v2c – A Verilog to C Translator Tool*

Rajdeep Mukherjee[1], Michael Tautschnig[2], and Daniel Kroening[1]

[1] University of Oxford, UK
[2] Queen Mary, University of London
{rajdeep.mukherjee,kroening}@cs.ox.ac.uk,mt@eecs.qmul.ac.uk

Appendix

Synthesizable Constructs

Our Verilog front-end support IEEE 1364.1 2005 Verilog standards. This includes the entire synthesizable fragment of Verilog. The detailed list of synthesizable Verilog constructs supported by our Verilog front-end is available in our website www.cprover.org/hardware/v2c/.

Non-Synthesizable Constructs

Most non-synthesizable constructs are not supported by *v2c*. These include *delay, repeat, wait, fork, join, event, deassign, force, release and time* constructs. *v2c* simply ignores all non-synthesizable constructs during parsing and does not generate any equivalent C code.

Netlist at different levels of abstractions

Figure 1 shows the Verilog RTL design described at different levels of granularity: *bit-level*, *word-level*, *software-level*. Note that the word-level netlist shown here is not a standard representation but is tool specific.

* Supported by ERC project 280053 and the Semiconductor Research Corporation (SRC) under task 2269.001.

Verilog	Bit-level Netlist	Word-level Netlist	Software-Netlist

`module top(Din,En, clk,Dout);`
`wire cs; reg ns;`
`input clk,Din,En;`
`output Dout;`
`assign Dout = cs;`
`always @(Din or cs or En)`
`begin`
`if (En) ns = Din;`
`else ns = cs;`
`end`
`ff ff(ns,clk,cs);`
`endmodule`
 	
`module ff(Din,clk, Dout);`
`input clk, Din;`
`output Dout;`
`reg q;`
`assign Dout = q;`
`always`
`@(posedge clk)`
`q <= Din;`
`endmodule`
	Variable Map:
	Inputs: top.clk=0,
	top.Din=1,
	top.En=2,
	top.ff.CLK=3,
	top.ff.Din=4,
	input[0]=6
	input[1]=7
	input[2]=11
	input[3]=12
	Wires: top.Dout=11,
	top.ff.Dout=5,
	top.cs=12,
	top.ns=!10
	Latch: top.ff.q=5
	Transition constraints:
	!(var(5)&!var(12))
	&!(!var(5)&var(12))
	!(var(4)&!(var(2)
	&var(1))&!(!var(2)
	&var(7)))&!(!var(4)
	&!(!(var(2)&var(1))
	&!(!var(2)&var(7))))
	!(var(3)&!var(0))&
	!(!var(3)&var(0))
	Next state functions:
	NEXT(top.ff.q)=var(4)
		State constraints:
		top.Dout==top.cs
		top.ff.Dout==
		top.ff.q
		top.ff.Din==
		top.ns
		top.ff.clk==
		top.clk
		top.ff.Dout==
		top.cs
		top.ns==top.En ?
		top.Din:top.cs
		
		Transisition constraints:
		next(top.ff.q)==
		top.ff.Din
			`_Bool nondet_bool();`
			`struct s_ff{_Bool q;};`
			`struct s_en{ _Bool ns;`
			`struct s_ff sff; }sen;`
			`_Bool ff(_Bool CLK,`
			`_Bool Din,_Bool *Dout)`
			`{ _Bool qold;`
			`qold = sen.sff.q;`
			`sen.sff.q = Din;`
			`*Dout = qold;`
			`return; }`
			`void top(_Bool clk,`
			`_Bool Din,_Bool En,`
			`_Bool *Dout){`
			`_Bool cs;`
			`if(En) {`
			`sen.ns = Din;`
			`else {`
			`sen.ns = cs; }`
			`ff(clk, sen.ns, &cs);`
			`*Dout = cs; }`
			`int main() {`
			`_Bool clk,En,Din,out;`
			`while(1) {`
			`Din = nondet_bool();`
			`En = nondet_bool();`
			`top(clk,Din,En,&out);}`
			`return; }`

Fig. 1. Verilog RTL design described at different levels of granularity: bit-level, word-level and software-netlist level

Software-netlist Model: The pseudo-code for the software-netlist model generated using *v2c* is shown in figure 2.

Command Line Options: The standard command-line usage for *v2c* is shown below.

```
v2c <Verilog-file-name> --module <Verilog-top-module> <options>
```

v2c has the following command line options:

```
-- for translation: v2c main.v --module main main.c
-- for debug information: v2c main.v --module main --show-parse-tree
```

Experimental Result

Table 1 shows the design statistics for various hardware circuits in Verilog and the equivalent Lines of Code (LOC) in ANSI-C design. The benchmarks are classified into data-path intensive and control-path intensive designs as shown in the table. Column 1 reports the name of the circuit and Column 2-5 reports the number of Latches/Flip-flops, input ports, output ports and gate count for each circuit. The design statistics is obtained using an open-source Verilog synthesis tool, *HANA*, which is available at sourceforge.net/projects/sim-sim/. Column 6 in Table 1 shows the number of lines of code (LOC) in the equivalent software-netlist model obtained using *v2c*. We do not report the translation times because these times are negligible even for large-scale SoC designs consisting of tens of Verilog design files. We now demonstrate one application

Pseudo-code for software-netlist model

```
// parameter definition
// macro definition
struct state_elements_design
  // declare all state-holding elements
  // of the current module
};
struct state_elements_design sdesign;
int initial_block() { //initialization of nets }
// Input are passed by value and output by reference
int design (data_type input, data_type *output) {

  // shadow variable declaration
  declare shadow variables for
  non-blocking assignments to
  the register elements

  // continuous assignments
  Place all continuous statements which
  are only dependant on input

  // always block
  Place the always block respecting
  intra-modular dependencies
  // procedural statements are bit-precise

  // continuous assignments
  Place all continuous statements which
  are updated by signals that are
  driven from the always block

  // Module instantiations
  Place all module instances
  with proper mapping of
  input and output ports

} // end of design module

int main() {
  // local varaibles
  declare all local variables
  which are passed to the design
  initial_block(); // call to initial block
  // check if the design is sequential.
  // if so, then put a  while(1) wrapper
  while(1) {
   // nondeterministic assignments
   nondeterministically assign inputs values
   // call the design
   design(input, &output);
  }

  // Assertions
  Place the C assertions here
} //end main
```

Fig. 2. Skeleton of the software-netlist model generated using *v2c*

of *v2c* in word-level property verification using two different case studies derived from real world Verilog circuits: *Traffic light controller* and *Buffer allocation protocol*. The traffic light controller is an unsafe design where the bug is manifested exactly after 64 clock cycles (iterations) in the software-netlist as well as Verilog design using software verifiers and bit-level model checkers respectively. On the other hand, the buffer allocation protocol design is safe and the properties are proven to be k-inductive for the same values of k in both the models. An important point to note here is that the generated word-level C program is structurally identical to the Verilog RTL. Thus, it is easier for debugging specially when the design is continuously evolving. *v2c* is integrated in

Circuit	Latches(L)/ FF	Input Ports	Output Ports	GATE Count	software-netlist (LOC)
Data-Path Intensive Designs					
UP-COUNTER	8 (FF)	3	1	66	22
FSM	4 (FF)	3	2	167	110
FIR	33 (FF)	3	1	15	198
SERIAL_ADDER	9(FF)	3	1	53	106
PIPELINED_ADDER	6(FF)	3	1	34	97
DIGITAL_AUDIO_VIDEO	10(FF)	10	8	269	329
BUFFER_ALLOCATION	5(FF)	5	2	100	102
Control-Path Intensive Designs					
ADPCM	196 (L)	5	4	3961	215
GCD CONTROLLER	4 (FF)	3	2	647	402
USB_PHY_IP	196 (FF)	8	10	977	862
CACHE COHERENCE	22 (FF)	9	6	112	828
ETHERNET_MAC_CONTROLLER	307(FF)	23	18	2818	9102
RCU	13(FF)	2	1	430	227
FIFO_CONTROLLER	37(FF)	5	1	83	140

Table 1. Design statistics for hardware and the software-netlist models

word-level formal hardware property verification (bounded and unbounded) tool flow to generate a word-level softlist-netlist representation for hardware circuits in Verilog.

Case Study 1: Traffic Light Controller Design

Verilog RTL

```
module traffic(reset, clk, time_left);
input reset, clk;
output [7:0] time_left;

parameter RED_LIGHT    = 0;
parameter GREEN_LIGHT  = 1;
parameter YELLOW_LIGHT = 2;
wire [5:0] RED_count;
wire [5:0] GREEN_count;
wire [2:0] YELLOW_count;
reg  [1:0] Light_Sign;
reg  [7:0] Counter;

assign RED_count    = 6'h3F;
assign GREEN_count  = 6'h3F;
assign YELLOW_count = 3'h3F;

assign time_left = Counter;

initial begin
 Counter = 0; Light_Sign = 0;
end

always @(posedge clk) begin
   if (reset) begin
      Light_Sign <= RED_LIGHT;
      Counter <= 8'd0;
   end
   else begin
    case (Light_Sign)
     RED_LIGHT :
       Light_Sign <= (Counter == 8'd0) ?
                  GREEN_LIGHT : RED_LIGHT;
       GREEN_LIGHT :
              Light_Sign <= (Counter == 8'd0) ?
              YELLOW_LIGHT : GREEN_LIGHT;
       YELLOW_LIGHT :
              Light_Sign <= (Counter == 8'd0) ?
              RED_LIGHT : YELLOW_LIGHT;
    endcase
    case (Light_Sign)
      RED_LIGHT :
           Counter <= (Counter == 8'd0) ?
           GREEN_count : Counter - 8'd1;
      GREEN_LIGHT :
           Counter <= (Counter == 8'd0) ?
           YELLOW_count : Counter - 8'd1;
      YELLOW_LIGHT :
           Counter <= (Counter == 8'd0) ?
           RED_count : Counter - 8'd1;
    endcase
  end
 end
 assert property (time_left != 8'd255);
 // this bug is manifested only after 64 iterations
 assert property (Light_Sign != 2'd2);
 assert property (Light_Sign != 2'd3);
endmodule
```

Software-netlist

```
struct state_elements_traffic {
  unsigned char Light_Sign;
  unsigned int Counter; };
struct state_elements_traffic straffic;
// parameters
int RED_LIGHT=0;int GREEN_LIGHT=1;
int YELLOW_LIGHT=2;int RED_count=63;
int GREEN_count=63;int YELLOW_count=63;

void traffic(_Bool reset, _Bool clk,
           unsigned int *time_left) {
  unsigned char Light_Sign_old;
  unsigned int Counter_old;
  // assignment statements
  Light_Sign_old = straffic.Light_Sign;
  Counter_old = straffic.Counter;
  if(!reset) {
    straffic.Light_Sign = RED_LIGHT;
    straffic.Counter = 0; }
  else {
   if(Light_Sign_old == RED_LIGHT)
    straffic.Light_Sign = (Counter_old == 0)
    ? GREEN_LIGHT : RED_LIGHT;
   else if(Light_Sign_old == GREEN_LIGHT)
    straffic.Light_Sign = (Counter_old == 0)
    ? YELLOW_LIGHT : GREEN_LIGHT;
   else if(Light_Sign_old == YELLOW_LIGHT)
    straffic.Light_Sign = (Counter_old == 0)
    ? RED_LIGHT : YELLOW_LIGHT;

   if(Light_Sign_old == RED_LIGHT)
    straffic.Counter = (Counter_old == 0)
    ? GREEN_count : Counter_old - 1;
   else if(Light_Sign_old == GREEN_LIGHT)
    straffic.Counter = (Counter_old == 0)
    ? YELLOW_count : Counter_old - 1;
   else if(Light_Sign_old == YELLOW_LIGHT)
    straffic.Counter = (Counter_old == 0)
    ? RED_count : Counter_old - 1; }
 // continuous assignment
 *time_left = straffic.Counter;
}
void main() {
 _Bool reset, clk;
 unsigned int time_left;
 while(1) {
  Traffic_Light(reset, clk, &time_left);
  assert((time_left != 0xffffffff));
  // this bug is manifested only after 64 iterations
  assert((straffic.Light_Sign != 2));
  assert((straffic.Light_Sign != 3));
 }
}
```

Fig. 3. Translation of traffic light controller circuit using *v2c*

Case Study 2: Buffer Allocation Protocol Design

Verilog RTL

```
module BufAl(clock,alloc_raw,nack,
alloc_addr,free_raw,free_addr_raw);
  input                clock;
  input                       alloc_raw;
  output                      nack;
  output [(4-1):0] alloc_addr;
  input                       free_raw;
  input [(4-1):0]   free_addr_raw;
  reg                  busy [0:(16 - 1)];
  reg [4:0]            count;
  reg                  alloc, free;
  reg [(4-1):0]     free_addr;
  integer                     i;

 initial begin
        for (i = 0; i < 16; i = i + 1)
          busy[i] = 0;
        count = 0;
        alloc = 0;
        free = 0;
        free_addr = 0;
 end

 assign nack = alloc & (count == 16);
 assign alloc_addr =
 ~busy[0] ? 0 :
                        ~busy[1] ? 1 :
                        ~busy[2] ? 2 :
                        ~busy[3] ? 3 :
                        ~busy[4] ? 4 :
                        ~busy[5] ? 5 :
                        ~busy[6] ? 6 :
                        ~busy[7] ? 7 :
                        ~busy[8] ? 8 :
                        ~busy[9] ? 9 :
                        ~busy[10] ? 10 :
                        ~busy[11] ? 11 :
                        ~busy[12] ? 12 :
                        ~busy[13] ? 13 :
                        ~busy[14] ? 14 :
                        ~busy[15] ? 15 :
                        0;

  always @ (posedge clock) begin
   alloc = alloc_raw;
        free = free_raw;
        free_addr = free_addr_raw;
  end
  always @ (posedge clock) begin
        count = count + (alloc & ~nack)
         - (free & busy[free_addr]);
if (free) busy[free_addr] = 0;
if (alloc & ~nack) busy[alloc_addr] = 1;
  end

  assert property: ((count[4] == 0
              || count[3:0] == 0));
endmodule
```

Software-netlist

```
_Bool nondet_bool();
unsigned char nondet_uchar();

struct state_elements_BufAl{
  _Bool alloc;
  _Bool free;
  unsigned char free_addr;
  _Bool busy[16];
  unsigned char count;
};
struct state_elements_BufAl smain;
void initial() {
        int i;
  for (i = 0; i < 16; i=i+1)
          smain.busy[i] = 0;
        smain.count = 0;
        smain.alloc = 0;
        smain.free = 0;
        smain.free_addr = 0;
}
void BufAl(_Bool clock, _Bool alloc_raw, _Bool *nack,
  unsigned char *alloc_addr, _Bool free_raw,
  unsigned char free_addr_raw)
{
  unsigned char tmpaddr, tmp;
  // always clocked block
  tmpaddr = smain.free_addr & 0xF;
  smain.count = ((smain.count & 0x1F) + (smain.alloc &
  !(*nack)) - (smain.free & smain.busy[tmpaddr])) & 0x1F;
  if(smain.free) smain.busy[smain.free_addr] = 0;
  tmp = *alloc_addr;
  if(smain.alloc & !nack) smain.busy[tmp] = 1;
  smain.alloc = alloc_raw;
  smain.free = free_raw;
  smain.free_addr = free_addr_raw;
  // continuous assignment statements
  *nack = smain.alloc && ((smain.count & 0x1F) == 16);
  *alloc_addr =
         !smain.busy[0] ? 0 : !smain.busy[1] ? 1 :
         !smain.busy[2] ? 2 : !smain.busy[3] ? 3 :
         !smain.busy[4] ? 4 : !smain.busy[5] ? 5 :
         !smain.busy[6] ? 6 : !smain.busy[7] ? 7 :
         !smain.busy[8] ? 8 : !smain.busy[9] ? 9 :
         !smain.busy[10] ? 10 : !smain.busy[11] ? 11 :
         !smain.busy[12] ? 12 : !smain.busy[13] ? 13 :
   !smain.busy[14] ? 14 : !smain.busy[15] ? 15 : 0;
}
void main() {
  _Bool clock, alloc_raw, nack, free_raw;
  unsigned char alloc_addr, free_addr_raw;
  // initial block
  initial();
  while(1) {
   alloc_raw = nondet_bool();
   free_raw = nondet_bool();
   free_addr_raw = nondet_uchar();
   BufAl(clock, alloc_raw, &nack,
   &alloc_addr, free_raw, free_addr_raw);
   assert(((((smain.count >> 4) & 1) == 0)
         || ((smain.count & 0xF) == 0)));
  }
}
```

Fig. 4. Translation of buffer allocation protocol circuit using *v2c*