Python-based Open Source Toolchain for RISC-V Development

Feng Li (李枫)
hkli2013@126.com
Jun 23，2021

Agenda

I. Overview

- Open Source EDA
- Evolution of HDL
- Testbed

II. Core/SoC Builder

- LiteX

III. Python-based FOSS FPGA Toolchain

- Open FPGA
- SymbiFlow
- Cocotb

IV. Practicing

- Cross Development of RISC-V on ARM

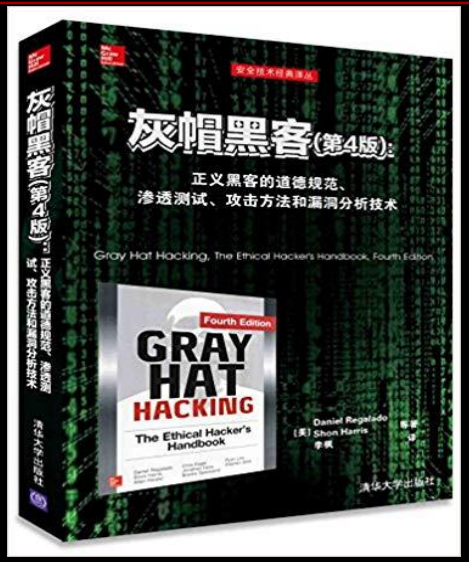
V. Python Runtimes

- Miscellaneous Python Implementations
- Benchmarking

VI. Wrap-up

Who Am I

- The main translator of the book «Gray Hat Hacking The Ethical Hacker's Handbook, Fourth Edition» (ISBN：9787302428671) & «Linux Hardening in Hostile Networks, First Edition» (ISBN: 9787115544384)



- Pure software developmement for ~15 years
- Actively participate in various activities of the open source community
- https://github.com/XianBeiTuoBaFeng2015/MySlides/tree/master/Conf
  https://github.com/XianBeiTuoBaFeng2015/MySlides/tree/master/LTS
- Recently, focus on infrastructure of Cloud/Edge Computing, AI, Virtualization, Program Runtimes, Network, 5G, RISC-V, EDA…

I. Overview

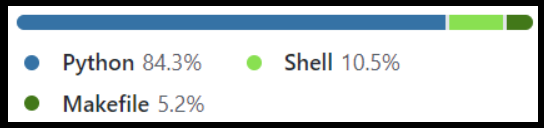
1) Open Source EDA

- https://en.wikipedia.org/wiki/Comparison_of_EDA_software
- https://semiwiki.com/wikis/industry-wikis/eda-open-source-tools-wiki/
- https://ieeexplore.ieee.org/document/9398963
- https://ieeexplore.ieee.org/document/9398960
- https://ieeexplore.ieee.org/document/9336682
- https://ieeexplore.ieee.org/document/9105619
- …

1.1 SkyWater Open Source PDK

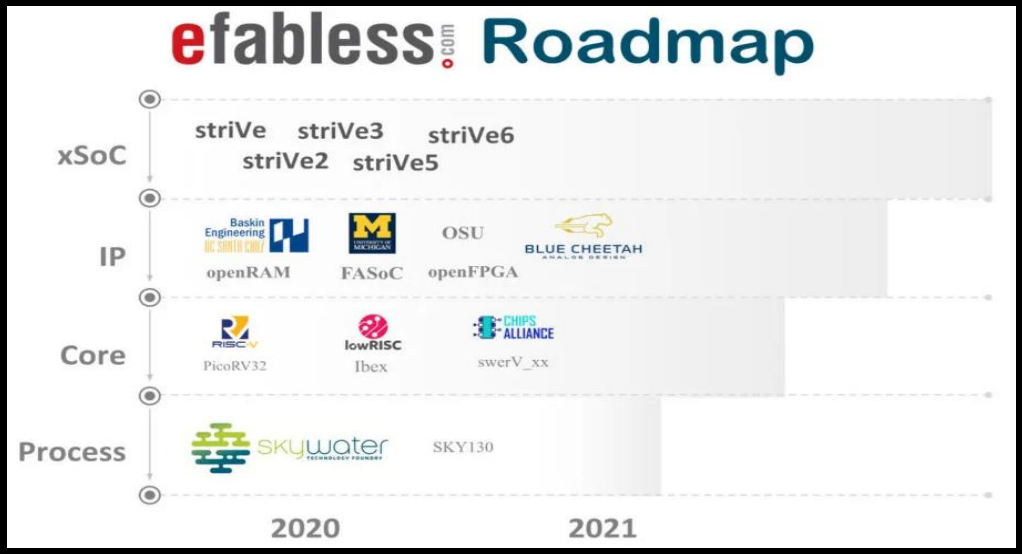
- https://skywater-pdk.rtfd.io/
- Open source process design kit for usage with SkyWater Technology Foundry's 130nm node.

- https://github.com/google/skywater-pdk



- https://en.wikipedia.org/wiki/Process_design_kit

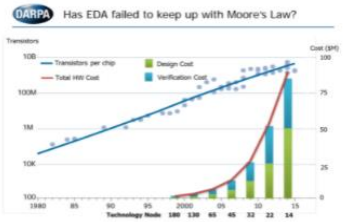
1.2 efabless

- https://efabless.com/
- https://github.com/efabless
- Features:
  - Complete and validated design flow based on open-source tools
  - Foundry process and PDK
  - Prototyping services
  - Packaging options
  - Marketplace library of IP and reference design
  - Evaluate IP from the marketplace prior purchasing
- 

1.3 OpenROAD

- https://theopenroadproject.org/
- https://github.com/The-OpenROAD-Project
-

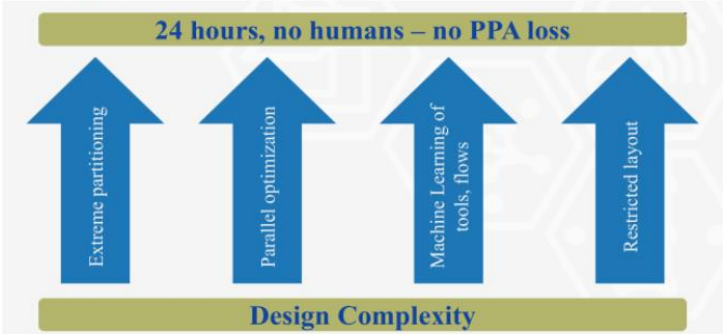
The Problem



Our Goal

- 24-hour, No-Human-In-The-Loop layout design for SOC, Package and PCB with no Power-Performance-Area (PPA) loss
- Tapeout-capable tools in source code form, with permissive licensing → seed future "Linux of EDA"

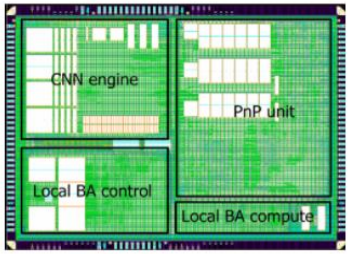
Our Approach

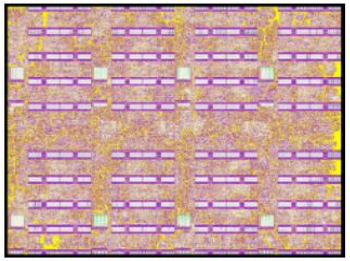


- No Humans: tools must adapt and self-tune, must never get stuck unexpectedly
- 24 hours: extreme partitioning of problems
  - parallel search on cloud
  - machine learning for predictability
- Mantra: Correctness and safety by construction
- Mantra: Embrace freedom from choice
- Mantra: Often, only one thread needs to succeed

Impact

- Create new "Base Technologies" that enable 24-hour, autonomous design
  - Extreme partitioning (bite-sized problems)
  - Parallel search and optimization
  - Machine learning: models of tools, designs
- New paradigm for design tools and methods: autonomy first
- Bring down barriers to democratize HW design
- Embedded vision chips (28nm/16nm) from Michigan Internal Design Advisors team
  - Layout @Michigan: 10+ weeks, significant resource
- OpenROAD and IDEA goal: 1 day, no humans (!)





2) Evolution of HDL

- Verilog/VHDL ⟶ HLS ⟶ eDSL

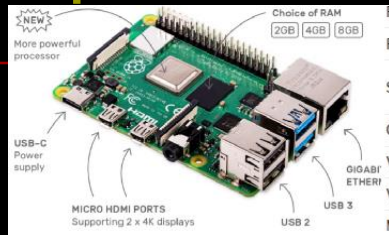
Typical eDSLs

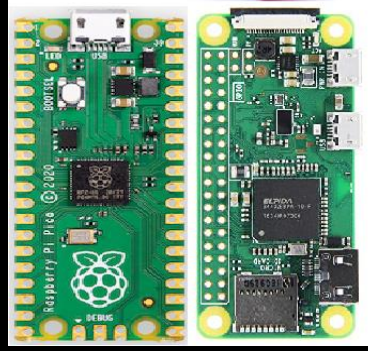
- Haskell as host
  Bluespec...
- Scala as host
  Chisel, SpinalHDL...
- Python as host
  FHDL...
- ...

3) Testbed

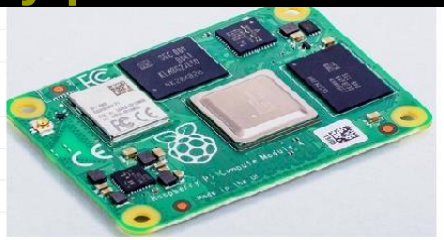
3.1 Raspberry Pi

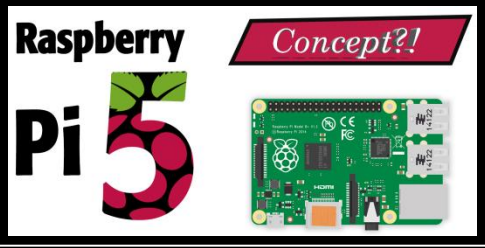
- https://www.raspberrypi.org/products/raspberry-pi-4-model-b/





Features/Specs	Raspberry Pi 4B	Raspberry Pi 3 B+
Release date	24th June 2019	14th March 2018
SoC	Broadcom BCM2711 quad-core Cortex-A72 @ 1.5 GHz	Broadcom BCM2837B0 quad-core Cortex-A53 @ 1.4 GHz
GPU	VideoCore VI with OpenGL ES 1.1, 2.0, 3.0	VideoCore IV with OpenGL ES 1.1, 2.0
Video Decode	H.265 4Kp60, H.264 1080p60	H.264 & MPEG-4 1080p30
Video Encode	H.264 1080p30	
Memory	1GB, 2GB, or 4GB LPDDR4	1GB LPDDR2
Storage	microSD card	
Video & Audio Output	2x micro HDMI ports up to 4Kp60 3.5mm AV port (composite + audio) MIPI DSI connector	1x HDMI 1.4 port up to 1080p60 3.5mm AV port (composite + audio) MIPI DSI connector
Camera	MIPI CSI connector	
Ethernet	Native Gigabit Ethernet	Gigabit Ethernet over USB (300 Mbps max.)
WiFi	Dual band 802.11 b/g/n/ac	
Bluetooth	Bluetooth 5.0 + BLE	Bluetooth 4.2 + BLE
USB	2x USB 3.0 + 2x USB 2.0	4x USB 2.0
Expansion	40-pin GPIO header	
Power Supply	5V via USB type-C up to 3A 5V via GPIO header up to 3A Power over Ethernet via PoE HAT	5V via micro USB up to 2.5A 5V via GPIO header up to 3A Power over Ethernet via PoE HAT
Dimensions	85×56 mm	
Default OS	Raspbian (after June 24, 2019)	Raspbian (after March 2018)
Price	$35 (1GB RAM), $45 (2GB RAM), $55 (4GB RAM)	$35 (1GB RAM)



- 

Fedora

- **A Linux distribution developed by the community-supported Fedora Project which is sponsored primarily by Red Hat**
- **https://en.wikipedia.org/wiki/Fedora_(operating_system)**

- **https://getfedora.org/**
- **https://alt.fedoraproject.org/alt/**
- **https://spins.fedoraproject.org/**
- **https://fedoraproject.org/wiki/Architectures/ARM**
- **https://fedoramagazine.org/**
- **https://silverblue.fedoraproject.org/**
- **Developer friendly!**

Manjaro

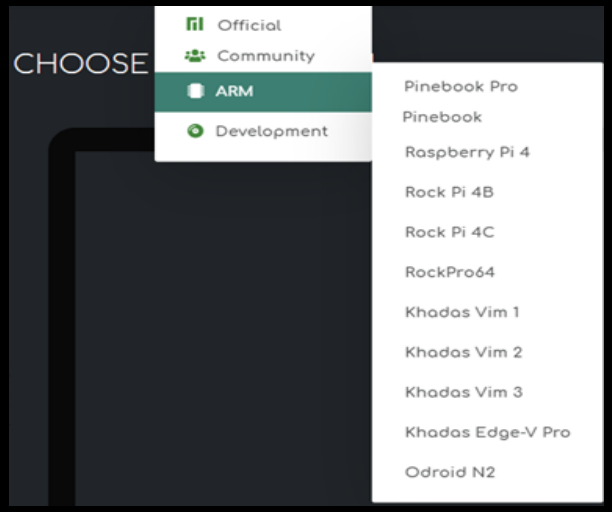
- an open-source Linux distribution based on the Arch Linux operating system
- https://en.wikipedia.org/wiki/Manjaro_Linux
- https://distrowatch.com/

Page Hit Ranking

Data span: Last 6 months Go

Rank	Distribution	HPD*
1	MX Linux	3369▼
2	Manjaro	2490▼
3	Mint	2172▼
4	Pop!_OS	1967▼
5	EndeavourOS	1643▲
6	Ubuntu	1378▼
7	Debian	1281–
8	elementary	1149▼
9	Fedora	1018▼
10	openSUSE	890▲

- https://manjaro.org

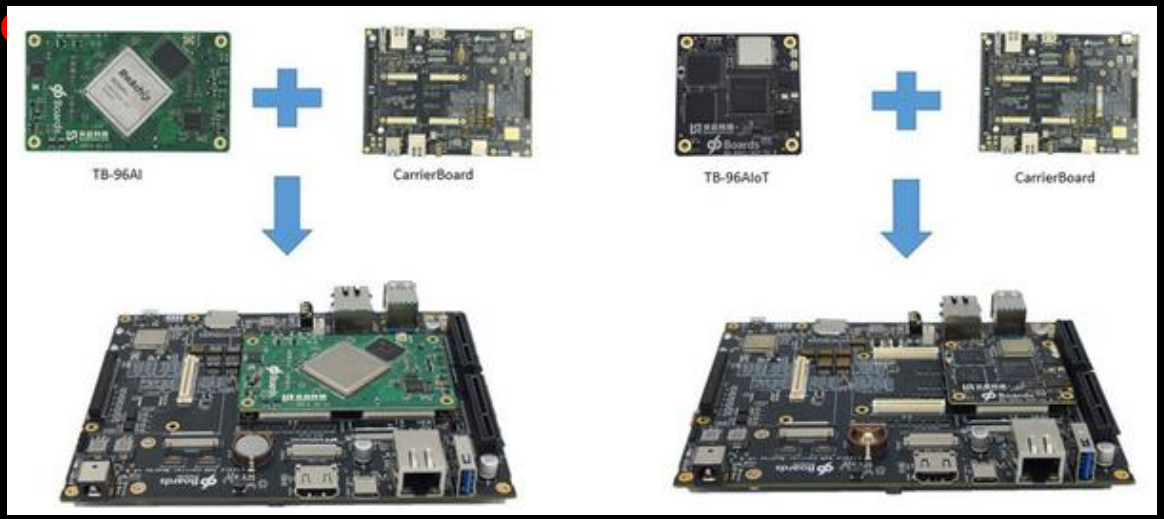


- More and more developer friendly!

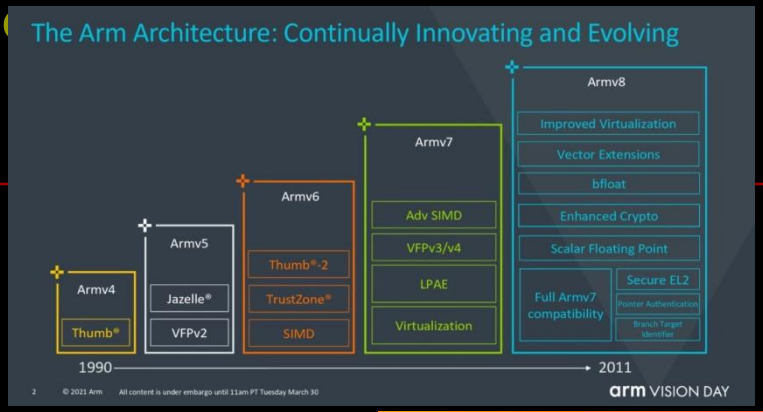
3.2 Modularization

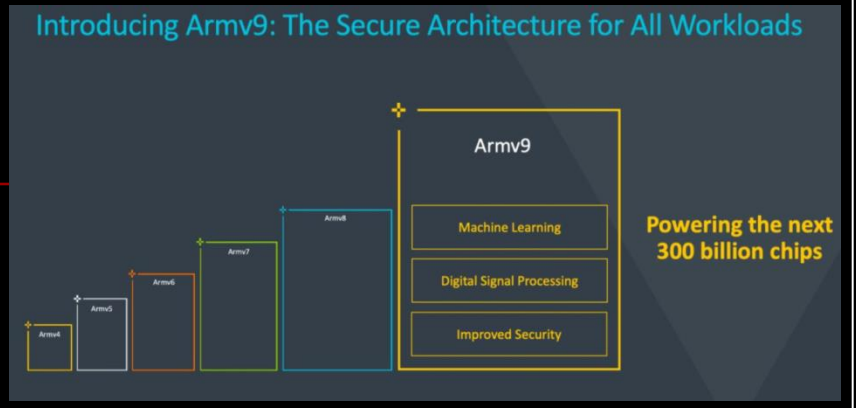
SOM/COM

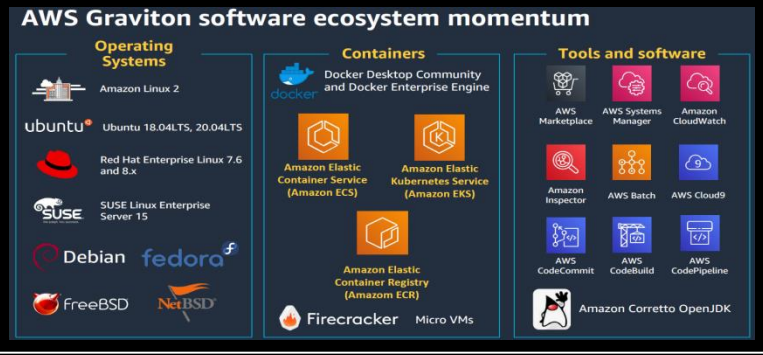
- https://www.linaro.org/news/linaro-announces-launch-of-96boards-system-on-module-som-specification/
- http://linuxgizmos.com/linaro-launches-two-96boards-som-specifications/
- http://static.linaro.org/assets/specifications/96BoardsComputeSoMSpecificationV1.0.pdf
- https://static.linaro.org/assets/specifications/96BoardsWirelessSoMSpecificationV1.0.pdf
-



3.3 ARM Ecosustem in 2021







3.4 Time to Migrating Cross Development from X86 to ARM?

EDA in the Cloud

- https://www.arm.com/company/news/2020/12/arm-moves-production-level-electronic-design-automation-to-the-cloud-with-the-help-of-aws
- ...

Moving Open Soruce EDA Projects to ARM

- **my patch to SpinalHDL for RPi4 (merged)**
  https://github.com/SpinalHDL/SpinalHDL/pull/412

16 sim/src/main/resources/SharedStruct.hpp

```
@@ -4,7 +4,9 @@
 #include<boost/interprocess/sync/scoped_lock.hpp>
 #include<boost/interprocess/sync/interprocess_condition.hpp>
 #include<boost/interprocess/containers/vector.hpp>
+#if !defined(__ARM_ARCH)
 #include<immintrin.h> //_mm_pause
+#endif
 #include<exception>
 #include<atomic>
 #include<thread>
@@ -21,6 +23,18 @@ using namespace boost::interprocess;
 typedef allocator<uint8_t, managed_shared_memory::segment_manager>  ShmemAllocator;
 typedef vector<uint8_t, ShmemAllocator> SharedVector;

+inline void _spin_pause() {
+    #if defined(__ARM_ARCH)
+        #if __ARM_ARCH == 8
+    __asm__ __volatile__("yield" ::: "memory");
+        #else
+    __asm__ __volatile__("yield");
+        #endif
+    #else
+    _mm_pause();
+    #endif
+}
+
 class VpiException: public std::exception
 {
     public:
@@ -68,7 +82,7 @@ class SharedStruct {
             for(uint32_t spin_count = 0; status == ProcStatus::ready; ++spin_count) {

                 if (spin_count < SPINLOCK_MAX_ACQUIRE_SPINS) {
-                    _mm_pause();
+                    _spin_pause();
                 } else {
                     std::this_thread::yield();
                     spin_count = 0;
```

4 sim/src/main/resources/SharedMemIface.cpp

```
@@ -234,8 +234,8 @@ void SharedMemIface::check_ready(){
 
         #ifndef NO_SPINLOCK_YIELD_OPTIMIZATION
         if (spin_count < SPINLOCK_MAX_ACQUIRE_SPINS) {
-            _mm_pause();
-        } else {
+            _spin_pause();
+        } else {
             std::this_thread::yield();
             spin_count = 0;
         }
```

16 sim/src/main/resources/SharedStruct.hpp

```
@@ -70,7 +84,7 @@ class SharedStruct {
                 if (spin_count < SPINLOCK_MAX_ACQUIRE_SPINS) {
-                    _mm_pause();
+                    _spin_pause();
                 } else {
                     std::this_thread::yield();
                     spin_count = 0;
```

3 sim/src/main/scala/spinal/sim/Verilator8ackend.scala

```
@@ -415,7 +415,8 @@ JNIEXPORT void API JNICALL ${jniPrefix}disableWave_1${uniqueId}
         jdk + "/include"
       }

-      val flags   = if(isMac) List("-dynamiclib") else List("-fPIC", "-m64", "-shared", "-Wno-attributes")
+      val arch = System.getProperty("os.arch")
+      val flags   = if(isMac) List("-dynamiclib") else if(arch == "arm" || arch == "aarch64") List("-fPIC", "-shared", "-Wno-attributes") else List("-fPIC", "-m64", "-shared"
 
       config.rtlSourcesPaths.filter(s => s.endsWith(".bin") || s.endsWith(".mem")).foreach(path => FileUtils.copyFileToDirectory(new File(path), new File(s"./")))
```

- **my patch to build riscv-tests on ARM (not merged)**
  https://github.com/riscv/riscv-tests/pull/335

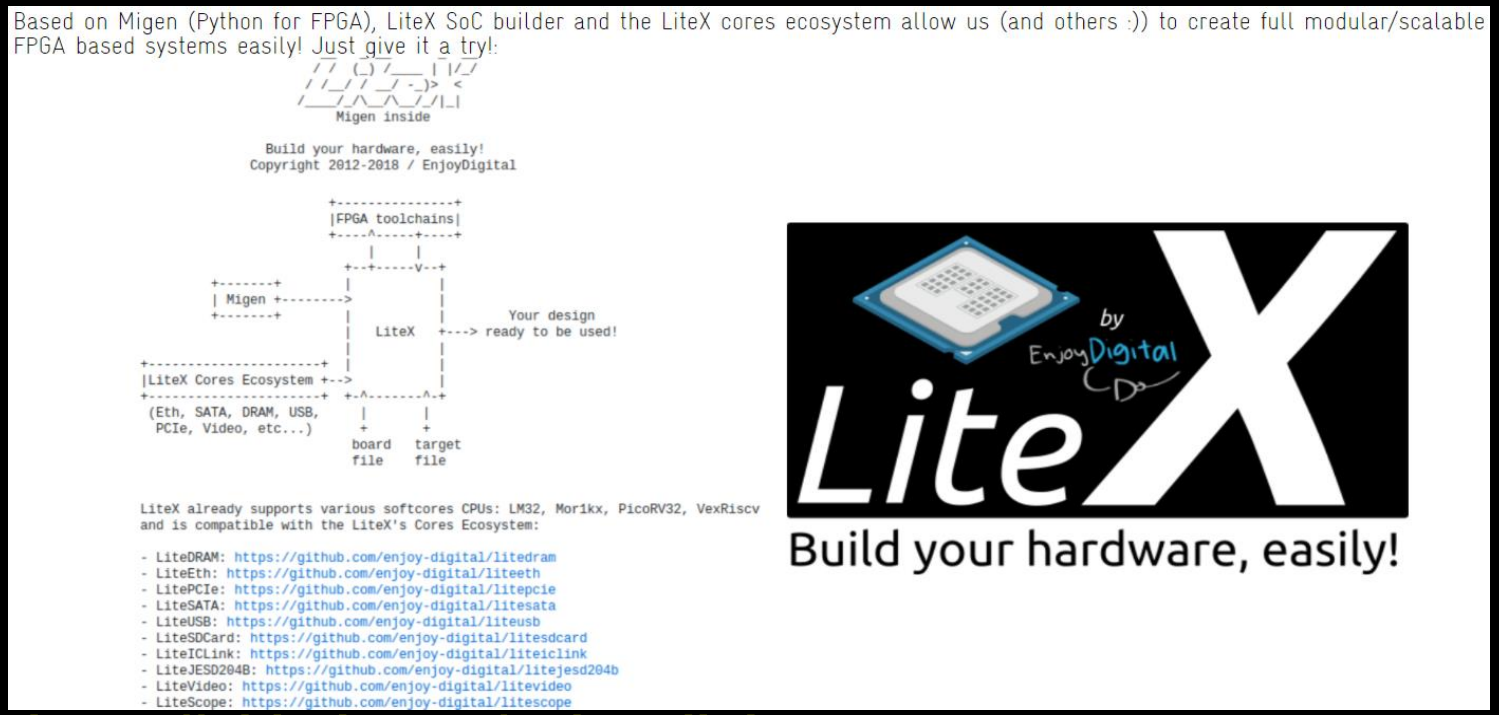
6 README.md

```
@@ -20,6 +20,12 @@ install path, and that the riscv-gnu-toolchain package is installed.
20 20        $ make
21 21        $ make install
22 22
   23 + If you are on an ARM machine, e.g., Raspberry Pi 4 with Fedora Spin 33
   24 + XFCE AArch64, you may use the similar configure arguments as below to
   25 + cross-compile riscv-tests:
   26 + ./configure --prefix=$RISCV/target --build=aarch64-redhat-linux --host=riscv64-unknown-elf --target=riscv64-unknown-elf
   27 + The above target triplets depends on your host system and the toolchain prefix according to $RISCV.
   28 +
23 29   The rest of this document describes the format of test programs for the RISC-V
24 30   architecture.
```

II. Core/SoC Builder

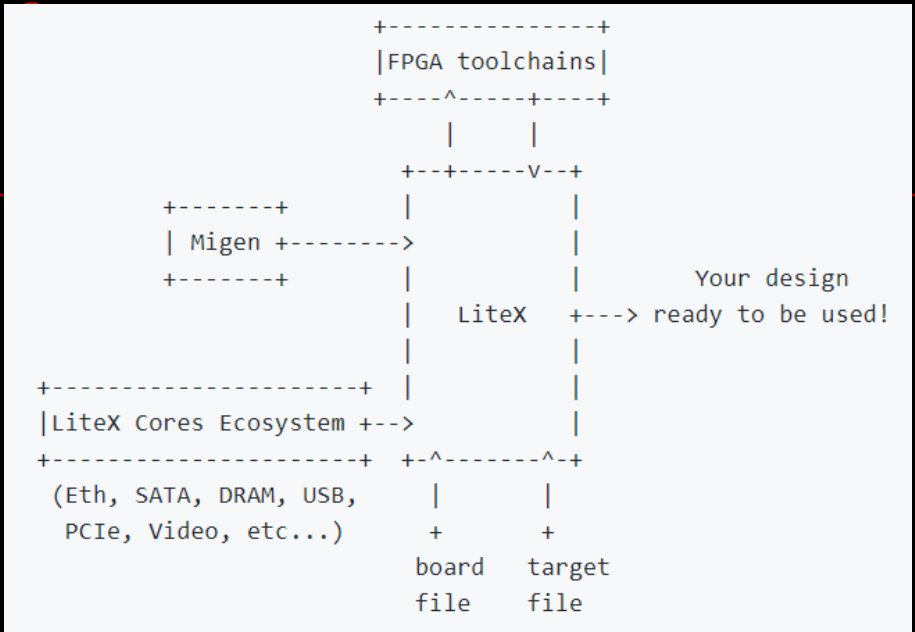
1) LiteX

1.1 Overview

- http://www.enjoy-digital.fr/
- 

- https://github.com/enjoy-digital
- https://github.com/litex-hub
- Similar to Rocket Chip Generator

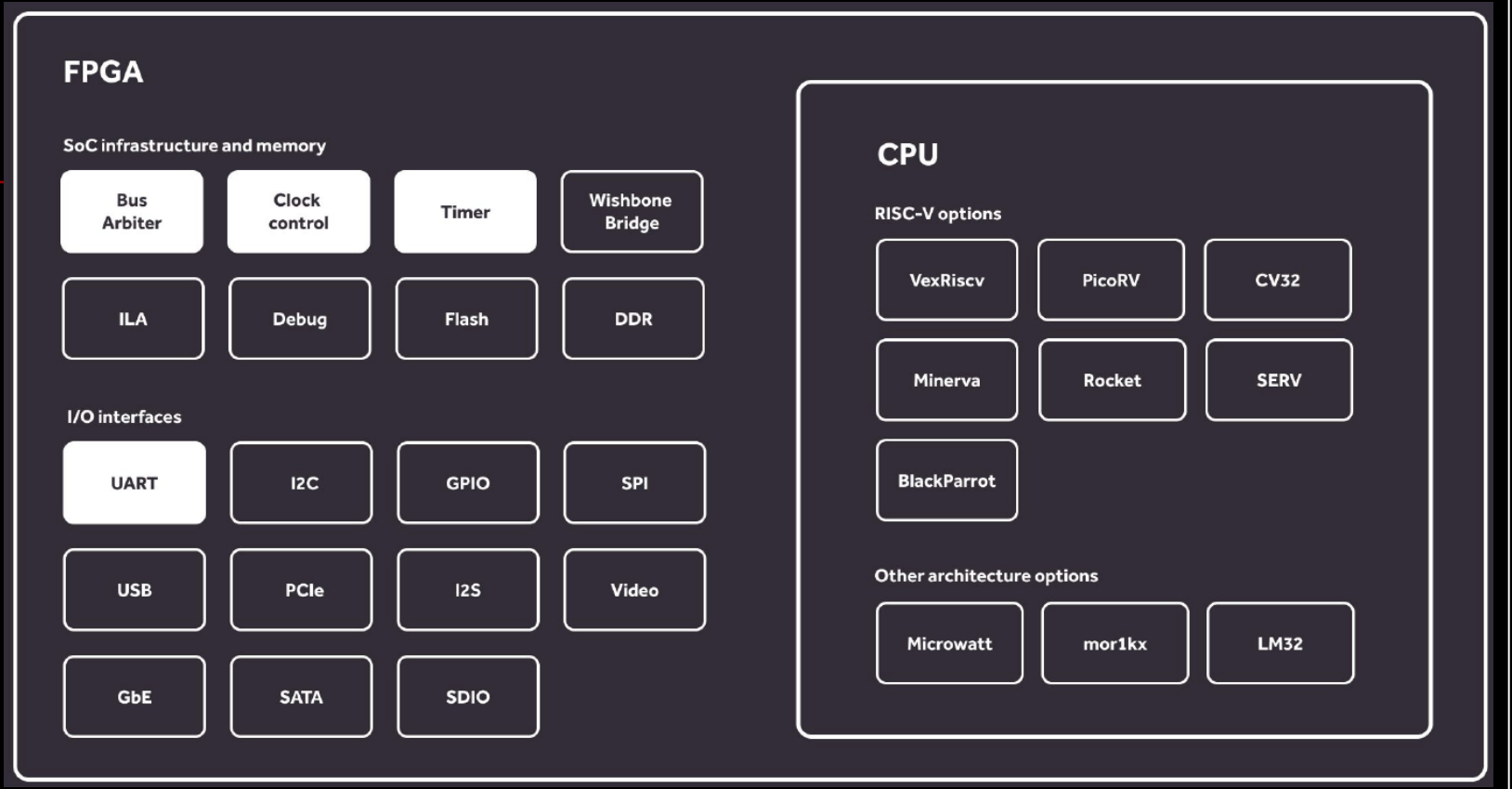
Design Flow



IP name	note
LiteDRAM	DRAM core, fully pipelined, SDRAM to DDR3
LiteEth	Ethernet core, PHYs, MAC, UDP/IP, up to 1Gbps
LitePCIe	DMA and MMAP PCIe core up to Gen2 X4
LiteSATA	SATA 1/2/3 core, DMA, RAID, Mirroring, up to 6Gbps
LiteUSB	USB2.0/3.0 Slave FIFO core + DMA
LiteSDCard	SD card core, DMA, up to UHS-1 (55MB/s R/W)
LiteICLink	Comm core, IOserdes, Transceivers, up to 10gbps
LiteJESD204B	JESD204B core, TX, Subclass1, up to 10Gbps
LiteVideo	Video core, HDMI RX/TX, Framebuffer, up to 1080p60
LiteScope	Logic Analyzer core, access via various bus protocols

Source: "LiteX: an open-source SoC builder and library based on Migen Python DSL", F. Kermarrec, S. Bourdeauducq, J.C. Le Lann, and H. Badier, OSDA 2019

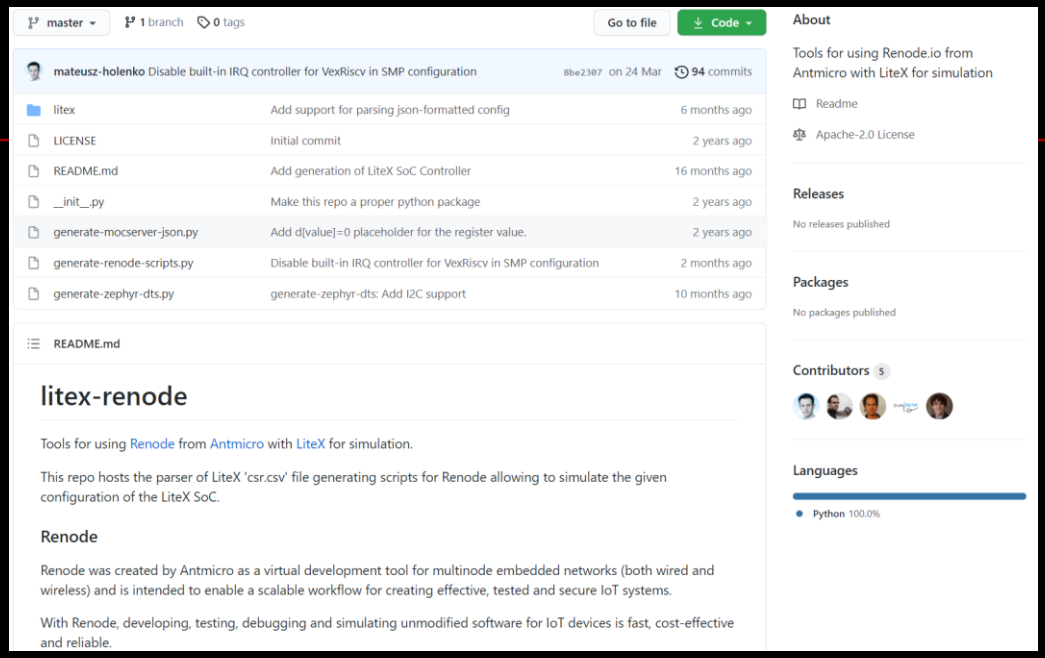
Ecosystem



Source: "Building your world out of blocks with Renode and LiteX", Piotr Zierhoffer, RISC-V Summit 2020

LiteX-Renode

- https://github.com/litex-hub/litex-renode



- LiteX generates a platform description in csr.csv output file
- Automatic generation from LiteX configuration (CSV)
  - platform (REPL)
  - script (RESC)
  - Zephyr DTS overlay
- Perfect for CI setup

Source: "Building your world out of blocks with Renode and LiteX", Piotr Zierhoffer, RISC-V Summit 2020

Setup

- **https://github.com/enjoy-digital/litex/blob/master/litex_setup.py**

...

```
def sifive_riscv_download():
    base_url  = "https://static.dev.sifive.com/dev-tools/"
    base_file = "riscv64-unknown-elf-gcc-8.3.0-2019.08.0-x86_64-"
```

...

1.2 Migen/MiSoC

- https://m-labs.hk/gateware/migen/
  a Python-based tool that automates further the VLSI design process.
- https://github.com/m-labs/misoc
  Built on Migen, MiSoC provides a high performance, flexible and lightweight solution to build system-on-chips for various applications.
- https://m-labs.hk/gateware/nmigen/
  a reboot of Migen, a Python toolbox for building complex digital hardware.

FHDL

- **https://m-labs.hk/migen/manual/fhdl.html**
  The FHDL domain-specific language
- The Fragmented Hardware Description Language (FHDL) is the basis of Migen. It consists of a formal system to describe signals, and combinatorial and synchronous statements operating on them. The formal system itself is low level and close to the synthesizable subset of Verilog, and we then rely on Python algorithms to build complex structures by combining FHDL elements. The FHDL module also contains a back-end to produce synthesizable Verilog, and some structure analysis and manipulation functionality.

  FHDL differs from MyHDL [myhdl] in fundamental ways. MyHDL follows the event-driven paradigm of traditional HDLs (see Background) while FHDL separates the code into combinatorial statements, synchronous statements, and reset values. In MyHDL, the logic is described directly in the Python AST. The converter to Verilog or VHDL then examines the Python AST and recognizes a subset of Python that it translates into V*HDL statements. This seriously impedes the capability of MyHDL to generate logic procedurally. With FHDL, you manipulate a custom AST from Python, and you can more easily design algorithms that operate on it.

```vhdl
-- Libraries imports
library ieee;
use ieee.std_logic_1164.all;

-- Module interface description
entity my_module is
    port(
        clk : in std_logic;
        o   : out std_logic
    );
end entity;

-- Module architecture description
architecture rtl of my_module is
   signal d : std_logic;
   signal q : std_logic;
begin
    -- Combinatorial logic
    o <= q;
    d <= not q;

    -- Synchronous logic
    process(clk)
    begin
        if rising_edge(clk) then
            d <= q
        end if;
    end process

end rtl;
```

What is Migen?

An alternative HDL based on Python


```python
from migen import *

class MyModule(Module):
    def __init__(self):
        self.o = Signal()

        # # #

        d = Signal()
        q = Signal()

        # combinatorial logic
        self.comb += [
            self.o.eq(q),
            d.eq(~q)
        ]

        # synchronous logic
        self.sync += d.eq(q)
```

Migen.FHDL DSL:

```
# types
Constant()
Signal()
ClockSignal()
ResetSignal()
Record()
Array()

# assignment
.eq()
.connect()

# statements
If().Elif().Else()
Case()

# specials
Instance()
Memory()

# structure
ClockDomain()
Module()
self.sync += []
self.comb += []
```

Python used for elaboration by manipulating these objects to create the logic.

Switch from hardware paradigm to software paradigm!

```
from migen import *
from migen.fhdl import verilog

class Blinker(Module):
    def __init__(self, sys_clk_freq, period):
        self.led = led = Signal()

        # # #

        toggle = Signal()
        counter_preload = int(sys_clk_freq*period/2)
        counter = Signal(max=counter_preload + 1)

        self.comb += toggle.eq(counter == 0)
        self.sync += \
            If(toggle,
                led.eq(~led),
                counter.eq(counter_preload)
            ).Else(
                counter.eq(counter - 1)
            )

# Create a 10Hz blinker from a 100MHz system clock.
blinker = Blinker(sys_clk_freq=100e6, period=1e-1)
print(verilog.convert(blinker, {blinker.led}))
```

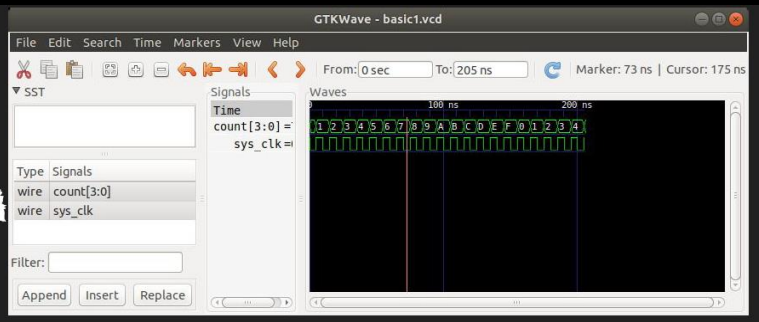
Led blinker

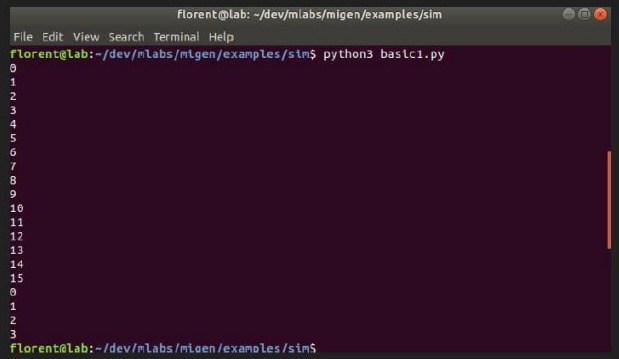
LiteX: SoC builder and library

OSDA Workshop (2019), Florence, March 29

Migen.Sim, a simulator in native Python:

```python
from migen import *


# Our simple counter, which increments at every cycle.
class Counter(Module):
    def __init__(self):
        self.count = Signal(4)

        # At each cycle, increase the value of the count signal.
        # We do it with convertible/synthesizable FHDL code.
        self.sync += self.count.eq(self.count + 1)


# Simply read the count signal and print it.
# The output is:
# Count: 0
# Count: 1
# Count: 2
# ...
def counter_test(dut):
    for i in range(20):
        print((yield dut.count))  # read and print
        yield  # next clock cycle
    # simulation ends with this generator


if __name__ == "__main__":
    dut = Counter()
    run_simulation(dut, counter_test(dut), vcd_name="basic1.vcd")

```





LiteX: SoC builder and library

OSDA Workshop (2019), Florence, March 29

1.3 Use Cases

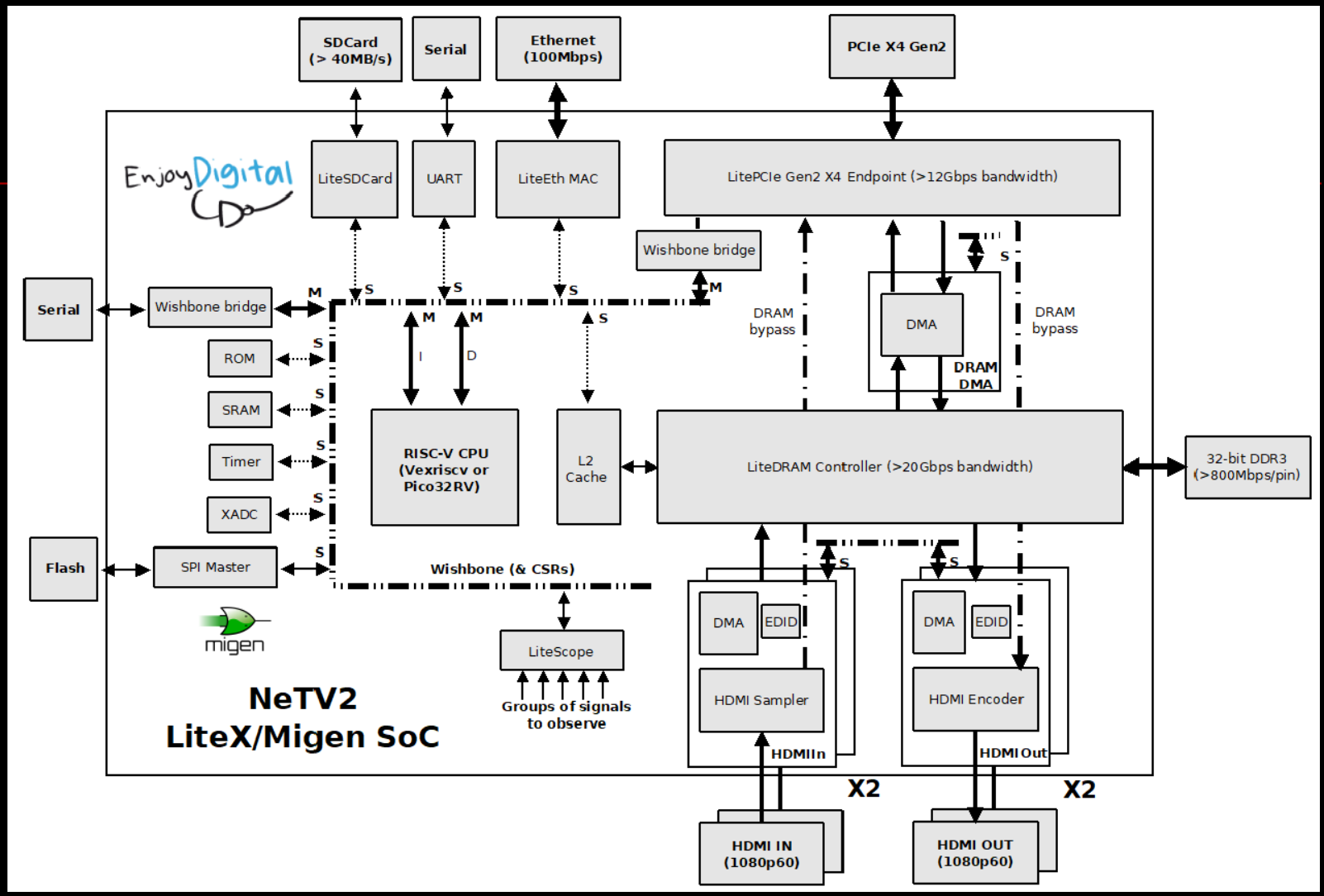

amarisoft
OPEN
BROADCAST SYSTEMS
SILEX
INSIGHT
HASSELBLAD
TechwaY
Mentor
A Siemens Business
SMARTDTV
THALES

NeTV2

- https://www.crowdsupply.com/alphamax/netv2
- An open video development board in a PCI express form factor that supports overlaying content on encrypted video signals

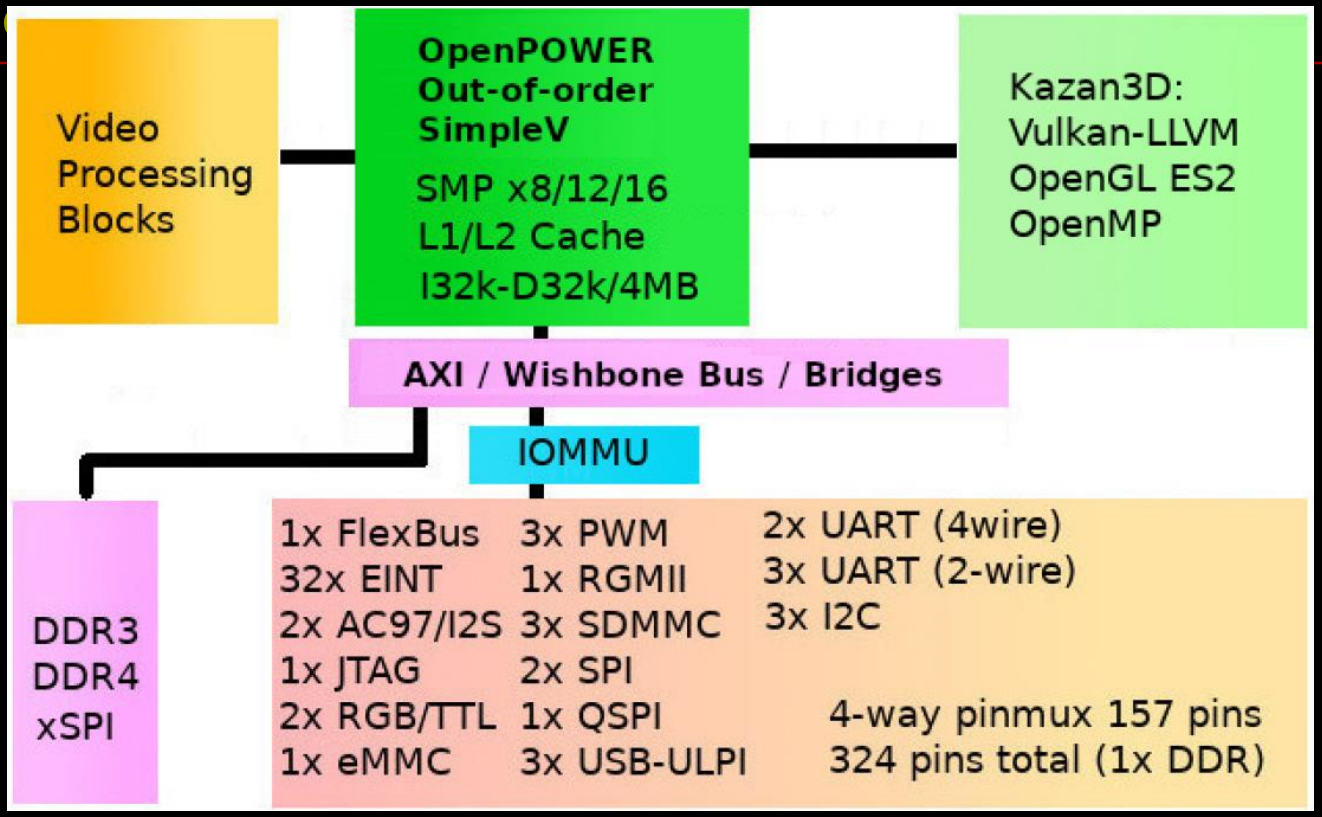
- https://www.adafruit.com/product/4248
- https://github.com/AlphamaxMedia/netv2-fpga
- https://github.com/enjoy-digital/netv2

NeTV2 Libre SoC architecture



LibreSOC

- https://en.wikipedia.org/wiki/Libre-SOC
- A hybrid 3D CPU / VPU / GPU based on OpenPOWER
-



- https://www.fosdem.org/2021/schedule/event/libresocproject/
- https://www.phoronix.com/scan.php?page=news_item&px=Libre-RISC-V-Eyeing-POWER

- https://git.libre-soc.org/

Project	Description
binutils-gdb.git	
c4m-jtag.git	
crowdsupply.git	
dev-env-setup.git	Development environment setup...
fosdem-stand.git	
freedom-sifive.git	Unnamed repository; edit this...
gcc.git	
gem5.git	
ieee754fpu.git	
kazan.git	
kvm-minippc.git	
libreriscv.git	http://libre-riscv.org ikiwiki
libresoc-isa-manual.git	
libresoc-litex.git	
litex.git	
mailman.git	Unnamed repository; edit this...
mesa.git	
nmigen-soc.git	
nmigen-type-annotations.git	
nmigen.git	
nmutil.git	
openpower-isa.git	
pinmux.git	Unnamed repository; edit this...
power-instruction-analyzer.git	
pythondata-cpu-libresoc.git	
riscv-isa-sim.git	Simple-V augmented version...
riscv-tests.git	
rv32.git	
sfpy.git	soft-float python bindings...
shakti-core.git	Unnamed repository; edit this...
shakti-iclass.git	Unnamed repository; edit this...
shakti-peripherals.git	Unnamed repository; edit this...
sifive-blocks.git	Unnamed repository; edit this...
simplev-cpp.git	
soc-cocotb-sim.git	
soc-cxxrtl-sim.git	
soc.git	
soclayout.git	
stands-website.git	
sv2nmigen.git	automatically converts systemv...
utils.git	
vector-math.git	

1.4 IceStorm

- http://www.clifford.at/icestorm/
  fully open source iCE40 flow
- Project IceStorm aims at documenting the bitstream format of Lattice iCE40 FPGAs and providing simple tools for analyzing and creating bitstream files. The IceStorm flow (Yosys, Arachne-pnr, and IceStorm) is a fully open source Verilog-to-Bitstream flow for iCE40 FPGAs.

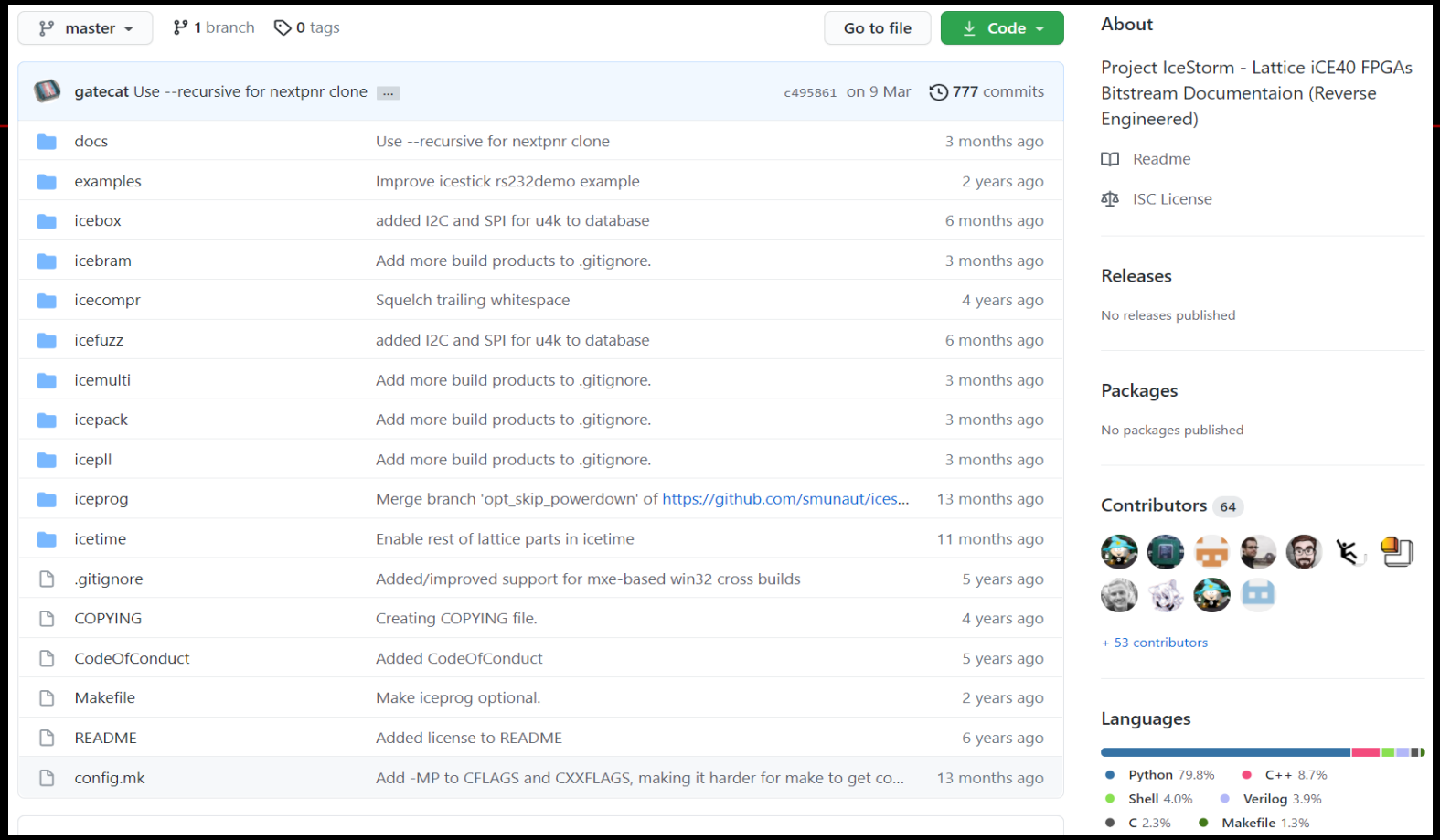
  The focus of the project is on the iCE40 LP/HX 1K/4K/8K chips. (Most of the work was done on HX1K-TQ144 and HX8K-CT256 parts.) The iCE40 UltraPlus parts are also supported, including DSPs, oscillators, RGB and SPRAM. iCE40 LM, Ultra and UltraLite parts are not yet supported.
- http://www.clifford.at/yosys/
  https://github.com/YosysHQ/yosys
- https://github.com/YosysHQ/nextpnr
- iCE40 Boards:
  - Lattice iCEstick
  - Lattice iCE40-HX8K Breakout Board
  - IcoBoard
  - wiggleport
  - ICEd = an Arduino Style Board, with ICE FPGA
  - CAT Board
  - eCow-Logic pico-ITX Lattice ICE40 board
  - Nandland Go Board
  - myStorm board (iCE40 + STM32)
  - DSP iCE board (another iCE40 + STM32 board)
  - BeagleWire iCE40 FPGA BeagleBone cape

  ...

- **Clifford Wolf**
  http://www.clifford.at/
  https://github.com/cliffordwolf

IceStorm Tools

- https://github.com/YosysHQ/icestorm



master | 1 branch | 0 tags | Go to file | Code

gatecat Use --recursive for nextpnr clone ... c495861 on 9 Mar 777 commits

docs	Use --recursive for nextpnr clone	3 months ago
examples	Improve icestick rs232demo example	2 years ago
icebox	added I2C and SPI for u4k to database	6 months ago
icebram	Add more build products to .gitignore.	3 months ago
icecompr	Squelch trailing whitespace	4 years ago
icefuzz	added I2C and SPI for u4k to database	6 months ago
icemulti	Add more build products to .gitignore.	3 months ago
icepack	Add more build products to .gitignore.	3 months ago
icepll	Add more build products to .gitignore.	3 months ago
iceprog	Merge branch 'opt_skip_powerdown' of https://github.com/smunaut/ices...	13 months ago
icetime	Enable rest of lattice parts in icetime	11 months ago
.gitignore	Added/improved support for mxe-based win32 cross builds	5 years ago
COPYING	Creating COPYING file.	4 years ago
CodeOfConduct	Added CodeOfConduct	5 years ago
Makefile	Make iceprog optional.	2 years ago
README	Added license to README	6 years ago
config.mk	Add -MP to CFLAGS and CXXFLAGS, making it harder for make to get co...	13 months ago

About

Project IceStorm - Lattice iCE40 FPGAs Bitstream Documentaion (Reverse Engineered)

Readme

ISC License

Releases

No releases published

Packages

No packages published

Contributors 64

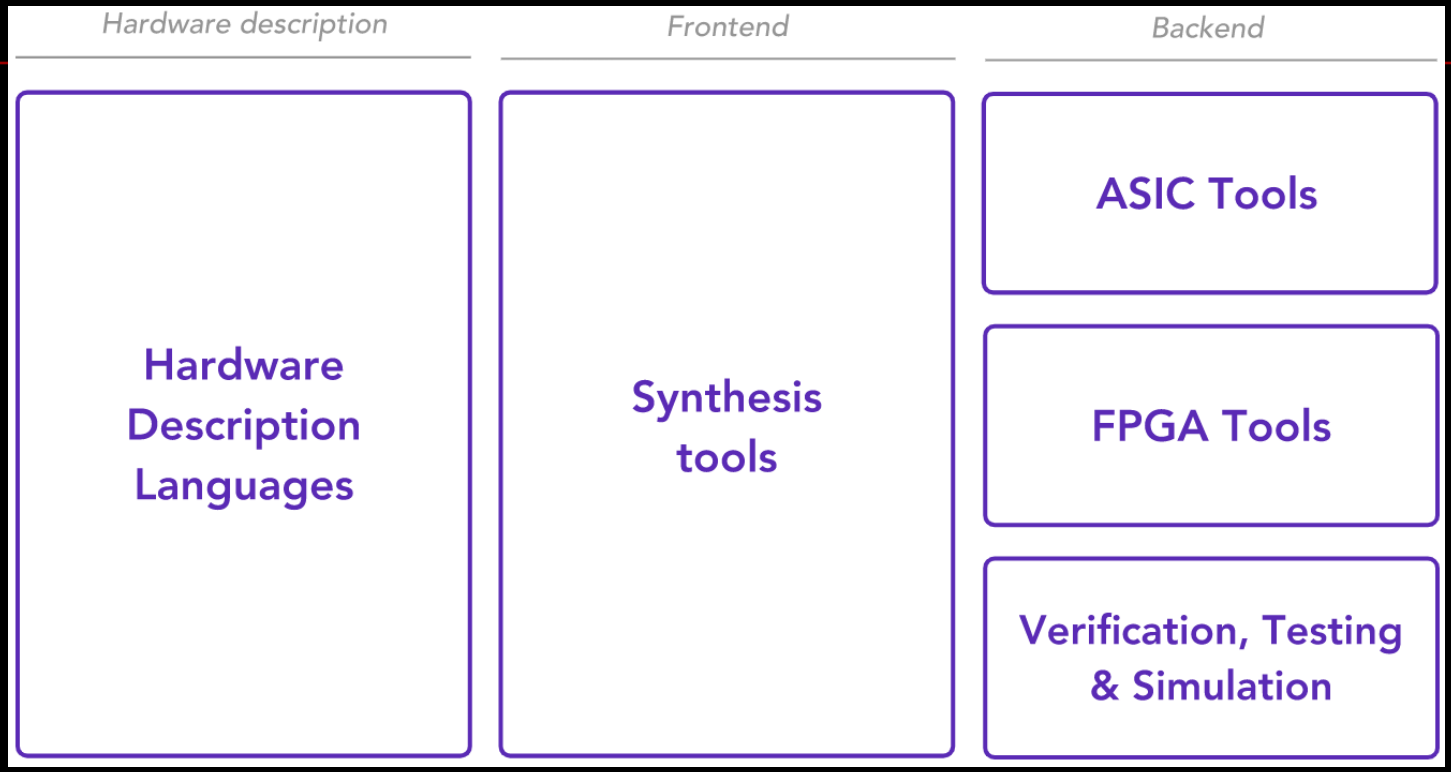
+ 53 contributors

Languages

Python 79.8% C++ 8.7% Shell 4.0% Verilog 3.9% C 2.3% Makefile 1.3%

III. Python-based FOSS FPGA Toolchain

1) Open FPGA



A little surprise

- replacing with Vivado with Trellis + Yosys + Nextpnr result in the same order of magnitude as for the commercial product experiment.

  From: "LiteX: an open-source SoC builder and library based on Migen Python DSL", F. Kermarrec, S. Bourdeauducq, J.C. Le Lann, and H. Badier, OSDA 2019

2.1 OSFPGA

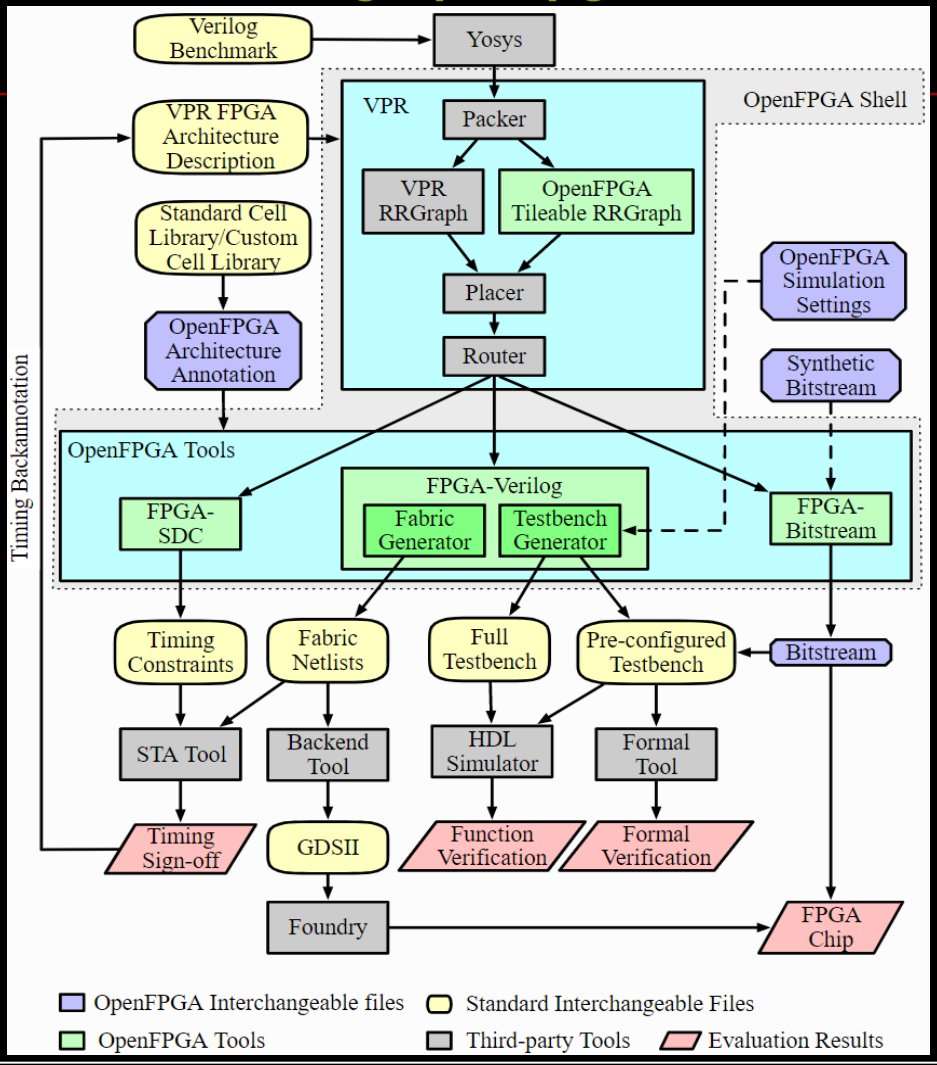
- https://osfpga.org/
- https://github.com/os-fpga
-

2.2 OpenFPGA

- https://github.com/lnis-uofu/OpenFPGA
- https://openfpga.readthedocs.io/en/master/

Tool Suites & Design Flows

- https://openfpga.readthedocs.io/en/master/tutorials/getting_started/tools/#fig-openfpga-tools

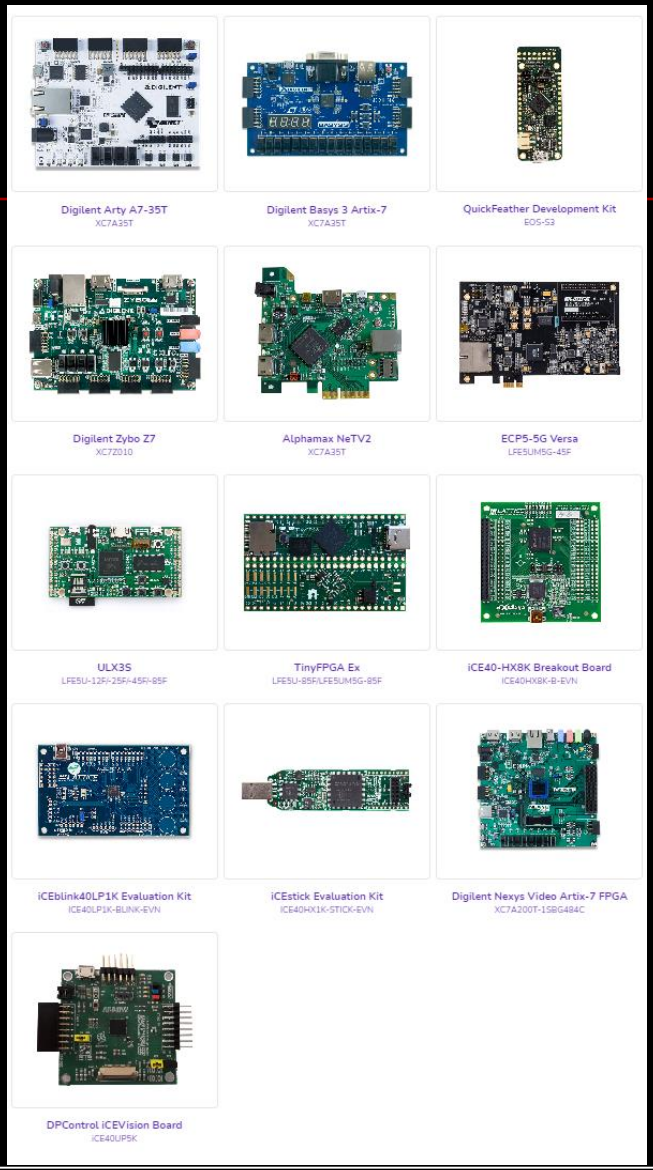


2) SymbiFlow

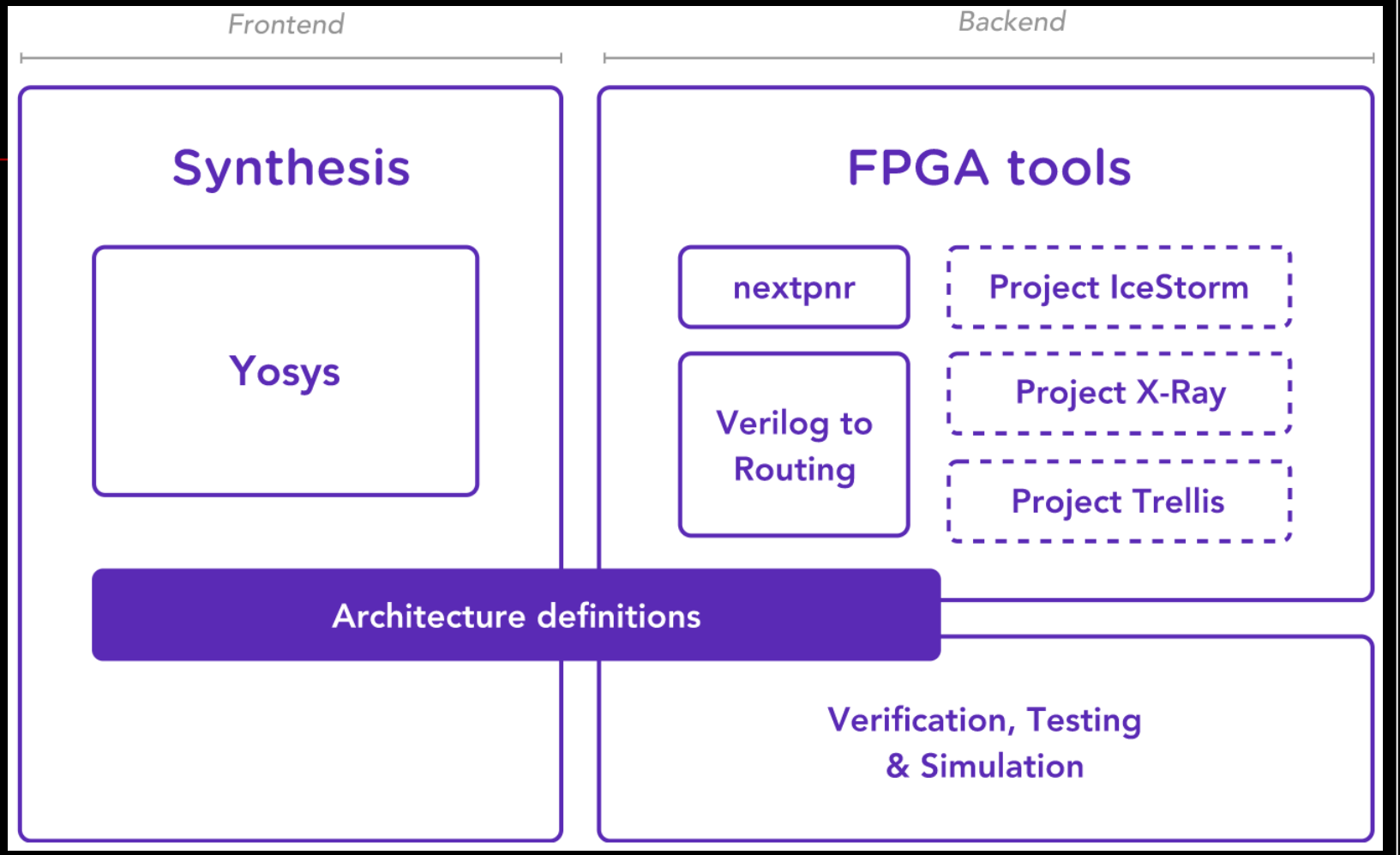
- https://symbiflow.github.io/
  the GCC of FPGAs
- https://symbiflow.readthedocs.io/en/latest/

- https://github.com/SymbiFlow/
- Supported the Xilinx 7-Series, Lattice iCE40/ECP5, QuickLogic EOS S3

Supported boards:

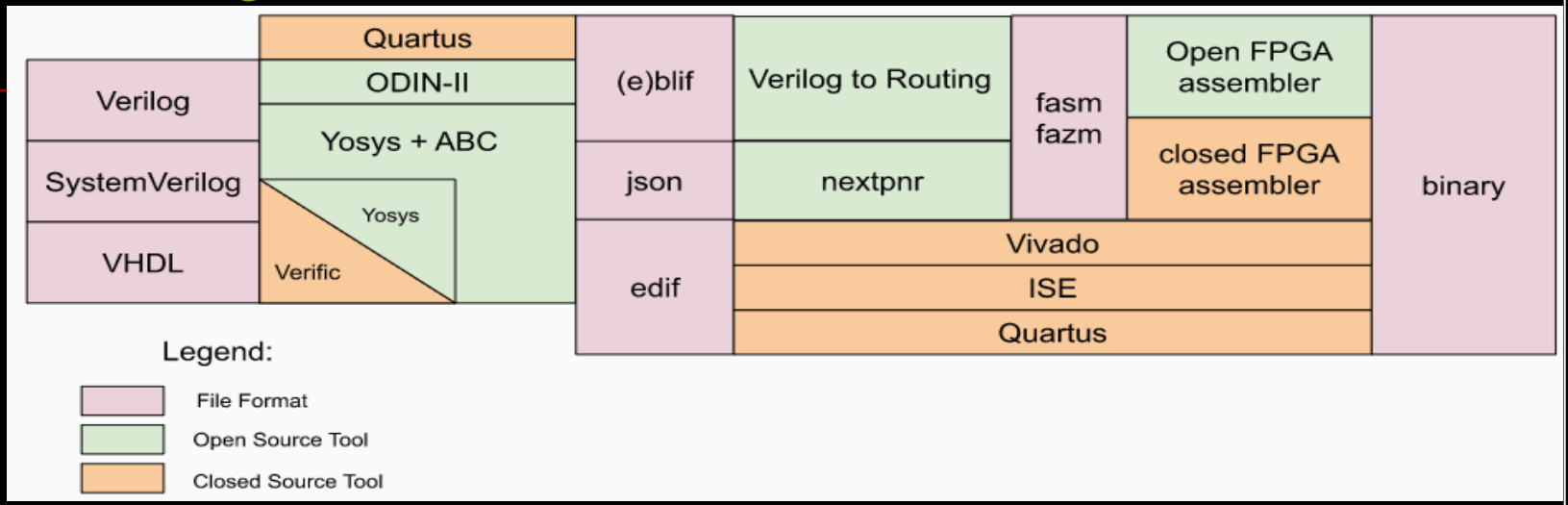


Structure


Frontend
Backend
Synthesis
Yosys
FPGA tools
nextpnr
Verilog to Routing
Project IceStorm
Project X-Ray
Project Trellis
Architecture definitions
Verification, Testing & Simulation

Design Flow

- https://symbiflow.readthedocs.io/en/latest/toolchain-desc/design-flow.html



3) Cocotb

- https://docs.cocotb.org/en/stable/
  a Coroutine based Cosimulation TestBench environment for verifying VHDL and SystemVerilog RTL using Python.
- https://github.com/cocotb/
- How is it different?

cocotb encourages the same philosophy of design re-use and randomized testing as UVM, however is implemented in Python.

With cocotb, VHDL or SystemVerilog are normally only used for the design itself, not the testbench.

cocotb has built-in support for integrating with continuous integration systems, such as Jenkins, GitLab, etc. through standardized, machine-readable test reporting formats.

cocotb was specifically designed to lower the overhead of creating a test.

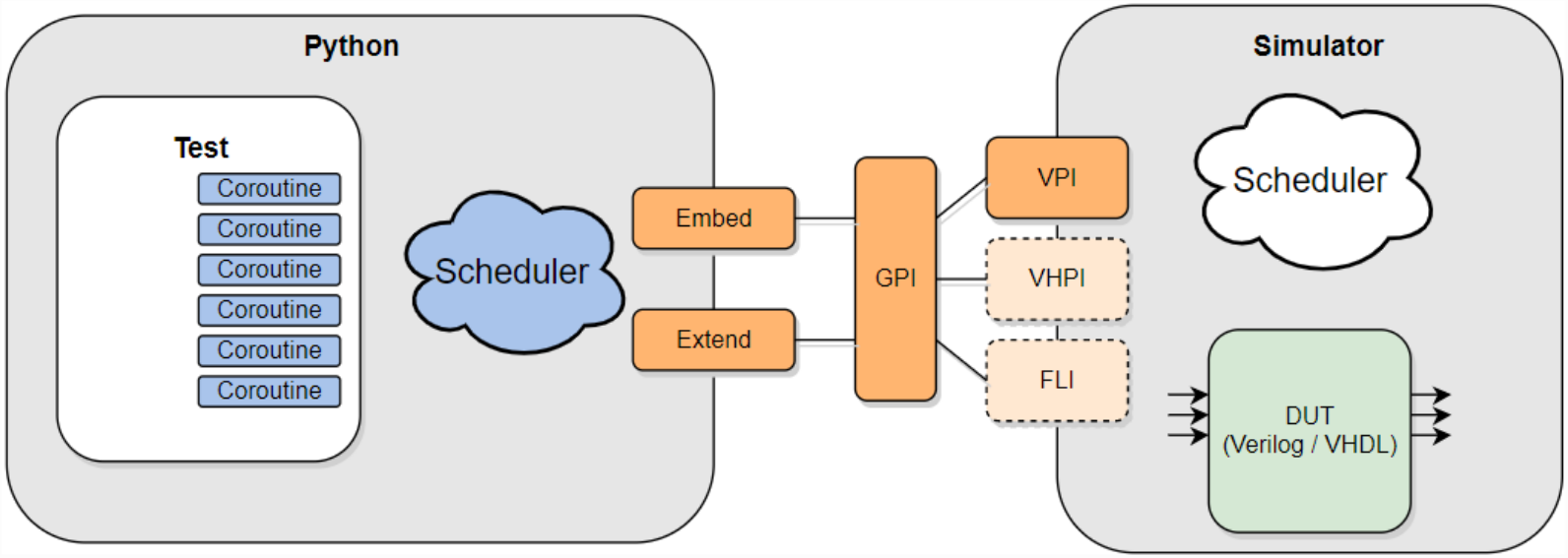
cocotb automatically discovers tests so that no additional step is required to add a test to a regression.

All verification is done using Python which has various advantages over using SystemVerilog or VHDL for verification:

- Writing Python is **fast** - it's a very productive language.
- It's **easy** to interface to other languages from Python.
- Python has a huge library of existing code to **re-use**.
- Python is **interpreted** - tests can be edited and re-run without having to recompile the design or exit the simulator GUI.
- Python is **popular** - far more engineers know Python than SystemVerilog or VHDL.

How it works

A typical cocotb testbench requires no additional RTL code. The Design Under Test (DUT) is instantiated as the toplevel in the simulator without any wrapper code. cocotb drives stimulus onto the inputs to the DUT (or further down the hierarchy) and monitors the outputs directly from Python. Note that cocotb can not instantiate HDL blocks - your DUT must be complete.



A test is simply a Python function. At any given time either the simulator is advancing time or the Python code is executing. The `await` keyword is used to indicate when to pass control of execution back to the simulator. A test can spawn multiple coroutines, allowing for independent flows of execution.

UVM-Python

- **https://github.com/tpoikela/uvm-python**
  UVM 1.2 port to Python
- **Current Status:**

Current status: Testbenches can already be written with all the typical UVM components. UVM Phasing is in place, and working. Stimulus can be generated using hierarchical sequences. Register layer supports already read/write to registers (via frontdoor), and to memories (frontdoor and backdoor). TLM 1.0 is implemented, put/get/analysis interfaces are done, and master/slave interfaces work. Initial implementation of TLM2.0 has also been added. The table below summarizes the status:

Feature	Status
TLM1.0	Done
TLM2.0	Started, 2/3 examples working
Components	Done
Phases	Done
Objections	Test and env-level objections work
Sequences	Partially done, hier sequences work
Registers	Reg/mem access working, built-in sequences partially done

Further Resources

- **https://github.com/cocotb/cocotb/wiki/Further-Resources**

...

Extension modules (cocotbext)

A list of cocotb extensions module packaged as described in documentation.

Please add yours here! If you publish to PyPI, consider replying to https://github.com/pypa/trove-classifiers/pull/24 to get a `Framework :: cocotb` classifier approved.

- cocotbext-eth: Ethernet (GMII, RGMII, XGMII, PTP clock)
- cocotbext-pcie: PCI Express (PCIe), and hard IP core models for UltraScale and UltraScale+
- cocotbext-axi: AXI, AXI lite, and AXI stream
- cocotbext-i2c: I2C interface modules
- cocotbext-uart: UART interface modules
- cocotbext-wishbone: Drive and monitor Wishbone bus
- cocotbext-uart: UART testing
- cocotbext-spi: Drive SPI bus
- cocomod-fifointerface: FIFO testing
- cocotbext-interfaces: "generalization of digital interfaces and their associated behavioral models"; Avalon ST
- cocotbext-ral: A port of the uvm-python RAL to use BusDrivers

...

IV. Practicing

1) Cross Development of RISC-V on ARM

1.1 LiteX

- https://github.com/enjoy-digital/litex/blob/master/litex_setup.py

Toolchain

- https://github.com/riscv/riscv-gnu-toolchain

for bare metal(make -j$(nproc)):

```
[mydev@MyRPi4Manjaro2 Official]$ tree -L 2 -C /opt/MyWorkSpace/DevSW/Toolchain/RISC-V/GCC/Official/ELF/
/opt/MyWorkSpace/DevSW/Toolchain/RISC-V/GCC/Official/ELF/
├── bin
│   ├── riscv64-unknown-elf-addr2line
│   ├── riscv64-unknown-elf-ar
│   ├── riscv64-unknown-elf-as
│   ├── riscv64-unknown-elf-c++
│   ├── riscv64-unknown-elf-c++filt
│   ├── riscv64-unknown-elf-cpp
│   ├── riscv64-unknown-elf-elfedit
│   ├── riscv64-unknown-elf-g++
│   ├── riscv64-unknown-elf-gcc
│   ├── riscv64-unknown-elf-gcc-10.2.0
│   ├── riscv64-unknown-elf-gcc-ar
│   ├── riscv64-unknown-elf-gcc-nm
│   ├── riscv64-unknown-elf-gcc-ranlib
│   ├── riscv64-unknown-elf-gcov
│   ├── riscv64-unknown-elf-gcov-dump
│   ├── riscv64-unknown-elf-gcov-tool
│   ├── riscv64-unknown-elf-gdb
│   ├── riscv64-unknown-elf-gdb-add-index
│   ├── riscv64-unknown-elf-gprof
│   ├── riscv64-unknown-elf-ld
│   ├── riscv64-unknown-elf-ld.bfd
│   ├── riscv64-unknown-elf-lto-dump
│   ├── riscv64-unknown-elf-nm
│   ├── riscv64-unknown-elf-objcopy
│   ├── riscv64-unknown-elf-objdump
│   ├── riscv64-unknown-elf-ranlib
│   ├── riscv64-unknown-elf-readelf
│   ├── riscv64-unknown-elf-run
│   ├── riscv64-unknown-elf-size
│   ├── riscv64-unknown-elf-strings
│   └── riscv64-unknown-elf-strip
├── include
│   └── gdb
├── lib
│   ├── bfd-plugins
│   ├── gcc
│   ├── libcc1.la
│   ├── libcc1.so -> libcc1.so.0.0.0
│   ├── libcc1.so.0 -> libcc1.so.0.0.0
│   ├── libcc1.so.0.0.0
│   └── libriscv64-unknown-elf-sim.a
├── libexec
│   └── gcc
├── riscv64-unknown-elf
│   ├── bin
│   ├── include
│   └── lib
└── share
    ├── gcc-10.2.0
    ├── gdb
    ├── info
    ├── locale
    └── man
```

directly built on RPi4 against “riscv64-unknown-elf-” for ~2.5h(8G RAM + 

)

for Linux(make linux -j$(nproc)):

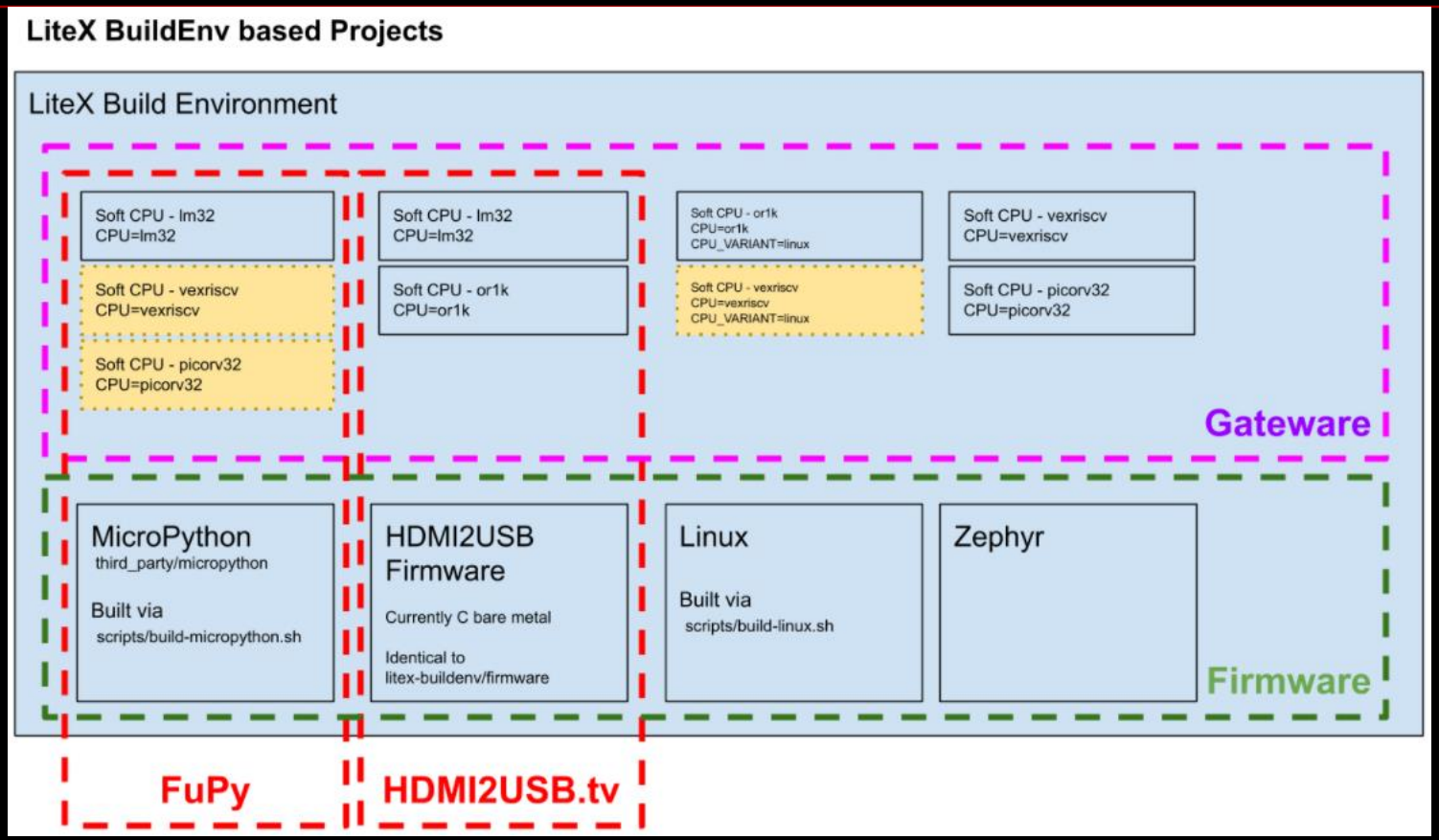
```
[mydev@MyRPi4Manjaro2 Linux]$ tree -L 2 -C /opt/MyWorkSpace/DevSW/Toolchain/RISC-V/GCC/Official/Linux
/opt/MyWorkSpace/DevSW/Toolchain/RISC-V/GCC/Official/Linux
├── bin
│   ├── riscv64-unknown-linux-gnu-addr2line
│   ├── riscv64-unknown-linux-gnu-ar
│   ├── riscv64-unknown-linux-gnu-as
│   ├── riscv64-unknown-linux-gnu-c++
│   ├── riscv64-unknown-linux-gnu-c++filt
│   ├── riscv64-unknown-linux-gnu-cpp
│   ├── riscv64-unknown-linux-gnu-elfedit
│   ├── riscv64-unknown-linux-gnu-g++
│   ├── riscv64-unknown-linux-gnu-gcc
│   ├── riscv64-unknown-linux-gnu-gcc-10.2.0
│   ├── riscv64-unknown-linux-gnu-gcc-ar
│   ├── riscv64-unknown-linux-gnu-gcc-nm
│   ├── riscv64-unknown-linux-gnu-gcc-ranlib
│   ├── riscv64-unknown-linux-gnu-gcov
│   ├── riscv64-unknown-linux-gnu-gcov-dump
│   ├── riscv64-unknown-linux-gnu-gcov-tool
│   ├── riscv64-unknown-linux-gnu-gdb
│   ├── riscv64-unknown-linux-gnu-gdb-add-index
│   ├── riscv64-unknown-linux-gnu-gfortran
│   ├── riscv64-unknown-linux-gnu-gprof
│   ├── riscv64-unknown-linux-gnu-ld
│   ├── riscv64-unknown-linux-gnu-ld.bfd
│   ├── riscv64-unknown-linux-gnu-lto-dump
│   ├── riscv64-unknown-linux-gnu-nm
│   ├── riscv64-unknown-linux-gnu-objcopy
│   ├── riscv64-unknown-linux-gnu-objdump
│   ├── riscv64-unknown-linux-gnu-ranlib
│   ├── riscv64-unknown-linux-gnu-readelf
│   ├── riscv64-unknown-linux-gnu-run
│   ├── riscv64-unknown-linux-gnu-size
│   ├── riscv64-unknown-linux-gnu-strings
│   └── riscv64-unknown-linux-gnu-strip
├── include
│   └── gdb
├── lib
│   ├── bfd-plugins
│   ├── gcc
│   └── libriscv64-unknown-linux-gnu-sim.a
├── libexec
│   └── gcc
├── riscv64-unknown-linux-gnu
│   ├── bin
│   ├── include
│   ├── lib
│   ├── lib32
│   └── lib64
├── share
│   ├── gcc-10.2.0
│   ├── gdb
│   ├── info
│   └── man
└── sysroot
    ├── etc
    ├── lib
    ├── lib32
    ├── lib64
    ├── sbin
    ├── usr
    └── var
```

directly built on RPi4 against “riscv64-unknown-elf-” for ~5h(8G RAM + 

)

The LiteX Build Environment

- https://github.com/timvideos/litex-buildenv
  an environment for building LiteX based FPGA designs. Makes it easy to get everything you need!



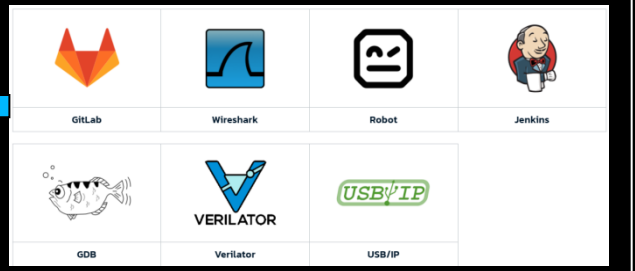
- https://github.com/timvideos/litex-buildenv/wiki

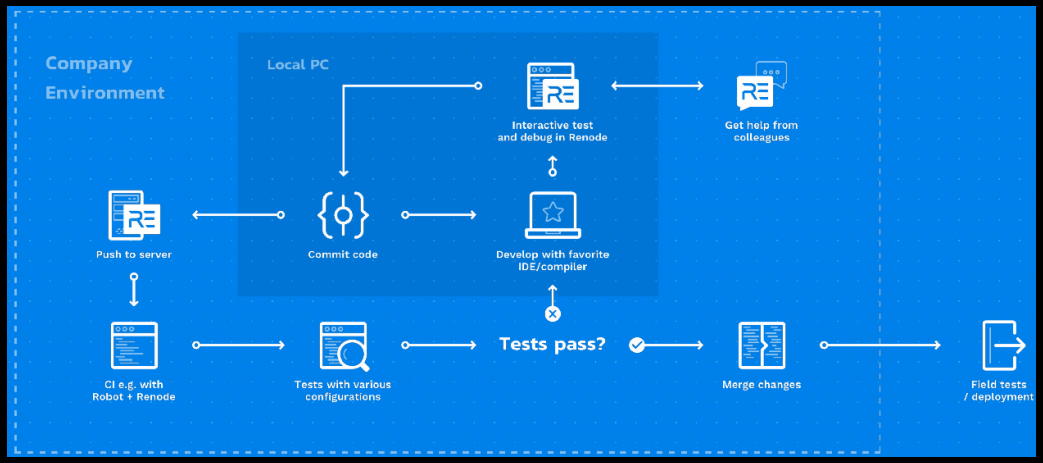
1.3 Try to port Renode to ARM

Overview

- https://renode.io/
  Antmicro's virtual development framework for complex embedded systems
- Strengths:

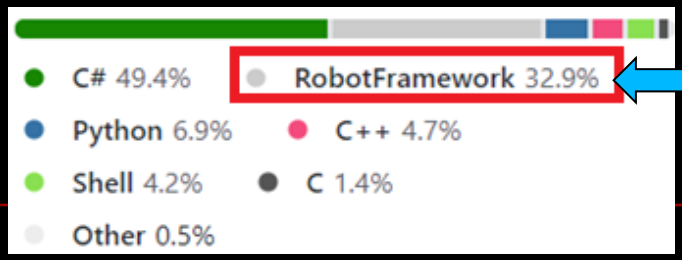
- **full determinism** of execution, shared virtual time
- **transparent & robust debugging**, tracing, analysis, even in multi-node setups
- **easy integration** with your everyday tools, plugins
- **rich model abstractions** with additional functionality "for free" + modular platform description format
- **automated tests and CI integrations**, other collaboration features





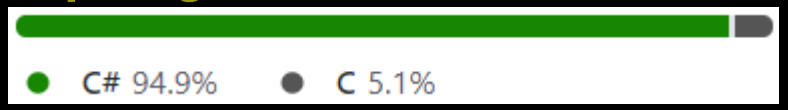
...

- https://github.com/renode/renode



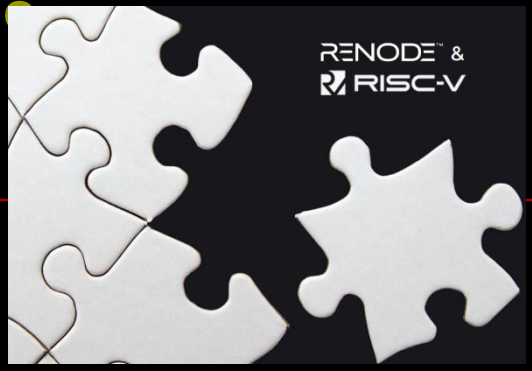
Generic automation framework for acceptance testing and RPA

- https://github.com/renode/renode-infrastructure



- https://github.com/renode/renode-resources
- depends on Mono(https://www.mono-project.com/) for building and running on Linux
- currently, targets X86 only

IoT/RISC-V/Linux/LiteX friendly

- 

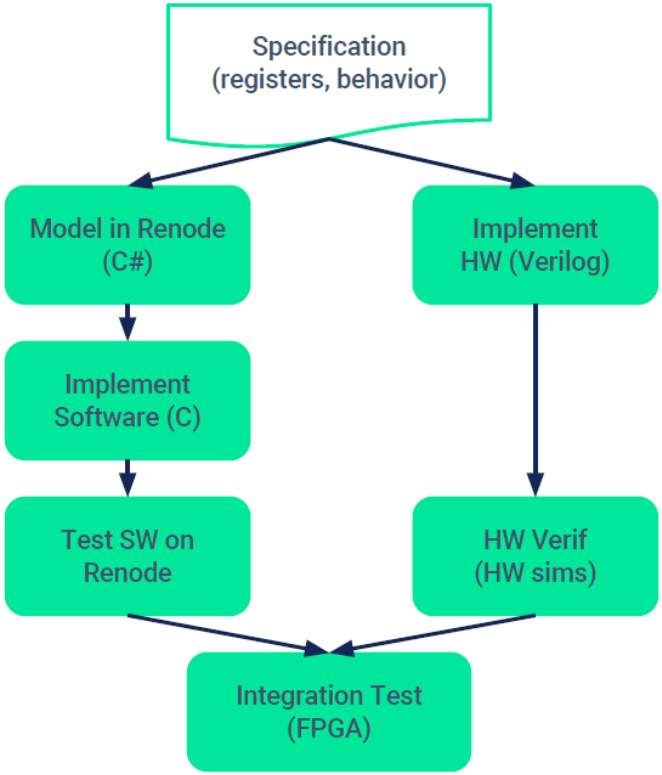
- **https://github.com/riscv/risc-v-getting-started-guide**
- **With Renode, developing, testing, debugging and simulating unmodified software for IoT devices is fast, cost-effective and reliable.**

Parallel HW/SW Development

- HW spec developed between HW and SW teams
- SW team implements spec in Renode and writes firmware against spec, testing on Renode
- In parallel, HW is implementing and testing HW design
- Integration test via FPGA and/or HW simulator

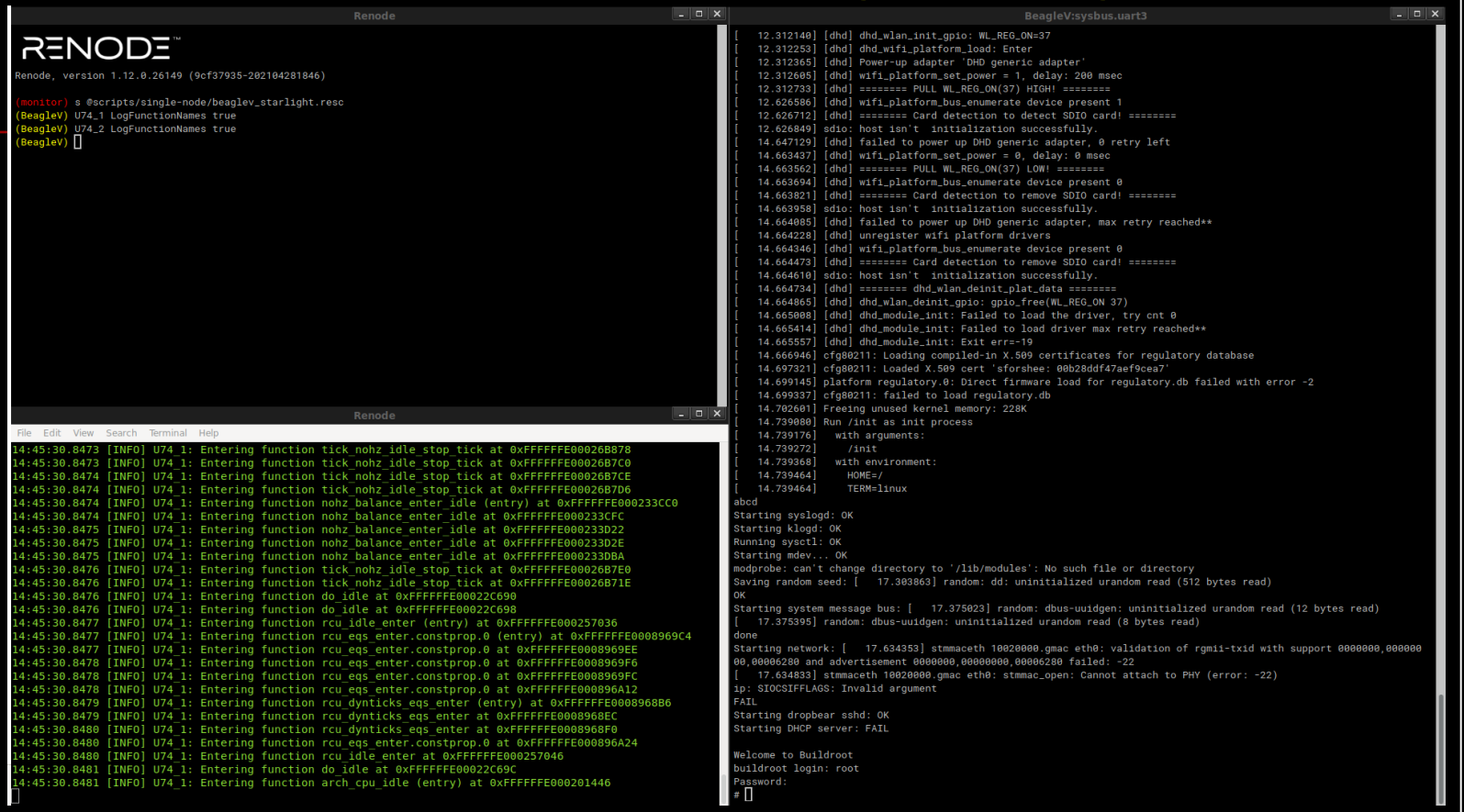
EXAMPLE

HDMI device. We had SW working against spec, under Renode, in advance of HW.



Source: "Hardware/Software Co-Design with the Open Source Renode Framework and RISC-V", Michael Gielda, Getting Started With RISC-V NA Tour 2019

- running BeagleV in Renode
  https://renode.io/news/Linux-on-BeagleV-Starlight-in-Renode/

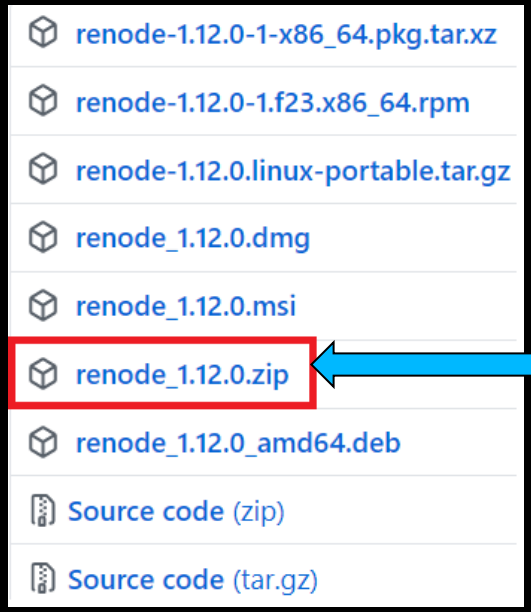


Porting Renode to ARM natively

- **Method 1:**
  1) reconstrcut the build system
  2) porting tlib(https://github.com/antmicro/tlib) to ARM
  3) migrating to .Net 6

  Heavy workload...

- **Method 2:**
  1) base on a special prebuilt release



Mono/.Net assembly only

blocking issue

```
ubuntu@ubuntu:/opt/MyWorkSpace/DevSW/Sim-Modeling/Renode/r1.11.0/bin$ /opt/MyWorkSpace/MyProjs/Runtime/DotNet/Mono/mono-main/build/Install/bin/m
no ./Renode.exe --port 55555
15:05:39.7592 [WARNING] Couldn't start UI - falling back to telnet mode
15:05:42.7343 [INFO] Loaded monitor commands from: /opt/MyWorkSpace/DevSW/Sim-Modeling/Renode/r1.11.0/./scripts/monitor.py
15:05:42.9878 [INFO] Monitor available in telnet mode on port 55555
15:11:59.6687 [INFO] System bus created.
Fatal error:
...
ERROR] FATAL UNHANDLED EXCEPTION: System.Reflection.TargetInvocationException: Exception has been thrown by the target of an invocation. ---> Sy
tem.DllNotFoundException: kernel32 assembly:<unknown assembly> type:<unknown type> member:(null)
 at (wrapper managed-to-native) Antmicro.Renode.Utilities.SharedLibraries.WindowsLoadLibrary(string)
 at Antmicro.Renode.Utilities.SharedLibraries.TryLoadLibrary (System.String path, System.IntPtr& address) [0x00000] in <c0b10643a8564222a33e17b8
5d53209>:0
 at Antmicro.Renode.Utilities.SharedLibraries.LoadLibrary (System.String path) [0x00000] in <c0b10643a8564222a33e17b845d53209>:0
 at Antmicro.Renode.Utilities.Binding.NativeBinder..ctor (Antmicro.Renode.IEmulationElement classToBind, System.String libraryFile) [0x00019] in
<c0b10643a8564222a33e17b845d53209>:0
 at Antmicro.Renode.Peripherals.CPU.TranslationCPU.Init () [0x000a4] in <ccd0ece1ea6442a093ef0525a57331c5>:0
 at Antmicro.Renode.Peripherals.CPU.TranslationCPU..ctor (System.UInt32 id, System.String cpuType, Antmicro.Renode.Core.Machine machine, ELFShar
.ELF.Endianess endianness, Antmicro.Renode.Peripherals.CPU.CpuBitness bitness) [0x000de] in <ccd0ece1ea6442a093ef0525a57331c5>:0
 at Antmicro.Renode.Peripherals.CPU.BaseRiscV..ctor (Antmicro.Renode.Peripherals.Timers.IRiscVTimeProvider timeProvider, System.UInt32 hartId, S
stem.String cpuType, Antmicro.Renode.Core.Machine machine, Antmicro.Renode.Peripherals.CPU.BaseRiscV+PrivilegeArchitecture privilegeArchitecture
 ELFSharp.ELF.Endianess endianness, Antmicro.Renode.Peripherals.CPU.CpuBitness bitness, System.Nullable`1[T] nmiVectorAddress, System.Nullable`1
T] nmiVectorLength) [0x0012c] in <38bea6321ff24c429448b1e05f8339cf>:0
 at Antmicro.Renode.Peripherals.CPU.RiscV32..ctor (Antmicro.Renode.Peripherals.Timers.IRiscVTimeProvider timeProvider, System.String cpuType, An
micro.Renode.Core.Machine machine, System.UInt32 hartId, Antmicro.Renode.Peripherals.CPU.BaseRiscV+PrivilegeArchitecture privilegeArchitecture,
LFSharp.ELF.Endianess endianness, System.Nullable`1[T] nmiVectorAddress, System.Nullable`1[T] nmiVectorLength) [0x00000] in <38bea6321ff24c42944
b1e05f8339cf>:0
 at Antmicro.Renode.Peripherals.CPU.VexRiscv..ctor (Antmicro.Renode.Core.Machine machine, System.UInt32 hartId, Antmicro.Renode.Peripherals.Time
s.IRiscVTimeProvider timeProvider, Antmicro.Renode.Peripherals.CPU.BaseRiscV+PrivilegeArchitecture privilegeArchitecture, System.String cpuType)
[0x0000b] in <38bea6321ff24c429448b1e05f8339cf>:0
 at (wrapper managed-to-native) System.Reflection.RuntimeConstructorInfo.InternalInvoke(System.Reflection.RuntimeConstructorInfo,object,object[]
System.Exception&)
 at System.Reflection.RuntimeConstructorInfo.InternalInvoke (System.Object obj, System.Object[] parameters, System.Boolean wrapExceptions) [0x00
05] in <78e2ffd775544956940f098edf0c4908>:0
  --- End of inner exception stack trace ---
```

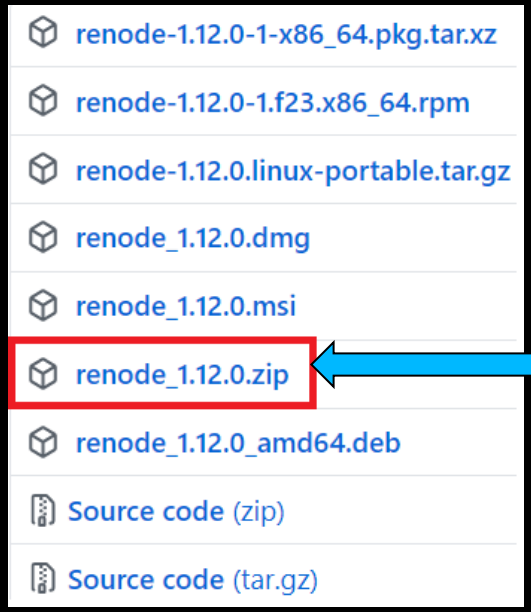
- **Note: different issues will be met on AArch64 and AArch32**

Porting Renode to ARM natively

- **Method 1:**
  1) reconstrcut the build system
  2) porting tlib(https://github.com/antmicro/tlib) to ARM
  3) migrating to .Net 6

  Heavy workload...

- **Method 2:**
  1) base on a special prebuilt release



Mono/.Net assembly only

Virtualization

- Running VM on RPi is heavyweight
- Attempting to make Renode docker image (https://github.com/renode/renode-docker) run on RPi4

Summary

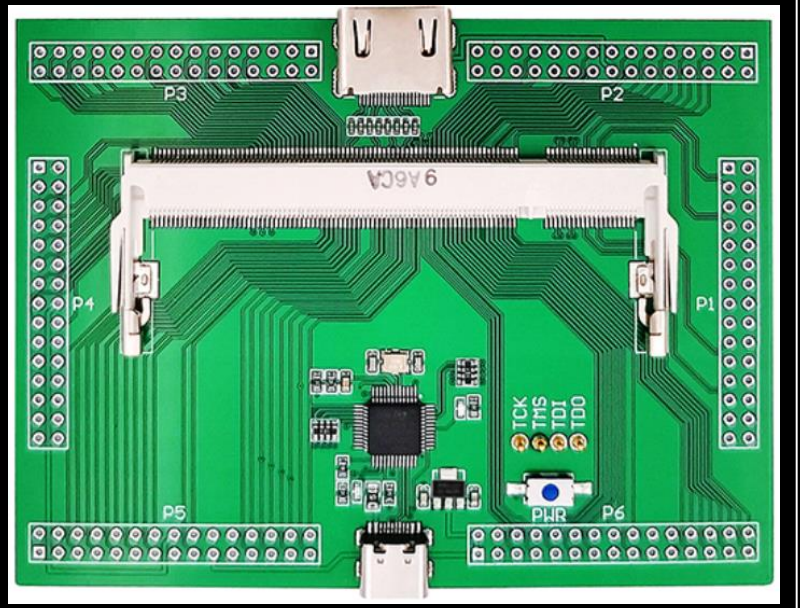
- Project Renode does not have built-in support for building and running on ARM(include unable to emulate cores for AArch64) by now, and a fair amount of work should be payed for corresponding porting if we want to do it by ourselves

---

- Renode has a special release that compiled to Mono/.Net assembly, which makes it possible to run on ARM, but we still need to overcome the blocking issues so as to make it work properly on Linux + ARM
- Natively porting is preferred, but we are trying the way of leveraging virtuliazation technologies

1.3 Development of iCESugar-Pro on ARM

- https://github.com/wuxx/icesugar-pro
- iCESugar-pro is a FPGA board base on Lattice LFE5U-25F-6BG256C, which is fully supported by the open source toolchain (yosys & nextpnr), the board is designed in DDR SODIMM form with 106 usable IOs, with on-board 32MB SDRAM, it can run RISC-V Linux. the on board debugger iCELink (base on ARM Mbed DAPLink) support drag-and-drop program, you can just drag the FPGA bitstream into the virtual disk to program, and with a additional USB CDC serial port direct connect to FPGA, so you can only use one TYPE-C cable to develop and test.
- 



Preparing dev env on RPi4

- as an alternative dev env for the official Ubuntu 18.04 VM image
- includes yosys, nextpnr, icestorm, gcc, sbt, openocd, ujprog...

- **make ujprog(https://github.com/f32c/tools, depends on libftdi, libhidapi, libusb...) available on RPi4**

Patching: https://github.com/wuxx/icesugar-pro/blob/master/tools/ujprog.patch + our patch

```
[mydev@fedora tools-master-patched]$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   ujprog/ft232r_flash.c
        modified:   ujprog/ujprog.c

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        ujprog/Makefile.linux.aarch64
```

```
[mydev@fedora tools-master-patched]$ cat ujprog/Makefile.linux.aarch64
CFLAGS += -Wall -D__linux__ -std=gnu99 -I/usr/include/libftdi1
LDFLAGS += -lftdi1

ujprogm: ujprog.c
        ${CC} ${CFLAGS} -I/usr/include $^ ${LDFLAGS} -lusb -o $@

ft232r_flash_m: ft232r_flash.c
        ${CC} ${CFLAGS} -I/usr/include/libusb-1.0 $^ ${LDFLAGS} -lusb-1.0 -o $@

install: ujprogm ft232r_flash_m
        install -m 4755 ujprogm /usr/local/bin
        install -m 4755 ft232r_flash_m /usr/local/bin

.PHONY: all clean

all: ujprogm ft232r_flash_m

clean:
        rm -f ujprogm *~
[mydev@fedora tools-master-patched]$
```

...

V. Python Runtimes

1) Overview

- https://en.wikipedia.org/wiki/Python_(programming_language)

A faster CPython

- https://thenewstack.io/guido-van-rossums-ambitious-plans-for-improving-python-performance/
- https://github.com/faster-cpython/
- http://hpyproject.org/
- ...

C-based

- **CPython**
  https://www.python.org/
  https://github.com/python/cpython

the official Python implementation, no JIT

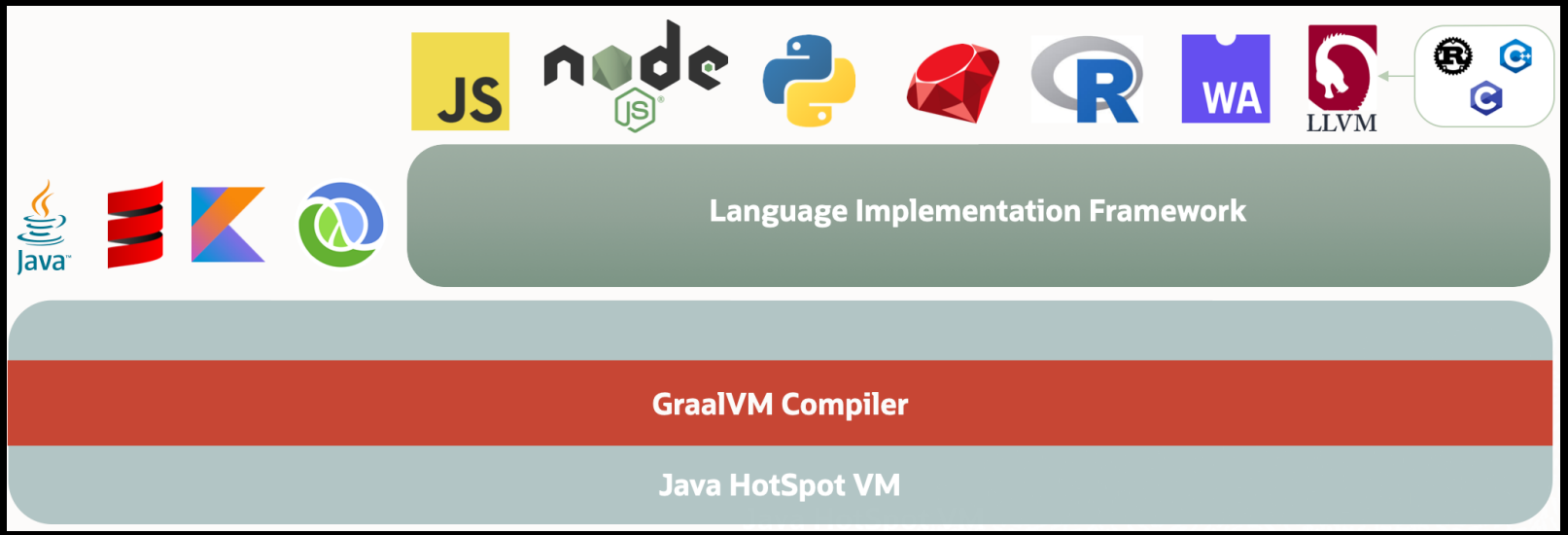
- **Cinder**
  https://github.com/facebookincubator/cinder

Cinder is Instagram's internal performance-oriented production version of CPython 3.8. It contains a number of performance optimizations, including bytecode inline caching, eager evaluation of coroutines, a method-at-a-time JIT, and an experimental bytecode compiler that uses type annotations to emit type-specialized bytecode that performs better in the JIT.

Currently, support X86 only, though the underlying AsmJit (https://asmjit.com/) is available on various architectures like ARM.

Java-based

- GraalPython
  https://github.com/oracle/graalpython
  a Python 3 implementation built on GraalVM

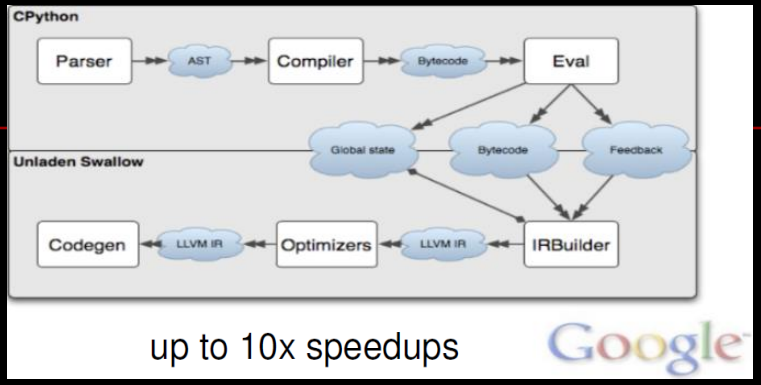


currently, only available for X86

- Jython
  https://github.com/jython/jython3
  not actively developed

LLVM-based (VMKit, MCJIT…)

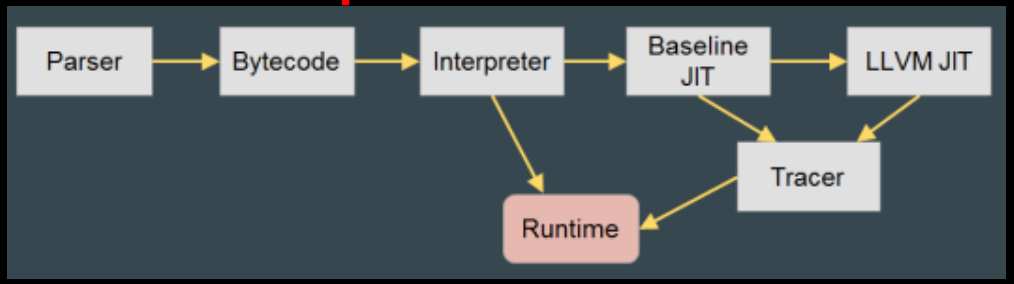
- **Unladen Swallow**



dropped off

- **PySton** Dropbox

still in development...



.Net-based

- IronPython
  https://ironpython.net/
  https://github.com/IronLanguages/ironpython3
  implementation of Python 3.x for .NET Framework that is built on top of the Dynamic Language Runtime.
  https://github.com/IronLanguages/ironpython3/tree/master/Documentation

- Pyjion
  https://github.com/tonybaloney/Pyjion
  Not a standalone Python runtime, just a JIT extension for CPython that compiles your Python code into native CIL and executes it using the .NET 5 CLR.
  Currently, support X86 only, though the underlying CoreCLR is cross-platform.

WASM-based

- **Pyodide**
  https://github.com/pyodide/pyodide
  Python with the scientific stack, compiled to WebAssembly.

Pyodide may be used in any context where you want to run Python inside a web browser.

Pyodide brings the Python 3.8 runtime to the browser via WebAssembly, along with the Python scientific stack including NumPy, Pandas, Matplotlib, SciPy, and scikit-learn. The packages directory lists over 75 packages which are currently available. In addition it's possible to install pure Python wheels from PyPi.

Pyodide provides transparent conversion of objects between Javascript and Python. When used inside a browser, Python has full access to the Web APIs.

- **Misc**
  wasmer-python…

Rust-based

- **RustPython**
  https://github.com/RustPython/RustPython
  a Python-3 (CPython >= 3.8.0) Interpreter written in Rust
  Current Status:

> RustPython is in development, and while the interpreter certainly can be used in interesting use cases like running Python in WASM and embedding into a Rust project, do note that RustPython is not totally production-ready.

awesome, but is not mature yet…

2) Benchmarking

2.1 Pyperformance

- https://github.com/python/pyperformance
- Python Performance Benchmark Suite (mainly targets CPython)
- https://pyperformance.readthedocs.io/
- https://pyperformance.readthedocs.io/usage.html#run-benchmarks

  ...
- Be fit for standalone Python runtime, but friendly for cases like IronPython, Pyjion, RustPython...

Summary

- prebuilt PyPy for Linux AArch64 is not stable during the test, crash or timeout often occurred
- RustPython is awesome, but it is still in development and lacks of some features to be used as a standalone Python runtime for testing
- made IronPython successfully running on RPi4B, but still got failed to run Pyperformance since the latter does not well handle the case of “Python executable” which need a “wrapper” (just like Mono for IronPython)

3) Conclusion

- Continuously improve CPython is the most practical solution, especailly brings JIT to CPython
- GraalPython is the most attrative candidate in the near future
- .Net and WASM/Rust based Python runtimes have great potential

VI. Wrap-up

- Next generation Core/SoC frameworks like LiteX that based eDSL bring higher productivity to hardware development, and are being more and more widely used in emerging hardware such as RISC-V's design & development;

- Open source EDA tools are blooming, and corresponding ecosystem is becoming mature;

- Python is playing a more and more important role in building open source EDA toolchains due to its built-in characteristics like simplicity, ease of use, high productivity, powerful ecosystem, the influence among both professional and non-professional developers, etc.

- Now ARM's HW/SW ecosystem is entering a new outbreak period, migrating traditional cross-platform development work from X86 hosts to that of ARM are bringing more choices and new experiences for developers.

Q & A

Thanks!

Reference

Slides/materials from many and varied sources:

- http://en.wikipedia.org/wiki/
- http://www.slideshare.net/
- https://programmersought.com/article/66077766598/
- https://www.gitmemory.com/mateusz-holenko
- https://en.wikipedia.org/wiki/Runtime_(program_lifecycle_phase)
- …