ME381 ROBOTICS

Experiment 2: Basics of microcontroller (Time dependent execution and communication)

Aim: Learn about the timing functions, Interrupt Service Routine (ISR), and UART communication using Arduino Uno microcontroller.

This module is aimed at imparting the knowledge required for making Arduino Uno perform time-dependent tasks. The student should be able to understand the blocking and non-blocking functions used by Arduino to fulfill these timing requirements. Additionally, this module also introduces the UART communication protocol used by Arduino to communicate with external devices. The following table shows in brief the tasks to be performed in this lab. The details of these tasks and reporting for evaluation are given in the 'Tasks' section. The 'Task Manual' describes the detailed procedure to be followed for performing each task.

Sr. No.	Experiment name	Lab work to be performed in brief
1	Basics of microcontrollers (Time dependent execution and communication)	1. LED blinking without delay 2. Handling interrupts and ISR 3. Serial communication: UART

Tasks:

- Each student in the group should perform all the tasks listed in the table below. The reported outputs obtained after performing each of these tasks should be mentioned in their report.
- Each student should get at least one task individually assigned to him/her by the TA. He/she is supposed to extensively present all the results for that task in his/her report such as taking snapshot of the circuit and snapshot of the serial monitor or any other output device.

Sr. no.	Topic	Task	Report
1	Using timer millis()	Unlike the previous lab, blink the built-in LED without using delay() function this time. Write a program to light up the built-in LED for 100ms within every elapsed second.	Tweak the provided program to light up an LED for 1 second and then turn it off for the next 3 seconds. Keep repeating this pattern. Report the lines of code where you made the changes to achieve this.
2	Interrupts and ISR	Write a program to light up the built-in LED (and stay ON forever) when interrupt pin 2 detects a FALLING edge. Use `pinMode(interruptPin, INPUT_PULLUP)` to maintain the interrupt pin 2 at 5V always. Connect pin 2 to GND once to obtain a falling edge. Remove the GND connection to pin 2. Use the RESET button on Arduino to reset the microcontroller operation.	2(a) Using a multimeter, report the voltage output at pin 2 before connecting it to GND. 2(b) Using a multimeter, report the voltage output at pin 2 after connecting it to GND once and then disconnecting it. 2(c) What happened to LED after connecting pin 2

			to GND once and then removing this connection? 2(d) Write your observation for pin2 voltage and LED status when RESET button of Arduino is pressed once.
3	Interrupts and ISR	The code for task 3 is very similar to task 2 code, except it introduces a ten-second delay in the main loop unlike task 2 code. Connect pin 2 to GND once to obtain a falling edge. Remove the GND connection to pin 2.	3(a) Report what happens to LED after connecting pin 2 to GND once and then removing this connection? Does it light up instantly or with some delay? 3(b) Is there any effect of changing the delay duration in the main loop? Specify reasons for the same.
4	Serial communication: UART (transmitter)	The code for task 4 transmits an integer varying from 1 to 10 printing each consecutive number after a delay of certain seconds (identify this delay duration given in the provided code for Task 4). Change the provided code to produce each consecutive number on the serial monitor after a delay of 2 seconds from the previously produced number.	4(a) Report after how many seconds is the provided code producing a new number on the serial monitor? 4(b) Report the changed line of code to produce each consecutive number on the serial monitor after a delay of 2 seconds from the previously produced number.
5	Serial communication: UART (receiver)	Upload the code for Task 5 to receive an integer from Serial Monitor (use `Serial.parseInt()`). Send any number between 0-10 from the computer by typing it into the Serial monitor and pressing enter. Blink the built-in LED that number of times within the next 5 seconds after receiving the number.	Report the number entered by you in the serial monitor and the LED blinking pattern with timing details (number of glows, duration of one such LED glow, and total time for all the glows).
6	Serial communication: UART (transmitter and receiver)	Upload the provided transmitting code (‘MicrocontrollerA_serial_transmitter.ino’) to board A and receiving code (‘MicrocontrollerB_serial_receiver.ino’) to board B. Follow the process mentioned for Task 6 in the ‘Task Manual’.	6(a) What time interval is Board A using between transmission of subsequent numbers? 6(b) Report the LED blinking pattern on Board B when Board A transmits the number ‘4’ along with the timing details of these blinks (number of glows, duration of one such LED glow, and total time for all the glows)

Task manual:

1. Using timer millis():

Upload the code for Task 1 and tweak the code as mentioned in the 'Tasks' table. Observe the built-in LED. The essential details to be learned after this task are as follows.

millis() function returns the number of milliseconds passed since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days. The return type of millis() is *unsigned long*.
Blocking functions: These functions bring the execution of program to a halt. It does not let the control to perform any other task until the blocking function has executed completely. The delay() as used in the previous lab is one such function.

Non-blocking functions: These functions initiate their desired purpose while allowing the control to proceed and execute the next lines of code. The control is not required to wait for execution of a non-blocking function to get completed. For example, a non-blocking function may initiate sampling of voltage by an external ADC (Analog to Digital Convertor). It can let the control to proceed to next lines of code without ADC sampling to get completed. The ADC may signal the microcontroller using an interrupt pin when it is ready with the sampled value. millis() is also a non-blocking function.

2. Interrupts and ISR:

Upload the code for Task 2. Connect the pin 2 to GND once as shown in the figure below and then remove this connection. Note down the observations mentioned in the 'Tasks' table.

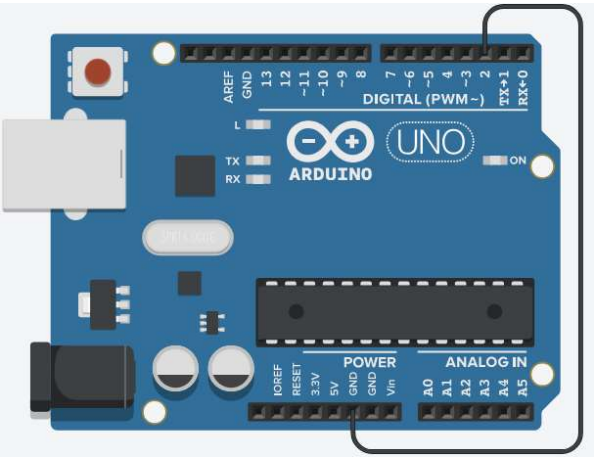

Figure 1 Pin 2 connected to GND

The essential details to be learned after this task are as follows:

Interrupt: An interrupt is a signal that temporarily halts the processor's current task to execute a separate routine. Once the routine is completed, the processor resumes its previous task.

The following function is used to interrupt the execution of current code for some time and execute an Interrupt Service Routine (ISR) which is typically a small piece of code that takes very less time to execute.

attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)

where the parameters of the function are:

pin: the Arduino pin number.

ISR: the ISR to call when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an interrupt service routine.

mode: defines when the interrupt should be triggered. Four constants are predefined as valid values:

- **LOW** to trigger the interrupt whenever the pin is low,
- **CHANGE** to trigger the interrupt whenever the pin changes value
- **RISING** to trigger when the pin goes from low to high,
- **FALLING** for when the pin goes from high to low.

3. Interrupts and ISR:

Upload the code for Task 3. Connect the pin 2 to GND once as shown in the figure below and then remove this connection. Note down the observations mentioned in the 'Tasks' table.

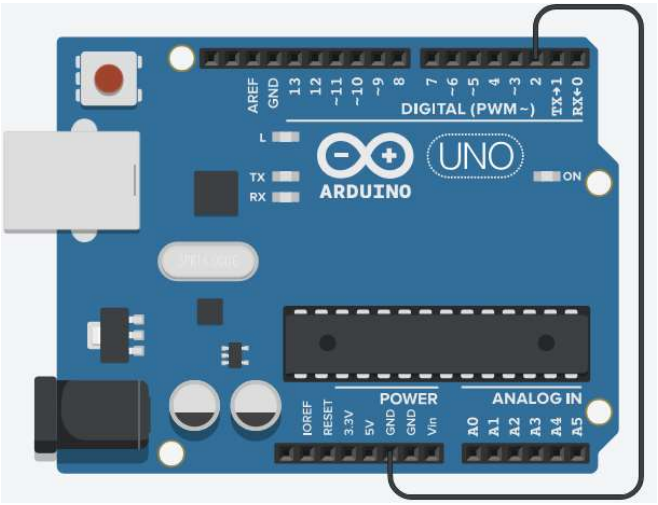

Figure 2 Pin 2 connected to GND

4. Serial communication: UART (transmitter)

Upload the code for Task 4. Observe the serial monitor. Note down the observations mentioned in the 'Tasks' table. Make changes in the code to print the numbers from 0 to 10 with numbers printing after every 2 seconds.

The Arduino microcontroller and the Serial Monitor communicate with each other using the UART communication protocol. The details of the UART protocol are given below.

UART (universal asynchronous receiver-transmitter), is one of the most used device-to-device communication protocols. Asynchronous means there is no clock signal to synchronize the output bits from the transmitting device going to the receiving end. The UART interface does not use a clock signal to synchronize the transmitter and receiver devices; it transmits data asynchronously. Instead of a clock signal, the transmitter generates a bitstream based on its clock signal while the receiver is using its internal clock signal to sample the incoming data. The point of synchronization is managed by having the same baud rate on both devices. Failure to do so may affect the timing of sending and receiving data that can cause discrepancies during data handling.

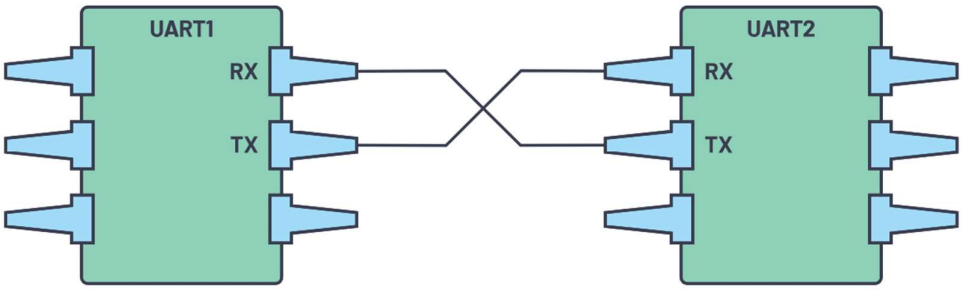

Figure 3 Tx of one UART is connected to Rx of the other UART and vice-versa

In UART, the mode of transmission is in the form of a packet. The piece that connects the transmitter and receiver includes the creation of serial packets and controls those physical hardware lines. A packet consists of a start bit, data frame, a parity bit, and stop bits.

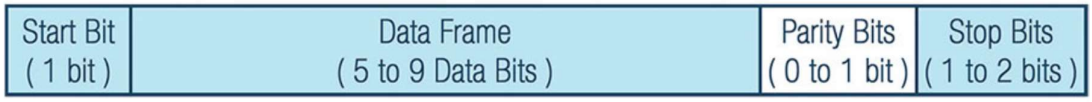

Figure 4 UART packet

- **Start Bit** The UART data transmission line is normally held at a high voltage level when it's not transmitting data. To start the transfer of data, the transmitting UART pulls the transmission line from high to low for one (1) clock cycle. When the receiving UART detects the high to low voltage transition, it begins reading the bits in the data frame at the frequency of the baud rate.
- **Data Frame** The data frame contains the actual data being transferred. It can be five (5) bits up to eight (8) bits long if a parity bit is used. If no parity bit is used, the data frame can be nine (9) bits long. In most cases, the data is sent with the least significant bit first.
- **Parity** Parity describes the evenness or oddness of a number. The parity bit is a way for the receiving UART to tell if any data has changed during transmission. Bits can be changed by electromagnetic radiation, mismatched baud rates, or long-distance data transfers.
  After the receiving UART reads the data frame, it counts the number of bits with a value of 1 and checks if the total is an even or odd number. If the parity bit is a 0 (even parity), the 1 or logic-high bit in the data frame should total to an even number. If the parity bit is a 1 (odd parity), the 1 bit or logic highs in the data frame should total to an odd number.
  When the parity bit matches the data, the UART knows that the transmission was free of errors. But if the parity bit is a 0, and the total is odd, or the parity bit is a 1, and the total is even, the UART knows that bits in the data frame have changed.
- **Stop Bits** To signal the end of the data packet, the sending UART drives the data transmission line from a low voltage to a high voltage for one (1) to two (2) bit(s) duration.

5. Serial communication: UART (receiver)

Upload the code for Task 5. Open the serial monitor and enter any number in the range 0-10 and observe. Report the LED glow pattern as mentioned in the 'Tasks' table.

6. Serial communication: UART (transmitter and receiver)

Upload the program for Microcontroller A on one of the Arduino boards. Upload the program for Microcontroller B on another Arduino board.

1. Close all instances of the Arduino IDE on your computer.
2. Disconnect Arduino board B from your computer and use the AC-DC adapter to power it.
3. Connect Arduino board A to your computer just for the sake of powering it.
4. Connect the Tx of board A to Rx of board B.

5. Connect the GND pins of both A and B boards.
6. Reset both the boards and observe the LED blinking.

Using these codes, ***Board A*** transmits an integer varying from 1 to 10 once every 6 seconds. ***Board B*** receives it and shows an LED blinking pattern based on the number received from board A.

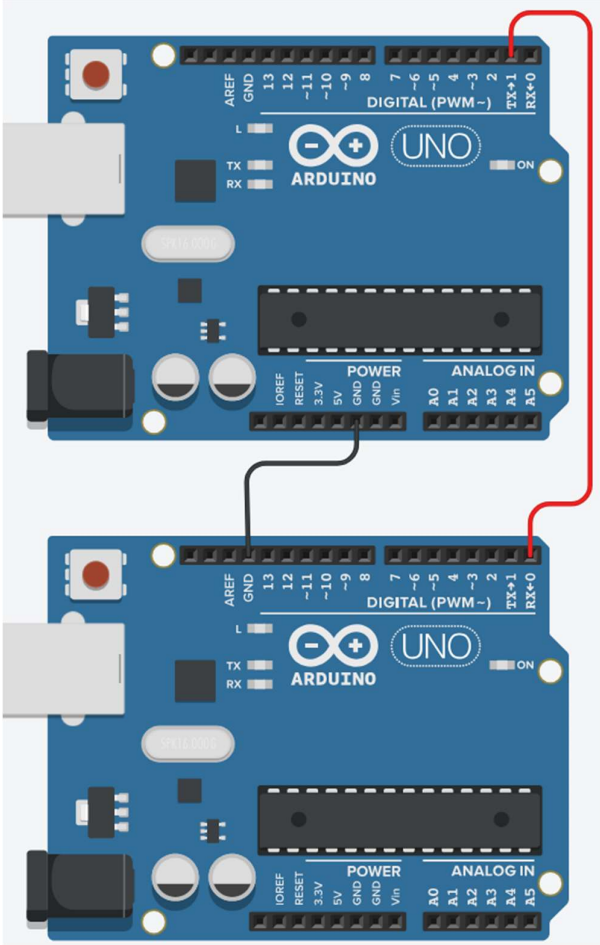

Figure 5 Board A and Board B communicating through UART communication

In the above case, since board B is not transmitting anything back to board A, so it is not required to connect the Tx of board B to Rx of board A. Otherwise, we need to connect the Tx and Rx pins of board A and board B as shown in Figure 3.

REPORT FORMAT

(Submit a single pdf file of your report to Hello IITK after renaming the file in the format: ME381_E2_Name_RollNo_GroupNumber)

Experiment No.

Experiment Title:

Group No.:

Name:

Roll no.

Results for the group task:

Task no.	Topic	Task	Report
1			
2			
3			
4			
5			
6			

Results for individually assigned task:

- Task description:
- Pictures of the circuit along with multimeter reading (if any):
- Pictures of the serial monitor (if any):