vc_apps_npi

1. npi_find_signal_regex

其采用的正规表达式是与vim一致的那种，不是perl的那种， 比如 `\(abc\|cde\)$` 表示匹配以abc或者cde结尾的信号

tcl语法：

```
proc npi_find_signal_regex { root_scope sig_regex sig_hdl_list }
```

从root_scope层次开始往下查找，所以子层次的信号也会被匹配到。

其正则表达式不支持`\d \w \s`等等类似的简写方式，需要使用`[a-z] [0-9]`类似的方式。

支持`. * +`等方式。

2. npi_find_inst_regex

```
proc npi_find_inst_regex { root_scope inst_regex inst_hdl_list }

Parameters
root_scope
Specify the hierarchical name of a root scope. This procedure will search
the target
handle from the scope. The top module will be used if it is empty.
inst_regex
Specify the instance regular expression naming rule. Any instances should be
collected if their names (not full names) match the specified naming rule.
inst_hdl_list
Specify the output list name. Collect all resultant handles into this list.

Return Value
Return the total number of matched instances. If none is found, return 0.
```

3. npi_nl_handle_by_name

```
npi_nl_handle_by_name -name <name> [-type <type>]
Parameters
-name <name>
<name> is a string containing the full hierarchical name of an object.
-type <type> (optional)
<type> is a string representing the type of object for which to obtain the
handle. If
```

```
-type is not specified, default value will be npiNlUndefined.
Return Value
A string representing the handle to an object.
```

用这个命令可以找到模块的port, 从而可以有办法知道port的方向。

```
set hdl [npi_nl_handle_by_name -name "top.a.datain" -type npiNlPort]
set dir [ npi_nl_get_str -property npiNlDirection -object $hdl ]
```

4. npi_trace_driver2

```
proc npi_trace_driver2 { sigHierName resListName { needPassThrough 1 }


proc npi_trace_driver2 { sigHierName resListName { needPassThrough 1 }
{boundaryHdlList {} } { trcOptionsList “1 0 1 1” } }
# 后面两个参数一般都没有使用
```

```
sigHierName
  Specify the hierarchical name of the target signal.
```

```
resListName
 它是返回一个list, list里面有多个元素, 每个元素也是一个list sigHdlList表示一个具体的driver赋值结构右边所有信号的一个list集合。
  Specify the name of the list where the driver statement and signal handles will be collected.
  The structure of the list is as follows:
  resultList ::= { {srcHdl scopeHdl isPassThrough numSigUse useHdl sigHdlList}* }
    srcHdl : The tracing source handle of this use handle.
    scopeHdl: The scope handle of this use handle.
    isPassThrough: Indicate this use handle is whether or not passed through.   --- 1, 表示端口信号驱动, 不是=号赋值
    numSigUse: Indicate how many times of source handle be traced in this use handle.
    If this value is 0, it means this useHdl is traced by other statement (statement structure based trace).
    useHdl: Target use handle. The definition of use handle, refer to Use and Vh Use.   --- 暂时没有感觉有什么用
    sigHdlList: The driver signal handles of this use handle.
```

```
needPassThrough
  A Boolean value specifying whether or not to pass through the port of module and map element of vh component.
  0: the boundary of npiModule or npiVhComponent object will not be passed through when traversing the model
  1: the boundary of npiModule or npiVhComponent object will be will not be passed through when traversing the model
```

```
  default value: 1
```

4.1 使用举例

```
# -------------------------------------------------------------------------
proc dump_res_list { resList targetFileHdl } {
    set size [ llength $resList ]
    for { set i 0 } { $i < $size } { incr i } {
        set dlStruct [lindex $resList $i]
        puts $targetFileHdl " $i source: [ npi_util_get_hdl_info [lindex
$dlStruct 0] ]"
        puts $targetFileHdl " scope: [ npi_util_get_hdl_info [lindex
$dlStruct 1] ]"
        puts $targetFileHdl " isPassThrought: [lindex $dlStruct 2] "
        puts $targetFileHdl " numSigUse: [lindex $dlStruct 3] "
        puts $targetFileHdl " useHdl: [ npi_util_get_hdl_info [lindex
$dlStruct 4] ]"
        set sigList [lindex $dlStruct 5]
        set sigSize [llength $sigList]
        for { set l 0 } { $l < $sigSize } { incr l } {
            puts $targetFileHdl " [ npi_util_get_hdl_info [lindex $sigList
$l] ]"
        }
    }
}
# -------------------------------------------------------------------------
set resList {}
set sigHdl [ npi_handle_by_name -name {top.f[2:3]} -scope ""]
::npi_L1::npi_trace_driver_by_hdl2 $sigHdl "resList"
dump_res_list $resList stdout
```

npi_util_get_hdl_info输出是字符串，输出格式是：赋值类型，赋值语句，{文件:行号}

5. npi_L1::npi_ut_get_hdl_info

```
proc ::npi_L1::npi_ut_get_hdl_info { hdl { isComposeFullName 0} }
```

主要用来看assign结构左右两边表达式的内容（文本内容），用于脚本分析赋值结构。

注：这个函数对常数的赋值没有把它的位宽给打印出来，会导致无法进行位宽分析，比如下面的情况 `assign a = {b[2:0], 5'b0};` 其hdl_info分析出来可能是 `b[2:0], 'b0`。即无法知道'b0的准确bit位宽。所以没有办法用上此工具。

相关例子，来自到vc_apps_npi.pdf

```
example.v
module top;
wire w1, w2, w3;
assign w1 = w2 & w3;
endmodule
```

```
demo.tcl

source $env(NPIL1_PATH)/npi_L1.tcl
debImport -sv example.v
set hdlList {}
set instHdl [ npi_handle_by_name -name "top" -scope "" ]
set itr [ npi_iterate -type npiNet -refHandle $instHdl ]
while { [ set subHdl [ npi_scan -iterator $itr ]] != "" } {
lappend hdlList $subHdl
}
# get the hdl information and print it
set hdlInfo [ ::npi_L1:npi_ut_get_hdl_info $instHdl ]
puts "$hdlInfo"
# dump the handle information usin L1 API
::npi_L1::npi_ut_dump_hdl_info $instHdl "stdout"
# dump and release the handles in the hdl vector
::npi_L1::npi_ut_dump_hdl_vec_info "hdlList" "stdout" 1
debExit
```

```
Result
npiModule, top, {example.v : 1}
npiModule, top, {example.v : 1}
npiNet, top.w1, {example.v : 2}
npiNet, top.w2, {example.v : 2}
npiNet, top.w3, {example.v : 2}
```

6. npi_mod_inst_get_instance

可以用来查找指定路径下所有实例化的模块，类似于dc里面的get_cells

```
proc npi_mod_inst_get_instance { hier_inst_name inst_hdl_list }
```

```
Parameters
hier_inst_name
Specify the hierarchical name of the target module instance.
inst_hdl_list
Specify the reference to a list of type npiHandle. The iterated instance
handle will be
put into the list.
Return Value
The total number of instances found is returned. 0 if it fails to find the
corresponding
```

```
module instance.
```

Example:

```
example.v
module MOD;
endmodule
module TOP;
MOD inst();
MOD inst2 [3:0] ();
endmodule
demo.tcl
source $env(NPIL1_PATH)/npi_L1.tcl
debImport -sv example.v
set hdlList {}
::npi_L1::npi_mod_inst_get_instance “TOP” “hdlList”
::npi_L1::npi_sv_ut_dump_hdl_vec_info “hdlList” “stdout”
debExit
```

Output:

```
npiModule, TOP.inst, {example.v : 5}
npiModule, TOP.inst2[3], {example.v : 6}
npiModule, TOP.inst2[2], {example.v : 6}
npiModule, TOP.inst2[1], {example.v : 6}
npiModule, TOP.inst2[0], {example.v : 6}
```

This code finds the module instance “TOP” in the loaded design, and collects the instance handles of the instance into the vector “hdlList”.

7. npi_mod_inst_get_port

使用方式跟npi_mod_inst_get_instance类似，用于获取一个模块下面的port列表。

```
proc npi_mod_inst_get_port { hier_inst_name port_hdl_list }
```

8. Properties

8.1 npiCellInstance

module properties, 如果模块定义文件被-y -v修辞或者是`celldefine 和`endcelldefine 表示该module被定义为library 所以其实例化的就是cell instance

有时候需要知道一下这个属性，比如在trace driver的时候，追到cell的passthrough pin就停止了，此时应该把这个Passthrough当做是一个正常的driver其它情况下的passthrough都直接被忽略掉了。 -- 这个是停止上cell的instance那一层，就是net driver log中不会显示cell的信息，所以无法定位到cell 所以想对

这个driver进行计数还得想其它的办法。

-- 不确定是不是可以用下面表格里面提到的tcl command来进一步做尝试识别。

```
set  bool_value [npi_get -property npiCellInstance -object $hdl]
```

8.2 vc_app connection

tcl command	说明
npi_nl_sig_2_mod_inst_conn	Identify the connected module instances from a signal. 只会在模块内部trace，不会trace出模块，这样会比较方便。 需要用stop at port/instport
npi_nl_sig_2_mod_inst_conn_dump	
npi_nl_sig_hdl_2_mod_inst_conn	
npi_nl_sig_hdl_2_mod_inst_conn_dump	
npi_nl_sig_2_primitive_inst_conn	Identify the connected primitive instances from a signal. 原语级别的，不太好用，而且数目特别大，不好分析。还是上面到mod模块级别的更好用。
npi_nl_sig_2_primitive_inst_conn_dump	
npi_nl_sig_hdl_2_primitive_inst_conn	
npi_nl_sig_hdl_2_primitive_inst_conn_dump	

npi_nl_sig_2_mod_inst_conn, nl 是netlist相关的get_str,需要使用`npi_nl_get_str` 命令，而不是`npi_get_str` 命令。

疑问： low_conn和high_conn有什么区别？ VC_APPS_NPI.pdf page1444

2. npiHighConn will indicate the hierarchically higher (closer to the top module) port connection.
3. npiLowConn will indicate the lower (further from the top module) port connection.