Using Symbolic Execution to Analyze Hardware TCP/IP Stacks Based on HLS Development

Nianhang Hu(Presenter), Witawas Srisa-an, Lisong Xu

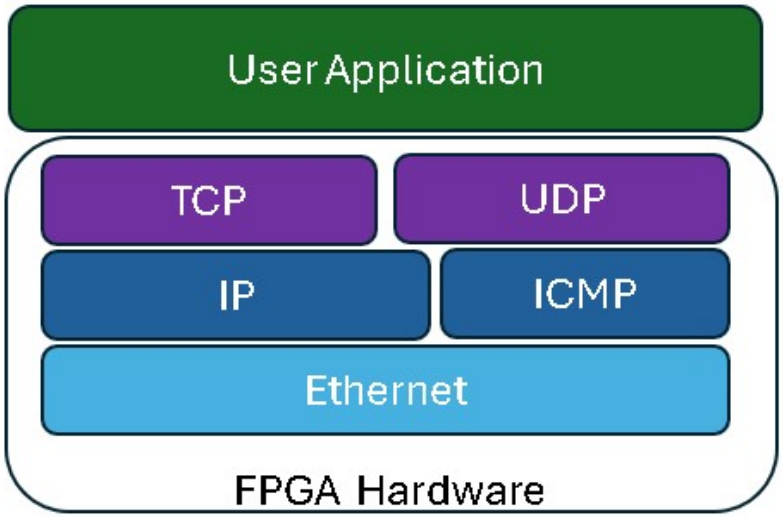
School of Computing

University of Nebraska-Lincoln

* This work is supported in part by NSF CNS-2135539

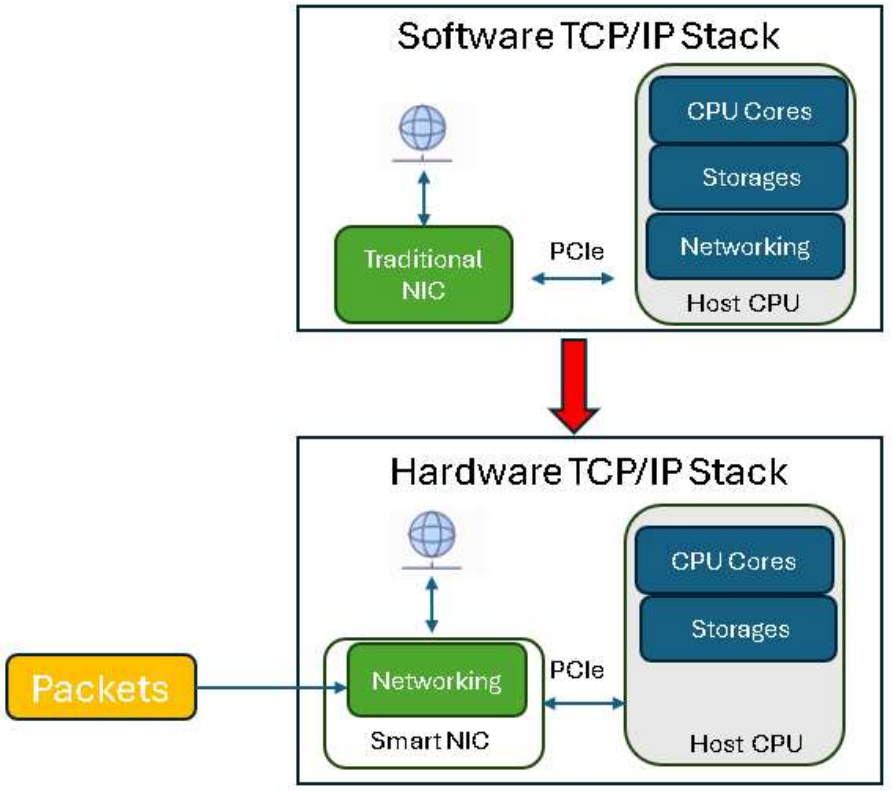
Hardware TCP/IP Stack (I)

- Implementing the TCP/IP protocol on the FPGA



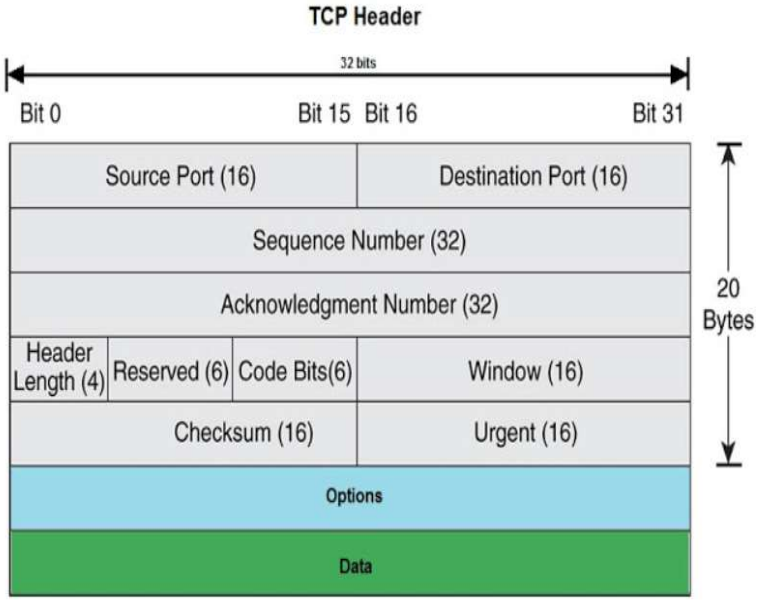
Hardware TCP/IP Stack (II)

- Improved performance.
- Increased throughput and reduced latency.
- Reduced CPU load



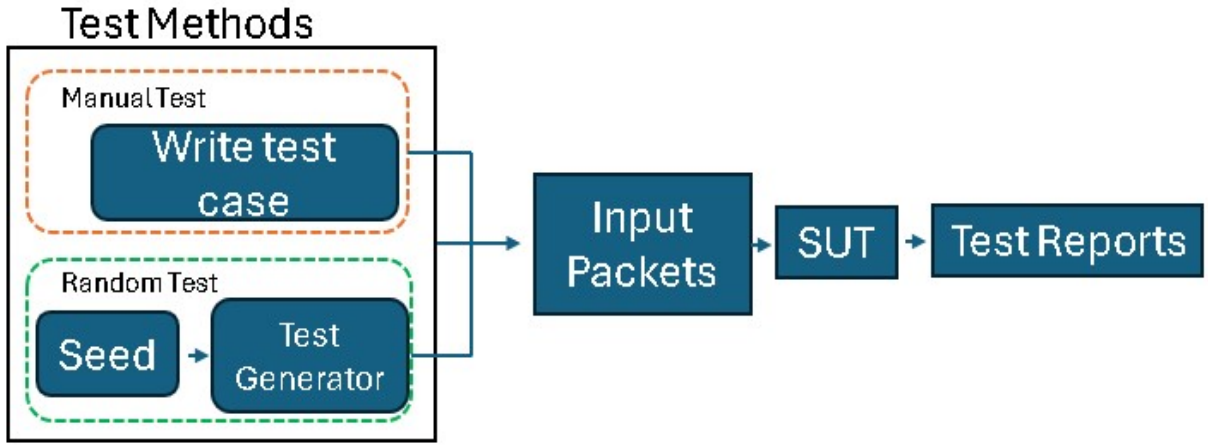
Current Hardware TCP/IP Stack Testing

- Construct test packets
- Compare results with gold.dat



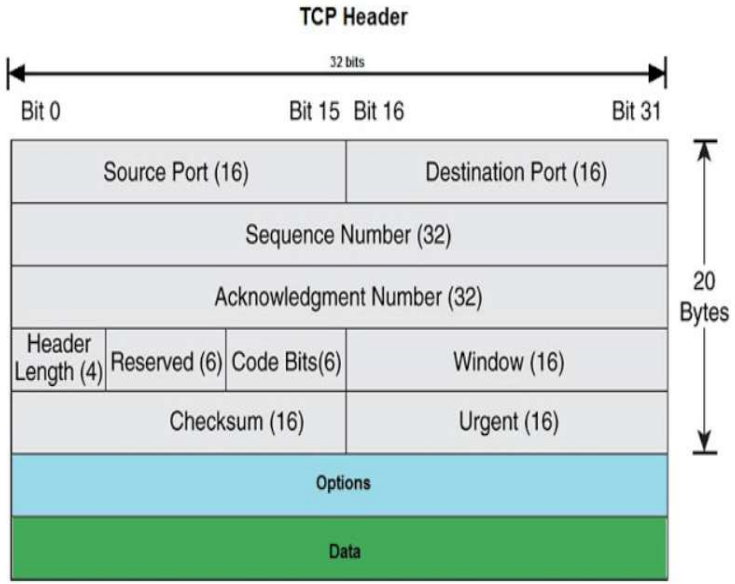
Current Test Methods

- Manual Test
- Random Test



Challenge of Current Testing

- TCP header 60 bytes long.
- Up to 2^{480} possible scenarios
- Limited scope of testing.
- Low testing efficiency.

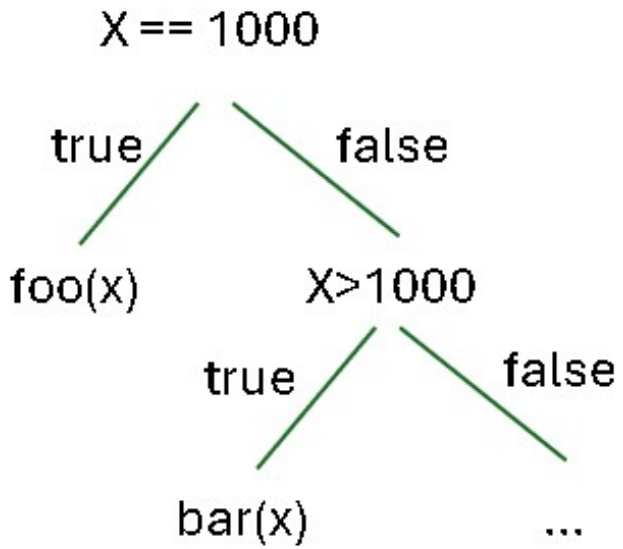


OUR WORK

- Propose a symbolic execution tool.
- Enhance code coverage for the hardware TCP/IP stack.

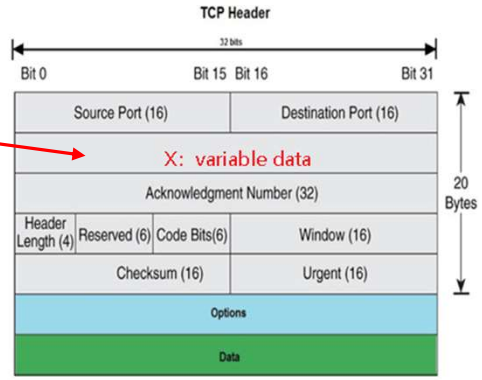
Symbolic Execution -- KLEE

```
void read(int x) {
    if (x == 1000) {
        foo(x);
    }
    else if (x > 1000)
    {
        bar(x);
    }
    else {
        ...
    }
}
```

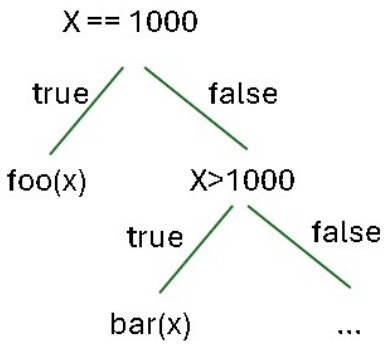


Hardware TCP Header Testing

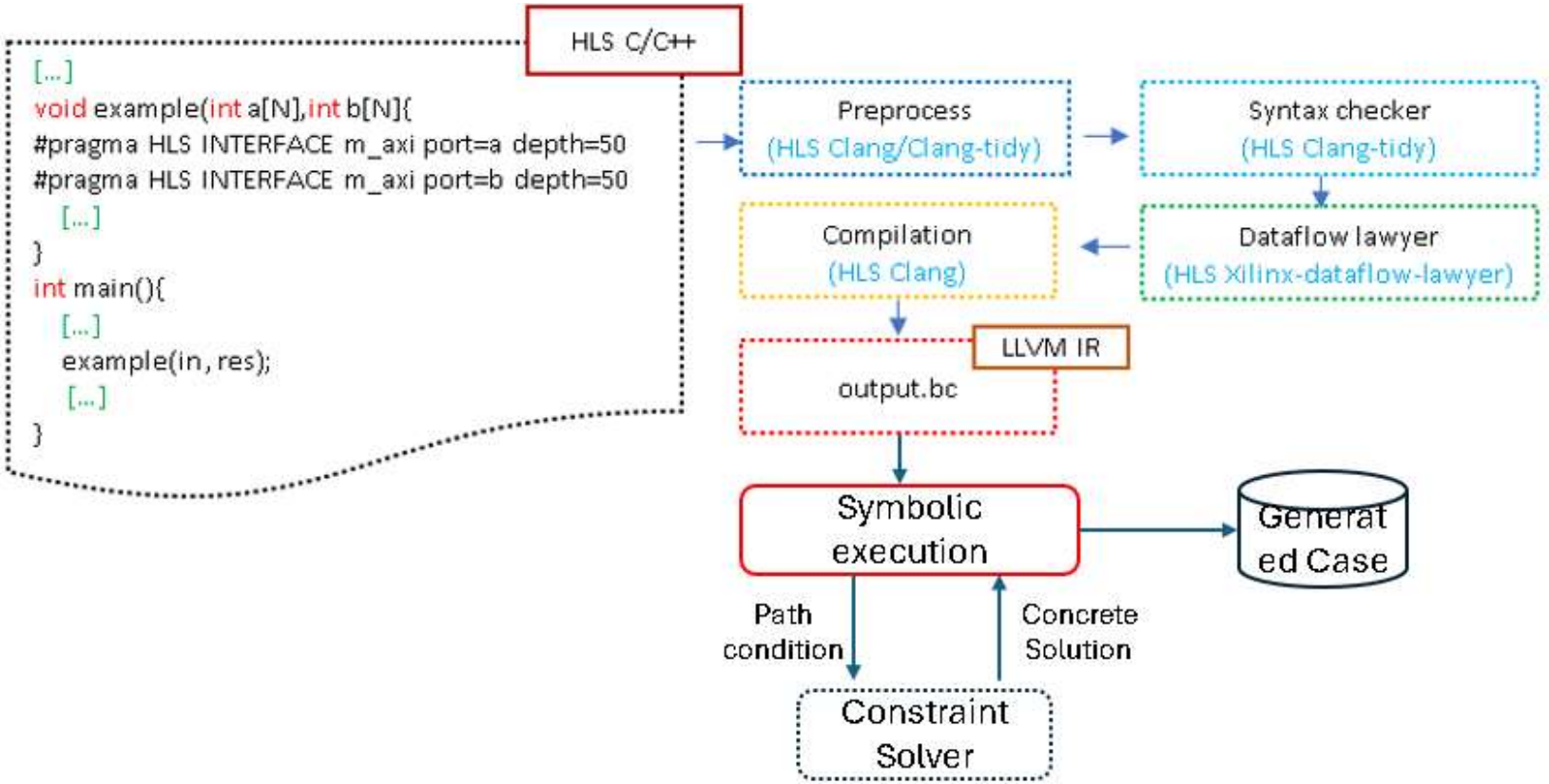
A given variable X in the TCP header:



Test Method	Case Num	Test Efficiency
Manual Test Method	Manually create a small number of cases	Depend on the expertise
Random Test Method	Need numerous cases	Uncertain probability
Symbolic Test Method	Just need 3 branches	High



Overview Hardware TCP Test Framework



Challenges of Using KLEE to Test Hardware TCP

1. KLEE-unsupported data types in HLS
2. KLEE-unsupported functions in HLS
3. Parallel computing in HLS

Issue 1: KLEE Unsupported Data Types in HLS

Solution：

Using basic data types to construct arbitrary bit-width data types

Data Type	sizeof( )
unsigned char	8bits
unsigned short	16bits
unsigned int	32bits
unsigned long	64bits

Basic data types

ap_uint<W*> data;

Special data type in HLS library

* W: bit width, from 1 to 4096

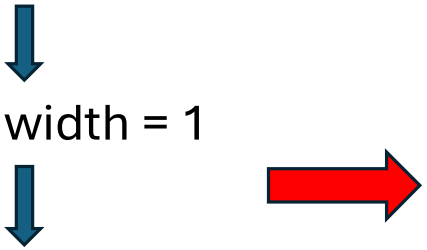
Principle - Construct special data type

- The range of data values
- Select the appropriate basic data type
- Optimize memory space
- Minimize testing time

HLS Special Data Type (I)

First example

ap_uint <1> tcpData;



data width = 1

Data range: [0,1]

unsigned char tmp


```
void make_symbolic_case () {
        ap_uint<1> tcpData;
        unsigned char tmp;
        make_symbolic(&tmp,sizeof(tmp), " tmp");
        klee_assume(tmp >=0 && tmp <=1)
        tcpData = tmp;
}
```

HLS Special Data Type (II)

Second example

ap_uint <130> tcpHeader;

data width = 130

Data range: [0,2^{130}-1]

unsigned long tmp[2]

unsigned char data

```
void make_symbolic_case () {
        ap_uint<130> tcpHeader;
        unsigned  long tmp[2];
        make_symbolic(&tmp,sizeof(tmp), " tmp");
        make_symbolic(&data,sizeof(data), " data");
        klee_assume(data >=0 && data <=3)
         tcpHeader.range(63,0) = tmp[0];
         tcpHeader.range(127,64) = tmp[1];
         tcpHeader.range(129,128) = data;
}
```

Issue 2: KLEE Unsupported Functions in HLS

Solution:

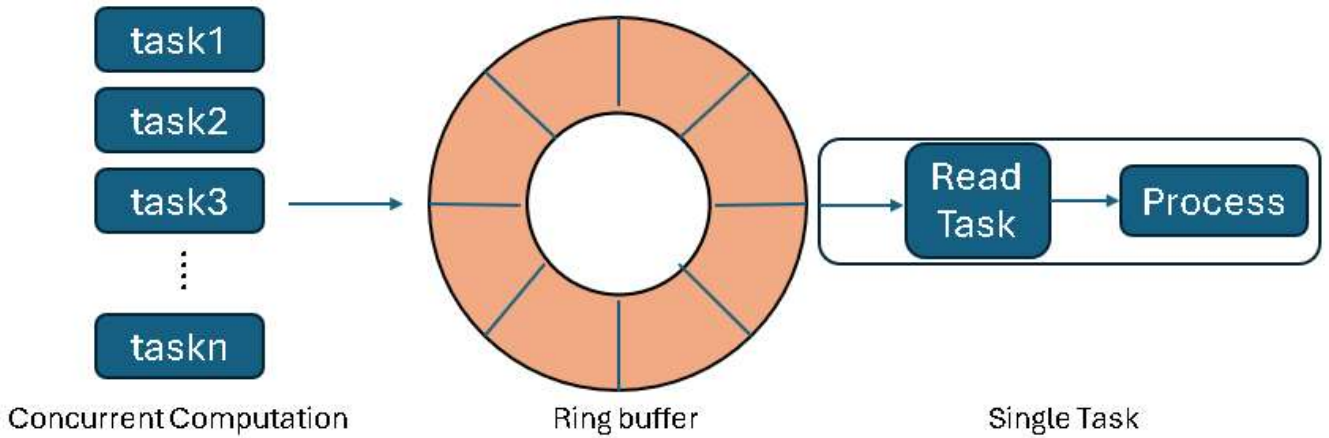
Re-implemented the methods mentioned below.

- Stream Extraction Operator : >>, <<
- Exception handling function:
  - try – catch
  - open and fail in std::ifstream file

Issue 3: Parallel Computing in HLS

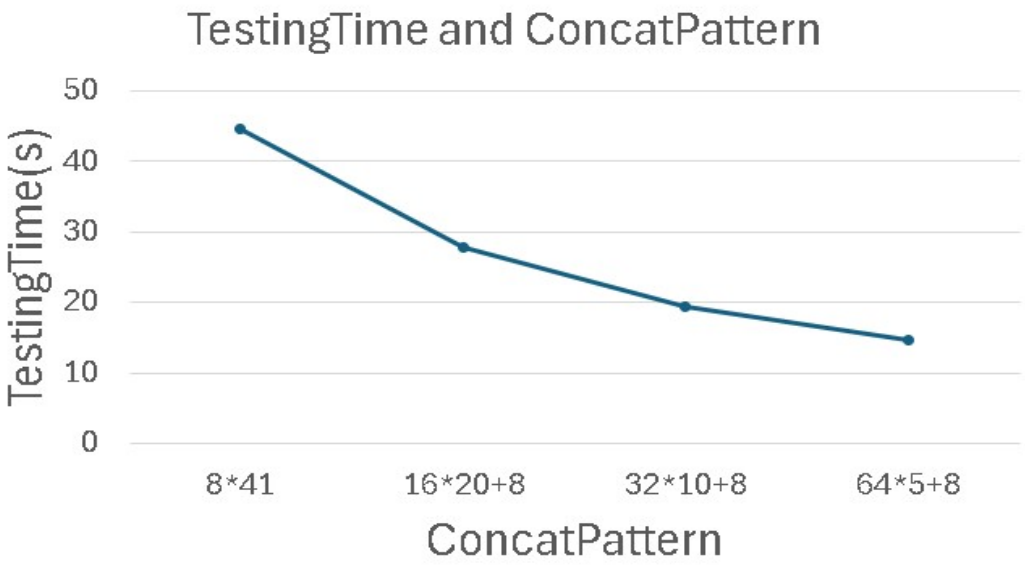
Solution

Modify multithread tasks to single-thread tasks

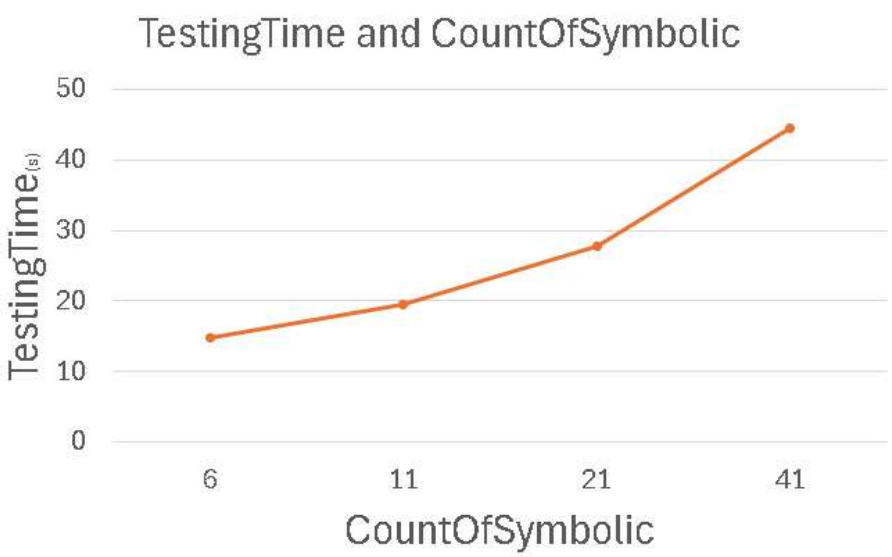


Experiment

- Optimize testing time.



Different data combinations result in varying testing times.



The fewer symbolic variables there are, the shorter the testing time will be.

Conclusion and Future Work

- Modified HLS so that HLS-based hardware TCP/IP stack can be executed using KLEE

- On-going
  - Test and compare the code coverage of random testing and symbolic execution testing
  - Continue to optimize HLS to make symbolic execution more efficient

Q & A

Nianhang Hu

hunianhang2001@gmail.com

402-853-6104

linkedin.com/in/nianhanghu-9527