Image Processing Toolkit 1 (IPT1) on FPGA

Digital Systems Week 1 Report

Problem Objective:
Establish UART Communication with Basys3 FPGA by writing a Python wrapper. Prove data transmission using a simple implementation of an adder circuit.

Description of different Modules:

1) UART Receive on FPGA:
Name:- RxUART
The UART receiver module operates based on a state machine design synchronized by the system clock (clk). It receives serial data (RxD) and synchronizes it to the system clock.The module however is activated only when 2063 (n) clock edges of the W5 pin of basys3 are counted, which is essentially a slow clock for the module. The number 2063 is decided by the clock frequency ($f = 100\,MHz$), baud rate ($b = 9600\,Hz$) and no of times the incoming bit is getting oversampled($k = 4$). The final count n is given by:-

$$n = \frac{f}{b \bullet k}$$

The module begins in an idle state waiting for the start bit, indicated by RxD transitioning from high to low. Once the start bit is detected, the module transitions to the receiving state. During reception, it samples the incoming data at multiple points within each bit interval to ensure accurate reception. After sampling, it shifts the received bits into a shift register (rxshiftreg). The module counts the received bits using a bitcounter and maintains a samplecounter to perform oversampling for robust data capture. Once all data bits have been received, the module signals the completion by asserting a flag. Additionally, it resets the bit and sample counters to prepare for the next reception. Finally, it outputs the received data (RxData) and a control signal (bhejdo), which can be used to indicate when valid data has been received.

2) UART Transmit module for FPGA:
Name:- TxUART
The UART transmission module is designed for sending 8-bit data in sequence through an FPGA. It relies on a clock signal (`clk`) to operate a state machine. The transmission module also activated only when 10416 clock signals of the W5 pin of basys3 are counted (this is to set a baud rate of 9600, calculated by dividing 100 MHz by 9600). Initially, it waits in an idle state until a transmission request (`transmit`) is received, with a reset signal (`reset`) ensuring synchronization. Upon receiving a transmission request and when not in a reset state, it

switches to a transmission state. During this phase, the data to be transmitted is loaded into a shift register (`rightshiftreg`) along with a start bit(0) at the LSB and a stop bit(1) at the MSB. Subsequently, the module shifts the data out serially, one bit at a time, by changing the `shift` signal. The `bitcounter` keeps track of the number of transmitted bits. Once all bits, including start and stop bits, have been sent, the module resets the `bitcounter` and returns to the idle state. The transmitted data is then available at the `TxD` output port, ready for transmission over the UART communication line.

3) UART Top Module to implement adder:

Name:- topmodule

The top module integrates UART receiving and transmission functionalities with an adder circuit on an FPGA. Upon receiving 8-bit numbers serially through the UART receiver module, the received data is accumulated to compute the sum. This data accumulation process is controlled by a flag triggered by the UART receiver's output signal (`bhejdo`). Each received 8-bit number is added to the `stored` variable, representing the running sum. Once all numbers are received and added, the sum is transmitted back to the computer via the UART transmitter module. The transmission process is initiated by the flag indicating the completion of data reception. The `stored` variable holding the sum is transmitted serially through the UART transmitter. The module is a parameterised code for n 8 bit number addition. The top module has a master reset that resets the overall sum by flushing out the stored data hence making it 0 and resetting the iterator i back to 0. Overall, this design enables bidirectional communication between the FPGA and the computer, allowing for the reception of data, computation, and subsequent transmission of results.

4) Python Code for Interfacing:

Name:- SerialSendRecieve.py

Our Python wrapper effectively establishes UART communication via the `pySerial` module, providing a straightforward method to interface with device (e.g., laptop) ports for serial data transmission and reception. The device-side baud rate, byte size, parity, and number of stop bits are configured to match the settings programmed on the FPGA, preventing communication mismatches.

The code prompts the user to input numbers, which are subsequently transmitted to the FPGA where the addition computation is performed. The numbers are sent one by one, and the FPGA adds the numbers to the final sum as they are received. The results are stored in an 8-bit register. Upon the user inputting the string 'add', the loop exits, signaling that all numbers have been entered and it's time to retrieve the computation result from the FPGA. The data is read back as a byte object in Python, which is then converted into an integer for display. The read operation retrieves a single byte which represents the final sum. The receiving order of the bits (that arrive one by one) is the LSB first to the MSB at the end. To avoid potential data corruption due to residual data in the buffers, both the output and input buffers are cleared regularly.