**8. WAP to Implement Error Detection: VRC**

**Steps to Implement VRC**

1. **Select Parity**: Choose either even or odd parity.
2. **Input Data Blocks**: Divide the data into blocks of equal size (e.g., 8 bits).
3. **Calculate Parity Bit**:
   * Count the number of 1s in the block.
   * For **even parity**, append a 0 if the count is even; otherwise, append a 1.
   * For **odd parity**, append a 1 if the count is even; otherwise, append a 0.
4. **Verify Parity**: At the receiver's end, check if the number of 1s matches the chosen parity.

**Implementation in C**

c

Copy code

#include <stdio.h>

#include <string.h>

// Function to calculate and append parity bits

void calculateVRC(char data[][9], int rows, char parityData[][10], int evenParity) {

for (int i = 0; i < rows; i++) {

int count = 0;

// Count the number of 1s in the block

for (int j = 0; j < 8; j++) {

if (data[i][j] == '1') {

count++;

}

}

// Determine parity bit

if (evenParity) {

// Even parity: append 0 if count is even, 1 if odd

parityData[i][8] = (count % 2 == 0) ? '0' : '1';

} else {

// Odd parity: append 1 if count is even, 0 if odd

parityData[i][8] = (count % 2 == 0) ? '1' : '0';

}

// Append parity bit and terminate the string

strncpy(parityData[i], data[i], 8);

parityData[i][9] = '\0';

}

}

// Function to verify parity at the receiver

int verifyVRC(char parityData[][10], int rows, int evenParity) {

for (int i = 0; i < rows; i++) {

int count = 0;

// Count the number of 1s in the block, including the parity bit

for (int j = 0; j < 9; j++) {

if (parityData[i][j] == '1') {

count++;

}

}

// Check if parity matches the chosen parity type

if ((evenParity && count % 2 != 0) || (!evenParity && count % 2 == 0)) {

return 0; // Parity check failed

}

}

return 1; // Parity check passed

}

int main() {

char data[4][9] = { // Input binary data blocks (8 bits each)

"11001101",

"10101010",

"11110000",

"00001111"

};

char parityData[4][10];

int rows = 4;

// Calculate and append even parity bits

calculateVRC(data, rows, parityData, 1); // Use 1 for even parity

printf("Data with Parity Bits (Even Parity):\n");

for (int i = 0; i < rows; i++) {

printf("%s\n", parityData[i]);

}

// Verify parity at the receiver

if (verifyVRC(parityData, rows, 1)) {

printf("Parity Check Passed: No Errors Detected.\n");

} else {

printf("Parity Check Failed: Errors Detected.\n");

}

return 0;

}

**How It Works**

**Sender Side:**

1. Input data blocks (8 bits each).
2. The **calculateVRC** function calculates and appends a parity bit to each block.

**Receiver Side:**

1. The **verifyVRC** function checks if the total number of 1s in each block matches the chosen parity.
2. If parity does not match, an error is detected.

**Compilation and Execution**

**Compilation:**

bash

Copy code

gcc -o vrc vrc.c

**Execution:**

bash

Copy code

./vrc

**Expected Output**

**Input Data Blocks:**

plaintext

Copy code

11001101

10101010

11110000

00001111

**Output Data with Parity Bits (Even Parity):**

plaintext

Copy code

110011010

101010100

111100000

000011111

**Result:**

plaintext

Copy code

Parity Check Passed: No Errors Detected.

**Key Notes**

1. **Parity Type**: Switch to odd parity by changing the evenParity parameter in the functions.
2. **Error Detection**:
   * Can only detect single-bit errors in a block.
   * Not effective for multi-bit errors.