

# LiteX: SoC builder and library

OSDA Workshop (2019), Florence, March 29

Florent Kermarrec, [florent@enjoy-digital.fr](mailto:florent@enjoy-digital.fr)

# Enjoy-Digital

Founded in 2011.

FPGA consulting / Full FPGA based systems design.

Reuse and create open-source tools/cores to be more efficient.



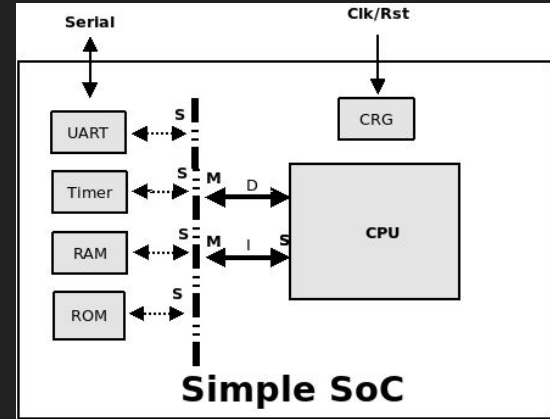
## Presentation:

- 1) Let's design an FPGA SoC the traditional way...
- 2) LiteX : Build your hardware, easily!
- 3) Systems using Migen/LiteX and Future!

Let's design an FPGA SoC the traditional way...

## Create a simple SoC:

- Choose a CPU.
- Add cores around it.
- Connect all that.
- Write low-level software.



[...]

Create a simple SoC:

[One day later...]

Great we got it working by only  
using open-source cores!

~ **500 LOC** for the wrapper.

Let's add our own core:

Let's create the new core, simulate it with an open-source simulator.

Everything is working fine, let's integrate it!

Let's add our own core:

It's... **almost working**, but not behaving as expected, is it a bug in the core or something we miss-understood in the datasheet of the chip we are interfacing with??

We need to see **what is going on...\***

(\*signal is not observable with an external logic analyzer)



**Let's add our own core:**

**No standard way to debug FPGA designs:**

**1) Create our own debug tools: ~a few days, limited functionalities/capabilities.**

**2) Use vendor's free embedded logic analyzer...**

Let's add our own core:

No standard way to debug FPGA designs:

1) Create our own debug tools: ~a few days, limited functionalities/capabilities.

2) Use vendor's free embedded logic analyzer...\*

\* Just this time...

**Let's add our own core:**

**OK, issue is understood (was a bad interpretation of the spec), core is adapted and now working fine...**

Let's add some RAM:

We need RAM to run an OS, our board has **DDR3**.

Let's look at open-source solutions... Nothing seems to exist except something called **LiteDRAM** written in ...Python, how can it be serious? :)

Let's **generate a controller with vendor's tools...\***

\* Just this time... (again)

We could continue with **several similar examples...:**

- How to support different vendors and reuse the same methodology/cores?
- How to build high performance systems at a reasonable cost.
- How to create complex SoCs and limit LOC
- ....

**In addition to open-source FPGA toolchains, there is a need to high level tools and high performance cores...**

# LiteX: Build your hardware easily!



# Migen : A python HDL toolbox.





# Migen : A python HDL toolbox.

```
-- 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*



```
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)
```

LiteX: SoC builder and library

OSDA Workshop (2019), Florence, March 29

FHDL is a Python DSL (Domain Specific Language) defined by Migen and allow generating Verilog or instantiating Verilog/VHDL from Python code.

Principle: Use Python to create a list of combinatorial and synchronous assignments.

```
self.comb += ["..."]
```

```
self.sync += ["..."]
```

And generate a verilog file from these assignments.

```
1 from migen import *
2 from migen.fhdl import verilog
3
4 class Blinker(Module):
5     def __init__(self, sys_clk_freq, period):
6         self.led = led = Signal()
7
8         ###
9
10        toggle = Signal()
11        counter_preload = int(sys_clk_freq*period/2)
12        counter = Signal(max=counter_preload + 1)
13
14        self.comb += toggle.eq(counter == 0)
15        self.sync += \
16            If(toggle,
17                led.eq(~led),
18                counter.eq(counter_preload)
19            ).Else(
20                counter.eq(counter - 1)
21            )
22
23 # Create a 10Hz blinker from a 100MHz system clock.
24 blinker = Blinker(sys_clk_freq=100e6, period=1e-1)
25 print(verilog.convert(blinker, {blinker.led}))
26
```

Led blinker

## Migen.FHDL DSL:

```
1 # types
2 Constant()
3 Signal()
4 ClockSignal()
5 ResetSignal()
6 Record()
7 Array()
8
9 # assignment
10 .eq()
11 .connect()
12
13 # statements
14 If().Elif().Else()
15 Case()
16
17 # specials
18 Instance()
19 Memory()
20
21 # structure
22 ClockDomain()
23 Module()
24 self.sync += []
25 self.comb += []
26
```

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



Switch from hardware  
paradigm to software  
paradigm!

```
1 from migen import *
2 from migen.fhdl import verilog
3
4 class Blinker(Module):
5     def __init__(self, sys_clk_freq, period):
6         self.led = led = Signal()
7
8         ###
9
10        toggle = Signal()
11        counter_preload = int(sys_clk_freq*period/2)
12        counter = Signal(max=counter_preload + 1)
13
14        self.comb += toggle.eq(counter == 0)
15        self.sync += \
16            If(toggle,
17                led.eq(~led),
18                counter.eq(counter_preload)
19            ).Else(
20                counter.eq(counter - 1)
21            )
22
23        # Create a 10Hz blinker from a 100MHz system clock.
24        blinker = Blinker(sys_clk_freq=100e6, period=1e-1)
25        print(verilog.convert(blinker, {blinker.led}))
26
```

Led blinker



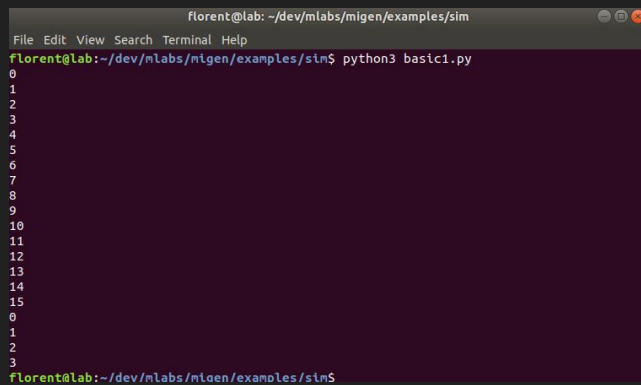
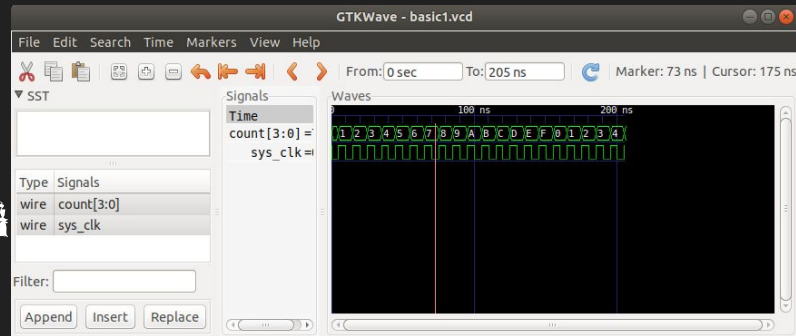
# Migen : A python HDL toolbox.

Migen.Genlib, a library with most of the base elements required to digital logic:

- Record. (Group signals together with direction)
- FSM. (Finite State Machine).
- Clock Domain Crossing.
- Memory.
- Instance. (reuse Verilog/VHDL)
- FIFO. (Synchronous, Asynchronous)
- ...

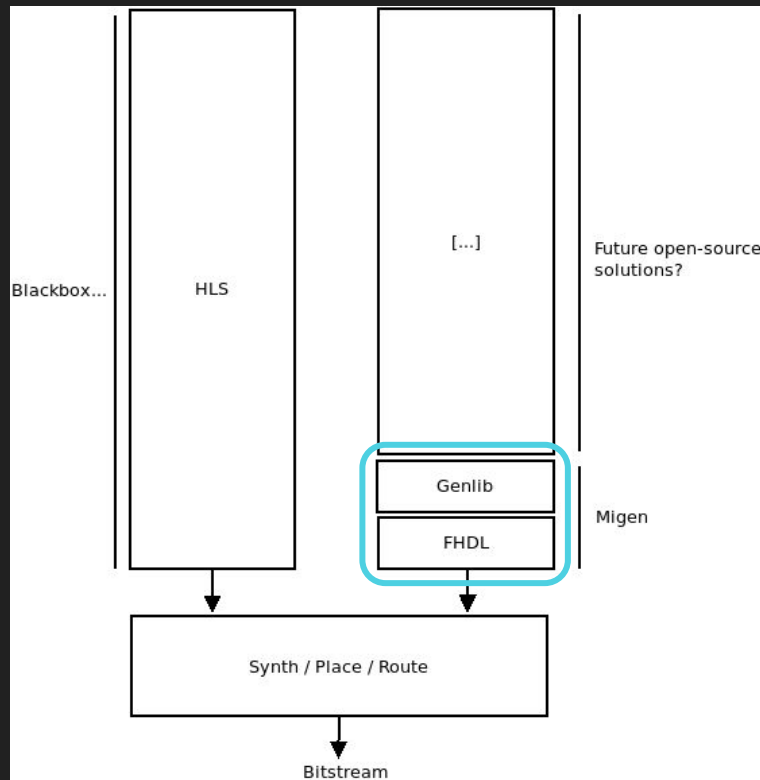
## Migen.Sim, a simulator in native Python:

```
1 from migen import *
2
3
4 # Our simple counter, which increments at every cycle.
5 class Counter(Module):
6     def __init__(self):
7         self.count = Signal(4)
8
9         # At each cycle, increase the value of the count signal.
10        # We do it with convertible/synthesizable FHDL code.
11        self.sync += self.count.eq(self.count + 1)
12
13
14 # Simply read the count signal and print it.
15 # The output is:
16 # Count: 0
17 # Count: 1
18 # Count: 2
19 # ...
20 def counter_test(dut):
21     for i in range(20):
22         print((yield dut.count)) # read and print
23         yield # next clock cycle
24     # simulation ends with this generator
25
26
27 if __name__ == "__main__":
28     dut = Counter()
29     run_simulation(dut, counter_test(dut), vcd_name="basic1.vcd")
30
```



```
florent@lab: ~/dev/mlabs/migen/examples/sim
File Edit View Search Terminal Help
florent@lab:~/dev/mlabs/migen/examples/sim$ python3 basic1.py
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
0
1
2
3
florent@lab:~/dev/mlabs/migen/examples/sim$
```

Migen != HLS  
(High Level Synthesis)  
but...

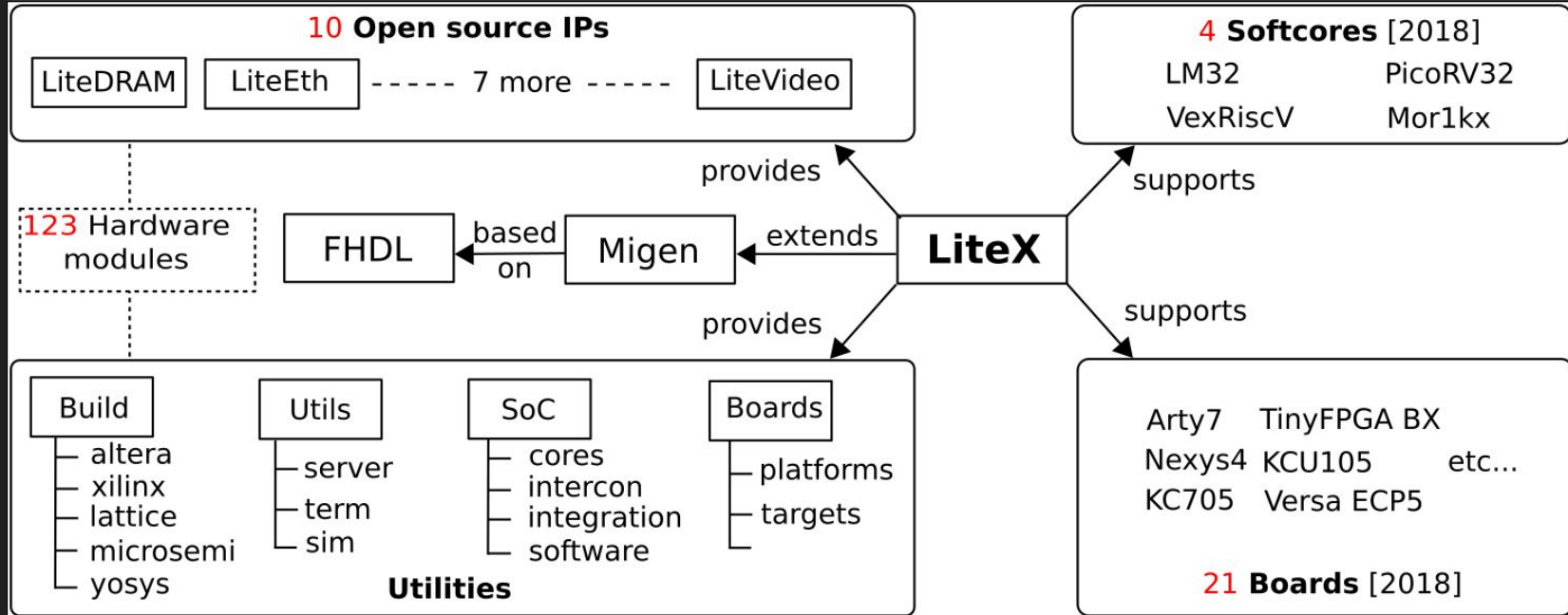


Could be seen as the  
lowest layer of an  
HLS.

Could be reused in the  
future to create an  
HLS but still full  
control on the  
generated logic.

# LiteX : Build your hardware, easily!

Provide the infrastructure to create complex SoCs with Python/Migen:



# LiteX : Build your hardware, easily!

Originated from a collaboration on MiSoC with **M-Labs**:



MiSoC still used successfully by M-Labs for physics equipment in



LiteX since 2015 focuses on embedded-systems and provides a wider collection of tools/cores.



# LiteX : Build your hardware, easily!

**LiteX's key features to simplify  
development / integration / debug:**



# LiteX : Build your hardware, easily!

## Scaling and portability

With open-source toolchain:  
Yosys/Trellis/Icestorm/Nextpnr  
(Even DDR3 / 1Gbps Ethernet!)

With vendor toolchains... for now :) Ultrascale

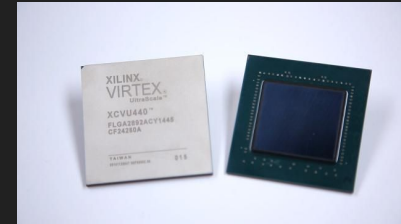
Cyclone V



Spartan6



7-Series



Stratix 10



2 Mluts / 40k \$

ECP5



iCE40



1 Kluts / ~few \$

Scale pretty well...

Used on very various FPGAs

LiteX: SoC builder and library

OSDA Workshop (2019), Florence, March 29

# LiteX : Build your hardware, easily!

## CPU's integration/support:

- LM32
- OpenRisc (Mor1kx)
- RISC-V (PicoRV32, Vexriscv, Minerva)
- ARM (\*through AXI on ZYNQ)
- X86 (\*through PCIe)



Switch CPU and in command line:

```
./arty.py --cpu_type=vexriscv --cpu_variant=lite
```

## Ecosystem of highly configurable and high performance cores:



SDR/DDR/DDR2/DDR3/DDR4  
(Up to > 80 Gbps with DDR4 - 64 bit)



SATA I/II/III: 1.5/3.0/6.0Gbps  
(400MB/s writes / 500MB/s reads)



Ethernet MAC/IP/UDP up to 1Gbps



SDCard up to 55MB/s



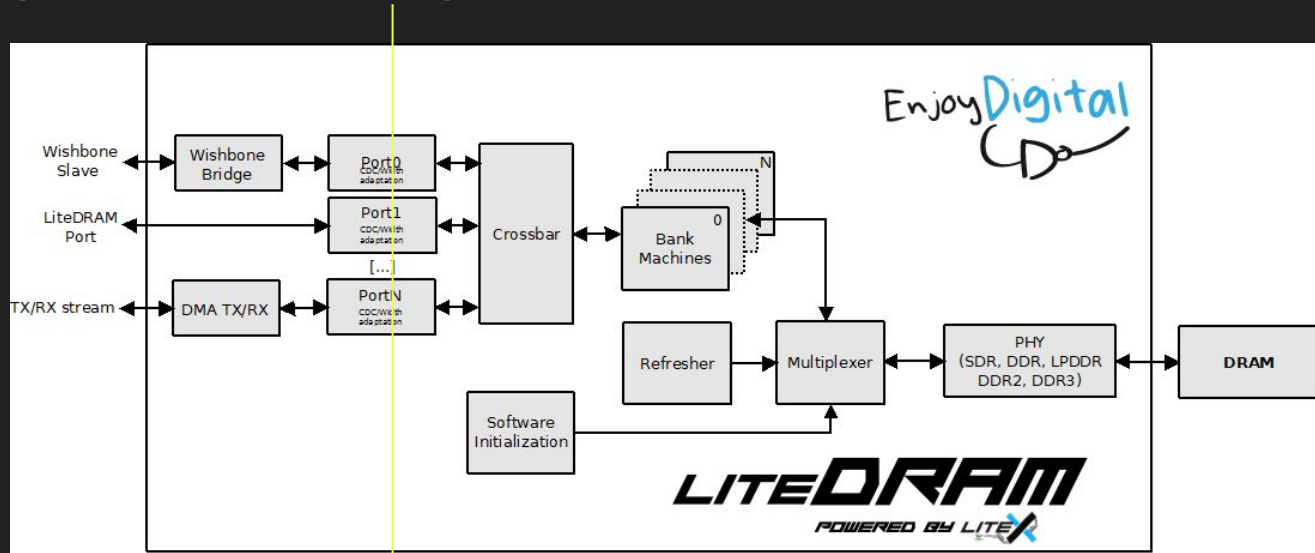
PCIe / DMA / Scatter-Gather  
(Up to Gen2 X4)



Embedded Logic Analyzer

Cores are designed to be used on various FPGAs, adding support for a new devices = just adding a new PHY.

## Constraint propagation / reduced duplication:



User defined constraints (specific to application)

Board specific constraints (automatically derived from DRAM config)

## Interactive simulation (emulation) with litex\_sim/Verilator:

```
SoC BIOS / CPU: LM32 / 1MHz
(c) Copyright 2012-2018 Enjoy-Digital
(c) Copyright 2007-2018 M-Labs Limited
Built Mar 12 2019 19:07:24

BIOS CRC passed (792d1b84)
Booting from serial...
Press Q or ESC to abort boot completely.
sL5DdSMmkekro
Timeout
No boot medium found
BIOS>
```

litex\_sim --cpu-type=lm32

```
SoC BIOS / CPU: VexRiscv / 1MHz
(c) Copyright 2012-2018 Enjoy-Digital
(c) Copyright 2007-2018 M-Labs Limited
Built Mar 12 2019 19:09:16

BIOS CRC passed (595b8b34)
Booting from serial...
Press Q or ESC to abort boot completely.
sL5DdSMmkekro
Timeout
No boot medium found
BIOS>
```

litex\_sim --cpu-type=vexriscv

```
SoC BIOS / CPU: MOR1KX / 1MHz
(c) Copyright 2012-2018 Enjoy-Digital
(c) Copyright 2007-2018 M-Labs Limited
Built Mar 12 2019 19:11:31

BIOS CRC passed (7bf6aa35)
Booting from serial...
Press Q or ESC to abort boot completely.
sL5DdSMmkekro
Timeout
No boot medium found
BIOS>
```

litex\_sim --cpu-type=or1k

<5s build time / ~1MHz execution speed, very useful for software dev.

Easy to add additional C++ plugins (JTAG, VGA, Ethernet, etc...)

LiteX: SoC builder and library  
OSDA Workshop (2019), Florence, March 29





## Automatic CSR registers collection / software header generation:

```
class SoCController(Module, AutoCSR):
    def __init__(self):
        self._reset = CSR()
        self._scratch = CSRStorage(32, reset=0x12345678)
        self._bus_errors = CSRStatus(32)

        ###

        # reset
        self.reset = Signal()
        self.comb += self.reset.eq(self._reset.re)

        # bus errors
        self.bus_error = Signal()
        bus_errors = Signal(32)
        self.sync += \
            If(bus_errors != (2**len(bus_errors)-1),
              If(self.bus_error,
                bus_errors.eq(bus_errors + 1)
              )
            )
        self.comb += self._bus_errors.status.eq(bus_errors)
```



```
#ifndef __GENERATED_CSR_H
#define __GENERATED_CSR_H
#include <stdint.h>
#ifdef CSR_ACCESSORS_DEFINED
extern void csr_write(uint8_t value, uint32_t addr);
extern uint8_t csr_read(uint32_t addr);
extern void csr_write(uint16_t value, uint32_t addr);
extern uint16_t csr_read(uint32_t addr);
extern void csr_write(uint32_t value, uint32_t addr);
extern uint32_t csr_read(uint32_t addr);
#else /* ! CSR_ACCESSORS_DEFINED */
#include <hw/common.h>
#endif /* ! CSR_ACCESSORS_DEFINED */

/* ctrl */
#define CSR_CTRL_BASE 0xe0000000
#define CSR_CTRL_RESET_ADDR 0xe0000000
#define CSR_CTRL_RESET_SIZE 1
static inline unsigned char ctrl_reset_read(void) {
    unsigned char r = csr_read(0xe0000000);
    return r;
}
static inline void ctrl_reset_write(unsigned char value) {
    csr_write(value, 0xe0000000);
}
#define CSR_CTRL_SCRATCH_ADDR 0xe0000004
#define CSR_CTRL_SCRATCH_SIZE 4
static inline unsigned int ctrl_scratch_read(void) {
    unsigned int r = csr_read(0xe0000004);
    r <<= 8;
    r |= csr_read(0xe0000008);
    r <<= 8;
    r |= csr_read(0xe000000c);
    r <<= 8;
    r |= csr_read(0xe0000010);
    return r;
}
static inline void ctrl_scratch_write(unsigned int value) {
    csr_write(value >> 24, 0xe0000004);
}
```

C header or CSV file that can be use by scripts/software

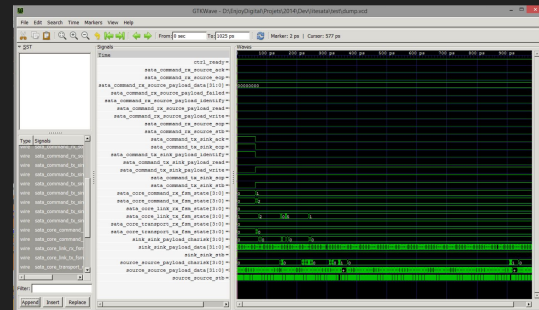
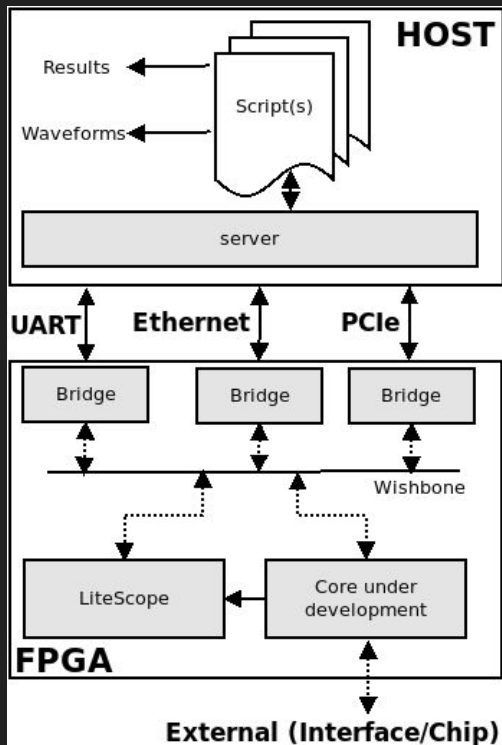
# LiteX : Build your hardware, easily!

Debug infrastructure with LiteX Server / LiteScope and the bridges:

```

1 #!/usr/bin/env python3
2
3 from litex import RemoteClient
4
5 wb = RemoteClient()
6 wb.open()
7
8 ###
9
10 def icap_send(addr, data):
11     wb.regs.icap_addr.write(addr)
12     wb.regs.icap_data.write(data)
13     wb.regs.icap_send.write(1)
14     while (wb.regs.icap_done.read() == 0):
15         pass
16
17 # iprog
18 icap_send(0x04, 0x0000000f)
19
20 ###
21
22 wb.close()
23
  
```

Python scripts to control/debug the hardware.



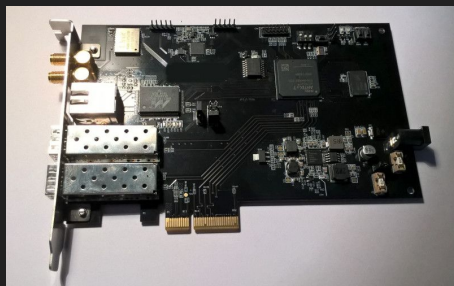
Speed up development, use the same tools for all devices!

# Systems using Migen/LiteX and Future!

# Systems using Migen/LiteX and Future!

Used for our commercial projects:

- Custom FPGA systems/designs for clients.
- Customizations of cores for clients to replace commercial/vendor's cores: LiteSATA on a 100Mpix camera, LiteDRAM on Ultrascale/DDR4 boards, etc...

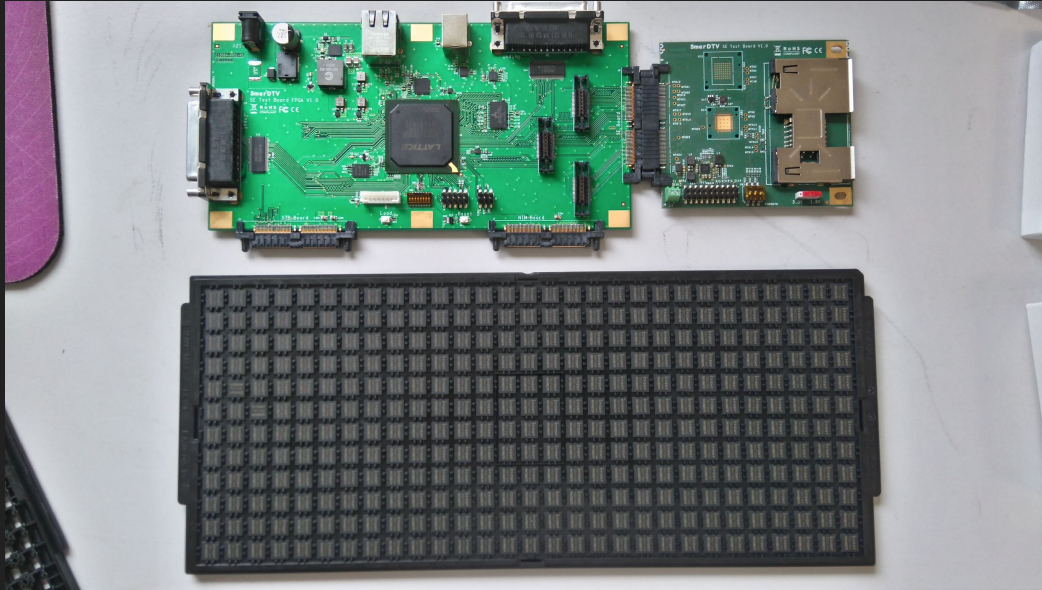


LiteX: SoC builder and library

OSDA Workshop (2019), Florence, March 29

# Systems using Migen/LiteX and Future!

- Custom ASIC characterization equipment



- ...

# Systems using Migen/LiteX and Future!


But also by others to create FPGA based systems:

**CROWD SUPPLY**

Imaging & Video  
Computers & Networking

## Numato Opsis: FPGA-based Open Video Platform

by Numato Lab  
An open platform for recording, routing, and manipulating HDMI and DisplayPort video signals.



00:16

\$40,430 raised  
of \$8,725 goal

463<sup>+</sup> Funded! Not Available


Opsis

**CROWD SUPPLY**

Imaging & Video  
Development Kits

## NeTV2

by Alphamax  
An open video development board in a PCI express form factor that supports overlaying content on encrypted video signals. Let's bring open video to the digital age!



00:35

\$84,135 raised  
of \$15,000 goal

560<sup>+</sup> Funded! Order Below


NeTV2

**CROWD SUPPLY**

Computers & Networking  
Open Hardware  
Development Kits

## Fomu

by Sutajin Kosugi  
An FPGA board that fits inside your USB port



01:09

\$26,976 raised  
of \$10,000 goal

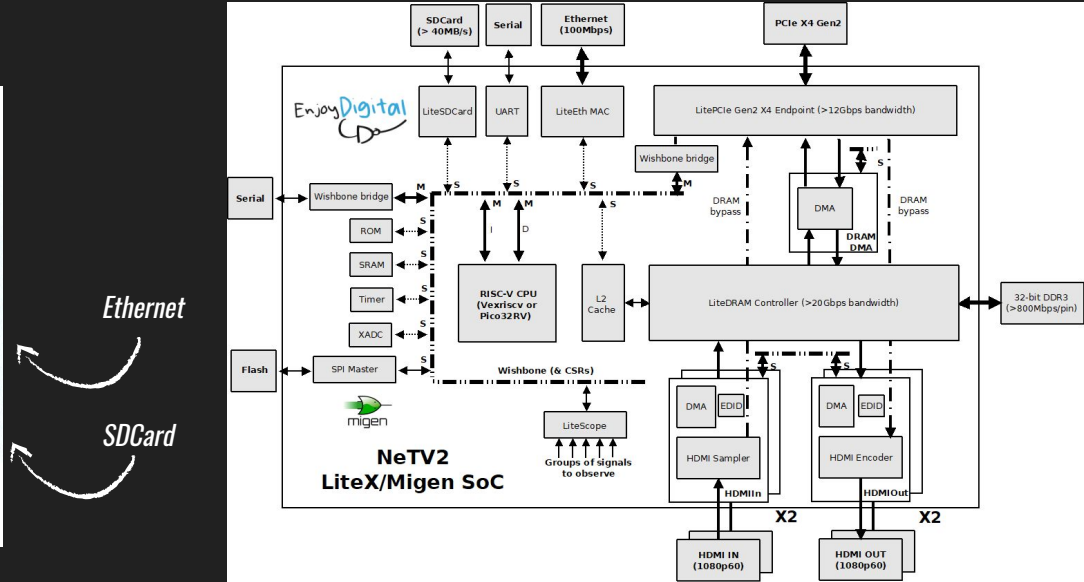
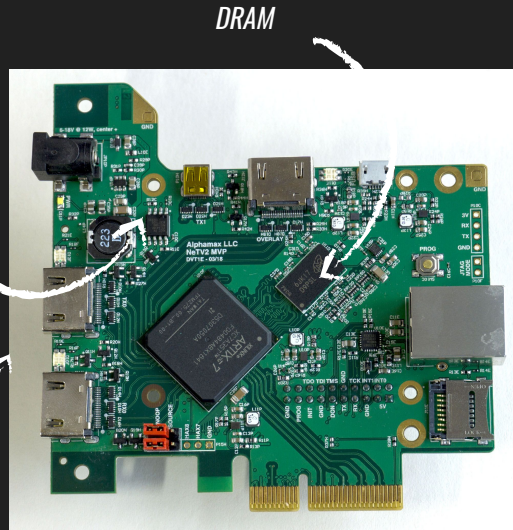
269<sup>+</sup> Funded! Order Below

Fomu



# Systems using Migen/LiteX and Future!

But also by others to create FPGA based systems:



NeTV2

# Systems using Migen/LiteX and Future!

Future: continue collaboration with others open-source enthusiasts to:

- Improve tools/cores/documentation/verification.
- Ease cores generation and reuse as standalone cores. (FuseSoC?).
- Add nMigen support (or switch to it) (almost 100% compatible, better doc/tests, adds formal verification).
- Improve current OS support/drivers and add drivers for others cores lite LiteSATA, LiteSDCard, LiteScope, LiteICLink, etc...
- Add others cores :)
- ...





You are a student and would like to work on this? (or know one)

Applications are now open and TimVideos/Symbiflow organizations can accept  
LiteX related projects.

The logo for SymbiFlow, consisting of the word "SymbiFlow" in a bold, purple, sans-serif font.



Migen/LiteX is not magic... but very powerful and could help you to be more efficient/productive.

Can be used to create full designs in Python or just as a tool for integration, core generation, simulation...

Want to try it or to help :) ? : <http://github.com/enjoy-digital/litex> (BSD License)

Get in touch on IRC (Freenode):

Migen: #m-labs

LiteX: #litex

# Questions ?