DEVELOPMENT OF PROGRAMMABLE LOGIC IN A RAPID, AUTOMATED, AND REPEATABLE FASHION FOR SAFETY-RELATED APPLICATIONS

Keith A. Harvey, Dan Ho, Phil Allen, Andrew Mortellaro, Louis Yu
Lockheed Martin Corporation
1701 W. Marshall Drive
Grand Prairie, Texas, USA 75051
keith.a.harvey@lmco.com

ABSTRACT

Digital safety platform considerations for power plant modernization or new build efforts involve a burdensome task of implementing a large number of software-based functions onto a designated platform to match the plant's architecture. This problem is not unique to any particular safety-related platform solution, but is prevalent across the landscape of current Qualified and Licensed Safety-related Instrumentation and Control (I&C) Solutions. As an example when implementing a RG 1.97 compliant Post Accident Monitoring System (PAMS) solution with the Lockheed Martin Nuclear Protection and Control (NuPAC) safety platform, 72 unique software loads are needed to configure the Qualified Display System (QDS).

Large scale development and customization efforts to upgrade existing plant functions or systems require the licensee to assume the cost and technical risk necessary to undergo a digital based upgrade. Often the cost, schedule, and technical risk outweighed the benefit of modernization efforts due to the unknown hurdles and past projects that overran cost and schedule projections.

Lockheed Martin's Engineering Production Suite (EPS) minimizes the licensee risk and expense associated with modernization efforts through a developer's toolkit for rapid, automated, and repeatable development of Programmable Logic used to implement plant functions and architectures for use on the Lockheed Martin NuPAC platform. This paper describes a use case where by EPS can create system level representations, simulate system functions, partition logic designs to accommodate hardware constraints through logic partitioning, automate the assembly of Application Specific Programmable Logic (ASPL), generate system test cases, program the NuPAC hardware, and perform testing throughout the design workflow.

1 Introduction

The EPS is a development environment used to develop system level functional representations, simulate system functions, produce programmable logic, generate test cases, test functions and HW, configuration control developed artifacts and ultimately program Field Programmable Gate Arrays (FPGA) for use on the NuPAC platform.

The current configuration of EPS is driven by the need for a developer to have full access to the system functions and logic needed to implement a system based on plant specified requirements, balanced with desire for Application Specific Programmable Logic (ASPL) to be developed in a quicker, more repeatable, and efficient manner than the current manual method of PL development. Not only does EPS have use cases that apply to a developer, but there is also a desire to allow customer access based upon a utility customer's need to have a development environment that offers them insight into how LM's NuPAC platform could be used within their plant's architectures. The EPS provides the flexibility for the customer to perform architecture trades and feasibility studies for the NuPAC platform in their plants; up to something more complex (if utility need drives it) like developing new PL specific to their applications with the EPS

and deploy it to NuPAC for use in their plant. The EPS is also used for programming replacement GLMs (Generic Logic Module) and aid in calibration of GLMs when plant maintenance is performed.

As an overview, EPS is a development environment used to

- Create System level functional representations,
- Simulate system functions,
- Partition logic design for HW and I/O sizing,
- Produce Application Specific Programmable Logic,
- Generate test cases and test system functions,
- Program NuPAC Field Programmable Gate Arrays (FPGA),
- Perform testing throughout the workflow with equivalent test cases.

1.1 Partitioning Feasibility

One of the major hurdles and potential discriminators for EPS is the ability to quickly, and repeatedly partition logic that is too large to fit on a single FPGA device. Trade studies covering the feasibility analysis showed that for the majority of the cases evaluated, I/O was as big of a driver as logic size. EPS was conceived through a desire to optimize size and communication cuts while allowing user feedback to provide an optimal solution package. Algorithms developed for EPS provide the ability to partition a circuit schematic onto two or more FPGAs while minimizing communication hops between logic cards within a NuPAC system. The goal of this algorithm is to provide a system architect/FPGA designer the freedom of building a system-level design without the added complexity of manually mapping interconnections. The algorithm set forth is an iterative approach in solving for the NP-Hard (Non-polynomial time) problem of separating logic across a boundary that separates two to 18 GLMs. EPS algorithms can prescribe layout plans for partitioned logic onto separate GLMs while minimizing the number of cuts for signals across partitions; cuts which represent signals going onto and off of a GLM. Figure 1-1 below demonstrates the initial state and final state for one circuit schematic as it is divided into two partitions. The solution shown on the right provides only 2 signals being cut across the entire schematic.

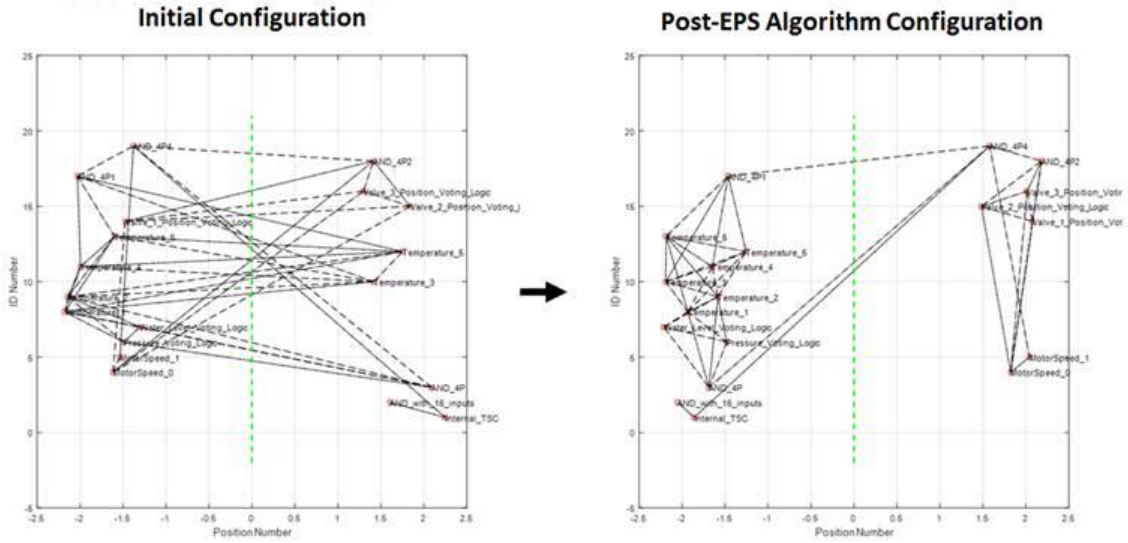

Figure 1-1 EPS NP-Hard Problem Solution

Plant diagrams are usually agnostic to specific architectures. Our tool takes a plant specific design and implements both the architecture and functional requirements with the NuPAC FPGA based platform. A design implemented with NuPAC still maintains linkages back to plant functional implementation even

though the individual logic partitions will be highly dependent on I/O constraints and the targeted plant architecture.

The team performed a number of feasibility analyses and case studies to determine the best algorithm to use going forward into detailed design, one that needed to be robust and repeatable enough for the logic and I/O partitioning use case.

1.1.1 NuPAC hardware routing platform

By understanding the underlying NuPAC hard-wired connections in the interconnected LVDS (low-voltage differential signaling) backplane mesh structure, a mathematical construct can be derived and served as an additional constraint on the algorithms when determining the best-placement onto the NuPAC GLM hardware. The algorithm also utilizes the layout of the NuPAC platform, which serves as a constraint that is applied into the equation to solve for the most optimal placement of any given design of 18 ASPLDs onto one NuPAC platform. This variant can also be extended onto multiple NuPAC chasses to include a complete Reactor Protection System (RPS) or Qualified Display System (QDS).

The NuPAC hardware platform is manufactured with hard-connections; there are four sets of four GLMs that have distinct interconnections to serve as high-traffic communication pathways. GLM 17 and GLM 18 are placed in locations so that these particular GLMs can drive inputs back/forth in-between the four sets.

1.2 Architecture and Hardware Development

1.2.1 Computer vision detection of scanned images

Digital logic diagrams serve to provide a functional layout on how a design is meant to behave. These digital logic diagrams are typically developed by one party, in this case a customer, and sent to another party to develop hardware that supports the design provided. The number of logic diagrams can span up into the hundreds and translation from a hardcopy to a model is a time-intensive manual process. The investigation utilizes computer vision techniques to detect ANSI (American National Standards Institute) or IEEE Std 91/91a-1991 Logic Gates, the interconnections, and character labels in order to translate it onto the EPS canvas.

The goal of the entire operation is quickly and efficiently convert a static drawing into model format that allows the developer to make changes as deemed necessary. As drawings can be drastically different from one another in terms of scale, component images, or other features – a full-blown translator is not feasible without setting restrictions. Therefore, the image processing algorithms utilized are developed with the intent of allowing a broad range of different drawings to be scanned and translated into a Model.

The diagram below shows the starting image and final result displayed side-by-side. The final results would be the starting point for an engineer to verify the translated diagrams match documentation and fix any issues that may have occurred in the translation process.

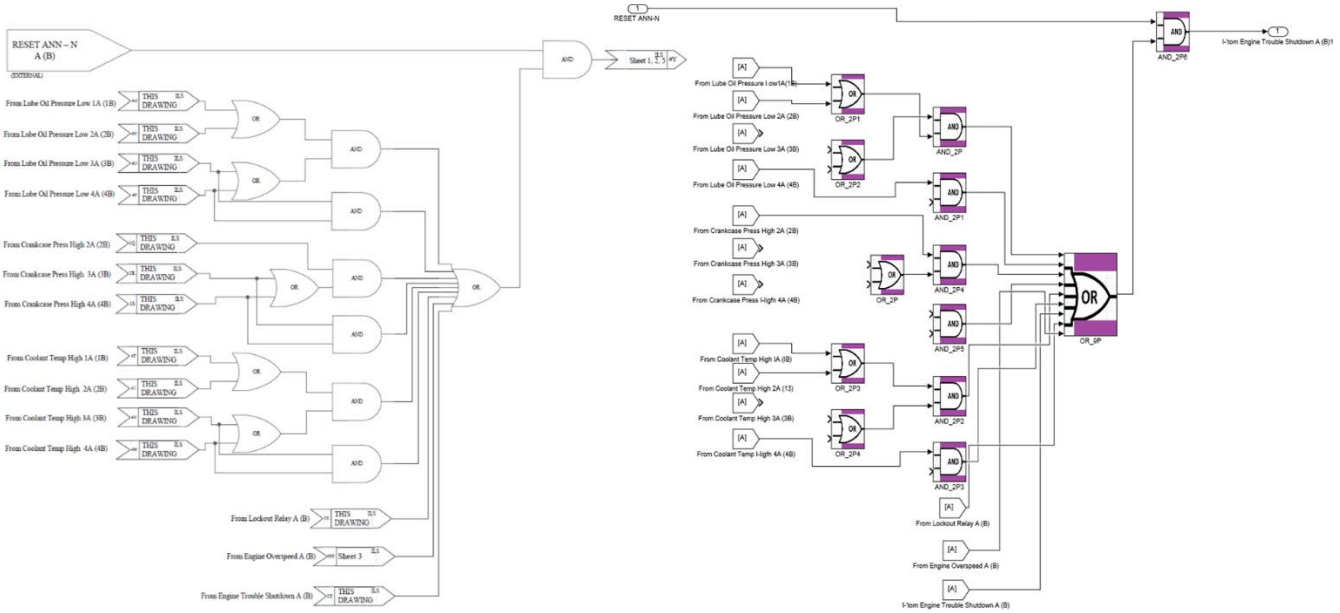

Figure 1-3 Translation Schematic Example

It should be understood that achieving 100% full-coverage translation is challenging. However, stress-test related results provide evidence that this technique provides a significant time savings over manually translating each schematic onto the EPS canvas. This technique provides a significant time savings over manually translating each schematic onto the EPS canvas. The most noticeable fallacies in the converted image are the missing connections and obscure character labels on some components.

The detection technique does fail in perceiving crossing connections and cannot decipher where multiple branches trace out to. Therefore, the algorithm does not associate the connections in order to allow the end-user to define these connections. In addition, the OCR detection utilized can at times misinterpret a character for another one. Most notable in the figure is the *"|-'tom Engine Trouble Shutdown A (B)"* which should actually read *"From Engine Trouble Shutdown A (B)"*. These issues should be resolved by an engineer to correct these translations. The above figure had a 100% conversion for all components on the canvas.

1.2.2 Common conversion file format

A common structure that can be manipulated in a modular format at the entity-level provides the basic functionality required to partition any complete functional logic diagram into any number of desired partitions. In addition, this structure allows complete forward and backward compatibility after any partitioning takes place. The configurable file can be fed back into modeling tools to be re-tested in a functional state and set-up for co-simulation with tools such as ModelSim for comparing the functional and physical mock-ups.

The structured file format was written with the intent of having a modular and easy to read format for PL engineers. The use of the structured file format is used when no pre-existing VHDL code exists and only a functional diagram exists. Other options include the ability to parse VHDL code and utilize that to partition within the EPS toolset. The goal is to characterize a functional structure that can be easily ported into the EPS tool while maintain all metadata parameters. This allows the EPS tool to partition out entities within the file and preserve the identity of the functional layout for all entities within a diagram in our prescribed format.

A typical file may typically contain hierarchical-level entities that only define a subset of the entities which contain actual I&C Element components. These hierarchical-level entities can either be ignored or utilized. If ignored, all lower-level entities within a hierarchical-level should be mapped at the highest level to avoid interdependent entities.

1.2.3 Structured file partitioning

The goal of having a file established in the structured format defined above is to allow partitioning to occur at an entity level, where all characteristics and appropriate architecture level characteristics can be preserved as entities are routed differently.

The component level can also be rearranged amongst GLMs, but doing so requires careful rearrangement of input/outputs and port mapping. The underlying basis for remapping a I&C_Element from one entity to another would be to route a I&C_Element that has certain I/O's that are dependent on one GLM as opposed to another.

1.2.4 EPS VHDL building blocks

Pre-built VHDL building blocks were created to be utilized for any black-box component a customer may desire. These VHDL building blocks consist of pre-designed and verified VHDL code-blocks. The VHDL blocks exist as primitives, and may be combined with others in a hierarchical fashion.

Each of these VHDL blocks are required to act as a wrapper of one or more I&C Library Blocks. Therefore, all programs utilizing any black-box structure would have an equivalent VHDL code-block that can easily be implemented when necessary. The EPS tool has the ability to align all of the appropriate signal mappings to any given I&C Component VHDL block which negates the need for a PL engineer to manually craft a VHDL file or test-bench file.

1.3 Partitioned system overview

One of the capabilities of EPS is that it may identify regions in which PL code can be partitioned. The motivation behind partitioning logic is two-fold: area and I/O constraints imposed by architecture or hardware. Area may constrain a design in the event that complex PL does not fit onto a single FPGA. On the other hand, I/O may constrain a design due to the fact that each GLM has a limited number of I/O channels. For these reasons, the partitioning feature of EPS can take in a large input logic design, identify the regions of code with minimum boundary crossings, and map these regions onto physical hardware resources.

All GLMs within a chassis are connected to a backplane, and are able to communicate with one another through a serial communications standard called LVDS. Therefore, any signals that are required to pass between GLMs as a result of partitioning may do so through the interconnected LVDS backplane.

In the example implementation below, EPS was used to partition an application between two separate GLMs. In its original form the application performed coincidence logic on trip sensor statuses coming from various GLMs. Depending on the voting results from each trip function, the application would either assert or de-assert its trip output signal.

Each partition has its own set of input and output signals. These signals represent individual sensor statuses, and can take on a binary logic '0' or '1' value. For example, if Division C's sensors were bypassed, then a notional Div_C_Sensors_Bypassed signal would read a logic '1' value.

The sensor status signals are provided to each GLM in the form of RS-422 serial messages. The messages require each of the active I/O Mezzanines to be configured for RS-422 communications. Intermediary messages may be passed between GLMs via the LVDS backplane

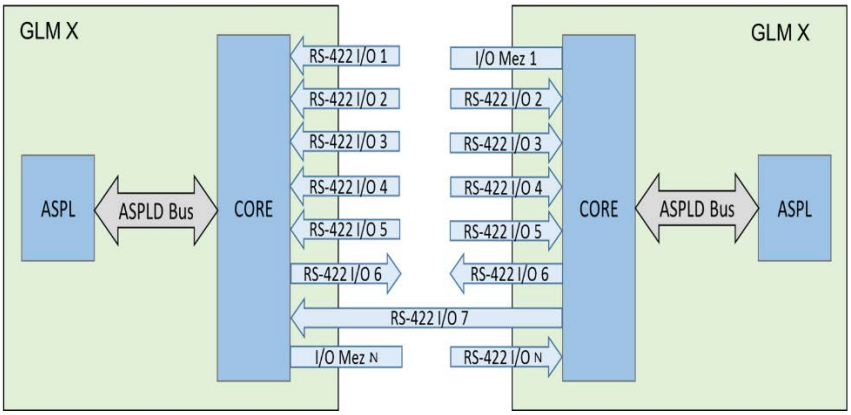

Figure 1-4 Two GLM I/O Configuration

1.3.1 Intra-GLM communications

The GLM has two main logic components: the Core and ASPL FPGAs. The Core serves to monitor/control signals coming from the I/O mezzanines, and to perform other functions common to any GLM (System monitoring, Built-In Test, etc.). The ASPL contains the application-specific programmable logic, which operates on the signals provided to it by the Core. Communication between ASPL and Core is accomplished via the ASPLD bus, which uses a Time Division Multiplexing protocol.

1.3.2 Application-Specific programmable logic

The ASPL in our implementation is composed of several logic modules to support communication between the Core and application partitions:

- **ASPLD Bus Interface Controller (AIC)** – handles bus communications between Core ASPL; implements protocol and error checking.
- **RS-422 Message Transceivers** – receives/transmits RS-422 messages**.**
- **LVDS Message Transceivers** – receives/transmits messages through the LVDS backplane.
- **Partition Logic** – represents the pure logic design output from EPS.
- **Finite State Machine** – handles the control of communication between sequential logic elements.

Figure 1-4 shows a generic, high-level block diagram of how an ASPL could be structured to support our architecture. The number of RS-422 transmitter/receiver blocks is dependent on the number of I/O mezzanines being used.

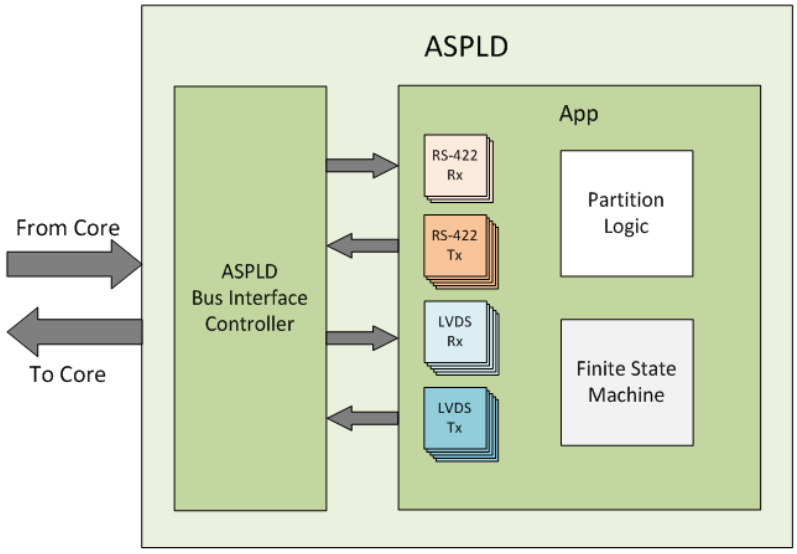

Figure 1-5 High Level Application Block Diagram

1.3.3 Test setup

Figure 1-6 below shows a basic test setup for our implementation. Only two GLM cards are shown in the figure above, but the setup can be adapted to any number of cards.

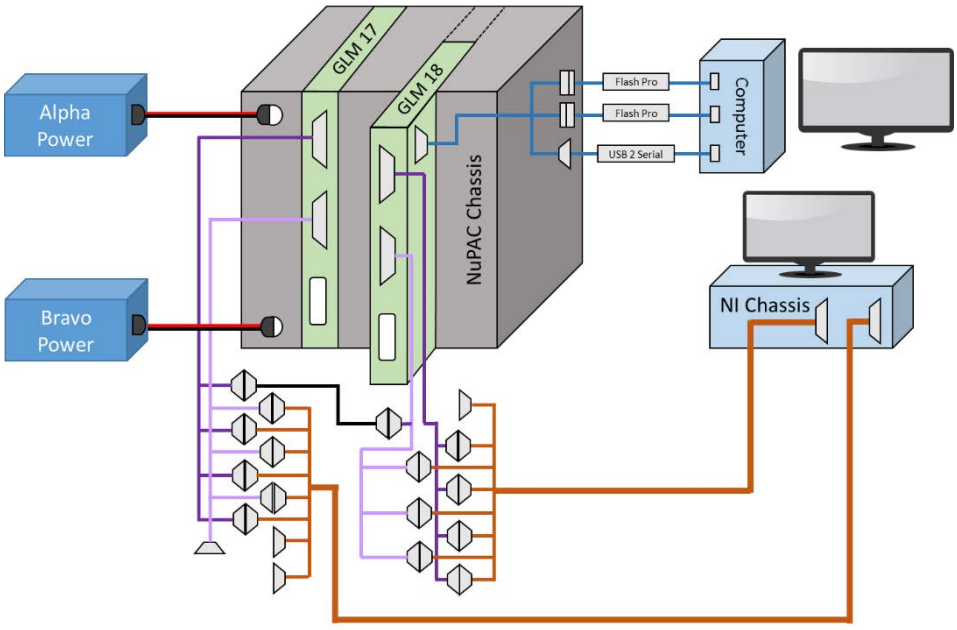

Figure 1-6 Engineering Production Suite Basic Test Setup

The logic development process is complete after a successful synthesis and place & route. Once ready to program, the user will select “Program Flash” in EPS, and choose the partition directory corresponding to the desired logic design. This process can be repeated for each GLM/partition that make up the system.

Testing the system is accomplished via a LabView GUI which operates on test vectors that are generated by the “Test Vectors” action in EPS. A test vector consists of a sequence of test cases in which signals are represented by binary values for logic “high” and “low”. EPS will generate test vectors for a user’s logic design, in the form of text files. Every I/O mezzanine that is being used in hardware will have its own test vector/text file.

2 Conclusion

Lockheed Martin’s Engineering Production Suite (EPS) minimizes the licensee risk and expense associated with modernization efforts through a developer’s toolkit for rapid, automated, and repeatable development of Programmable Logic used to implement plant functions and architectures for use on the Lockheed Martin NuPAC platform. As has been discussed in this paper EPS helps minimize cost, schedule, and technical risk exposure that often outweighed the benefit of upgrade efforts due to the unknown. Utilizing EPS, shown in Figure 2-1 below, users can create system level representations, simulate system functions, partition logic designs to accommodate hardware constraints through logic partitioning, automate the assembly of Application Specific Programmable Logic (ASPL), generate system test cases, program the NuPAC hardware, and perform testing throughout the design workflow.

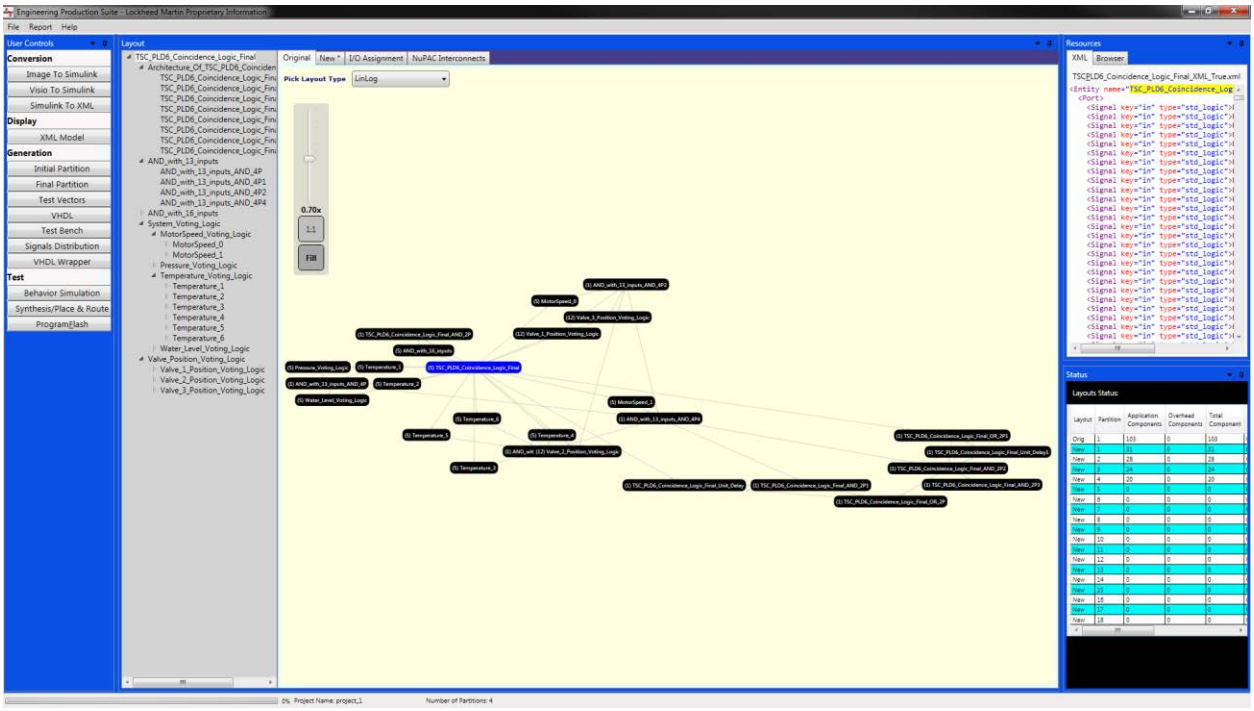

Figure 2-1 Engineering Production Suite Screen Shot