BIRZEIT UNIVERSITY

Department of Electrical & Computer Engineering
ENCS4110 – Computer Design Laboratory

Experiment 04

General-Purpose Input/Output and Peripheral Interfacing

Date: October 2025

4 TM4C123 Microcontroller: Architecture and Digital Outputs

Learning Objectives

After completing this experiment, you will be able to:

- Identify the main components and subsystems of the TM4C123 microcontroller and understand their roles within an embedded system.
- Recognize the organization of the ARM Cortex-M4 core, memory map, and peripheral buses.
- Understand the role of core peripherals including NVIC, SysTick, and System Control Block (SCB).
- Configure and control digital output pins using the General-Purpose Input/Output (GPIO) module.
- Write Assembly programs to drive on-board LEDs through direct register access.
- Relate register operations to physical hardware behavior and verify functionality through observation on the LaunchPad board.

Experiment Overview

This experiment introduces the architecture of the TM4C123 microcontroller and its most fundamental feature — the ability to interface with the outside world through digital I/O pins.
You will first explore the internal structure of the device, including the ARM Cortex-M4 core, core peripherals (NVIC, SysTick, SCB), buses, memory regions, and peripheral interconnect. Then, you will apply this understanding by writing Assembly programs that toggle the on-board LEDs connected to `PORTF`.
In this experiment, you will:

- Understand how peripheral registers are mapped into memory and accessed by the CPU.
- Learn about Cortex-M4 core peripherals and their functions.
- Learn the initialization sequence required to enable and configure GPIO ports.
- Develop Assembly programs that control LEDs using software delays.
- Form the foundation for subsequent experiments involving interrupts, timers, and analog peripherals.

By the end of this lab, you will understand the TM4C123 architecture, Cortex-M4 core peripherals, be able to configure GPIO ports for digital output, and control external hardware through register manipulation in Assembly.

Contents

1 Theoretical Background

1.1 TM4C123 Microcontroller Architecture

The TM4C123GH6PM microcontroller is built around the ARM Cortex-M4F processor core, implementing the ARMv7-M architecture. It integrates the CPU core, memory, peripherals, and I/O interfaces on a single chip — a complete system-on-chip (SoC) solution for embedded applications.

1.1.1 Core Components

The TM4C123 contains the following major components:

- **ARM Cortex-M4F Processor Core**: 32-bit RISC processor with hardware Floating-Point Unit (FPU)
- **Memory**: 256 KB Flash, 32 KB SRAM, 2 KB EEPROM
- **System Buses**: Advanced High-Performance Bus (AHB) and Advanced Peripheral Bus (APB)
- **Core Peripherals**: NVIC, SysTick Timer, Memory Protection Unit (MPU), System Control Block (SCB)
- **General-Purpose Timers**: Six 16/32-bit and six 32/64-bit timers
- **Communication Interfaces**: UART, I^2C, SSI (SPI), CAN
- **Analog Peripherals**: 12-bit ADC with 12 channels, analog comparators
- **GPIO Ports**: Six ports (A-F) with up to 43 programmable pins

1.2 ARM Cortex-M4 Core Peripherals

The ARM Cortex-M4 processor includes several **core peripherals** that are common across all Cortex-M4-based microcontrollers. They are tightly integrated with the CPU and provide essential system functions such as interrupt handling and timing.

1.2.1 Nested Vectored Interrupt Controller (NVIC)

The NVIC manages all interrupt and exception handling. It supports up to 240 interrupt sources (138 on the TM4C123), hardware priority levels, and automatic context saving for efficient servicing. Key features include nesting, tail-chaining, and late-arrival handling for minimal interrupt latency.
Important NVIC registers (base address `0xE000E100`):

- `NVIC_ENx` — Set-Enable
- `NVIC_DISx` — Clear-Enable
- `NVIC_PRIx` — Priority Configuration
- `NVIC_ACTIVEx` — Active Status

1.2.2 SysTick Timer

The SysTick timer is a simple 24-bit down-counter integrated in the core. It provides a consistent time base for system delays or periodic tasks. Many operating systems use it for time-keeping or scheduling.
SysTick registers (base address `0xE000E010`):

- `STCTRL` — Control and Status
- `STRELOAD` — Reload Value
- `STCURRENT` — Current Counter Value

Other core peripherals such as the System Control Block (SCB) are present but not covered in this course.

1.3 Memory Map

The Cortex-M4 processor uses a unified 4 GB address space for all code, data, and peripherals. Every peripheral and memory region occupies a unique address range, allowing direct access through normal load and store instructions.

Region	Address Range	Description
Flash Memory	0x00000000 - 0x0003FFFF	Program storage (256 KB)
SRAM	0x20000000 - 0x20007FFF	On-chip data memory (32 KB)
Peripherals	0x40000000 - 0x400FFFFF	Peripheral registers
GPIO Ports	0x40004000 - 0x40025FFF	GPIO A-F registers
Core Peripherals	0xE0000000 - 0xE00FFFFF	NVIC, SysTick, SCB, etc.

Table 4.1: Simplified TM4C123 Memory Map

Each peripheral's registers are **memory-mapped**, meaning they are accessed just like variables in memory. For example, writing to address `0x400253FC` directly updates the GPIO Port F data register.

1.4 General-Purpose Input/Output (GPIO)

GPIO (General-Purpose Input/Output) ports form the primary interface between the microcontroller and external devices such as LEDs, switches, and sensors. Each pin can be configured as either an input or an output.

When a pin is an **output**, software drives it high or low to control external hardware (e.g., turn an LED on/off). When a pin is an **input**, software reads its logic level (e.g., a button press).

1.4.1 GPIO Ports Overview

The TM4C123GH6PM provides six GPIO ports (A-F), each with up to eight programmable pins. Not all pins are available on the LaunchPad, and some are reserved for debugging or special functions.

Port	Pins Available	Notes
Port A	PA0-PA7	UART0 (PA0, PA1) shared with USB debug interface
Port B	PB0-PB7	I^2C0, SSI2, ADC channels
Port C	PC0-PC7	PC0-PC3 used for JTAG (avoid modification)
Port D	PD0-PD7	PD7 requires unlock for GPIO use
Port E	PE0-PE5	ADC and UART5 functionality available
Port F	PF0-PF4	On-board LEDs (PF1-PF3) and switches (PF0, PF4)

Table 4.2: GPIO Port Summary — TM4C123GH6PM

Each port exposes its own control and data registers, allowing independent configuration and operation.

1.4.2 Memory-Mapped GPIO Registers

GPIO modules are accessed via **memory-mapped registers**: fixed addresses that the CPU reads/writes with standard `LDR`/`STR` instructions.

Every port has a **base address**; key registers sit at fixed offsets from that base.

Port	Base Address
Port A	0x40004000
Port B	0x40005000
Port C	0x40006000
Port D	0x40007000
Port E	0x40024000
Port F	0x40025000

Table 4.3: GPIO Port Base Addresses (TM4C123GH6PM)

In this experiment we use Port F, so:

`GPIODIR` = `0x40025400`, `GPIODEN` = `0x4002551C`, `GPIODATA` = `0x400253FC`.

1.4.3 Address Masking in `GPIODATA`

The `GPIODATA` register supports **address masking**: address bits [9:2] form a bit mask that selects which pins are affected.

- **Full access:** Base + `0x3FC` — all 8 pins.
- **Masked access:** Base + (`mask` $\ll$ 2) — only the pins in `mask`.

Example (PF1 only):

`0x40025000 + (0x02 << 2) = 0x40025008`.

This is **address masking** (often confused with "bit-banding"); it enables fast, selective reads/writes to individual pins.

1.4.4 Protected and Special-Function Pins

Some pins are locked or reserved at reset due to critical roles:

- **PF0** doubles as the Non-Maskable Interrupt (NMI) input and is locked. To use it as GPIO, write `0x4C4F434B` to `GPIO_LOCK` and set the desired bits in `GPIO_CR`.
- **PC0-PC3** carry the JTAG debug interface and should not be reassigned in normal operation.

Always confirm pin availability in the datasheet before repurposing special-function pins.

1.4.5 Port F on the LaunchPad

Port F connects to the on-board RGB LEDs and user push buttons:

Pin	Function	Connected Component	Active Level
PF1	Output	Red LED	Active High
PF2	Output	Blue LED	Active High
PF3	Output	Green LED	Active High
PF0	Input (locked)	SW2 (Push Button)	Active Low (enable pull-up)
PF4	Input	SW1 (Push Button)	Active Low (enable pull-up)

Table 4.4: GPIO Port F Pin Assignments — TM4C123 LaunchPad

LEDs are **active-high** (write 1 to turn on). Push buttons pull to ground when pressed, so inputs read **0 when pressed** and require internal pull-ups.

1.4.6 GPIO Configuration Workflow

In practice, GPIO usage has two phases:

- **Initialization (setup, runs once):** enable the port clock, unlock protected pins (if needed), set direction, and enable digital function.
- **Runtime (operation, repeats):** read inputs or write outputs by accessing `GPIODATA` in the main loop or in interrupt service routines (ISRs).

A typical structure is:

```
InitGPIO();                // RCGCGPIO, optional LOCK/CR, DIR, DEN
while (1) {                // Runtime phase
    // Read buttons (inputs) and/or drive LEDs (outputs)
    // Optionally, some I/O is handled inside ISRs
}
```

Listing 4.1: Typical GPIO program structure (setup vs. runtime)

The detailed register-level steps (clock enable, unlock, direction, digital enable, and data access) are demonstrated in the following subsections.

Step 1 — Enable the GPIO Port Clock

Register: `RCGCGPIO` — Run Mode Clock Gating Control for GPIO (`0x400FE608`)
Before accessing any GPIO port, its clock must be enabled. The `RCGCGPIO` register controls the clock to all GPIO modules. Each bit corresponds to a single port (A-F). Writing a '1' to a bit enables the clock to that port, while '0' disables it.

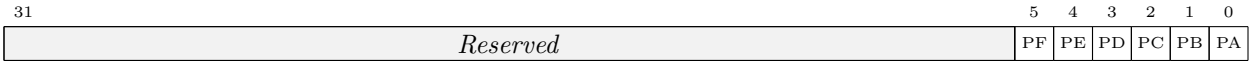

Figure 4.1: RCGCGPIO Register (`0x400FE608`) — GPIO Clock Control

To activate Port F, bit 5 must be set to 1 as shown below.

```
LDR     R1, =0x400FE608     ; RCGCGPIO register
LDR     R0, [R1]
ORR     R0, R0, #0x20       ; Set bit 5 (Port F)
STR     R0, [R1]
```

Listing 4.2: Enable clock for Port F

Once the clock is enabled, a short delay or status polling ensures the peripheral is ready before further configuration.

Step 2 — Unlock Protected Pins

Registers: `GPIO_LOCK` (`0x40025520`), `GPIO_CR` (`0x40025524`)
Certain pins such as PF0 are protected because they share critical alternate functions (e.g., the NMI input). To modify these pins, the port must be unlocked by writing the key value `0x4C4F434B` ("LOCK") into the `GPIO_LOCK` register. The `GPIO_CR` register (Commit Register) then determines which pins can be altered.

Figure 4.2: GPIO_CR Register — Commit Control

The following code unlocks Port F and enables modification of all pins.

```
LDR     R1, =0x40025520     ; GPIO_LOCK
LDR     R0, =0x4C4F434B     ; Unlock key
STR     R0, [R1]
LDR     R1, =0x40025524     ; GPIO_CR
MOV     R0, #0xFF           ; Allow changes to all pins
STR     R0, [R1]
```

Listing 4.3: Unlock Port F

In this experiment, PF0 is not used, so unlocking is optional. After unlocking, configuration registers such as direction and digital enable can be safely modified.

Step 3 — Configure Pin Direction

Register: `GPIODIR` (`0x40025400`) — GPIO Direction Control (Port F)
The `GPIODIR` register determines whether each GPIO pin functions as an input or an output. Writing '0' configures the pin as an input, while '1' configures it as an output.

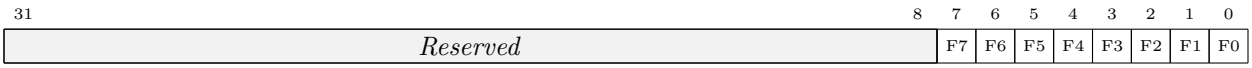

Figure 4.3: GPIODIR Register (Port F) — Direction Control

For the on-board RGB LED, PF1-PF3 must be configured as outputs. Bits 1-3 are therefore set to '1'.

```
LDR     R1, =0x40025400     ; GPIODIR register (Port F)
LDR     R0, [R1]
ORR     R0, R0, #0x0E       ; Set PF1, PF2, PF3 as outputs
STR     R0, [R1]
```

Listing 4.4: Set PF1, PF2, PF3 as outputs

Unmodified bits remain unchanged, allowing input pins (such as PF0 or PF4) to retain their default configuration.

Step 4 — Enable Digital Functionality

Register: `GPIODEN` (`0x4002551C`) — Digital Enable Register (Port F)
Each GPIO pin can serve analog or digital functions. The `GPIODEN` register enables the digital circuitry for selected pins. Pins configured as digital inputs or outputs must have their corresponding bits set to '1'.

Figure 4.4: GPIODEN Register (Port F) — Digital Enable

Bits 1-3 are set to enable PF1-PF3 as digital outputs for the LED.

```
LDR     R1, =0x4002551C     ; GPIODEN register (Port F)
LDR     R0, [R1]
ORR     R0, R0, #0x0E       ; Enable PF1-PF3 as digital pins
STR     R0, [R1]
```

Listing 4.5: Enable digital function for PF1-PF3

Failing to enable `GPIODEN` leaves the pins electrically inactive even if their direction is set.

Step 5 — Write to the Data Register

Register: `GPIODATA` (`0x400253FC`) — Data Input/Output Register (Port F)
The `GPIODATA` register reflects the current logic levels on all GPIO pins. Writing to a bit drives the corresponding output high ('1') or low ('0'). Bits 1-3 correspond to the RGB LED pins on the LaunchPad: PF1 = Red, PF2 = Blue, PF3 = Green.

Figure 4.5: GPIODATA Register (Port F) — Data Output/Input

The example below drives PF1 (Red) and PF3 (Green) simultaneously to produce a yellow color.

```
LDR     R1, =0x400253FC     ; GPIODATA register (Port F)
MOV     R0, #0x0A           ; Set PF1 (Red) and PF3 (Green) = Yellow
STR     R0, [R1]
```

Listing 4.6: Turn on Yellow LED (Red + Green = PF1 + PF3)

By writing different bit combinations, various LED colors can be generated:

Color	**Red (PF1)**	**Blue (PF2)**	**Green (PF3)**	**Hex**	**Combination**
Red	●	○	○	0x02	Red only
Blue	○	●	○	0x04	Blue only
Green	○	○	●	0x08	Green only
Yellow	●	○	●	0x0A	Red + Green
Cyan	○	●	●	0x0C	Blue + Green
Magenta	●	●	○	0x06	Red + Blue
White	●	●	●	0x0E	All on

Table 4.5: LED Color Combinations on TM4C123 LaunchPad (PF1-PF3)

2 Procedure

2.1 Examples

The following examples illustrate the complete process of initializing Port F and controlling the on-board LEDs using Assembly language.

2.1.1 Example 1 — Simple LED Blink

This example initializes Port F and toggles the Red LED (PF1) on and off using address masking with a software delay. Address Masking for PF1:

```
0x40025000 + (0x02 « 2) = 0x40025008
```

```
        AREA    RESET, CODE, READONLY
        THUMB
        EXPORT  __main

__main  PROC
        LDR     R1, =0x400FE608         ; RCGCGPIO
        LDR     R0, [R1]
        ORR     R0, R0, #0x20           ; Enable Port F
        STR     R0, [R1]
        NOP
        NOP

        LDR     R1, =0x40025400         ; GPIODIR
        LDR     R0, [R1]
        ORR     R0, R0, #0x02           ; PF1 output (R-M-W)
        STR     R0, [R1]

        LDR     R1, =0x4002551C         ; GPIODEN
        LDR     R0, [R1]
        ORR     R0, R0, #0x02           ; Enable PF1 digital
        STR     R0, [R1]

loop    LDR     R1, =0x40025008         ; DATA masked address for PF1
        MOV     R0, #0x02               ; LED on (PF1)
        STR     R0, [R1]

        LDR     R0, =1000000
        BL      DELAY

        MOV     R0, #0x00               ; LED off (PF1)
        STR     R0, [R1]

        LDR     R0, =1000000
        BL      DELAY

        B       loop
        ENDP

DELAY   PROC                            ; R0 = count
        SUBS    R0, R0, #1
        BNE     DELAY
        BX      LR
```

```
        ENDP
        END
```

Listing 4.7: Assembly code to blink the Red LED (PF1) on the TM4C123 LaunchPad

2.1.2 Example 2 — Cycle Through RGB Colors

This example extends the previous one by cycling through RGB LED colors (Red, Green, Blue) using a software delay. Address Masking for PF1, PF2, PF3:

```
0x40025000 + ((0x02 | 0x04 | 0x08) « 2) = 0x40025038
```

```
        AREA    RESET, CODE, READONLY    ; Code section
        THUMB                            ; Use Thumb instruction set
        EXPORT  __main                   ; Export symbol
__main  PROC
        BL      PF_Init                  ; Initialize Port F
loop
        LDR     R1, =0x40025038          ; Masked DATA address for PF1-PF3

        MOV     R0, #0x02                ; Turn ON red LED (PF1)
        STR     R0, [R1]                 ; Write to GPIODATA
        LDR     R0, =10000000            ; Load delay count
        BL      DELAY                    ; Call delay

        MOV     R0, #0x04                ; Turn ON blue LED (PF2)
        STR     R0, [R1]
        LDR     R0, =10000000
        BL      DELAY

        MOV     R0, #0x08                ; Turn ON green LED (PF3)
        STR     R0, [R1]
        LDR     R0, =10000000
        BL      DELAY
        B       loop                     ; Repeat
        ENDP

DELAY   PROC
        SUBS    R0, R0, #1               ; Decrement counter
        BNE     DELAY                    ; Loop until zero
        BX      LR                       ; Return
        ENDP

PF_Init PROC
        LDR     R1, =0x400FE608          ; RCGCGPIO address
        LDR     R0, [R1]                 ; Read current value
        ORR     R0, R0, #0x20            ; Enable Port F clock
        STR     R0, [R1]
        NOP                              ; Small delay
        NOP
        LDR     R1, =0x40025400          ; GPIODIR address
        LDR     R0, [R1]
        ORR     R0, R0, #0x0E            ; Set PF1-PF3 as outputs
        STR     R0, [R1]

        LDR     R1, =0x4002551C          ; GPIODEN address
        LDR     R0, [R1]
        ORR     R0, R0, #0x0E            ; Enable digital function for PF1-PF3
        STR     R0, [R1]
        BX      LR                       ; Return
        ENDP
        END
```

Listing 4.8: Assembly code to cycle through RGB colors on the TM4C123 LaunchPad

2.2 Tasks

2.2.1 Task 1 — Adjust the Blink Rate

Modify the delay routine in **Example 1** to change the blinking speed of the Red LED. Experiment with different delay values until the LED blinks at approximately **1 Hz** (about one second ON and one second OFF).

2.2.2 Task 2 — Cycle Through Multiple Colors

Expand **Example 2** to include additional colors by combining the Red, Green, and Blue LEDs. Create a program that automatically cycles through all color combinations listed in Table 4.5.

Hint: Use a loop to step through the color sequence repeatedly instead of writing separate code for each color.