Engineering Educators Bringing the World Together
2025 ASEE Annual Conference & Exposition
Palais des congrès de Montréal, Montréal, QC • June 22–25, 2025
Paper ID #48474

Case Studies of ChapGPT for Embedded Systems Teaching

Dr. Guoping Wang, Purdue University Fort Wayne

Guoping Wang, Ph.D. is an Associate Professor in the Department of Electrical and Computer Engineering at Purdue University Fort Wayne. He earned his Ph.D. from the University of Oklahoma in 2003, following a Master's from Nanjing University and a Bachelor's from Tsinghua University. Dr. Wang's research interests include the Internet of Things, edge computing in machine learning, embedded systems, FPGA for DSP applications, and computer security. He has received numerous awards for teaching excellence and secured multiple grants for innovative projects. A senior member of IEEE, he actively contributes to the field through publications and conference presentations.

Case Studies of ChatGPT for Embedded Systems Teaching

Abstract

The rise of AI technology, particularly Generative AI, has significantly transformed the landscape of higher education. Generative AI, such as ChatGPT, has been extensively studied in fields like Computer Science to assess its effectiveness in enhancing learning. However, its impact on more specialized areas, such as bare-metal embedded systems, remains underexplored. Bare-metal embedded systems, which include hardware (e.g., microcontrollers, memory, input/output interfaces) and software (e.g., firmware drivers, real-time operating systems), present unique challenges compared to traditional areas of study.

ChatGPT has proven to be a useful tool in these embedded systems courses, helping students work through complex documentation, such as lengthy datasheets and technical manuals. It also assists with debugging, clarifying system architectures, and explaining hardware-software interactions in a way that makes difficult concepts more approachable. By breaking down complex topics and offering quick insights, ChatGPT can help students navigate the steep learning curve of embedded systems. Whether ChatGPT can generate solutions for embedded systems design problems remains an unanswered question.

In this paper, several case studies are examined to explore the role of ChatGPT in generating embedded systems solutions for lab practices. These case studies are based on actual student project assignments in a sequence of embedded systems courses, including 1 - Introduction to Microprocessors, 2 - Embedded Systems, and 3 - Real-Time Operating Systems. Our studies have found that though ChatGPT is a valuable tool in embedded systems teaching, it cannot replace the foundational knowledge essential for mastering embedded systems. Practical experience and a deep understanding of embedded systems' intricacies are still essential for success in this field. In the era of ChatGPT, instructors teaching embedded systems design should incorporate pop-up quizzes, lab check-outs, and other assessment methods instead of relying too heavily on traditional project assignments.

Introduction

The emergence of Generative AI tools such as ChatGPT has brought significant changes to higher education, offering new ways for students to access knowledge and interact with learning materials. ChatGPT serves as an on-demand tutor, offering personalized assistance, solving problems, and clarifying complex concepts. This creates a more flexible and accessible learning environment, allowing students to study at their own pace and receive real-time feedback, which is particularly beneficial in the context of online and self-directed learning.

Despite its benefits, the integration of ChatGPT into higher education presents challenges, particularly in terms of academic integrity and the ethical implications of AI-generated content. The potential for misuse, such as plagiarism or over-reliance on AI-generated solutions, is a growing concern. This has led institutions to rethink traditional assessments and establish guidelines for ethical AI use. As AI continues to evolve, higher education must balance the potential of these technologies with the need to maintain critical thinking, creativity, and intellectual integrity.

In previous research, numerous studies have explored the impacts of ChatGPT on various educational domains, including computer science, engineering, mathematical modeling, and construction management. For instance, a study [1] examines how ChatGPT can enhance the learning experience for computer science students by improving comprehension of complex concepts, boosting problem-solving skills, and increasing student engagement. Another study [2] discusses ChatGPT's role in engineering education, noting its potential benefits in generating relevant questions, providing instant feedback, and assisting with problem-solving and content creation. Additional research has explored the use of ChatGPT in teaching programming courses, particularly in C++ [3]. The study demonstrates that AI can help explain complex programming concepts, assist with homework, and improve student performance, with students showing better average scores and reduced failure rates. However, it stresses the importance of using AI tools to complement traditional teaching methods rather than replace them. Furthermore, in a study on ChatGPT's impact on engineering report writing [4], improvements were observed in grammar, organization, and conclusions, although there was little progress in enhancing visuals, equations, and references. The research highlights that while AI-generated content can sometimes include "unnatural" language or errors, students strongly support AI integration, believing it enhances both their learning experience and the quality of their work.

With the rise of generative AI (GenAI), tools like ChatGPT can significantly enhance embedded systems education by simplifying access to complex technical documentation. A typical ARM Cortex-M microcontroller (MCU) comes with hundreds of pages of datasheets and thousands of pages of firmware library notes. In the past, students spent considerable time searching through these documents to find relevant information, often struggling with dense technical jargon.

ChatGPT streamlines this process by quickly retrieving relevant details, summarizing key concepts, and explaining complex topics in a more digestible way. Instead of manually searching for how to configure a timer or set up an interrupt, students can get concise, targeted answers instantly. It also helps clarify ambiguities, provides example code, and suggests troubleshooting steps when projects don't work as expected.

Beyond documentation support, ChatGPT acts as a tutor, reinforcing key embedded concepts, guiding debugging strategies, and explaining system design trade-offs. While it cannot replace hands-on experience or critical thinking, it serves as a valuable tool that improves efficiency, enhances accessibility to knowledge, and accelerates learning in embedded systems courses.

However, due to the nature of embedded systems, which involve both hardware from different companies and software that heavily relies on firmware libraries provided by microcontroller manufacturers, how effective ChatGPT can be in generating solutions for project assignments remains a question to be answered. Unlike general-purpose programming languages such as C, C++, Python, and JavaScript, which have consistent syntax across different platforms, ChatGPT has proven to be very effective in generating project solutions. However, the design of embedded systems depends on the knowledge of datasheets and microcontroller (MCU) user manuals from various vendors. Firmware programming requires an understanding of register organization and the firmware driver libraries provided by these MCU vendors, and the IDE tools vary between companies. For example, some popular ARM Cortex-M4 IDE toolchains include Keil [5], IAR [6], STM32CubeIDE [7], MCUXpresso [8], ARM Studio [9],TI Code Composer Studio [10], and the open-source ARM GNU toolchain [11]. Some toolchains support a broad range of MCUs from multiple vendors, while others only support MCUs from their respective companies, and each toolchain has its own unique features.

In the following, several project assignments from three embedded systems sequence classes are studied using ChatGPT to generate solutions. The solutions are carefully analyzed and tested. The results demonstrate that while ChatGPT is highly effective in generating project solutions, it cannot be completely relied upon. We have also proposed teaching and learning methodologies to adapt our approach to embedded systems education in the era of ChatGPT.

Embedded Systems Sequence Courses

Computer Engineering students at our university are required to take a sequence of three embedded systems courses: ECE 36200 - Microprocessor Systems and Interfacing, ECE 46500 - Embedded Microprocessor and ECE 48500 - Embedded Real-Time Operating Systems. ECE 36200 - Microprocessor Systems and Interfacing, introduces students to the fundamentals of computer organization, focusing on ARM microprocessor architecture and assembly language programming. It covers topics such as instruction sets, digital and analog interfaces, and communication between microprocessors and peripherals. The course includes both theoretical and hands-on laboratory components, emphasizing practical experience with software applications and debugging techniques. By the end, students gain skills in solving engineering problems using assembly language and interfacing CPUs with various devices. ECE 46500 – Embedded Microprocessor focuses on the design of embedded systems using microcontrollers. The course covers both hardware and software perspectives, emphasizing applications like data acquisition, audio processing, and communication systems. Key topics include programming in C, interfacing with analog sensors, real-time debugging, and understanding communication protocols like UART. The course also explores system timing, noise analysis, and the use of IDE tools for embedded system design. ECE 48500 – Embedded Real-Time Operating Systems introduces students to embedded real-time operating systems (RTOS) with a focus on software development tasks, inter-task communication, synchronization, and network software. Students learn to program embedded systems using C and assembly, explore RTOS concepts like task scheduling and communication mechanisms (e.g., semaphores, mutexes), and design systems incorporating RTOS. The course also provides hands-on experience with RTOS development and practical case studies.

In all three courses, many projects are assigned to students to help them grasp the knowledge and practical skills in embedded systems. The hardware platform – TI EM4C123GXL Launchpad

[12] is used in all three classes. The Keil IDE toolchain is used to write the embedded system software. The TivaWare firmware peripheral library [13] is only used in ECE 48500 for RTOS system design. In both ECE 36200 and ECE 46500, students must write their firmware using MCU register access.

Project Solutions using ChatGPT

The case studies in this paper were selected based on actual student project assignments from a structured sequence of embedded systems courses: Introduction to Microprocessors, Embedded Systems, and Real-Time Operating Systems. The goal was to evaluate ChatGPT's effectiveness in assisting with learning and problem-solving across different levels of embedded systems education.

Projects were selected to represent typical challenges in embedded systems coursework, including interpreting microcontroller datasheets, configuring firmware, writing assembly language, and implementing RTOS tasks. To ensure a balanced assessment, assignments were taken from different points in the semester—some from early foundational exercises and others from more complex, end-of-term projects. The selected projects covered essential embedded system topics, including assembly language programming, peripheral interfacing, system timing, and RTOS-based task management.

For each case study, ChatGPT was prompted to generate solutions, which were then tested using the Keil IDE toolchain and deployed on the TI EK-TM4C123GXL Launchpad hardware platform. In cases where the initial output had errors or inefficiencies, additional iterative prompts were provided to refine the responses. The effectiveness of ChatGPT's solutions was evaluated based on correctness and applicability in an embedded systems setting, helping to determine both the advantages and limitations of using AI assistance in this domain.

Table 1 presents a summary of these case studies, briefly describing the project descriptions, the output from ChatGPT, and additional comments. Each case study is detailed in the following, along with links to corresponding ChatGPT-generated solutions.

ECE 36200 - Microprocessor Systems and Interfacing

Lab 02 - Practice of ARM Assembly Language

Write a properly commented and formatted Cortex-M3 assembly language program to compute the result of this expression:

$15 - 71 + (17 \div 2) + (5 \times 4) + 42 + 21$

- The expression, as written above without any simplification, must be evaluated from left to right, observing the standard algebraic order of operations.
- The only arithmetic instructions that may be used are addition and subtraction.
- Multiplication and division operations must be implemented using a combination of shift, addition, and/or subtraction instructions.

Table 1: Case Studies of ChatGPT

Class	Project	ChatGPT Results	Comments
ECE 36200	Assembly Language Practice	Nearly perfect solution from ChatGPT with detailed comments	Only minor modifications required
ECE 36200	System Tick and ADC Sampling	Working solution on TI hardware	Further prompt inputs improve the solution
ECE 46500	C Conversion Function	Working solution	Initial output used snprintf(); after prompting "do not use snprintf()," ChatGPT provided a correct alternative
ECE 46500	RC Controller PWM Width Measurement	Working solution	Correctly distinguishes between rising and falling edges
ECE 48500	FreeRTOS LED/Button Practice	Working code with a minor glitch due to mechanical switch behavior	After prompting "add debouncing for the switches," ChatGPT produced a perfect solution
ECE 48500	FreeRTOS UART/-Timer	Initial code was not thread-safe due to snprintf usage	After prompting "do not use snprintf," ChatGPT generated a correct, thread-safe solution

- Any constants needed must be moved into registers as values before using them. (Exception: For shifts to implement multiply or divide, the shift amount may be an immediate value in the instruction.)
- The values in the expression must be declared or obtained as follows:
  - 15, 71, 17, 5, 42, and 21: These are predefined constants saved in ROM as data.
  - 2 and 4: Use assembler EQU as DIV2 and MULT4, respectively, for the number of bits to shift.
- The result must be stored in ARM RAM memory.

Solutions from ChatGPT: ChatGPT is able to generate almost perfect solutions except a few lines of NVIC and stack pointer setup before running the main program. Besides, detailed explanations are also included for the solutions. The ChatGPT generated results can be found in List 1.

```
; Program to compute the result of the expression:
; 15-71 + (17/2) + (5x4) + 42+21

; Expression: ; 15-71 + (17/2) + (5x4) + 42+21

; The result will be stored in memory at RESULT_ADDR.

; Define constants using assembler directives
```

```
DIV2    EQU 1          ; Shift 1 bit to divide by 2 (equivalent to dividing by 2)
MULT4   EQU 2          ; Shift 2 bits to multiply by 4 (equivalent to multiplying by 4)

    AREA    CODE, READONLY
    THUMB
    EXPORT  __main      ; Export the main program entry

__main
    ; Load constants into registers
    LDR     R0, =ROM_DATA   ; Load the base address of data (ROM)

    ; Load values from ROM into registers
    LDR     R1, [R0, #0]    ; Load 15 into R1
    LDR     R2, [R0, #4]    ; Load 71 into R2
    LDR     R3, [R0, #8]    ; Load 17 into R3
    LDR     R4, [R0, #12]   ; Load 5 into R4
    LDR     R5, [R0, #16]   ; Load 42 into R5
    LDR     R6, [R0, #20]   ; Load 21 into R6

    ; Begin computation

    ; Step 1: 15 - 71
    SUBS    R1, R1, R2      ; R1 = R1 - R2 (15 - 71)

    ; Step 2: 17 / 2
    LSRS    R3, R3, #DIV2   ; Divide 17 by 2, R3 = R3 >> DIV2 (R3 = 17 / 2)

    ; Step 3: 5 x 4
    LSLS    R4, R4, #MULT4  ; Multiply 5 by 4, R4 = R4 << MULT4 (R4 = 5 * 4)

    ; Step 4: Add (17 / 2) to result of (15 - 71)
    ADDS    R1, R1, R3      ; R1 = R1 + R3

    ; Step 5: Add (5 x 4) to result
    ADDS    R1, R1, R4      ; R1 = R1 + R4

    ; Step 6: Add 42 to result
    ADDS    R1, R1, R5      ; R1 = R1 + R5

    ; Step 7: Add 21 to result
    ADDS    R1, R1, R6      ; R1 = R1 + R6

    ; Store the result in memory
    LDR     R7, =RESULT_ADDR   ; Load address of result storage
    STR     R1, [R7]           ; Store final result into RESULT_ADDR

    ; End of program, loop here forever
    B       .

    ; Define data section
    AREA    DATA, READONLY
ROM_DATA
```

```
    DCD     15                  ; Predefined constant 15
    DCD     71                  ; Predefined constant 71
    DCD     17                  ; Predefined constant 17
    DCD     5                   ; Predefined constant 5
    DCD     42                  ; Predefined constant 42
    DCD     21                  ; Predefined constant 21
    ; Define result storage
    AREA    DATA, READWRITE
RESULT_ADDR
    SPACE   4                   ; Reserve 4 bytes for the result
    END
```

Listing 1: ECE 36200 Lab02 ChatGPT Results

Lab 11 - System Tick and AD Sampling Write code for the TI TM4C123GXL LaunchPad to periodically start an ADC conversion every 10 ms. The ADC conversion should be triggered using the SysTick timer within an interrupt service routine. The program will poll the ADC registers to read the values. After every 16 sampled data points, the program will compute the average. DO NOT use the TivaWare firmware peripheral library.

Solutions from ChatGPT for Lab11: The first solution provided by ChatGPT from the Lab 11 description above is well-organized, commented, and explained. It passes testing using the Keil IDE tool on the TI TM4C123GXL Launchpad, assuming a 16 MHz system clock frequency in the first iteration.

In the second iteration, the following prompt is entered: "Add additional code for the PLL to generate a 50 MHz system clock from the external 16 MHz crystal on the TI TM4C123GXL Launchpad." The result from ChatGPT includes all the necessary functions with a clear structure. The generated codes can be found in List 2.

```
#include <stdint.h>
#include "tm4c123gh6pm.h"

#define SAMPLES_COUNT 16

volatile uint32_t adc_values[SAMPLES_COUNT];
volatile uint32_t sample_index = 0;
volatile uint32_t adc_sum = 0;
volatile uint32_t adc_average = 0;

void SysTick_Handler(void) {
    // Start ADC conversion by setting the start bit (SS3)
    ADC0_PSSI_R |= 0x0008; // Set bit 3 to start ADC conversion (SS3)
}

// Initialize SysTick Timer to generate interrupts every 10 ms
void SysTick_Init(void) {
    NVIC_ST_CTRL_R = 0;                // Disable SysTick during setup
    NVIC_ST_RELOAD_R = 160000 - 1;     // Reload value for 10 ms (assuming 16
   MHz clock)
    NVIC_ST_CURRENT_R = 0;             // Clear current value
    NVIC_ST_CTRL_R = 0x07;             // Enable SysTick with core clock and
   interrupts
```

```
}

// Initialize ADC
void ADC_Init(void) {
    SYSCTL_RCGCADC_R |= 0x01;         // Enable clock for ADC0
    SYSCTL_RCGCGPIO_R |= 0x10;        // Enable clock for port E (for ADC input)

    GPIO_PORTE_AFSEL_R |= 0x08;       // Enable alternate function on PE3
    GPIO_PORTE_DEN_R &= ~0x08;        // Disable digital function on PE3
    GPIO_PORTE_AMSEL_R |= 0x08;       // Enable analog function on PE3

    ADC0_ACTSS_R &= ~0x08;            // Disable sample sequencer 3 (SS3) during configuration
    ADC0_EMUX_R = (ADC0_EMUX_R & 0xFFFF0FFF); // Use software trigger for SS3
    ADC0_SSMUX3_R = 0x00;             // Set SS3 to sample AIN0 (PE3)
    ADC0_SSCTL3_R = 0x06;             // Configure SS3: single-ended, interrupt enabled
    ADC0_ACTSS_R |= 0x08;             // Enable SS3
}

// Function to poll ADC and read value
void ADC_Read(void) {
    if ((ADC0_RIS_R & 0x08) == 0) {
        return;  // Conversion not complete
    }

    adc_values[sample_index] = ADC0_SSFIFO3_R & 0xFFF;  // Read 12-bit ADC value
    ADC0_ISC_R = 0x08;  // Clear interrupt flag for SS3

    adc_sum += adc_values[sample_index];
    sample_index++;

    if (sample_index == SAMPLES_COUNT) {
        adc_average = adc_sum / SAMPLES_COUNT;  // Compute average after 16 samples
        adc_sum = 0;  // Reset sum
        sample_index = 0;  // Reset index
    }
}

int main(void) {
    SysTick_Init();   // Initialize SysTick Timer
    ADC_Init();       // Initialize ADC

    while (1) {
        ADC_Read();  // Poll ADC values in main loop
    }
}
```

Listing 2: ECE36200 Lab11 ChatGPT Results

ECE 46500 - Lab02 The first project tested using ChatGPT comes from a question from Lab02 as follows:

Write a function using C language with this prototype sDec2BCD() defined as:

```
void sDec2BCD(int32\_t hex, char *dec)

//----------------------------------------------

//     sDec2BCD() convert -9999 to 9999 signed number to a BCD string for LCD display
/* Input: hex: - a  signed integer, *dec, a point pointing to the BCD ASCII code.
Below are the testing vectors:
-100000  *.***
-10000   *.***
-9999     -9.999
-999      -0.999
-1          -0.001
 0           0.000
 123       0.123
1234       1.234
9999       9.999
10000      *.***
*/
//  hex    - 32-bit signed number, $-2^{31} - 2^{31}-1 $ \\
//  dec    - pointer to 6 8-bit (char) in ASCII code, as shown above\\
//  Output None
```

The first result from ChatGPT includes the body code of the main() and sDec2BCD() functions. Inside the main function, test vectors are also included. A library function from stdio.h, snprintf(), is used to convert the integer to a BCD ASCII string.

The second prompt is given to ChatGPT: “Do not use snprintf().” This time, the embedded C code is generated using only the stdio.h library. The ChatGPT results from both prompts can be found in the List 3.

```c
#include <stdint.h>
#include <stdio.h>

void sDec2BCD(int32_t hex, char *dec) {
    // Handle out-of-range values
    if (hex < -9999 || hex > 9999) {
        snprintf(dec, 6, "*.***");
        return;
    }
```

```

    // Format the number correctly with a fixed-point format
    if (hex < 0) {
        snprintf(dec, 6, "-%1d.%03d", (-hex) / 1000, (-hex) % 1000);
    } else {
        snprintf(dec, 6, " %1d.%03d", hex / 1000, hex % 1000);
    }
}

// Test function
int main() {
    int32_t test_values[] = {-100000, -10000, -9999, -999, -1, 0, 123, 1234, 9999, 10000};
    char output[6];

    for (int i = 0; i < 10; i++) {
        sDec2BCD(test_values[i], output);
        printf("%d -> %s\n", test_values[i], output);
    }

    return 0;
}



#include <stdint.h>
#include <stdio.h>

void sDec2BCD(int32_t hex, char *dec) {
    // Handle out-of-range values
    if (hex < -9999 || hex > 9999) {
        dec[0] = '*';
        dec[1] = '.';
        dec[2] = '*';
        dec[3] = '*';
        dec[4] = '*';
        dec[5] = '\0';
        return;
    }

    // Handle negative numbers
    int32_t abs_hex = (hex < 0) ? -hex : hex;
    int is_negative = (hex < 0);

    // Convert integer to individual digits
    int d4 = (abs_hex / 1000) % 10;
    int d3 = (abs_hex / 100) % 10;
    int d2 = (abs_hex / 10) % 10;
    int d1 = abs_hex % 10;

    // Format output string
    if (is_negative) {
        dec[0] = '-';
    } else {
```

```c
        dec[0] = (abs_hex >= 1000) ? ('0' + d4) : ' ';
    }

    dec[1] = (abs_hex >= 1000) ? '.' : ((abs_hex >= 100) ? ('0' + d3) : '0');
    dec[2] = (abs_hex >= 100) ? ('0' + d3) : ((abs_hex >= 10) ? ('0' + d2) : '0');
    dec[3] = (abs_hex >= 10) ? ('0' + d2) : '0';
    dec[4] = '0' + d1;
    dec[5] = '\0';  // Null-terminate the string
}

// Test function
int main() {
    int32_t test_values[] = {-100000, -10000, -9999, -999, -1, 0, 123, 1234, 9999, 10000};
    char output[6];

    for (int i = 0; i < 10; i++) {
        sDec2BCD(test_values[i], output);
        printf("%d -> %s\n", test_values[i], output);
    }

    return 0;
}
```

Listing 3: ECE46500 Lab02 ChatGPT Results

ECE 46500 - Remote Controller RC Signal Width Measurement

Lab assignment: Write a C code to find the width of a 1.0ms to 2.5 ms Remote Control signal using Tivaware firmware peripheral driver

The result, generated using ChatGPT 4.0, correctly distinguishes between the rising and falling edges of the RC signal. The results can be found at List 4.

The generated results are properly commented with detailed explanations.

```c
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_ints.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"

// Global variables to store the capture times
volatile uint32_t g_ui32StartTime = 0;
volatile uint32_t g_ui32EndTime = 0;
volatile uint32_t g_ui32PulseWidth = 0;

void TimerCaptureIntHandler(void)
{
```

```c
    uint32_t ui32Status;

    // Get the timer interrupt status and clear the interrupt flag
    ui32Status = TimerIntStatus(TIMER0_BASE, true);
    TimerIntClear(TIMER0_BASE, ui32Status);

    // Check if the interrupt was caused by a rising or falling edge
    if (ui32Status & TIMER_CAPA_EVENT)
    {
        // If it's a rising edge, store the start time
        g_ui32StartTime = TimerValueGet(TIMER0_BASE, TIMER_A);

        // Switch to capture the falling edge
        TimerControlEvent(TIMER0_BASE, TIMER_A, TIMER_EVENT_NEG_EDGE);
    }
    else if (ui32Status & TIMER_CAPB_EVENT)
    {
        // If it's a falling edge, store the end time and calculate the pulse
   width
        g_ui32EndTime = TimerValueGet(TIMER0_BASE, TIMER_A);

        // Calculate the pulse width (taking into account timer rollover if
   needed)
        if (g_ui32EndTime > g_ui32StartTime)
        {
            g_ui32PulseWidth = g_ui32EndTime - g_ui32StartTime;
        }
        else
        {
            // Timer rollover case
            g_ui32PulseWidth = (0xFFFFFFFF - g_ui32StartTime) + g_ui32EndTime;
        }

        // Switch back to capture the rising edge for the next pulse
        TimerControlEvent(TIMER0_BASE, TIMER_A, TIMER_EVENT_POS_EDGE);
    }
}

int main(void)
{
    // Set the system clock to 50 MHz
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
   SYSCTL_XTAL_16MHZ);

    // Enable peripherals for GPIO port and Timer
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);

    // Wait for the peripherals to be ready
    while (!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOB)) {}
    while (!SysCtlPeripheralReady(SYSCTL_PERIPH_TIMER0)) {}

    // Configure PB6 as T0CCP0 (Timer capture pin)
    GPIOPinConfigure(GPIO_PB6_T0CCP0);
```

```c
    GPIOPinTypeTimer(GPIO_PORTB_BASE, GPIO_PIN_6);

    // Configure the timer for edge time capture
    TimerConfigure(TIMER0_BASE, TIMER_CFG_SPLIT_PAIR | TIMER_CFG_A_CAP_TIME_UP);

    // Configure Timer0A to capture the rising edge initially
    TimerControlEvent(TIMER0_BASE, TIMER_A, TIMER_EVENT_POS_EDGE);

    // Enable interrupts for Timer0A capture
    TimerIntEnable(TIMER0_BASE, TIMER_CAPA_EVENT);
    IntEnable(INT_TIMER0A);
    IntMasterEnable();

    // Enable Timer0A
    TimerEnable(TIMER0_BASE, TIMER_A);

    while (1)
    {
        // Main loop - the pulse width will be updated by the interrupt handler
        // g_ui32PulseWidth contains the pulse width in timer ticks

        // To convert timer ticks to time (in microseconds):
        // Pulse width (in microseconds) = (g_ui32PulseWidth / (SysCtlClockGet() / 1000000));
    }
}
```

Listing 4: ECE46500 RC PWM Width Measurement ChatGPT Results

ECE 48500 - Embedded Real-Time Operating Systems

In the testing using ChatGPT for ECE 48500 project assignments, the following assignments in FreeRTOS are fed into ChatGPT. Each question is entered in sequence and the generated code are compiled, tested on TI TM4C123GXL lanchpad and can runs smoothly with small changes.

ECE 48500 - Lab02a - FreeRTOS Practice Using FreeRTOS and the TivaWare firmware peripheral driver library on the TI TM4C123GXL Launchpad, design a small RTOS system where the Red LED toggles when the left key is pressed, and the Blue LED toggles when the right key is pressed.

Notes:

1. Start with the FreeRTOS demo project and ensure that you create two tasks: an LED task and a Switch task.
2. The Switch task should be configured with a higher priority.
3. The LED task should be configured with a lower priority.
4. Create a queue to pass the switch press messages (left or right push button) to the LED task.

5. Suggested task priorities can be found in (priorities.h).

Here are the suggested priorities of these two tasks: (priorities.h)

```
#ifndef __PRIORITIES\_H__
#define __PRIORITIES\_H__

//*****************************************************
//
// The priorities of the various tasks.
//
//*****************************************************
#define PRIORITY\_SWITCH\_TASK    2
#define PRIORITY\_LED\_TASK       1

#endif // __PRIORITIES_H__
```

ChatGPT is able to generate working code using FreeRTOS. The code can be compiled and run on the TI EK-TM4C123GXL Launchpad with a minor glitch. The glitch occurs because the buttons are mechanical switches that need to be debounced. Mechanical switch bounce happens due to the imperfect nature of mechanical switches. When a switch is pressed or released, there is a brief period where the electrical signal is unstable. List 5 provides the generated codes.

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "priorities.h"

// Define constants for the buttons
#define LEFT_BUTTON  GPIO_PIN_4
#define RIGHT_BUTTON GPIO_PIN_0

// Define LED pins
#define RED_LED GPIO_PIN_1
#define BLUE_LED GPIO_PIN_2

// Declare the queue handle
xQueueHandle g_pQueue;

// Function prototypes
void LEDTask(void *pvParameters);
void SwitchTask(void *pvParameters);
void SetupHardware(void);

int main(void) {
    // Set up system clock and hardware peripherals
```

```c
    SetupHardware();

    // Create the queue with space for 10 unsigned integers
    g_pQueue = xQueueCreate(10, sizeof(uint32_t));

    if (g_pQueue != NULL) {
        // Create the Switch Task
        xTaskCreate(SwitchTask, "Switch Task", 128, NULL, PRIORITY_SWITCH_TASK, NULL);

        // Create the LED Task
        xTaskCreate(LEDTask, "LED Task", 128, NULL, PRIORITY_LED_TASK, NULL);

        // Start the FreeRTOS scheduler
        vTaskStartScheduler();
    }

    // The program should never reach here
    while (1) {
    }
}

// LED Task: Reads the queue and toggles LEDs based on switch presses
void LEDTask(void *pvParameters) {
    uint32_t button_press;

    while (1) {
        // Wait for data from the queue
        if (xQueueReceive(g_pQueue, &button_press, portMAX_DELAY) == pdPASS) {
            // Check which button was pressed and toggle the corresponding LED
            if (button_press == LEFT_BUTTON) {
                GPIOPinWrite(GPIO_PORTF_BASE, RED_LED,
                             GPIOPinRead(GPIO_PORTF_BASE, RED_LED) ^ RED_LED);
            } else if (button_press == RIGHT_BUTTON) {
                GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED,
                             GPIOPinRead(GPIO_PORTF_BASE, BLUE_LED) ^ BLUE_LED);
            }
        }
    }
}

// Switch Task: Detects button presses and sends messages to the LED task
void SwitchTask(void *pvParameters) {
    uint32_t left_button_state, right_button_state;
    uint32_t left_button_last_state = LEFT_BUTTON;
    uint32_t right_button_last_state = RIGHT_BUTTON;

    while (1) {
        // Read the current state of the buttons
        left_button_state = GPIOPinRead(GPIO_PORTF_BASE, LEFT_BUTTON);
        right_button_state = GPIOPinRead(GPIO_PORTF_BASE, RIGHT_BUTTON);

        // Detect if the left button was pressed (falling edge detection)
```

```c
        if ((left_button_last_state != 0) && (left_button_state == 0)) {
            uint32_t message = LEFT_BUTTON;
            xQueueSend(g_pQueue, &message, portMAX_DELAY);
        }

        // Detect if the right button was pressed (falling edge detection)
        if ((right_button_last_state != 0) && (right_button_state == 0)) {
            uint32_t message = RIGHT_BUTTON;
            xQueueSend(g_pQueue, &message, portMAX_DELAY);
        }

        // Save the current button states for edge detection
        left_button_last_state = left_button_state;
        right_button_last_state = right_button_state;

        // Delay to avoid rapid polling
        vTaskDelay(50 / portTICK_RATE_MS);
    }
}

// Function to set up hardware peripherals
void SetupHardware(void) {
    // Set the system clock to 50 MHz
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
    SYSCTL_XTAL_16MHZ);

    // Enable peripherals for Port F (for LEDs and buttons)
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    // Unlock PF0 (for right button)
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= GPIO_PIN_0;
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;

    // Configure the LEDs (PF1 for red, PF2 for blue) as output
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, RED_LED | BLUE_LED);

    // Configure the switches (PF0 for right, PF4 for left) as input with pull
    -up resistors
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, LEFT_BUTTON | RIGHT_BUTTON);
    GPIOPadConfigSet(GPIO_PORTF_BASE, LEFT_BUTTON | RIGHT_BUTTON,
    GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
}
```

Listing 5: ECE48500 Lab02a FreeRTOS LED/Button

A further prompt, ”Add debouncing functions to the switches,” was sent to ChatGPT, and this time, the result came back remarkably well, as shown in List 6.

```c
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
```

```c
#include "driverlib/gpio.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "priorities.h"

// Define constants for the buttons
#define LEFT_BUTTON  GPIO_PIN_4
#define RIGHT_BUTTON GPIO_PIN_0

// Define LED pins
#define RED_LED GPIO_PIN_1
#define BLUE_LED GPIO_PIN_2

// Declare the queue handle
xQueueHandle g_pQueue;

// Function prototypes
void LEDTask(void *pvParameters);
void SwitchTask(void *pvParameters);
void SetupHardware(void);
bool DebounceButton(uint32_t button, uint32_t port, uint32_t delay_ms);

int main(void) {
    // Set up system clock and hardware peripherals
    SetupHardware();

    // Create the queue with space for 10 unsigned integers
    g_pQueue = xQueueCreate(10, sizeof(uint32_t));

    if (g_pQueue != NULL) {
        // Create the Switch Task
        xTaskCreate(SwitchTask, "Switch Task", 128, NULL, PRIORITY_SWITCH_TASK
    , NULL);

        // Create the LED Task
        xTaskCreate(LEDTask, "LED Task", 128, NULL, PRIORITY_LED_TASK, NULL);

        // Start the FreeRTOS scheduler
        vTaskStartScheduler();
    }

    // The program should never reach here
    while (1) {
    }
}

// LED Task: Reads the queue and toggles LEDs based on switch presses
void LEDTask(void *pvParameters) {
    uint32_t button_press;

    while (1) {
        // Wait for data from the queue
        if (xQueueReceive(g_pQueue, &button_press, portMAX_DELAY) == pdPASS) {
```

```
            // Check which button was pressed and toggle the corresponding LED
            if (button_press == LEFT_BUTTON) {
                GPIOPinWrite(GPIO_PORTF_BASE, RED_LED,
                             GPIOPinRead(GPIO_PORTF_BASE, RED_LED) ^ RED_LED);
            } else if (button_press == RIGHT_BUTTON) {
                GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED,
                             GPIOPinRead(GPIO_PORTF_BASE, BLUE_LED) ^ BLUE_LED);
            }
        }
    }
}

// Switch Task: Detects button presses, debounces them, and sends messages to the LED task
void SwitchTask(void *pvParameters) {
    while (1) {
        // Check if the left button is pressed and debounced
        if (DebounceButton(LEFT_BUTTON, GPIO_PORTF_BASE, 50)) {
            uint32_t message = LEFT_BUTTON;
            xQueueSend(g_pQueue, &message, portMAX_DELAY);
        }

        // Check if the right button is pressed and debounced
        if (DebounceButton(RIGHT_BUTTON, GPIO_PORTF_BASE, 50)) {
            uint32_t message = RIGHT_BUTTON;
            xQueueSend(g_pQueue, &message, portMAX_DELAY);
        }

        // Delay between checking button states
        vTaskDelay(50 / portTICK_RATE_MS);
    }
}

// Debouncing function for buttons
bool DebounceButton(uint32_t button, uint32_t port, uint32_t delay_ms) {
    // Read the current state of the button
    uint32_t currentState = GPIOPinRead(port, button);

    // Check if the button is pressed (low state, active low button)
    if (currentState == 0) {
        // Wait for the debounce delay
        vTaskDelay(delay_ms / portTICK_RATE_MS);

        // Read the state again to confirm the press
        if (GPIOPinRead(port, button) == 0) {
            return true; // Button press confirmed
        }
    }

    return false; // No valid press detected
}

// Function to set up hardware peripherals
```

```c
void SetupHardware(void) {
    // Set the system clock to 50 MHz
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
   SYSCTL_XTAL_16MHZ);

    // Enable peripherals for Port F (for LEDs and buttons)
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    // Unlock PF0 (for right button)
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= GPIO_PIN_0;
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;

    // Configure the LEDs (PF1 for red, PF2 for blue) as output
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, RED_LED | BLUE_LED);

    // Configure the switches (PF0 for right, PF4 for left) as input with pull
   -up resistors
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, LEFT_BUTTON | RIGHT_BUTTON);
    GPIOPadConfigSet(GPIO_PORTF_BASE, LEFT_BUTTON | RIGHT_BUTTON,
   GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
}
```

Listing 6: ECE48500 Lab02a FreeRTOS LED/Button with Switch Debouncing

In part 2 of Lab 2a, the following assignment was given to ChatGPT: ”Continue with the previous code and add the following functionality:

- When a message is sent in the switch_task function or received in the LED_task function, a message is also sent to the UART0 terminal.
- To protect the shared UART0 resource between these two tasks, a mutex needs to be created.
- Here are the four messages that will be sent to the serial terminal when each event occurs:

```c
#define LEFTSEND       "LEFT SEND\r\n"   // when the left
   button is pressed/released in the switch task
#define RIGHTSEND      "RIGHT SEND\r\n"  // when the right
   button is pressed/released in the switch task
#define LEFTRECEIVE    "LEFT RECV\r\n"   // when the left
   button message is received in the LED task
#define RIGHTRECEIVE   "RIGHT RECV\r\n"  // when the right
   button message is received in the LED task
```

This time ChatGPT is able to generates fully functional code, as shown in List 7.

```c
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
```

```c
#include "driverlib/uart.h"
#include "driverlib/pin_map.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"
#include "priorities.h"

// Define constants for the buttons
#define LEFT_BUTTON  GPIO_PIN_4
#define RIGHT_BUTTON GPIO_PIN_0

// Define LED pins
#define RED_LED GPIO_PIN_1
#define BLUE_LED GPIO_PIN_2

// Define UART messages
#define LEFTSEND     "LEFT SEND\r\n"
#define RIGHTSEND    "RIGT SEND\r\n"
#define LEFTRECEIVE "LEFT RECV\r\n"
#define RIGHTRECEIVE "RIGT RECV\r\n"

// Declare the queue handle and mutex handle
xQueueHandle g_pQueue;
xSemaphoreHandle g_pUARTMutex;

// Function prototypes
void LEDTask(void *pvParameters);
void SwitchTask(void *pvParameters);
void SetupHardware(void);
void UARTSend(const char *message);
bool DebounceButton(uint32_t button, uint32_t port, uint32_t delay_ms);

int main(void) {
    // Set up system clock and hardware peripherals
    SetupHardware();

    // Create the queue with space for 10 unsigned integers
    g_pQueue = xQueueCreate(10, sizeof(uint32_t));

    // Create a mutex to protect UART0 access
    g_pUARTMutex = xSemaphoreCreateMutex();

    if (g_pQueue != NULL && g_pUARTMutex != NULL) {
        // Create the Switch Task
        xTaskCreate(SwitchTask, "Switch Task", 128, NULL, PRIORITY_SWITCH_TASK
    , NULL);

        // Create the LED Task
        xTaskCreate(LEDTask, "LED Task", 128, NULL, PRIORITY_LED_TASK, NULL);

        // Start the FreeRTOS scheduler
        vTaskStartScheduler();
    }
```

```c

    // The program should never reach here
    while (1) {
    }
}

// LED Task: Reads the queue and toggles LEDs based on switch presses
void LEDTask(void *pvParameters) {
    uint32_t button_press;

    while (1) {
        // Wait for data from the queue
        if (xQueueReceive(g_pQueue, &button_press, portMAX_DELAY) == pdPASS) {
            // Take the UART mutex before sending a message
            if (xSemaphoreTake(g_pUARTMutex, portMAX_DELAY)) {
                if (button_press == LEFT_BUTTON) {
                    UARTSend(LEFTRECEIVE); // Send "LEFT RECV\r\n"
                    GPIOPinWrite(GPIO_PORTF_BASE, RED_LED,
                                 GPIOPinRead(GPIO_PORTF_BASE, RED_LED) ^ RED_LED);
                } else if (button_press == RIGHT_BUTTON) {
                    UARTSend(RIGHTRECEIVE); // Send "RIGT RECV\r\n"
                    GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED,
                                 GPIOPinRead(GPIO_PORTF_BASE, BLUE_LED) ^ BLUE_LED);
                }
                xSemaphoreGive(g_pUARTMutex); // Release the mutex
            }
        }
    }
}

// Switch Task: Detects button presses, debounces them, and sends messages to the LED task
void SwitchTask(void *pvParameters) {
    while (1) {
        // Check if the left button is pressed and debounced
        if (DebounceButton(LEFT_BUTTON, GPIO_PORTF_BASE, 50)) {
            uint32_t message = LEFT_BUTTON;

            // Send the "LEFT SEND" message to UART
            if (xSemaphoreTake(g_pUARTMutex, portMAX_DELAY)) {
                UARTSend(LEFTSEND); // Send "LEFT SEND\r\n"
                xSemaphoreGive(g_pUARTMutex);
            }

            xQueueSend(g_pQueue, &message, portMAX_DELAY);
        }

        // Check if the right button is pressed and debounced
        if (DebounceButton(RIGHT_BUTTON, GPIO_PORTF_BASE, 50)) {
            uint32_t message = RIGHT_BUTTON;

            // Send the "RIGHT SEND" message to UART
```

```
            if (xSemaphoreTake(g_pUARTMutex, portMAX_DELAY)) {
                UARTSend(RIGHTSEND); // Send "RIGT SEND\r\n"
                xSemaphoreGive(g_pUARTMutex);
            }

            xQueueSend(g_pQueue, &message, portMAX_DELAY);
        }

        // Delay between checking button states
        vTaskDelay(50 / portTICK_RATE_MS);
    }
}

// UART send function to send strings over UART0
void UARTSend(const char *message) {
    while (*message) {
        UARTCharPut(UART0_BASE, *message++);
    }
}

// Debouncing function for buttons
bool DebounceButton(uint32_t button, uint32_t port, uint32_t delay_ms) {
    // Read the current state of the button
    uint32_t currentState = GPIOPinRead(port, button);

    // Check if the button is pressed (low state, active low button)
    if (currentState == 0) {
        // Wait for the debounce delay
        vTaskDelay(delay_ms / portTICK_RATE_MS);

        // Read the state again to confirm the press
        if (GPIOPinRead(port, button) == 0) {
            return true; // Button press confirmed
        }
    }

    return false; // No valid press detected
}

// Function to set up hardware peripherals
void SetupHardware(void) {
    // Set the system clock to 50 MHz
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
   SYSCTL_XTAL_16MHZ);

    // Enable peripherals for Port F (for LEDs and buttons) and UART0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    // Unlock PF0 (for right button)
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= GPIO_PIN_0;
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;
```

```

    // Configure the LEDs (PF1 for red, PF2 for blue) as output
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, RED_LED | BLUE_LED);

    // Configure the switches (PF0 for right, PF4 for left) as input with pull-up resistors
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, LEFT_BUTTON | RIGHT_BUTTON);
    GPIOPadConfigSet(GPIO_PORTF_BASE, LEFT_BUTTON | RIGHT_BUTTON, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);

    // Configure UART0 for communication with the terminal
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

    // Enable UART0 FIFO and interrupts
    UARTFIFOEnable(UART0_BASE);
    UARTEnable(UART0_BASE);
}
```

Listing 7: ECE48500 Lab02a FreeRTOS UART/Timer

ECE 48500 - Lab02b - FreeRTOS Practice

In Lab02b, the following assignment is sent to ChatGPT, Continue to add the following functions: add the following functions:

- Configure Timer0 to count periodically every 20 ms, with interrupts enabled and priority 1 (very high).
- Inside the Timer0 ISR, read ADC channel 1. After 2 seconds with 100 samples, calculate the average value of channel 1's ADC, turn on the Blue LED, and then signal the ADCSample task.
- For the ADCSample task, wait for the 2-second signal from the Timer0 ISR, and then convert the ADC average value to the 6.2d format (xxxx.xx).
- Send this average value to the UAR0 serial terminal.
- After that, delay the task for 500 ms and turn off the Blue LED.

The priorities of these three tasks are defined as:

```
#define PRIORITY_SWITCH_TASK    3
#define PRIORITY_LED_TASK       2
#define PRIORITY_ADCSAMPLE_TASK 4
```

As expected, ChatGPT is able to significantly enhance working code by providing detailed comments and explanations. However, it uses the snprintf() function and does not recognize that snprintf() may not be thread-safe, as shown in List 8.

```c
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/uart.h"
#include "driverlib/pin_map.h"
#include "driverlib/timer.h"
#include "driverlib/adc.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"
#include "priorities.h"

// Define constants for the buttons and LED
#define LEFT_BUTTON  GPIO_PIN_4
#define RIGHT_BUTTON GPIO_PIN_0
#define RED_LED      GPIO_PIN_1
#define BLUE_LED     GPIO_PIN_2

// Define UART messages
#define LEFTSEND    "LEFT SEND\r\n"
#define RIGHTSEND   "RIGT SEND\r\n"
#define LEFTRECEIVE "LEFT RECV\r\n"
#define RIGHTRECEIVE "RIGT RECV\r\n"

// Define task priorities
#define PRIORITY_SWITCH_TASK    3
#define PRIORITY_LED_TASK       2
#define PRIORITY_ADCSAMPLE_TASK 4

// Global variables
xQueueHandle g_pQueue;
xSemaphoreHandle g_pUARTMutex;
xSemaphoreHandle g_pADCSampleSignal;
volatile uint32_t g_adcAccumulator = 0;
volatile uint32_t g_adcSampleCount = 0;
volatile uint32_t g_adcAverage = 0;

// Function prototypes
void LEDTask(void *pvParameters);
void SwitchTask(void *pvParameters);
void ADCSampleTask(void *pvParameters);
void SetupHardware(void);
void UARTSend(const char *message);
bool DebounceButton(uint32_t button, uint32_t port, uint32_t delay_ms);
void Timer0ISR(void);

// Timer configuration for 20 ms interrupts
void Timer0Config(void) {
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
```

```c

    // Set the timer period for 20 ms
    uint32_t ui32Period = (SysCtlClockGet() / 50);  // 20ms = 50Hz
    TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period - 1);

    // Enable Timer interrupts
    IntPrioritySet(INT_TIMER0A, 0x20);  // Priority 1
    IntEnable(INT_TIMER0A);
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

    TimerEnable(TIMER0_BASE, TIMER_A);
}

// Timer0 ISR: Accumulates ADC samples every 20 ms
void Timer0ISR(void) {
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

    uint32_t adcValue;

    // Trigger ADC conversion on channel 1
    ADCProcessorTrigger(ADC0_BASE, 1);

    // Wait for conversion to complete
    while (!ADCIntStatus(ADC0_BASE, 1, false)) {}
    ADCIntClear(ADC0_BASE, 1);
    ADCSequenceDataGet(ADC0_BASE, 1, &adcValue);

    // Accumulate the ADC samples
    g_adcAccumulator += adcValue;
    g_adcSampleCount++;

    // If 100 samples (2 seconds) have been collected
    if (g_adcSampleCount >= 100) {
        // Calculate the average
        g_adcAverage = g_adcAccumulator / g_adcSampleCount;

        // Reset the accumulator and sample count
        g_adcAccumulator = 0;
        g_adcSampleCount = 0;

        // Turn on the Blue LED
        GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED, BLUE_LED);

        // Signal the ADCSample task
        xSemaphoreGiveFromISR(g_pADCSampleSignal, pdFALSE);
    }
}

// UART send function to send strings over UART0
void UARTSend(const char *message) {
    while (*message) {
        UARTCharPut(UART0_BASE, *message++);
    }
}
```

```c

// ADCSample Task: Waits for the 2-second signal and converts the ADC average to 6.2d format
void ADCSampleTask(void *pvParameters) {
    char uartMessage[20];
    while (1) {
        // Wait for the signal from the Timer0 ISR
        if (xSemaphoreTake(g_pADCSampleSignal, portMAX_DELAY) == pdTRUE) {
            // Convert the average ADC value to 6.2d format
            float adcValue = (float)g_adcAverage * 3.3 / 4096;  // Assuming 12-bit ADC with 3.3V reference
            snprintf(uartMessage, sizeof(uartMessage), "AVG: %04d.%02d\r\n", (int)adcValue, (int)((adcValue - (int)adcValue) * 100));

            // Take the UART mutex and send the formatted string
            if (xSemaphoreTake(g_pUARTMutex, portMAX_DELAY)) {
                UARTSend(uartMessage);
                xSemaphoreGive(g_pUARTMutex);
            }

            // Delay the task for 500 ms and turn off the Blue LED
            vTaskDelay(500 / portTICK_RATE_MS);
            GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED, 0);
        }
    }
}

// Switch Task: Detects button presses and sends messages to the LED task
void SwitchTask(void *pvParameters) {
    while (1) {
        // Check if the left button is pressed and debounced
        if (DebounceButton(LEFT_BUTTON, GPIO_PORTF_BASE, 50)) {
            uint32_t message = LEFT_BUTTON;

            // Send the "LEFT SEND" message to UART
            if (xSemaphoreTake(g_pUARTMutex, portMAX_DELAY)) {
                UARTSend(LEFTSEND);  // Send "LEFT SEND\r\n"
                xSemaphoreGive(g_pUARTMutex);
            }

            xQueueSend(g_pQueue, &message, portMAX_DELAY);
        }

        // Check if the right button is pressed and debounced
        if (DebounceButton(RIGHT_BUTTON, GPIO_PORTF_BASE, 50)) {
            uint32_t message = RIGHT_BUTTON;

            // Send the "RIGHT SEND" message to UART
            if (xSemaphoreTake(g_pUARTMutex, portMAX_DELAY)) {
                UARTSend(RIGHTSEND);  // Send "RIGT SEND\r\n"
                xSemaphoreGive(g_pUARTMutex);
            }

            xQueueSend(g_pQueue, &message, portMAX_DELAY);
```

```c
        }

        // Delay between checking button states
        vTaskDelay(50 / portTICK_RATE_MS);
    }
}

// LED Task: Reads the queue and toggles LEDs based on switch presses
void LEDTask(void *pvParameters) {
    uint32_t button_press;

    while (1) {
        // Wait for data from the queue
        if (xQueueReceive(g_pQueue, &button_press, portMAX_DELAY) == pdPASS) {
            // Take the UART mutex before sending a message
            if (xSemaphoreTake(g_pUARTMutex, portMAX_DELAY)) {
                if (button_press == LEFT_BUTTON) {
                    UARTSend(LEFTRECEIVE);  // Send "LEFT RECV\r\n"
                    GPIOPinWrite(GPIO_PORTF_BASE, RED_LED, GPIOPinRead(GPIO_PORTF_BASE, RED_LED) ^ RED_LED);
                } else if (button_press == RIGHT_BUTTON) {
                    UARTSend(RIGHTRECEIVE);  // Send "RIGT RECV\r\n"
                    GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED, GPIOPinRead(GPIO_PORTF_BASE, BLUE_LED) ^ BLUE_LED);
                }
                xSemaphoreGive(g_pUARTMutex);  // Release the mutex
            }
        }
    }
}

// Debouncing function for buttons
bool DebounceButton(uint32_t button, uint32_t port, uint32_t delay_ms) {
    uint32_t currentState = GPIOPinRead(port, button);
    if (currentState == 0) {
        vTaskDelay(delay_ms / portTICK_RATE_MS);
        if (GPIOPinRead(port, button) == 0) {
            return true;
        }
    }
    return false;
}

// Function to set up hardware peripherals
void SetupHardware(void) {
    // Set the system clock to 50 MHz
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);

    // Enable peripherals for Port F and ADC0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);

    // Enable UART0 for terminal communication
```

```
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
   UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE);

    // Configure the buttons and LEDs
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, RED_LED | BLUE_LED);
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, LEFT_BUTTON | RIGHT_BUTTON);
    GPIOPadConfigSet(GPIO_PORTF_BASE, LEFT_BUTTON | RIGHT_BUTTON,
   GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);

    // Configure ADC channel 1 (AIN1, PE2)
    GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_2);
    ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
    ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_CH1 | ADC_CTL_IE |
   ADC_CTL_END);
    ADCSequenceEnable(ADC0_BASE, 1);

    // Configure Timer0
    Timer0Config();
}
```

Listing 8: ECE48500 Lab02b FreeRTOS Generated Codes using snprintf Function

In the prompt, it was specified: "Do not use the snprintf() function as it may not be thread-safe in an RTOS system." After further prompting, ChatGPT addresses the concern of avoiding snprintf() in a FreeRTOS system (since it may not be thread-safe) by manually formatting the ADC value into the xxxx.xx format using basic string manipulation. This ensures that the formatting operation does not introduce thread-safety issues.

The updated version of the code replaces the snprintf() function with manual conversion to the required format. The generated code is shown in List 9.

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/uart.h"
#include "driverlib/pin_map.h"
#include "driverlib/timer.h"
#include "driverlib/adc.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"
#include "priorities.h"

// Define constants for the buttons and LED
#define LEFT_BUTTON  GPIO_PIN_4
```

```c
#define RIGHT_BUTTON GPIO_PIN_0
#define RED_LED      GPIO_PIN_1
#define BLUE_LED     GPIO_PIN_2

// Define UART messages
#define LEFTSEND    "LEFT SEND\r\n"
#define RIGHTSEND   "RIGT SEND\r\n"
#define LEFTRECEIVE "LEFT RECV\r\n"
#define RIGHTRECEIVE "RIGT RECV\r\n"

// Define task priorities
#define PRIORITY_SWITCH_TASK    3
#define PRIORITY_LED_TASK       2
#define PRIORITY_ADCSAMPLE_TASK 4

// Global variables
xQueueHandle g_pQueue;
xSemaphoreHandle g_pUARTMutex;
xSemaphoreHandle g_pADCSampleSignal;
volatile uint32_t g_adcAccumulator = 0;
volatile uint32_t g_adcSampleCount = 0;
volatile uint32_t g_adcAverage = 0;

// Function prototypes
void LEDTask(void *pvParameters);
void SwitchTask(void *pvParameters);
void ADCSampleTask(void *pvParameters);
void SetupHardware(void);
void UARTSend(const char *message);
bool DebounceButton(uint32_t button, uint32_t port, uint32_t delay_ms);
void Timer0ISR(void);

// Timer configuration for 20 ms interrupts
void Timer0Config(void) {
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);

    // Set the timer period for 20 ms
    uint32_t ui32Period = (SysCtlClockGet() / 50);  // 20ms = 50Hz
    TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period - 1);

    // Enable Timer interrupts
    IntPrioritySet(INT_TIMER0A, 0x20);  // Priority 1
    IntEnable(INT_TIMER0A);
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

    TimerEnable(TIMER0_BASE, TIMER_A);
}

// Timer0 ISR: Accumulates ADC samples every 20 ms
void Timer0ISR(void) {
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

    uint32_t adcValue;
```

```

    // Trigger ADC conversion on channel 1
    ADCProcessorTrigger(ADC0_BASE, 1);

    // Wait for conversion to complete
    while (!ADCIntStatus(ADC0_BASE, 1, false)) {}
    ADCIntClear(ADC0_BASE, 1);
    ADCSequenceDataGet(ADC0_BASE, 1, &adcValue);

    // Accumulate the ADC samples
    g_adcAccumulator += adcValue;
    g_adcSampleCount++;

    // If 100 samples (2 seconds) have been collected
    if (g_adcSampleCount >= 100) {
        // Calculate the average
        g_adcAverage = g_adcAccumulator / g_adcSampleCount;

        // Reset the accumulator and sample count
        g_adcAccumulator = 0;
        g_adcSampleCount = 0;

        // Turn on the Blue LED
        GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED, BLUE_LED);

        // Signal the ADCSample task
        xSemaphoreGiveFromISR(g_pADCSampleSignal, pdFALSE);
    }
}

// UART send function to send strings over UART0
void UARTSend(const char *message) {
    while (*message) {
        UARTCharPut(UART0_BASE, *message++);
    }
}

// Convert ADC average value to string in 6.2d format
void FormatADCValue(uint32_t adcValue, char *buffer) {
    // Assuming a 12-bit ADC with 3.3V reference
    float voltage = (float)adcValue * 3.3 / 4096;

    // Convert to integer values for "xxxx.xx" format
    int integerPart = (int)voltage;
    int fractionalPart = (int)((voltage - integerPart) * 100);

    // Manually format the string
    buffer[0] = (integerPart / 1000) + '0';  // Thousands place
    buffer[1] = ((integerPart / 100) % 10) + '0';  // Hundreds place
    buffer[2] = ((integerPart / 10) % 10) + '0';  // Tens place
    buffer[3] = (integerPart % 10) + '0';  // Ones place
    buffer[4] = '.';  // Decimal point
    buffer[5] = (fractionalPart / 10) + '0';  // Tenths place
    buffer[6] = (fractionalPart % 10) + '0';  // Hundredths place
```

```
    buffer[7] = '\r';
    buffer[8] = '\n';
    buffer[9] = '\0';  // Null terminator
}

// ADCSample Task: Waits for the 2-second signal and converts the ADC average to 6.2d format
void ADCSampleTask(void *pvParameters) {
    char uartMessage[20];
    while (1) {
        // Wait for the signal from the Timer0 ISR
        if (xSemaphoreTake(g_pADCSampleSignal, portMAX_DELAY) == pdTRUE) {
            // Format the ADC average value into the buffer
            FormatADCValue(g_adcAverage, uartMessage);

            // Take the UART mutex and send the formatted string
            if (xSemaphoreTake(g_pUARTMutex, portMAX_DELAY)) {
                UARTSend(uartMessage);
                xSemaphoreGive(g_pUARTMutex);
            }

            // Delay the task for 500 ms and turn off the Blue LED
            vTaskDelay(500 / portTICK_RATE_MS);
            GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED, 0);
        }
    }
}

// Switch Task: Detects button presses and sends messages to the LED task
void SwitchTask(void *pvParameters) {
    while (1) {
        // Check if the left button is pressed and debounced
        if (DebounceButton(LEFT_BUTTON, GPIO_PORTF_BASE, 50)) {
            uint32_t message = LEFT_BUTTON;

            // Send the "LEFT SEND" message to UART
            if (xSemaphoreTake(g_pUARTMutex, portMAX_DELAY)) {
                UARTSend(LEFTSEND);  // Send "LEFT SEND\r\n"
                xSemaphoreGive(g_pUARTMutex);
            }

            xQueueSend(g_pQueue, &message, portMAX_DELAY);
        }

        // Check if the right button is pressed and debounced
        if (DebounceButton(RIGHT_BUTTON, GPIO_PORTF_BASE, 50)) {
            uint32_t message = RIGHT_BUTTON;

            // Send the "RIGHT SEND" message to UART
            if (xSemaphoreTake(g_pUARTMutex, portMAX_DELAY)) {
                UARTSend(RIGHTSEND);  // Send "RIGT SEND\r\n"
                xSemaphoreGive(g_pUARTMutex);
            }

```

```
            xQueueSend(g_pQueue, &message, portMAX_DELAY);
        }

        // Delay between checking button states
        vTaskDelay(50 / portTICK_RATE_MS);
    }
}

// LED Task: Reads the queue and toggles LEDs based on switch presses
void LEDTask(void *pvParameters) {
    uint32_t button_press;

    while (1) {
        // Wait for data from the queue
        if (xQueueReceive(g_pQueue, &button_press, portMAX_DELAY) == pdPASS) {
            // Take the UART mutex before sending a message
            if (xSemaphoreTake(g_pUARTMutex, portMAX_DELAY)) {
                if (button_press == LEFT_BUTTON) {
                    UARTSend(LEFTRECEIVE);  // Send "LEFT RECV\r\n"
                    GPIOPinWrite(GPIO_PORTF_BASE, RED_LED, GPIOPinRead(
    GPIO_PORTF_BASE, RED_LED) ^ RED_LED);
                } else if (button_press == RIGHT_BUTTON) {
                    UARTSend(RIGHTRECEIVE);  // Send "RIGT RECV\r\n"
                    GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED, GPIOPinRead(
    GPIO_PORTF_BASE, BLUE_LED) ^ BLUE_LED);
                }
                xSemaphoreGive(g_pUARTMutex);  // Release the mutex
            }
        }
    }
}

// Debouncing function for buttons
bool DebounceButton(uint32_t button, uint32_t port, uint32_t delay_ms) {
    uint32_t currentState = GPIOPinRead(port, button);
    if (currentState == 0) {
        vTaskDelay(delay_ms / portTICK_RATE_MS);
        if (GPIOPinRead(port, button) == 0) {
            return true;
```

Listing 9: ECE48500 Lab02b FreeRTOS Thread-Safe Generated Code

Discussions

In the era of Generative AI, such as ChatGPT, even embedded systems design—which requires specific knowledge of hardware and software—can benefit from AI-generated solutions. However, these results must be carefully analyzed and tested. Essential knowledge and practical skills remain crucial, even in the age of AI. It has become challenging to assign students traditional projects, as ChatGPT can replace much of the students' work. Therefore, innovative approaches need to be explored.

The findings in this paper highlight both the advantages and limitations of using ChatGPT in embedded systems teaching. While it helps students navigate documentation, debug code, and grasp complex topics, it falls short in key areas—it can't interact with hardware, sometimes provides inaccurate information, and lacks real-world context. These gaps emphasize the importance of hands-on learning and critical thinking. To address this, educators can incorporate more in-class debugging exercises, mini-projects, and assignments that challenge students to critically evaluate AI-generated solutions. Teaching AI literacy—guiding students to critically assess ChatGPT's responses—ensures they use it as a tool rather than a substitute for learning. By balancing AI assistance with practical experience, students can build both technical expertise and problem-solving skills.

Assessing students' learning in embedded system design in the era of ChatGPT requires innovative approaches that go beyond traditional methods, ensuring that students learn effectively and apply their knowledge creatively. Assessing students' achievements remains challenging, particularly in distinguishing between the following three categories:

- Students' work without any AI assistance
- Students' work with some AI assistance
- Students' work with lots of AI assistance

With these considerations in mind, how can educators design assignments that leverage AI while maintaining rigorous learning outcomes? The key is to integrate AI and modern tools thoughtfully, ensuring students develop essential skills rather than passively relying on automation.

To address these challenges, we propose several pedagogical strategies that balance AI's benefits while ensuring students develop fundamental embedded systems expertise. Here are some strategies for crafting effective programming assignments in embedded systems—assignments that encourage learning, foster problem-solving, and account for AI's role in the development process.

1. **Pop Quiz**

   Assignment: Students will be given unannounced, short tests designed to quickly evaluate their understanding of embedded systems material. These pop quiz questions should be brief, typically 5-10 questions, to quickly gauge students' grasp, such as, "How would you connect a sensor to an MCU using a CAN bus?"

   Objective: Evaluate their understanding of key concepts such as microcontroller functionality, sensor interfacing, and communication protocols, in a real-time, practical manner.

2. **Lab Checkout Questions**

   Assignment: During the hands-on lab checkout, students will be asked specific questions related to that lab practice to assess whether they have grasped and understood the necessary knowledge. Their answers will be graded, and similar questions may appear in future exams.

Objective: Evaluate students' understanding of key concepts such as microcontroller functionality, sensor interfacing, and communication protocols for that lab practice.

Note: We have adopted this approach in ECE 46500 - Embedded Microprocessor Lab and found it to be effective. By incorporating verbal assessments during lab checkout, we ensure that students engage with the material beyond simply completing the lab exercises. This method encourages them to think critically, articulate their understanding, and apply concepts in real-time scenarios. Additionally, it provides instructors with immediate feedback on common misconceptions, allowing for timely intervention. As a result, students develop a deeper comprehension of embedded systems principles, better preparing them for exams and real-world applications.

3. **Mini-Project in Classroom**

   Assignment: Carefully designed, quick, focused mini-projects are assigned to students, which they can complete within an hour in the classroom. Each project is designed to build foundational skills and confidence with embedded concepts.

   Objective: These mini-projects are hands-on and achievable within an hour, even without the assistance of AI, providing a manageable introduction to key concepts in embedded systems. Each mini-project allows students to learn essential concepts while developing practical skills in designing, testing, and debugging embedded systems.

4. **Code Refactoring and Optimization**
   Assignment: Provide students with a fully functioning, yet inefficient embedded system code (e.g., code that controls an IoT sensor network or motor control in a robot). Their task is to refactor and optimize the code to improve performance, memory usage, or energy efficiency.

   Objective: Assess students' ability to optimize embedded system code for constraints like speed or memory.

   AI Integration: ChatGPT can assist students in suggesting optimizations, but students must document how they identified and applied each optimization. Their own explanations of the changes should demonstrate independent understanding.

5. **AI-Assisted Code Debugging Challenge**
   Assignment: Give students buggy embedded system code, such as code with timing issues in real-time systems, incorrect sensor data handling, or faulty communication protocols. Their task is to debug the code and provide a report on how they fixed the issues.
   Objective: Test students' debugging and problem-solving skills, especially in low-level programming environments.
   AI Integration: Students can use ChatGPT to get suggestions or debugging tips, but they must submit a step-by-step explanation of how they solved each issue and what specific role AI played in the process.

6. **Hardware-Software Co-Design Project**
   Assignment: Students are tasked with developing both the software and hardware interface for a small embedded system (e.g., an automatic door system, temperature control system,

or smart light control using a microcontroller). The code should interface with sensors and actuators.
Objective: Assess the ability to write embedded system code that interacts with hardware components (sensors, actuators) and manage real-time constraints.
AI Integration: ChatGPT can help students with code structure or offer advice on hardware interfacing, but students must explain their hardware choices and the specifics of their custom implementation. For example, they should justify why they used particular sensor libraries and how they configured interrupt handling.

7. **Code Review and AI Critique**
   Assignment: Students are asked to write embedded system code for a given task (e.g., controlling a motor or reading data from a sensor). After submission, they must use ChatGPT to review their own code, focusing on efficiency, potential bugs, and best practices.
   Objective: Improve students' ability to critically evaluate their own code and identify areas for improvement.
   AI Integration: The final submission includes the student's original code, ChatGPT's feedback, and a report on what changes they made based on AI suggestions. Students must justify why they agreed or disagreed with the AI feedback.

Summary

The paper explores the integration of ChatGPT in embedded systems education. It highlights the challenges of using AI tools in specialized fields like bare-metal embedded systems, where hardware and software intricacies are critical. ChatGPT proves beneficial in assisting students with tasks such as code generation and understanding complex documentation. However, practical experience and foundational knowledge remain indispensable for success. The paper presents case studies from courses like Introduction to Microprocessors, Embedded Systems, and Real-Time Operating Systems, illustrating how ChatGPT complements but cannot replace traditional learning methods. Educators are encouraged to adapt their teaching strategies to ensure effective learning in the AI era. As educators continue to adapt, incorporating AI thoughtfully into curricula will be key to preparing students for an evolving engineering landscape.

References

[1] L. Kloub and A. Gupta, "Chatgpt in computer science education: Exploring benefits, challenges, and ethical considerations," in *2024 ASEE North East Section*, no. 10.18260/1-2–45758. Fairfield, Connecticut: ASEE Conferences, April 2024, https://peer.asee.org/45758.

[2] S. Onal, "Exploring the potential benefits and risks of chatgpt in engineering education," in *2023 IL-IN Section Conference*, no. 10.18260/1-2-118-45100. Edwardsville, IL: ASEE Conferences, December 2023, https://peer.asee.org/45100.

[3] A. Kavianpour, "Teaching programming languages by two teachers: Instructor and chatgpt," in *2024 ASEE Annual Conference & Exposition*, no. 10.18260/1-2–48065. Portland, Oregon: ASEE Conferences, June 2024, https://peer.asee.org/48065.

[4] R. D. Manteufel and A. Karimi, "Student use of chatgpt to write an engineering report," in *2024 ASEE Annual Conference & Exposition*, no. 10.18260/1-2–48021. Portland, Oregon: ASEE Conferences, June 2024, https://peer.asee.org/48021.

[5] Keil MDK. [Online]. Available: https://developer.arm.com/ToolsandSoftware/KeilMDK

[6] IAR Embedded Workbench for ARM. [Online]. Available: https://www.iar.com/products/architectures/arm/iar-embedded-workbench-for-arm/

[7] STM32CubeIDE Integrated Development Environments for STM32. [Online]. Available: https://www.st.com/en/development-tools/stm32cubeide.html

[8] MCUXpresso Integrated Development Environment. [Online]. Available: https://www.nxp.com/design/design-center/software/development-software/mcuxpresso-software-and-tools-/mcuxpresso-integrated-development-environment-ide:MCUXpresso-IDE

[9] Arm Development Studio. [Online]. Available: https://developer.arm.com/ToolsandSoftware/ArmDevelopmentStudio

[10] TI Code Composer Studio Integrated Development Environment. [Online]. Available: https://www.ti.com/tool/download/CCSTUDIO/12.8.0

[11] ARM GNU Toolchain. [Online]. Available: https://developer.arm.com/ToolsandSoftware/GNUToolchain

[12] EK-TM4C123GXL Launchpad from TI. [Online]. Available: https://www.ti.com/tool/EK-TM4C123GXL

[13] SW-TM4C TivaWare for C Series. [Online]. Available: https://www.ti.com/tool/SW-TM4C