FPGA Project

Reza Nematollahi - AmirHossein Taslimi

OverView:

This project is about designing and implementing a digital system for image coding using the chain code algorithm. The chain code algorithm is a method of encoding the edges of an object in an image by assigning a code to one of the eight directions (up, down, left, right and diagonal). The encoder module reads the binary image from a memory block and generates the code for the image. The decoder module receives the code and reconstructs the original image. The communication between the encoder and the decoder is done by a UART module

Members' Participation Level:

The **Encoder** module has been implemented by Reza Namatollahi.
The **Decoder** module has been implemented by Amirhossein Taslimi.
The **UART** section has been implemented by both group members.

Encoder Module:

The encoder module is designed to convert a binary image into vectors using the chain code algorithm. The chain code algorithm represents a boundary by encoding the direction of transitions between pixels along the boundary. The module is implemented in Verilog and consists of a state machine that manages the entire encoding process, from finding area and perimeter to transmitting the encoded vectors. This state machine processes the input binary image and generates a corresponding vector representation.

Module Interface:

- Inputs:
  - reset: Asynchronous reset signal.
  - clk: Clock signal.
  - Start: Signal to initiate the encoding process.
- Outputs:
  - code: Output vector code.
  - done: Indicates when the encoding process is complete.
  - error: Indicates any error during encoding (e.g., incorrect starting point).
  - perimeter_output: Output representing the perimeter of the binary image.
  - area_output: Output representing the area of the binary image.
  - start_pixel_x, start_pixel_y: Coordinates of the starting pixel for encoding.

Internal Variables and Data Structures:

- picture: Two-dimensional array representing the binary image.
- edges: Two-dimensional array to store processed edges of the image.
- temp: Temporary two-dimensional array used during processing.
- current_x, current_y: Current coordinates in the image.
- start_x, start_y: Starting coordinates for encoding.
- temp_index_x, temp_index_y: Temporary indices used during processing.
- perimeter, area: Variables to store the perimeter and area of the image.
- vector_table: Table to store the generated vector codes.
- vector_table_index: Index to keep track of the current position in the vector table.
- sending_vector_index, sending_vector_bit_index: Indices for sending vectors.
- sending_data_clk: Clock for controlling the transmission of vector data.
- state: State variable for the state machine.
- transmission_pattern_state: State variable for the transmission state machine.

State Machine:

The module utilizes a state machine to control the encoding process. The states include:

- idle_state: Initial state waiting for the start signal.
- finding_area_perimeter_state: Computes area and perimeter of the image.
- finding_start_point_state: Identifies the starting point for encoding.
- encoding_state: Generates vectors using the chain code algorithm.
- transmission_state: Transmits the generated vector codes.

Processing Steps:

- Initialization: Resets internal variables and sets the state to idle.
- Finding Area and Perimeter State: Computes the area and perimeter of the binary image using the chain code algorithm.
- Finding Start Point State: Identifies the starting point for encoding by searching for the first boundary pixel.
- Encoding State: Generates vector codes using the chain code algorithm, updating the vector table and coordinates.
- Transmission State: Transmits the generated vector codes, including start and stop bits.

Decoder Module:

The Decoder module is designed to receive serial data, decode chain codes, update pixel coordinates based on the chain codes, store processed pixels in memory, and output relevant information such as completion status and potential errors.

Module Interface:

- Inputs:
  - CLK: Clock input.
  - input_serial_bit: Input bit for serial data.
  - Area: 16-bit input for area data.
  - perimeter: 16-bit input for perimeter data.
  - reset: Reset signal.
  - start: Start signal.
  - start_pixel_x: 6-bit input for the starting pixel's x-coordinate.
  - start_pixel_y: 6-bit input for the starting pixel's y-coordinate.

- Outputs:
  - Packet_Done_output: Output indicating the completion of a packet.
  - ChainCode_ouput: 4-bit output representing the chain code.
  - current_x: 6-bit output representing the current x-coordinate in processing pixels.
  - current_y: 6-bit output representing the current y-coordinate in processing pixels.
  - error: Output indicating an error condition.
  - done: Output indicating the completion of the decoding process.

Internal Variables and Data Structures:

- memory: 2D array of size 64x64 to store pixel information.
- Idle, Start_Bit, Data_Bit, Stop_Bit, finish: Parameters defining the states of the state machine.
- r_Clock_Count: 8-bit register to keep track of clock cycles.
- r_Bit_Index: 3-bit register representing the current bit index during data reception.
- Chain_Code: 4-bit register to store the received chain code.
- Packet_Done: 1-bit register indicating the completion of receiving a packet.
- state: 3-bit register representing the state of the state machine.
- start_pix: 1-bit register to track the initial pixel calculation.

State Machine:

- Idle: Initial state, waiting for a start bit.
- Start_Bit: Checking the middle of the start bit, transitioning to the Data_Bit state if valid.
- Data_Bit: Waiting to sample serial data for each bit, updating the chain code accordingly.
- Stop_Bit: Receiving and waiting for the stop bit, indicating the completion of a packet.
- finish: Transition state to return to Idle after completing a packet.

Processing Steps:

- Initialization:
  The memory array is initialized as a 64x64 array, representing the pixels. All elements are set to zero.
  Parameters, such as Idle, Start_Bit, Data_Bit, Stop_Bit, and finish, are defined to represent different states of the state machine.
  Registers (r_Clock_Count, r_Bit_Index, Chain_Code, Packet_Done, state, start_pix) are declared to store various internal states and control variables.
  The initial values for these registers are set.
- Serial Data Reception:
  The state machine starts in the Idle state, waiting for the start bit.
  Upon detecting a start bit, it transitions to the Start_Bit state.
  In the Start_Bit state, it checks the middle of the start bit to ensure it's still low. If valid, it transitions to the Data_Bit state; otherwise, it goes back to the Idle state.
  In the Data_Bit state, the module waits for CLKS_PER_BIT-1 clock cycles to sample serial data. It updates the Chain_Code based on the received bits.
  After receiving all bits, it transitions to the Stop_Bit state.
- Pixel Processing:
  If the Packet_Done signal is high (indicating completion of receiving a packet), the module processes the received chain code.
  A case statement is used to determine the direction based on the chain code and update the current pixel coordinates (current_x and current_y).
  If the chain code indicates a completed packet (4'b1000), the done signal is set high.
- Initial Start Pixel:
  If the start signal (start) is active and the start_pix flag is set, the module initializes the pixel coordinates (current_x and current_y) with the values provided (start_pixel_x and start_pixel_y). This is done only once, ensuring that the start pixel is set at the beginning of the process.
- Pixel Storage:
  After a packet is done, the module stores the processed pixels in the memory array.
  It opens a file ("pixels.txt") and writes the content of the memory array into the file, representing the pixels in a binary format.
- Error Checking:
  The module checks if the current pixel coordinates (current_x and current_y) are within the valid range (0 to 63). If not, it sets the error signal high to indicate an error condition.
- Packet Completion:
  The state machine signals the completion of receiving and processing a packet through the Packet_Done signal.
  After processing a packet, the state transitions to the finish state to reset and prepare for the next packet.

Universal Asynchronous Receiver-Transmitter (UART) Communication:

Universal Asynchronous Receiver-Transmitter (UART) is a widely used serial communication protocol that enables data exchange between two devices. It is particularly prevalent in embedded systems, where low-cost and simplicity are essential. UART communication is asynchronous, meaning that the transmitting and receiving devices are not synchronized by a shared clock signal. Instead, they agree on a specific data rate, known as the baud rate, to facilitate communication.

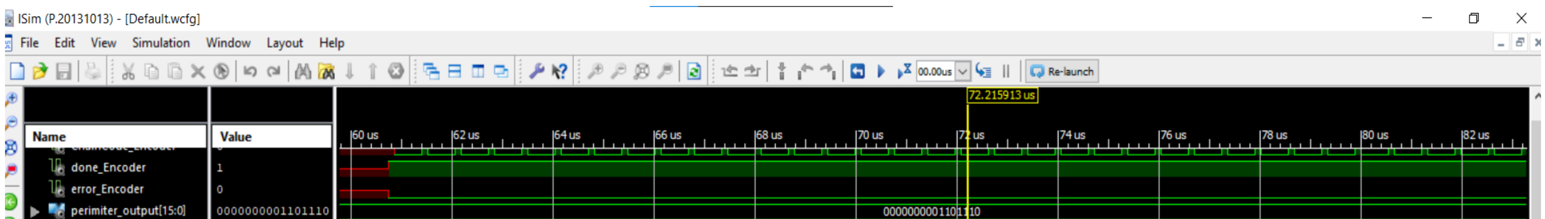
Key Components:

- Start Bit:
  A UART communication frame begins with a start bit, which is a high-to-low transition on the communication line. This start bit signals the beginning of a data packet and allows the receiver to synchronize its internal clock with the incoming data.

- Data Bits:
  Following the start bit, a fixed number of data bits (typically 8 bits) represent the actual information being transmitted. These bits encode the data in binary form, allowing for a wide range of characters and data types to be conveyed. In this project we send 4 bit data.

- Stop Bit(s):
  The data frame concludes with one or more stop bits (usually one or two). Stop bits are high-level signals that indicate the end of the data packet and provide the receiver with time to prepare for the next frame.

Simulation screenshot:



Name	Value
done_Encoder	1
error_Encoder	0
perimiter_output[15:0]	0000000001101110