# Transitioning from the Behavioral Synthesis to Physical Synthesis

Peter Maina Mwihaki

June,1,2024

## 1 Introduction

After completing the behavioral synthesis of the simple adder using ALLIANCE CAD tools, we need to proceed to the physical synthesis. The physical synthesis requires a structural file (vst) in order for the flow to work. In this document, we will look into how to generate the structural file for the synthesis transition.

## 2 Preview

The physical synthesis stage comes after the behavioral synthesis. We obtained these top-level modules from the behavioral synthesis:

1. `file_boog.vst`

```
entity adder_boog is
   port (
      a    : in      bit;
      b    : in      bit;
      cin  : in      bit;
      sum  : out     bit;
      cout : out     bit;
      vdd  : in      bit;
      vss  : in      bit
 );
end adder_boog;

architecture structural of adder_boog is
Component oa2ao222_x2
   port (
      i0  : in      bit;
      i1  : in      bit;
      i2  : in      bit;
      i3  : in      bit;
      i4  : in      bit;
      q   : out     bit;
      vdd : in      bit;
      vss : in      bit
 );
end component;

Component xr2_x1
   port (
      i0  : in      bit;
      i1  : in      bit;
      q   : out     bit;
      vdd : in      bit;
      vss : in      bit
 );
end component;

signal xr2_x1_sig : bit;

begin

cout_ins : oa2ao222_x2
   port map (
      i0  => a,
      i1  => b,
      i2  => b,
      i3  => a,
      i4  => cin,
      q   => cout,
      vdd => vdd,
      vss => vss
   );

xr2_x1_ins : xr2_x1
   port map (
      i0  => a,
      i1  => b,
      q   => xr2_x1_sig,
      vdd => vdd,
      vss => vss
   );

sum_ins : xr2_x1
   port map (
      i0  => xr2_x1_sig,
      i1  => cin,
      q   => sum,
      vdd => vdd,
      vss => vss
   );

end structural;
```

2. file_loon.vst

```
entity adder_loon is
   port (
      a    : in      bit;
      b    : in      bit;
      cin  : in      bit;
      sum  : out     bit;
      cout : out     bit;
      vdd  : in      bit;
      vss  : in      bit
 );
end adder_loon;

architecture structural of adder_loon is
Component oa2ao222_x2
   port (
      i0  : in      bit;
      i1  : in      bit;
      i2  : in      bit;
      i3  : in      bit;
      i4  : in      bit;
      q   : out     bit;
      vdd : in      bit;
      vss : in      bit
 );
end component;

Component xr2_x1
   port (
      i0  : in      bit;
      i1  : in      bit;
      q   : out     bit;
      vdd : in      bit;
      vss : in      bit
 );
end component;

signal xr2_x1_sig : bit;
```

```
begin

cout_ins : oa2ao222_x2
   port map (
      i0  => a,
      i1  => b,
      i2  => b,
      i3  => a,
      i4  => cin,
      q   => cout,
      vdd => vdd,
      vss => vss
   );

xr2_x1_ins : xr2_x1
   port map (
      i0  => a,
      i1  => b,
      q   => xr2_x1_sig,
      vdd => vdd,
      vss => vss
   );

sum_ins : xr2_x1
   port map (
      i0  => xr2_x1_sig,
      i1  => cin,
      q   => sum,
      vdd => vdd,
      vss => vss
   );

end structural;
```

These files are essential for the next stage. In this transition, we will use a tool called genlib to come up with the top-level description file (`.vst`).

## 2.1 GENLIB

Genlib is a tool that uses C language to compile the modules and produce a single `vst` file as output. In our case of a simple adder, we will need to observe what kind of inputs we have and create the following C program:

```c
#include "/home/pierre-tfie/alliance/install/include/genlib.h"

int main() {
    // Define the top-level circuit
    GENLIB_DEF_LOFIG("top_level");

    // Define ports of the top-level design
    GENLIB_LOCON("a", IN, "a");
    GENLIB_LOCON("b", IN, "b");
    GENLIB_LOCON("cin", IN, "cin");
    GENLIB_LOCON("sum", OUT, "sum");
    GENLIB_LOCON("cout", OUT, "cout");
    GENLIB_LOCON("vdd", IN, "vdd");
    GENLIB_LOCON("vss", IN, "vss");

    // Instantiate the existing modules
    GENLIB_LOINS("adder_boog", "U_boog", "a", "b", "cin", "sum", "cout", "vdd
    GENLIB_LOINS("adder_loon", "U_loon", "a", "b", "cin", "sum", "cout", "vdd

    // Save the top-level design
    GENLIB_SAVE_LOFIG();
    return 0;
}
```

### 2.1.1 Concept of U_boog and U_loon

In our top-level design, `U_boog` and `U_loon` are instances of the modules `file_boog` and `file_loon`, respectively. These modules are defined in `file_boog.vst` and `file_loon.vst`.

### 2.1.2 Explaining the entry of the existing module

```
GENLIB_LOINS("adder_boog", "U_boog", "a", "b", "cin", "sum", "cout", "vdd", "vss", 0);
```

- `"adder_boog"`: The name of the model being instantiated.
- `"U_boog"`: The instance name.
- `"a", "b", "cin", "sum", "cout", "vdd", "vss"`: These are the signal names connected to the ports of `adder_boog`.
- `0`: This terminates the list of signal names. It indicates that there are no more ports to connect for this instance.

The zero is crucial for the function to understand where the list of port connections ends. Without it, the function wouldn't know where the argument list terminates, leading to potential errors or undefined behavior.

## 2.2 The Next Stage (Top Level Description)

We need to obtain a top-level description of the two files which have using `genlib` command:

```
genlib top_level.c -o top_level.vst
```

If `genlib` runs without errors, our output will be the `top_level.vst`:

```
entity top_level is
   port (
      a    : in      bit;
      b    : in      bit;
      cin  : in      bit;
      sum  : out     bit;
      cout : out     bit;
      vdd  : in      bit;
      vss  : in      bit
 );
end top_level;

architecture structural of top_level is
Component adder_boog
   port (
      a    : in      bit;
      b    : in      bit;
      cin  : in      bit;
      sum  : out     bit;
      cout : out     bit;
      vdd  : in      bit;
      vss  : in      bit
 );
end component
Component adder_loon
   port (
      a    : in      bit;
      b    : in      bit;
      cin  : in      bit;
      sum  : out     bit;
      cout : out     bit;
      vdd  : in      bit;
      vss  : in      bit
 );
end component;

begin
```

```
u_boog : adder_boog
   port map (
      a    => a,
      b    => b,
      cin  => cin,
      sum  => sum,
      cout => cout,
      vdd  => vdd,
      vss  => vss
   );

u_loon : adder_loon
   port map (
      a    => a,
      b    => b,
      cin  => cin,
      sum  => sum,
      cout => cout,
      vdd  => vdd,
      vss  => vss
   );

end structural;
```

We can now use the `top_level.vst` to proceed with the physical synthesis.