SOCGEN Project

Naming Guidelines

As designs continue to grow in size along with the size of the design team, it is mandatory that all code must written to adhere to name space guidelines. The days when a designer could simply pull names out of thin air are faster disappearing. Guidelines have two functions. First they ensure that no two designers select the same name for different objects and have a name collision. The second function is to ensure that the chosen names are meaningful to all of the design team. It is quite common for designers to select names that only make sense to themselves and no one else on the team.

Signal ,Port and Pad Names

Signals define the nodes inside of a component and each node must have a unique name. That signal name becomes the port name when a node is ported up the hierarchy. The port names become the pad names at the top level. All of these exist in the same name space along with other items such as instance names. Managing this name space is crucial.

There are two distinct groups that use these names. The IC design team is one group and it will use all three. The other group consists of System designers,PCB designers, Board Test engineers etc.
They only access the chip via the pad names and never see the internal ones. These two groups have incompatible objectives. The IC design team is dealing with millions of names and needs a naming scheme that produces long descriptive names that won't collide and conveys information about the signals function.

The rest of the world is only dealing with a few hundred or thousand names. They also have their own naming requirements. These typically are:

- Short Names that fit on a schematic graphic symbol. If you have 99 short names and 1 long one then you have a long column and wasted white space on your schematic.

- Capital Letters. They make a packed schematic readable. You don't want your board designers trying to guess if it's a 1 or a l.

- ATE naming requirements. Do you know what the IEEE 1149.1 pad naming rules are? If not then you shouldn't be selecting pad names.

The guideline for selecting pad names is that the IC design team should not attempt to pick pad names based on the internal signal names. They should first meet all of the PCA customers requirements without regard to what names are chosen for the internal signals. Name collisions are avoided by ensuring that ALL pad names start with a capital letter and that all internal names start with a small one.

For internal signal and port names you must first find the four pieces of information that will uniquely identify every signal in the design. These are:

- Interface Name You don't want 5 different ways to spell clock in a design. Each team must agree on common signal names and everyone must follow the rules. These are called standard interfaces. The team must create a document that lists all the standard interfaces and their names. It is ESSENTIAL that once a standard is chosen then all signals covered by that standards MUST follow the naming rules and the no signals that are not covered by the standard are allowed to use its name.

- Sub_member If the standard interface has more than one signal then you must also define the names for each sub_member as part of the standard

- ad hoc If a signal is not defined by a standard interface then an ad hoc signal can be created based on the designers insight. If a module has 2 or more signals with the same standard interface then a ad hoc field is needed to distinguish between them.

- Driving Instance This is the instance name that is driving the signal. Wired or tristate logic is not allowed. There will be one and only one driver per node.

You can create signal names by simply gathering this information and concatenating it into a name but it is perfectly acceptable to drop any field(s) if they are not needed to uniquely identify a node.
For example a IC design may have a signal named "clk". Clk is the standard interface name for a clock signal so we know that it is a clock. The clock interface has two sub_members - rising edge and falling edge. If you have N sub_members then you only have to identify N-1 of them. In this case the standard chooses _n for falling edge clocks and nothing for rising edge. clk is a rising edge clock. An ad hoc field is needed if the design has more than one clock and we have several - 2x, 4x 1.5 x etc. But again we only have to add this to

all but one of the clocks. clk is a 1x clock. This design only has one clock generator so we don't need to add the driving instance. If a second clock generator is added then all of those clocks must include the driving instance in their name.

Besides defining all of the standard interfaces the design team must also define a field separator such as _ (underscore) as the way to separate the different fields that are combined to make a signal or port name. But the most important decision of all is the order that the fields are assembled to make up a name. This is like the Big Endian/Little Endian issue. They both have their strengths and weaknesses and it really doesn't matter which one you pick. BUT it is essential that the design team picks one and everybody does it that way.

Signal and port names are even worse because with four fields you can have 24 possible signal names for each node. Unless everyone on the design team adheres to one order then it will be chaos when you try to architect and synthesize a design.

The recommended order for fields in a signal/port name is

Driving_instance_(sep)_Ad hoc_(sep)_Interface_(sep)_Sub_member

This ordering gives us the ability to have our signal names follow their function as the signals pass up and down the hierarchy. It also gives us an easy rule to follow when we need to pick a signal name. All that you need to do is find the instance name of the module that is driving that signal and combine it with the port name from that module. Since instance names are unique inside a design and port names are unique inside of a module then this rule guarantees that no other signal will use that name.

By following this rule a signal name will grow as it progresses up the hierarchy. At each new level a new instance name is stuck on the front end and the instance name from the lower level becomes part of the ad hoc field. Each name contains a history of how it was created and what it does.

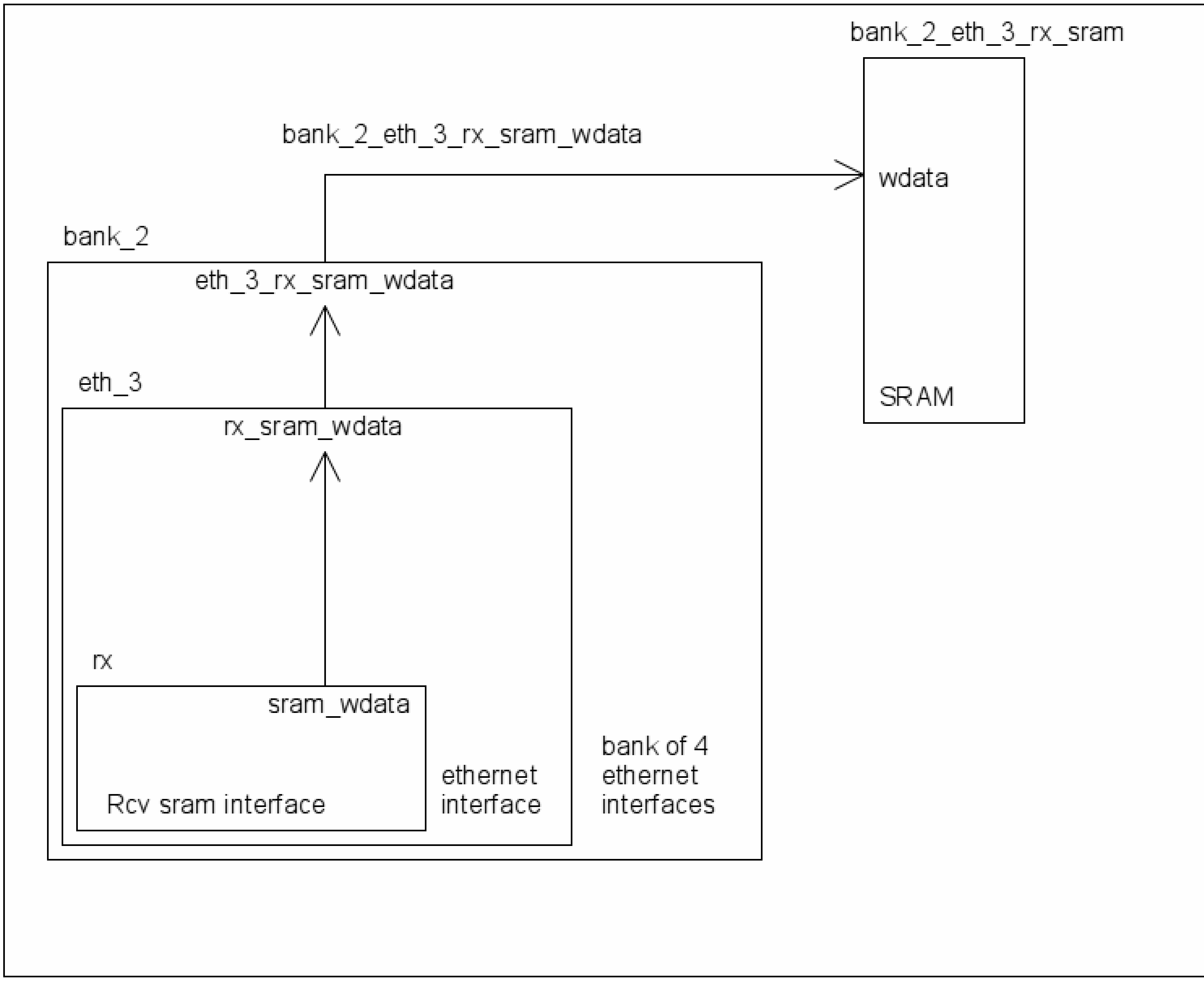
There are some special cases that can occur and these rules should be followed:

- There is one sub_member that can be used on a ad hoc signal that is not defined as a standard interface. That sub_member is active low (_n).

- If a standard interface includes a signal that is itself defined as another standard interface then the interface name of the child becomes the sub member name for the parent. This usually occurs when a clock or reset is included in a bus interface. This ensures that when parsing the signal name it will match on both interfaces. If there are multiples of this interface then a ad hoc field must be perpended to the sub_member interface.

- If the driving instance is not known such as a module where the signal is an input port then the instance and port of a receiving instance may be used instead.

Again remember that any field may be dropped if it is not needed to uniquely identify the node



Here is an example of how this works in a real design. A router IC has 24 instances of a ethernet interface. Each instance controls a transmit sram buffer and a receive sram buffer. There are four instances in a bank and the bank is instantiated six times. The receive write data for the third interface in the second bank originates in a register bank deep inside a submodule. The name of this register is sram_wdata and that was chosen because the sram bus is a standard interface and wdata is the sub_member for the wdata. As it passes through the hierarchy the driving instance name is prepended on the front. It always parses as a sram wdata signal but the ad hoc field keeps growing.

If the clock signal is also included in the sram interface then it's name would be:

```
bank_2_eth_3_sram_clk
```

It would parse as both a clock signal and a sram signal. If this were a dual port sram then the signal would be:

```
bank_2_eth_3_sram_a_clk
```

Notice that there is an ad hoc field both before and after the sram interface name and it still parses as both a clock and a sram signal.

If you want to synthesize the bank of 4 controllers then you will need to set an output delay on the sram outputs as a placeholder for the setup and routing delays in the full chip. To do this you need the full instance name of the source registers as seen from the top level. Assuming you use the standard _reg convention it would be:

```
eth_3/rx/sram_wdata_reg
```

(tic) statements

(tic) statements are an extension to the verilog language where variables may be defined using a `define statement and these values are then used to reconfigure the components. (tic) statements all share a global name space so managing these names is crucial. The following names are reserved and must never be used for any other purpose.

`define TIMESCALE 1ns/1ns

loads in the simulations default timescale value

`define TIMEFORMAT $timeformat(-9, 2, " ns", 14);

contains the verilog function that controls the timeformat in the log file

`define PERIOD 40.000

set by the simulation to the simulations master clock period in `TIMESCALE units `define SYNTHESIS

set by synthesis tools to exclude all non-synthesizable constructs

`define VCD

set in a simulation when a value change data dump file is needed