Lockstep Analysis for Safety - Critical Embedded Systems

Stylianos Ganitis

AALBORG UNIVERSITY
MASTER THESIS FOR MSC. IN EMBEDDED SOFTWARE SYSTEMS
AUGUST 2015

Lockstep Analysis for Safety - Critical Embedded Systems

Stylianos Ganitis

AALBORG UNIVERSITY
STUDENT REPORT

Electronics and IT
Aalborg University
http://www.aau.dk

Title:
Lockstep Analysis
for Safety Critical
Embedded Systems

Theme:
Safety in Embedded Systems

Project Period:
Spring Semester 2015

Project Group:
-

Participant(s):
Stylianos Ganitis

Supervisor(s):
Ulrik Nyman

Copies: 5

Page Numbers: 115

Date of Completion:
August 1, 2015

Abstract:

Lockstep is a state of the art architecture of microcontrollers employed in safety-critical systems. It is a safety feature for detecting core-level faults. Complemented with other safety features for memory, peripherals and communication buses, they constitute systems for use in fields that require high functional safety.

The current project was conducted in cooperation with Airbus Defence and Space. Thus, one of the objectives is to determine if Lockstep could be beneficial to use in Avionic Systems which is the field with the strictest safety requirements.

The project focuses mainly on the Texas Instruments Lockstep architecture which was the object under investigation. Nevertheless, patents of other semiconductor companies are presented, leading the reader to realize the challenges in a Lockstep architecture.

Lockstep can be beneficial for avionic product solutions, but due to the nature of the certification process in this field, it cannot be stated that such an MCU is targeting systems of a specific DAL criticality level.

Contents

List of Figures

List of Abbrevations

1oo1D	One out of One with Diagnostics
ADC	Analog-to-Digital Converter
ASIL	Automotive Safety Integrity Level
BIST	Built In Self Test
CAN	Control Area Network
CandM	Control and Monitoring device
ECC	Error Correction Code
LBIST	Logic Built-In Self Test
PBIST	Programmable Built-In Self Test
STC	CPU Self-Test Controller Module
CCMKEYR	CCMR4F Key Register
CCMSR	CCMR4F Status Register
CCS	Code Composer Studio
CEC	Core Electronics Component
CEH	Complex Electronic Hardware
CEO	Chief Executive Office
CIS	Communications Intelligence and Security
COTS	Commercial Off The Shelf
DMA	Direct Memory Access
DPM	Decoupled Parallel Mode

DRAM	Direct Random Access Memory
EADS	European Aeronautical Defence and Space
eDMA	enhanced Direct Memory Access
EEPE	Electrical Electronic or Programmable Electronic
EEPROM	Electrically Erasable Programmable Read-Only Memory
EMIF	External Memory Interface
ESD	ElectroStatic Discharge
ESM	Error Signaling Module
ESP	Electronic Stability Program
EUROCAE	European Commission for Civil Aviation Equipment
FAA	Federal Aviation Administration
FCCU	Fault Collection and Control Unit
FMPLL	Frequency Modulated Phase Locked Loop
FT	Fault Tolerant
GPIO	General Purpose Input Output
HalCoGen	HAL Code Generator
HAL	Hardware Abstraction Layer
HTU	High-end Timer Transfer Unit
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
INTC	INterrupt Controller
ISR	Interrupt Service Routine
LED	Light Emitting Diode
LIN	Local Interconnect Network
LSM	LockStep Mode

MCU	Microprossecor Control Unit
MibADC	Multi-Buffered Analog-to-Digital Converter
MibSPI	Multi-Buffered Serial Peripheral Interface
MPU	Memory Protection Unit
N2HET	Next High-End Timer
PBIST	Programmable Built In Self Test
PCR	Peripheral Central Resource
PD	Power Domain
PBRIDGE	Peripheral Bridge
RAM	Random Access Memory
RCCU	Redundancy Control Checker Unit
RTI	Real Time Interrupt
SCI	Serial Communication Interface
SDUT	Safety Device Under Test
SECDED	Single Error Correction - Double Error Detection
SEH	Simple Electronic Hardware
SIL	Safety Integrity Level
SoR	Sphere of Replication
SPI	Serial Peripheral Interface
SPOF	Single Point Of Failure
SRAMC	Static Random Access Memory Controller
SRAM	Static Random Access Memory
STC	Supplemental Type Certificate
SWT	Software Watchdog Timer
TC	Type Certificate

TRM	Technical Reference Manual
TSOA	Technical Standard Order Authorization
VandV	Verification and Validation

Acknowledgments

First of all, I would like to thank from the bottom of my heart, my father Georgios Ganitis and my brother Giannis Ganitis for their financial but mainly for the emotional support that were providing during my whole period of studies and especially during the conduction of this Master Thesis.

Moreover, I would like to thank my professor Dr. Ulrik Nyman for his continuous support and the frequent meetings and for his willingness to propose solutions for each problem with the experience of his strong academic background.

Thank to Airbus Defence and Space, the Head of TEOHD5 department Mr. Andreas Schacht that trusted me to conduct this research and my supervisor Mr. Frank Glaser that was always there for any technical problems and questions.

Furthermore, thanks to Texas Instruments support and especially to Mr. Chuck Davenport from TI forum for the long conversations around several issues and questions. I admired his professionalism that demonstrated with the detailed and simplified answers for all of my questions.

Last but not least, I would like to thank my lovely mother Despoina Ganiti that passed away one month before I start this academic journey for my Master studies. She sacrificed everything for me and my brother. She will be always with me, in my heart and my mind. She was physically absent during these two years of my studies but the thought of her was the largest source of power for me in any challenge that I met.

Preface

This thesis is made as a completion of the Master education in Embedded Software Systems for Stylianos Ganitis. The author after the completion of his Bachelor's studies on Informatics and Communications at the Technological Education Institute of Serres in Greece, continued his academic journey at Aalborg University of Denmark in September 2013.

In July of 2014 he had the honor to conduct his six months internship in the leading European Aerospace company Airbus Defence and Space on the topic of "Continuous Integration Implementation for in-house Applications" in Friedrichshafen of Germany.

This Master Thesis is a product of six months work, beginning on February of 2015. The topic is considered state-of-the-art in the field of safety-critical embedded systems. Nevertheless, the main challenge was to find study material for the Lockstep concept, while the majority of the literature is well-protected from the semiconductor companies to protect their Intellectual Property.

This project is intended for electrical and embedded system students and for engineers interested in safety around embedded systems. It could be used as information material and guideline to employ a Lockstep architecture microcontroller in any embedded system.

Aalborg University, August 1, 2015

[illegible]

Stylianos Ganitis
<sganit13@student.aau.dk>

Chapter 1

Introduction

The increasing integration of computer systems into our daily routine and especially in systems where their behavior is vital has strengthen the need to continuously investigate the means to make such systems more reliable and safe. The safe operation of systems that may affect the integrity of human lives is essential. However, it is a prohibitively expensive process to ensure the functional correctness of a system. Hardware and software errors individually or in combination may lead to devastating consequences for humans and/or for the environment. As a result appropriate measures must be applied during the design, implementation, installation and maintenance of safety-critical systems both in terms of software and hardware [37].

At the beginning of a safety-critical system development, a hazard and risk analysis is required to determine the safety integrity level for each component individually and for the whole system. Therefore, different standards exist for each sector that provide guidelines for the lifecycle of hardware and software development. As a result, safety-critical systems should not only be safe, but they need to "look" safe in order to be in compliance with the respective safety standard by the certification authority.

A vast majority of safety-critical systems is in the form of embedded systems, where a microprocessor is employed and it is not visible from the outside environment. The highest volume production of safety-critical embedded systems is observed in automotive and domestic product development companies.

Safety is an abstract term. Although, in the field of the embedded systems the following definition could give a good interpretation:

*"**Safety** is a property of a system that it will not endanger human life or the environment"*

and a **safety-related or safety-critical** system may be defined as *"The system by which the safety of equipment or plant is assured"*[37].

The field of avionics comprises of safety-critical systems that need to fulfill the stringent safety requirements. Reliability in avionic is required to ensure the efficient management and control of aircraft and space systems. A fault occurrence in such systems may lead to a range of consequences, from disruption of schedule or aircraft operations to accident conditions [4]. Electronics is a major part of avionic systems. Thus, reliability of hardware electronic components is a main concern of the avionic system designers.

Redundancy of components or systems is a common approach in the field of avionics. In redundant architectures the main goal is that when a part of the system fails, the rest of the system is able to continue operating efficiently. Beside the fact that redundancy is an efficient way to mitigate faults, there are factors that can affect the reliability of such architectures (i.e common cause failures, load sharing).

The current research approaches a safety feature / architecture of microcontrollers that is already established in automotive, medical and industrial applications. Lockstep, is a fault detection mechanism to prevent errors that may occur at the core level. The microcontroller designers of Lockstep architecture were inspired by the concept of redundancy and came up with a multi-core processor, while at the same time attempting to deal with common cause failures that could affect the efficiency of such a diagnostic.

Lockstep is just a piece of the puzzle of the safety features that modern microcontrollers provide to help the application designers to achieve functional safety and to comply with the required certification standards. However, currently it is not broadly known in the avionics community and one aspect of this project is to investigate how it could be employed in the production of avionic systems.

This report contributes to the academic community by describing a safety feature that is used in the industry, but lacks of reliable scientific documentation sources. Furthermore, while this project was conducted under the support and supervision of the Airbus Defence and Space, it was easier for the author to obtain information by semiconductor companies due to the established partnership of the company with the majority of them.

This project is intended for electrical and embedded system students and for engineers interested in safety around embedded systems. It could be used as information material and guideline to employ a Lockstep architecture microcontroller in any embedded system.

1.1 Airbus Defence and Space

Airbus Group is a European global leader in the aeronautics, space and defence industry. Founded in July 2000 under the name European Aeronautics Defence and Space (EADS) is currently Europe's leading company in the defence and space industry. It is the second largest space company in the world and one of the top 10 defence companies globally with revenues of around 14 billion per year and approx. 40.000 employees operating in more than 170 locations worldwide. The Chief Executive Officer (CEO) of Airbus Defence and Space is Bernhard Gerwert. Airbus Defence and Space focuses on core businesses: Space, Military Aircraft, Missiles and related systems and services.

Figure 1.1: Airbus Defence and Space logo

The duty of Airbus Defence and Space is to develop cutting-edge reliable products in the field of defence and space. The main concern of the company is to help governments and institutions to protect natural resources, societies and individual freedom. The aircrafts, satellites and services help to monitor climate and crops, and to secure borders.

Airbus Defence and Space consists of four Business Lines: Military Aircraft; Space Systems; Communications, Intelligence & Security (CIS); and Electronics. Among the innovative products are the transport aircraft A400M, the military jet Eurofighter and, in the framework of the Airbus Safran Launcher joint venture, the Ariane launcher.

1.1.1 Creation of the Company

In 1997, America appeared as the global flagship of the Aeronautics, Space and Defence industry. Thus, in order to counterbalance the American leadership, the French, German and Spanish governments asked their space and defence manufacturers to present a project of fusion into a single entity, a European company.

The EADS was founded the 10th of July 2000 by the fusion of the German aerospace and defence company Daimler-Chrysler, the French missile and aircraft company Aerospatiale-Matra and the Spanish aircraft company Construcciones Aeronauticas SA.

This new company became immediately the second largest global aerospace company after its American competitor Boeing and the second largest European defence company after the British company BAE Systems.

1.1.2 The Electronics Business Line

The entity of Electronics, is headed by Thomas Mueller, and is responsible for providing equipment for system integrators that serve both Airbus Defence and Space within the Airbus Group and external customers all over the world. The products are basically addressing the civil, defence and security markets by providing ground, maritime, airborne and space applications. The products of the Electronics business line include radars and IFF systems, electronic warfare devices, avionics, space platform electronics, space payload electronics as well as optronic sensors.

1.1.3 TEOHD5 Department

The current research was conducted on behalf of TEOHD5 department of Avionics Center in Friedrichshafen in Germany. The title of the department describes what it has to deal with: "TEOHD5 - Platform Software Avionics Computer". The mission of the department is the development, delivery and maintenance of Basis Software. The definitions of *Basis Software*, *Platform*, *Support Applications / Functions* as they are presented in the internal documentation of the company are following:

- The **Basis Software** provides a Platform and additional optional Support Applications or Support Functions for the integration and installation of application software through standard interfaces and standard communication methods.

- The **Platform** provides a standard interface to the Application Software. Covers all Parts for abstraction of the Hardware. The Platform includes the Operating System, Board Support Package, Device Drivers, Libraries & Communication Interfaces (comparable to Windows).

- **Support Applications / Functions** are utilized to support and / or to maintain a complete system (1 .. X processors in a box), e.g. Loader, Maintenance Software, Redundancy Communication Concepts etc.

The expertise of the department can be categorized under the following 5 sections:

1. *Equipment Engineering Support* - Including system design, selection of processors and hardware modules, software architecture
2. *Certification Activities* - Development of Certification artifacts, Support Verification & Validation (V&V) Process, Support Safety Analysis
3. *Tools / Development Environment* - Requirement Analysis and Software Design with UML, RTOS Configuration
4. *Test & Integration* - Support Hardware Bring Up, Hardware / Software Integration and Test, Support Equipment Integration
5. *Basis Software Engineering* - Requirements Engineering according to development standard, Software Development according to RTCA - DO178B Level A-D, Software Development according to standard processes (ABD0100, VM-XT and so on).

The permanent employees of the department are equipped with a multi-role education following a continuous training on new technologies, new development methodologies and certification issues.

1.2 Problem Statement

The current research focuses on the Lockstep architecture that is employed in safety-critical systems mainly in the automotive, medical and industrial markets. The motivation behind this topic is that the Lockstep architecture is not broadly known in the scientific community. It is a state of the art feature in a variety of safety critical applications. Furthermore, it has already a history of use in a variety of safety-critical applications, but the aforementioned domains that it is employed in, are very special. Consequently, it lacks documentation in terms of a safety-feature/architecture and the only source of information is the safety-manuals of semiconductor companies for their own specific implementation.

This research focuses on clarifying what Lockstep as a broad term means, highlighting the benefits and potential drawbacks of Lockstep as a safety mechanism for error detection in embedded systems and investigating the different implementations from the several semiconductor companies that have already integrated it in their microcontrollers.

The available hardware for this research was the TMS570LS3137 Hercules Development Kit provided by Texas Instruments that has an integrated delayed-lockstep mechanism and we focused on this specific implementation. The following research questions were the primary concern of this project:

1. What is Lockstep in terms of embedded systems?
2. What are the reasons for using Lockstep as an error-detection mechanism? What kind of errors can it detect?
3. How to use Lockstep? (specifically for TMS570LS31x/21x)
4. How Lockstep should be treated in the certification process of an avionic system?
5. What are the limitations and drawbacks of a Lockstep architecture?

Chapter 2

Functional Safety and Certification in Embedded Systems

The goal of this chapter is to describe the terms around safety in embedded systems and to highlight the differences among them. Additionally, safety standards and their objectives are briefly presented. While the object under investigation is the Hercules family of microcontrollers we will point out the safety standards that it complies with and the approach that Texas Instruments follows for this purpose. Finally, we focus on the avionics standards that is the sector of the department that supported the current project.

2.1 Functional Safety and Fault Tolerance

An increasing amount of electronic devices involved in our daily routine is inevitable in the modern world and their faulty operation may harm humans and/or animals and the environment [13]. Functional safety is a fundamental concept for any field involving safety-related systems. For instance, in automotive it ensures the right operation of the airbags at the right time, in transportation it ensures that the doors will open and close at the right time, in medical sector it ensures the right operation of a pump that could lead to an over-dose of a medicine to a patient and in manufacturing it could ensure the right mix of chemicals through an automatic valve closure mechanism.

The employment of safety-mechanisms focusing on electronics and software of the system facilitates the reduction of risks to a tolerable level. It is not possible to eliminate the risk, but what the functional safety is targeting to is to reduce its negative impact. Additionally, functional safety is aiming to bring the system or the device in a tolerable level while it provides the means to measure the probability of an event occurrence and the harm that it may cause. As a result, a system is functional safe only if it contains tolerable risks.

In IEC 61508 Standard the **Safety** is defined as *"The freedom from unacceptable risk of physical injury or of damage to the health of people, either directly, or indirectly as a result of damage to property or to the environment"* and the **Functional Safety** as *"It is a part of the overall safety that depends on a system or equipment operating correctly in response to its inputs"*. In ISO 26262 the definition of **Functional Safety** is described as *"Absence of unacceptable risk due to hazards caused by mal-functional behavior of electrical and/or electronic systems"* [22].

A variety of Standards exists for specific or general markets, to facilitate the compliance of the systems to the functional safety principles. The standard for application-specific functional safety development in electronics is the IEC 61508. Various domain-specific standards have been derived from that [30]:

- IEC 60730 for white goods
- ISO 26262 for automotive passenger vehicles
- EN50128 for software development of railway applications
- IEC61513 for nuclear power plants
- IEC61511 for the process industry and associated instrumentation
- IEC62061 and ISO 13849 for machinery electrical control systems
- IEC62304 for medical systems

These standards offer guidelines to assess risk and assign safety goals for safety-related systems, they provide methods to reduce systematic failures, frameworks for quantitative analysis of random failure rates and effectiveness of diagnostics to detect them and finally, they provide guidelines for the maintenance of functional safety after the deployment of the product.

The functional safety term should not be confused with product characteristics as reliability and availability. Reliability describes the likelihood for a system to execute efficiently an assigned task in a specified amount of time. Availability is the percentage of the entire system service life during it can be used to execute the assigned task [31].

With the term **Fault Tolerance** is described the ability of a functional unit to continue the execution of a required task even in presence of faults or errors [19]. In terms of hardware, the fault tolerance is its ability to continue performing a required task even when faults occur. For instance, hardware fault tolerance 0 is interpreted as the inability of the system to continue operating efficiently even in presence of a single error. Generally fault tolerance N means that the system will lead to the loss of a safety function in occurrence of N+1 errors [43].

2.1.1 IEC 61508

The challenge in electrical, electronic or programmable electronic systems (E/E/PE) is to utilize a variety of safety functions to prevent or mitigate in a safe way a fault that may occur which would lead to an unexpected behavior of the system that could be catastrophic for the surrounding environment.

The assessment of risk to minimize this kind of failures in any sector is supported by the IEC 61508 standard series. They are international standards for E/E/PE safety-related systems (including both hardware and software) developed by the International Electrotechnical Commission (IEC). Initially they were developed for application on system level but they were extended on product and component level. They consist of seven parts:

- IEC 61508-1, General requirements
- IEC 61508-2, Requirements for E/E/PE safety-related systems
- IEC 61508-3, Software requirements
- IEC 61508-4, Definitions and abbreviations
- IEC 61508-5, Examples and methods for the determination of safety integrity levels
- IEC 61508-6, Guidelines on the application of IEC 61508-2 and IEC 61508-3
- IEC 61508-7, Overview of techniques and measures

The IEC 61508 include a series of actions that are necessary to be followed during the design, implementation, operation and maintenance of the system to comply with the required Safety Integrity Level (SIL). There are four available SILs each corresponding to a range of target likelihood of failures of a safety function. So, the safety integrity level is a factor dependent on a safety function rather than on the system or any part of it [18]. The SIL4 is considered the strictest while it targets to the protection against the highest severity level risks.

For a given process SILs are measures of the safety risk. The assignment of a SIL reveals to what extent a process is expected to perform safely and in case of a failure to what extent it can react by failing safely [43].

On a device level, IEC 61508 is used by safety equipment vendor companies to verify and document that their devices are suitable for use in SIL rated systems [34].

2.1.2 ISO 26262

ISO 26262 is a sector-specific standard for the automotive industry that focuses on functional safety. It is an extension of the IEC 61508 which covers industrial systems in general. In this standard the safety systems are divided in two categories:

- *Active safety systems* that react proactively to prevent an accident, such as ACC, ABS and ESP.

- *Passive safety systems* that refer to the reactive actions after an accident occurrence, i.e safety belts, airbags, belt tensioners.

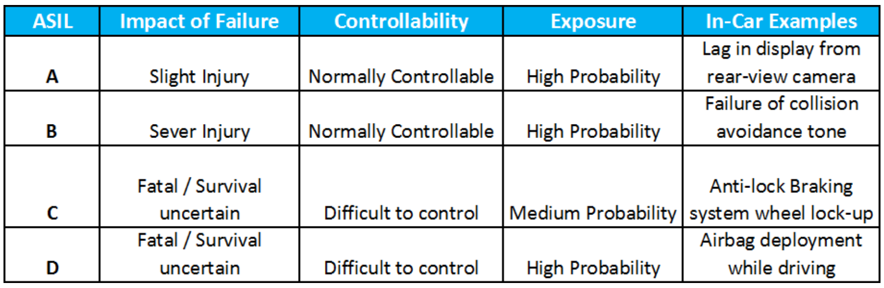
These systems and especially the electronics must also be secure, because a malfunction could lead to personal injury.

The safety lifecycle that is followed in order to ensure functional safety according to ISO 26262 starts with a definition of the system at vehicle level. For instance, investigating an airbag system, a hazard analysis and risk assessment have to be initially performed. A potential hazard for this system could be an unexpected inflation of the airbag. For each of these hazards a safety goal has to be set. For instance, how to prevent the unintended inflation of the airbag. Similarly to the IEC 61508, each of these safety goals is categorized in Automotive Safety Integrity Level (ASIL) terms. There are 4 available ASILs, from ASIL A to ASIL D, with an increasing safety classification level in this order.

The assignment of ASIL (very similar to SIL assignment in IEC 61508) to each safety goal is dependent on 3 parameters:

- *Exposure*, how often the situation is risky for the passengers or other road users
- *Controllability*, how involved people can handle the violation of the safety goal

- *Severity*, how serious could a missed safety goal become



ASIL	Impact of Failure	Controllability	Exposure	In-Car Examples
A	Slight Injury	Normally Controllable	High Probability	Lag in display from rear-view camera
B	Sever Injury	Normally Controllable	High Probability	Failure of collision avoidance tone
C	Fatal / Survival uncertain	Difficult to control	Medium Probability	Anti-lock Braking system wheel lock-up
D	Fatal / Survival uncertain	Difficult to control	High Probability	Airbag deployment while driving

Figure 2.1: ASIL classification examples [28]

In this example, the unexpected inflation of the airbag would most likely belong to ASIL D level. A functional safety concept is required for each potential hazard. For the airbag example, the functional concept could rely on a redundancy concept where the system would consist of a control and monitoring channel. The airbag would only inflate if both of the channels produce identical output. The technical aspects of these functional safety concepts (i.e sufficient number of independent sensors and the method to independently enable the trigger circuit) compromise the technical safety concept. Software and hardware requirements are determined then according to the technical safety concept.

2.1.3 Avionics Certification (DO - 178C and DO - 254)

The significance of the correct operation of avionic systems, due to the consequences of the hazards that they may cause in a potential malfunction, makes the avionic certification a complex process with stricter criteria than the aforementioned standards.

The DO-178B and DO-254 are documents that provide guidelines for dealing with respectively software and hardware in the process of avionic systems development. These two documents are written by people with formal software expertise and are quite similar. Initially the hardware was not requiring the strict standards as the software did. The evolution in the field of aviation revealed the need for a set of rules that have to be followed in the hardware development life-cycle as well, because the majority of the modern avionic systems consist of both hardware and software components [45].

The series of the DO-178 (DO-178A, DO-178B, DO-178C) and DO-254 were developed by the commercial avionics community in cooperation with RTCA (a Federal Advisory Committee) in an attempt to establish common guidelines for system developers and government certifying agencies for the certification of avionic products.

It is worth to note at this point what is “certified” in the avionics world. The Federal Aviation Administration (FAA), the national aviation authority of USA, is responsible for ensuring that all air travel is safe. There is a commonly erroneous notion about what is certifiable in the avionics world. The FAA does not specifically certify software or hardware, but systems. The way and the standard that a system will be certified varies. The required certificate depends on the product and its usability:

- Technical Standard Order Authorization (TSOA) - Is used to approve that a specific device (i.e GPWS, Radio, Weather Radar) can be used on a different aircraft
- Type Certificate (TC) - Certifies systems for new aircraft
- Supplemental Type Certificate (STC) - Certifies changes of design not requiring a new TC

The real intention of the certification process and the DO documents is to help the industry to prove that each software or hardware product meets its intended functionality. The terminology that is used in the field of avionics reveals the different levels of compliance for a system [45]:

1. “Certified” is an entire system in which each component may have different certification level
2. “Certifiable” is a component of a system that achieves its higher certification status before certifying it within the whole system
3. “Qualified” is used in terms of a tool. Since it is not participating in the “fly” operations does not require to be “certified”
4. “Compliant” is a certified system from a different entity than FAA (i.e military or non commercial avionics)

In avionics, after a safety assessment of the system that is driven by SAE standard ARP4754 (*Certification Considerations For Highly-Integrated or Complex Aircraft Systems*), a criticality level is determined. It might be from Level A to Level E. Informally speaking, the Level A is the most stringent where a potential failure might lead to loss of life of all passengers, at Level B a few people (passengers or crew) may die, at Level C there are major injuries, at Level D minor injuries and finally at Level E no impact on the people and the environment.

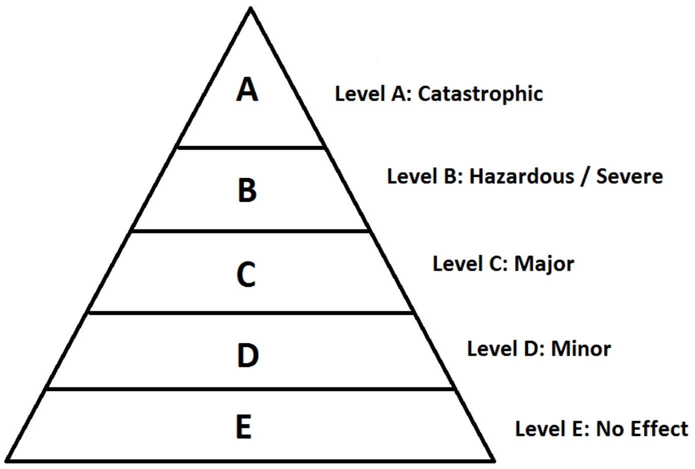


Figure 2.2: Criticality Level Pyramid for Avionic Systems [45]

It should be noticed that even appliances that are not critical for the safety of the flight require at least a Level D certification. A real-life example is an aircraft crash that was caused by a seat-back video screen for the entertainment of the passengers. The crash analysis revealed that these appliances were sharing a bus for power supply with critical devices in the cockpit of the aircraft. The malfunction of these safety unrelated devices affected the safety of the whole aircraft.

Software Considerations in Airborne Systems and Equipment Certification - DO-178

As was already mentioned the DO-178 is aiming at establishing guidelines in terms of software in the development of avionic systems. The initial version, DO-178 was addressing the software life-cycle in avionics. A later version, the DO-178A was enhanced with the classification of the system into criticality levels and component testing methods to increase the quality of the overall system. The next version, DO-178B was developed from scratch. Software quality was increased by the addition of more planning methods, more application development practices, the integration of COTS products and tools and the integration of techniques for continuous testing and monitoring of the system in real-world conditions. The latest DO-178C version addresses known issues of the previous version while it provides new tool qualification guidance.

The benefits derived from following the DO-178 series are not restricted only at the certification. The guidelines facilitate the verifiable software quality, higher reliability, greater re-usability, decreased maintenance, lower life cycle costs, faster integration to the hardware and increased probability of error detection during the development phase.

Design Assurance Guidance for Airborne Electronic Hardware - DO-254

DO-254 was released in 2000 and was formally required by FAA from 2005. The European commission EUROCAE (European Commission for Civil Aviation Equipment) provides the ED-80 containing the same content with DO-254. It is a counterpart of the well-established DO-178, addressing electronic hardware issues to reach the certification compliance.

DO-254 follows a top-down approach on electronic hardware starting from the system level. The scope of this document is to address certification issues and to ensure the functional safety on electronic hardware components where the complexity of these devices is increasing and their size is shrinking [4]. It relates exclusively to airborne systems and it is worth mentioning that electronic-hardware is not certified as a stand-alone entity.

There is a distinction that derives from an analysis of a hardware component: It may be either a Simple Electronic Hardware (SEH) or a Complex Electronic Hardware (CEH). If it is determined that a component belongs to the former category after the conduction of the appropriate analysis (i.e the component is fully testable, or it does not contain silicon-based logic) the DO-254 document is not required. More specifically, a simple component is one that provides the capability of performing exhaustive testing on it, examining all the potential inputs.

Further steps in an abstract form could be defined as follows:

- Systems aspects are considered first - System or function Development Assurance Level (DAL) is determined for a hazard that may occur at aircraft level. Additionally, a hazard assessment should be performed. System functions and DAL are allocated to hardware. Requirements of the system are allocated to the hardware.

- Identification of functions and their effects - For each function that is expressed by a requirement and has an assigned DAL, hardware components are designed to fulfill the respective requirement.

- Design assurance approach for hardware - Requirements based verification is performed to determine the criteria of the assigned DAL.

2.1.4 The Meaning of 1oo1D

For the Hercules family of microcontrollers the default Lockstep mode is described as 1oo1D (one-out-of-one with diagnostics). A 1oo1D system is comprised of a single channel with a diagnostic channel.

The terminology of these architectures is mentioned in the IEC-61508 standard. Generally, the definition of MooN in IEC 61508-4 is given as "M out of N channels architecture, where either of the two channels can perform the safety function" and the definition of MooND is "M out of N channel architecture with Diagnostics" [19]. As is also mentioned in the same safety standard, in case of a dangerous failure, the safety function fails as well. In simple words, a MooN system contains N identical components and if at least M out of N components are operating correctly, the system is error-free. Beyond the hardware this redundancy concept may also be applied at software, time or information level.

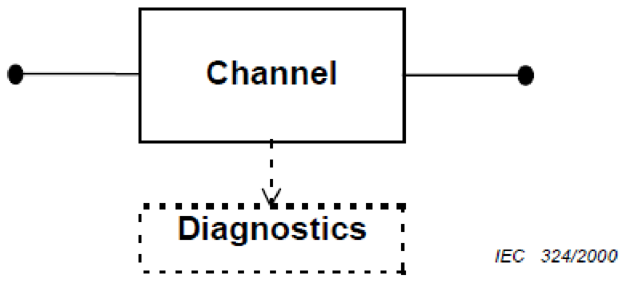


Figure 2.3: 1oo1 Block Diagram [19]

Anthony Vaughan, the manager of Hercules safety group of Microcontrollers in North America, describes the usability of this 1oo1D architecture in the aerospace area and how it can be utilized to meet the strict safety criteria that are required in this industry with the required highest level functional safety [46].

A common approach to ensure the functional safety in embedded aerospace electronic systems is the employment of the 2oo3 architecture (see Figure 2.4). Systems following this architecture comprise of 3 (probably dissimilar) embedded controllers and a voting circuit. In case of a fault occurrence in one of the three controllers the system availability remains unaffected while the other two control the system. This fail operational architecture is proven to be efficient for flight critical applications where is required high system availability and extremely low failure rate.

Beyond the fact that 1oo1D is already used in a variety of safety critical applica-

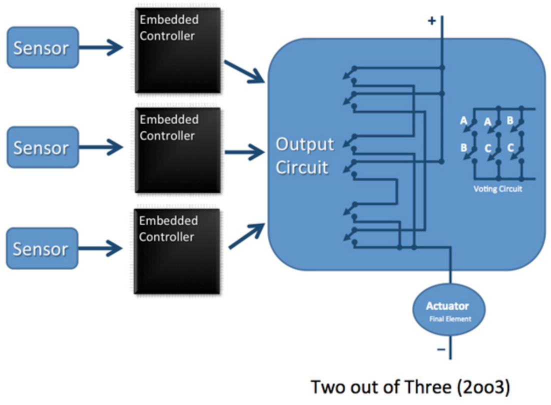


Figure 2.4: Two Out Of Three Architecture (2003) [46]

tions, is not enough for the safety requirements of the aerospace area. Instead, a Two out of Two with Diagnostics (2oo2D) architecture may be utilized to create fail-safe systems. This proposal that is analyzed in the article of A. Vaughan deploys two separate channels each consisting of the 1oo1D controller (Lockstep architecture) and a signal that is sent bidirectional between the controllers, to notify that a fault occurred, maintaining the system availability.

Two main advantages are pointed out by this implementation: It is simpler than the 2oo3 architecture and could be less susceptible to random failures while the system consists of fewer controllers.

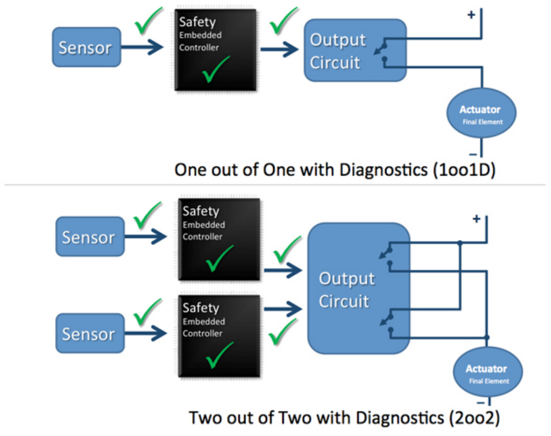

Sensor
Safety Embedded Controller
Output Circuit
+
Actuator
Final Element
–
One out of One with Diagnostics (1oo1D)
Sensor
Sensor
Safety Embedded Controller
Safety Embedded Controller
Output Circuit
+
Actuator
Final Element
–
Two out of Two with Diagnostics (2oo2)

Figure 2.5: One out of One with Diagnostics (1oo1D) - Two out of Two (2oo2) Architectures [46]

2.1.5 Texas Instruments Approach To Functional Safety

One of the initial tasks in functional safety development is a hazard and risk analysis of the target application. Depending on this assessment, the level or the acceptable risk is determined regarding the SILs as already discussed in Section 2.1.1. Then, the process of software, hardware and system development must comply with the requirements of the derived SIL in order to handle efficiently systematic failures. Hardware and systems can be assessed via quantitative metrics to reveal the likelihood of failure per hour and failure fraction in the final application product [30].

A SIL is only achieved by evaluating the whole system and at component level we could only say that "a component is developed according to SILx" or " this component is suitable for use in SILx systems". The derived SIL of a system consisting of a variety of components with different SILs is the weakest link.

As mentioned before the hardware and system can be assessed regarding the failure rate metrics. On the other hand, in the software there are not such metrics while there are not random failure modes. The IEC 61508-3 describes the requirements for the software life cycle to comply with a determined SIL. The highest SIL that a broad range of application is targeting is the SIL3. For SIL4 which is an ultrahigh safety level, redundancy architectures are used (i.e multiple channels) such as two versions of software serving the same purpose. For instance, in a single system two RTOSs from different suppliers can be employed to achieve compliance with SIL4.

Developing a system with use of a certified COTS (Commercial Off The Self) software such as a RTOS does not necessarily mean that the final product complies with the COTS's SIL. It ensures that the RTOS provides the mechanisms for several tasks that are required for achieving functional safety such as memory allocation, task synchronization, maintenance of temporal constraints and so on. In essence the compliance depends on the loyalty to the specific SIL development and testing requirements. The artifacts of following these rules should be well documented and they are provided to the safety assessor.

The TMS570LSxxx and RM48x are two processor families developed by Texas Instruments that are suitable for use in SIL3 systems. They have been independently assessed by Exida[1] and they were developed to address the risks of MCU development and to restrict the random failures that have been noticed in safety critical systems during operation.

[1]Exida is one of the leading authorities in the field of functional safety

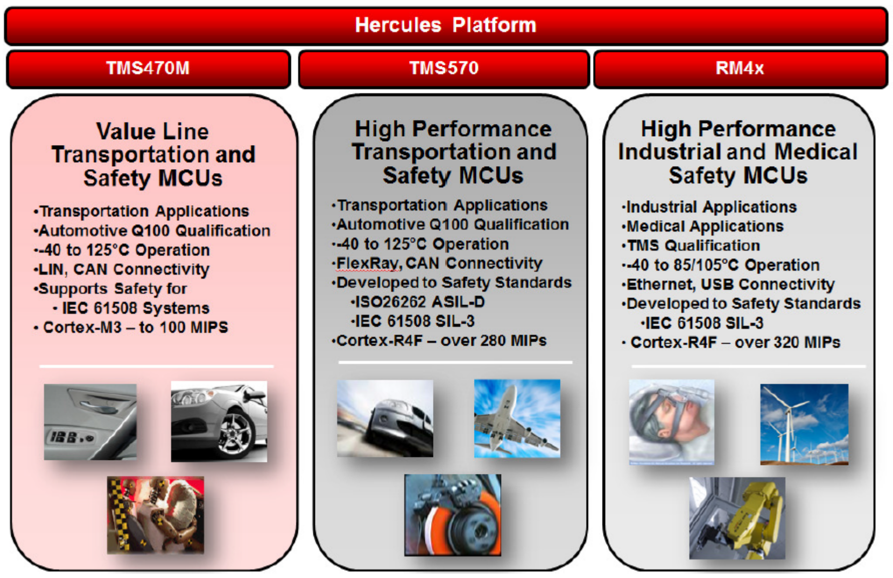

Hercules Platform
TMS470M
TMS570
RM4x
Value Line Transportation and Safety MCUs
•Transportation Applications
•Automotive Q100 Qualification
•-40 to 125°C Operation
•LIN, CAN Connectivity
•Supports Safety for
• IEC 61508 Systems
• Cortex-M3 – to 100 MIPS
High Performance Transportation and Safety MCUs
•Transportation Applications
•Automotive Q100 Qualification
•-40 to 125°C Operation
•FlexRay, CAN Connectivity
•Developed to Safety Standards
•ISO26262 ASIL-D
•IEC 61508 SIL-3
•Cortex-R4F – over 280 MIPs
High Performance Industrial and Medical Safety MCUs
•Industrial Applications
•Medical Applications
•TMS Qualification
•-40 to 85/105°C Operation
•Ethernet, USB Connectivity
•Developed to Safety Standards
•IEC 61508 SIL-3
• Cortex-R4F – over 320 MIPs

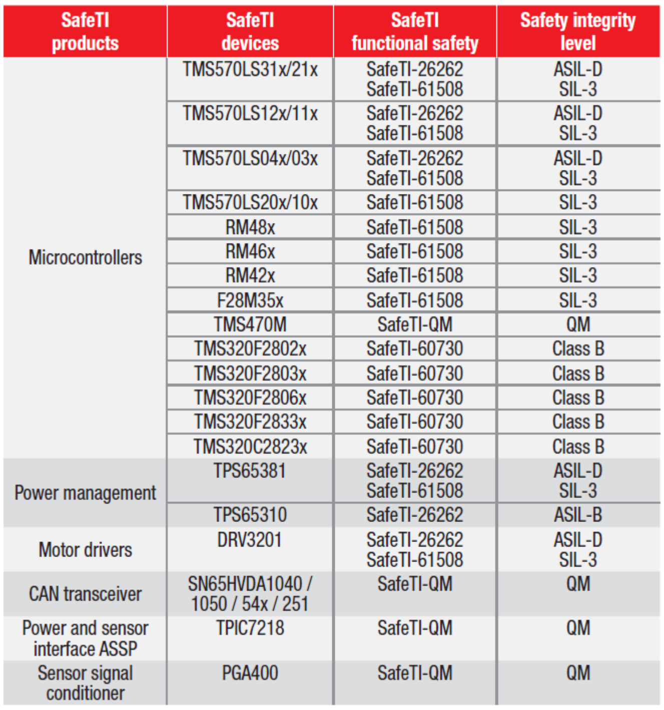
Figure 2.6: The Hercules ARM Family of safety MCUs [30]

Combining such an MCU that is specifically developed to comply with a SIL and a RTOS that ensures the requested requirements for this safety level simplifies the certification process for both the certification authority and the development company. In this direction, Texas Instruments provides the SafeTI design package that helps their customers to build applications capable of easily achieving compliance to safety functional standards. The SafeTI design package consists of 5 components [32]:

1. Safety-related semiconductor components
2. Safety-related documents, tools and software
3. Complementary embedded processing and analog components
4. Quality manufacturing process
5. Safety development process

In Figure 2.7 is depicted a list of the currently available SafeTI devices and the integrity levels according to what they are developed respectively.

The two categories of errors that the functional safety has to deal with are known as systematic and random faults. Random faults are inherent to the application and the metrics that already discussed to determine the failure rate are not applicable to them. Random errors could occur by a permanent failure of a circuit,



SafeTI products	SafeTI devices	SafeTI functional safety	Safety integrity level
Microcontrollers	TMS570LS31x/21x	SafeTI-26262 SafeTI-61508	ASIL-D SIL-3
	TMS570LS12x/11x	SafeTI-26262 SafeTI-61508	ASIL-D SIL-3
	TMS570LS04x/03x	SafeTI-26262 SafeTI-61508	ASIL-D SIL-3
	TMS570LS20x/10x	SafeTI-61508	SIL-3
	RM48x	SafeTI-61508	SIL-3
	RM46x	SafeTI-61508	SIL-3
	RM42x	SafeTI-61508	SIL-3
	F28M35x	SafeTI-61508	SIL-3
	TMS470M	SafeTI-QM	QM
	TMS320F2802x	SafeTI-60730	Class B
	TMS320F2803x	SafeTI-60730	Class B
	TMS320F2806x	SafeTI-60730	Class B
	TMS320F2833x	SafeTI-60730	Class B
	TMS320C2823x	SafeTI-60730	Class B
Power management	TPS65381	SafeTI-26262 SafeTI-61508	ASIL-D SIL-3
	TPS65310	SafeTI-26262	ASIL-B
Motor drivers	DRV3201	SafeTI-26262 SafeTI-61508	ASIL-D SIL-3
CAN transceiver	SN65HVDA1040 / 1050 / 54x / 251	SafeTI-QM	QM
Power and sensor interface ASSP	TPIC7218	SafeTI-QM	QM
Sensor signal conditioner	PGA400	SafeTI-QM	QM

Figure 2.7: SafeTI Devices [32]

temporary failure of SRAM or shorting of adjacent signals in the MCU and so on.

For the prevention of such errors, semiconductor companies utilize safety mechanisms that execute during the operation of the system within the control loop. Because of the temporal constraints in real time embedded systems the burden of the continuously or even periodically execution of these diagnostic tests became a significant issue for the semiconductor companies. As a result the need for parallel execution of such tests became vital. The Hercules MCU platform provide integrated into hardware diagnostics facilitating tight control loops while the checks run in parallel.

The application developer has to be aware of common mode failure when implementing such safety mechanisms. Common mode failure describes the case that faults occur in commonly shared signals. The temporal and physical diversity as is presented in Sections 4.4.6.1 and 4.4.6.2 are measures to prevent the common mode failure in the redundant CPU architecture.

The approach of Texas Instruments in the TMS570 family of processors for facilitating the application developers to achieve functional safety is called "Safe Island". This term describes the combination of hardware and software diagnostics while at the same time concerning the cost. In this philosophy elements like power, clock, reset, CPU, Flash, SRAM and associated interconnect to Flash and SRAM operate continuously hardware safety mechanisms to insure the right execution of software. Therefore, after ensuring the correct operation of the aforementioned elements, software-based diagnostics can be executed to verify the correctness of other device elements, such as peripherals [21].

Chapter 3

Embedded Software Development

In this section we present the hardware and software that was used during this project. The hardware selection was predetermined by the company, while the TMS570LS3137 fulfills all the safety and communication features that are needed to be employed as a coordinator between two modules of a large radar project that is currently in progress. Regarding the software, the recommendations of Texas Instruments were used to provide stability and flexibility regarding a potential certification process, such as the Code Composer Studio IDE, the HALCoGen for automatic code generation and the TI compiler.

3.1 Hardware

The hardware that was available for the conduction of the experiments in this project is described in this section. The TMS570LS3137HDK evaluation board from Texas Instruments and the SafeTI Hitex Safety Kit - TMS570LS3137 by Hitex in cooperation with Texas Instruments are briefly demonstrated.

3.1.1 Hercules TMS570LS3137 microcontroller by Texas Instruments

TMS570 are the first Microprocessor Control Units (MCU) based on ARM® Cortex^{TM}-R4F based floating point that meet the IEC61508/SIL3 standards. This family of microprocessors is designed specifically to comply with automotive and transportation safety standards, while it consists of dual Cortex-R4F processors operating in Lockstep. System-wide protection is insured by the error detection in the processor and the memory modules. Both single and double precision is achieved with the integrated high-performance Floating Point Unit (FPU). They are targeting to safety applications of transportation industry including [39]:

- Automotive chassis and stability control
- Electric power steering
- Hybrid and electric vehicles
- Aerospace
- Railway communications
- Off-road vehicle engine controlp

The CPU offers 1.66 DMIPS/MHz[1] and can provide up to 298 DMIPS with configurations that can run at 180MHz. The big-endian [BE32] word-invariant format is supported on this device. The TMS570LS3137 has 3MB flash and 256KB RAM which are protected with Single-bit Error Correction and Double-bit Error Detection (SECDED). The flash is an Electrically Erasable and Programmable Memory (EEPROM) with a 64 bit wide data bus. Furthermore, program and read operations are performed with a 3.3V supply input to the flash memory. Same input supply is used for the I/O operations.

The TMS570LS3137 device has 3MB of integrated flash and 256KB of data RAM. Both the flash and RAM have single-bit error correction and double-bit error detection. The flash memory on this device is a nonvolatile, electrically erasable, and programmable memory implemented with a 64-bit-wide data bus interface. The flash operates on a 3.3-V supply input (same level as I/O supply) for all read, program, and erase operations. When in pipeline mode, the flash operates with a system clock frequency of up to 180 MHz. The SRAM supports single-cycle read and write accesses in byte, halfword, word, and double-word modes.

Additionally, two Next Generation High-End Timer (N2HET) co-processors and two Analog-to-Digital Converters (ADC) with up to 24 inputs are integrated in the MCU. The N2HET module is an intelligent timer that is used in real-time applications while it provides sophisticated timing functions. A High-End Time Transfer Unit (HTU) with a built-in Memory Protection Unit (MPU), facilitates the DMA-type transactions of N2HET and main memory.

Moreover, two Multi-Buffered Analog-to-Digital Converters (MibADCs) of 24 channels with 64 words and parity-protection are integrated. A conversion can be performed on an individual channel or a group of them that is set by the software.

[1]DMIPS: Dhrystone Million Instructions Per Second is a benchmark to measure the integer performance of processors and compilers [7].

Three Multi-Buffered Serial Peripheral Interfaces (MibSPIs), two Serial Peripheral Interfaces (SPI), one Local Interconnect Network (LIN), one Serial Communication Interface (SCI), one Inter-Integrated Circuit (I2C), three Control Area Network (DCAN), one Ethernet interface and one Flexray controller are the communication interfaces integrated into the device.

Two Frequency-Modulated Phase-Locked Loop (FMPLL) clock modules are employed. These modules are capable of multiplying the external frequency reference when higher frequency is required for internal use.

A Direct Memory Access (DMA) controller with 16 channels, 32 control packets and parity-protected is integrated, with a built-in MPU to protect the rest of the memory in case of a malfunction of the DMA (the DMA is restricted to a prescribed area of the memory).

All the device errors are controlled by the Error Signaling Module (ESM) while it determines if an interrupt will be triggered or an external pin (ERROR) is asserted for different kinds of errors.

Finally, expansion of the device with external modules such as synchronous/asynchronous memory devices, peripherals or FPGAs can be achieved with the External Memory Interface (EMIF) module [41].

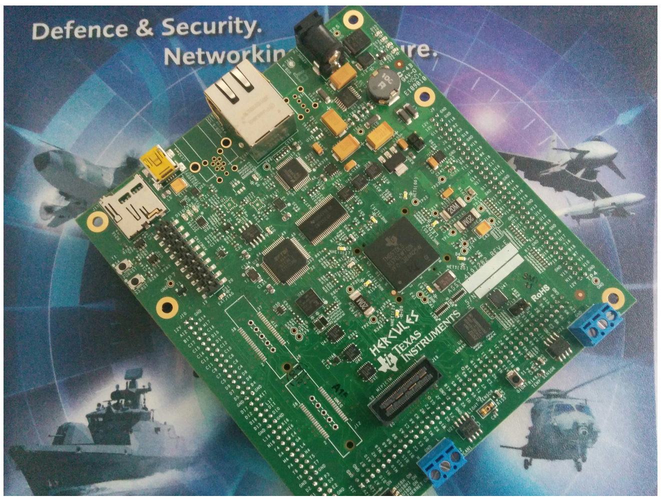


Figure 3.1: TMS570LS3137 Hercules Development Kit

The TMS570LS3137 Hercules Development Kit was employed in this project to evaluate and start the development with the Hercules platform of these safety series of MCUs by Texas Instruments. The development board provides the following features:

- 337p BGA TMS570LS3137 MCU, on board
- On Board USB XDS100vs JTAG
- On Board SCI to USB Serial
- 20 pin ARM and MIPI JTAG connectors
- 6 White NHET LEDs
- 2 Tri-Color RGB NHET LEDs
- 8MB SDRAM (EMIF)
- Ambient light and temperature sensor
- 2 CAN transceivers
- 1 RJ-45 ethernet port
- 1 Micro SD card slot (SPI mode)

3.1.2 SafeTI Hitex Safety Kit - TMS570LS31x/21x

The SafeTI is a tool developed by Hitex in cooperation with Texas Instruments and facilitates the evaluation of safety features of TI's Hercules safety microcontrollers [33]. The kit consists of two Hercules microcontrollers placed on the same evaluation board, where the first acts as a Safety Device Under Test (SDUT) and the second as a Control and Monitoring Device (C&M). There are two variations regarding the model of the SDUT device: it may be either an TMS570LS3137 or an RM48L952 MCU. In the current project was used the former. The C&M device is in both of the cases an RM48L952.

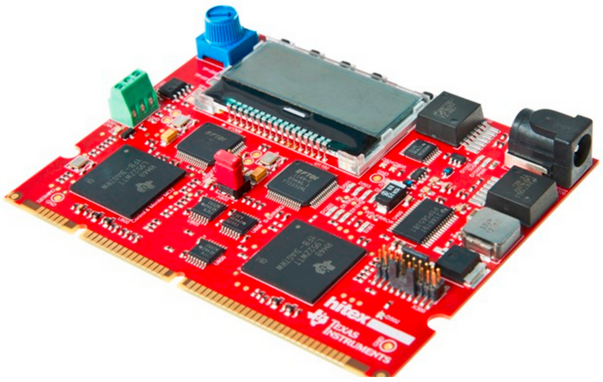


Figure 3.2: SafeTI Hitex Safety Kit [17]

Additionally, a TPS65381-Q1 companion chip is integrated to monitor the operation of the SDUT device. This companion chip is a power source that contains a watchdog which requires specific messages from the SDUT in a predetermined amount of time, in order to detect potential hardware faults.

The software stack of the system consists of four layers from bottom to top:

1. The hardware abstraction layer
2. The Texas Instruments SafeTI Diagnostic Library
3. The SAFERTOS operating system
4. Example user application

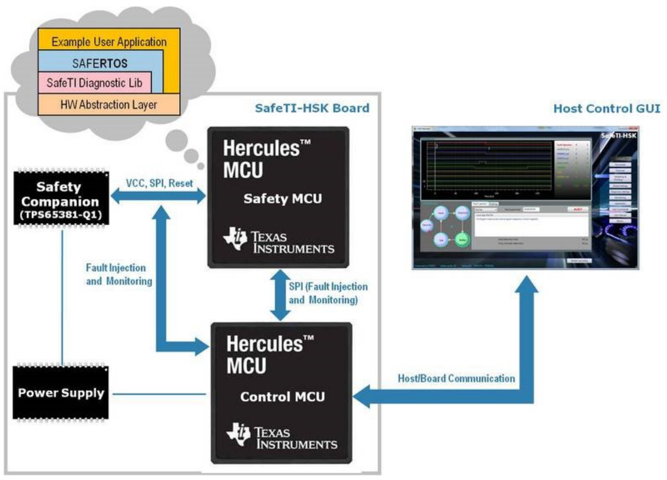


Figure 3.3: SafeTI HSK Block Diagram [33]

Essentially, the purpose of the kit is to monitor the reactions of the SDUT device when faults are injected in real-time by the C&M.

The C&M is responsible for monitoring the behavior of the device under test and provide the results and the measurement of system response times in a GUI based application in a graphical manner.

3.2 Software

The software tools that were used during the conduction of the experiments and the evaluation of the Lockstep architecture are presented in this section. Code Composer Studio (CCS) for developing and debugging, HALCoGen for configuration of the peripherals and automatic code generation and the GNU ARM toolchain that was used as an alternative to the CCS integrated toolchain are briefly presented.

3.2.1 Code Composer Studio (CCS) and Tools Suite

Code Composer Studio is an Eclipse-base integrated development environment (IDE) supporting Texas Instrument's portfolio of microcontrollers and embedded processors. It is used for developing and debugging of embedded applications. It provides features such as source code editor, project build environment, debugger, profiler and simulators [5]. Additionally, an optimizing C/C++ compiler is integrated.

The **debugger** provides conditional or hardware breakpoints that are based on local variables, registers or C expressions. Moreover, a memory window is provided for all levels of memory to inspect potential issues, such as cache coherency problems. The debugger even supports complex multi-core or multi-processor systems, providing synchronous operations and global breakpoints.

Measurement of code performance and efficient use of resources in debugging or development mode is achieved through the **profiler**. Instruction cycles of a function, cache misses/hits, pipeline stalls and branches are events that can be measured and monitored by the profiler. The profiling facilities are active through the whole development cycle and the goal is to end up in a finely-tuned code when it is used efficiently by developers.

Testing and performance benchmarking are time consuming tasks that should be performed continuously during the development process. CCS supports a **scripting** environment suitable for the automation of such processes.

The flexibility of the Eclipse-based environment that provides the capability of adding plugins and extensions, the aforementioned features plus others such as image analysis and visualization, simulation and support for TI's real-time operating systems, make the CCS an ideal IDE for embedded application development for TI's systems.

3.2.2 HALCoGen

The lowest software layer of an embedded system that directly interacts with the hardware is called Hardware Abstraction Layer (HAL). It is usually responsible for the system initialization and the direct access to the several components of the MCU.

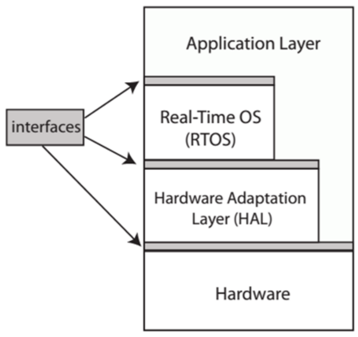


Figure 3.4: Hardware Abstraction Layer in an Embedded system [36]

Texas Instruments provides a GUI tool called HAL Code Generator (HALCoGen) that the user can configure and generate device specific code for initialization of the system and configuration even for the safety related functions [15]. It could also be described as a driver code generator tool that insures a stable HAL code.

3.2.3 GNU ARM Toolchain

As an alternative toolchain to the CCS integrated tools, the GNU ARM was employed. The GNU ARM toolchain provides GCC compiler support for development on embedded ARM processors with the contribution of ARM employees. Specifically it supports Cortex-R/Cortex-M processor families, covering Cortex-M0, Cortex-M3, Cortex-M4, Cortex-M0+, Cortex-M7, Cortex-R4, Cortex-R5 and Cortex-R7 [14].

The executables for the GNU ARM toolchain that are officially provided do not support the big-endian [BE32] word-invariant format that is required for the Cortex-R4F CPU of the TMS570LS3137. Building manually the binaries that are also provided from GNU ARM community was necessary, applying first a diff patch (see Appendix D) on the source code.

After applying the patch to the binaries that are provided from the GNU ARM webpage [14], the process for building them is as described in the Appendix E.

The reason for the investigation of an alternative toolchain is a potential future need of the company to use the GCC compiler in a project that employs such a microcontroller. Additionally, the type of license for the CCS integrated toolchain varies regarding the specific model of the microcontroller. For some models is provided free, while for others it has code size limitations in a free version.

Chapter 4

Safety Features of Hercules Family

Texas Instruments provides a series of safety features for the Hercules family of microcontrollers that target to the functional safety of the applications. The current research focuses on the Lockstep architecture and the benefits that it may provide in a safety-critical system, but this section aims to give an overview of the other provided features that make Hercules a competent family of microcontrollers in a variety of safety applications, currently in the transportation, industrial and medical markets.

Instead of using software mechanisms to detect hardware errors, Hercules microcontrollers implement safety in hardware that reduces the software overhead while it increases the performance. The Lockstep architecture is presented thoroughly in Chapter 5 and eliminates redundant system requirements and consequently additional cost: In absence of the Lockstep architecture, redundant subsystems are employed to comply with functional safety requirements. This redundancy of the subsystems increases the cost and the space of the system significantly.

Except of Lockstep as a safety feature of the Hercules family, the CPU hardware built-in self test (BIST) is employed for detection of latent defects without extra software overhead. Furthermore, for the protection of memories and buses Error Correction Code (ECC) logic is integrated in the CPU. Moreover, high diagnostic coverage is achieved while all Random Access Memories (RAM) can be tested using HW BIST and a Memory Protection Unit is employed to assure protection against deterministic errors in the application software [16].

4.1 CPU Logic Built-In Self Test (LBIST) Self-Test Controller (STC)

Due to the high complexity of a Cortex-R4F CPU, TI does not recommend several CPU software-based tests that are performed by other safety critical microcontrollers. Though, a hardware-based CPU Logic Built-In Self Test (LBIST) Self-Test Controller (STC) scheme is proposed for providing high diagnostic coverage on the CPUs operating in Lockstep at a transistor level. The STC is utilized to test the CPU core using the deterministic LBIST controller as the test engine. This hardware-based solution is more efficient in terms of required memory, detection capability and execution time of the tests [35].

The LBIST tests are triggered by the software, where the developer may select to run all or part of them based on the execution time. The execution time that the application sets for the LBIST tests in each safety-critical loop can be allocated to the LBIST diagnostic. The version used in TMS570LS31x/21x family is capable of running the LBIST tests on both cores simultaneously [40].

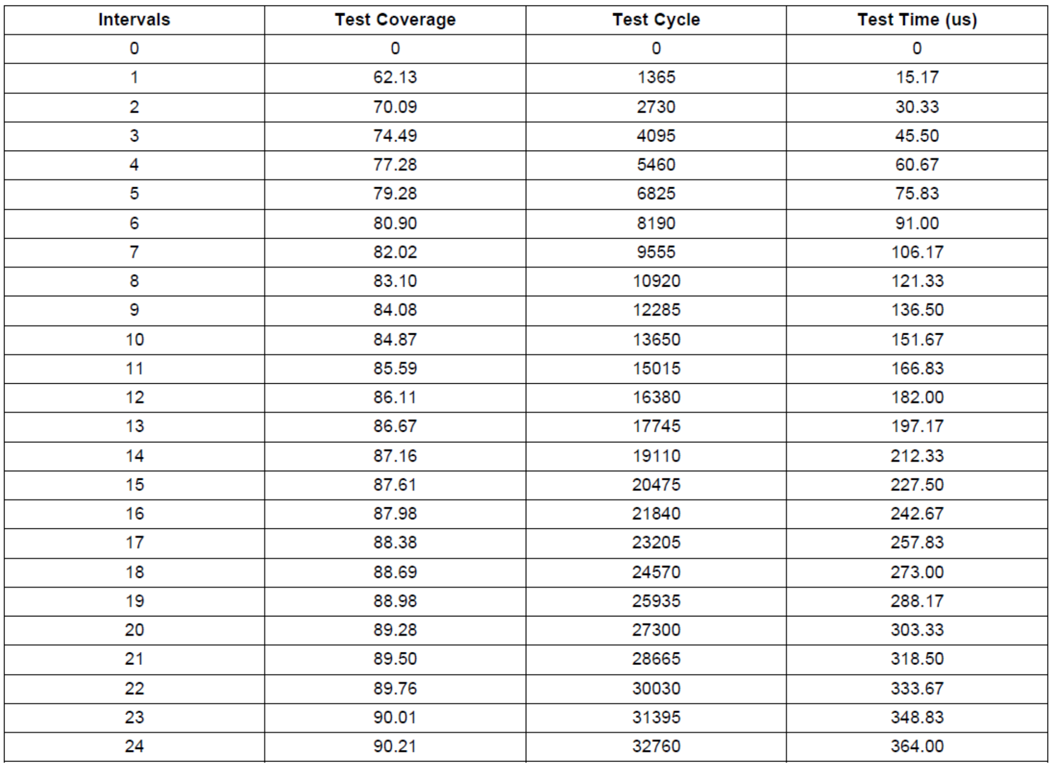
Running the LBIST STC is much more costly in terms of power than a normal application due to the increase of level of transistor switching per clock cycle that is required. The user is able to reduce the CPU clock for the duration of the tests. It is a trade-off between current consumption and time to complete the execution of the diagnostic tests.

Before the execution of the LBIST test, a context save is performed and the CPU is isolated from the rest of the system. The remainder logic of the system does not interrupt its operation. After the completion of the test execution a CPU reset is required, the system is notified that the reset was performed due to the completion of this test (through the SYSER register) and the normal operation of the system continues by restoring the CPU context. Essentially, a context saving is not required when the STC runs on start-up, while the start-up configurations will be performed.

The LBIST provides the option to test a proper operation of the diagnostic. The testing process is deterministic and in order to ensure that a test is performed within expected time, a timeout counter is employed to detect a potential non-deterministic situation that indicates an error. Moreover, the STC can operate in an error-forcing mode, where an input error is injected to verify the right functionality of the module in the error detection and propagation to the system level.

The STC supports 24 intervals that the test process can be divided into and run a few intervals at a time. The user is able to run either all the intervals together or in slices specifying if the STC will continue from the next interval or restarting from interval 0.

The table in Figure 4.1 depicts the coverage percent and time that is required for each interval in a typical case of the device running at HCLK = 180 MHz, VCLK = 90 MHz, and STCCLK = 90 MHz.



Intervals	Test Coverage	Test Cycle	Test Time (us)
0	0	0	0
1	62.13	1365	15.17
2	70.09	2730	30.33
3	74.49	4095	45.50
4	77.28	5460	60.67
5	79.28	6825	75.83
6	80.90	8190	91.00
7	82.02	9555	106.17
8	83.10	10920	121.33
9	84.08	12285	136.50
10	84.87	13650	151.67
11	85.59	15015	166.83
12	86.11	16380	182.00
13	86.67	17745	197.17
14	87.16	19110	212.33
15	87.61	20475	227.50
16	87.98	21840	242.67
17	88.38	23205	257.83
18	88.69	24570	273.00
19	88.98	25935	288.17
20	89.28	27300	303.33
21	89.50	28665	318.50
22	89.76	30030	333.67
23	90.01	31395	348.83
24	90.21	32760	364.00

Figure 4.1: Test Coverage and Duration [40]

- HCLK: Clock domain used by the high-speed system modules: Flash, memory interfaces, TCRAM interface, Error Signaling Module (ESM), DMA
- VCLK: Clock domain used by some system modules (VIM), peripheral modules accessed via the Peripheral Central Resource (PCR) controller, and all other register interfaces also accessed via the PCR
- STCCLK: Determines the self-test execution speed, by dividing HCLCK

4.2 Programmable Built-In Self-Test (PBIST) Module

The Programmable Built-In Self-Test Module integrated on the Hercules microcontrollers, facilitates the testing of on-chip memories via a host processor interface. A dedicated on-chip PBIST ROM is employed to store information regarding on-chip memories, memory groupings, memory background patterns and test algorithms. Moreover, a test of the memories at their maximum possible speed in application is provided and an intelligent clock gating is implemented in order to conserve power. Finally, it provides the ability to test its own PBIST ROM itself [40].

Regarding the architecture, a small co-processor with a dedicate instruction set specialized in memory testing is employed which executes routines that are stored in the PBIST ROM. In comparison to software-based techniques that are used for memory testing from the main Cortex-R4F processor, the PBIST provides significant advantages:

- A dedicated memory path is used specifically for the self-test of the memories, resolving the problem of the long access paths that embedded CPUs have to face.

- The dedicated instruction set that is specifically formulated for memory testing facilitates the development of such test routines.
  The generic instruction set of an embedded CPU may do the development and execution of such self-tests a complex task.

- The algorithm code on a conventional embedded CPU for this purpose is significantly larger than the code of PBIST.

- The size of a PBIST controller is significantly smaller than the size of an embedded CPU.

The execution of the tests is triggered by the application where the developer has the option to select the algorithms that will run against the memory modules of the system. PBIST tests are destructive for the memory contents and if they execute during operation of the system, the application may copy the data of the memory under test to a non-tested memory and restore the data after the completion of the current test.

A potential error is indicated in PBIST status registers and a further feature is the capability of keeping log of the errors.

Similarly to the LBIST controller (see Section 4.1), PBIST is a deterministic diagnostic method. Therefore, a timeout counter could be programmably implemented using the Real Time Interrupt (RTI) module to indicate potential failure in test completion within expected amount of time [35].

Some of the test algorithms that are used for PBIST are following [40]:

1. March13N:

   March13N is the algorithm for SRAM testing that provides the highest overall coverage. The following algorithms are complementary to March13N which is the baseline. The basic concept is to check if a bit cell can be written and read as 1 and 0 and if a bit cell remains unaffected by the bits around it. The operation is simple: Firstly, the array is initialized with a known pattern and then a different pattern through the memory is marched. This algorithm detects the following types of errors:

   - Address decoder faults
   - Stuck-At faults
   - Coupled faults
   - State coupling faults
   - Parametric faults
   - Write recovery faults
   - Read/Write logic faults

2. Map Column:

   Line sensitivities of the memory array are detected by the Map Column algorithm. The first row is filled with all 0s, the second with all 1s and repeated through the whole array in this way. Afterwards the values of each column is read on consecutive cycles and the pattern is inverted to proceed with the complementary check. This algorithm is targeting to the detection of the following errors:

   - Leakage due to a low resist path in a bit
   - An Open in the bit cell
   - Leakage on a BIT or BITN line
   - Miss-balance in the sense amp
   - Leakage in the sense

- High resist in the sense amp
- Failure of the pre-charge circuits after read operations

3. Pre-Charge:

   A similar algorithm to Map Column is employed to insure the correct pre-charge capability of the SRAM array. The same pattern with 0s and 1s is used in this algorithm, but a write operation is performed between two reads to force the pre-charge circuits in the array to the worst-case conditions. A failure occurs when an increasing frequency of the system approaches the minimum access time of the SRAM array, at this boundary:

   - High voltage should operate better than low
   - Low temperature should operate better than high

4. DOWN1a:

   DOWN1a is an additional test algorithm for CPU/memory read/write operations that is targeting row/column decode in the SRAM array, the sense amp and sense amp multiplexers, the memory array output buffers and with aggressive writes checks potential at-speed write failures.

5. DTXN2a:

   The DTXN2a algorithm is targeting to insure the correct functionality of the global column decode logic

4.3 Error Correction Code (ECC) on Flash and SRAM

As already mentioned, Error Correction Code (ECC) is employed in the R4F CPUs. In microcontrollers, a potential corruption of the stored data in the memory is possible due to several reasons. Error detection techniques are employed by the semiconductor companies to verify the correctness of the data before or during the execution of the software in the memory modules [8]. Adding extra ECC bits to the original data is a technique that facilitates the error detection insuring the data integrity to the system. The implementation on the TMSx570 devices is capable of detecting up to 2-bit errors and correct single bit errors.

The ECC controllers are located inside each of the CPUs. This placement of the ECC controller provides two basic advantages: The diagnostic covers a potential error detection between the CPU and the memory. Moreover, the ECC logic is tested itself for potential errors as produced faults are detected by the CCM-R4F module.

The Single-Error-Correction Double-Error-Detection (SECDED) ECC logic is a diagnostic for the on-chip SRAM and Flash memory modules. Accesses to the memory modules are protected with this embedded in the CPU logic. 8 bits of ECC code are provided for 64 bits of data (or instructions) fetched from the memory. The expected ECC code is calculated by CPU and compared with the one received from the memory. This logic is capable of correcting a single bit error. Therefore, in case of a single bit error, it is flagged and corrected by the CPU. In case of multi-bit errors, they are only flagged by the CPU and the application has to act appropriately in such a case.

The logic is different for each memory type and depending on the CPU. In case of flash memory, the ECC check bits are programmed along the program data. In RAM modules, the ECC code is generated by hardware with each write operation and is stored in specific ECC memory location. The detection of an error is performed during a read operation where the ECC bits from the special ECC space are read along the data and if it is a single bit error it is corrected directly.

Chapter 5

Lockstep

The term lockstep is used in a variety of circumstances. In the military the marching where the legs and the arms of a drilling soldier were absolutely in sync with the corresponding of the soldier in front of, was called lockstep marching [24]. Additionally, it is also referred to a protocol in multiplayer computer games that is used to solve partially the look-ahead problem. This problem came up when the latency of one client was used as a cheat method, while after this delay she knew the steps of the other players. The lockstep protocol synchronizes the actions of all the clients relatively to the higher latency client [26]. Moreover, a compensation system is called lockstep where for example all the graduates of a law school that are hired by a company receive the same salary, bonuses, promotions, regardless their skills and working experience [25]. Generally, the term in all the aforementioned cases is used to indicate a synchronous or imitating movement. It is usually expressed that something is "in lockstep with ...", to show that it follows the latter.

In this research and in the embedded systems' world, the Lockstep is a safety feature that is integrated from semiconductor companies in families of microcontrollers that are used especially in safety-critical systems. The idea of the concept is the same, but the implementation varies. In this section the Lockstep architecture is described in a general manner, although the implementation, the architecture, the placement of the cores and the details of its usability are specific for the Hercules family of Texas Instruments microcontrollers. Additionally, an alternative implementation of Lockstep by Freescale is presented with significant differences than the one by Texas Instruments. The purpose is to highlight a different way that another leading company in the world of safety-critical systems targets the functional safety of such applications.

5.1 What is Lockstep?

The term "Lockstep" in the field of the embedded systems describes an error-detection mechanism, employed by semiconductor vendor companies that construct systems for safety-critical applications. Safety reliability and functional safety requirements that are prerequisites in avionic and automotive electronic systems, are often satisfied with a hardware redundant architecture [10]. Lockstep is a method to monitor and verify the correctness in the operation of a system [44] by employing at least two processing modules performing identical operations. Typically, the processors are synchronized to the same state during the system start-up. The same tasks are independently processed by the processors operating in lockstep and a real-time comparison of the output signals is performed. In case of discrepancy in the output signals, an error signal is generated and further execution of the task is inhibited. Several mitigation techniques can be employed to recover from the detected error by leading the system to a safe state where potentially dangerous operations cannot be executed [10].

5.2 Reasons to use Lockstep

The lockstep concept is used in safety-relevant systems which require the detection of temporary or permanent errors during the execution of a task or a program. Data corruptions occurring in a processor are more often undetected ("silent" errors) [1].

In the literature, the kind of detected errors by the lockstep architecture are categorized in several different ways. One approach is the soft and hard errors and these terms are used to describe the persistence of the error. Soft errors describe these that occur by a temporary anomaly (i.e Electrostatic Discharge (ESD) hit, noise glitch) and lead to a bit or a logic gate flip in the core. Soft errors may be derived from transient events such as cosmic radiation, radioactive decay, power supply variations or cross talking [29], leading to unintended transient signals or conditions in CPU cores or in the peripherals and their lifetime is typically 2 milliseconds or less. This kind of errors can affect an instruction execution or computation, but the next time that the same operation will be performed the error will not be reproducible. On the other hand, hard errors are permanent and are caused by a long term degradation of the silicon/logic over time. Hard errors may also occur by corrupted memory cells or other circuit components due to ionizing radiation, manufacturing inconsistencies [27], or exposure to high current which may cause metal migration. Generally, they are caused by physical defect of the hardware and they are not recoverable.

In other source literature, the errors that lockstep architecture detects are classified as follows [10]:

1. Permanent faults
2. Intermittent faults
3. Transient faults

In this classification permanent and transient faults are alternative terms to describe the hard and soft errors respectively. However, intermittent faults are a special case that describe errors that occur either periodically, or at irregular intervals under circumstances that several events performed simultaneously. This kind of errors are difficult to detect as all the factors that contribute for their occurrence should be present to reproduce them.

The lockstep architecture is an efficient way to detect all the aforementioned faults and in some cases to isolate and/or correct them using a variety of enhancements that are presented in the following section.

5.3 Lockstep implementation by Freescale

Freescale Semiconductor Inc., one of the leading companies in the world in the construction of safety systems for automotive, industrial and consumer systems has introduced the Lockstep concept into the dual-core MPC564xL MCU family. The safety features of this family is targeting to applications that need to comply with IEC61508 (SIL3) and ISO26262 (ASILD) safety standards [11]. This level of safety is needed for systems such as:

- Electric power steering
- Short- and mid-range adaptive cruise control (up to 100m), RADAR and LIDAR
- Vehicle dynamic and chassis control
- ABS braking systems
- Electronic stability program (ESP)
- Blind spot detection
- Pre-crash detection

- Hybrid electric vehicles

The implementation of this MCU is dual-core, dual-issue (To increase instruction throughput, the processor can issue certain pairs of instructions simultaneously - two instructions per clock cycle) [6]. Instead of using two MCUs with a primary core executing the software and a checker core executing some safety diagnostic software, MPC564xL family contains two "channels", each consisting of a core, bus, interrupt controller, memory controller and other core-related modules.
This MCU can be pre-configured to operate in two distinct modes:

1. Decoupled parallel mode (DPM) - independent core operation
2. Lockstep mode (LSM) - parallel core operation

Depending on the nature of the application, the developer has to employ the right operating mode. The trade-off between performance and software transparency has to be considered under the choice of the mode. In LSM, the two cores result in the performance of one, while high diagnostic coverage and short detection intervals without software interaction is achieved. On the other hand, in DPM, the two channels work independently, providing higher performance, but lacking of diagnostic coverage without software interaction at the hardware level [3].

More specifically, the redundancy in the architecture of MPC564xL family of freescale is extended to more modules than only the core. The term in the safety manual of the MPC5643LSM is called "Sphere of Replication" (SoR) and stands for the replication of the following components in order to detect permanent, dormant, latent, and transient faults:

- e200z4 core (including Memory Management Unit)
- Enhanced Direct Memory Access (eDMA)
- Interrupt Controller (INTC)
- Crossbar Switch (XBAR)
- Memory Protection Unit (MPU)
- Flash Memory Controller (PFlashC)
- Static RAM Controller (SRAMC)
- System Timer Module (STM)
- Software Watchdog Timer (SWT)
- Peripheral Bridge (PBRIDGE)

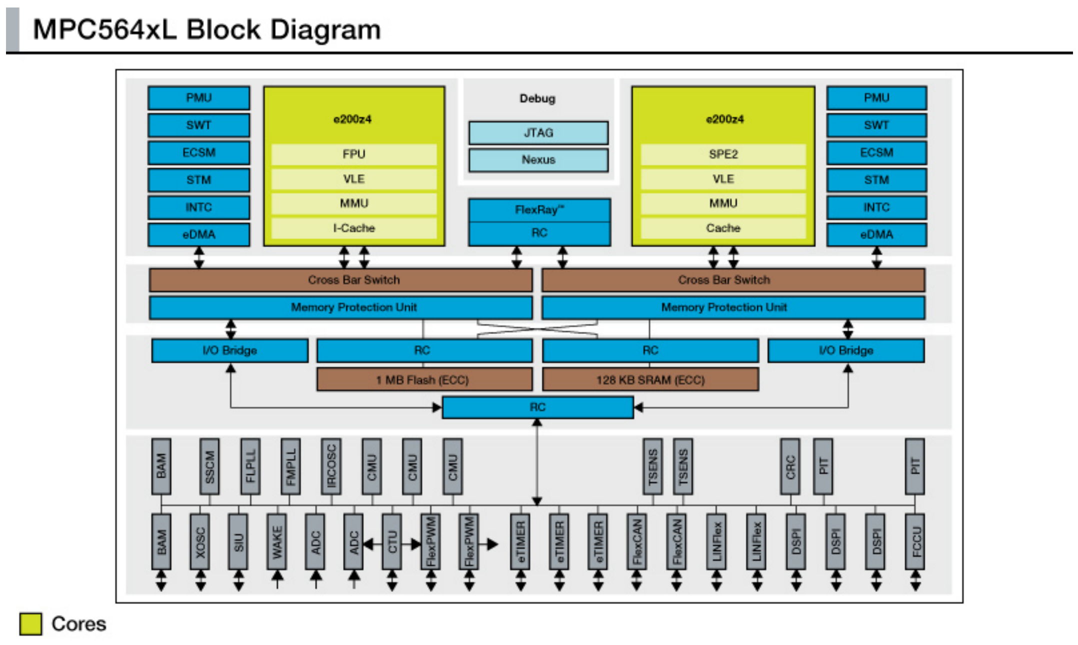


Figure 5.1: MPC564xL Block Diagram [12]

In LSM mode all the aforementioned components operate the same operations and transactions and the supervisor module for the smooth and equal execution in sync is called Redundancy Control Checker Units (RCCU). The role of the RCCUs is the detection but not the prevention of failures at the point where the outputs of the modules in sync are merged. The isolation of the fault is performed with an error signaling by the Fault Collection and Control Unit (FCCU), which therefore allows the device or the application to perform the appropriate operations.

Nevertheless, in the current research the main concern is the replication of the core that is a common point between the various semiconductor companies and the purpose is to reveal how this redundancy contributes in cooperation with the other safety features that each microcontroller provides, to comply with the standards for the safety-critical systems.

5.4 Lockstep implementation on TMS570LS31x/21x by Texas Instruments

In this section we describe the lockstep implementation on TMS570LS31x/21x of the Hercules family of microcontrollers. The current research was based on a Texas Instruments TMS570LS3137 Hercules Development Kit and focused on the implementation, the architecture and the capabilities that are presented in this section.

Safety-critical systems require a run-time detection of errors that are derived from a faulty behavior on the processor level. Two instances of Cortex-R4F processors operate in a delayed lockstep (see Section 6.1) in order to prevent errors that may lead to unsafe operating conditions. The CPU Compare Module for Cortex-R4F (CCM-R4F) is employed in the Texas Instruments architecture to compare the output signals of the two processors that run in a 1oo1D (one-out-of-one, with diagnostics). In case of a mismatch, an error is signaled to the Error Signaling Module. Furthermore, a self-test method for the CCM-R4F module is applied at boot time to detect any potential hardware errors on the module itself that could lead the module to an unintended behavior. The three main characteristics of the CCM-R4F module are [42]:

- Run-time detection for errors
- Self-test capability
- Error-forcing capability

5.4.1 Architecture and CCM-R4F module operation

The lockstep architecture and the interconnection diagram of the CCM-R4F module is depicted in Figure 5.2. Two Cortex-R4F CPUs are fed with identical input and their outputs are driven to the CCM-R4F for a match check. There is a delay of 2 cycles before the entrance of the input signal to the Checker CPU and an equal amount delay is coupled to the output of the Master CPU after it is propagated to the system. This delay serves temporal diversity purposes that are described in Section 5.4.6. In case of a mismatch in the comparison of the outputs from master and checker CPU, a compare error signal is generated to notify the Error Signaling Module. After a reset, 6 CPU clock cycles are required till the start of the comparison, in order to ensure that CPU output signals have been set to a known state [42].

The CCM-R4F can operate in four distinct operating modes:

1. 1oo1D lockstep
2. self-test
3. error forcing
4. self-test error forcing

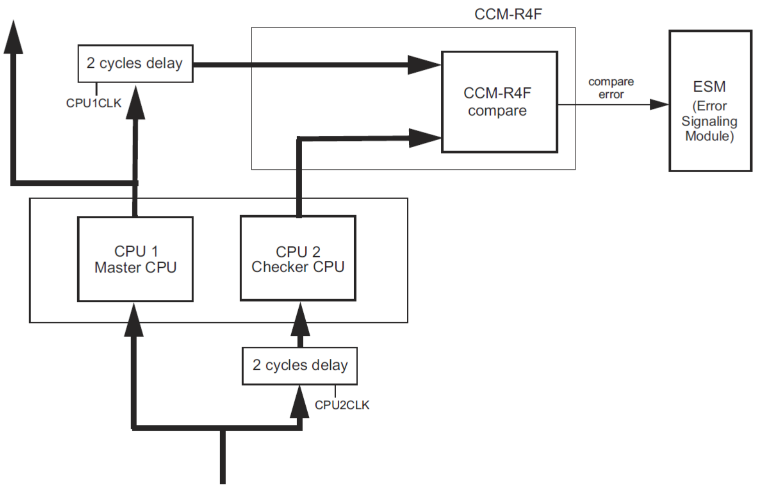


Figure 5.2: Interconnection diagram of CCM-R4F [35]

5.4.2 1oo1D Lockstep Mode

The 1oo1D is the default mode of the module after a start-up, where the aforementioned comparison of the output signals from master and checker CPUs are compared and an error flag is raised to the ESM. Upon a reset, the values of internal registers for the Cortex-R4F CPUs may vary. Therefore, to avoid an erroneous computation in comparison, the application software must initialize registers of both master and checker CPU with identical values, before they are used. In this operating mode, the CPU compare error module generates two signals as a safety mechanism to interface with the ESM: in case of a mismatch both "CCM-R4F - compare" and "CCM-R4F self-test error" flags are raised to the ESM to ensure that the error is detected even if one of the paths fail.

5.4.3 Self-test mode

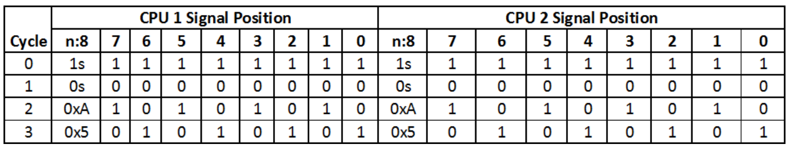
In self-test mode, test patterns are automatically generated by the CCM-R4F to determine its correct functionality. In case of an error detection that indicates a hardware fault on the module itself, a "CCM-R4F - selftest" flag will be raised to the ESM. After the completion or termination of the self-test, if no error occurred, the self-test complete flag is set to notify the system to proceed appropriately. Moreover, after the completion of this integrity test, the module remains in the self-test mode until the appropriate register (MKEY: Mode Key Register) is set to indicate the operation in a different mode. The following two tests are generated by CCM-R4F in this order, to verify its proper functioning:

- Compare Match Test
- Compare Mismatch Test

Each of these test patterns is applied on both of the inputs of CCM-R4F and clocked for one cycle with a total duration of the self-test at 3615 CPU clock cycles. During the self-test which is a non-interruptible task, both CPUs can work independently of the test execution, but the comparison of their output signals is skipped while the module is busy with its self diagnostic test. Furthermore, the self-test checks only the integrity of the CCM-R4F module and not its memory mapped register controls.

Compare Match Test

Four different test patterns are fed to the CCM-R4F in the compare match test. These patterns contain identical vectors that are fed to both of the input ports of the module simultaneously to check for output equality. In case of discrepancy, the test is terminated, the self-test error flag is set and the self-test error signal is generated. The table in Figure 5.3 depicts the vectors that fed to the input ports:



	CPU 1 Signal Position									CPU 2 Signal Position								
Cycle	n:8	7	6	5	4	3	2	1	0	n:8	7	6	5	4	3	2	1	0
0	1s	1	1	1	1	1	1	1	1	1s	1	1	1	1	1	1	1	1
1	0s	0	0	0	0	0	0	0	0	0s	0	0	0	0	0	0	0	0
2	0xA	1	0	1	0	1	0	1	0	0xA	1	0	1	0	1	0	1	0
3	0x5	0	1	0	1	0	1	0	1	0x5	0	1	0	1	0	1	0	1

Figure 5.3: Compare Match Test vectors [42]

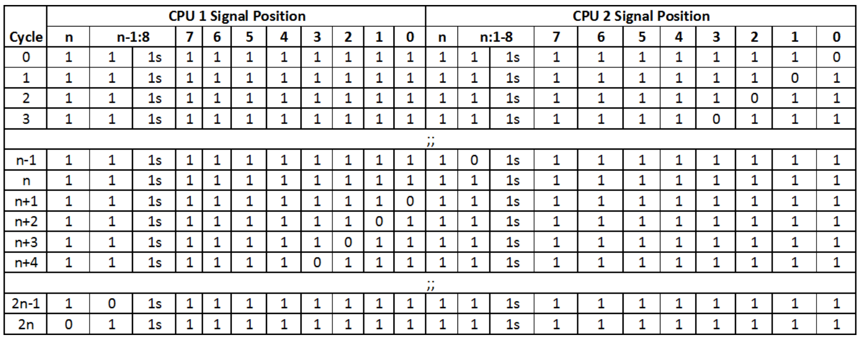
The duration of this test is 4 cycles where:

- 1st cycle: both CPU signal ports are fed with 1s
- 2nd cycle: both CPU signal ports are fed with 0s
- 3rd cycle: both CPU signal ports are fed with As
- 4th cycle: both CPU signal ports are fed with 5s

Compare Mismatch Test

In the compare mismatch test, patterns with one different bit are applied to the input ports of the CCM-R4F module (see Figure 5.4). For instance, when a signal with all 1s is applied to the first line, a sequence of signals are applied to the second, starting by flipping the least significant bit and on (Cycle 0,1,2,3, ...). The comparison of the unequal vectors is expected to generate a mismatch in the proportional bit (i.e bit 0 in cycle 0, bit 1 in cycle 1, ...) to indicate a correct functionality without though propagating an error signal to the ESM.

In case that a match is detected in any of the unequal comparisons, the self-test error flag is set, the self-test error signal is raised and the test is terminated, revealing an unintended behavior of the module.



	CPU 1 Signal Position												CPU 2 Signal Position										
Cycle	n	n-1:8		7	6	5	4	3	2	1	0	n	n:1-8		7	6	5	4	3	2	1	0
0	1	1	1s	1	1	1	1	1	1	1	1	1	1	1s	1	1	1	1	1	1	1	0
1	1	1	1s	1	1	1	1	1	1	1	1	1	1	1s	1	1	1	1	1	1	0	1
2	1	1	1s	1	1	1	1	1	1	1	1	1	1	1s	1	1	1	1	1	0	1	1
3	1	1	1s	1	1	1	1	1	1	1	1	1	1	1s	1	1	1	1	0	1	1	1
												;;										
n-1	1	1	1s	1	1	1	1	1	1	1	1	1	0	1s	1	1	1	1	1	1	1	1
n	1	1	1s	1	1	1	1	1	1	1	1	1	1	1s	1	1	1	1	1	1	1	1
n+1	1	1	1s	1	1	1	1	1	1	1	0	1	1	1s	1	1	1	1	1	1	1	1
n+2	1	1	1s	1	1	1	1	1	1	0	1	1	1	1s	1	1	1	1	1	1	1	1
n+3	1	1	1s	1	1	1	1	1	0	1	1	1	1	1s	1	1	1	1	1	1	1	1
n+4	1	1	1s	1	1	1	1	0	1	1	1	1	1	1s	1	1	1	1	1	1	1	1
												;;										
2n-1	1	0	1s	1	1	1	1	1	1	1	1	1	1	1s	1	1	1	1	1	1	1	1
2n	0	1	1s	1	1	1	1	1	1	1	1	1	1	1s	1	1	1	1	1	1	1	1

Figure 5.4: Compare Mismatch Test Vectors [42]

5.4.4 Error Forcing Mode

In error-forcing module, two non-identical signals are forced to the input ports of the CCM-R4F module. More specifically, a hardcoded test pattern with a value of 0x5 is applied to the first input and 0xA to the second input respectively, expecting a "CCM-R4F compare" error to be propagated to the ESM. As is described in the 1oo1D Lockstep mode (see section 5.4.2) the "CCM-R4F self-test" is also raised as a safety mechanism in case of a path failure of the "CCM-R4F compare" signal.

This test ensures that an error between the CCM-R4F and ESM path is successfully detected. If a fault is not detected in this mode, a hardware failure is present. The duration of this mode is 1 CPU clock cycle and after its completion the mode is automatically switched to 1oo1D Lockstep mode.

5.4.5 Self-Test Error Forcing Mode

In this mode an error is injected during the self-test of the module. In a normal operation of the module, a "CCM-R4F self-test" flag is expected to be raised to ESM. The error can be cleared through the application after the right operation of the mode is verified.

5.4.6 Mitigation of common cause failures in CPU

In order to prevent common cause errors that can affect both of the pair of the CPU cores, Texas Instruments employs some special techniques that at least can reduce this possibility of simultaneously identical error appearance in both of the cores. Temporal and physical diversity are practices to mitigate this common mode failure and the details of their implementation on the TMS570LS31x/21x family of TI microcontrollers are described in this section [35].

Temporal diversity

The first measure that is already presented in the Texas Instruments patent (see Section 6.1) is related to timing and guarantees a temporal diversity between the cores. The term "delayed lockstep" is used to describe the Lockstep implementation with this timing enhancement.

The clock input to one of the cores is delayed 2 cycles on the input side while the output from the other is delayed 2 cycles on the output (see Figure 5.2). Therefore, each core is executing a different operation at any given point in time. This way, if there is a disturbance that might result in errant operation of the core such as radiation, a voltage glitch, a voltage dip, etc., both cores will, again, fail in different ways since they are executing different instructions at the point of the disturbance.

Physical diversity

Another consideration that is given to the design are placement/orientation of the cores on the silicon. The 2 cores are rotated 90 degrees and flipped in relation to each other with at least 100 um space in between. This is to give a physical diversity. Physical diversity gives some assurance of a different failure mechanism if there is a manufacturing flaw causing physical damage to the die.

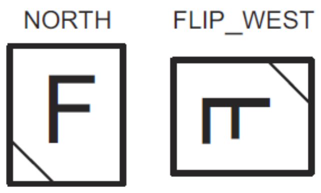


Figure 5.5: Example of orientation of the cores [35]

In Figure 5.5 a placement example of two cores on the silicon is illustrated where the one is north-oriented and the other flipped-west. The common cause failures that physical diversity is called to restrict might be due to manufacturing flaws or even susceptibility to radiation due to the direction of some of the components used in the core design. A simple example for a potential manufacturing flaw that might be common to both cores without the physical diversity, suppose that there is a geometry of a certain component that was prone to collecting moisture and it was processed in a way that this component was traveling parallel to some chemical spray in the process. This might lead to trapped materials at this component in the silicon but if the components were rotated and flipped it would make the likelihood of trapping contaminates less likely. In this way, making the two cores physically diverse can keep both cores from having the same defective component.

5.4.7 Lockstep behavior in CPU Debug mode

During a debug session, where the execution of the code is halted, several asynchronous halting events occur, with a possibility to lead to a loss of Lockstep (indication of non-existing errors). For this reason, when halting debug events are detected, the CCM-R4F is automatically disabled. In order to set the Lockstep mode on and the CCM-R4F in operational state, a CPU reset is required [42].

5.4.8 Error Signaling Module (ESM)

A brief overview of the Error Signaling Module is required in this chapter as it is strongly related to the CCM-R4F module. The ESM collects and categorize the errors that are derived by the hardware diagnostic tools, regarding their severity. Low priority interrupt, high priority interrupt or an external pin action are potential responses to the hardware detected errors depending on their severity level. The severity levels are divided into 3 groups, supporting up to 128 error channels in total [42]:

Group	No. of Channels	Severity Level	Interrupt Generation	ERROR pin Behavior
Group 1	64	Low Severity	Configurable	Configurable
Group 2	32	High Severity	Predefined	Predefined
Group 3	32	High Severity	No Interrupt Generation	Predefined

Table 5.1: Groups of errors in ESM

The errors belonging to Group 1, indicating low severity, have a configurable interrupt response and ERROR pin behavior. The Group 2 are considered as high severity errors and an interrupt is always generated as well as the output on the ERROR pin. Finally, the detection of the errors in Group 3, are high severity errors that have already generated a CPU abort response and therefore they do not generate an interrupt. However, they always generate an ERROR pin output.

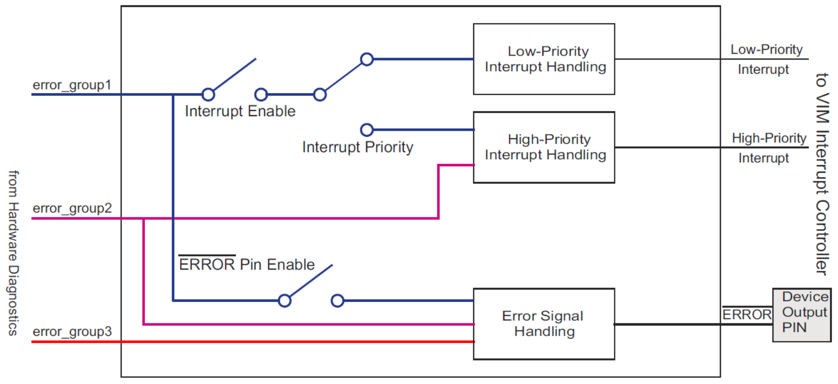


Figure 5.6: ESM Block Diagram [42]

As already mentioned in the different operating modes of the CCM-R4F, all errors that are related to this module fall into a Group2 category in the ESM. This means that the interrupt and error pin behavior is fixed. i.e., an interrupt will always be generated and the ERROR pin will be asserted.

5.5 Lockstep implementation on TMS570LC43x by Texas Instruments

TMS570LC43x is currently the most recent microcontroller of the Hercules family. The fist major difference is that the core has been upgraded to Cortex-R5F which provides some enhancements to improve the performance of the device. The main advantage of this migration to the Cortex-R5F is a lower latency in accessing peripherals and a hardware support for flushing lines in cache that have been modified in the main memory. The Hercules products with an integrated Cortex-R4F core are built with only tightly coupled memory and by default no cache (i.e cache memory is an option in these devices). With the integration of cache into the recent microcontrollers a higher frequency is achieved [38].

The current research has focused primarily on the systems with Cortex-R4F CPU cores. This section aims to emerge the main differences of the Lockstep implementation in the most recent microcontoller of Texas Instruments that is targeting to safety-critical systems with the Lockstep implementation discussed in section 5.4.

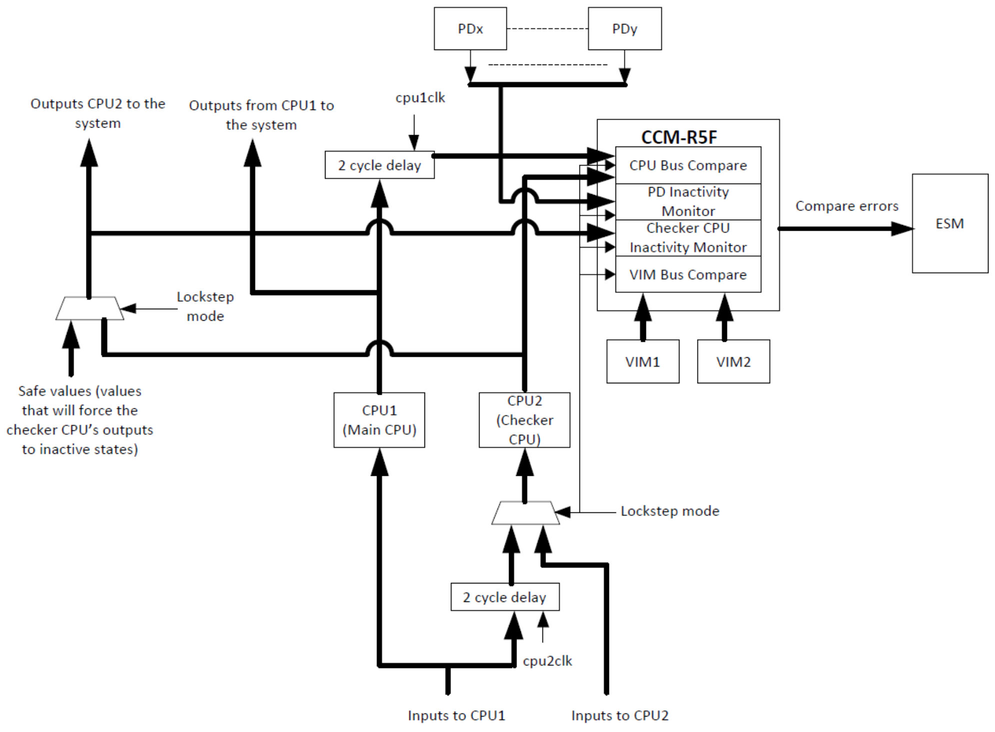


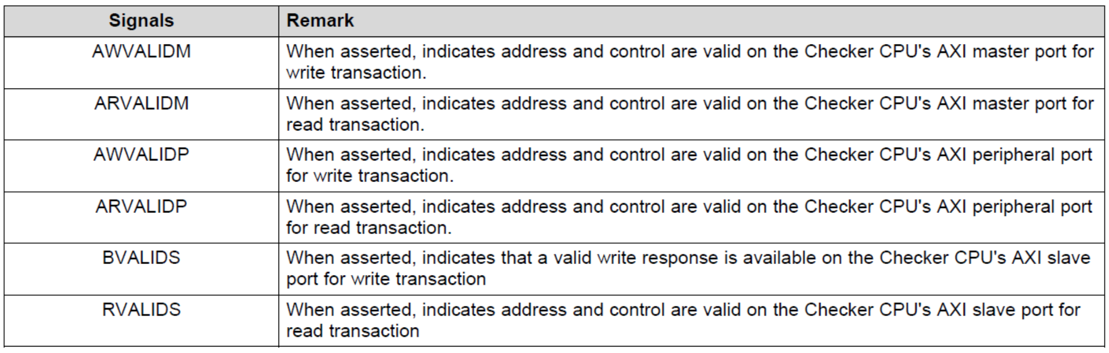
Figure 5.7: Block Diagram of CCM-R5F [40]

Regarding the comparison of the CPU outputs the concept is the same with the implementation of TMS570LS31x/21x microcontrollers. A 2 cycles delay is applied to the output of the Main CPU and an identical amount of delay to the input of the Checker CPU. Before the delay stage, the output of the Master CPU propagates to the system while after the delay stage is fed to the CPU Compare Module for Cortex-R5F. CCM-R5F module receives the output signals of both CPUs and checks for a mismatch. In case of a discrepancy, a compare error signal is raised to the ESM module (see Figure 5.7).

The new lockstep-related feature of this microcontroller, is the implementation of the VIM (see Section 5.5.1) in Lockstep mode. Two VIMs are running in lockstep and their outputs are fed for comparison to the CCM-R5F. In case of a mismatch a compare error is signaled to the ESM in the same way as with a core compare error. Additionally two new run-time diagnostic features are integrated [40]:

- Checker CPU Inactivity Monitor
- Power Domain (PD) Inactivity Monitor

The former measure monitors the key bus signals of the Checker CPU to the interconnect. The functionality of the Checker CPU is restricted to the execution of the same instructions with the Master CPU and propagation of the output signal to the CCM-R5F module. It is prohibited to interact with the rest of the system, while only the Master CPU is allowed to perform the transactions with memory and peripherals. For this reason a number of key bus signals are monitored and must be inactive. If an activity is indicated on them, an unintended interaction of the Checker CPU occurred to the system. In case of detection of activity in any of these signals (see Figure 5.8) is flagged as an error.



Signals	Remark
AWVALIDM	When asserted, indicates address and control are valid on the Checker CPU's AXI master port for write transaction.
ARVALIDM	When asserted, indicates address and control are valid on the Checker CPU's AXI master port for read transaction.
AWVALIDP	When asserted, indicates address and control are valid on the Checker CPU's AXI peripheral port for write transaction.
ARVALIDP	When asserted, indicates address and control are valid on the Checker CPU's AXI peripheral port for read transaction.
BVALIDS	When asserted, indicates that a valid write response is available on the Checker CPU's AXI slave port for write transaction
RVALIDS	When asserted, indicates address and control are valid on the Checker CPU's AXI slave port for read transaction

Figure 5.8: List of the signals that must be inactive in Checker CPU [40]

The latter diagnostic follows the same principle with the Checker CPU Inactivity Monitor, but in terms of power domains. The Power Domain Inactivity Monitor monitors the key bus signals of the inactive power domains that in a correct operation must be inactive and isolated from the rest of the system. Any activity of these signals indicates a transaction onto the interconnect which is an unintended behavior (in case of a turned of power domain) and an error is flagged.

5.5.1 Vectored Interrupt Manager (VIM)

As aforementioned the Vectored Interrupt Manager (VIM) Module is operating in lockstep in the same manner that the CPU cores do. VIM is the responsible module for controlling and prioritizing the several interrupt sources that may occur during the program flow. It is the same module for interrupt handling that is used on TMS570Ls31x/21x microcontrollers, but it is enhanced with a redundant one to operate in lockstep for increasing safety. Temporal diversity is also supported with a delay stage of 2 CPU clock cycles (same as in the delayed-lockstep architecture of the CPU cores) [40].

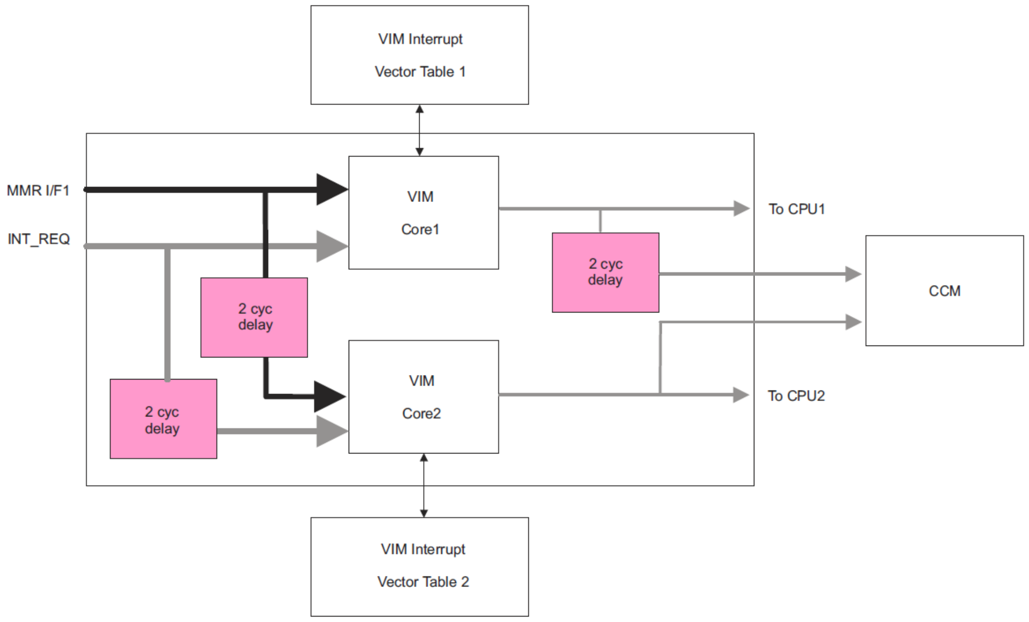


Figure 5.9: Dual VIM for Safety [40]

The scope of this research is restricted to the lockstep operation. For more information about the kinds and the priorities of the generated interrupts and the interrupt handling in the Hercules family of microcontrollers someone can refer to the Technical Reference Manual (TRM) of the specific microcontroller.

Chapter 6

Lockstep Related Patents

Various semiconductor companies have settled patents for distinct lockstep architectures and several features for enhancement of its functionality and capabilities. In this chapter the lockstep related patents are presented, emphasizing on the disparity and the special characteristics that each invention has to demonstrate.

6.1 Delayed lockstep CPU compare (Texas Instruments)

The lockstep invention of Texas Instruments addresses a dual CPU architecture microcontroller where the outputs of the CPUs are compared [29]. The two CPUs execute the same program code, where a delay stage is applied to the output bus of the first (CPU1) and a second delay stage is coupled to the input of the second CPU (CPU2). The CPU compare unit is responsible for the comparison of the two signals: the delayed output of the CPU1 and the output of the CPU2. The amount of delay stages must be predetermined and equal and it can be a number of clock cycles or fractions of clock cycles of the system clock. The signals that are compared in the CPU compare unit are data that belong to the same operation of the program code, while the time shift is compensated before reaching the comparator.

The slight time difference in the execution of an instruction by the two CPUs (referred as execution in a delayed lockstep) ensure the detection of a common cause errors such as a short voltage drop or a glitch in the clock signal. Practical implementations have shown that the appropriate delay to detect the majority of the common cause errors is between 0.5 and 2 cycles of the system clock. The CPU compare unit is adapted to report a match or mismatch in the comparison of the two signals. In case of a mismatch, the system should handle the error detection appropriately.

The system performance remains unaffected while the output of the CPU1 is directly fed to the system before experiencing the delay stage and reach the CPU compare unit. The role of the CPU2 is to feed the CPU compare unit with a second signal that under normal conditions (without an error detection) should be identical to the delayed output signal of the CPU1. Therefore, the output of the CPU2 is not propagated to the system and it does not affect the internal states of memories or registers.

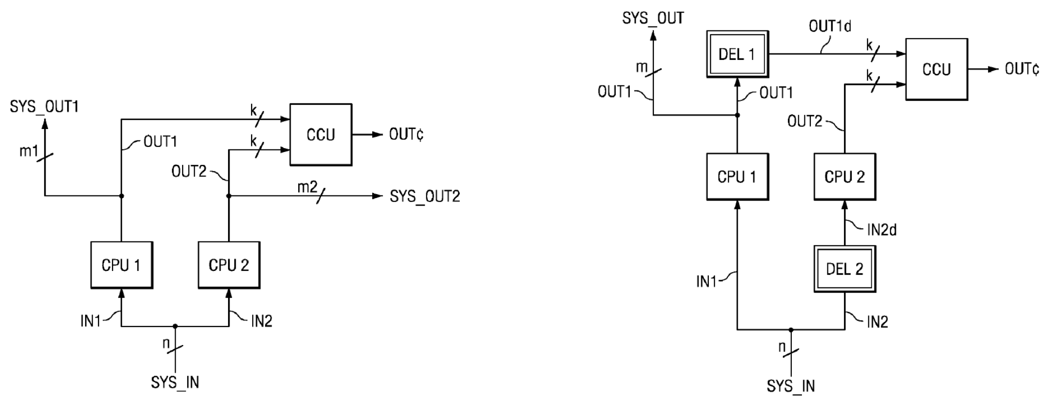


Figure 6.1: Prior art and Texas Instruments invention [29]

In the prior art as shown in Figure 6.1 on the left side, the CPU1 and CPU2 are executing the same program code that are coupled via the input SYS_IN line at the same time.

The output signals of the CPUs (OUT1, OUT2) are fed to the CCU (CPU Compare Unit) and they are checked for match or mismatch. Moreover, they are propagated to the system via the output buses SYS_OUT1 and SYS_OUT2. On the right side of Figure 6.1 is depicted the current invention of Texas Instruments. The input bus SYS_IN is directly connected to the CPU1 that is known as the master CPU while a delay stage DEL2 interferes before feeding the checker CPU (CPU2) with the input signal. The output of the CPU1, OUT1, is propagated directly to the system via the SYS_OUT bus, but it is also coupled to a delay stage DEL1 which should be equal with the delay stage used in the CPU2 (DEL2). The delayed output of CPU1, OUT1d and the output of the CPU2 are fed to CCU to be checked if they are identical. In case of discrepancy the system is noticed via the compare output OUTc. Regarding the read and write operations, both CPUs read data from the common system memory, but only the master CPU is permitted to write and modify the system state.

6.2 Error detection and communication of an error location in multi-processor data processing system having processors operating in lockstep (Freescale Semiconductor Inc.)

The current invention of Freescale Semiconductor Inc. facilitates the prevention of a wrong lockstep operation due to some internal core errors. More specifically, some internal errors, such as soft errors in a cache are not replicated to both of the cores that operate in lockstep. As a result, the behavior of each core will be different, leading the cores to fall out of lockstep [47].

The patent is referring to multi-core systems (>= 2 cores) that operate in lockstep. The principle is that when a soft error is detected internally and an exception is triggered, the other cores are forced to enter in the same altered state. This imitation of the behavior of the first core from the others is achieved with an error-signaling interface. It receives information regarding the altered internal state from the core that experiences it and provides this information to the other cores that they have to emulate the same altered state. The information that is propagated may be the type and the location of the error, facilitating to distinguish it from different predetermined types of errors.

When a machine check exception is generated in one core internally indicating a hardware failure, the cross-signaling interface forces all the other cores to generate the same exception. Potential causes of such an exception could be a parity error, a decode error, logic error, and single and multi-bit errors discovered using error correction code (ECC), parity, or other error detection mechanisms. An alternative implementation to the machine check exception could be a cache control operation like a "miss" condition to be forced by the error-signaling interface where all the cores perform an auto-invalidation of the cache location and the location of the cache is reloaded. In case of simultaneous errors, a logic placed in the error-signaling interface can decide which errors will be imitated by the other cores or which cores should be reset.

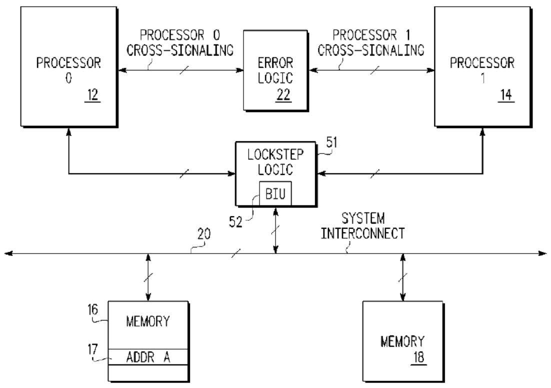


Figure 6.2: Error-signaling Interface in a lockstep architecture by Freescale [47]

A simplified model of the current patent is illustrated in Figure 6.2. The system includes two substantially identical processors operating in lockstep, two memories that are bi-directional connected to the system interconnect and may be static random access memory (SRAM) or dynamic random access memory (DRAM) or flash, an error logic coupled in between of the two processors which bi-directionally sends and receives control signals to the processors and the lockstep logic including a bus interface unit (BIU) that is employed for the connection of the lockstep logic to system interconnect.

The lockstep logic includes the comparator logic that compares the outputs of the two (or more) processors and reports any potential error indicating that the processors do not operate in lockstep mode anymore. Furthermore, it may acts as a coordinator to couple the non-failed processor to the system interconnect via BIU or trigger an exception or directly reset one or both processors.

The current embodiment is referring to lockstep operation of the processors, where both execute the same program code simultaneously, accessing the same addresses and anticipating identical outputs. A read request accesses the same data, saving the content in their respective cache memories exactly on the same clock cycle and in the same location. On the other hand, they can also operate independently executing different instructions and operating as a multi-core system.

To conclude, this patent presents a solution in the case of an internal soft error in one of the processors that operate in lockstep. In case of an exception signaling in one of the processors as a response for example in a cache error, the processor will execute a different set of instructions (i.e interrupt service routine). The simultaneous occurrence of such an error in both of the processors is rare and as a result the lockstep operation will fall out of the correct functionality. The imitation of the erroneous state in the processor that operates without experiencing the error through an error signaling interface in order to keep all the utilized processors in a sync manner, is the proposal of Freescale that is analyzed in this patent.

6.3 Method and system for fault containment (Infineon Technologies AG)

The current invention by Infineon Technologies AG. enhances the prior art delayed lockstep implementation with a method to prevent the propagation of the detected error to the system [2].

As shown in Figure 6.3 on the upper side, which is substantially the implementation of Texas Instruments delayed lockstep mechanism (see Section 6.1), an error is detected after the amount of delay unit2 (usually equal to delay unit1 and amounts between 0.5 and 2.5 clock cycles) when the output signals of the CPUs are checked in the comparator unit and in case of a mismatch the system is getting noticed. As a result, till the detection of the error it is possible that erroneous data in a write transaction are propagated to a system interconnect or a local SRAM. The embodiments of this patent are targeting to the prevention or containment to this error propagation to the system.

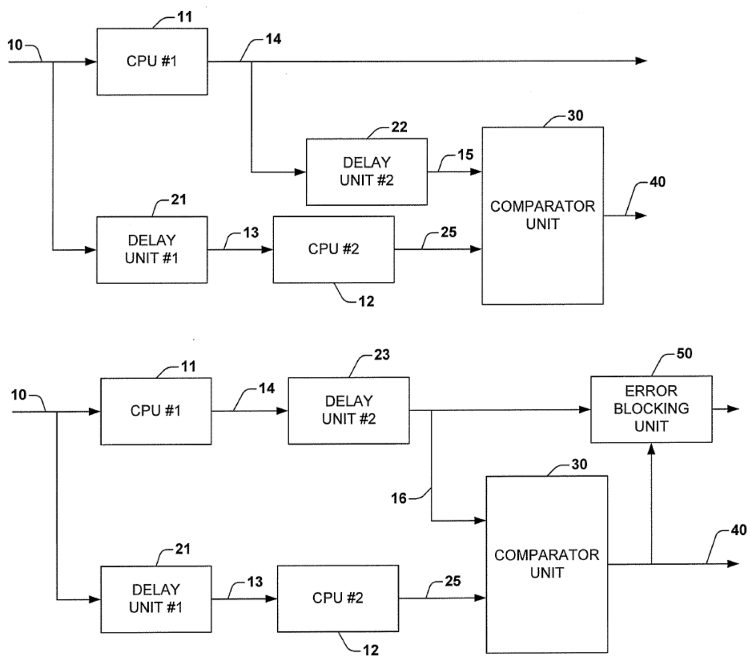


Figure 6.3: Prior art delayed lockstep implementation and Infineon's enhancement [2]

In a first approach for corruption protected delayed lockstep CPUs, a method is utilized as is demonstrated on the bottom of Figure 6.3. A further delay unit2 is placed directly to the output signal of CPU1 and this delayed version is fed to the comparator unit for a match check with the output of the CPU2 before it is fed to the system in subsequent stages. In this regard, the comparator unit has the time to generate an error signal in case of a mismatch, before the corrupted data propagate to the system. In this way, the delay unit2 may delay a signal that is related to a write transaction and prevent an erroneous write operation. For this purpose, the error blocking unit is employed and with this architecture write transactions can be blocked or totally be aborted. The corruption of the subsequent stages of the system can also be prevented by operating these faulty write transaction to known memory ranges while part of the system will still be functional.

The invention suggests different implementations of the delay unit2: instead of delaying all the signals, only the write operations may be delayed or only a category of write transactions, i.e write operations to peripherals and to a memory attached to a shared internal bus.

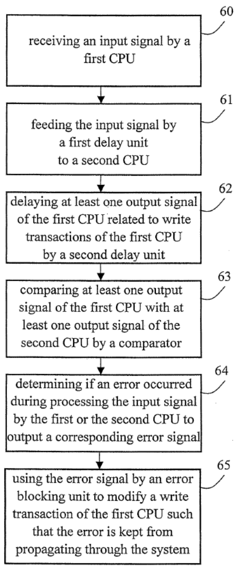

60
receiving an input signal by a first CPU
61
feeding the input signal by a first delay unit to a second CPU
62
delaying at least one output signal of the first CPU related to write transactions of the first CPU by a second delay unit
63
comparing at least one output signal of the first CPU with at least one output signal of the second CPU by a comparator
64
determining if an error occurred during processing the input signal by the first or the second CPU to output a corresponding error signal
65
using the error signal by an error blocking unit to modify a write transaction of the first CPU such that the error is kept from propagating through the system

Figure 6.4: Flow diagram of the Infineon's invention for avoiding error propagation in multi-CPU systems [2]

Variety of implementations are also proposed for the error blocking unit (alternative methods for step 65 in Figure 6.4): The first that is already mentioned is to block a write transaction that is performed by the first CPU. A second case may be to let a write transaction to propagate to the system and in case of a mismatch in the comparator unit to send a command to abort it. Moreover it may modify the address of the write transaction in a predetermined location that is considered as a "safe" storage for this purpose. Finally, an implementation could be to utilize error detection/correction code in a write operation and an abort will be generated by the error blocking unit by corrupting the address signature of the write transaction.

Another approach is presented by employing two extra comparator units to verify said protection condition that the output of the comparator unit1 (error signal) and the output of the comparator unit2 (negated error signal) correspondingly agree.

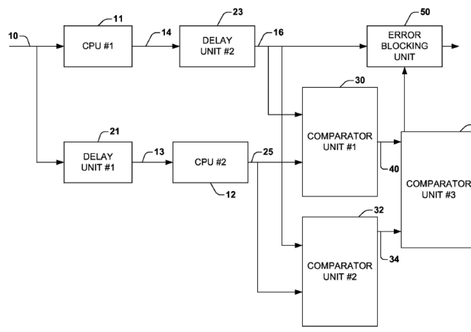


Figure 6.5: Two extra comparator units for comparison of the error and the negated error signals [2]

One issue that is mentioned in the current patent is the protection of the error blocking unit. Specifically, to integrate a safety mechanism to resolve a potential dangerous behavior of the error blocking unit where it could generate an error signal in absence of a lockstep error. The full compliance with the ISO26262 standard requires the protection of this unit, without analyzing though a specific method for this enhancement.

6.4 Method and apparatus for recovery from loss of lock step (Hewlett-Packard Development Company, L.P)

Hewlett-Packard demonstrates a method for recovery from loss of lockstep [23]. In case of a delayed correction of this loss, the whole system may crash. In a dual-core system, in case of one failed processor the whole system may halt processing if the other core is still full functional. As was already discussed in Section 6.2 that is described the proposal of Freescale for the same issue, the loss of lockstep may occur from a data cache error. A prior art is also presented for the loss of lockstep (see Figure 6.6) where in case of malfunction of the one of the cores, the architected state of the other core (considered as "good") is saved to the memory. Then, both of the cores are reset and reinitialized and the architected state is copied from the memory.

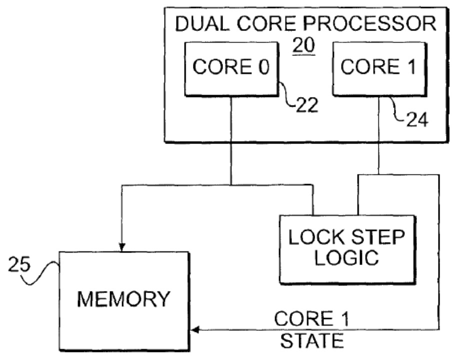


Figure 6.6: Prior art for prevention of loss of lockstep [23]

This prior art method is not efficient as it makes the cores unavailable for an amount of time with burden on the performance of the system. Additionally, if this amount of time that is required to recover from the loss of lockstep is long, it may lead to the crash of the whole computer system.

The invention of Hewlett-Packard employs an apparatus comprising of at least two processor units operating in lockstep, at least one idle processing unit and a controller unit that is capable of copying an architected state of an operational processor unit to the idle one. The processor unit that experiences the loss of lockstep generates a signal and after the transfer of its architected state to the idle processing unit, it goes offline.

In Figure 6.8 is depicted a system that contains three pairs of CPUs, each of them operating in lockstep (CPU0 in lockstep with CPU1, CPU5 in lockstep with CPU4, CPU2 in lockstep with CPU3). In lockstep mode, each pair is connected to a lockstep logic (see Figure 6.7):

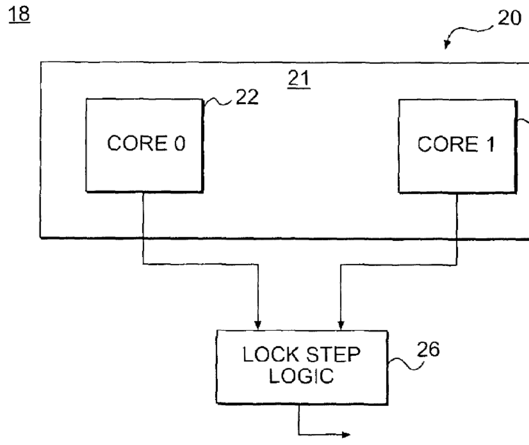


Figure 6.7: Pair of processors in lockstep mode connected to a lockstep logic [23]

The node controller is coupled in between and from its perspective each pair appears as a single processor. In lockstep mode the pairs are first connected to a lockstep logic (Figure 6.7) and then to the node controller (Figure 6.8). In one implementation the node controller is aware of the architected state of all the pairs that are connected to it. Another case may be that the node controller is used just as means for communication between the processors. In this way, it could store internally or to an external component of the system the architected state of the processors. A further scenario could be that the node controller will facilitate the storage of its architected state to another processor of the system. Finally, one processor could have the role "hot standby" that will be idle and in case of a failure (loss of lockstep) on another processor, it could recover it by adapting its architected state with the help of the node controller.

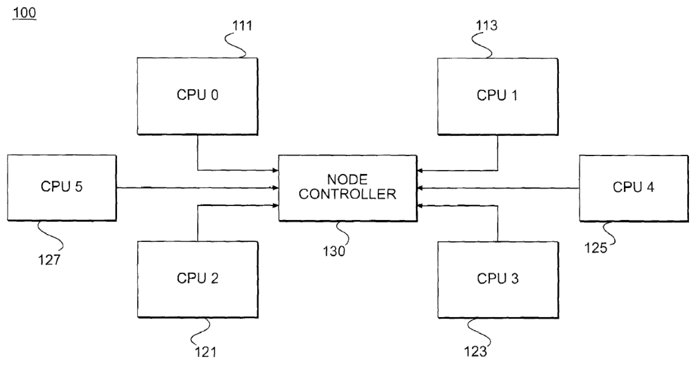


Figure 6.8: Architectural details for recovery from loss of lockstep [23]

A flow diagram for the recovery of lockstep via the "hot standby" processor is demonstrated in Figure 6.9. An example case would be a process depicted in the first step of the flow diagram is executed in lockstep on the pair of CPU0 and CPU1 of the Figure 6.8. An error event is detected by CPU0 indicating an impending loss of lockstep and it signals the node controller for this failure. The node controller copies the architected state of the pair CPU0/CPU1 to the idle pair CPU2/CPU3. At this point, the system operates without the existence of a hot standby pair until the appropriate actions execute on the first "infected" pair (i.e all caches are flushed on the CPU0/CPU1). Finally, the node controller reboots the CPU0/CPU1 pair of processors and set them as the new hot/standby pair.

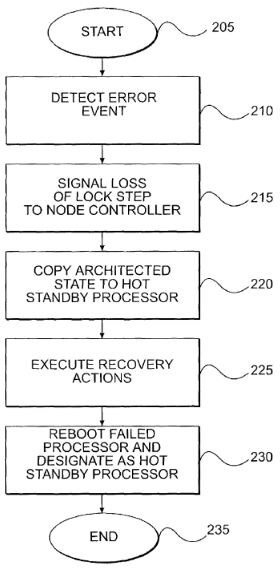

START
205
DETECT ERROR EVENT
210
SIGNAL LOSS OF LOCK STEP TO NODE CONTROLLER
215
COPY ARCHITECTED STATE TO HOT STANDBY PROCESSOR
220
EXECUTE RECOVERY ACTIONS
225
REBOOT FAILED PROCESSOR AND DESIGNATE AS HOT STANDBY PROCESSOR
230
END
235

Figure 6.9: Flow diagram for recovery from loss of lockstep - "hot standby" processor [23]

6.5 System and method to increase lockstep core availability (Infineon Technologies AG.)

An alternative method for increasing lockstep core availability is presented by Infineon Technologies AG [27]. In a simple implementation of a lockstep architecture in prior art, when a core-related error occurs the entire system (both CPUs and peripherals) is placed in a reset state to recover by the error. This method has the drawback of a delay of tens of milliseconds that could not satisfy the temporal requirements of a real time safety-critical system. Another technique that efficiently increases the core availability is the employment of one more CPU (three CPUs running in lockstep mode) and with a majority voting system to detect the defect CPU and recover, while the functional CPUs continue with the execution of the software. Although this method is robust and efficient, the cost, the power consumption and area on the silicon are increased.

The invention of Infineon involves two CPUs operating in lockstep (master and checker in a delayed lockstep or not) and a state buffer where the state of the master CPU is stored. In case of a mismatch in the comparison of the output signals of the two CPUs a control signal is generated and the stored state is loaded to both of the CPUs from the state buffer.

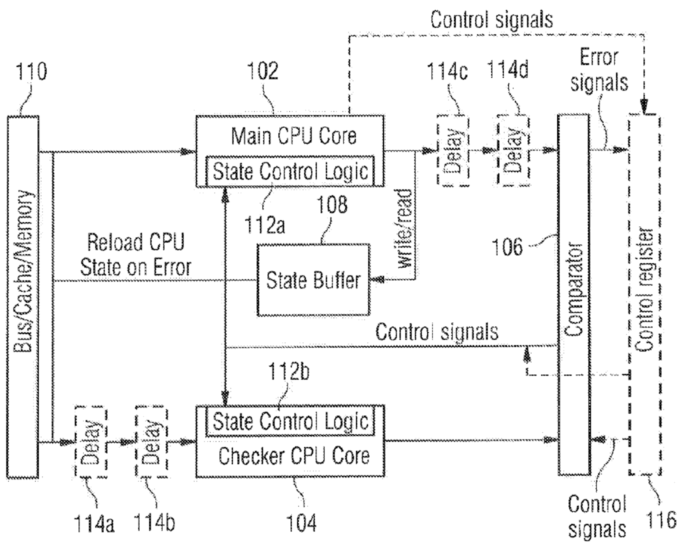


Figure 6.10: Schematic diagram of the Infineon's patent, introducing the state buffer [27]

As shown in Figure 6.10, each of the CPU cores include a state control logic. Various implementations that are proposed in the current patent, propose different state control logic functionalities. In one case they control the operation of their respective CPU core, being capable of pausing and restarting them and writing their current state to the state buffer. The saved state from the state buffer is also loaded with the help of this state control logic. In other implementations it samples periodically the master CPU saving its state to the state buffer or a sampling is triggered on an event occurence such as the start of a task execution.

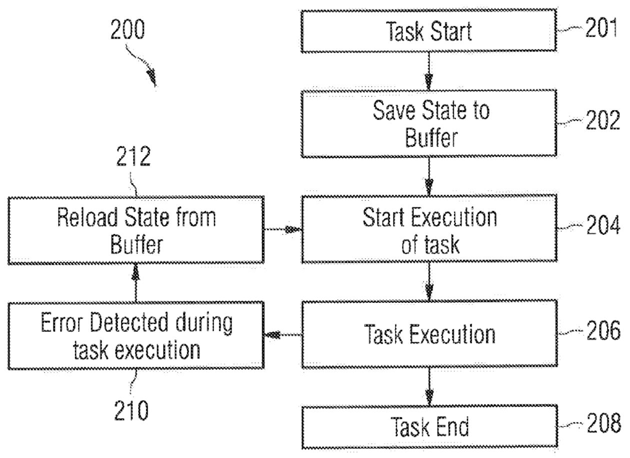


Figure 6.11: Flow diagram where the save state is triggered on a task start [27]

In the case that the flow diagram in Figure 6.11 demonstrates, the state control logic of the master CPU saves the current state to the state buffer by generating a control signal. The current state is interpreted in the voltages and/or current values, or their logical high and low values of the circuit components that the CPU contains, or the values of the local registers of the main CPU. After the completion of the storage of the current state, the first instruction of the task is received by the memory module and is executed in lockstep mode. If the comparator detects a mismatch, it generates a control signal notifying the state control logic of the CPU cores (see Figure 6.10). In another implementation a control register is employed as shown in Figure 6.10 that is notified by the comparator for a mismatch and it generates the control signals for controlling the state control logic of the CPU cores. In any of the aforementioned scenarios, the state control logic of the CPU cores receive a control signal that there is a discrepancy in the comparison of the outputs. Therefore, it generates a read signal to the state buffer, the saved state is retrieved and the execution of the task restarts.

To sum up, a known "good" state of the master CPU core is saved in the state buffer before the execution of a task. In case of a comparison error in the lock-step execution, the core states are reverted back to the known "good" state and the instructions are re-executed. The benefit of this method is that in an erroneous situation only the CPU cores are winding back to a safe state and not the entire system (i.e peripherals). As a result, the core availability, meaning the time that the core is operational and able to execute instruction sets is significantly increased.

Several enhancements are proposed in this patent, such as using a counter to prevent an endless loop of reverting to the same state when executing the same task which could derive from a permanent error. However, the purpose of this chapter is to give an insight about the potential problems and the suggested solutions for the lockstep architecture and not to analyze all the details of each invention.

6.6 Computing with both lockstep and free-step processor modes (Hewlett-Packard Development Company, L.P.)

The current invention demonstrates a method for systems comprised of a set of two or more processors with capability to operate in two step modes [1]: lockstep or free-step mode. Strongly depending on the severity in the safety requirements of a task it can be assigned to a processor to work in a free-step mode, or to be coupled in a set of two processors for execution in lockstep mode. The benefit is an optimal way to trade-off between high performance and system integrity by using free-step or lockstep mode respectively.

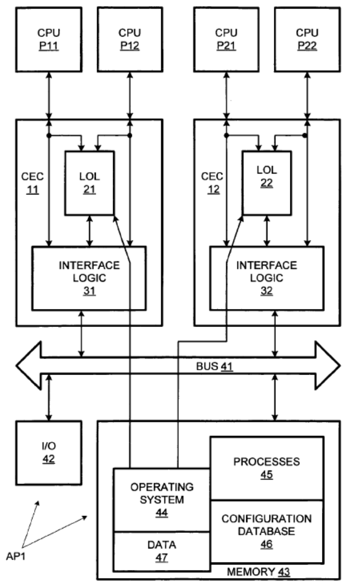


Figure 6.12: A computer system including four processors with the suggested architecture by Hewlett-Packard [1]

The example system in Figure 6.12 includes four processors placed in pairs (P11-P12, P21-22) and each pair interfaces to the rest of the system by the Core Electronics Component (CEC). The rest of the system consists of memory (RAM or hard disk, etc.) and I/O channels. The CEC includes an interface logic for the communication of the processors with the rest system and a loss-of-lockstep logic that controls the error detection by the comparison of the outputs of the processors when the system is operating in lockstep mode. In case of operation in free-step mode the output data from the processors propagate to the system via the interface logic bypassing the LOL. In memory beside the data, processes and the operating system, is stored a configuration database that provide information for the operation of the system:

- Distinguishing the step mode of each CPU (free-step or lockstep mode)
- Some criteria that the above mode assignment could change dynamically
- A list of processes that must run in lockstep mode (By default, processes run in free-step mode if they do not exist in this list)

In each process call, a look-up operation is performed in the database to check if the process requires execution on a single processor in free-step mode or on a pair of processors in lockstep mode. Moreover some special rules could determine the running mode of the process such as the time of execution or resource requirements of the process. For instance, in a nightly execution could be set a rule for forcing a lockstep mode as the performance is not as important as the error detection at that time. In one of the variations of the invention, resource utilization by operating system could also be employed to determine if a switch to free-step mode of an overloaded pair of processors, or a reallocation of the process to another pair should be performed.

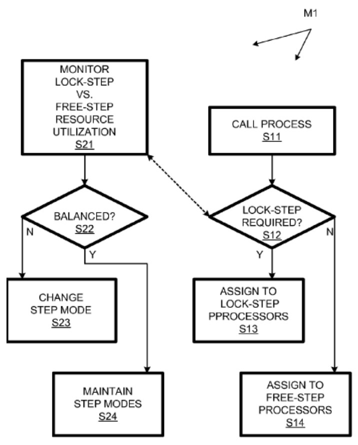

M1
MONITOR LOCK-STEP VS. FREE-STEP RESOURCE UTILIZATION S21
CALL PROCESS S11
BALANCED? S22
N
Y
LOCK-STEP REQUIRED? S12
Y
N
CHANGE STEP MODE S23
ASSIGN TO LOCK-STEP PPROCESSORS S13
MAINTAIN STEP MODES S24
ASSIGN TO FREE-STEP PROCESSORS S14

Figure 6.13: Flow graph of a potential scenario using the current invention [1]

In the scenario presented in the flow graph of Figure 6.13, the steps S21-S24 are executed in parallel with S11-S14. As aforementioned, the steps S11-S14 check the configuration database and assign the process to a single or a pair of processors for free or lockstep mode respectively. At the same time, resource allocation of the processors is monitored by the operating system. In step S22 if one of the processors is stressed, a reallocation is performed in terms of step mode in S23, which may be performed dynamically or after a restart of the system.

Chapter 7

Results

7.1 Experiments

Two experiments were conducted in the current project related to the Lockstep architecture of Texas Instruments TMS570LS3137 microcontroller. The purpose of the first is to demonstrate how the Lockstep architecture can be employed in a safety-critical application in order to detect soft and hard errors that may occur during the execution of the software. The second experiment presents the behavior of the CPU through a GUI that is provided with the HiTex SafeTI kit. It calculates the time that is required from an error occurrence till the error detection.

The purpose of the experiments is to demonstrate how the Lockstep architecture can be employed in a safety-critical application on the TMS570LS3137 microcontroller and to calculate the time that an error will be detected. The hardware that was used is the TMS570LS3137 Hercules Development Kit (see Section 3.1.1) and the SafeTI Hitex Safety Kit - TMS570LS3137 (see Section 3.1.2).

7.1.1 Error-Forcing Experiment on TMS570LS3137HDK

The purpose of the experiment is to give to the reader an overview of how the TMS570LS3137 microcontroller has to be configured in order to operate efficiently in Lockstep mode. Regarding the hardware, the TMS570LS3137 Hercules Development Kit was used to create a simplified application with the programmable push-button and the LEDs that are integrated on it. In terms of software the HALCoGen 04.03.00 was used to configure and generate the initialization code for the microcontroller and the peripherals that were employed in this experiment and the Code Composer Studio IDE 6.1.0 for the development and debugging (partially) the application.

From the aforementioned in the Lockstep implementation of TMS570LS3137 and CCM-R4F, the Lockstep is active in “1oo1D Lockstep Mode” by default and permanently. The developer is not able to disable it.

The current experiment is kept as simple as possible in terms of functionality, in order to avoid the complexity in code that a more sophisticated application could involve. The validity of the experiment is not affected, while the error detection is independent of the application complexity and the size of the code. The errors that may occur are exclusively related to the hardware and the conditions that the system could be physically exposed to.

What is visually perceived during the execution of the code is that when the programmable push-button is pressed, it turns two LEDs on, on the evaluation board:

- The red LED indicating that a hardware error is present on the system and is turned automatically on
- The white LED that is set by the code when the button is pressed

If the button gets pressed for a second time, it turns both of the LEDs off. The behavior of this application was designed with the purpose to artificially generate an error when the user presses the programmable button and monitor if it is efficiently detected by the comparator module. When the error is detected the execution of the software is halted. With a second press of the button the error is cleared from the ESM module and the execution of the software continues.

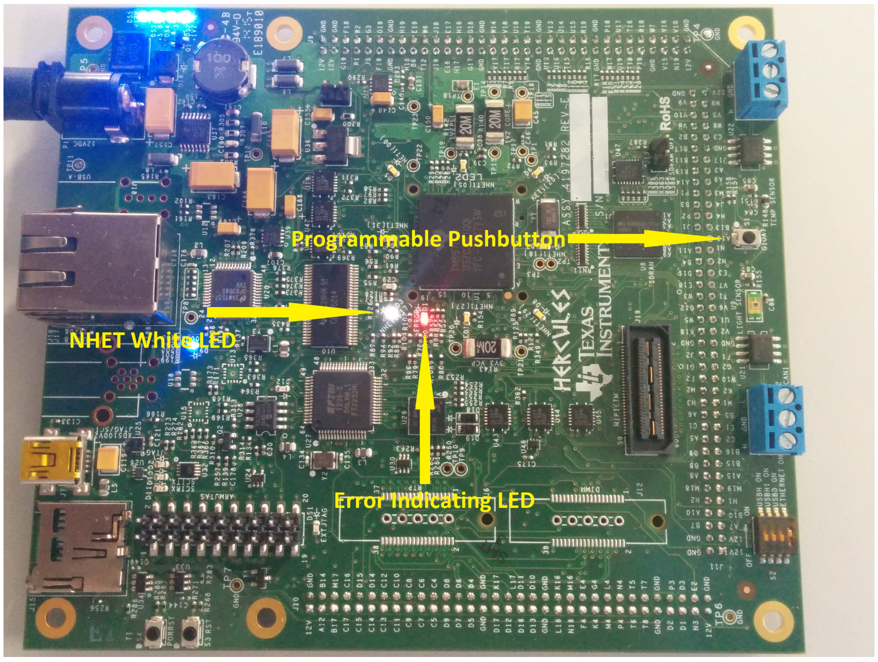


Figure 7.1: Entering Error-forcing mode by pressing the pushbutton

More specifically, in HALCoGen the RTI and GIO drivers need to be set active (see Appendix F for the HALCoGen configuration screenshots). The RTI driver is utilized to integrate a small delay (configured for 500 ms delay in the Compare 0 Period of RTI1 Compare) after the button is pressed to avoid the button bouncing. The GIO driver is enabled in order to use the GIOA7 pin of the evaluation board that is assigned to the programmable push-button. Additionally, the following interrupt channels should be activated:

- 0 for ESM High,
- 20 for ESM Low,
- 2 for RTI Compare 0,
- 9 for GIO Int A

Finally, the interrupt for the GIOA7 pin (Bit 7 in GIO PortA tab) should be enabled. After setting correctly this configuration to the HALCoGen, the code should be generated and we are ready to develop the application in CCS. The main advantage of the generated code by HALCoGen is that it generates all the necessary start-up code for the initialization of the controller and driver-specific code that the user can easily configure via its graphical user interface. For example, a file “sys_core.asm” is generated automatically to set both of the cores to an identical state (i.e their

register values) as is a prerequisite for the efficient use of the Lockstep diagnostic (see Section 5.4.2).
The scope of this experiment is the CCM-R4F configuration. It is a simple module in terms of configuring and manipulating the detected errors. It is controlled by two 32-bit registers: The CCM-R4F Status Register (CCMSR) and the CCM-R4F Key register (CCMKEYR). The former contains information about whether the self-test is completed, or an error is detected and what was the specific pattern that the self-test failed. The latter is used to set or read the mode of the module. For more information regarding the registers of the CCM-R4F module, please refer to Appendix A.

The CCM-R4F module is set to "Error-forcing Mode" by setting the CCMKEYR register to 0x09 (see Appendix A). The input patterns of 0xAs and 0x5s are fed to the CCM-R4F module and a compare error is generated.
The current example application has employed a flag *ESM_High_Int_Flag* that is set to the ESM Interrupt Service Routine (ISR). The HALCoGen generated code determines the priority group that each detected error belongs to (see Section 5.4.8 for error categories) and executes the corresponding code regarding the severity of the error. This is the point where the application developer has to deal with the detected error depending on the safety requirements and the temporal constraints of the application. For the sake of this experiment the ISR is empty while the purpose is to wait for the user to press the button and clear the error. Therefore, with a second button press, the ERROR pin is cleared and the execution of the application continues.

7.1.2 CCM-R4F Experiment with Hitex Safety Kit- TMS570LS3137

The SafeTI Hitex Kit is an evaluation board implemented by Hitex in cooperation with Texas Instruments for assessing the safety features of the MCU. The kit has 2 Hercules MCUs on it. One is the primary MCU and the second is used to inject faults. The software tool can be used to choose and profile the reactions of the main MCU.

The purpose of this experiment is to highlight the capabilities that this evaluation kit provides related to the Lockstep architecture and the CCM-R4F module.

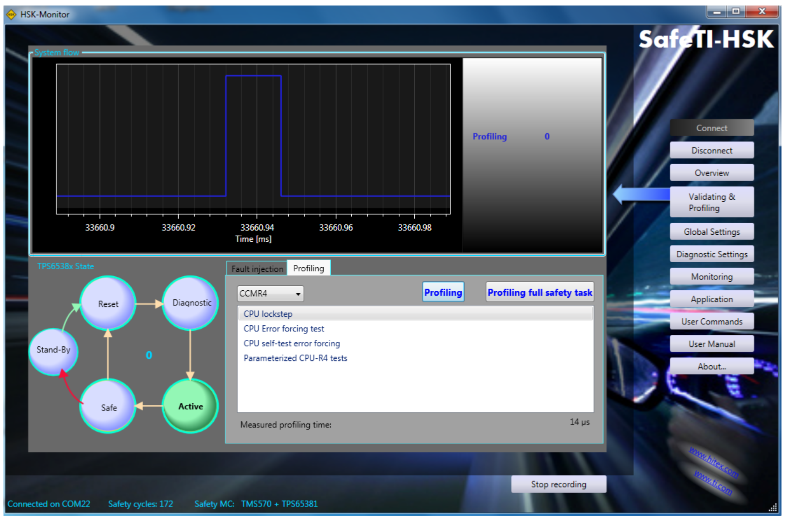
Measurement of Test Execution Time

The profiling window in the HSK Monitor GUI provides the capability to perform time measurements for certain tests. In the drop-down menu when selecting the

CCMR4 module which is the main concept of this project, 3 tests are available and they are executed when the user presses the "Profiling" button:

- *CPU Lockstep*:

The "CPU Lockstep" performs a self-test of the CCM-R4F module (see Section 5.4.3) and the result of this test is depicted in the Figure 7.2. The amount of time that is required for a self-test of the CCM-R4F module is 14 usec as is shown on the right bottom corner of the HSK-Monitor GUI.



CPU lockstep
CPU Error forcing test
CPU self-test error forcing
Parameterized CPU-R4 tests

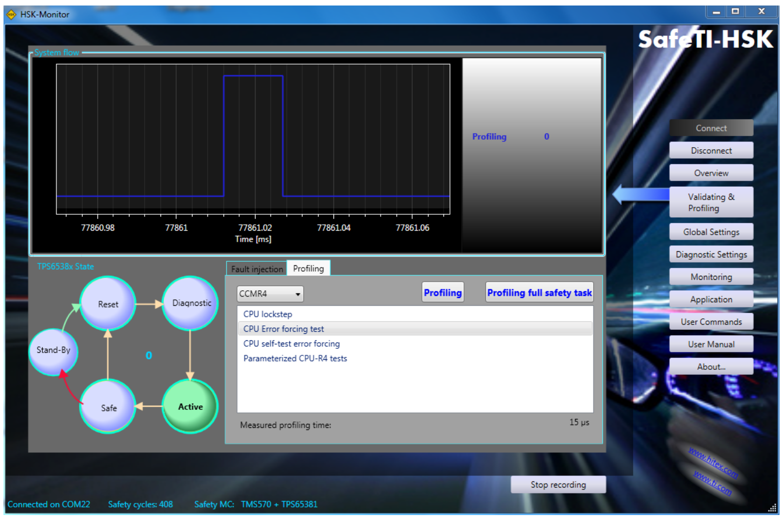
Measured profiling time: 14 µs

Connected on COM22 Safety cycles: 172 Safety MC: TMS570 + TPS65381

Figure 7.2: CPU Lockstep Test Time Measurement

- *CPU Error Forcing Test*:

The execution of the "CPU Error Forcing" test reveals the duration of an error detection that is applied on the compare error output signal of the compare unit (see Section 5.4.4). The derived time that is displayed in the HSK-Monitor is 15 usec.



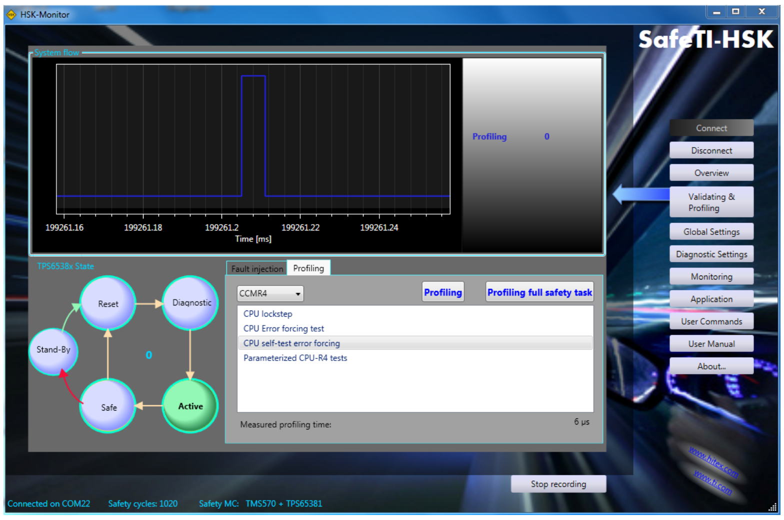
Measured profiling time: 15 µs

Connected on COM22 Safety cycles: 408 Safety MC: TMS570 + TPS65381

Figure 7.3: CPU Error Forcing Test Time Measurement

- *CPU Self-test Error Forcing*:

This test forces an error at the self-test error signal (see Section 5.4.5). Regarding the HSK-Monitor GUI the duration of this test is 6 usec.



Measured profiling time: 6 µs

Connected on COM22 Safety cycles: 1020 Safety MC: TMS570 + TPS65381

Figure 7.4: CPU Self Test Error Forcing Time Measurement

Run-time Injection of Error and Time Detection Measurement

The application that is integrated into the HSK, monitors and implements the appropriate conversions of the temperature and acceleration values that are derived from the corresponding sensors that are integrated on the board. In Figure 7.5 we can observe the behavior of the application when shaking the evaluation board and the acceleration values change rapidly. The tab “Application” displays continuously these values as shown in Figure 7.5.

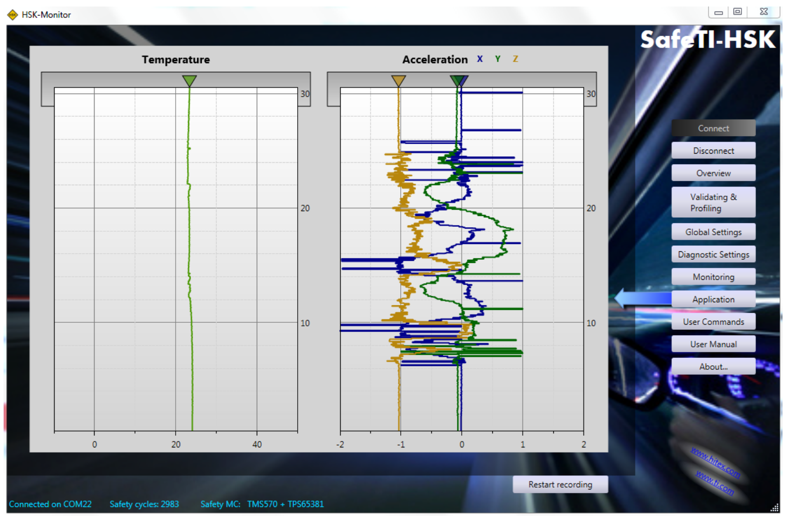
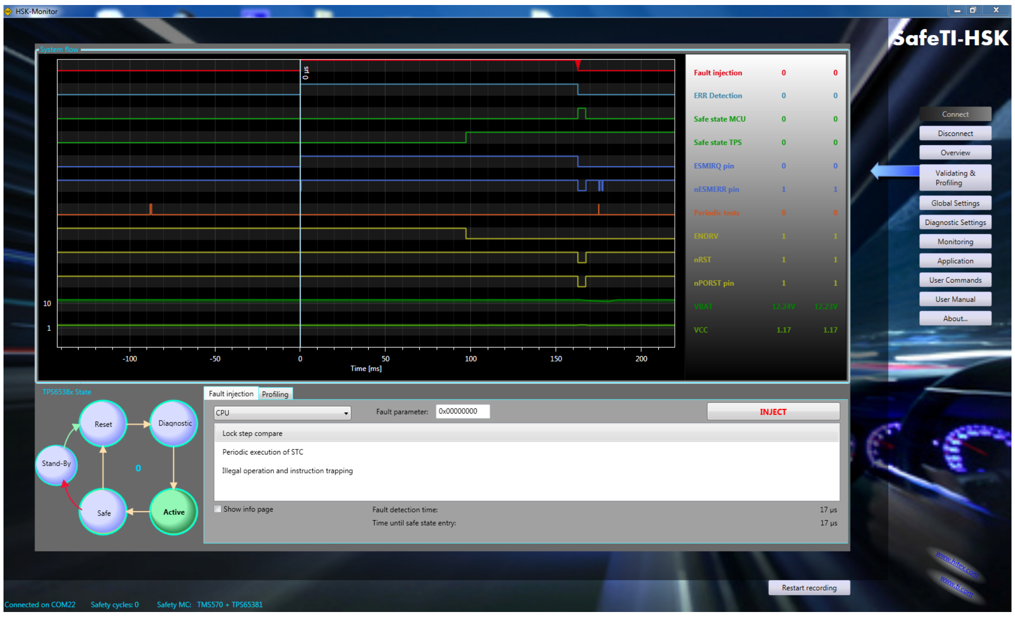


Figure 7.5: Temperature and Acceleration Values in HSK-Monitor GUI

This part of the experiment is targeting to demonstrate the basic feature of the HSK that is the run-time injection of errors. In “Validating & Profiling” tab when “Lock Step Compare” is selected and the user presses the “INJECT” button, an error is injected from the C&M MCU to the SDUT MCU (see Section 3.1.2). The error detection time of the injected error is displayed in the right bottom corner of the HSK-Monitor GUI and in the measurement that is depicted at Figure 7.6 is 17 usec.



Fault injection	0	0
ERR Detection	0	0
Safe state MCU	0	0
Safe state TPS	0	0
ESMIRQ pin	0	0
nESMERR pin	1	1
Periodic tests	0	0
ENDRV	1	1
nRST	1	1
nPORST pin	1	1
VBAT	12.24V	12.23V
VCC	1.17	1.17

CPU — Fault parameter: 0x00000000

Lock step compare
Periodic execution of STC
Illegal operation and instruction trapping

Show info page
Fault detection time: 17 µs
Time until safe state entry: 17 µs

Connected on COM22 Safety cycles: 0 Safety MC: TMS570 + TPS65381

Figure 7.6: Lockstep Compare Run-time Fault Injection

7.1.3 Conclusions

The first experiment is a demonstration of setting the CCM-R4F module in "Error Forcing" mode. The fault injection / forcing is the only mechanism to make an error occur purposefully and reliably. In order to conduct more sophisticated experiments it would require to create a hard fault in the silicon by disassembling the MCU and create a hard fault in. To create a soft error it would be possible by subjecting the MCU to a concentrated amount of radiation (alpha and beta particles) without ensuring that it would create only a core compare error without affecting other elements in the silicon.

The aforementioned testing methods were not possible to be performed in this project due to lack of the special required equipment in the company's laboratories and the high cost of acquiring this. Similar experiments are performed by the semiconductor companies to ensure the right operation of their Lockstep architectures.

The idea of run-time fault injection by a secondary MCU is the innovation of the Hitex-SafeTI-Kit and is useful to assess the behavior of the primary MCU via the well structured user interface that is provided. What was observed during the conduction of the experiments for the time measurement is variations in the fault detection time. For instance, 2 subsequent "CPU Lockstep" error injections result in different fault detection time. Initially, we did not expect a variation while this time should be deterministic. A Lockstep error occurrence is critical and generates the highest priority interrupt. After investigating the way that an error is injected in the primary CPU, the variation in time detection was justified. When the safety application gets the command to "INJECT" a fault the following sequence of steps happens:

1. An I/O pin is raised to signal the fault injection request to the monitor
2. A few instructions are executed to check which kind of failure to inject
3. A function from the SafeTI Diagnostic Library is called to cause the fault injection
4. After the fault injection in the Hardware, an ESM error signal is raised
5. In the ESM IRQ handler another pin is raised indicating to the monitor that the fault is detected.

The steps that occur in a fault injection in the C&M and SDUT devices are demonstrated in Figures 7.7 and 7.8 respectively.

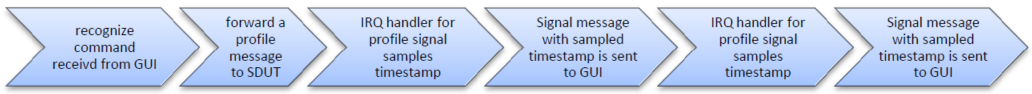

recognize command receivd from GUI
forward a profile message to SDUT
IRQ handler for profile signal samples timestamp
Signal message with sampled timestamp is sent to GUI
IRQ handler for profile signal samples timestamp
Signal message with sampled timestamp is sent to GUI

Figure 7.7: Dataflow of C&M Device In a Fault Injection Operation [33]

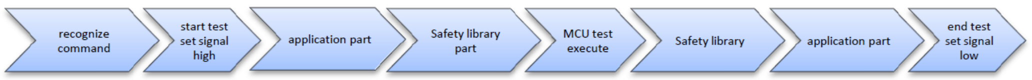
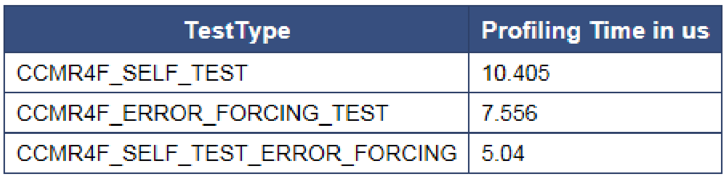


Figure 7.8: Dataflow of SDUT Device In a Fault Injection Operation [33]

As a result, during the pre-work for the error injection, peripherals or OS interrupts may occur and they add overhead to the detection time. This overhead makes the measurement of the detection time non-deterministic. In simple words, the reason is that the fault is injected by software and is not a realistic case where a hardware error generates the highest priority interrupt. More specifically the error detection time in "Error forcing" mode would expected to be 1 CPU clock cycle, as is the duration of the Error-Forcing test (see Section 5.4.4).

HSK integrated software utilizes the functions of the SafeTI Diagnostic Library provided by Texas Instruments for the operation of the safety features. The profiling time measurements of our experiments deviate from the timestamps that are presented in the manual of the SafeTI library:



TestType	Profiling Time in us
CCMR4F_SELF_TEST	10.405
CCMR4F_ERROR_FORCING_TEST	7.556
CCMR4F_SELF_TEST_ERROR_FORCING	5.04

Figure 7.9: SafeTI CCM-R4F Related Functions Time in usec[20]

The CCMR4F_SELF_TEST in Figure 7.9 corresponds to the "CPU Lockstep" profiling test (14 usec), the CCMR4F_ERROR_FORCING_TEST to the *"CPU Error Forcing Test"* (15 usec) and the CCMR4F_SELF_TEST_ERROR_FORCING to *"CPU Self-test Error Forcing"* (6 usec) respectively.

To sum up, the current project covers all the potential experiments that could be conducted related to Lockstep architecture for the available hardware. The restrictions in Texas Instruments implementation (i.e Lockstep mode is disabled in Debug Mode) do not give the opportunity for more sophisticated experiments. Finally, the Hitex SafeTI Kit provides an innovative way of run-time fault injection to the main MCU using a "helper" MCU, but the fault detection time measurement is not a useful feature for our experiments while the values are not realistic and may vary in different consequent fault injections.

Chapter 8

Lockstep In Avionics

This chapter focuses on the process of verifying a COTS hardware component in the special field of Avionics and how the Lockstep architecture should be treated during the certification process. Moreover, an example use case of Lockstep architecture in an avionic system is presented.

8.1 Certification Actions

One of the research questions for this project is *"How Lockstep should be treated in the certification process of an avionic system?"*. As was already mentioned in chapter 2.1.3 the certification process is applied at a system level. As a result we cannot conclude if the Lockstep architecture can be employed in avionic systems or not and under which DAL is suitable to use such an MCU. It is strongly dependent on the nature of the application and the hazards that could derive.

For a COTS hardware component as is the TMS570LS3137 MCU a series of analysis is required (i.e SEH or CEH component, hazard analysis, allocation of hardware component to each function and so on). Although, such an analysis is out of the scope of this project and impossible in absence of a specific system and application. Instead, we focused on how this processor should be treated in a certification process. Following a list of requirements that are included in the "CAST-32 Multi-core Processors" paper provided by FAA (the analysis regarding the Lockstep architecture is attached in Appendix G) we are confident to claim that the TMS570LS3137 MCU should be treated as a single-core processor. Beside the fact that the CAST-32 paper mentions that:

"This paper does not apply to the following MCP architectures:

- *Two core processors in which both cores host the same software and execute that*

> *same software in Lockstep so that their outputs, based on identical input data, can be compared for use in a safety-critical application."*

we followed thoroughly the list of requirements that are provided because as we have already noticed, each Lockstep implementation may be different in terms of shared resources, system availability and so on.

This analysis contributes to a potential certification process in a system that will employ such an MCU. Certification experts from Airbus Defence and Space have the knowledge and the expertise to follow the guidelines dealing with single-core processor systems. Therefore, this project may be the "spark" for the integration of Lockstep architecture MCUs in avionic systems.

8.2 Example Lockstep Use Case

It should have already become clear that it is not the optimal way to present a specific example of a project without having performed a functional-hazard analysis and clear requirements. Nevertheless, a simplified example application of a Lockstep architecture, presented in a very abstract way could help the reader to understand how such an MCU could be employed in a safety-critical system and the extra benefits that this could offer.

Beside the fact that the rate of aircraft engine failures have dramatically decreased, at the point that flight crews is most likely that will not ever meet one in their whole career, it still remains a possibility. Because of this infrequent occurrence of an engine failure the crew is not always able to identify and to handle such a malfunction [9]. As a result, erroneous operations of the crew in such a case could lead to devastating consequences for them and for the passengers.

The engine is likely the most complex and crucial component on the aircraft. It is responsible for hundred calculations and as a result a main part of the safety relies on this component. In our example, we will focus on the fire detection scenario of the engine. In avionics, there is the principle of keeping the aircraft trajectory as the highest priority duty [9]. Thus, in case of an engine malfunction, the system should stabilize the aircraft trajectory first, before proceeding with further actions to resolve the engine problem. This could probably cause a larger damage to the engine but it would help to ensure the safety of humans and the aircraft itself.

In our scenario we introduce a lockstep processor into the aircraft's engine. One of the functions of this MCU is to receive the raw values from temperature sensors, to convert them to Celsius degrees and in case of exceeding the acceptable limit to notify the crew and the system controller via the Ethernet interface to prevent a fire. After taking the appropriate actions to stabilize the aircraft trajectory (i.e activate a redundant engine) the system sets the defective engine to idle state. In high altitude the level of radiation that occurs from cosmic rays is significantly high. As is already mentioned, such a radiation can cause a bit flip in the core of the system that executes this temperature calculation. In case of such a bit flip during the conversion from a single-core MCU without any diagnostic, an erroneous fire alarm could be raised. The erroneous calculation could end up in a non-realistic over-limit temperature value, leading to an undesirable deactivation of the engine and getting the flight crew into trouble. In presence of a Lockstep MCU, such an error would be detected and could temporarily set the system into a safe-state. The two cores operating in Lockstep, most likely would not experience the same bit flip and they would produce divergent outputs. This discrepancy in the comparison of the output signals would generate an error.

From this point and on, everything would depend on the system requirements. For instance, if the system could handle to "wait" the time that is required for a soft-reset of the MCU, it could be determined if the error is temporal or permanent. In case of permanent error the system could finally set the engine idle and enable a redundant one. In case of a soft error, a reset of the system could be enough to continue operating efficiently.

In this example, we tried to eliminate the complexity of such a system and focus only on one functionality and one hazard: How the radiation could affect the conversion of the temperature values and how the Lockstep architecture could facilitate the system designer to mitigate this malfunction. The general safety architecture (i.e a voting system compromising of multiple instances of the Lockstep MCUs, or multiple dissimilar temperature sensors, needed to comply with DAL A requirements that such a system would most probably require) is purposefully skipped. The purpose of this Lockstep use case is to introduce the reader in thinking of the usability of this architecture in complex systems that need to comply with stringent standards and fulfill safety requirements to prevent hazards that could affect humans and the environment.

Chapter 9

Retrospective

In this chapter we summarize the project objectives, highlighting what has been achieved and how both company and scientific community could benefit by this research. The discussion section describes the challenges and restrictions during this project and the philosophy of Texas Instruments for functional safety from the Hercules family MCUs point of view. Moreover, the Future Work section includes ideas how this current project could be extended in scientific and practical terms.

9.1 Discussion

The objectives of the current project were successfully covered providing evidence for each aspect of a Lockstep architecture and especially for the available hardware, the TMS570LS3137 microcontroller by Texas Instruments.

Initially we investigated the term “Lockstep” and its relation to safety-critical embedded systems. The experiments demonstrate how to configure the Lockstep in the desired mode and how to perform the only potential experiment by setting the CCM-R4F module in “Error-Forcing” mode, causing artificially an error. Additionally, the HiteX SafeTI Kit was employed while it is advertised as a great tool for the assessment of the safety features of the Hercules family of microcontrollers. It utilizes a clever concept of a run-time injection of a fault with a simple button click via the provided GUI. Although, the time measurements are not realistic, while the errors are produced by software which adds overhead to the real fault detection time. Most likely it is more realistic for the assessment of other safety features that are out of the scope of this project.

The current project contributes to the company as information material for a potential future use of Lockstep architecture in the innovative avionics systems that it produces. Moreover, it highlights an obscure, in terms of literature, but interesting safety feature that is already a state of the art in automotive, medical and industrial applications.

The following research question that was thoroughly investigated was the reasons to employ a Lockstep based MCU. A simplified answer would be that it is a diagnostic for core-level error detection. Both soft and hard errors are detected (or even corrected, depending on each implementation) that occur by exposure of the system to high radiation levels or a malfunction that occurred during the manufacturing process.

Furthermore, the main available literature that is not strongly dependent on the Hercules family of Texas Instrument was derived from the patents of other semiconductor companies. The demonstration of alternative methods either in terms of hardware architecture or in error detection / correction mechanisms reveal the two main challenges in such a system:

- Synchronization of two cores - Different methods are utilized for the efficient in Lockstep operation of the two cores. In case of running out of sync, erroneous fault detection would be signaled by the comparator module.

- System availability - Some systems require the continuation of the application execution even when an error is detected. Due to the high criticality of the executed function, methods were employed to continue the operation of the system after a fault detection by the Lockstep architecture.

While the current project is strongly related to safety critical aspect of embedded systems it was necessary to include a chapter introducing the reader to the terminology around safety. Moreover, a brief overview of the functional safety related certification standards and processes was crucial while the final products of the discussed microcontroller families need to comply with them in order to be permitted for use in safety-critical applications. While the company under the current project was supported belongs to avionics industry, an introduction to the related certification standards is provided.

One of the crucial points for the company research is *"How Lockstep should be treated in the certification process of an avionic system?"*. As is described in Chapter 2.1.3, the certification process in avionic systems varies from other sectors. Beyond the most stringent requirements to ensure the functional safety, the certification process is done on a system level. As a result, we cannot say that an MCU employing a Lockstep architecture is certifiable for an avionic system or not. It is always dependent on the application and the system and consequently on the DAL level that will be derived from the appropriate analysis. Nonetheless, we performed an initial task that is inevitable for a piece of COTS hardware in an avionic system certification: We conducted an analysis, concluding that the TMS570LS3137 beyond a dual-core processor system, it should be treated as a single-core. The redundancy exists due to the diagnostic channel that monitors and facilitates the error detection in one of the two cores (1oo1D Safety Architecture).

As is already mentioned, the Hercules family of microcontrollers was developed targeting the relevant component level requirements of IEC 61508 and ISO 26262. For DO 254 the situation is more complicated. It requires evidence of the internal component design, including source code of embedded IP components. As a result, confidentiality challenges for the semiconductor companies exist while they cannot expose these information to the public. Thus, most DO 254 compliant systems consist of non-compliant components (The certification is applied on system level).

An important point to note for the Hercules family of microcontrollers is the mitigation of a detected error. In previous chapters we presented the actions for detecting an error but not how to deal with a presence of it. The main reason is that there is not a certain answer how to deal with a detected error. The Hercules devices are not supposed to be fault tolerant. The documentation of these devices including the technical reference manuals and safety analysis documents have an indication of FT = 0 (Fault Tolerance). Essentially, the main focus of the Hercules family is the efficient detection of an error, without though maintaining the operation of the system afterwards. Then, it is strongly dependent on the safety requirements of the system how each error should be handled. Thus, the real functional safety is implemented at system level. Several safety features of these MCUs are provided, but the system design and the implementation of the safety functions are the key elements to achieve high reliability and comply with the application requirements. A typical case for a CCM error detection could be to run the LBIST test to ensure the correctness of the core(s) (if the LBIST test passes, it means that it was a soft error). In a successful LBIST result, it is safe to continue the execution of the application. If the CCM error continues, a soft reset is probably required to re-sync the cores. All these actions are strongly related to safety requirements of the system. The aforementioned scenario though, could not be applicable to a system that executes safety-critical real time operations and the temporal burden of LBIST test execution could lead to a disaster. Actions like turning the system off, setting it to a degraded mode (i.e stop executing safety-critical operations while transferring them to another operational module) or notifying a human could be just a few scenarios of CCM error mitigation.

As was already mentioned in this project, it is not possible to eliminate risks in any system. The real goal of functional safety is to detect and correct a potential fault or to lead the system to fail safely. In any system, there is a Single Point of Failure (SPOF). In the case of the Lockstep architecture, what is essentially achieved is to transfer the SPOF from the CPU core(s) to the CCM comparator module. This is always a concern of safety-system designers. Accepting that it is not feasible to avoid a SPOF, they try to transfer it to a simpler component with lower failure rate, where the error detection is easier and the self-test to ensure the right operation of the component itself is faster. Definitely, the CCM comparator module is much simpler than the core. Additionally, the time execution of CCM self-test is much less than the CPU self-test (10.405 and 364.000 usec respectively).

Along the current report we point out several advantages of the Lockstep architecture. It is a run-time safety mechanism to detect errors without adding any burden in the development process. From the developer's point of view a single-core programming model is required, avoiding the complexity in programming multi-core processors. Additionally, while it is a hardware implemented diagnostic, the error detection is faster than implementing any software that checks for errors. The only disadvantage we can notice is the increased power consumption that such an MCU require in comparison to a single-core. Obviously, in any safety-related system the designer has to decide on a trade-off between either performance or cost (in terms of silicon area or power consumption) and safety-related functions.

9.2 Conclusion

To sum up, the objectives of the current project were accomplished, giving to the reader an overview of the Lockstep architecture in general and detailed information about the Lockstep implementation of Texas Instruments using the Cortex-R4F processor. The research questions that we called to answer covered the Lockstep concept drawing all the available information that after a big effort we managed to collect:

- ✓ What is Lockstep in terms of embedded systems? (see Chapter 5.1)
- ✓ What are the reasons for using Lockstep as an error-detection mechanism? What kind of errors can it detect? (see Chapter 5.2)
- ✓ How to use Lockstep? (see Chapter 7.1.1)

- ✓ How Lockstep should be treated in the certification process of an avionic product? (see Chapter 8.1)

- ✓ What are the limitations and drawbacks of a Lockstep architecture? (see Chapter 9.1)

Due to the kind of errors that Lockstep is targeting, the experiments were restricted while we do not own the equipment to artificially produce such errors. Additionally, as is already mentioned in Section 5.4.7, the Lockstep is automatically disabled in debugging mode. Thus, it was not possible even to check the behavior of the system (i.e register values) when an error-forcing was performed. Nevertheless, the current project is a good base for a future researcher or a designer of safety-critical systems to comprehend the "Lockstep" as a safety concept and to identify the advantages and disadvantages of such an architecture. Every potential aspect of safety should be thoroughly considered while the protection of environment and human lives should be a top priority for everyone.

9.3 Future Work

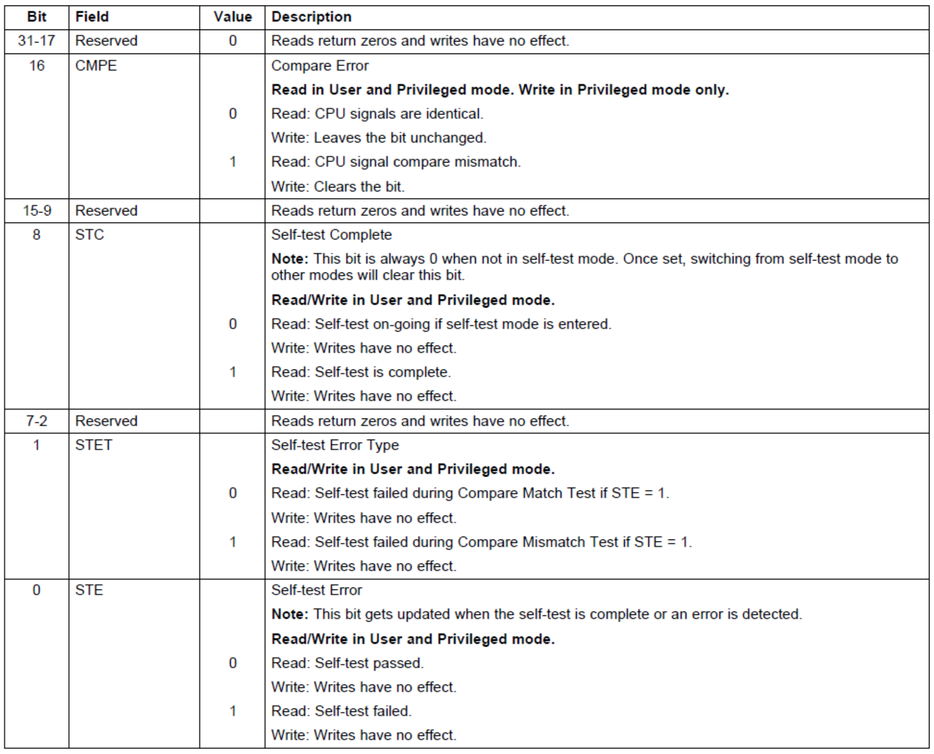
From the academic point of view the main objectives of the Lockstep as a safety concept have been accomplished. One aspect of extending this project could be the investigation of the low-level architecture of such an MCU on transistor level. In fact, it is not possible to obtain such information from the semiconductor companies while they protect their intellectual property and they do not provide so detailed information to the public.

In practical terms, the next step for Lockstep MCUs is their integration in real-world projects in other sectors that require functional safety. More specifically, this research was conducted under the supervision and support of Airbus Defence and Space in order to obtain deeper knowledge regarding the Lockstep architecture, to highlight the pros and cons and how it could be integrated in avionic systems that need to fulfill extremely stringent safety requirements.

From the semiconductor companies' point of view, we would expect over the next few years, a Lockstep architecture with two dissimilar cores. It would be more susceptible to common cause failures while it is unlikely for differently implemented cores to fail in the same way. Obviously there are challenges for this step, basically due to the complexity of synchronizing dissimilar components (i.e synchronize two cores of different architectures to execute the same instructions at the same time).

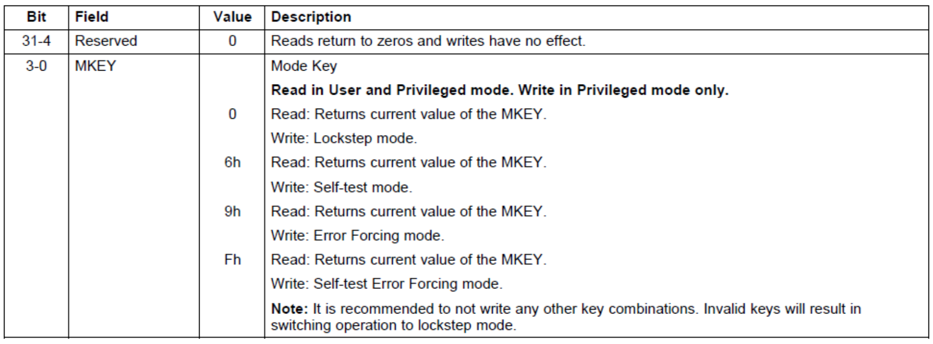
Appendix A

CCM-R4F Register Tables



Bit	Field	Value	Description
31-17	Reserved	0	Reads return zeros and writes have no effect.
16	CMPE		Compare Error
			Read in User and Privileged mode. Write in Privileged mode only.
		0	Read: CPU signals are identical.
			Write: Leaves the bit unchanged.
		1	Read: CPU signal compare mismatch.
			Write: Clears the bit.
15-9	Reserved		Reads return zeros and writes have no effect.
8	STC		Self-test Complete
			Note: This bit is always 0 when not in self-test mode. Once set, switching from self-test mode to other modes will clear this bit.
			Read/Write in User and Privileged mode.
		0	Read: Self-test on-going if self-test mode is entered.
			Write: Writes have no effect.
		1	Read: Self-test is complete.
			Write: Writes have no effect.
7-2	Reserved		Reads return zeros and writes have no effect.
1	STET		Self-test Error Type
			Read/Write in User and Privileged mode.
		0	Read: Self-test failed during Compare Match Test if STE = 1.
			Write: Writes have no effect.
		1	Read: Self-test failed during Compare Mismatch Test if STE = 1.
			Write: Writes have no effect.
0	STE		Self-test Error
			Note: This bit gets updated when the self-test is complete or an error is detected.
			Read/Write in User and Privileged mode.
		0	Read: Self-test passed.
			Write: Writes have no effect.
		1	Read: Self-test failed.
			Write: Writes have no effect.

Figure A.1: CCMSR Resgister [42]



Bit	Field	Value	Description
31-4	Reserved	0	Reads return to zeros and writes have no effect.
3-0	MKEY		Mode Key
			Read in User and Privileged mode. Write in Privileged mode only.
		0	Read: Returns current value of the MKEY.
			Write: Lockstep mode.
		6h	Read: Returns current value of the MKEY.
			Write: Self-test mode.
		9h	Read: Returns current value of the MKEY.
			Write: Error Forcing mode.
		Fh	Read: Returns current value of the MKEY.
			Write: Self-test Error Forcing mode.
			Note: It is recommended to not write any other key combinations. Invalid keys will result in switching operation to lockstep mode.

Figure A.2: CCMKEYR Register [42]

Appendix B

Main function of the first experiment

```
/** @file  sys_main.c
*   @brief  Application  main  file
*   @date 15.Jun.2015
*   @version  04.03.00
*
*   This  file  contains  the  main function ,
*   for  setting  the  CCM−R4F module
*   in  "Error−forcing"  mode when the programmable pushbutton
*   is  pressed
*/

/*  USER CODE BEGIN (0) */
/*  USER CODE END */

/*  Include  Files  */

#include "sys_common.h"

/*  USER CODE BEGIN (1) */
#include "ccmr4.h"
#include "esm_demo.h"
#include " rti .h"
#include "gio.h"
#define BUTTON_LOW 0
#define ERROR_ABSENCE 0
```

```
#define ERROR_PRESENCE 1
#define BUTTON_BIT_NO 7
#define LED_BIT_NO 17
#define LED_OFF 0
#define LED_ON 1

/* USER CODE END */

/* USER CODE BEGIN (2) */
uint16 buttonPressed;
/* USER CODE END */

void main(void)
{
/* USER CODE BEGIN (3) */

        /* Initialize ESM driver */
        esmInit();

        /* Enable the Interrupts */
        _enable_interrupt_();

        /* Initialize the flag that indicates if the button is pressed */
        buttonPressed = BUTTON_LOW;

        /* Initialize the flag that indicates a presence of an error */
        uint32 uiError = ERROR_ABSENCE;

        /* Initialize RTI driver */
         rtiInit ();

    /* Set high end timer GIO port hetPort pin direction to all output */
        gioSetDirection(hetPORT1, 0xFFFFFFFF);

    /* Enable RTI Compare 0 interrupt notification */
        rtiEnableNotification(rtiNOTIFICATION_COMPARE0);

        /* Enable GIO driver */
        gioInit ();

        while(1)
```

```
{
        /* Enable Notification on the 7th pin of GPIO PortA that is
            assigned to the programmable pushbutton */
        gioEnableNotification(gioPORTA, BUTTON_BIT_NO);
        switch (buttonPressed) {
                case 1:
                        if (uiError)
                        {
                                /* Clears the ERROR pin */
                                esmClearErrPin();
                                /* Turns the white LED off */
                                gioSetBit(hetPORT1, LED_BIT_NO,
                                    LED_OFF);
                                /* Disables the button notification */
                                gioDisableNotification(gioPORTA,
                                    BUTTON_BIT_NO);
                                /* Resets the error indicator flag */
                                uiError = ERROR_ABSENCE;
                        }
                        else
                        {
                                /* Turns the white LED on */
                                gioSetBit(hetPORT1, LED_BIT_NO,
                                    LED_ON);
                                /* Sets the CCM-R4 Module in Error-
                                    forcing Mode */
                                CCM_R4_Compare();
                                /* Disables the button notification */
                                gioDisableNotification(gioPORTA,
                                    BUTTON_BIT_NO);
                                /* Sets the error indicator flag */
                                uiError = ERRO_PRESENCE;
                        }
                        /* Insert a delay of 500ms to avoid button bounce
                            */
                        rtiStartCounter(rtiCOUNTER_BLOCK0);
                        /* Reset the button pressed indicator flag */
                        buttonPressed = BUTTON_LOW;
                        break;
        }
```

```
            }
/* USER CODE END */
}

/* USER CODE BEGIN (4) */
/* USER CODE END */
```

Appendix C

CCMR4F-Compare Function

```
/** @file ccmr4.c
*   @brief CCMR4 Driver Source File to check CCMR4 compare error
*   @date 15.June.2015
*   @version 1.00.000
*.\\
*/
/* (c) Texas Instruments 2009, All rights reserved. */
#include "ccmr4.h"
#include "gio.h"
#include "het.h"
#include <stdio.h>
extern unsigned int ESM_High_Int_Flag;

/** @fn void CCM_R4_Compare(void)
*   @brief CCMR4 Compare fail Error creation and check routines.\\
*/
void CCM_R4_Compare(void)
{
        /* Setting the Error forcing mode  */
        CCMR4Reg->CCMKEYR = 0x00000009;

    /* Waiting for interrupt ESM Interrupt flag */
        while(!ESM_High_Int_Flag);

        /* Clear the Interrupt Flag */
        ESM_High_Int_Flag=0;

}
```

Appendix D

ARM GCC 4.9 2015q1 Big-Endian Patch

```
--- /home/steliosgan/gcc-arm-none-eabi-4_9-2015q1-20150306/
    src/gcc/gcc/config/arm/t-rmprofile.orig        2015-02-27
    05:16:50.000000000 +0100
+++ /home/steliosgan/gcc-arm-none-eabi-4_9-2015q1-20150306/
    src/gcc/gcc/config/arm/t-rmprofile    2015-03-20
    10:59:56.870119629 +0100
@@ -13,8 +13,10 @@
 MULTILIB_OPTIONS   += mfloat-abi=softfp/mfloat-abi=hard
 MULTILIB_DIRNAMES += softfp fpu
 MULTILIB_OPTIONS   += mfpu=fpv4-sp-d16/mfpu=vfpv3-d16/mfpu=
     fpv5-sp-d16/mfpu=fpv5-d16
 MULTILIB_DIRNAMES += fpv4-sp-d16 vfpv3-d16 fpv5-sp-d16 fpv5-
     d16
+MULTILIB_OPTIONS   += mbig-endian
+MULTILIB_DIRNAMES += big-endian

 MULTILIB_MATCHES    = march?armv6s-m=mcpu?cortex-m0
 MULTILIB_MATCHES   += march?armv6s-m=mcpu?cortex-m0plus
 MULTILIB_MATCHES   += march?armv6s-m=mcpu?cortex-m1
@@ -89,11 +91,20 @@
 ifneq (,$(filter armv7 armv7-r armv7-a,$(subst $(comma),$(
     space),$(with_multilib_list))))
 MULTILIB_REQUIRED     += mthumb/march=armv7
 MULTILIB_REQUIRED     += mthumb/march=armv7/mfloat-abi=softfp/
     mfpu=vfpv3-d16
 MULTILIB_REQUIRED      += mthumb/march=armv7/mfloat-abi=hard/
    mfpu=vfpv3-d16
+MULTILIB_REQUIRED      += mthumb/march=armv7/mbig-endian
+MULTILIB_REQUIRED      += mthumb/march=armv7/mfloat-abi=softfp/
   mfpu=vfpv3-d16/mbig-endian
+MULTILIB_REQUIRED      += mthumb/march=armv7/mfloat-abi=hard/
   mfpu=vfpv3-d16/mbig-endian
 MULTILIB_OSDIRNAMES  += mthumb/march.armv7=!armv7-ar/thumb
 MULTILIB_OSDIRNAMES  += mthumb/march.armv7/mfloat-abi.hard/
    mfpu.vfpv3-d16=!armv7-ar/thumb/fpu
 MULTILIB_OSDIRNAMES  += mthumb/march.armv7/mfloat-abi.softfp/
    mfpu.vfpv3-d16=!armv7-ar/thumb/softfp
+MULTILIB_OSDIRNAMES  += mthumb/march.armv7/mbig-endian=!armv7
   -ar-bigE/thumb
+MULTILIB_OSDIRNAMES  += mthumb/march.armv7/mfloat-abi.hard/
   mfpu.vfpv3-d16/mbig-endian=!armv7-ar-bigE/thumb/fpu
+MULTILIB_OSDIRNAMES  += mthumb/march.armv7/mfloat-abi.softfp/
   mfpu.vfpv3-d16/mbig-endian=!armv7-ar-bigE/thumb/softfp
 MULTILIB_REUSE         += mthumb/march.armv7=marm/march.armv7
 MULTILIB_REUSE         += mthumb/march.armv7/mfloat-abi.softfp/
    mfpu.vfpv3-d16=marm/march.armv7/mfloat-abi.softfp/mfpu.
    vfpv3-d16
 MULTILIB_REUSE         += mthumb/march.armv7/mfloat-abi.hard/
    mfpu.vfpv3-d16=marm/march.armv7/mfloat-abi.hard/mfpu.
    vfpv3-d16
+MULTILIB_REUSE         += mthumb/march.armv7/mbig-endian=marm/
   march.armv7/mbig-endian
+MULTILIB_REUSE         += mthumb/march.armv7/mfloat-abi.softfp/
   mfpu.vfpv3-d16/mbig-endian=marm/march.armv7/mfloat-abi.
   softfp/mfpu.vfpv3-d16/mbig-endian
+MULTILIB_REUSE         += mthumb/march.armv7/mfloat-abi.hard/
   mfpu.vfpv3-d16/mbig-endian=marm/march.armv7/mfloat-abi.
   hard/mfpu.vfpv3-d16/mbig-endian
 endif
```

Appendix E

How to Build the GNU ARM Toolchain on Ubuntu

E.1 Build GNU Tools on Ubuntu 8.10

The following instructions are located in the GNU ARM Toolchain website [14] and are integrated into this report due to the frequent changes that occur by the active GNU ARM community.

E.1.1 Install Ubuntu Ubuntu 8.10

ISO image is available from http://old-releases.ubuntu.com /releases/8.10/ubuntu-8.10-desktop-i386.iso.
You can install it as a native system or a virtual machine. The command lines provided in this document are all using user id 'build' as an example, so please create a new user called 'build' in the system. Otherwise, you have to replace user id 'build' with your own one.

E.1.2 Tune environment and install required software

Change /bin/sh to bash

Some shell scripts in gcc and other packages are incompatible with the dash shell, which is the default /bin/sh for Ubuntu 8.10. You must make /bin/sh a symbolic link to one of the supported shells: saying bash. Here on Ubuntu 8.10 system, this can be done by running following command firstly: $ sudo dpkg-reconfigure -plow dash Then choose 'No' in the 'Configuring dash' popup dialog and press enter. You can run following command and check that /bin/sh points to 'bash':
$ ls -l /bin/sh

...... /bin/sh -> bash

Change software sources to Main server

On Ubuntu 8.10 system, click 'System->Administration->Software Sources' to open 'Software Sources' dialog, choose 'Main server' in 'Download from:' list box, then click 'close'.
You will be prompted by a window saying 'The information about available software is out-of-date', please click 'Reload'. And then there will be a warning message box popped up, which can be just ignored by clicking 'Close'.

Edit the file using command line:
$ sudo vi /etc/apt/sources.list
replace all 'http://*.ubuntu.com' with 'http://old-releases.ubuntu.com' in that file, save and exit. Run following command to update package list. It should not fail, or else something has been wrong.
$ sudo apt-get update

Install common tools and libraries

Install common tools and libraries needed by build process with below command:

```
$ sudo apt-get install $ sudo apt-get install apt-src \
p7zip-full \
gawk \
gzip \
perl \
autoconf \
m4 \
automake \
libtool \
libncurses5-dev \
gettext \
gperf \
dejagnu \
expect \
tcl \
autogen \
guile-1.6 \
flex \
flip \
bison \
tofrodos \
texinfo \
g++ \
gcc-multilib \
libgmp3-dev \
libmpfr-dev \
debhelper \
texlive \
texlive-extra-utils
```

Note that the package management software might complain that several packages cannot be installed properly while installing texlive and texliveextra- utils. It won't harm our building process, please just ignore it now. Some of those tools might be unnecessary, but it won't hurt if installed.

Download and deploy prebuilt native tools

In order to save effort to prepare the native build tools, we provide prebuilt ones at website https://launchpad.net/gcc-arm-embedded-misc /native-build-tools/20140701. The related source package and script are also provided. Please download the tool and decompress it to a proper place, it will be used in subsequent steps to build gcc arm embedded toolchain. The command to decompress it looks like:

```
tar xf prebuilt-native-tools.tar.lzma –lzma
```

Please be noted that those prebuilt tools are for Ubuntu 8.10 32-bit and not suitable for any other build platforms. For those working on other build platforms, please either prepare your own build tools and use them through option –build tools of build script or just use the ones from your system by not specifying the –build tools option.

E.1.3 Build GNU Tools for ARM Embedded Processors

If you download and decompress the prebuilt tools successfully, you have set up the building environment. You can now start to build the toolchain by yourself with below commands:

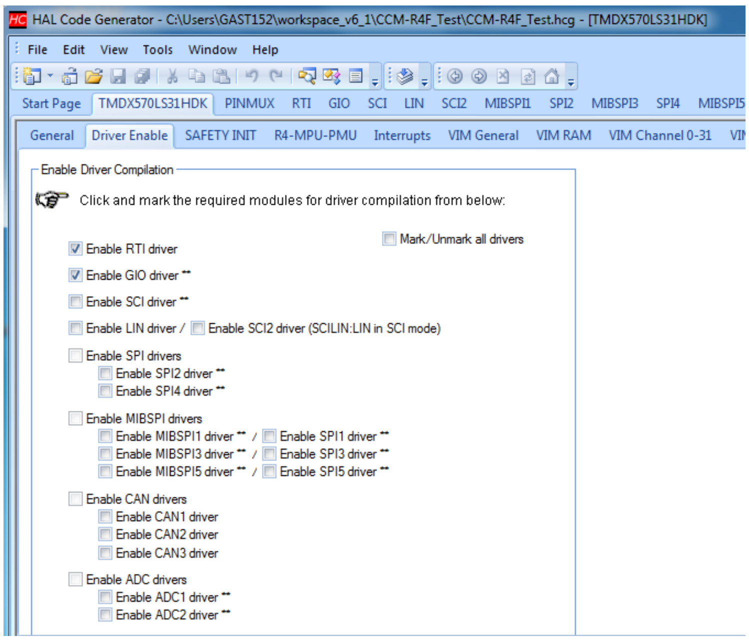
```
#Copy the src release package into ~/toolchain/ directory
$ cp gcc-arm-none-eabi-4_9-2015q1-20150306-src.tar.bz2 ~/toolchain
#Prepare source codes
$ cd ~/toolchain
$ tar -xjf gcc-arm-none-eabi-4_9-2015q1-20150306-src.tar.bz2
$ cd ./gcc-arm-none-eabi-4_9-2015q1-20150306/src
$ find -name '*.tar.*' | xargs -I% tar -xf %
$ cd ../
```

#Start building the toolchain.
#Can specify "–skip_steps=mingw32" option to skip building windows host
#toolchain, and if specify that option when building prerequisites,
#you have to specify it when building toolchain too.
$ ./build-prerequisites.sh –build_tools=YOUR_PATH
$ ./build-toolchain.sh –build_tools=YOUR_PATH
After this, you can 'cd' into
'~/toolchain/gcc-arm-none-eabi-4_9-2015q1-20150306/pkg' and find the built toolchain/-source code packages and the md5 checksum file

Appendix F

HALCoGen Configuration Screenshots

The following figures depict the configuration that is required in the GUI of HALCoGen for the first experiment (see Section 7.1.1). After setting the appropriate options the code is generated to proceed further with the development of the application.

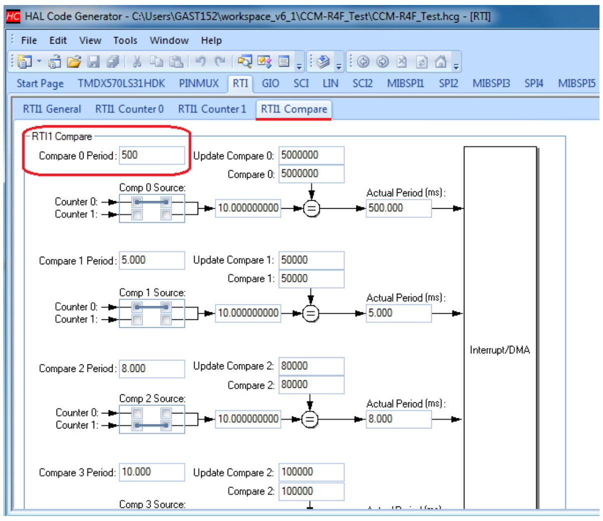

HAL Code Generator - C:\Users\GAST152\workspace_v6_1\CCM-R4F_Test\CCM-R4F_Test.hcg - [TMDX570LS31HDK]
File Edit View Tools Window Help
Start Page TMDX570LS31HDK PINMUX RTI GIO SCI LIN SCI2 MIBSPI1 SPI2 MIBSPI3 SPI4 MIBSPI5
General Driver Enable SAFETY INIT R4-MPU-PMU Interrupts VIM General VIM RAM VIM Channel 0-31

Enable Driver Compilation

Click and mark the required modules for driver compilation from below:

- [ ] Mark/Unmark all drivers
- [✓] Enable RTI driver
- [✓] Enable GIO driver **
- [ ] Enable SCI driver **
- [ ] Enable LIN driver / [ ] Enable SCI2 driver (SCILIN:LIN in SCI mode)
- [ ] Enable SPI drivers
  - [ ] Enable SPI2 driver **
  - [ ] Enable SPI4 driver **
- [ ] Enable MIBSPI drivers
  - [ ] Enable MIBSPI1 driver ** / [ ] Enable SPI1 driver **
  - [ ] Enable MIBSPI3 driver ** / [ ] Enable SPI3 driver **
  - [ ] Enable MIBSPI5 driver ** / [ ] Enable SPI5 driver **
- [ ] Enable CAN drivers
  - [ ] Enable CAN1 driver
  - [ ] Enable CAN2 driver
  - [ ] Enable CAN3 driver
- [ ] Enable ADC drivers
  - [ ] Enable ADC1 driver **
  - [ ] Enable ADC2 driver **

Figure F.1: Enable RTI - GIO Drivers


HAL Code Generator - C:\Users\GAST152\workspace_v6_1\CCM-R4F_Test\CCM-R4F_Test.hcg - [RTI]
File Edit View Tools Window Help
Start Page TMDX570LS31HDK PINMUX RTI GIO SCI LIN SCI2 MIBSPI1 SPI2 MIBSPI3 SPI4 MIBSPI5
RTI1 General RTI1 Counter 0 RTI1 Counter 1 RTI1 Compare
RTI1 Compare
Comp 0 Source:
Counter 0:
Counter 1:
Comp 1 Source:
Comp 2 Source:
Comp 3 Source:
Interrupt/DMA

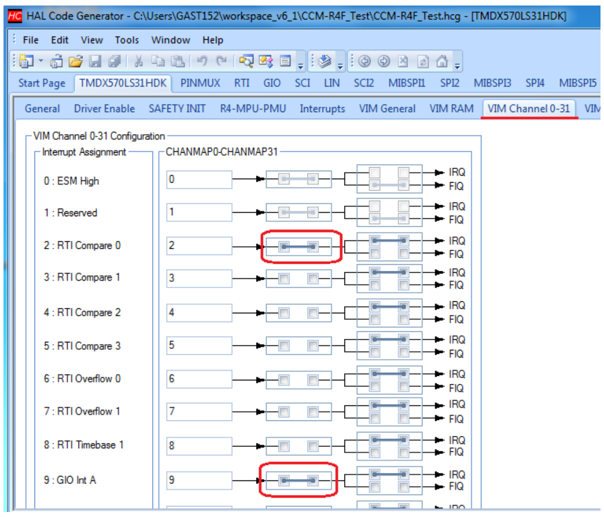
Compare 0 Period: 500
Update Compare 0: 5000000
Compare 0: 5000000
10.000000000
Actual Period (ms): 500.000

Compare 1 Period: 5.000
Update Compare 1: 50000
Compare 1: 50000
10.000000000
Actual Period (ms): 5.000

Compare 2 Period: 8.000
Update Compare 2: 80000
Compare 2: 80000
10.000000000
Actual Period (ms): 8.000

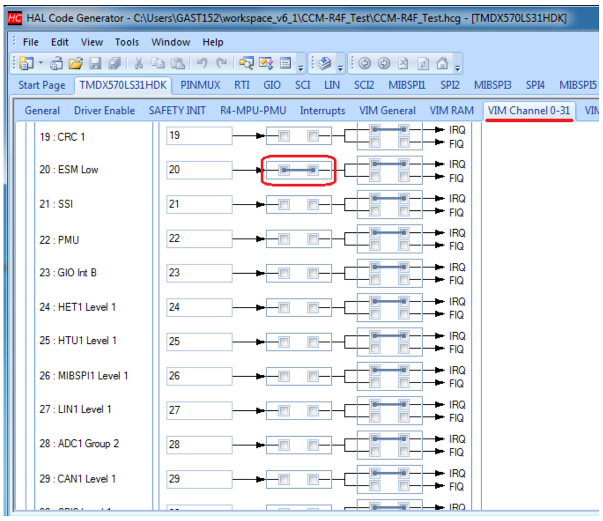
Compare 3 Period: 10.000
Update Compare 2: 100000
Compare 2: 100000

Figure F.2: Configure RTI Compare 0 to generate 500 usec delay


HAL Code Generator - C:\Users\GAST152\workspace_v6_1\CCM-R4F_Test\CCM-R4F_Test.hcg - [TMDX570LS31HDK]
File Edit View Tools Window Help
Start Page TMDX570LS31HDK PINMUX RTI GIO SCI LIN SCI2 MIBSPI1 SPI2 MIBSPI3 SPI4 MIBSPI5
General Driver Enable SAFETY INIT R4-MPU-PMU Interrupts VIM General VIM RAM VIM Channel 0-31 VIM
VIM Channel 0-31 Configuration
Interrupt Assignment
CHANMAP0-CHANMAP31
IRQ
FIQ

Interrupt Assignment	CHANMAP0-CHANMAP31
0 : ESM High	0
1 : Reserved	1
2 : RTI Compare 0	2
3 : RTI Compare 1	3
4 : RTI Compare 2	4
5 : RTI Compare 3	5
6 : RTI Overflow 0	6
7 : RTI Overflow 1	7
8 : RTI Timebase 1	8
9 : GIO Int A	9

Figure F.3: Activation of RTI Compare 0 and GIO Int A in VIM



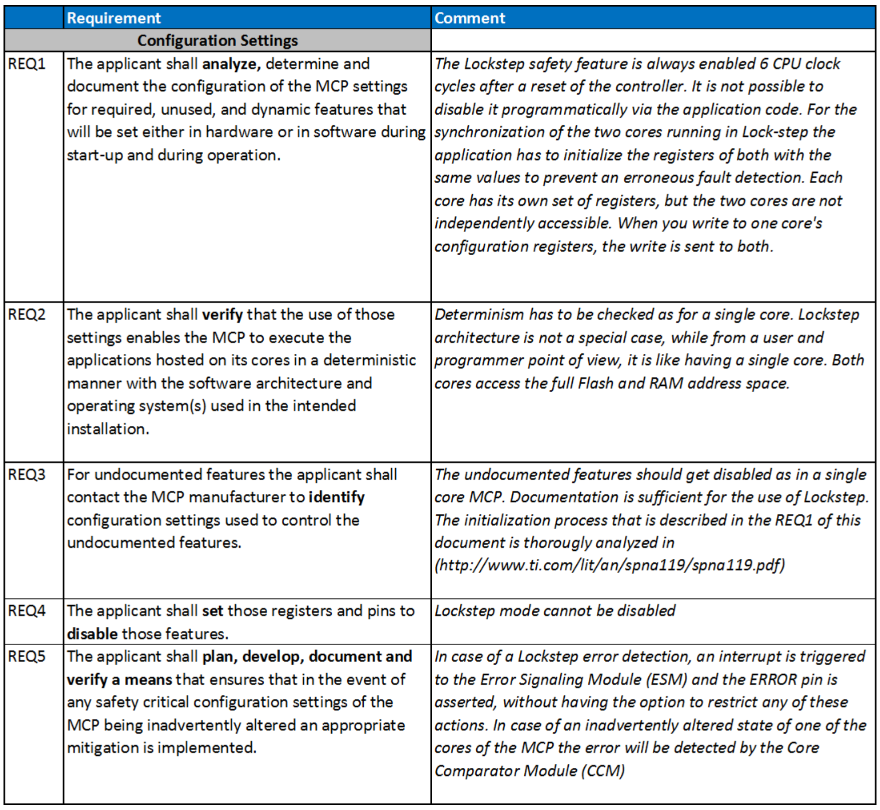
HAL Code Generator - C:\Users\GAST152\workspace_v6_1\CCM-R4F_Test\CCM-R4F_Test.hcg - [TMDX570LS31HDK]

19 : CRC 1	19
20 : ESM Low	20
21 : SSI	21
22 : PMU	22
23 : GIO Int B	23
24 : HET1 Level 1	24
25 : HTU1 Level 1	25
26 : MIBSPI1 Level 1	26
27 : LIN1 Level 1	27
28 : ADC1 Group 2	28
29 : CAN1 Level 1	29

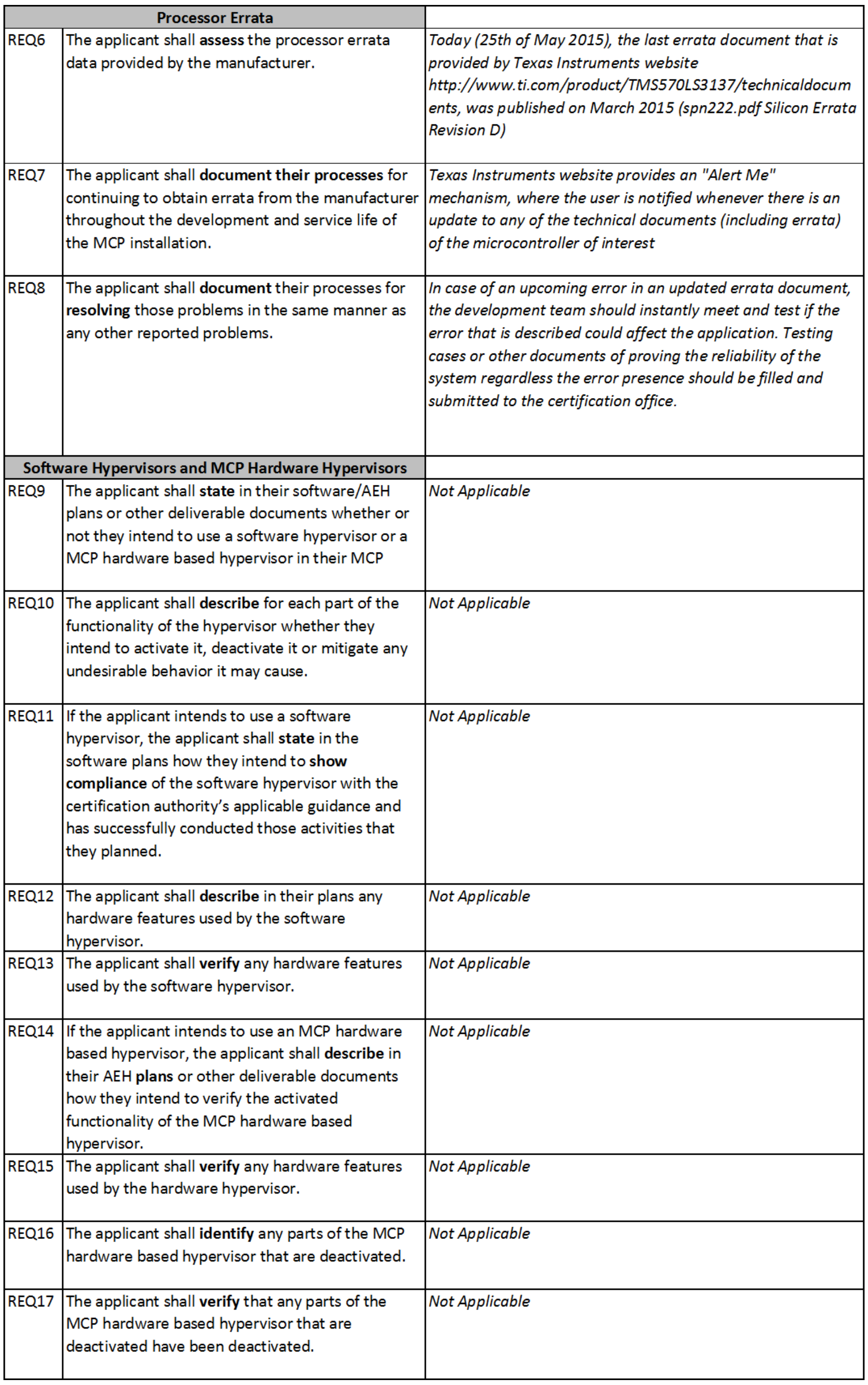
Figure F.4: Activation of ESM Low Interrupt in VIM Module

Appendix G

Requirements for Multi-core Processors



	Requirement	Comment
	Configuration Settings	
REQ1	The applicant shall **analyze,** determine and document the configuration of the MCP settings for required, unused, and dynamic features that will be set either in hardware or in software during start-up and during operation.	*The Lockstep safety feature is always enabled 6 CPU clock cycles after a reset of the controller. It is not possible to disable it programmatically via the application code. For the synchronization of the two cores running in Lock-step the application has to initialize the registers of both with the same values to prevent an erroneous fault detection. Each core has its own set of registers, but the two cores are not independently accessible. When you write to one core's configuration registers, the write is sent to both.*
REQ2	The applicant shall **verify** that the use of those settings enables the MCP to execute the applications hosted on its cores in a deterministic manner with the software architecture and operating system(s) used in the intended installation.	*Determinism has to be checked as for a single core. Lockstep architecture is not a special case, while from a user and programmer point of view, it is like having a single core. Both cores access the full Flash and RAM address space.*
REQ3	For undocumented features the applicant shall contact the MCP manufacturer to **identify** configuration settings used to control the undocumented features.	*The undocumented features should get disabled as in a single core MCP. Documentation is sufficient for the use of Lockstep. The initialization process that is described in the REQ1 of this document is thorougly analyzed in (http://www.ti.com/lit/an/spna119/spna119.pdf)*
REQ4	The applicant shall **set** those registers and pins to **disable** those features.	*Lockstep mode cannot be disabled*
REQ5	The applicant shall **plan, develop, document and verify a means** that ensures that in the event of any safety critical configuration settings of the MCP being inadvertently altered an appropriate mitigation is implemented.	*In case of a Lockstep error detection, an interrupt is triggered to the Error Signaling Module (ESM) and the ERROR pin is asserted, without having the option to restrict any of these actions. In case of an inadvertently altered state of one of the cores of the MCP the error will be detected by the Core Comparator Module (CCM)*

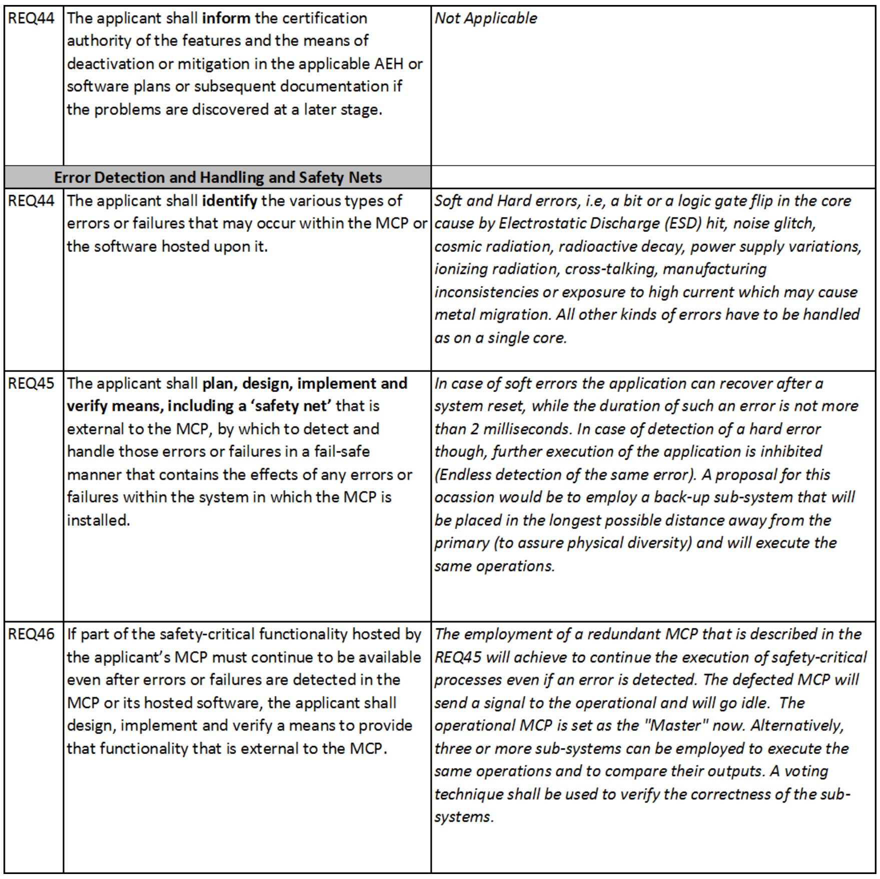


Processor Errata		
REQ6	The applicant shall **assess** the processor errata data provided by the manufacturer.	*Today (25th of May 2015), the last errata document that is provided by Texas Instruments website http://www.ti.com/product/TMS570LS3137/technicaldocuments, was published on March 2015 (spn222.pdf Silicon Errata Revision D)*
REQ7	The applicant shall **document their processes** for continuing to obtain errata from the manufacturer throughout the development and service life of the MCP installation.	*Texas Instruments website provides an "Alert Me" mechanism, where the user is notified whenever there is an update to any of the technical documents (including errata) of the microcontroller of interest*
REQ8	The applicant shall **document** their processes for **resolving** those problems in the same manner as any other reported problems.	*In case of an upcoming error in an updated errata document, the development team should instantly meet and test if the error that is described could affect the application. Testing cases or other documents of proving the reliability of the system regardless the error presence should be filled and submitted to the certification office.*
Software Hypervisors and MCP Hardware Hypervisors		
REQ9	The applicant shall **state** in their software/AEH plans or other deliverable documents whether or not they intend to use a software hypervisor or a MCP hardware based hypervisor in their MCP	*Not Applicable*
REQ10	The applicant shall **describe** for each part of the functionality of the hypervisor whether they intend to activate it, deactivate it or mitigate any undesirable behavior it may cause.	*Not Applicable*
REQ11	If the applicant intends to use a software hypervisor, the applicant shall **state** in the software plans how they intend to **show compliance** of the software hypervisor with the certification authority's applicable guidance and has successfully conducted those activities that they planned.	*Not Applicable*
REQ12	The applicant shall **describe** in their plans any hardware features used by the software hypervisor.	*Not Applicable*
REQ13	The applicant shall **verify** any hardware features used by the software hypervisor.	*Not Applicable*
REQ14	If the applicant intends to use an MCP hardware based hypervisor, the applicant shall **describe** in their AEH **plans** or other deliverable documents how they intend to verify the activated functionality of the MCP hardware based hypervisor.	*Not Applicable*
REQ15	The applicant shall **verify** any hardware features used by the hardware hypervisor.	*Not Applicable*
REQ16	The applicant shall **identify** any parts of the MCP hardware based hypervisor that are deactivated.	*Not Applicable*
REQ17	The applicant shall **verify** that any parts of the MCP hardware based hypervisor that are deactivated have been deactivated.	*Not Applicable*

MCP Interference Channel		
REQ18	The applicant shall **conduct** a functional interference **analysis** to identify all the interference channels between the software hosted on the cores of the MCP.	*There are no interference channels between the two cores as the system behave as a single-core. The secondary core is truly only a checker for the first. It is not connected to the data/instruction buses directly. It will still do everything that the primary core does in regard to instantiating reads and writes but only the reads and writes from the primary core will reach the rest of the system.*
REQ19	The applicant shall **design, implement and verify** a means of mitigation for each of those interference channels.	
Shared Memory and Cache		
REQ20	The applicant shall **state** in their software plans whether or not they intend to use shared memory (between the processing cores).	*The application is not able to handle the shared memory. Both of the cores share the same address space (RAM and FLASH) simultaneously. The read/write operations are performed by the "master" core. The "checker" core receives the data that the master read.*
REQ21	The applicant shall **describe** in their software **plans** the means they intend to use to control **access to shared memory** locations and to prevent the disruptions to deterministic software execution caused by problems such as race conditions, data starvation, deadlocks or live-locks.	*Both cores use the same data bus without risk of deadlocks, data starvation or live-locks while they execute the same instructions at a time.*
REQ22	If the applicant uses shared memory between the processing cores, the applicant shall **test the means** that they have designed to control the **access to shared memory**.	*The cores are not able to execute different instructions at a time*
REQ23	The applicant shall **ensure** that the implemented means provides uninterruptible access to the shared memory locations from either core of the MCP and prevents either core being locked out from accessing the shared memory.	*The cores cannot independently access the shared resources*
REQ24	The applicant shall **state** in their software **plans** whether or not they intend to use shared cache between the processing cores.	*L1&L2 Cache are exclusive resources for each core.*
REQ25	If shared cache is used between the processing core, the applicant shall **describe** in their **plans** their strategy for managing and verifying cache usage.	*Not Applicable*
REQ26	If the applicant uses shared cache between the processing cores, the applicant shall **conduct analyses and tests** to determine the **worst-case effects** that the use of **shared cache and memory** can have on the execution of the specific software applications hosted on the two cores of the MC.	*Not Applicable*
REQ27	The applicant shall **describe** the effects of the shared cache usage on the WCET to the certification authority.	*Not Applicable*
REQ28	The applicant shall **implement** and verify a means to mitigate the effects of using shared cache.	*Not Applicable*

Planning and Verification of Resource Usage		
REQ29	The applicant shall **describe** in their software/AEH **plans** or other deliverable documents how they **intend to allocate, manage and measure the use of resources and the use of the interconnect** by the applications hosted on the MCP and by other MCP peripherals so as to avoid contention for MCP resources and to prevent the capacity of the interconnect and the resources of the MCP from being exceeded.	*Not Applicable*
REQ30	The applicant shall **allocate** the usage of the MCP resources to the software applications hosted on the MCP and shall **verify** that the total of the resource demands when all applications are executing in the worst-case situation does not exceed the total of the resources available.	*Not Applicable*
REQ31	The applicant shall **determine** the maximum **capacity of any interconnect mechanism** of their MCP to sustain transactions in a deterministic manner.	*Not Applicable*
REQ32	The applicant shall **verify** that the demands made on that mechanism by the software hosted on the MCP or by any peripherals of the MCP do not exceed its maximum capacity during any phase of operation of the system.	*Not Applicable*
REQ33	The applicant shall **identifiy the non-deterministic effects** that any **coherency mechanism** of their MCP could cause, such as excessive jitter in data arrival times, or transactions being lost or serviced in an undesirable order.	*Not Applicable*
REQ34	The applicant shall **design, implement and verify** means to deactivate the features concerned to cause non-deterministic effects or to mitigate these effects.	*Not Applicable*
Software Planning and Development Processes		
REQ35	In the software **plans,** the applicant shall **identify** their MCP **software architecture and operating system(s)** and has provided details of how they will develop and verify all the software loaded onto the MCP so as to ensure that the software applications hosted by the MCP execute deterministically including details of any special methods or tools that are necessary due to the use of an MCP or the selected MCP architecture.	*Not Applicable*
Software Verification		
REQ36	The applicant shall **describe** in their Software Verification Plan (SVP) the environment to be used for each software test activity.	*Not Applicable*

REQ37	For any testing that will not be conducted using the target MCP, the applicant shall **describe** the environment that they intend to use, their rationale for using a different test environment and why they consider that test environment to be sufficiently representative of the target MCP.	*Not Applicable*
REQ38	The applicant shall **verify** that each operating system and/or software interface with the MCP hardware when installed on the MCP target complies with applicable objectives in DO-178B/ED-12B or DO-178C/ED-12C (e.g. DO-178C reference A-5 numbers 1 through 9, A-6 numbers 1 through 5) .	*Not Applicable*
REQ39	The applicant shall **verify** that each individual software application that is hosted on the MCP complies with the applicable objectives in DO-178B/ED-12B or DO-178C/ED-12C (e.g. DO-178C reference A-5 numbers 1 through 9, A-6 numbers 1 through 5) when all the applications hosted on the MCP are executing in the intended final configuration of the processor and its hosted software.	*Not Applicable*
REQ40	The applicant shall **verify** that the **data and control coupling** between all software components hosted on the MCP has been exercised during software requirement-based testing including exercising any implicit (e.g. through interconnect features) or explicit interfaces between the applications via shared memory and any mechanisms to control the access to shared memory, and that the data and control coupling is correct.	*Not Applicable*
REQ41	The applicant shall **conduct robustness testing of the interfaces** and the features of the MCP both when software applications are executing individually and when all the software hosted on the MCP is executing, and has verified the compliance of all the hosted software with the applicable objectives in DO-178B/ED-12B or DO-178C/ED-12C (e.g. DO-178C reference A-6 numbers 1 through 5) and with the resource allocation to each application under these conditions.	*Not Applicable*
Discovery of Additional Features or Problems		
REQ42	The applicant shall **identify** any features of the MCP not specifically mentioned in this paper that could cause non-deterministic behavior of the software hosted on the MCP, has designed.	*Not Applicable*
REQ43	The applicant shall **implement and test means** to deactivate those features or mitigate their effects.	*Not Applicable*



REQ44	The applicant shall **inform** the certification authority of the features and the means of deactivation or mitigation in the applicable AEH or software plans or subsequent documentation if the problems are discovered at a later stage.	*Not Applicable*
Error Detection and Handling and Safety Nets		
REQ44	The applicant shall **identify** the various types of errors or failures that may occur within the MCP or the software hosted upon it.	*Soft and Hard errors, i.e, a bit or a logic gate flip in the core cause by Electrostatic Discharge (ESD) hit, noise glitch, cosmic radiation, radioactive decay, power supply variations, ionizing radiation, cross-talking, manufacturing inconsistencies or exposure to high current which may cause metal migration. All other kinds of errors have to be handled as on a single core.*
REQ45	The applicant shall **plan, design, implement and verify means, including a 'safety net'** that is external to the MCP, by which to detect and handle those errors or failures in a fail-safe manner that contains the effects of any errors or failures within the system in which the MCP is installed.	*In case of soft errors the application can recover after a system reset, while the duration of such an error is not more than 2 milliseconds. In case of detection of a hard error though, further execution of the application is inhibited (Endless detection of the same error). A proposal for this ocassion would be to employ a back-up sub-system that will be placed in the longest possible distance away from the primary (to assure physical diversity) and will execute the same operations.*
REQ46	If part of the safety-critical functionality hosted by the applicant's MCP must continue to be available even after errors or failures are detected in the MCP or its hosted software, the applicant shall design, implement and verify a means to provide that functionality that is external to the MCP.	*The employment of a redundant MCP that is described in the REQ45 will achieve to continue the execution of safety-critical processes even if an error is detected. The defected MCP will send a signal to the operational and will go idle. The operational MCP is set as the "Master" now. Alternatively, three or more sub-systems can be employed to execute the same operations and to compare their outputs. A voting technique shall be used to verify the correctness of the sub-systems.*

Bibliography

[1] Ken G. Pomaranski Roseville CA (US) Andrew H. Barr Roseville CA (US). "Computing with both lock-step and free-step processor modes". Pat. US 8,826,288 B2. 2014.

[2] Andre Roger Munlch (DE) Antonio Vilela Mering (DE). "Method and system for fault containment". Pat. US 8,819,485 B2. 2014.

[3] M. Baumeister. *Using Decoupled Parallel Mode for Safety Applications, User's Manual*. Freescale.

[4] T. Ferrell C. Spitzer U. Ferrell, ed. *Digital Avionics Handbook*. 2014.

[5] *Code Composer Studio wiki page*. Available: http://processors.wiki.ti.com/index.php/GSG:CCSv5-Overview [Accessed: 08-06-2015].

[6] *Cortex-R4 and Cortex-R4F, Technical Reference Manual*. r1p3. ARM.

[7] *Dhrystone Benchmarking for ARM Cortex Processors, User's Manual*. ARM.

[8] *ECC Handling in TMSx70-Based Microcontrollers, User's Manual*. Texas Instruments, February 2011.

[9] *Flight Operations Briefing Notes - Handling Engine Malfunctions, User's Manual*. Airbus Defence and Space.

[10] Christopher Temple Munich (DE) Florian Bogenberger Poing (DE). "Method And Apparatus For Handling An Ouput Mismatch". Pat. US 8,373,435B2. 2013.

[11] Freescale. *MPC564xL Family 32-bit Architecture MCUs [Brochure]*.

[12] *Freescale MPC564xL Page*. Available: http://www.freescale.com/webapp/sps/site/prod-summary.jsp?code=MPC564xL [Accessed: 12-05-2015].

[13] *Functional Safety, User's Manual*. IEC - International Electrotechnical Commission.

[14] *GNU Tools for ARM Embedded Processors*. Available: https://launchpad.net/gcc-arm-embedded [Accessed: 14-06-2015].

[15] *HALCoGen wiki page*. Available: http://processors.wiki.ti.com/index.php/HALCoGen [Accessed: 09-06-2015].

[16] *Hercules Microcontrollers [Brochure]*. Texas Instruments. 2014.

[17] Hitex. *SafeTI Hitex Safety Kit*. http://www.ti.com/tool/safeti-hsk-rm48 [Accessed: 10-06-2015].

[18] *IEC FAQ*. Available: http://iec.ch/functionalsafety/faq-ed2/page5.htm [Accessed: 17-06-2015].

[19] *IEC61508*. International Electrotechnical Commission.

[20] Texas Instruments. *SafeTI Library, User's Manual*.

[21] D. Pradhan K. Greb. *Hercules Microcontrollers: Real-Time MCUs for safety-critical products, User's Manual*. Texas Instruments. 2011.

[22] Anthony Seely Karl Greb. *Design of Microcontrollers for Safety Critical Operation, User's Manual*. Texas Instruments.

[23] Fort Collins CO (US) Kevin David Safford. "Method and apparatus for recovery from loss of lock step". Pat. US 7,085,959 B2. 2006.

[24] *Lockstep*. Available: http://wikipedia.org/wiki/Lockstep [Accessed: 2-05-2015].

[25] *Lockstep Compensation*. Available: http://en.wikipedia.org/wiki/Lockstep-compensation [Accessed: 02-05-2015].

[26] *Lockstep-protocol*. Available: http://en.wikipedia.org/wiki/Lockstep-protocol [Accessed: 02-05-2015].

[27] Simon Brewerton Trowbridge (GB) Neil Hastie Gloucestershire (GB). "System and method to increase lockstep core availability". Pat. US 2014/0258684 A1. 2014.

[28] *News about Autosar and Iso26262 A new Approach To Vehicle Network Design And Automotive Safety*. Available: http://www.automotive-eetimes.com/en/autosar-and-iso26262-a-new-approach-to-vehicle-network-design-and-automotive-safety.html [Accessed: 21-06-2015].

[29] KranZberg (DE) Bernard Fuessl MooSburg-Aich (DE) Rainer Troppmann. "Delayed Lock-step Cpu Compare". Pat. US 2008/0244305 A1. 2008.

[30] K. Greb S. Cozart A. Longhurst. *An Introduction to Software Development for Functional Safety on TI Processors, User's Manual*. Texas Instruments.

[31] B. Arends P. Metz B. Enser S. Kriso C. Temple. *Functional Safety in accordance with ISO 26262, User's Manual*. ZVEI - German Electrical and Electronic Manufacturers' Association e.V.

[32] *SafeTI System Design Packages for Functional Manual, User's Manual*. Texas Instruments. 2014.

[33] *SafetTI - Hitex Safety Kit (HSK), User's Manual*. v1.2. hitex.

[34] *Safety Integrity Level - Quick Guide*. Sorinc.

[35] *Safety Manual for TMS570LS31x and TMS570LS21x Hercules ARM-Based Safety Critical Microcontrollers - User's Guide*. Texas Instruments, November 2014 - Revised March 2015.

[36] W.; Rothermel G. ; Tingting Yu Srisa-an. "Testing Inter-layer and Inter-task Interactions in RTES Applications". In: *Software Engineering Conference (APSEC)*. 2010 17th Asia Pacific.

[37] Neil Storey. *Safety-Critical Computer Systems*. Addison Wesle Longman, 1996.

[38] *TMS570 LS and TMS570 LC differences*. Available: http://e2e.ti.com/support/microcontrollers /hercules/f/312/p/357489/1254554 [Accessed: 13-06-2015].

[39] *TMS570 wiki page*. Available: http://processors.wiki.ti.com/index.php/Category:TMS570 [Accessed: 07-06-2015].

[40] *TMS570LC43x 16/32-Bit RISC Flash Microcontroller - Technical Reference Manual*. Texas Instruments, May 2014.

[41] *TMS570LS3137 page*. Available: http://www.ti.com/product/tms570ls3137 [Accessed: 04-06-2015].

[42] *TMS570LS31x/21x 16/32-Bit RISC Flash Microcontroller - Technical Reference Manual*. Texas Instruments, November 2012 - Revised 2013.

[43] *Understanding Safety Integrity Level, User's Manual*. Magnetrol.

[44] William C. Moyer Dripping Springs TX (US). "Dynamic Lockstep Cache memory Replacement Logic". Pat. US2012/0272006 A1. 2012.

[45] T. Baghai V. Hilderman. *Avionics Certification - A Complete Guide to DO-178 (Software), DO-178C (Update), DO-254 (Hardware)*. Second Edition. Avionics Communications Inc., 2013.

[46] Anthony Vaughan. *Lockstep microcontrollers advance aerospace electronics safety*. Available: http://www.ecnmag.com/articles/2013/03/lockstep-microcontrollers-advance-aerospace-electronics-safety [Accessed: 10-07-2015].

[47] MichaeI J. Rochford Round Rock TX (US) Davlde M-Santo Munich (DE) William C. Moyer Dripping Springs TX (US). "Error detection and communication of an error location in multi-processor data processing system having processors operating in lockstep". Pat. US 8,090,984 B2. 2012.