WINTECHCON

# Automation of Verification for Application Specific Instruction Set Processors

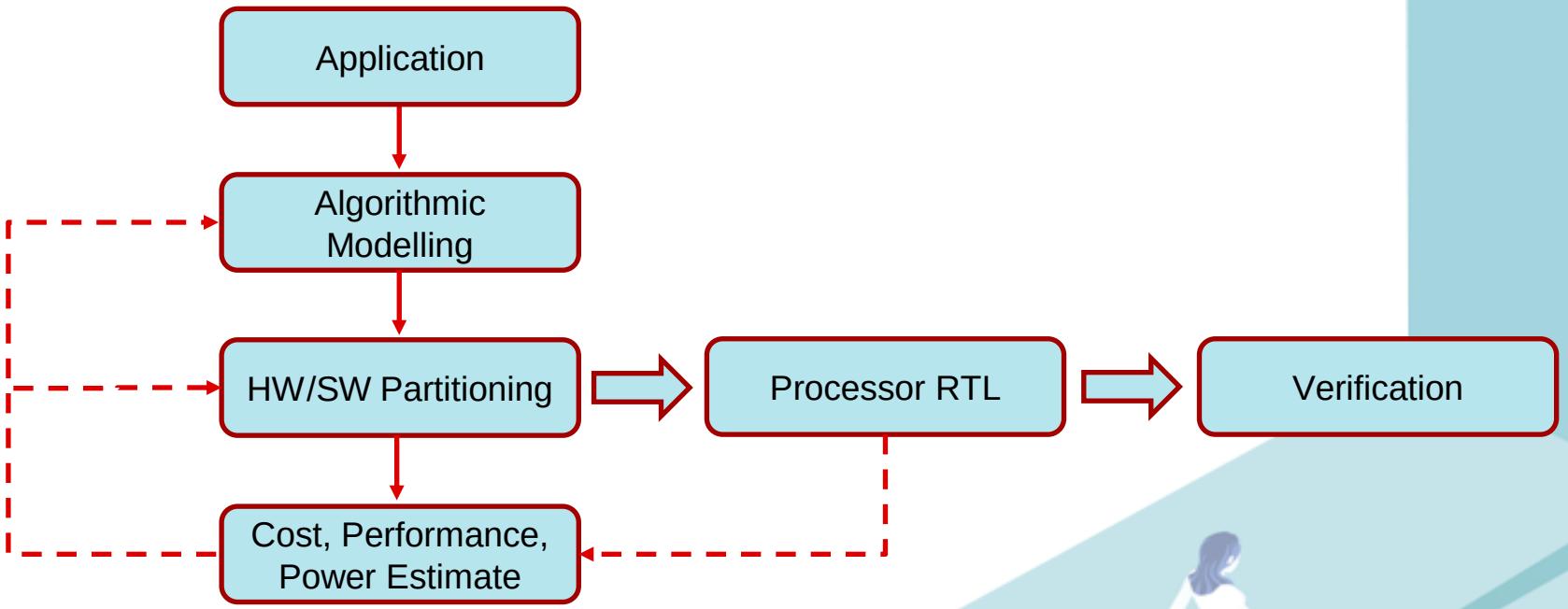
September 27, 2019

**Chaithanya Kolikipudi**

**Sreenivas Machavaram**
**Tal Milea**
**Parakalan Venkataraghavan**

**Intel Technologies India Pvt Ltd, Bangalore.**

# HW/SW Co-Design for ASIP Processor



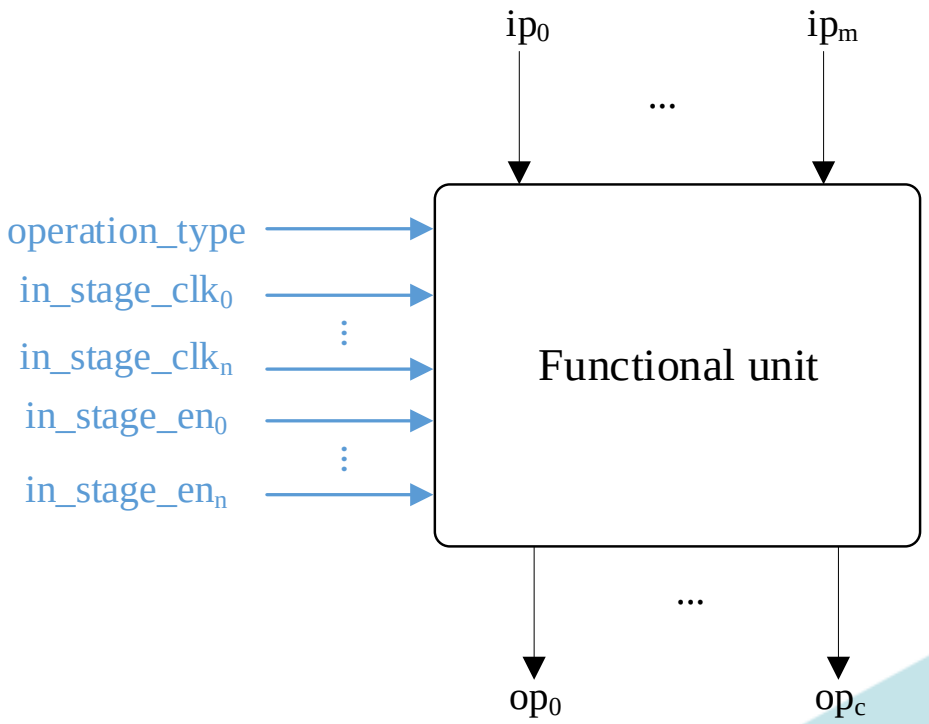
# Challenges in Instruction Verification

- Algorithmic optimizations
- C and RTL changes during HW & SW partitioning
- Hardware optimizations for area, timing, registers, pipelining and resource sharing, layout etc.
- New feature additions
- Providing quick design confidence

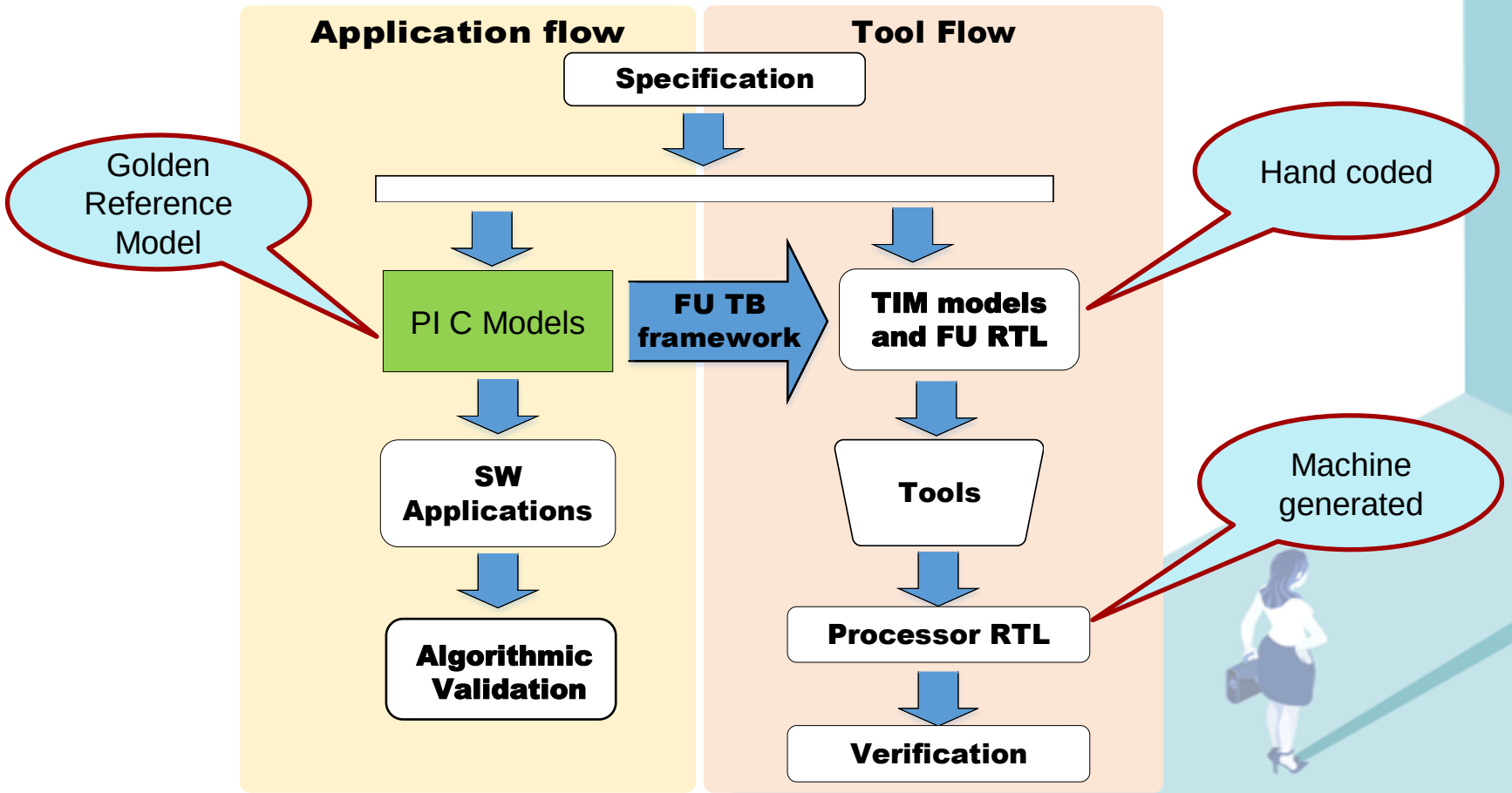
***Need for an Automated way to generate verification environment for instruction set verification !!!***

# Functional Unit

- Hardware implementation of one or many custom instructions.
- Multiple Functional Units build an ASIP.



# Custom Processor Design Flow



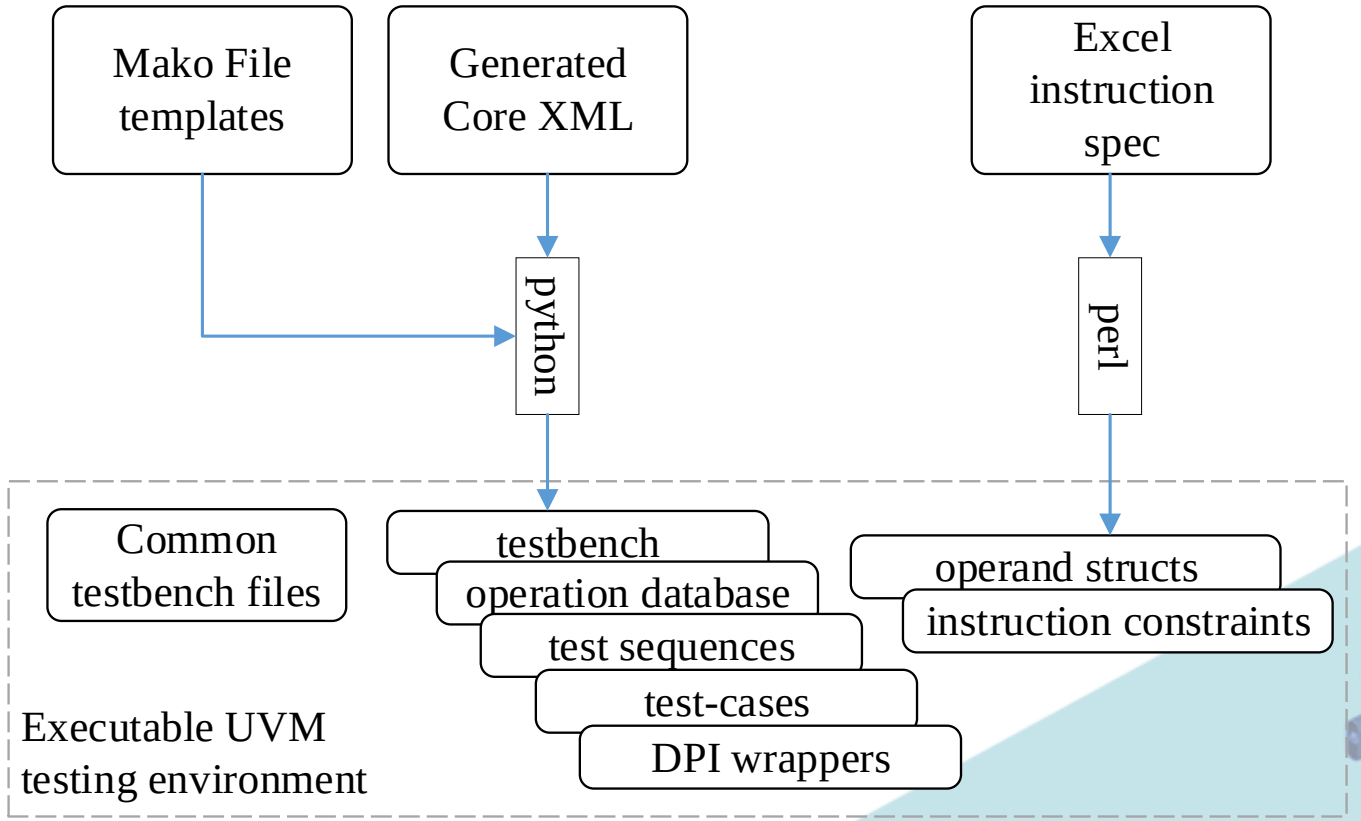
# Automatic Function Unit TB Goals

- Avoid manual test bench changes
- Check the coherency between all the design abstractions (C, TIM, and RTL)
- Consolidate the different resources into a single test bench
- Generic automation flow for any FU in any processor.

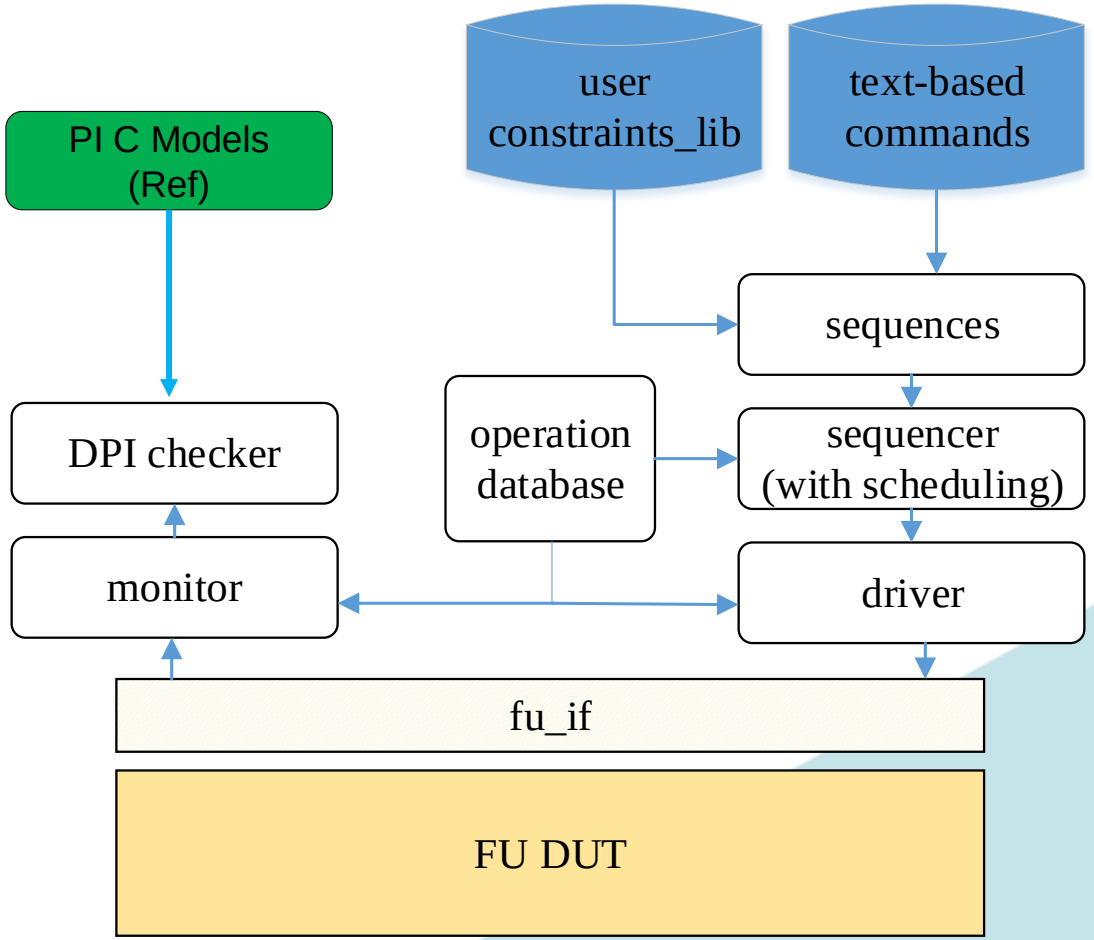
# Possible Changes in FU Implementation

- Number of instructions grouped per FU
- Number of operands per instruction
- Operand width of each instruction
- Time shape of operands and results
- Number of pipeline stages
- Resource sharing

# Automation of Verification Environment



# UVM FU TB Environment



# Generated Files : Operation Database

```
//optype 3 (op_add)
  op_db[3] = new();
  //input A
  op_db[3].operands[0].pin_id = IP0;
  op_db[3].operands[0].cycle = 0;
  //input B
  op_db[3].operands[1].pin_id = IP1;
  op_db[3].operands[1].cycle = 0;
  //output C
  op_db[3].results[0].pin_id = OP0;
  op_db[3].results[0].cycle = 3;
  //output R
  op_db[3].results[1].pin_id = OP1;
  op_db[3].results[1].cycle = 3;
```

# Generated Files : Instruction specification XLS

| A | B | D | E | F | G | H | |
|---|---|---|---|---|---|---|---|
| *S. NO* | *INSTRUCTION :* | *OPERAND* | *OPERA* | *TOTAL WIDTH* | *FIELD NAME* | *BIT POS* | *DESCRIPTION* |
| 1 | ADD | OPERAND[0] | A | [255:0] | size | [3:0] | b00:16-bit elements<br>b01: 32-bit elements<br>b10: reserved<br>b11: reserved |
| | | | | | negate | [4:4] | ANY:When set, perform the negation of data |
| | | | | | mask | [7:5] | ANY:Mask bits for data |
| | | | | | RSVD | [31:8] | ANY:RSVD bits |
| | | | | | Value0 | [255:32] | ANY:Data |
| | | OPERAND[1] | B | [255:0] | size | [3:0] | b00: reserved.<br>b01: 16-bit elements.<br>b10: reserved<br>b11: reserved |
| | | | | | negate | [4:4] | ANY:When set, perform the negation of data |
| | | | | | mask | [7:5] | ANY:Mask bits for data |
| | | | | | RSVD | [31:8] | ANY:RSVD bits |
| | | | | | Value0 | [255:32] | ANY:Data |
| | | RESULT[0] | R | [255:0] | R | [4095:0] | R = OP_ADD (A, B); |
| | | | | | | | |

# Generated Files : Structs and Constraints

```
constraint c_op_add{
 if(opcode == OP_ADD){
   operand0.op_add_ctrlA.rsvd1 == 0;
   operand0.op_add_ctrlA.rsvd0 == 0;
   operand0.op_add_ctrlA.size inside {0,1};

   operand1.op_add_ctrlB.rsvd1 == 0;
   operand1.op_add_ctrlB.rsvd0 == 0;
   operand1.op_add_ctrlB.size inside{1};
 }
}

typedef struct packed{
  bit[MAX_OPERAND_WIDTH-1:256] rsvd1;
  bit[255:32] value0;
  bit[31:8] rsvd0;
  bit[7:5] mask;
  bit negate;
  bit[3:0] size;
} op_add_ctrlA_s;

typedef struct packed{
  bit[MAX_OPERAND_WIDTH-1:256] rsvd1;
  bit[255:32] value0;
  bit[31:8] rsvd0;
  bit[7:5] mask;
  bit negate;
  bit[3:0] size;
} op_add_ctrlB_s;
```

# Generated Files : High Level Tests and Vector files

```
//syntax:
//<optype> <#instructions> <in0>, …, <inN>
//Examples:
OP_ADD 0x2 RANDOM 0xfffe0aaa
OP_ADD 0x2 NEG_MAX SAME_AS_PREV
OP_FRM 0x4 ZERO RANDOM_POS RANDOM 0x11111111
```

```
//syntax:

//<Op_type> <IP0> … <IPn> <OP0> <Op_Valid0> … <OPn> <OP_Validn>

//Example:

0xf 0x11111111 0x22222222 0x00000000 0x0 0x333333333 0x000000 0x0 0x0 0x0 0x0 0x4444444444 0x1
0x555555555 0x1

0x2 0xaabb8800 0x77777777 0xffffffff 0x0110110 0xabababab 0xcccccccc 0x0 0x0 0x11112222 0x1 0x555555555
0x1
```

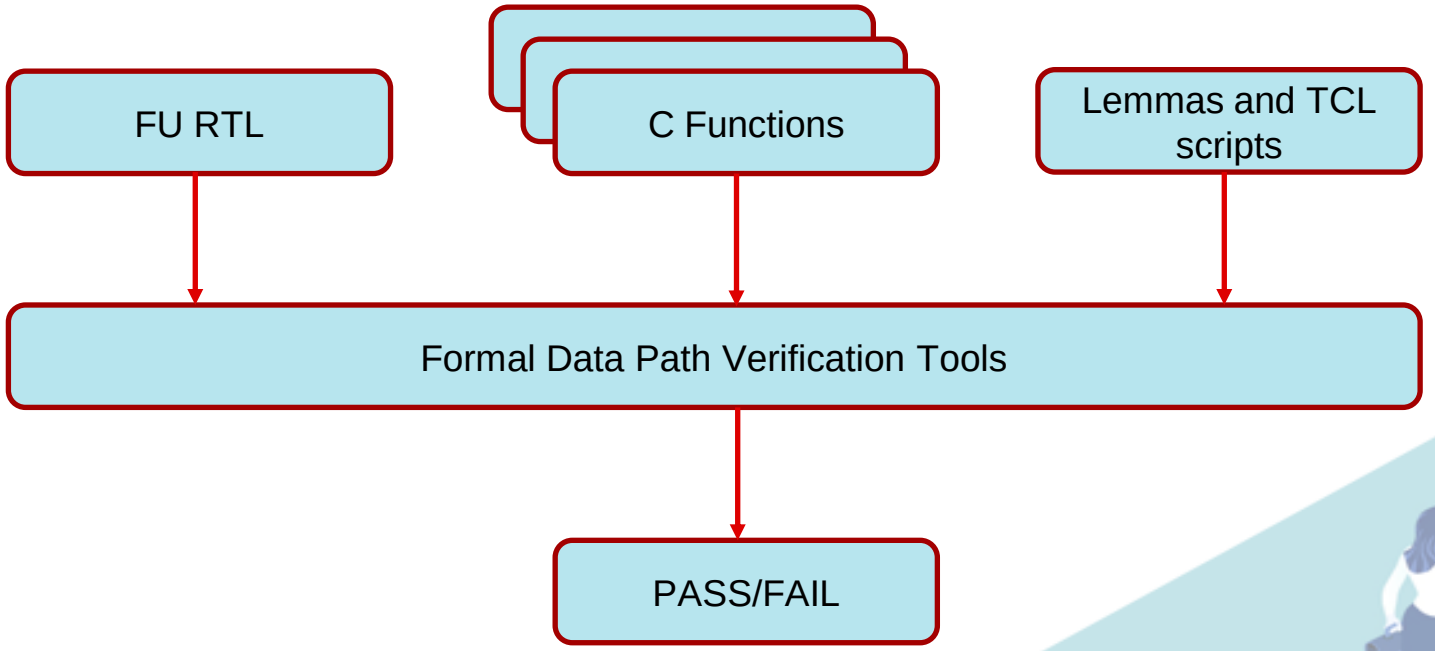
# DPI Wrappers

```
void op_add(
 const svOpenArrayHandle A_in,
 const svOpenArrayHandle B_in,
 const svOpenArrayHandle C_out,
 const svOpenArrayHandle R_out
){
   _t__int256 A;
   _t__int256 B;
   _t__int32 C;
   _t__int256 R;

   dpi_copy_256_sv2c(A_in, &A, SV2C);
   dpi_copy_256_sv2c(B_in, &B, SV2C);

   R = op_add_R(A, B);
   C = op_add_C(A, B);

   dpi_copy_256_sv2c(R_out, &R, C2SV);
   dpi_copy_256_sv2c(C_out, &C, C2SV);
}
```

# Results

- Early Design Confidence
  - 30 min to complete verification of an FU with 8 logical operations.
  - 10min to start validating changes in an arithmetic block which has partition, slicing and pipeline changes
  - Algorithmic vector regression validated in an hour
- Bugs identified
  - saturation in an arithmetic-logic FU
  - control path bugs
  - C/RTL non-compliance with specification
  - Time shape issues
  - Opcode mismatches

# Future Work : Formal Verification for Data Path Coverage



# Conclusions

- Automation of complete test suite and environment
- Bridges HW , C and TIM models
- Easy to use
- Ensures quick turn around time

# References


- J. Leijten, G. Burns, J. Huisken, E. Waterlander, and A. van Wel, “A massively parallel reconfigurable accelerator,” in Proc. International Symposium on System-on-Chip, pp. 165-168, November 2003
- J. Leijten and M. Lindwer, “Multiprocessing template for media applications,” in Proc. Eighth IEEE International Symposium on Multimedia, pp. 475-480, December 2006
- A. S. Nery, N. Nedjah, F. M. Franca, L. Jozwiak, and H. Corporaal, “Automatic complex instruction identification for efficient application mapping onto application-specific instruction set processors,” IEEE 5th Latin American Symposium on Circuits and Systems, pp. 1-4, 2014
- Universal Verification Methodology (UVM): www.accellera.org/downloads/standards/uvm
- Mako Templates for Python: www.makotemplates.org


# Thank you