DESIGN DOCUMENT: CRC-12 Error Detection

1. Module: `crc_common.h` (Helper File)

- **Sub-Modules:**
  - Packet Definition
  - Network Configuration
  - Global Constants
- **FSM:** N/A (Header file)
- **Algorithm:** N/A (Header file)
- **Code (Functions / Subroutines):**
  - `struct Packet` : A C++ struct to represent our data packet.
    - `char data[100]` : Holds the 100-byte data chunk from the file.
    - `uint16_t crc_checksum` : Holds the 12-bit CRC remainder. (We use `uint16_t` as it's the smallest standard type that can hold 12 bits).
    - `bool is_last_packet` : A flag to tell the receiver when the file transfer is complete.
  - `#define PORT 8080` : Network port.
  - `#define HOST "127.0.0.1"` : Localhost IP.
  - `#define POLY "1100000001111"` : The 13-bit CRC-12 polynomial.
  - `#define CRC_BITS 12` : The degree of the polynomial.
  - `#define PACKET_SIZE 100` : Data chunk size in bytes.
- **Documentation:** A common header file to be included by all programs. It ensures all components use the same packet structure, polynomial, and network port.

2. Module: `crc_module.h` (Helper File)

- **Sub-Modules:**
  - CRC-12 Logic
  - Binary/String Conversion Utilities
- **FSM:** N/A (Header file with functions)
- **Algorithm:**
  - **CRC Division:** Implements binary long division (using XOR) to divide a data string by the polynomial and return the remainder.
  - **Byte-to-Binary:** Converts a 100-byte `char` array into an 800-bit `std::string` (e.g., "01000001...").
  - **Binary-to-Checksum:** Converts a 12-bit `std::string` (e.g., "101010101010") into a `uint16_t` for storage in the packet.

  - **Checksum-to-Binary:** Converts a `uint16_t` checksum back into a 12-bit `std::string` for verification.
- **Code (Functions / Subroutines):**
  - `std::string bytesToBinary(char* data, int len)`: Implements "Byte-to-Binary".
  - `uint16_t binaryToChecksum(std::string binary_crc)`: Implements "Binary-to-Checksum".
  - `std::string checksumToBinary(uint16_t crc)`: Implements "Checksum-to-Binary".
  - `std::string crcMod2Division(std::string data_with_padding)`: Implements the core "CRC Division" algorithm.
- **Documentation:** A helper file containing the core CRC-12 logic. This is kept separate so both the Sender and Receiver can use the *exact same* functions for calculation and verification.

3. Module: `crc_sender.cpp`

- **Sub-Modules:**
  - File Reader
  - CRC Calculator
  - Socket Sender
- **FSM:**
  1. START
  2. Open `input.txt` file.
  3. LOOP: While not End-of-File: a. Read 100 bytes (or less for the last packet). b. Call `bytesToBinary()` on the data. c. Append 12 '0's to the binary string. d. Call `crcMod2Division()` to get the 12-bit binary remainder. e. Call `binaryToChecksum()` to get the `uint16_t` CRC. f. Create `Packet` struct with data and CRC. g. `send()` packet to Server.
  4. Send a final "last packet" marker.
  5. END
- **Algorithm:** Implements the sender-side logic from the lab sheet (Part a, b, c). It reads the file in 100-byte chunks, calculates the 12-bit CRC for each, and sends it as a `Packet`.
- **Code (Functions / Subroutines):**
  - `main()`: Main program loop.
  - Calls `socket()`, `connect()`, `send()`, `close()`.
  - Calls functions from `crc_module.h`.

4. Module: `crc_server.cpp` (Optional, but good for error sim)

- **Sub-Modules:**
  - Connection Listener
  - Packet Relay
- **FSM:**

1. START
2. `socket()`, `bind()`, `listen()`.
3. Accept Sender.
4. Accept Error Inducer.
5. Accept Receiver.
6. LOOP: a. `recv()` packet from Sender. b. `send()` packet to Error Inducer. c. `recv()` (possibly corrupted) packet from Error Inducer. d. `send()` packet to Receiver.
7. If "last packet" seen, END.

- **Algorithm:** Acts as the central hub, passing the packet from Sender -> Error Inducer -> Receiver.
- **Code (Functions / Subroutines):**
  - `main()`: Main loop.
  - Calls `socket()`, `bind()`, `listen()`, `accept()`, `send()`, `recv()`.

5. Module: `crc_error_inducer.cpp`

- **Sub-Modules:**
  - Packet Receiver
  - Bit-flipper
  - Packet Sender
- **FSM:**
  1. START
  2. Connect to Server.
  3. LOOP: a. `recv()` packet from Server. b. **Error Logic:** If not the last packet, flip one bit (e.g., `packet.data[10] = 'X'`). c. `send()` packet back to Server.
  4. If "last packet" seen, END.
- **Algorithm:** Simulates a transmission error by receiving a valid packet, intentionally corrupting one byte in the data, and forwarding it.
- **Code (Functions / Subroutines):**
  - `main()`: Main loop.
  - Calls `socket()`, `connect()`, `send()`, `recv()`.

6. Module: `crc_receiver.cpp`

- **Sub-Modules:**
  - Socket Receiver
  - CRC Verifier
  - Output Display
- **FSM:**
  1. START

2. Connect to Server.
3. LOOP: a. `recv()` packet from Server. b. If "last packet," break loop. c. Call `bytesToBinary()` on `packet.data`. d. Call `checksumToBinary()` on `packet.crc_checksum`. e. Concatenate: `binary_data + binary_crc`. f. Call `crcMod2Division()` on the combined string. g. Check if the remainder is all '0's. h. Print "No error detected" or "ERROR DETECTED".
4. END

- **Algorithm:** Implements the receiver-side logic. It re-calculates the CRC on the *entire* received frame (data + checksum). If the result is a zero-remainder, the packet is valid.
- **Code (Functions / Subroutines):**
  - `main()`: Main loop.