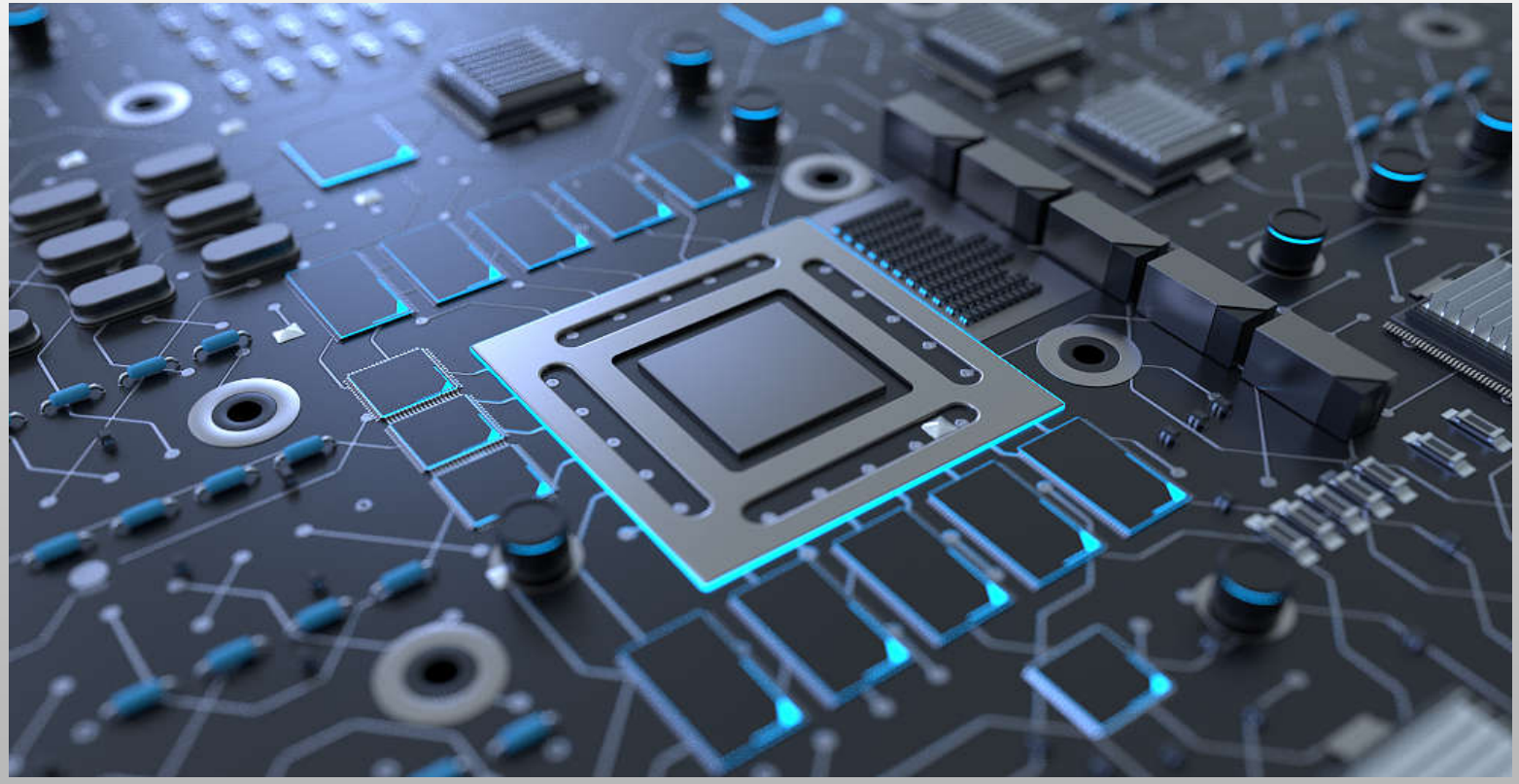
HDLGEN INTRODUCTION

WILSON CHEN

2022-10



CONTENT

OVERVIEW

HDLGen is a tool for HDL generation, it enables embedded Perl or Python scripts in Verilog source code, and support Perl style variable anyway, to generate desired HDL in an easy and efficient way.

It supports all syntax and data structure of Perl or Python, and has a few predefined functions for signal define, module instance, port connection etc.

This tool also supports extended API functions in Perl format(Python API not on plan yet), for any function or module that you want or have.

HDL and script mixed design file can be any name, while final generated RTL file will be Verilog(.v)

Assume line starting with "//:" to be single line Perl script;

Assume line starting with "//:Begin" and ending with "//:End" are multi-line Perl script;

Assume line starting with "//#" to be single line Python script;

Assume line starting with "//#Begin" and ending with "//#End" are multi-line Python script;

```
//: for my $i (0..63) {
//:   print("[7:0] exp_test_$i;\n");
//: }
```

```
//#Begin
    for i in [0,1,2,3]:
        print("wire [15:0] test_pydata%d;" % i )
        print("wire [15:0] test_pyDATA%d;" % i )
//#End
```

```
//:Begin
    for my $i (0..1) {
        &Instance mul_int8_4x4 u_mul_4x4_inst$i;
    }

    for my $i (0..3) {
        &Instance mul_int8_4x4 u_mul_4x4_inst$i;
            Connect /(.*)/  \${1}_mul$i;
    }
//:End
```

```
//: our $reset= " or negedge clk";

always @(posedge clk ${reset})
begin
    q <= d;
    $display("%t:%m: this is a test string\n");
end
```

USAGE - INPUT

So far **HDLGen** supports single source file input only(multi-file is easy to support), generate Verilog HDL with same name. It only need 1 input option default, like:

HDLGen.pm -i my_design.src

and **my_design.v** will be generated as a pure Verilog HDL file.

Other options usage:

-u[usage] : print usage or helping message
-o[output] : override output file
-d[debug] : debug, several intermedia files will be saved to help debug
-v[verbose] : verbal mode, will print a lot of information on screen, if -debug turned on too(rarely used)

Suggestion:

1. Only use (-i) in most case;
2. Turn on (-d) if error message not understood;

```
--- This is a script to read in HDL design file with emdedded Perl/Python scripts in
--- and generate final HDL files with Perl/Python scripts parsed & executed
EX:
   --> ./HDLGen.pm -i HDL_Design.src
   --> ./HDLGen.pm -i HDL_Design.src -o HDL_Design_NewName.v ( default is HDL_Design.v )
   --> ./HDLGen.pm -i HDL_Design.src -d ( run with debug option )
         will print info and store internal data structures,
         Perl/Python scripts are stored in .eperl.pl or .epython.py
   NOTE:
         Currently "&AutoDef" for auto wire a/o reg defines is not perfect, or has bugs yet
         this tool can only parse wire signals as simple as:
                                          assign wire_sig[m:n] = left_sig<[q:p]>
         or parse reg  signals as simple as:
                                            reg_sig<[m:n]> <= dd'h/b...
                                        or: reg_sig        <= dd{1'x...
                                        or: reg_sig        <= left_sig[q:p]
```

USAGE – FLEXIBLE VARIABLES

You can use Perl style variable wherever in Verilog code, as long as you defined such variable before using it.

```
//: our $reset= " or negedge clk";

assign test_wires = test_input[3:0];

always @(posedge clk ${reset})
begin
    q <= d;
    $display("%t:%m: this is a test string\n");
end
```

Note:

1. Such variable can be used wherever as native Verilog code, without any "**//:**";
2. But such variable must be defined as "**our**" type;
3. And these variables must be used with "**{}**", like **${reset},** to differentiate from Verilog embedded functions;

EMBEDDED AND EXTENDED FUNCTIONS

HDLGen has a few embedded functions, which help to achieve RTL generation and IP integration.

NOTE:

1. Function call has 2 ways: **&Function()**, or **Function();**
   "**&**" is optional but suggested;
1. Function parameters have 2 style:
   a) Direct string split by **","**, order is critical;
   b) Linux command **-option** like, order is meaningless;
   Details refer to each function.
3. Embedded function can be used directly;
4. Extended functions need to add package prefix: **&eFunc::**fun(…)

```
//: &eFunc::ClkGen("-clk clk_m");
//: &eFunc::RstGen("-clk clk_m");
//: &eFunc::FuseGen("-clk clk_m");
//: &eFunc::PmuGen("-clk clk_m");
//: &eFunc::MemGen("-clk clk_m");
//: &eFunc::FifoGen("-clk clk_m");
//: &eFunc::AsyncGen("-clk clk_m");
```

```
&Instance test_sys_ctrl_apb_regs;
  Connect -final -interface APB3  -up \${1}_suffix ;
```

```
&Instance NV_NVDLA_CMAC_CORE_MAC_mul u_mul_$i;
  &Connect exp_sft      exp_sft_$ii;
   Connect op_a_dat     wt_actv_data${i};
   Connect op_a_nz      wt_actv_nz${ii};
   Connect op_a_pvld     wt_actv_pvld[${i}];
  &Connect op_b_dat     dat_actv_data${i};
  &Connect op_b_nz      dat_actv_nz${i};
  &Connect op_b_pvld     dat_actv_pvld[${i}];
   Connect /(res_.*)/    \${1}_$ii;
   Connect -final (res_tag)    \${1}_$i; ### override above line
```

```
//:Begin
  my $sv = "
  interface test_if(input clk);
    logic rst_n,
    wire  [1:0] port_a_0 ;
    logic [12:0] port_a_1 ;
    wire port_b_0 ;
    logic port_b_1 ;
   endinterface
    ";
    &AddIntfBySV($sv);
    &PrintIntfPort("-intf test_if -up");
//:End
```

EMBEDDED FUNCTION - print

There are 2 types of print functions in this tool:

1. Perl standard "**print**";
2. Verilog line(s) print as "**vprintl**";

Usage:

//: print("string to print\n");

//: vprint("string to print\n");

NOTE:

1. the difference between 2 functions is that **vprintl** will NOT print to screen at all, it only update internal data structures, while **print** can print info on screen if you write correctly;

2. it's more safe to use "**print STDOUT** …" if you want to print on screen for debug;

3. if you want to print a bulk of code, you can use **"print <<EOF; …EOF";**

5. Python is different! --- it only has Python native **print** function.

```
//: for my $i (0..63) {
//:   print("wire [7:0] exp_test_$i;\n");
//: }
```

```
//:Begin
    for my $i (0..63) {
        my $ii = sprintf("%02d",$i);
        print STDOUT " ---\$ii == $ii ---\n";
        &Instance NV_NVDLA_CMAC_CORE_MAC_mul u_mul_$i;
//:End
```

```
//:Begin
    for my $i (0..9) {
        ### for bulk print
        print <<EOF;
`ifdef DESIGNWARE_NOEXIST
NV_DW02_tree #(8, 36) u_tree_10n0$i (
   .INPUT              (pp_in_10n0${i}[287:0])     //|< r
  ,.OUT0               (pp_out_10n0${i}_0[35:0])   //|> w
  ,.OUT1               (pp_out_10n0${i}_1[35:0])   //|> w
  );
`else
DW02_tree #(8, 36) u_tree_10n00 (
   .INPUT              (pp_in_10n0${i}[287:0])     //|< r
  ,.OUT0               (pp_out_10n0${i}_0[35:0])   //|> w
  ,.OUT1               (pp_out_10n0${i}_1[35:0])   //|> w
  );
`endif
EOF
    }
//:End
```

EMBEDDED FUNCTION - SRC

This function is used to update source file search path, and can be used multiple times for multiple paths. When other functions need a file not in current path, then tool will search in all search paths to find target file(will report error if no file in all paths)

Usage：

```
//: SRC ./incr;
//: &SRC ./cfg;
```

NOTE：

you can use absolute path in other functions, but sometime it's not so convenient;

EMBEDDED FUNCTION - AutoDef

This function is used to automatically generate wire or reg definition for all logic code in 1 source file, that's to say, you can write logic code directly without signal defined first.

But, please note this tool supports very limited syntax so far(**welcome any suggestion a/o solution to improve!**) , like:

assign wire_sig[m:n] = left_sig[q:p]
reg_sig[m:n] <= dd'h/b...
reg_sig <= dd{1'x...
reg_sig <= left_sig[q:p]

Usage:
//: &AutoDef;

NOTE: this function need to put before any wire/reg define lines;

```
//:AutoDef;
```

```
///|: &AutoDef;
///| ==========================================================================
///| ============ Below Wires & Regs are auto-generated by &AutoDef ===========
///| ============ these definition may be not perfect or correct    ===========
///| ============ yout may need to manually update/correct          ===========
///| ==========================================================================
wire [3:0]        test_wires   ;
reg  [64:0]       cfg_is_int8_d0  ;
reg               cfg_reg_en_d0   ;
reg               mac_out_pvld    ;
///| ============================== End of Auto Wires/Regs ======================
///| ==========================================================================
```

```
assign test_wires = cfg_is_wd_d3[3:0];
```

```
always @(posedge nvdla_core_clk or negedge nvdla_core_rstn) begin
  if (!nvdla_core_rstn) begin
    cfg_reg_en_d0 <= 1'b0;
  end else begin
  cfg_reg_en_d0 <= cfg_reg_en;
  end
end
always @(posedge nvdla_core_clk or negedge nvdla_core_rstn) begin
  if (!nvdla_core_rstn) begin
    cfg_is_int8_d0 <= {65{1'b0}};
  end else begin
  if ((cfg_reg_en) == 1'b1) begin
    cfg_is_int8_d0 <= {65{cfg_is_int8}};
  // VCS coverage off
  end else if ((cfg_reg_en) == 1'b0) begin
  end else begin
    cfg_is_int8_d0 <= 'bx;
  // VCS coverage on
  end
  end
end
```

- Suggest to enable;
- But keep in mind this is not perfect
  - It's only a nice to be or backup solution
- Complex logic's signals suggested to manual declare

EMBEDDED FUNCTION - AutoInstWire

This function is to automatically generate module instance's port connected wire define, but it only supports those modules instanced by embedded function of "**&Instance**" , other module instanced by Verilog syntax is not supported yet (can support if requirement exist) **:**

Usage:

//: &AutoInstWire;

NOTE:

1. **This function can be in anywhere, but suggest to be around wire/reg lines;**
2. **port connection is assumed be "wire" type only, if it's "reg" then need manual define and this function will bypass those signals.**

```
//:AutoInstWire
```

```
//:Begin
    for my $i (0..1) {
       &Instance mul_int8_4x4 u_mul_4x4_inst$i;
    }

    for my $i (0..3) {
       &Instance mul_int8_4x4 u_mul_4x4_inst$i;
           Connect /(.*)/  \${1}_mul$i;
    }
//:End
```

```
//|: &AutoInstWire;
//| ========================================================================================
//| ======== Below Wires are for all &Instance modules by &AutoInstWire  ===============
//| ============ these definition may be not correct(for reg signals)     ===============
//| ============ youk may need to manually update/correct                 ===============
// ------ wires of Instance: my_spi & my_spi ------
wire              IPX_clk_i                 ;
wire              IPX_rst_i                 ;
wire              MISO_I_IPX                ;
wire              MOSI_O_IPX                ;
wire              SCK_O_IPX                 ;
wire              SS_O_IPX                  ;
wire              ack_o                     ;
wire [2:0]        adr_i                     ;
wire              cyc_i                     ;
wire [7:0]        dat_i                     ;
wire [7:0]        dat_o                     ;
wire              inta_o                    ;
wire              stb_i                     ;
wire              we_i                      ;
// ------ wires of Instance: u_exp ------
wire [191:0]      dat_pre_exp               ;
wire [63:0]       dat_pre_mask              ;
```

Strongly recommended, it ease your work

EMBEDDED FUNCTION -Instance - 1

This function is used to instance sub-module, its syntax is very similar to Verilog instance, like:

&Instance NV_NVDLA_CMAC_CORE_MAC_mul u_mul_0

Or:

&Instance NV_NVDLA_CMAC_CORE_MAC_mul #(.param0(xxx), .param1(yy) ..) u_mul_0

Or without instance name:

&Instance NV_NVDLA_CMAC_CORE_MAC

NOTE:

1. &Instance codes can be with or without starting head of "**//:**"or"**//:Begin**" "**//:End**", which mean it can be treated as native Verilog code(suggested mode);
2. If &Instance only has module name, then instance will be default as: u_module
3. &Instance need to be used together with Connect in most case, like:

&Instance NV_NVDLA_CMAC_CORE_MAC_mul u_mul_0
&Connect exp_sft exp_sft_00[3:0];
&Connect /op_a_(\w*)/ wt_actv_\${1}0;
&Connect /op_b_(.*)/ dat_actv_\${1}0;

4. If no &Connect line after &Instance line, then all ports of this instance will be connected to wires as same name of port
5. If &AutoInstWire function is called before &Instance, then all wires connected to this instance's ports will be auto-defined at right place;
6. If there is no &AutoInstWire function before &Instance, then all wires connected to this instance's ports will be generated below the instance as commented lines (starting with //), and you can manually copy/change later.

```
&Instance test_sys_ctrl_apb_regs;
  Connect -final -interface APB3  -up \${1}_suffix ;
```

```
//| --- ePerl generated code Begin (DO NOT EDIT BELOW!) ---
//| &Instance("test_sys_ctrl_apb_regs");
//|   Connect -final -interface APB3  -up \${1}_suffix ;
test_sys_ctrl_apb_regs  u_test_sys_ctrl_apb_regs (
    .paddr                      (PADDR_SUFFIX )                 //|<-i
   ,.pclk                       (PCLK_SUFFIX )                  //|<-i
   ,.penable                    (PENABLE_SUFFIX )               //|<-i
   ,.presetn                    (PRESETN_SUFFIX )               //|<-i
   ,.psel                       (PSEL_SUFFIX )                  //|<-i
   ,.pwdata                     (PWDATA_SUFFIX )                //|<-i
   ,.pwrite                     (PWRITE_SUFFIX )                //|<-i
   ,.prdata                     (PRDATA_SUFFIX )                //|>-o
   ,.pready                     (PREADY_SUFFIX )                //|>-o
   ,.pslverr                    (PSLVERR_SUFFIX )               //|>-o
   ,.sys_ctrl0_l2c_strip_mode   (sys_ctrl0_l2c_strip_mode)      //|>-o
   ,.sys_ctrl0_mem_repair_done  (sys_ctrl0_mem_repair_done)     //|>-o
   ,.sys_ctrl0_mem_repair_en    (sys_ctrl0_mem_repair_en)       //|>-o
   ,.sys_ctrl0_pdc_use_arm_ctrl(sys_ctrl0_pdc_use_arm_ctrl)    //|>-o
   ,.sys_ctrl0_smmu_mmusid      (sys_ctrl0_smmu_mmusid)         //|>-o
   ,.test_reg_test_field0       (test_reg_test_field0)          //|>-o
   ,.test_reg_test_filed1       (test_reg_test_filed1)          //|>-o
   );
```

```
&Instance NV_NVDLA_CMAC_CORE_MAC_mul u_mul_2;
  &Connect -final -interface APB   -up ${1}_suffix ;
  Connect exp_sft exp_sft_00[3:0];
  &Connect /op_a_(\w*)/  wt_actv_${1}0;
  Connect /op_b_(.*)/  dat_actv_${1}0;
  &Connect /(res_.*)/    \${1}_00;
  Connect -final res_tag     res_tag_0[7:0];
  &Connect -final -input res_tag     res_tag_0[7:0];
```

```
NV_NVDLA_CMAC_CORE_MAC_mul  u_mul_0 (
    .cfg_is_fp16     (cfg_is_fp16)       //|<-i
   ,.cfg_is_int8     (cfg_is_int8)       //|<-i
   ,.cfg_reg_en      (cfg_reg_en)        //|<-i
   ,.exp_sft         (exp_sft_00[3:0])   //|<-i
   ,.nvdla_core_clk  (nvdla_core_clk)    //|<-i
   ,.nvdla_core_rstn (nvdla_core_rstn)   //|<-i
   ,.op_a_dat        (wt_actv_dat0)      //|<-i
   ,.op_a_nz         (wt_actv_nz0)       //|<-i
   ,.op_a_pvld       (wt_actv_pvld0)     //|<-i
   ,.op_b_dat        (dat_actv_dat0)     //|<-i
   ,.op_b_nz         (dat_actv_nz0)      //|<-i
   ,.op_b_pvld       (dat_actv_pvld0)    //|<-i
   ,.res_a           (res_a_00)          //|>-o
   ,.res_b           (res_b_00)          //|>-o
   ,.res_tag         (res_tag_0[7:0])    //|>-o
   );
```

Thanks NVIDIA for NVDLA as a testing source!

EMBEDDED FUNCTION – Instance - 2

&Instance function has another way to use: **IPXACT** direct instance --- take IPXACT as a module to instance, like:

&Instance simple_spi.xml my_spi;

Or:

&Instance simple_spi.xml #(.param0(xxx), .param1(yy) ..) my_spi;

Or no Instance name: as

&Instance simple_spi.xml;

NOTE:

1. IPXACT file name can be identical or different to module, module name will be defined by IPXACT's "name" field;
   if IPXACT has no "name" field then file name will be used;
2. When instancing from IPXACT file, then all the interfaces defined in the IPXACT file will be automatically updated into internal interface list;
   so you can use those interfaces directly;
3. Other requirements/functions are common for &Instance;

```
&Instance simple_spi.xml my_spi;
 Connect -final -interface spi  -up ${1}_IPX ;
 Connect /(clk.*)/ IPX_${1};
 Connect /(rst.*)/ IPX_${1};
```

```
simple_spi  my_spi (
    .adr_i (adr_i)          //|<-i
   ,.clk_i (IPX_clk_i)      //|<-i
   ,.cyc_i (cyc_i)          //|<-i
   ,.dat_i (dat_i)          //|<-i
   ,.miso_i(MISO_I_IPX )    //|<-i
   ,.rst_i (IPX_rst_i)      //|<-i
   ,.stb_i (stb_i)          //|<-i
   ,.we_i  (we_i)           //|<-i
   ,.ack_o (ack_o)          //|>-o
   ,.dat_o (dat_o)          //|>-o
   ,.inta_o(inta_o)         //|>-o
   ,.mosi_o(MOSI_O_IPX )    //|>-o
   ,.sck_o (SCK_O_IPX )     //|>-o
   ,.ss_o  (SS_O_IPX )      //|>-o
   );
```

EMBEDDED FUNCTION – Instance - 3

&Instance function supports multi-line parameter, but has special requirements, like:

```
&Instance simple_spi.xml
    #( .parm0(0),
       .param1(1),
       .param2(2)
    )
    my_spi;
```

NOTE:

1. When Instancing with multi-line parameter, Instance command must be 3 parts:
   1. 1st line for module or IPXACT name, and has **no ";"** ;
   2. Second line to the line ending of **")"** is for all parameters;
   3. Last line is instance name, ending with **";"** ;

```
&Instance simple_spi.xml
    #( .parm0(0),
       .param1(1),
       .param2(2)
    )
    my_spi;
 Connect -final -interface spi  -up ${1}_IPX ;
 Connect /(clk.*)/ IPX_${1};
 Connect /(rst.*)/ IPX_${1};
```

```
simple_spi
     #( .parm0(0),
        .param1(1),
        .param2(2)
     )
my_spi (
    .adr_i (adr_i)          //|<-i
   ,.clk_i (IPX_clk_i)      //|<-i
   ,.cyc_i (cyc_i)          //|<-i
   ,.dat_i (dat_i)          //|<-i
   ,.miso_i(MISO_I_IPX )    //|<-i
   ,.rst_i (IPX_rst_i)      //|<-i
   ,.stb_i (stb_i)          //|<-i
   ,.we_i  (we_i)           //|<-i
   ,.ack_o (ack_o)          //|>-o
   ,.dat_o (dat_o)          //|>-o
   ,.inta_o(inta_o)         //|>-o
   ,.mosi_o(MOSI_O_IPX )    //|>-o
   ,.sck_o (SCK_O_IPX )     //|>-o
   ,.ss_o  (SS_O_IPX )      //|>-o
   );
```

EMBEDDED FUNCTION – Connect

This function must be working along with **&Instance**, to achieve module's port connections. The function support regular expression for name matching , also support signal grouping by interface (interface can be standard AMBA bus, or manual defined --- as following intro), like:

&Instance NV_NVDLA_CMAC_CORE_MAC_mul u_mul_0
&Connect exp_sft exp_sft_00[3:0];
&Connect /op_a_(\w*)/ wt_actv_\${1}0
&Connect -input /op_b_(.*)/ dat_actv_${1}0;
&Connect -final -interface APB3 -up ${1}_${suffix} ; //connect APB3 bus to wires has "_$suffix", and all wires upcased

NOTE:

1. Must follow &Instance line, no blank line from Instance line (blank line means "ending") ;
2. Can control if only apply to input port(-input) or out port(-output);
3. Regular express is native format, with 2 strings :
   a) 1st is match pattern for port name;
      a) Has "/" or has no "/" will get same result;
   b) 2nd is name change with **$n** supported;
   c) Support variable in express, like **$var**;
   d) Please add **"\"** for regular express matched pattern **(\$1, \$2**);
   e) Please add "**{}**" on variable to avoid any mistake;
4. The wire going to connect can be upcase (-up) or lowcase (-low);
   a) But keep in mind: **-low** is higher priority then-up, only –low action if both enabled.
5. subsequent line's command will override previous lines;
6. But override will be disabled if you enabled with "**-final**"
   a) note: **-final** is highly recommended in most case
7. If you want grouping by interface, then just use "**-interface intf_name**"
   a) But please make sure interface does exist!
      a) Default only standard AMBS bus exist
   b) If need other interface, you need to manually add--- as following intro;
   c) Interface default is "slave" mode --- can be changed by "**-master**" option;

```
for my $i (0..63) {
    my $ii = sprintf("%02d",$i);
    &Instance NV_NVDLA_CMAC_CORE_MAC_mul u_mul_$i;
      &Connect exp_sft      exp_sft_$ii;
       Connect op_a_dat     wt_actv_data${i};
       Connect op_a_nz      wt_actv_nz${ii};
       Connect op_a_pvld     wt_actv_pvld[${i}];
      &Connect op_b_dat     dat_actv_data${i};
      &Connect op_b_nz      dat_actv_nz${i};
      &Connect op_b_pvld     dat_actv_pvld[${i}];
       Connect /(res_.*)/    \${1}_$ii;
       Connect -final (res_tag) \${1}_$i; ### override above line
```

```
NV_NVDLA_CMAC_CORE_MAC_mul  u_mul_0 (
    .nvdla_core_clk  (nvdla_core_clk)          //|<-i
   ,.nvdla_core_rstn (nvdla_core_rstn)         //|<-i
   ,.cfg_is_fp16     (cfg_is_fp16)             //|<-i
   ,.cfg_is_int8     (cfg_is_int8)             //|<-i
   ,.cfg_reg_en      (cfg_reg_en)              //|<-i
   ,.exp_sft         (exp_sft_00[3:0])         //|<-i
   ,.op_a_dat        (wt_actv_data0[15:0])     //|<-i
   ,.op_a_nz         (wt_actv_nz00[1:0])       //|<-i
   ,.op_a_pvld       (wt_actv_pvld[0])         //|<-i
   ,.op_b_dat        (dat_actv_data0[15:0])    //|<-i
   ,.op_b_nz         (dat_actv_nz0[1:0])       //|<-i
   ,.op_b_pvld       (dat_actv_pvld[0])        //|<-i
   ,.res_a           (res_a_00[31:0])          //|>-o
   ,.res_b           (res_b_00[31:0])          //|>-o
   ,.res_tag         (res_tag_0[7:0])          //|>-o
   );
```

```
&Instance test_sys_ctrl_apb_regs;
  Connect -final -interface APB3  -up \${1}_suffix ;
```

```
//| --- ePerl generated code Begin (DO NOT EDIT BELOW!) ---
//| &Instance("test_sys_ctrl_apb_regs");
//|   Connect -final -interface APB3  -up \${1}_suffix ;
test_sys_ctrl_apb_regs  u_test_sys_ctrl_apb_regs (
    .paddr                    (PADDR_SUFFIX )              //|<-i
   ,.pclk                     (PCLK_SUFFIX )               //|<-i
   ,.penable                  (PENABLE_SUFFIX )            //|<-i
   ,.presetn                  (PRESETN_SUFFIX )            //|<-i
   ,.psel                     (PSEL_SUFFIX )               //|<-i
   ,.pwdata                   (PWDATA_SUFFIX )             //|<-i
   ,.pwrite                   (PWRITE_SUFFIX )             //|<-i
   ,.prdata                   (PRDATA_SUFFIX )             //|>-o
   ,.pready                   (PREADY_SUFFIX )             //|>-o
   ,.pslverr                  (PSLVERR_SUFFIX )            //|>-o
   ,.sys_ctrl0_l2c_strip_mode (sys_ctrl0_l2c_strip_mode)   //|>-o
   ,.sys_ctrl0_mem_repair_done(sys_ctrl0_mem_repair_done)  //|>-o
   ,.sys_ctrl0_mem_repair_en  (sys_ctrl0_mem_repair_en)    //|>-o
   ,.sys_ctrl0_pdc_use_arm_ctrl(sys_ctrl0_pdc_use_arm_ctrl)//|>-o
   ,.sys_ctrl0_smmu_mmusid    (sys_ctrl0_smmu_mmusid)      //|>-o
   ,.test_reg_test_field0     (test_reg_test_field0)       //|>-o
   ,.test_reg_test_filed1     (test_reg_test_filed1)       //|>-o
   );
```

EMBEDDED FUNCTION – Interface

There are several functions for Interface management, you can use them to add interface from RTL file, JSON file, or Perl hash, or embedded SV code, then subsequent code can use these interfaces to print or connect, or do whatever you want, like:

```
//: &AddIntfByIPX("./cfg/simple_spi.xml");
//: &AddIntfByJson("./cfg/MyIntf.json");
//: &PrintIntfPort("-intf spi");
```

```
//:Begin
  my %APB3 = (
       'pSELx' => "input: 1",
       'pADDR' => "input: 32",
       'pENABL' => "input: 1",
       'pRDATA' => "output: 32",
       'pREADY' => "output: 1",
       'pSLVERR' => "output: 1",
       'pDAT' => "input: 32"
  );
  &AddIntfByHash(\%APB3,"test_APB3");
  &PrintIntfPort("-intf test_APB3 -awd 12 -d 32 -pre my_ -port -end ;");
//:End
```

You'll get code as:

```
port                    ss_o            ;
port                    sck_o           ;
port                    mosi_o          ;
port                    miso_i          ;
```

```
output                  my_pSLVERR      ;
input   [31:0]          my_pDAT         ;
output  [31:0]          my_pRDATA       ;
output                  my_pREADY       ;
input                   my_pENABL       ;
input   [11:0]          my_pADDR        ;
input                   my_pSELx        ;
```

NOTE: detail usage please refer to sample code, or user guide released later

EMBEDDED FUNCTION – Interface

- Add or modify embedded interfaces

AddIntfByIPX	Read in PXACT file， Parse IPXACT(xml)file for interface and add all of them into embedded list	**&AddIntfByIPX("./cfg/simple_spi.xml");**
AddIntfByJson	Read in JSON file， parse JSON signal define， add all of then as an interface into embedded list	**&AddIntfByJson("MyIntf.json");**
AddIntfByRTL	Read in RTL file, parse all port, filter with keyword(optional)， add as an interface into embedded list	**&AddIntfByRTL("MyIntf.v“. “MyIntfName”, “key_word”);**
AddIntfBySV	use SystemVerilog code to define interfaces, add all of them into embedded list	**&AddIntfBySV($sv_code);**
AddIntfByHash	Add Perl hash as an interface, into embedded list	**&AddIntfByHash(\%MyIntf, “MyIntf”, “key_name”);**
AddIntfByName	Add 1 port into an existing embedded interface	**&AddIntfByName("clk “， ”input:1", "intf_name");**
RmIntfPort	remove a port from an existing embedded interface	**&RmIntfPort(“clk“, “intf_name”);**

- for 3rd part or inhouse IP integration

PrintIntfPort	Print out an interface as signal list, according to config options	**&PrintIntfPort("-intf MyIntf -awd 18 -dwd 32 -pre Test_ -low-port -master -end ,");**
PrintAmbaBus	Print out standard AMBS bus signal list, according to config options	**&PrintAmbaBus("-type test_APB3 -awd 18 -dwd 32 -pre Test_ -suf _End -up -wire -end ,");**
ShowIntf	Show and interface into a file as an hash array(for debug)	**&ShowIntf(“intf_name”);**

EMBEDDED FUNCTION - IPXACT

ReadIPX	Read in IPXACT file, parse IPXACT(xml) interface info， add all of then io embedded list	**&ReadIPX("./cfg/simple_spi.xml");**
ShowIPX	Read in IPXACT file, parse IPXACT(xml) interface info, and write interface info into a file for debug	**&ShowIPX("./cfg/ipx.xml");**
GenIPX	Generate a standard IPXACT file (xml) as what RTL looks like	**&GenIPX("my_design", "MyCorp");** **On plan but not start**

- For 3d part IP integration
- And SOC integration

EXTENDED API

- For inhouse design development
- & IP a/o SOC integration

ClkGen	Generate Clock Generation Module，including MUX, DIV, ICG, with parameters	**On plan but not start**
RstGen	Generate Reset Generation Module, with parameters	**On plan but not start**
PMUGen	Generate PMU module, with parameters	**On plan but not start**
FuseGen	Generate Fuse module, with parameters	**On plan but not start**
AsyncGen	Generate Async interface, with or without fifo	**On plan but not start**
FifoGen	Generate various fifo design, according to different parameters	**On plan but not start**
MemGen	Generate sram design, based on foundary config and input constraints	**On plan but not start**
…	…	…

Any suggestion or solution or contribution is warmly welcomed!

OTHER FUNCTIONS

CallCmd	call Shell/Perl/Python command	**&CallCmd("create_design.py -n my_design -d 32 -a 18");**
DTIWire	Generate DTI interface as wire signals，name prefix and data width is necessary	**&DTIWire("top2me", 512);**
DTISlave	Generate DTI interface as slave signals，name prefix and data width is necessary	**&DTISlave("top2me", 512);**
DTIMaster	Generate DTI interface as master signals，name prefix and data width is necessary	**&DTIMaster"top2me", 512);**
…	…	…

- For inhouse design development

Thanks PyGear for the name of "DTI"

Verilog is the King

Connection is all you need

Flexibility is cool

Thanks and Notice

Thanks NVIDIA for giving me the chance to know how powerful Perl is to run a big ASIC factory;
Thanks NVIDIA's VIVA to let me know how Perl can make Verilog easy, interesting and amazing;

Please note this tool was developed from scratch during the special spring time in Shanghai in 2022;
The things related to NVIDIA are:

- several function names are identical;
- several HDL files of open sourced NVDLA are used to be test source

Please kindly let me know if there is any license issue