CEC Continuing Education Center

DesignNews

Leveraging AI to Accelerate Embedded Software Development

DAY 2 : Writing Better Prompts for Code Generation

Sponsored by

DigiKey

BENINGO EMBEDDED GROUP

informa markets

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.

CEC Continuing Education Center

BENINGO EMBEDDED GROUP

DigiKey

THE SPEAKER

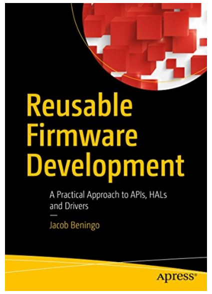
Jacob Beningo

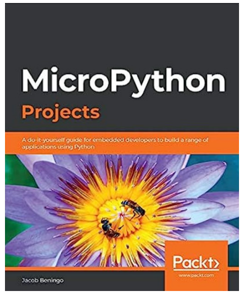
Jacob@beningo.com

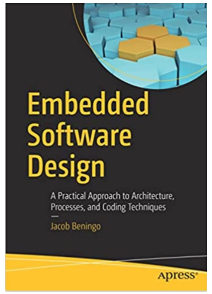
Beningo Embedded Group - President

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 to learn more

01

Writing Code with ChatGPT

CEC Continuing Education Center | BENINGO EMBEDDED GROUP | DigiKey

Writing Code with ChatGPT

Be careful what you ask for . . .

You
Write a function that calculates a running average with configurable number of values.

ChatGPT
Certainly! Below is an example of a Python function that calculates a running average. The function is designed to keep track of the last `n` values (where `n` is configurable) and computes the average of these values. It uses a deque from the `collections` module to efficiently handle this functionality.

python	Copy code

```python
from collections import deque

class RunningAverage:
    def __init__(self, size):
        """
        Init                                        specified size
        :par                                        the running av
        """
        self
        self
        self

    def add(
        """
        Add
        :par
        :ret
        """
        if l
            self.sum -= self.values[0]

        self.values.append(value)
        self.sum += value
        return self.sum / len(self.values)
```

Writing Code with ChatGPT

Principles for Effective Prompts

1) Be specific and clear
2) Use Technical Language when appropriate
3) Provide context
4) Outline the desired logic or algorithm
5) Mention the programming language
6) Specify the scope
7) Include examples or use cases
8) State any constraints
9) Iterate and refine

Writing Code with ChatGPT

Being more specific . . .

JA **You**

In the C programming language, write a function that calculates the running average of a buffer. The buffer size is configurable, but static allocation should be used. Malloc is not allowed.

The buffer will be used to store adc values that are then used by the application. Configuration values should be stored in a separate file. Assertions should be used to verify function parameters are valid.

Only allow functions to have a single return point. Use fixed point math, rather than floating point.

Make the function and any supporting ones reusable and in their own separate module.

Provide examples on how to use the code separately.

ChatGPT

To create a C module for calculating the running average of ADC values with the specified constraints, you should structure your code across two files: a header file (`RunningAverage.h`) for the declaration of the module's interface and a source file (`RunningAverage.c`) for the implementation. Additionally, a configuration file (`config.h`) will be used to set the buffer size. The code will use fixed-point arithmetic and include assertions for parameter validation. The functions will be designed to have only one return point.

Writing Code with ChatGPT

Being more specific . . .

Configuration File: `config.h`

```c
// config.h
#ifndef CONFIG_H
#define CONFIG_H

#define MAX_BUFFER_SIZE 100   // Max

#endif // CONFIG_H
```

Module Header: `RunningAverage.h`

c Copy code

```c
// RunningAverage.h
#ifndef RUNNING_AVERAGE_H
#define RUNNING_AVERAGE_H

#include "config.h"

typedef struct {
    int buffer[MAX_BUFFER_SIZE];
    int maxSize;
    int currentSize;
    int head;
    long long sum;
} RunningAverage;

void initRunningAverage(RunningAverage *ra, int size);
void addValue(RunningAverage *ra, int value);
int getRunningAverage(const RunningAverage *ra);

#endif // RUNNING_AVERAGE_H
```

```c
// RunningAverage.c
#include "RunningAverage.h"
#include <assert.h>

void initRunningAverage(RunningAverage *ra, int size) {
    assert(size > 0 && size <= MAX_BUFFER_SIZE);
    ra->maxSize = size;
    ra->currentSize = 0;
    ra->head = -1;
    ra->sum = 0;
}

void addValue(RunningAverage *ra, int value) {
    assert(ra != NULL && ra->maxSize > 0);

    ra->head = (ra->head + 1) % ra->maxSize;
    if (ra->currentSize < ra->maxSize) {
        ra->currentSize++;
    } else {
        ra->sum -= ra->buffer[ra->head];
    }
    ra->buffer[ra->head] = value;
    ra->sum += value;
}

int getRunningAverage(const RunningAverage *ra) {
    assert(ra != NULL && ra->currentSize > 0);

    int average = 0;
    average = (int)(ra->sum / ra->currentSize);
    return average;
}
```

Audience POLL Question

Which of the following is critical to successfully using AI to generate code?

a) Being specific and clear
b) Providing context
c) Iterating
d) All the above
e) None of the above

02

Teach it your coding style

Teach it your coding style

Coding Style

Coding Styles define how your software should look. It defines:

- Naming conventions
- Formatting
- Comments and documentation
- File structure
- Error handling
- Memory management
- Performance optimization
- etc

```c
#include <stdio.h>

/**
 * Adder - Adds two integers.
 * @num1: First integer to be added.
 * @num2: Second integer to be added.
 *
 * Return: Sum of num1 and num2.
 */
int Adder(int num1, int num2) {
    return num1 + num2;
}

int main() {
    int a = 5, b = 3;
    printf("Sum: %d\n", Adder(a, b));
    return 0;
}
```

Teach it your coding style

Pulling the format from ChatGPT

Potential issues with your existing style guide:

- It's too big to copy in at once
- It's not in a compatible format

Might consider the following:

- Have ChatGPT tell you what it needs
- Feed in your examples one step at a time

JA **You**

Can you provide an example of a c coding style guide for embedded systems that would help you to generate code?

ChatGPT

Certainly! Below is an example of a C coding style guide tailored for embedded systems. This guide emphasizes practices that enhance code readability, maintainability, and performance, which are crucial for embedded systems development.

C Coding Style Guide for Embedded Systems

1. **Naming Conventions**

- **Variables**: Use lowerCamelCase for variable names. Prefix boolean variables with `is`, `can`, or `has`.
- **Constants**: Use UPPER_CASE with underscores.
- **Functions**: Use UpperCamelCase for function names. Function names should be verbs and clearly describe the action performed.
- **Macros & Preprocessor Directives**: Use UPPER_CASE with underscores.
- **Enumerations**: Prefix enums with `e` and use UpperCamelCase for the type and UPPER_CASE for values.
- **Structures and Unions**: Prefix with `s` for structures and `u` for unions, followed by UpperCamelCase.
- **Typedefs**: Use UpperCamelCase and prefix with `t`.

2. **Formatting**

Teach it your coding style

Test it!

JA **You**

Develop an example implementation for an STM32 USART driver module.

ChatGPT

Developing an example implementation of a USART driver module for an STM32 microcontroller involves several key aspects, such as configuring the USART peripheral, implementing functions for initialization, sending and receiving data, and handling interrupts. This implementation will be based on your coding style guide, focusing on naming conventions, file structure, and error handling.

This example will be simplified and may not cover all the intricacies of an actual driver implementation. It assumes the use of STM32 HAL libraries for hardware abstraction.

Teach it your coding style

Test it!

File: stm32_usart_driver.c

c Copy code

```c
#include "stm32_usart_driver.h"

eUsartStatus UsartInit(sUsartConfig* config) {
    if (config == NULL) return eUSART_ERROR_INIT;

    config->huart.Instance = USART1; // Example: using USART1
    config->huart.Init.BaudRate = config->baudRate;
    config->huart.Init.WordLength = config->wordLength;
    config->huart.Init.StopBits = config->stopBits;
    config->huart.Init.Parity = config->parity;
    config->huart.Init.Mode = config->mode;
    config->huart.Init.HwFlowCtl = config->hwFlowCtl;
    config->huart.Init.OverSampling = UART_OVERSAMPLING_16;

    if (HAL_UART_Init(&(config->huart)) != HAL_OK) {
        return eUSART_ERROR_INIT;
    }

    return eUSART_OK;
}
```

File: stm32_usart_driver.h

c Copy code

```c
#ifndef STM32_USART_DRIVER_H
#define STM32_USART_DRIVER_H

#include "stm32f4xx_hal.h"

// Error codes for USART operations
typedef enum {
    eUSART_OK = 0,
    eUSART_ERROR_INIT,
    eUSART_ERROR_TX,
    eUSART_ERROR_RX
} eUsartStatus;

// USART configuration structure
typedef struct {
    USART_HandleTypeDef huart;
    uint32_t baudRate;
    uint32_t wordLength;
    uint32_t stopBits;
    uint32_t parity;
    uint32_t mode;
    uint32_t hwFlowCtl;
} sUsartConfig;

eUsartStatus UsartInit(sUsartConfig* config);
eUsartStatus UsartTransmit(USART_HandleTypeDef* huart, uint8_t* pData,
eUsartStatus UsartReceive(USART_HandleTypeDef* huart, uint8_t* pData, u

#endif // STM32_USART_DRIVER_H
```

Teach it your coding style

Test it!

```
eUsartStatus UsartTransmit(USART_HandleTypeDef* huart, uint8_t* pData,
    if (huart == NULL || pData == NULL) return eUSART_ERROR_TX;

    if (HAL_UART_Transmit(huart, pData, Size, 1000) != HAL_OK) { // 100
        return eUSART_ERROR_TX;
    }

    return eUSART_OK;
}

eUsartStatus UsartReceive(USART_HandleTypeDef* huart, uint8_t* pData, u
    if (huart == NULL || pData == NULL) return eUSART_ERROR_RX;

    if (HAL_UART_Receive(huart, pData, Size, 1000) != HAL_OK) { // 1000
        return eUSART_ERROR_RX;
    }

    return eUSART_OK;
}
```

Audience POLL Question

How well do you feel that the AI followed our coding style?

a) Surprising well
b) Well, but there is room for improvement
c) Not well
d) The worst. Where are the interns?

03 Prompt Examples

Prompt Examples

Temperature Sensor Interface

JA **You**

Develop firmware for an STM32 microcontroller to interface with a DS18B20 temperature sensor. The firmware should initiate temperature readings every 5 minutes, store the readings in an internal buffer, and transmit the data via UART to a connected computer. Include error handling for sensor disconnection and implement a low-power sleep mode between readings.

ChatGPT

Developing firmware for an STM32 microcontroller to interface with a DS18B20 temperature sensor and meet your specified requirements involves several steps. This example will provide a basic framework, following your coding style guide, for these functionalities:

1. **Sensor Initialization and Reading**: Interface with the DS18B20 sensor to initiate and read temperatures.
2. **Timing Control**: Use a timer to trigger temperature readings every 5 minutes.
3. **Buffer Management**: Store readings in an internal buffer.
4. **UART Communication**: Transmit the buffered data to a connected computer.
5. **Error Handling**: Implement checks for sensor disconnection and error conditions.
6. **Low-Power Management**: Enter a low-power sleep mode between readings.

Note: This is a high-level description. The actual implementation will depend on the specifics of your STM32 microcontroller and development environment.

```c
#include "stm32f4xx_hal.h"  // Adjust for your specific STM32 series
#include "ds18b20.h"        // Include DS18B20 sensor driver

#define BUFFER_SIZE 12      // Number of readings to store
#define READ_INTERVAL 300000 // 5 minutes in milliseconds

// Global variables
float temperatureBuffer[BUFFER_SIZE];
uint8_t bufferIndex = 0;

// Function prototypes
void SystemClock_Config(void);
void USART2_Init(void);
void Timer_Init(void);
void EnterSleepMode(void);
float ReadTemperature(void);
void TransmitData(float* buffer, uint8_t size);
void HandleSensorError(void);

int main(void) {
    HAL_Init();
    SystemClock_Config();
    USART2_Init();
    Timer_Init();
    DS18B20_Init(); // Initialize DS18B20 sensor

    uint32_t lastReadTime = 0;

    while (1) {
```

Prompt Examples

Iteration #2

Iteration is the key

Not perfect . . .

```
// Error codes for USART operations
typedef enum {
    eUSART_OK = 0,
    eUSART_ERROR_INIT,
    eUSART_ERROR_TX,
    eUSART_ERROR_RX
} eUsartStatus;
```

Audience POLL Question

What do you think AI is more likely to do?

a) Take your job
b) Act as a tool to make you more efficient
c) Teach you to communicate more clearly
d) Improve how you develop requirements

Next Steps

04

Additional Resources

Please consider the resources below:

- Jacob's AI Blogs
- Jacob's CEC courses
- Jacob's ML Blogs

- Embedded Bytes Newsletter
  - http://bit.ly/1BAHYXm

www.beningo.com

Consulting | Coaching | Training

Next Steps

- ✓ The Rise of AI in Embedded Software
- ✓ Writing Better Prompts for Code Generation
- Optimizing your Build System with AI
- Abstracting your Hardware with an AI-Generated HAL
- Managing AI and ML Code

Thank You

Sponsored by