Towards a Hardware DSL Ecosystem

RubyRTL and Friends

Jean-Christophe Le Lann
Labsticc UMR CNR 6285
ENSTA Bretagne, France

Hannah Badier
Labsticc UMR CNR 6285
ENSTA Bretagne, France

Florent Kermarrec
Enjoy Digital
Landivisiau, France

contact : jean-christophe.le_lann@ensta-bretagne.fr

What is **RubyRTL** ?

A new Ruby internal domain-specific language (DSL)
for Hardware description,
inspired by Python-based Migen

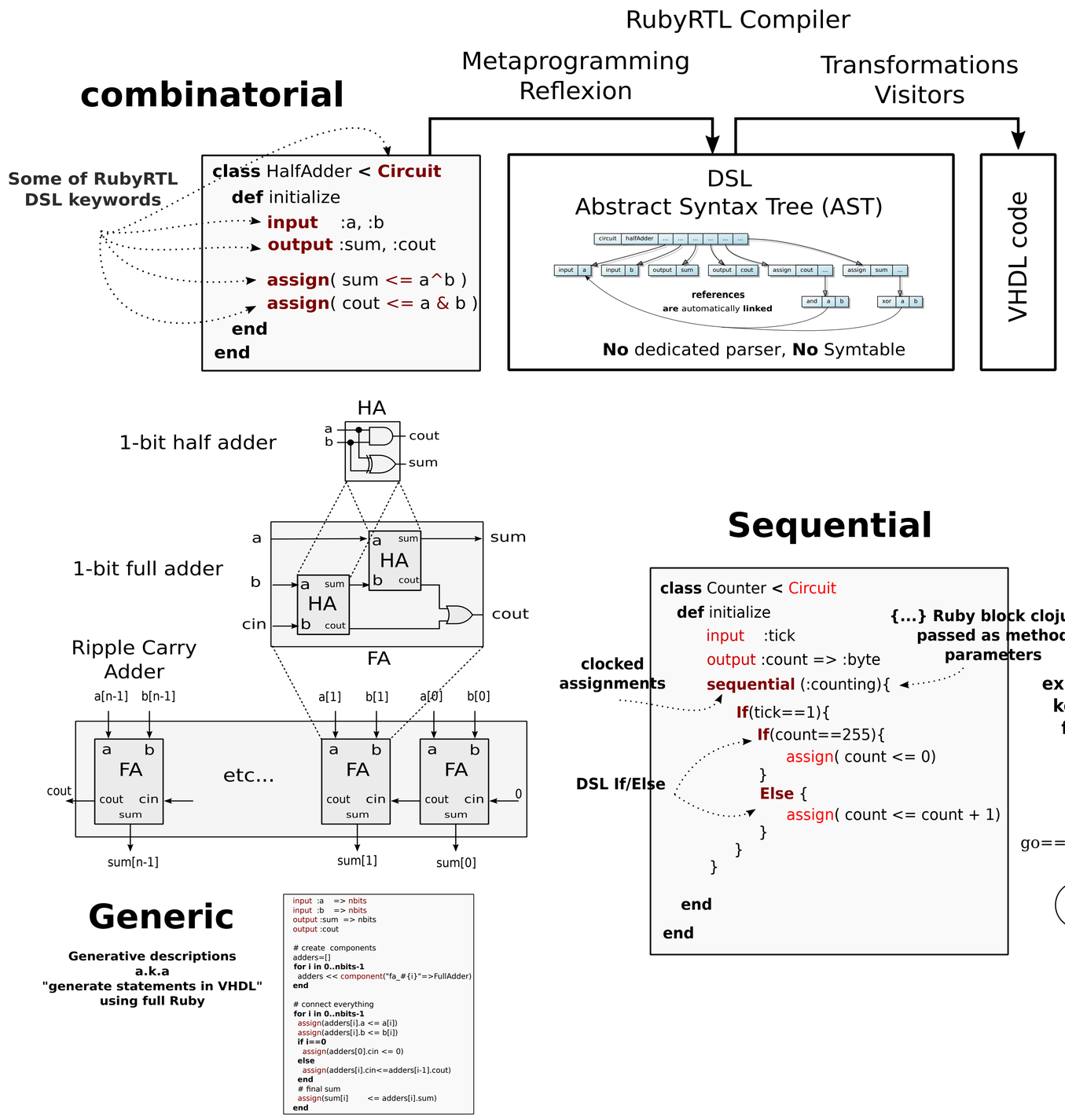
Why **Ruby** ?

- DSLs are commonplace in Ruby
- Metaprogramming is easy
- "Ruby makes you Happy !"

松本行弘
Yukihiro Matsumoto
"Matz", creator of Ruby

Motivations

- Opening hardware and FPGA design to the Ruby community by providing an approachable DSL.
- Encouraging IP interchange between various hardware DSLs. .
- Exploring the practicality of metaprogramming and syntax maleability for open source hardware design.



components

```
class FullAdder < Circuit
  def initialize
    input    :a, :b, :cin
    output :sum, :cout
    component :ha1 => HalfAdder
    component :ha2 => HalfAdder
    assign( ha1.a <= a )
    assign( ha1.b <= b )
    assign( ha2.a <= cin )
    assign( ha2.b <= ha1.sum )
    assign( sum <= ha2.sum )
    assign( cout <= ha1.cout | ha2.cout )
  end
end
```

Component Re-use

Pointed notation

Sequential

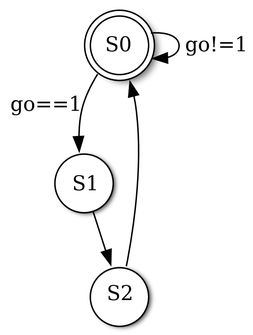
```
class Counter < Circuit
  def initialize
    input    :tick
    output :count => :byte
    sequential (:counting){
      If(tick==1){
        If(count==255){
          assign( count <= 0)
        }
        Else {
          assign( count <= count + 1)
        }
      }
    }
  end
end
```

clocked assignments

DSL If/Else

{...} Ruby block clojures passed as method parameters

FSM



explicit DSL keywords for FSM

```
input :go
output :f => :bv2

fsm(:simple){
  assign(f <= 0)
  state(:s0){
    assign(f <= 1)
    If(go==1){
      next_state :s1
    }
  }
  state(:s1){
    assign(f <= 2)
    next_state :s2
  }
  state(:s2){
    assign(f <= 3)
    next_state :s0
  }
}
```

Much more...

- enum/struct/array types

```
typedef :cplx => Record(:re=>:int16,:im=>int16)
typedef :cplx_ary => Array(256, :cplx)
wire :mem => :cplx_ary
(0..255).each{ assign(mem[i] <= {re:i i, im: i*2} }
puts mem[21][:im] # prints 42
```

- automatic type conversion

```
e.g  wire a => :bit
     wire w => :int8
     assign( w <= a + 5 )
     (cumbersome in VHDL)
```

experimental

Generic

Generative descriptions
a.k.a
"generate statements in VHDL"
using full Ruby

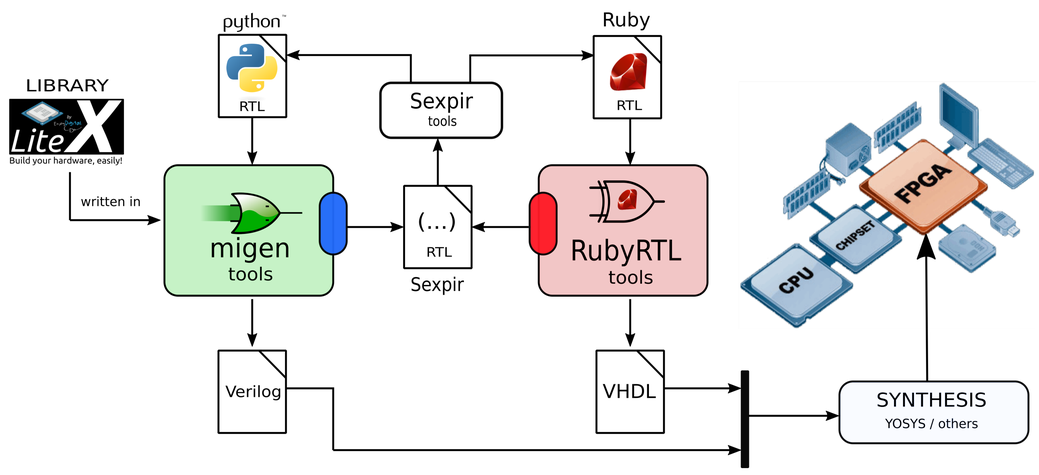
```
input  :a    => nbits
input  :b    => nbits
output :sum  => nbits
output :cout

# create  components
adders=[]
for i in 0..nbits-1
  adders << component("fa_#{i}"=>FullAdder)
end

# connect everything
for i in 0..nbits-1
  assign(adders[i].a <= a[i])
  assign(adders[i].b <= b[i])
  if i==0
    assign(adders[0].cin <= 0)
  else
    assign(adders[i].cin<=adders[i-1].cout)
  end
  # final sum
  assign(sum[i]      <= adders[i].sum)
end
```

IP Interchange

- Objectives : translate IPs back and forth.
- **Sexpir** : new s-expression based interchange format.
- Experiments :
  - Regenerate simple VHDL code : UART, etc
  - Dump LiTeX IPs
  - Translate Migen Verilog-oriented to RubyRTL VHDL



Future work

- Experimenting with larger designs
  - RISC-V SoC, Regular Architectures like CGRA and MPPA.
- Providing supplemental tools :
  - visualization, animation, cycle-based simulation
- Introducing multiple clocks

Open source

- http://www.github.com/JC-LL/ruby_rtl
- http://www.github.com/JC-LL/sexpir
- http://www.github.com/enjoy-digital