New Innovative Way to Verify Package Connectivity

Mike Walsh (mike.walsh@siemens.com) Siemens EDA Cary, NC
Jin Hou (hou.jin@siemens.com), Siemens EDA Fremont, CA

***Abstract*-Integration of multiple ICs in a single package is critical for high performance computing. Due to the huge number of connections after package the ICs, it is hard to verify the correctness of the connections. The traditional way to verify the connections requires a lot of manpower and time and is either not exhaustive or too late in the process. This paper will introduce a new way to verify the package connectivity using formal verification that can exhaustively verify all interconnections between the IC blocks. The flow is automatic for all steps from creating connectivity spec to verify package output connectivity. The automatic parallel algorithms on compute grid can verify huge numbers of connections in minutes even seconds. The script for the flow is simple and only takes a few minutes to setup. Once the script is ready, it can be reused for different package projects.**

I. Introduction

Historically IC Package design has been a relatively simple task which allowed the die bumps to be fanned out to a geometry suitable for connecting to a printed circuit board. The package netlist was often captured by the package designer typically using Excel to manually assign net names to the desired die bumps and BGA balls to achieve the intended connection.

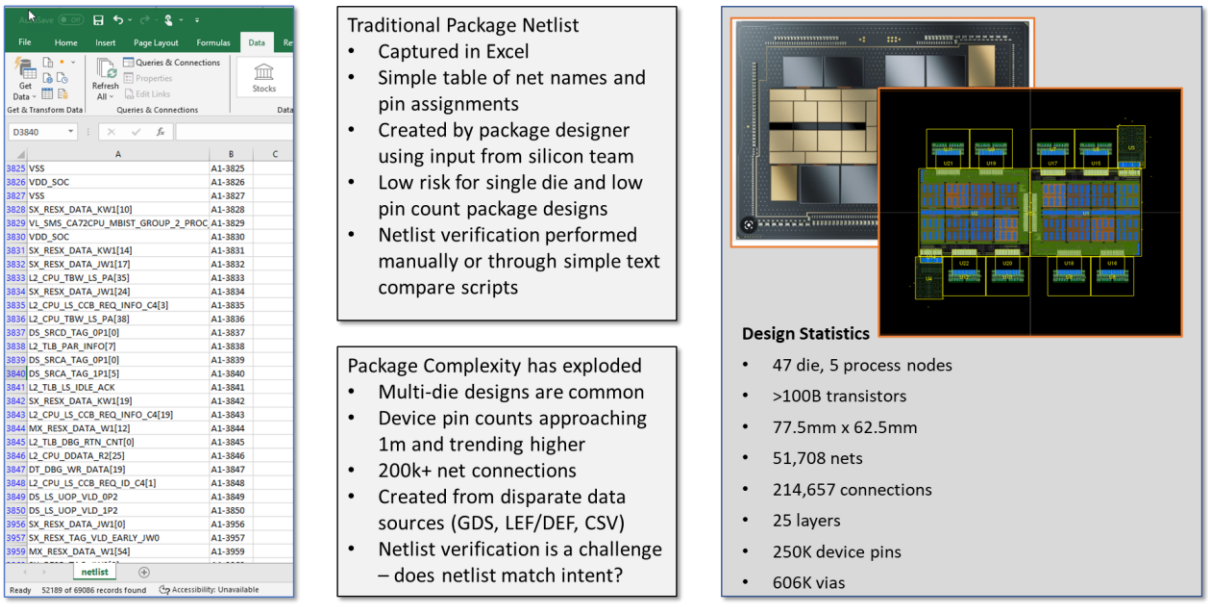


Figure 1. History of IC package design

Modern package and interposer design has become a system integration task: The designer has the responsibility to take input from various stakeholders, who are often designing their content at the same time the package or interposer is being designed and create a design which is both electrically and physically correct, but functions as designed too.

For the purposes of this paper, the term "substrate" will be used to generically represent both the package and interposer design. For many design and verification tasks, the target implementation of either package or interposer can be used interchangeably as the design task and verification tasks are similar despite the differences in manufacturing technology and implementation tools.

II. Challenges

With the rapid advances in package technology, the explosion of AI and HPC applications, substrate designers are facing design challenges which are breaking their existing methodologies. Designs with 500,000 bumps and several hundred thousand connections are becoming commonplace. With hybrid bonding technology promising millions of bump connections, a spreadsheet is no longer capable of managing the complex connectivity.

Heterogeneous integration has brought with it some challenges for the package designer. The source data is being supplied in a myriad of data formats:

- Ball map CSV files
- LEF/DEF from P&R tools
- GDS
- Verilog RTL
- Spreadsheet Data
- Plain text files

As each component of the design is introduced, it must be connected to the other components in the system. Spreadsheet based design requires every connection be defined as a scalar. One can quickly see how High Bandwidth Memory (HBM) based design with its 1024[1] bit width would become tedious and error prone to define the connectivity one bit at a time.

To handle the explosion of die-to-die connections, substrate designers are beginning to embrace language-based design to define the connectivity of the system. It is far more efficient and less error prone to write Verilog RTL using proper bus notation to connect the various components of a system together than it is to define the connectivity one bit at a time in a spreadsheet or develop specialized Excel macros to populate a connectivity table. Spreadsheet based solutions, even with custom macro development, do not scale to hundreds of thousands or millions of bumps.[2]

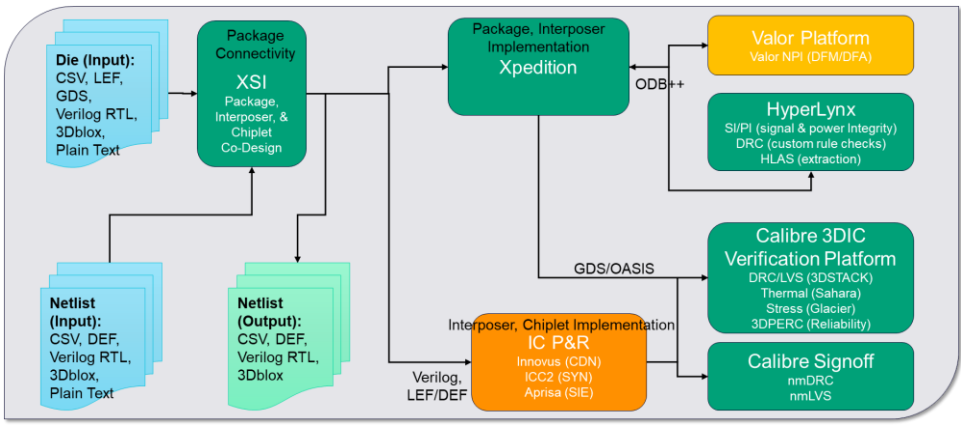


Figure 2. Current flow of package design and verification

Xpedition Substrate Integrator (XSI[3]) from Siemens EDA, helps engineers address the challenges of modern substrate design. In the Siemens substrate design flow illustrated above, the source data on the left is aggregated from the source die, BGA, and netlist data. Once all the data has been imported into XSI, the netlist and physical data can be written out for the appropriate implementation tool(s). Below is an example of Verilog source imported into XSI.

[1] HBM memory bus is very wide in comparison to other DRAM memories such as DDR4 or GDDR5. An HBM stack of four DRAM dies (4-Hi) has two 128-bit channels per die for a total of 8 channels and a width of 1024 bits in total. (from Wikipedia: https://en.wikipedia.org/wiki/High_Bandwidth_Memory)

[2] The maximum number of rows in Excel is 1,048,576 (from https://support.microsoft.com/en-au/office/excel-specifications-and-limits-1672b34d-7043-467e-8e27-269d656771c3)

[3] Xpedition Substrate Integrator, XSI, from Siemens EDA enables heterogeneous and homogeneous 2.5/3D IC Package Connectivity Planning, assembly prototyping, & system technology co-optimization. See: https://eda.sw.siemens.com/en-US/ic-packaging/software/substrate-integrator/

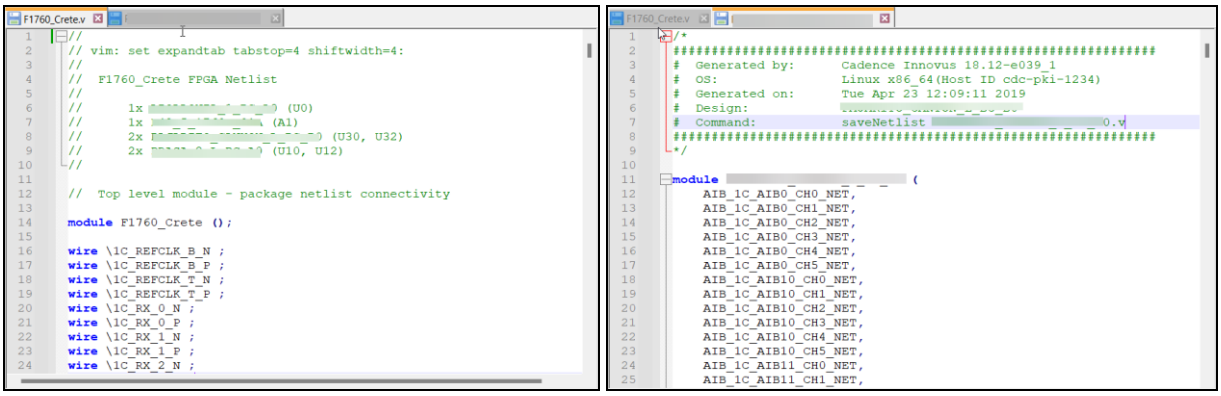


Figure 3. Example of Verilog source imported into XSI

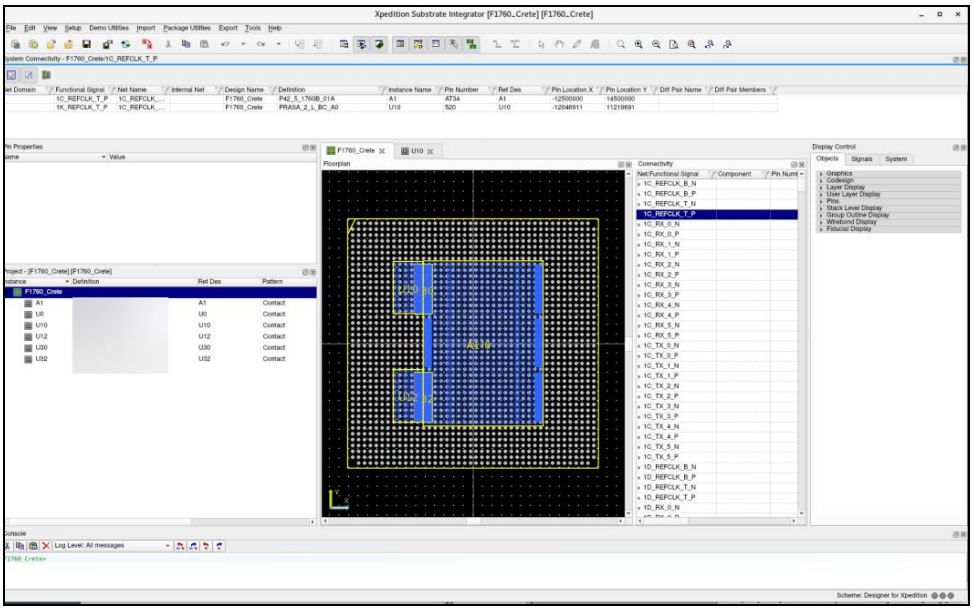


Figure 4. Example of aggregate netlist

Once the source design data has been aggregated, XSI can export connectivity in a number of formats for verification purposes. The aggregate netlist can then be validated against the reference netlist the design was assembled from.

III. Functional Verification

The idea of running functional verification of a complex substrate assembly is appealing as it would validate the system is connected correctly and functions as expected. From a distance, functional verification seems like a solvable problem. The core of the design is typically a chip being design in-house so one might assume a model and testbench exist and are readily available. However, functional verification of a substrate requires more than just the main silicon device. There is a tremendous amount of additional information to run a simulation, some examples are:

- Functional models for each die in the system
- Models or provisions for discrete components
- Testbench to exercise the verification and verify correctness

When a design employs chiplets (e.g., HBM), functional models may not be available as these components often come from third party suppliers. Often the chip RTL and verification environment aren't available to the substrate designers. Even if it were available, does the substrate design team have the expertise to adapt it to the full package or interposer? Do they have the expertise to run the verification tools? If everything were available along with the resources who can run such a verification, how long would a run take? Performing a functional verification on a modern package design is like the challenges faced by PCB designers where functional simulation has never been embraced mostly due to the lack of models.

Another functional verification approach is to write specialized models which exercise the die-to-die connectivity of the substrate. These sorts of models are often much simpler and don't need to account for complex clock schemes. They are written to push randomized data through the substrate in a representative fashion with automatic checking to ensure the data sent into the system matches the data which comes out of the system.

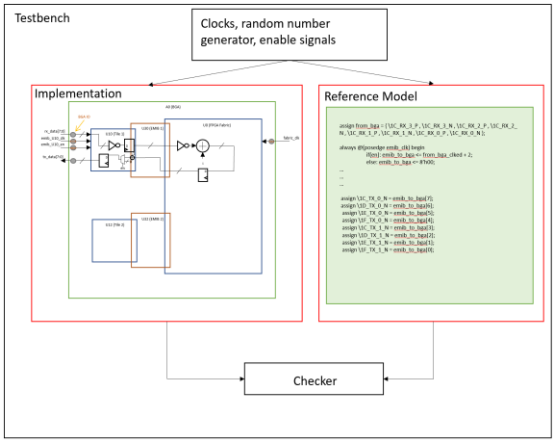


Figure 5. Example of functional verification

While this approach to functional verification is typically easier to implement and runs very quickly, it does require development of specialized Verilog models and testbenches specifically to verify substrate connectivity. The skillset required to do this development is not typically found in the domain of package and interposer designers so it needs to be staffed accordingly.

While it is technically possible to perform functional verification of a complex substate assembly, it is often impractical to do so for a number of reasons. Functional verification on a modern substrate design is similar to the challenges faced by PCB designers where functional simulation has never been embraced primarily due to the lack of models.

IV. LVS (Layout vs Schematic)

What about LVS? Can't LVS verify our system? Yes and no.

While LVS at the system assembly level is gaining popularity thanks to tools such as Calibre 3DSTACK, LVS can only tell us if the physical implementation of the design matches the source netlist. LVS can identify shorts and opens and other similar physical issues but it cannot tell us if the design is actually "correct". LVS does not address these questions: Does it work as designed? Does it perform the function as intended?

LVS can only tell us if the source netlist matches the layout netlist. But what if the source netlist is incomplete or has errors? If the netlist is wrong but the layout matches the incorrect netlist, LVS will pass giving the designer a false sense of security.

V. Automatic Formal-based Approach

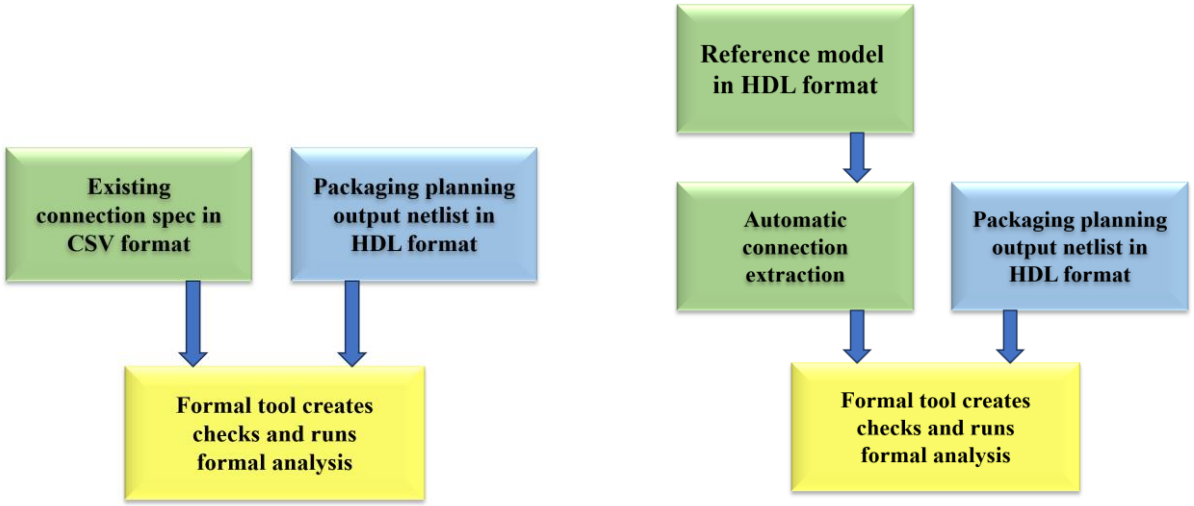
The formal verification tools specializing in connectivity check are good fit for verifying package connection early in the process. The formal connectivity solutions don't require package designers to write lengthy testbenches and assertions for possible millions of connections while otherwise required by using simulation. Formal connectivity solutions don't require package designer to understand the function of the design or know how to run simulation either. For verifying the connections between multiple dies, formal tools don't need the functional models of the dies, and only need the system top module that instantiates the multiple dies and has the connectivity information of the dies, and the module port definitions of the dies, normally the "black box" modules received by a package team.

Formal verification is powerful for verifying package connections in chiplets. Here's how formal methods can help in this context:

1. *Detecting connection errors*: Formal connectivity solution can detect errors related to package connections. It can mathematically analyze the package output module against the connectivity spec and identify inconsistencies or violations of intended connections that may lead to electrical faults. When it finds broken connections, it provides waveforms to show the issue, and has all other necessary features for debugging the issue.
2. *Ensuring Correctness of connections*: Since formal is doing exhaustive mathematically analysis considering all possible stimulus, it can verify that signals are routed as intended by proving connections adhere to the specified requirements. When it proves a connection, there is no scenario that can break the connection, i.e., no overlooked scenarios.

3. *Avoiding Short Circuits*: Formal verification can identify potential short circuits or other connectivity issues in the package by finding unintended connection paths.
4. *Complex Systems with millions of connections*: The formal flow generates golden connection spec automatically without manually creating csv files of connections and can verify large numbers of connections using parallel algorithms.
5. *Early Detection*: Formal verification can be applied right after package prototyping, before package physical implementation. Any mistakes in the planning and prototyping stage can be caught early, which can save lots of time and money.
6. *Safety and Reliability*: Only formal verification can do exhaustive analysis, it is crucial to use formal verification to verify the correctness of package connections of ICs for industries like aerospace, automotive, and medical devices.
7. *Compliance*: The formal tools we are using are ISO 26262 certified which can be essential for audits and quality control for industries with strict regulatory standards.

When using formal tools to verify the correctness of the package connections after package prototyping is done, we need the specification of the connections. Some companies manually create connection spec in CSV file and write script to generate the system top to instantiate the multiple dies following the connection spec. In this situation, the existing CSV file can be used as spec for verifying package output file. When there is no existing CSV spec of the connections, either the user manually creates the connection spec in a format accepted by formal tool or uses connectivity extraction tool to generate the specification if the design/verification team has already created system top module and has done functional verification. We can use the system top module and the black boxes of the dies as golden reference design and its connections as specification. When we have the spec of the connections and the output netlist file from the package tool, we can run formal connectivity tool to verify if the package output design satisfies the connection specifications. The two flows of using formal verification tool to verify package connectivity are shown below.



(a) Using existing connection spec in CSV (b) Using reference model to extract connection spec
Figure 6. Formal verification flows for verifying package connectivity.

In our testcases, we used the flow shown in Figure 6 (b) by using golden reference models[4] and Siemens EDA tool Connectivity Explorer to extract connections automatically from the reference model and export them into CSV file, using the package tool XSI to generate Verilog netlists, and using formal tool Check Connect to automatically generate checks for the connections in the CSV file and formally verify if the package output file satisfies the connections in the CSV spec file, i.e., if package process has broken any connections in the golden reference model. Here is the detail of the flow we are using.

[4] A "golden" reference model is typically developed by the design engineering team and is created over time through the design phase and extensive simulation. Once the design team is satisfied with the simulation results, the representative model is often referred to as the "golden model" which other simulations and analysis must correlate against. Development of the reference model is typically a combination of hand written and automatically generated RTL exercised and verified against hand written and automatically generated testbenches and stimulus.

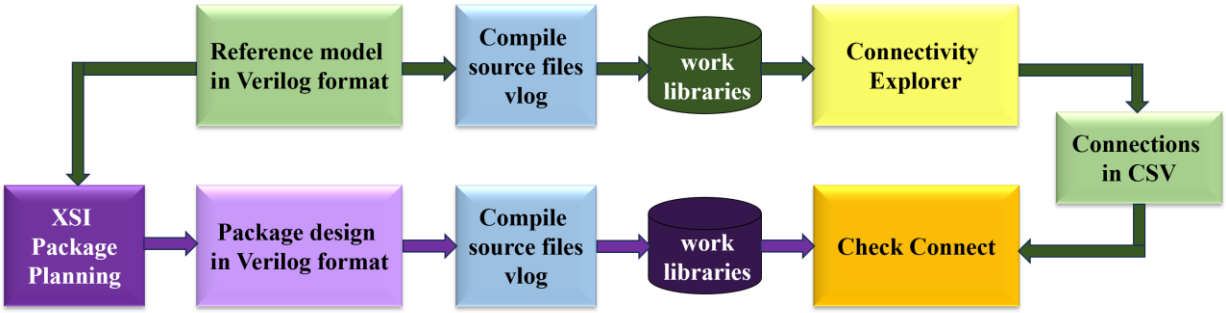


Figure 7. The detail of the flow using formal verification

When using formal connectivity solutions, the script is very simple and only need a few minutes to create. After the script is ready, the run is automatic. Here is an example of the Makefile to run all the steps. The config.txt has the simple definition "-inst *" that tells the Explorer to extract all connections for all instances.

```
#### Compile designs
  Compile_vl:
       vlog -sv -f flist_golden.txt -work lib_golden
       vlog -sv -f flist_package.txt -work lib_package
#### Generate Connectivity Spec
  Generate_conn_csv:
       qconnect_check -explore -od log_csv \
       -infile config.txt \
       -dut F1760_Crete -work ./lib_golden
#### Run Formal Analysis
  Check_connect:
       qverify -od log_cc -do "\
       connectcheck compile -d F1760_Crete -work lib_package;\
       connectcheck load csv log_csv/qconnect_explore_F1760_Crete.csv;\
       connectcheck verify "
```

When using the formal tool to verify package connections, we only care about the connections between the blocks, and don't need the function inside each block. The package tool can export modules without internal logic and only keep ports and connections between blocks. Package teams often only gets the system top module and "black box" of dies from system verification team. Even when package teams get the full functional modules from system verification team for the golden reference model, Connectivity Explorer can treat the die modules as black boxes and extract the connections between the dies. Due to "black box" modules of dies, the formal tool can run very fast with any size of the system. When verifying the package design against the connection spec CSV, Check Connect also runs fast and can handle big systems with many dies since there are not many sequential logics for analyzing.

When running Check Connect on the package design with the connection spec CSV, if the tool exhaustively proves all the connections, we are assured that the package design satisfies the connectivity spec; if the tool finds any violations, we know that the package design has broken some connections and routed signals in wrong paths. When the tool finds a violation, it provides a short counterexample in Wave tab to show the violation of the connection. With the waveforms and rich debug features such as source tracing and schematic view, provided by the formal tool, we can easily find the root cause of the issue.

Is there any possibility that the reference model or the connection specification miss some connections? It is possible since system functional verification may not be exhaustive, or package planning may be started before

system functional verification completes. Is there any possibility that the package design accidentally adds extra pins or connections? This situation is rare, but not entirely error prone, such as typos of pins that actually add new pins. One way to help on this issue is using a unique feature of Check Connect to generate a missing-port report that lists the ports of the top module and the dies not covered by any connections. If there is a port uncovered by the connection specification, first we will check if the port definition in the package design is correct, not a mistake. If the port definition is correct, the problem may lie either in the reference model from which the CSV file is extracted or in the originally manually created connection specification. For missing connection definitions, we can manually add them, and rerun Check Connect on the package design.

We have run two designs using the flow mentioned in Figure 7. The wall time including compile and extracting the connections of the reference model for the two test cases is 15 seconds and 35 seconds respectively. The wall time including compile and checking connections of the package design for the two test cases is 30 seconds and 56 seconds respectively. Here is the result table for the two testcases.

	Time for extracting connection spec	The total number of connections	Verification results of the package design	Time for verifying the package design
Design 1	15 seconds	21367	All proven	30 seconds
Design 2	35 seconds	43440	All proven	56 seconds

When using Check Connect to generate the missing-port report for Design 1, it identified that two ports "U12.DFX_THERMO0" and "U12.DFX_THERMO1" are not covered by the connection specification. Checking the source files of both the reference model and the package design, both don't have connections for these two ports shown in Figure 8. The root cause of missing connections for the two ports in the package design is caused by missing connections in the reference model. The tool can not only verify package designs, but also find possible missing connections in the reference models.

In the case identified below, the designer has purposely left two thermal sensor pins unconnected. Check Connect identified these two as suspicious and the designer would have to decide how to disposition them. It is not uncommon to leave unused pins floating. The design team later confirmed leaving these two pins floating was indeed intentional but was also impressed it was identified.

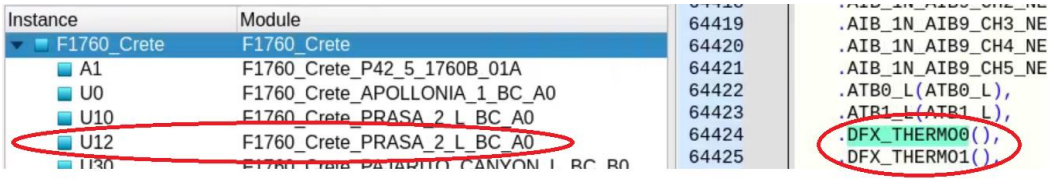


Figure 8. Missing two connections

VI. Conclusions

We have addressed the challenges of verifying package connectivity and illustrated how to use formal tools to verify connectivity for package designs. The setup for running the formal tools is simple, only needs a few minutes. The connection extraction and the package connection verification run fast and can work on big systems that have many dies. This method requires minimum manual work. Once the setup is done for one design, we can reuse the setup for another design with a little tweak such as changing the design name and source file list.

When starting the connectivity verification earlier right after package planning and prototyping using formal, the quality of the physical implementation can be improved dramatically and the time to market can be shorten significantly. The ease of use of the method makes its adoption a no-brainer decision.

References

[1] Daniel Han, Walter Sze, Benjamin Ting, and Darrow Chu, "A Reusable, Scalable Formal App for Verifying Any Configuration of 3D IC Connectivity," DVCon US 2013

[2] Murugesh Palaniswamy, et.al., "SoC Verification of Analog IP Integration through Automated, Formal-based, Rule-driven Spec Generation," DVCon US 2018

[3] Cone, Chris and Ali, Zain, "The smart path to chiplets using hierarchical device planning and pin regions", GoMACTech23

[4] John Ferguson, "Successful 3DIC design, verification, and analysis requires an integrated approach", Chiplet Summit 2023