EE533 Lab 3 Report

Project GitHub Link: https://github.com/jeremycai99/EE533-Labs/tree/main/Lab3

Introduction

In this lab, we are building an Intrusion Detection System (mini-IDS) using schematic and dual port BRAM IP core with simple verilog. We simulate the design in both schematic and extracted verilog source files from the schematic.

Part1: Recreate Schematic in Lab3 Project

Since the schematics are given, we can simply reproduce the schematics listed in the lab manual:

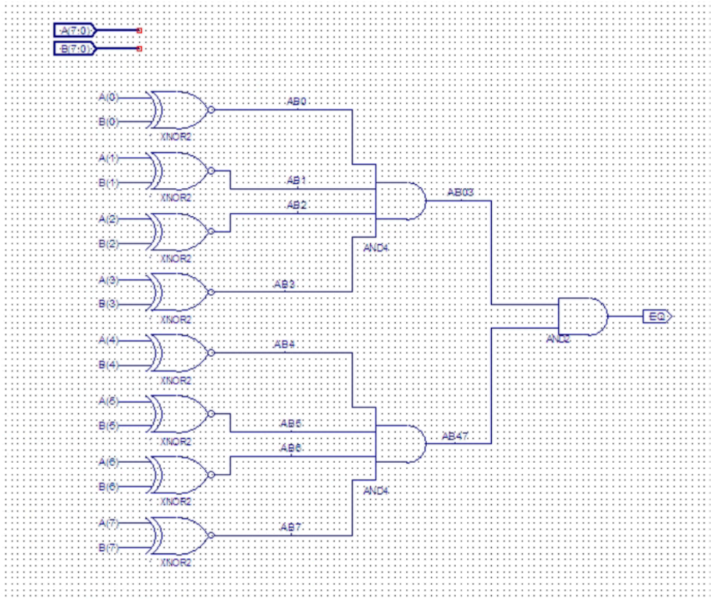


Figure 1. comp8.sch (Optional Since Built-In in ISE)

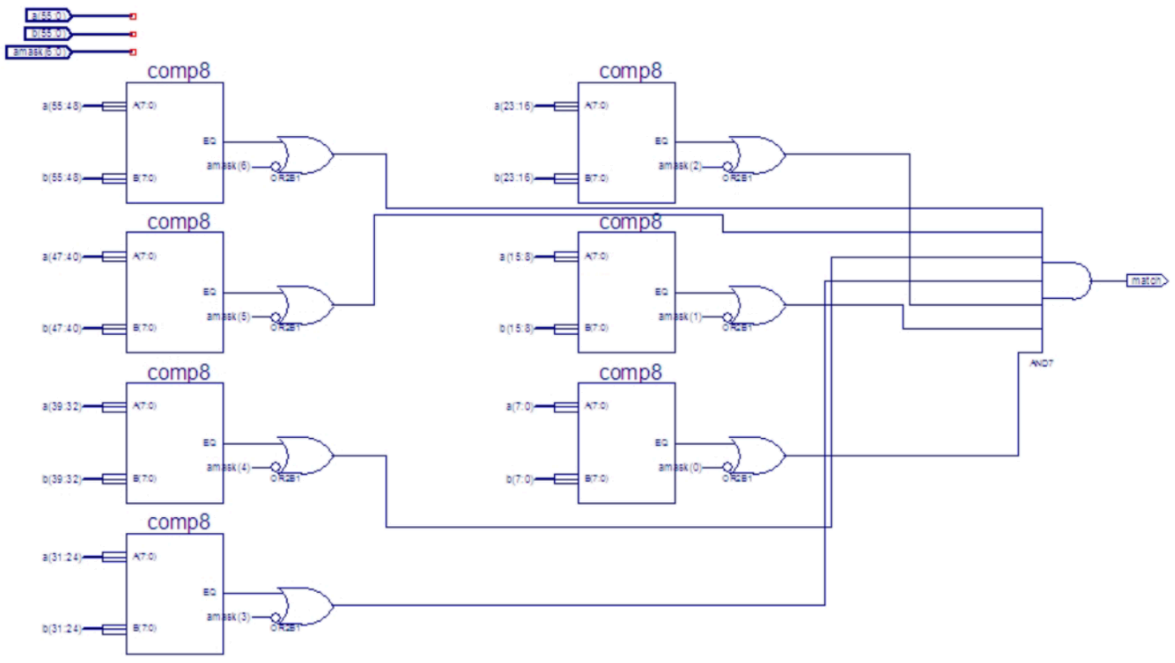


Figure 2. comparator.sch

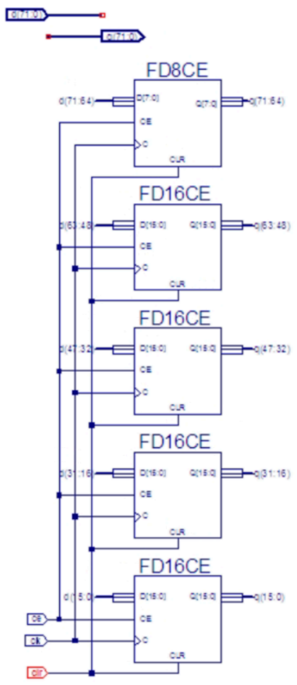


Figure 3. reg9B.sch

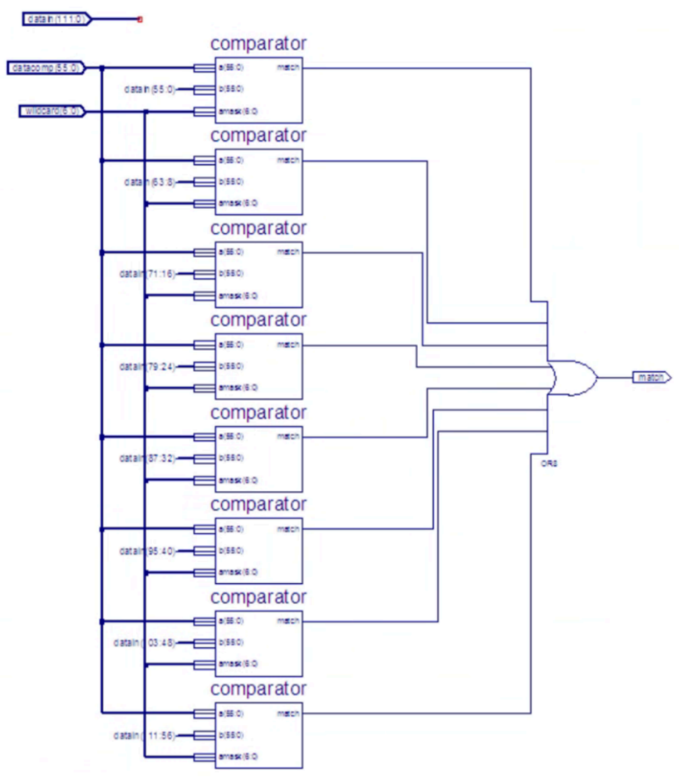


Figure4. wordmatch.sch

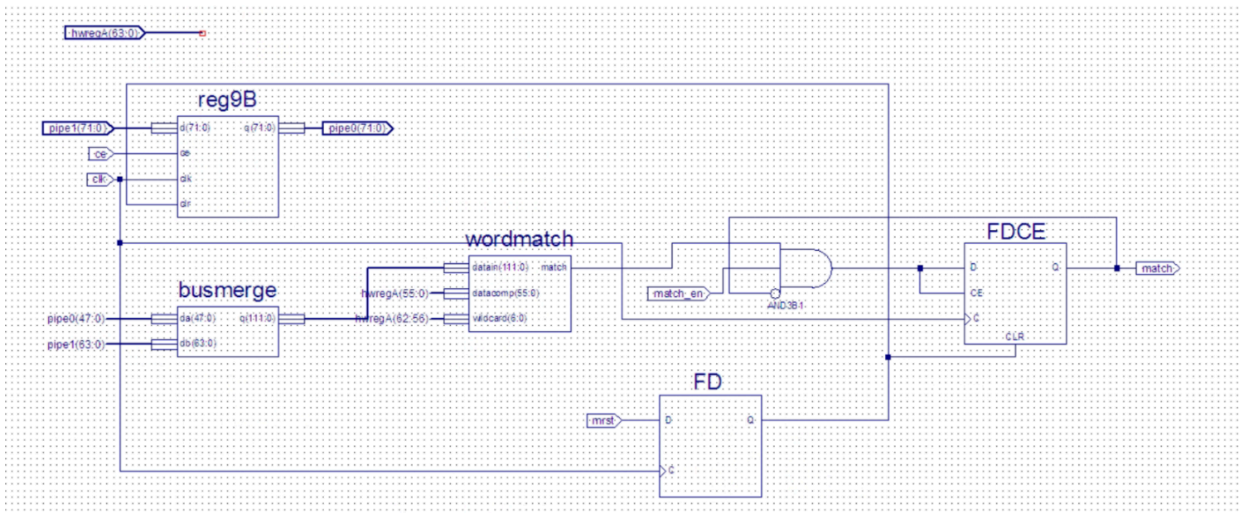


Figure 5. detect7B.sch

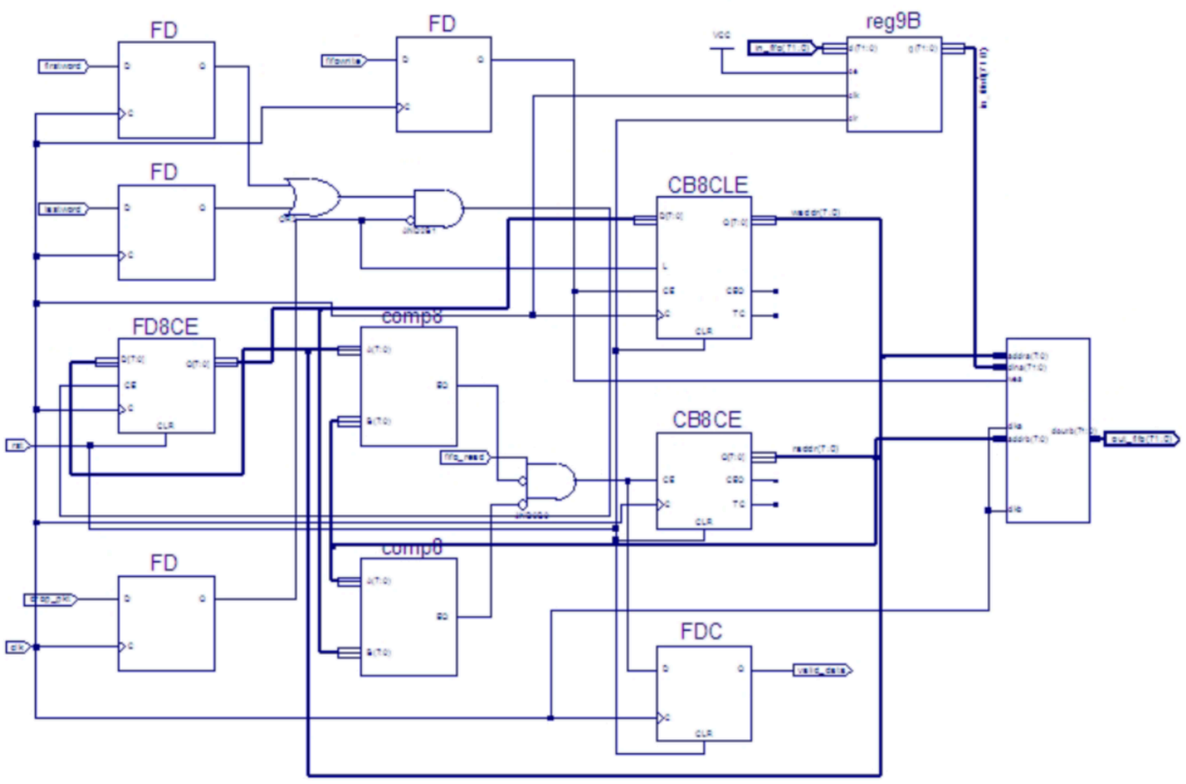
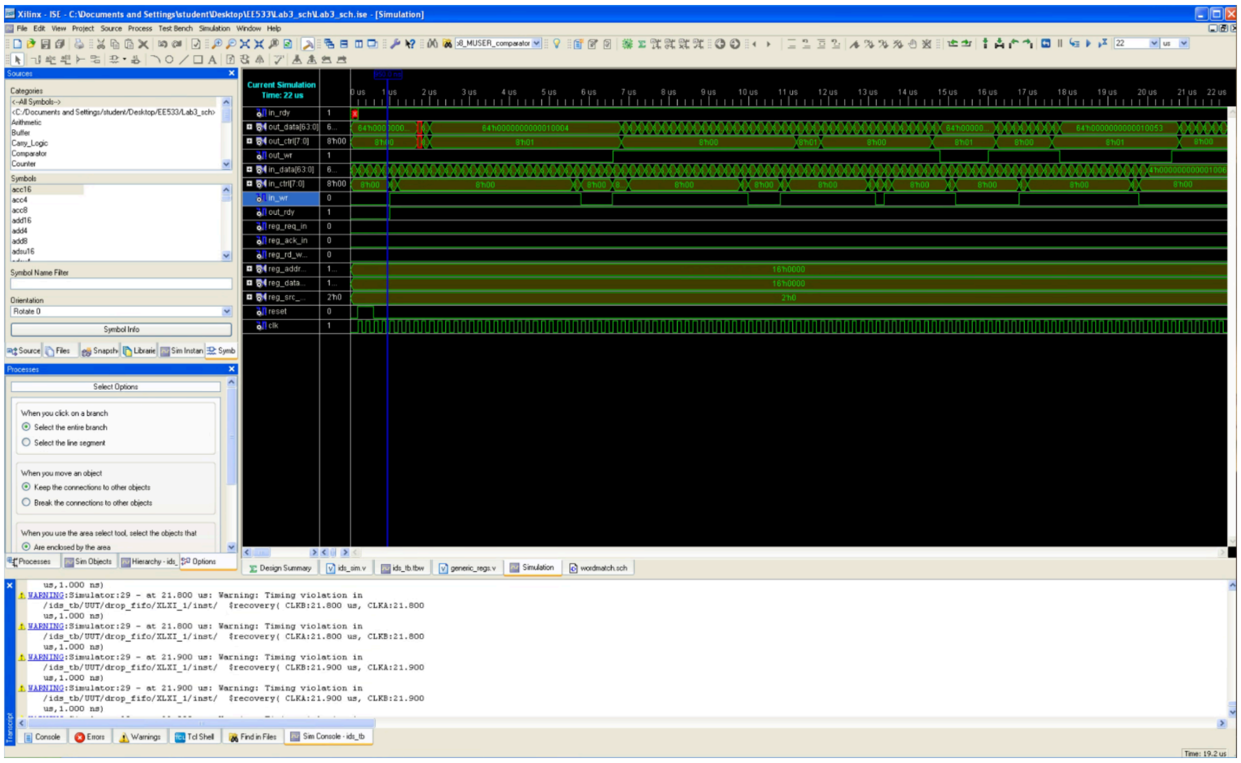


Figure 6. dropfifo.sch

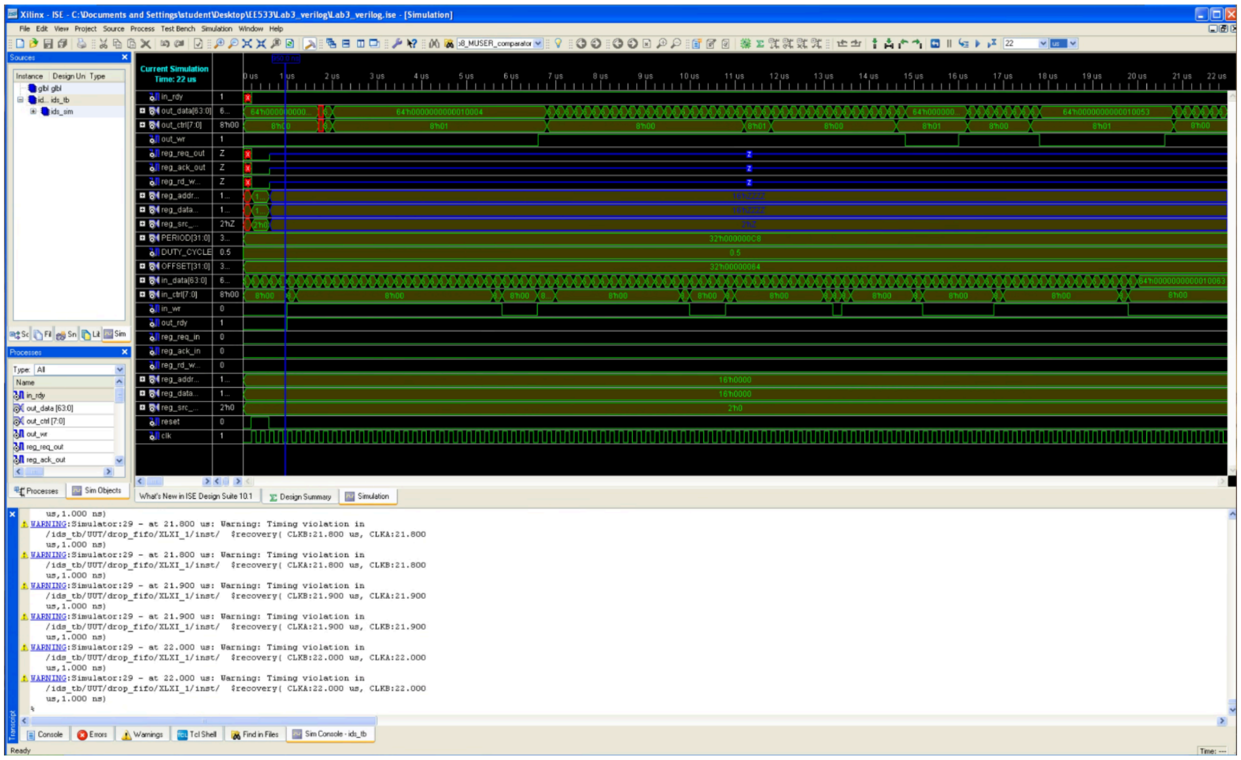
Part 2: Simulation Result Using `ids_sim.v` and `ids_tb.tbw`

In this part, we use provided source code and testbench waveform to perform behavioral simulation on both schematic based design and verilog based project extracted from schematic by Xilinx sch2verilog tool.



```
    us,1.000 ns)
WARNING:Simulator:29 - at 21.800 us: Warning: Timing violation in
    /ids_tb/UUT/drop_fifo/XLXI_1/inst/  $recovery( CLKB:21.800 us, CLKA:21.800
    us,1.000 ns)
WARNING:Simulator:29 - at 21.800 us: Warning: Timing violation in
    /ids_tb/UUT/drop_fifo/XLXI_1/inst/  $recovery( CLKA:21.800 us, CLKB:21.800
    us,1.000 ns)
WARNING:Simulator:29 - at 21.900 us: Warning: Timing violation in
    /ids_tb/UUT/drop_fifo/XLXI_1/inst/  $recovery( CLKB:21.900 us, CLKA:21.900
    us,1.000 ns)
WARNING:Simulator:29 - at 21.900 us: Warning: Timing violation in
    /ids_tb/UUT/drop_fifo/XLXI_1/inst/  $recovery( CLKA:21.900 us, CLKB:21.900
    us,1.000 ns)
```

Figure 7. Simulation Result in Schematic Based Project



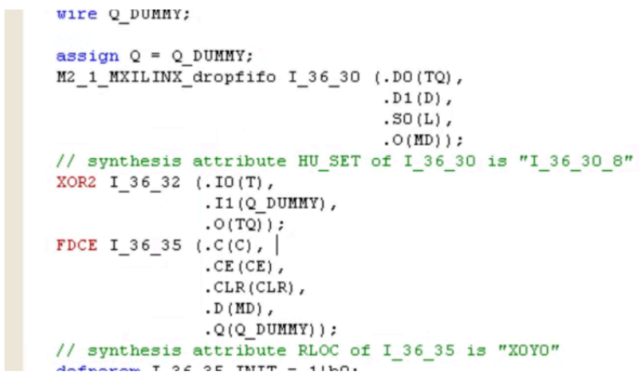
```
    us,1.000 ns)
WARNING:Simulator:29 - at 21.800 us: Warning: Timing violation in
    /ids_tb/UUT/drop_fifo/XLXI_1/inst/  $recovery( CLKB:21.800 us, CLKA:21.800
    us,1.000 ns)
WARNING:Simulator:29 - at 21.800 us: Warning: Timing violation in
    /ids_tb/UUT/drop_fifo/XLXI_1/inst/  $recovery( CLKA:21.800 us, CLKB:21.800
    us,1.000 ns)
WARNING:Simulator:29 - at 21.900 us: Warning: Timing violation in
    /ids_tb/UUT/drop_fifo/XLXI_1/inst/  $recovery( CLKB:21.900 us, CLKA:21.900
    us,1.000 ns)
WARNING:Simulator:29 - at 21.900 us: Warning: Timing violation in
    /ids_tb/UUT/drop_fifo/XLXI_1/inst/  $recovery( CLKA:21.900 us, CLKB:21.900
    us,1.000 ns)
WARNING:Simulator:29 - at 22.000 us: Warning: Timing violation in
    /ids_tb/UUT/drop_fifo/XLXI_1/inst/  $recovery( CLKB:22.000 us, CLKA:22.000
    us,1.000 ns)
WARNING:Simulator:29 - at 22.000 us: Warning: Timing violation in
    /ids_tb/UUT/drop_fifo/XLXI_1/inst/  $recovery( CLKA:22.000 us, CLKB:22.000
    us,1.000 ns)
```

Figure 8. Simulation Result in Verilog HDL Based Project

Part 3: Q & A

TAKE A LOOK AT THE CREATED VERILOG. DO THEY MAKE SENSE?

From the generated verilog, apart from the auto-naming of the module, it reflects the hierarchical design what we implement in the schematic. However, for the `dropfifo.vf`, it can be not straightforward since there are logic gates generated that never appear in the schematic design. For example, XOR gate and dummy wire (Q_DUMMY) are generated out of nowhere but the overall logic is legit.



```
wire Q_DUMMY;

assign Q = Q_DUMMY;
M2_1_MXILINX_dropfifo I_36_30 (.D0(TQ),
                               .D1(D),
                               .S0(L),
                               .O(MD));
// synthesis attribute HU_SET of I_36_30 is "I_36_30_8"
XOR2 I_36_32 (.I0(T),
              .I1(Q_DUMMY),
              .O(TQ));
FDCE I_36_35 (.C(C),
              .CE(CE),
              .CLR(CLR),
              .D(MD),
              .Q(Q_DUMMY));
// synthesis attribute RLOC of I_36_35 is "X0Y0"
```

Figure 9: XOR Instance and Dummy Logic Created

Since this phenomenon doesn't happen for simple designs, we argue that it's due to logic optimization or slight tweak from the tool for a better but logically equivalent design. Since the change is minor, we don't dive deep to root cause of it.

WHICH DO YOU THINK EASIER: ENTERING THE SCHEMATICS OR WRITING VERILOG? WHY? IN WHICH CASES MIGHT YOU DO THE OTHER?

Writing Verilog is easier than schematic design. Verilog can do both behavioral coding and hierarchical coding while schematic design can only be hierarchical/structural. Schematic design is a lot difficult since we need to understand the gate-level logic to an application. However, Schematic design can be highly competitive if the design is relatively small and the resource is extremely limited. Verilog requires design synthesis provided by the tool to map the logic into FPGA resources, and it's not an efficient way. For optimization-heavy design, schematic-based project can deliver most cost-efficient design with minimal resource usage and low latency. Well, considering the complexity of extensive optimization, I'd choose Verilog for more high-level design rather than gate-level awareness

EXPLAIN THE PATTERN MATCHING ALGORITHM IN THE REPORT

The pattern matching algorithm can detect 7-byte data packet or message without strict message begin or end boundaries. It enables the configuration of per-byte wildcard so that it can used to mask any bit that should not participate in the matching algorithm from the 56-bit (7B) input. To make sure it can find the signature even when it starts at different byte positions, the design builds a bigger buffer called datain[111:0] (14 bytes). It does this by keeping some bytes from the previous cycle and merging them with new bytes, so it can look across a boundary between two words. Then it checks the signature at 8 possible starting offsets inside that 14-byte buffer (start at byte 0, 1, …, 7). For each offset, it compares 7 bytes of data against the 7-byte signature. A byte matches if it is equal **or** that byte is marked as wildcard. If all 7 bytes match for an offset, that offset is a "hit." Finally, it ORs all 8 offset results together. If **any** offset hits, the module asserts match, meaning the signature was found somewhere in the recent data.

WHAT IS THE PURPOSE OF AMASK[6:0]?

AMASK[6:0] is set up as a wildcard to implement the masking logic for input 7B data. The byte would match any value if we set its corresponding AMASK bit to "0", meaning it’s a "don’t care" of these bytes.

WHAT EXACTLY DOES BUSMERGE.V DO?

busmerge.vmerges the lower 6 bytes of previous message (pipe0) and lower 8 bytes of current message (pipe1), which ensure that any consecutive 7-byte data can be detected by the wordmatch logic.

WHAT DO THE COMP8 MODULES DO IN THIS SCHEMATIC?

COMP8 performs a bit-by-bit comparison of the two inputs, and it only outputs 1 if every corresponding bit is equal. In comparator.sch, COMP8 compares the 7-byte data stream with target message. In dropfifo.sch, COMP8 compares read address with write address of dual port memory to generate output valid data.

WHAT IS THE PURPOSE OF DUAL9BMEM IN DROPFIFO.SCH?

In our design, the BRAM IP core is named dpmem9B but the configuration is identical. The dual9Bmem as a dual-port BRAM, is used as a data buffer in the dropfifo module. It can be configured to either retain or drop a data packet by controlling the read and write enable signals.