Rutgers University
Application Specific Electronic Modules
(ASEM)
Rapid Application Specific Electronic Module Design and Test
ARPA Contract # F33615-93-C-4309
Rutgers University ARPA Contract Rutgers Format Netlist Compiler

Prof. Michael L. Bushnell

February 7, 2001

This report describes the Rutgers Format netlist language and parser that are provided to ARPA as part of the Rutgers CAD tool release of January, 1996. This software contains parsers that make it vastly easier to write CAD tools that can parse the Rutgers format.

1 Rutgers Netlist Language

1.1 Cell and Port Naming Conventions

In the Rutgers format, the input and output ports of a logic gate are simply referred to by the logic gate number, and the context of the reference determines whether an input or output port is being referenced. This format is by far the easiest of the three to type into a file by hand.

1.2 Rutgers Schematic File Format

It is assumed that a file called *circuit.rutmod* contains the circuit schematic. In this schematic file format all data is decimal:

```
The first field is the gate output number.

The second field is a two-character gate type as follows:
     an (and)        na (nand)       or (or)        po (primary output)
     in (inverter)   xo (xor)        eq (xnor)      pi (primary input)
     no (nor)        bu (buffer)     cx (xor)       df (D Master-Slave flip-flop)
```

The numbers of the gates driving this gate follow the gate type field. The maximum fanin of each gate is 9 except for xor (2), xnor (2), inverter (1), po (1), pi (0), bu (1), and df (1). The number of gate fanouts is unlimited, except for po's where it must be 0.

All gates must be numbered consecutively beginning with 1. Primary inputs must be numbered consecutively from 1 to the number of pi's. These must be followed by all logic gates in the circuit, which must also be consecutively numbered. Finally, all primary outputs must be declared, and followed by the number of the gate that generates their signal. PO's must also be numbered consecutively. Failure to declare the PO's means that tools will malfunction, since they assume that no circuit outputs are observable.

There is some checking for graph connectivity but no checking for feedback loops in combinational circuits. All gates are assumed to be reachable from one or more PI's. White space separates fields.

Each line declared in this format can be followed by one or two optional line delay specifications, of the form **r** # to specify a rising transition delay time for the logic gate output, **f** # to specify a falling transition delay time, or **d** # to specify the same time for both rising and falling delays. These delay specifiers are separated from the rest of the line by white space. Figure 1 is an example circuit schematic (Schneider's circuit): Figure 2 shows the **s27.rutmod** file translated from the ISCAS '89 benchmark circuit *s27* and Figure 3 shows its gate-level schematic. Files are available for all ISCAS benchmark circuits in Rutgers format.

```
1 pi
2 pi
3 pi
4 pi
5 no 1 3 r 2 f 3
6 no 2 3 r 2 f 3
7 no 2 4 r 2 f 1
8 no 2 5 d 3
9 no 1 6 d 3
10 no 4 6 d 3
11 no 3 7 d 3
12 no 8 9 10 11 r 4 f 1
13 po 12
```

Figure 1: Schneider's Circuit in Rutgers Format

2 Record Data Structure Definition

All three language parsers maintain netlist information in their data structures using these common data types:.

Record format and predefined data types:

1. CIRCUIT * – a pointer data type that points to an entire parsed circuit in memory. This is returned to you by the parser, and contains the entire circuit that was read in.

2. NODE * – a pointer data type that points to an entire logic gate, primary input, or primary output.

3. P_GATETYPE – an enumerated type giving the logic gate type. Acceptable values are: p_pi, p_po, p_nand, p_and, p_nor, p_or, p_equiv, p_xor, p_not, p_cxor, p_f_out, p_d_ff, p_buffer,

```
1 pi
2 pi
3 pi
4 pi
5 in 9
6 df 7
7 no 12 9
8 df 9
9 no 6 17
10 df 11
11 no 3 15
12 in 1
13 an 12 8
14 or 15 13
15 no 2 10
16 or 4 13
17 na 16 14
18 po 5
```

Figure 2: The **s27.rutmod** File

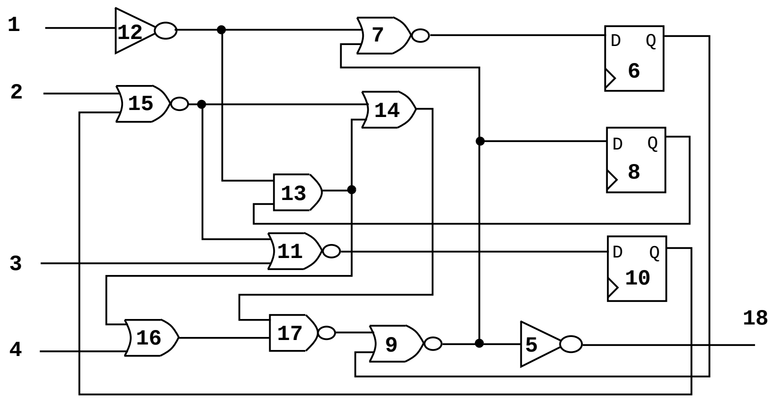

Figure 3: ISCAS '89 Benchmark Circuit **s27** Schematic

p_unknown. This is the type of primitive that this component represents (primary input, primary output, nand, and, nor, or, equivalence, exclusive-or, inverter, parity-flipper exclusive-or, fanout-pointer, D Flip-Flop, buffer, or unknown, respectively). These names were chosen to avoid identifier conflicts with names used in the CAD tools. p_fan_out is unused in Rutgers format.

4. gate_nmbr – Rutgers integer gate number for creating an output Rutgers format file. Usually, components are consecutively numbered starting with primary inputs, D Flip-Flops, all other logic gates, and ending with primary outputs.

5. nmbr_outputs – Number of outputs from this gate (total fanout).

6. output_list – FANOUT structure (a variable-extent array) with pointers to component number, component_name, node pointer tuples for each fanout to a component.

7. nmbr_inputs – Number of inputs to this gate (total fanin).

8. input_list – FANIN structure (a variable-extent array) with pointers to each component for each gate input.

9. level_number – Count of the number of logic levels from the PI to this component (sometimes this is a count of the levels from the nearest PO to this component).

10. rising_gate_delay – Integer delay value for the output stem for this gate (in whatever units your tool assumes). Default is 0.

11. falling_gate_delay – Integer delay value for the output stem for this gate (in whatever units your tool assumes). Default is 0.

12. logic_val – the 32-bit current logic value for the gate.

3 Rutgers Unified Parser Access Function Definitions

The purpose of this parser is to allow us to parse syntactically correct Rutgers files and convert them into the internal data structures that all of our CAD programs currently use. This parser currently flattens the entire netlist. After parsing is complete, all wires for every net emanate only from the driver of the net and go to each recipient of the net signal. We have a complete set of data base netlist access primitives with this parser.

Please note that the parser can maintain MULTIPLE netlists at the same time in memory in one program. That is because each netlist can have its own CIRCUIT *, and all compiled netlist access is done through the pointer.

The following are C access functions and subroutines that return information about the parsed netlist. The first group returns information about a parsed netlist. The second group has functions that modify the in-memory version of the netlist.

3.1 Netlist Information Extraction and Parsing

- Name: assign_gate_numbers
  - Function: Loops through all the PI's, gates and PO's and assigns gate numbers. Assumes that the PI's, gates and PO's are already in levelorder.

- Calling sequence: assign_gate_numbers (ckt);
- Inputs:
  1. ckt – CIRCUIT * data structure for parsed circuit
- Outputs: NONE
- Returns: NOTHING
- Author: M. Bushnell

- Name: faningate
  - Function: Sets or returns a pointer to the ith fanin of the node (gate or po) in the circuit, depending on whether it appears on the left or right hand side of a statement.
  - Calling Sequence: faningate (node, i)
  - Inputs:
    1. node – A NODE * referring to the node.
    2. i – An integer giving the fanin to return (numbered from 0 to number fanins - 1)
  - Returns: A NODE * pointer to the fanin.
  - Author: M. Bushnell

- Name: fanoutgate
  - Function: Sets or returns a pointer to the ith fanout of the node (pi or gate) in the circuit, depending on whether it appears on the left or right hand side of a statement.
  - Calling Sequence: fanoutgate (node, i)
  - Inputs:
    1. node – A NODE * referring to the node.
    2. i – An integer giving the fanout to return (numbered from 0 to number fanouts - 1)
  - Returns: A NODE * pointer to the fanout.
  - Author: M. Bushnell

- Name: fanoutgatenmbr
  - Function: Sets or returns the integer gate number of the ith fanout of the node (pi or gate) in the circuit, depending on whether it appears on the left or right hand side of a statement.
  - Calling Sequence: fanoutgatenmbr (node, i)
  - Inputs:
    1. node – A NODE * referring to the node.
    2. i – An integer giving the fanout to return (numbered from 0 to number fanouts - 1)
  - Returns: The integer Rutgers gate number of the fanout.
  - Author: M. Bushnell

- Name: gatefunction
  - Function: Sets or returns the logic gate type of the node, depending on whether it appears on the left or right hand side of a statement.

  - Calling Sequence: gatefunction (node)
  - Inputs: node – A NODE * referring to the node.
  - Returns: A P_GATETYPE enumerated type giving the logic gate type of this node.
  - Author: M. Bushnell

- Name: gate_list
  - Function: Returns or sets the entire binary variable extent array having the logic gates and flip-flops of the circuit, depending on whether it appears on the right or left hand side of a statement.
  - Calling Sequence: gate_list (circuit)
  - Inputs: circuit – a CIRCUIT * binary parsed netlist pointer.
  - Returns: A NODE ** variable extent array whose length is the number of gates and flip-flops.
  - Author: M. Bushnell

- Name: gate_number
  - Function: Sets or returns the Rutgers gate number of this node, depending on whether it appears on the left or right hand side of a statement.
  - Calling Sequence: gate_number (node)
  - Inputs: node – A NODE * referring to the node.
  - Returns: The Rutgers gate number of this node.
  - Author: M. Bushnell

- Name: gatesx
  - Function: Sets or returns the NODE pointer for the ith logic gate in the circuit, depending on whether it appears on the left or right hand side of a statement.
  - Calling Sequence: gatesx (circuit, i)
  - Inputs:
    1. circuit – a CIRCUIT * binary parsed netlist pointer.
    2. i – an integer specifying the ith gate to be returned (gates are numbered from 0 to number of gates - 1.
  - Returns: A NODE * to the actual logic gate record.
  - Author: M. Bushnell

- Name: levelnumber
  - Function: Sets or returns the integer level number of this node, depending on whether it appears on the left or right hand side of a statement.
  - Calling Sequence: levelnumber (node)
  - Inputs: node – A NODE * referring to the node.
  - Returns: The integer level number of this node.
  - Author: M. Bushnell

- Name: logicvalue
  - Function: Returns or sets the current logic value of the node (gate, pi, or po) in the circuit, depending on whether it appears on the right or left hand side of an statement.
  - Calling Sequence: logicvalue (node)
  - Inputs: node – A NODE * referring to the node.
  - Returns: An integer giving the current 32-bit logic value for the node.
  - Author: M. Bushnell
- Name: nmbr_gates
  - Function: Sets or returns the integer number of logic gates and flip-flops in the circuit, depending on whether it appears on the left or right hand side of the statement.
  - Calling Sequence: nmbr_gates (circuit)
  - Inputs: circuit – a CIRCUIT * binary parsed netlist pointer.
  - Returns: An integer giving the number of logic gates and flip-flops.
  - Author: M. Bushnell
- Name: nmbrinputs
  - Function: Sets or returns the integer number of fanins to this node, depending on whether it appears on the left or right hand side of a statement.
  - Calling Sequence: nmbrinputs (node)
  - Inputs: node – A NODE * referring to the node.
  - Returns: The integer number of fanins to this node.
  - Author: M. Bushnell
- Name: nmbroutputs
  - Function: Sets or returns the number of fanouts of the node (pi or gate) in the circuit, depending on whether it appears on the left or right hand side of a statement.
  - Calling Sequence: nmbroutputs (node)
  - Inputs: node – A NODE * referring to the node.
  - Returns: An integer giving the number of fanouts of the node.
  - Author: M. Bushnell
- Name: nmbr_pi
  - Function: Sets or returns the integer number of primary inputs in the circuit, depending on whether it appears on the left or right hand side of a statement.
  - Calling Sequence: nmbr_pi (circuit)
  - Inputs: circuit – a CIRCUIT * binary parsed netlist pointer.
  - Returns: An integer giving the number of primary inputs.
  - Author: M. Bushnell
- Name: nmbr_po

  - Function: Sets or returns the integer number of primary outputs in the circuit, depending on whether it appears on the left or right hand side of a statement.
  - Calling Sequence: nmbr_po (circuit)
  - Inputs: circuit – a CIRCUIT * binary parsed netlist pointer.
  - Returns: An integer giving the number of primary outputs.
  - Author: M. Bushnell

- Name: p_comp_type
  - Function: Sets or returns the type of a component (p_pi, p_po, p_nand, p_and, p_nor, p_or, p_equiv, p_xor, p_not, p_cxor, p_f_out, p_d_ff, p_buffer, p_unknown), depending on whether it appears on the left or right hand side of a statement. p_f_out is a fanout node and p_cxor is a parity flipper XOR gate (used only by BIST tools).
  - Calling Sequence:

```
P_GATETYPE p_comp_type (int gate_number, struct node * gate_array, int
    num_gates);
                 typedef enum {p_pi, p_po, p_nand, p_and, p_nor, p_or, p_equiv,
                               p_xor, p_not, p_cxor, p_f_out, p_d_ff, p_buffer,
                               p_unknown} P_GATETYPE;
```

  - Inputs:
    1. gate_number – Rutgers gate number for component.
    2. gate_array – POINTER to array of structure pointers, with one pointer for each primitive gate in the circuit.
    3. num_gates – number of gates in circuit.
  - Returns:
    * A P_GATETYPE enumerated type value giving the gate type, or
    * p_unknown if the component does not exist or is marked as unknown.
  - Author: M. Bushnell

- Name: p_dump_new_netlist
  - Function: Dump the internal Rutgers netlist data structure built in main memory on the results output file. This same routine works for both EDIF and L parsers.
  - Calling sequence: p_dump_new_netlist (CIRCUIT * ckt, FILE * resultsout);
  - Inputs:
    1. ckt – A pointer the the Rutgers CIRCUIT data structure for the netlist.
    2. resultsout – A file pointer the output CAD tool commentary file on which to dump the netlist. The file pointer must already be open.
  - Returns: Nothing
  - Author: M. Bushnell

- Name: pi_list
  - Function: Sets or returns the entire binary variable extent array having the primary inputs of the circuit, depending on whether it appears on the left or right hand side of a statement.

  - Calling Sequence: pi_list (circuit)
  - Inputs: circuit – a CIRCUIT * binary parsed netlist pointer.
  - Returns: A NODE ** variable extent array whose length is the number of primary inputs.
  - Author: M. Bushnell

- Name: PIs
  - Function: Sets or returns the NODE pointer for the ith primary input in the circuit, depending on whether it appears on the left or right hand side of a statement.
  - Calling Sequence: PIs (circuit, i)
  - Inputs:
    1. circuit – a CIRCUIT * binary parsed netlist pointer.
    2. – an integer specifying the ith primary input to be returned (pis are numbered from 0 to number of pis - 1).
  - Returns: A NODE * to the actual primary input record.
  - Author: M. Bushnell

- Name: po_list
  - Function: Sets or returns the entire binary variable extent array having the primary outputs of the circuit, depending on whether it appears on the left or right hand side of a statement.
  - Calling Sequence: po_list (circuit)
  - Inputs: circuit – a CIRCUIT * binary parsed netlist pointer.
  - Returns: A NODE ** variable extent array whose length is the number of primary outputs.
  - Author: M. Bushnell

- Name: POs
  - Function: Sets or returns the NODE pointer for the ith primary output in the circuit, depending on whether it appears on the left or right hand side of a statement.
  - Calling Sequence: POs (circuit, i)
  - Inputs:
    1. circuit – a CIRCUIT * binary parsed netlist pointer.
    2. i – an integer specifying the ith primary output to be returned (po's are numbered from 0 to number of po's - 1.
  - Returns: A NODE * to the actual primary output record.
  - Author: M. Bushnell

- Name: r_parse
  - Function: Specify the Rutgers file to be parsed.
  - Calling Sequence: int r_parse (CIRCUIT ** Rutgers_CIRCUIT, char * Rutgers_File_Name, int wantlevelsort);

- Inputs:
  1. Rutgers_CIRCUIT – CIRCUIT ** pointer to a pointer to a Rutgers internal circuit data structure. This will be automatically allocated and filled in by L_parse, so you only need to provide a dummy pointer here.
  2. Rutgers_File_Name – character string (with optional directory qualifiers) naming the Rutgers file to be parsed.
  3. wantlevelsort – an integer set to 1 if you want the netlist to be level sorted as it is created, 0 if you do not want that done.
- Returns:
  * -52 if the Rutgers file cannot be parsed.
  * 1 if the parsing succeeded and the circuit is sequential.
  * 0 if the parsing succeeded and the circuit is combinational.
  * Various negative numbers when other parsing errors occurred.
- Author: M. Bushnell

• Name: sourcegate
  - Function: Sets or returns a pointer to the ith input of the node (gate or po) in the circuit, depending on whether it appears on the left or right hand side of a statement.
  - Calling Sequence: sourcegate (node, i)
  - Inputs:
    1. node – A NODE * referring to the node.
    2. i – An integer giving the input number (0 to number inputs - 1) to be returned.
  - Returns: A NODE * to the node representing the logic gate input.
  - Author: M. Bushnell

• Name: verify_ckt_topology
  - Function: Verifies that the various connections of the ckt graph are correct.
  - Calling sequence: verify_ckt_topology (ckt);
  - Inputs:
    1. ckt – pointer to well-formed binary CIRCUIT
  - Outputs: NONE
  - Returns: NOTHING
  - Author: John Giraldi

3.2 Netlist Modification Access Functions

• Name: add_wire
  - Function: Adds a wire from "from" to "to".
  - Calling sequence: add_wire (from, to);
  - Inputs:
    1. from – NODE * to logic gate/PI that is driver of wire.

    2. to – NODE * to logic gate/PO that is load of wire.
  - Outputs: NONE
  - Returns: NOTHING
  - Author: M. Bushnell

- Name: delete_logic_gate
  - Function: Deletes and garbage collects the logic gate.
  - Calling sequence: delete_logic_gate (ckt, logic_gate_ptr);
  - Inputs:
    1. ckt – binary parsed circuit
    2. logic_gate_ptr – NODE * to logic gate to be freed
  - Outputs: NONE
  - Returns: NOTHING
  - Author: M. Bushnell

- Name: delete_pi
  - Function: Deletes and garbage collects the primary input
  - Calling sequence: delete_pi (ckt, pi_ptr);
  - Inputs:
    1. ckt – well-formed binary netlist
    2. pi_ptr – NODE * to Primary Input to be deleted
  - Outputs: NONE
  - Returns: NOTHING
  - Author: M. Bushnell

- Name: delete_po
  - Function: Deletes and garbage collects the primary output.
  - Calling sequence: delete_po (ckt, po_ptr);
  - Inputs:
    1. ckt – well-formed binary netlist
    2. po_ptr – NODE * to Primary Output to be deleted
  - Outputs: NONE
  - Returns: NOTHING
  - Author: M. Bushnell

- Name: delete_wire
  - Function: Deletes the wire from ”from” to ”to”.
  - Calling sequence: delete_wire (from, to);
  - Inputs:
    1. from – NODE * to logic gate/PI that is driver of wire.

    2. to – NODE * to logic gate/PO that is load of wire.
  - Outputs: NONE
  - Returns: NOTHING
  - Author: M. Bushnell

- Name: free_circuit
  - Function: Garbage collects and frees entire ckt.
  - Calling sequence: free_circuit (ckt);
  - Inputs:
    1. ckt – CIRCUIT * circuit data structure
  - Outputs: NONE
  - Returns: NOTHING
  - Author: M. Bushnell

- Name: make_new_logic_gate
  - Function: Creates the new logic gate.
  - Calling sequence: logic_gate_ptr = make_new_logic gate (ckt, gate_type, gate_no, gate_name, level);
  - Inputs:
    1. ckt – well-formed binary netlist
    2. gate_type – P_GATETYPE code (see sim.h) for type of gate to make
    3. gate_no – Rutgers logic gate number to be assigned to new PI
    4. gate_name – character string giving gate name
    5. level – integer level number to be assigned to gate.
  - Outputs: NONE
  - Returns: NODE * to newly created logic gate that was installed in the ckt.
  - Author: M. Bushnell

- Name: make_new_pi
  - Function: Creates the new primary input.
  - Calling sequence: logic_gate_ptr = make_new_pi (ckt, gate_no, gate_name);
  - Inputs:
    1. ckt – well-formed binary netlist
    2. gate_no – Rutgers logic gate number to be assigned to new PI
    3. gate_name – character string giving gate name
  - Outputs: NONE
  - Returns: NODE * to newly created PI that was installed in the ckt.
  - Author: M. Bushnell

- make_new_po

  - Function: Creates a new primary output.
  - Calling sequence: logic_gate_ptr = make_new_po (ckt, gate_no, gate_name, level);
  - Inputs:
    1. ckt – well-formed binary netlist
    2. gate_no – Rutgers logic gate number to be assigned to new PO
    3. gate_name – character string giving gate name
    4. level – integer giving level number for new PO
  - Outputs: NONE
  - Returns: NODE * to newly created PO that was installed in the ckt.
  - Author: M. Bushnell

- Name: r_write
  - Function: Rewrite entire flattened netlist back into an ASCII text file in Rutgers language format. The netlist is flattened, and the numbers of all entities appear consecutively in the netlist, with PI's followed by logic gates followed by PO's.
  - Calling Sequence: int r_write (CIRCUIT * ckt, FILE * sch_outputfile);
  - Inputs:
    1. ckt – A pointer to a Rutgers CIRCUIT data structure for the internal netlist.
    2. sch_outputfile – Open file pointer to ASCII file where the Rutgers output netlist will be written.
    3. design – Character string pointer to name of entire design.
  - Returns:
    * 1 if Rutgers netlist writing was successful,
    * -1 if sch_outputfile is not open.
  - Author: M. Bushnell

4 Include Files for Using the Rutgers Parser

In your main C program, please include these mandatory files:

```
<stdio.h>
~bushnell/docum/arpa/standard/c/sInclude/sim.h
~bushnell/docum/arpa/standard/c/sInclude/cstring.h
```

sim.h will include these files internally:

```
~bushnell/docum/arpa/standard/c/sInclude/sizes.h
~bushnell/docum/arpa/standard/c/sInclude/errno.h
~bushnell/docum/arpa/standard/c/sInclude/proto.h
```

cstring.h will include these files internally:

```
~bushnell/docum/arpa/standard/c/sInclude/cset.h
```

Sometimes, problems occur such that a second level include file cannot be found by the C compiler. When that happens, use the -I switch to specify places to look (like the sInclude directory) for the second level include files.

These C files must always be compiled and linked with your program:

1. b̃ushnell/docum/arpa/standard/c/r_interface.c – basic parser
2. b̃ushnell/docum/arpa/standard/c/fatalerror.c – basic error message facility
3. b̃ushnell/docum/arpa/standard/c/serialqueue.c – serial queue package
4. b̃ushnell/docum/arpa/standard/c/levelqueue.c – level sorted queue package
5. b̃ushnell/docum/arpa/standard/c/levelorder.c – arrange in level order package

If you write out a netlist, you must compile and include:

- b̃ushnell/docum/arpa/standard/c/r_writer.c

If you call verifytop to verify the netlist topology, you must compile and include:

- b̃ushnell/docum/arpa/standard/c/verifytop.c

If you use any of the functions to add and delete netlist hardware, you must also compile and include:

- b̃ushnell/docum/arpa/standard/c/addhardware.c

Some of these functions may like to write out error messages on a file pointer known as *resultsout*. If that happens, you need to copy *stdout* to *resultsout* before using these functions. In the event of any problems with these functions, please see Prof. Bushnell for help in fixing them. The good news is that all of these functions have been extensively used, and the code is known to be 100 % bug free.