ILA Package User Manual

Last updated: 09/08/2016

The first part of this document will provide instruction on how to install the ILA package. The second part is the specification of ILA syntax and semantics.

Package Requirements:

python: version 2.7 or above

Z3: version 4.4.2

boost: version 1.60.0

Files:

Z3 package: https://github.com/Z3Prover/z3

boost package: http://www.boost.org/

ILA package: See released package

Installation

Boost:

Please see the boost documents for installation instructions. You may need to export the following system paths to corresponding paths:

1. PATH
2. LD_LIBRARY_PATH
3. LIBRARY_PATH
4. C_INCLUDE_PATH
5. CPLUS_INCLUDE_PATH

Z3:

Please see the Z3 documents for installation instructions.

ILA:

- *cd [root path]/synthesis/libcpp*
- *vim Jamroot*

  Comment/delete the line:

  "*testing.make-test run-pyd : ila test/export_verilog.py : : test_export_verilog ;*"

- *bjam*
- *export PYTHONPATH=[root path]/synthesis/libcpp/build/:$PYTHONPATH*

ILA Syntax and Semantics

ILA library provides you an interface to create abstraction in python environment. All expressions represent a node in the abstract syntax tree.

To use the library, you have to import the package.

```
import ila
```

The below will provide the syntax and semantics of ILA operators.

Abstraction components:

- **Abstraction**
  - Semantics: Create a container for abstraction.
  - Syntax: ila.Abstraction([name])
  - Example: m = ila.Abstraction('soc')
- **add_microabstraction**
  - Semantics: Create a micro-abstraction container with the specified active condition.
  - Syntax: [abstraction].add_microabstraction([name], [active condition])
  - Example: um = m.add_microabstraction('aes_compute', state != 0)
- **get_microabstraction**
  - Semantics: Get the symbolic link of the micro-abstraction.
  - Syntax: [abstraction].get_microabstraction([name])
  - Example: um = m.get_microabstraction('aes_compute')
- **connect_microabstraction**
  - Semantics: Connect the micro-abstraction to the macro-abstraction.
  - Syntax: [abstraction].connect_microabstraction([name], [micro-abstraction])
  - Example: m.connect_microabstraction('aes_compute', um)
- **inp**
  - Semantics: Create a input variable.
  - Syntax: [abstraction].inp([name], [bit length])
  - Example: mode = m.inp('mode', 32)
- **reg**
  - Semantics: Create a register (bitvectpr variable).
  - Syntax: [abstraction].reg([name], [bit length])
  - Example: eax = m.reg('eax', 32)
- **getreg**
  - Semantics: Get the symbolic link of the register with the specified name.
  - Syntax: [abstraction].getreg([name])
  - Example: eax = m.getreg('eax')
- **bit**
  - Semantics: Create a bit (boolean variable).
  - Syntax: [abstraction].bit([name])
  - Example: flag = m.bit('flag')
- **getbit**
  - Semantics: Get the symbolic link of the bit with the specified name.
  - Syntax: [abstraction].getbit([name])
  - Example: flag = m.getbit('flag')

- **mem**
  - Semantics: Create a memory variable.
  - Syntax: [abstraction].mem([name], [address bit length], [data bit length])
  - Example: sram = m.mem('sram', 32, 8)
- **getmem**
  - Semantics: Get the symbolic link of the memory with the specified name.
  - Syntax: [abstraction].getmem([name])
  - Example: sram = m.getmem('sram')
- **fun**
  - Semantics: Create an un-interpreted function.
  - Syntax: [abstraction].fun([name], [output bit length], [list of inputs bit length])
  - Example: aes = m.fun('aes', 32, [128, 128])
- **getfun**
  - Semantics: Get the symbolic link of the function with the specified name.
  - Syntax: [abstraction].getfun([name])
  - Example: aes = m.getfun('aes')
- **const**
  - Semantics: Create a constant bitvector variable..
  - Syntax: [abstraction].const([value], [bitlength])
  - Example: ZERO = m.const(0x0, 32)
- **bool**
  - Semantics: Create a constant Boolean variable.
  - Syntax: [abstraction].bool([value])
  - Example: TOP = m.bool(True)
- **MemValues**
  - Semantics: Create a memory value variable.
  - Syntax: ila.MemValues([address bit length], [data bit length], [default value])
  - Example: ZMEM = ila.MemValues(32, 8, 0x0)
  - Note: Memory variable can only be initialized by MemValues.
- **set_init**
  - Semantics: Set initial value for the reg/bit/mem with the specified name.
  - Syntax: [abstraction].set_init([name], [initial value])
  - Example: m.set_init('eax', m.const(0x0, 32))
    m.set_init('flag', m.bool(True))
    m.set_init('sram', ila.MemValues(32, 8, 0x0))
- **set_ipred**
  - Semantics: Set initial constraints for the reg/bit/mem with the specified name.
  - Syntax: [abstraction].set_ipred([name], [initial condition])
  - Example: m.set_ipred('eax', eax >= 0x2)
    m.set_ipred('flag', flag ^ flag2)
    m.set_ipred('sram', ila.load(sram, 0x0) == 0x0)
- **set_next**
  - Semantics: Set next state function for the reg/bit/mem with the specified name.
  - Syntax: [abstraction].set_next([name], [next state function])
  - Example: m.set_next('eax', eax + ebx)
    m.set_next('flag', ~flag)
    m.set_next('sram', ila.store(sram, 0x0, 0x0))

- **get_next**
  - Semantics: Get the next state function for the reg/bit/mem with the specified name.
  - Syntax: [abstraction].get_next([name])
  - Example: eax_nxt = m.get_next(‘eax’)
    flag_nxt = m.get_next(‘flag’)
    sram_nxt = m.get_next(‘sram’)

Abstraction operation:

- **areEqual**
  - Semantics: Functionally compare two expressions.
  - Syntax: [abstraction].areEqual([node1], [node2])
  - Example: eq = m.areEqual(2*eax+2*ebx, 2*(eax+ebx))
- **add_assumption**
  - Semantics: Add assumptions to the abstraction for synthesis.
  - Syntax: [abstraction].add_assumption([assumption expression])
  - Example: m.add_assumption(eip <= 0x0000ffff)
  - Note: The synthesis algorithm will only use distinguishing input that satisfy the assumptions.
- **get_all_assumptions**
  - Semantics: Get all the assumptions that has been added.
  - Syntax: [abstraction].get_all_assumptions()
  - Example: ass_list = m.get_all_assumptions()
- **fetch_expr**
  - Semantics: Define fetch expression.
  - Syntax: [abstraction].fetch_expr = [fetch expression]
  - Example: m.fetch_expr = rom[eip]
- **fetch_valid**
  - Semantics: Define when the fetch is valid.
  - Syntax: [abstraction].fetch_valid = [fetch valid constraint]
  - Example: m.fetch_valid = (state == 0)
- **decode_exprs**
  - Semantics: Define decode expressions.
  - Syntax: [abstraction].decode_exprs = [list of decode expressions]
  - Example: m.decode_exprs = [(rom[eip] == i) for i in xrange(0, 0x100)]
  - Note: The synthesis algorithm will try to find out the correct configuration for each decode expression in the decode_exprs. That is to say, try to find distinguishing input that satisfy the underlying decode expression.
- **synthesize**
  - Semantics: Synthesize the given state by using the given simulator.
  - Syntax: [abstraction].synthesize([name], [simulator])
  - Example: m.synthesize(‘eax’, sim)
- **exportOne**
  - Semantics: Export one expression to the specified file.
  - Syntax: [abstraction].exportOne([expression], [filename])
  - Example: m.exportOne(eax_nxt, ‘res/eax_nxt.ila’)
  - Note: The exported expression must not contain synthesis primitives.

- **exportAll**
  - Semantics: Export the whole abstraction to the specified file.
  - Syntax: [abstraction].exportAll([filename])
  - Example: m.exportAll('res/soc.ila')
  - Note: The exported expression must not contain synthesis primitives.
- **importOne**
  - Semantics: Import one expression from the specified file.
  - Syntax: [abstraction].importOne([filename])
  - Example: eax_nxt = m.importOne('res/eax_nxt.ila')
- **importAll**
  - Semantics: Import the whole abstraction from the specified file.
  - Syntax: [abstraction].importAll([filename])
  - Example: m.importAll('res/eax_nxt.ila')
- **generateSim**
  - Semantics: Generate a C++ simulator for the abstraction to the specified file.
  - Syntax: [abstraction].generateSim([filename])
  - Example: m.generateSim('soc_sim.cpp')
- **generateSimToDir**
  - Semantics: Generate a C++ simulator for the abstraction to the specified directory.
  - Syntax: [abstraction].generateSimToDir([director])
  - Example: m.generateSimToDir('soc_sim')
  - Note: The simulator will be partitioned into several files. Large abstraction is recommended to use this function rather than generateSim.
- **bmc**
  - Semantics: Bounded model check two states are functionally equivalent.
  - Syntax: ila.bmc([step1], [abstraction1], [state1], [step2], [abstraction2], [state2])
  - Example: ila.bmc(10, soc, eax, 1, golden, eax)

Synthesis Primitives:

- **choice**
  - Semantics: Define a set of possible values.
  - Syntax: ila.choice([name], [list of possible values])
  - Example: src = ila.choice('src_choice', [eax, ebx, ecx, m.const(0x0, 32)])
- **inrange**
  - Semantics: Define a range of possible values.
  - Syntax: ila.inrange([name], [lower bound], [upper bound])
  - Example: offset = ila.inrange('offset', m.const(0x0, 32), m.const(0xffff, 32))
- **readslice**
  - Semantics: Define the bitvector to read from and the bit length to read. The read can start from any bit.
  - Syntax: ila.readslice([name], [bitvector], [bitlength])
  - Example: cnt = ila.readslice('counter', ecx, 8)
  - Note: The above example is the same as choice within [ecx[i+7, i] for i in xrange(0, 24)].

- **readchunk**
  - Semantics: Define the bitvector to read from and the bit length to read. The read can only start from the multiple of the bit length.
  - Syntax: ila.readchunk([name], [bitvector], [bitlength])
  - Example: cnt = ila.readchunk('counter', ecx, 8)
  - Note: The above example is the same as choice within [ecx[i*8+7, i*8] for i in xrange(0, 4)]. The width of the bitvector should be the multiple of the input bitlength.
- **writeslice**
  - Semantics: Define the bitvector to write to and the value to write. The write can start from any bit.
  - Syntax: ila.writeslice([name], [destination bitvector], [value bitvector])
  - Example: ila.writeslice('write_eax', eax, ebx[7:0])
- **writechunk**
  - Semantics: Define the bitvector to write to and the value to write. The write can only start from the multiple of the bit length.
  - Syntax: ila.writechunk([name], [destination bitvector], [value bitvector])
  - Example: ila.writechunk('write_eax', eax, ebx[7:0])
  - Note: The width of the destination should be the multiple of the width of value.

Expression Operations:

- **~**
  - Semantics: Invert the bit/bits, bitwise logical negation.
  - Syntax: ~[expression]
  - Example: res = ~flag
- **-**
  - Semantics: Numerically negate the bitvector.
  - Syntax: -[expression]
  - Example: res = -eax
- **&**
  - Semantics: Logical bitwise AND.
  - Syntax: [expression/immediate] & [expression/immediate]
  - Example: res = eax & 0xff
    res = flag1 & flag2
- **|**
  - Semantics: Logical bitwise OR.
  - Syntax: [expression/immediate] | [expression/immediate]
  - Example: res = eax | ebx
    res = m.bool(False) | flag
- **^**
  - Semantics: Logical bitwise XOR.
  - Syntax: [expression/immediate] ^ [expression/immediate]
  - Example: res = eax ^ m.const(0xffffffff, 32)
    res = flag1 ^ flag2

- +
  - Semantics: Addition.
  - Syntax: [expression/immediate] + [expression/immediate]
  - Example: res = eax + ebx
    res = eax + 0x1a
- –
  - Semantics: Subtraction.
  - Syntax: [expression/immediate] – [expression/immediate]
  - Example: res = eax – ebx
    res = eax – 0x1a
- *
  - Semantics: Multiplication.
  - Syntax: [expression/immediate] * [expression/immediate]
  - Example: res = eax * (ebx + ecx)
    res = eax * 0x2
- /
  - Semantics: Signed division.
  - Syntax: [expression/immediate] * [expression/immediate]
  - Example: res = 0xff / ebx
    res = eax / 0x2
- %
  - Semantics: Unsigned modulo.
  - Syntax: [expression/immediate] % [expression/immediate]
  - Example: res = eax % (ebx + edx)
    res = 0xff % eax
- <<
  - Semantics: Left shift.
  - Syntax: [expression] << [expression/immediate]
  - Example: res = eax << ebx
    res = eax << 0x2
- >>
  - Semantics: Logical right shift.
  - Syntax: [expression] >> [expression/immediate]
  - Example: res = eax >> (ebx + ecx)
    res = eax >> 0x3
- ==
  - Semantics: Equal.
  - Syntax: [expression] == [expression/immediate]
  - Example: res = eax == ebx
    res = eax == 0xff
- !=
  - Semantics: Not equal.
  - Syntax: [expression] != [expression/immediate]
  - Example: res = eax != ebx
    res = flag1 != flag2

- <
    - Semantics: Unsigned less than.
    - Syntax: [expression] < [expression/immediate]
    - Example: res = eax < ebx
- >
    - Semantics: Unsigned greater than.
    - Syntax: [expression] > [expression/immediate]
    - Example: res = eax > 0xff
- <=
    - Semantics: Unsigned less than or equal to.
    - Syntax: [expression] <= [expression/immediate]
    - Example: res = eax <= ebx
- >=
    - Semantics: Unsigned greater than or equal to.
    - Syntax: [expression] >= [expression/immediate]
    - Example: res = eax >= ebx
- **[]**
    - Semantics: Load one data from memory.
    - Syntax: *memory*[*expression/immediate*]
    - Example: instruction = rom[eip]
      instruction = rom[0xff]
- **[:]**
    - Semantics: Slice from bitvector.
    - Syntax: *expression*[*immediate* : *immediate*]
    - Example: res = eax[7:0]
- **load**
    - Semantics: Load data from memory.
    - Syntax: ila.load([memory], [address])
    - Example: var = ila.load(sram, ptr)
      var = ila.load(sram, m.const(0xff10, 32))
- **loadblk**
    - Semantics: Load long data from the memory in little endian.
    - Syntax: ila.loadblk([memory], [address], [chunk number])
    - Example: instr_32 = ila.loadblk(rom, eip, 0x4)
    - Note: Chunk number is the number of data unit to be read. In the above example, if the memory has 8-bit data, the instr_32 will have 32 bits.
- **loadblk_big**
    - Semantics: Load long data from the memory in big endian.
    - Syntax: ila.loadblk_big([memory], [address], [chunk number])
    - Example: instr_32 = ila.loadblk_big(rom, eip, 0x4)
    - Note: Chunk number is the number of data unit to be read. In the above example, if the memory has 8-bit data, the instr_32 will have 32 bits.
- **store**
    - Semantics: Store data to memory.
    - Syntax: ila.store([memory], [address], [data])
    - Example: ila.store(sram, ptr, m.const(0xff10, 32))

- **storeblk**
  - Semantics: Store long data to the memory in little endian.
  - Syntax: ila.storeblk([memory], [address], [data])
  - Example: ila.storeblk(sram, ptr, longdata)
- **storeblk_big**
  - Semantics: Store long data to the memory in big endian.
  - Syntax: ila.storeblk_big([memory], [address], [data])
  - Example: ila.storeblk_big(sram, ptr, longdata)
- **nand**
  - Semantics: Logical bitwise NAND.
  - Syntax: ila.nand([expression], [expression])
  - Example: res = ila.nand(eax, ebx + ecx)
- **nor**
  - Semantics: Logical bitwise NOR.
  - Syntax: ila.nor([expression], [expression])
  - Example: res = ila.nor(eax + ebx, ecx)
- **xnor**
  - Semantics: Logical bitwise XNOR.
  - Syntax: ila.xnor([expression], [expression])
  - Example: res = ila.xnor(eax, ebx)
- **sdiv**
  - Semantics: Signed devision.
  - Syntax: ila.sdiv([expression/immediate], [expression/immediate])
  - Example: res = ila.sdiv(0x2, eax)
- **smod**
  - Semantics: Signed modulo.
  - Syntax: ila.smod([expression/immediate], [expression/immediate])
  - Example: res = ila.smod(eax, ebx)
- **srem**
  - Semantics: Signed remainder
  - Syntax: ila.srem([expression/immediate], [expression/immediate])
  - Example: res = ila.srem(eax, 0x3)
- **ashr**
  - Semantics: Arithmetic right shift.
  - Syntax: ila.ashr([expression/immediate], [expression/immediate])
  - Example: res = ila.ashr(eax, 0x2)
- **concat**
  - Semantics: Concat two bitvector.
  - Syntax: ila.concat([expression], [expression]) or ila.concat([list of expression])
  - Example: res1 = ila.concat(eax, ebx)
res2 = ila.concat([eax, ebx, ecx])
  - Note: eax = 0xff; ebx = 0x00 → res1 = 0xff00
- **lrotate**
  - Semantics: Left rotate the bits in the bitvector.
  - Syntax: ila.lrotate([expression], [immediate])
  - Example: res = ila.lrotate(eax, 0x2)

- **rrotate**
  - Semantics: Right rotate the bits in the bitvector.
  - Syntax: ila.rrotate([expression], [immediate])
  - Example: res = ila.rrotate(eax, 0x3)
- **zero_extend**
  - Semantics: Zero extend the bitvector to the specified bit length.
  - Syntax: ila.zero_extend([expression], [immediate])
  - Example: res = ila.zero_extend(eax[7:0], 32)
- **sign_extend**
  - Semantics: Signed extend the bitvector to the specified bit length.
  - Syntax: ila.sign_extend([expression], [immediate])
  - Example: res = ila.sign_extend(m.const(0xff, 8), 32)
- **slt**
  - Semantics: Signed less than.
  - Syntax: ila.slt([expression], [expression])
  - Example: res = ila.slt(eax, ebx + ecx)
- **sle**
  - Semantics: Signed less than or equal to.
  - Syntax: ila.sle([expression], [expression])
  - Example: res = ila.sle(ebx + ecx, m.const(0xffff, 32))
- **sgt**
  - Semantics: Signed greater than.
  - Syntax: ila.sgt([expression], [expression])
  - Example: res = ila.sgt(eax, ebx)
- **sge**
  - Semantics: Signed greater than or equal to.
  - Syntax: ila.sge([expression], [expression])
  - Example: res = ila.sge(eax, ebx)
- **nonzero**
  - Semantics: Check if is non-zero.
  - Syntax: ila.nonzero([expression])
  - Example: res = ila.nonzero(eax – ebx)
- **implies**
  - Semantics: Logical implies.
  - Syntax: ila.implies([expression], [expression])
  - Example: cond = ila.implies(eax != 0, eip != 0)
- **ite**
  - Semantics: If then else.
  - Syntax: ila.ite([expression], [expression], [expression])
  - Example: res = ila.ite(flag, eax + ebx, eax - ebx)
- **appfun**
  - Semantics: Apply function on inputs.
  - Syntax: ila.appfun([function], [list of inputs])
  - Example: res = ila.appfun(fun1, [eax, ebx, ecx])