



**DesignNews**

Machine Learning in Microcontrollers

# DAY 2: Writing Embedded Software with ChatGPT and Open.AI

Sponsored by



## Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Group Chat’ by maximizing the chat widget in your dock.

## THE SPEAKER



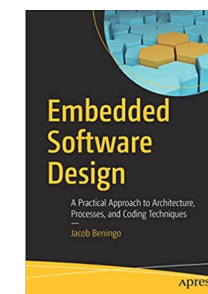
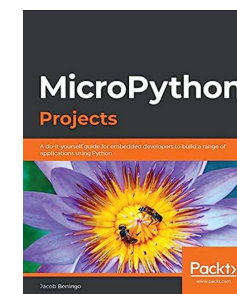
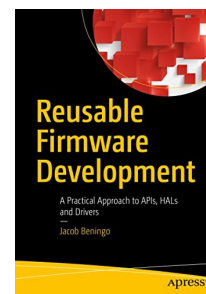
# Jacob Beningo

Visit 'Lecturer Profile'

## Beningo Embedded Group - President

Focus: Embedded Software Consulting and Training

Specializes in creating and promoting embedded software **excellence** in businesses around the world.



Blogs for:

- DesignNews.com
- Embedded.com
- EmbeddedRelated.com
- MLRelated.com

Visit [www.beningo.com](http://www.beningo.com) to learn more ...

## Course Sessions

- AI and ML for Microcontrollers
- **Writing Embedded Software with ChatGPT and Open.AI**
- Tools for Machine Learning in Microcontrollers
- Training a Model for the STM32
- Deploying Machine Learning Models

1

# AI Tools for Embedded Software

# AI Tools for Embedded Software

```
int main() {
    if (wiringPiSetup() == -1) {
        fprintf(stderr, "Failed to initialize wiringPi.\n");
        return 1;
    }

    pinMode(LED_PIN, OUTPUT);

    while (1) {
        digitalWrite(LED_PIN, HIGH);
        delay(BLINK_DELAY_MS);
        digitalWrite(LED_PIN, LOW);
        delay(BLINK_DELAY_MS);

        printf("Hello, world!\n");
    }

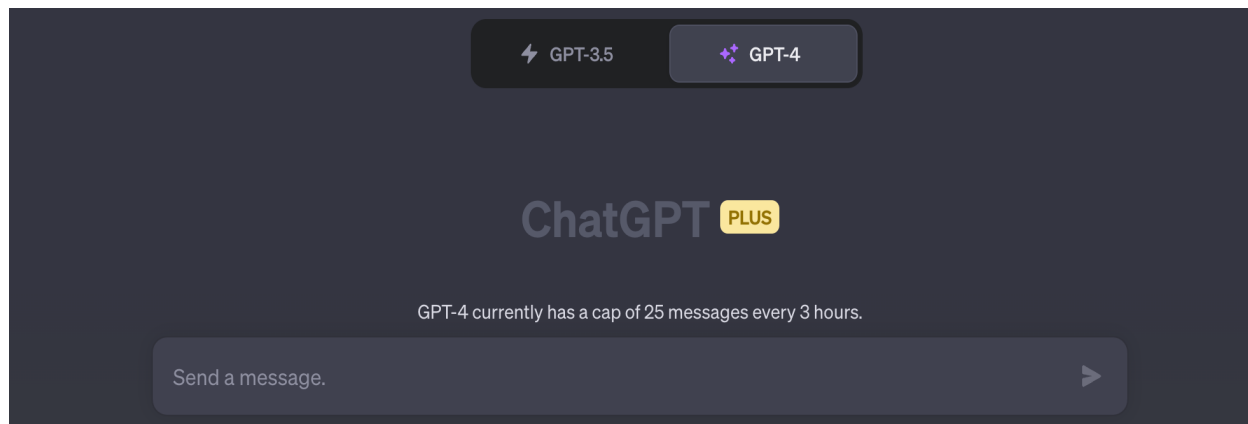
    return 0;
}
```

## Github Copilot

- Powered by OpenAI Codex
  - Generative pretrained model
- Suggests code to the developer
- Emulates pair programming with AI
- 26% - 40% percent of code suggestions are accepted

<https://github.com/features/copilot>

# AI Tools for Embedded Software



## ChatGPT

- GPT-3.5 or GPT-4 LLM
- ChatGPT is an AI language model by OpenAI.
- It creates human-like text from given prompts.
- GPT-4 architecture enables context understanding and response generation.
- Trained on diverse internet text, specifics are unknown.
- No access to personal data unless explicitly provided.
- Versatile: drafts emails, writes, tutors, translates, simulates, codes.
- Despite sophistication, ChatGPT has limitations.
- It doesn't understand like humans, may err.
- Aim is to continuously refine and improve it.

# AI Tools for Embedded Software

AI Tools can help embedded developers in several ways:

- Debugging code by suggesting potential causes and solutions.
- Writing embedded software documentation for specific hardware components.
- Providing code snippets for common embedded system tasks.
- Assisting in real-time troubleshooting of embedded systems issues.
- Offering guidance on best practices in embedded software development.
- Generating skeleton code for communication protocols (SPI, I2C, UART).
- Reviewing existing code and suggesting improvements or refactoring.
- Simulating responses from different hardware components for testing.
- Tutoring in advanced embedded systems concepts and programming languages.
- Assisting in project management tasks for embedded systems development.



Do you trust AI generated code?

- Yes
- No
- Maybe so . . .

## 2

# Generating a USART Interface using ChatGPT

# Generating a USART Interface using ChatGPT

JA We are going to write an embedded software interface in C for a usart peripheral. We are going to use the Google C++ coding standard. Don't use built-in types such as char, but use fixed width integer types from stdint.h. Please create an initial draft of the usart interface

## Analysis:

- I like the configuration structure for initialization.
- The interface is a bit verbose and repetitive.
  - SendByte, SendBytes, etc
- Interface support only a single USART device
- Not a fan of variables with \_. Would rather use camel case
- The use of const in parameters is a good practice
  - USART\_Init should really have a const config
- There's no considerations for interrupts or DMA through callback functions

Note: You may get different results. I've done this exercise several times.

```
#ifndef USART_H
#define USART_H

#include <stdint.h>

// USART configuration struct
typedef struct {
    uint32_t baud_rate; // Baud rate for USART communication
    uint8_t data_bits; // Number of data bits (5-9)
    uint8_t stop_bits; // Number of stop bits (1 or 2)
    uint8_t parity; // Parity mode (0: none, 1: odd, 2: even)
} USART_Config;

// Initialize USART with the given configuration
void USART_Init(USART_Config config);

// Send a single byte over USART
void USART_SendByte(uint8_t byte);

// Receive a single byte from USART
uint8_t USART_ReceiveByte(void);

// Send a byte array over USART
void USART_SendBytes(const uint8_t* data, uint32_t size);

// Receive a byte array from USART
void USART_ReceiveBytes(uint8_t* buffer, uint32_t size);

// Send a null-terminated string over USART
void USART_SendString(const uint8_t* str);

// Receive a null-terminated string from USART
void USART_ReceiveString(uint8_t* buffer, uint32_t buffer_size);

#endif // USART_H
```

Regenerate response

# Generating a USART Interface using ChatGPT

JA

This is a good start. Can you update the usart interface based on the following changes 1) Can you simplify the interface so that there are not so many repetitive functions 2) The interface only supports one device. Can you add support for multiple usart devices 3) Instead of using \_ in parameter variable names, please use camel. For example, buffer\_size would become bufferSize

## Analysis:

- The interface still has the string functions which I would not include in a low-level driver. This is a library function.
- The device has been added but coupled with USART\_Config.
- Each function does have a device pointer, but these should be const
- There are a lot of useless comments in the code
- There's no considerations for interrupts or DMA through callback functions

Note: You may get different results. I've done this exercise several times.

```
#ifndef USART_H
#define USART_H

#include <stdint.h>

// USART configuration struct
typedef struct {
    uint32_t baudRate;    // Baud rate for USART communication
    uint8_t dataBits;    // Number of data bits (5-9)
    uint8_t stopBits;    // Number of stop bits (1 or 2)
    uint8_t parity;      // Parity mode (0: none, 1: odd, 2: even)
} USART_Config;

// USART device struct
typedef struct {
    uint32_t usartId;    // Identifier for the USART device
    USART_Config config; // Configuration for USART communication
} USART_Device;

// Initialize USART device with the given configuration
void USART_Init(USART_Device* device);

// Send data over USART
void USART_Send(USART_Device* device, const uint8_t* data, uint32_t size);

// Receive data from USART
void USART_Receive(USART_Device* device, uint8_t* buffer, uint32_t bufferSize);

// Send a null-terminated string over USART
void USART_SendString(USART_Device* device, const uint8_t* str);

// Receive a null-terminated string from USART
void USART_ReceiveString(USART_Device* device, uint8_t* buffer, uint32_t bufferSize);

#endif // USART_H
```

Regenerate response

# Generating a USART Interface using ChatGPT

JA Update the latest version of the interface with the following: 1) remove the string functions from the interface. 2) Remove the comments above each function and structure. 3) USART\_Device is not a useful structure. Remove it. 4) Update USART\_Device to be an enum of usartDevice\_t that lists the available usart interfaces.

## Analysis:

- The interface now looks slim and cleaner!
- We have a typedef list of USARTs to interact with.
- Unfortunately, USART\_Config comments were removed.
- Not a big fan of USART\_COUNT. I prefer USART\_DEVICE\_ID\_MAX.
- Would prefer usartDevice\_t be usartDeviceID\_t.
- There's no considerations for interrupts or DMA through callback functions

Note: You may get different results. I've done this exercise several times.

```
c Copy code
#ifndef USART_H
#define USART_H

#include <stdint.h>

typedef enum {
    USART_1,
    USART_2,
    USART_3,
    // Add more USART interfaces as needed
    USART_COUNT
} usartDevice_t;

typedef struct {
    uint32_t baudRate;
    uint8_t dataBits;
    uint8_t stopBits;
    uint8_t parity;
} USART_Config;

void USART_Init(usartDevice_t device, USART_Config config);
void USART_Send(usartDevice_t device, const uint8_t* data, uint32_t size);
void USART_Receive(usartDevice_t device, uint8_t* buffer, uint32_t bufferSize);

#endif // USART_H
```

# Generating a USART Interface using ChatGPT

JA Update the latest interface version with the following changes: 1) change usartDevice\_t to usartDeviceID\_t 2) Change USART\_COUNT to USART\_MAX 3) Change USART\_Cong to usartConfig\_t 4) When passing the usartConfig\_t to USART\_Init, the parameter should be const

## Analysis:

- We're zeroing in on a reasonable interface!
- Our changes have been implemented.

## What's missing?:

- The functions don't return error codes
- We are ignoring the dependency inversion rule
- There's no considerations for interrupts or DMA through callback functions

Note: You may get different results. I've done this exercise several times.

```
c Copy code
#ifndef USART_H
#define USART_H

#include <stdint.h>

typedef enum {
    USART_1,
    USART_2,
    USART_3,
    // Add more USART interfaces as needed
    USART_MAX
} usartDeviceID_t;

typedef struct {
    uint32_t baudRate;
    uint8_t dataBits;
    uint8_t stopBits;
    uint8_t parity;
} usartConfig_t;

void USART_Init(usartDeviceID_t device, const usartConfig_t config);
void USART_Send(usartDeviceID_t device, const uint8_t* data, uint32_t size);
void USART_Receive(usartDeviceID_t device, uint8_t* buffer, uint32_t bufferSize);

#endif // USART_H
```

# Generating a USART Interface using ChatGPT

JA Update the latest interface with the following changes: 1) Each function in the interface should return an error code of type `usartError_t` 2) The functions should be wrapped in a typedef struct where each function of the interface is a function pointer. 3) Add a callback function to the `usartConfig_t` structure.

## Analysis:

- At this point, we have a pretty well-defined interface. There are still minor changes that should be fixed by hand:
  - The callback function probably would have void parameters
  - The USART\_Receive function pointer has a `bufferSize` parameter which should really just be `size`
  - The header file is `USART_H` when it should be `HAL_USART` or `USART_INTERFACE`
  - The AI decided to define an extern `usartInterface` variable that can be removed.

Note: You may get different results. I've done this exercise several times.

```
#ifndef USART_H
#define USART_H

#include <stdint.h>

typedef enum {
    USART_1,
    USART_2,
    USART_3,
    // Add more USART interfaces as needed
    USART_MAX
} usartDeviceID_t;

typedef enum {
    USART_OK,
    USART_ERROR
} usartError_t;

typedef struct {
    uint32_t baudRate;
    uint8_t dataBits;
    uint8_t stopBits;
    uint8_t parity;
    void (*callback)(usartError_t error); // Callback function for error handling
} usartConfig_t;

typedef struct {
    usartError_t (*USART_Init)(usartDeviceID_t device, const usartConfig_t* config);
    usartError_t (*USART_Send)(usartDeviceID_t device, const uint8_t* data, uint8_t size);
    usartError_t (*USART_Receive)(usartDeviceID_t device, uint8_t* buffer, uint8_t bufferSize);
} usartInterface_t;

extern const usartInterface_t USART_Interface;

#endif // USART_H
```

Regenerate response

What do you think about using ChatGPT in this manner?

- Worthless, I can do it faster by hand
- An interesting experiment, but would still do it myself
- Fascinating, I can see and would use this to accelerate my own development
- I'm all in! The AI Overlord can now write all my code!
- Other



**3**

# Generating SPI and I2C Interfaces using ChatGPT

# Generating SPI and I2C Interfaces using ChatGPT

Best Practices for generating code:

- 1) Be very specific one what you want. The more information you feed it, the closer it will get on the first try.
- 2) Use iterative feedback to get to the results you want
- 3) Feed in examples to more closely get the results that you want.

# Generating SPI and I2C Interfaces using ChatGPT

JA Based on the usart interface design and coding style, can you create an interface for a SPI peripheral?

## Analysis:

- With an example, it generated a pretty decent Spi interface!
- There may be a few tweaks needed, (I'd just simplify the interface to SPI\_Transfer), but it could be used as is!

```
#ifndef SPI_INTERFACE_H
#define SPI_INTERFACE_H

#include <stdint.h>

typedef enum {
    SPI_1,
    SPI_2,
    SPI_3,
    // Add more SPI interfaces as needed
    SPI_MAX
} spiDeviceID_t;

typedef enum {
    SPI_OK,
    SPI_ERROR
} spiError_t;

typedef struct {
    uint32_t clockSpeed;
    uint8_t dataMode;
    uint8_t bitOrder;
    void (*callback)(void);
} spiConfig_t;

typedef struct {
    spiError_t (*SPI_Init)(spiDeviceID_t device, const spiConfig_t* config);
    spiError_t (*SPI_Transmit)(spiDeviceID_t device, const uint8_t* data, uint
    spiError_t (*SPI_Receive)(spiDeviceID_t device, uint8_t* buffer, uint32_t
    spiError_t (*SPI_TransmitReceive)(spiDeviceID_t device, const uint8_t* txD
} spiInterface_t;

#endif // SPI_INTERFACE_H
```

Regenerate response

# Generating SPI and I2C Interfaces using ChatGPT

JA based on the spi interface design and coding style, can you create an interface for a I2C peripheral?

## Analysis:

- With an example, it generated a pretty decent I2C interface!
- There may be some small adjustments needed, but once I fed in an example, it took ~5 seconds to get the results.

```
#ifndef I2C_INTERFACE_H
#define I2C_INTERFACE_H

#include <stdint.h>

typedef enum {
    I2C_1,
    I2C_2,
    I2C_3,
    // Add more I2C interfaces as needed
    I2C_MAX
} i2cDeviceID_t;

typedef enum {
    I2C_OK,
    I2C_ERROR
} i2cError_t;

typedef struct {
    uint32_t clockSpeed;
    uint8_t address;
    void (*callback)(void);
} i2cConfig_t;

typedef struct {
    i2cError_t (*I2C_Init)(i2cDeviceID_t device, const i2cConfig_t* config);
    i2cError_t (*I2C_Write)(i2cDeviceID_t device, const uint8_t* data, uint32_t size);
    i2cError_t (*I2C_Read)(i2cDeviceID_t device, uint8_t* buffer, uint32_t size);
    i2cError_t (*I2C_WriteRead)(i2cDeviceID_t device, const uint8_t* txData, uint8_t* rxData, uint32_t size);
} i2cInterface_t;

#endif // I2C_INTERFACE_H
```

How appealing is using a generative AI tool to you to augment your design and development?

- Not at all
- It's interesting, and I may try it
- I'll definitely be using AI to help accelerate my development
- Other

3

# Implementing the USART Interface

# Implementing the USART Interface (for STM32L475)

JA

Using the usart interface, can you show an example implementation for the STM32L475?

## Analysis:

- Dang! It took about 20 seconds and I received 190 lines of code.
- It's not a full implementation. There are comments throughout where the developer can add implementation details such as in an interrupt.
- A quick glance through the code, and most of it seems reasonable . . .

```
#include "usart_interface.h"
#include "stm32l475xx.h"

// USART registers
#define USART1_BASE_ADDR (USART1_BASE)
#define USART2_BASE_ADDR (USART2_BASE)
#define USART3_BASE_ADDR (USART3_BASE)

// USART control register (CR1) bits
#define USART_CR1_TE (USART_CR1_TE_Msk) // Transmitter enable
#define USART_CR1_RE (USART_CR1_RE_Msk) // Receiver enable
#define USART_CR1_UE (USART_CR1_UE_Msk) // USART enable

// USART status register (SR) bits
#define USART_SR_TXE (USART_SR_TXE_Msk) // Transmit data register
#define USART_SR_RXNE (USART_SR_RXNE_Msk) // Read data register not

// USART interrupt enable register (CR1) bits
#define USART_CR1_TXEIE (USART_CR1_TXEIE_Msk) // Transmit data register
#define USART_CR1_RXNEIE (USART_CR1_RXNEIE_Msk) // Read data register not

// USART interrupt flag register (SR) bits
#define USART_SR_TXE (USART_SR_TXE_Msk) // Transmit data register
#define USART_SR_RXNE (USART_SR_RXNE_Msk) // Read data register not

// USART IRQ numbers
#define USART1_IRQ (USART1_IRQn)
#define USART2_IRQ (USART2_IRQn)
#define USART3_IRQ (USART3_IRQn)

// USART IRQ handlers
#define USART1_IRQHandler USART1_IRQHandler
#define USART2_IRQHandler USART2_IRQHandler
#define USART3_IRQHandler USART3_IRQHandler

// USART peripheral registers
```

Regenerate response

Continue generating

## Implementing the USART Interface (for STM32L475)

```
// USART peripheral pointers
USART_TypeDef *const USARTs[] = {
    USART1,
    USART2,
    USART3
};

// USART interrupt handlers
void (*const USART_IRQHandlers[])(void) = {
    USART1_IRQHandler,
    USART2_IRQHandler,
    USART3_IRQHandler
};
```

```
// USART interrupt service routine
void USART_IRQHandler(USART_TypeDef *device)
{
    USART_TypeDef *usart = USARTs[device];

    if ((usart->SR & USART_SR_TXE) && (usart->CR1 & USART_CR1_TXEIE)) {
        // Handle transmit data register empty interrupt
        // Implement your transmit logic here
    }

    if ((usart->SR & USART_SR_RXNE) && (usart->CR1 & USART_CR1_RXNEIE)) {
        // Handle read data register not empty interrupt
        // Implement your receive logic here
    }
}
```



## Implementing the USART Interface (for STM32L475)

```
// USART initialization
USART_Error_t USART_Init(USART_DeviceID_t device, const USART_Config_t* config)
{
    if (device >= USART_MAX || config == NULL) {
        return USART_ERROR;
    }

    // Enable USART clock
    if (device == USART_1) {
        RCC->APB2ENR |= RCC_APB2ENR_USART1EN;
    } else if (device == USART_2) {
        RCC->APB1ENR1 |= RCC_APB1ENR1_USART2EN;
    } else if (device == USART_3) {
        RCC->APB1ENR1 |= RCC_APB1ENR1_USART3EN;
    }

    USART_TypeDef *usart = USARTs[device];

    // Configure USART CR1 register
    usart->CR1 = 0;
    usart->CR1 |= (config->baudRate << 4);
    usart->CR1 |= (config->
```

```
        // Configure USART CR1 register
        usart->CR1 = 0;
        usart->CR1 |= (config->baudRate << 4);
        usart->CR1 |= (config->dataBits << 12);
        usart->CR1 |= (config->stopBits << 12);
        usart->CR1 |= (config->parity << 9);

        // Enable transmitter and receiver
        usart->CR1 |= USART_CR1_TE | USART_CR1_RE;

        // Enable USART
        usart->CR1 |= USART_CR1_UE;

        return USART_OK;
    }
```

## Implementing the USART Interface (for STM32L475)

```
usartError_t USART_Send(usartDeviceID_t device, const uint8_t* data, uint32_t size)
{
    if (device >= USART_MAX || data == NULL || size == 0) {
        return USART_ERROR;
    }

    USART_TypeDef *usart = USARTs[device];

    for (uint32_t i = 0; i < size; i++) {
        // Wait until transmit data register is empty
        while (!(usart->SR & USART_SR_TXE)) {}

        // Write data to transmit data register
        usart->DR = data[i];
    }

    return USART_OK;
}
```

```
usartError_t USART_Receive(usartDeviceID_t device, uint8_t* buffer, uint32_t size)
{
    if (device >= USART_MAX || buffer == NULL || size == 0) {
        return USART_ERROR;
    }

    USART_TypeDef *usart = USARTs[device];

    for (uint32_t i = 0; i < size; i++) {
        // Wait until read data register is not empty
        while (!(usart->SR & USART_SR_RXNE)) {}

        // Read data from read data register
        buffer[i] = usart->DR;
    }

    return USART_OK;
}
```

## Implementing the USART Interface (for STM32L475)

```
// USART1 interrupt handler
void USART1_IRQHandler(void)
{
    USART_IRQHandler(USART_1);
}

// USART2 interrupt handler
void USART2_IRQHandler(void)
{
    USART_IRQHandler(USART_2);
}

// USART3 interrupt handler
void USART3_IRQHandler(void)
{
    USART_IRQHandler(USART_3);
}
```

```
void exampleUsage(void)
{
    // Configure USART1 with desired settings
    usartConfig_t config = {
        .baudRate = 115200,
        .dataBits = 8,
        .stopBits = 1,
        .parity = 0,
        .callback = NULL // Add your callback function here if needed
    };
    USART_Init(USART_1, &config);

    // Example transmit data
    uint8_t txData[] = "Hello, USART!";
    USART_Send(USART_1, txData, sizeof(txData));

    // Example receive data
    uint8_t rxBuffer[16];
    USART_Receive(USART_1, rxBuffer, sizeof(rxBuffer));
}
```

## Implementing the USART Interface (for STM32L475)

```
int main(void)
{
    // Initialize the system and peripherals

    // Example usage of USART interface
    exampleUsage();

    while (1) {
        // Main program loop
    }
}
```

4

# Going Further

## AI and ML Resources

- [Jacob's AI Blogs](#)
- [Jacob's CEC courses](#)
- [Jacob's ML Blogs](#)
- Embedded Bytes Newsletter
  - <http://bit.ly/1BAHYXm>

[www.beningo.com](http://www.beningo.com)

**BENINGO**  
EMBEDDED GROUP



# Thank You

Sponsored by

