EE533: Network Processor Design & Programming

Lab #8: Convertible First-In-First-Out (SRAM) Memory

Instructor: Prof. Young Cho, PhD

Team Number: **#3**

Project Partners:

Member #1: **Sarthak Jain**

Member #2: **Archit Sethi**

Member #3: **Justin Santos**

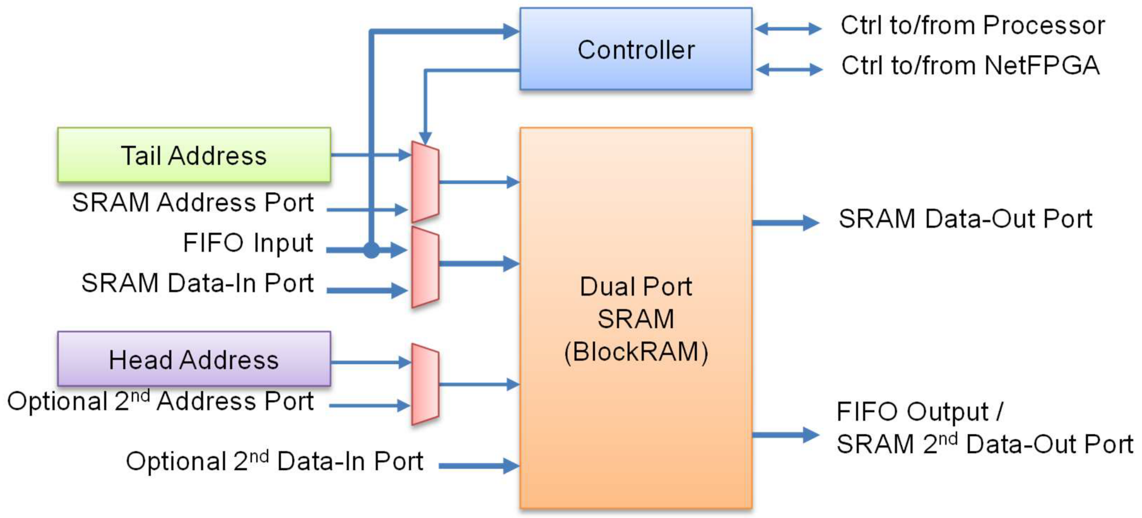
Designed, Created, and Submitted by Team #3

University of Southern California

Los Angeles, CA 90007

Team #4 GitHub Repository Link: *Team#3 Link*

1. Design and Implementation
    a. FIFO Structure: The FIFO Structure is built using dual-port SRAM which enables the read and write operations simultaneously. The architecture consists of:
        i. Memory block (SRAM-based)
            1. Stores the incoming network packets
            2. It is organized as a Circular Buffer
        ii. Head and Tail Pointers
            1. The HEAD pointer displays the read position
            2. The TAIL pointer marks the write position
            3. These 2 pointers are updated when the data is being read or written
        iii. Control signals
            1. FIFO FULL: This control signal prevents overwriting when the buffer is FULL.
            2. FIFO EMPTY: It indicates no data is available.
            3. Data Ready: It signals when a complete packet is stored in FIFO.
    b. Block Diagram

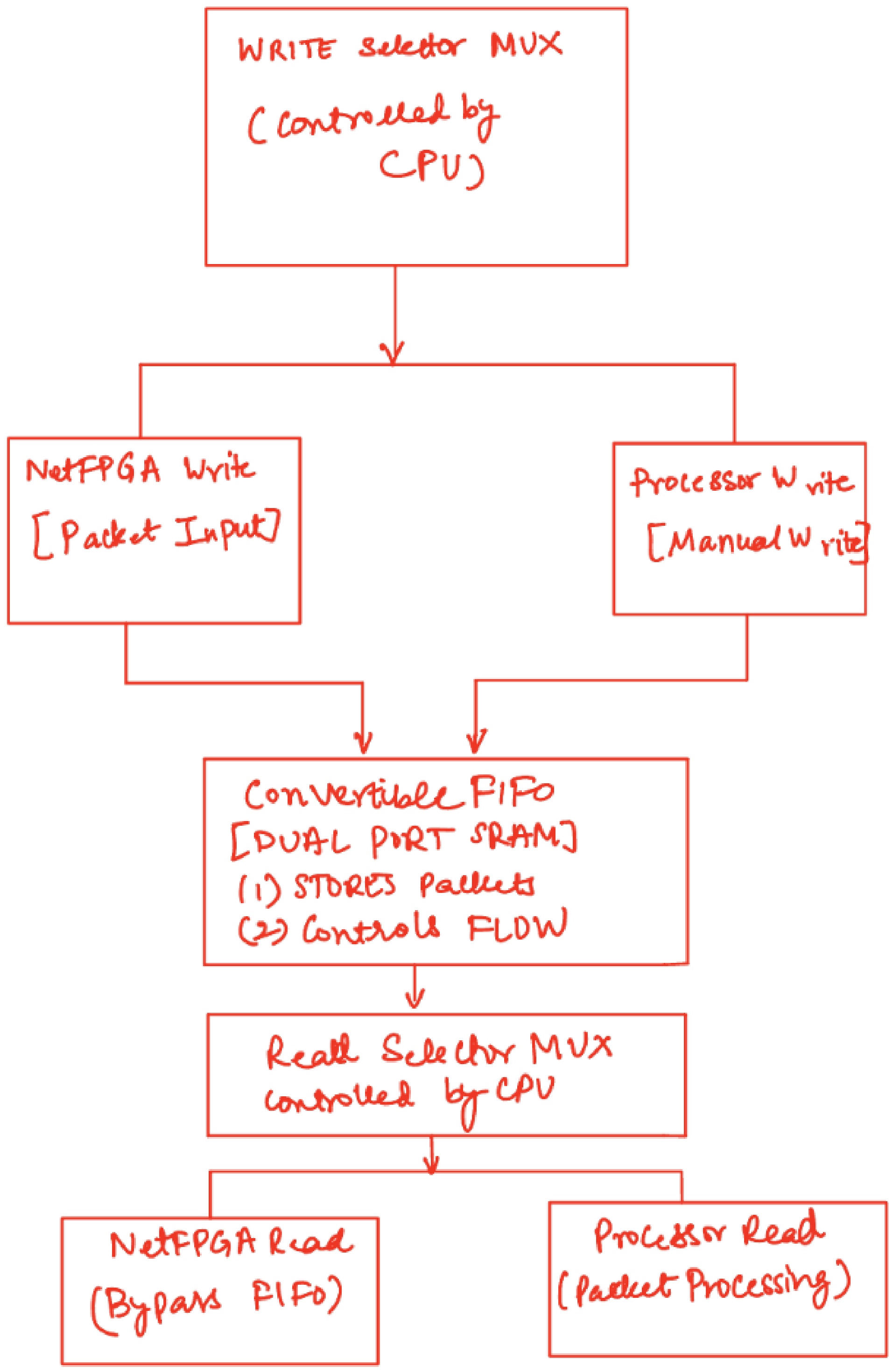


Let us explain the complete Block diagram data flow step-by-step:
Dual Port SRAM: This is the actual memory storage for our FIFO Design. It has 2 independent address and data ports which allows to write data and another port to read data.

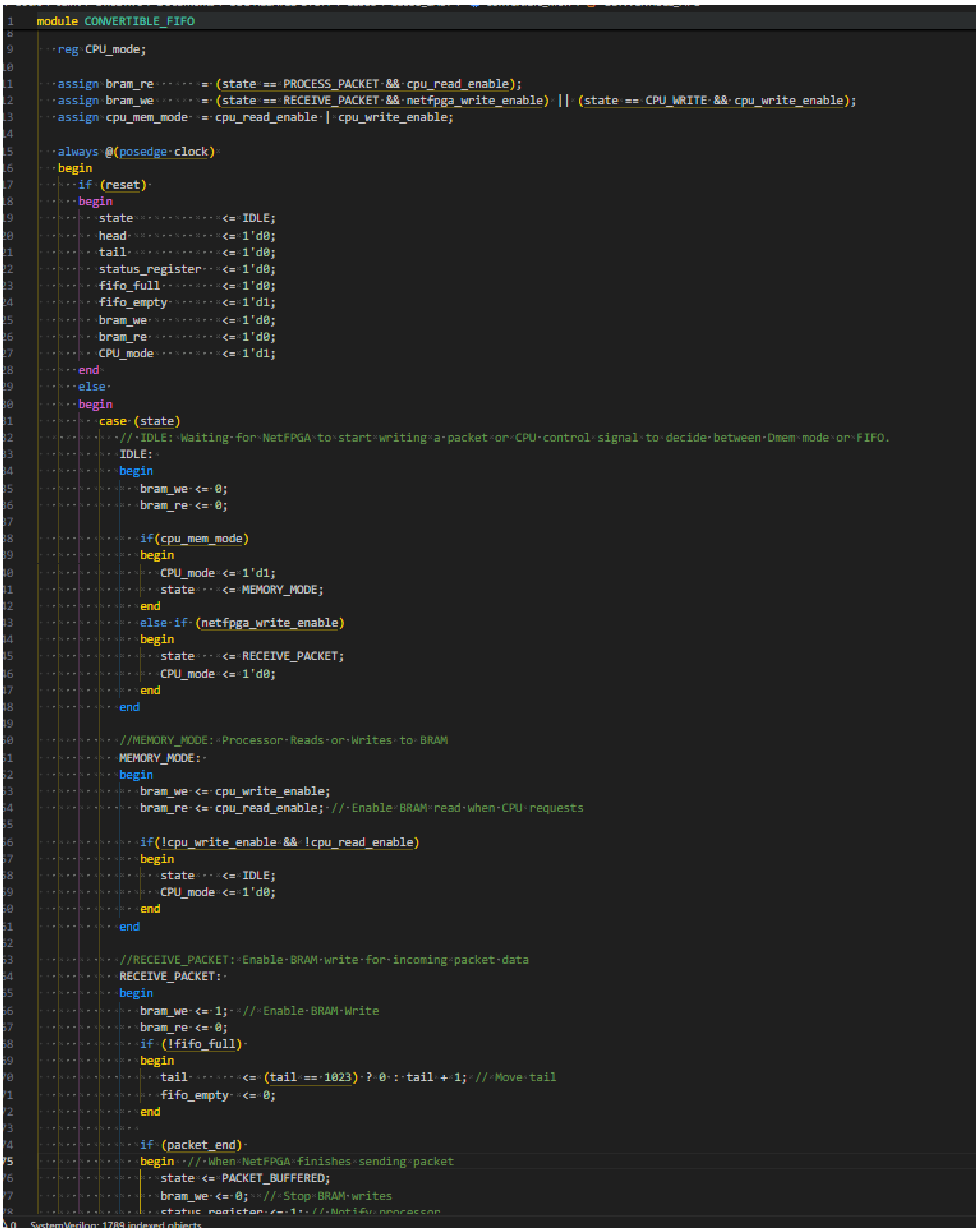
Address and Data Ports:

Port Name	**Purpose**	**Controlled By**
SRAM Address Port	Points to the memory location where data will be read/written	Tail Address (write), Head Address (read)
SRAM Data-In Port	Data input into the FIFO (from NetFPGA)	FIFO Input
FIFO Output / SRAM 2nd Data-Out Port	Data output (to the processor)	Head Address (read pointer)
Optional 2nd Address/Data Port	Allows a second processor or interface to read/write data	Not always used, but allows more flexibility

Multiplexer (MUX) Integration: In order to allow the NetFPGA and the processor to access FIFO, we used a MUX to select between (a) NetFPGA's Datapath-for writing incoming packets & (b) Processor's MEM stage for reading and processing packets.The BRAM is acting as both FIFO and Processor's MEM stage.

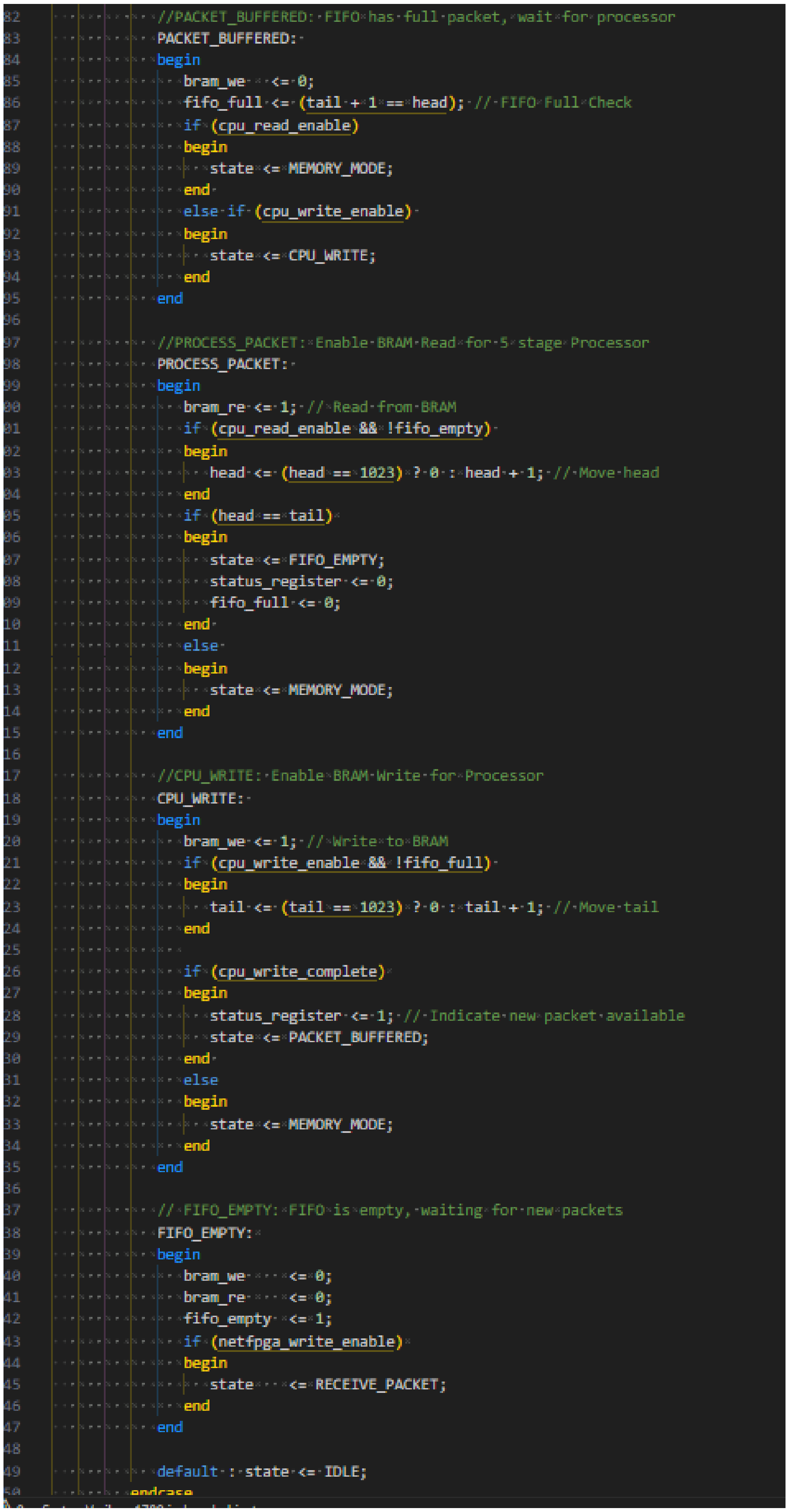

WRITE selector MUX
(Controlled by CPU)
NetFPGA Write
[Packet Input]
Processor Write
[Manual Write]
Convertible FIFO
[DUAL PORT SRAM]
(1) STORES Packets
(2) Controls FLOW
Read Selector MUX
controlled by CPU
NetFPGA Read
(Bypass FIFO)
Processor Read
(Packet Processing)

2. Simulation *Convertible FIFO.v:*



```
module CONVERTIBLE_FIFO

  reg CPU_mode;

  assign bram_re       = (state == PROCESS_PACKET && cpu_read_enable);
  assign bram_we       = (state == RECEIVE_PACKET && netfpga_write_enable) || (state == CPU_WRITE && cpu_write_enable);
  assign cpu_mem_mode  = cpu_read_enable | cpu_write_enable;

  always @(posedge clock)
  begin
    if (reset)
    begin
      state             <= IDLE;
      head              <= 1'd0;
      tail              <= 1'd0;
      status_register   <= 1'd0;
      fifo_full         <= 1'd0;
      fifo_empty        <= 1'd1;
      bram_we           <= 1'd0;
      bram_re           <= 1'd0;
      CPU_mode          <= 1'd1;
    end
    else
    begin
      case (state)
        // IDLE: Waiting for NetFPGA to start writing a packet or CPU control signal to decide between Dmem mode or FIFO.
        IDLE:
        begin
          bram_we <= 0;
          bram_re <= 0;

          if(cpu_mem_mode)
          begin
            CPU_mode <= 1'd1;
            state    <= MEMORY_MODE;
          end
          else if (netfpga_write_enable)
          begin
            state    <= RECEIVE_PACKET;
            CPU_mode <= 1'd0;
          end
        end

        //MEMORY_MODE: Processor Reads or Writes to BRAM
        MEMORY_MODE:
        begin
          bram_we <= cpu_write_enable;
          bram_re <= cpu_read_enable; // Enable BRAM read when CPU requests

          if(!cpu_write_enable && !cpu_read_enable)
          begin
            state    <= IDLE;
            CPU_mode <= 1'd0;
          end
        end

        //RECEIVE_PACKET: Enable BRAM write for incoming packet data
        RECEIVE_PACKET:
        begin
          bram_we <= 1;  // Enable BRAM Write
          bram_re <= 0;
          if (!fifo_full)
          begin
            tail        <= (tail == 1023) ? 0 : tail + 1; // Move tail
            fifo_empty  <= 0;
          end

          if (packet_end)
          begin  // When NetFPGA finishes sending packet
            state <= PACKET_BUFFERED;
            bram_we <= 0;  // Stop BRAM writes
            status_register <= 1; // Notify processor
```

```
0  SystemVerilog: 1789 indexed objects
```



```verilog
            //PACKET_BUFFERED: FIFO has full packet, wait for processor
            PACKET_BUFFERED: 
            begin
                bram_we   <= 0;
                fifo_full <= (tail + 1 == head); // FIFO Full Check
                if (cpu_read_enable)
                begin
                    state <= MEMORY_MODE;
                end 
                else if (cpu_write_enable) 
                begin
                    state <= CPU_WRITE;
                end
            end

            //PROCESS_PACKET: Enable BRAM Read for 5 stage Processor
            PROCESS_PACKET: 
            begin
                bram_re <= 1; // Read from BRAM
                if (cpu_read_enable && !fifo_empty) 
                begin
                    head <= (head == 1023) ? 0 : head + 1; // Move head
                end
                if (head == tail) 
                begin
                    state <= FIFO_EMPTY;
                    status_register <= 0;
                    fifo_full <= 0;
                end 
                else 
                begin
                    state <= MEMORY_MODE;
                end
            end

            //CPU_WRITE: Enable BRAM Write for Processor
            CPU_WRITE: 
            begin
                bram_we <= 1; // Write to BRAM
                if (cpu_write_enable && !fifo_full) 
                begin
                    tail <= (tail == 1023) ? 0 : tail + 1; // Move tail
                end
                
                if (cpu_write_complete) 
                begin
                    status_register <= 1; // Indicate new packet available
                    state <= PACKET_BUFFERED;
                end 
                else
                begin
                    state <= MEMORY_MODE;
                end
            end

            // FIFO_EMPTY: FIFO is empty, waiting for new packets
            FIFO_EMPTY: 
            begin
                bram_we     <= 0;
                bram_re     <= 0;
                fifo_empty  <= 1;
                if (netfpga_write_enable) 
                begin
                    state     <= RECEIVE_PACKET;
                end
            end

            default : state <= IDLE;
        endcase
```