EE533: Network Processor Design & Programming

Lab #5: Debugging and Testing Custom NetFPGA Design

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 Lab-5 Link*

1. Introduction

a. Overview of Lab Objectives & Explanation of NetFPGA-based mini IDS

In this lab#5, we aim to debug and test a custom NetFPGA -based Intrusion Detection System (mini-IDS). The objective is to implement and verify the functionality of a passthrough module, integrate a logic analyzer, and incrementally test our mini-IDS design.

The system will be validated using packet injection, pattern matching, and performance analysis under high-speed network conditions.

The primary task in this lab include:

- Integrating and testing a passthrough module for baseline validation
- Implementing a logic analyzer to capture and analyze internal signals
- Incrementally adding and testing mini-IDS components
- Modifying a register access script (idsreg) to interact with hardware registers
- Testing the pattern matching mechanism using traffic generation tools (iperf)
- Verifying correct packet blocking behavior tcpdump

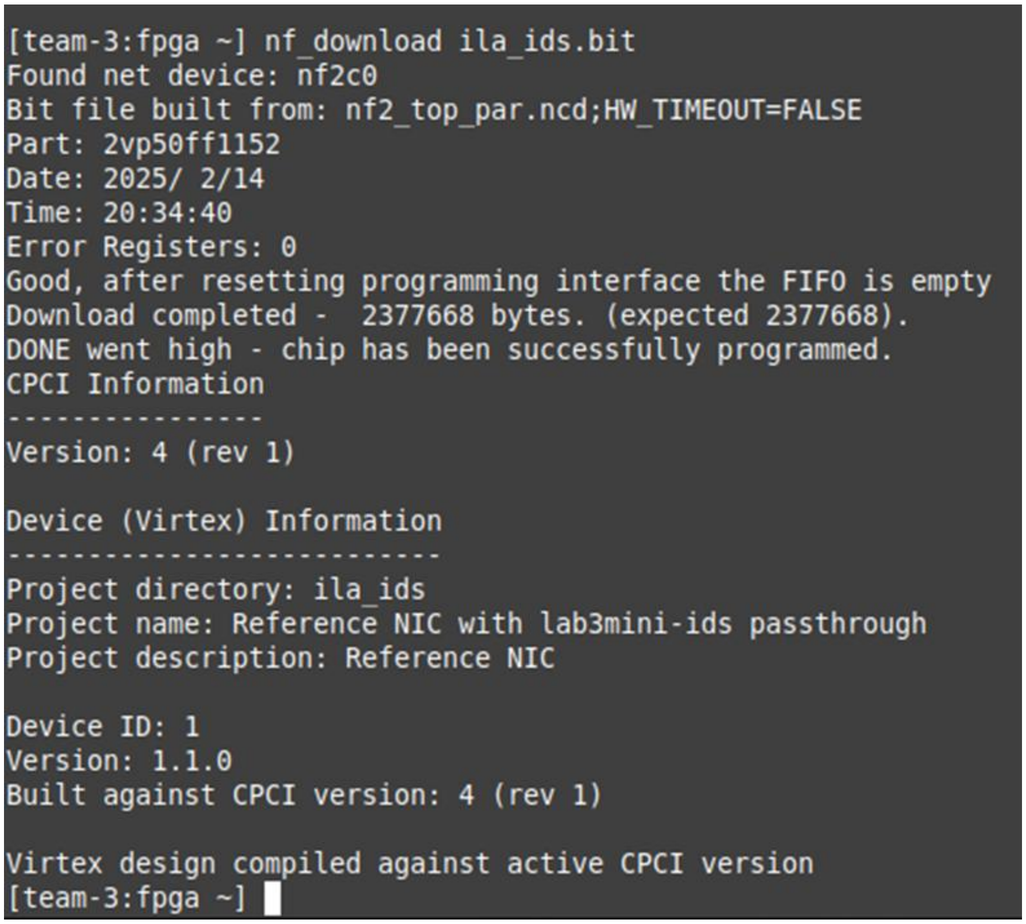
The final goal is to ensure that the mini-IDS functions correctly, detects predefined string patterns in network traffic, and performs efficiently without introducing excessive latency or packet drops.

2. Procedure

a. Passthrough Implementation

i. Description of adding passthrough.v to the NetFPGA reference NIC/Router

To ensure that our NetFPGA is set up correctly, a passthrough module (ids.v) was inserted into the NetFPGA reference NIC/router package. This module simply connects input to output using wire type variables, ensuring unmodified packet forwarding.



```
[team-3:fpga ~] nf_download ila_ids.bit
Found net device: nf2c0
Bit file built from: nf2_top_par.ncd;HW_TIMEOUT=FALSE
Part: 2vp50ff1152
Date: 2025/ 2/14
Time: 20:34:40
Error Registers: 0
Good, after resetting programming interface the FIFO is empty
Download completed -  2377668 bytes. (expected 2377668).
DONE went high - chip has been successfully programmed.
CPCI Information
----------------
Version: 4 (rev 1)

Device (Virtex) Information
---------------------------
Project directory: ila_ids
Project name: Reference NIC with lab3mini-ids passthrough
Project description: Reference NIC

Device ID: 1
Version: 1.1.0
Built against CPCI version: 4 (rev 1)

Virtex design compiled against active CPCI version
[team-3:fpga ~] █
```

Fig.1. Passthrough ids with ila plkaced to mintor output data of ids

ii. **Steps for compiling and generating the bitfile, then uploading the bitfile to the FPGA using *'scp'* and reconfiguring it**

1. Implemented passthrough ids.v, which directly connects inputs to outputs.
2. Compiled the design using the same method as we did in Lab#4 to generate the bitfile
3. Uploaded the bitfile to the NetFPGA using the Linux scp command
4. Reconfigured the NetFPGA with the generated bitfile and validated that it behaves the same as the reference NetFPGA bitfile

Expected behavior is that the passthrough module should forward all packets without modification. We verified by pinging between the nodes to confirm correct packet forwarding.

b. **Logic Analyzer Development**

i. **Implementation of a logic analyzer using block RAM**

A Logic Analyzer (ILA) was developed using block RAM (BRAM) to capture internal signals for debugging purposes. The ILA was controlled through a register interface, allowing it to record data and export it for analysis.

ii. **Description of register interface usage for recording internal signals**

We designed a register-mapped logic analyzer inside the FPGA. Then, modified the register interface to control the analyzer's behavior. Implemented a script to extract recording signals from the FPGA and send them to a server. Verified that internal signals were being captured correctly.

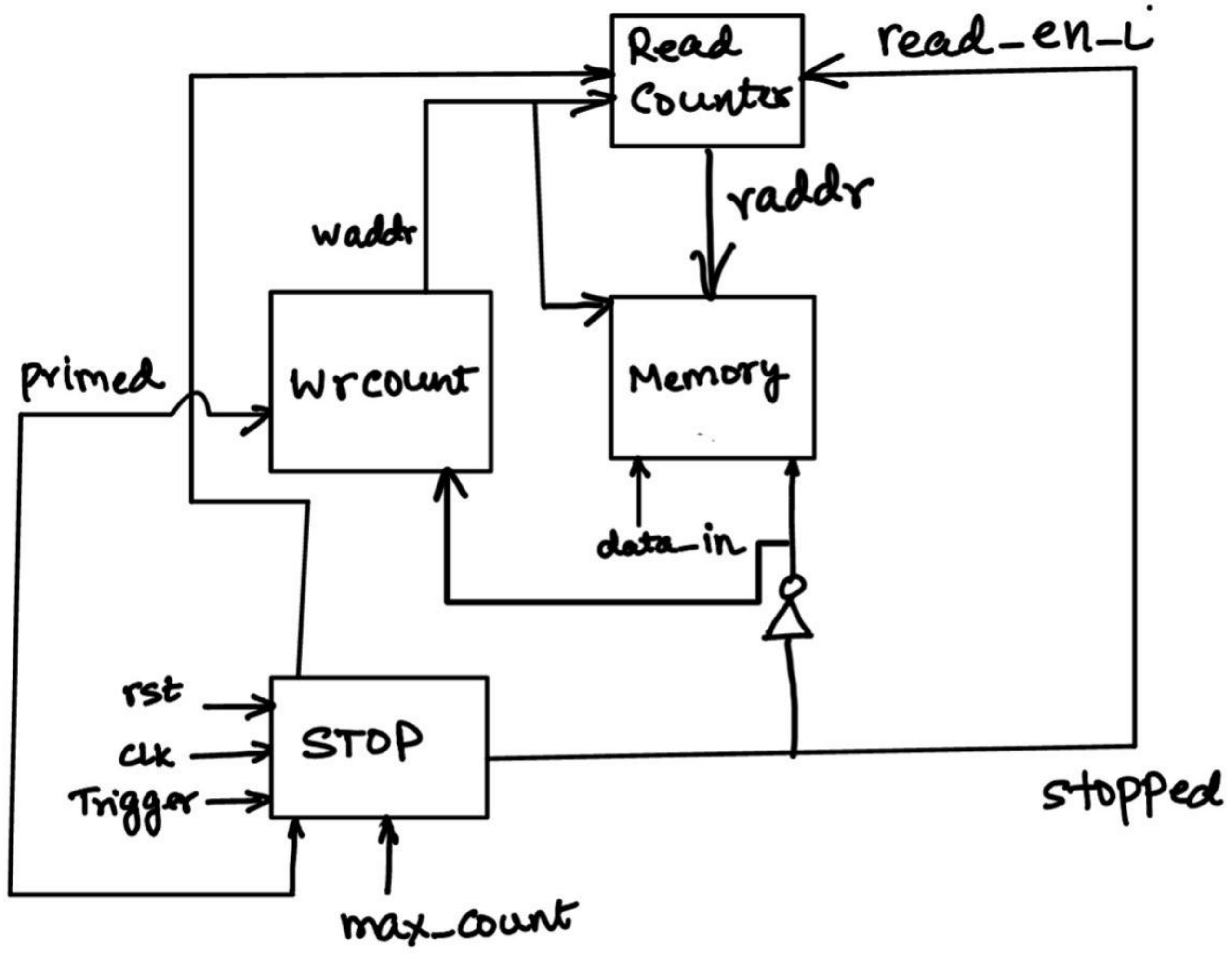


Fig. This is the Logic-Block diagram for custom ILA

The logic analyzer should record selected internal signals and allow data retrieval via the register interface.

c. **Incremental Design Testing**

   i. **Adding mini-IDS components step-by-step**

   To ensure reliability, mini-IDS components were added one at a time with separate compilation and testing at each stage.

   ii. **Loading and testing the bitfile using *'nf_download'***

   Steps:

   1. Integrated individual IDS components incrementally.
   2. Verified correctness at each step before moving forward.
   3. Used 'nf_download' to load the bitfile and ensured proper interface configuration.
   4. Conducted connectivity tests by ensuring that NetFPGA nodes could ping each other.

   Each newly added module should function correctly without affecting each other part of the system.

d. **Register-Based Testing**

   i. **Explanation of modifying *idsreg* Perl script**

   The perl script *idsreg* was used to interact with NetFPA registers. The script was modified to match the assigned register addresses and to facilitate testing of register-based data processing.

   Steps:

   1. Copied idsreg script to the FPGA node using scp.

   2. Analysed the script to understand how it interacts with registers.

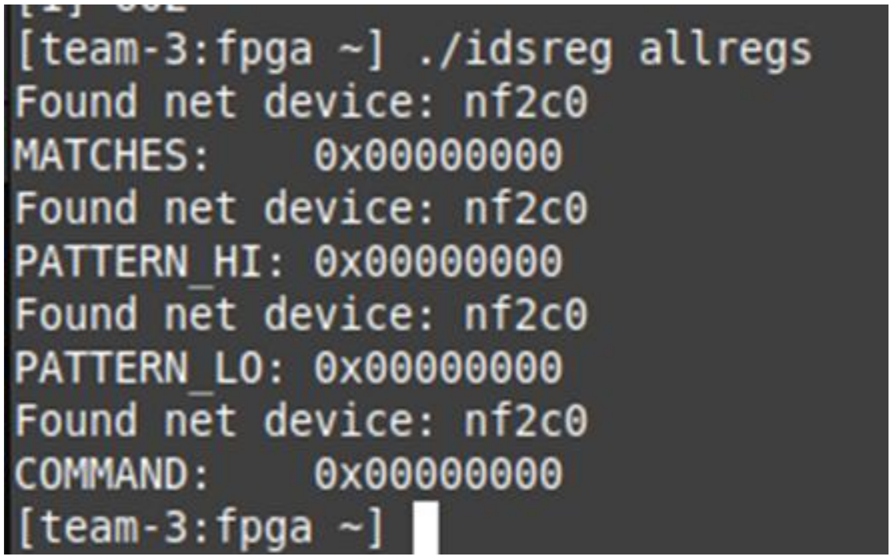
   3. Modified script:

   3.1. Use correct NetFPGA compiler-assigned register addresses.

   3.2. Support and writing custom register values.

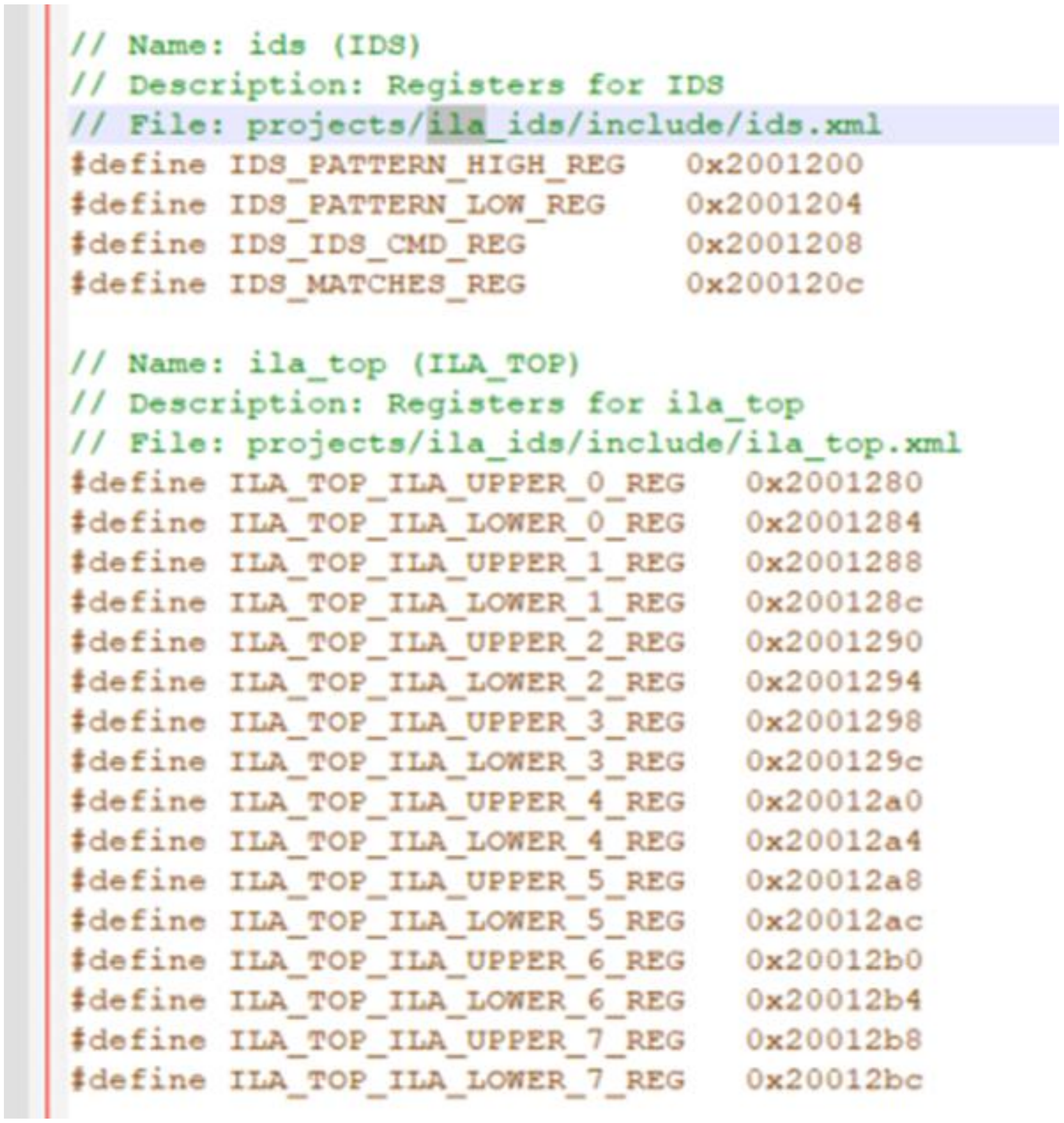
   3.3. Test data transformations within the hardware.

   Register reads/writes will work correctly allowing the FPGA to process data and return results.



```
[team-3:fpga ~] ./idsreg allregs
Found net device: nf2c0
MATCHES:    0x00000000
Found net device: nf2c0
PATTERN_HI: 0x00000000
Found net device: nf2c0
PATTERN_LO: 0x00000000
Found net device: nf2c0
COMMAND:    0x00000000
[team-3:fpga ~]
```

Fig.2. Successfully edited idsreg script to have the correct addresses of our ids registers. I.e. deadbeef is not shown anymore

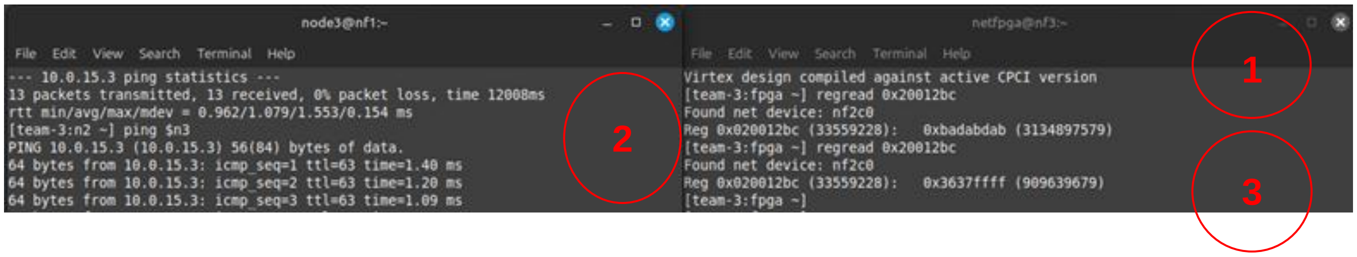


```
// Name: ids (IDS)
// Description: Registers for IDS
// File: projects/ila_ids/include/ids.xml
#define IDS_PATTERN_HIGH_REG   0x2001200
#define IDS_PATTERN_LOW_REG    0x2001204
#define IDS_IDS_CMD_REG        0x2001208
#define IDS_MATCHES_REG        0x200120c

// Name: ila_top (ILA_TOP)
// Description: Registers for ila_top
// File: projects/ila_ids/include/ila_top.xml
#define ILA_TOP_ILA_UPPER_0_REG   0x2001280
#define ILA_TOP_ILA_LOWER_0_REG   0x2001284
#define ILA_TOP_ILA_UPPER_1_REG   0x2001288
#define ILA_TOP_ILA_LOWER_1_REG   0x200128c
#define ILA_TOP_ILA_UPPER_2_REG   0x2001290
#define ILA_TOP_ILA_LOWER_2_REG   0x2001294
#define ILA_TOP_ILA_UPPER_3_REG   0x2001298
#define ILA_TOP_ILA_LOWER_3_REG   0x200129c
#define ILA_TOP_ILA_UPPER_4_REG   0x20012a0
#define ILA_TOP_ILA_LOWER_4_REG   0x20012a4
#define ILA_TOP_ILA_UPPER_5_REG   0x20012a8
#define ILA_TOP_ILA_LOWER_5_REG   0x20012ac
#define ILA_TOP_ILA_UPPER_6_REG   0x20012b0
#define ILA_TOP_ILA_LOWER_6_REG   0x20012b4
#define ILA_TOP_ILA_UPPER_7_REG   0x20012b8
#define ILA_TOP_ILA_LOWER_7_REG   0x20012bc
```

Fig.3. Successfully placed ila registerse into the memory map

a. We did regread 0x20012bc which is the last ila register and read 0xbadabdab which is our initialization value.
b. Then I started ping $n3 on n2
c. After pinging, we did regread 0x20012bc again, and this time. We got a new value 0x3637ffff indicating that our ila is updating properly and we are able to read from it.

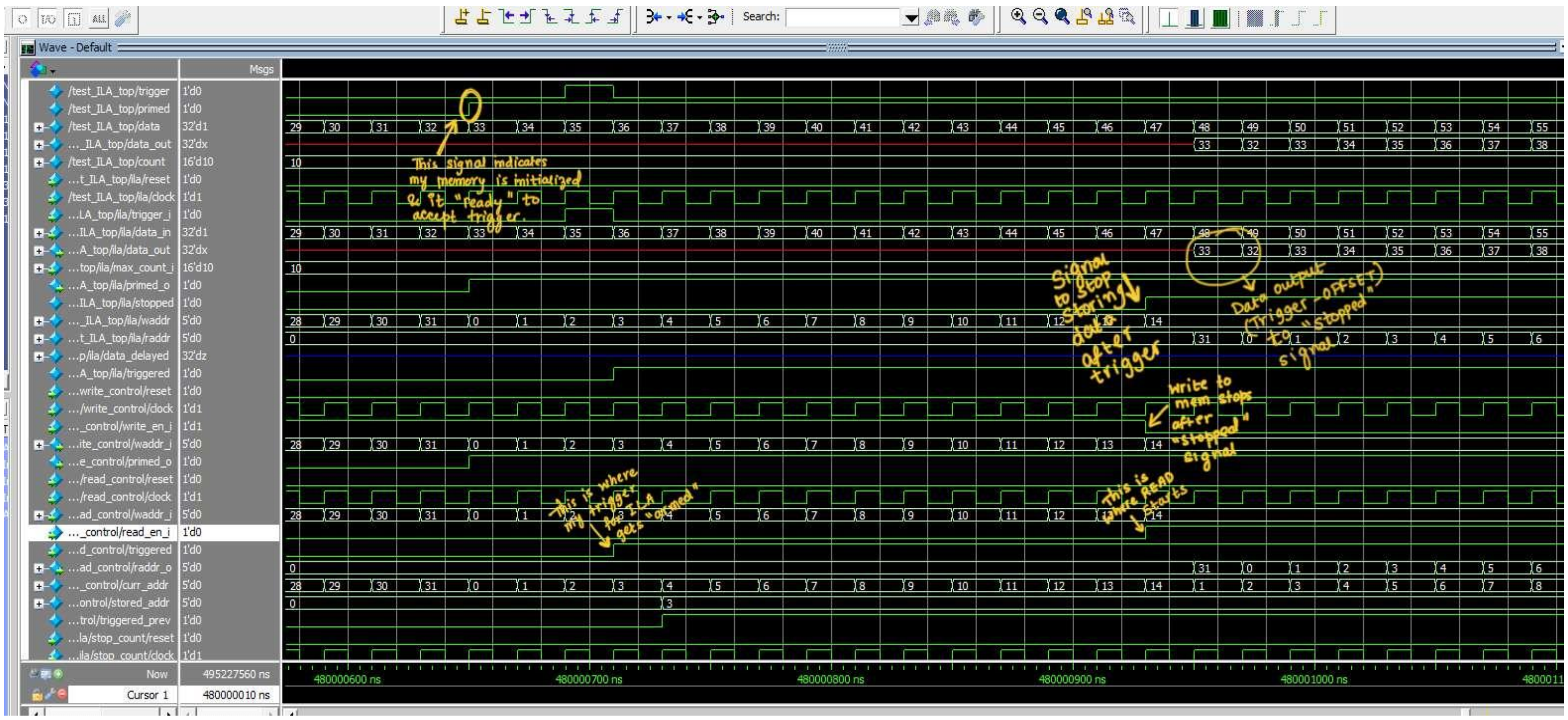


```
node3@nf1:~
File  Edit  View  Search  Terminal  Help
--- 10.0.15.3 ping statistics ---
13 packets transmitted, 13 received, 0% packet loss, time 12008ms
rtt min/avg/max/mdev = 0.962/1.079/1.553/0.154 ms
[team-3:n2 ~] ping $n3
PING 10.0.15.3 (10.0.15.3) 56(84) bytes of data.
64 bytes from 10.0.15.3: icmp_seq=1 ttl=63 time=1.40 ms
64 bytes from 10.0.15.3: icmp_seq=2 ttl=63 time=1.20 ms
64 bytes from 10.0.15.3: icmp_seq=3 ttl=63 time=1.09 ms
```

```
netfpga@nf3:~
File  Edit  View  Search  Terminal  Help
Virtex design compiled against active CPCI version
[team-3:fpga ~] regread 0x20012bc
Found net device: nf2c0
Reg 0x020012bc (33559228):   0xbadabdab (3134897579)
[team-3:fpga ~] regread 0x20012bc
Found net device: nf2c0
Reg 0x020012bc (33559228):   0x3637ffff (909639679)
[team-3:fpga ~]
```

e. **Traffic Analysis with iperf**
   i. **Running iperf servers/clients on each node**
   ii. **Sending:**
      1. **"Good" packets (without the specific string)**
      2. **"Bad" packets (with the designated string)**
   iii. **Observing behavior in TCP and UDP modes**
   iv. **Using tcpdump to verify packet blocking**

f. **Final Testing**
   i. **Running idsreg matches to verify the count of dropped packets**



```
Wave - Default                      Msgs
/test_ILA_top/trigger               1'd0
/test_ILA_top/primed                1'd0
/test_ILA_top/data                  32'd1
..._ILA_top/data_out                32'dx
/test_ILA_top/count                 16'd10
...t_ILA_top/ila/reset              1'd0
/test_ILA_top/ila/clock             1'd1
...LA_top/ila/trigger_i             1'd0
...ILA_top/ila/data_in              32'd1
...A_top/ila/data_out               32'dx
...top/ila/max_count_i              16'd10
...A_top/ila/primed_o               1'd0
...ILA_top/ila/stopped              1'd0
..._ILA_top/ila/waddr               5'd0
...t_ILA_top/ila/raddr              5'd0
...p/ila/data_delayed               32'dz
...A_top/ila/triggered              1'd0
...write_control/reset              1'd0
.../write_control/clock             1'd1
..._control/write_en_i              1'd1
...ite_control/waddr_i              5'd0
...e_control/primed_o               1'd0
.../read_control/reset              1'd0
.../read_control/clock              1'd1
...ad_control/waddr_i               5'd0
..._control/read_en_i               1'd0
...d_control/triggered              1'd0
...ad_control/raddr_o               5'd0
..._control/curr_addr               5'd0
...ontrol/stored_addr               5'd0
...trol/triggered_prev              1'd0
...la/stop_count/reset              1'd0
...ila/stop_count/clock             1'd1
Now                                 495227560 ns
Cursor 1                            480000010 ns
```

Search:
This signal indicates my memory is initialized & it "ready" to accept trigger.
Signal to stop storing data after trigger
Data output (Trigger - OFFSET) to "stopped" signal
write to mem stops after "stopped" signal
This is where my trigger for ILA gets "armed"
This is where READ starts
480000600 ns
480000700 ns
480000800 ns
480000900 ns
480001000 ns

Please note the above-mentioned waveform that has been uploaded:
The *primed* signal indicated that our memory is initialized and it is ready to accept the trigger signal.

The *data out* signal is DATA-OUTPUT (Trigger -Offset) to ‘stopped’ signal

The write address (waddr) is used to stop storing data after the trigger

The write enable (wr_en_i) writes to memory stops after ‘stopped’ signal

The read enable (rd_en_j) reads where READ starts

The trigger signal (triggered) is where our trigger for ILA gets armed.

3. **Answering Report Questions**
   a. **Identifying a Bug at High-Speed Traffic**
      i. **Explanation of any discovered issues when the mini-IDS operates near 1 Gbps**
         At high-speed network traffic (~1Gbps) a performance issues:
         1. Bug: The mini-IDS drops legitimate packets or introduces delays
      ii. **Possible causes (e.g. buffer overflow, processing delays)**
         a. FIFO/Buffer Overflow: Limited memory can’t handle high-speed traffic.
         b. Pattern Matching delay: The IDS takes long time to analyze packets
         c. Synchronization issues: Packet order may be incorrect under heavy loads.

         Solutions:
         Increase buffer size to handle traffic bursts.
         Optimize pattern matching logic for high-speed performance.
         Parallelize packet processing to improve throughput.
   b. **How the mini-IDS works**
      i. **Summary of the mini-IDS operation and how it interacts with other NetFPA modules**
         Mini-IDS is a hardware-based intrusion detection system implemented on NetFPGA.
         Functionality:
         Receives network packets via NetFPGA.
         Extracts payloads and checks for a predefined pattern.
         Drops malicious packets if they contain the pattern.
         Forwards legitimate traffic to the destination.
      ii. **Explanation for a technical audience (NetFPGA developers)**
         user_data_path.v → Manages network packet routing.
         ids.v → Implements pattern matching logic.

Register Interface → Allows software-based IDS configuration.

c. **Pattern Matching Algorithm**

   i. **Explanation of the system detects patterns in network traffic**
      Reads packet payload data.
      Extracts a substring from the payload.
      Compares it against the predefined pattern (e.g., "ABCDEFG").
      If a match is found, the packet is dropped.
      Else, it is forwarded normally

   ii. **Algorithm efficiency and performance considerations**
      Use hardware-based comparators to speed up matching.
      Implement parallel string comparison to process packets efficiently