DatenLord

Agile Digital Chip Design with SpinalHDL and Cocotb

WanZheng Weng, DatenLord

About Us

DatenLord aims to break cloud barrier by deeply integrating hardware and software to build a unified storage-access mechanism to provide high-performance and secure storage support for applications across clouds.

We are using advanced HDLs, like Bluespec and SpinalHDL, and new verification tools, like Cocotb, to develop high-performance RDMA network interface card.

GitHub: https://github.com/datenlord;
Website: https://datenlord.github.io/;

DatenLord

CONTENTS

- Introduction -

Why we need agile hardware design?

- SpinalHDL -

A Scala package featured in efficient hardware design.

- Cocotb -

Python-based testbench environment for verifying RTL designs

DatenLord

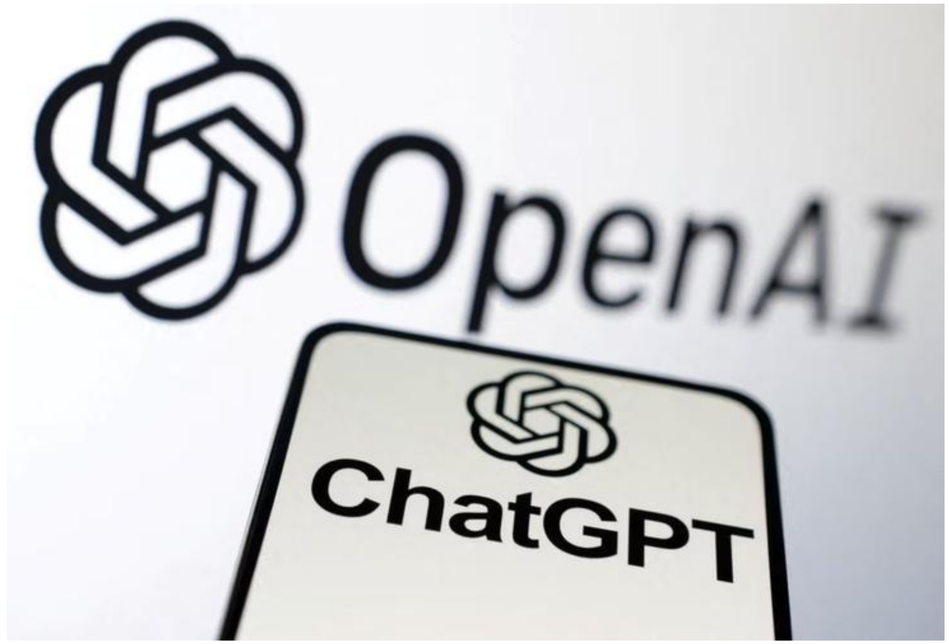
Introduction

- Why Agile Chip Design?

01

Why Agile Chip Design?

- Moore Law and Dennard Scaling slows down, hard to improve performance further;
- Emerging Applications, like LLM and Crypto, demand huge computation power;



Why Agile Chip Design?

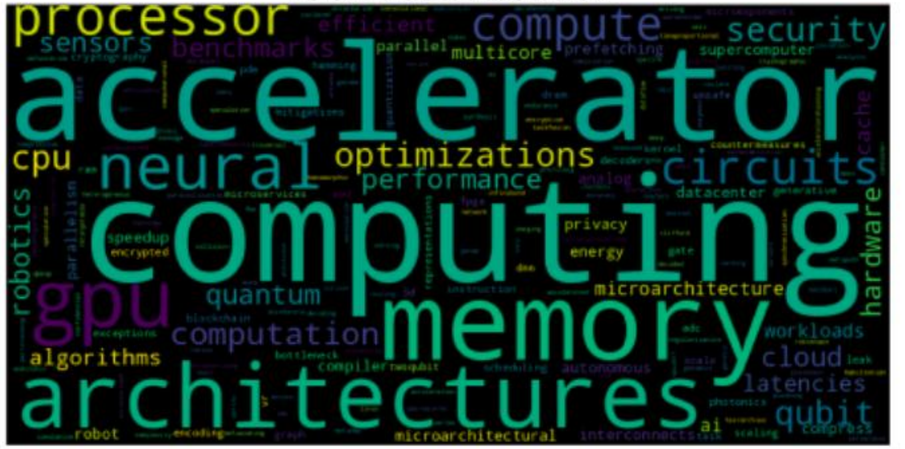
- Adopt domain-specific architecture to gain better performance-power ratio:
  - AI accelerator;
  - Smart NIC;
  - Cryptography;
  - Video coding/encoding;

- The architecture of chip becomes more diverse and design complexity surges;

- Democratize chip design for more companies and designers;

ISCA 1993 Abstract

ISCA 2023 Abstract



Design with SpinalHDL

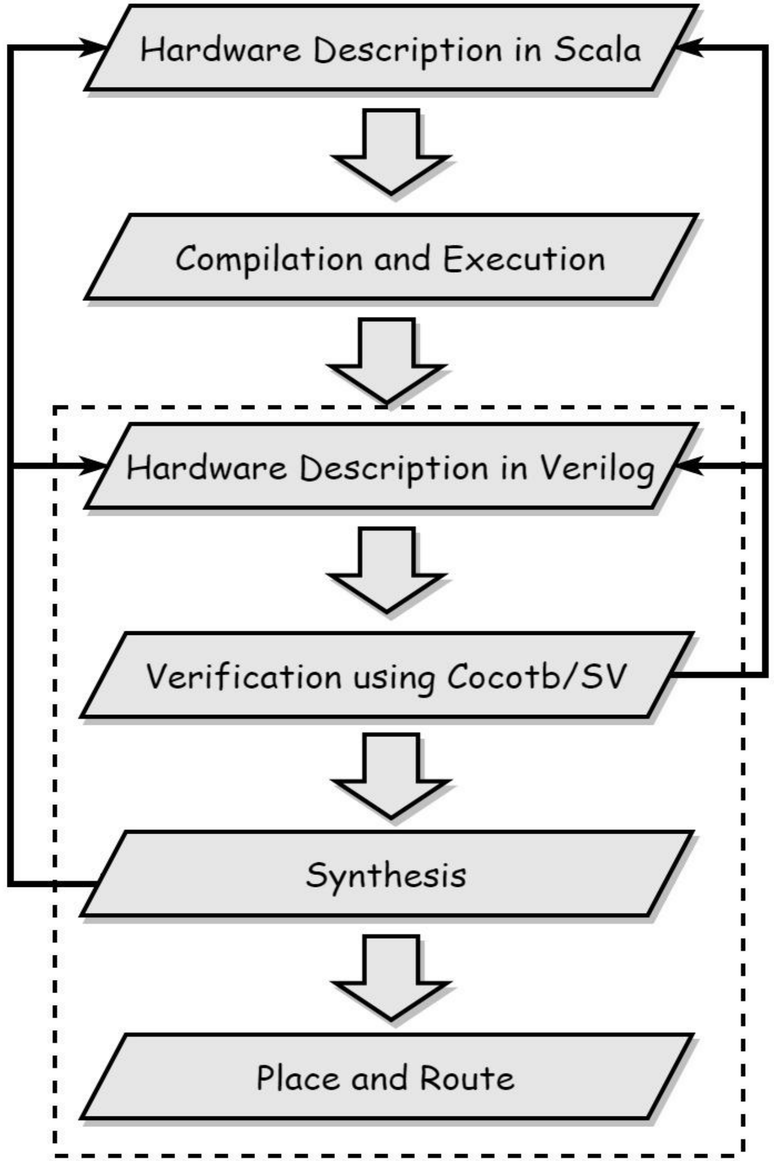
- What is SpinalHDL?
- Why SpinalHDL?

03

What's SpinalHDL?

- An efficient HCL(Hardware Construction Language) embedded in Scala;
- A Scala package providing APIs to describe circuits and generate Verilog codes;
- Problems with Verilog/VHDL:

  - Poor parameterization ability;
  - Loose error checking;
  - Tedious net connections;
  - ...



What's SpinalHDL?

- Not HLS;
- Same description granularity as traditional HDLs;
- SpinalHDL is a super set of Verilog/VHDL;

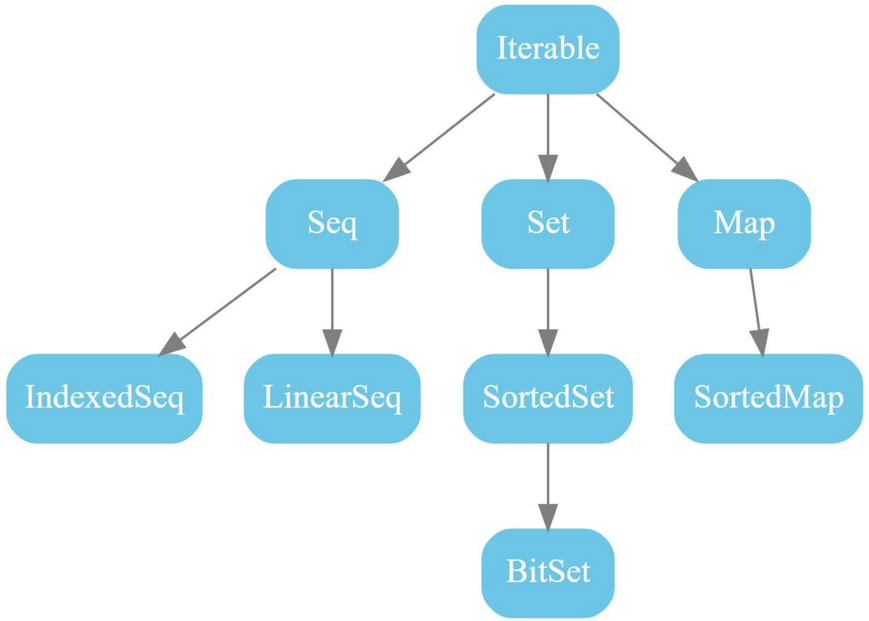
SpinalHDL elements	System Verilog element	category
in/out	input/output	I/O
Bundle	interface	I/O
ClockDomain	sensitivity list and always block	clock domain
signals(Bool,Bits,UInt,Sint and Vec)	wire/reg/logic and its modifiers	signals and connections
connection(:=)	assignment(=, <=)	signals and connections
when/elseWhen/others	if/else if/else	combinational
switch/is/default	case/default	combinational
logical operators	logical operators	combinational
relational operators	relational operators	combinational
arithmetic operators	arithmetic operators	combinational
registers	-	sequential
memories	-	sequential

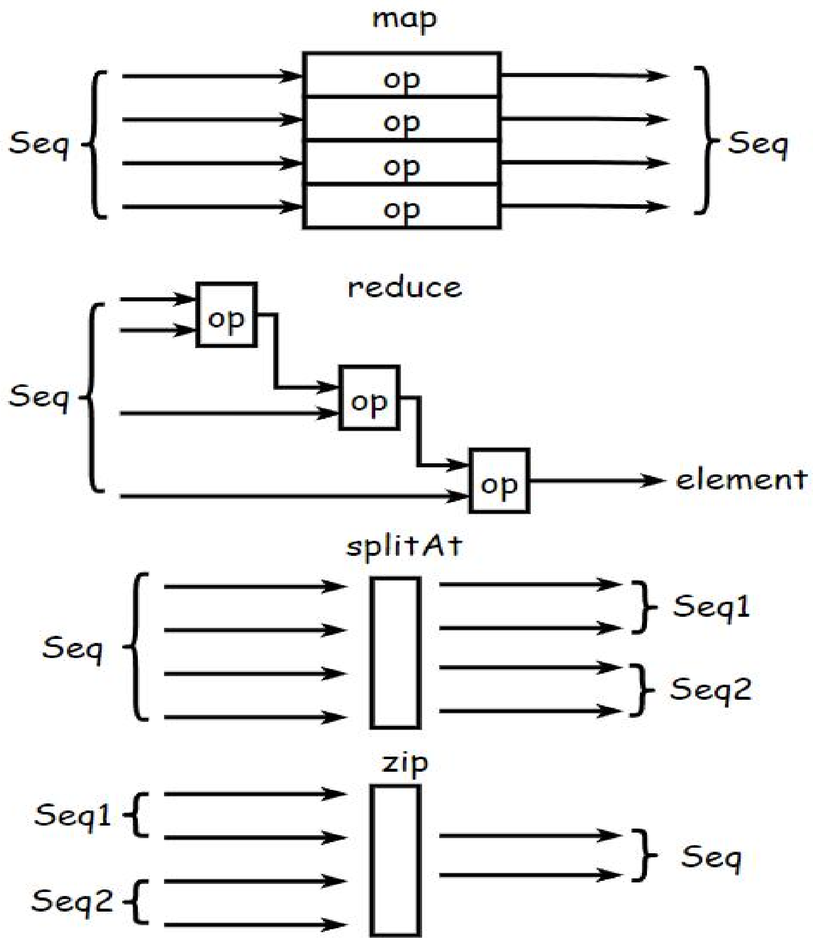
General Digital Circuit:

- Clock Domain;
- Combinational paths;
- Sequential parts;
- Signals and connections;
- I/O;

Why SpinalHDL? - Expressiveness

- Utilize advanced language features of Scala to describe circuits:
  - Object Oriented;
  - Recursion;
  - Collection types and methods;
  - Function programming;



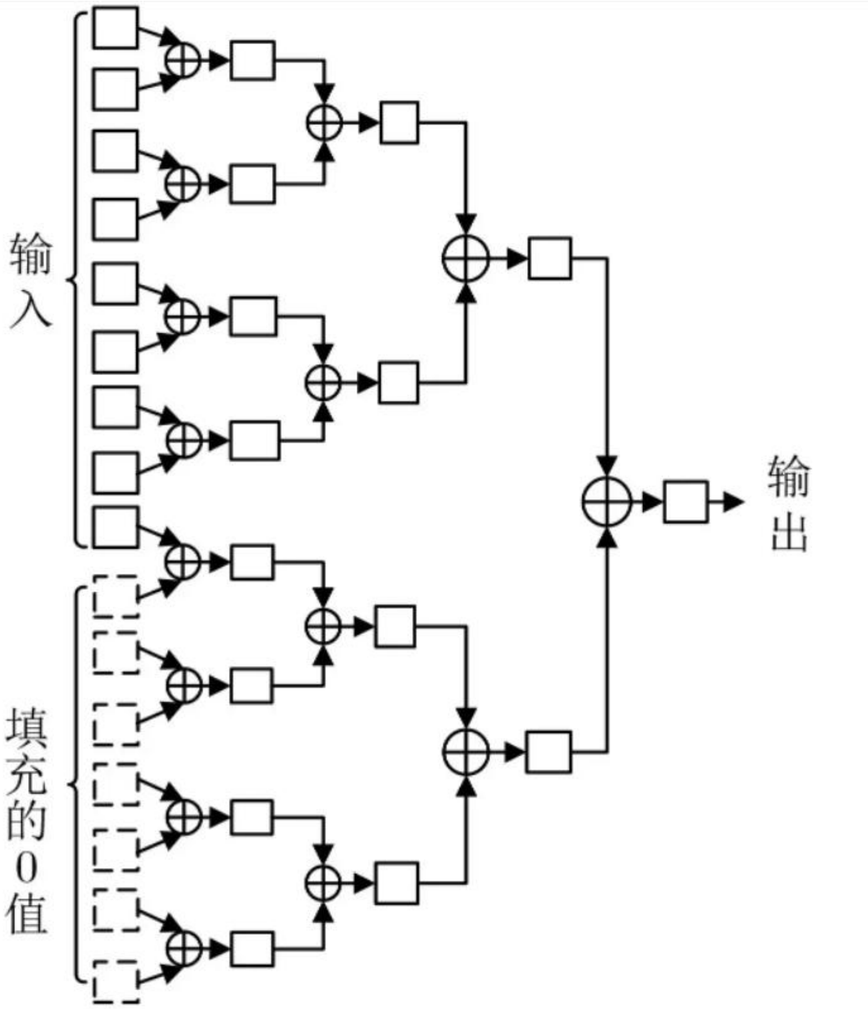


DatenLord

Why SpinalHDL? - Expressiveness

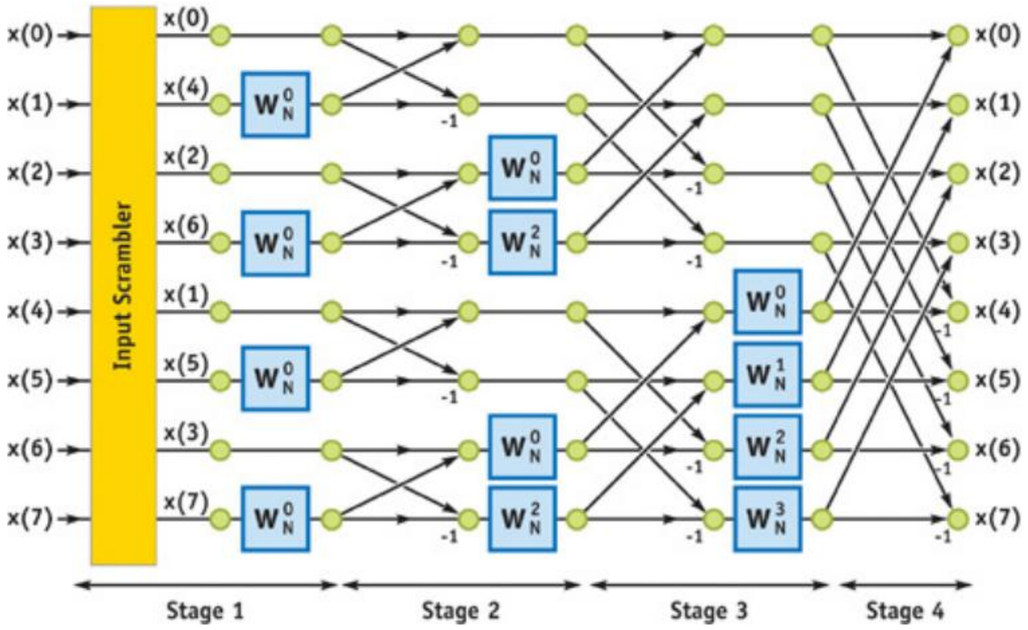
- A Simple Demo - Adder Tree:
  - Implemented through recursion
  - reduceBalanceTree method

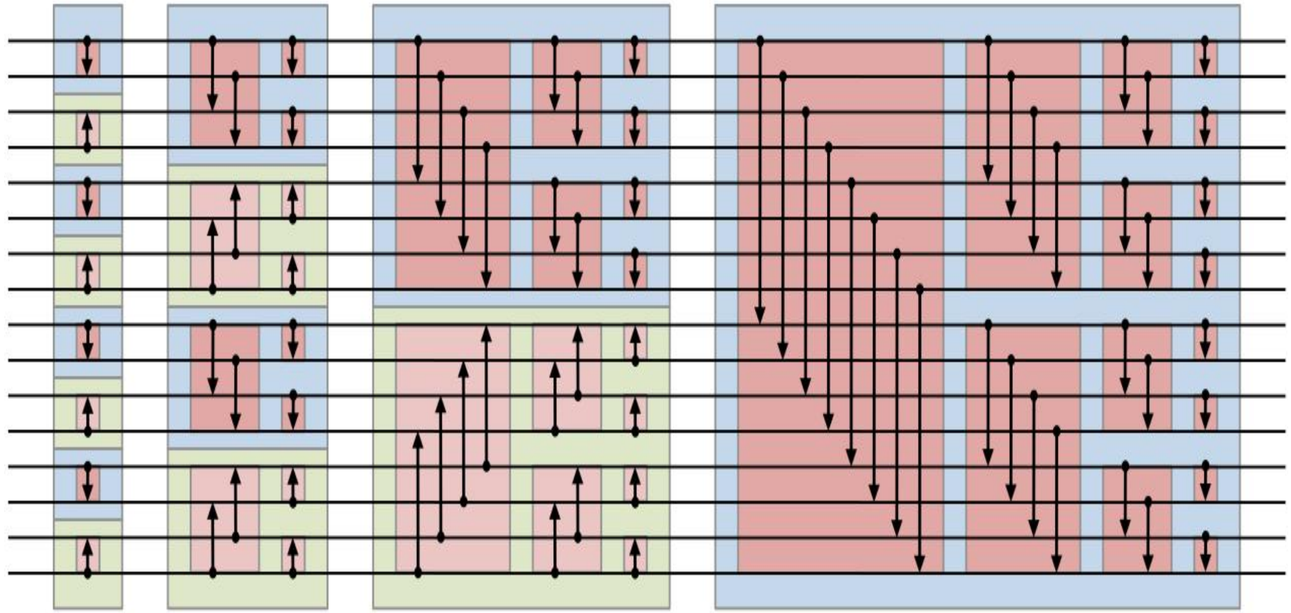
```
class AdderTree extends Component {
  val io = new Bundle {
    val inputs = in Vec(Bits(4 bits), 16)
    val output = out Bits(4 bits)
  }

  io.output := io.inputs.reduceBalanceTree(_ + _)
}
    if (inputs.length == 2) {
        inputs(0) + inputs(1)
    } else {
        val splitIdx = inputs.length/2
        val (firstHalf, secondHalf) = inputs.splitAt(splitIdx)
        adderTree(firstHalf) + adderTree(secondHalf)
    }
  }
  io.output := adderTree(io.inputs)
}
```



Why SpinalHDL? - Expressiveness

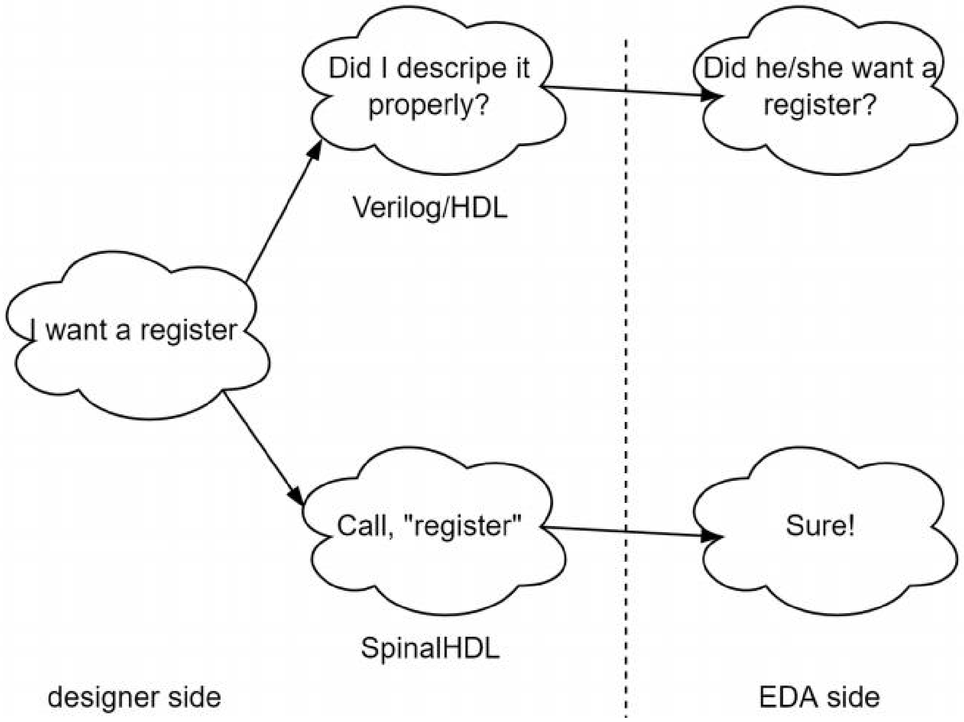
- Other Regular Circuits:
  - FFT(Fast Fourier Transformation)
  - Bitonic sorter





Why SpinalHDL? - Reliability

- More accurate model:
  - Registers in Verilog:
    - always block + sensitive list;
    - if else;
    - blocking/non-blocking assignments;
  - For SpinalHDL, reg is embedded in language level:
    - Reg/RegNext/RegInit;
    - ClockDomain;
  - SpinalHDL provides richer finer types for signal:
    - Bits / UInt / SInt / Vec



Why SpinalHDL? - Reliability

- Early DRCs:

  - Semantic problems could be found by the type system;
  - Problems introduced by implicit width expansion or reduction;
  - Latches could be found before generating Verilog/VHDL code;

- Separate Design and Simulation Elements:

  - Verilog can be divided into synthesizable and unsynthesizable;
  - It is confused to separate these two parts;
  - No needs to consider “synthesizable” in SpinalHDL;

Why SpinalHDL? - Reusability

- Object Oriented and Strong Parameterization:
  - Handle relations and restrictions between parameters;
  - Optimize architecture for a specific set of parameters;
- Community and tools:
  - Reuse Java packages;
  - Convenient build tools sbt/mill, intelligent IDE;
- Abundant Libraries of Circuit Components:
  - FIFO, RAM/ROM, Counter;
  - Bus: AXI/APB/AHB;
  - VexRiscv, NaxRiscv and SoC framework

```
val cpu = new VexRiscv(
  //Provide a configuration instance
  config = VexRiscvConfig(
    //Provide a list of plugins which will futher add t
    plugins = List(
      new IBusSimplePlugin(
        resetVector = 0x00000000l,
        cmdForkOnSecondStage = true,
        cmdForkPersistence  = true
      ),
      new DBusSimplePlugin(
        catchAddressMisaligned = false,
        catchAccessFault = false
      ),
      new DecoderSimplePlugin(
        catchIllegalInstruction = false
      ),
      new RegFilePlugin(
        regFileReadyKind = Plugin.SYNC,
        zeroBoot = true
      ),
      new IntAluPlugin,
      new SrcPlugin(
        separatedAddSub = false,
        executeInsertion = false
      ),
      new LightShifterPlugin,
      new HazardSimplePlugin(
        bypassExecute           = false,
        bypassMemory            = false,
        bypassWriteBack         = false,
        bypassWriteBackBuffer   = false
      ),
      new BranchPlugin(
        earlyBranch = false,
        catchAddressMisaligned = false
      ),
      new YamlPlugin("cpu0.yaml")
    )
  )
)
```

Verification with Cocotb

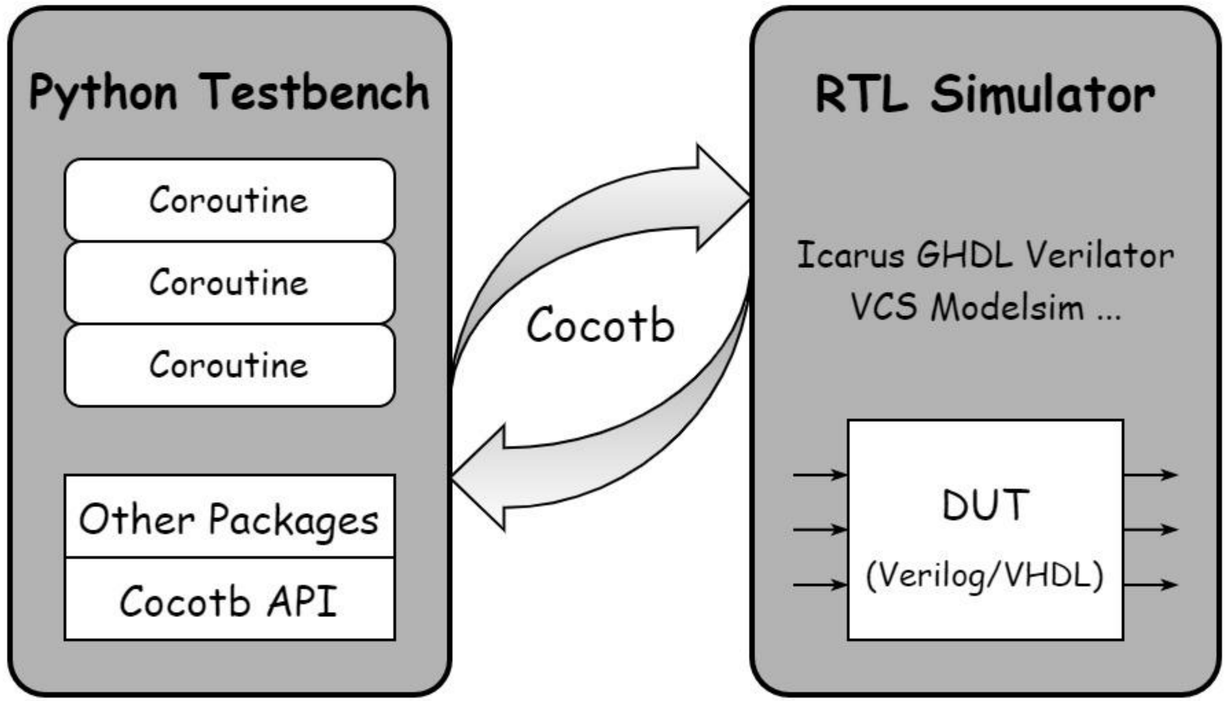
- What is Cocotb?
- Why Cocotb?
- A Specific Cocotb Demo

03

DatenLord

What's Cocotb?

- A bridge/interface between Python testbench and RTL simulator;
- Enable you write hardware testbench like software in Python;



Components in Cocotb Verification:

- RTL simulator providing VHPI/VPI/GPI;
- Hardware Design in Verilog/VHDL;
- Python testbench using Cocotb API;

Why Cocotb?

- In the perspective of language itself:

  - more productive and succinct than VHDL/SV;
  - abundant language features: OO, FP;
  - Easy to learn and master;
  - Popular language: easy to find engineer;

Example: Big Integer Multiplication in Crypto

- Simplest C++ implementation uses 40 lines;
- Only 1 line in Python;

```c
/* big_int_arith.h */
#ifndef BIG_INT_ARITH_H
#define BIG_INT_ARITH_H
#define MAX_SIZE 10
typedef struct big_int{
    int width;
    int numbers[MAX_SIZE];
} big_int;
big_int big_int_init(char input[]);
big_int big_int_add(big_int op1, big_int op2);
#endif
/* big_int_arith.c */
#include "big_int_arith.h"
#include <string.h>
big_int big_int_init(char input[]){
    big_int res;
    int len = strlen(input);
    res.width = len;
    if(len > MAX_SIZE) {res.width = MAX_SIZE;}
    for(int i=0; i<MAX_SIZE; i++){
        if(i < res.width){ res.numbers[i] = input[len-1-i] - '0';}
        else{ res.numbers[i] = 0;}
    }
    return res;
}

big_int big_int_add(big_int op1, big_int op2){
    int max = op1.width;
    if(max < op2.width) max = op2.width;
    for(int i; i< MAX_SIZE-1; i++){
        op1.numbers[i] += op2.numbers[i];
        op1.numbers[i+1] += op1.numbers[i]/10;
        op1.numbers[i] = op1.numbers[i]%10;
    }
    op1.numbers[MAX_SIZE-1] += op2.numbers[MAX_SIZE-1];
    op1.numbers[MAX_SIZE-1] = op1.numbers[MAX_SIZE-1]%10;
    if(max == MAX_SIZE){ op1.width = max;}
    else{op1.width = (op1.numbers[max]==0)?max:max+1;}
    return op1;
}
```

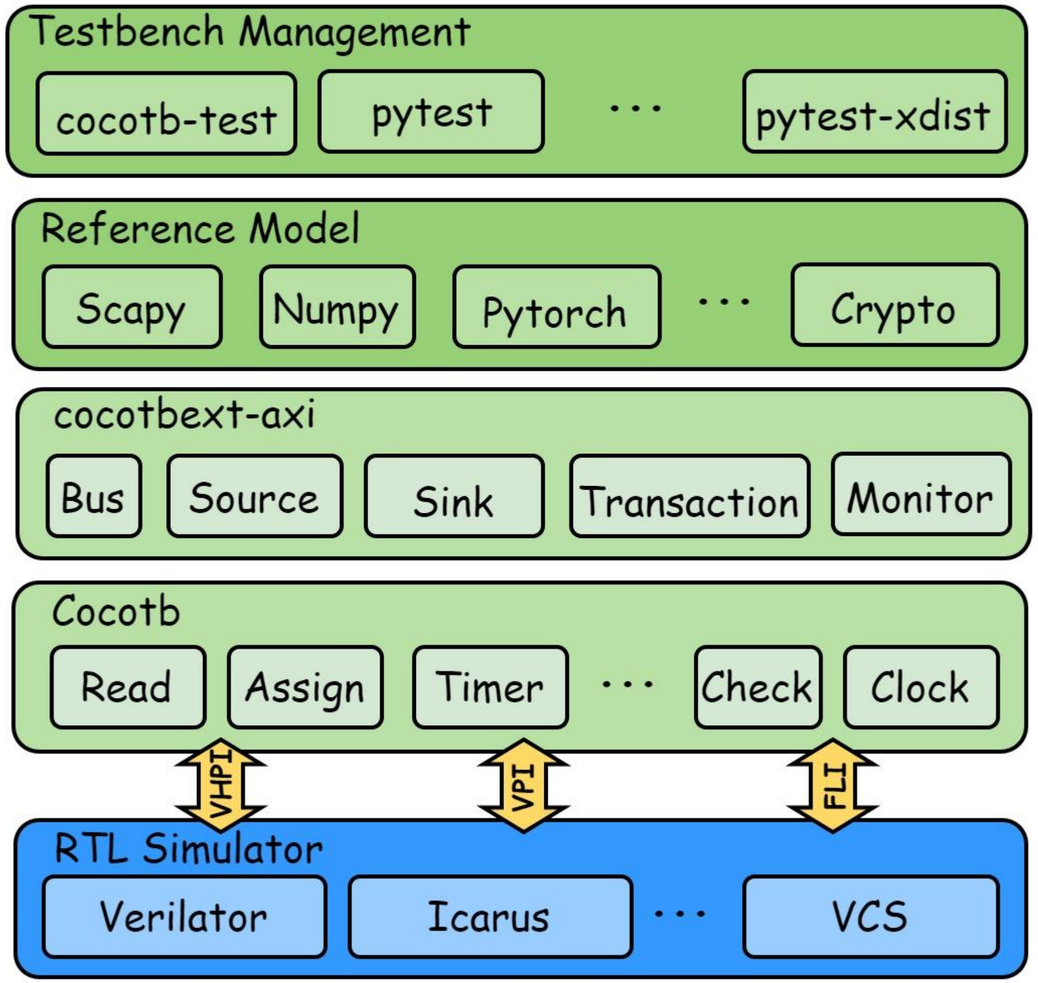
```python
# python implements addition of integers with any bit width
c = a + b
```

Why Cocotb?

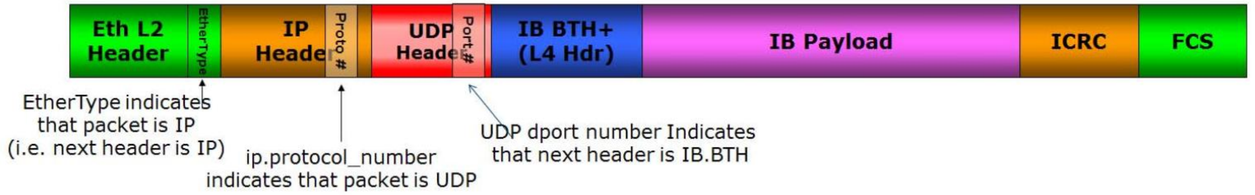
- In the perspective of Community:

  - Prosperous open-source community;
  - Abundant and diverse libraries;
  - Easy-to-use package management;
  - Reduce workload and ensure correctness;

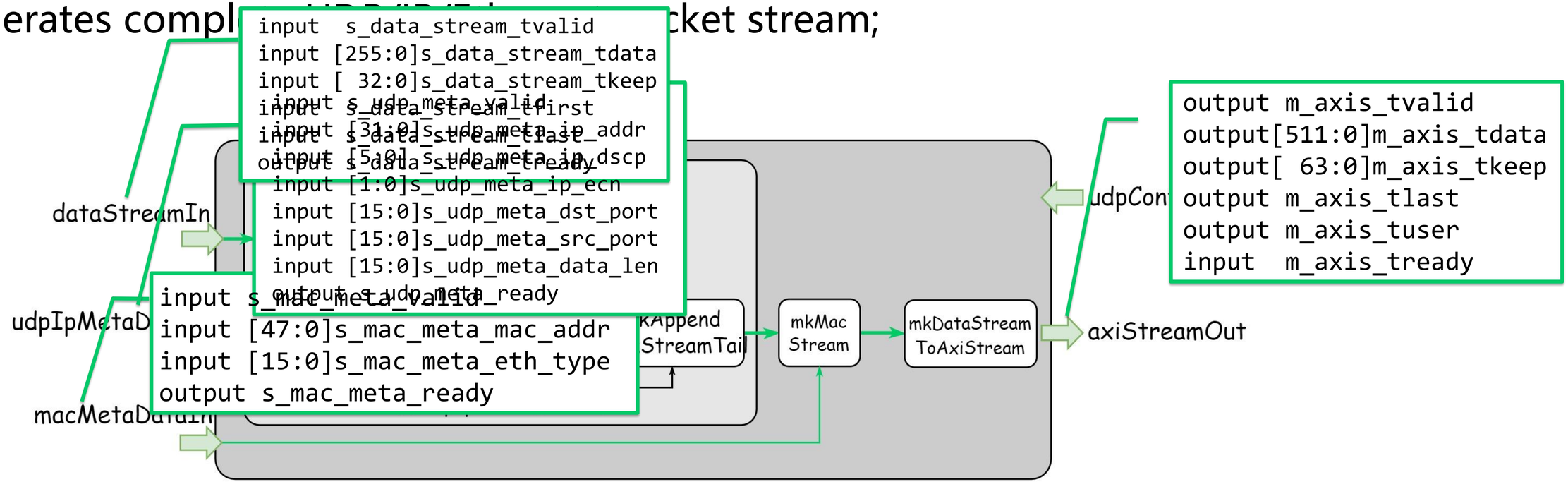
Package Stack for Cocotb-based Verification



A Specific Cocotb Demo

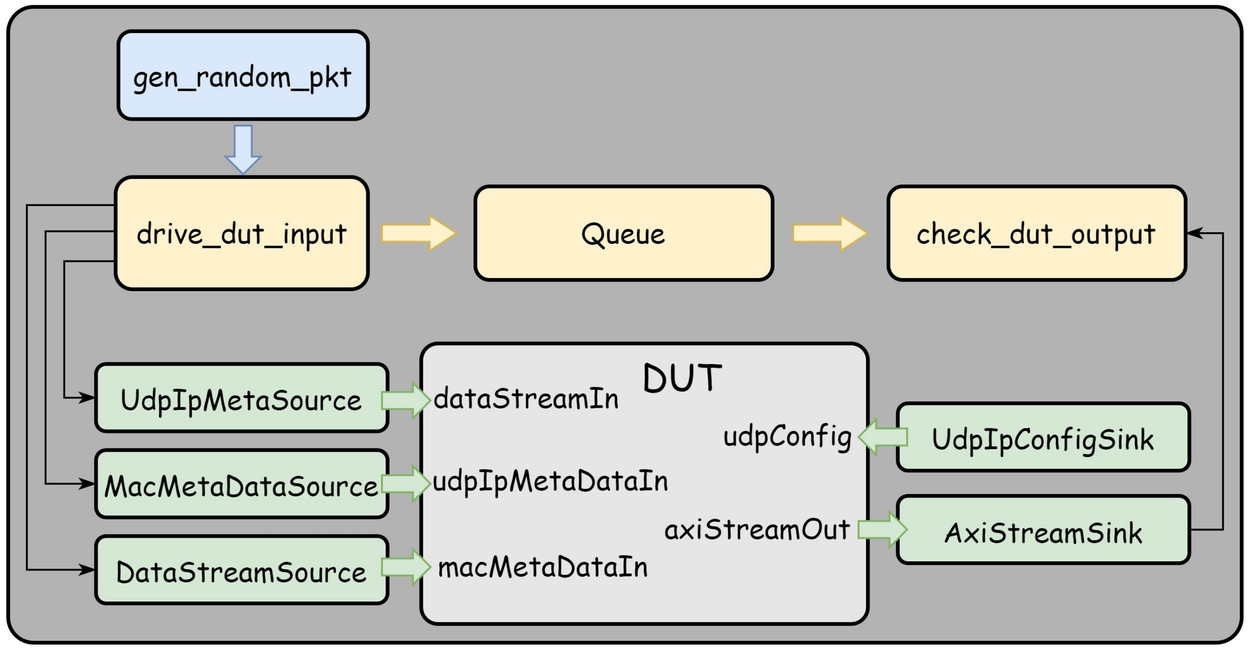


- Design Under Test:
  - Generate UDP/IP/Ethernet packet generator with support for RoCE;
  - Additional ICRC is appended in the tail of IP packet;
  - Takes in payload stream and header information;
  - Generates compl cket stream;



A Specific Cocotb Demo

- Testbench Architecture:
  - Sink/Source: abstract valid-ready ports into a python function using **cocotb-axi;**
  - gen_random_pkt: generate random UDP/IP/Ethernet packet using **scapy;**
  - drive_dut_input: drive input ports by calling corresponding function;
  - check_dut_output: receives and check reference packets;



A Specific Cocotb Demo

- Key Codes:

  - gen_random_pkt: generate random UDP/IP/Ethernet packet using **scapy;**
  - drive_dut_input: drive input ports by calling corresponding function;
  - check_dut_output: receives and check reference packets;

```
dut_I  await self.udp_ip_meta_src.send(udp_ip_meta)
ref_I  await self.mac_meta_src.send(mac_meta)
       await self.data_stream_src.send(frame)
if dI  self.ref_buffer.put(raw(packet))
    ref_packet_hex = ref_packet.hex("-")
    self.log.error(f"DUT Packet {case_idx}: {dut_packet_hex}")
    self.log.error(f"REF Packet {case_idx}: {ref_packet_hex}")
    Ether(dut_packet).show()
    Ether(ref_packet).show()
assert raw(dut_packet) == raw(ref_packet), f"Test Case {case_idx} Fail"
self.log.info(f"Pass testcase {case_idx}")
```

Related Links

- Datenlord GitHub: https://github.com/datenlord;
- SpinalHDL: https://github.com/SpinalHDL/SpinalHDL;
- Cocotb: https://www.cocotb.org/;
- blue-ethernet: https://github.com/wengwz/blue-ethernet;

Thanks