Automatic Netlist Modifications required by Functional Safety

Harald Lüpken, Dirk Hönicke, Michael Rohleder
AMCU/New Product Development Center Munich
Freescale Semiconductor
Munich, Germany
harald.luepken@freescale.com, dirk.hoenicke@freescale.com, michael.rohleder@freescale.com

***Abstract*—**Developing semiconductor products targeted towards a functional safety application impose an additional set of challenges for the involved development teams. Requirements like "freedom of interference", redundant execution and cross-checking of functionality, as well as countermeasures for possible common cause failures need to be taken into account for every step of the design cycle. The implementation of corresponding features and the reuse of IP and subsystems/platforms involves often netlist modifications of non-trivial nature; i.e. new or modified hierarchy levels, additional device modes, and the insertion of additional logic or logic blocks.

Functional safety standards, like the ISO26262 for the Automotive Industry, require a high degree of repeatability of the corresponding work. Here, the early adoption of the IP-XACT standard for netlist assembly allowed us to exploit related benefits for several earlier products. The most recent family of safety devices developed by Freescale™ for one automotive customer is extending on these capabilities. Automation of further netlist modifications is employed to implement many safety relevant features, like logic built-in-self-test (LBIST), online memory built-in-self-test (MBIST), independent physical hierarchies and their separation by X-bounding, as well as the required bypass and comparison logic.

Vendor independence is another benefit of using an industry standard like IP-XACT. It enables easier extension of the required capabilities but also enabled a faster switching between tool vendors.

Keywords—ISO26262; IP-XACT

I. Introduction

The IP-XACT standard, originally developed by The SPIRIT Consortium, Inc. (now merged into the Accellera Systems Initiative [1]), has meanwhile matured into an IEEE standard (IEEE Std 1685™-2009 [2]). The intention of this standard is to provide a well-defined and unified specification for the meta-data representing the components and designs within an electronic system.

The flexibility of this standard makes it suitable for many areas of design work; the most common ones is by tools for netlist generation and for processing and maintaining register descriptions, e.g. for header file generation and memory map definition. With the ever-increasing complexity of state-of-the-art systems and semiconductor devices, this standard enables to automate tasks that had to be done manually in earlier days. It further permits to reuse information that is common for multiple devices in various aspects, which enables the reuse of the related data, even across product families [3].

Systems that are targeted towards applications in the field of functional safety have always required a rigid, well organized and structured design style to ensure a repeatable and well documented development effort. Automating many steps of the related effort is a welcome way of doing things less manually which usually translates into less error prone. This, and the increased reuse capabilities that could be achieved when performing the corresponding activities in a tool supported environment, have led to an early adoption of the IP-XACT standard for several aspects of the development work within our organization.

This paper describes the experiences when further extending an already automated development step, the netlist assembly, with additional capabilities that are required due the system targeting an application in the field of functional safety.

II. Functional Safety

Electrical and/or electronic (E/E) elements have been used for many years to perform safety functions. Computer-based systems – also referred to as Electrical, Electronic and Programmable Systems (E/E/PS) – are increasingly being used to perform safety functions.

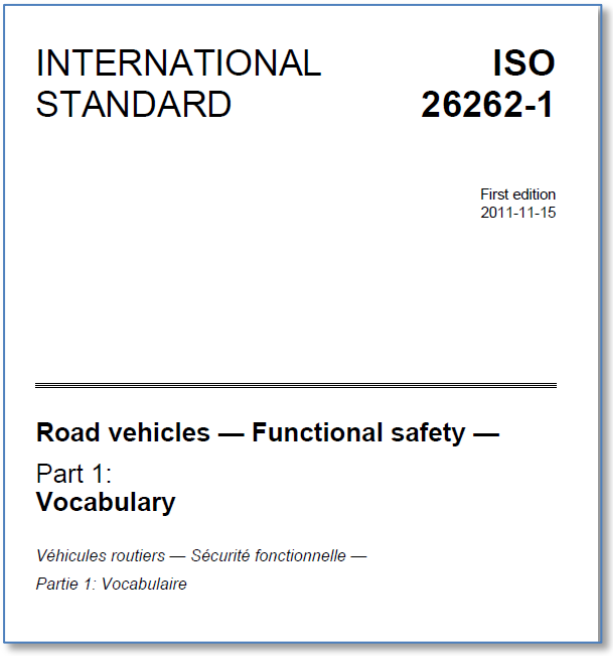
Functional safety is a concept applicable across all industry sectors that is fundamental to the enabling of complex technology used for safety-related systems [4][5]. It is the part of the overall safety of a system or piece of equipment that depends on the system or equipment operating correctly in response to its inputs, including the safe management of likely operator errors, hardware failures and environmental changes. Its objective is the freedom from unacceptable risk of physical injury or of damage to the health of people either directly or indirectly (through damage to property or to the environment). As such it is fundamental to the enabling of complex technology used for safety-related systems by providing the assurance that these systems will offer the necessary risk reduction required to achieve safety for the equipment.

Functional safety is intrinsically end-to-end in scope in that it has to treat the function of a component or subsystem as part of the function of the whole system. This means that whilst Functional Safety standards focus on the E/E/PS, the end-to-end scope means that in practice functional safety methods have to extend to the non-E/E/PS parts that those systems actuates, controls or monitors. Furthermore, safety standards are not only concerned with the parts of a system; they are also covering many aspects of the development of those parts. As such, functional safety features form an integral part of each development phase of a safety product, ranging from the specification, to design, implementation, integration, verification, validation, and production release.

III. The ISO26262 Standard

The standard ISO 26262 (official title "Road Vehicles – Functional Safety") [4] is an adaptation of the Functional Safety standard IEC 61508 [5] for Automotive Electric/Electronic Systems. It defines functional safety for automotive equipment applicable throughout the lifecycle of all automotive electronic and electrical safety-related systems.

ISO 26262 is intended to be applied to safety-related systems that include one or more electrical and/or electronic systems and that are installed in "series production passenger cars with a maximum gross vehicle mass up to 3500 kg". It addresses possible hazards caused by malfunctioning behavior of E/E safety-related systems, including interaction of these systems. It does not address the nominal performance of E/E systems.


INTERNATIONAL STANDARD

ISO 26262-1

First edition
2011-11-15

Road vehicles — Functional safety —

Part 1:
Vocabulary

Véhicules routiers — Sécurité fonctionnelle —
Partie 1: Vocabulaire

Figure 1: from the Title page of ISO26262 - Part 1

The standard consists of 9 normative parts (first edition published on November 15th, 2011) and a guideline for the ISO 26262 as the 10th part (published July 25th, 2012) [6].

The following table provides an overview of those parts:

Part 1: Vocabulary

Part 2: Management of functional safety

Part 3: Concept phase

Part 4: Product development at the system level

Part 5: Product development at the hardware level

Part 6: Product development at the software level

Part 7: Production and operation

Part 8: Supporting processes

Part 9: Automotive Safety Integrity Level(ASIL)-oriented and safety-oriented analyses

Part 10: Guideline on ISO 26262

Like its parent standard (IEC 61508) [7], the ISO 26262 is a risk-based safety standard, where the risk of hazardous operational situations is qualitatively assessed and safety measures are defined to avoid or control systematic failures and to detect or control random hardware failures, or mitigate their effects. In particular, the ISO26262:

- provides an automotive safety lifecycle and supports tailoring the necessary activities during its phases.
- covers functional safety aspects of the entire development process.
- provides an automotive-specific risk-based approach for determining risk classes (Automotive Safety Integrity Levels, ASILs).
- uses ASILs for specifying the item's necessary safety requirements for achieving an acceptable residual risk.
- provides requirements for validation and confirmation measures to ensure a sufficient and acceptable level of safety is being achieved.

Since functional safety is an essential element for many systems targeting an application in the automotive world, the requirements defined by ISO26262 are becoming an essential need for many electrical and electronic elements being developed for this market segment.

IV. Safety Functionality and Features

Semiconductor devices that are targeting an application in the area of functional safety often provide or implement some specific features or functionality. The intention of those features is in many cases a better or easier detection of faults that might lead to a malfunction of the device. Other features might support fault avoidance or be beneficial to prevent failures that could not be detected or avoided otherwise.

One common method in this area is *redundancy*; with its many nuances of implementation. Redundancy can have many forms; e.g. two identical parts or subsystems that are processing their inputs concurrently and check each other in every clock cycle (so named "lock-step architectures"), or one subsystem that is continuously checked by another subsystem having only reduced functionality. The following figure depicts the redundant portion of the block diagram of a lock-

step architecture, the SPC56EL60xx [8]. The majority of blocks colored in blue within this picture is implemented redundantly, while the yellow blocks (FlexRay, Flash memory, and SRAM) are only available once. All redundancy checker elements (RC) shown in red are itself replicated.

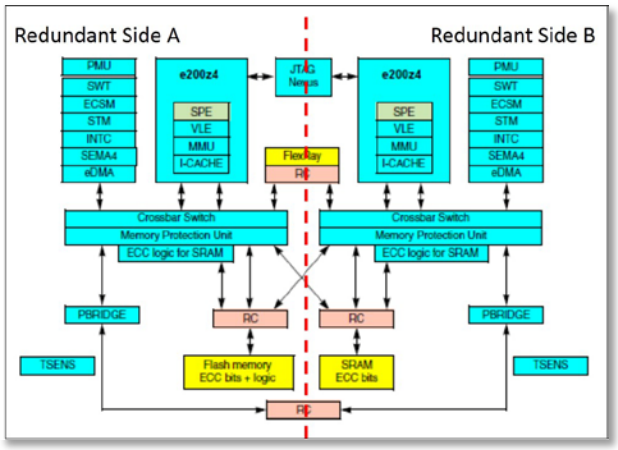


Figure 2: Redundant portion of a lock-step architecture

Most common implementations of such architectures are replicating one or more processor cores including the corresponding coprocessor elements, related caches, and associated control IP, like memory controllers, cross-bar switches, interrupt controllers and memory management and protection units. The redundant implementation of the most important processing elements permit a symmetric redundant processing as shown in Figure 3 [9]. In some cases, the sensors and actuators may be replicated as well, to provide a maximum of redundancy for the safety application.

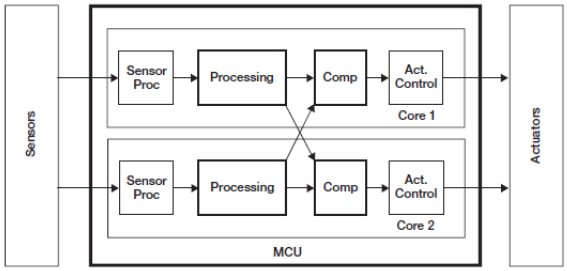


Figure 3: Symmetric redundant processes implemented by a lock-step architecture

The benefit of these lock-step architectures is their fast reaction on potential failures (often dependent on the amount and type of signals being compared) and the relatively high capability to detect and identify failures that arise during the processing.

Redundancy is a very important, but not the only architectural feature implemented in semiconductor devices targeting a functional safety application. When an architecture relies on redundant elements for detecting a certain class of failures, then there is always a need to detect, avoid or at least reduce the possibility of so named “common cause” faults. Functional safety refers to such faults in the context of redundancy, when it is possible that a single fault has an equivalent or not distinguishable impact on both redundant elements – and is as such preventing any chance to detect such incidents with only the redundant implementation of a part of the system.

Another important topic for functional safety devices is the “freedom of interference” between parts, which is important to be maximized as much as possible. With this term functional safety experts refer to features, which ensure that an unintended behavior in one part of the system does not result in impacting another part of the system. In some cases features that avoid common cause faults can also be used for providing an increased freedom of interference.

The rest of this section discusses three functional safety specific features that are central for the reported work.

A. Lakes

Lakes are design/physical hierarchies within a semiconductor device which contain specific sub blocks, as shown in Figure 4. The composition of a lake depends on the safety concept. Lakes can be used to separate safety-relevant blocks from other blocks that might itself be safety-relevant or non-safety-relevant; thus providing a certain amount of independence between those blocks. The usage of lakes enables the physical implementation to create separate placement areas, decouple common aspects like power supply, clock or reset control, and to implement routing rules intended to ensure the compliance with specific safety requirements. Common requirements imposed by safety features are the avoidance of signal crossings, a separate power supply, independent clock and reset control or other features with the intention to ensure the freedom of interference between logic that is separated by the lakes. It might also be used to control and meet diversity requirements for replicated blocks; e.g. when it is required to use different cell sets for replicated blocks.

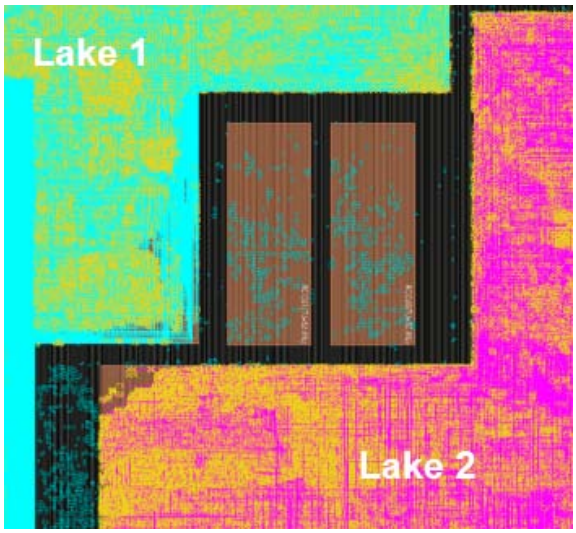


Figure 4: Lakes in a semiconductor device

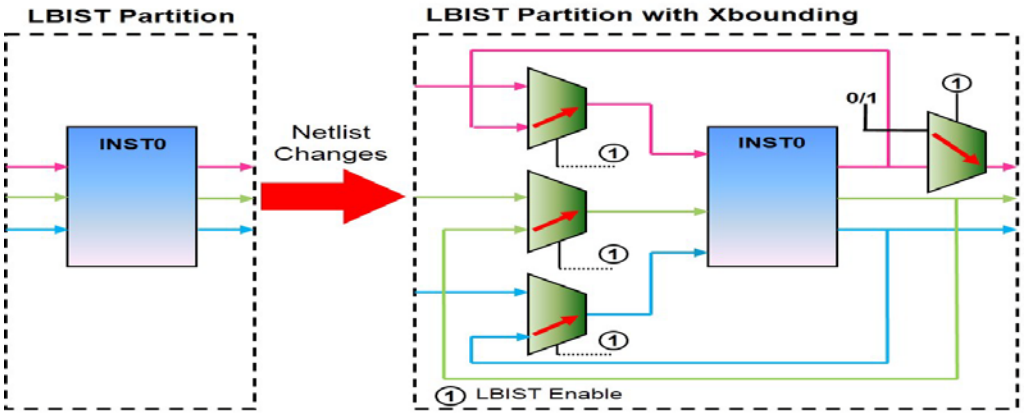


Figure 5: Logic BIST (LBIST) partitions and x-bounding

B. *LBIST and x-bounding*

Logic Built-in Self-Test (Logic BIST or LBIST) is a technique based on pseudo random scan patterns generated at runtime by a LBIST controller; with the intention to identify incorrect behavior of the corresponding logic including combinatorial and sequential elements and thus providing a capability to find non-transient errors within this logic. For this purpose dedicated IP blocks are assigned to a LBIST partition, which might coincide with or being part of a lake. Such a partition can be a substructure of a lake, but never span multiple lakes. In principle, the set of LBIST partitions create a new hierarchy layer within a semiconductor device that can only be used in a specific "Logic BIST" mode.

To ensure repeatability of the LBIST operation ("LBIST sequence"), the inputs to a LBIST partition have to be isolated to ensure that the LBIST functionality has full control over the logic to be tested. Furthermore, partition outputs shall not influence sensitive parts of the designs, to inhibit any negative impact of the operations being performed. The isolation of the inputs and outputs of a partition is managed by an approach called "x-bounding". For this purpose, multiplexers with predefined or rule based feedback are assigned to the inputs and the outputs of a partition (refer also to Figure 5). The controlling rules can be based on structural information, clock domain specifics or other data of interest. These multiplexers are then used to ensure that the LBIST controller has full control over the logic within a partition, and that there is no impact on other parts of a system when a partition undergoes a LBIST sequence.

C. *Bypass logic*

The implementation of lakes requires further that some of the corresponding connectivity is receiving special treatment, in particular:

- there are minimum distance rules, which are often related to the root cause of a potential incident (e.g. alpha particle).
- inter-lake connections must be buffered to ensure that a failure condition in one lake cannot propagate backwards across lake boundaries.
- crossing of inter-lake and potential common-cause signals must be prevented or avoided to ensure that a short or electrical overstress condition cannot result in a common cause failure.

A special topic of interest is the connectivity required for providing data from singular elements that are eventually implemented in one of those lakes to the other redundant processing elements within the other lake. The so named "bypass logic" must implement the connection of the corresponding signals in a manner that minimizes the potential for common cause failures and also maximizes the freedom of interference. This can be best illustrated by a related safety requirement for this logic:

Requirement <reqID>: *The ports of all modules that accomplish the coupling of both lakes should be placed in that manner that the A-side signals and the B-side signals are not mixed and crossed.*

For this purpose, the bypass logic blocks are implemented providing a predefined number (e.g. 128) of bypass signals. These blocks have been hardened to support a backend flow that addresses the above (and some more) safety requirements.

V. Automated Netlist Assembly

The netlist assembly flow being central for the reported work has been developed several years ago by a joint effort between Freescale™ and ST Microelectronics™. It is based on the IP-XACT standard to ensure the reusability of various design items and provide means to import information from other sources or export data for subsequent usage. Another intent for the development of this flow was the migration towards third party tools and industry standards, to eliminate existing dependencies on company proprietary tools and databases.

The following Figure 6 "IP-XACT based Netlist Assembly Flow" provides an overview of this flow, including its most important data and processing steps. All IP blocks are "packaged" (*Packaged IP*); where "packaging" refers to adding/extracting IP-XACT information to the original IP database. This packaging step is a sub-flow, and is also using a common set of *Protocol Definitions* that describe common and reusable interfaces being used by the IP blocks.

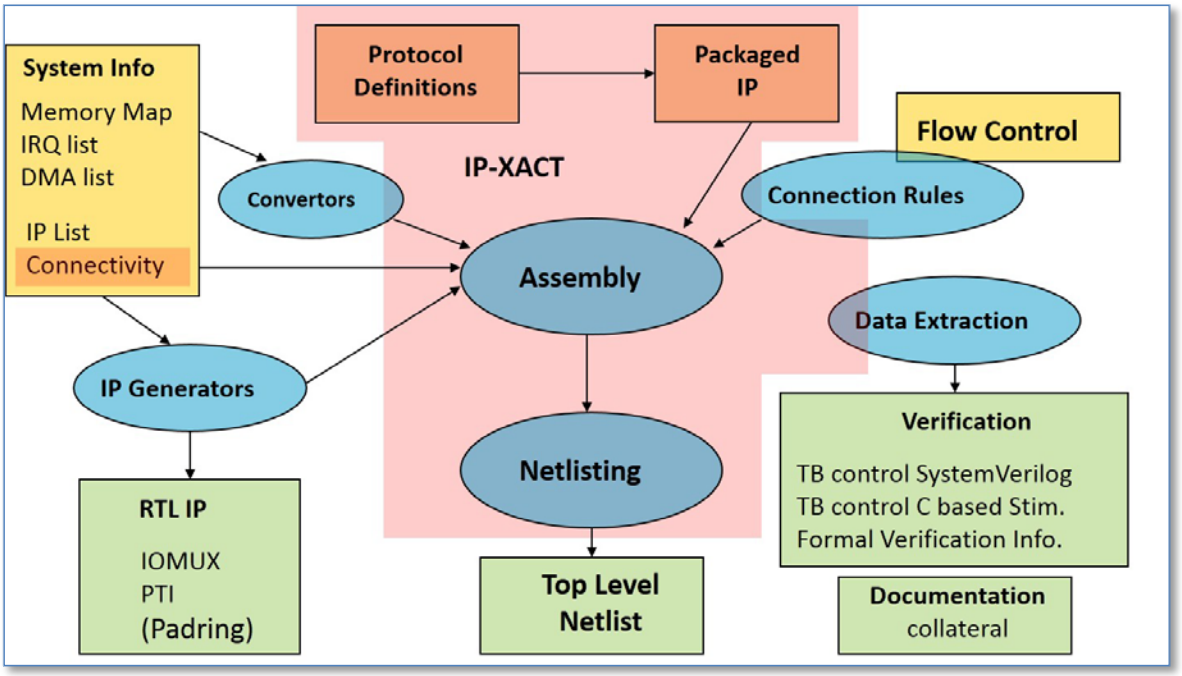


Figure 6: IP-XACT based Netlist Assembly Flow

The core of the assembly flow is the "*Assembly*" step that combines the packaged IP with connectivity and other parametric information according to provided connection rules. In addition to creating connections between design objects, further automation steps are defined. These include the generation of device specific system information like memory maps, interrupt assignments and DMA connectivity. Furthermore, *IP Generators* are capable to create specific IP blocks and generation rules provide iterators/macros to avoid repetitive specifications. After assembling the design and creating the desired hierarchies, the "*Netlisting*" step converts the created design into either HDL output and/or IP-XACT designs. An additional data extraction step permits to extract information that is intended to support the verification effort and improves the documentation.

The described setup enables the usage of generic databases holding connectivity and parametric information instead of manually defining the design connectivity and its hierarchies using a specific hardware description language (HDL). Portions of a particular SoC are often of very generic and common nature, but many of their details are then specific for a SoC. This is in particular the case when the corresponding functionality is required but the implementation itself is dependent on topology, feature mix, interfaces, etc. Examples for the corresponding, IP is the processor platform, the clock/reset logic, the logic for configuring and multiplexing I/O signals, or common blocks implementing ATPG features.

The resulting connectivity databases are highly configurable and easy to maintain due to their simple format (EXCEL based comma separated value [.csv] files) to permit their reuse within multiple members of a product family. A major portion of all required parameters and connections of a semiconductor device can be provided as part of such a generic database. Using IP-XACT as backbone for the some of the associated development steps enabled the usage of existing 3rd party tools and standard definitions, instead of having to develop and design a proprietary description format for this purpose.

The intention of this approach is to facilitate the (re-)use of IP blocks together with the corresponding connectivity, which is especially beneficial when these devices are part of a device family or are covering a similar application space. The capability to reuse not only the IP blocks, but also related connectivity data, permits to predefine or generate a significant amount of the corresponding connections. This concept enables product teams to concentrate their efforts on adding differentiating IP and generating the related connectivity. Reentering or modifying similar portions of a design by hand can be avoided. The described methodology not only supports the creation of complex systems, but also ensures a high quality by eliminating manual design effort and by re-using previously verified portions of a SoC.

VI. Automated Netlist Modifications

Automation of netlist generation and associated modifications was a necessary prerequisite to achieve high design efficiency, ultra-short response on change requests, as well as enhanced controllability, repeatability and reuse of design modifications and design data. Additionally, it was an enabler for further netlist changes that have been imposed by functional safety requirements; as these have been already described earlier.

Having many aspects of the semiconductor device specified with abstractions, like the definition of memory map aspects, interrupt tables and DMA connectivity turned out to be very beneficial. Related design data is often already made available, a common method are tables that can be found in

specifications or are intended to end up in user documentation. Extracting this information, and using generators to create the corresponding logic and connections enabled further automation and reduces the need for error prone manual edits. The resulting peripheral bridges, read/write multiplexers, protection wrappers, memory block interface logic, interrupt request lines, DMA request/acknowledge signals are generated by a repeatable process, that can be easily adjusted or extended to cover additional requirements.

One first application is the creation or removal of hierarchy levels; e.g. for the definition of power of high-speed clock domains. Such hierarchies are also required for a hierarchical backend flow, which intends to reduce tool runtime by a separate physical implementation of certain portions of a semiconductor device. Similar hierarchies may also be generated in 3rd party subsystems, which are delivered in standardized formats, like structural RTL or as IP-XACT design.

A. *Safety lakes*

Safety lakes define boundaries for many aspects to be taken into account by tools used for the physical implementation work; e.g. for power domains, clock distribution, LBIST partitions, but also for placement and routing restrictions that have to be applied to a certain subset of a semiconductor device. They are essential elements to implement a lock-step architecture, ensure diversity or the freedom of interference of replicated blocks. Often, a safety lake is represented by an additional hierarchy that is specially treated by the physical implementation. However, several layers of hierarchy levels may be implemented; e.g. one or multiple LBIST partitions within a safety lake.

Using the capabilities of an automated netlist assembly flow, creation of a safety lake became a significantly simpler and less tedious task. Modifications that are heavily impacting the interface and/or content of a hierarchy can be managed by changing a few lines in the input data or associated scripts. Creation of a correct netlist becomes a quick and repeatable step within the design flow. In comparison, the corresponding manual changes would be more error prone and take much longer (e.g. adding/removing a sub-block). Incremental changes of an interface could be addressed by simply re-running the flow. As such the automated netlist assembly flow using IP-XACT for components and the SoC netlist enabled to quickly apply complex hierarchy manipulations involving many substantial changes; e.g. in the number of instances, the interface size or modifications of the interface itself.

B. *LBIST partitions and x-bounding*

LBIST partitions and x-bounding are both necessary elements to implement Logic BIST on the RTL level. A single SoC may have several LBIST partitions within different hierarchy levels. In many cases, the set of logic assigned to a LBIST partition may relate to different hierarchies or hierarchy levels; involving similar changes than the creation of safety lakes.

Further requirements resulted from the need to apply x-bounding to the boundaries of a LBIST partition. For this purpose a flow has been developed that is using structural information and clock domain crossing data derived from the SoC RTL code using standard static verification tools (e.g. LEC or Spyglass®) for evaluating related aspects of the netlist. The extracted data is maintained incrementally and used to drive the x-bounding modifications. Also in this case, the automated netlist creation process became an enabler for the implementation of the x-bounding step within the SoC assembly flow. Applying those modifications on RTL level using IP-XACT did avoid time consuming loops via the backend implementation flow and facilitated quick iterations.

Basically, the x-bounding flow classifies the input and output paths of a LBIST partition according to the following attributes (as is shown in Figure 7):

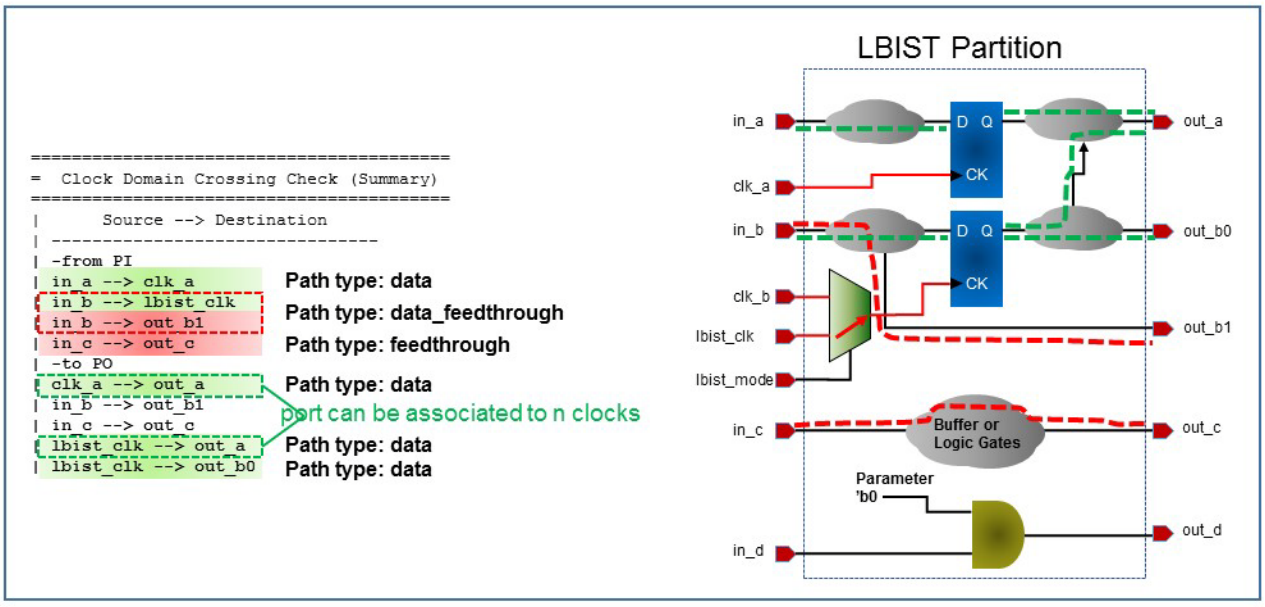


Figure 7: Possible path types and their association with an LBIST partition

- Clock association (input-to-capture clock, launch clock-to-output)
- Registered input/output path (direct path from input to FF, direct path from FF to output)
- Combinational feed-through path
- Mixture of registered input/output and combinational feed-through path.
- Constant input (input always constant)
- Constant output (output always constant)
- Clock input (clock during LBIST)
- Reset input
- Mode inputs (constraint for LBIST modes)
- No clock association

Using the above classification, some rules can be defined for the implementation of the x-bounding feedback:

Rule	
#1	Outputs are used as feedback source if the clock association is the same as for the input to be x-bounded.
	Purpose: avoid critical timing paths or clock domain crossing problems.
#2	Constant outputs are never used as feedback source.
	Purpose: avoid coverage issues as constant outputs cannot be controlled.
#3	Rule #1 is applied to select a feedback source for constant inputs.
	Purpose: allow control of the related logic.
#4	Outputs which are the end point of a combinatorial feedback path are never used as feedback source.
	Purpose: avoid combinational loops and critical timing paths.

Applying these rules on tables that are defined and maintained by the automated netlist assembly flow was a simple extension of this flow that has been proven to be very beneficial for the implementation of the x-bounding step.

C. *Bypass flow*

A similar approach has been used for implementing the inter-lake connections; in particular the bypass logic described earlier. Those inter-lake interfaces can comprise hundreds of connections having different attributes (e.g. standard signal, clock, reset, special net, safety relevant, debug) which have to be considered. Also here, static verification tools (e.g. LEC or Spyglass®) have been used to extract all interconnections, the resulting connection data and rules are stored in tables that are incrementally maintained. The resulting connection database is then driving the automated implementation of the required bypass connections, thus providing a robust flow. The Figure 8 depicts the structural and implementation of the bypass flow.

VII. Vendor Independence

There is a wide variety of tools available which provide the functionality required to perform the steps of an automated netlist assembly flow; such as netlist modifications, inserting additional objects as well as hierarchical transformations like grouping and ungrouping of levels of instances. These tools often define very different tasks which are – even despite their common usage of IP-XACT – not always compatible with each other. Changing a tool or tool set for a subsequent project can then quickly become a problem; since this might involve a new set of scripts and supporting assembly inputs. Re-use of assembly data would then become very difficult to achieve, if possible at all.

To secure the investment in building-up a machine-readable connectivity database, and to provide some flexibility in choosing the corresponding tool set an abstraction layer has been defined for all essential and required netlist modification functions. This layer consists mostly of a set of structured and well-defined comma-separated-value (CSV) files that usually define tables. There are three classes of tables defined:

1. object definition tables,
2. parameter tables, and
3. connectivity definition tables.

During assembly, the objects referenced by the object definition tables (instance and hierarchy table) are combined with information from the associated parameter tables. These tables specify the version of a module to be integrated, the parameters for an instance and the hierarchy level(s) of every instance. Connectivity definition tables contain mostly the source and target of a particular connection. Connections can be made from pins or interface ports of real as well as virtual instances assuming a flat design without any hierarchies. Separating connectivity from hierarchy information provides many benefits, when it is required to quickly create or remove hierarchies.

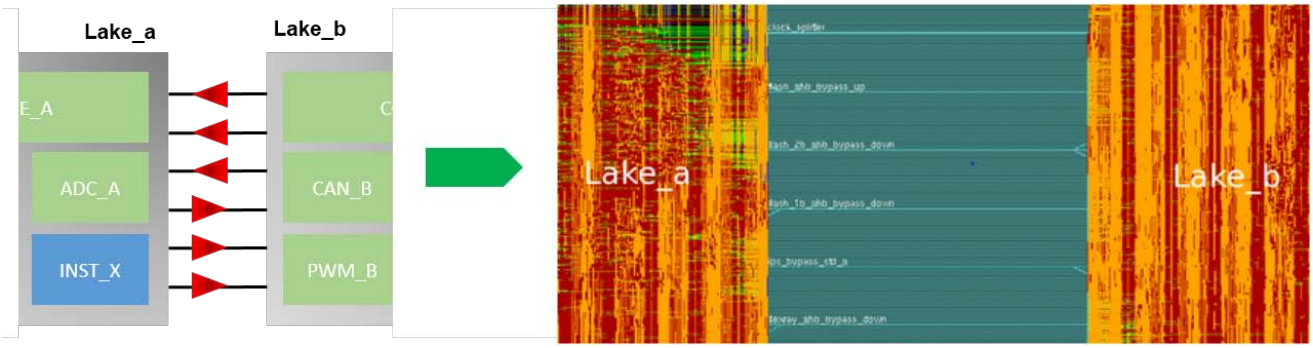


Figure 8: Bypass flow; structural and implementation view

Before the assembly step, these abstract design definition tables are being processed and mapped to the command set of the assembly tool of choice. This pre-processing step requires little time but ensures some flexibility when it is desired to switch the tool set to be used. Furthermore, this method enables the usage of different assembly tools concurrently, thus providing the opportunity to cross-check the resulting design assemblies.

The usage of simple CSV files and IP-XACT as interchange format in combination with this abstraction layer has been proven as key to stay vendor independent. This became very valuable when the vendor of the initially selected assembly tool A decided to discontinue its support. Replacing this tool with another tool set was straight forward and could be completed within a few days. It was only required to re-define the translation of the abstract definition tables to the assembly command set of the alternatively selected assembly tool B. Another benefit was the reduced training effort, since designers could continue without learning new formats and commands required/provided by the new assembly tool B.

VIII. Results

The usage of the IP-XACT standard for the definition of IP views and interfaces helped to successfully establish a joint development flow for complex SoCs. IP-XACT interfaces allow a significant reduction of connectivity data by bundling several signals to a single interface connection. Connection rules included in interfaces can be checked automatically. Incorrect connections can be reported at implementation time which reduces the verification effort.

In general the usage of IP-XACT for the netlist assembly automation helps to increase design efficiency; enables repeatable implementation and further automation and reduces the possibility of manual errors:

- Complex netlist operations (e.g. hierarchy creation) can be performed quickly, repeatable and with high quality results.
- Appropriate partitioning of the design data (e.g. IP integration, clock/reset, analog, DFT, safety, debug) enables a parallel development approach. Concurrent development is supported and allows quick progress in several areas (e.g. concurrent implementation of clock/reset logic, the DFT sub-system, and IP integration), resulting in reduced turnaround times.
- Reuse of connectivity data and quick adaption of connection tables allows continuous improvement and fast generation of prototypes.

Usage of IP-XACT for IP views and protocol definitions has been proven to be a big benefit to ensure independence from EDA tools and vendors. Most 3rd Party IP providers provide IP-XACT views as part of the delivery package. This enables quick and seamless integration of IP.

Data extracted from other sources or by other tools and incrementally maintained in machine readable format (e.g. EXCEL tables) can be used to drive complex netlist manipulations.

IX. Summary and Outlook

The implementation of safety requirements within a semiconductor device requires complex and massive manipulations of the RTL database. The expectation of functional safety standards on the development process is a repeatable and state-of-the-art implementation with high quality results.

An automated netlist assembly flow based on IP-XACT has been proven to become an enable to perform many required processing steps in short time and good quality, by providing appropriate means for automation, configuration and reuse of the involved information.

Based on these results, the usage of IP-XACT for netlist assembly is a success story, that will be continued and further extended to also support the reuse of verification and backend properties.

References

[1] www.accellera.org

[2] IEEE Std 1685™-2009, "IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows", Institute of Electrical and Electronics Engineers, Inc. Published 18 February 2010. ISBN 978-0-7381-6160-0

[3] Stefan Doll et al., "Skeleton, an approach to maximize reuse across multiple product families", DesignCon 2009

[4] http://www.iec.ch/functionalsafety

[5] Portions of the following text are citing statements taken from http://en.wikipedia.org/wiki about: Functional_Safety, IEC_61508, ISO_26262

[6] www.iso.org

[7] http://www.iec.ch/functionalsafety/standards

[8] SPC56EL60L3/L5, "32-bit Power™ Architecture microcontroller for automotive SIL3/ASILD chassis and safety applications". Data Brief. Doc ID 15461 Rev 3, Feb. 2010, Freescale Semiconductor, Inc.

[9] Markus Baumeister, "Using Decoupled Parallel Mode for Safety Applications". White Paper 2009. MPC564XLWP, Freescale Semiconductor, Inc.