# CRC for error detection

Introduction: Embedded systems often encounter data transmission or storage scenarios where data integrity is of utmost importance. Cyclic Redundancy Code (CRC) is a powerful error detection technique widely used in embedded systems to ensure reliable data transmission and storage. In this essay, we will delve into the concept of CRC, discuss its significance in achieving robust data integrity, and provide an example embedded C code to demonstrate its implementation.

Understanding CRC: Cyclic Redundancy Code (CRC) is an error detection algorithm that uses polynomial division to generate a checksum or a CRC code for a given set of data. It appends this code to the original data, allowing the recipient to verify the integrity of the received data. CRC employs a cyclic shift register and bitwise XOR operations to compute the CRC code, which is based on a predefined polynomial.

Working Principle: The working principle of CRC involves the following steps:

1. Selection of CRC Polynomial: A suitable polynomial is chosen based on the desired error detection capabilities. Commonly used polynomials include CRC-8, CRC-16, and CRC-32. The polynomial is represented as a binary value, and its degree determines the number of bits in the CRC code.
2. Data Calculation: The data to be transmitted or stored is divided by the CRC polynomial using polynomial division. This division process involves shifting the polynomial one bit at a time and performing bitwise XOR operations on the data and the polynomial.
3. CRC Code Generation: After completing the polynomial division, the remainder obtained represents the CRC code. The CRC code is appended to the original data, forming the transmitted or stored data packet.
4. Verification at the Receiving End: Upon receiving the data packet, the recipient performs the same CRC calculation on the received data, using the same CRC polynomial. If the calculated CRC code matches the received CRC code, it indicates that the data is likely to be error-free. However, if the CRC codes differ, it suggests the presence of errors in the data.

Example: CRC Calculation in Embedded C Code Let's consider an example where a microcontroller-based embedded system performs CRC calculation on a packet of data using CRC-8.

Here's an example embedded C code snippet illustrating the CRC calculation:

```
#include <stdint.h>

#define CRC_POLYNOMIAL 0x07 // CRC-8 polynomial: x^8 + x^2 + x + 1

uint8_t calculateCRC(uint8_t* data, uint16_t length) {
    uint8_t crc = 0;

    for (uint16_t i = 0; i < length; i++) {
        crc ^= data[i]; // XOR with data byte

        for (uint8_t bit = 0; bit < 8; bit++) {
            if (crc & 0x80) { // MSB is 1
                crc = (crc << 1) ^ CRC_POLYNOMIAL;
            } else {
                crc <<= 1;
            }
        }
    }

    return crc;
}
```

```
int main() {
    uint8_t data[] = {0x55, 0xAA, 0x12, 0x34}; // Example data
    uint16_t dataLength = sizeof(data);

    uint8_t crc = calculateCRC(data, dataLength);

    // Append CRC to data and transmit data packet
    // ...

    return 0;
}
```

In this code, we have a function called **calculateCRC()** that takes an array of data and its length as input. The CRC register, **crc**, is initialized to 0xFFFF, following the initialization step of CRC calculation. The function then processes each byte of the data array and updates the CRC register accordingly, applying the CRC-16 polynomial.

In the **main()** function, an example data array, **data[]**, and its length, **dataLength**, are defined. The **calculateCRC()** function is called, passing the data and length as arguments, to obtain the CRC checksum. Once the CRC checksum is calculated, it can be transmitted along with the data for error detection and verification at the receiving end.

Significance of CRC: Cyclic Redundancy Check offers several benefits in embedded systems:

1. Robust Error Detection: CRC algorithms can detect a wide range of errors, including single-bit errors, burst errors, and most common multiple-bit errors. This makes CRC highly effective in ensuring data integrity during transmission or storage.

2. Efficiency: CRC calculations are computationally efficient, making them suitable for resource-constrained embedded systems. The simplicity of the bitwise operations involved in CRC calculations allows for fast and reliable error detection.
3. Widely Used: CRC is a widely adopted error-detection technique in various industries and protocols, including telecommunications, storage systems, networking, and data transmission standards like Ethernet and USB. Its ubiquity signifies its reliability and proven effectiveness.

Conclusion: Cyclic Redundancy Check (CRC) is a powerful error-detection technique used in embedded systems to ensure data integrity. By performing polynomial division on the data, CRC generates a checksum that can be used for error detection and verification. The example embedded C code demonstrates the implementation of CRC-16 calculation, highlighting its practical usage. By employing CRC algorithms, embedded systems can enhance the reliability and integrity of transmitted and stored data, ensuring the smooth operation of critical applications.