Test Automation Tool for “M/S Fitwel Tools and Forgings Pvt. Ltd.”

Mr. Varun Bhargav a, Mr. Naimeesh C.H.M b, Mr. Shamanth T c, Mr. Sachin B.R d

a Guide & Technology Consultant, M/S Fitwel Tools and Forgings Pvt. Ltd., Tumkur, Karnataka, India.

b Research Engineer and Mechanical Engineer, Bangalore, Karnataka, India.

c Research Engineer and Mechanical Engineer, Tumkur, Karnataka, India.

d Student, Mechanical Engineering, S.I.E.T, Tumkur, Karnataka, India.

Abstract: Increasing productivity along the development and verification process of safety-related projects is an important aspect in today’s technological developments, which need to be ever more efficient. The increase of productivity can be achieved by improving the usability of software tools and decreasing the effort of qualifying the software tool for safety-related project. For safety-critical systems, the output of software tools has to be verified in order to ensure the tools’ suitability for safety-relevant applications. Verification is particularly important for test automation tools that are used to run hardware-in-the-loop (HIL) tests of safety-related software automatically 24/7. This qualification of software tools requires advanced knowledge and effort. This problem can be solved if a tool is suitable for developing safety-related software. This paper explains how this can be achieved for test automation of tool from manual methods which is being implemented in M/S Fitwel Tools and Forgings Pvt. Ltd. Maximizing the productivity of a software tool’s use involves more than just considering the pre-qualification of the tool in accordance to safety standards. Enhancements in the tool’s usability with automation can also increase the productivity. Test automation tools usually let users develop tests by dragging blocks those define test steps. Signal-based tests are an advancement of this method, which is described in this paper. The aim of signal-based testing is to define test descriptions as if sketching graphs on a sheet of paper with a plotter-like editor to intuitively describe stimulus signals and reference signals as a sequence of signal segments, thereby lowering the initial hurdle towards setting up efficient test automation. In recent years, automotive components have become more sophisticated and the electronic control unit (ECU) has employed more complex large-scale software. As the product scale becomes larger, an increasing number of tests are required to assure product quality. Even in the case that the auto-testing tools are used, test patterns need to be input manually. This process requires significant amounts of man-hours particularly when the product types vary and the test patterns need to be modified for input signal changes. In order to improve the efficiency of this product development process, we have developed a tool that converts the simulation patterns used in model-based design into those for product tests. This tool automatically adjusts the input signals, and thus, successfully reduces the man-hours by 50% and improves the test quality with common test patterns. With the ever-increasing complexity of embedded software applications, and the emergence of more and more safety critical applications, thorough validation and verification of the code is needed. To address this need, many embedded software development groups are using models and doing upfront engineering before testing on the final product. Using the old style of testing late in the development cycle resulted in very long and expensive release cycles. Ford estimated that 60% of work tasks were to correct requirements or design defects that had been released to downstream developers. With today’s increasing need to get to market quickly with a safe product, this old style of testing is not adequate. Ford also used randomly generated unit test vectors, due to the lack of a commercially available tool, which only had approximately 75% coverage. Because of the need for safe systems, this level of testing is insufficient.

This paper presents requirements for model checking and unit test generation tools so that the tools are practical in a large production environment that is typical in the automotive industry.

Keywords – Automation, Productivity, verification, usability, software, and test method, model based development.
1. INTRODUCTION

The current automotive tools process is complex and multidisciplinary products. The design of tools is passed hundreds of times through hundreds of personnel from various organizations from the initial concept through construction and road test. The dense integration of embedded systems with physical processes in a vehicle is critical to achieve good fuel economy, low emissions, and better safety, and strength of material of tools among others. Currently, automotive design of tools has been siloed into specific disciplines and supported by highly specialized but domain-specific model based design (MBD) automation tools. For example, mechanical engineering is done in computer-aided design (CAD) and engineering (CAE) tools; electrical engineering is done in electronic design automation (EDA) and wire harness design tools; control engineering is done in Matlab/Simulink and Modelica and software engineering is done in UML and in-house software development environments. This paper reports our experience in the development of a novel multi-disciplinary integrated design automation tool for automotive cyber-physical systems. Our tool shows that the various disciplines in automotive design can be brought together to enhance the communication and requirements negotiation among engineers and organizations, enable multi-disciplinary simulations to evaluate the system-level impact of domain-specific design of tool decisions, and reduce the overall design cycle with automation. Our method relies on functional modeling to create a technology-independent description of what the system does, and uses a Functional Modeling Compiler (FMC) to synthesize technology-dependent solutions that can be directly used in multidisciplinary simulations for validation and design-space exploration. We can also implement electrical control units (ECUs). ECU software (hereinafter referred as “vehicle-embedded software”) is less likely to have a completely new design, and in many cases, the former model design is used in the development process of the software. When the former design is used, the former test pattern is also used in order to ensure software quality. However, there is the possibility that the entire design of the tool may be reviewed for functional decomposition to multiple ECUs and the design changes of connection interfaces. In this case, test patterns should be modified due to the changes of signal I/O interfaces for each ECU, even though the basic functions of the tools are not changed. On the other hand, the tool-embedded software tends to be more complex and larger. To solve the issues of the increase in the number of test items and risks for validation errors, the model-based development method has been promoted in the automobile industry. In this method, design verification is conducted by simulation using a design drawing (model) (MILS) (Fig. 1). This method enables us to improve the design quality of tool being manufactured, minimize rework, and detect bugs during the test process. Moreover, by using the common test patterns in the phases of design simulation and ECU testing, the efficiency of the entire development process can be improved. However, signal I/O timing may vary due to the variations of the signal I/O interfaces between the model and ECUs intended for verification (HILS). These issues may form a bottleneck in achieving the standardization. This paper describes how we can improve the efficiency in embedded software development by reusing the former design. Modeling allows for a well-defined algorithm from which verification and validation are practical as well as provides a mechanism for a high degree of automation. Today’s tools allow for a broad spectrum of uses for the models being developed. Some of these uses include: requirements capture, algorithm specification, algorithm validation and verification, documentation, automatic code generation, automatic unit test vector generation, hardware-in-the-loop testing, rapid prototype testing, and architecture specification. One of the biggest remaining problems is making these tools practical for the “typical” engineer working in a production environment. Most of today’s tools have been used very successfully by “high end” users, such as researchers and advance groups. These high end users are typically very motivated individuals with extensive training and ample time to learn the tools and experiment with them until they work. Unfortunately, the production engineers often have neither the training nor the time to experiment with the new modeling tools. These engineers need tools that are easy to learn, intuitive, and nearly push-button to use. Also, due to their overbooked workload, these engineers need analysis tools that can work on a single model file. They do not have the time to implement and double check the same algorithm in multiple tools. Model checking is an emerging technology for analysis of model based software designs. Model checking can also be used to automatically generate test vectors. While unit test vectors can be generated using specialized algorithms, many of the emerging automated test vector generation tools use model checking technology. These tools negate the property of interest and present it to the model checker. The counter example returned is the desired test vector, which exactly exercises the property. Currently, the standard practice in the automotive industry is to do a significant amount of in-vehicle testing but very little upfront testing. This is a very costly manner of conducting...
business, and the industry is trying to move towards a virtual environment in which most testing is done early in the development process. From the software testing point of view, the implication is that any testing is better than no testing. Thus, a tool that can help with any piece of automating the model checking or unit test vector generating would be useful. However, any testing that is done needs to be nearly push-button due to schedule constraints in the production environment. In other words, a highly automated tool which does part of the testing could potentially gain widespread use, whereas a partly automated tool that does everything many not get used at all. The rest of this paper will describe the types of model checking and unit test vectors that are of interest to the automotive industry, provide a brief overview of some of the available tools for modeling, model checking, and generation of unit test vectors, and describe an effort to make model checking and automated unit test vector generation practical for the automotive industry. We believe this also applies to related embedded industries such as aerospace, robotics, and medical devices also.

## 2. OBJECTIVES

### 2.1 Functional Modelling Compiler for Component-Based Simulation which can be implemented in FITWELL TOOLS AND FORGING

Automotive engineering consists of multiple design iterations starting at concept design of tools where the requirements of the product are defined, to detail design where the physical system is realized and tested. Functional modeling is a design activity where the informal or semi-formal requirements are formally specified in terms of functions. A function describes what the system does in terms of energy, material, and signal transformations yet it remains technology independent. Functional models are written in the Functional Basis, a high-level language close to natural language with well defined syntax and semantics to facilitate interdisciplinary communication among engineers from different domains. In the existing workflow, the functional model is a static document used by the domain engineers to translate requirements into engineering specifications in each of the disciplines that allocate functions to actual components. Unfortunately, it is very difficult for computer aided design tools from different disciplines to exchange data and take advantage of the model-based design at the system-level. To overcome the limitations of the current siloed development, this paper presents a FMC capable of automating the allocation of functions to components and generating feasible multi-disciplinary simulation models to validate different architectures and various components (e.g. ECUs, transmissions, engines, etc.) at the system-level. Our FMC is intended to be used as a design-space exploration tool at the concept design phase to evaluate how different combinations of automotive components can be leveraged to achieve the functional and non-functional requirements. Our FMC leverages the technological advances in hybrid simulation languages such as AMESim and Modelica to generate multi-disciplinary simulation models to validate the interactions between the physical and the cyber components of a vehicle. The commercial support of simulation component libraries written in these languages allows us to generate high-fidelity functional simulation models with components that have been validated by the vendors and are very close to the behavior of their real-life counterpart. These components include multi-physics components, and ECUs and control algorithms for internal combustion engines, automatic gearboxes, fuel cells, series- and parallel-electric cars. Figure 2 shows how the FMC is used to perform a design-space exploration of different architectures by combining different simulation components (e.g. engines, ECUs, controllers, etc.) stored and classified (e.g. mechanical, cyber, software, etc.) in a simulation component repository. The input to the FMC is a functional model and it generates multidisciplinary simulation models as an output.
2.12 TYPES OF DESIRED MODEL CHECKING AND UNIT TEST VECTORS IN FITWEL TOOLS & FORGING

Now we can present some model checks and types of unit test vectors which can be implemented in Fitwel that would be useful.

One particular challenge for the research community is that many of the models being made, especially for automotive powertrain applications, contain a mixture of control and data. The data consists of mathematical equations, which often have floating point variables. Most model checking tools cannot handle such data, since the state space is too large. Some of the emerging model checking tools is finding innovative techniques to deal with this large state space and produce results both in a timely manner and within the memory available on a standard PC. One alternative to completely exploring the entire state space is to use a form of depth-first search. A second alternative is to abstract floating point variables into a few Boolean conditions, for instance, replacing $x > 4.2$ with a Boolean $x$ Too Large.

Another challenge is that the models can be quite large. Depending on the item under test, the test tool may only need to deal with a small piece of the total application. Some of these pieces can be quite big as well. The test tools, while utilizing a minimum of time and computer memory, will need to analyze large models.

“Passive” Model Checks

The goal of model checking is to check that the specification is sound. One set of checks that are important can be termed “passive” checks, that is, the tester does not need to specify anything beyond the original model. They are predefined and commonly agreed upon. Some of these checks include: all states reachable, no unnecessary states, no graphical dependencies, all outcomes accounted for, no writes before a definition, no algebraic loops, and array indexes are all within bounds. In addition to helping validate the specification, passive checks may help the practical economics, too. Automotive applications are extremely cost sensitive. As a result adding off-chip memory is only done in exceptional cases, usually requiring the approval of someone high in the management chain. The preference is for the entire program to reside on-chip. Even though current microprocessors have more memory than their predecessors, wasting code is very undesirable. In can force the use of more chips. Consequently, identifying and removing unreachable or unnecessary states increases the efficiency of the code implementation, especially when an automatic code generation tool is used. Some tools, such as Mat lab’s & State flow, allow for the graphical position of model elements to determine how the model executes. This is inherently dangerous when the models are also used for documentation since apparently cosmetic changes in the layout may lead to subtle behavioral changes. This type of check should be optional as it may be acceptable and even needed by certain groups. For consistency, all outcomes of an expression should be accounted. Most tools, especially those that provide an executable specification, will flag algebraic loops before running a simulation. For those tools that do not have this built in, the model checker should perform this check. Another safety check is to ensure that array indexes are within the bounds for the given variable.

“Active” Model Checks

The next category of model checks can be termed “active” tests in that user input is required. For this case, an easy-to-use GUI is needed so the tester can input the checks in an intuitive or at least easily learned language. Most model checking tools, such as Z or SMV, require a very specialized input format, which is unfamiliar to the typical production engineer. Model checking is more traditionally used for these kinds of active tests. For practicality, a GUI should allow the tester to create the desired checks. These tests should be able to be saved to a file for future use.

Unit Test Vectors

One way to break down the problem of testing into manageable pieces is use different coverage levels for the test vectors. The most common coverage levels are statement coverage, decision coverage, MC/DC coverage and some form of all paths.
Another coverage type is boundary values; see definition below. A tool is needed in which the tester can select the desired coverage level and for which a minimal set of tests should be generated to accomplish the selected coverage criteria. The different coverage levels and a minimal set of test vectors have the same end purpose: identify errors in a minimal fashion. Due to time constraints, if the test engineers get multiple tests that fail for the same reason, they will probably get frustrated and stop using the tool. Coverage levels allow the tester to progressively increase the thoroughness of the testing. Hopefully the less stringent coverage levels identify major bugs. Once those are fixed, the more thorough coverage levels will find the more subtle bugs. The more thorough coverage levels typically take longer for the tools to generate the test vectors. Therefore, by starting with the lower coverage levels, the more time consuming tests can be run fewer times, saving overall testing time. Also, producing a minimal set of test vectors for a given coverage reduces time to execute tests and analyze the results of each test.

The coverage levels listed are the standard coverage measures that have been defined in the literature for years. However, these are defined for testing the source code. These coverage levels will need to be modified to apply to models. For data flow models, such as gains, addition, and multiplication, the definitions are straightforward to convert. Special care should be taken for blocks that require control logic to implement, the boundary value coverage is intended to test the values just above and below a decision value, such as, \( x > 10 \). If \( x \) is an integer, the values that should be tested are 9, 10, and 11. If \( x \) is a floating point variable, the values should be \( 10 + \delta \), where \( \delta \) could be a user defined quantity. For state machines, the standard coverage measures need some redefinition. For example, statement coverage can be redefined as touching every state or using every transition between the states. The mixing of data and control flow is particularly important for state machines, as many of the transitions depend on variables which are potentially calculated outside of the state machine. Notice that to test state machines, a sequence of test vectors is needed. Some testing tools expect to be told the state variable and set this state variable to the desired state for the test in question. This approach does not work when an unknown source, such as a person, is generating the code for this state machine. Thus, the test vector generator tool needs to produce a sequence of test vectors that starts with the default state and progresses through the state machine to get to the state under test. Another tool that would be beneficial is one where the tester can specify the desired end state. This will place the model in the desired state and keep the rest of the model “legal,” while letting the tester manually continue the testing from this point.

3. METHODOLOGY

As we all know how the basic processing is done, here we will be reviewing the further process of raw material which comes to FITWEL and what processes it goes through.

Right now the manual methods are being used and implemented which needs to modernized from the objectives which has been mentioned and taken into consideration. All process will move towards automation in near future from this implementation.

The raw materials which are usually got in form rolled steel is being received by forging unit in the form of bars, then there is cross check for quality, visual & dimensional inspection as few are already been mentioned in the objective sector. The bars of rolled steel are cached in the raw material yard and are processed into appropriately sized cut blanks based on the specifications set by the engineering department. These cut blanks are then sent to the forging lines as per the set schedule, Once the forging process is complete. A quality check is done, Here the parts are segregated to move either towards further process or towards quarantine, parts sent to quarantine are either reworked (hot working or fettling), scrapped or yellow tagged for clearance from the customer for the deviations. When the product is OK, if all standards are met; the part moves on for the heating treatment.
The heating treatment includes

- Normalizing/Hardening
- Tempering/annealing

Upon the completion of the above procedures, there is a check for hardness & micro-structure. If the product has a fault in either at the microscopic level or in surface hardness; it again goes back for re-heat treatment only if the composition of the steel and the customer quality specifications align to accommodate for multiple heat treating operations, if all specifications are met, the part moves to the shot blasting section which is followed by coining (or cold working). After the cold working, there is coining inspection being done. Cold working of parts causes strain hardening and hence reduces the scope for re-work. This leads to a higher set-up time for the coining process and more frequent in-process quality checks. Visual and dimensional inspection of the parts is conducted in tandem with the cold working operation to ensure minimum losses in case of deviation. If any dimensional deviations are found in the part at this stage, the process is stopped and the tooling is corrected, if the deviation is beyond the correctable bandwidth, there are chances for total rejection which becomes designated as “scrap” (No use). We need to note that ‘quality control re check (random)’, ‘re-work’ and ‘scrap yard’ are taken into consideration. For the yellow tagged re-work production process being followed as mentioned above or else it is given a statement- “Cleared for Dispatch”.

For our implementation all these process will be aligned in a single way from raw material to finish product from our objective called automation.

4. CONCLUSION

Its described development of a test-supporting tool for more efficient testing and its effectiveness in dealing with sophisticated software. First of all, we established a technology that uses common test patterns for simulation before and after ECU design change to secure product quality at the common software development. As a result, we achieved 20% reduction of man-hours required for changing the test patterns. Secondly, as part of our efforts towards the model based development, we established a technology that uses the common test patterns at a phase of design simulation and ECU testing. Using the auto-testing tools, we achieved approximately 50% reduction of man-hours compared to previous conditions. We believe that it is possible to secure product quality in a short time and minimize the rework rate at the product testing phase. We continue to make further efforts for improvement of the evaluation process for the model-based development, in order to achieve high efficiency.

There is a significant need for more upfront engineering in today’s embedded software design of tool process. Within the automotive area, very little upfront testing has been done. With the introduction of executable modeling tools, this upfront testing is more feasible. It is the job of the tool vendors to make this testing technology available and practical to the end user.

Due to the constraints placed on the production engineers, principally limited time, the test tools need to be: nearly push-button to use, intuitive to learn, and connect to the tools that they are already using. Some additional characteristics that will make the test tools practical include: identify any error just once, allow the user to select the coverage level of interest, minimize the time to check the model or generate the test vectors, use the language of the production engineer, run on a standard desktop PC, generate a sequence of test vectors for testing state machines, and handle large models of tools and tools that consist of both data and control flow with a large state space.

These challenges in M/S Fitwel Tools and Forgings Pvt. Ltd may seem daunting for the tool vendors, but providing a partial solution is better than no testing at all. And many tools are on the verge of being practical in today’s production environment.

5. TECHNICAL TERMS

- ECU (Electronic Control Unit): An electrical control unit mounted on a vehicle.
- MILS (Model in the Loop Simulation): Software verification process intended for a designed model.
- HILS (Hardware in the Loop Simulation): ECU verification process in a simulated operating environment. Direct connection: Cable connection to transmit a single signal.
- CAN (Controller Area Network): A network standard for connecting electrical circuits or devices

6. REFERENCES


16) ETAS, www.etasinc.com

17) I-Logix, www.ilogix.com

18) Rational, www.rational.com

19) ADI, www.adi.com

20) T-VEC, www.t-vec.com