Lab 10: UART

- **Link**
  - Document
  - Youtube

- **Basic** (50%)
  - **Description**

    The objective of this assignment is to design and implement a digital counter system using a push-button input, UART output, and an LED display. Upon system initialization, the UART terminal must present an initial value of 0. Each time the user presses the push-button, the counter must increment by 1, and the updated value must be immediately transmitted through the UART interface in its decimal representation.

    In parallel with the UART output, the LED array must display the counter value **modulo 16**, represented in binary format. The design must guarantee that only one increment is registered per physical button press, requiring proper handling of mechanical button bounce.

    To maintain a clean, real-time UART display, the output must not scroll. Instead, each updated value must overwrite the previously displayed value using the **Backspace character (\b)**.

  - **Standard of grading**
    1. Both C and Assembly language implementations are acceptable.
    2. The program must overwrite the previous UART value.
    3. The LED output must match the value shown on the UART terminal.

- **Advance** (30%)
  - **Description**

    Develop a **cyclic up counter** that continuously counts from **0x00** up to **0x0F**, then wraps around to 0x00 to repeat the cycle. The counter output must be displayed on **RD7–RD4** LEDs.

    When the system starts up, the LED update interval must be 1.0 second. Users can change this interval at any time via the UART interface by inputting a value between 0.1 and 1.0 seconds (the spacing is 0.1, eg. 0.1, 0.2, 0.3 … 1.0). After the user presses the Enter key, the system must immediately apply the new delay to the counter, and the LED update frequency must change accordingly. Importantly, the counter cannot restart counting after the delay value is changed, it must continue counting from the current value. All timing functions must be implemented using a timer (e.g., TIMER2).

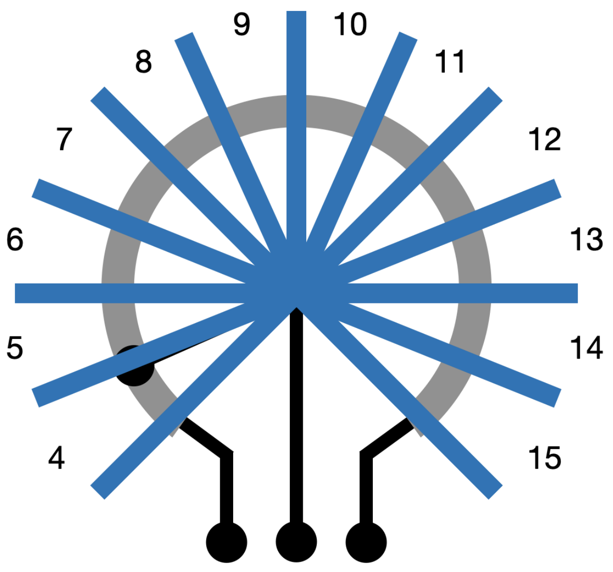
    The program must enable interrupt priority and assign a higher priority to the UART interrupt than to the timer interrupt. This configuration ensures that any incoming UART data can interrupt and preempt the timer operation immediately.

- **Standard of grading**
  1. Both C and Assembly language implementations are acceptable.
  2. Correct operation of 4-bit output via LEDs RD7–RD4.
  3. Must use TIMER2.
  4. Must enable interrupt priority and assign a higher priority to the UART interrupt than to the timer interrupt.

- **Hard** (20%)
  - **Description**

    The objective of this assignment is to develop a monitoring and real-time processing of an analog input signal derived from a connected potentiometer. The program must first sample the 10-bit Analog-to-Digital Converter (ADC) input, denoted as R_ADC, which spans the range of [0, 1023]. Subsequently, a non-linear mapping transformation must be applied: the entire R_ADC range must be mathematically partitioned into 12 equal-width segments, with each segment corresponding to a specific output value selected from the set {4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}. The mapping logic must strictly adhere to the following segment boundaries, and resulting output values:

ADC Range	Output Value
[0, 85)	4
[85,170)	5
[170, 256)	6
[256, 341)	7
[341, 426)	8
[426, 512)	9
[512, 597)	10
[597, 682)	11
[682, 767)	12
[767, 852)	13
[852, 938)	14
[938, 1024)	15



The **output values** must be represented on the connected LED array. At the same time, the program must transmit the decimal representation of **ADC values** via the UART serial interface. A critical requirement for the serial output is the **Mandatory In-Place Update**: when the potentiometer is adjusted, the UART update must only be triggered when **the R_ADC value results in a change** in the output value state, not on every ADC read. The system must utilize the **Backspace character (\b)** to erase the previously displayed value and rewrite the new value in the same terminal position, thereby preventing standard scrolling log output and ensuring a clean, dynamic display on the terminal.

- **Standard of grading**
  1. Both C and Assembly language implementations are acceptable.
  2. The program must use the Mandatory In-Place Update.
  3. The value is represented on the LEDs, and the terminal must match.
  4. Ensure the mapping logic strictly adheres to the boundaries.

- Note: If you have any questions, please send an email to the TA mailbox (mprocessor2025@gmail.com). Emails sent to individual TAs will not be answered.