Manual Latch & Flip-Flop Recognition in AccuCell and AccuCore

Introduction

AccuCell and AccuCore use an automatic algorithm for recognizing logic functions, latches and flip-flops. Manual methods are also possible in cases where greater control is or necessary. The procedure and syntax are outlined here and examples given.

Manual Function Extraction

If a cell cannot be processed automatically, writing a function equation (.eqn) file should be attempted before using a vector table (.tbl) file. When writing equations the following rules of construction apply:

1) Every output must have at least one set of equations.
2) Logical operators with parentheses can be used to specify the order of operation.
3) Logical AND "&" takes precedence over logical OR "|"
4) Logical NOT "~" takes precedence over logical AND "&"
5) For states 0 and 1, equations are expressed in terms of inputs and storage states.
6) Manually written equations take precedence over those generated automatically.

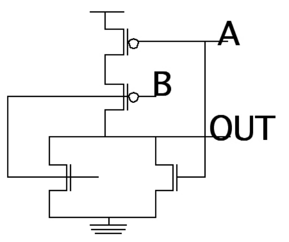
.eqn EBNF Equation File Syntax

```
<legal_char> ::= {{a…z} | {A…Z} | {0…9} | "_" |
"."} // under-score period
<illegal_char> ::= ~ <legal_char> // NOT(<legal_
char>)
<escaped_char> ::= "\" <illegal_char>
<char> ::= <legal_char> | <illegal_char>
<string> ::= <char> [<char>]
<legal_signal_name_char> ::= <legal_char> |
<escaped_char>
<legal_signal_name> ::= <legal_signal_name_char>
[<legal_signal_name_char>]
<semi> ::= ";" // semi-colon
<assignment> ::= ":=" // colon equal-sign
<not_back_slash> ::= ~ "\" // NOT(back-slash)
<new_line> ::= <not_back_slash> \n // new_line_char
<line_continue> ::= "\" \n // back-slash must be
the last char in the line
<white_space> ::= {" " | \t} [<white_space>] //
tab-char
<separator> ::= <white_space> [<line_continue>]
<comment_to_end_of_line> ::= "/" "/" <string> <new_
line> // double fwd-slash
<term> ::= <semi> <new_line> [<new_line>]
<hyphen> ::= "-"
<not> ::= "~"
<and> ::= "&"
<or> ::= "|"
<oper> ::= <and> | <or>
<rising_edge_suffix> ::= "+"
<falling_edge_suffix> ::= <hyphen>
<previous_state_prefix> ::= <hyphen>
<input_signal> ::= <legal_signal_name> // defined in
.cfg INPUTS cmd
<bidir_signal> ::= <legal_signal_name> // defined in
.cfg INOUTS cmd
<clock_signal> ::= <legal_signal_name> // defined in
.cfg CLOCKS cmd
<output_signal> ::= <legal_signal_name> // defined
in .cfg OUTPUTS cmd
<input> ::= {[<previous_state_prefix>]
<input_signal>} |\
{[<previous_state_prefix>] <bidir_signal>} |\
<clock_signal> {<rising_edge_suffix> |
<falling_edge_suffix>}}
<output> ::= <output_signal> | <bidir_signal>
<not_input> ::= <not> <input>
```

```
<grouping> ::= "(" [<separator>] <expr>
[<separator>] ")"
<logical_exp> ::= {<grouping> | <not_input> |
<input>} [<oper>] [<grouping> |\
<not_input> | <input>] // bidir_signal may not
self-refer
<output.0_expr> ::= <output>".0" [<separator>]
<assignment> [<separator>]\
<logical_expr> <term>
<output.1_expr> ::= <output>".1" [<separator>]
<assignment> [<separator>]\
<logical_exp> <term>
<output.z_expr> ::= <output>".z" [<separator>]
<assignment> [<separator>]\
<logical_expr> <term>
<output.x_expr> ::= <output>".x" [<separator>]
<assignment> [<separator>]\
<logical_expr> <term>
<eqn_file> ::= [<comment_to_end_of_line>] |
<separator> | <new_line>]\
<output.0_expr> [<comment_to_end_of_line> |
<separator> | <new_line>]\
<output.1_expr> [<comment_to_end_of_line> |
<separator> | <new_line>]\
[<output.z_expr>] [<comment_to_end_of_line> |
<separator> | <new_line>]\
[<output.x_expr>] [<comment_to_end_of_line> |
<separator> | <new_line>] <eof>
```

Examples

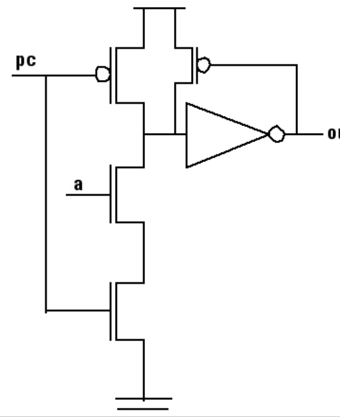
The following examples are for demonstration purposes ONLY and are NOT representative of functions requiring manual definition.



```
out.0:=a | b;
out.1=:~a & ~b
```

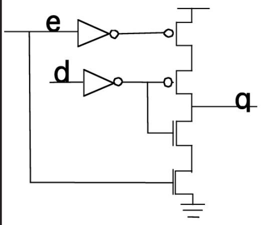
NOR Gates with Inputs A and B and Equation

Figure 1. Multiplexer and Equation.



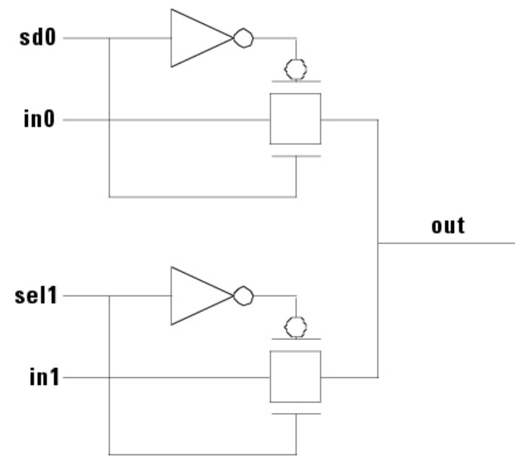
```
out.0 := ~pc
out.1 := a&pc
```

Figure 2. Standard Footed Domino BUFFER and Equation.



```
q.0 := en & ~d;
q.1 := en & d;
q.z := ~en
```

Figure 3. Tristate BUFFER and Equation



```
out.0 := (~sel1&sel0&~in0) |
(sel1&~in1&~sel0) | (~in1&sel0&~in0)
; // out is 0
out.1 := (in1&sel0&in0) |
(~sel1&sel0&in0) | (sel1&in1&~sel0) ;
// out is 1
out.z := (~sel1&~sel0) ; //out is
floating
out.x := (sel1&in1&sel0&~in0) |
(sel1&~in1&sel0&in0) ; // out is un-
known
```

Figure 4. Multiplexer and Equation.

Sequential Devices

For sequential cells, the CLOCKS command must be included in each cell configuration file. Then, the cells must be grouped by domain and within a domain, identified by phases.

Flip-flops with asynchronous set and clear lines can also be specified using equations. Consider a flip-flop with an active high signal "set" and active high signal "clr". Asserting "set" causes the flop to produce a 1 on output, but asserting "clr" causes it to produce a 0.

In the example below, "d" and the clocks "m" (main), "!m" (main) are irrelevant for the "set" and "clr" terms.

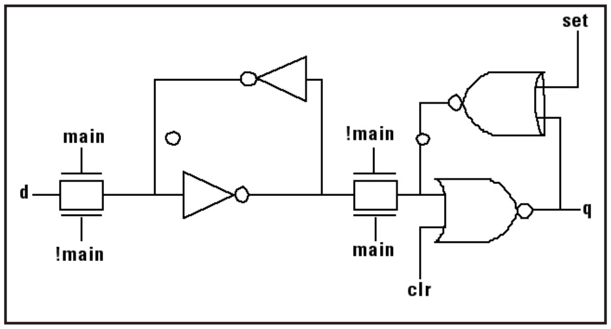


Figure 1.

Flip-flop with Asynchronous set, clear and equation

```
q.0 := ~d & main+ & ~set & ~clr | clr;
q.1 := d & main+ & ~set & ~clr | set & ~clr;
```

Pre-Processing in .cfg files

In AccuCell all that is necessary to utilize .egn or .tbl files is to add either:

1) EQN_FILE_NAME myfile.eqn or

2) TBL_FILE_NAME myfile.tbl to the cell level .cfg file.

In AccuCore the block level .cfg file must specify either:

1) KEEP_SUBCKT (for hierarchical netlist)

2) FIND_SUBCKT (for flat netlists) command to first partition the function and point to the .egn or .tbl file that defines the override.

```
KEEP_SUBCKT <subckt_name> {<inputs>}
{<outputs>} {bidirs>} {<clocks>} {EQN
<eqn_file>|<tbl_file>}
FIND_SUBCKT <subckt_name><pattern_file>
{<powers>}{<grounds>} {<inputs>} {<out-
puts>} {bidirs>} {<clocks>} {EQN <eqn_
file>|<tbl_file>}
```

NOTE: flat netlists will need a pattern file for matching.

Conclusion

While a .tbl file may be necessary to completely control the characterization process for some functions, .egn files are often a more suitable and easily designed solution to function extraction issues. The also take advantage of AccuCell's and AccuCore's automatic vector generation capabilities.