TCP Offloading Engine

Clementine Barbet (cb3022)
Christine Chen (cpc2143)
Qi Li (ql2163)

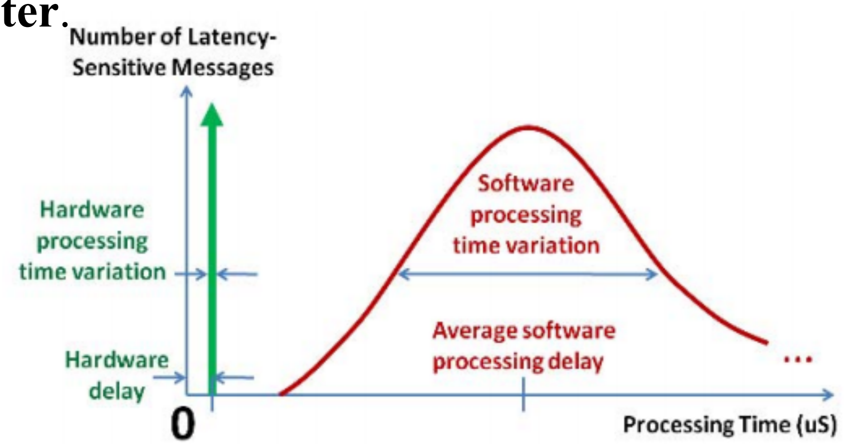
Project Goals

- Software
  - Understand how the TCP/IP protocol enables reliable communications
  - Implement a TCP stack that bypasses the Linux Kernel
  - Verify implementation through comparison with Golden Model

- Hardware
  - Implementation of a TOE IP software Core using System Verilog
  - Implementation of a qsys design that allows streaming packets in/out of the TOE
  - Verification using simulation with ModelSim and JTAG.

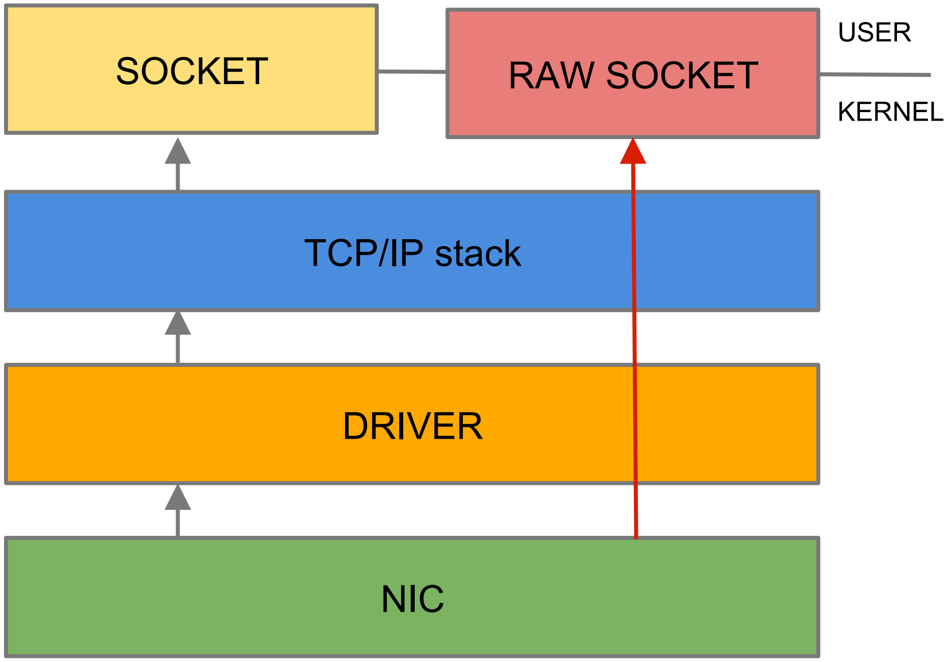
Motivations

Offloading the handling of TCP to an FPGA IP core allows to minimise **processing latency** and **jitter**.


Number of Latency-Sensitive Messages
Hardware processing time variation
Hardware delay
0
Software processing time variation
Average software processing delay
...
Processing Time (uS)

Potential Applications : HFT, data-center servers, …

Software Implementation (1) : Bypassing the Linux Kernel using Raw Sockets



```
sd = socket (PF_PACKET,
 SOCK_RAW, htons(ETH_P_ALL))
```

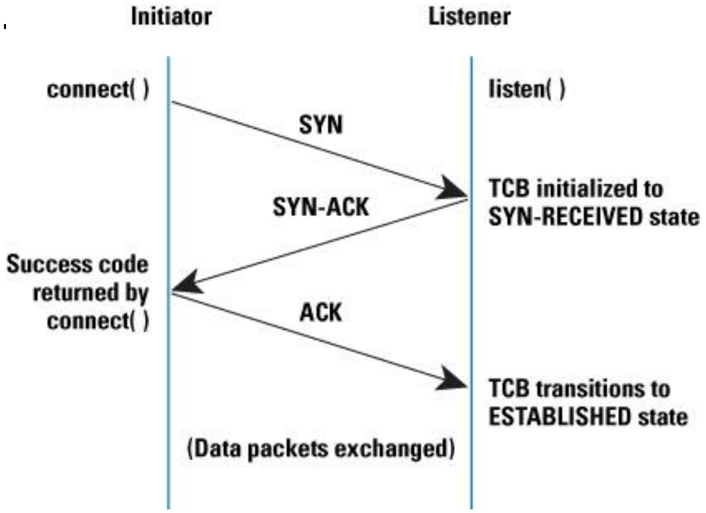
Issue encountered with linux kernel sending spontaneous RST :

```
sudo iptables -A OUTPUT -o wlan0
 -p tcp --dport 52000 --tcp-
 flags RST RST -j DROP
```

Software Implementation (2) : TCP stack

Features :

- Initiate connections by performing 3-handshake (SYN, SYN-ACK, ACK)
- Can handle several connections at once
- Performs reliable data transfer with appropriate ACK sending
- Close connections by performing 3-handshake protocol (FIN, FIN-ACK, ACK)
- User can switch from hardware implementation to software implementation while using same function calls.


Initiator
Listener
connect( )
listen( )
SYN
TCB initialized to SYN-RECEIVED state
SYN-ACK
Success code returned by connect( )
ACK
TCB transitions to ESTABLISHED state
(Data packets exchanged)

Not implemented TCP features :

- Window-size advertising
- Management of retransmissions

Software Implementation (3) : Implementation

- Structure that holds the connection data :

```
struct tcp_ctrl{
                int sd;
                char *interface, *target, *src_ip, *dst_ip;
                uint8_t *src_mac, *dst_mac, *ether_frame;
                int *ip_flags, *tcp_flags;
                struct sockaddr_ll device;
                int seq, rcv_ack;
                uint16_t sport, dport;
                uint8_t *sdbuffer;
                struct tcphdr *tcphdr;
                struct ip *iphdr;
                int mtu;
                state_t state;};
```

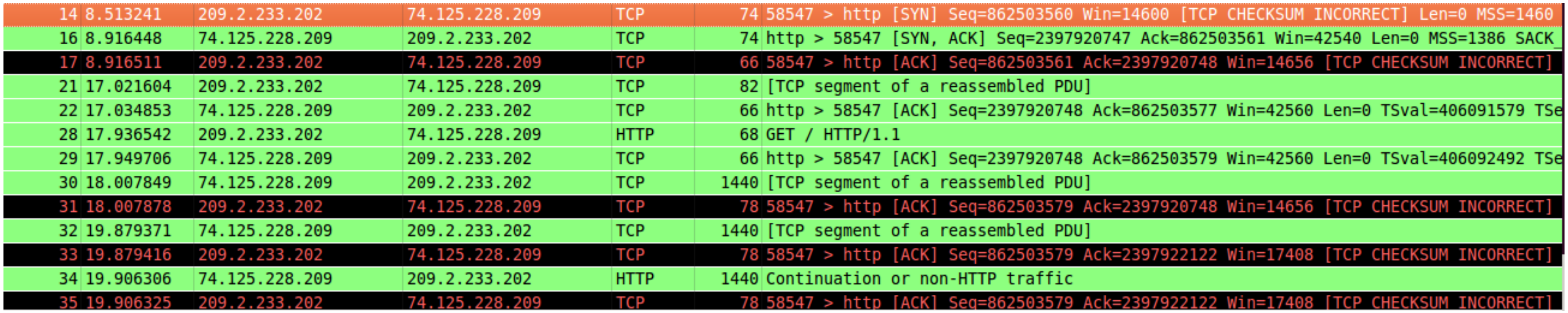
- Function to chose the software API :

```
int tcp_set_rawsck(void);
```

- Software API :

```
struct tcp_ctrl *(*tcp_new)(void);
int (*tcp_bind)(struct tcp_ctrl*, char*, uint16_t, char*);
int (*tcp_connect)(struct tcp_ctrl *, char *);
struct tcp_ctrl *(*tcp_listen)(struct tcp_ctrl *);
int (*tcp_close)(struct tcp_ctrl *);
```

Software Implementation (4) : Golden Model



```
14 8.513241  209.2.233.202  74.125.228.209 TCP    74 58547 > http [SYN] Seq=862503560 Win=14600 [TCP CHECKSUM INCORRECT] Len=0 MSS=1460
16 8.916448  74.125.228.209 209.2.233.202  TCP    74 http > 58547 [SYN, ACK] Seq=2397920747 Ack=862503561 Win=42540 Len=0 MSS=1386 SACK_
17 8.916511  209.2.233.202  74.125.228.209 TCP    66 58547 > http [ACK] Seq=862503561 Ack=2397920748 Win=14656 [TCP CHECKSUM INCORRECT]
21 17.021604 209.2.233.202  74.125.228.209 TCP    82 [TCP segment of a reassembled PDU]
22 17.034853 74.125.228.209 209.2.233.202  TCP    66 http > 58547 [ACK] Seq=2397920748 Ack=862503577 Win=42560 Len=0 TSval=406091579 TSe
28 17.936542 209.2.233.202  74.125.228.209 HTTP   68 GET / HTTP/1.1
29 17.949706 74.125.228.209 209.2.233.202  TCP    66 http > 58547 [ACK] Seq=2397920748 Ack=862503579 Win=42560 Len=0 TSval=406092492 TSe
30 18.007849 74.125.228.209 209.2.233.202  TCP  1440 [TCP segment of a reassembled PDU]
31 18.007878 209.2.233.202  74.125.228.209 TCP    78 58547 > http [ACK] Seq=862503579 Ack=2397920748 Win=14656 [TCP CHECKSUM INCORRECT]
32 19.879371 74.125.228.209 209.2.233.202  TCP  1440 [TCP segment of a reassembled PDU]
33 19.879416 209.2.233.202  74.125.228.209 TCP    78 58547 > http [ACK] Seq=862503579 Ack=2397922122 Win=17408 [TCP CHECKSUM INCORRECT]
34 19.906306 74.125.228.209 209.2.233.202  HTTP 1440 Continuation or non-HTTP traffic
35 19.906325 209.2.233.202  74.125.228.209 TCP    78 58547 > http [ACK] Seq=862503579 Ack=2397922122 Win=17408 [TCP CHECKSUM INCORRECT]
```

- Extensively used Wireshark to track packets sent.

- Golden Reference generated by sending an http request to Google using Telnet

- Comparison to Golden reference has been done manually.

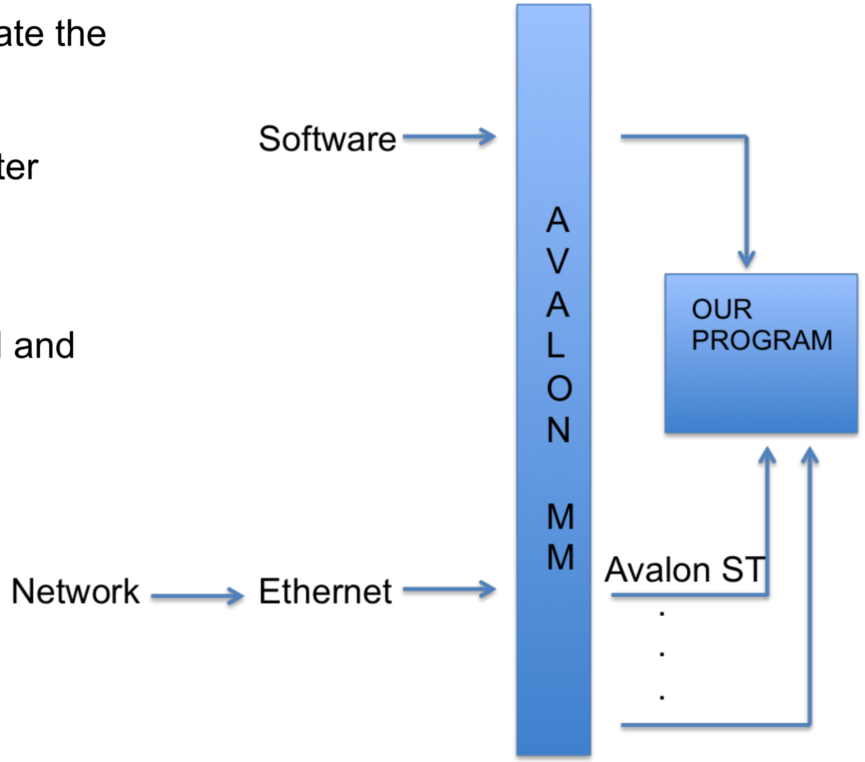
Qsys

We use Qsys to generate the interconnect.

-processor: 32-bit master

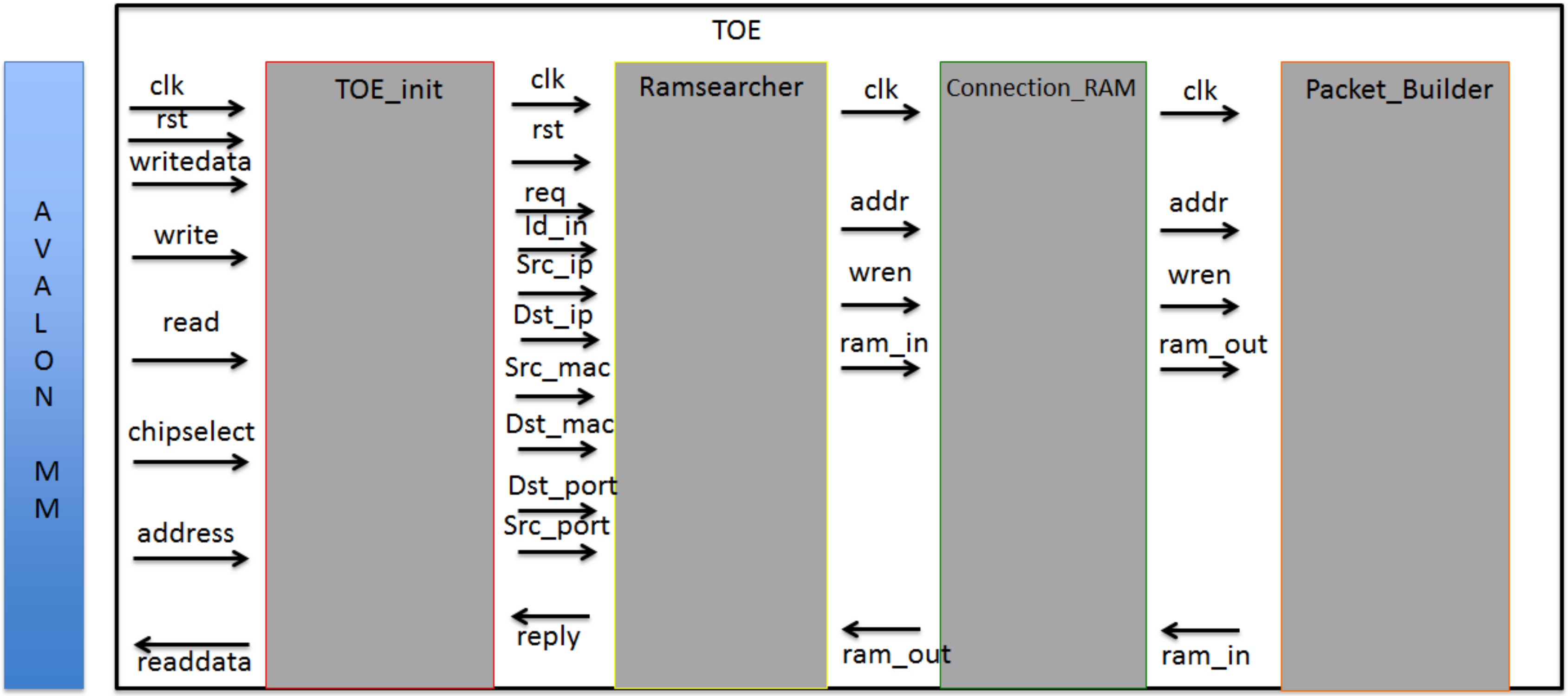
-slaves: 8-bit

-Have both Avalon MM and Avalon ST signals



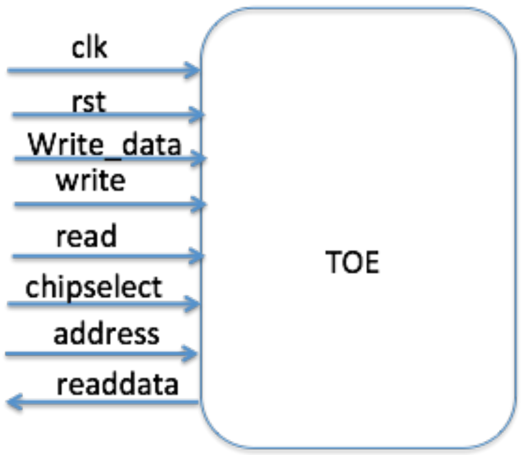
Hardware Module Interfaces

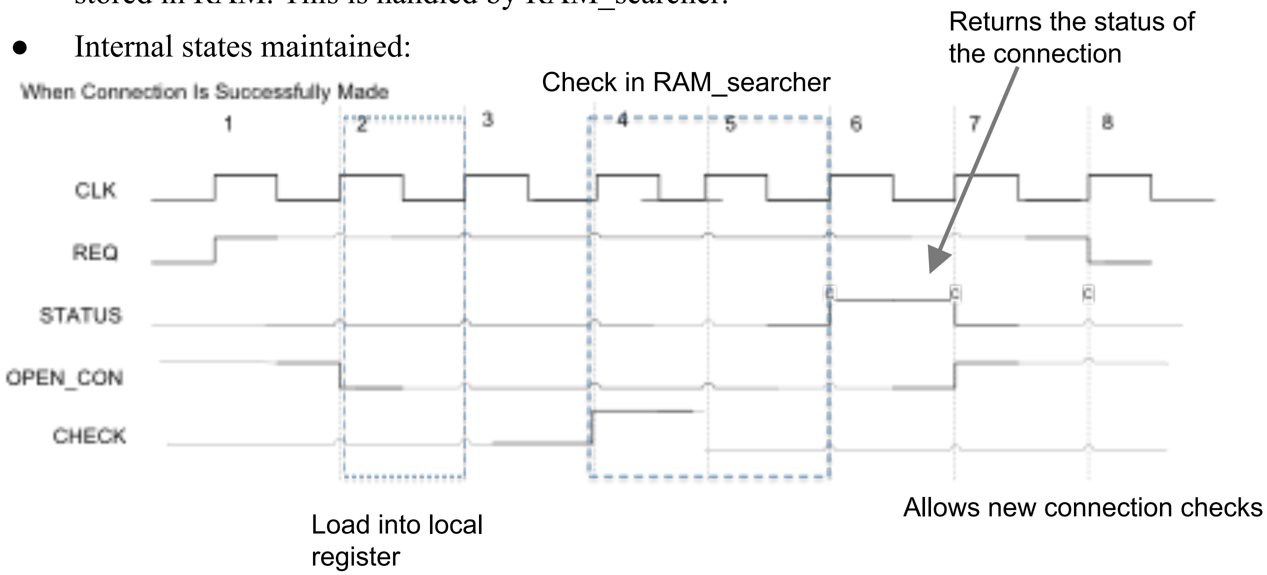
- Maintaining state of connection in TOE
- RAM_searcher searches/inserts/deletes a connection
- Packet builder goes through connection_RAM, generating packets when connection is set



TOE Connection

- Takes value of bits from Avalon MM. It interface with Avalon MM slaves.
- If a new request comes in, and the current connection is open, it loads the data into the local register, and compares this connection data with the previous existing ones stored in RAM. This is handled by RAM_searcher.
- Internal states maintained:





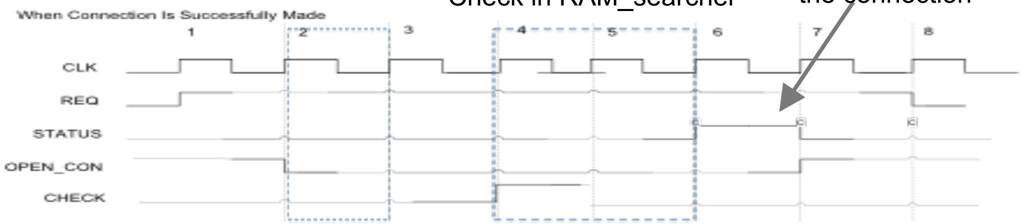
Check in RAM_searcher

Returns the status of the connection

Load into local register

Allows new connection checks

TOE Connection

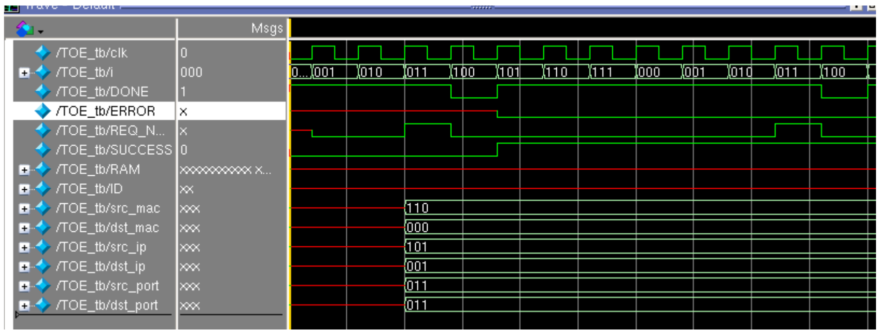


Check in RAM_searcher

Returns the status of the connection

Load into local register

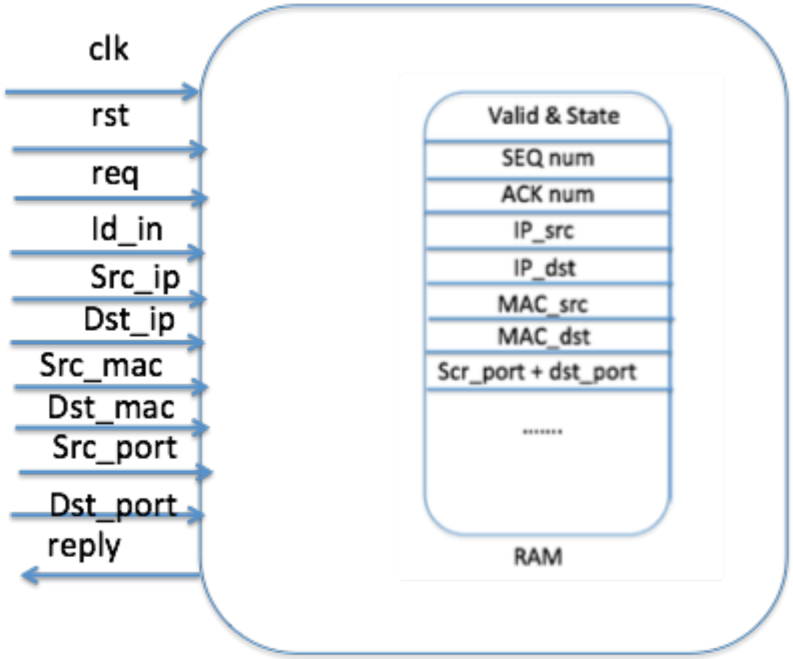
Allows new connection checks



	Msgs
/TOE_tb/clk	0
/TOE_tb/i	000
/TOE_tb/DONE	1
/TOE_tb/ERROR	x
/TOE_tb/REQ_N...	x
/TOE_tb/SUCCESS	0
/TOE_tb/RAM	xxxxxxxxxxx x...
/TOE_tb/ID	xx
/TOE_tb/src_mac	xxx
/TOE_tb/dst_mac	xxx
/TOE_tb/src_ip	xxx
/TOE_tb/dst_ip	xxx
/TOE_tb/src_port	xxx
/TOE_tb/dst_port	xxx

intermediate module testing with Modelsim

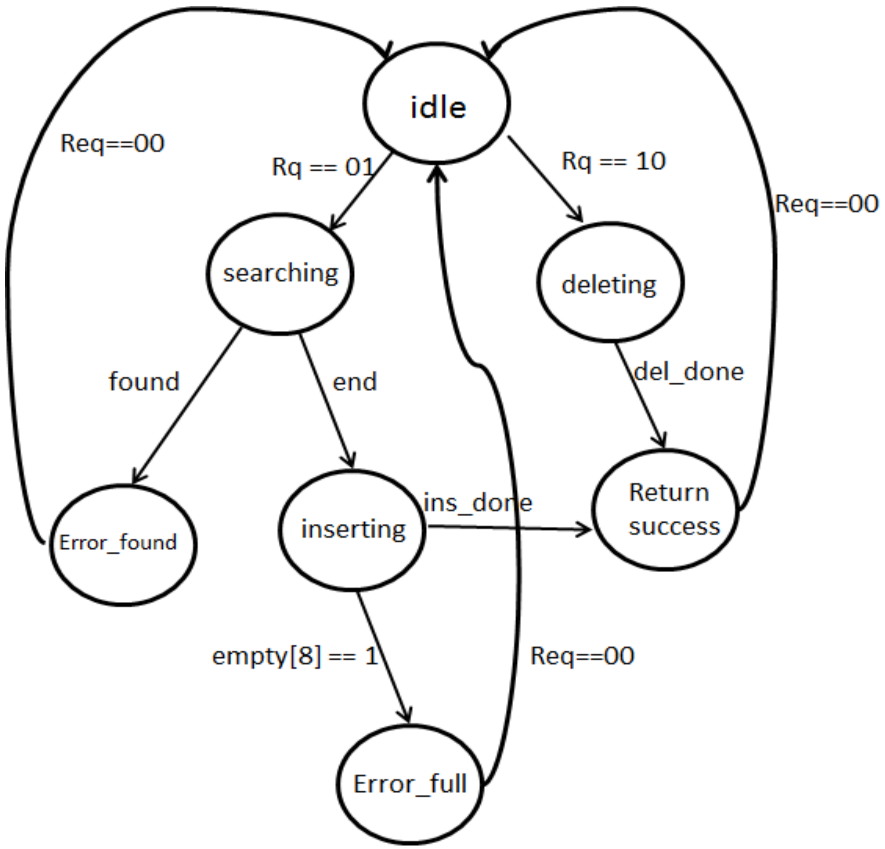
RAM_searcher Structure



RAM_searcher

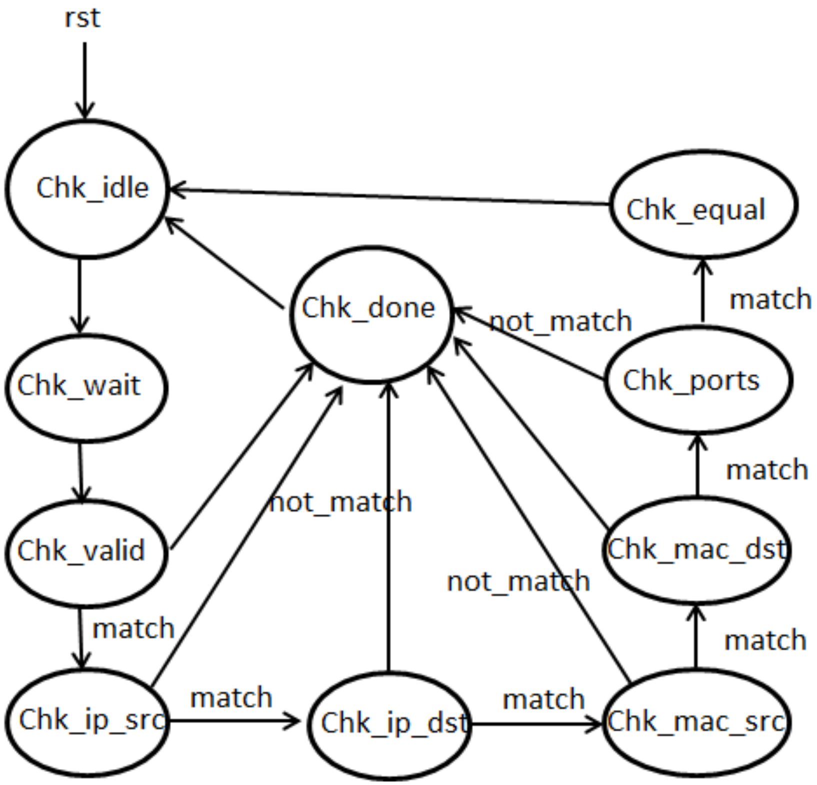
- RAM_searcher establish/delete a TCP connection depending on the request
- RAM maintains a list of existing TCP connection
- Layout of data stored in the RAM
  - First slot is valid and TCP states
  - seq number (32 bits)
  - ack number (32 bits)
  - ip_src (32 bits)
  - ip_dst (32 bits)
  - mac_src (48 bits)
  - mac_dst (48 bits)
  - src_port (16 bits) + dst_port (16 bits)

Ram_searcher State Diagram



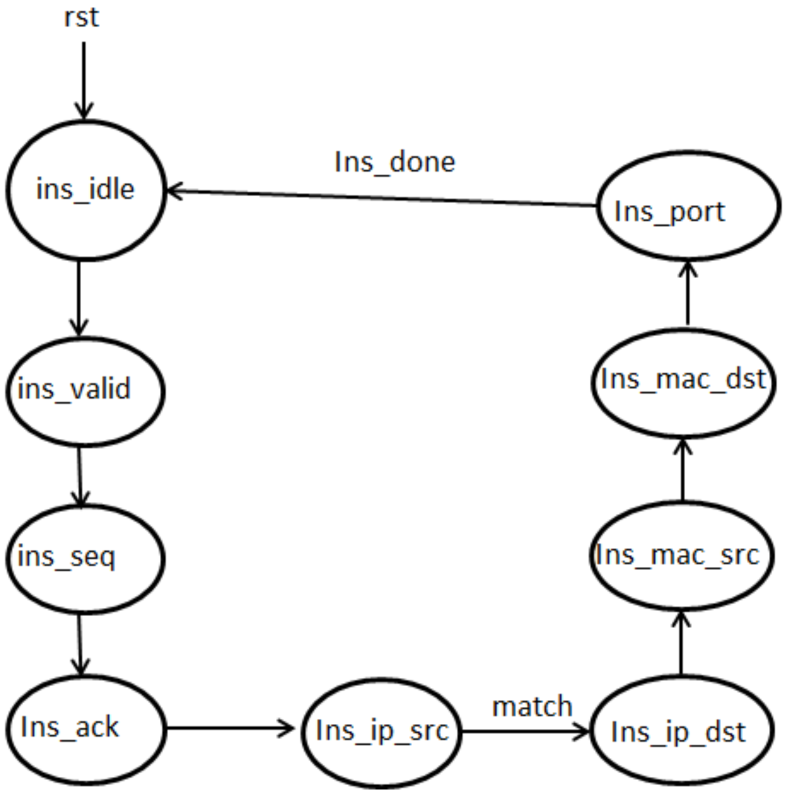
- RAM_searcher could establish/delete a TCP connection depending on the request
- When establishing a new TCP connection first try to search the RAM if there is already an existing connection
  - If found then an error would be returned
  - if not found then a new connection is created/inserted

Ram_searcher: searching for a connection



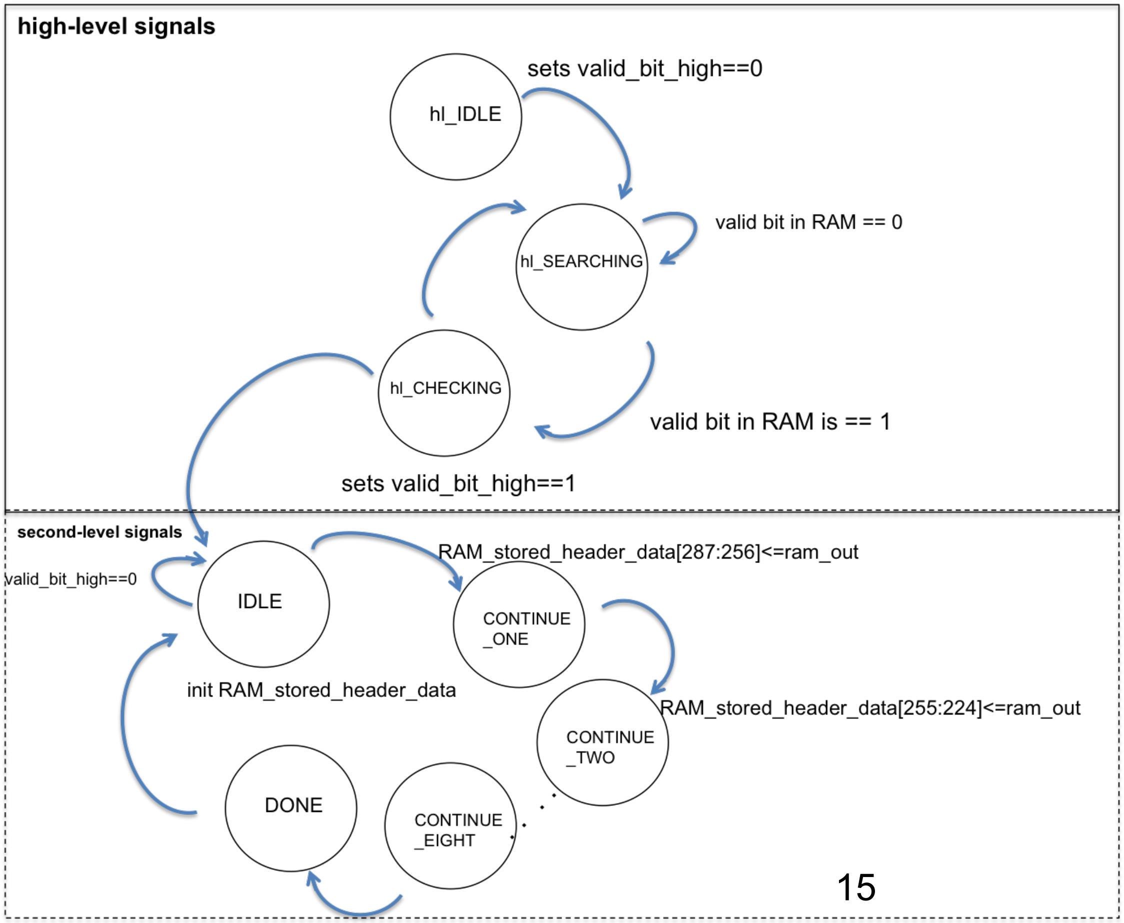
- Checks if there is already a connection
    - RAM addr incremented if going to next state
    - chk_equal means found existing connection
    - would return error
    - if not found base_address is set as the address to write into

Ram_searcher: inserting a connection



- Insert the fields into RAM sequentially
    - RAM address incremented by 1 in the next state

Hardware Module Interfaces: Packet_builder



- Header data is stored
  - -3 main Ethernet fields
  - -12 main IP fields
  - -11 main TCP fields

-Check of valid bit of each address before proceeding with process

-Syn bit in seq num is set high

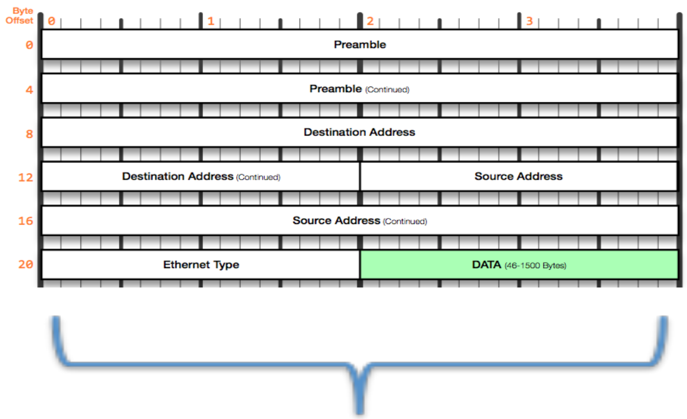
Hardware Module Interfaces: Packet_builder

-Results in a header for a packet

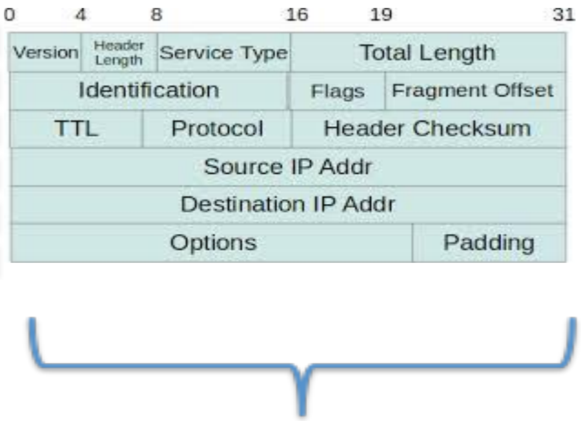
-Does what the software implementation did for making a packet

-Transmit out through Avalon ST in top-level file

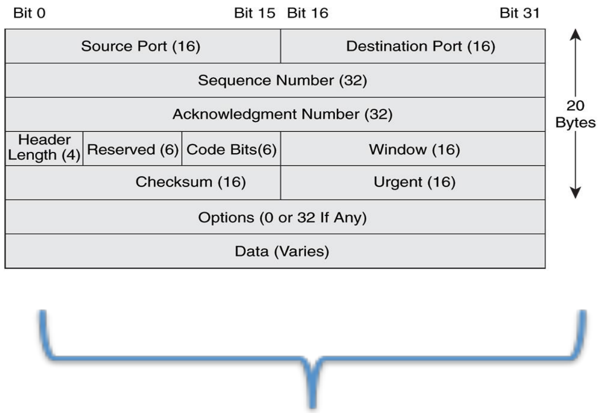
- Packet_decomposer: opposite functionality (test using SignalTap)



Ethernet



IP header



TCP header

Conclusion

- We show:
  - Software implementation of successful request for connection, with raw socket API, and Wireshark verification, for hardware implementation of TCP processing
  - Hardware implementation of integrated modules for starting a connection to send a syn packet. Verification using ModelSim.
    - Modules include: TOE_init, RAM_searcher, RAM, packet_builder

References

[1] Lockwood, J. W. “A Low Latency Library in FPGA Hardware for High Frequency Trading.” 2012 IEEE Symposium on High-Performance Interconnects. San Jose. 2012.