A Case Study in Modeling a Fault-tolerant Satellite System through Implementation of Dynamic Reconfiguration via Handshake

Kashif Javed
Turku Centre for Computer Science (TUCS)
Department of Information Technologies
Abo Akademi University
Turku, FIN-20520, Finland
Kashif.Javed@abo.fi

Elena Troubitsyna
Department of Information Technologies
Abo Akademi University
Turku, FIN-20520, Finland
Elena.Troubitsyna@abo.fi

***Abstract*— Fault tolerance of satellite systems is critical for ensuring the success of the space mission. To minimize redundancy of the on-board equipment, the satellite systems should rely on dynamic reconfiguration in case of failures of some of their components. In this paper, modeling and implementation of a handshake procedure has been presented that becomes a crucial part of the dynamic reconfiguration process of a satellite subsystem for data processing. The model for handshake methodology is specialized software for quickly and successfully recovering from the crisis and failure situation of the satellite system.**

Keywords – dynamic reconfiguration; fault tolerance; advanced software for handshake procedure; modeling and verification.

I. INTRODUCTION

To ensure high reliability during long-term missions, the satellite systems rely on redundancy to achieve fault tolerance and guarantee that the system would be able to deliver its services despite component failures. However, the use of redundancy in the satellites is restricted by the constraints put on the weight and volume of the on-board equipment.

Despite a careful analysis performed to ensure the desired degree of reliability, recently one of the satellites has experienced a double-failure problem with a system that samples and packages scientific data [6]. The system consisted of two identical modules. When one of the subcomponents of the first module failed, the system switched to the use of the second module. However, after a while a subcomponent of the spare module also failed, so it became impossible to produce scientific data. In order to avoid failure of the entire mission, the company controlling the operation of the system has invented a solution that relies on healthy subcomponents of both modules and provides complex communication mechanism based on the handshake procedure to restore functioning and to resume production of scientific data.

In this paper, we present a case study in modeling and implementation of Control and Data Management Unit (CDMU) [1] - a generic subsystem of satellites. In particular, we focus on modeling fault tolerance aspect of the system that is implemented as a handshake procedure between two redundant systems. This mechanism is introduced to achieve the dynamic reconfiguration. For this purpose, a formal model of the handshake procedure has been designed and implemented in Promela. Handshake modeling is an advanced software application to deal with dynamic reconfiguration for ensuring fault-tolerance when the mission-critical satellite system encounters faults in its component and errors in data communication.

This paper is structured as follows. Section II describes the state-of-the-art model of CDMU and Section III presents the architecture of the control and data management unit. Section IV describes the handshake procedure performed to reconfigure the system from simple redundant two-module architecture to the Master-Slave architecture. The proposed system model for handshake is explained in Section V covering all relevant details of master and slave modules. Section VI discusses the handshake model between the two reconfiguration modules that has been implemented and verified using SPIN/PROMELA. Finally, conclusions and future work are summarized in Section VII.

II. STATE-OF-THE-ART MODEL

CDMU is a state-of-the-art platform to monitor and control the satellites system and to organize the collected on-board data. The major objective of CDMU is to acquire and transmit the data to the ground after carrying out appropriate processing. Moreover, it also distributes and decodes the given commands to its all redundant systems consisting of processor, reconfiguration and telemetry modules. Whenever any failure or data error takes place during the operation of the satellite system, there is an emergent requirement to dynamically reconfigure the components of CDMU for its smooth and crisis-free control and data management. Processing and storing of satellite data at the right time is of top-most importance during the working and recovery procedure of the proposed system. In case of experiencing any failure, the implemented CDMU structure and the developed model of handshake procedure immediately adapts to the well-defined and specialized switchover mechanism for shifting from one redundant processor to another in order to reconfigure and provide safe operation of the satellite system during its critical mission.

 ISBN: 978-1-61208-230-1

III. ARCHITECTURE

The CDMU consists of two Processor Modules (PM1 and PM2), two Reconfiguration Modules (RM1 and RM2), and two Telemetry Modules (TMM1 and TMM2). It their own turns, each PM consists of Random Access Memory (RAM), Integer Unit (IU), Floating Point Unit (FPU), and Erasable Electrically Programmable Memory (EEPROM). Each Reconfiguration Module (RM) has two components -- Mass Memory (MM) and On-Board Reference Time (OBRT). Telemetry Modules generate Telemetries (TMs) that are processed by Processor Modules.

In CDMU, only one Processor Module (PM1 or PM2) is in active mode and can access one or both RM1 and RM2. TMs are received by the active processor module and accumulated only in MM of its local RM. However, TMs can be retrieved from the MM of partner RM after switching is done from one processor module to another. When each particular PM has experienced a failure, the Master and Slave policy is introduced for error recovery. It aims at ensuring that the CDMU functionality can be preserved even when failures are present in the system.

In our case study, we consider the following two consecutive errors in CDMU that might occur during the execution of the system:

1) PM1 fails due to the failure in FPU.
2) TM ceases to function due to the failure in the link between TMM2 and PM2.

The basis of the Master and the Slave is to prepare a work-around in order to address above mentioned failures. In this case, PM1 and PM2 are converted into the Slave and the Master respectively. Similarly, Master and Slave comprise of the functional program running in PM2 and PM1 respectively and it is mainly established to execute the system without the FPU and connection link.

At a time, both the Master and the Slave interface with RM1 and RM2, respectively, as shown in the CDMU structure. However, RM1 and RM2 are not capable to hold simultaneous access to both of them.

Despite the error in the connection link of PM2, the PM2 is still in operational mode and stores TM in the MM. Similarly, PM1 is also in operational mode by using only IU program (without FPU) that recovers TM from the MM and sends to the operator. The operator interacts with the Master and the Slave by sending Tele-Commands (TCs). Figure 1 shows that each processor module is connected to both RM1 and RM2 and to both TMM1 and TMM2. The TeleCommand (TC) receiver is also linked to both PM1 and PM2.

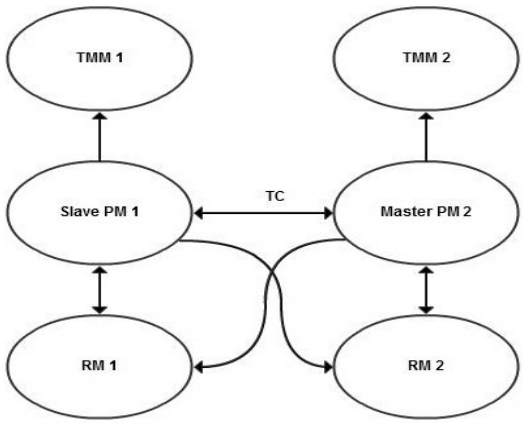

Figure 1: CDMU Structure [1]

IV. FACTORS CONTRIBUTING IN HANDSHAKE

The important key factors that are involved in the handshake procedure are as follows:

1) Time Event Register (TER) is used for messaging between the Master and the Slave. As there is no direct link between the Master and the Slave, so TER is used as a shared device. Both can access TER to read and write messages. RM1 and RM2 have their own TER devices.
2) The two interrupts -- Time Event Interrupt (TEI) and Time Synchronization Interrupt (TSI) caused by RM1 and RM2 are sent to the Slave and the Master respectively. If the Master uses RM1 and interrupt triggers, then interrupt is only sent to the Slave because it is a local processor module of RM1.
3) The interrupts can be used as a signal from the Master to the Slave for the acknowledgement of the messages because the Master has a charge of the interrupt timing.
4) OBRT Status Register is used to find out that interrupt has triggered in the system. The Master holds the check of this register and clears the interrupt flag for allowing the coming up interrupts.
5) The Master and the Slave cannot use the same RM at a time. However, both the Master and the Slave are informed through handshake procedure in order to choose required RM at a given time interval.
6) Handshaking is done through Communication Channel (CCH) between the Master and the Slave. RM1 or RM2 is used as CCH. The TER in the CCH is expressed as Communication Time Event Register (CTER).
7) The selection of RM1 or RM2 as CCH depends on the Master as it utilizes both RM1 and RM2. On getting the TC instruction from the operator, it

 ISBN: 978-1-61208-230-1

switches to one module of RM (RM1 or RM2) and releases the other RM for CCH. If the Master is using only one RM module initially, the unused RM will be selected as CCH. The Master can switch the RM at the end of the handshake procedure.

8) The handshake message contains the phase content and timing of the message that is encoded in the CTER. The timing of the interrupt is slightly affected by the phase content that is encoded in the four Least Significant Bits (LSB) of the CTER, but this affect of interrupt timing is less than 0.3 ms and is, therefore, ignored.
9) The phase content in the four least significant bits of the CTER is as under:
    i. When 4 LSB of CTER has value '1', then the Master informs the Slave to communicate through RM1. Similarly, when 4 LSB of CTER has value '2', then the Master informs the Slave to communicate through RM2. This phase is known as "Select Communication RM".
    ii. If the value is '4' in the 4 LSB of CTER, the Slave updates the Master to confirm the communication through RM1. Likewise, if the value is '5' in the 4 LSB of CTER, then the Slave informs the Master that it confirms the communication through RM2. This phase of the handshake procedure is called "Confirm Communication RM".
    iii. Upon setting the value of '10' in 4 LSB of CTER, the Slave is informed by the Master that if RM1 is not in use then switch to it and use it. For the value '11', the Slave has to switch to use RM2. When the value is '14', then the Master instructs the Slave to release both RM1 and RM2. This phase is named as "Command Slave".
    iv. The Master sends a message to the Slave in which it verifies the RM1 or RM2 selection by putting the value '8' in 4 LSB of CTER. This phase is entitled as "Confirm Command".

10) The encoding of the handshake messages is done within one second (s) - Pulse Per Second (PPS). The interrupts according to the PPS time slot are given below:
    i. When interrupts occur from 0.10 to 0.40 s, RM1 and RM2 are not selected in this time slot. It means that the Master instructs the Slave to confirm the change to use no RM.
    ii. For the selection of RM1, interrupts take place in the time slot ranging from 0.42 to 0.70 s. The Master orders the Slave either to communicate with RM1 or confirm change to use RM1 during the handshake procedure.
    iii. In the 0.72 - 1.00 s time slot, interrupts are taken into account. This selection is encoded for RM2 where master notifies the Slave either to communicate with RM2 or confirm change to use RM2 during the handshake procedure.
    iv. The purpose of the remaining unused slots 0.00 – 0.10 s, 0.40 – 0.42 s and 0.70 – 0.72 s is to avoid overlaps. Any interrupts appearing in these timing slots will be ignored.

11) The minimum time between two TSIs is greater than 0.3s to ensure that two TSIs do not trigger during the same time slot. On the other hand, interrupt can be triggered two times during the same time slot.

V. PROPOSED SYSTEM MODEL FOR HANDSHAKE

The handshake procedure [2] has been modeled for the Master and the Slave as shown in Figure 2. Handshake is a procedure in which the Master communicates with the Slave to update the selection of RM1 and RM2. It is a complicated process as there is no direct communication link between them.

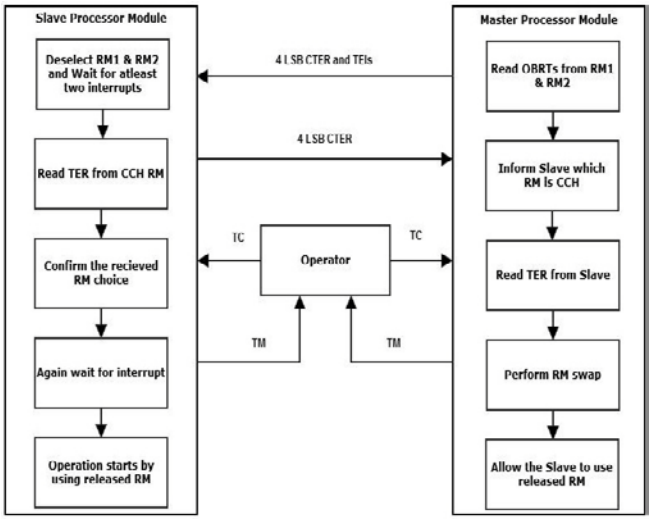

Figure 2: Model of Handshake Procedure

A. *Master Handshake Procedure*

The handshake procedure that is executed by the Master Module is shown in Figure 2. Below we give its brief description:

Upon the reception of TC from the operator, the handshake procedure is started by the Master. The Master

informs the Slave that other RM will be used as CCH by updating the value of 4 LSB TER. If the Master is using RM2 and storing TM, then the Slave will be informed to make RM1 as CCH. Likewise, if RM1 is operated by the Master, then the Slave has to use RM2 as CCH. When CCH is RM1, then system operation is performed from 0.42 to 0.70 s PPS slot. Similarly, for RM2, 0.72 to 1.00 s, PPS slot is used for the system operation. System has to wait for starting of the right PPS slot according to the CCH.

In order to send information to Slave, interrupts are triggered from the Master after setting the value of OBRT Status Register to zero. For accuracy, the value of TER for the Slave RM is set to 0.04 s. The interval between two interrupts is 0.06 s. The Master ensures by reading the CTER value from the Slave that selection of CCH is done. The Master can swap the CCH selection at the end of handshake procedure. The Master commands the Slave by setting the future CCH selection value in the 4 LSB CTER and triggers a TEI only. The time value of TEI is not relevant to the CTER, so the time slot of TEI makes no changes in the end result of the system. Only operator is responsible for the new RM selection and determining which RM is used as CCH as stated in Section IV. In the system, operator initially notifies the RM selection to the Master, it changes CCH selection from used RM to other RM according to the swapping information that is encoded in 4 LSB CTER and also confirms the RM selection. The confirmation message is also forwarded to the Slave by sending two interrupts within the correct time slot. At this moment, the Master ends the handshake procedure and updates the operator for successful working by sending the corresponding TM.

B. Handshake Procedure: Slave Behaviour

When the operator starts the handshake, the following operations are carried out by the Slave as shown in Figure 2.

If the Slave is using RM1 or RM2, then it will deselect the current RM on the reception of TC command from the operator. When RM is discontinued from the Slave, then OBRT Status Register will be set to zero and no more interrupts will be triggered. The Slave waits for 0.03 s to get the new command along with two interrupts (i.e. TEI and TSI) which will be generated from the Master during the expected PPS slot. When the Slave receives a message from the Master, then it decodes it from the interrupts time slot as mentioned in Section IV (para # 10). For verification, the Slave also interprets the value of 4 LSB CTER as described in Section IV (para # 9). If the values derived from the interrupts time slot and 4 LSB CTER are the same, then the Slave achieves the specified CCH selection. After that, the Slave sends acknowledgement of confirmation to the Master by setting the value of 4 LSB CTER according to Section IV. Now, the Slave has to wait again for 0.02 s for the new response or interrupt from the Master according to the PPS slot. On the arrival of message from the Master, the Slave is triggered by TEI. The Slave has no opportunity to change the decision of new selection and waits for 10s for the confirmation message from the Master. Again, the Slave receives two interrupts with the CTER message and compares the time slot of interrupts with previous CTER value. If both are same, then the Slave begins the operation with released RM. Finally, the Slave also completes the handshake procedure by sending TM to the operator.

VI. VERIFICATION OF THE HANDSHAKE MODEL

The handshake model has been implemented by using PROMELA (PROcess MEta LAnguage) high level modeling language with SPIN model checker for verifying the required results. SPIN [3,4] is extensively used in formal verification of distributed and parallel processing systems. SPIN has greatly facilitated the process of verification in the areas of mission-critical algorithmic applications, message and data communication in the client-server environment, synchronization and coordination of large number of processes in the parallel and distributed systems, deadlock handling methodologies in the modern multi-tasking operating systems, verification of the mission-oriented control models for space aircrafts, utilization of intelligent models for determining most suitable and economical paths over wide area networks, checking performance of routing protocols [5], testing of fault-tolerant strategies and implementation of a wide variety of switching techniques. The literature review reveals that most of the software-based systems/models are checked and verified by the SPIN model checker.

The handshake model between two processors in control and data management unit has been successfully implemented and verified using SPIN/PROMELA. The flow chart for handshake procedure model is shown in Figure 3. The following algorithm along with description of each condition of the processes shows part of the implemented SPIN/PROMELA model.

```
/*Variable Declarations */
active proctype Slave_starts_HP()
{S_TC=true;
if
::(S_TC==true)->RM1=0;RM2=0;
::( S_TC!=true)-> printf("\n\nExit Handshake Procedure.\n\n");
fi
S_TM=true;}
```

The above code depicts that when TC command is received to Slave from the operator, Slave starts handshake procedure by deselecting the RM selection. After successful execution of the TC command, Slave sends TM to operator and waits for Master's response. In any other condition, handshake procedure will be terminated.

```
active proctype Master_starts_HP()// time value is taken in (ms)
{M_TC=true; RM1=0;RM2=1; // set by the operator
if
::(RM1==0 && RM2==1)->// I_time denotes timing of interrupts
{CTER_4_LSB=1;I_time=500;TEI=true;TSI=true;OBRT_SR=1;
run Slave_read_wrtie_operation(CTER_4_LSB,I_time,TEI,TSI);}
::(RM1==1 && RM2==0)->
{CTER_4_LSB=2;I_time=800;TEI=true;TSI=true;OBRT_SR=1;
```

```
run Slave_read_wrtie_operation(CTER_4_LSB,I_time,TEI,TSI);}
fi}
```

The code associated with the above process describes that Master starts handshake on the operator command. When operator selects RM2 for Master, then Master uses RM2 and notifies Slave (by sending CTER and interrupts) to use RM1 as CCH. Likewise, if operator selects RM1, then Master uses RM1 and updates the Slave (through CTER and interrupts) to use RM2 as CCH. After that, it waits for Slave's response.

```
proctype Slave_read_wrtie_operation(int CTER_4_LSB,I_time;bool
TEI,TSI)
{if
::((CTER_4_LSB==1) && (TEI==true && TSI==true) && (I_time>=420
&& I_time<=700))->
{CTER_4_LSB=4;run Master_decides_future_selection(CTER_4_LSB);}
::((CTER_4_LSB==2) && (TEI==true && TSI==true) && (I_time>=720
&& I_time<=1000))->
{CTER_4_LSB=5;run Master_decides_future_selection(CTER_4_LSB);}
::((CTER_4_LSB!=1) || !(I_time>=420 && I_time<=700))->
{printf("\n\nExit Handshake Procedure.\n\n");}
::((CTER_4_LSB!=2) || !(I_time>=720 && I_time<=1000))->
{printf("\n\nExit Handshake Procedure.\n\n");}
fi}
```

The above piece of code illustrates that when timing of interrupts is in line with the information that is encoded in CTER 4 LSB, then Slave confirms the selection to Master and waits for 0.02 s in order to get Master's response. So, when interrupts occurs between 0.42 to 0.70 s time slot and CTER 4 LSB is '1', it means Slave confirms to use RM1 as CCH by encoding the value '4' in CTER 4 LSB. Similarly, if time slot for interrupt is 0.72 to 1.00 s and CTER 4 LSB is '2' then RM2 is confirmed as CCH by the Slave through updating the value '5' in CTER 4 LSB. If timing of the interrupts is not compatible with the encoded information in CTER 4 LSB, handshake procedure exits at this stage.

```
proctype Master_decides_future_selection(int CTER_4_LSB)
{if
::(CTER_4_LSB==4)->
{OBRT_SR=0;CTER_4_LSB=11;TEI=true;OBRT_SR=1;
if
::(CTER_4_LSB==11)->
{RM1=1;RM2=0;aa= CTER_4_LSB;OBRT_SR=0;CTER_4_LSB=8;
I_time=800;TEI=true;TSI=true;OBRT_SR=1;M_TM=true;
run Slave_interprets_message(aa,I_time,TEI,TSI);}
::(CTER_4_LSB==14)->
{RM1=0;RM2=0;OBRT_SR=0;aa=CTER_4_LSB;CTER_4_LSB=8;
I_time=200;TEI=true;TSI=true;OBRT_SR = 1;M_TM=true;
run Slave_interprets_message(aa,I_time,TEI,TSI);}
fi;}
```

The above fragment of the code describes that when Slave is using RM1, Master updates the up-coming selection of RM by placing the value '11' or '14' in CTER 4 LSB with only TEI. If Master selects RM1, it releases RM2 to be used as CCH by putting the value '11' in CTER 4 LSB. When Master picks RM1 and does not release RM2 to be used as CCH, it writes the value '14' in CTER 4 LSB. After a half second to give the Slave sufficient time to read value of CTER, the Master confirms the selection to the Slave by encoding the value '8' in CTER 4 LSB on the specified time slot according to Section IV and exits the handshake procedure.

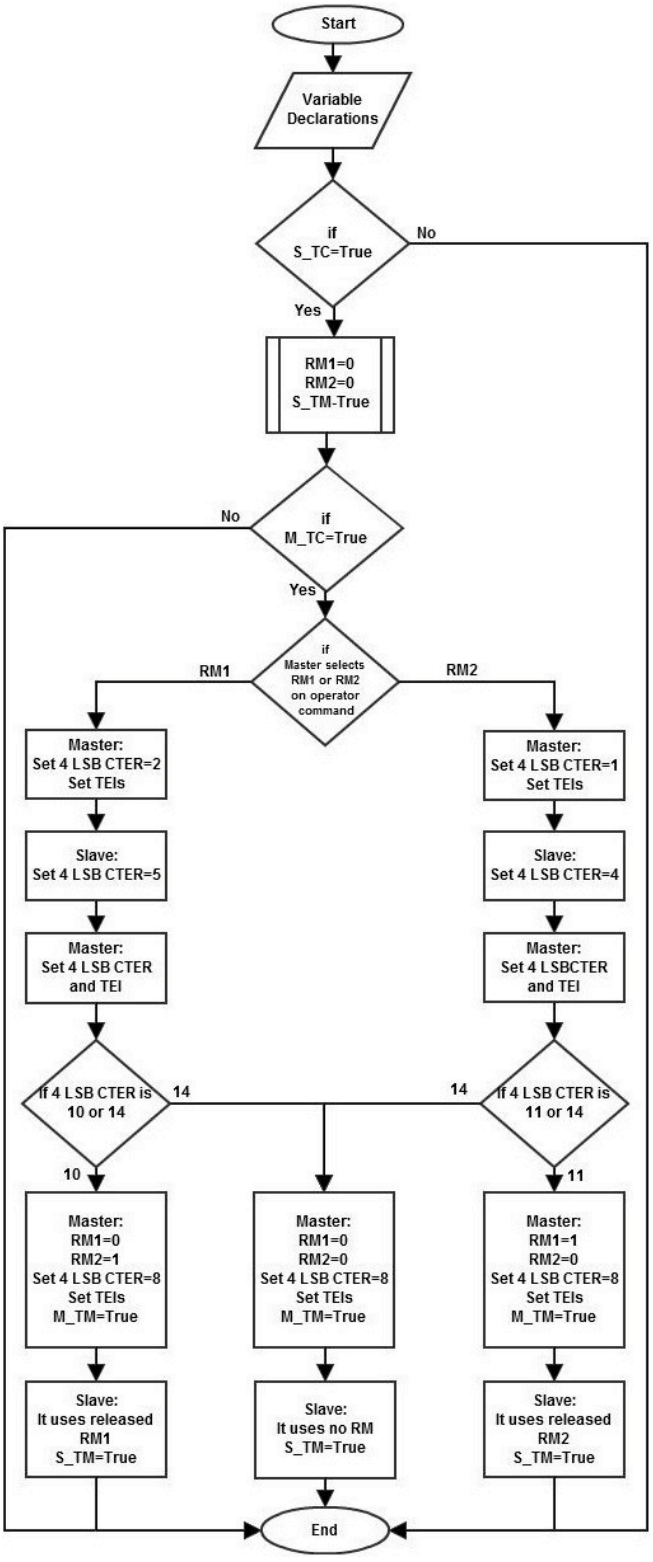

Figure 3: Flow Chart of Handshake Procedure Model

```
::(CTER_4_LSB==5)->
{OBRT_SR=0;CTER_4_LSB=10;TEI=true;OBRT_SR=1;
```

The associated code with above condition illustrates this.

```
if
::(CTER_4_LSB==10)->
{RM1=0;RM2=1;OBRT_SR=0;bb=CTER_4_LSB;CTER_4_LSB=8;
I_time=800;TEI=true;TSI=true;OBRT_SR=1;M_TM=true;
run Slave_interprets_message(bb,I_time,TEI,TSI);}
::(CTER_4_LSB==14)->
{RM1=0;RM2=0;OBRT_SR=0;bb=CTER_4_LSB;CTER_4_LSB=8;
I_time=200;TEI=true;TSI=true;OBRT_SR = 1;M_TM=true;
run Slave_interprets_message(bb,I_time,TEI,TSI);}
fi;}
fi}
```

The above part of the code shows that when the Master is using RM1, it updates the up-coming selection of RM by setting the value '10' or '14' in CTER 4 LSB with only TEI. If the Master selects RM2, it releases RM1 to be used as CCH by putting the value '10' in CTER 4 LSB. When the Master picks RM2 and does not release RM1 to be used as CCH, it writes the value '14' in CTER 4 LSB. After a half second to give the Slave sufficient time to read value of CTER, the Master confirms the selection to the Slave by encoding the value '8' in CTER 4 LSB on the specified time slot according to Section IV and exits the handshake procedure.

```
proctype Slave_interprets_message(int previous_CTER,I_time;bool
TEI,TSI)
{if
::((I_time>=420 && I_time<=700) && (previous_CTER==10) &&
(TEI==true && TSI==true))->
{S_TM=true;}
::((I_time>=720 && I_time<=1000) && (previous_CTER==11) &&
(TEI==true && TSI==true))->
{S_TM=true;}s
::((I_time>=100 && I_time<=400) && (previous_CTER==14) &&
(TEI==true && TSI==true))->
{S_TM=true;}
::(!(I_time>=420 && I_time<=700) || (previous_CTER!=10))->
{ printf("\n\nExit Handshake Procedure.\n\n");}
::(!(I_time>=720 && I_time<=1000) || (previous_CTER!=11))->
{ printf("\n\nExit Handshake Procedure.\n\n");}
::(!(I_time>=100 && I_time<=400) || (previous_CTER!=14))->
{ printf("\n\nExit Handshake Procedure.\n\n");}
fi}
init
{atomic// Atomic is used to reduce the complexity.
{run Slave_starts_HP();
run Master_starts_HP();}
}
```

The code given above indicates that after waiting for 10 s, Slave receives the confirmation message with two interrupts from Master. The timing of interrupts is matched with the information that is encoded in previous CTER 4 LSB as mentioned in Section IV. Therefore, when timing of the interrupts lies between 0.42 to 0.70 s time slot and previous CTER 4 LSB is '10', it notifies that Slave uses RM1 as CCH that is released by the Master. Similarly, timing of the interrupts lies between 0.72 to 1.00 s time slot and previous CTER 4 LSB is '11', it notifies that Slave uses RM2 as CCH that is released by the Master. Also, when interrupts timing lies between 0.10 to 0.40 s and the value of previous CTER 4 LSB is '14', then Slave uses neither RM1 nor RM2 as CCH. After then Slave exits the handshake procedure. If interrupts timing is not in line with the information that is encoded in earlier CTER 4 LSB, handshake procedure exits at this stage too.

VII. Conclusions and Future Work

In this paper, we have proposed a formal approach for modeling a fault-tolerant satellite system that relies on the handshake procedure for dynamic reconfiguration. We have demonstrated how to create a Promela model of the handshake and carry out its analysis. Since the handshake procedure has a number of non-trivial properties caused by the distributed nature of the system, such a model allows the designers to ensure correctness of the handshake implementation. In our future work, we are planning to extend the proposed approach to derive the generic modeling patterns. Moreover, it would be interesting to explore the handshake in the presence of more complex network architecture.

References

[1] "DEPLOY – Software Requirement Specification, Master/Slave Software", Space Systems Finland, Ltd., July 2011.

[2] J. Kashif, and E. Troubitsyna, "Designing a Fault-Tolerant Satellite System in SystemC", ICONS 2012, The Seventh International Conference on Systems, IEEE Computer Press, pp. 49–54, March 2012.

[3] C.Baier and J.-P. Katoen. "Principles of Model Checking". MIT Press, 2008.

[4] N. A. S. A. Larc, "What is Formal Methods?", NASA Langley Methods, http://shemesh.larc.nasa.gov/fm/fmwhat.html, formal methods program, 2001.

[5] J. Kashif, A. Kashif, and E. Troubitsyna,, "Implementation of SPIN Model Checker for Formal Verification of Distance Vector Routing Protocol", International Journal of Computer Science and Information Security (IJCSIS), Vol 8, No 3, USA, ISSN 1947-5500, pp. 1-6, June 2010.

[6] A. Tarasyuk, I. Pereverzeva, E. Troubitsyna, T. Latvala, and L. Nummila, Formal Development and Assessment of a Reconfigurable On-board Satellite System, In: Frank Ortmeier, Peter Daniel (Eds.), Proceedings of 31st International Conference on Computer Safety, Reliability and Security (SAFECOMP 2012), LNCS 7612, pp.210-222, Springer, 2012.

 ISBN: 978-1-61208-230-1