Project 2: 9/25/2025

Document name: CE Task Manager Project Report
Document reference: andrewOmer.cmpe311.fall25.project#2
Date of publication: November 11, 2025

LEAD ENGINEER: Andrew Omer, University of Maryland Baltimore County, Baltimore, Maryland, USA

STAKEHOLDERS:
Prof Kidd, Instructor, University of Maryland Baltimore County, Baltimore, Maryland, USA
MD Safwan Zaman, TA, University of Maryland Baltimore County, Baltimore, Maryland, USA

HIGH-LEVEL DESCRIPTION: This document is the project document for Project#2 of the CMPE311 class. It contains customer, technical and testing requirements as well as the design and results of the validation testing.

DESCRIPTION: This design is to create a circuit on an Arduino Uno R3 development platform that allows the user of the program to independently specify an LED number and its blink interval. The blinking of the LEDs must continue asynchronously with any input from the user. Included in this document are the customer requirements obtained from the customer, the high-level technical requirements derived from those customer requirements, the design, the testing scenarios and requirements, and the results of testing. Appended to this document is the code executed and a video of those tests that required video documentation.

RESULT SUMMARY: The project was a success, the embedded system design meeting all testing and high-level requirements.

REFERENCES AND GLOSSARY

REFERENCES:

- Asynchronous Tasking Project Report – Prior CMPE 311 project accomplishing the same task with different software
- projectTaskMgmtCE.pdf– The definition of the project this document addresses
- CMPE310 Project #5 – A similar project assigned in CMPE310 in Spring 2025
- Arduino UNO R3 Product Reference Manual SKU A000066, 12/03/2024
- Async Programming in Arduino: Unleashing the Power of Non-Blocking Code, Mahdi Valizadeh, medium.com, 4/8/2024

***DEFINITIONS*:**

"The User" – The person operating (not programming) the embedded system
"The System" – The embedded system being operated by The User
"The Customer" – The person(s) paying for the embedded system being designed and built
"The Developer" – The person(s) designing and building the System
"The Evaluator" – The person(s) that determine whether or not The System satisfies The Customer-requirements.
"The Customer-requirements" – The requirements defined by The Customer as satisfying The Contract.
"The Requirements" – The System's high-level technical requirements derived from The Customer-requirements.
"The Educational-constraints" – Requirements imposed by the instructor unrelated to the embedded system that allow The System to be evaluated.
"The Company" – The organization The Customer has contracted with to build The System.
"The Contract" – The business document that legally binds The Company to provide some service or product to The Customer.
"serial-monitor" – The serial port used by the Arduino IDE to communicate with The User.
"The Reference-platform" – The configuration of The System used by The Developer to test and validate The System. For this class, The System is the Arduino compatible ELEGOO Uno R3 development board.
"PROJECT-ASYNC" – The previous project upon which this project is based

ACRONYMS AND ABBREVIATIONS:

Arduino – an Italian open-source hardware and software company; also refers to a development board created by the company
arduino.h – header for a library of convenience functions specific to the Arduino development platform

AVR – A family of microcontrollers, originally developed by Atmel, and currently owned by Microchip Technology
ELEGOO – A Chinese company that develops and markets 3D printers and accessories
IDE – Integrated Development Environment
gcc – front end for the GNU Compiler Collection
Github – A widely used distributed SVC (Software Version Control) system
LED – Light Emitting Diode
CE – Cyclic Executive

REQUIREMENTS

CONVENTIONS:
Must, shall or will – your design must satisfy the requirement
May – your design may satisfy the requirement but doesn't have to
Informative – the intent of the following description is to make the requirement more understandable
All customer requirements are started with "C.#".
All high-level requirements are started with "HL.#".
All testing/validation requirements ar5e started with "T.#"

CUSTOMER REQUIREMENTS:

The customer requirements, C1 through C5, are identical to that of PROJECT-ASYNC.
Ref: andrewOmer.cmpe311.fall25.project#1, CUSTOMER REQUIREMENTS section.

HIGH-LEVEL TECHNICAL REQUIREMENTS:

The high level technical requirements, HL.1 through HL.3, are identical to that of PROJECT-ASYNC.p
Ref: andrewOmer.cmpe311.fall25.project#1, HIGH-LEVEL TECHNICAL REQUIREMENTS section.

EDUCATIONAL REQUIREMENTS:

E.1.0 Implement a cyclic executive task manager to dispatch the asynchronous tasks you created
in the previous project, PROJECT-ASYNC.
E.1.1 The system testing and validation requirements must contain those defined in PROJECT-ASYNC.
E.1.1.1 The testing and validation requirements must contain any additional tests necessary to properly evaluate/demonstrate the function of the system.
E.1.2 The final report must be a formal design document as in PROJECT-ASYNC.

DESIGN:

DESIGN PRE-REQUISITES:

1. ELEGOO Arduino Uno R3 clone
2. Arduino IDE 2.3.3 or better

DEVELOPMENT PLATFORM:

1. See DESIGN PRE-REQUISITES above

ANY ADDITIONAL DESIGN CONSIDERATIONS:

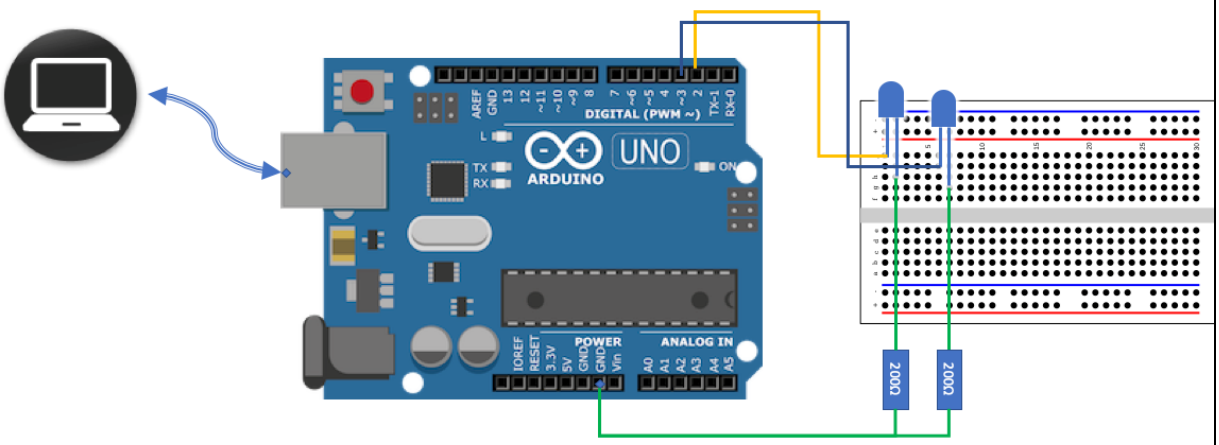
1. None

See Appendix A for the logical flow of the project program.
See Appendix B for the code executing on the Arduino Uno R3.
See Appendix C for testing video.

TESTING AND VALIDATION

DESCRIPTION: The tests that have to be performed and validated are shown in Table 1. These tests were performed on the testbed shown Figure 1. Table 1 shows a dialog that must be successfully performed by the embedded system. The results of testing are shown in Table 3. A video of the test has been linked within this report, and attached to it's submission.

Figure 1. Testbed Setup. LEDs connected to digital pins 2 & 3.



TESTING PLATFORM:

1. ELEGOO Arduino Uno R3 clone
2. Arduino IDE 2.3.3 or better
3. Power: Testing platform powered through USB cable, LEDs connected directly through Digital Outputs

Table 1. Required Test Dialog (***blue*** are the user inputs).

Serial port I/O	**Notes**
What LED? (1 or 2) ***2***	
What interval (in msec)? ***600***	LED2 starts blinking at an interval of 300ms on and 300ms off. LED1 is unaffected.
What LED? (1 or 2) ***1***	
What interval (in msec)? ***1200***	LED1 starts blinking at an interval of 600ms on and 600ms off. LED2 is unaffected.
What LED? (1 or 2)	Waiting for next LED# interval pair
What LED? (1 or 2) ***2***	
What interval (in msec)? ***1200***	LED2 starts blinking at an interval of 600ms on and 600ms off. LED1 is unaffected.

TESTING AND VALIDATION REQUIREMENTS:

Table 2. Testing and Validation Requirements

T.0 All testing and validation must be done on the testbed illustrated in Figure 1.
T.1 The dialog above (or similar) must work as shown
T.1A The blink rate of an LED must be able to be set without interfering with the blinking of the other LED
T.1.1 Setting of the blink rate (and LED#) must be through the IDE serial monitor
T.2 The blink rate of the LED being set must not change until the input is complete
T.3 The user must specify the blink rate in milliseconds per blink
T.4 The blink rate specified by the user must be correctly reflected on the testbed LEDs.
T.4.1 The blink rate specified by the user must be reflected on the LED specified by the user
T.5. The setting of an LED's blink rate must be able to be repeated at least 2 times
T.5.1. Successively for the same LED
T.5.2. Alternately between the different LEDs

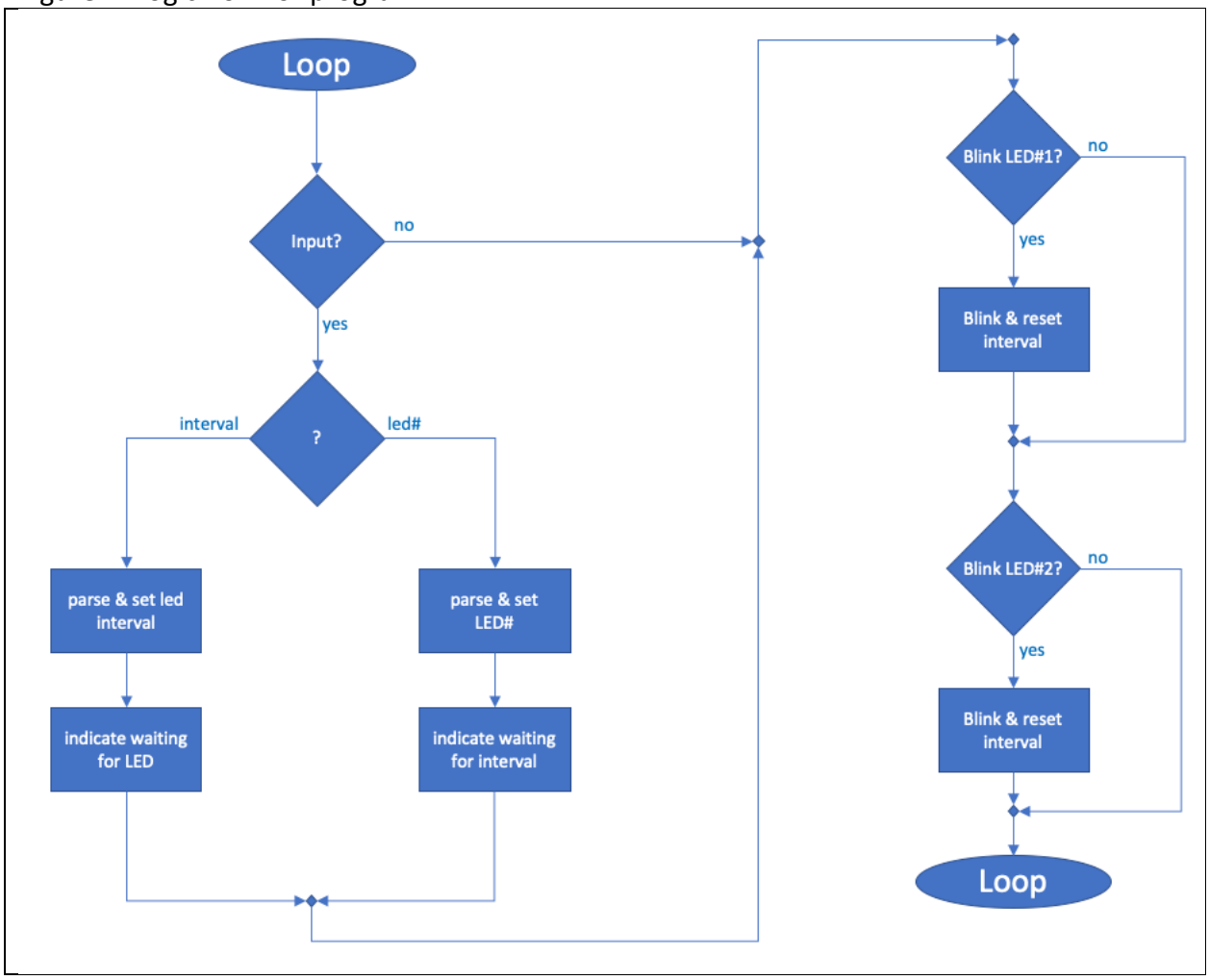
TESTING RESULTS:

Table 3. Results of tests

Test performed	Results
T.1.1	Satisfied. See video of test.
T.1	Satisfied
T.2	Satisfied. See video of test.
T.3	Satisfied
T.4	Satisfied
T.4.1	Satisfied. See video of test.
T.5.1	Satisfied. See video of test.
T.5.2	Satisfied. See video of test.

Appendix A. Logic Flow Chart

Figure 2. Logic flow for program.



APPENDIX B. DESIGN CODE

```
#define LED1 2
#define LED2 3

// Task timing parameters (ms)
#define FRAME_TIME 10  // base tick for scheduler, 10ms per frame

// Shared state variables
int ledSelect = 0;
int timing = 0;

unsigned long prevTime1 = 0;
unsigned long prevTime2 = 0;
long timeInt1 = 0;
long timeInt2 = 0;

int ledState1 = LOW;
int ledState2 = LOW;

bool flagPrint1 = true;
bool flagPrint2 = false;
bool flag3Check = true;
bool flag4Check = false;

// ---------- Task 1: Handle serial input ----------
void taskSerial() {
  if(flagPrint1) {
    Serial.println("What LED? (1 or 2)?");
    flag3Check = true;
    flagPrint1 = false;
  }

  if(flag3Check && Serial.available()) {
    ledSelect = Serial.parseInt();
    if(ledSelect != 0) {
      flagPrint2 = true;
      flag3Check = false;
    }
  }

  if(flagPrint2) {
    Serial.println("What interval (in msec)?");
```

```
    flagPrint2 = false;
  }

  if(!flag3Check && Serial.available()) {
    timing = Serial.parseInt();
    if(timing != 0) flag4Check = true;
  }

  if(flag4Check && ledSelect == 1) {
    timeInt1 = timing;
    if(timeInt1 < 0) ledState1 = LOW;
  }
  else if(flag4Check && ledSelect == 2) {
    timeInt2 = timing;
    if(timeInt2 < 0) ledState2 = LOW;
  }
  else if(flag4Check) {
    // invalid LED number
  }

  if(flag4Check) {
    flagPrint1 = true;
    flagPrint2 = false;
    flag3Check = true;
    flag4Check = false;
    ledSelect = 0;
    timing = 0;
  }
}

// ---------- Task 2: Blink LED1 ----------
void taskBlinkLED1() {
  unsigned long currentTime = millis();
  if(timeInt1 != 0 && (currentTime - prevTime1 >= timeInt1)) {
    prevTime1 = currentTime;
    ledState1 = (ledState1 == HIGH) ? LOW : HIGH;
    digitalWrite(LED1, ledState1);
  }
}

// ---------- Task 3: Blink LED2 ----------
void taskBlinkLED2() {
  unsigned long currentTime = millis();
  if(timeInt2 != 0 && (currentTime - prevTime2 >= timeInt2)) {
    prevTime2 = currentTime;
```

```
    ledState2 = (ledState2 == HIGH) ? LOW : HIGH;
    digitalWrite(LED2, ledState2);
  }
}

// ---------- Setup ----------
void setup() {
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  Serial.begin(9600);
  prevTime1 = millis();
  prevTime2 = millis();

}

typedef void (*TaskFunc)();
TaskFunc taskList[] = {
  taskSerial,
  taskBlinkLED1,
  taskBlinkLED2
};
const int NUM_TASKS = sizeof(taskList) / sizeof(TaskFunc);

// ---------- Cyclic Executive ----------
void loop() {
  unsigned long frameStart = millis();

  // Loop through all tasks in round-robin order
  for (int i = 0; i < NUM_TASKS; i++) {
    taskList[i]();  // Call each task function
  }

  // Enforce fixed frame duration
  while(millis() - frameStart < FRAME_TIME) {
    // idle until frame completes (keeps consistent timing)
  }
}
```

APPENDIX C. Testing Video

https://drive.google.com/file/d/1WtdlQBX-LNTQs0siEShGoddsu2ogLj-0/view?usp=sharing

END OF DOCUMENT