Verifiable Computing Project

“How do we build trusted, open hardware?”

Joyce Ng

Talk Outline

- Why care about verifiable computing?
- Why use FPGAs?
- Existing Works
- FPGA Design Flow
- Toolchains
- Implementation
- Demo
- Planned Projects

What will NOT be covered

- Computer Architecture
- Digital Design
- VLSI design

Why care about verifiable computing?

What do these devices have in common?

Why care about verifiable computing?

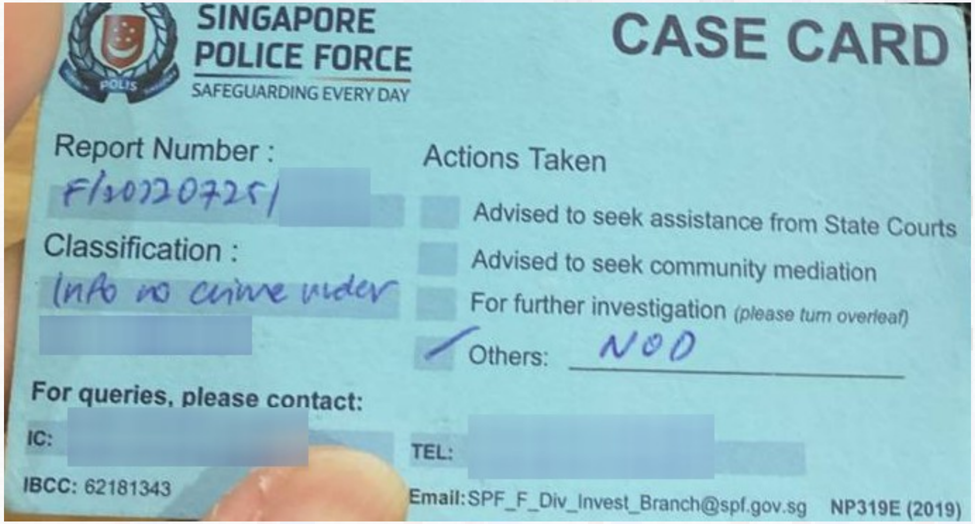
They all are computers!

Why care about verifiable computing?

- End user:
  ”How do I know that the devices I use are trustworthy?”

  Chip designer: “How do I know that the chips I get from the fab are exactly as per design and not tampered with?

Why care about verifiable computing?



SINGAPORE POLICE FORCE

SAFEGUARDING EVERY DAY

CASE CARD

Report Number : F/20220725/ [redacted]

Classification : Info no crime under [redacted]

Actions Taken

□ Advised to seek assistance from State Courts

□ Advised to seek community mediation

□ For further investigation *(please turn overleaf)*

☑ Others: NOD

For queries, please contact:

IC: [redacted] TEL: [redacted]

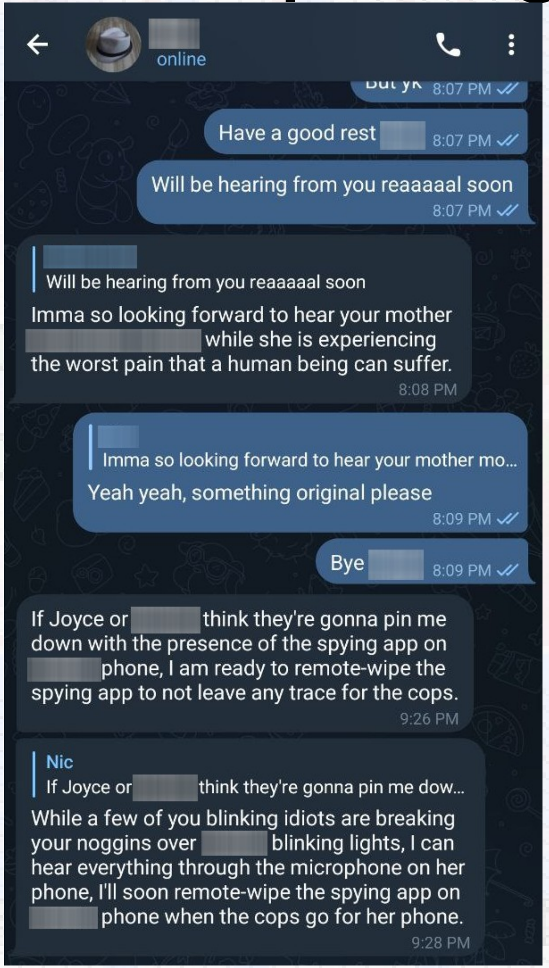
IBCC: 62181343

Email:SPF_F_Div_Invest_Branch@spf.gov.sg NP319E (2019)

Real life example – iPhone X compromised!

Why care about verifiable computing?

- Phone known to be compromised as adversary messages contact with things said on a live Zoom call
- Police forensics unable to find entry point nor traces of stalkerware
- Phone not jailbroken by owner
- Many consumer might as well be black boxes to their owners!



[redacted]
online

But yk 8:07 PM

Have a good rest [redacted] 8:07 PM

Will be hearing from you reaaaaal soon 8:07 PM

> [redacted]
> Will be hearing from you reaaaaal soon

Imma so looking forward to hear your mother [redacted] while she is experiencing the worst pain that a human being can suffer. 8:08 PM

> [redacted]
> Imma so looking forward to hear your mother mo...

Yeah yeah, something original please 8:09 PM

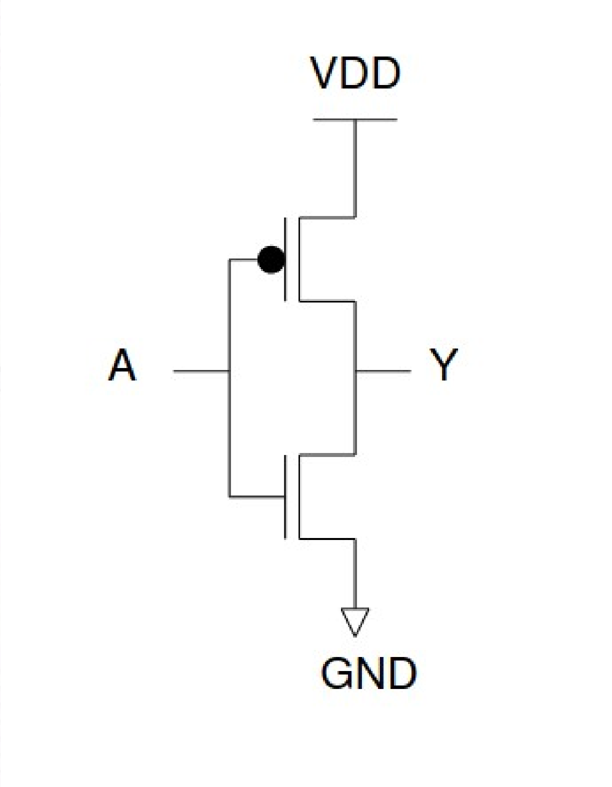
Bye [redacted] 8:09 PM

If Joyce or [redacted] think they're gonna pin me down with the presence of the spying app on [redacted] phone, I am ready to remote-wipe the spying app to not leave any trace for the cops. 9:26 PM

> Nic
> If Joyce or [redacted] think they're gonna pin me dow...

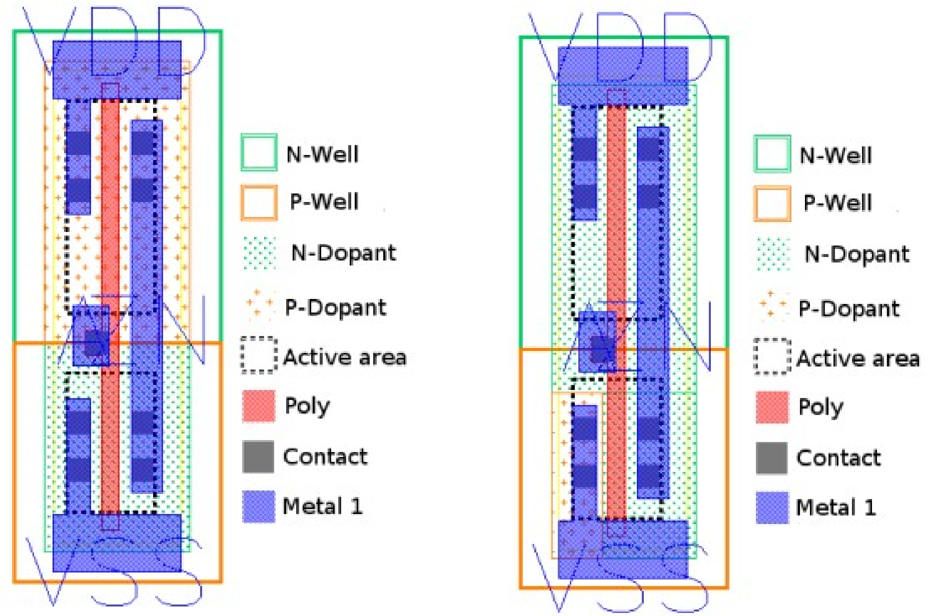
While a few of you blinking idiots are breaking your noggins over [redacted] blinking lights, I can hear everything through the microphone on her phone, I'll soon remote-wipe the spying app on [redacted] phone when the cops go for her phone. 9:28 PM

Why care about verifiable computing?



Second example - Simple CMOS Inverter

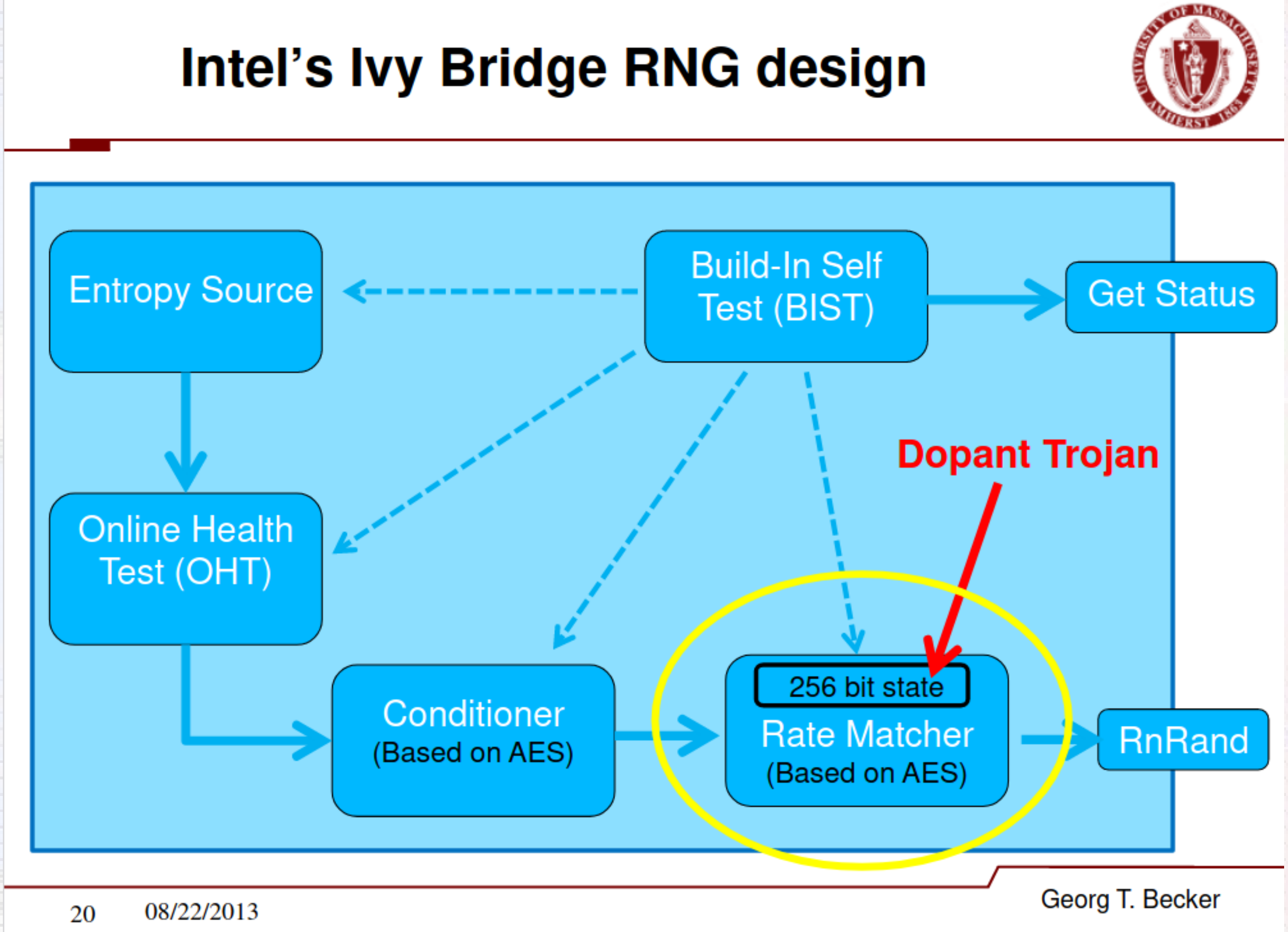
Why care about verifiable computing?



(a) Original (b) Trojan

Try spotting the difference!

Why care about verifiable computing?



Intel’s Ivy Bridge RNG design

Entropy Source

Build-In Self Test (BIST)

Get Status

Online Health Test (OHT)

Dopant Trojan

Conditioner (Based on AES)

256 bit state

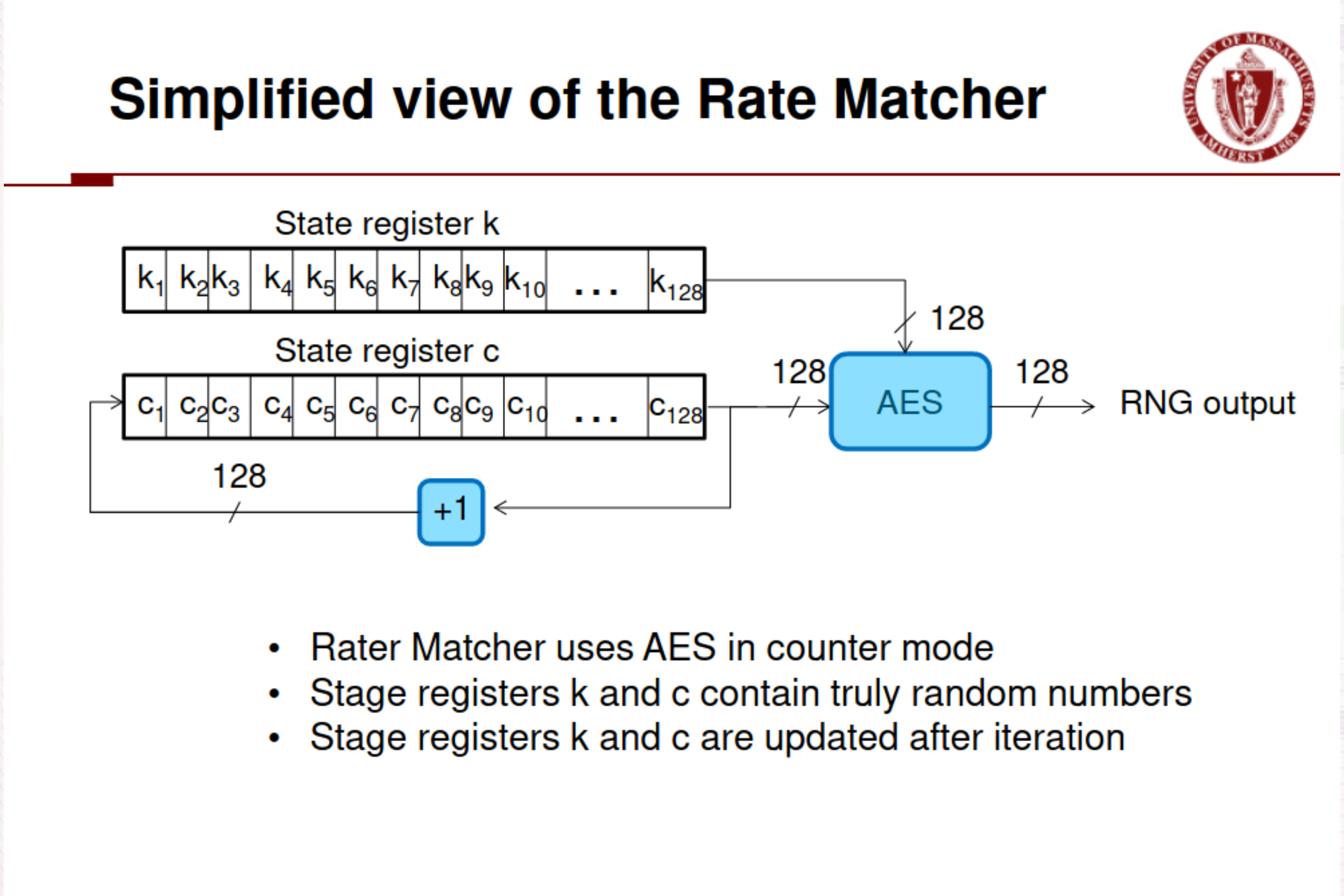
Rate Matcher (Based on AES)

RnRand

20 08/22/2013

Georg T. Becker

Why care about verifiable computing?



Simplified view of the Rate Matcher

State register k

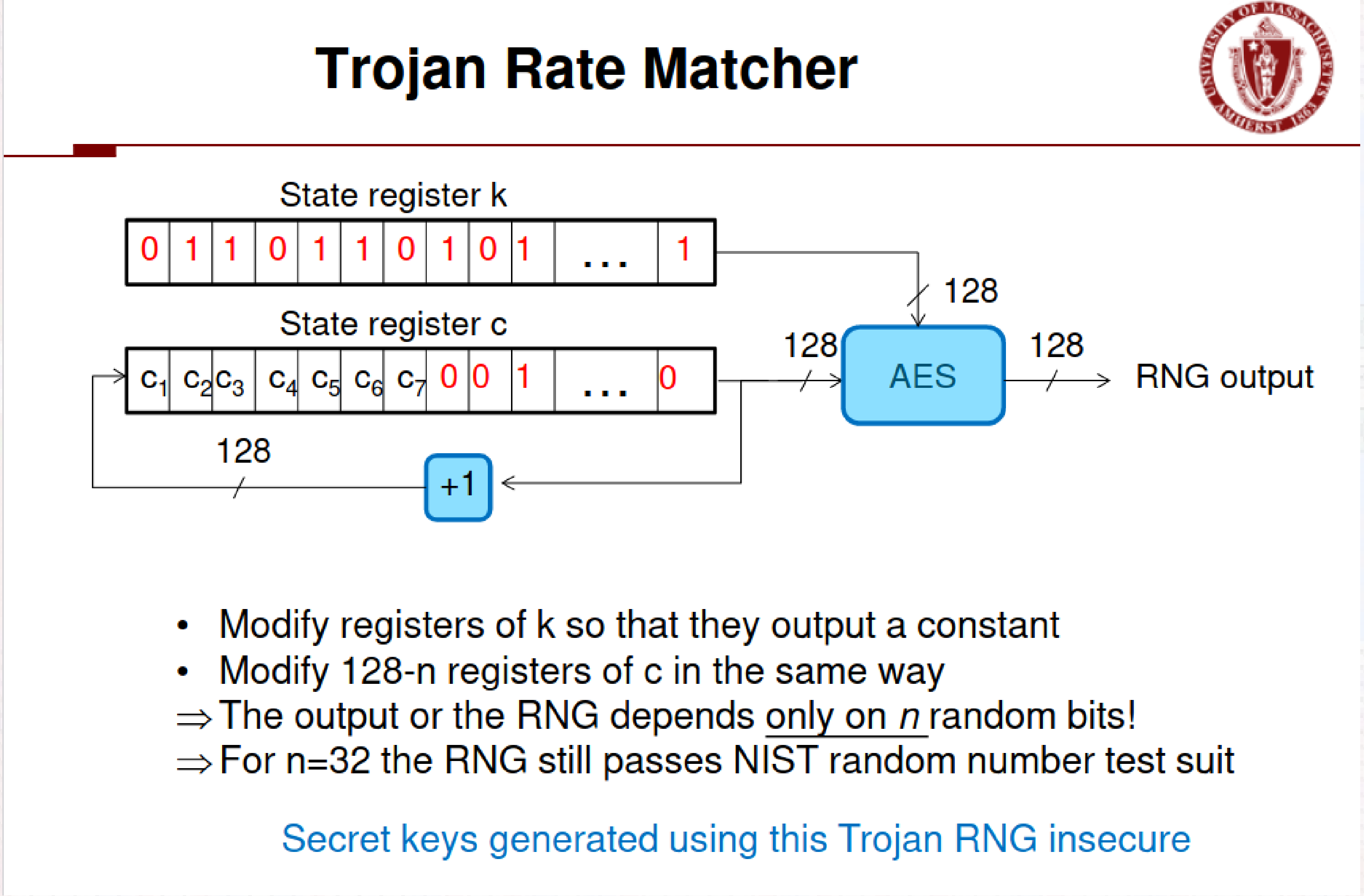
k$_1$ k$_2$ k$_3$ k$_4$ k$_5$ k$_6$ k$_7$ k$_8$ k$_9$ k$_{10}$ . . . k$_{128}$

State register c

c$_1$ c$_2$ c$_3$ c$_4$ c$_5$ c$_6$ c$_7$ c$_8$ c$_9$ c$_{10}$ . . . c$_{128}$

- Rater Matcher uses AES in counter mode
- Stage registers k and c contain truly random numbers
- Stage registers k and c are updated after iteration

Why care about verifiable computing?



Trojan Rate Matcher

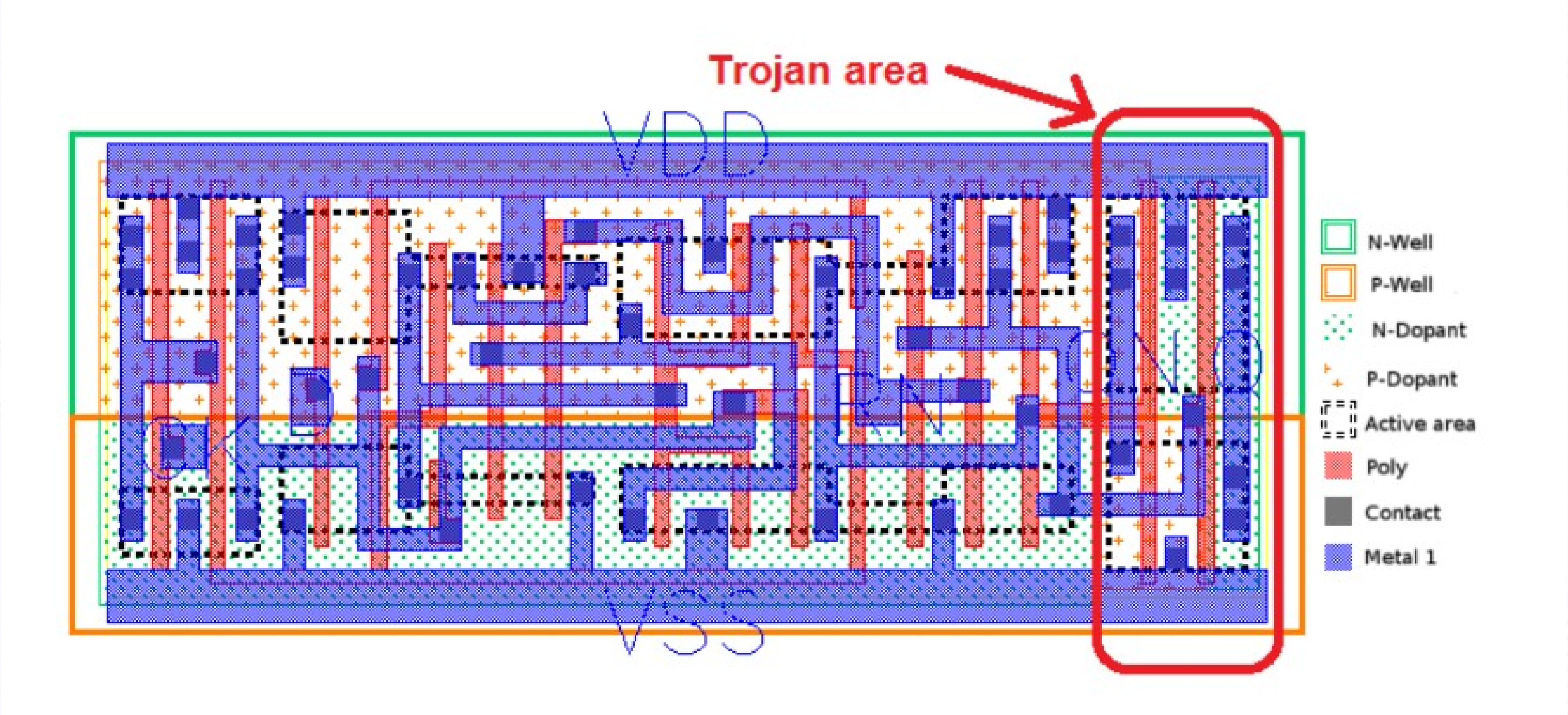
- Modify registers of k so that they output a constant
- Modify 128-n registers of c in the same way

⇒ The output or the RNG depends only on *n* random bits!

⇒ For n=32 the RNG still passes NIST random number test suit

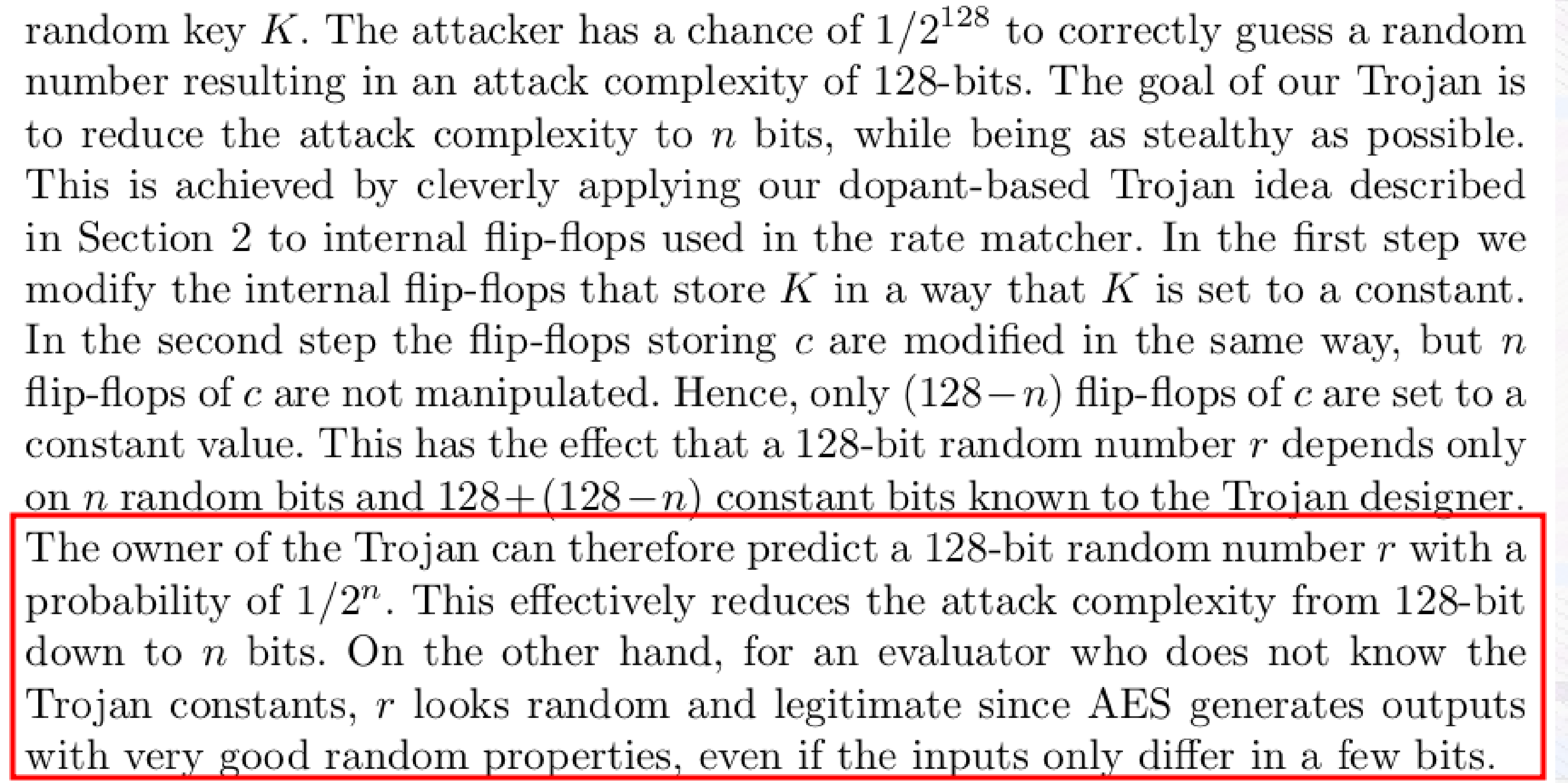
Secret keys generated using this Trojan RNG insecure

Why care about verifiable computing?



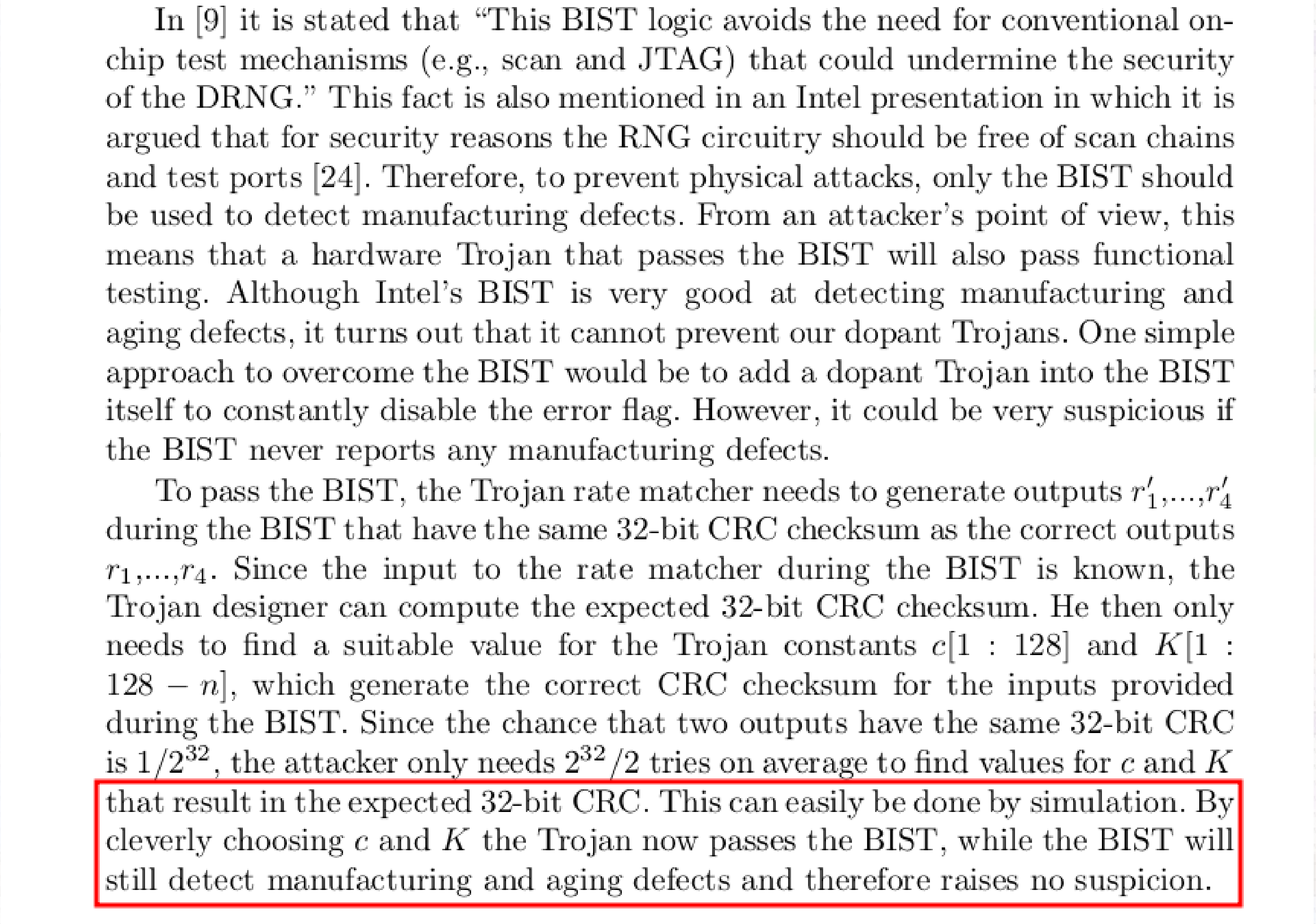
RNG with dopant level trojan

Why care about verifiable computing?



random key K. The attacker has a chance of $1/2^{128}$ to correctly guess a random number resulting in an attack complexity of 128-bits. The goal of our Trojan is to reduce the attack complexity to n bits, while being as stealthy as possible. This is achieved by cleverly applying our dopant-based Trojan idea described in Section 2 to internal flip-flops used in the rate matcher. In the first step we modify the internal flip-flops that store K in a way that K is set to a constant. In the second step the flip-flops storing c are modified in the same way, but n flip-flops of c are not manipulated. Hence, only $(128-n)$ flip-flops of c are set to a constant value. This has the effect that a 128-bit random number r depends only on n random bits and $128+(128-n)$ constant bits known to the Trojan designer. The owner of the Trojan can therefore predict a 128-bit random number r with a probability of $1/2^n$. This effectively reduces the attack complexity from 128-bit down to n bits. On the other hand, for an evaluator who does not know the Trojan constants, r looks random and legitimate since AES generates outputs with very good random properties, even if the inputs only differ in a few bits.

Why care about verifiable computing?



In [9] it is stated that “This BIST logic avoids the need for conventional on-chip test mechanisms (e.g., scan and JTAG) that could undermine the security of the DRNG.” This fact is also mentioned in an Intel presentation in which it is argued that for security reasons the RNG circuitry should be free of scan chains and test ports [24]. Therefore, to prevent physical attacks, only the BIST should be used to detect manufacturing defects. From an attacker’s point of view, this means that a hardware Trojan that passes the BIST will also pass functional testing. Although Intel’s BIST is very good at detecting manufacturing and aging defects, it turns out that it cannot prevent our dopant Trojans. One simple approach to overcome the BIST would be to add a dopant Trojan into the BIST itself to constantly disable the error flag. However, it could be very suspicious if the BIST never reports any manufacturing defects.

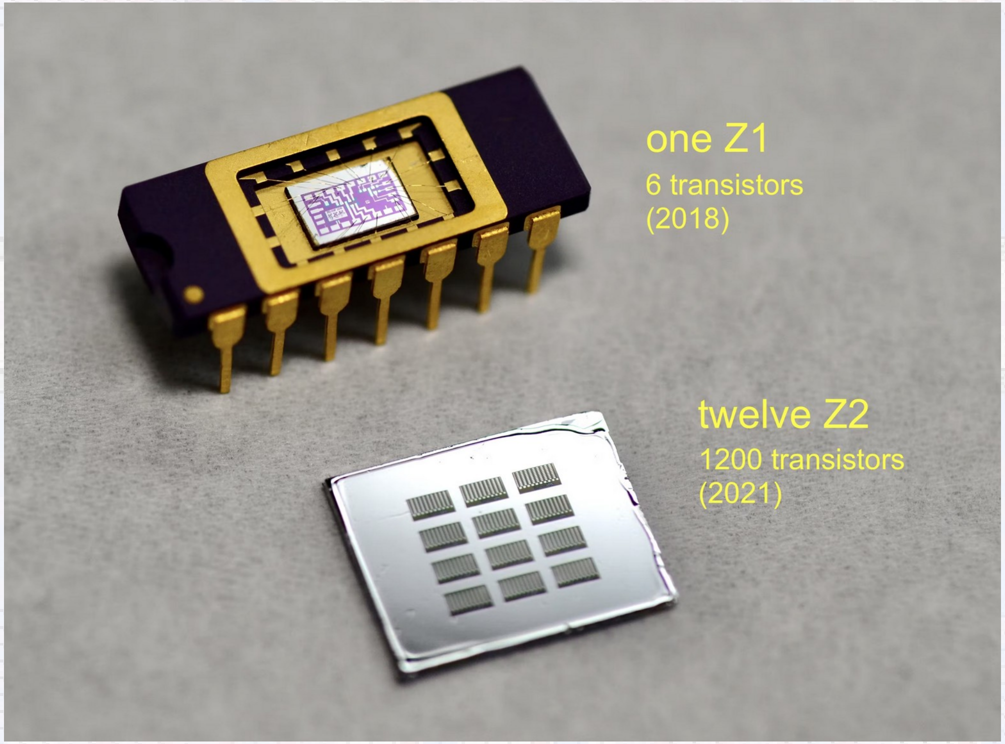
To pass the BIST, the Trojan rate matcher needs to generate outputs $r'_1,\ldots,r'_4$ during the BIST that have the same 32-bit CRC checksum as the correct outputs $r_1,\ldots,r_4$. Since the input to the rate matcher during the BIST is known, the Trojan designer can compute the expected 32-bit CRC checksum. He then only needs to find a suitable value for the Trojan constants $c[1 : 128]$ and $K[1 : 128 - n]$, which generate the correct CRC checksum for the inputs provided during the BIST. Since the chance that two outputs have the same 32-bit CRC is $1/2^{32}$, the attacker only needs $2^{32}/2$ tries on average to find values for c and K that result in the expected 32-bit CRC. This can easily be done by simulation. By cleverly choosing c and K the Trojan now passes the BIST, while the BIST will still detect manufacturing and aging defects and therefore raises no suspicion.

Why care about verifiable computing?

- Dopant level attacks cannot be found even with Scanning Electron Microscopy!
- Only way to detect is to compare production chips with “golden chip”
- Hard to know if chip is even tampered unless you already are looking out for a specific attack!
- Difficult to inspect fab process; easy for state level actor to carry out said attack

Why care about verifiable computing?

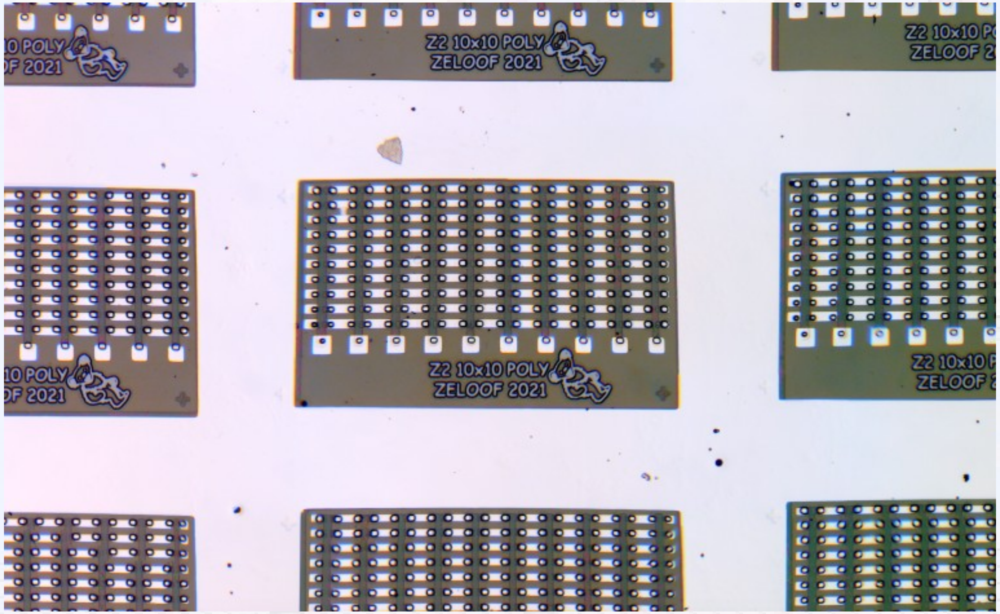
- Unfortunately, an open source chip fabrication process doesn’t yet exist
- Work is being done in that space. (see image!)
- What can we do for now?

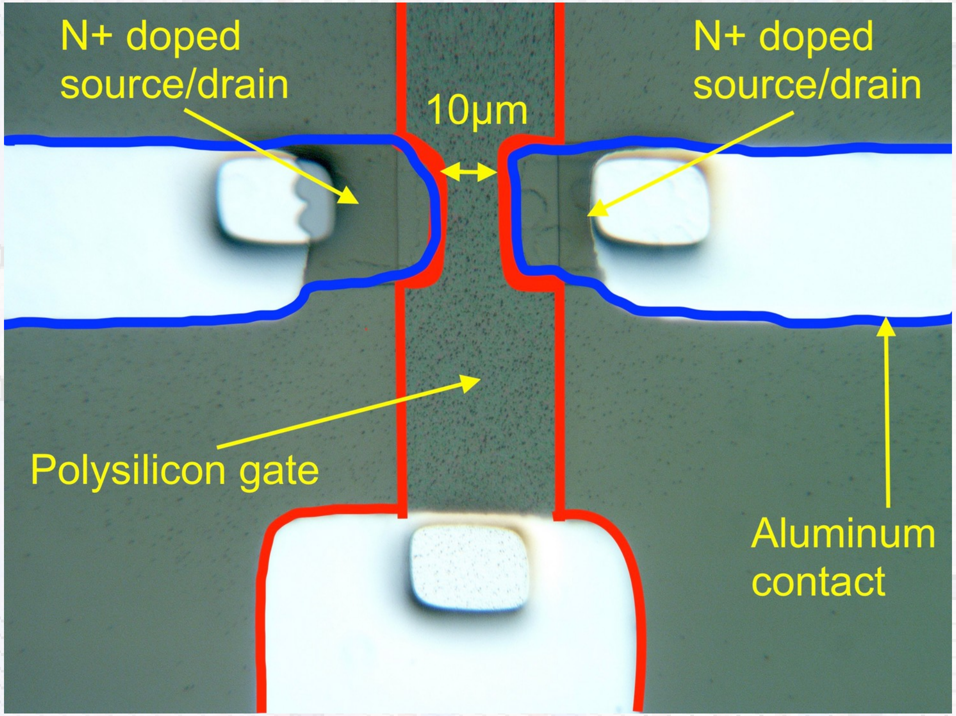


one Z1
6 transistors
(2018)

twelve Z2
1200 transistors
(2021)

Sam Zeloof DIY Z2 Chip – Photos!



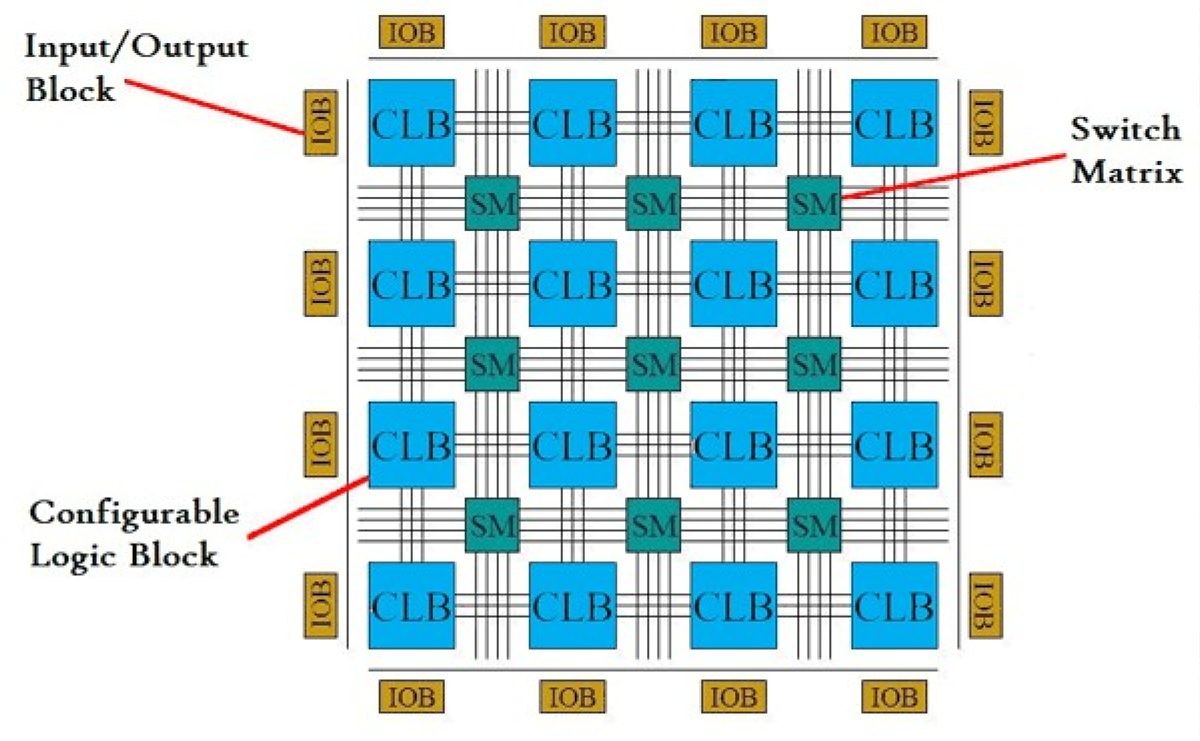


Sam Zeloof Chip Lab

Why use FPGAs?

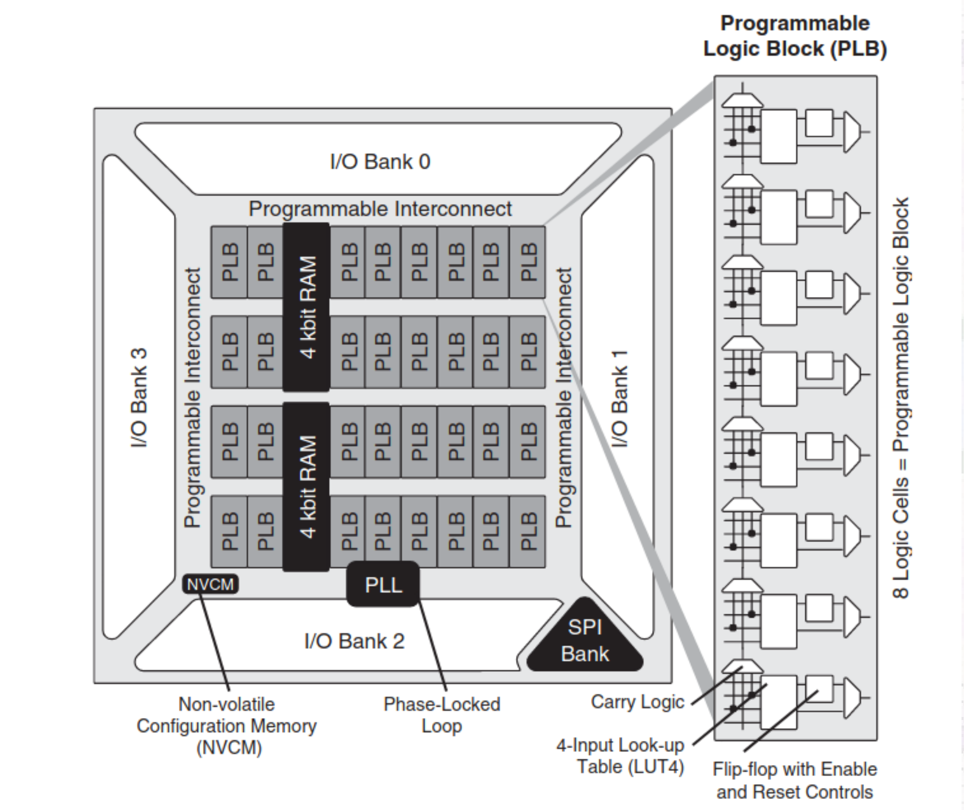
- General Purpose device
- FPGA used in various devices and applications, thus difficult to tamper in silicon
- Allows a user to verify the correct operation of a device by inspecting HDL (Hardware Description Language) code.
- User can customize/hack their devices if desired.

Why use FPGAs? - FPGA Architecture



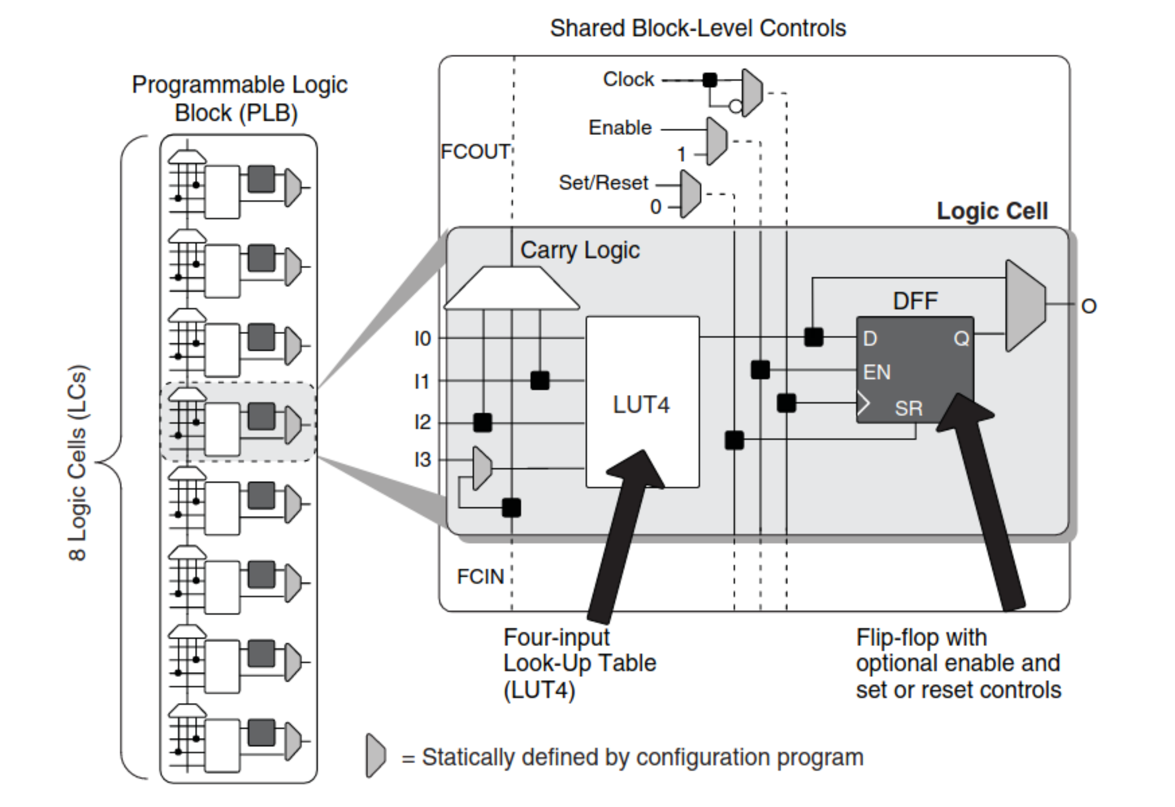
General Structure of a FPGA

Why use FPGAs? - FPGA Architecture


Programmable Logic Block (PLB)
I/O Bank 0
Programmable Interconnect
I/O Bank 3
Programmable Interconnect
PLB
4 kbit RAM
4 kbit RAM
Programmable Interconnect
I/O Bank 1
8 Logic Cells = Programmable Logic Block
NVCM
PLL
I/O Bank 2
SPI Bank
Non-volatile Configuration Memory (NVCM)
Phase-Locked Loop
Carry Logic
4-Input Look-up Table (LUT4)
Flip-flop with Enable and Reset Controls

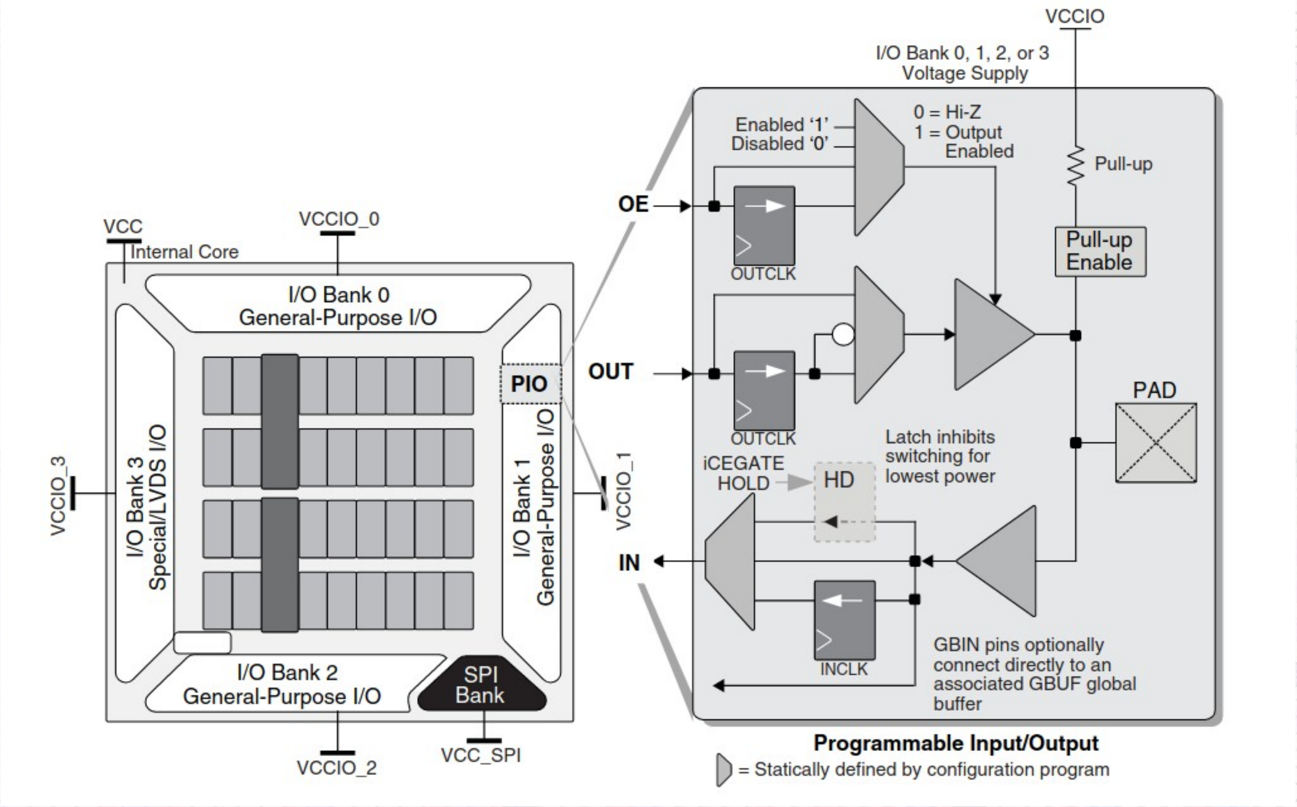
Lattice iCE40 Internal Block Diagram

Why use FPGAs? - FPGA Architecture


Shared Block-Level Controls
Programmable Logic Block (PLB)
8 Logic Cells (LCs)
FCOUT
Clock
Enable
1
Set/Reset
0
Logic Cell
Carry Logic
DFF
D
Q
EN
SR
O
I0
I1
I2
I3
LUT4
FCIN
Four-input Look-Up Table (LUT4)
Flip-flop with optional enable and set or reset controls
= Statically defined by configuration program

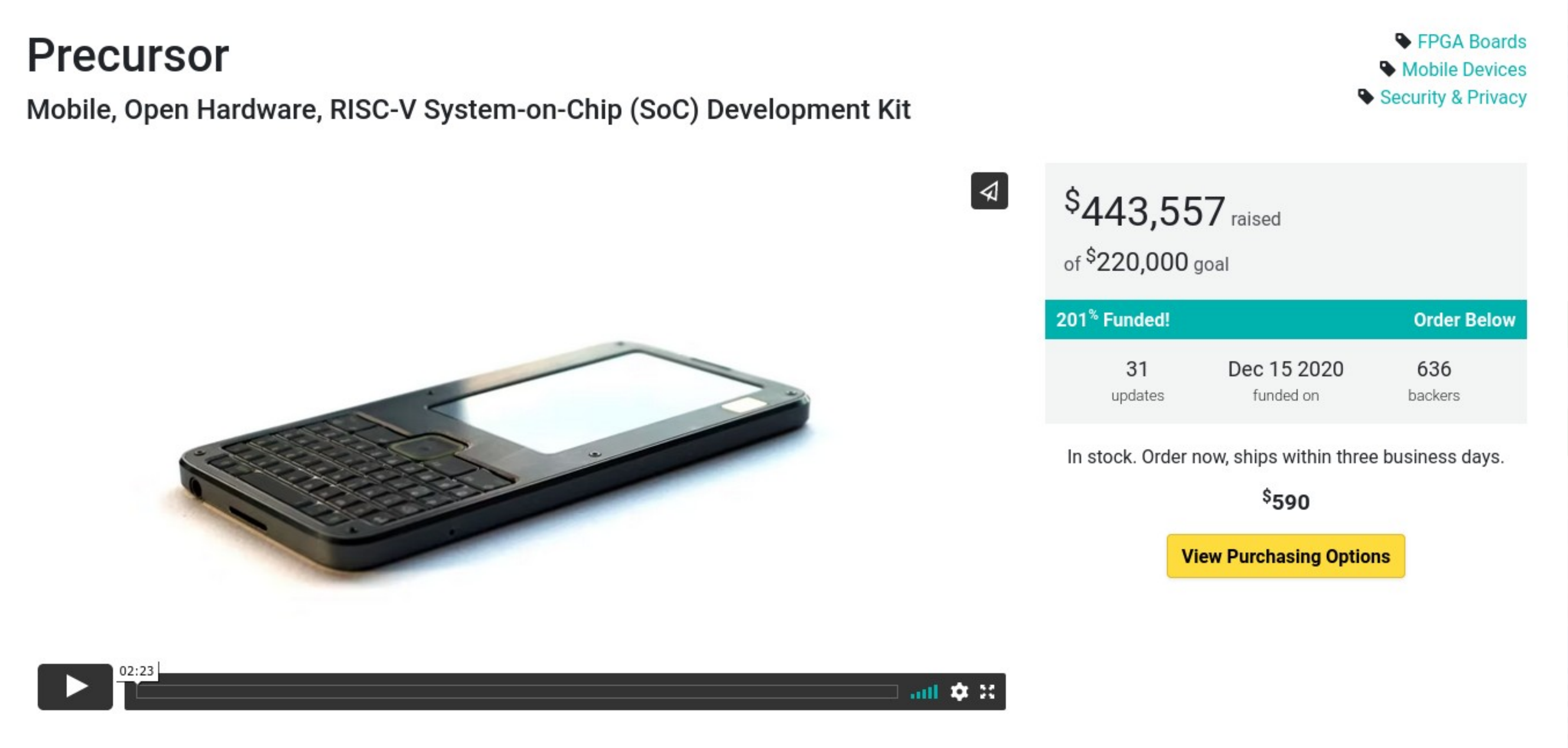
Lattice iCE40 – Programmable Logic Block

Why use FPGAs? - FPGA Architecture



Lattice iCE40 – I/O Bank and I/O Cell

Existing Works



Precursor

Mobile, Open Hardware, RISC-V System-on-Chip (SoC) Development Kit

- FPGA Boards
- Mobile Devices
- Security & Privacy

$443,557 raised

of $220,000 goal

201% Funded! Order Below

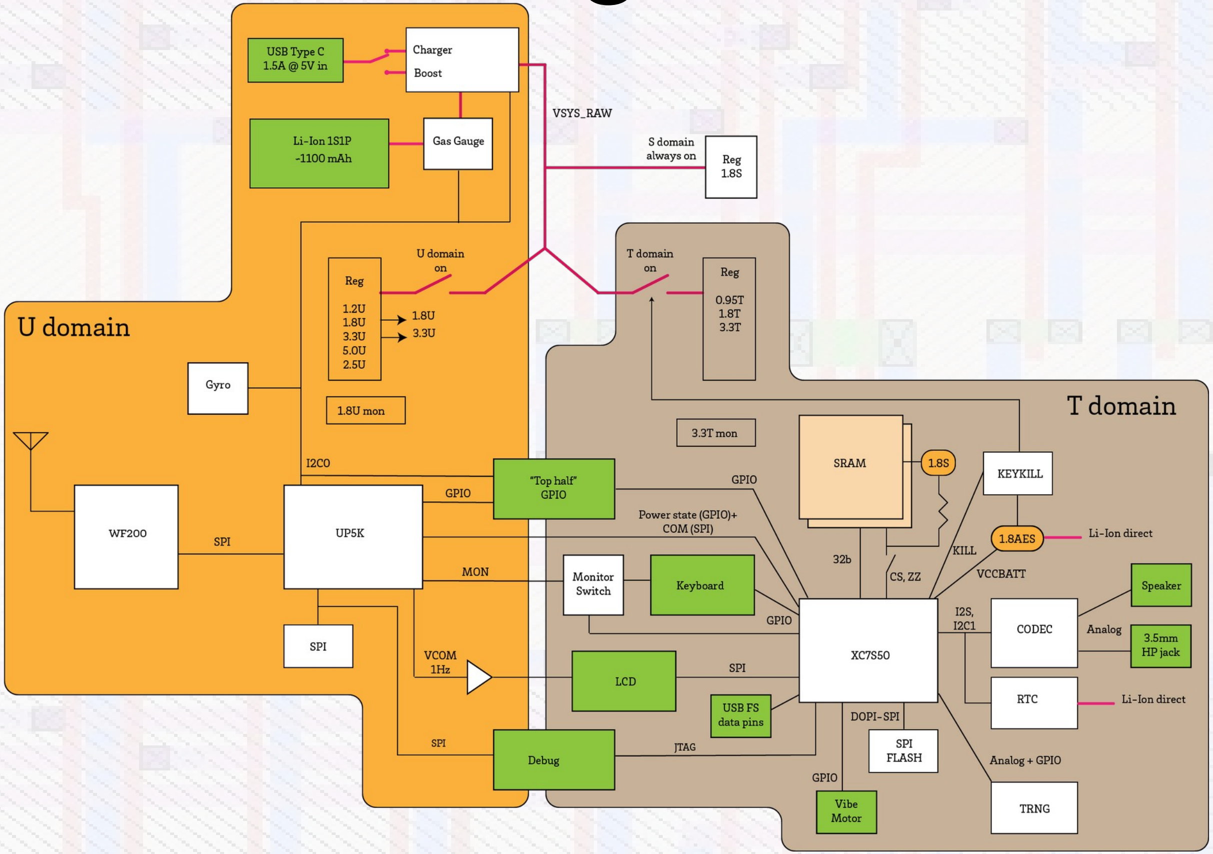
31	Dec 15 2020	636
updates	funded on	backers

In stock. Order now, ships within three business days.

$590

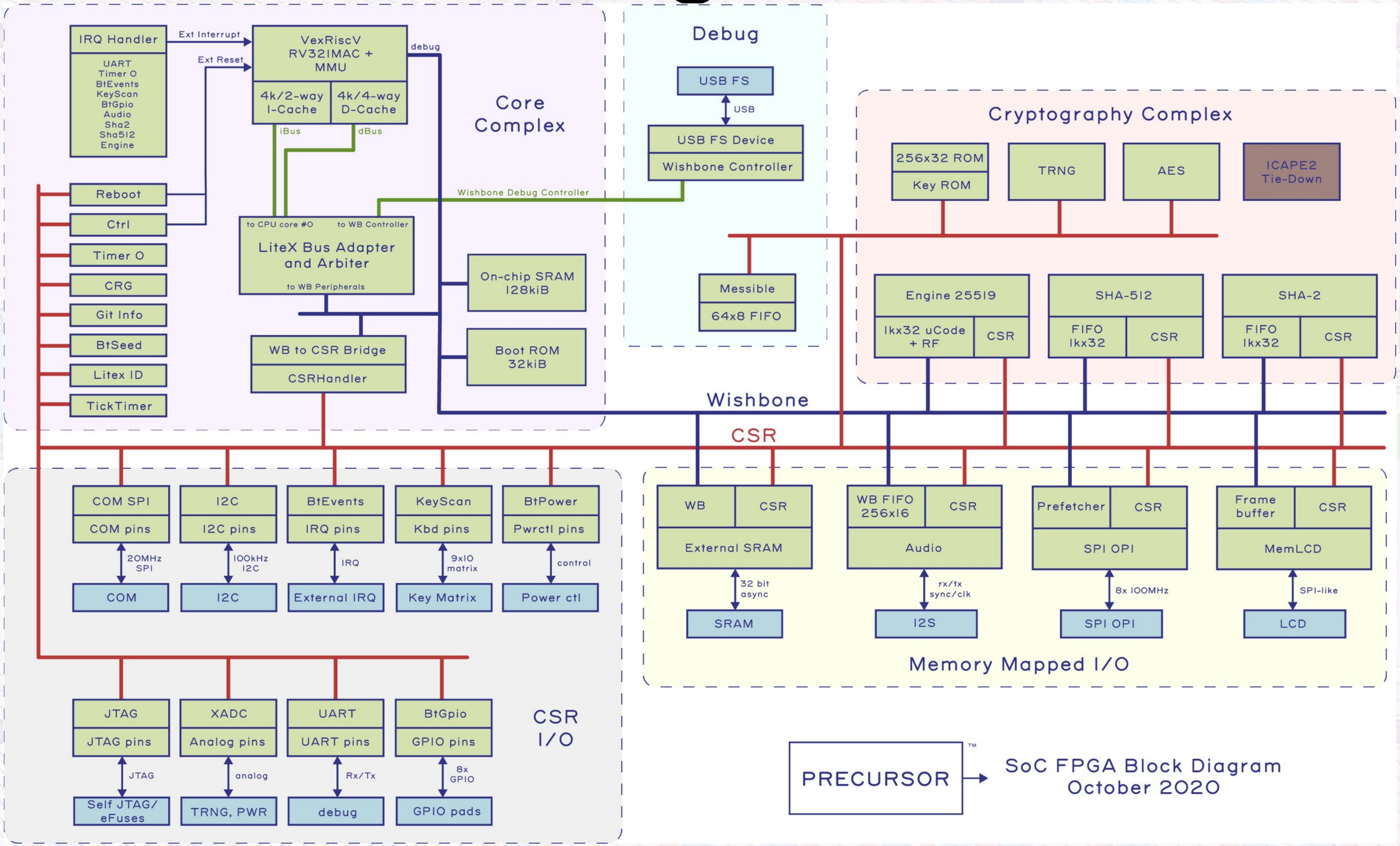
Bunnie Studios Precursor

Existing Works



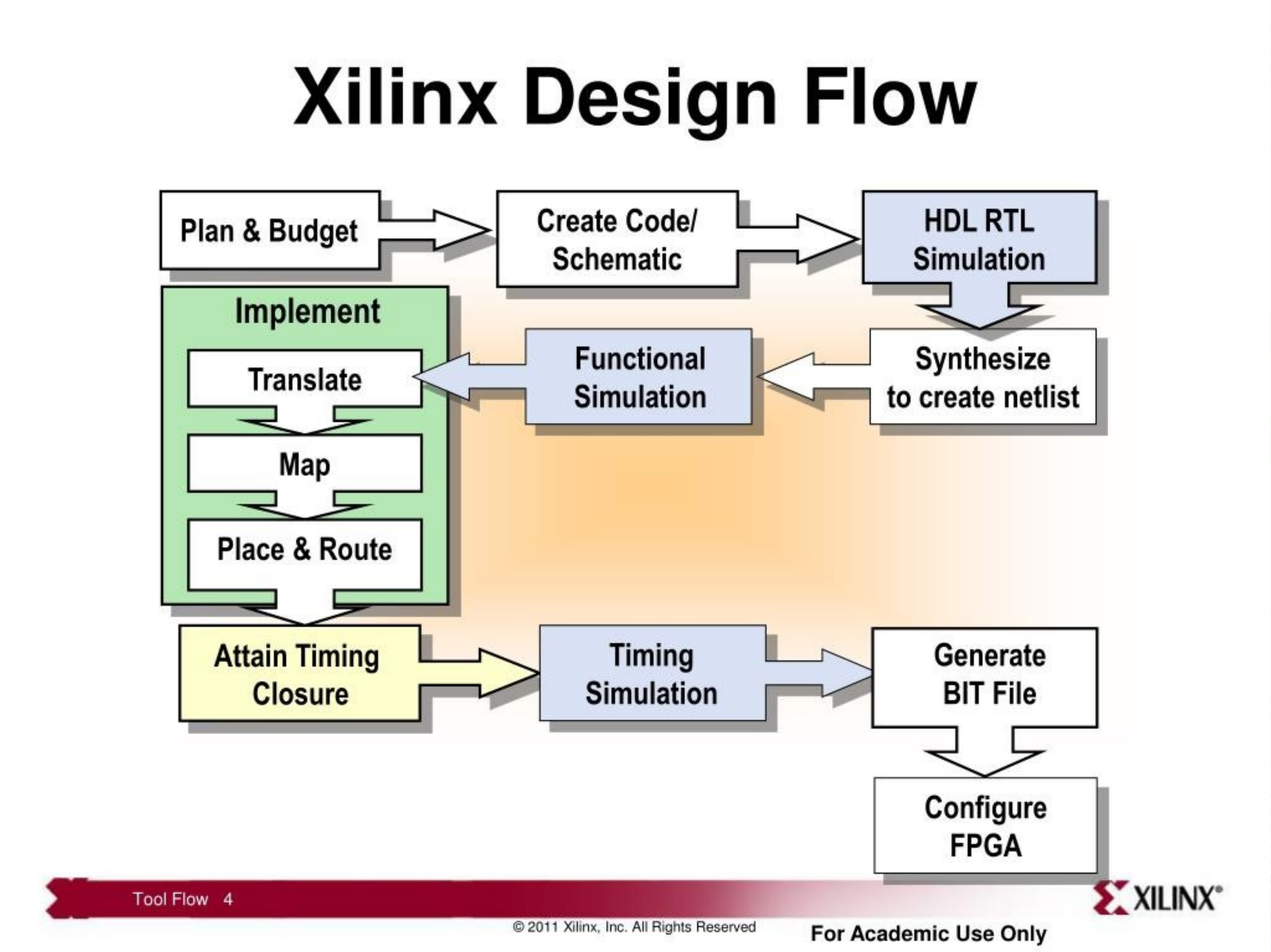
Precursor – Block Diagram (Mainboard)

Existing Works



Precursor – Block Diagram (SoC)

FPGA Design Flow



Xilinx Design Flow

FPGA Design Flow - Processes

- Synthesis:
  Converts HDL code into netlist file describing logical connections between blocks.
- Place and Route (PnR):
  Takes synthesized design and turns it into a physical implementation based on the target FPGA device.
- Bitstream Generation:
  Generates a bitstream file containing instructions for Configurable Logic Blocks (CLB) on target device.

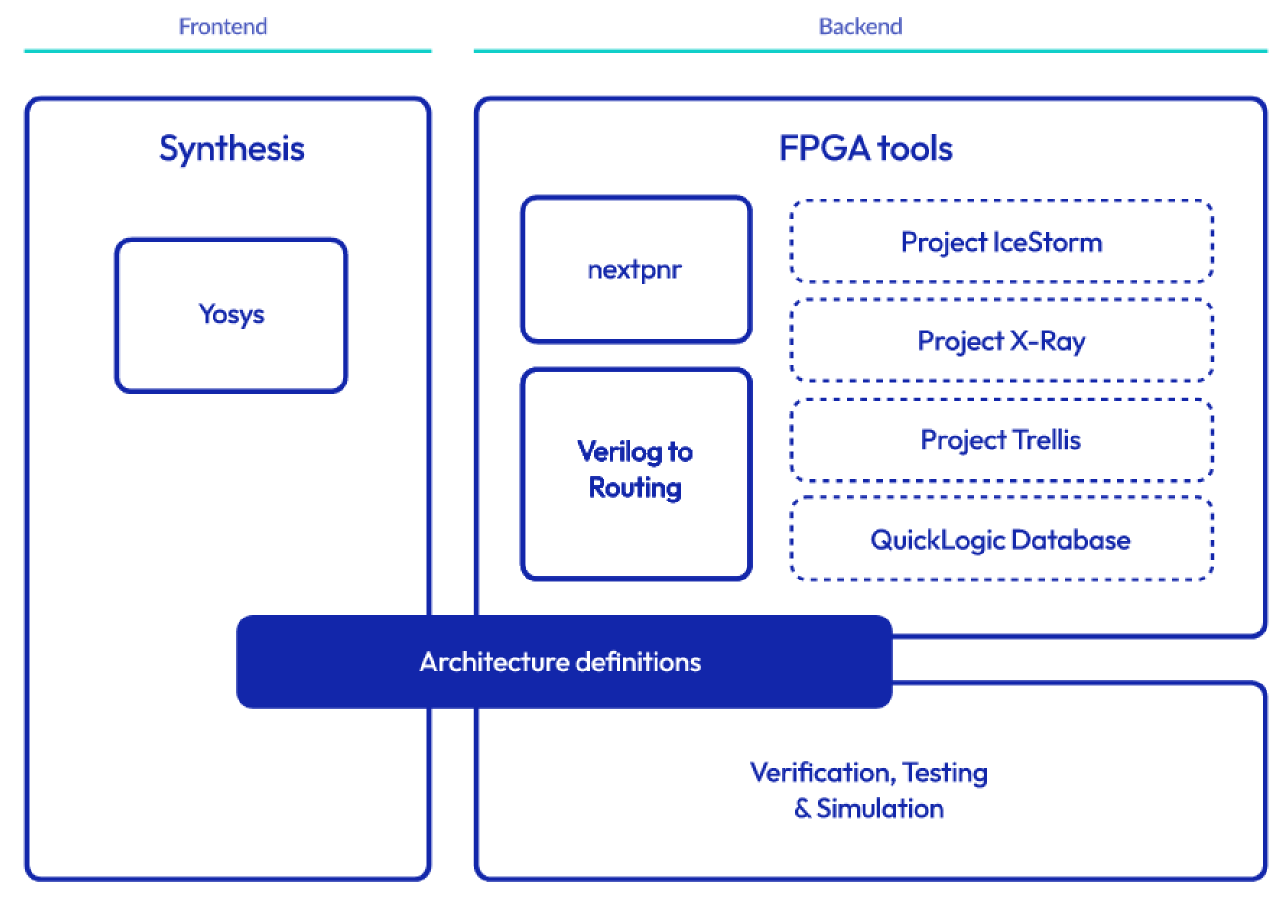
Toolchains – Process Tools

- Synthesis - Yosys
- Place and Route - nextpnr
- Bitstream Generation – various projects under F4PGA

Toolchains - F4PGA

- Aims to be the “GCC” for FPGAs
- Open source FPGA toolchain
- Focused on FPGA architecture definitions
- Supports Xilinx 7-Series, Lattice iCE40 and ECP5, as well as QuickLogic EOS-S3

Toolchains - F4PGA



Frontend

Backend

Synthesis

Yosys

FPGA tools

nextpnr

Verilog to Routing

Project IceStorm

Project X-Ray

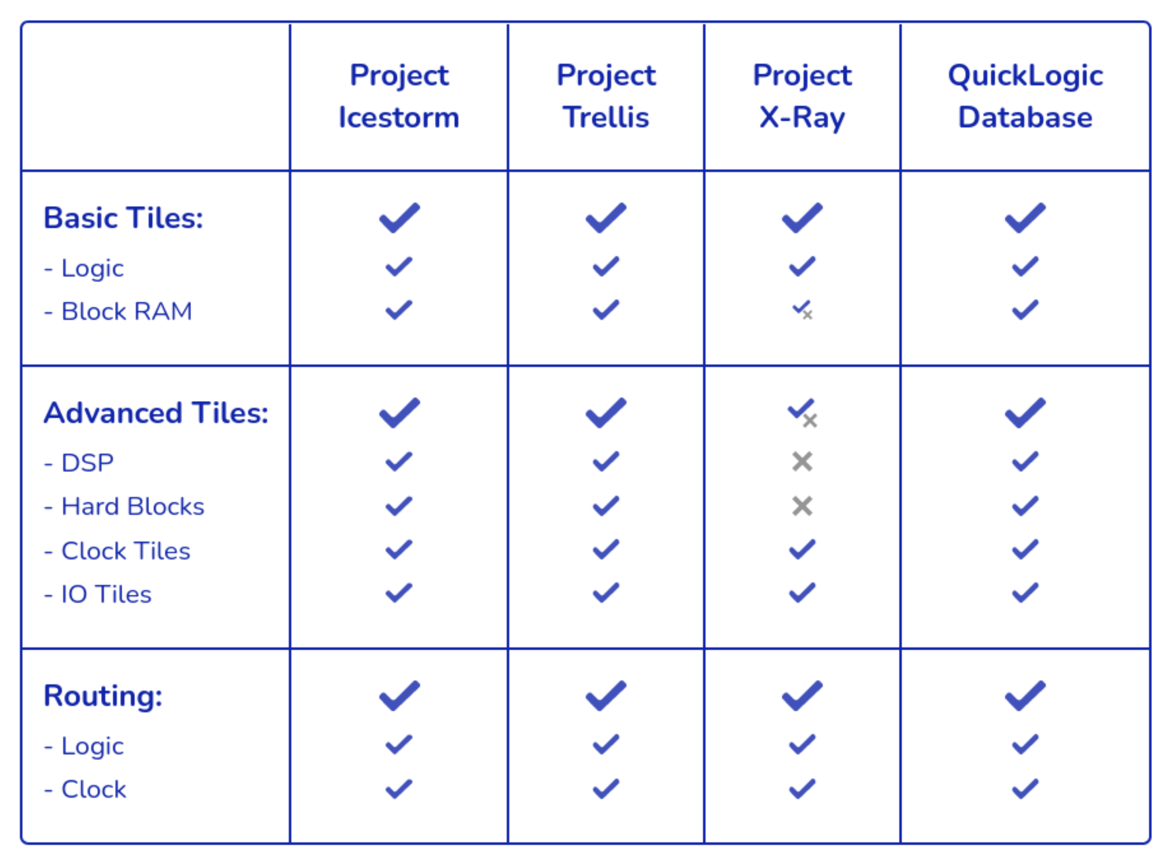
Project Trellis

QuickLogic Database

Architecture definitions

Verification, Testing & Simulation

Toolchains - F4PGA



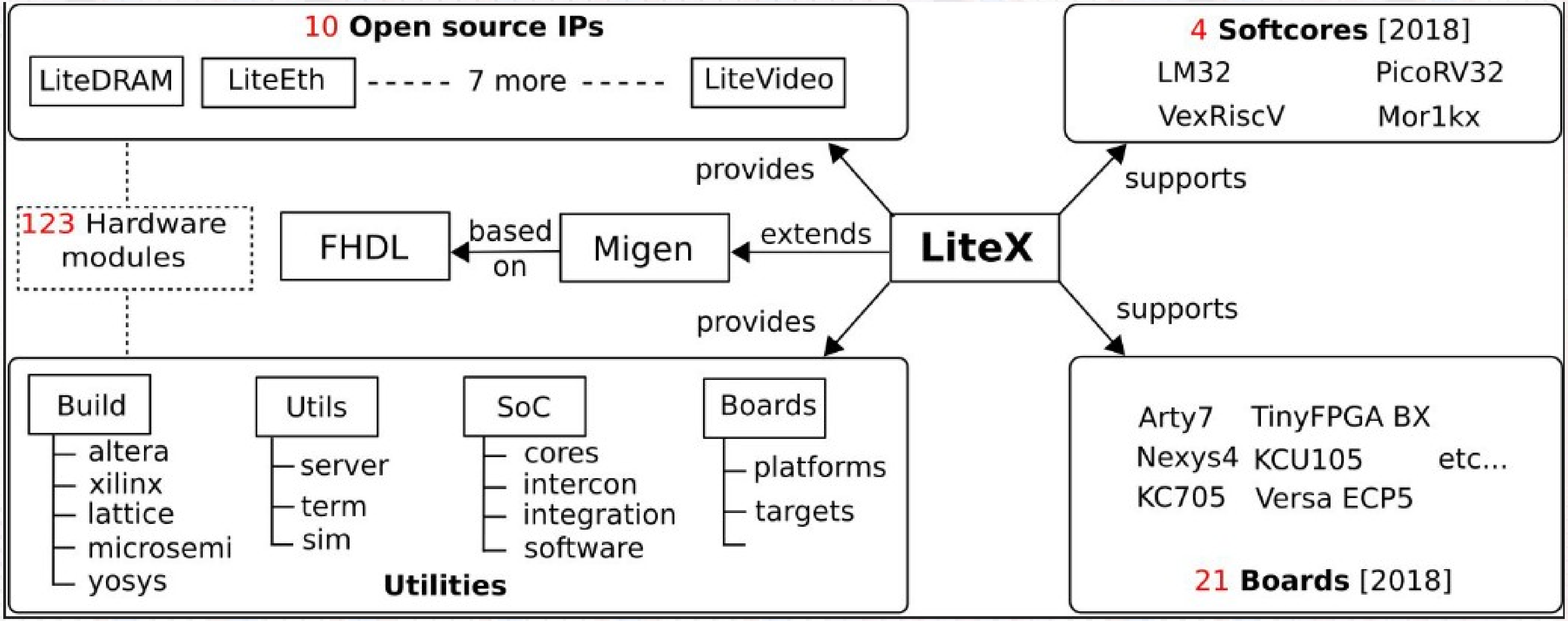
	Project Icestorm	Project Trellis	Project X-Ray	QuickLogic Database
Basic Tiles:	✓	✓	✓	✓
- Logic	✓	✓	✓	✓
- Block RAM	✓	✓	✓✗	✓
Advanced Tiles:	✓	✓	✓✗	✓
- DSP	✓	✓	✗	✓
- Hard Blocks	✓	✓	✗	✓
- Clock Tiles	✓	✓	✓	✓
- IO Tiles	✓	✓	✓	✓
Routing:	✓	✓	✓	✓
- Logic	✓	✓	✓	✓
- Clock	✓	✓	✓	✓

F4PGA Device Support Status

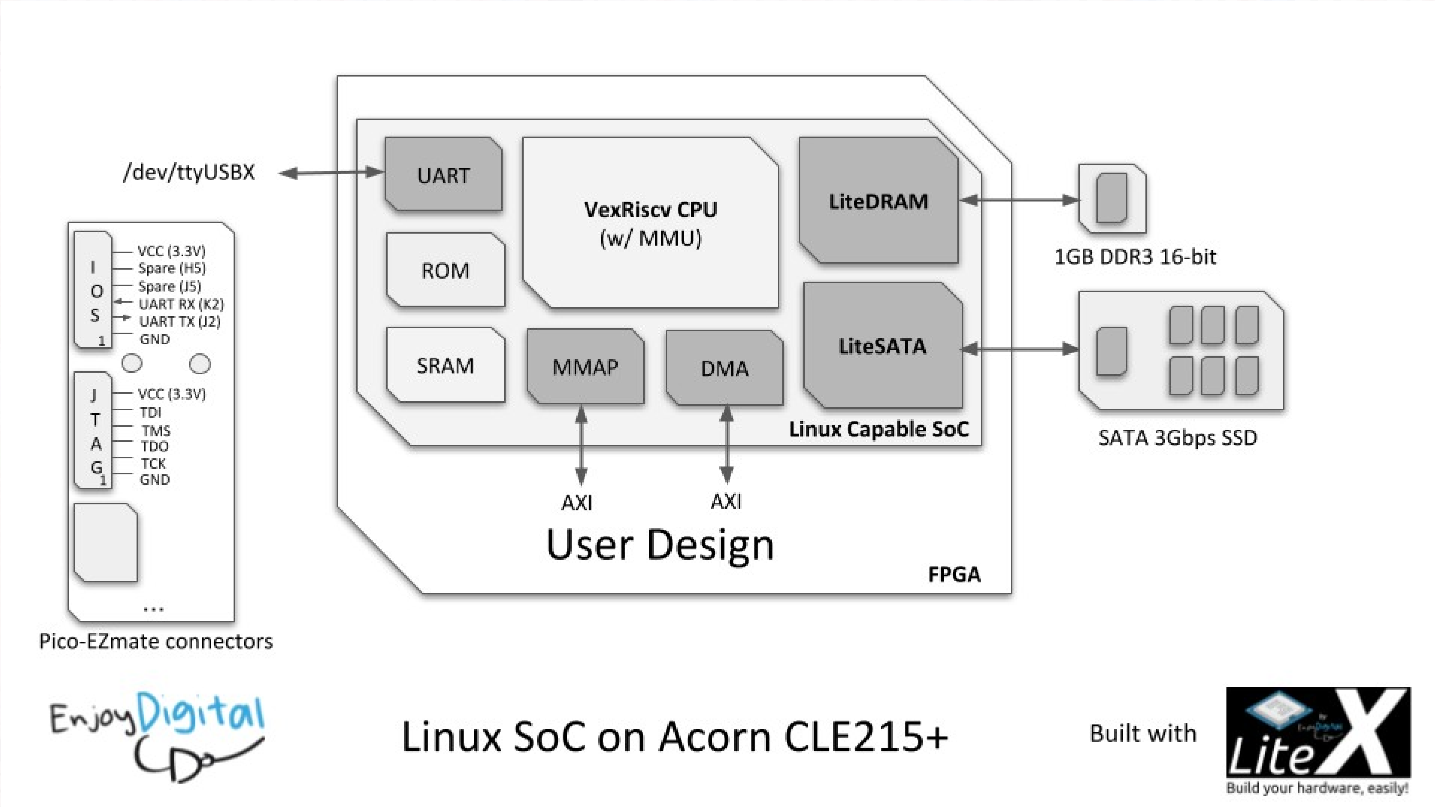
Toolchains - LiteX

- Framework for building FPGA SoCs
- Supports building various RISC-V and other soft core CPUs
- Uses Migen, a Python based HDL
- Easily customize your own SoC design with various peripheral cores (serial, PCIe, Ethernet, etc)

Toolchains - LiteX

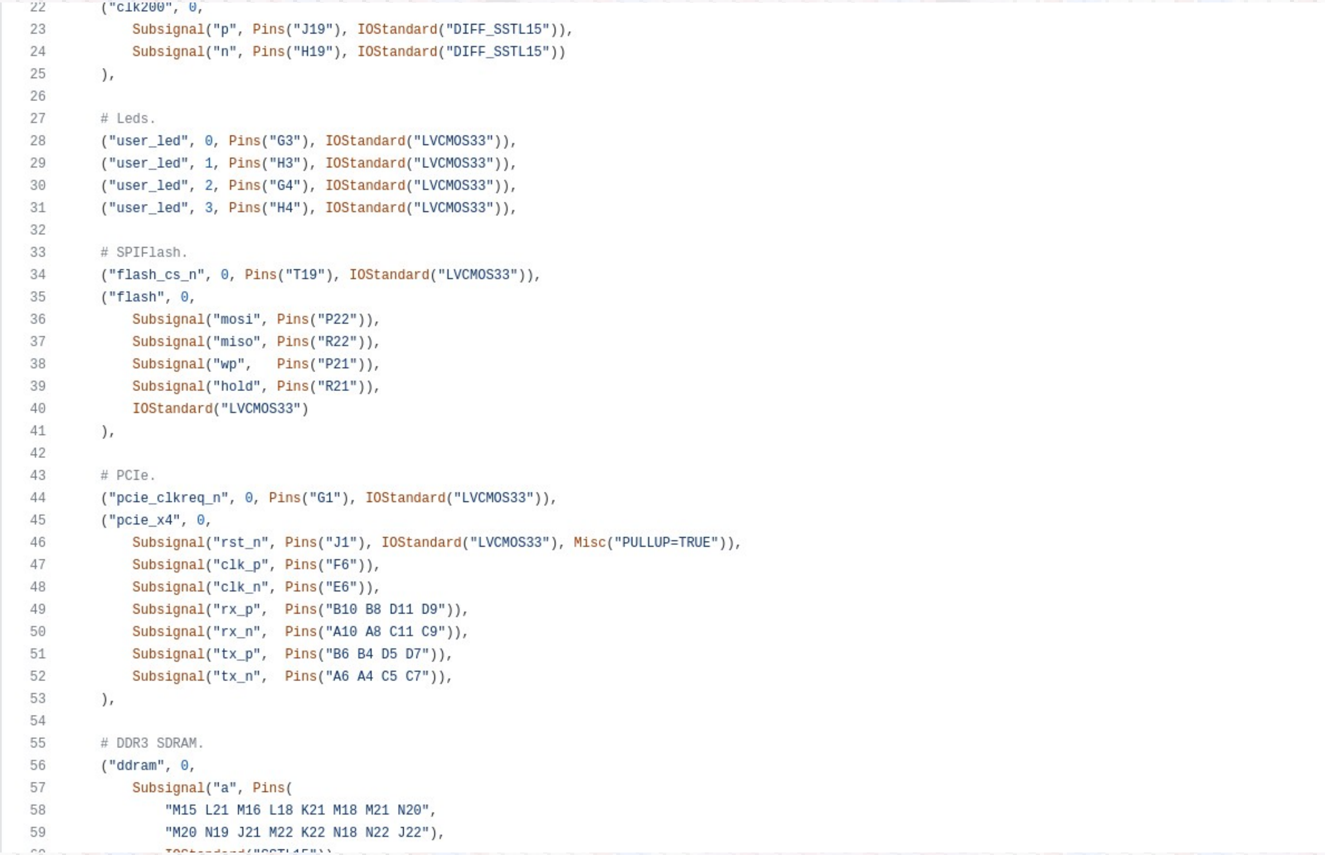


Toolchains - LiteX



Example SoC design on Acorn CLE-215+ board

Toolchains - LiteX



```
    ("clk200", 0,
        Subsignal("p", Pins("J19"), IOStandard("DIFF_SSTL15")),
        Subsignal("n", Pins("H19"), IOStandard("DIFF_SSTL15"))
    ),

    # Leds.
    ("user_led", 0, Pins("G3"), IOStandard("LVCMOS33")),
    ("user_led", 1, Pins("H3"), IOStandard("LVCMOS33")),
    ("user_led", 2, Pins("G4"), IOStandard("LVCMOS33")),
    ("user_led", 3, Pins("H4"), IOStandard("LVCMOS33")),

    # SPIFlash.
    ("flash_cs_n", 0, Pins("T19"), IOStandard("LVCMOS33")),
    ("flash", 0,
        Subsignal("mosi", Pins("P22")),
        Subsignal("miso", Pins("R22")),
        Subsignal("wp",   Pins("P21")),
        Subsignal("hold", Pins("R21")),
        IOStandard("LVCMOS33")
    ),

    # PCIe.
    ("pcie_clkreq_n", 0, Pins("G1"), IOStandard("LVCMOS33")),
    ("pcie_x4", 0,
        Subsignal("rst_n", Pins("J1"), IOStandard("LVCMOS33"), Misc("PULLUP=TRUE")),
        Subsignal("clk_p", Pins("F6")),
        Subsignal("clk_n", Pins("E6")),
        Subsignal("rx_p",  Pins("B10 B8 D11 D9")),
        Subsignal("rx_n",  Pins("A10 A8 C11 C9")),
        Subsignal("tx_p",  Pins("B6 B4 D5 D7")),
        Subsignal("tx_n",  Pins("A6 A4 C5 C7")),
    ),

    # DDR3 SDRAM.
    ("ddram", 0,
        Subsignal("a", Pins(
            "M15 L21 M16 L18 K21 M18 M21 N20",
            "M20 N19 J21 M22 K22 N18 N22 J22"),
            IOStandard("SSTL15")),
```

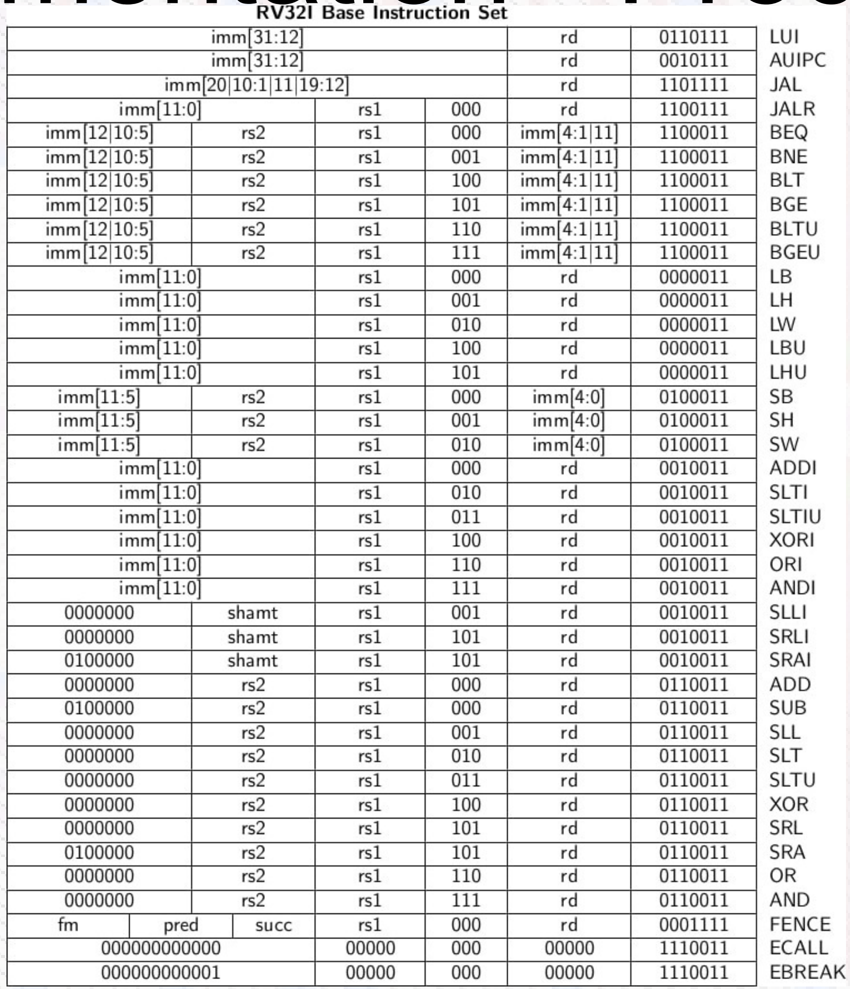
Acorn CLE-215+ board definition

Implementation - Processor

- Why RISC-V?

- Open Source ISA
- <50 instructions for RV32I
- Frozen Base Instructions + Standard Instructions
- Custom extensions possible

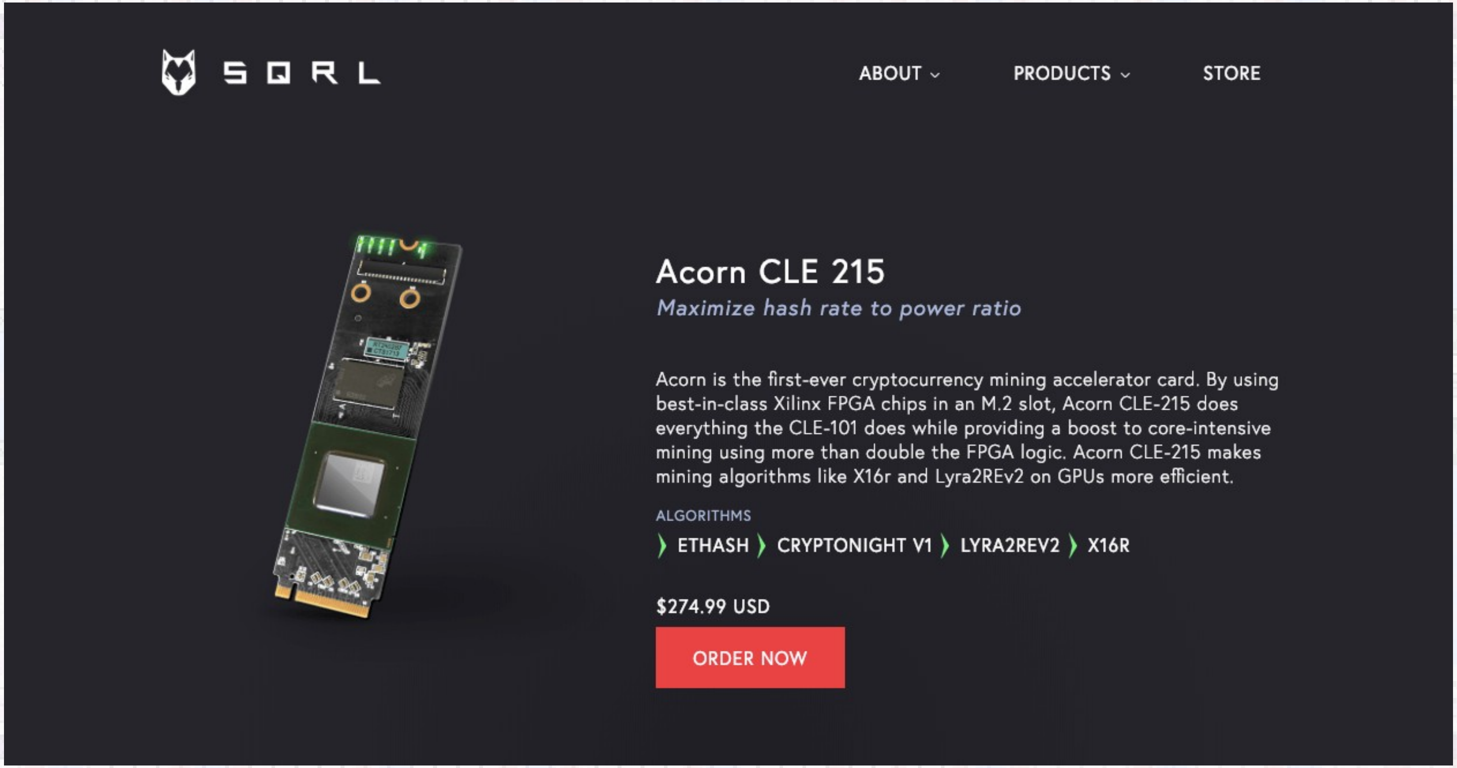
Implementation - Processor



RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm pred	succ	rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

Implementation



Acorn CLE 215

Maximize hash rate to power ratio

Acorn is the first-ever cryptocurrency mining accelerator card. By using best-in-class Xilinx FPGA chips in an M.2 slot, Acorn CLE-215 does everything the CLE-101 does while providing a boost to core-intensive mining using more than double the FPGA logic. Acorn CLE-215 makes mining algorithms like X16r and Lyra2REv2 on GPUs more efficient.

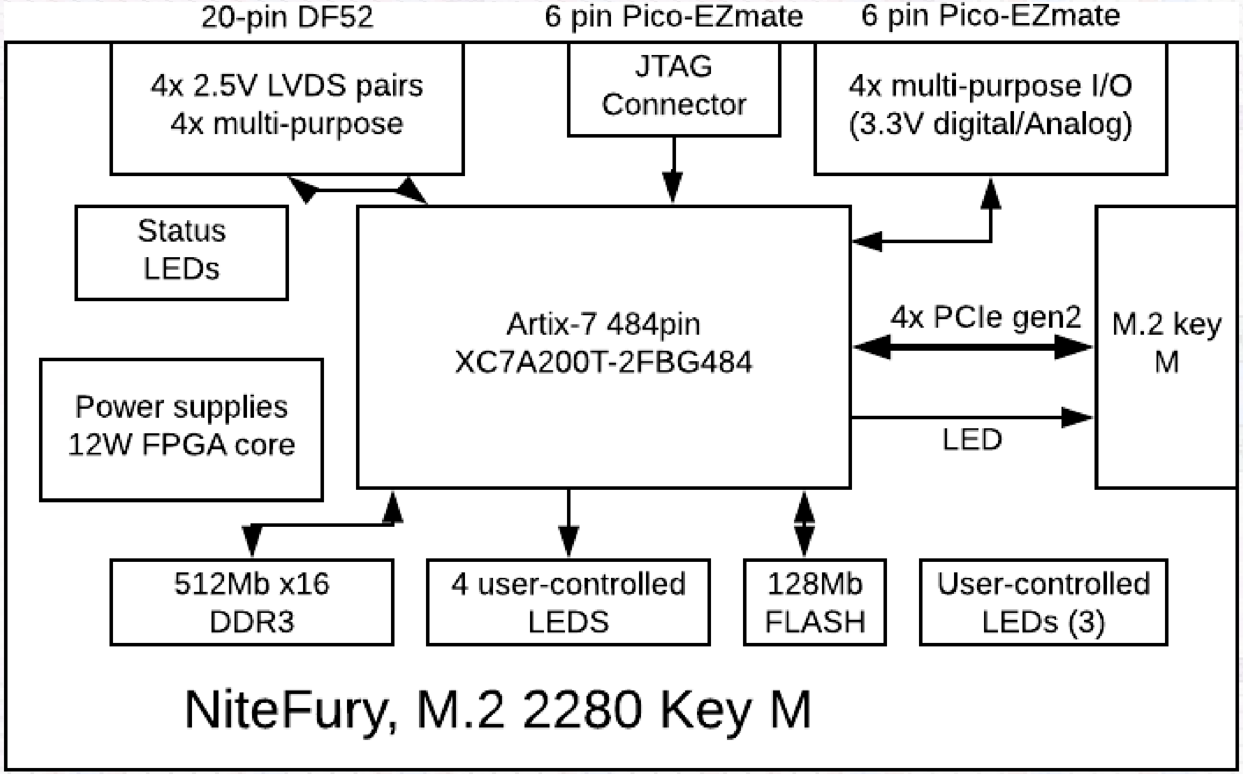
ALGORITHMS

〉 ETHASH 〉 CRYPTONIGHT V1 〉 LYRA2REV2 〉 X16R

$274.99 USD

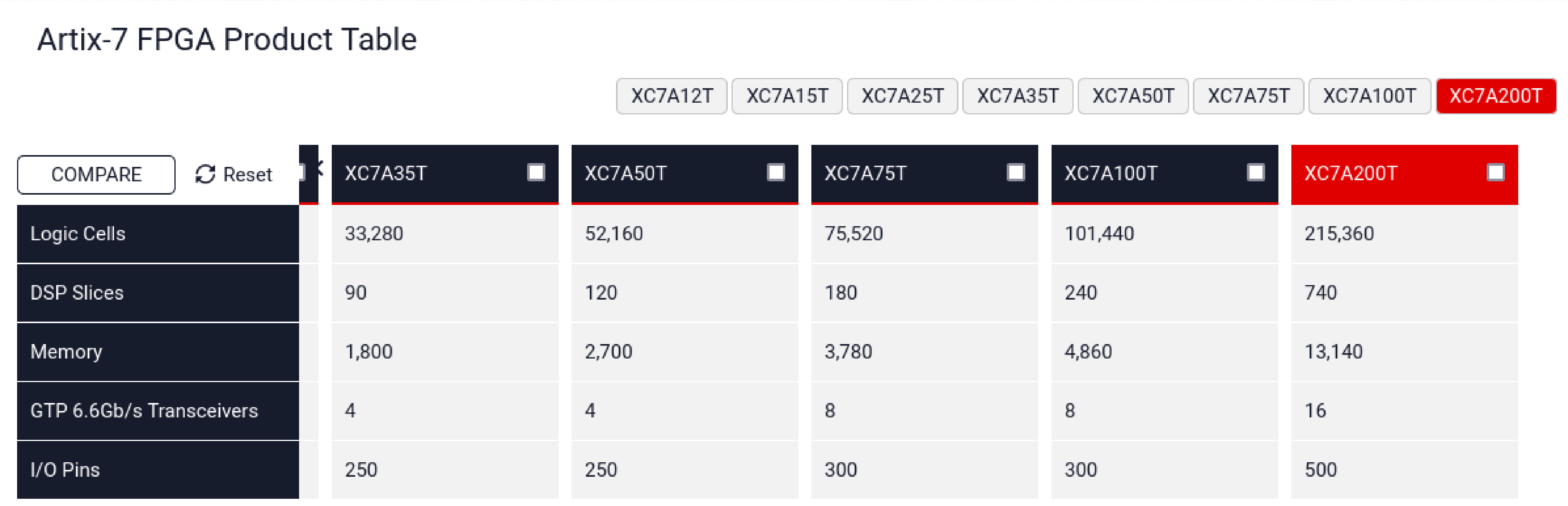
Acorn SQRL CLE-215+

Implementation



Acorn SQRL CLE-215+ (Block Diagram)

Implementation



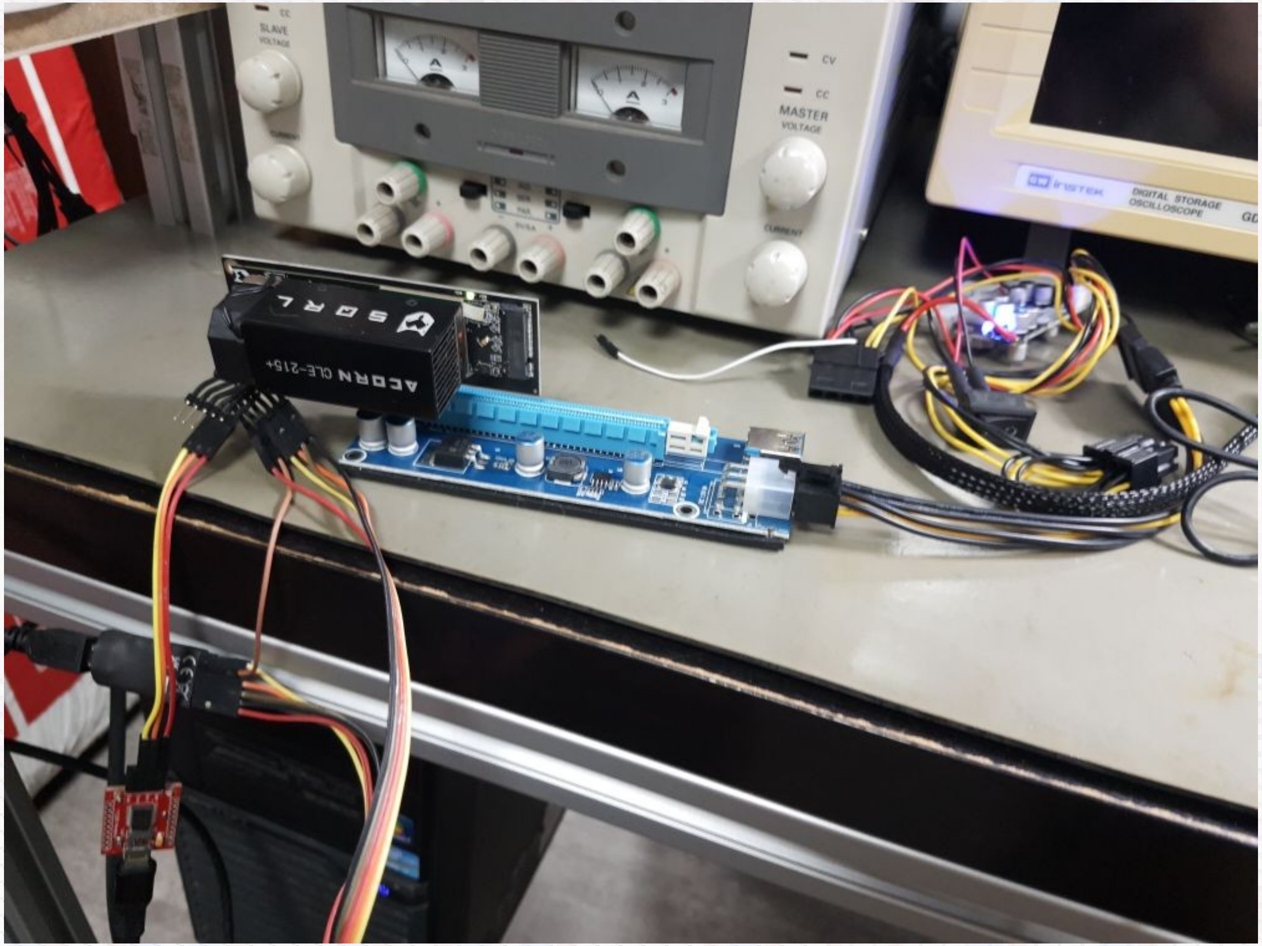
Artix-7 FPGA Product Table

XC7A12T | XC7A15T | XC7A25T | XC7A35T | XC7A50T | XC7A75T | XC7A100T | XC7A200T

COMPARE ⟳ Reset	XC7A35T □	XC7A50T □	XC7A75T □	XC7A100T □	XC7A200T □
Logic Cells	33,280	52,160	75,520	101,440	215,360
DSP Slices	90	120	180	240	740
Memory	1,800	2,700	3,780	4,860	13,140
GTP 6.6Gb/s Transceivers	4	4	8	8	16
I/O Pins	250	250	300	300	500

Xilinx Artix-7 Family

Implementation - Setup

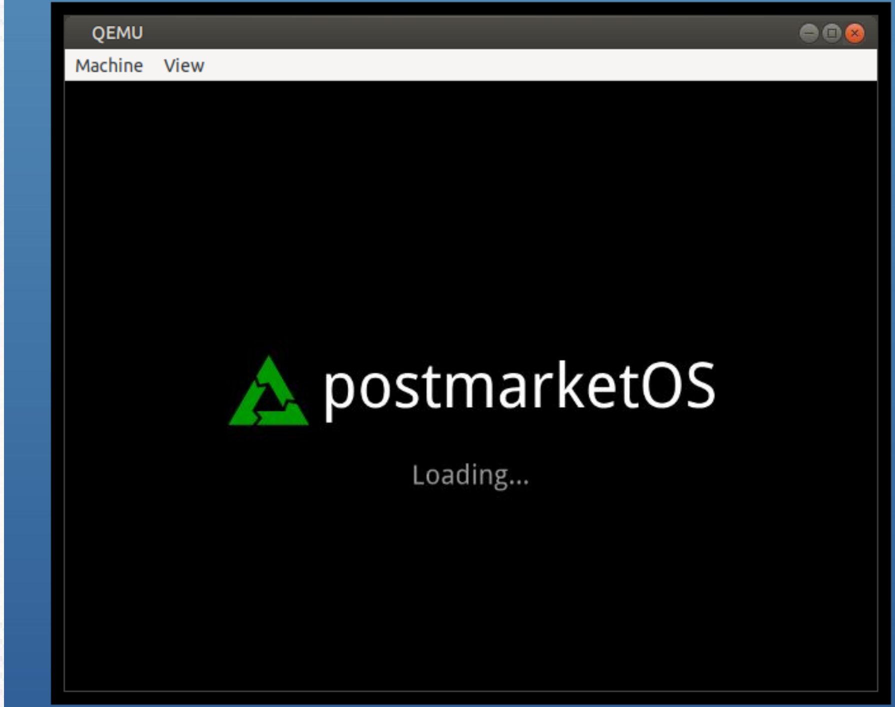


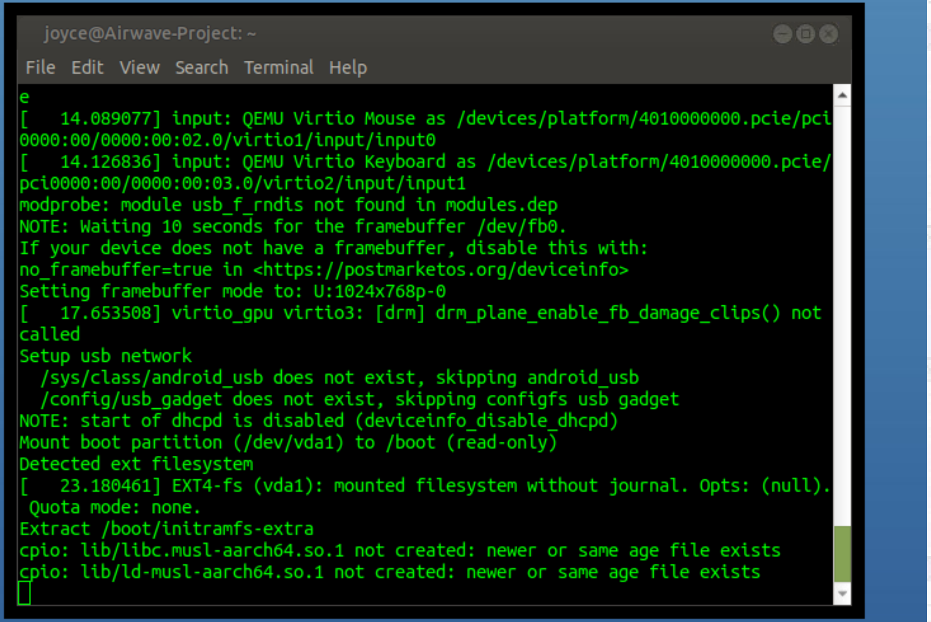
Demo

Future Projects

- RISC-V Laptop:
  Verifiable laptop design using RISC-V on FPGA for processor.
- Airwave Project:
  Investigation into cellular device security; with focus on cellular communications. Possible phone implementation based on Software Defined Radio (SDR) for baseband?

Future Projects





```
e
[   14.089077] input: QEMU Virtio Mouse as /devices/platform/4010000000.pcie/pci
0000:00/0000:00:02.0/virtio1/input/input0
[   14.126836] input: QEMU Virtio Keyboard as /devices/platform/4010000000.pcie/
pci0000:00/0000:00:03.0/virtio2/input/input1
modprobe: module usb_f_rndis not found in modules.dep
NOTE: Waiting 10 seconds for the framebuffer /dev/fb0.
If your device does not have a framebuffer, disable this with:
no_framebuffer=true in <https://postmarketos.org/deviceinfo>
Setting framebuffer mode to: U:1024x768p-0
[   17.653508] virtio_gpu virtio3: [drm] drm_plane_enable_fb_damage_clips() not
called
Setup usb network
  /sys/class/android_usb does not exist, skipping android_usb
  /config/usb_gadget does not exist, skipping configfs usb gadget
NOTE: start of dhcpd is disabled (deviceinfo_disable_dhcpd)
Mount boot partition (/dev/vda1) to /boot (read-only)
Detected ext filesystem
[   23.180461] EXT4-fs (vda1): mounted filesystem without journal. Opts: (null).
 Quota mode: none.
Extract /boot/initramfs-extra
cpio: lib/libc.musl-aarch64.so.1 not created: newer or same age file exists
cpio: lib/ld-musl-aarch64.so.1 not created: newer or same age file exists
```

Messing with postmarketOS for Airwave Project

References and Resources

- Precursor Wiki: https://github.com/betrusted-io/betrusted-wiki/wiki
- F4PGA: https://f4pga.org/
- LiteX: https://github.com/enjoy-digital/litex
- Bootstrapping a Libre, Self-Hosting RISC-V Computer: https://open-src-soc.org/2021-03/media/slides/3rd-RISC-V-Meeting-2021-03-31-13h30-Gabriel-Somlo.pdf

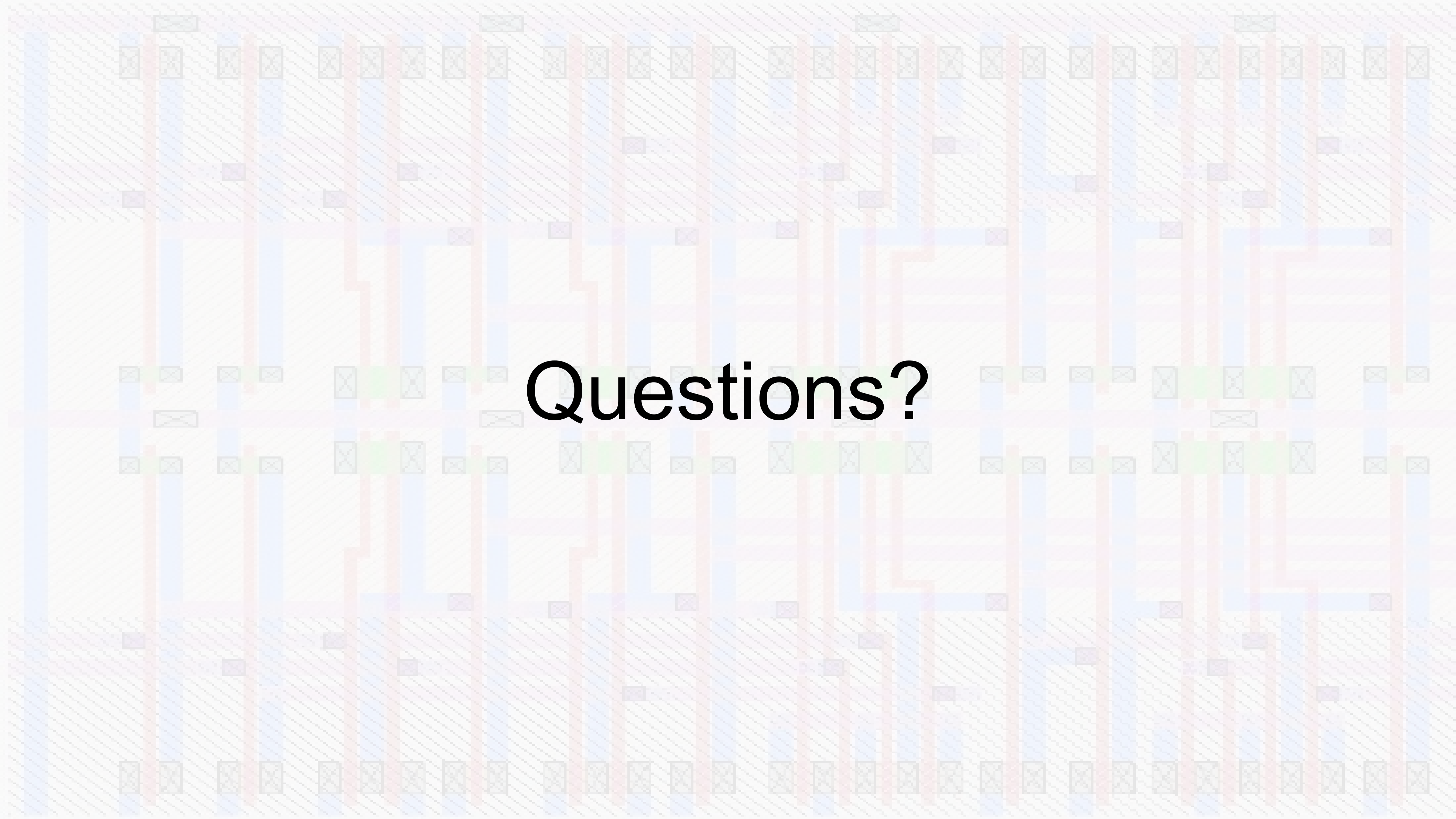
References and Resources

- Stealthy Dopant-Level Hardware Trojans: https://sharps.org/wp-content/uploads/BECKER-CHES.pdf

Where to find me:

- Twitter: @quantumcatgirl
  Mastodon: @sleepyowl@chaos.social

  For direct contact:
  Email: joyce_ng@hyantechnologies.com
  Discord: Schrödinger's Catgirl (Joyce)#3724



Questions?