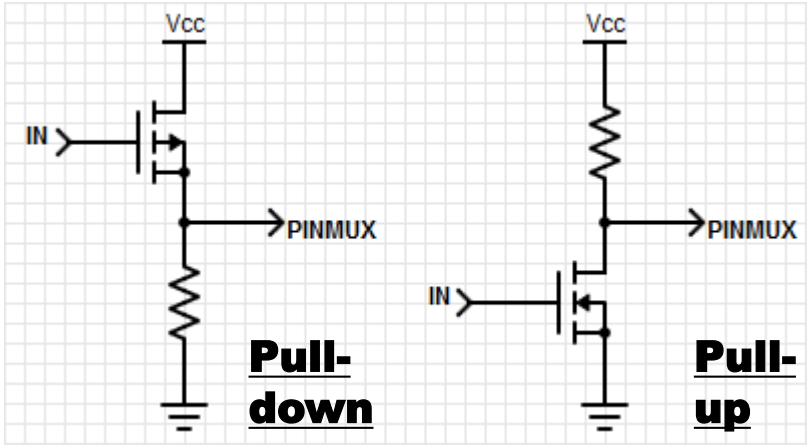
  - E.g., if V_{DD} is 3.3 V, V_{OH} is between 2.8 V and 3.3 V, and V_{OL} is between 0 and 0.5 V.

- Note: Output voltages shall depend on the current drawn by load on pin.
  - The above values only hold true when current < 5 mA and V_{DD} > 2.7 V.

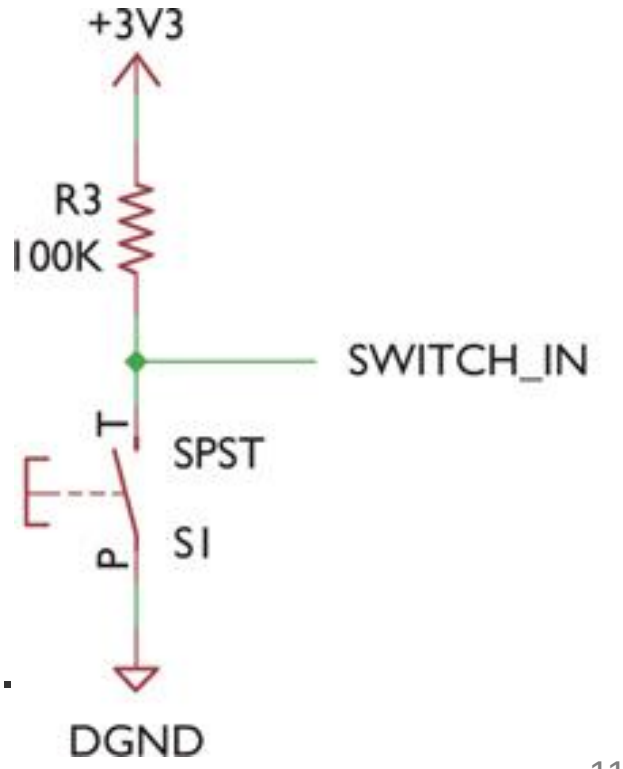


Switch (Input): How does it work?

- **Pull-up/pull-down resistors** ensure a known value on the output if a pin is left floating.

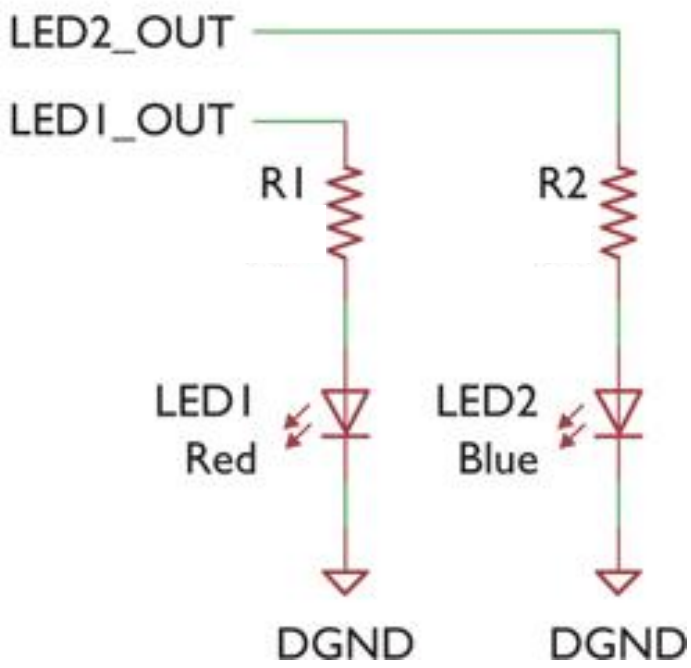


- The signal SWITCH_IN is connected to a switch and a pull-up resistor.
  - When the switch is pressed, the signal is set to low (logic zero);
    - The high value pull-up resistor is to minimize the current consumption.
  - Otherwise, it is pulled high (logic one).



LED (Output): How does it work?

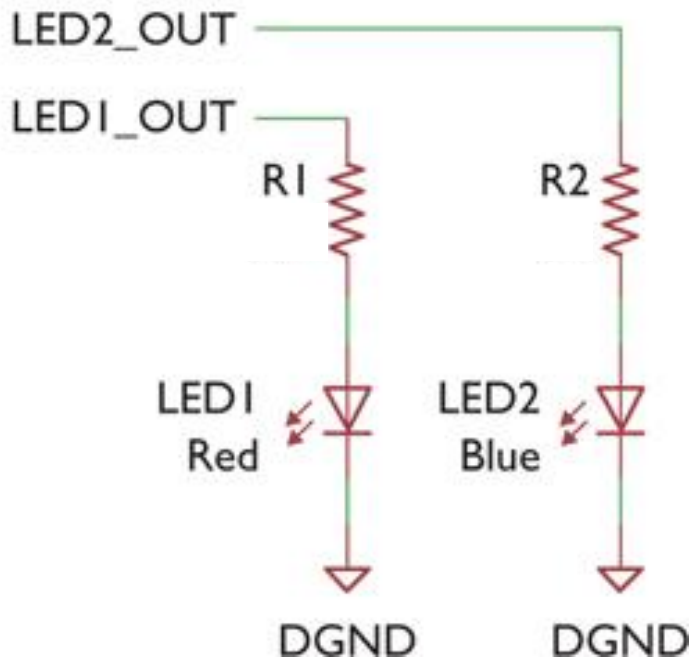
- Each LED has its anode (+) connected to an output signal (i.e., LED1/2_OUT) and its cathode (-) connected to ground.
- The resistor (i.e., R1/R2) is needed to limit the current to a value which is safe for both LED and MCU port driver.
  - V_{LED} for the **Red** LED: $\sim 1.8\ V$
  - V_{LED} for the **Blue** LED: $\sim 2.7\ V$



- An LED's brightness is roughly proportional to the current flowing through it.

Class Exercise 2.2

- Recall that
  - V_{LED} for the **Red** LED: $\sim 1.8\ V$
  - V_{LED} for the **Blue** LED: $\sim 2.7\ V$
- Suppose I_{LED} = 4 mA and V_{DD} = 3.0 V, derive the resistor values.

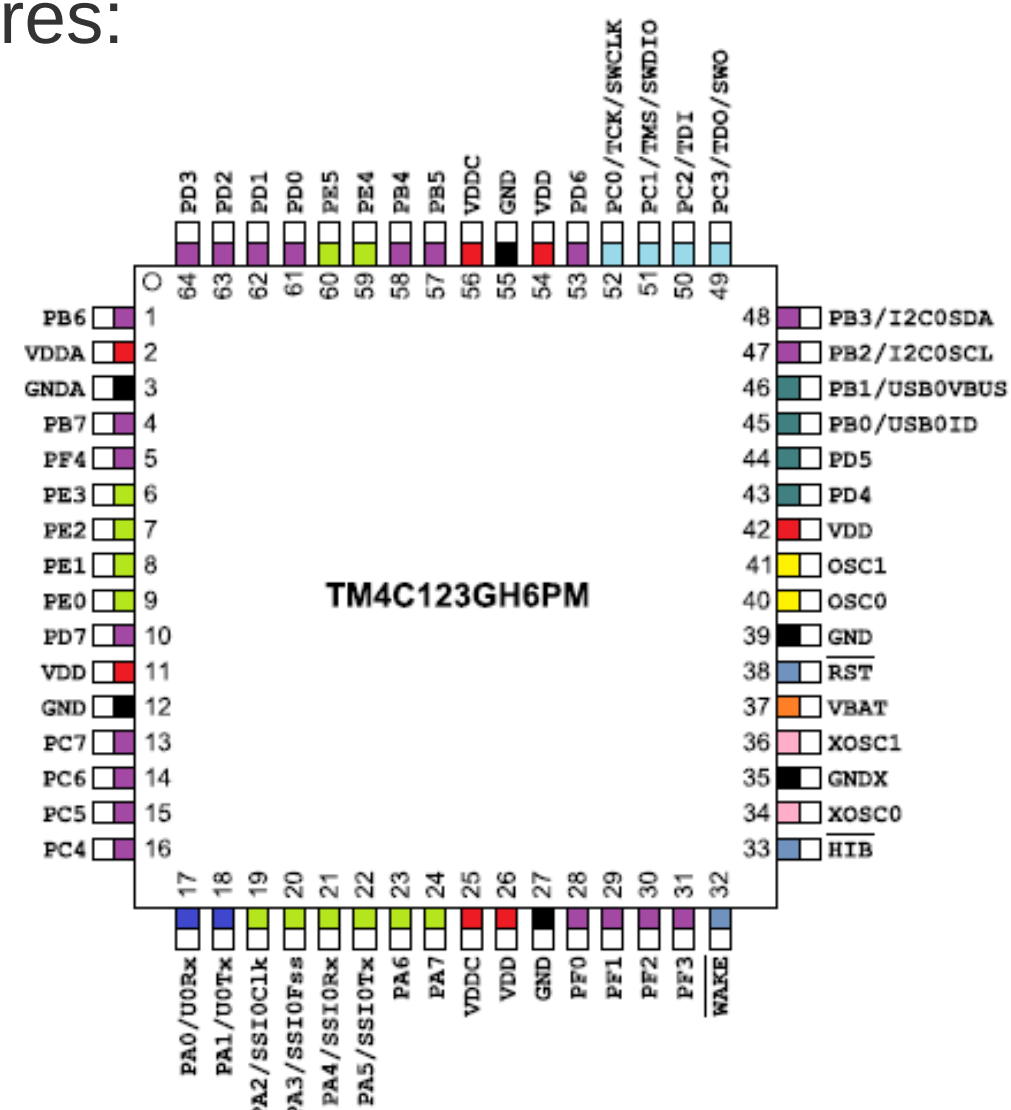


Outline

- **GPIO Basics**
  - What's a one? A zero?
  - How does it work?
- **GPIO Module on Tiva™ TM4C123GH6PM**
  - Typical Structure of a GPIO Pin
  - Memory-mapped I/O
  - Control Registers
- **TivaWare™ Peripheral Driver Library**
  - Direct Register Access Model
  - Software Driver Model
  - Programming Example: Toggling LEDs

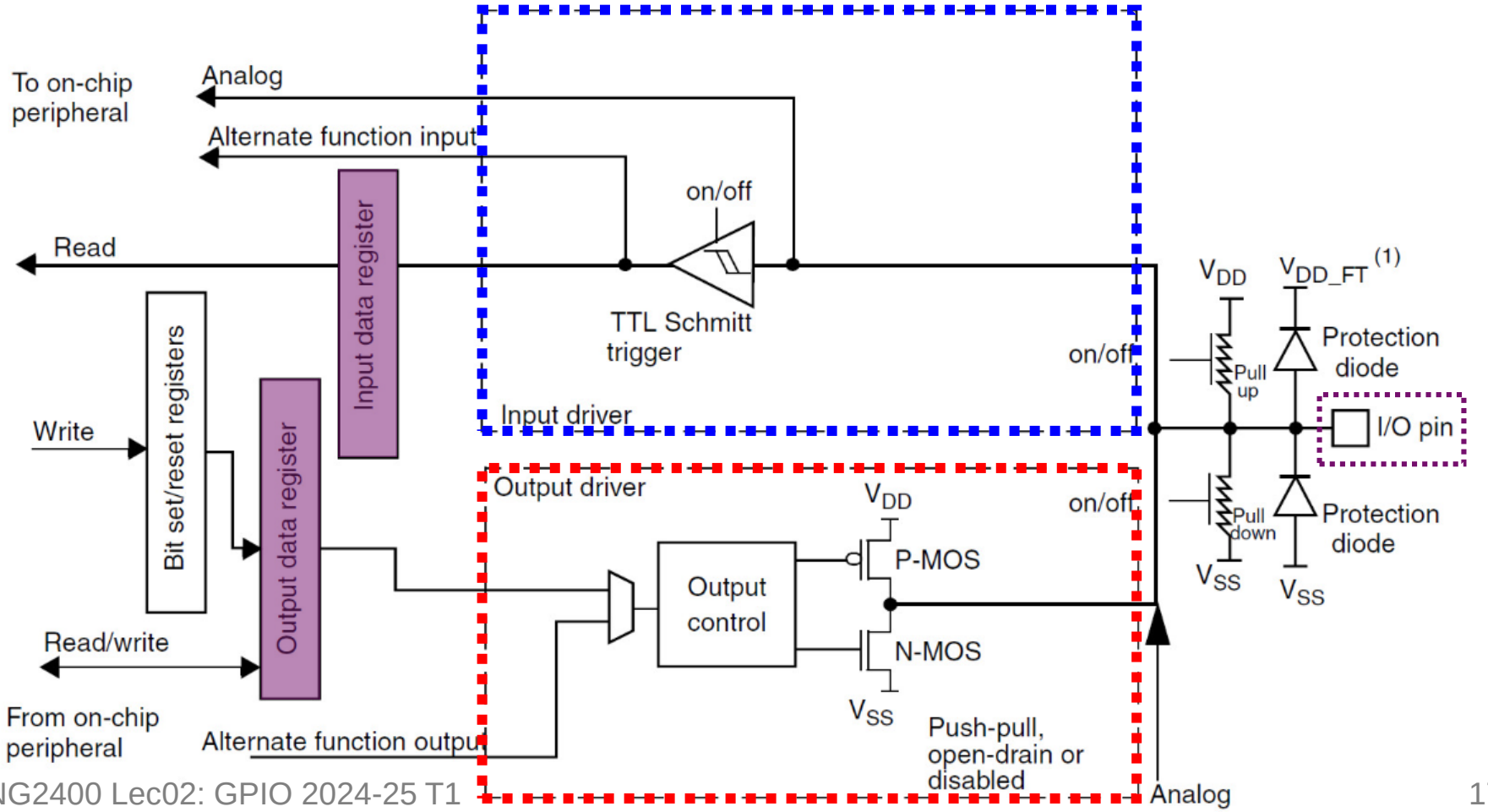
GPIO Module on Tiva™ TM4C123GH6I

- The GPIO module features:
  - Six GPIO blocks, each corresponding to an GPIO port (i.e., PA~PF);
  - Up to 43 programmable I/O pins;
  - Flexible pin “MUX”ing;
    - Use as GPIO or alternate peripheral functions;
  - 5-V-tolerant in input;
  - Programmable control for GPIO interrupts;
  - Bit masking in both R/W through address lines;
  - Etc.



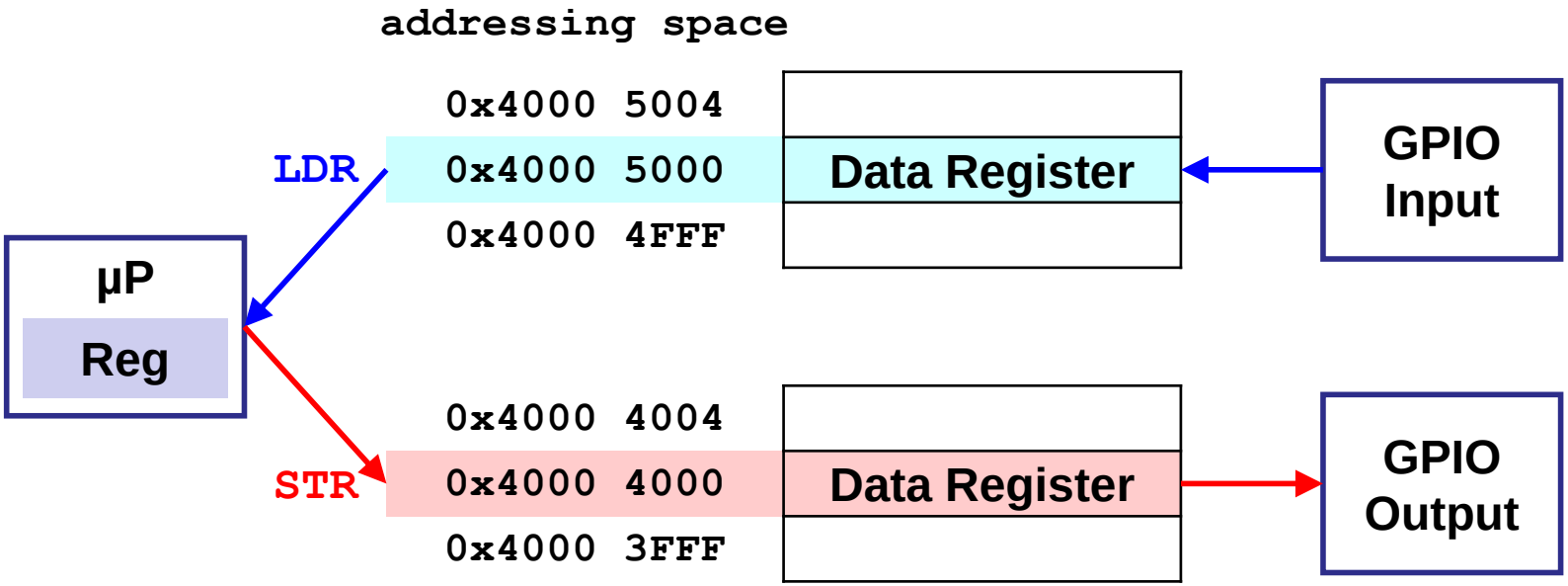
Typical Structure of a GPIO Pin

- The I/O pin can be configured to an input/output driver.
  - We can interact with a physical pin by reading/writing "memory-mapped controller registers".



Memory-mapped I/O

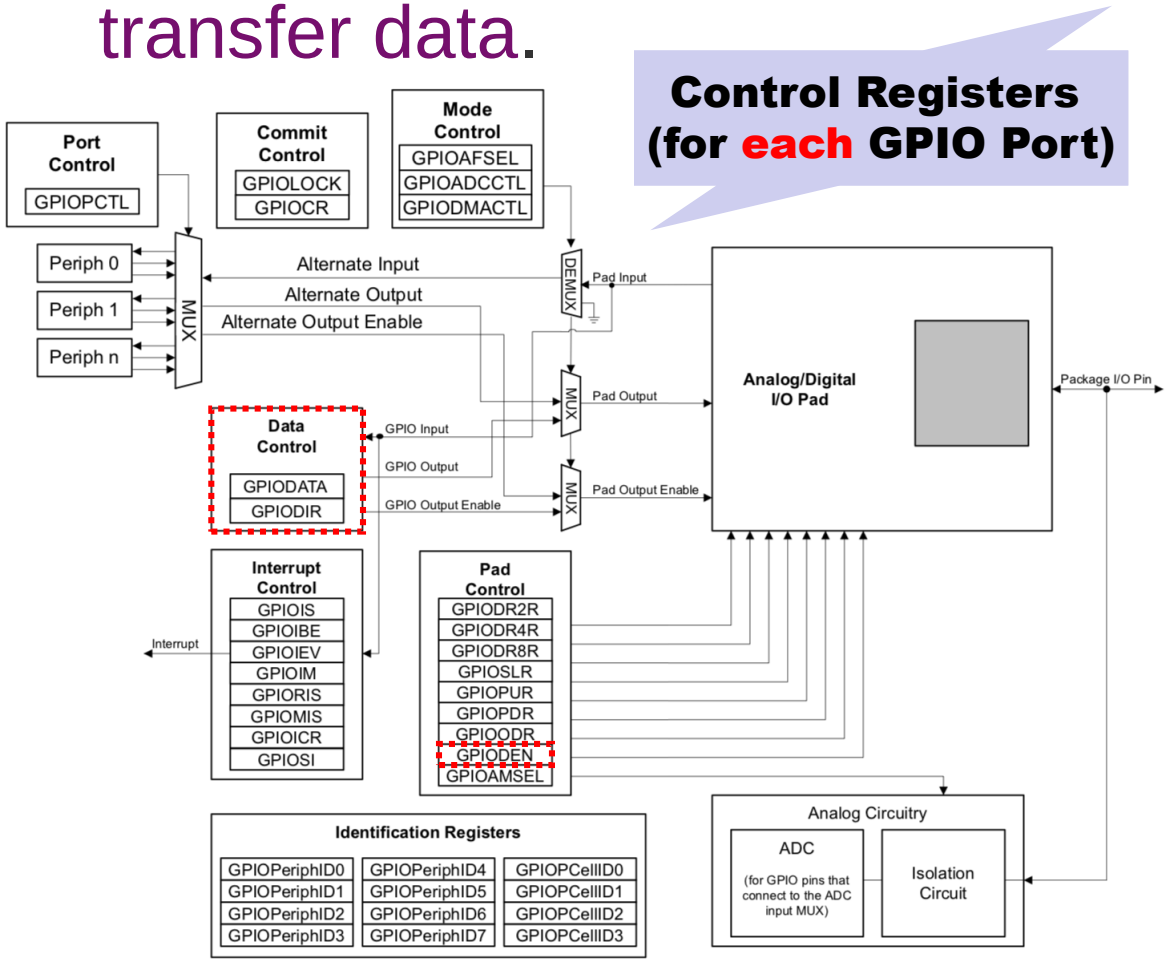
- **Memory-mapped (vs. Port-mapped) I/O**
  - Each device register associated with a **physical address** in the addressing space of µP (rather than a dedicated port).
  - Use native CPU instructions (rather than special ones).
    - E.g., **`LDR`**/**`STR`** **`Reg, [address]`**



A **simpler** and **more convenient** way to interface peripherals!

Control Registers

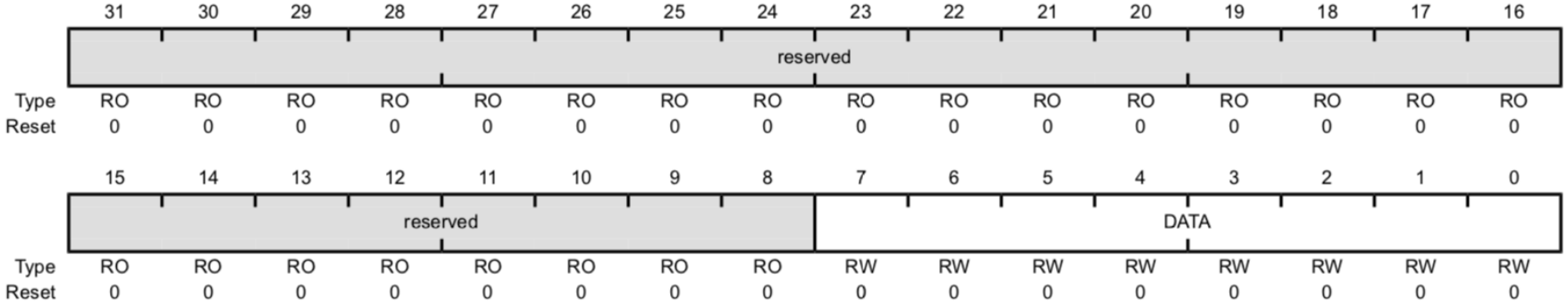
- **Control registers** allow us to configure peripherals, determine their status, and transfer data.



Offset	Name	Type	Reset	Description
0x000	GPIODATA	RW	0x0000.0000	GPIO Data
0x400	GPIODIR	RW	0x0000.0000	GPIO Direction
0x404	GPIOIS	RW	0x0000.0000	GPIO Interrupt Sense
0x408	GPIOIBE	RW	0x0000.0000	GPIO Interrupt Both Edges
0x40C	GPIOIEV	RW	0x0000.0000	GPIO Interrupt Event
0x410	GPIOIM	RW	0x0000.0000	GPIO Interrupt Mask
0x414	GPIORIS	RO	0x0000.0000	GPIO Raw Interrupt Status
0x418	GPIOMIS	RO	0x0000.0000	GPIO Masked Interrupt Status
0x41C	GPIOICR	W1C	0x0000.0000	GPIO Interrupt Clear
0x420	GPIOAFSEL	RW	-	GPIO Alternate Function Select
0x500	GPIODR2R	RW	0x0000.00FF	GPIO 2-mA Drive Select
0x504	GPIODR4R	RW	0x0000.0000	GPIO 4-mA Drive Select
0x508	GPIODR8R	RW	0x0000.0000	GPIO 8-mA Drive Select
0x50C	GPIOODR	RW	0x0000.0000	GPIO Open Drain Select
0x510	GPIOPUR	RW	-	GPIO Pull-Up Select
0x514	GPIOPDR	RW	0x0000.0000	GPIO Pull-Down Select
0x518	GPIOSLR	RW	0x0000.0000	GPIO Slew Rate Control Select
0x51C	GPIODEN	RW	-	GPIO Digital Enable
0x520	GPIOLOCK	RW	0x0000.0001	GPIO Lock
0x524	GPIOCR	-	-	GPIO Commit
0x528	GPIOAMSEL	RW	0x0000.0000	GPIO Analog Mode Select
0x52C	GPIOPCTL	RW	-	GPIO Port Control
0x530	GPIOADCCTL	RW	0x0000.0000	GPIO ADC Control
0x534	GPIODMACTL	RW	0x0000.0000	GPIO DMA Control
0xFD0	GPIOPeriphID4	RO	0x0000.0000	GPIO Peripheral Identification 4
0xFD4	GPIOPeriphID5	RO	0x0000.0000	GPIO Peripheral Identification 5
0xFD8	GPIOPeriphID6	RO	0x0000.0000	GPIO Peripheral Identification 6
0xFDC	GPIOPeriphID7	RO	0x0000.0000	GPIO Peripheral Identification 7
0xFE0	GPIOPeriphID0	RO	0x0000.0061	GPIO Peripheral Identification 0
0xFE4	GPIOPeriphID1	RO	0x0000.0000	GPIO Peripheral Identification 1
0xFE8	GPIOPeriphID2	RO	0x0000.0018	GPIO Peripheral Identification 2
0xFEC	GPIOPeriphID3	RO	0x0000.0001	GPIO Peripheral Identification 3
0xFF0	GPIOPCellID0	RO	0x0000.000D	GPIO PrimeCell Identification 0
0xFF4	GPIOPCellID1	RO	0x0000.00F0	GPIO PrimeCell Identification 1
0xFF8	GPIOPCellID2	RO	0x0000.0005	GPIO PrimeCell Identification 2
0xFFC	GPIOPCellID3	RO	0x0000.00B1	GPIO PrimeCell Identification 3

Example 1: GPIO Data (1/2)

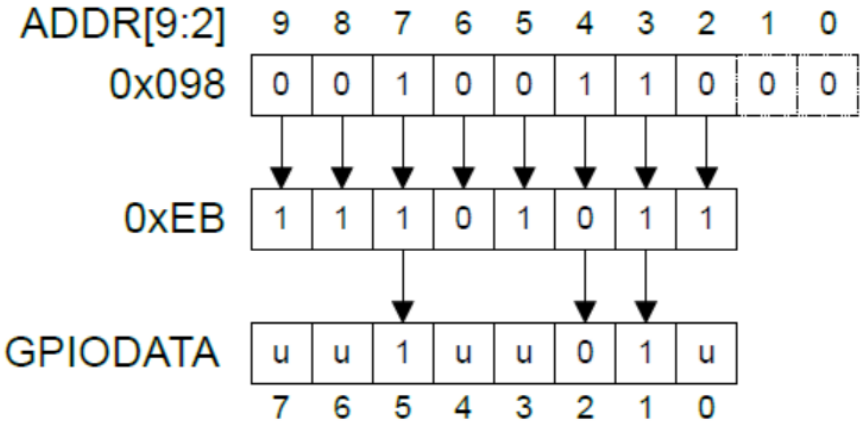
- **GPIODATA** (**offset 0x000**) register carries the to-be-read/to-be-written data (**masked by the address [9:2]*).
  - **This method allows individual GPIO pins to be modified in a single instruction for better software efficiency.*



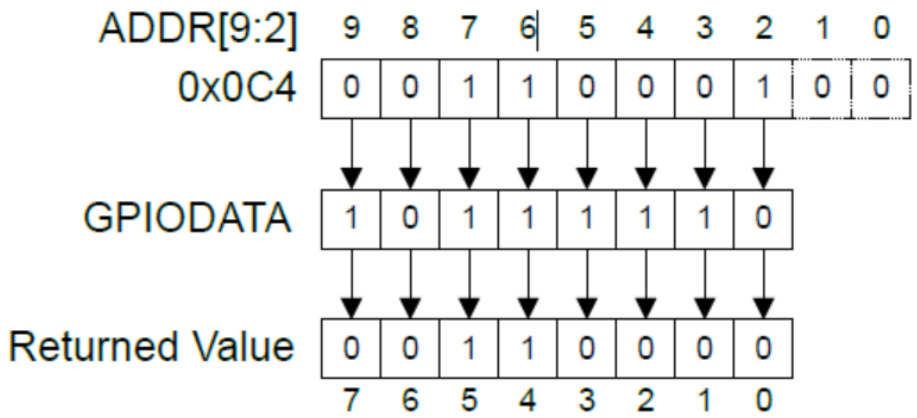
Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0x0000.00	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7:0	DATA	RW	0x00	GPIO Data This register is virtually mapped to 256 locations in the address space. To facilitate the reading and writing of data to these registers by independent drivers, the data read from and written to the registers are masked by the eight address lines [9:2]. Reads from this register return its current state. Writes to this register only affect bits that are not masked by ADDR[9:2] and are configured as outputs. See “Data Register Operation” on page 654 for examples of reads and writes.

Example 1: GPIO Data (2/2)

- **GPIODATA Write**
  - *If the address bit associated with that data bit is set*, the value of the GPIODATA register is altered.
  - Otherwise, the data bit is left unchanged.

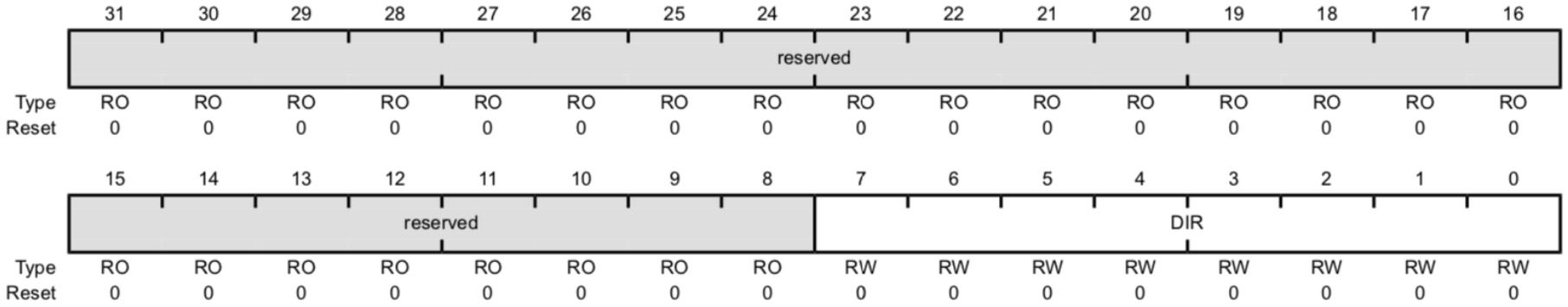


- **GPIODATA Read**
  - *If the address bit associated with that data bit is set*, the value is read.
  - Otherwise, the data bit is read as a zero, regardless of its actual value.



Example 2: GPIO Direction

- **GPIODIR** (**offset 0x400**) register is used to configure each individual pin as an input or output.
  - Setting/clearing a bit in **GPIODIR** configures the corresponding pin (of the GPIO port) to be an output/input.

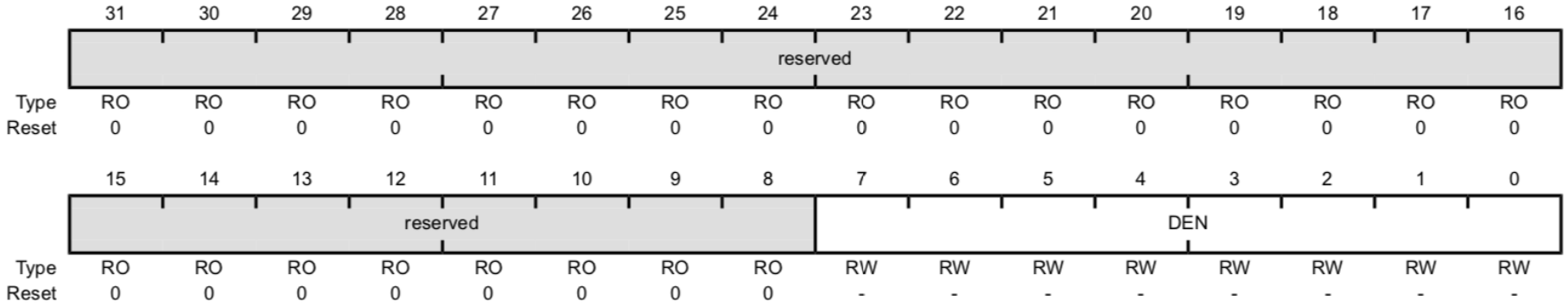


Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0x0000.00	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7:0	DIR	RW	0x00	GPIO Data Direction

Value	Description
0	Corresponding pin is an input.
1	Corresponding pins is an output.

Example 3: GPIO Digital Enable

- **GPIODEN** (**offset 0x51C**) is digital enable register.
  - To use the pin as a digital input/output (either GPIO or alternate function), the corresponding bit must be set.



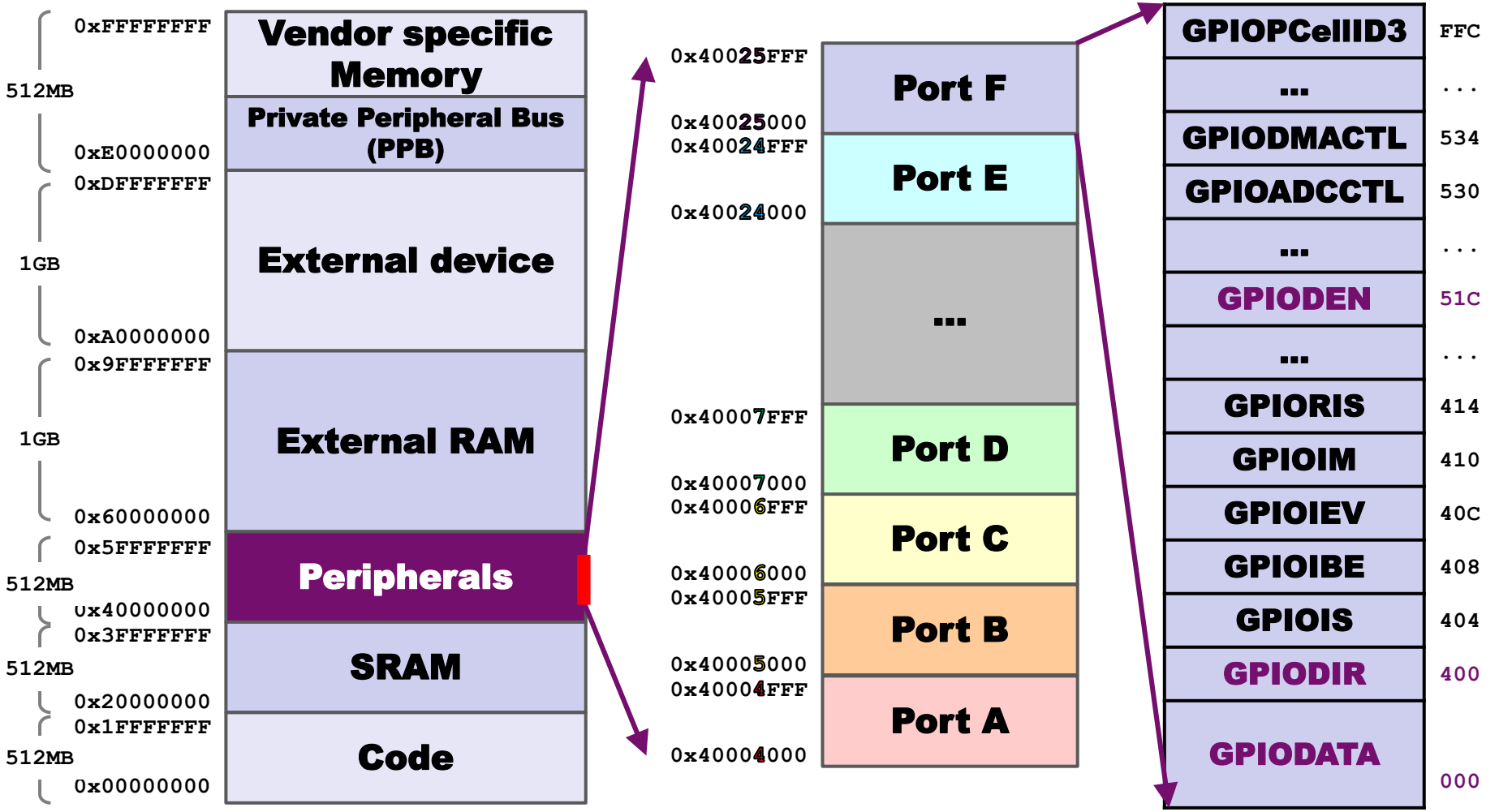
Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0x0000.00	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7:0	DEN	RW	-	Digital Enable

Value	Description
0	The digital functions for the corresponding pin are disabled.
1	The digital functions for the corresponding pin are enabled.

The reset value for this register is 0x0000.0000 for GPIO ports that are not listed in Table 10-1 on page 650.

ARM Cortex-M4 Memory Map

- ARM Cortex-M4 has 4 GB of memory address space.
  - It is architecturally defined as the following regions:

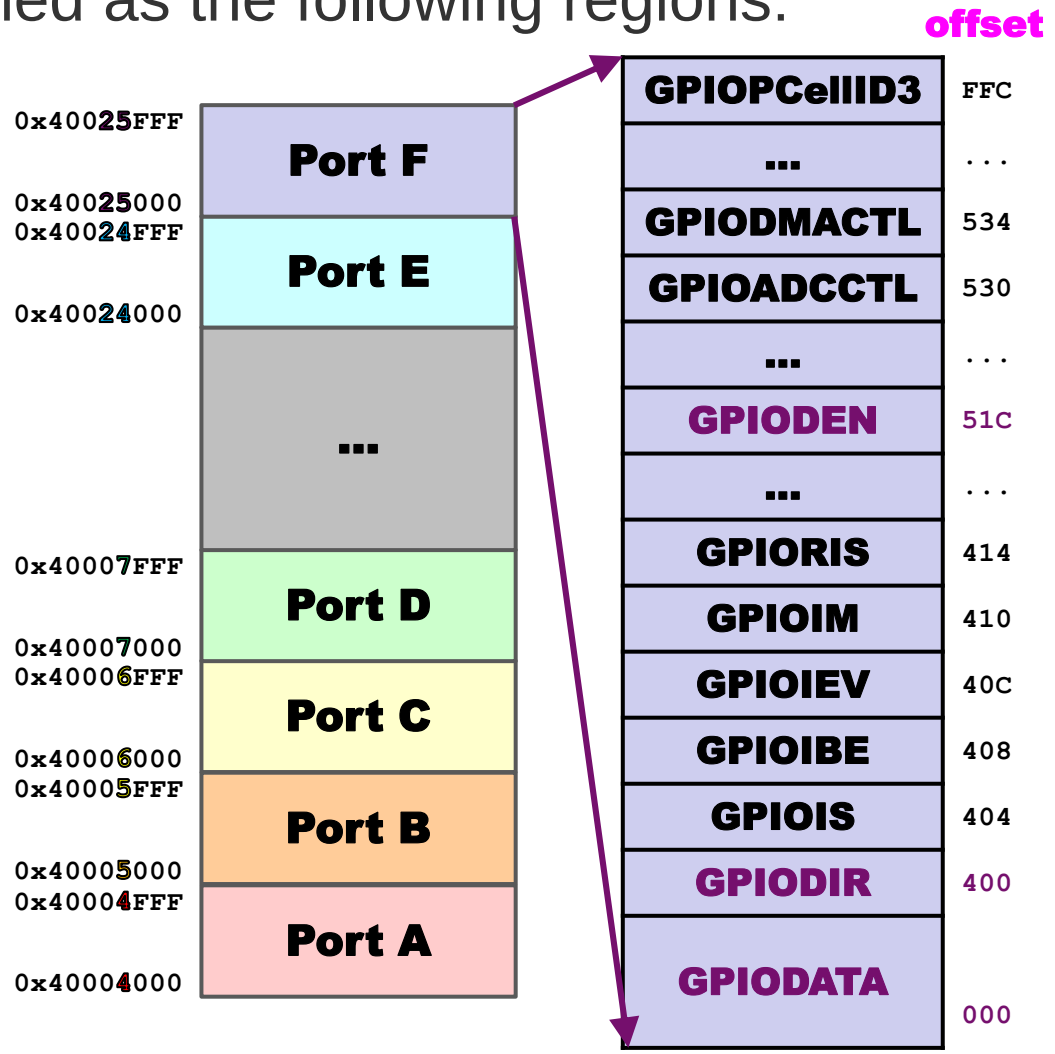


Class Exercise 2.3

- ARM Cortex-M4 has 4 GB of memory address space.
  - It is architecturally defined as the following regions:

- **Question 1**:
  How much memory space is assigned to each GPIO port?

- **Question 2**:
  What is the physical memory address of **GPIODEN** for **Port C**?



Outline

- **GPIO Basics**
  - What's a one? A zero?
  - How does it work?
- **GPIO Module on Tiva™ TM4C123GH6PM**
  - Typical Structure of a GPIO Pin
  - Memory-mapped I/O
  - Control Registers
- **TivaWare™ Peripheral Driver Library**
  - Direct Register Access Model
  - Software Driver Model
  - Programming Example: Toggling LEDs

TivaWare™ Peripheral Driver Library (1/3)

- This library facilitates the peripheral accesses for on the **Tiva™ family of ARM Cortex-M based MCUs**.
  - **Pros**:
    - Written entirely in C except where absolutely not possible;
    - Demonstrate how to use the peripheral in a common way;
    - Easy to understand;
    - Reasonably efficient in terms of memory/processor usage.
    - As self-contained as possible;
    - Where possible, computations that can be performed at compile time are done there instead of at run time;
    - Built with more than one tool chain.
  - **Cons**:
    - Not necessarily as efficient as they could be (for better readability;
    - Do not support the full capabilities of the hardware;
    - Remove all error checking code (for better code size/speed).

TivaWare™ Peripheral Driver Library (2/3)

- Source Code Overview:

`EULA.txt`	The full text of the End User License Agreement that covers the use of this software package.
`driverlib/`	This directory contains the source code for the drivers.
`hw_*.h`	Header files, one per peripheral, that describe all the registers and the bit fields within those registers for each peripheral. These header files are used by the drivers to directly access a peripheral, and can be used by application code to bypass the peripheral driver library API.
`inc/`	This directory holds the part specific header files used for the direct register access programming model.
`makedefs`	A set of definitions used by make files.

TivaWare™ Peripheral Driver Library (3/3)

- This library supports two programming models (each model can be used independently or combined):

① **Direct Register Access Model**

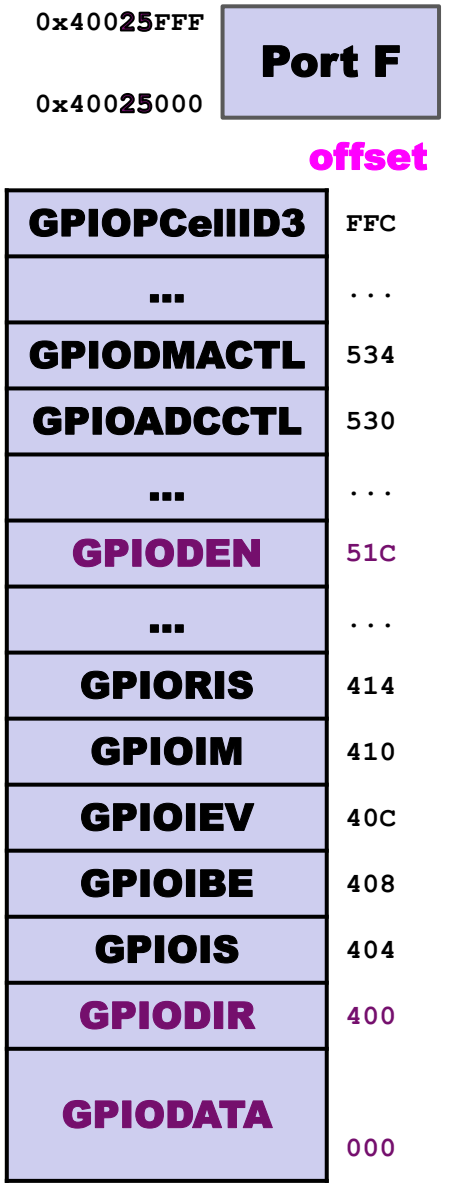
- The peripheral's control registers are manipulated directly.
  - Requiring detailed knowledge but delivering more efficient codes.
- A set of macros is provided that simplifies this process.
  - The header file for TM4C123GH6PM MCU: `"inc/tm4c123gh6pm.h"`.
  - Naming Convention:
    - » Register name macros: `module_name & instance_number`;
    - » Register bit fields: `module_name & register_name & bit_field_name`;
    - » Values that end in `_R` are used to access the value of a register;
    - » Values that end in `_M` represent the mask for a multi-bit field in a register;
    - » Values that end in `_S` represent the number of bits to shift for alignment.

② **Software Driver Model**

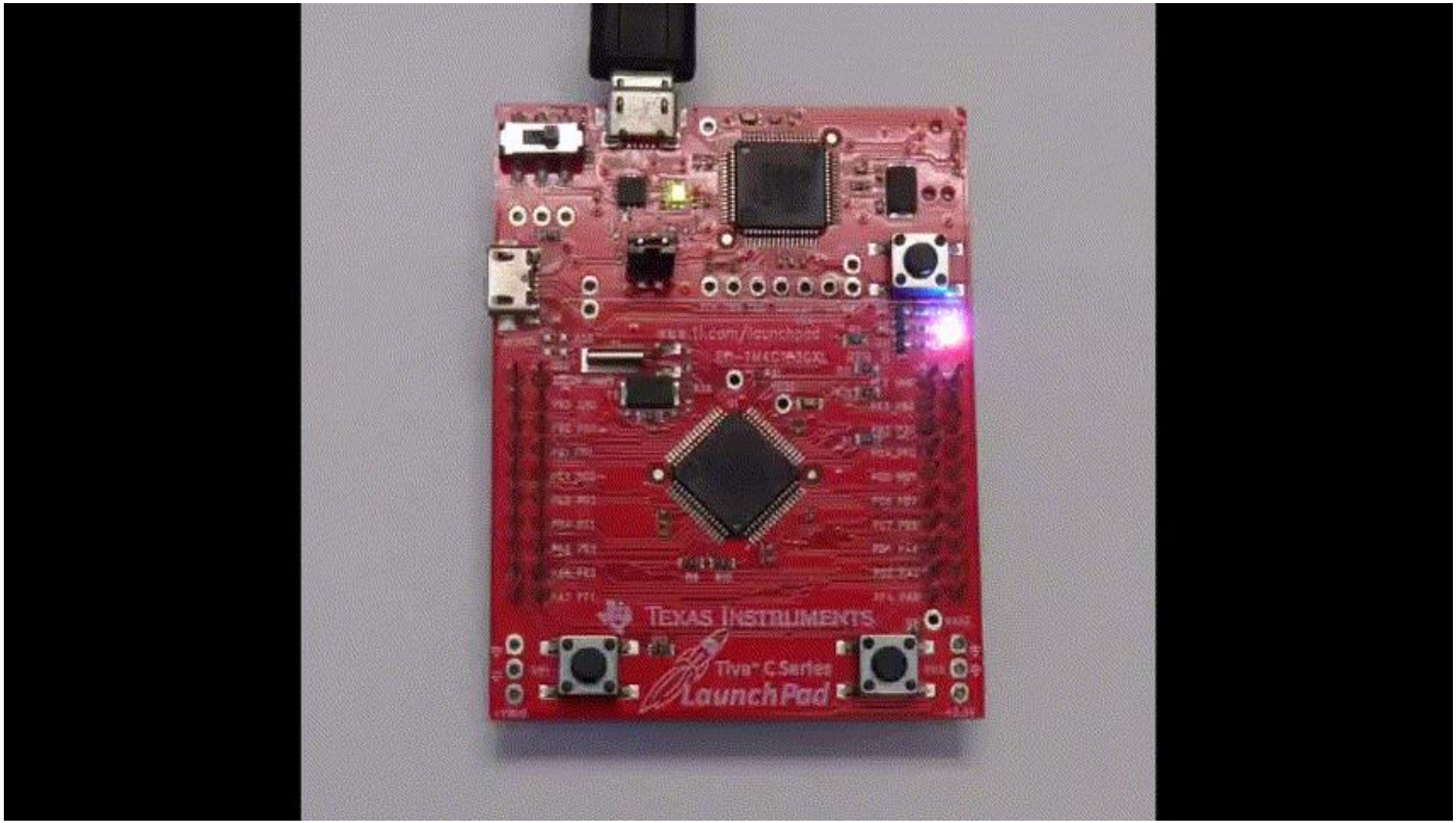
- The API provided by the library is used to control the peripherals.
  - Requiring less time to develop applications.

Header File for TM4C123GH6PM MCU

```
//*****************************************************************************
// The following are defines for the bit fields in the SYSCTL_RCGCGPIO
//*****************************************************************************
#define SYSCTL_RCGCGPIO_R5      0x00000020  // GPIO Port F Run Mode
                                            // Clock Gating Control
...
//*****************************************************************************
// GPIO registers (PORTF)
//*****************************************************************************
#define GPIO_PORTF_DATA_BITS_R  ((volatile unsigned long *)0x40025000)
#define GPIO_PORTF_DATA_R       (*((volatile unsigned long *)0x400253FC))
#define GPIO_PORTF_DIR_R        (*((volatile unsigned long *)0x40025400))
#define GPIO_PORTF_IS_R         (*((volatile unsigned long *)0x40025404))
#define GPIO_PORTF_IBE_R        (*((volatile unsigned long *)0x40025408))
#define GPIO_PORTF_IEV_R        (*((volatile unsigned long *)0x4002540C))
#define GPIO_PORTF_IM_R         (*((volatile unsigned long *)0x40025410))
#define GPIO_PORTF_RIS_R        (*((volatile unsigned long *)0x40025414))
...
#define GPIO_PORTF_DEN_R        (*((volatile unsigned long *)0x4002551C))
...
```



Programming Example: Toggling LEDs (1/2)



Programming Example: Toggling LEDs (2/2)

```c
/*** ① Direct Register Access Model ***/
#include "inc/TM4C123GH6PM.h"
void delayMs (int n);
int main(void) {
  /* enable clock to GPIOF */
  SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R5;
  /* enable the GPIO pins for the LED
  (PF3, 2 1) as output */
  GPIO_PORTF_DIR_R = 0x0E; /* 0000 1110 */
  /* enable the GPIO pins for
  digital function */
  GPIO_PORTF_DEN_R = 0x0E; /* 0000 1110 */

  while(1) {
    GPIO_PORTF_DATA_R = 0x0E; /* all on */
    delayMs(500); /* sleep 500 ms */
    GPIO_PORTF_DATA_R = 0; /* all off */
    delayMs(500); /* sleep 500 ms */
  }
}
```

```c
/*** ② Software Driver Model ***/
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
void delayMs (int n);
int main(void) {
  /* enable clock to GPIOF */
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
  /* enable the GPIO pins for the LED */
  GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,
            GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
  while(1){
    GPIOPinWrite(GPIO_PORTF_BASE,
        GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 14);
    delayMs(500); /* sleep 500 ms */
    GPIOPinWrite(GPIO_PORTF_BASE,
         GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
    delayMs(500); /* sleep 500 ms */
  }
}
```

GPIO Run Mode Clock Gating Control

- **RCGCGPIO** (offset 0x608) is one of system control registers that provides software the capability to enable and disable GPIO modules in Run mode.
  - **Important**: This register should be used to control the clocking for the GPIO modules.

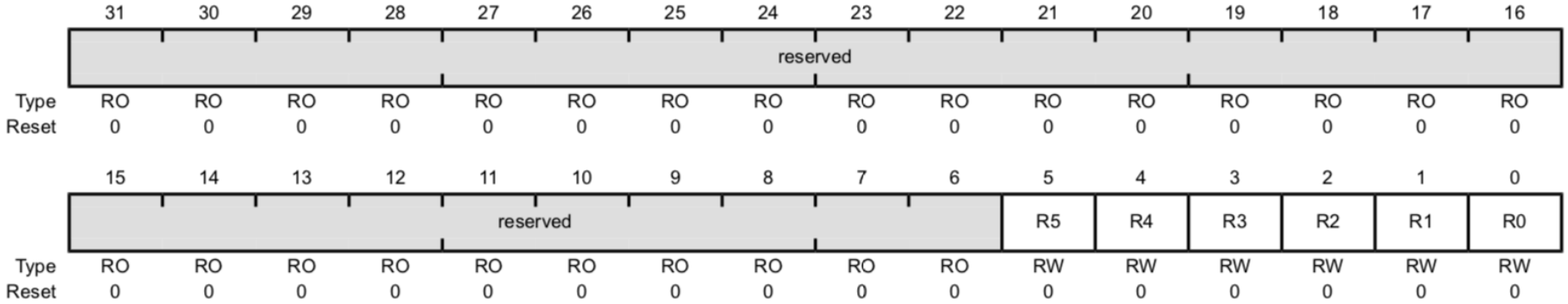
In **Run mode**, the processor is actively executing code.

(Note: Three modes of operation are supported by the Tiva family: Run mode, Sleep mode, and Deep-Sleep mode.)

Clock Gating is a method to disable circuit by blocking clock signal, reducing power consumption.

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

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



Bit/Field	Name	Type	Reset	Description
31:6	reserved	RO	0	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.
5	R5	RW	0	GPIO Port F Run Mode Clock Gating Control Value Description 0 GPIO Port F is disabled. 1 Enable and provide a clock to GPIO Port F in Run mode.
4	R4	RW	0	GPIO Port E Run Mode Clock Gating Control Value Description 0 GPIO Port E is disabled. 1 Enable and provide a clock to GPIO Port E in Run mode.

(more)

Selected API Functions

- `SysCtlPeripheralEnable(uint32_t ui32Peripheral)`
  - Enables a peripheral.
  - **Parameters**:
    - **`ui32Peripheral`** is the peripheral to enable.

- `GPIOPinTypeGPIOOutput(uint32_t ui32Port, uint8_t ui8Pins)`
  - Configures pin(s) for use as GPIO outputs.
  - **Parameters**:
    - **`ui32Port`** is the base address of the GPIO port.
    - **`ui8Pins`** is the bit-packed representation of the pin(s).

- `GPIOPinWrite(uint32_t ui32Port, uint8_t ui8Pins, uint8_t ui8Val)`
  - Writes a value to the specified pin(s).
  - **Parameters**:
    - **`ui32Port`** is the base address of the GPIO port.
    - **`ui8Pins`** is the bit-packed representation of the pin(s).
    - **`ui8Val`** is the value to write to the pin(s).

Class Exercise 2.4

- The given program lights up all the LEDs (i.e., Red, Blue, and Green). How should the program be modified if we only want to light up the Red LED?

```
/*** ② Software Driver Model ***/
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
void delayMs (int n);
int main(void) {
  /* enable clock to GPIOF */
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
  /* enable the GPIO pins for the LED */
  GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,
            GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
  while(1){
    GPIOPinWrite(GPIO_PORTF_BASE,
        GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 14);
    delayMs(500); /* sleep 500 ms */
    GPIOPinWrite(GPIO_PORTF_BASE,
         GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
    delayMs(500); /* sleep 500 ms */
  }
}
```

Summary

- **GPIO Basics**
  - What's a one? A zero?
  - How does it work?
- **GPIO Module on Tiva™ TM4C123GH6PM**
  - Typical Structure of a GPIO Pin
  - Memory-mapped I/O
  - Control Registers
- **TivaWare™ Peripheral Driver Library**
  - Direct Register Access Model
  - Software Driver Model
  - Programming Example: Toggling LEDs

Important References

- Tiva C Series TM4C123G LaunchPad Evaluation Kit User's Manual
- Tiva™ C Series TM4C123GH6PM Microcontroller Data Sheet datasheet (Rev. E)
- TivaWare™ Peripheral Driver Library for C Series User's Guide (Rev. E)