ANALYZING DEPENDABILITY MEASURES AT THE ELECTRONIC SYSTEM LEVEL

Marc Michael *Daniel Große* *Rolf Drechsler*

Institute of Computer Science
University of Bremen
28359 Bremen, Germany
{mmichael,grosse,drechsle}@informatik.uni-bremen.de

ABSTRACT

Raising the level of abstraction to design the next generation of embedded systems has become mandatory. This design methodology is commonly referred to *Electronic System Level* (ESL) design. Simultaneously, dependability of embedded systems becomes a major concern. To satisfy these demands already at ESL, we present a dependability analysis approach working directly at this level. The approach analyzes the effectiveness of dependability measures in SystemC-based virtual prototypes. Errors are injected into SystemC transactions using an XML-based configuration mechanism. This is combined with the specification of the expected behavior with respect to the injected errors. The developed analysis approach allows for validation of dependability measures as well as localization of missing or buggy measures. Experimental results for a complex image processing system, which determines the position of a game controller in video data, demonstrate the advantages of our approach.

1. INTRODUCTION

The preferred approach to address the rising complexity of embedded systems is *Electronic System Level* (ESL) design [1]. The key feature of ESL design is abstraction resulting in virtual prototypes for the complete hardware and software platform of embedded systems. This allows validation, architectural exploration, performance analysis and software development much earlier in comparison to traditional design flows. In this context, SystemC [2, 3, 4] has become the standard language for ESL design. Clearly, the key enabler for the success of SystemC was *Transaction Level Modeling* (TLM) [5, 6]. In the last years a substantial body of academic and industrial progress in methods for SystemC TLM has been made. In addition, most EDA companies provide SystemC-based ESL tools nowadays which support the design from abstract models down to hardware.

At the same time dependability of embedded systems becomes a critical issue. A major reason is the steadily shrinking of the silicon process technology leading to unreliable circuit components. As a consequence, system failures may result from transistor failures, wear out or radiation. Thus, many methods and techniques have been developed to increase the robustness of embedded systems. In particular, fault-tolerant design techniques are used for this task [7, 8, 9, 10]. Basically, some form of redundancy is added to the system for error detection (and error correction). Moreover, approaches to analyze the achieved fault tolerance and reliability have been proposed, e.g. [11, 12, 13, 14, 15]. However, these approaches do no target dependability at ESL or focus on mixed fault simulation (more details are discussed in the related work section).

In this paper, we present an approach to analyze whether dependability measures of SystemC virtual prototypes are effective. After functional verification, the designers address the dependability challenges by integrating respective measures. These include for instance error correcting codes, checksums or algorithmic redundancy and are implemented in hardware or software in the abstract SystemC model. Then, to estimate their effectiveness errors have to be injected and it has to be checked that the system continues to operate properly. Obviously, for this task a flexible way for both, injecting errors and evaluating the effects of errors with respect to the specification is necessary. Due to the high level of abstraction of SystemC TLM our analysis approach injects errors by mutation of the TLM communication. That is, the parameters (payload, address, etc) of the TLM communication functions are mutated. Thereby, we can cover a wide range of faults since single bits can be changed as well as the behavior of a complete module can be changed modeling a complex error. In our approach this fault/error injection is controlled by XML-based configuration files. Besides the types of errors also their frequency, complex scenarios over time and the expected result with respect to the concrete dependability measure can be specified. As a consequence, if for some error the system malfunctions it is possible to identify modules which are not robust. In this case the integrated dependability

measures needs to be fixed or additional measures need to be integrated. Overall, we summarize the contributions of this paper as follows:

- Configurable TLM error injection and expected behavior specification
- Validation of implemented dependability measures
- Identification and localization of missing or incorrect dependability measures

The remainder of the paper is structured as follows: Section 2 describes related work. In Section 3 the proposed analysis approach for dependability measures in SystemC ESL designs is introduced. The experimental evaluation is given in Section 4. Finally, the paper is concluded in Section 5.

2. RELATED WORK

The application of mutation-techniques for different verification problems of abstract SystemC designs has been considered in literature. A general mutation model for SystemC TLM 2.0 communication interfaces has been presented in [16]. This model has been used to measure the quality of functional verification with respect to the verification environment in [17]. Mutation testing with the focus on concurrent designs has been proposed in [18]. A general tool for error and mutation injection of SystemC models has been presented in [19]. But all these approaches do not target dependability.

A virtual platform of the LEON3 processor with fault injection has been developed in [20]. In principle this platform can be used as basis for our approach.

There are approaches which target the modeling and simulation of faults across different levels of abstraction including TLM [21, 22]. However, they focus on the evaluation of the fault effects and efficient simulation. The analysis of the effectiveness of implemented dependability measures (including identification and localization of missing or incorrect measures) is not considered in these papers.

3. ANALYSIS OF DEPENDABILITY MEASURES

In this section the proposed analysis approach for dependability measures in SystemC ESL models is introduced. Before we give the details, the overall flow is explained.

3.1. Overall Flow

Fig. 1 depicts the overall flow of the proposed approach: Starting with a SystemC TLM virtual prototype of the embedded system, functional validation is performed to check whether the system conforms to the functional specification. Then, dependability measures are implemented and their effectiveness has to be ensured. For this task, the design needs to be simulated with specific errors. To make the required

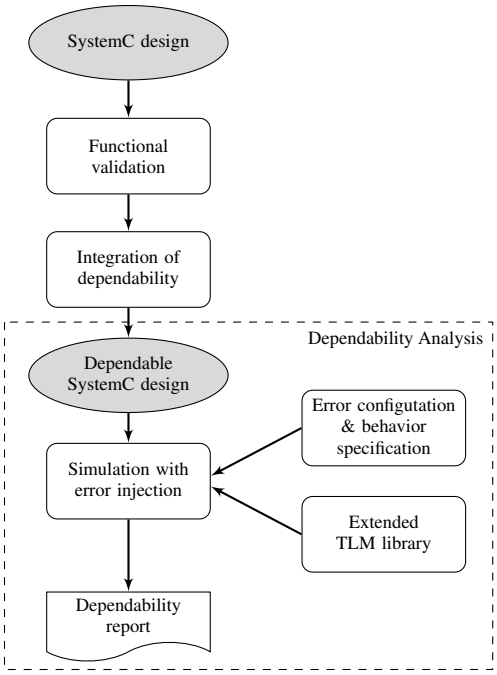

Fig. 1. Overall flow for analyzing dependability measures

error injection process convenient for the user we have developed a configuration mechanism. The user can specify the errors and complex error scenarios for injection together with the expected behavior of the system. To carry out the error injection in the SystemC TLM design, we have extended the TLM library. Basically, we mutate the transactions of the SystemC model to inject an error. To check whether the dependability measures take effect dedicated logging and comparison methods are provided in our library. Finally, after the simulation a dependability report is created by our approach. This report gives information about the quality of the design dependability: the positive cases where for example an error has been corrected, but also the negative cases where the error is not treated as specified are reported. Since the erroneous transactions and their communication partners are known information can be derived where the design robustness needs to be improved.

In the following we describe the error injection, the configuration mechanism, and the analyzing methods in detail.

3.2. Error Injection by Transaction Mutation

Due to the high level of abstraction of SystemC TLM models it is sufficient to inject errors at the communication boundaries, i.e. in the transactions. By this, low-level faults as well as complex errors can be covered. To inject an error we mutate a transaction. Table 1 lists the possible mutations for the elements of the transaction payload. As can be seen the command and the response can be substituted by another message.

Table 1. Mutations for a transaction

Element	Type	Modification
Command	enum	substitute
Address	unsigned int	mask
Data	unsigned int*	mask
Length	unsigned int	mask
Response	enum	substitute

Table 2. Mask attributes

Mask attributes	Range	Default value
random	on | off	off
percentage	-1..100	100
start_pos	-1..max_size	0
length	-1..max_size	max_size

-1 sets value to random

The address, the data and the length of the data can be modified with a mask. Additional attributes can be defined for the mask to support different types of modifications. These attributes, its ranges and the default value are shown in Table 2. By enabling randomization the effect of the mask is randomly distributed over the data according to the percentage, the start position and length. The percentage describes how many data elements are modified by applying the mask to each element. The start position defines the absolute start position where the data should be modified and length defines the amount of data to be modified, respectively. Alternatively the value of percentage, the start position and the length can be set to random (encoded by -1 as can be seen in Table 2).

It remains to explain where the mutations are applied to inject the errors. For this task, we have extend the SystemC TLM-2.0 library [6] such that if the standardized transport functions (*b_transport* or *nb_transport_fw, _bw*) are called via a target socket, a function in our library is executed in between.

Fig. 2 shows the principle in a simplified way: The initiator sends a transaction to the target. This transaction can be mutated using the error configuration mechanism before it reaches the target. Moreover, information about the concrete mutation and the origin of the transaction is logged for the analysis later. In general, for each target socket individual error configurations can be specified.

Based on these error injection methods we introduce the configuration mechanism in the next section.

3.3. Error Configuration and Behavior Specification

We have developed a user-friendly XML-based configuration mechanism to describe the types of errors to be injected, complex error scenarios and the expected behavior.

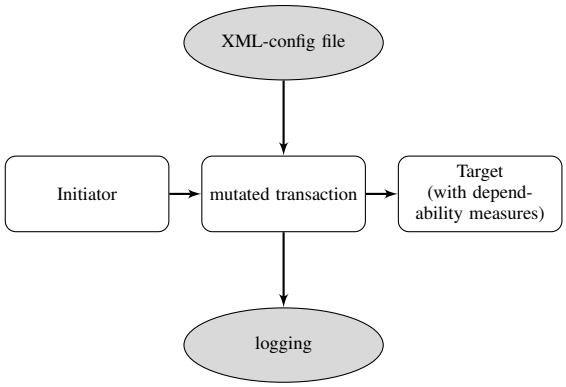

Fig. 2. TLM mutation

Table 3. XML elements for error injection

	Sequence	Attributes
Scenario	sequential or randomized	random, weight
Modification	sequential or randomized	random, weight
Transaction	once	identifiable, correctable

In general, a configuration file is a hierarchical structure based on the following elements:

- Scenario
- Modification
- Transaction

A *scenario* is the highest element in this structure and represents a sequence of defined modifications. Every configuration file must have at least one or more scenarios. The scenarios in the configuration file are processed sequentially. Every scenario includes a non-empty list of modifications which are also processed sequentially. A *modification* represents one mutated *transaction* using the features as introduced in the Section 3.2 and is illustrated below by examples.

The configuration file structure is demonstrated by the following simple example:

```
<scenario name="scenario1">
  <modification detectable="1" correctable="1">
    /* one specified mutation */
  </modification>
</scenario>
```

As can be seen two attributes can be set for a modification. The goal is that the user specifies the *expected behavior* when the respective modification is applied, i.e. whether a dependability measure detects the injected error or even corrects it. Please note that if none of these attributes are given the idea is that the effect of the corresponding error is acceptable.

General configuration options for a scenario (and contained modifications) are given in Table 3. The execution order of scenarios and modifications can be set from sequential to random. If it is desired, the scenarios and the modifications

can also be weighted differently to generate different mutation occurrence probabilities.

The elements of a transaction payload as shown in Table 1 can be separated in two classes: Enums (for command and response) and unsigned integer (for address and length and an integer array for data).

Commands and responses can be substituted by a fixed message or a randomized weighted message. A concrete example (belonging to a modification in the configuration hierarchy) demonstrating the substitution of a TLM command is:

```
<commands>
  <request name="READ" />
</commands>
```

The next example shows a randomized weighted substitution:

```
<commands random="1">
  <request name="READ" weight="2" />
  <request name="IGNORE" weight="1" />
  <request name="ORIGINAL" weight="9" />
</commands>
```

For error injection into address, length and data we use masks as already mentioned in Section 3.2. The mask can be randomly distributed over the data structure according to percentage, start position and length. A mask can also be defined as a list of two or more masks which are applied step-by-step. The following example demonstrates the error injection into the address of a transaction:

```
<address>
  <destination>
    <and mask="0xaa" />
    <or mask="0x55" />
  </destination>
</address>
```

Modifying certain elements of the payload data can be done as follows. Here, 90% of the bytes in the range from 200 to 300 are modified:

```
<data>
  <transfer random="1" percentage="90" start_pos="200"
  length="100">
    <or mask="0xff" />
  </transfer>
</data>
```

Please note that if in a modification an element of a transaction (command, address, data, length or response) is not specified, this element remains unchanged.

To load a configuration file and perform the defined error injection for a target socket using mutation, only two lines of code are necessary:

```
Mutation mutation("path/to/configFile");
module.target_socket.set_mutation(&mutation);
```

The mutation can also be used to modify other target sockets. Alternatively it is possible to load for every single target socket a separate configuration file.

3.4. Analyzing Dependability Measures

So far we have only considered error injection. In the following we describe our automatic analysis method to evaluate whether the implemented dependability measures have been effective with respect to the injected errors.

Therefore, we create a dependability report after simulation with detailed information about the effect of errors and the potentially involved dependability measures. Based on this report the designers can correct or improve the measures and identify parts of the system which are not robust.

For the analysis each injected error as defined by the configuration, i.e. each mutated transaction, is logged automatically. The logged *unchanged* information includes:

- the current simulation time,
- the name of the module the target socket belongs to,
- the command,
- the address,
- the data length,
- a hash of the data,
- if the injection is detectable and
- if the injection is correctable.

Since we need the information whether a implemented dependability measure was able to detect (and potentially correct) the error, the following method needs to be called when a dependability measure handles an error:

```
log_detected_error(module_name, transaction);
```

Similar data is logged as before. However, when this method is called the erroneous transaction is actually handled by a specific dependability measure. Hence, at this point we can conclude that the error has been detected. In addition, in a post processing step after simulation we analyze if the error has been corrected based on the logged information. Overall, for the dependability report we can compute the number of detected erroneous transactions in relation to all injected errors as well as the number of detectable erroneous transactions (according to the user expectation specified in the configuration) in relation to all errors. Thus, errors which have not been detected by a measure or are not treated as specified are revealed. Since information is available at which target port the error has been injected we are able to identify modules which are not robust or use an incorrect measure.

In the following section we present the experimental evaluation.

4. EVALUATION

This section gives the experimental evaluation of the proposed approach. At first, the test environment used for the evaluation is described. Then, the results are presented and discussed.

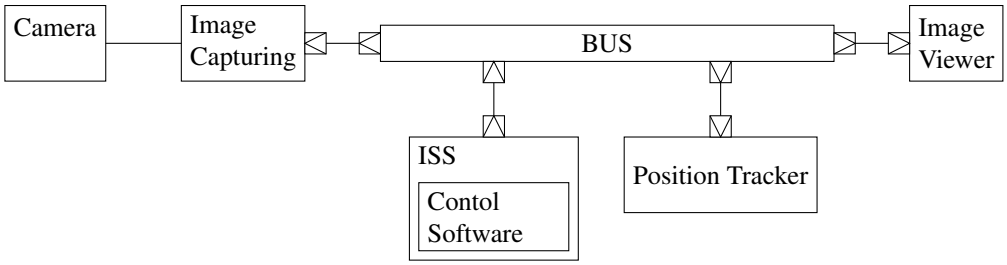

Fig. 3. System overview

4.1. Test Environment

We have implemented an image processing system which determines the position of the PlayStation™ Move Motion Controller [23] on a video stream. Basically, the controller illuminates a ball on its top. We recognize this ball and hence can determine a 3D position, i.e. x, y, z coordinates where the z coordinate can be derived from the radius of the ball.

Fig. 3 shows the architecture of our SystemC TLM implementation. We use an *Instruction Set Simulator* (ISS), namely Or1ksim [24], which runs the control software of the system. The system works as follows: At first the software in the ISS loads an image from the image capturing module which grabs the frames from a connected camera. Then, this image is send via the bus to the position tracker module which performs several image processing algorithms to calculate the position of the controller. In particular, this module converts at first a color image to a gray scale image, then the Sobel operator for edge detection is used and finally Hough transformation [25] is performed to identify circles which represents the 3D position of the controller in the picture. Basically, these steps are shown in Fig. 4. The control software waits until the position tracker module is ready and loads the new information from the module. After receiving the information the image together with the position of the controller are send to the image viewer. The image viewer displays the received image and marks additionally the position of the controller in the image with a circle as shown in the bottom of Fig. 4. By drawing the position directly on the image it is easy to see if the result of the position tracker module is correct.

In our image processing system the implemented dependability measures are not protecting the entire content of a transaction. In compliance with the dependability specification of the system we only ensure the correctness of the TLM command, the address, data_length and the response_status using parity bits.

For the evaluation we use a video file instead of live video from the camera to guarantee deterministic behavior. This is necessary to compare the behavior of the system with and without errors.

All following experiments have been carried out on an AMD Phenom II X4 Quad-Core with 8 GB of main memory. Depending on the injected error type, the run-time overhead for error injection was in the range of a factor of 1 to 3. Essentially, the mutation time is dominated by the size of the modified TLM transaction data element.

4.2. Test Scenarios

To analyze the implemented dependability measures we have built different configuration files for error injection. They can be divided into several scenarios.

Scenario 1 and 2 represent the incorrect usage of the camera or a damaged one. Such errors lead to image artifacts (as exemplified in Fig. 5). The artifacts in the first scenario are small: 5% of the image is corrupted. In the second scenario 33% of the image is corrupted. In both scenarios the artifacts are distributed over the image randomly. To represent these errors we mutated the data of the transactions between the image capturing module and the ISS.

Scenario 3 represents software errors of the control software which is executed on the ISS. In this scenario the software errors lead to incorrect transactions to the image capturing module. One bug sends TLM_WRITE commands instead of the TLM_READ command and the second modifies the data_length value.

Scenario 4 simulates a corrupted connectivity between the position tracker module and the ISS. A corrupted connection leads to faulty transactions. These errors were represented as mutated TLM commands, addresses and data_length. Every fourth transaction is not mutated to simulate a temporally broken connection.

4.3. Results and Discussion

The results for all scenarios are summarized in Table 4. The first column *Scenario* gives the number of the test scenario. Scenario 0 represents the implemented system without any error injection and is used as a reference for the remaining scenarios. Column *Correct results* shows how often the controller has been found at the same position as in Scenario 0. The column *Tolerated results* gives similar to column *Correct results* how often the controller was found at the correct position but with a tolerance of 10 pixels for the x, y and

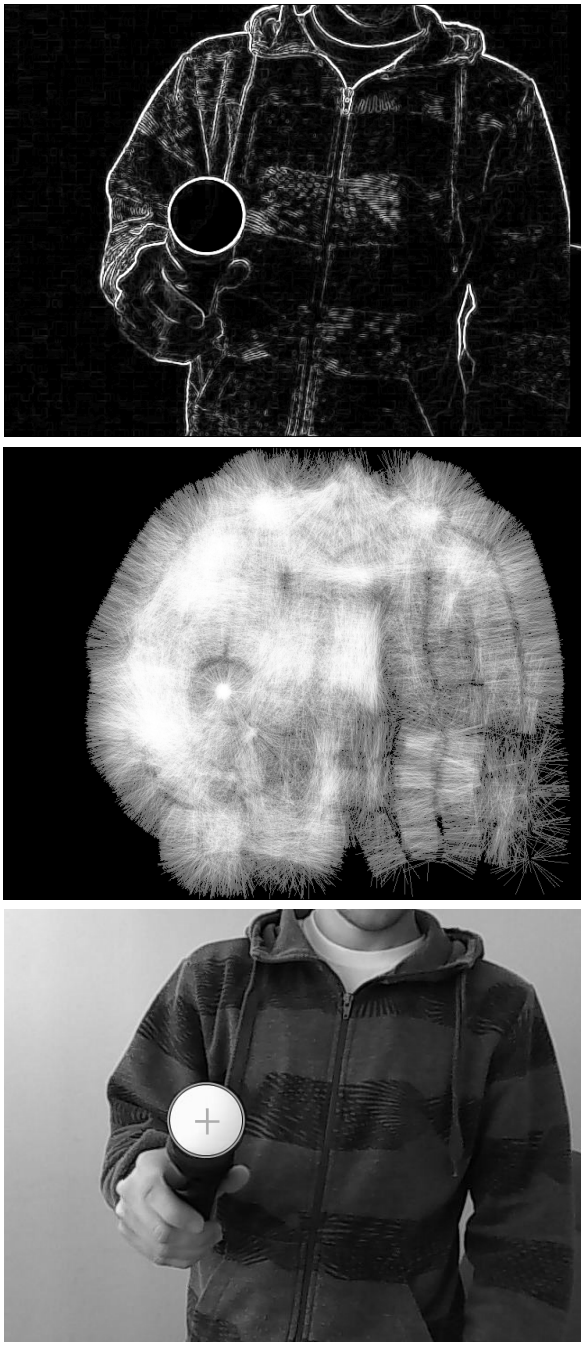

Fig. 4. Image processing flow: Sobel image (top), Hough image (middle), result (bottom).

z coordinates of the controller. In both columns the fraction with respect to the value in Scenario 0 is provided in brackets. The numbers of injected errors are shown in column *Errors*. How many errors have been detected is provided in column *Detected errors*, as well as the percentage of the injected errors. The number of how many detectable errors have been identified as such by a dependability measure is shown in column *Detectable errors*, and again the percentage of detectable errors as specified in the XML configuration. The last column *Corrected errors* provides the number of errors which have been fixed by the implemented dependability measure, i.e. gave the same result after error correction in comparison to the original unchanged transaction payload. Here, also the percentage of correctable errors as specified in the XML configuration is shown.

Fig. 5. Result after error injection.

In Scenario 1 and 2 no errors have been detected. The dependability measures in the system are not able to detect mutated data of a transaction. The implemented dependability measures are only able to detect erroneous TLM commands, addresses and data_length since we accept if images are broken in compliance with the specification. The results have been correct in 83% of the cases when mutating 5% of the image (Scenario 1) and in only 49% of the cases when mutating 33% (Scenario 2). The figures of the tolerated results (third column) are clearly better: 97% for Scenario 1 and 66% for Scenario 2, respectively. The concept in the dependability specification for the considered test scenario is confirmed by these results because if camera faults lead to too many errors the overall image quality becomes too low and hence the camera has to be replaced by a new one.

The position results for Scenario 3 are always correct. No errors have been injected into the image data which leads to a correct calculation of the controller's position. However, only about half of the errors have been detected. By looking closer into the logged data of the dependability report the reason could be found very fast: Usually, the camera module receives only TLM_READ commands and ignores other TLM commands. As a consequence, all mutated TLM command transactions have been ignored by the camera module, but later retransmitted since for these transactions the TLM communication was not successful. Obviously, we have found a missing dependability measure, i.e. the TLM command is not checked. If every element of a transaction except the TLM

Table 4. Experimental results

Scenario	Correct results	Tolerated results	Errors	Detected errors	Detectable errors	Corrected errors
0	813 (100%)	813 (100%)	-	-	-	-
1	678 (83%)	789 (97%)	1626	0 (0%)	0 (-)	0 (-)
2	398 (49%)	537 (66%)	1626	0 (0%)	0 (-)	0 (-)
3	813 (100%)	813 (100%)	1626	813 (50%)	813 (50%)	0 (0%)
3b	813 (100%)	813 (100%)	813	813 (100%)	813 (100%)	407 (100%)
4	609 (75%)	710 (87%)	1221	813 (66%)	813 (66%)	407 (100%)
4b	813 (100%)	813 (100%)	1221	1221 (100%)	1221 (100%)	407 (100%)

```
1  ...
2  unsigned int parity_bits = calc_parity_bits(trans);
3  if(  trans.get_command_parity()           == (parity_bits >> 4) % 2
4    && trans.get_address_parity()           == (parity_bits >> 3) % 2
5    && trans.get_data_length_parity()       == (parity_bits >> 3) % 2
6    && trans.get_response_status_parity()   == (parity_bits >> 1) % 2
7    && trans.get_parity_bit()               ==  parity_bits       % 2
8  ) return true;
9  ...
```

Fig. 6. Incorrect dependability measure

command is okay the transaction could be corrected by replacing the wrong TLM command with TLM_READ. The results after adding this dependability measure are shown in Scenario 3b. The number of errors is only half of the number in Scenario 3 because no erroneous transaction will be ignored by the camera module (such a transaction is corrected). Overall, 100% of the errors have been detected and the TLM command mutations (50% of all mutations) have been corrected.

The results in Scenario 4 are correct in 75% of the cases. The reason is similar to Scenario 3: The image data has not been changed but 25% of the transactions contains an erroneous TLM command which results in no information (command TLM_WRITE) or old information (TLM_READ) about the actual position of the controller. In this scenario only 66% of the detectable errors have been found due to a copy-paste error when implementing the dependability measure. An excerpt of the code of the dependability measure is shown in Fig. 6. The parity bit of the data_length was also compared to the parity bit of the address bit (line 5, shift again by 3 instead of 2). The parity bits of both were always set to 1 in the scenario. Thus, every erroneous data_length will not be detected by the dependability measure. After fixing the code the achieved results are shown as Scenario 4b in Table 4. Every error has been detected and the position of the controller has always been correct.

In summary, with our approach we can validate dependability measures, identify missing ones and localize incorrect measures.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have presented an approach to analyze the effectiveness of dependability measures of SystemC virtual prototypes. The approach uses an XML-based configuration mechanism to specify the errors injected into the system as well as the expected system response. The error injection is performed by mutating the elements of SystemC transactions. This allows covering a wide range of errors, i.e. low-level faults but also complex errors. After simulation a dependability report is generated summarizing whether all injected errors have been treated in compliance with the dependability specification; that is, whether the detectable (correctable) errors have been detected (corrected). In addition, information where the design robustness needs to be improved can be derived since the communication partners are known at ESL as well as where the error was not handled in contradiction to the specification. For a complex image processing system we have demonstrated the advantages of our approach. Besides successfull validation of the integrated measures, we have found a missing measure as well as a buggy one.

For future work, we plan to extend our analysis method such that timing information, which is for example available in approximately timed TLM models, can also be used as dependability criterion. Since the overall analysis quality depends on the simulated scenarios we plan to use coverage techniques to ensure the testbench quality, e.g. [26]. Moreover, the qualification of the injected errors is another important issue such that the specified error scenarios can be improved to cover a larger error space.

6. ACKNOWLEDGEMENTS

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project SANITAS under contract no. 01M3088.

7. REFERENCES

[1] B. Bailey, G. Martin, and A. Piziali, *ESL Design and Verification: A Prescription for Electronic System Level Methodology*, Morgan Kaufmann/Elsevier, 2007.

[2] OSCI, "SystemC," 2011, Available at http://www.systemc.org.

[3] IEEE Std. 1666, *IEEE Standard SystemC Language Reference Manual*, 2005.

[4] D. Große and R. Drechsler, *Quality-Driven SystemC Design*, Springer, 2010.

[5] L. Cai and D. Gajski, "Transaction level modeling: an overview," in *IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, 2003, pp. 19–24.

[6] J. Aynsley, *OSCI TLM-2.0 LANGUAGE REFERENCE MANUAL*, Open SystemC Initiative (OSCI), 2009.

[7] D. P. Siewiorek and R. S. Swarz, *Reliable computer systems - design and evaluation (3. ed.)*, A K Peters, 1998.

[8] J. Gaisler, "A portable and fault-tolerant microprocessor based on the SPARC V8 architecture," in *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, 2002, pp. 409–415.

[9] C. McNairy and R. Bhatia, "Montecito: A dual-core, dual-thread itanium processor," *IEEE Micro*, vol. 25, pp. 10–20, March 2005.

[10] A. Israr and S. A. Huss, "Specification and design considerations for reliable embedded systems," in *Design, Automation and Test in Europe*, 2008, pp. 1111–1116.

[11] R. J. Martínez, P. J. Gil, G. Martín, C. Pérez, and J.J. Serrano, "Experimental validation of high-speed fault-tolerant systems using physical fault injection," in *Proceedings of the conference on Dependable Computing for Critical Applications*, 1999, pp. 249–265.

[12] C. Constantinescu, "Experimental evaluation of error-detection mechanisms," *Reliability, IEEE Transactions on*, vol. 52, no. 1, pp. 53 – 57, march 2003.

[13] A. Pellegrini, K. Constantinides, D. Zhang, S. Sudhakar, V. Bertacco, and T. M. Austin, "Crashtest: A fast high-fidelity FPGA-based resiliency analysis framework," in *Int'l Conf. on Comp. Design*, 2008, pp. 363–370.

[14] G. Fey, A. Sülflow, and R. Drechsler, "Computing bounds for fault tolerance using formal techniques," in *Design Automation Conf.*, 2009, pp. 190–195.

[15] M. Glaß, M. Lukasiewycz, C. Haubelt, and J. Teich, "Towards scalable system-level reliability analysis," in *Design Automation Conf.*, 2010, pp. 234–239.

[16] N. Bombieri, F. Fummi, and G. Pravadelli, "A mutation model for the SystemC TLM 2.0 communication interfaces," in *Design, Automation and Test in Europe*, 2008, pp. 396–401.

[17] N. Bombieri, F. Fummi, G. Pravadelli, M. Hampton, and F. Letombe, "Functional qualification of tlm verification," in *Design, Automation and Test in Europe*, 2009, pp. 190–195.

[18] A. Sen and M. S. Abadir, "Coverage metrics for verification of concurrent SystemC designs using mutation testing," in *IEEE International High Level Design Validation and Test Workshop*, 2010, pp. 75 –81.

[19] P. Lisherness and K.-T. Cheng, "SCEMIT: a SystemC error and mutation injection tool," in *Design Automation Conf.*, 2010, pp. 228–233.

[20] A. da Silva and S. Sanchez, "LEON3 ViP: A virtual platform with fault injection capabilities," in *Euromicro Conference on Digital System Design*, 2010, pp. 813 – 816.

[21] G. Beltra, C. Bolchini, and A. Miele, "Multi-level fault modeling for transaction-level specifications," in *ACM Great Lakes Symposium on VLSI*, 2009, pp. 87–92.

[22] M. A. Kochte, C. G. Zoellin, R. Baranowski, M. E. Imhof, H.-J. Wunderlich, N. Hatami, S. Di Carlo, and P. Prinetto, "Efficient simulation of structural faults for the reliability evaluation at system-level," in *Asian Test Symp.*, 2010, pp. 3–8.

[23] "Playstation™ move motion controller," http://uk.playstation.com/psmove.

[24] Jeremy Bennett, *Or1ksim User Guide*, 2010.

[25] D.H. Ballard, "Generalizing the hough transform to detect arbitrary shapes," *Pattern Recognition*, vol. 13, no. 2, pp. 111 – 122, 1981.

[26] D. Große, H. Peraza, W. Klingauf, and R. Drechsler, "Measuring the quality of a SystemC testbench by using code coverage techniques," in *Forum on specification and Design Languages*, 2007, pp. 146–151.