Blackwire™
WireGuard in HDL
100+ Gbit/s
on an FPGA NIC
Leon Woestenberg
state-of-the-art clean-slate protocol for building network overlays with encryption and authentication, IP/IP, many use cases, trusted crypto.

- a WireGuard peer is a host in the VPN, identified by its public key. each peer can potentially reach every remote peer directly: meshing

- a WireGuard endpoint is the public (or outer) UDP address. may change when endpoint roams; the tunnel stays up.

- each peer has a set of allowed IP address prefixes within tunnel. multicasting is possible
256 bit session key

- **IPv4 Header**
  - **UDP**
  - **UDP Payload**
  - **WireGuard Payload**
  - **IP Packet**
    - **encrypt: ChaCha20**
    - **auth: Poly1305**

**Ethernet Frame**
WireGuard Type 4 packet for data packets

- IPv4 Header
- IP Payload
- UDP Payload
- WireGuard Payload
- encrypt: ChaCha20
- auth: Poly1305

Ethernet Frame
Why WireGuard on an FPGA?

- Speed growth curve of Ethernet is steeper than that of CPUs. Offloading 'instructure tasks' such as VPN/encryption and authentication to smartNIC FPGAs becomes cost effective.
- Deterministic guaranteed behaviour vs. best-effort on CPU.

Example use case:

- 9th February 2023 – The Video Services Forum (VSF), has further enhanced the Reliable Internet Streaming Transport (RIST) protocol with the use of WireGuard VPN in RIST devices.
Map WireGuard into our FPGA design space

WireGuard Type 4 data packets

WGT1-3

Ethernet

new session handshake

sessions/peer

previous

current

next

decrypt & authenticate peer on RX

encrypt & add authentication on TX

PCle or Ethernet
Map WireGuard into our FPGA design space

- Headers
- Decrypt
- Auth
- Prevent replay
- Crypto firewall

Handshake:
- Peer public keys
- Timers
- x25519

Session:
- Keys
- Indices

Allowed IP's
- Crypto routing
- Crypto firewall

Endpoint UDP addr

Headers
- Auth
- Encrypt
- Nonce
- Crypto routing
100 Gbit/s, 512 bits wide AXI Streaming 250 MHz
worst case one packet header in each clock cycle

roughly one handshake per minute per active peer connection

1024 peers → 58 ms budget per handshake take conservative design budget: 25 ms?
Let's do this in RTL

3. Let's do this on a RISC-V VexRiscv maybe with some x25519 accelerator (RISC-V s/w, custom instructions or custom accelerator).

Let's do this in RTL
Design choices: (De)coupling data/control

- decrypt
- auth
- prevent replay
- crypto firewall
- nonce counter
- control software on VexRiscv RISC-V
- AXI4 x-bar
- RTL logic and per-session lookup table

- headers
- auth
- encrypt
- nonce
- crypto routing
Design choices: (De)coupling data/control

headers decrypt auth

prevent replay

crypto firewall

control software on VexRiscv RISC-V

nonce counter

AXI4 x-bar

previous current next

headers auth encrypt nonce crypto

routing
Design choices: (De)coupling data/control

Control software on VexRiscv RISC-V

AXI4 x-bar

Allowed IP sets

Endpoints

Headers

Decrypt

Auth

Prevent replay

Nonce counter

Nonce counter

Counter

Counter

UDP addr

Session index

Session tx key

Nonce

Nonce

Encrypt

Route

Crypto

Firewall
3. Let's do this on a RISC-V VexRiscv and build everything with SpinalHDL

Programmatically generate RTL, but also SoC design at much higher level!
Example: Allowed IP address prefix lookup

headers - decrypt - auth - prevent replay - crypto firewall

session rx key
nonce
encrypt
crypto

control software on VexRiscv RISC-V

allowed IP set

endpoint UDP addr - session index - session tx key - nonce
crypto routing

allowed IP set

nonce counter

AXI4 x-bar
Allowed IP lookup:
Longest prefix match using binary tree

Address Lookup Using Tries

- Prefixes “spelled out” by following path from root
- To find the best prefix spell out address in trie.
Pipeline: one IP address bit per clock

- Multi stage pipeline, one stage for each address bit.
- 32 bits/stages for IPv4, 128 for IPv6, 129 for both.
- Each stage has a memory (LUT, distributed or BRAM).
- Use optimally balanced tree (reduce worst case RAM use!).

Result: longest prefix match

Match Addr w/ Prefix Set (Tree)

<table>
<thead>
<tr>
<th>Lookup Stage 0</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup Stage 1</td>
<td>RAM</td>
</tr>
<tr>
<td>Lookup Stage 2</td>
<td>RAM</td>
</tr>
<tr>
<td>Lookup Stage 3</td>
<td>RAM</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Lookup Stage 31</td>
<td>RAM</td>
</tr>
</tbody>
</table>

12:00
Logic for one stage:

Vivado: report_design_analysis -logic_level_distribution
➔ Two pipeline stages per bit
➔ Balanced combinatorial logic (≤ 4 levels) between registers
➔ 400 MHz on Ultrascale+
➔ True dual port RAM (RX reads, TX reads, RISC-V writes).
➔ **800 million Allowed IP address prefix lookups per second.**
Blackwire builds upon top-notch OSH:

➔ Blackwire uses **SpinalHDL** (Thanks Charles Papon!)
  ◆ write cycle efficient RTL in less code
  ◆ zero cost (no overhead) abstractions
  ◆ the Spinal (building blocks) library is a piece of art

➔ Blackwire uses **Corundum** (Thanks Alex Forencich!)
  ◆ SGDMA NIC design for PCIe and Ethernet FPGA boards
  ◆ comes with Linux Kernel device driver

and tools... Verilator, GHDL, CocoTB, GTKWave, SymbiYosys, ... thanks to everyone committing to those projects.

Thanks for explaining **formal verification** (Thanks Matt Venn!)
Blackwire Project Status (9/2023)

➔ Open sourced HDL on GitHub, some WIP to follow
https://github.com/brightai-nl/BlackwireOverview

FAQ: Actual code repositories listed in README!

➔ Are We WireGuard Yet (AWWY)?
◆ 75% done;
◆ 25% to do, see README on GitHub for TODOs.
Blackwire IP Core

AXI4 Stream ➔ detunnel ➔ decrypt ➔ allow ➔ AXI4 Stream

manage remote peers, sessions and keys

AXI4 Stream ➔ AXI4 Configuration

AXI4 Stream ➔ tunnel ➔ encrypt ➔ route ➔ AXI4 Stream

→ https://github.com/brightai-nl/BlackwireOverview
Blackwire: 'wg0' but implemented on FPGA

tunneled encrypted traffic

(Internet facing)

network interface 'wg0' with plaintext IP packets
Blackwire: integrated in network infrastructure

tunneled
encrypted
traffic

plaintext
traffic

control interface
over PCIe (or other
interface)
Blackwire  WireGuard

Thanks!    Questions?

→ https://github.com/brightai-nl/BlackwireOverview

→ Q&A e-mail: Leon Woestenberg <leon@brightai.nl>
Complementary Slides
Blackwire FPGA Resources ~ (for 100 Gbit)

Alveo U50 example with Corundum + WireGuard (BRAM, no URAM)

RX path is 128 Gbit/s, TX path is 64 Gbit/s in this design

<table>
<thead>
<tr>
<th>Name</th>
<th>^1 CLB LUTs (871680)</th>
<th>^1 CLB Registers (1743360)</th>
<th>^1 CARRY8 (108960)</th>
<th>^1 F7 Muxes (435840)</th>
<th>^1 F8 Muxes (217920)</th>
<th>^1 CLB (108960)</th>
<th>^1 LUT as Logic (871680)</th>
<th>^1 LUT as Memory (403200)</th>
<th>Block RAM Tile (1344)</th>
<th>^1 URAM (640)</th>
<th>^1 DSPs (5952)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N fpga</td>
<td>31.39%</td>
<td>32.04%</td>
<td>18.67%</td>
<td>0.36%</td>
<td>0.11%</td>
<td>58.96%</td>
<td>26.90%</td>
<td>9.71%</td>
<td>25.78%</td>
<td>6.09%</td>
<td>21.91%</td>
</tr>
</tbody>
</table>

Resources (roadmap: move more registers into BRAM/URAM)

<table>
<thead>
<tr>
<th>Name</th>
<th>^1 CLB LUTs (871680)</th>
<th>^1 CLB Registers (1743360)</th>
<th>^1 CARRY8 (108960)</th>
<th>^1 F7 Muxes (435840)</th>
<th>^1 F8 Muxes (217920)</th>
<th>^1 CLB (108960)</th>
<th>^1 LUT as Logic (871680)</th>
<th>^1 LUT as Memory (403200)</th>
<th>Block RAM Tile (1344)</th>
<th>^1 URAM (640)</th>
<th>^1 DSPs (5952)</th>
</tr>
</thead>
<tbody>
<tr>
<td>app.app_block_inst (mqnic_app_block)</td>
<td>211501</td>
<td>462593</td>
<td>19298</td>
<td>649</td>
<td>8</td>
<td>50192</td>
<td>182601</td>
<td>28900</td>
<td>153</td>
<td>21</td>
<td>1300</td>
</tr>
</tbody>
</table>

add the following numbers for 100 Gbit/s full duplex:
subtract the following numbers for ~60 Gbit/s full duplex

<table>
<thead>
<tr>
<th>Name</th>
<th>^1 CLB LUTs (871680)</th>
<th>^1 CLB Registers (1743360)</th>
<th>^1 CARRY8 (108960)</th>
<th>^1 F7 Muxes (435840)</th>
<th>^1 F8 Muxes (217920)</th>
<th>^1 CLB (108960)</th>
<th>^1 LUT as Logic (871680)</th>
<th>^1 LUT as Memory (403200)</th>
<th>Block RAM Tile (1344)</th>
<th>^1 URAM (640)</th>
<th>^1 DSPs (5952)</th>
</tr>
</thead>
<tbody>
<tr>
<td>packetTx_tx (BlackwireTransmit)</td>
<td>67906</td>
<td>156822</td>
<td>6397</td>
<td>185</td>
<td>4</td>
<td>17931</td>
<td>59587</td>
<td>8319</td>
<td>20.5</td>
<td>0</td>
<td>432</td>
</tr>
</tbody>
</table>
Olof Kindgren (He/Him) (He/Him) • 1st
Award-winning Engineer and Actor at Qamcom

This is super interesting. I have been looking at doing exactly the same thing. Is this a proprietary or open source implementation? Would love to read some more about the work.
Olof Kindgren (He/Him) (He/Him) • 1st
Award-winning Engineer and Actor at Qamcom

This is super interesting. I have been looking at doing exactly the same thing. Is this a proprietary or open source implementation? Would love to read some more about the work.

Olof Kindgren (He/Him) (He/Him) • 1st
Award-winning Engineer and Actor at Qamcom

Ah! It looks like it is written in SpinalHDL too :D I'm sure Charles Papon must be happy to see that :)

Like · 6 | Reply · 3 Replies