MetaRTL: Raising the Abstraction Level of RTL Design

Jianwen Zhu

Electrical and Computer Engineering
University of Toronto

March 16, 2001

jzhu@eecg.toronto.edu
http://www.eecg.toronto.edu/~jzhu

Outline

- *Motivation*
- Previous work
- Problems of synthesizable HDL
- MetaRTL
- Results

Importance of RTL Abstraction

- RTL abstraction
  - Semantics: FSMD (Gajski)
  - Synthesizable VHDL/Verilog (IEEE)
- Established industrial standard for ASIC design
- Backend interface for higher level design
- *de facto* soft IP exchange standard

Problems of Synthesizable HDLs

- Designed as a simulation language.
- Not designed with design reuse in mind.
- Not designed with a type system as powerful as that of software languages.
- Do not provide any abstractions for memories.
- Do not simulate fast enough at the RTL level.

Outline

- Motivation
- *Previous work*
- Problem of synthesizable HDL
- MetaRTL
- Results

Library-Based Approach

- Syntactically existing language:
  - Defines a programming style for hardware modeling
  - Simulation convenience: simulator for free (g++)
- Semantically new language:
  - Squeeze new semantics into old constructs
  - Synthesis headache: which semantics should I apply?
- Efforts:
  - SystemC: `http://www.systemc.org`
  - Cynlib: `http://www.cynapps.com`
  - OCAPI: Schaumont et. al. DAC'99

Language-Based Approach

- New language
  - Defines a computational model (semantics)
  - Defines language construct (syntax) to capture the model
- Compilation overhead: needs a new compiler
- Unambiguous synthesis
- Efforts
  - Numerous work on parallel extension of C/C++
  - OOVHDL: IEEE working group
  - VHDL+: extension of VHDL
  - Superlog: extension of Verilog with C constructs
  - V++: complete new language for synchronous reactive semantics

Our Approach

- Semantic level:
  - Rethink what should be abstracted away, what should not
  - Address the problems to be described
  - Raise the abstraction level of RTL design
- Syntax level
  - Can be applied to library-based SLDL
  - Can be embedded into C-based SLDL

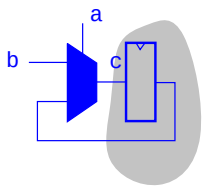
Outline

- Motivation
- Previous work
- *Problems of synthesizable HDLs*
- MetaRTL
- Results

Simulation Semantics

- HDL designed for simulation
  - How to synthesize delay
  - How to synthesize signal
- synthesis subset
  - Problematic constructs excluded
  - Infer hardware that exhibits discrete event semantics

```
...
if( a = '0' ) then
    c <= b;
end if;
...
```

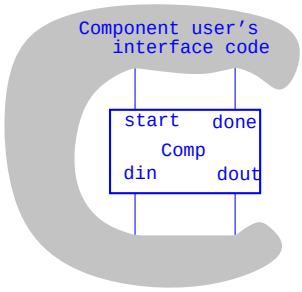


Design Reuse

- Hardware reuse by component instantiation
- Not sufficient for sequential components

  No interface protocol captured

```
u1 : Comp( start, done, din, dout );
...
...
start = '1';
for in in 0 to 8 do
    wait until clk'event and clk = '1';
    din <= a(i);
end for;
while( done = '0' ) do
    wait until clk'event and clk = '1';
    start <= '0';
end while;
while( done = '1' ) do
    wait until clk'event and clk = '1';
    b(j) := dout;
    j := j + 1;
end while;
...
```



Type System

- Type system: most effective error prevention in software
- Untyped system in synthesizable HDL
  - enumerate type
  - bit vector
- More abstract data type needed at RTL level

Memory Abstraction

- Any interesting application involves the use of memory
  - Multimedia: data sample storage
  - Networking: routing table, protocol states
- No abstraction of memory at RTL level
  - Interface protocol with memory
  - Dynamic allocation: pointer concept
  - Address calculation: array and record access

```
...
struct {
    int    field1;
    struct {
        char field3;
        } field2;
    } *p1;
short    a, *p2;
char     b[10];

...
a = 0;
b[3] = ‘a’;
p1->field1 = 1;
p1->field2.field3 = 2;
p2 = &a;
...
```


Outline

- Motivation
- Previous work
- Problems of synthesizable HDLs
- *MetaRTL*
- Results

MetaRTL Overview

- Syntactic-sugar-free, object-oriented, polymorphic language
- Difference from imperative program
  - Specifies hardware objects
  - Field has modifiers: storage class
  - `always` method: component logic
  - `public` method: interface protocol

MetaRTL Syntax

MetaRTL	::=	(Class)*
Class	::=	**class ID** [**[** Formal (, Formal)* **]**]
		{ (Field | Method)* }
Type	::=	**ID** [**[** Actual | (, Actual)* **]**]
Formal	::=	**class ID** | Type **ID**
Actual	::=	Type | Expr
Field	::=	sclass Type **ID** [= Expr]**;**
sclass	::=	**in** | **out** | **inout** | **reg** | **wire** | **latch** | Type
Method	::=	[**always** | **public**] Type **ID**
		[**(** Param (, Param)*] **)** { (Stmt)* }
Param	::=	[**in** | **out** | **inout**] Type **ID**

Stmt	::=	[LeftValue =] Expr **;**
	|	**if (** Expr **)** Stmt [**else** Stmt]
	|	**switch (** Expr **)** (CaseStmt)+
	|	**[ID] :** Stmt
	|	**do** Stmt **while(** Expr **) ;**
	|	**while** (Expr) Stmt
	|	**break ;**
	|	**return** Expr **;**
CaseStmt	::=	**case** Expr **:** Stmt | **default :** Stmt
Expr	::=	**Literal**
	|	**this**
	|	LeftValue
	|	Expr **. ID (** [Expr (, Expr)*] **)**
LeftValue	::=	**ID** | Type **. ID**
	|	Expr **. ID**

Synthesis Semantics

- Field $\mapsto$ Storage
  - `in, out, inout` fields $\mapsto$ ports
  - `wire` field $\mapsto$ wire
  - `latch, reg` fields $\mapsto$ registers
  - Normal field $\mapsto$ memory
- Method $\mapsto$ Logic
  - Assignment: predicated connection semantics
  - Method dispatch: protocol inlining
  - Statement: logic
  - State label: state boundary

Type System

- Arithmetic data types used in place of bit vectors
- Polymorphism
  - Parameterize over constants
  - Parameterize over type
- Subtyping: extensible hardware design

Design Reuse

- Type system improves reuse
- Explicit capture of interface protocol
  - Combinational components: connection and glue logic
  - Sequential components: partial FSMD
- Port mapping is replaced by method dispatch

Memory Abstraction

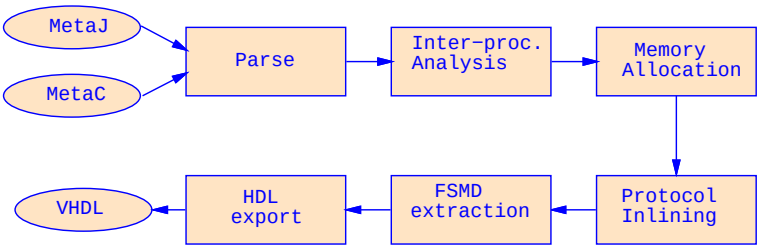
- Object reference and array, field access
- Access by name vs access by explicit address
- Memory bank and address allocation left to synthesis tool
- Automatic insertion of memory interface protocol

Outline

- Motivation
- Previous work
- Problems of synthesizable HDLs
- MetaRTL
- *Results*

Implementation

- Embed MetaRTL in experimental SLDL
- MetaSyn: MetaRTL $\mapsto$ synthesizable VHDL



Experimental Result

- Toronto DSP benchmark set
- TSMC 0.35um technology
- Completed by Mr. Prakash: first year undergraduate before any digital logic courses

Benchmark	MetaC (#lines)	VHDL (#lines)	Area (um^2)
fft	45	1028	500652
fir	29	355	272206
iir	46	1201	476529
latnrm	37	748	408111
lmsfir	39	796	405976
mmult	28	451	331058
smult	12	94	57157
sra	16	288	98599

Conclusion

- RTL semantics can be made more abstract
- RTL syntax can be as simple as one page
- Future work
  - Synthesis algorithms
  - Embedding in SLDL