Ethernet ARP DPI Project

Author

Artem Voropaev
Email: voropaev.art@gmail.com
Created: 2025-11-06

Functional Description

This project implements an **ARP (Address Resolution Protocol) responder** in SystemVerilog RTL that interfaces with a real Linux network stack using SystemVerilog DPI-C (Direct Programming Interface for C). The design can respond to real ARP requests from the network through a TAP (network tap) interface.

Key Features:

- **ARP Parser**: Parses incoming ARP request packets from Ethernet MAC, validates frame structure and checks if the request is for the configured IP address
- **ARP Sender**: Generates and transmits ARP reply packets in response to valid requests
- **Clock Domain Crossing**: Dual-clock FIFO for safe transfer of ARP packets between RX and TX clock domains
- **TAP Interface Integration**: Connects RTL simulation to Linux TAP interface for real network packet exchange
- **DPI-C Bridge**: Provides seamless communication between SystemVerilog testbench and C code for packet handling

How It Works:

1. A TAP interface (`tap0`) is created on the Linux host with IP address 192.168.43.1/24
2. The C application (`test_bfm`) reads packets from the TAP interface and forwards them to the RTL via DPI-C
3. The RTL (`arp_top`) parses incoming ARP requests, validates them, and generates ARP replies
4. ARP replies are sent back through the DPI-C interface to the TAP interface
5. Real network tools like `arping` can send ARP requests that are processed by the simulated hardware
6. Traffic can be monitored using Wireshark on the TAP interface

Default Configuration:

- DUT MAC Address: `00:11:22:33:44:55`
- DUT IP Address: `192.168.1.1` (0xC0A80101)
- TAP Interface: `tap0` at `192.168.43.1/24`

to change the configuration, edit the `arp_tb.sv` file:

```
        my_mac  = 48'h001122334455;
        my_ipv4 = 32'hC0A80101; // 192.168.1.1
```

File Structure

```
eth_dpi/
├── README.md                          # This file
├── run.sh                             # Main simulation runner script
│
├── scripts/
│   └── create_tap_iface.sh            # Script to create TAP network interface
│
├── src/
│   ├── rtl/                           # RTL source files
│   │   ├── arp_pkg.sv                 # ARP protocol package with data structures
│   │   ├── arp_parser.sv              # Parses incoming ARP request packets
│   │   ├── arp_sender.sv              # Generates and sends ARP reply packets
│   │   ├── arp_top.sv                 # Top module connecting parser and sender
│   │   ├── dc_fifo_wrapper.sv         # Dual-clock FIFO for CDC
│   │   └── altera_mf.v                # Altera megafunction library (FIFO
implementation)
│   │
│   └── tb/                            # Testbench files
│       ├── arp_tb.sv                  # Top-level testbench with DPI-C integration
│       ├── start_sim.tcl              # QuestaSim simulation script
│       ├── wave.do                    # Waveform configuration
│       ├── work/                      # QuestaSim work library (generated)
│       └── alt_verilog_libs/          # Compiled Altera libraries (generated)
│
└── sw/                                # Software/C code for DPI-C
    ├── test_bfm.c                     # Main BFM: TAP interface handler
    ├── dpi_tasks.h                    # DPI-C task declarations
    ├── Makefile                       # Builds shared library for DPI
    └── test_bfm.so                    # Compiled shared library (generated)
```

Key Files Description:

RTL Files:

- `arp_pkg.sv`: Defines ARP frame structure (`ether_arp_frame_t`) and validation functions
- `arp_parser.sv`: State machine that receives bytes from MAC, assembles ARP frame, validates it
- `arp_sender.sv`: State machine that generates ARP reply and serializes it byte-by-byte to MAC
- `arp_top.sv`: Integrates parser, sender, and CDC FIFO with separate RX/TX clock domains
- `dc_fifo_wrapper.sv`: Clock domain crossing FIFO wrapper for Altera megafunctions

Testbench Files:

- `arp_tb.sv`: SystemVerilog testbench with mailbox-based packet queues and DPI-C exports/imports
- `start_sim.tcl`: Compiles RTL and testbench, starts simulation with DPI-C library

Software Files:

- `test_bfm.c`: Creates TAP interface, reads/writes packets, interfaces with RTL via DPI-C tasks

- `dpi_tasks.h`: Header with DPI-C function declarations for SystemVerilog-C interface

---

Tools Used

1. **Ubuntu 18.04**
2. **QuestaSim/ModelSim v2021.2**
3. **gcc 7.5.0** (C/C++ Compiler)
    - Compiles C code into shared library (`.so`) for DPI-C interface
4. **GNU Make 4.1**
    - Builds the DPI-C shared library from C sources
5. **Linux Kernel TUN/TAP Driver**
    - Creates virtual network interface for packet injection/capture
    - Requires `/dev/net/tun` device
6. **arping** (from `iputils` package)
    - Command-line tool to send ARP requests
    - Used to test the ARP responder: `arping -I tap0 192.168.1.1`
7. **Wireshark 2.6.10**
    - Network protocol analyzer for capturing and analyzing packets

---

How to Run Simulation

Prerequisites

Install Required Packages:

- Install arping

```
sudo apt-get install arping
```

- Install Wireshark

```
sudo add-apt-repository ppa:wireshark-dev/stable
sudo apt-get update
sudo apt-get install wireshark
```

Update QuestaSim Paths:

Edit `run.sh` to set your QuestaSim installation path:

```
export MSIM_HOME=/home/vae/Questa_sim/questasim
export LM_LICENSE_FILE=/home/vae/Questa_sim/license.dat
```

Create Linux TAP Interface:

Important: TAP interface creation requires root privileges.

```
# Create TAP interface with IP 192.168.43.1/24
sudo ./scripts/create_tap_iface.sh
```

This script will:

- Create a TAP interface named `tap0`
- Assign IP address `192.168.43.1/24` to it
- Configure reverse path filtering
- Bring the interface up

Verify the interface:

```
ifconfig tap0
```

You should see:

```
tap0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.43.1  netmask 255.255.255.0  broadcast 192.168.43.255
        ether e6:07:6f:82:09:1b  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Run QuestaSim Simulation:

Important: run the simulation with sudo privileges.

Run the simulation in **console mode** (no GUI):

```
sudo ./run.sh
```

Or run with **GUI** for waveform viewing:

```
sudo ./run.sh -gui
```

What happens during simulation:

1. The `Makefile` compiles `test_bfm.c` into `test_bfm.so` shared library
2. QuestaSim compiles RTL and testbench files
3. Simulation starts and the C application opens the `tap0` interface
4. The testbench enters an infinite loop, waiting for packets from the TAP interface
5. You should see: `HOST: ifr_name=tap0, ready`

Note: The simulation runs indefinitely (does not auto-terminate). You'll need to stop it manually or it will run until packets are exchanged.

Send ARP Requests with arping

In a separate terminal, while the simulation is running, send ARP requests:

```
# Send ARP request to the DUT's IP address (192.168.1.1) via tap0
sudo arping -I tap0 192.168.1.1
```

Expected Output:

```
sudo arping -I tap0 192.168.1.1
ARPING 192.168.1.1
42 bytes from 00:11:22:33:44:55 (192.168.1.1): index=0 time=7.158 msec
42 bytes from 00:11:22:33:44:55 (192.168.1.1): index=1 time=5.262 msec
42 bytes from 00:11:22:33:44:55 (192.168.1.1): index=2 time=10.202 msec
42 bytes from 00:11:22:33:44:55 (192.168.1.1): index=3 time=4.356 msec

...
```

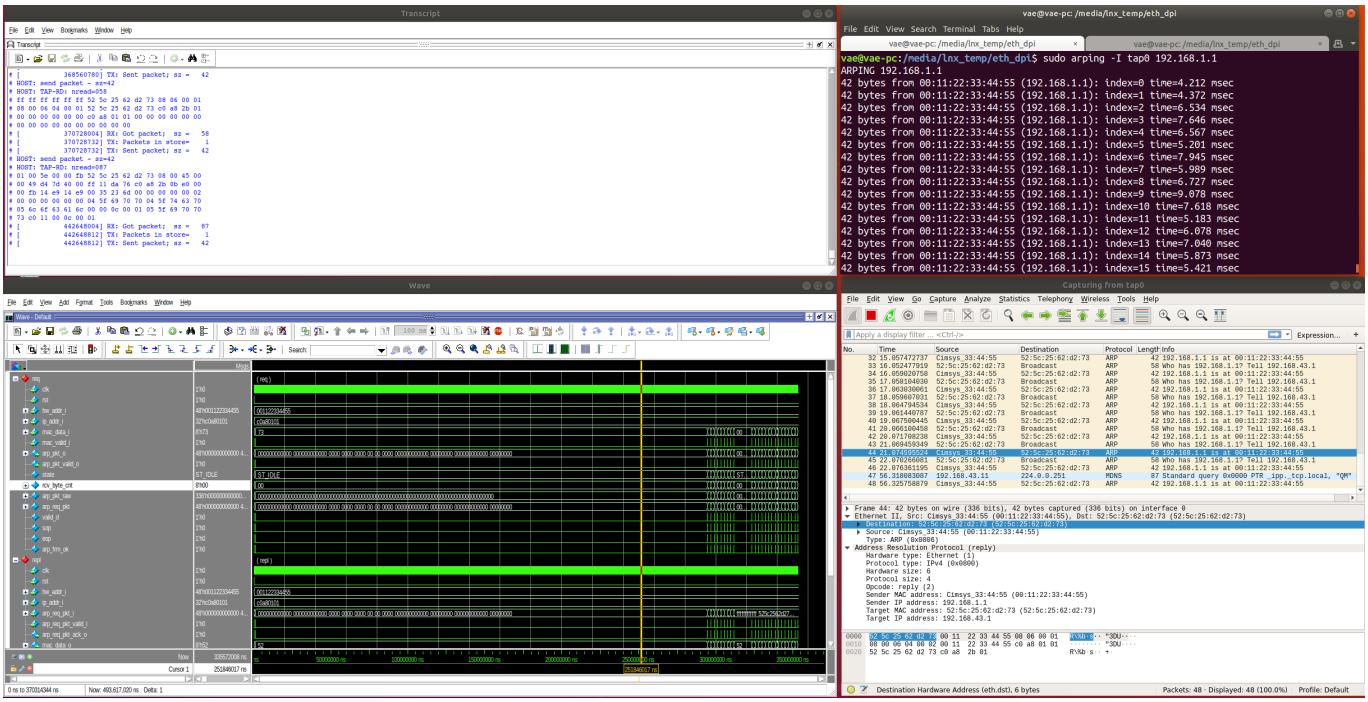
In the QuestaSim console, you should see:

```
# HOST: TAP-RD: nread=058
# ff ff ff ff ff ff e6 07 6f 82 09 1b 08 06 00 01
# 08 00 06 04 00 01 e6 07 6f 82 09 1b c0 a8 2b 01
# 00 00 00 00 00 00 c0 a8 01 01 00 00 00 00 00 00
# 00 00 00 00 00 00 00 00 00 00
# [            25224004] RX: Got packet;  sz =   58
# [            25224780] TX: Packets in store=    1
# [            25224780] TX: Sent packet; sz =   42
```

Step 4: Monitor Traffic with Wireshark

In another terminal, launch Wireshark to observe packet exchange:

Sim results:



Simulation Workflow Diagram

```
flowchart TD
    A["arping Tool
(Send ARP Request)"] --> B["Linux TAP Interface (tap0)
IP: 192.168.43.1/24"]
    B --> C["test_bfm.c (C)
- Read from TAP
- Write to TAP"]
    C -->|DPI-C calls| D["arp_tb.sv (SystemVerilog TB)
- Mailbox queues (to_hdl_mbx/from_hdl_mbx)
- Clock generation (CLK_RX, CLK_TX)
- DPI-C export/import tasks"]
    D --> E["arp_top (DUT)"]
    E --> F["arp_parser
(RX clock domain)"]
    F --> G["DC FIFO
(Clock crossing)"]
    G --> H["arp_sender
(TX clock domain)"]
    H --> I["ARP Reply Generated"]
    I -->|Packets flow back via DPI-C| C
    C --> J["Wireshark
(Monitor ARP Request/Reply)"]
    B --> J
```