Fault Tolerant Distributed Embedded Architecture and Verification

Chandrasekaran Subramaniam
Research and Development
Rajalakshmi Engineering College/ AUT
Chennai, India
chandrasekaran_s@msn .com

Prasanna Vetrivel, Srinath Badri
Electrical and Electronics Engineering
Easwari Engineering College/ AUT
Chennai, India
prasannavetrivel1990@gmail.com,
srinath.badri@live.com

Sriram Badri
Electronics and Communication Engineering
Sri Venkateswara College of Engineering/ AUT
Chennai, India
b-sriram@hotmail.com

***Abstract*— The objective of the work is to propose a distributed embedded architecture model for tolerating faults while performing security functions using multiple field programmable gate arrays (FPGA). The hardware encryption and decryption modules are used as customized modules within the devices to act as a cooperative system to tolerate omission and commission faults. The different security functions are communicating through a common UART channel and security operations are synchronized with standard protocols initiated by an embedded micro controller. The decision in locating the working and available modules among a pool of devices is carried out by the microcontroller using an intelligent F-map mechanism and directs the control instructions. The model is scalable with increased number of similar devices when connected across the common communication channel. The model is verified for all its paths using Symbolic Model Verifier, NuSMV to assert the dynamic behavior of the architecture in case of different faulty conditions.**

Keywords- Distributed architecture; Fault tolerance; Security module; Model verification; Assertion technique.

I. INTRODUCTION

The security architecture of embedded systems depends not only on the functional and performance requirements but also on the cost and spatial requirements suited to the target platforms. For example, the data path should be secured against many privacy attacks in the case of embedded systems used in the mobile applications. The data flow based on the dependence graph is solely determined by the components embedded as intellectual properties to perform the expected computations in real time. The architecture suitable for such computations with the security primitive components should be formally verified in order to avoid security errors like communication and synchronization errors. Due to the heterogeneity of various security hardware components from different component vendors, integration of them may lead to further challenges in the architectural design. The earlier SAFES architecture [1] focuses on the reconfigurable hardware that monitors the system behavior to realize the intrusion detections. The other proposed SANES architecture [2] focuses on security controller and component controller components to monitor the abnormal behavior in the system run time. Irrespective of the cryptographic algorithm used in the security primitive, the primitive components are to be self tested since the data path may vary dynamically in the case of a distributed embedded system. The components may not be available at some point of time when they are in need and they should be available in a fault free condition within the distributed mesh of devices. The correct selection of the primitive components either in the source or in the destination FPGA is to be decided in a power efficient manner among a pool of similar devices. The security processes are to be completed in the correct sequence and the operations are to be enabled as in the same dataflow form when they are completed [3]. The components are treated as resources within a single device to complete the submitted task and other similar devices are considered as coordinating devices controlled by a centralized controller. A field mapping technique is proposed through which the resource components available in the devices will be connected to form a distributed system considering the power consumption and the propagation delay involved in the on demand architecture. The distributed security architecture model has to be formally verified for its behavior to meet the reachability and fail free conditions. The standard NuSMV tool [5] supports LTL model checking [5] where the individual parameters can be inspected to investigate the effect of choices. Existing embedded system architectures are not capable of keeping up with the computational demands of security processing, due to increasing data rates and complexity of security protocols [7]. High assurance cryptographic applications require a design to be portioned and physically independent to ensure information cannot leak between secure design partitions. Ensuring this partition separation in the event of independent hardware faults is one of the principle tenets of a high-assurance design [8]. FPGAs are highly promising devices for implementing private-key cryptographic algorithms. Compared to software based solutions FPGA [9] based implementations can achieve superior performance and security. Hence the main focus of the work is to propose a fault tolerant distributed architecture model for security hardware and check the model for its expected behavior meeting the specifications or against it.

The organization of the paper is as follows: Section 2 proposes the distributed embedded architecture for the security hardware using a single reconfigurable FPGA device with all the needed security primitive components. Section 3 discusses the sequences of processes to meet the performance requirements of the security architecture. The next section 4 extends the single system on chip to multi FPGA model where the data packet is routed between four similar devices under different faulty situations that lead to the worst case performance of the model. The next section explains the role of the micro controller in managing the field mapping of the devices when needed security components are faulty. Section 7 illustrates the model checking using NuSMV and its verified output and explores the scalability of the architecture along with its limitations like the issue of packet conflicts along the communication path in UART.

II. DISTRIBUTED EMBEDDED SYSTEM ARCHITECTURE

The architecture selection depends on the resource availability and reliability requirements of the security system. A reconfigurable platform like FPGA in addition to a high speed micro controller can organize the different computations needed to control the sequence of operations in the system. The resources are available in the form of intellectual properties (IPs) within the chip and these are to be assigned with the tasks in an efficient manner by the micro controller as the central manager. The best suitable architecture is the distributed embedded architecture where the resources are different sets of workers and the operations are performed in data flow form. The basic security hardware architecture consists of an encryption and decryption modules for 64 bit size along with their corresponding activation switches. The data transmission and reception takes place through the transmitter and receiver modules placed in the same chip and configured for different baud rates and actual process takes places through a universal asynchronous receiver transmitter (UART) module. The configuration of the security modules are self tested by the wired built-in self test (BIST) components connected with them to tolerate functional faults and controlled by a BIST Controller to regulate the data flow. All the components are selected or enabled by the control signals from device selector (DEVICE SEL) as shown in Fig. 1. The device reads the plain text from the keyboard and routes through the encryption and decryption modules as per the instructions received the micro controller that may reside inside or outside the chip assembly and connected through a bus. The deciphered text will be displayed in the activated display unit of the selected chip.

The micro controller is responsible for maintaining the queue of tasks and allocates the tasks to computational module as and when they are fail free. Because of the different execution times needed for different resources for processing the tasks, the micro controller has to run an intelligence algorithm to decide the available resource for that instant of time to forward the data. The micro controller gets the updated components information by regular sample intervals and refreshes its resource status (RST) and device status table (DST) in the private registers. The control signals are issued by the device select component as and when needed to send or receive. In case where there are concurrent requests from multiple devices over the limited bandwidth channel, then the micro controller forms a priority table to decide the order in which the queue of tasks may be completed. The micro controller is embedded with the algorithm to assign the priority to various requests.

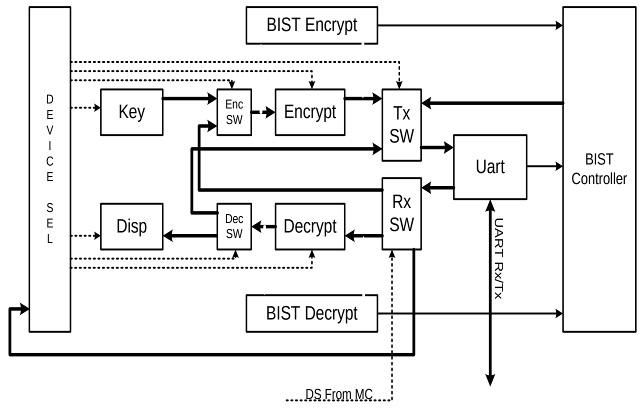

Figure 1. Distributed Embedded Architecture on Chip.

III. SEQUENCE OF PROCESSES

The security components needed to perform cryptographic operations are to be initialized with requests from the central manager component that is the micro controller. The sequence of processes is determined based on the availability of those components in their fault free conditions so that the data flow can be triggered. The enabling and disabling of the components in the pool of resources is taken care by the intelligent algorithm called F-Map embedded in the micro controller. For encryption of the text that is coming through an input peripheral say, keyboard connected to any one of the FPGA device, the sequence of processes is different from that of the decryption processes towards the output peripheral say, display device connected to other or the same FPGA device. For any FPGA to work it is mandatory that the Transmission, Reception switches & the UART connection to function properly. The microcontroller maintains two table a Device Status Table (DST), where a Farm of FPGAs that can take part in the security process are listed and Resource Status Table (RST), where the resource status of each FPGA in the device farm is listed. During runtime micro controller forms a Run Time Table in which all the working set of FPGAs, obtained from DST with all its resources in fault-free condition (resource status obtained from the RST) for the current operation i.e. encryption or decryption, are enlisted depending on power dissipation (Power Aware mode) or propagation delay (Performance mode) of the FPGAs.

A. *Sequence of Operation*

- Check for updates in the run-time table_encryption
- Get data from the SOURCE_FPGA.
- Check the status of resources in the SOURCE_FPGA.
- If Fault free, send data to FPGA for encryption.
- Else, select next FPGA based upon the mode of operation in the runtime table to encrypt the data.
- Check updates in run-time table_decryption.
- Check the status of resources in the DESTINATION_FPGA
- If Fault free, send encrypted data to DESTINATION_FPGA for decryption.
- Else, select next FPGA in the runtime table and instructs to decrypt
- Transmit data to destination.
- Wait and Go to 1.

IV. MULTI FPGA BASED FAULT TOLERANT DISTRIBUTED EMBEDDED SYSTEM

The distributed embedded architecture on chip proposed in the work can be integrated with similar devices so as to make a distributed fault tolerant model. A micro controller is connected to instruct and manage the resource allocation between the devices based on the fault free conditions of the needed components. The algorithm that is embedded in the controller accepts the status of all the devices in terms of their resource components and decides the routing that the plain or cipher text has to follow as shown in Fig 2. The reliability of the distributed system is enhanced by component dynamic redundancy technique as and when the fault gets detected. Even though the functional component is static within a device, the controller calls the fault free components dynamically. The availability and system reliability is enhanced at the cost of communication overhead between the controller and the status table register multiple times. Since the performance depends on the speed of completion of the submitted task, the reliability of the system gets improved over the expected performance level.

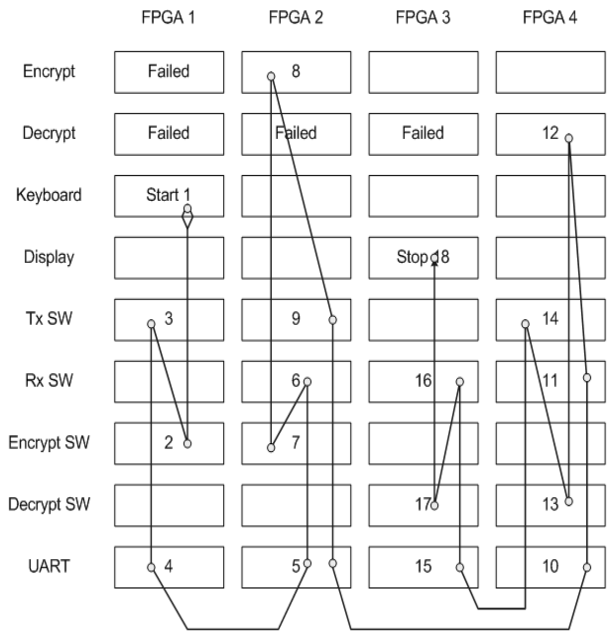

Figure 2. Multi FPGA based Distributed Embedded System.

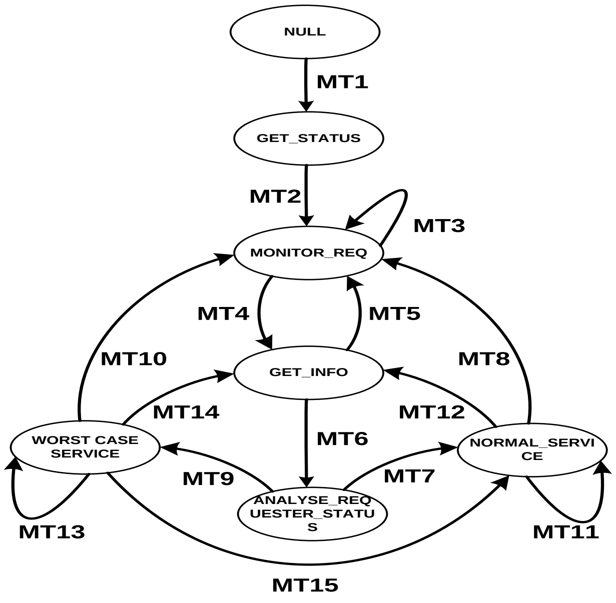

Figure 3. Microcontroller System Level Behaviour.

In the distributed architecture, the microcontroller behavior starts from instructing the distributed architectutre of FPGAs in its Null state. After a time interval (when an implicit or watchdog timer expires) its state transits from Null to Get_Status state (MT1). The state changes from Get_Status to Monitor_Req when the microcontroller successfully reads the status of all the FPGAs (MT2). The coordinating micro controller continues to remain in

Monitor_Req state until any request is made (MT3). The state transits from Monitor_Req to Get_Info if a request is made by any one of the FPGAs (MT4). On occurance of any data error during the request, the controller tracks back from Get_Info to Monitor_Req (MT5). On successfully aquiring the information from FPGAs, a state transition occurs from Get_Info to Analyse_Requester_Status (MT6). If all the resource primitive components at the source and destination FPGAs are fault-free, a state changes from Analyse_Requester_Status to Normal_Service (MT7). On successful completion of the task state transits from Normal_Service state to Monitor_Req state (MT8). If source or destination devices or both have non availability errors in any of their resources, the controller changes state from Analyse_Requester_Status to Worst_Case_Service (MT9). On successful completion of the event, the state changes from Worst_Case_Service state to Monitor_Req (MT10). The control remains in the Normal_Service state until the predefined time elapses on occurance of a communication error (MT11). The microcontroller changes state from Normal_Service to Get_Info state if the error continues to persist at the end of the predefined time interval (MT12). While encountering various errors, the control remains in the Worst_Case_Service state until the predefined time elapses (MT13). The control changes Worst_Case_Service to Get_Info state if the error exists when the predefined time is exhausted (MT14). The control of operations shift from Worst_Case_Service to Normal_Service when the micro controller finds the updated status of the resouce primitive components at the source and destination FPGAs to be fault-free (MT15) as in Fig 3. The fault tolerant technique using available fault free components can be extended to the situation when multiple FPGA devices are interconnected. Assuming similar devices, the security operations are executed in different devices based on the primitive components availability. If not, the correct routing instruction will be issued by the micro controller based on its updated device table and corresponding resource tables. The reliability of the communication path between the devices is a major challenge in the design of the above multi FPGA based distributed embedded security system.The synchronization in the execution of primitive operations is taken care by the embedded algorithm residing in the micro controller.

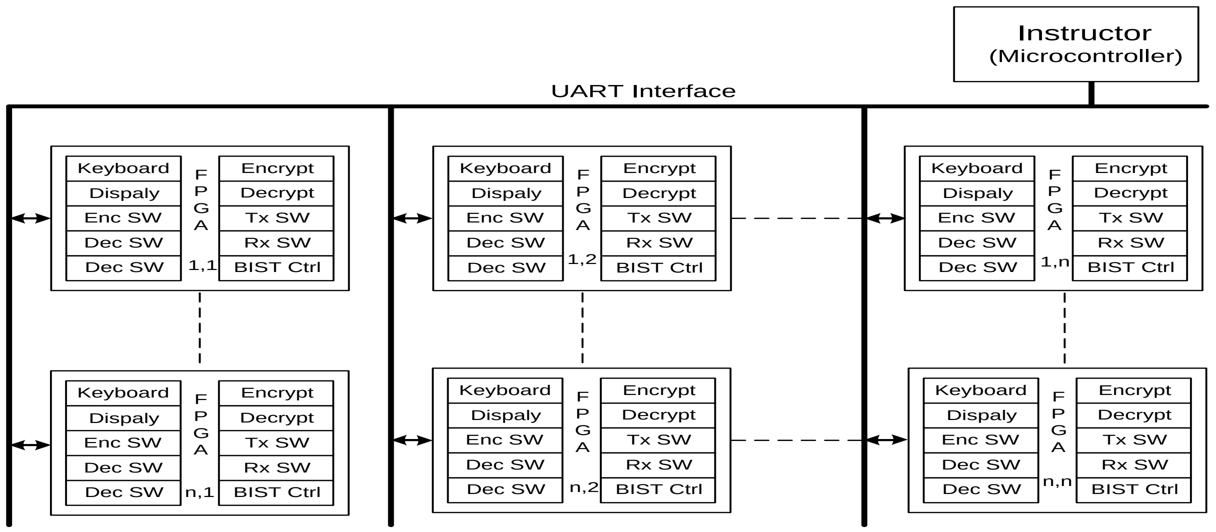

Figure 4. F Map for Power Aware Application.

V. F MAP FOR RESOURCE SELECTION

The Farm map (F-Map) is an intelligent algorithm by which the micro controller identifies the next device that has the needed security primitive components. The mapping brings the current status of the resources and fills the cell as ‘1’ in the *START FPGA* which indicates that the *DEVICE SELECT* and the *BIST for Encryption* components are in failed states as shown in Fig 5. The algorithm searches the next device, depending upon whether the FPGAs are operated in power save mode or performance mode and in which all the needed components are fault free that is indicated by 1 cell for encryption process. Similarly the same algorithm is iterated for the decryption process also to get the data packet encrypted and communicated to the destination. The decryption may take place in the destination FPGA device. If multiple ‘1’ cells are appearing at any time among different devices, then the controller selects the shortest path to minimize the communication delay to avoid any attack on the device itself that is possible in the distributed FPGA architecture. The *Device Select* at the destination FPGA is in failed state which makes the micro controller iterate for the next FPGA with the required resources in Fault-free state. The alternate FPGA for decryption process is selected depending on whether the distributed FPGA architecture is operated in performance mode or power aware mode. ACTIVE_DECRYPT_FPGA1 is chosen when operating in performance aware mode and

ACTIVE_DECRYPT_FPGA2 is chosen while operating in power aware mode. The performance characteristics of the distributed FPGA vary with the operating mode. For quicker response, the Distributed architecture may be operated in performance mode which causes higher power dissipation in the device. While operating in power save mode, the propagation delay is high, but the power consumed is minimal, as shown in Table. 1.

Propagation Delay = n * T (1)

Where, n= R + C - 1

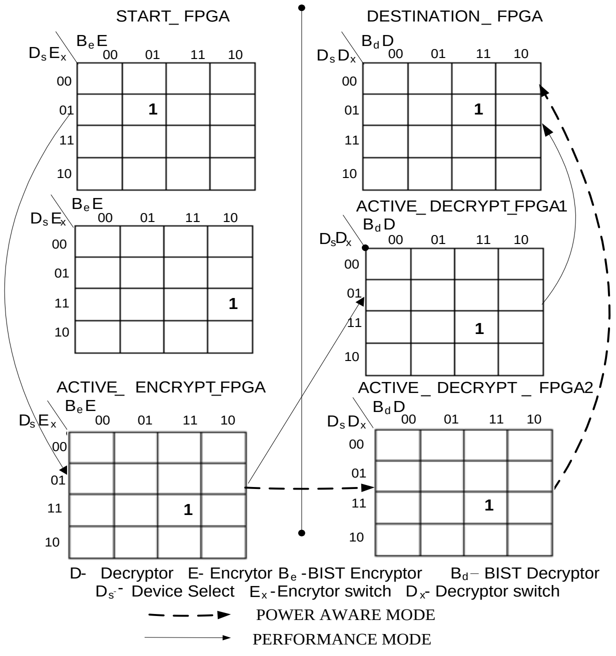

Figure 5. F Map for Power Aware Application.

VI. Performance in Best Case and Worst Case Scenario

In the distributed embedded system, if all the resources in the source and destination FPGAs are fault-free then the micro controller chooses the normal operating mode which is the best case scenario. The behavior of the embedded system during its best case performance is shown in Fig. 6 and its transitions are mentioned in Table. 2. The behavior of the distributed embedded system varies when a resource in source or destination FPGA is in failed state, depending upon the mode of operation viz. Performance Mode, Power Aware Mode, Safe Mode, depicted by the worst case behavior is shown in Fig. 7.

The microcontroller iterates various paths from the source to the destination depending upon the mode of operation and selects the FPGAs in source or destination side, with its required resources in fault free condition.

'R' Denotes the Row & 'C' Denotes Column in which the FPGA is located in the DST.

TABLE I. Power Dissipation And Propogation Delay for Various Modes

Order in DST (R,C)	Power Dissipation	Propagation Delay(ms)	Remarks
1,1	6.41W	T	Performance
2,3	4.23W	4*T	Normal/Safe
1,9	3.47W	9*T	Power Aware
6,12	7.9W	17*T	Poor

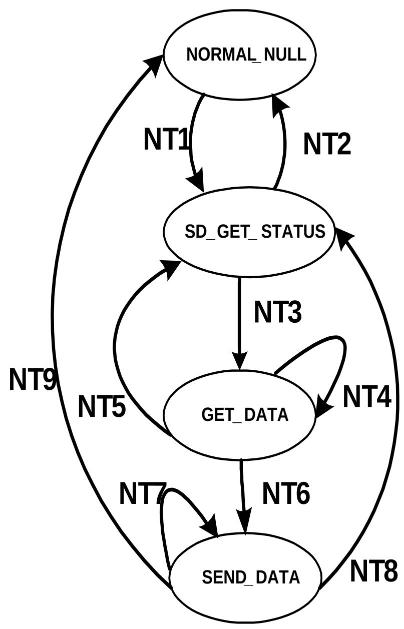

Figure 6. Best Case Scenario.

TABLE II. Best Case Transitions

Transition ID	Events
NT1	If main state machine reaches NORMAL_SERVICE state
NT2	Failure of getting source status || Failure of getting destination status
NT3	After getting successful status of source and destination
NT4	If not getting correct data from source
NT5	If not getting correct data from source && reaches maximum tries
NT6	Getting successful data from source
NT7	Not sending correct data to destination
NT8	If not sending correct data to destination
NT9	If sending correct data to destination

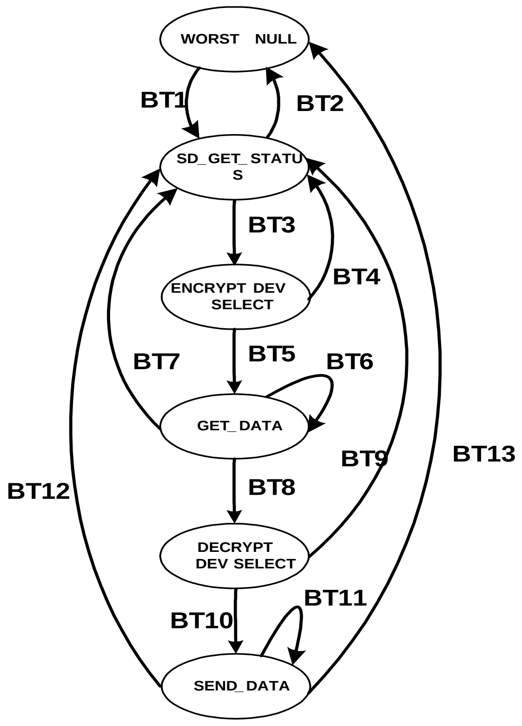

Figure 7. Worst Case Scenario.

While the distributed Embedded system is in its Worst Case behavior, the microcontroller acquires the status of all the FPGAs present in the distributed embedded system (BT1). In case of failure of acquisition of the status, the microcontroller waits in a null state until the status of the FPGAs are updated (BT2). Depending upon the mode of operation during resource failure in the source FPGA, the microcontroller searches for an alternate FPGA for encryption process(BT3). On selection of a suitable FPGA, the data from the source FPGA is transferred for encryption process (BT5). The microcontroller continues to exist in Get_Data state until the data is completely transferred to the alternate FPGA (BT6). On occurance of any errors during the data acquisition process, the microcontroller tracks back to its initial state (BT7). On completion of encryption process, a FPGA is selected by the microcontroller for decryption on the destination side (BT8). On successful selction of FPGA with all its resources in fault free condition, process of decryption takes place (BT10). The microcontroller traces itself to its initial state on failure of the decryption process (BT9). The decrypted data is ultimately sent to the destination FPGA until which the microcontroller remains in its current state (BT11). On successful reception of the complete decrypted data by the destination FPGA, the microcontroller prepares itself for the next encryption process (BT13), else the microcontroller resets to repeat the failed process (BT12).

VII. Model Checking of Architecture using NuSMV

There are two extreme cases in which the fault tolerant capability can be measured. In one case, all the needed security components are fail free and available in a single chip and thereby the communication delay will be only due to transmit and receive signal propagation making power consumption very high. The other extreme is when one set of encryption components may be available in nearby device where as the decryption set of components in other end of the array or pool of devices and vice versa [6]. This condition leads to huge delay due to multiple enable, multiple address and data signals along with multiple receive, transmit and multiple device select signals. The worst case performance where the micro controller searches for the fail free encryption and decryption components is verified using the symbolic model verifier as shown below. The model checking is done to check the reliability of the model during the worst case performance and to verify the availability of alternate resources to complete the specified task. The LTL and CTL operations are applied as specifications to verify the availability of devices and the needed security resources for the encryption and decryption processes as given in NuSMV code below:

NuSMV Code For availability in the worst case Scenario

```
MODULE main
VAR
state : { null_state, get_update_status, s_device_select,
get_data, d_device_select, send_data };
status: boolean;
source_device_select : boolean;
data_get : boolean;
data_sent : boolean;
time : boolean;
dest_device_select : boolean;
INIT
state=null_state;
ASSIGN
next(state):= case
state = null_state : get_update_status ;
state = get_update_status & !status : null_state;
state = get_update_status & status : s_device_select;
state = s_device_select & !source_device_select :
get_update_status; state = s_device_select &
source_device_select : get_data;
state = get_data & !data_get & time : get_data;
state = get_data & !data_get & !time : get_update_status;
state = get_data & data_get : d_device_select;
state = d_device_select & !dest_device_select :
get_update_status;
state = d_device_select & dest_device_select : send_data;
state = send_data & !data_sent & time : send_data;
state = send_data & !data_sent & !time : get_update_status;
state = send_data & data_sent : null_state;
esac;
SPEC
EF (state=null_state)
SPEC
AG AF( !time -> state= get_update_status)
SPEC
AG EF(!status -> state=null_state)
SPEC
AG AG AF( time -> !data_get ->state= get_update_status)
```

The model checking algorithm reports true if specification holds true for every state of the system model. Otherwise, states not satisfying the specification are identified. A transition path from a defined initial state to a state identified as not satisfying the specification is called counterexample [4]. The CTL properties of the proposed security farm architecture is verified using NuSMV for different specifications and the result with instances and counter example is shown below:

SPECIFICATUIONS FOR THE NUSMV CODE USING LTL AND CTL PROPERTIES

```
-- specification EF state = null_state is true
-- specification AG (AF (!time -> state = get_update_status)) is true
-- specification AG (EF (!status -> state=null_state)) is true
-- specification AG (AG (AF (time -> (!data_get -> state = get_update_status)))) is false
-- as demonstrated by the following sequence
Trace DEscription: CTL Conterexample
Trace Type: Counterexample
-> State: 1.1 <-
 state = null_state
 status = FALSE
 source_device_select = FALSE
 data_ get = FALSE
data_sent = FALSE
 time = FALSE
 dest_device_select + FALSE
-> State: 1.2 <-
 state = get_update_status
 status = TRUE
-> State: 1.3 <-
 state = s_device_select
 status = FALSE
 source_device_select = FALSE
 time = TRUE
-- Loop starts here
-> State: 1.4 <-
 status=get_data
 source_device_select = FALSE
-> State: 1.5 <-
```

VIII. CONCLUSION AND FURUTE WORKS

A distributed embedded architecture model for security hardware with fault tolerant features is proposed to tolerate non availability faults and resource component faults. The model is a centralized system in which sequential control is exercised by a micro controller through the F-map algorithm embedded in it and can be scaled as multi FPGA devices pool with power or performance awareness. The collective behavior of the architecture model is formally verified using a model checker based on LTL and CTL properties. The limitation of the proposed model is in the communication overhead when all devices want to send the data concurrently over the limited bandwidth of UART channel. The constraints are the assumptions that the basic transmission and reception switches must always be fault free even to intimate the error report to the central controller. The actual implementation of the project with multiple FPGAs and triple micro controllers to enhance the reliability is the future work planned.

REFERENCES

[1] G. Gogniat, T. Wolf and W. Burleson: *Reconfigurable Hardware for High-Security / High-Performance Embedded Systems: The SAFES Perspective*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 16, No. 2, February 2008, pp. 1-10.

[2] *Reconfigurable Security Architecture for Embedded Systems*, http://vcsgi.ecs.umass.edu/essg/papers/ MOCHASubmit.pdf, pp. 1-7.

[3] S. Hauck and A. Dehon: Morgan Kaufmann Publications, *Reconfigurable Computing*, pp 107-110.

[4] Sebastian Steinhorst: Dissertation, *Formal Verification Methodologies for Nonlinear Analog Circuits*, Frankfurt, 2011, pp. 55-69.

[5] A.Cimatti, E.Clarke, F.Giunchiglia and M.Roveri: *NuSMV :A New Symbolic Model Verifier*,pp.1-5.

[6] G. K. Palshikar: *An Introduction to Model Checking*, html page, pp. 1-8.

[7] S. Ravi, A. Raghunathan, P. Kocher and S. Hattangady: *Security in Embedded Systems: Design Challenges*, ACM Transactions on Embedded Computing Systems, Vol 3, No. 3, August 2004, pp. 6-10.

[8] P. Quintana: *Fail-Safe FPGA Design Features for High-Reliability Systems*, Paper ID: 900566, IEEE 2009, pp. 3-5.

[9] A. Dandalis, V.K. Prasanna and D.P. Rolim: An Adaptive Cryptography Engine for IPSec Architectures, ACM Transactions on Design of Automation of Electronic Systems, Vol. 9, July 2004, pp. 333-353.