Hierarchical System Builder Toolset

Madhav P. Desai
Department of Electrical Engg.
IIT Bombay, Mumbai India

January 10, 2025

Overview

- The hierarchical system builder toolset is intended for assembly of a system in terms of sub-systems.
- The interfaces of the system and sub-systems are either *pipes* or *signals*. Pipes can be blocking or non-blocking.
- It is possible to embed register transfer level fragments in the system description in order to model glue logic.

Hierachical system description format

A system description takes the following form:

```
$system system_name $library system_library
   $in
        ... input pipes and signals ...
   $out
        ... output pipes and signals ...
{
   ... (optional) internals of the system ....
}
```

The description is kept in a file with extension ".hsys". For example

```
system_name.hsys
```

A simple system

```
// comments start with //
$system sysmem $library sysmem_library
   $in
       // input pipe bus_request, width 32, depth 2
       $pipe bus_request 32 $depth 2
   $out
       // output pipe bus_response, width 32, depth 2
       $pipe bus_response 32 $depth 2
{
   // empty..
}
```

This describes a leaf level system which could have been implemented by translation from Aa to vhdl by the AHIR V2 toolset. It's VHDL description is supposed to be compiled into the library sysmem_library.

Hierarchy

A 4-stage shift register is constructed out of two simpler 2-stage shift registers.

```
$system shift_reg_4 $library shiftreg_library
    $in
        $pipe in_data 32 $depth 2
    $out
        $pipe out_data 32 $depth 2
{
    // internal pipe description..
    $pipe mid_data 32 depth 2
    $instance sr_left shiftreg_library :  shift_reg_2
                  in_data => in_data
                  out_data => mid_data
    $instance sr_right shiftreg_library :  shift_reg_2
                  in_data => mid_data
                  out_data => out_data
}
```

Hierarchy: continued

shift_register_2 may itself be build out of 1-stage shift registers.

```
$system shift_reg_2 $library shiftreg_library
    $in
        $pipe in_data 32 $depth 2
    $out
        $pipe out_data 32 $depth 2
{
    // internal pipe description..
    $pipe mid_data 32 depth 2
    $instance sr_left shiftreg_library :  shift_reg_1
                  in_data => in_data
                  out_data => mid_data
    $instance sr_right shiftreg_library :  shift_reg_1
                  in_data => mid_data
                  out_data => out_data
}
```

Hierarchy: reached the leaf

```
$pipe in_data out_data: $uint<32> $depth 2

$module [shift_reg_1_daemon] $in () $out () $is
{
  $branchblock[loop] {
    $merge $entry loopback $endmerge
    X := in_data
    out_data := X

    $place [loopback]
  }
}
```

Hierarchy: build organization and methodology. See examples/shift_reg_4.

Organize the system into directories which reflect the hierarchy.

```
Top directory
    shift_reg_4/

Contains
        shift_reg_2/

which contains
            shift_reg_1/
```

Hierarchy: build organization and methodology continued

At the leaf directory shift_reg_1, we may have an **Aa** project which we used to generate VHDL and an aa2c model etc. At non-leaf directories, we describe the hierarchy at that level in an hsys file.

```
shift_reg_4/
    contains hsys file
    shift_reg_2/
      contains hsys file
      shift_reg_1/
         contains Aa project
```

Hierarchy: build scripts and make-files at each level

```
shift_reg_4/
    build.sh script generates aa2c of entire hierarchy
    and VHDL at this level.
    shift_reg_2/
       Makefile generates VHDL model at this level
      shift_reg_1/
          Makefile generates aa2c and VHDL of leaf
          shift_reg_1
```

Testbench: See examples/shift_reg_4/tb

- See tb/tb.c, and compile.sh
- Use testbench tb_aa2c to validate the aa2c model.
- Use testbench tb_ghdl to validate the VHDL model.

Glue logic insertion using RTL

- In addition to structural descriptions, we can use the hsys file to describe glue logic using an RTL description.
- For example, suppose we wish to process the result coming out of the left shift_register_2 by complementing it.
- We will insert a state machine between the left and right instances of shift_register_2 in the top level shift_register_4.
- Note: this mechanism cannot be verified by C simulations, but only by VHDL simulations.

RTL State Machine

```
$thread complementerFsm
   $in $pipe input_data: $unsigned<32>
   $out $pipe ouput_data: $unsigned<32>
   $variable  data_reg: $unsigned<32>
   $constant z1  : $unsigned<1> := ($unsigned<1>) 0
   $constant o1  : $unsigned<1> := ($unsigned<1>) 1
$default
   //... continued...
```

RTL State Machine: continued

```
$now input_data$req  := o1 // default value of pipe reac
$now output_data$req := z1 // default valu of pipe write
<st_empty> {
        $if input_data$ack { // input pipe has data
            data_reg := (~ input_data) // register!
            $goto st_full
        }
}
<st_full> {
        $now output_data$req := o1 // request write to pi
        $now output_data := data_reg // data to pipe
        $if output_data$ack { // pipe accepts..
              $goto st_empty
        }
        $else { $goto st_full }
}
$now $tick
```

RTL State Machine: instantiating

```
$system shift_reg_4 $library shiftreg_library
   $in
       $pipe in_data 32 $depth 2
   $out
       $pipe out_data 32 $depth 2
{
   $pipe mid_data_left mid_data_right 32 depth 2
   $instance sr_left shiftreg_library :  shift_reg_2
                in_data => in_data
                out_data => mid_data_left
   $instance sr_right shiftreg_library :  shift_reg_2
                in_data => mid_data_right
                out_data => out_data

   // thread description of complementFsm not shown...

   $string complement_inst: complementFsm
      input data => mid data left output data => mid data
```

Using RTL State Machines: example shift_reg_4_rtl

- Essentially the same as shift_reg_4 example, but without aa2c simulation setup.
- Examine the build and make files to see the differences.
- Note: only VHDL sim verificationis supported for now.

Clocks and resets

- By default, each block in the system operates on clock *clk* and reset *reset*. The reset is active high and synchronized to *clk*.
- It is possible to define alternative clock and reset signals used in a system.
- It is possible to map one of these alternative clocks and resets to the default clock/reset on a subsystem instance.

Declaring clock-like and reset-like signals

```
$system Top $library TopLib
    $in
      $pipe A
      $signal $clk EXTCLK 1 $depth 1
      $signal $reset EXTRESET 1 $depth 1
    $out
      $pipe B
{
}
```

Here, we are stating that the input signal EXTCLK is clock-like and the input signal RESET is reset-like.

Using clock-like and reset-like signals

```
// ... as above ...
{
 $pipe TMP $clk => EXTCLK $reset =>  EXTRESET
 $instance i0  S1:Stage1
          ..
          $clk => EXTCLK
          $reset => EXTRESET

 $instance i1  S2:Stage2
          ...
          $clk => EXTCLK
          $reset => EXTRESET
```

Using clock-like and reset-like signals

- We have declared EXTCLK as clock-like and EXTRESET as reset-like.
- Pipe TMP and instance i0 will operate with EXTCLK connected to their default clock input and EXTRESET connected to their default reset input.
- We have generated a local clock EXTCLK_REPEATED using an RTL machine.
- Instance i0 will operate with EXTCLK_REPEATED connected to its default clock input and EXTRESET connected to its default reset input.
- Only a clock-like signal can be mapped to the default clock.
- Only a reset-like signal can be mapped to the default reset.

Example: using clock-like and reset-like signals

See the example clocks_and_resets.

Summary

- Convenient way to assemble large systems which use pipes and signals to communicate.
- Build script: buildHierarchicalModel.py can generate aa2c and vhdl models.
- Lots of error checking: dangling connections, width mismatch, direction mismatch.
- Can embed glue RTL in description files
  - If you use embedded RTL, then only vhdl sims are possible for verification.
- Tried and proven tools in AJIT, NAVIC project!