Lab 13. Assembly Programming in ESP32-C3 – using Wokwi Simulation

Objective

In this practical session, students will become familiar with the ESP32-C3 kit, which is based on the RISC-V architecture. Students will use assembly language to program simple applications to control the input/output ports of the kit and run simulations using the Wokwi emulator.

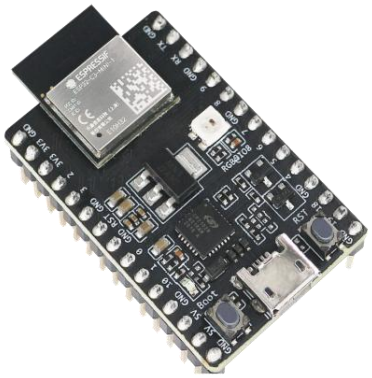
Reference

- ESP32-C3 Technical Reference Manual

Preparation

Introduction to MCU ESP32-C3

The ESP32-C3 is a System on Chip (SoC) developed by Espressif, featuring a microcontroller based on the open-source RISC-V architecture. It achieves a balanced combination of performance, peripheral connectivity, and security, offering a cost-effective solution for connected devices. The 32-bit RISC-V microcontroller can operate at a maximum clock speed of 160 MHz. With 22 configurable General Purpose Input/Output (GPIO) pins, 400 KB of RAM, and low-power mode support, it is suitable for various applications involving connected devices.



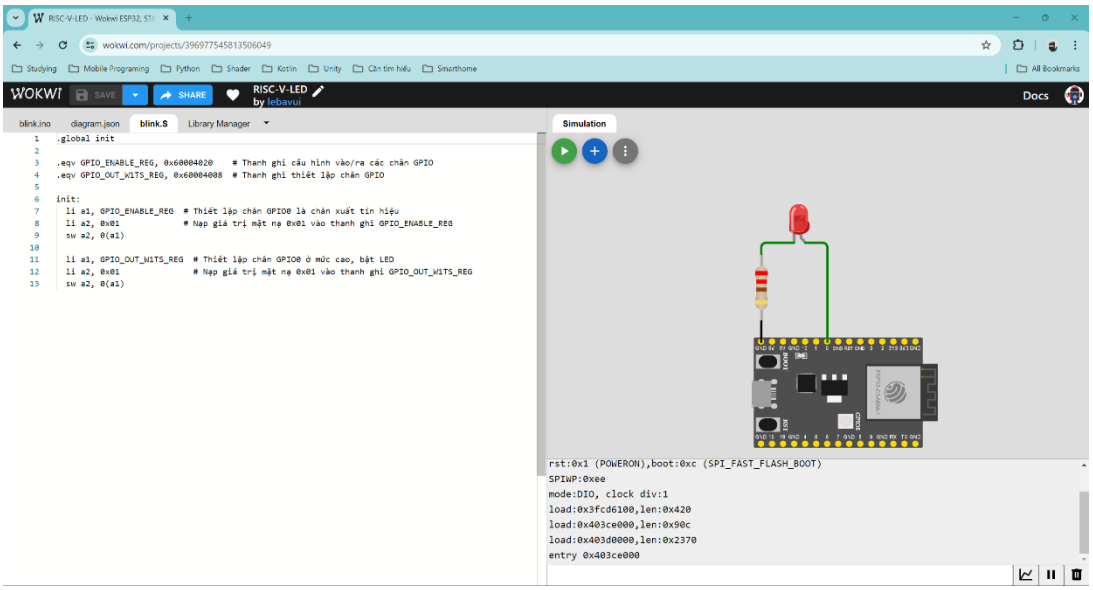
Electronics manufacturers often produce development boards (also known as kits). In addition to the primary microcontroller module, these kits integrate electronic circuits to enhance usability, such as USB programming interfaces, voltage control circuits, status LEDs, and more. The image on the right shows the ESP32-C3-DevKitM-1 board developed by Espressif. Other versions by different manufacturers are also available.

When working with these development boards, students need to refer to the datasheet to understand how to use the board, including pin layouts, operating modes, and other specifications.

Simulating the ESP32-C3 Kit with Wokwi

Wokwi is a web-based application that allows the simulation of popular embedded system development boards such as Arduino, STM32, and ESP32. The platform provides an

interface where developers can write code and test it on simulated embedded systems. Additionally, Wokwi offers basic electronic components, which act as peripherals connected to the embedded system, including LEDs, 7-segment displays, LCD screens, buttons, switches, resistors, and more. These features make it possible to simulate and test simple embedded applications effectively..



Editor and Simulator of Wokwi

Wokwi Assembly Programming on ESP32-C3 Using Wokwi

The steps to create a project, write source code, and simulate using Wokwi are as follows:

1. Create a New Project Using the ESP32-C3
    a. Go to the Wokwi homepage: https://wokwi.com
    b. Select **ESP32**
    c. Select **ESP32-C3**
2. In source code editor, rename ino file if nessecessary (click to right arrow **Library Manager**, select **Rename**). Change the **ino** file as follows:

```
void setup() {}

void loop() {}
```

3. Add assembly file (extension .S), click to the right arrow of **Library Manager**, select **New file …**, the file name match **ino** file name but with the extension **.s** (e.g., if the **ino** file is named **sketch.ino**, the assembly file should be named **sketch.S**)
4. The assembly program should follow the structure below:

```
# Define the init function for Wokwi to execute the assembly program
.global init

# (Optional) Use the .eqv directive to define constants
.eqv CONST1, 0x0001
```

```
# (Optional) Use the .data directive to declare data in memory
.data

# The .text directive marks the beginning of the instruction section
# If there is no .data section, the .text directive is not necessary
.text
init:
  # The assembly program starts here!!!
```

5. Press the + button to add the necessary electronic components. Connect the component pins to the appropriate pins of the ESP32-C3 board according to the schematic diagram
6. Press the Start button to compile the code and run the simulation.

ESP32-C3 General Purpose Input/Output (GPIO) Overview

The ESP32-C3 includes 22 General Purpose Input/Output (GPIO) pins, which can be configured for output or input functions. Each GPIO pin can also perform other functions (e.g., peripheral communication via the I2C protocol). The ESP32-C3 provides registers for configuring GPIO functionality, typically set during initialization.

Key GPIO Registers

- **GPIO_ENABLE_REG** (GPIO output enable register)
  - Address: 0x60004020
  - Enables output functionality for GPIO pins.
  - Bits 0 to 21 correspond to GPIO0 through GPIO21.
  - A bit value of 1 configures the corresponding GPIO pin as an output.
- **GPIO_ENABLE_W1TS_REG** (GPIO output enable set register)
  - Address: 0x60004024
  - Supports setting bits in the **GPIO_ENABLE_REG** register.
  - A bit value of 1 sets the corresponding bit in **GPIO_ENABLE_REG**, leaving other bits unchanged.
- **GPIO_ENABLE_W1TC_REG** (GPIO output enable clear register)
  - Address: 0x60004028
  - Supports clearing bits in the **GPIO_ENABLE_REG** register.
  - A bit value of 1 clears the corresponding bit in **GPIO_ENABLE_REG**, leaving other bits unchanged.
- **GPIO_IN_REG** (GPIO input register)
  - Address: 0x6000403C
  - Reads the state of GPIO pins configured as input.
  - Bits 0 to 21 correspond to GPIO0 through GPIO21.
  - A bit value of 1/0 indicates a high/low logic level on the corresponding GPIO pin.
- **GPIO_OUT_REG** (GPIO output register)
  - Address: 0x60004004
  - Configures output values for GPIO pins.

    - Bits 0 to 21 correspond to GPIO0 through GPIO21.
    - A bit value of 1/0 sets the corresponding GPIO pin to a high/low logic level.
- **GPIO_OUT_W1TS_REG** (GPIO output set register)
    - Address: 0x60004008
    - Supports setting bits in the **GPIO_OUT_REG** register.
    - A bit value of 1 sets the corresponding bit in **GPIO_OUT_REG**, leaving other bits unchanged.
- **GPIO_OUT_W1TC_REG** (GPIO output clear register)
    - Address: 0x6000400C
    - Supports clearing bits in the **GPIO_OUT_REG** register.
    - A bit value of 1 clears the corresponding bit in **GPIO_OUT_REG**, leaving other bits unchanged.
- **IO_MUX_GPIOn_REG** (where n is 0 to 21)
    - Address: 0x60009004 + 4*n
    - Configures the functionality of GPIO pins (GPIO0 to GPIO21).
    - These registers are used to select functions and configure GPIO pin operations.

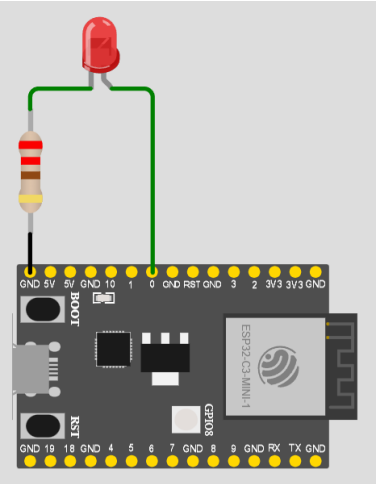
Notes

- Use **GPIO_ENABLE_W1TS_REG** and **GPIO_ENABLE_W1TC_REG** to modify specific bits in the **GPIO_ENABLE_REG**, avoiding unintended changes to unrelated bits.
- Use **GPIO_OUT_W1TS_REG** and **GPIO_OUT_W1TC_REG** to modify specific bits in the **GPIO_OUT_REG**, avoiding unintended changes to unrelated bits.
- Refer to the datasheet for detailed information about GPIO pin functions and select appropriate configurations.

Students are advised to consult the documentation or datasheet for a deeper understanding of these registers.

Home Assignment 1 – Turn LED On/Off

The following example demonstrates how to control an LED to turn it on and off.



Circuit Diagram

The circuit consists of an LED with:

- **Anode** (positive terminal) connected to GPIO0.
- **Cathode** (negative terminal) connected to a 220 Ohm resistor, which is connected to GND.
- When GPIO0 is at logic level 0, the LED is off.
- When GPIO0 is at logic level 1, the LED lights up.

Use **Wokwi** to build the circuit as illustrated in the provided diagram.

```
.global init

.eqv GPIO_ENABLE_REG, 0x60004020    # Register to configure GPIO pins as
input/output
.eqv GPIO_OUT_W1TS_REG, 0x60004008  # Register to set GPIO pins

init:
  li a1, GPIO_ENABLE_REG  # Load register address to setup GPIO0
  li a2, 0x01             # Load the mask 0x01 for register GPIO_ENABLE_REG
  sw a2, 0(a1)

  li a1, GPIO_OUT_W1TS_REG  # Load register to output GPIO0
  li a2, 0x01               # Load the mask 0x01 for register GPIO_OUT_W1TS_REG
  sw a2, 0(a1)
```

Load the program into Wokwi, run and observe the result

Home Assignment 2 – Blink LED

This example demonstrates how to control an LED to blink. The circuit diagram is the same as in Home Assignment 1. To create a blinking effect, the program alternates between turning the LED on and off, with a delay in between.

Source code:

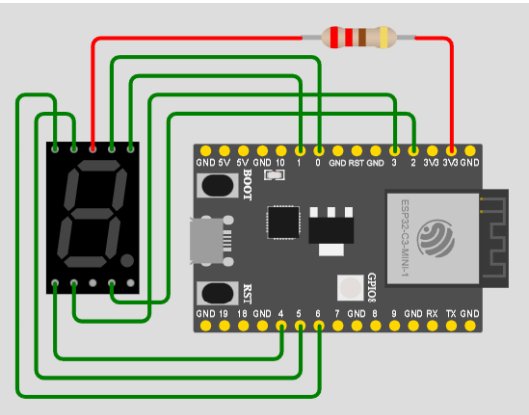
```
.global init

.eqv GPIO_ENABLE_REG, 0x60004020
.eqv GPIO_OUT_W1TS_REG, 0x60004008
.eqv GPIO_OUT_W1TC_REG, 0x6000400C
```

```
init:
  li a1, GPIO_ENABLE_REG    # Setup GPIO0 as output
  li a2, 0x01
  sw a2, 0(a1)

main_loop:
  li a1, GPIO_OUT_W1TS_REG  # GPIO0 -> HIGH
  li a2, 0x01
  sw a2, 0(a1)
  call delay_asm            # Delay

  li a1, GPIO_OUT_W1TC_REG  # Clear GPIO0
  li a2, 0x01
  sw a2, 0(a1)
  call delay_asm            # Delay

  j main_loop               # Loop

# Delay
delay_asm:
  li a3, 0                  # counter
  li a4, 5000000            # wait time (counting times)
loop_delay:
  addi a3, a3, 1
  blt a3, a4, loop_delay
  ret
```

Load the program into Wokwi, run and observe the result.

Home Assignment 3 – Show number in LED 7 segments

The following example illustrates how to implement a circuit that displays the number 0 on a common anode 7-segment LED display. The circuit setup is as shown in the provided diagram. The LED segments a, b, c, d, e, f, g are connected to GPIO pins GPIO0, GPIO1, GPIO2, GPIO3, GPIO4, GPIO5, GPIO6, respectively. These GPIO pins need to be configured as output pins to send signals.

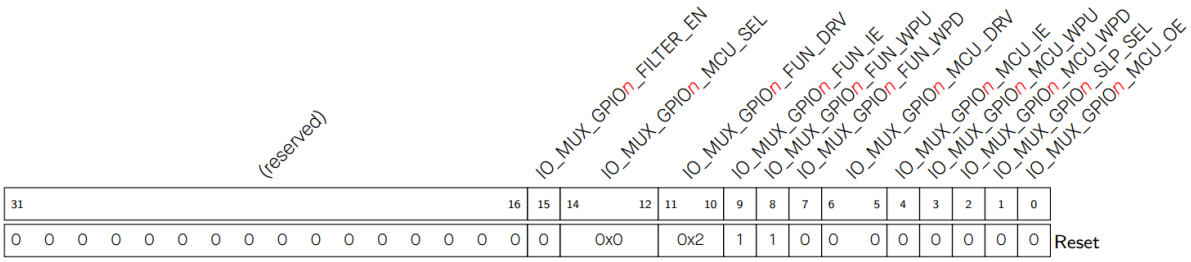


The GPIO pins GPIO4, GPIO5, GPIO6, and GPIO7 have default functionality for the SPI protocol. To output signals on these pins, the function must be selected using the **IO_MUX_GPIOn_REG** register (n from 4 to 7).

Table 2-3. IO MUX Pin Functions

Pin No.	IO MUX / GPIO Name	IO MUX Function [1,4] 0	Type [3]	1	Type	2	Type
4	GPIO0	GPIO0	I/O/T	GPIO0	I/O/T		
5	GPIO1	GPIO1	I/O/T	GPIO1	I/O/T		
6	GPIO2	GPIO2	I/O/T	GPIO2	I/O/T	FSPIQ [5d]	I1/O/T
8	GPIO3	GPIO3 [5a]	I/O/T	GPIO3	I/O/T		
9	GPIO4	MTMS	I1	GPIO4	I/O/T	FSPIHD	I1/O/T
10	GPIO5	MTDI	I1	GPIO5	I/O/T	FSPIWP	I1/O/T
12	GPIO6	MTCK	I1	GPIO6	I/O/T	FSPICLK	I1/O/T
13	GPIO7	MTDO	O/T	GPIO7	I/O/T	FSPID	I1/O/T

Register 5.21. IO_MUX_GPIO*n*_REG (*n*: 0-21) (0x0004+4**n*)



The IO_MUX_GPIOn_MCU_SEL field (bits 12–14) is used to select the function for the GPIOn pin. This field must be set to a value of 1 to configure the pin as GPIO.
***Note**: When simulating with Wokwi, it is not necessary to configure the function for GPIO4, GPIO5, GPIO6, and GPIO7. However, when using a physical development board, this configuration must be set.*

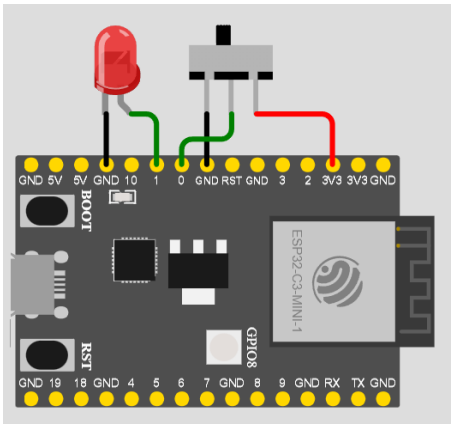
Source code:

```
.global init

.eqv GPIO_ENABLE_REG, 0x60004020    # Enable outpu GPIO
.eqv GPIO_OUT_REG, 0x60004004       # setup output

.eqv IO_MUX_GPIO4_REG, 0x60009014   # Setup function GPIO4
.eqv IO_MUX_GPIO5_REG, 0x60009018   # Setup function GPIO5
.eqv IO_MUX_GPIO6_REG, 0x6000901C   # Setup function GPIO6
.eqv IO_MUX_GPIO7_REG, 0x60009020   # Setup function GPIO7

.text

init:
  li a1, GPIO_ENABLE_REG
  li a2, 0xFF         # output from GPIO0 to GPIO7 (8 bits)
  sw a2, 0(a1)        # setupt bits in GPIO_ENABLE_REG
```

```
  # setup function in GPIO4, GPIO5, GPIO6, GPIO7
  # in default, they are used for SPI function
  # we need to change to GPIO function

  li a2, 0x1000

  li a1, IO_MUX_GPIO4_REG
  sw a2, 0(a1)

  li a1, IO_MUX_GPIO5_REG
  sw a2, 0(a1)

  li a1, IO_MUX_GPIO6_REG
  sw a2, 0(a1)

  li a1, IO_MUX_GPIO7_REG
  sw a2, 0(a1)

  # a1 contains the address of state register GPIO
  li a1, GPIO_OUT_REG
  li a2, 0xC0
  sw a2, 0(a1)            # Output to GPIO
```

Home Assignment 4 – Read states of switch / button

The following example demonstrates a circuit that reads an input signal from GPIO0 and controls the LED on GPIO1.



The connection diagram is as illustrated below.

The **GPIO_IN_REG** register holds the input values of the GPIO pins. By default, GPIO pins are configured as input pins. However, to enable signal input, the **IO_MUX_GPIOx_FUN_IE** bit must be set in the **IO_MUX_GPIOx_REG** register corresponding to the GPIOx pin.

Source code

```
.global init

.eqv GPIO_OUT_W1TS_REG, 0x60004008  # set register
.eqv GPIO_OUT_W1TC_REG, 0x6000400C  # clear register
.eqv GPIO_ENABLE_REG, 0x60004020    # enable output register
.eqv GPIO_IN_REG, 0x6000403C        # state register GPIO
.eqv IO_MUX_GPIO0_REG, 0x60009004   # function register GPIO0

init:
  li a1, GPIO_ENABLE_REG # Set GPIO1 as input
```

```
  li a2, 0x02
  sw a2, 0(a1)

  li a1, IO_MUX_GPIO0_REG # Enable GPIO0 as input
  lw a2, 0(a1)
  ori a2, a2, 0x200       # Set bit IO_MUX_GPIO0_FUN_IE
  sw a2, 0(a1)

loop:
  li a1, GPIO_IN_REG    # Read status of GPIO
  lw a2, 0(a1)
  andi a3, a2, 0x01     # Check GPIO0
  beq a3, zero, clear   # If GPIO0 = 0 => turn off LED
set:
  li a1, GPIO_OUT_W1TS_REG  # turn on LED: Set  GPIO1 = 1
  li a2, 0x02
  sw a2, 0(a1)
  j next
clear:
  li a1, GPIO_OUT_W1TC_REG  # off LED: Clear GPIO1 = 0
  li a2, 0x02
  sw a2, 0(a1)
next:
  j loop                    # Loop
```

Assignment 1

Create a project to implement and test Home Assignment 1. Update the source code to test with other GPIO pins (GPIO2, GPIO3, GPIO4).

Assignment 2

Create a project to implement and test Home Assignment 2. Update the source code to test with other GPIO pins (GPIO2, GPIO3, GPIO4) and adjust the LED blinking duration.

Assignment 3

Create a project to implement and test Home Assignment 3. Update the source code to display different digits (from 0 to 9).

Assignment 4

Create a project to implement and test Home Assignment 4. Update the source code to use other GPIO pins (GPIO2, GPIO3, GPIO4) as signal input pins.

Assignment 5

Create a project to implement a circuit that counts from 0 to 9 on a 7-segment LED display