

Intelligent and Communicating Systems, ICS

2nd Year Specialty SIL G1

Lab Report n°07

Title:

Arduino-Raspberry Wired Communications UART-I2C

Studied by:

First Name: Yasmine

Last Name: DINARI

E_mail: ly_dinari@esi.dz

Contents

1	Theory	2
1.	Definition UART and particularly USB	2
2.	Introduction and Comparing Arduino vs Raspberry UART	2
2.1.	Theoretical study of UART of an Arduino MKR1010 pins and software related to UART	2
2.2.	Theoretical study of UART of Raspberry pins and software related to UART	3
3.	Theoretical study on I2C and SPI communication and comparison of SPI, I2C, and UART protocols	4
3.1.	Theoretical study of I2C of an Arduino MKR1010 pins and software (Library) related to I2c and Analog-to-Digital Converter ADS1115	4
3.2.	Theoretical study of I2C of Raspberry pins and software (Library) related to I2c and to Analog to Digital Converter ADS1115	4
2	Activity	6
1.	Raspberry -I2C (with CAN ADS1115)	6
1.1.	Hardware Setup	6
1.2.	Software Configuration:	7
1.3.	Results and Observations:	9
3	Conclusion	10

Theory

1. Definition UART and particularly USB

UART (Universal Asynchronous Receiver-Transmitter) is a hardware communication protocol that enables asynchronous serial communication between devices. It transmits data bit by bit over 02 wires: one for **transmitting** (TX) and one for **receiving** (RX). Unlike synchronous protocols, UART does not require a shared **clock** signal; instead, both devices (connected directly) must agree on a **common baud rate** (data transmission speed). UART is widely used for low-speed, short-distance communication, such as between microcontrollers and peripheral devices. It implements basic error-checking mechanisms, like parity bits, to ensure data integrity during transmission.

USB (Universal Serial Bus) is a separate and more advanced standard for serial communication. USB provides higher data transfer rates, supports plug-and-play device recognition, and can supply power to connected devices. Unlike UART, USB communication is typically managed by a host (such as a PC) and supports multiple device connections through a tiered star topology. USB is often used for connecting peripherals like keyboards, mouse, storage devices...

2. Introduction and Comparing Arduino vs Raspberry UART

2.1. Theoretical study of UART of an Arduino MKR1010 pins and software related to UART

UART Pins :

On the Arduino MKR1010, the hardware UART interface is available on specific pins:

- TX (Transmit): Pin 14
- RX (Receive): Pin 13

These pins are used for connecting the board to other UART-compatible devices, such as sensors or another microcontroller. **Software Functions :** The Arduino IDE provides built-in support for UART communication through the Serial1 object such as:

- Serial1.begin(baudrate) to initialize communication at a chosen speed
- Serial1.write() to send data

- `Serial1.read()` to receive data

2.2. Theoretical study of UART of Raspberry pins and software related to UART

The Raspberry Pi implements UART communication through GPIO pins and software configuration:

Primary UART Pins (BCM mode):

- TX (Transmit): GPIO 14 (Physical Pin 8)
- RX (Receive): GPIO 15 (Physical Pin 10)

These pins are shared with Bluetooth on Pi 4 model.

Voltage Levels:

Uses 3.3V logic (incompatible with 5V devices without level shifting).

Software Configuration

Enable UART via: Terminal: `sudo raspi-config > Interface Options > Serial Port > Disable shell/Enable hardware Edit /boot/config.txt: Add enable_uart=1.`

Software Functions :

- Python (pyserial): `serial.Serial("/dev/ttyS0", baudrate)` for Pi 3/4
- C/WiringPi: `wiringPi.serialOpen("/dev/ttyAMA0", 9600)`
- Terminal tools: `screen /dev/ttyAMA0 115200` for debugging

3. Theoretical study on I2C and SPI communication and comparison of SPI, I2C, and UART protocols

3.1. Theoretical study of I2C of an Arduino MKR1010 pins and software (Library) related to I2c and Analog-to-Digital Converter ADS1115

Hardware:

The Arduino MKR1010 supports I2C communication using two dedicated pins: SDA (data line) on pin 11 and SCL (clock line) on pin 12. These pins allow the board to connect to multiple I2C devices, such as sensors or analog to digital converters, using only two wires for both data and clock signals. The ADS1115 analog to digital converter connects to the same I2C bus and offers 04 input channels with 16-bit resolution, enabling analog measurements. Its I2C address can be set by hardware configuration, and it is powered by the same voltage as the MKR1010, ensuring compatibility for data transfer.

Software:

I2C communication on the **Arduino** MKR1010 is managed through the built in Wire library, which provides simple functions for initializing the bus, sending, and receiving data. For the ADS1115, dedicated libraries such as Adafruit_ADS1X15 make it easy to configure the converter and read analog values. In a typical program, the Wire library is initialized with Wire.begin(), and the ADS1115 library is used to set up the converter and retrieve measurements from its input channels, allowing the integration of analog sensing.

3.2. Theoretical study of I2C of Raspberry pins and software (Library) related to I2c and to Analog to Digital Converter ADS1115

Hardware:

The Raspberry Pi supports I2C communication using two dedicated GPIO pins: SDA (data line) on GPIO 2 and SCL (clock line) on GPIO 3. These pins are equipped with pull-up resistors to 3.3V and are designed specifically for I2C communication, allowing multiple devices to share the same bus as long as each has a unique address. The ADS1115 analog to digital converter connects to these I2C pins and provides four analog input channels with 16-bit resolution, enabling accurate analog measurements. The ADS1115 operates with a supply voltage between 2.0V and 5.5V and communicates over I2C using a configurable 7-bit address, ensuring compatibility with the Raspberry Pi's 3.3V logic levels.

Software:

On the Raspberry Pi, I2C communication can be managed in Python using the `smbus` or `smbus2` library, which provides functions for reading from and writing to I2C devices. To interact with the ADS1115, you create an SMBus object (e.g. `bus = smbus2.SMBus(1)`) and use methods such as `write_byte_data()` and `read_i2c_block_data()` to configure the ADC and retrieve analog measurements. This approach allows the Raspberry Pi to communicate with the ADS1115 and integrate analog sensor readings.

Activity

1. Raspberry -I2C (with CAN ADS1115)

The goal of this activity is to control the brightness of an LED connected to the Raspberry Pi, based on the ambient light measured by an LDR (Light Dependent Resistor). The analog value from the LDR is read using an ADS1115 analog-to-digital converter via the I2C protocol.

1.1. Hardware Setup

- The ADS1115 module is connected to the Raspberry Pi as follows:
 - GND to GND
 - VDD to 3.3V
 - SCL to GPIO 3 (SCL, Pin 5)
 - SDA to GPIO 2 (SDA, Pin 3)

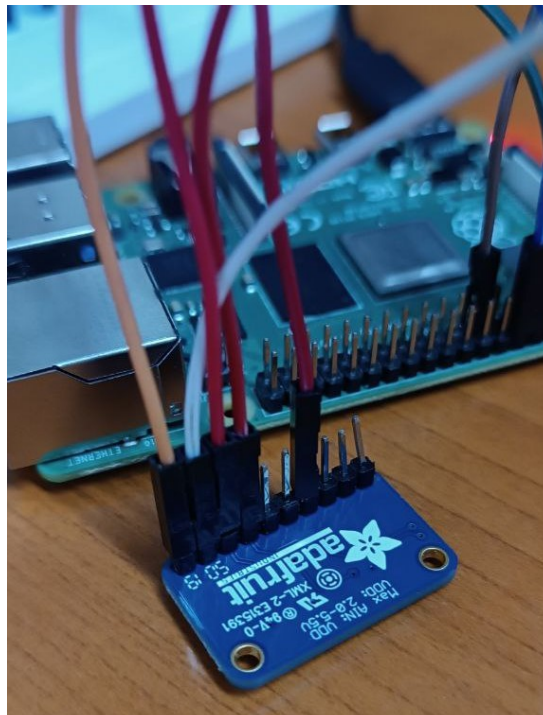


Figure 2.1: Connection of ADS1115 to the Raspberry Pi.

To enable I2C communication on the Raspberry Pi, open the configuration tool using

```
1 sudo raspi-config
```

Then navigate to **Interface Options** and activate the I2C interface. After that, install the necessary I2C tools with

```
1 sudo apt install i2c-tools
```

To verify the connection and detect the address of the ADS1115 ADC, use the command :

```
1 i2cdetect -y 1
```

and confirm that the ADC appeared at its default I2C address (0x48).

- The LDR is connected in series with a 10k Ω resistor to form a voltage divider:
 - One end of the LDR to 3.3V
 - The other end to A0 of ADS1115 and to one end of the 10k Ω resistor
 - The other end of the resistor to GND
- The LED (with a 220 Ω resistor) is connected to a PWM-capable pin (GPIO 18, Pin 12) of the Raspberry Pi.

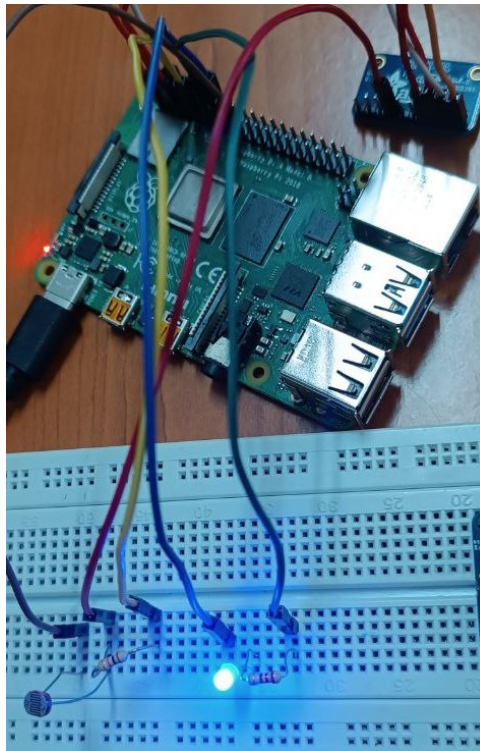


Figure 2.2: Complete breadboard setup: Raspberry Pi, ADS1115, LDR, and LED.

1.2. Software Configuration:

- The code initializes the ADS1115 for single-ended input on channel A0, sets the gain, and configures the data rate.

- In a loop, the Raspberry Pi reads the raw analog value from the LDR via the ADS1115.
- The raw value is normalized and mapped to a PWM value between 0 and 1, which is then used to set the LED brightness.
- Realtime output are shown in Figure 2.3.

```

1 # Import necessary libraries
2 import smbus2          # For I2C communication with ADS1115
3 import time            # For delays
4 from gpiozero import PWMLED # For controlling LED brightness (PWM)
5
6 # Set up the LED on GPIO 18 (supports PWM)
7 ledPin = PWMLED(18)
8
9 # Set up I2C bus and ADS1115 address
10 I2C_BUS = 1            # I2C bus number on Raspberry Pi
11 ADS1115_ADDRESS = 0x48 # Default I2C address for ADS1115
12 bus = smbus2.SMBus(I2C_BUS) # Create an I2C bus object
13
14 # ADS1115 register addresses
15 ADS1115_CONVERSION = 0x00 # Register to read ADC value
16 ADS1115_CONFIG = 0x01     # Register to configure ADC
17
18 # Function to configure the ADS1115
19 def initialize_ads1115():
20     # Build config value for single-ended AIN0, gain, mode, and data rate
21     config = (
22         (0x40 << 8) # Select channel AIN0
23         | (0x04 << 8) # Set gain to ±4.096V
24         | (0x80)      # Continuous conversion mode
25         | (0x03)      # Data rate: 128 samples/sec
26     )
27     config_bytes = [config >> 8, config & 0xFF] # Split config into two
bytes
28     bus.write_i2c_block_data(ADS1115_ADDRESS, ADS1115_CONFIG, config_bytes)
# Send config
29
30 # Function to read a value from the ADS1115
31 def read_ads1115():
32     result = bus.read_i2c_block_data(ADS1115_ADDRESS, ADS1115_CONVERSION,
2) # Read 2 bytes
33     raw_value = (result[0] << 8) | result[1] # Combine bytes to get 16-bit
value
34     if raw_value > 0x7FFF:                    # Convert to signed number if
needed
35         raw_value -= 0x10000
36     return raw_value
37
38 # Initialize the ADS1115
39 initialize_ads1115()
40
41 # Main loop
42 try:
43     while True:
44         ldr_raw = read_ads1115() # Read value from LDR

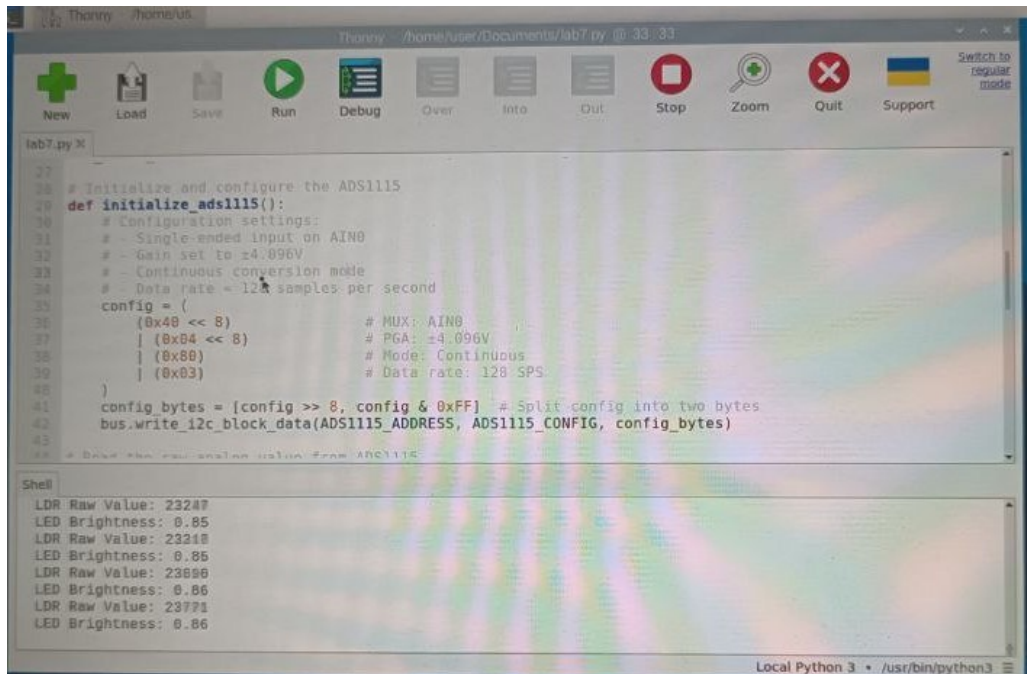
```

```

45     print(f"LDR Raw Value: {ldr_raw}") # Print the value
46
47     # Convert raw value (-32768 to +32767) to range 0.0 to 1.0 for PWM
48     brightness = (ldr_raw + 32768) / 65535
49     print(f"LED Brightness: {brightness:.2f}") # Print brightness
50
51     ledPin.value = brightness # Set LED brightness
52
53     time.sleep(0.5) # Wait before next reading
54
55 except KeyboardInterrupt:
56     print("Exiting ...")
57
58 finally:
59     bus.close() # Close the I2C bus when done

```

Listing 2.1: Code for ADS1115 LDR readings and LED brightness values



```

lab7.py X
27
28 # Initialize and configure the ADS1115
29 def initialize_ads1115():
30     # Configuration settings:
31     # - Single-ended input on AIN0
32     # - Gain set to ±4.096V
33     # - Continuous conversion mode
34     # - Data rate = 128 samples per second
35     config = (
36         (0x40 << 8) # MUX: AIN0
37         | (0x84 << 8) # PGA: ±4.096V
38         | (0x80) # Mode: Continuous
39         | (0x03) # Data rate: 128 SPS
40     )
41     config_bytes = [config >> 8, config & 0xFF] # Split config into two bytes
42     bus.write_i2c_block_data(ADS1115_ADDRESS, ADS1115_CONFIG, config_bytes)
43
44 # Read raw value and calculate LED brightness from ADS1115
45
46 Shell
47 LDR Raw Value: 23247
48 LED Brightness: 0.85
49 LDR Raw Value: 23218
50 LED Brightness: 0.85
51 LDR Raw Value: 23698
52 LED Brightness: 0.86
53 LDR Raw Value: 23771
54 LED Brightness: 0.86

```

Figure 2.3: Live output: LDR raw values and corresponding LED brightness.

1.3. Results and Observations:

- As the light intensity on the LDR increases, the raw value read from the ADS1115 increases, resulting in a higher PWM value and brighter LED.
- Conversely, reducing the light on the LDR decreases the LED brightness.
- The system demonstrates real-time analog-to-digital conversion and PWM control using I2C communication.

Conclusion

This lab demonstrated the successful implementation of I2C communication between the Raspberry Pi and the ADS1115 analog-to-digital converter. Through configuring both the hardware and software, analog data from an LDR sensor was accurately acquired and used to control the brightness of an LED via PWM. Overall, this experiment provided a clear understanding of both the theoretical and practical aspects of I2C communication and external ADC usage in embedded applications.