INTRODUCTION

CHAPTER1

1.1 INTRODUCTION TO EMBEDDED C

Embedded C is a specialized programming language used for developing software that runs on embedded systems. These systems are often small, resource-constrained, and designed for specific tasks, making them different from general-purpose computers. Embedded C is a variant of the C programming language, customized to suit the needs and limitations of embedded systems.

1.1.1 Embedded Systems: Embedded systems are found in a wide range of applications, from consumer electronics and automotive control systems to industrial automation and medical devices. They are dedicated, task-specific devices with limited resources, including memory, processing power, and energy.

1.1.2 C Programming Language: C is a widely used and highly efficient programming language known for its low-level capabilities and portability. Embedded C is essentially a subset of the C language, customized for embedded systems. It shares the basic syntax and semantics of C but often includes extensions, constraints, and optimizations specific to embedded development.

Embedded C is a specialized programming language that is essential for developing software on embedded systems. It takes into account the unique requirements and constraints of these systems, offering low-level control and real-time capabilities. Embedded C programmers must have a deep understanding of hardware and be skilled in optimizing code for efficiency and reliability in resource-constrained environments.

1.2 INTRODUCTION TO SOFTWARE -STM32CUBE IDE

STMicroelectronics' STM32Cube IDE is an integrated development environment (IDE) specifically designed for programming and developing applications for STM32 microcontrollers. STM32 microcontrollers are a popular family of microcontrollers used in a wide range of embedded systems. STM32Cube IDE offers a comprehensive platform to streamline the development process.

1.2.1 IDE for STM32 Microcontrollers: STM32Cube IDE is tailored for STM32 microcontroller development. It provides a dedicated environment that includes everything you need to develop, compile, and debug applications for STM32 devices.

1.2.2 Built on Eclipse: STM32Cube IDE is built on the Eclipse IDE, which is a well-established and highly extensible open-source development platform. It leverages Eclipse's capabilities and adds STM32-specific tools and features.

1.2.3 Code Development and Editing: STM32Cube IDE offers a code development and editing environment, making it easy to write, edit, and manage your embedded C code for STM32 microcontrollers. It supports various programming languages, including C and C++.

1.2.4 Code Generation: STM32Cube IDE integrates STM32CubeMX, a tool that helps in configuring and initializing STM32 peripherals. It assists in generating initialization code and drivers for your specific microcontroller, reducing the effort required to set up the hardware.

STM32Cube IDE is a specialized development environment for STM32 microcontrollers. It streamlines the process of developing embedded applications by providing code editing, project management, code generation, debugging, and peripheral configuration tools, along with integration with STM32CubeMX and STM32CubeRTOS. This IDE simplifies the development of applications for STM32 microcontrollers and offers a rich set of features for embedded system development.

1.3 INTRODUCTION TO MICROCONTROLLER

The STM32 microcontroller family is a popular series of microcontrollers developed by STMicroelectronics. These microcontrollers are widely used in various embedded systems, and they come with a wide range of features and capabilities. Here's an introduction to the STM32 microcontroller family

1.3.1 Overview:

- Manufacturer: STMicroelectronics
- Core Architectures: STM32 microcontrollers are based on ARM Cortex-M processor cores, making them powerful and energy-efficient.

1.3.2 Key Features:

1.3.2.1 Processor Cores: STM32 microcontrollers use ARM Cortex-M cores, with various models supporting Cortex-M0, M0+, M3, M4, and M7.

1.3.2.2 Memory: They come with flash memory for program storage and RAM for data storage. Some models have external memory interfaces for further expansion.

1.3.2.3 Power Management: Many STM32 devices offer low-power modes and features to conserve energy.

1.3.2 Applications:

1.3.3.1 Consumer Electronics: Smartphones, tablets, smartwatches, and remote controls.

1.3.3.2 Automotive: Engine control units infotainment systems, and advanced driver-assistance systems .

1.3.3.3 Industrial Automation: Programmable logic controllers (PLCs), motor control, and factory automation.

1.3.3.4 Medical Devices: Vital sign monitors, infusion pumps, and medical imaging equipment.

1.3.3.5 IoT (Internet of Things): Sensors, communication devices, and edge computing nodes.

1.3.3.6 Robotics: Robotic control systems for automation and research.

1.3.3.7 Home Automation: Smart thermostats, security systems, and lighting control.

1.3.4 Programming:

STM32 microcontrollers can be programmed in C, C++, and assembly languages.

the STM32 microcontroller family, based on ARM Cortex-M cores, is a versatile and widely adopted solution for embedded systems. With a range of features, extensive peripheral options, and strong support from STMicroelectronics and the developer community, STM32 microcontrollers are well-suited for a wide variety of applications across different industries.

CHAPTER 2

2.1 MOVING AVERAGE FILTER

A Moving Average Filter, also known as a running average filter, is a simple and widely used digital signal processing technique to smooth or reduce noise in time-series data. It calculates the average value of a series of data points over a specific window or interval and replaces the original data points with the computed average. This helps to eliminate short-term fluctuations or high-frequency noise while preserving the overall trend or pattern in the data.

2.1.1 Window or Interval: You define a fixed-size window or interval, which determines how many data points will be included in the average calculation. This window size is often represented as 'N' and can be any positive integer.

2.1.2 Data Points: As new data points become available over time, you continually update the window to include the most recent 'N' data points. Older data points drop out of the window.

Implementation Steps:

Define the Input Data: The input data is a 1D sequence of values, such as a list or a NumPy array.

Set Window Size: Specify the size of the moving window. It should be an odd number to have a symmetric filter.

Calculate Averages: If the data point is near the edges, you keep it as is since there may not be enough points to form a complete window. If the data point is not near the edges, calculate the average of the data points within the window. This involves summing the values within the window and dividing by the window size.

Replace Data Point: Replace the central data point (the one currently being processed) with the calculated average.

Repeat: Continue this process for all data points in the input sequence.

Output Filtered Data: The result of the filter is a new sequence of data points, where the central values have been replaced with the calculated averages.

The window size you choose and the characteristics of your data will determine the level of smoothing and noise reduction achieved. Larger window sizes result in smoother output but may lose fine details and responsiveness to rapid changes in the data, while smaller window sizes provide less smoothing but preserve more detail.

2.2 USES OF MOVING AVERAGE FILTER

2.2.1 .Noise Reduction: It is often used to remove high-frequency noise or fluctuations from sensor data, such as temperature measurements, stock prices, or audio signals.

2.2.2.Smoothing: It smooths out jagged or irregular time-series data to reveal underlying trends or patterns. For example, it can be applied to financial data to visualize long-term trends.

2.2.3.Averaging and Data Compression: It helps to summarize a series of data points, reducing the dataset's size while preserving important characteristics.

2.2.4.Time-Series Analysis: Moving averages are frequently used in time-series analysis and forecasting, such as in financial analysis, demand forecasting, and trend analysis.

There are different variations of moving average filters, including the simple moving average (SMA) discussed above, as well as weighted moving averages and exponential moving averages (EMAs), which give more weight to recent data points. The choice of the moving average type and the window size depends on the specific application and the nature of the data.

It's important to note that while moving average filters are effective for noise reduction and smoothing, they introduce a time delay in the filtered output, which can be a consideration in real-time applications. Additionally, they may not be suitable for all types of data and noise characteristics, so other signal processing techniques should be considered depending on the specific requirements and data properties.

2.3 APPLICATIONS OF MOVING AVERAGE FILTER

Moving average filters, such as the Simple Moving Average (SMA), find applications in various fields where data smoothing, noise reduction, or trend analysis is required. Here are some common applications:

Financial Analysis:

Stock Price Analysis: Moving averages are widely used in stock and financial markets to identify trends and generate trading signals. Exponential Moving Averages (EMAs) are particularly popular for this purpose.

Epidemiology:

Disease Outbreak Analysis: Moving averages can be applied to epidemiological data to smooth out fluctuations and identify trends in disease outbreak patterns, aiding in decision-making and resource allocation.

Economics:

Economic Data Analysis: Economists use moving averages to analyze economic indicators, such as GDP, unemployment rates, and inflation, to understand long-term trends and eliminate short-term fluctuations.

Environmental Science:

Climate Data Analysis: Moving averages help analyze climate data to identify long-term trends, seasonal variations, and underlying patterns, making it easier to predict climate changes and assess environmental impact.

Signal Processing:

Noise Reduction: Moving average filters are used to eliminate noise from signals in various domains, such as audio and image processing. For example, a moving average can smooth out audio signals to reduce background noise.

Engineering:

Sensor Data Filtering: In robotics and automation, moving averages can be used to filter noisy sensor data, improving the accuracy of measurements and control systems.

Quality Control:

Manufacturing: Moving averages can be applied in manufacturing processes to monitor product quality over time and detect deviations from the expected performance.

Weather Forecasting:

Weather Data Analysis: Moving averages are employed to analyze historical weather data, identify climate trends, and improve the accuracy of weather forecasts.

Stock Market Analysis:

Technical Analysis: Traders and analysts use various moving averages, such as the Simple Moving Average (SMA) and Exponential Moving Average (EMA), to assess price trends, support, and resistance levels.

Time Series Analysis:

General Data Smoothing: Moving average filters are used in time series analysis to smooth data and extract underlying trends and patterns, which can be valuable for forecasting and modeling.

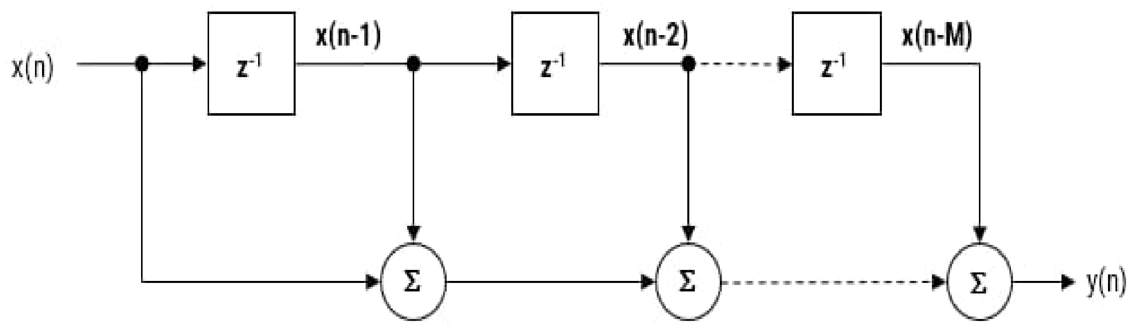
Inventory Management:

Demand Forecasting: Moving averages are used in supply chain and inventory management to predict future demand for products based on historical sales data, helping companies optimize stock levels.

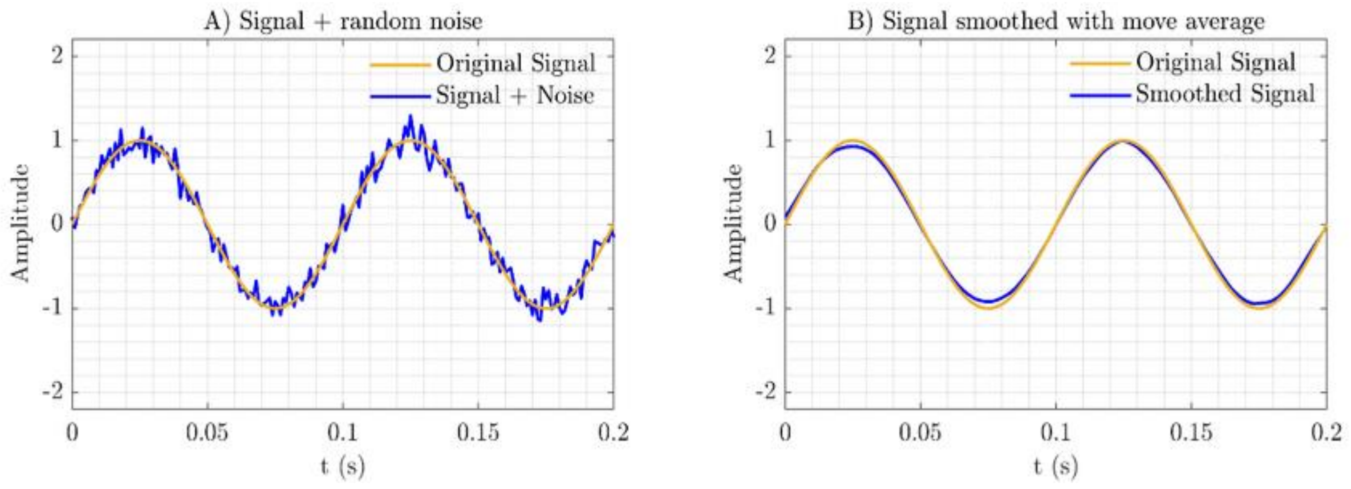
Biomedical Research:

Biological Data Analysis: In areas like genetics and physiology, moving averages are used to process biological data, identify patterns in experiments, and analyze trends in biological measurements.

2.4 BLOCK DIAGRAM



2.5 EXAMPLE



2.5.1 NOISE REDUCTION

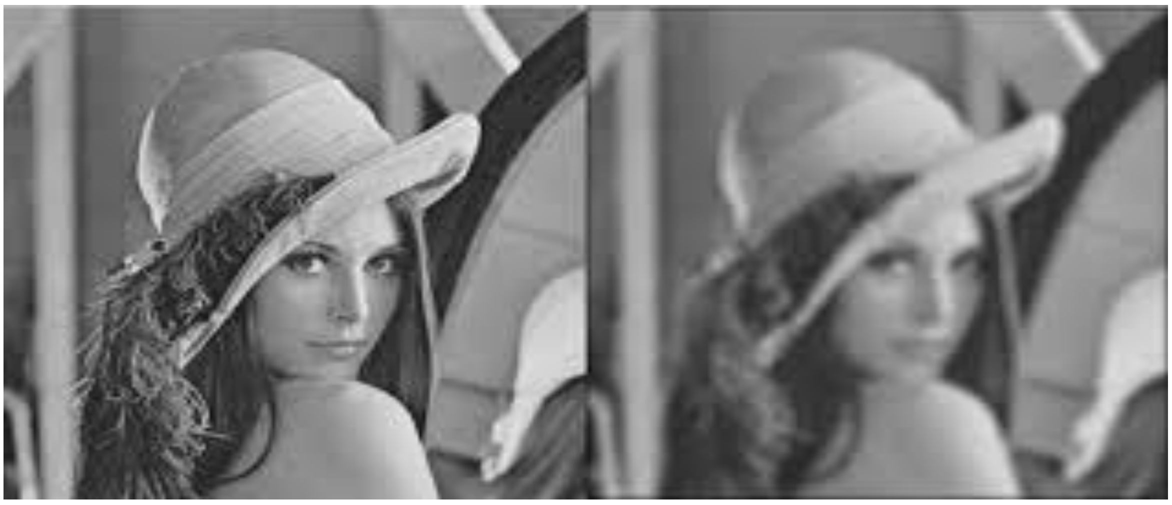


Image mean filtered

2.5.2 IMAGE FILTERING

CHAPTER 3

CODE FOR MOVING AVERAGE FILTER IMPLEMENTATION

```
#include <stdint.h>
// Define the window size
#define WINDOW_SIZE 3
// Define the array of sensor readings
int16_t sensor_readings[20] = {10, 12, 15, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50};
// Calculate the moving average
int16_t calculate_moving_average(int16_t *readings, int16_t window_size) {
  int16_t sum = 0;
  for (int i = 0; i < window_size; i++) {
    sum += readings[i];
  }
  return sum / window_size;
}
// Print the moving average
void print_moving_average(int16_t moving_average) {
  printf("Moving average: %d\n", moving_average);
}
// Main function
int main(void) {
  // Initialize the moving average filter
  int16_t moving_average = 0;
  // Calculate the moving average for each sensor reading
  for (int i = 0; i < 20; i++) {
    moving_average = calculate_moving_average(sensor_readings + i, WINDOW_SIZE);
    // Print the moving average
    print_moving_average(moving_average);
  }
  return 0;
}
```

OUTPUT

```
Moving average: 12
Moving average: 15
Moving average: 17
Moving average: 20
Moving average: 22
Moving average: 24
Moving average: 26
Moving average: 28
Moving average: 30
Moving average: 32
Moving average: 34
Moving average: 36
Moving average: 38
Moving average: 40
Moving average: 42
Moving average: 44
Moving average: 46
Moving average: 48
Moving average: 32
Moving average: 16
```

CONCLUSION

The moving average filter is a widely used digital signal processing technique that helps smooth data and reduce noise by averaging neighbouring data points over a defined window. Its simplicity and effectiveness make it a valuable tool for various applications, including signal processing, finance, and image processing. However, it's important to choose an appropriate window size to balance smoothing and preserving relevant information in the data.

In summary, implementing a moving average filter involves calculating the average of data points within a specified window size, which is slid across the dataset. The choice of window size greatly influences the trade-off between noise reduction and maintaining signal details. Efficient implementation and understanding the impact of window size are essential for successful application of the moving average filter in various domains such as finance, signal processing, and sensor data analysis.

REFERENCES

https://www.researchgate.net/publication/311257038_Use_Moving_Average_Filter_to_Reduce_Noises_in_Wearable_PPG_During_Continuous_Monitoring

https://www.sciencedirect.com/topics/engineering/moving-average-filter

https://codemonk.in/blog/moving-average-filter/

*VIEDIO LINKS

https://youtu.be/yargH0L3B68?si=KZX8lJF_RNipkNrV

https://youtu.be/rttn46_Y3c8?si=9B9mWpujWczlLxYI