SYSTEMVERILOG
HIGH-LEVEL SYNTHESIS

Bluespec SystemVerilog™ Training

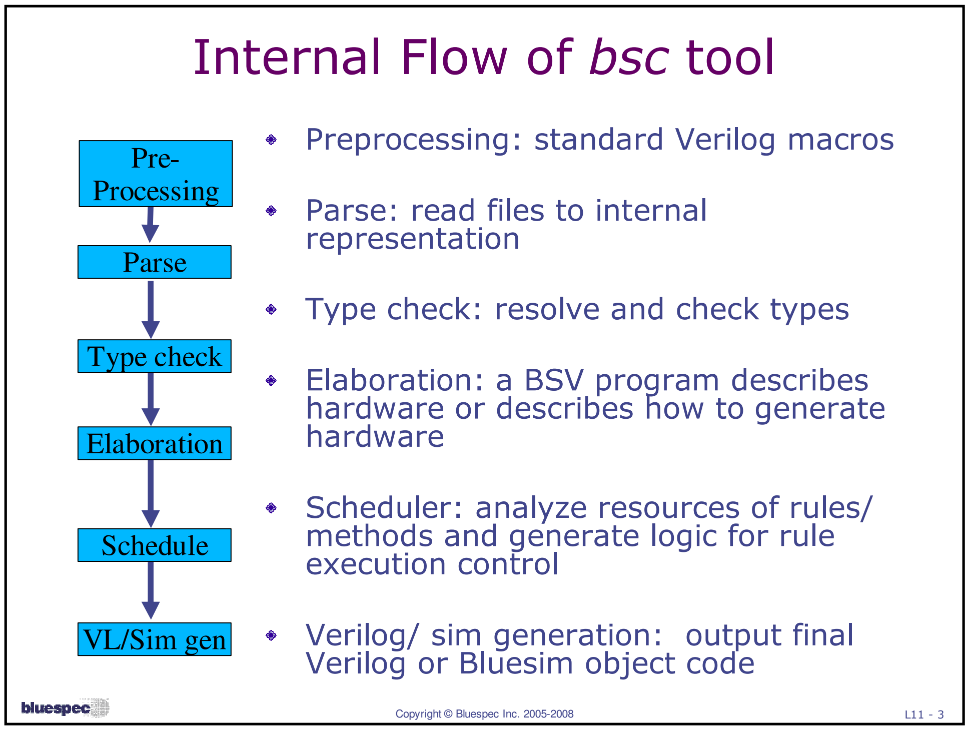
Lecture 11: Compiling, Running and Debugging

Copyright © Bluespec, Inc., 2005-2008

Lecture 11: Compiling, Running and Debugging

- Using bsc, the synthesis tool
- Simulating with a Verilog simulator
- Simulating with Bluesim, the direct simulator
- Understanding type-checking error messages
- Understanding scheduler error messages

Copyright © Bluespec Inc. 2005-2008

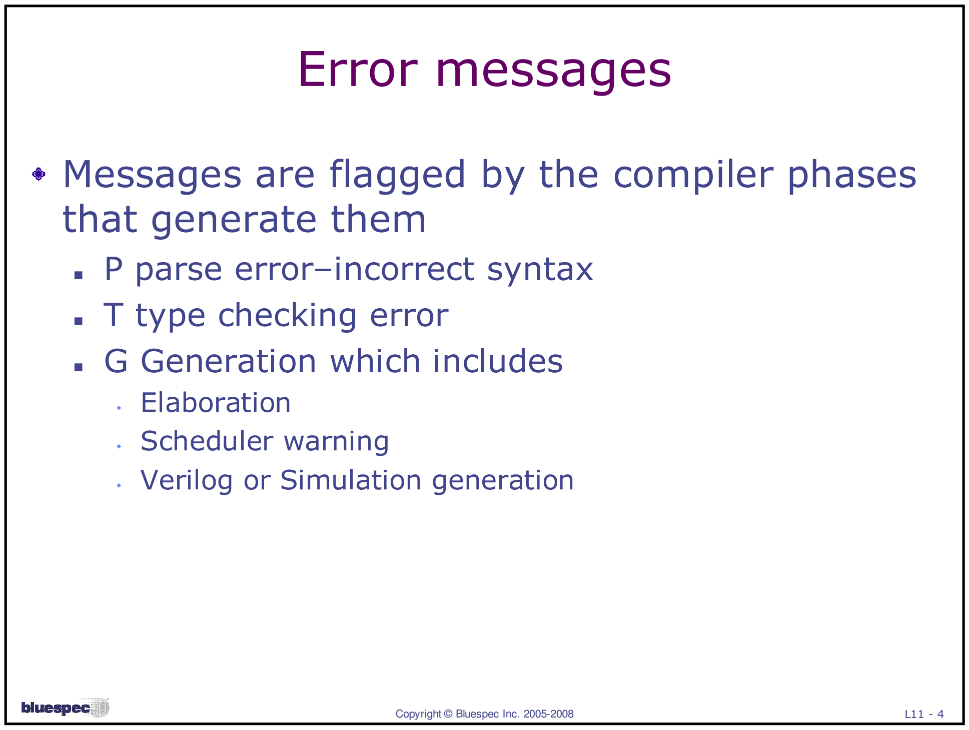


Internal Flow of *bsc* tool

$$\begin{tikzcd}
\text{Pre-Processing} \arrow[d] \\
\text{Parse} \arrow[d] \\
\text{Type check} \arrow[d] \\
\text{Elaboration} \arrow[d] \\
\text{Schedule} \arrow[d] \\
\text{VL/Sim gen}
\end{tikzcd}$$

- Preprocessing: standard Verilog macros
- Parse: read files to internal representation
- Type check: resolve and check types
- Elaboration: a BSV program describes hardware or describes how to generate hardware
- Scheduler: analyze resources of rules/ methods and generate logic for rule execution control
- Verilog/ sim generation: output final Verilog or Bluesim object code

Copyright © Bluespec Inc. 2005-2008



Error messages

- Messages are flagged by the compiler phases that generate them
  - P parse error–incorrect syntax
  - T type checking error
  - G Generation which includes
    - Elaboration
    - Scheduler warning
    - Verilog or Simulation generation

Copyright © Bluespec Inc. 2005-2008

Some useful bsc command line options

- List all command-line options
  - -help
- Verbose messages
  - -v
- Verilog or Bluesim
  - -verilog, -sim
- Setting output directories
  - -bdir, -simdir, -vdir, -info-dir
- Package management
  - -u recompiles imported modules if necessary
  - -p sets search path (+ is current path)
- Debugging
  - -keep-fires keeps scheduling signals
  - -show-schedule dumps details of schedule

Details in $BLUESPECDIR/../doc/user-guide.pdf

Initial and Don’t-Care Values

- Initial values for state elements are the same as in Verilog: 0xAAAAAA...
- Don’t-care values are given a value according to the `–unspecified–to` flag, during compilation
  - Just as Verilog uses
  - Choices are 0, 1, A, X, Z
  - X and Z are not allowed for Bluesim

Simulating with Verilog

- –verilog flag causes Verilog modules to be created for modules
  - marked by (* synthesis *) attribute, and/or
  - named by –g command-line flag
- Combined with –e flag indicating the top module and the necessary Verilog files, it will invoke a third party Verilog simulator and make the simulation executable.
  - Example: bsc –verilog –e myTop –vsim <simulator> *.v
- Verilog simulators and synthesizers need access to Bluespec's verilog primitives library–in $BLUESPECDIR/Verilog
  - Example: *simulator* +libext+.v -y $BLUESPECDIR/Verilog

Bluespec's main.v

- Bluespec provides a driver module main.v to stimulate your design with a per default clock and reset
- It contains other as options, dump vcd files, print cycle count, etc.
- Verilog simulators need access to Bluespec's simulation file $BLUESPECDIR/Verilog main.v
- Example:
  - *simulator* … $BLUESPECDIR/Verilog/main.v +define+TOP=<top-module>
- You can replace this with your own main.v

Debugging Options

- -keep-fires turns off optimization which may remove **`CAN_FIRE_rule`** and **`WILL_FIRE_rule`** signals
  - Useful during "performance debugging"

- -show-schedule dumps details of schedule phase. i.e, conflicting rule, predicates, etc.

Debugging Simulation

- Modules
  - those with the (* synthesize *) attribute are preserved in Verilog as separate modules
  - Bluespec library primitives – e.g., Reg, FIFOs, etc.
- Interfaces – methods become Verilog ports (example follows)
- Rules – show actions occurring in design
- Two signals show activity:
  - CAN_FIRE_rulename – True when rule predicate (both implicit and explicit) conditions are all true
  - WILL_FIRE_rulename – True when CAN_FIRE is true *and* scheduler allows rule to fire

Library Module ports

- Primitives use instance$port for signal names
- E.g.,

```
FIFO2 #(.width(16)) testfifo(.CLK(CLK),
                             .RST_N(RST_N),
                             .D_IN(testfifo$D_IN),
                             .ENQ(testfifo$ENQ),
                             .DEQ(testfifo$DEQ),
                             .D_OUT(testfifo$D_OUT),
                             .CLR(testfifo$CLR),
                             .FULL_N(testfifo$FULL_N),
                             .EMPTY_N(testfifo$EMPTY_N));
```

```
// instance testfifo
  assign testfifo$D_IN = dout + 16'd148 ;
  assign testfifo$DEQ = WILL_FIRE_RL_store || WILL_FIRE_RL_write ;
  assign testfifo$ENQ = WILL_FIRE_RL_get || WILL_FIRE_RL_update ;
  assign testfifo$CLR = 1'b0 ;
```

Methods to Ports

```
interface Example_ifc#(type any_t);
        method  any_t      first() ;
        method  Action     clear() ;
        method  Action     enqueue( any_t din ) ;
endinterface: Example_ifc
```

Synthesized Verilog

```
Output   RDY_first ;  // Boolean
Output   first ;      // of type any_t

Output   RDY_clear ;  // Boolean
Input    EN_clear ;   // Boolean

Output   RDY_enqueue ; // Boolean
Input    EN_enqueue ;  // Boolean
Input    enqueue_din ; // of type any_t
```

- All methods have RDY signal
- Action methods have EN signal
- A value method's output is its name
- Method arguments become inputs

Running Bluesim

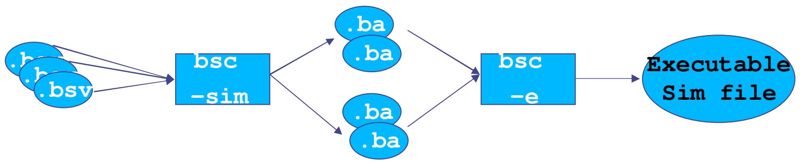
- bsc generates .ba files (among others!)

```
bsc –sim <file>.bsv
```

- Link these together with

```
bsc –sim –e <top-level-mod> [–o <exefile>] *.ba
```

- Creates an executable <exefile>



bluespec
Copyright © Bluespec Inc. 2005-2008

Simulating and debugging with Bluesim

Simulator command line options

- -help show command line options
- -m *n* run simulator for *n* clock cycles

Powerful interactive capabilities via tcl interface

(simulate until clock, single step, examine signals and state values, etc.)

Details in $BLUESPECDIR/../doc/user-guide.pdf

bluespec
Copyright © Bluespec Inc. 2005-2008

Bluesim can give you a significant speedup

Designs	Bluesim (secs)	Fastest RTL Simulator (secs)	Speedup (factor)
Wide GCD	5.53	39.7	7.18x
Life Game	8.37	283	33.9x
FIR Filter	4.23	61.8	14.6x
Upsize Converter	53.98	98.7	1.83x
Wallace Multiplier	48.66	417.0	8.57x
FIFO	11.73	124.2	10.6x
Mesa	37.45	126.4	3.37x
DIV3	8.24	116.8	14.2x
DES Core	28.55	89.36	3.13x
IDCT	15.69	16.32	1.04x

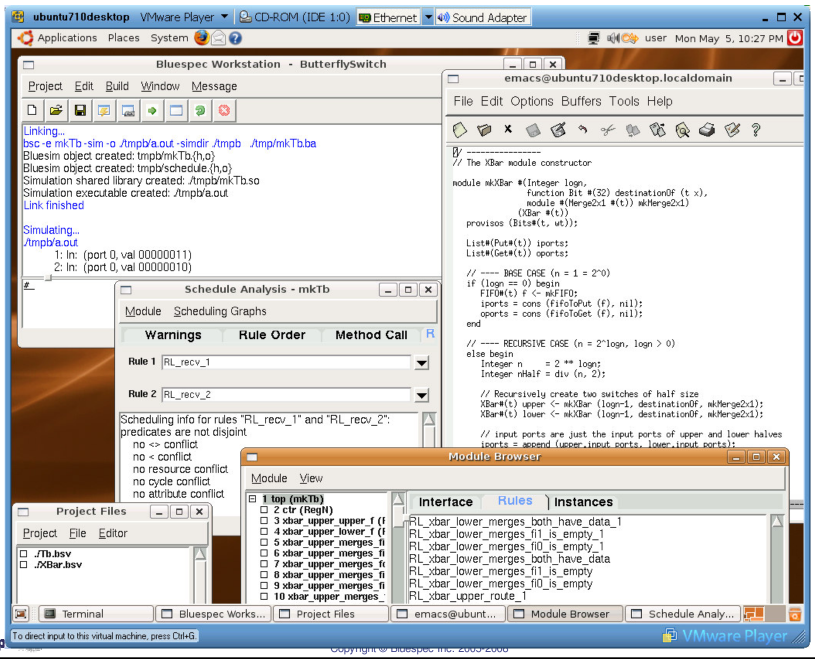
Copyright © Bluespec Inc. 2005-2008

Bluespec Development Workstation

- GUI-based “project management”
  - Windows for project file listings, browsing packages, modules, types, rule schedules, with hyperlinks to jump between different views
- Hyperlinked to standard text editors (emacs, gvim)
- Single-click compile, link, execute
- Supports standard Verilog simulators (Modelsim, VCS, NCSim, iverilog, cver, ...)
- Hyperlinked to standard waveform viewers (Novas, Modelsim)
- More in Lecture 15.

Copyright © Bluespec Inc. 2005-2008

Development Workstation screenshot



```
Linking...
bsc -e mkTb -sim -o ./tmpb/a.out -simdir ./tmpb   ./tmp/mkTb.ba
Bluesim object created: tmpb/mkTb.{h,o}
Bluesim object created: tmpb/schedule.{h,o}
Simulation shared library created: ./tmpb/mkTb.so
Simulation executable created: ./tmpb/a.out
Link finished

Simulating...
./tmpb/a.out
        1: In:  (port 0, val 00000011)
        2: In:  (port 0, val 00000010)
```

```
// ----------------
// The XBar module constructor

module mkXBar #(Integer logn,
                function Bit #(32) destinationOf (t x),
                module #(Merge2x1 #(t)) mkMerge2x1)
              (XBar #(t))
   provisos (Bits#(t, wt));

   List#(Put#(t)) iports;
   List#(Get#(t)) oports;

   // ---- BASE CASE (n = 1 = 2^0)
   if (logn == 0) begin
      FIFO#(t) f <- mkFIFO;
      iports = cons (fifoToPut (f), nil);
      oports = cons (fifoToGet (f), nil);
   end

   // ---- RECURSIVE CASE (n = 2^logn, logn > 0)
   else begin
      Integer n     = 2 ** logn;
      Integer nHalf = div (n, 2);

      // Recursively create two switches of half size
      XBar#(t) upper <- mkXBar (logn-1, destinationOf, mkMerge2x1);
      XBar#(t) lower <- mkXBar (logn-1, destinationOf, mkMerge2x1);

      // input ports are just the input ports of upper and lower halves
      iports = append (upper.input_ports, lower.input_ports);
```

Rule 1 RL_recv_1

Rule 2 RL_recv_2

```
Scheduling info for rules "RL_recv_1" and "RL_recv_2":
predicates are not disjoint
   no <> conflict
   no < conflict
   no resource conflict
   no cycle conflict
   no attribute conflict
```

```
./Tb.bsv
./XBar.bsv
```

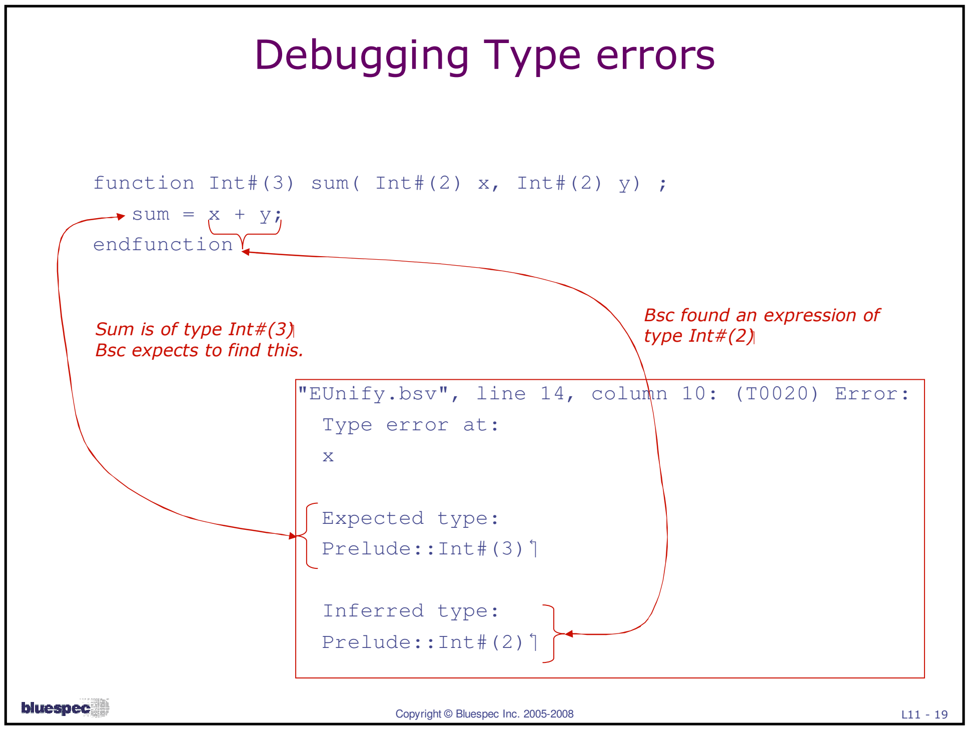
```
1 top (mkTb)
   2 ctr (RegN)
   3 xbar_upper_upper_f (F
   4 xbar_upper_lower_f (F
   5 xbar_upper_merges_fi
   6 xbar_upper_merges_fi
   7 xbar_upper_merges_fo
   8 xbar_upper_merges_fi
   9 xbar_upper_merges_fi
   10 xbar_upper_merges_
```

```
RL_xbar_lower_merges_both_have_data_1
RL_xbar_lower_merges_fi1_is_empty_1
RL_xbar_lower_merges_fi0_is_empty_1
RL_xbar_lower_merges_both_have_data
RL_xbar_lower_merges_fi1_is_empty
RL_xbar_lower_merges_fi0_is_empty
RL_xbar_upper_route_1
```

ubuntu710desktop VMware Player
CD-ROM (IDE 1:0)
Ethernet
Sound Adapter
Applications Places System
user Mon May 5, 10:27 PM
Bluespec Workstation - ButterflySwitch
Project Edit Build Window Message
emacs@ubuntu710desktop.localdomain
File Edit Options Buffers Tools Help
Schedule Analysis - mkTb
Module Scheduling Graphs
Warnings Rule Order Method Call
Project Files
Project File Editor
Module Browser
Module View
Interface Rules Instances
Terminal
Bluespec Works...
Project Files
emacs@ubunt...
Module Browser
Schedule Analy...
To direct input to this virtual machine, press Ctrl+G.
VMware Player

Type checking errors

- The Bluespec compiler performs strong type checking to guarantee that *values are only used in ways that make sense*, according to their type.
- Type checking errors can be frustrating… but
  - It catches a huge class of design errors and typos statically (without simulation)!
  - Strong Type checking replaces simulation-time debugging
  - Frequent experience:
    - “once it type-checked, it just worked, first time!”
  - Overall design time is shortened



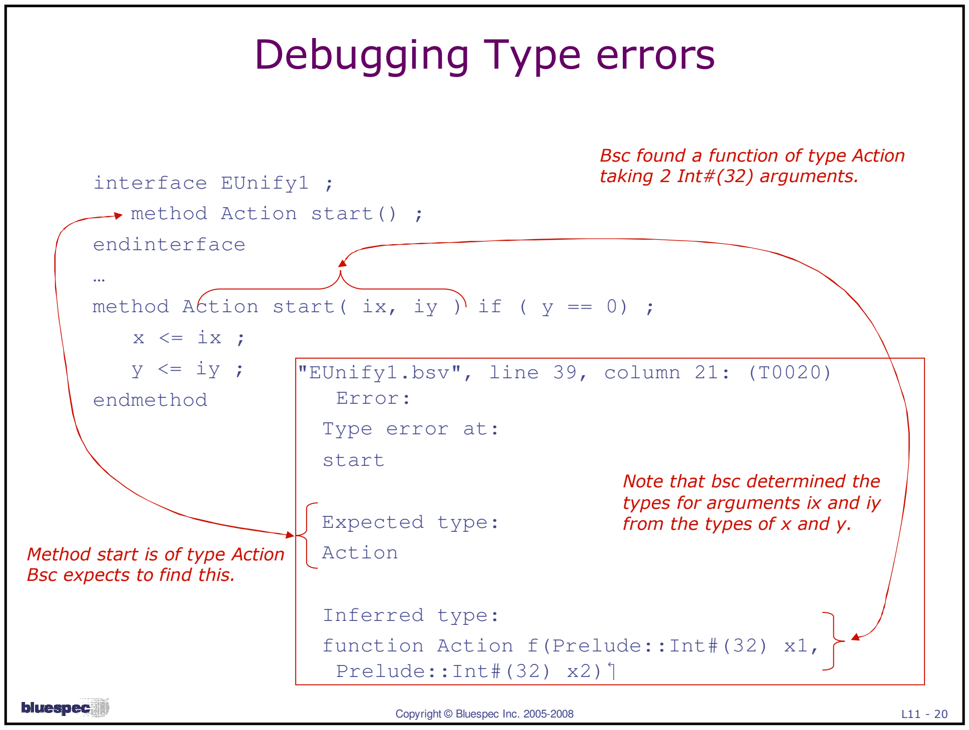
Debugging Type errors

```
function Int#(3) sum( Int#(2) x, Int#(2) y) ;
   sum = x + y;
endfunction
```

Sum is of type Int#(3)
Bsc expects to find this.

Bsc found an expression of type Int#(2)

```
"EUnify.bsv", line 14, column 10: (T0020) Error:
  Type error at:
  x

  Expected type:
  Prelude::Int#(3)

  Inferred type:
  Prelude::Int#(2)
```



Debugging Type errors

Bsc found a function of type Action taking 2 Int#(32) arguments.

```
interface EUnify1 ;
   method Action start() ;
endinterface
…
method Action start( ix, iy ) if ( y == 0) ;
   x <= ix ;
   y <= iy ;
endmethod
```

```
"EUnify1.bsv", line 39, column 21: (T0020)
  Error:
  Type error at:
  start

  Expected type:
  Action

  Inferred type:
  function Action f(Prelude::Int#(32) x1,
   Prelude::Int#(32) x2)
```

Note that bsc determined the types for arguments ix and iy from the types of x and y.

Method start is of type Action
Bsc expects to find this.

Debugging: scheduling

- Some sanity checks needed for scheduling
- if bsc can schedule the rules it will
- -show-schedule option to bsc displays the schedule
- -sched-dot option generate graph files for visualizing schedule (See development workstation)

- Guideline: There should be no error or (unexpected) warning about scheduling
  - e.g., dead rules, unexpected urgency
  - Also check rule conflicts

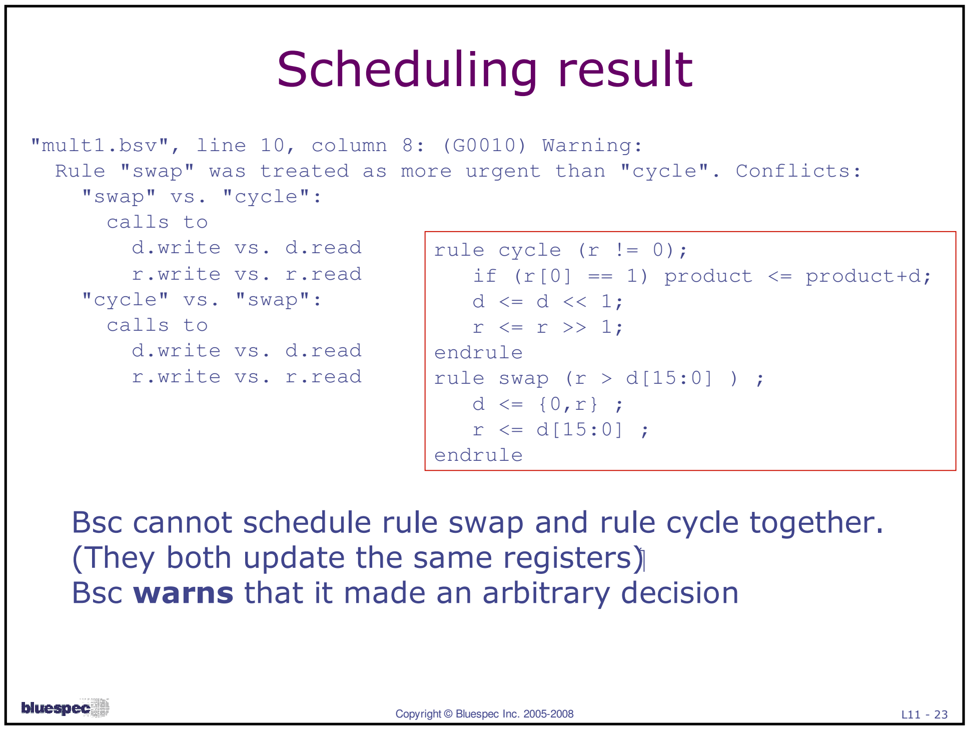
bluespec Copyright © Bluespec Inc. 2005-2008

Scheduling example

```
module mkMult1 (Mult_ifc);
   Reg#(Tout) product <- mkReg(0);
   Reg#(Tout) d       <- mkReg(0);
   Reg#(Tin)  r       <- mkReg(0);

   rule cycle (r != 0);
      if (r[0] == 1) product <= product + d;
      d <= d << 1;
      r <= r >> 1;
   endrule

   rule swap (r > d[15:0] ) ;
      d <= {0,r} ;
      r <= d[15:0] ;
   endrule

   method Action start (x,y) if (r == 0);
      d <= {0,x} ; r <= y; product <= 0;
   endmethod

   method result () if (r == 0);
      return product;
   endmethod
endmodule: mkMult1
```

- Multiplier example from earlier lecture
- This version includes a new “swap” rule

bluespec Copyright © Bluespec Inc. 2005-2008

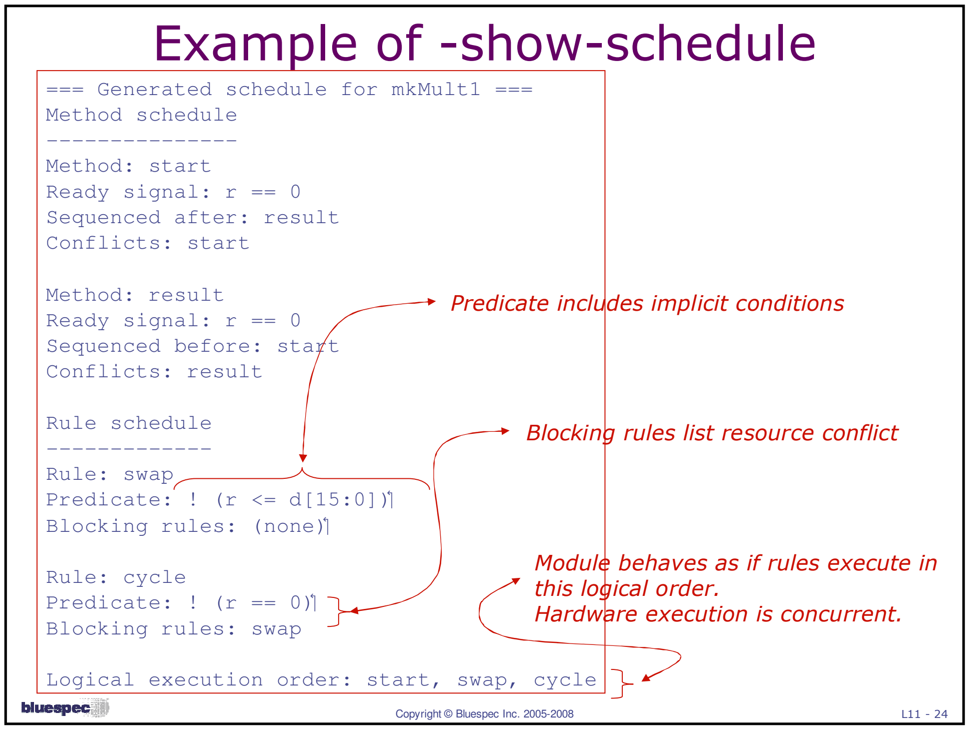


Scheduling result

```
"mult1.bsv", line 10, column 8: (G0010) Warning:
  Rule "swap" was treated as more urgent than "cycle". Conflicts:
    "swap" vs. "cycle":
      calls to
        d.write vs. d.read
        r.write vs. r.read
    "cycle" vs. "swap":
      calls to
        d.write vs. d.read
        r.write vs. r.read
```

```
rule cycle (r != 0);
   if (r[0] == 1) product <= product+d;
   d <= d << 1;
   r <= r >> 1;
endrule
rule swap (r > d[15:0] ) ;
   d <= {0,r} ;
   r <= d[15:0] ;
endrule
```

Bsc cannot schedule rule swap and rule cycle together.
(They both update the same registers)
Bsc **warns** that it made an arbitrary decision



Example of -show-schedule

```
=== Generated schedule for mkMult1 ===
Method schedule
---------------
Method: start
Ready signal: r == 0
Sequenced after: result
Conflicts: start

Method: result
Ready signal: r == 0
Sequenced before: start
Conflicts: result

Rule schedule
-------------
Rule: swap
Predicate: ! (r <= d[15:0])
Blocking rules: (none)

Rule: cycle
Predicate: ! (r == 0)
Blocking rules: swap

Logical execution order: start, swap, cycle
```

Predicate includes implicit conditions

Blocking rules list resource conflict

Module behaves as if rules execute in this logical order.
Hardware execution is concurrent.

Specifying rule urgency

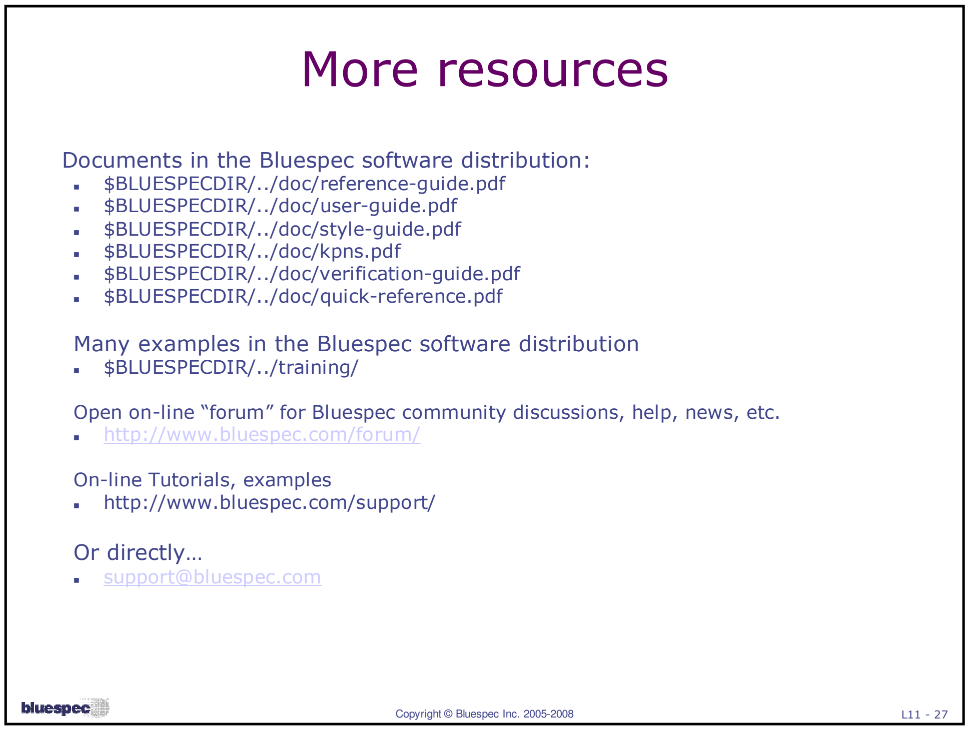
- Urgency annotation
  - **`(* descending_urgency = “swap, cycle” *)`**
  - Note position of quotation marks
- SystemVerilog “attribute” syntax

Removing conflicts

- Add rule conditions to disable swap and cycle from occurring together.

```
let swapem = (r[13:0] > d[15:2]) ;

rule cycle ((r != 0)&& (!swapem));
   if (r[0] == 1)
      product <= product + d;
   d <= d << 1;
   r <= r >> 1;
endrule

rule swap (swapem);
   d <= {0,r} ;
   r <= d[15:0] ;
endrule
```

- Explicit condition `swapem` added
- Rules predicates are provably mutually exclusive
- Need to reason that only one swap will occur



More resources

Documents in the Bluespec software distribution:

- $BLUESPECDIR/../doc/reference-guide.pdf
- $BLUESPECDIR/../doc/user-guide.pdf
- $BLUESPECDIR/../doc/style-guide.pdf
- $BLUESPECDIR/../doc/kpns.pdf
- $BLUESPECDIR/../doc/verification-guide.pdf
- $BLUESPECDIR/../doc/quick-reference.pdf

Many examples in the Bluespec software distribution

- $BLUESPECDIR/../training/

Open on-line “forum” for Bluespec community discussions, help, news, etc.

- http://www.bluespec.com/forum/

On-line Tutorials, examples

- http://www.bluespec.com/support/

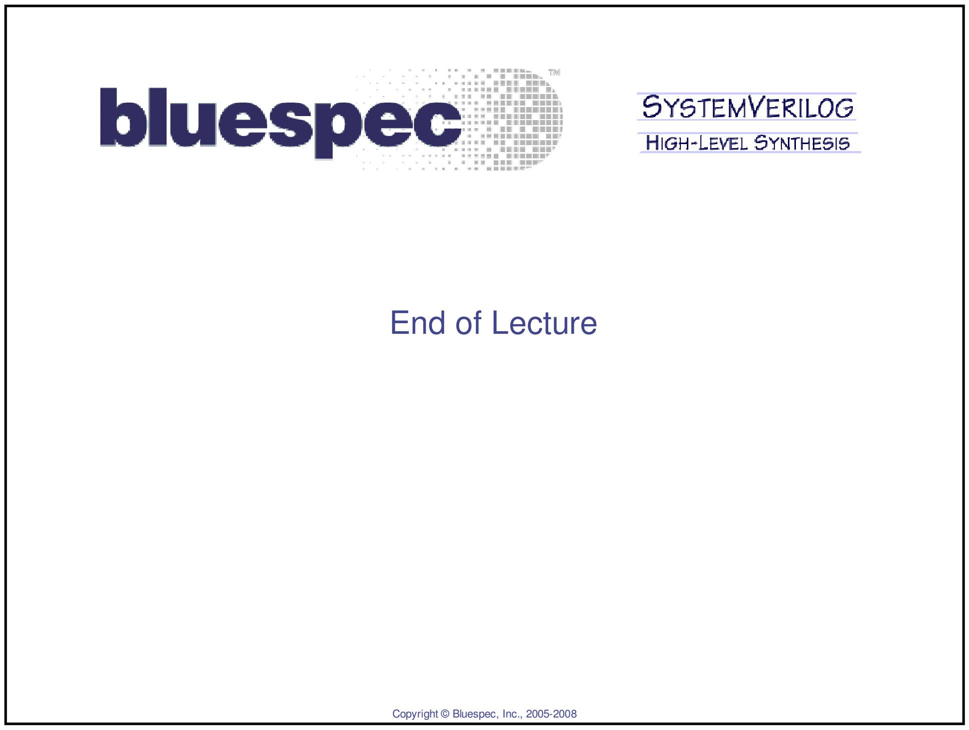
Or directly…

- support@bluespec.com

bluespec

Copyright © Bluespec Inc. 2005-2008

L11 - 27



bluespec™

SYSTEMVERILOG
HIGH-LEVEL SYNTHESIS

End of Lecture

Copyright © Bluespec, Inc., 2005-2008