EE533 Network Processor Design & Programming

Lab #3: Mini Intrusion Detection System (mini-IDS)

Instructor: Prof. Young Cho, PhD

Name: Archit Sethi

University of Southern California

Los Angeles, CA 90007

GITHUB LINK to my REPOSITORY

Aarch0811/EE533/LAB3/

Archit's GITHUB: LINK

1. Take a look at the created Verilog. Do they make sense? Which do you think easier: entering the schematics or writing Verilog? Why? In which cases might you do the other?
   A.1. The script that I had generated appears unintelligible because the tool assigns highly specific instance names, wire names, and module names. Although the simulation functions correctly, interpreting the generated code is quite challenging.

   Personally, writing RTL logic for a large circuit like this would be far more manageable since the code can be modular and reused efficiently. High-level behavioural modelling in HDL significantly reduces the effort required for logic design compared to schematic-based implementation. Additionally, debugging is much more straightforward if any issues arise.

2. Download the Lab5_mini_ids_src.zip file.
   a. In this file you will find two directories.
   b. Extract ids_sim directory to your ISE project. You should now be able to simulate the mini-IDS using the ids_tb.tbw testbench. The other files are needed to emulate the pieces of the NetFPGA that are around your design. Run the testbench and take a screenshot. Describe what the testbench does and how it shows that the mini-IDS is functioning.

      Ans: For a mini-intrusion detection (mini-IDS) system, three key components are required: an input data stream, a pattern matching logic to identify specific intrusion patterns, and an output FIFO wrapper that discards detected intrusive packets while transferring the remaining data to a dual-port FIFO. This FIFO serves to isolate intrusion-free packets from the subsequent processing stages.

      The testbench consists of four main components:

      Detect7B Matcher: *This module identifies intrusive data bit patterns within the input stream received from the fall-through FIFO.*

      Fallthrough fifo: *This acts as a buffer for feeding packets into the pattern detection logic. It functions as a decoupler, isolating the input data stream from the pattern matching mechanism.*

      Dropfifo: *A First-In-First-Out (FIFO) designed to eliminate packets that contain identified intrusion patterns while passing the rest of the data through.*

      Packet Separation Logic: *A state machine responsible for extracting the header and payload from incoming packets. This enables a byte-by-byte comparison of the payload against predefined intrusion patterns for detection.*

3. Write and submit your lab report.
    a. Explain the pattern matching algorithm in the report.

Intrusion Detection System Overview: The concept of this intrusion detection system is to identify 7-byte malicious patterns within an incoming network data stream. The system processes 9-byte packets, consisting of 8 bytes of data and 1 byte for control signals. These packets pass through a fall-through FIFO before being analysed by the 7B pattern-matcher module. Since malicious patterns can be dispersed across multiple packets in various ways, all potential edge cases must be considered.

Consider a scenario where only 1 byte of the malicious sequence appears at the end of one packet, while the remaining 6 bytes continue into the next packet, must be addressed.

In addition, other combinations, such as 3 bytes in one packet and 4 bytes in the next, must be handled to ensure comprehensive detection. Addressing the worst-case scenario allows the system to be robust against fragmented malicious patterns.

System Components and Functionality: To construct a mini-IDS system, the following components are necessary:

Detect7B Matcher: *This module is responsible for scanning the incoming data stream to identify occurrences of the malicious 7-byte pattern. It takes input from the fall-through FIFO and performs pattern detection.*

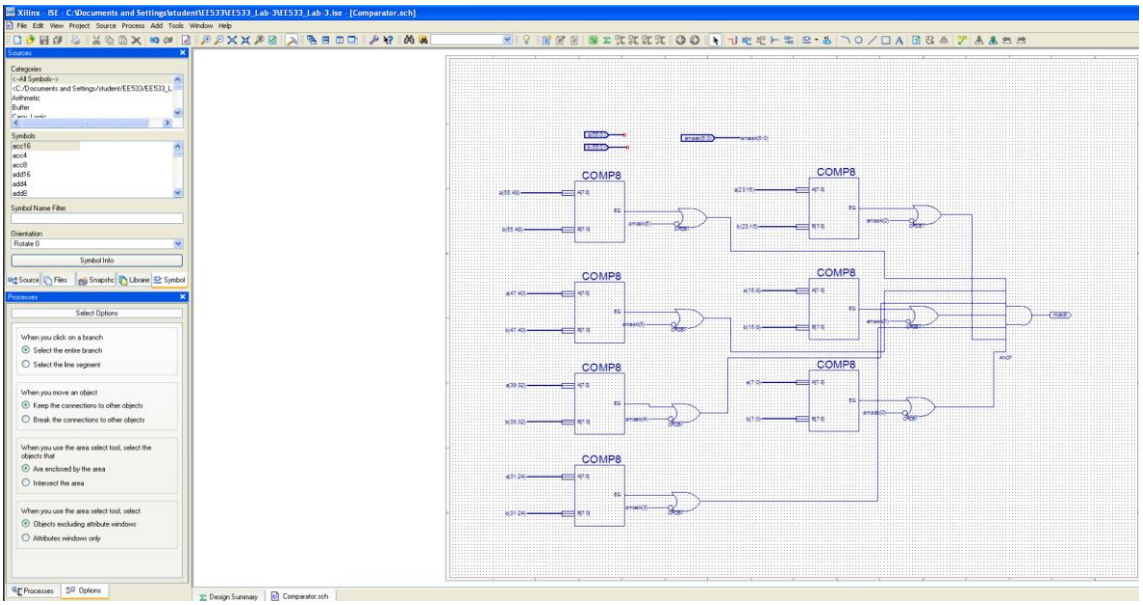
Fall-through fifo: *This FIFO serves as a buffer for feeding packets into the pattern detection logic. It acts as a decoupler, isolating the input data stream from the pattern matching process to prevent interference.*

Packet Separation Logic: *This module functions as a state machine, responsible for extracting headers and payloads from incoming packets. It enables byte-by-byte comparison of packet content with predefined intrusion patterns*

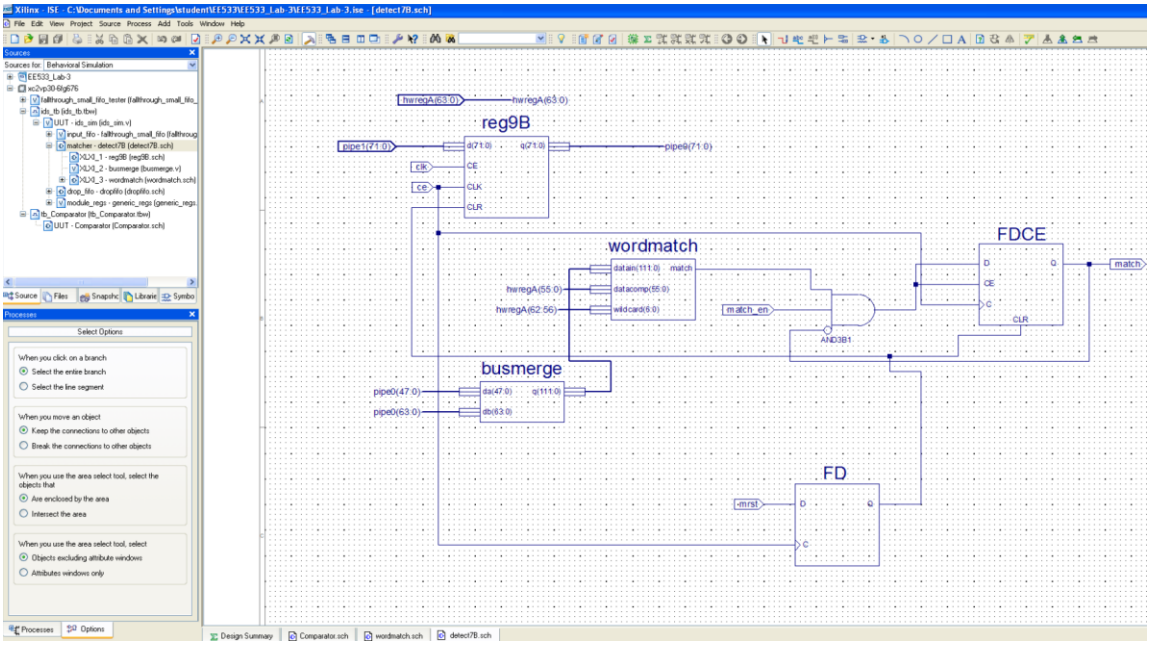
Dropfifo: *This FIFO is designed to eliminate packets that contain detected malicious patterns. Only packets that do not match the intrusion pattern are forwarded for further processing.*

Please find attached the figures of Schematics for comparator, Detect7B, wordmatch, dropfifo as follows:

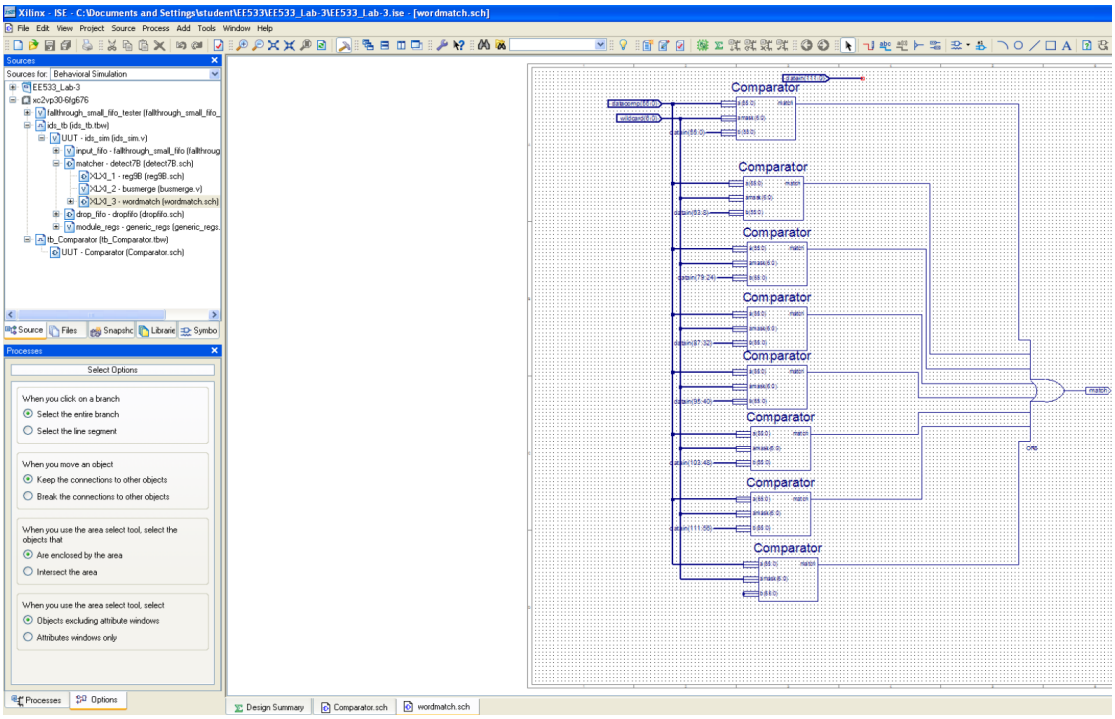
Comparator:



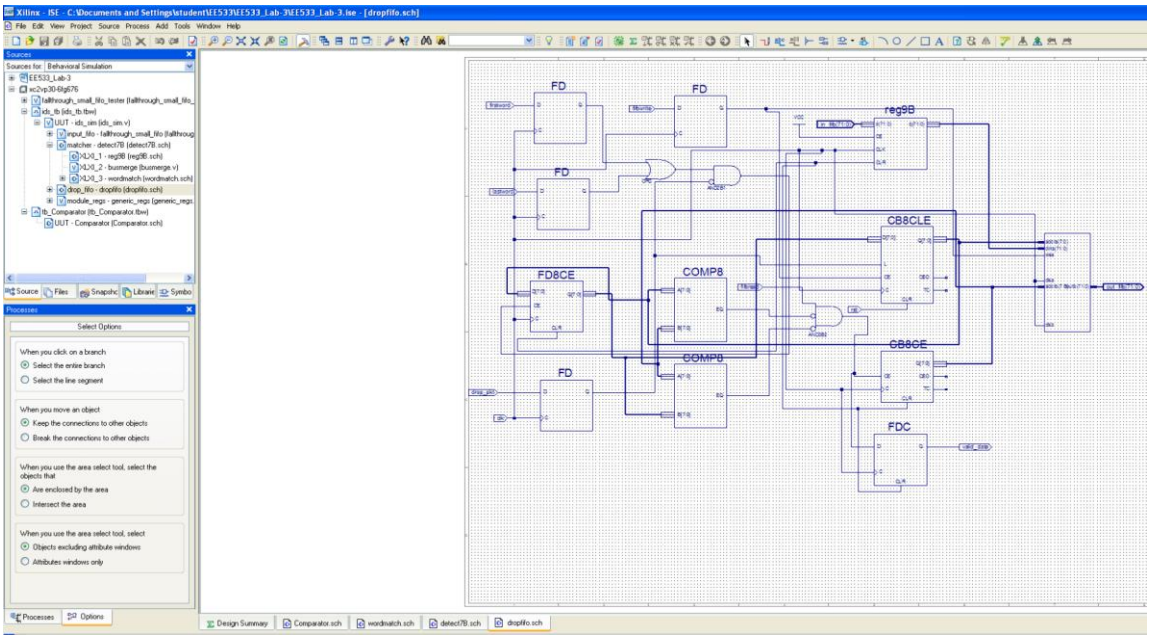
Detect7B:



Wordmatch:



Dropfifo:



b. Include the answers to your lab in this report.

   i. What is the purpose of AMASK[6:0]?

   Ans: The AMASK[6:0] register determines which bytes of the predefined pattern are compared against the incoming data stream. When we set to 7'b1111111, all 7 bytes are actively matched against the corresponding bytes in the input packet.

However, if any bit is cleared (it is set to 0), the respective byte is ignored in the comparison and treated as a don't care value. This selective matching approach enhances flexibility, allowing the system to adaptively compare relevant bytes rather than rigidly enforcing a full-pattern match in all scenarios.

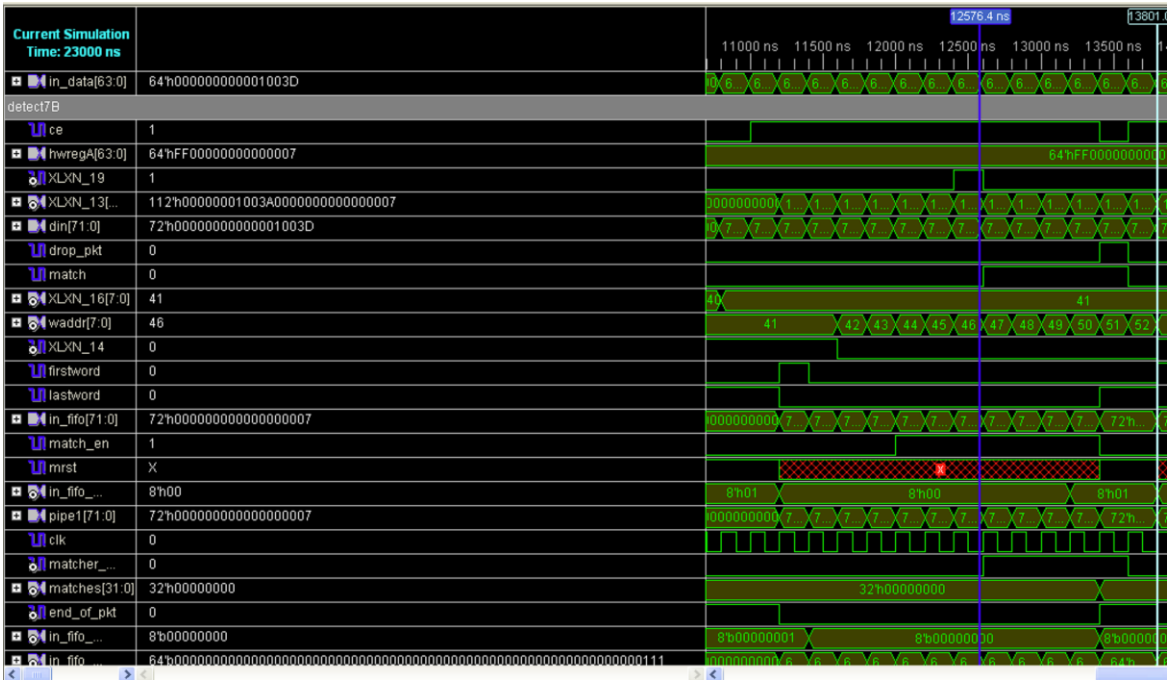
ii. What exactly does busmerge.v do?
*Ans: To perform a continuous **7-byte comparison**, we require two overlapping **7-byte segments** of input data. This ensures that if part of a pattern appears at the end of one chunk and the remaining portion appears in the next, it can still be detected.*

*The **busmerge.v** facilitates this by **concatenating 6 bytes from the previous data chunk** with the **8 bytes of newly received data** (effectively merging data from consecutive clock cycles). This combined data stream is then passed to the pattern matching logic for comparison.*

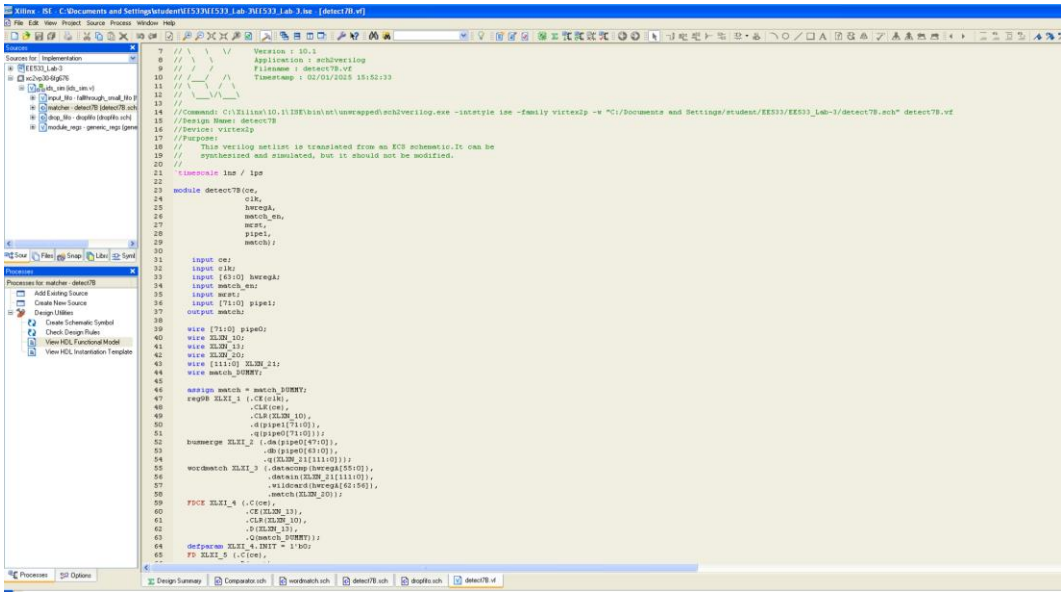
iii. What do the comp8 modules do in this schematic?
Ans: The comparator is responsible for analysing each byte of the incoming data packet and determining whether it matches the reference data byte from the pattern stream. It outputs a signal indicating whether there is a match or equality between the two.

iv. What is the purpose of dual19Bem in dropfifo.sch?
*Ans: The dual-port **9-byte memory** is designed to filter out pattern-matched malicious data, ensuring that only non-malicious data is stored. It consists of **8 bytes of data** along with **1 byte of control signals**, making a total of **9 bytes**. This FIFO, which exclusively admits verified safe data, serves as a buffer that effectively **isolates the processed data from the consuming system**.*

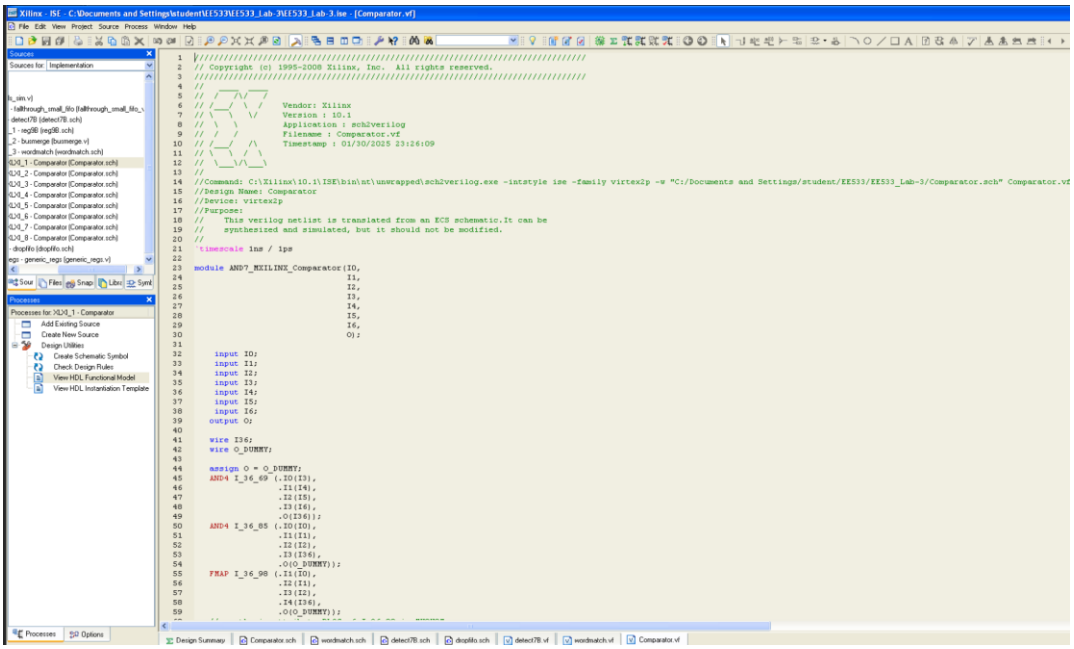
c. Please also include all of the screen capture as well as generated Verilog.
   i. Generated Waveform:



ii. Detect7b generated Verilog:

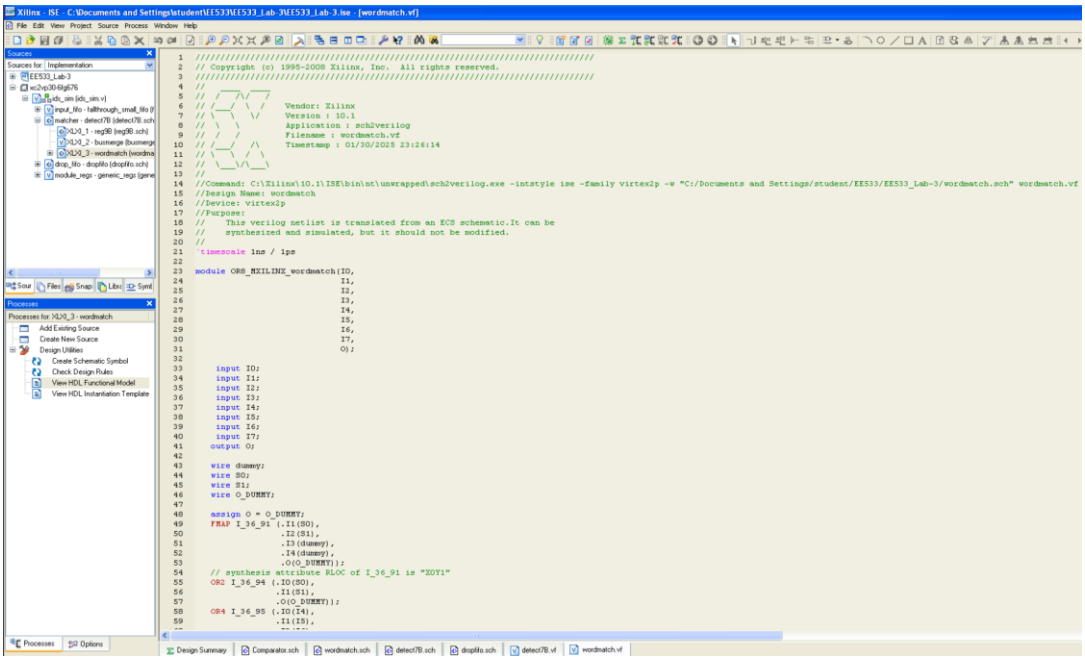


iii. Comparator generated Verilog:



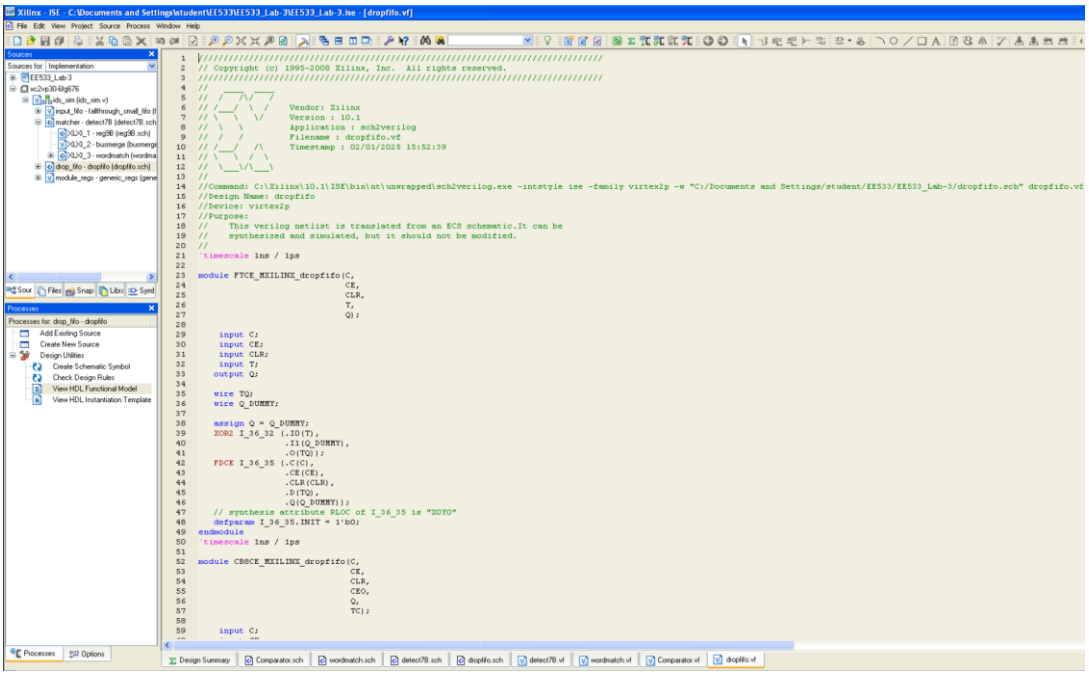
```
//////////////////////////////////////////////////////////////////////////////////
// Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.
//////////////////////////////////////////////////////////////////////////////////
//   ____  ____
//  /   /\/   /
// /___/  \  /    Vendor: Xilinx
// \   \   \/     Version : 10.1
//  \   \         Application : sch2verilog
//  /   /         Filename : Comparator.vf
// /___/   /\     Timestamp : 01/30/2025 23:26:09
// \   \  /  \
//  \___\/\___\
//
//Command: C:\Xilinx\10.1\ISE\bin\nt\unwrapped\sch2verilog.exe -intstyle ise -family virtex2p -w "C:/Documents and Settings/student/EE533/EE533_Lab-3/Comparator.sch" Comparator.vf
//Design Name: Comparator
//Device: virtex2p
//Purpose:
//    This verilog netlist is translated from an ECS schematic.It can be
//    synthesized and simulated, but it should not be modified.
//
`timescale 1ns / 1ps

module AND7_MXILINX_Comparator(IO,
                               I1,
                               I2,
                               I3,
                               I4,
                               I5,
                               I6,
                               O);

  input IO;
  input I1;
  input I2;
  input I3;
  input I4;
  input I5;
  input I6;
  output O;

  wire I36;
  wire O_DUMMY;

  assign O = O_DUMMY;
  AND4 I_36_69 (.IO(I3),
                .I1(I4),
                .I2(I5),
                .I3(I6),
                .O(I36));
  AND4 I_36_85 (.IO(IO),
                .I1(I1),
                .I2(I2),
                .I3(I36),
                .O(O_DUMMY));
  FMAP I_36_98 (.I1(IO),
                .I2(I1),
                .I3(I2),
                .I4(I36),
                .O(O_DUMMY));
```

iv. Wordmatch generated Verilog:

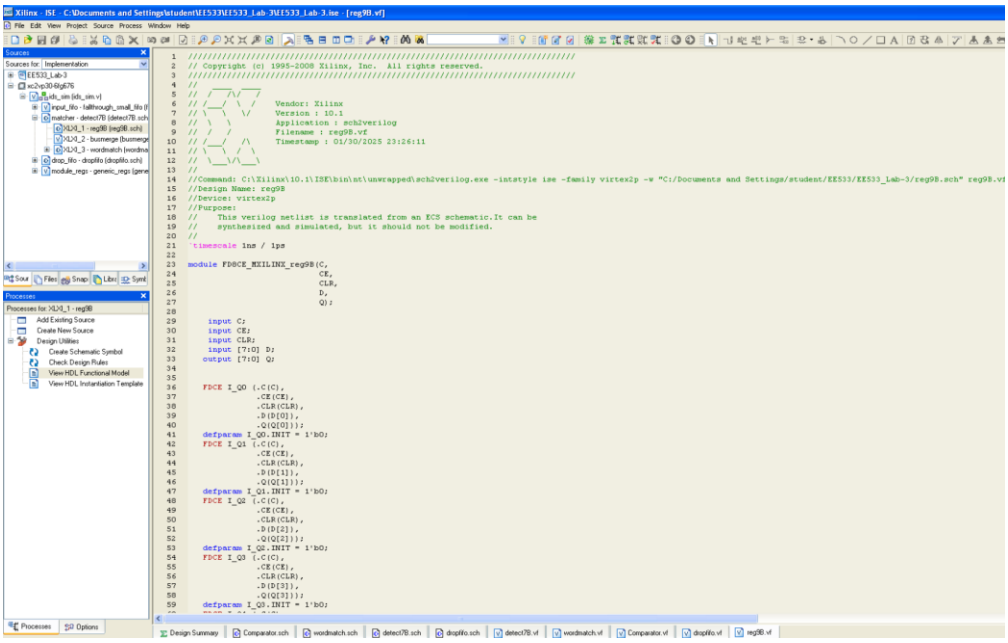


```
//////////////////////////////////////////////////////////////////////////////////
// Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.
//////////////////////////////////////////////////////////////////////////////////
//   ____  ____
//  /   /\/   /
// /___/  \  /    Vendor: Xilinx
// \   \   \/     Version : 10.1
//  \   \         Application : sch2verilog
//  /   /         Filename : wordmatch.vf
// /___/   /\     Timestamp : 01/30/2025 23:26:14
// \   \  /  \
//  \___\/\___\
//
//Command: C:\Xilinx\10.1\ISE\bin\nt\unwrapped\sch2verilog.exe -intstyle ise -family virtex2p -w "C:/Documents and Settings/student/EE533/EE533_Lab-3/wordmatch.sch" wordmatch.vf
//Design Name: wordmatch
//Device: virtex2p
//Purpose:
//    This verilog netlist is translated from an ECS schematic.It can be
//    synthesized and simulated, but it should not be modified.
//
`timescale 1ns / 1ps

module OR8_MXILINX_wordmatch(IO,
                             I1,
                             I2,
                             I3,
                             I4,
                             I5,
                             I6,
                             I7,
                             O);

  input IO;
  input I1;
  input I2;
  input I3;
  input I4;
  input I5;
  input I6;
  input I7;
  output O;

  wire dummy;
  wire S0;
  wire S1;
  wire O_DUMMY;

  assign O = O_DUMMY;
  FMAP I_36_91 (.I1(S0),
                .I2(S1),
                .I3(dummy),
                .I4(dummy),
                .O(O_DUMMY));
  // synthesis attribute RLOC of I_36_91 is "X0Y1"
  OR2 I_36_94 (.IO(S0),
               .I1(S1),
               .O(O_DUMMY));
  OR4 I_36_95 (.IO(I4),
               .I1(I5),
```

v. Dropfifo generated Verilog:



vi. Reg9B:



GitHub commits & history: