9/14/2016

[redacted]

END OF INTERNSHIP REPORT

ANALYSE DE PERFORMANCES BASSE CONSOMMATION DE CONTROLEURS AUTOMOBILES | Prerana SHAMSUNDAR PUNJABI

Institut supérieur de l'aéronautique et de l'espace

Advanced Master's degree in Embedded Systems of ISAE

(2015 – 2016)

Professional Thesis Report

on

ANALYSE DE PERFORMANCES BASSE CONSOMMATION DE CONTROLEURS AUTOMOBILES

(Division **Body & Security**)
(04/04/2016 – 30/09/2016)

Company Supervisor: **Mr. LAFUENTE Romain**
Mr. MULLER Yann
Mr. LIBAUD Romain

School Supervisor: **Dr. HUGUES Jérôme**
Heads of Program: **Dr. HUGUES Jérôme**
Dr. SCHARBARG Jean-Luc
Academic Tutor: **Mr. Arnaud DION**

Prerana SHAMSUNDAR PUNJABI **Continental Automotive SAS – Toulouse**

ACKNOWLEDGEMENT

I take this opportunity to thank people for trusting me and imparting their knowledge to me. Nothing is more precious than knowledge 'It is indeed the stepping stone to successes.

Firstly, I want to thank **Mr. Romain Lafuente -** My Tutor, for selecting me for this internship and providing me this unique opportunity. Next to **Mr. Romain Libaud** for explaining complex technical terminologies with simple examples so that I could relate to the practical scenarios seen in everyday life. This made me to appreciate the development being done in '**Continental**'. Next and Most Importantly to **Mr. Yann Muller** you have had a wonderful patience level throughout this Internship, you have encouraged me not only in developing embedded skills but also indulged my fascination with the hardware and microcontrollers, with learning French and for having fun.

I also want to Thank **Ms. Vasilica Iliescu** a.k.a '**Mari**' as you motivated me to be better and to try harder everyday while working on SPI communication, for the times you supported me even when you were on vacations. For sending over the hardware units when I required it the most. A sincere Thank You to **Mr. Patrick Habouzit** as I know it would have been hard for you to find the necessary time but somehow you managed helping me even with your busy schedule.

Next I want to acknowledge my team mates from '**M**ode **M**anagement **T**eam': **Thomas Labarbe**, **Mathieu Valiente** and **Flavius Raiescu** you all kept a cool, cheerful atmosphere and helped me to be better at my work, there was learning even in our status meeting and review sessions.

I also want to Thank **Mr. Jean-Claude Ferrari**, **Mr. David Cella**, **Mr. Pascal Leman** and **Mr. Jerome Soulas** who taught me how to handle and fix hardware related issues.

Next, I want to appreciate my cubicle mates **Nicolas Jean-Baptiste**, **Jean-Claude De-Pellegrin**, **David Delafosse**, **Noe Figuet** (for teaching me about bootloaders), **Alexandre Sol**, **Thomas Betous** (for taking all the test bench photographs), **Jerome Martinez, Laurent Cardoner** and **Francois Planes** for being a very entertaining, friendly group and for integrating me into various team activities.

I want to Thank my complete Team from **Software Platform** who welcomed me into '**Continental'** with broad minds and tried their level best to teach me French day in and day out.

I also want to convey my Gratitude to my Teachers for their support and encouragement. I especially want to thank **Ms. Claire Pagetti** as you have always made me feel that, even the difficult chapters are very simple to solve and for believing in me. I want to thank **Dr. Jerome HUGUES** for the guidance to take up this internship and **Dr. Scharbarg Jean Luc** for the classes at ENSEEIHT. The lessons were really interesting even if the course was hectic these classes were very useful.

A Special Thanks to **My Classmates** for the immense support I have received from you, you made things amiable and light even when it was getting harder, every game and competition we have had was fun and a welcomed break. It was great being a student again and I am happy to have joined this class.

And last but not the least I want to acknowledge the most important part of my life, **My Family** and **My Friends** who have persuaded me to follow my dreams even if at the times they were not convinced as to what I was looking for.

Thank You to all and I sincerely appreciate what each one of you has done for me.

ABSTRACT

As per Embedded Systems Advanced Master's academic requirement I had the opportunity to perform an Internship in '**Continental AG**' in the Body & Security Division for Software Platform – Mode Management Team. During this period I was motivated to learn the working of microcontrollers and programming for microcontrollers. I was able to apply the knowledge that has been acquired during my studies.

This internship was mainly oriented in Power management features of automotive microcontrollers. The Internship can be distinguished into two sectors:

[I] To implement a logging feature for low power module over Serial Peripheral Interface for a single core microcontroller. In Low Power mode most components are shutdown, hence to have the logging facility in representative environment without the connection of a debugger. This feature would meet the Software Platform team's need for improvement in the software's debug efficiency and validation of the Low Power Module functionalities.

[II] The second part of my internship is to evaluate the Dual Core mode for low power topics and to develop a prototype for dual core systems to be able to have the two cores entering and exiting in Low Power mode. This prototype development shall provide a firsthand feedback to the team, of the different challenges that could be faced while designing the basic software on dual core microcontrollers. In this report some of these design choices and their strengths and shortcomings have been discussed.

We shall discuss the details of each of these implementations in detail in the following chapters.

TABLE OF CONTENTS

1 INTRODUCTION 1

1.1 ABOUT THE COMPANY 1
1.2 TEAM ORGANIZATION 4

2 PROJECT CONTEXT 7

2.1 POWER MODES IN RH850 MICROCONTROLLERS 7
2.2 OBJECTIVE OF INTERNSHIP 9

3 DEBUG FEATURE IN LOW POWER MODE 11

3.1 TECHNICAL CONTEXT 11
3.2 SPECIFICATIONS 12
3.3 HARDWARE SET-UP AND CONFIGURATIONS 13
3.4 IMPLEMENTATION 15
3.5 IMPROVEMENTS 18

4 DUAL CORE PROTOTYPE DEVELOPMENT 19

4.1 TECHNICAL CONTEXT 19
4.1.1 POWER MODE FURTHER IN DETAIL 20
4.2 TASK BREAKDOWN 21
4.3 INITIAL GROUNDWORK AND SET-UP 22
4.3.1 MISSING LINK IN THE INITIAL DESIGN 22
4.4 DESIGN IMPROVEMENTS BROUGHT IN 23
4.4.1 GUIDELINES ESTABLISHED 24
4.4.2 DRAWBACKS IN FIRST APPROACH 24
4.4.3 IMPROVEMENTS NEEDED IN THE SECOND APPROACH 26
4.4.4 FINAL APPROVED DESIGN 27
4.5 HARDWARE SETUP 30
4.6 IMPLEMENTATION 31
4.6.1 LAUTERBACH DEBUGGING 31
4.6.2 DESIGN CHOICES MADE FOR IMPLEMENTATION 33
4.6.2.1 EXCLUSIVE REGISTER CONTROL 33
4.6.2.2 SYNCHRONIZATION BETWEEN THE TWO CORES TO ENTER LPM 34
4.6.2.3 RESET OF THE ‘readyToEnterIntoLowPower’ FLAG 34
4.6.2.4 PROBLEMS DISCOVERED DURING IMPLEMENTATION 36
4.7 RESULTS FROM SIMULATION 38
4.8 IMPROVEMENTS 40

5 EVALUATION AND ASSESSMENT 41

6 CONCLUSION 42

7 APPENDIX – A 43

7.1 MICROCONTROLLER PRESENTATION 43
7.1.1 RH850 MICROCONTROLLERS 43
7.1.1.1 F1L -176 PIN MICROCONTROLLER 44
7.1.1.1.1 PRIMARARY LOCAL RAM AREA 45
7.1.1.1.2 SECONDARARY LOCAL RAM AREA 45
7.1.1.1.3 RETENTION RAM AREA 45
7.1.1.1.4 EDGE / LEVEL DETECTION 45
7.1.1.1.5 INTERRUPT CONTROLLER REGISTERS 45
7.1.1.1.6 CLOCK GENERATION CIRCUITS 45
7.1.1.1.7 CLOCKED SERIAL INTERFACE H - (CSIH) 45
7.1.1.2 F1H -272 PIN MICROCONTROLLER 46
7.1.1.2.1 CPU1 (CORE 1) 46
7.1.1.2.2 CPU2 (CORE 2) 46
7.1.1.2.3 LOCAL RAM 46
7.1.1.2.4 CODE FLASH 46
7.1.1.2.5 INTC1, INTC2 46
7.1.1.2.6 GLOBAL RAM / RETENTION RAM 47
7.1.1.2.7 SLAVE GUARD 47
7.1.1.2.8 BASIC SYSTEM REGISTERS 47
7.1.1.2.9 INTER-PROCESSOR INTERRUPTS 48

8 APPENDIX – B 48

8.1 ADVANCED POWER DOWN CONCEPTS 48

9 APPENDIX – C 51

9.1 INTERESTING TECHNICAL ASPECTS LEARNT 51

10 BIBLOGRAPHGY 54

LIST OF FIGURES

FIGURE—1 OVERVIEW OF CONTINENTAL CORPORATION 1
FIGURE—2 PRODUCTS DELIVERED BY BODY & SECURITY DIVISION 2
FIGURE—3 SOFTWARE PLATFORM TEAM COMPOSITION 3
FIGURE—4 COMPANIES INVOLVED IN AUTOSAR DEVELOPMENT PARTNERSHIP 4
FIGURE—5 AUTOSAR STACK OVERVIEW 5
FIGURE—6 MODE MANAGEMENT TEAM & AGILE SCRUM BOARD 6
FIGURE—7 POWER MODES IN BCM MICROCONTROLLER 8
FIGURE—8 LOW POWER PERIODIC CYCLE 8
FIGURE—9 INITIAL ESTIMATE FOR INTERNSHIP 10
FIGURE—10 ACTUAL SCHEDULE FOR INTERNSHIP 10
FIGURE—11 DEBUGGING FEATURE - MASTER-SLAVE CONFIGURATION 13
FIGURE—12 DEBUGGING FEATURE - F1L CORE BOARD MODIFICATIONS 14
FIGURE—13 DEBUGGING FEATURE - COMPLETE SET-UP 15
FIGURE—14 DEBUGGING FEATURE – OUTPUT FROM-LEFT: OSCILLOSCOPE; RIGHT: DIOLAN 15
FIGURE—15 DEBUGGING FEATURE - EMBEDDED TRACE TOOL BUGS 15
FIGURE—16 DEBUGGING FEATURE - SALEAE LOGIC ANALYSER OUTPUT 16
FIGURE—17 ON LEFT- EMBEDDED TRACE TOOL ON RIGHT – DIOLAN BOARD INTERFACE 16
FIGURE—18 DEBUGGING FEATURE - FINAL OUTPUT 17
FIGURE—19 DUAL CORE PROTOTYPE - COMMUNICATION MECHANISIM 20
FIGURE—20 AUTOSAR SPECIFICATION FOR: ON LEFT–GENERAL ECU SW LAYER; ON RIGHT-LPM 20
FIGURE—21 SEQUENCE TO ENTER AND EXIT LOW POWER MODE 21
FIGURE—22 DUAL CORE PROTOTYPE - INITIAL DESIGN 23
FIGURE—23 DUAL CORE PROTOTYPE - FIRST DESIGN APPROACH 25
FIGURE—24 DUAL CORE PROTOTYPE – SECOND DESIGN APPROACH 26
FIGURE—25 DUAL CORE PROTOTYPE - STATE DIAGRAM – POWER MODES TRANSITIONS 27
FIGURE—26 DUAL CORE PROTOTYPE - STATE DIAGRAM – SYNCHRONIZATION DURING LP TRANSITION 28
FIGURE—27 DUAL CORE PROTOTYPE - SEQUENCE DIAGRAM – TRANSITION [HP TO LP] – CORE 1 29
FIGURE—28 DUAL CORE PROTOTYPE - SEQUENCE DIAGRAM – TRANSITION [HP TO LP] – CORE 2 29
FIGURE—29 DUAL CORE PROTOTYPE - SEQUENCE DIAGRAM – COMMUNICATION BETWEEN CORES 30
FIGURE—30 DUAL CORE PROTOTYPE - SETUP FOR RH850 / F1H 30
FIGURE—31 DUAL CORE PROTOTYPE - DEPLOYMENT DIAGRAM 31
FIGURE—32 DUAL CORE PROTOTYPE - STATIC SMP LAUTERBACH DEBUGGING 31
FIGURE—33 DUAL CORE PROTOTYPE - LAUTERBACH DEBUGGING ISSUES 32
FIGURE—34 DUAL CORE PROTOTYPE - LAUTERBACH DEBUGGING WORKING ENVIRONMENT 33
FIGURE—35 DUAL CORE PROTOTYPE - PROCESS TO ENTER LPM - TOP CORE2, BOTTOM CORE1 38
FIGURE—36 DUAL CORE PROTOTYPE - OVERVIEW OF HP AND LP - TOP CORE2, BOTTOM CORE1 38
FIGURE—37 DUAL CORE PROTOTYPE - TASK SCHEDULING – TOP CORE2, BOTTOM CORE1 39
FIGURE—38 DUAL CORE PROTOTYPE - TIME TAKEN TO ENTER LOW POWER 39
FIGURE—39 DUAL CORE PROTOTYPE - OUTPUT FROM SALEAE LOGIC ANALYSER 40
FIGURE—40 RH850 40NM MCU SERIES FOR VEHICLE CONTROL 43
FIGURE—41 [F1L – 176 PIN] MICROCONTROLLER 44
FIGURE—42 RH850 / F1L SUB-CPU COMPONENT 44
FIGURE—43 PORT FOR SPI COMMUNICATION 45
FIGURE—44 RH850 [F1H – 272 PIN] MICROCONTROLLER 46
FIGURE—45 BLOCK DIAGRAM OF RH850 / F1H 47
FIGURE—46 IPIR_CHN — INTER-PE INTERRUPT REGISTER N (N = 0 TO 3) CONTENTS 48
FIGURE—47 HARDWARE DESIGN FOR F1L – 176 PIN MICROCONTROLLER 49

FIGURE—48 TRANSITION TO STAND-BY MODE 50
FIGURE—49 TIMING DIAGRAM – POWER MODES 51
FIGURE—50 CURRRENT DRAIN OFFERED BY RENESAS MICROCONTROLLERS 52
FIGURE—51 BINARY FILES GENERATED ON COMPILATION 52
FIGURE—52 STRUCTURALISATION OF FLASH BOOTLOADER 53

LIST OF TABLES

TABLE—1 DEBUGGING FEATURE - INITIAL REQUIREMENTS 12
TABLE—2 DEBUGGING FEATURE - TASK BREAKDOWN OF THE USER STORIES 12
TABLE—3 DEBUGGING FEATURE - CORE BOARD PIN CONFIGURATION 14
TABLE—4 DEBUGGING FEATURE - INTERRUPT CONFIGURATION 14
TABLE—5 DEBUGGING FEATURE – API ADDED 16
TABLE—6 DEBUGGING FEATURE - FINAL OUTPUT LOGGINGS 18
TABLE—7 DUAL CORE PROTOTYPE - TASK BREAKDOWN 21
TABLE—8 DUAL CORE PROTOTYPE - GUIDELINES OUTLINED FOR THE DESIGN 24
TABLE—9 SPI PORT CONFIGURATION 45
TABLE—10 POWER MODES HANDLED BY THE MICROCONTROLLER 50

LIST OF ABBREVATIONS

ABBREVIATION	EXPLANATION
AMP	Asymmetric MultiProcessing
AUTOSAR	AUTomobile Open System Architecture
BCM	Body Control Module
BSWM	Basic SoftWare Manager
CAN	Controller Access Network
CES	Continental Engineering Services
Cessar-CT	Continental Engineering Services Solution for AUTOSAR Configuration Tool – developed by Continental Engineering Services for generating AUTOSAR compliant code
CPU	Central Processing Unit, internal core of the microcontroller.
Deepstop	RH850 specific, least power consuming mode, where in power to most of the components can be cut off except for retention RAM.
DMAC	Direct Memory Access Controller
ECUM	Electronic Control Unit Mode Manager
G3M	32-bits RISC core for V850 Series
HALT	RH850 specific, a machine instruction that stops the clock of the CPU core. Other peripherals like CAN, UART are still clocked.
HMI	Human Machine Interface
HP	High Power
HW	Hardware
INTC	INTerrupt Controller
IO	Input / Output
ISR	Interrupt Service Routine
JTAG	Joint Test Action Group
LPM	Low Power Mode

ABBREVIATION	EXPLANATION
LPS	Low Power Sampler
MEV	Mutual Exclusion Variable
OS	Operating System
PE	Processor Equipment (CORE 1 & CORE 2 refer to as CPU1 & CPU2 respectively)
RAM	Random Access Memory
RH	Renesas High End Microcontroller unit Brand
RISC	Reduced Instruction Set Computing
RTE	Runtime Environment
SMP	Symmetric MultiProcessing
SPARC	Software Platform AutosaR Configuration tool – developed by software platform team of as a alternative to Cessar-CT
SPI	Serial Peripheral Interface
STBC	STand By Controller
STOP	RH850 specific, Low Power Consumption mode where in power to some components can be stopped.
SW	Software
SWP	Software Platform
TDI	External interrupt jumper pin. (Specific to Renesas microcontroller design)
UMA	Uniform Memory Access

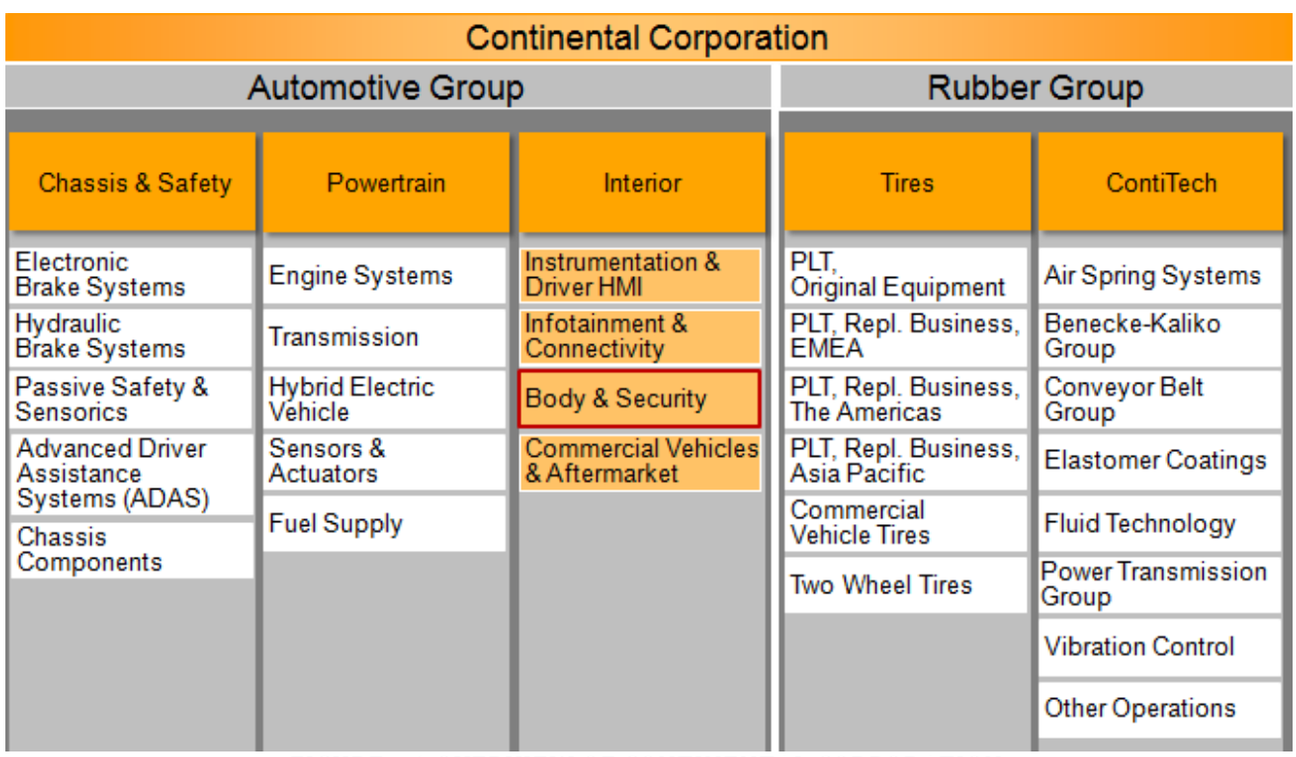
1 INTRODUCTION

"**Necessity is the mother of Invention**". Embedded System is basically a combination of hardware and software, designed to control and handle data communication. These systems are inventions that have made the world to think and bring about different innovations and development in many fields. One such sector is the 'Field Of Automobile Industry'. There has been a drastic change over the years in this field. Automobiles now are built around controllers that act as the brain of the system. Complex microcontrollers and digital signal processors have made their way into the automobile industry. Today many cars and luxury vehicles come with large number of embedded controllers (as many as 100). This automobile market is growing and there is increasing demand and competition between vendors in various sectors.

1.1 ABOUT THE COMPANY

'**CONTINENTAL AG**' ranks among the top five automotive suppliers worldwide. As a supplier of brake systems, systems and components for powertrains and chassis, instrumentation, infotainment solutions, vehicle electronics, tires and technical elastomers, '**Continental**' contributes to enhance driving safety and global climate protection. '**Continental**' is also a competent partner in networked automobile communication.

The '**Continental Corporation**' is divided into the Automotive Group and the Rubber Group, and consists of five divisions:

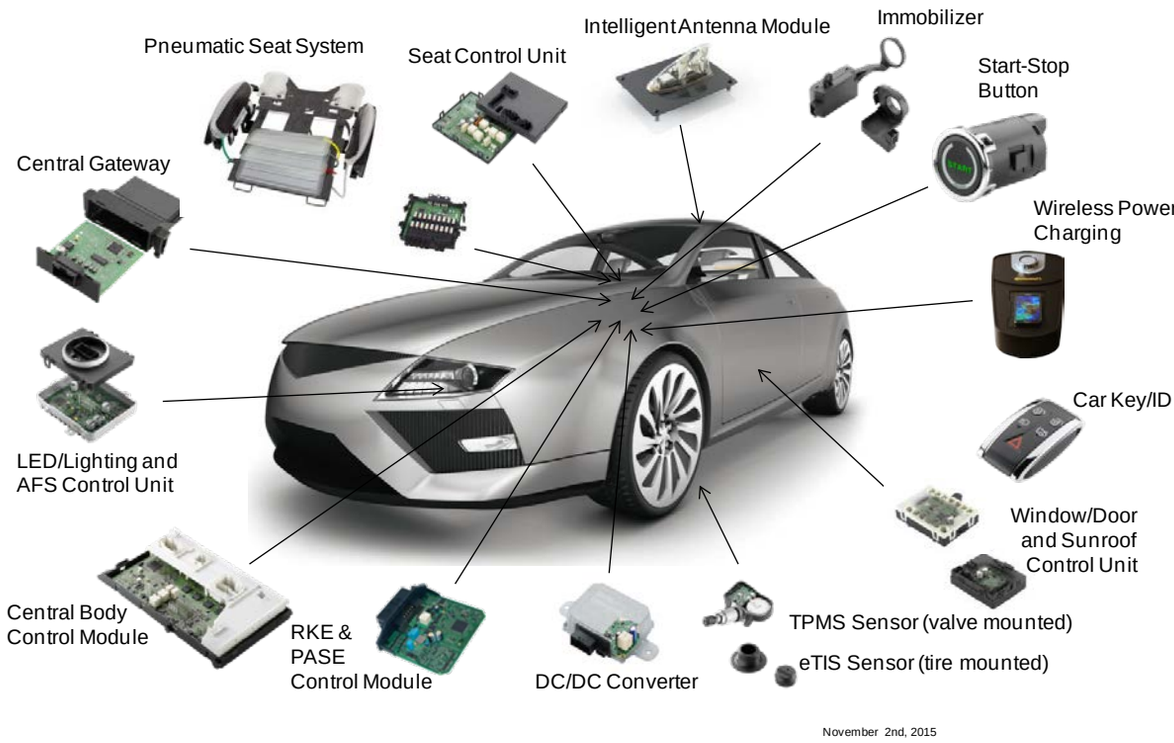


FIGURE—1 OVERVIEW OF CONTINENTAL CORPORATION

Each division consists of business units or BUs, which are responsible for specific product areas. As '**Continental**' develops products in each of these divisions, it allows them to work together and to share expertise and perspectives from several areas while developing new products and solutions. This, in turn, provides them with a huge knowledge base and makes them the '**Market Leader**' that they are.

The department in which this Internship has been served is - '**The Interior Division**' in '**Body & Security**' business unit mainly within the '**Software Platform Team**'.

The Body & Security business unit focuses on the development and production of electronic systems that enable access to vehicles control driving authorization and ensure the availability of security and comfort features within the vehicle.



FIGURE—2 PRODUCTS DELIVERED BY BODY & SECURITY DIVISION

Body Control Modules (BCMs) are central elements of the vehicle electronics. They combine several functions in one housing and control practically all the vehicle electronic basis, comfort and security functions. In the market there exist different variants of vehicles each offering different ranges of functionalities. If we look towards the software aspect of this, as each variant has a different configuration different software would be integrated. Subsequently, the production and maintenance for each variant would be very expensive. To have a cost effective solution for maintaining different variants, AUTOSAR* guidelines were followed. Furthermore, thanks to the scalable architecture of Body Control Modules and their AUTOSAR* compatibility, the BCM can freely be adapted to satisfy customer requirements and harmonize within a specific vehicle in order to respond to market demands worldwide.

'**Continental**' corporation is a multi-national company spread across various locations worldwide.

Please Note: For details on AUTOSAR, please refer to section 1.2

The '**Software Platform**' team is widely spread across many locations as shown in the below figure:

FIGURE—3 SOFTWARE PLATFORM TEAM COMPOSITION

- **Team's Vision**
  - We supply Scalable, Safe, Secure, Reliable, Optimized and Customized SoftWare Platform solutions to fit customer needs

- **Team's Mission**
  - We develop and integrate embedded low layer software
    - **H**igh **E**nd and **L**ow **E**nd **S**oft**W**are **P**latform
    - **AUTOSAR S**oft**W**are **P**latform (**AUT**omotive **O**pen **S**ystem **AR**chitecture)*
  - We deploy our SWP solutions within **Body & Security Worldwide customer projects**
    - More than 150 supported projects from 14 locations
  - We support Engineering Technologies
    - SW Architecture solutions
    - Advanced AUTOSAR tooling (SPARC, Cessar-CT)
    - Integration of generated code from Model Based Development

1.2 TEAM ORGANIZATION

In Automotive the Interior division has a very competitive challenge, as there are various requirements from the end customer hence there are many features that can be integrated in a vehicle. Accommodation of these features depends on the market demand i.e. end customer choice which in turn effects the production and the cost of automobile.

In-order to generalize, there was a need to introduce de-coupled modules (to facilitate easy integration or removal of features) and therefore to have a cost effective production. Hence AUTOSAR* methodologies came into existence.

AUTOSAR (**AUT**omobile **O**pen **S**ystem **AR**chitecture) – is a **Layered Software Architecture** norm that has been established by the collaboration of many companies, for the sole purpose of managing and facilitating the increasing demand and growth of the variety of ECUs included and to satisfy customer requirements in the Automobile Industry. AUTOSAR is dedicated for Automotive ECUs.

FIGURE—4 COMPANIES INVOLVED IN AUTOSAR DEVELOPMENT PARTNERSHIP

An AUTOSAR compliant Interface is defined and included into the architecture for each module and the ECU software components are organized as layers (in other words – AUTOSAR provides a Layered architecture). This results in an architecture that is decoupled.

- The AUTOSAR Architecture distinguishes on the highest abstraction level between 3 software layers: Application, Runtime Environment (RTE) and Basic Software (BSW) which run on a Microcontroller.
- Communication between Application Software Components and access to BSW is done via RTE
- The AUTOSAR Basic Software is divided in 3 major layers and Complex Drivers: Services, ECU Abstraction and Microcontroller Abstraction.
- The Basic Software Layers are further divided into functional groups. Examples of Services are System, Memory and Communication Services.
- The services are further grouped into stacks. The stacks provide a set of features to the Basic Software and to the Application Software through the RTE.

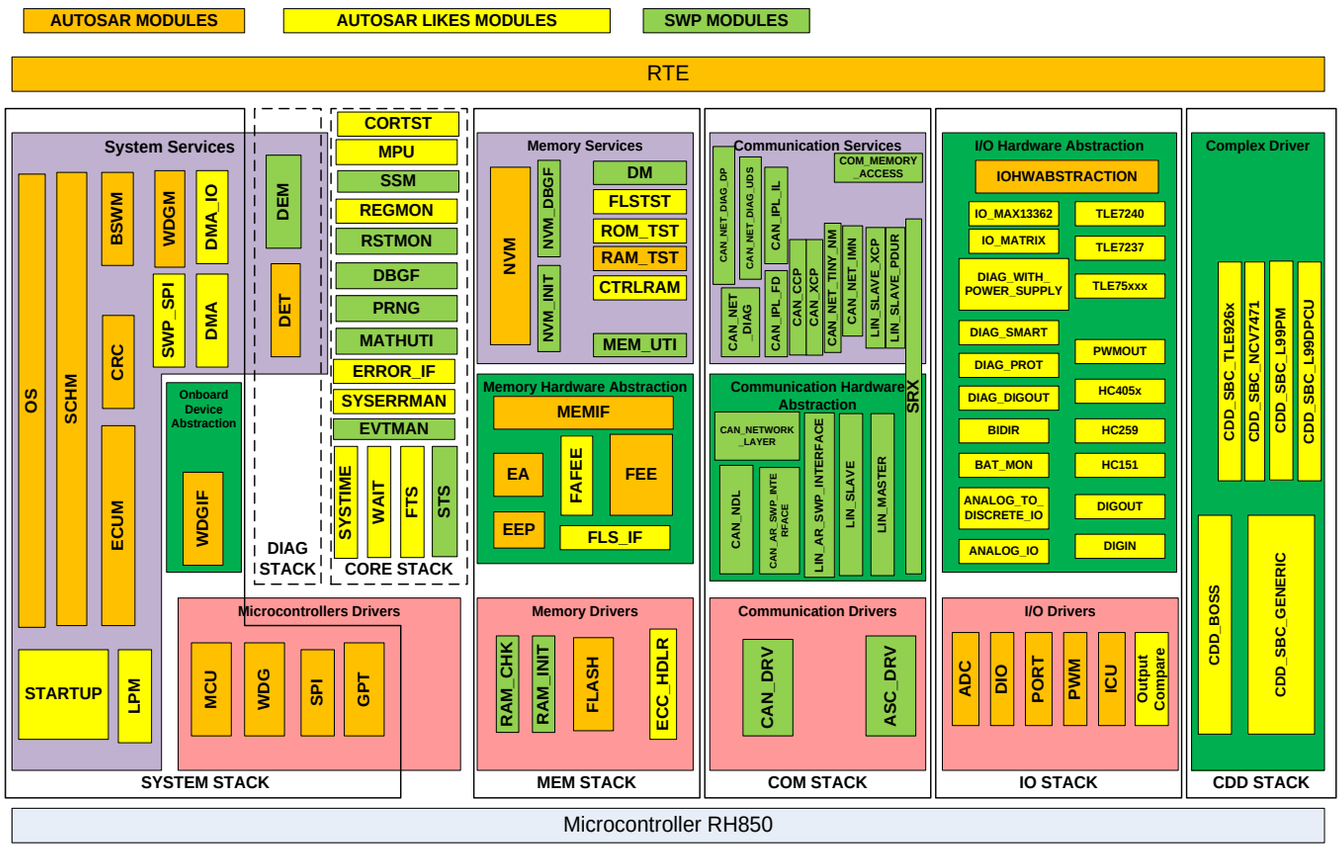
It can be mentioned that on a high level the teams in Software Platform are organized according to the stacks of AUTOSAR. '**Renesas**' microcontroller family - RH850s' AUTOSAR platform is organized into stacks.

Each feature is implemented by one or several modules which can be part of the same stack or of several stacks. The **Basic Software** can be sub-divided into the following types of services:

- **Input / Output (I/O)** - Standardized access to sensors, actuators and ECU onboard peripherals. This stack regroups all the modules which handle all the kind of IOs involved on the platform and the external devices dealing with the IOs

- **Memory** - Standardized access to internal/external memory (non volatile memory). This stack contains all the software modules into the platform which interact with the memories
- **Communication** - Standardized access to: vehicle network systems, ECU onboard communication systems and ECU internal SW
- **System** - The SYSTEM stack regroups all the software modules which have a close interaction with the CPU subsystem of the microprocessor

System Stack is further comprised of many software modules among them is the '**Low Power Manager**'. This module shall be discussed in detail in the following chapters as my internship mainly revolves around this module.



FIGURE—5 AUTOSAR STACK OVERVIEW

To have a methodical and disciplined approach in the product development and testing phases '**Continental**' follows V-Model or V-Cycle process. In this process a top down step wise procedure is applied. In V-Model the projects are executed in phases, right from requirement analysis to Verification and Validation. V-Model is a very well know process in Embedded Domain. In the first part of my internship the 'System Stack' Team followed this process. The V-Cycle model is best for long predictive development cycles, for which requirements are known at the beginning of the cycle and mostly do not change during the development.

This is not the case with Software Platform Team here in '**Continental**' as:

1. The requirements and their priority vary strongly during one development cycle as the projects tend to send their requirements step by step while they receive their own from their customers.
2. In addition new micro-controllers are sourced every two years with hardware specificities discovered during the development cycle and therefore a significant in certainty exists in the required development effort.

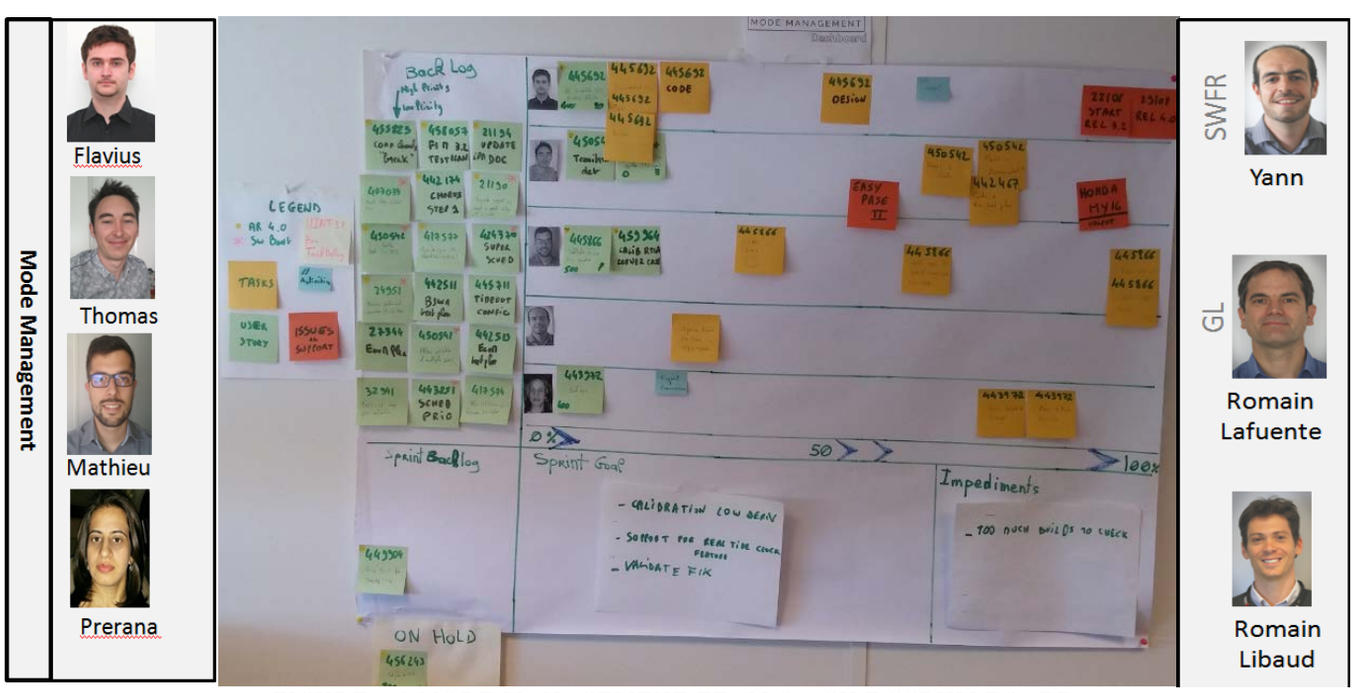
A better suited process '**Agile**' is now being followed, wherein the project requirements and their priority constantly changes which the varying demands.

The concept of Agile challenges that of the V-Model. Here the teams are split into small units, each not exceeding in size from 5-6 (that can be managed easily). The Mode Management Team was formed with this notion. The idea behind forming this team is that the members of the team are actively involved in solving issues related to various mode changes that are present in body control unit. The '**Mode Management Team**' is composed of the ECU Management module, Basic Software Management module and the Low Power Manager modules.

Thanks to the agile methodology it brings about the following main improvements:

- The Concept of Agile is to perform continuous small deliveries with evolving requirements to be delivered that depends on the product demand. The small deliveries are known as sprints.
- The features selected to be delivered in a sprint are assigned to the developers. These features are either termed as Epic or User Stories
- The developer and the team together decide on the complexity of a user story to be developed and the time it might take to deliver it
- These features are broken down into smaller task, and these tasks are tracked to understand the progress made
- Very small, frequent stand-up meetings are conducted to be in-line with every ones work and their progress (hence avoiding lengthy meetings and working efficiently)

We were five members in our team **Mr. Yann Muller** is the **Scrum Master** and the **Product Owner** of this team. The below figure shows the sprint board used to track the team members progress.



FIGURE—6 MODE MANAGEMENT TEAM & AGILE SCRUM BOARD

2 PROJECT CONTEXT

For in-vehicle high-end microcomputer the best choice for body control units would be:

1. A product that could respond to a variety of systems (as there are many vehicles with different facilities that support different demands of the end user)
2. The demand of the automotive industry to reduce CO2 emission is further increasing. This leads also to strong requirement to reduce the current consumption of all electronic modules. The microcontroller should be capable of reducing the supply current in various applications (i.e. wide ranges of **power reducing measures** are available)

I had got the opportunity to work with two of these microcontrollers RH850/F1L (low end, single core) and RH850/F1H (high end, multi core).

2.1 POWER MODES IN RH850 MICROCONTROLLERS

With the increasing number of ECUs used in a car, the power or current required to keep the complete unit powered on and functional at all times is high and expensive.

When the engine is switched off - The vehicle operates on battery and the ECUs in the car are not required to be functional any more. Hence to reduce the current drain through the system the power to most of the ECUs can be cut off and only the essential components can be kept alive. This mode of the system is termed as '**Low Power Mode**'.

The Basic Software Platform supports two main power modes

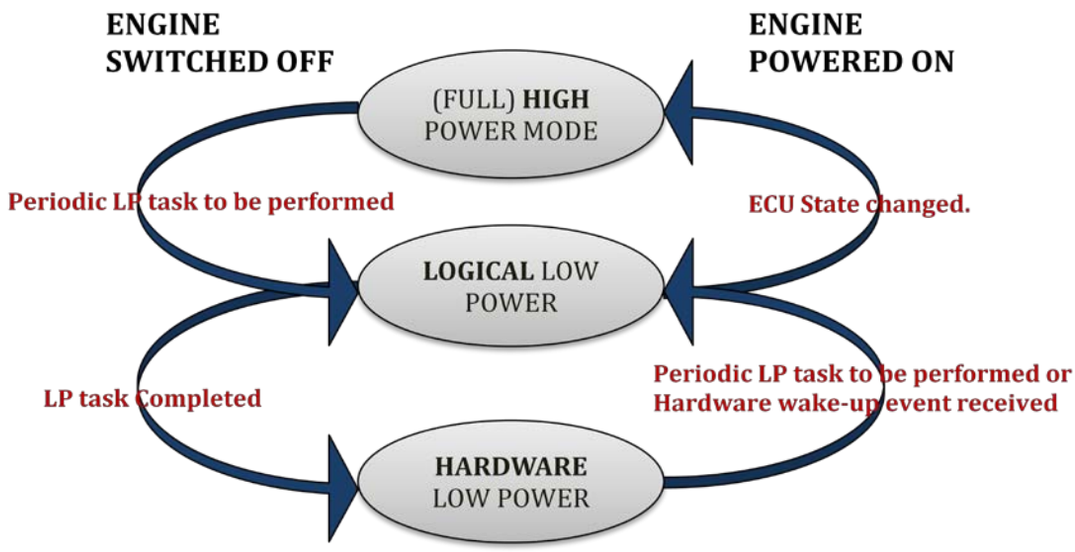
- **HIGH Power Mode (RUN MODE)**: When the Engine is Powered On: Complete - Full functionality of the vehicle shall be operable. In this mode the ECU states are managed by a module '**ECU Manager – EcuM'**. This module is responsible for monitoring the state of the ECU and performing the transition between the various states. And the task in High Power Mode are scheduled by the OS '**OS_Scheduler'**
- **LOW Power Mode**: In this Mode the ECU runs on battery, i.e. when the engine is Switched Off: In this mode reduced functionality of the vehicle is offered. As described in the above chapters to reduce the current through the BCM many components which are not used when the vehicle is switched off are shutdown or the supply / clock gating to these components are stopped. This mode occurs when ECU goes to SLEEP State and is no more needed to be functional. In this mode very minimal functionalities are periodically performed like monitoring of events, such as key press, or external interrupts like key inserted or door knob opened, or theft etc.

The **low power mode** is further divided in two stages:

- LOGICAL LOW POWER / RUN ECO MODE
- HARDWARE LOW POWER MODE

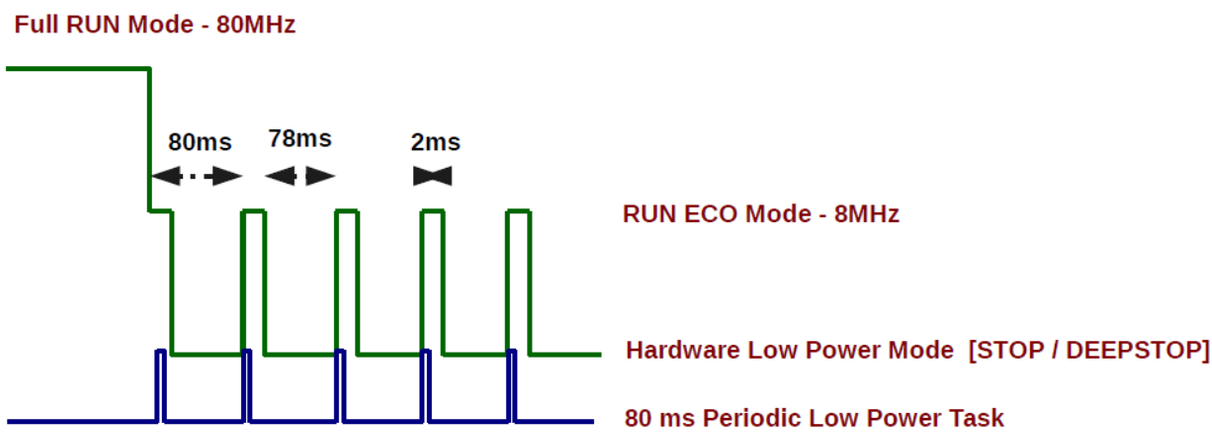
In Low power very few activities are performed with respect to monitoring the state of the vehicle and ensure safety of the vehicle. These activities are performed in low power task. And these Low power tasks are executed in **Logical low power** mode, by a **low power scheduler**.

As in Low Power the CPU is switched-off and the OS is shutdown, a low power scheduler has been developed to be executed instead. Once all the low power tasks are completed then the system enters a mode called as Hardware Low power (the system is shutdown or goes to sleep) and remains there until an interrupt is received to either perform Low power task or to get back to high power.



FIGURE—7 POWER MODES IN BCM MICROCONTROLLER

Furthermore, it can be summarized that the low power activities are performed in a **periodic task** that is scheduled by a Low Power scheduler. And hence the low power task cycle shall be similar to the figure shown below:



FIGURE—8 LOW POWER PERIODIC CYCLE

Please note: In this example the frequency at which full mode execution takes place in is 80MHz

And in Run Eco Mode the execution takes place in 8MHz. By reducing the frequency in Logical low power mode it is ensured that there is reduction in switching loses hence it saves power.

Run Eco mode is where the Low power activities are performed. Once the LP activities are completed the system enters Hardware low power where the system just goes to sleep / halt state.

The LPM Scheduler shall be responsible for the following task:

- It shall manage the LP task execution timings
- It can launch the task with an offset delay that can be configured*
- Puts the controller in Hardware Low Power**
- Shall perform the transition back to high power when the wake-up occurs

Please Note: (1). Refer to section 8 – Appendix - B for detailed information on the ways introduced by '**Renesas**' in order to reduce current drain through the system and the different modes offered.
(2). * Refer to section 9 – Appendix C for details
(3). ** Refer to section 4.1 (note section) for details

2.2 OBJECTIVE OF INTERNSHIP

As mentioned in the previous chapters, this internship was oriented in the '**Mode Management**' sector namely related to the '**Low Power Module**'. There were two distinct objectives in this internship:

- The **First Part** relates to logging of debug statements using Serial Peripheral Interface Communication in LPM:

Goal: To propose, develop and test in the real-time environment a solution to log and store critical information regarding LPM activities for example: mode transitions, resources activation / deactivation and component state, using features like SPI and tools like Embedded Trace tool and DioLAN. This feature should be able to send all the data to an external PC for further analysis offline.

Merit of this Goal: Accomplishing this objective successfully shall facilitate the team to validate Low Power Mode features. This is important as in Low power most of the components including communication buses are shutdown to reduce current drain of the system. Hence a light weight logging mechanism in a representative environment (i.e. in an environment that represents the real system without an addition of a debugger), would help the team to reproduce and monitor the state changes, activity timings, task and interrupts in low power mode.

The Goal was further divided into the following sub-goals:

- Setup a working environment for F1L microcontroller with LPM, SPI and Debug configurations enabled with embedded tracing facility in High power
- Establish SPI communication between F1L microcontroller and the DioLAN board.
- Implement logging feature in Low Power, test the software to ensure correct frames shall be received and entering hardware low power was not prevented

- The **Second Part** relates to the development of a prototype for a dual core microcontroller to enter and exit low power mode of the BCM.

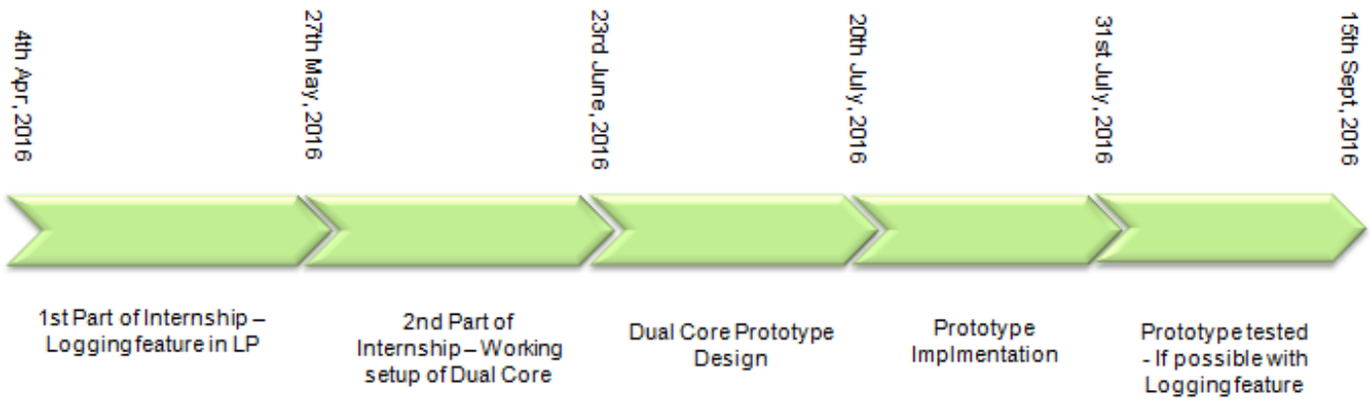
Goal: To propose a prototype design wherein several synchronization and activity sharing mechanisms must be analyzed to allow the two cores to enter and exit low power mode and to implement this solution.

Merit of this Goal: This prototype shall form the initial design for the actual development for a commercial product that shall aim not only to improve the processing speed and current drain through the body control unit but also reduce the cost for production.

- Analysis of the initial LP requirements and existing design for Dual Core.
- Setup a working environment for F1H microcontroller with LPM and dual core configuration enabled and modify the Lauterbach scripts used to flash the software.
- Evaluate the different design approaches to synchronize the communication, status of both cores
- Implement the prototype and test the code for reliability under different scenarios

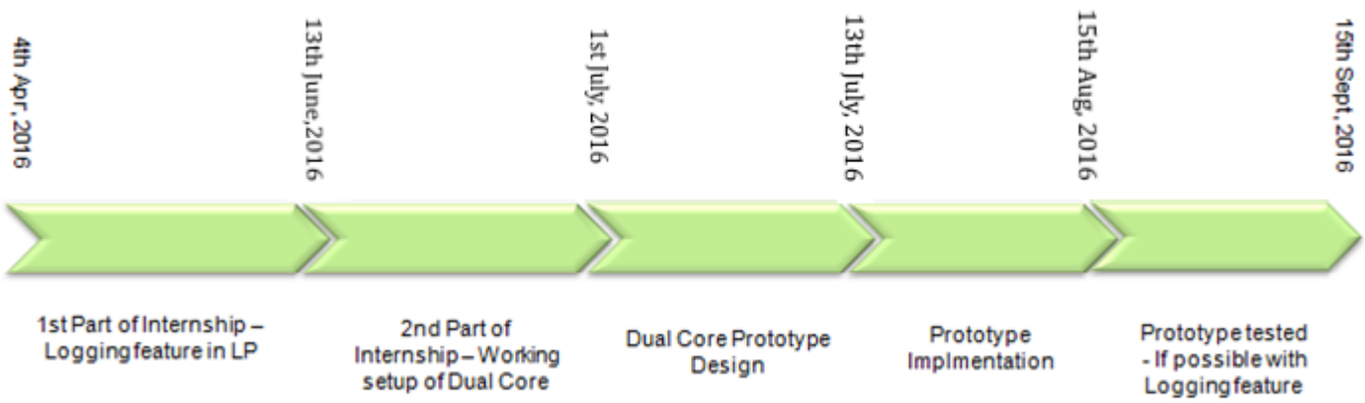
Please Note: *These sub-goals were further broken down into simpler task and were executed in a task by task approach

Though in Mode Management Team Agile methodologies is followed, wherein there are no defined timelines but sprints are mentioned, as my internship is time bound a rough estimate of sub-goals was done to be able to keep track of efforts and time. The below schedule was outlined earlier in the Internship.



FIGURE—9 INITIAL ESTIMATE FOR INTERNSHIP

Additionally the optional feature of adding logging feature in the dual core prototype has not been implemented as it was of a lower priority; however other mechanisms have been utilized to validate the implementation done for the dual core prototype.



FIGURE—10 ACTUAL SCHEDULE FOR INTERNSHIP

3 DEBUG FEATURE IN LOW POWER MODE

3.1 TECHNICAL CONTEXT

Due to the complexity of the embedded real-time application, it is essential to include a debugging feature that can trace and validate the features and state transitions in a system.

In BCM there are two different Schedulers:

- The **OS Scheduler** that executes during High Power (during this period the system is configured to execute with all features enabled)
- The **Low Power Scheduler** that shall be enabled to execute once the board is in low power

Each of these modes has their own state machines and contains different tasks that are scheduled. The tasks utilize different resources for their completion. The microcontroller's architecture is very complex that includes many different units, for instance there are 4 oscillators, 5-6 communication buses and 5-6 timers (refer to section 7.1.1). To be able to trace each of these components state and to utilize all features in an efficient manner would be challenging.

In High Power as all the system components are operable, there are many logging and debugging solutions available that are already been developed. For instance – JTAG or Nexus communication, CAN bus communication (Diagnostic tool, Calibration tool, Embedded Trace tool that uses Can communication for logging in High Power). But to implement this feature in Low Power it is more challenging as the communication buses are not available in Low power and many ports and components are disabled.

<u>THE NECESSITY FOR DEBUGGING FEATURE IN LOW POWER</u>:

- To monitor the state change, during High power to Low power transition
- Various resources (or components) that have been activated or requested during low power mode and their duration of usage (this is essential as in low power as most of the unused components are to be cut-off to reduce the current drain through the system). Hence after a resource is utilized if no other low power task is using the resource it needs to be turned off, therefore there exists a need to monitor these resource status
- The duration for which each low power task runs

Information related to these aspects can help the team to **validate low power features** that have been implemented and to increase the efficiency of the system by monitoring the logging under different scenarios of execution modes.

The already existing methods and tools cannot be used in low power mode as

- The supply to the CPU is cut off in HW Low power
- Having communication buses (e.g. LIN, CAN) during low power is not representative due to additional power consumption it brings
- The supply to the RAM (except for Retention RAM) is cut off in Deepstop* mode
- As the diagnostic tools use many essential components of the system, this would prevent the system from entering low power, hence this tool cannot be used in Low Power

Please Note: Refer to section 8 – Appendix - B for detailed information

Therefore it was proposed to use a light communication bus like SPI to log short messages as many projects are already using SPI for communication during Low Power. The messages transmitted by this SPI would be interpreted using tools like Embedded trace tool (that has been developed by '**Continental**').

3.2 SPECIFICATIONS

Sl. Number	Specification
REQ-1	SPI communication feature can be utilized for implementing debugging feature in LPM
REQ-2	A master Slave configuration shall be maintained to implement this feature where in the logger is the slave the microcontroller is the Master
REQ-3	It shall be ensured that when the SPI channel is being utilized for logging (i.e. logging is ongoing), the system should not enter Hardware Low Power
REQ-4	It shall be ensured in case there is no activity in logical low power mode the system should enter Hardware low power mode
REQ-5	The system time could be inaccurate (for instance in deepstop* mode the system time is not available). Hence a alternate mechanism shall be explored

TABLE—1 DEBUGGING FEATURE - INITIAL REQUIREMENTS

As '**Agile**' process was followed the goals listed in section 2.2 were broken down into simple tasks as listed below:

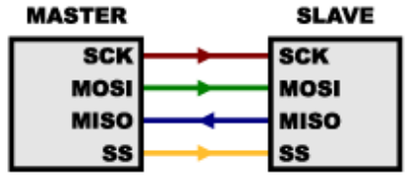
Task	Description
1	Setup the F1L microcontroller with the build environment and execute the code
2	Enable all configurations required ->. Enable option to enter LPM ->. Enable debug feature ->. Enable the use of SPI in LPM ->. Enable the use of Embedded trace tool
3	Understand the usage of Embedded trace tool and logging mechanism used in high power mode and learn how to use the emulator E1 for debugging
4	Connect the dioLAN as the slave and transmit message to this board from microcontroller in HIGH Power using SPI
5	Configure the microcontroller to enter Low Power mode - Stop / Deepstop*
6	Establish communication between the two boards in Low Power mode
7	Implement logging feature in Low Power and ensure the traces are captured and displayed on to the Embedded Trace tool
8	Ensure that with the current implementation the Low Power feature is working correctly. (REQ-3 and REQ-4) are taken care off

TABLE—2 DEBUGGING FEATURE - TASK BREAKDOWN OF THE USER STORIES

Please Note: *The Power modes are explained in the section 8 – Appendix – B, it is not explained here as most implementation and test were done in Stop Mode.

3.3 HARDWARE SET-UP AND CONFIGURATIONS

To accomplish this task a Master - Slave configuration had been established wherein the microcontroller is the Master it sends debug information as SPI **frames** to the DioLAN board that is operated as Slave and shall receive these SPI frames and record them.



FIGURE—11 DEBUGGING FEATURE - MASTER-SLAVE CONFIGURATION

As mentioned in the above chapters a step by step by task implementation approach was adapted to accomplish this part of the internship. First task was to set up, configure and execute the existing application on F1L microcontroller board, for establishing SPI communication and to have a mechanism to enter low power mode.

The configuration of this set-up was challenging as these tools (DioLAN board, SPI comm. for logging, embedded trace tool) were never used together by anyone, the set-up was to be done by referring to the datasheets and available user manuals. Hence there were many trial and error mechanisms done to get the final set-up up and running. And thanks to the '**Continentals**' global network, we were able to the final results.

- First steps was to use configuration tools like SPARC* to enable features listed in the above table.

Note: As the basic software is highly configurable standardized tools like SPARC* and Cessar-CT* are used to enable and disable features into the code. These tools are internally developed by '**Continental**' and follow AUTOSAR specifications.

Later the microcontroller datasheet was referred to enable SPI, Master out Slave in (MOSI) and Chip Select pin on the Hardware. For this the hardware was soldered to enable the appropriate ports on the board. The below diagram show the hardware modifications done on the microcontroller board and the DioLAN board. (Refer to Figure 12)

The main hardware related changes done on the board were:

1. Enabling of the Chip select port on the core board
2. Enabling of the push button pins to configure external interrupts to enter Low Power mode
3. Shorting of CAN communication ports so that it could be disabled in low power mode

Next as the DioLAN board required a supply voltage of 3.3 volts but the supply to the board is 5 volts appropriate registers had to be introduced to ensure correct voltage across the circuit.

470 Ω was introduced as shown in figure 13. The E1 Emulator is used to flash and debug the application.

The RH850 / F1L core board has the SPI signal configurations different to that of the Slave board (DioLAN board). For example the DioLAN board requires the chip select pin to be active low and when triggered to be high. Similarly, the MOSI connection from the core board to DioLAN board has different configurations hence it was necessary to read the microcontroller datasheets and the DioLAN board specifications to set up the two boards correctly and establish communication.

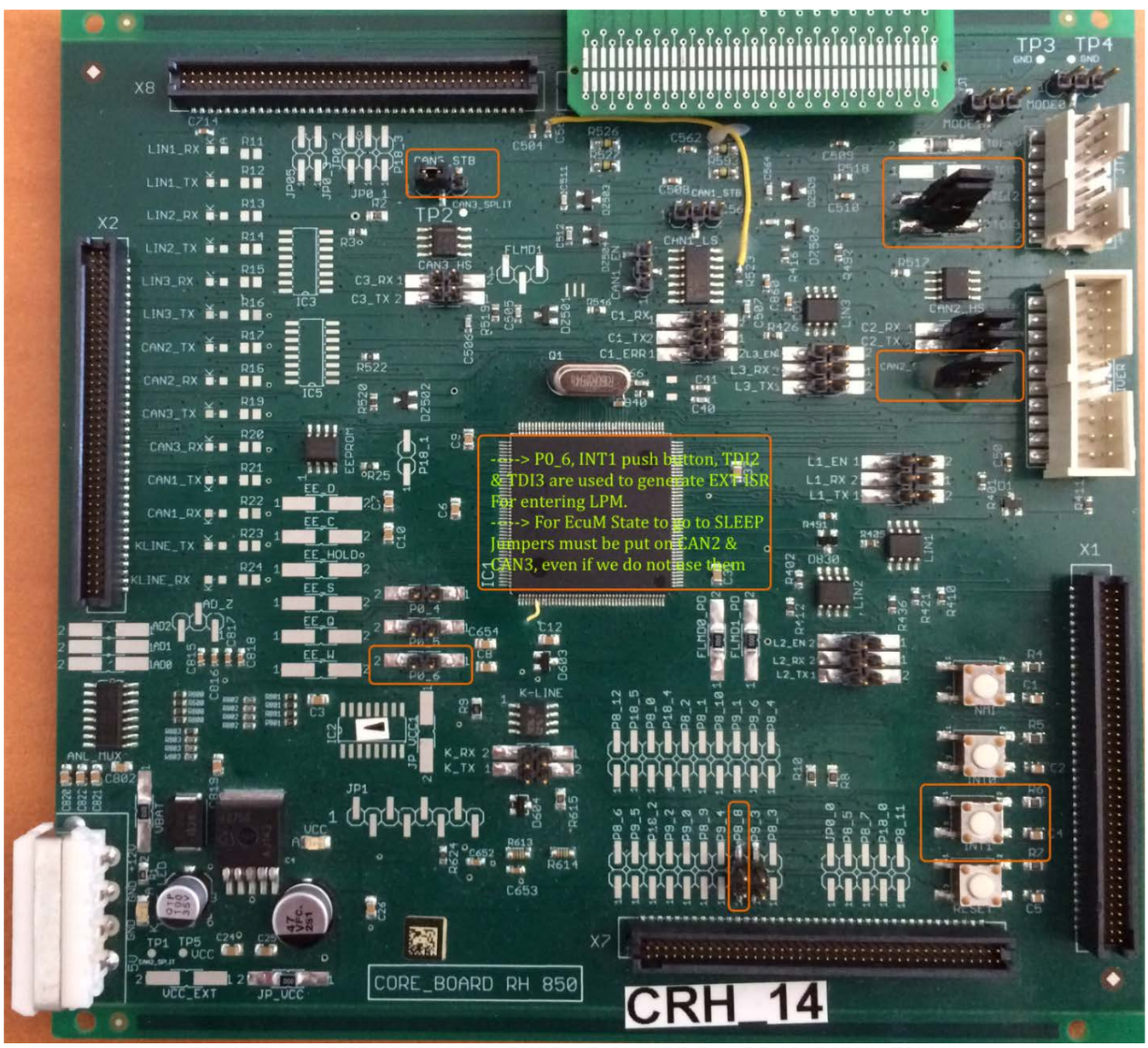
Pin	Additional Functionality	Description	Pin Mapping
P8_8	Chip Select	Is used to select the active Slave line (active low, output from Master)	Core board P8_8 shorted
P11_5	MISO	Master In Slave Out (output from Slave)	X5 Pin 14
P11_6	MOSI	Master Out Slave In (output from Master)	X5 Pin 13
P11_7	Clock	Serial Clock (output from Master)	X 5 Pin 12
	GND	Ground	X5 Pin 1 or 2

TABLE—3 DEBUGGING FEATURE - CORE BOARD PIN CONFIGURATION

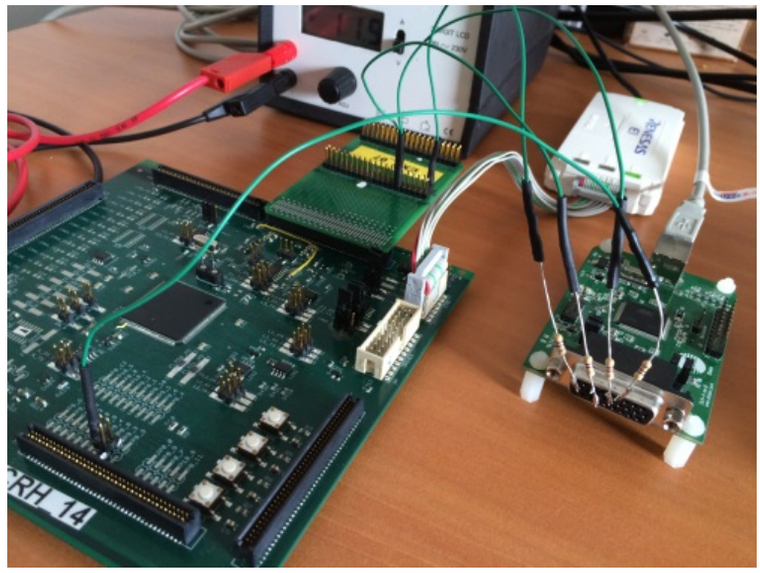
The push buttons in the core board (as indicated above) are enabled and configured in order to trigger an external interrupt. Used here for entering Low Power Mode.

RH850/F1L			Button	
F1L		Jumper	Button	Jumper
INT1	P0_6	TDI3-2	INT1	TDI3-1

TABLE—4 DEBUGGING FEATURE - INTERRUPT CONFIGURATION



FIGURE—12 DEBUGGING FEATURE - F1L CORE BOARD MODIFICATIONS



FIGURE—13 DEBUGGING FEATURE - COMPLETE SET-UP

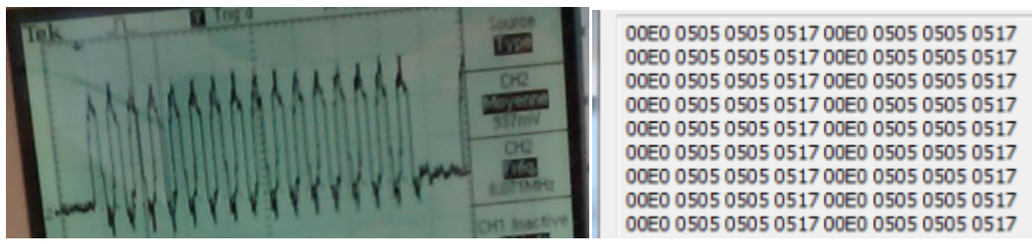
The serial clock level shall be specified by setting of the register Serial Communication Clock registers, care should be taken that the settings match the ones expected by the dioLAN board.

3.4 IMPLEMENTATION

For establishing communication a Core Stack module – 'the debug Module APIs' where used which had 2 options:

1. To send immediate frame of data less than 8 bytes – used to send - System Information
2. To send a buffer (array) of frames, once the array filled – used to send – debug information

To test the connections the first option was chosen and static data was sent on MOSI line to observe the frames received at the slave terminal.



FIGURE—14 DEBUGGING FEATURE – OUTPUT FROM-LEFT: OSCILLOSCOPE; RIGHT: DIOLAN

After setting up the board and signals correctly, the next challenge was to get debug logs to work in High Power using SPI as in general scenario CAN communication was used for this feature.

Another tool that has is developed by '**Continental**' Core stack team was be used to interpret these frames in context of the Basic Software Application.

As this tool is a continuous development tool there were bugs discovered in the tool, where in the buffer used to send immediate data, would get corrupted and the SPI frames that were transmitted were not getting captured in the logs.

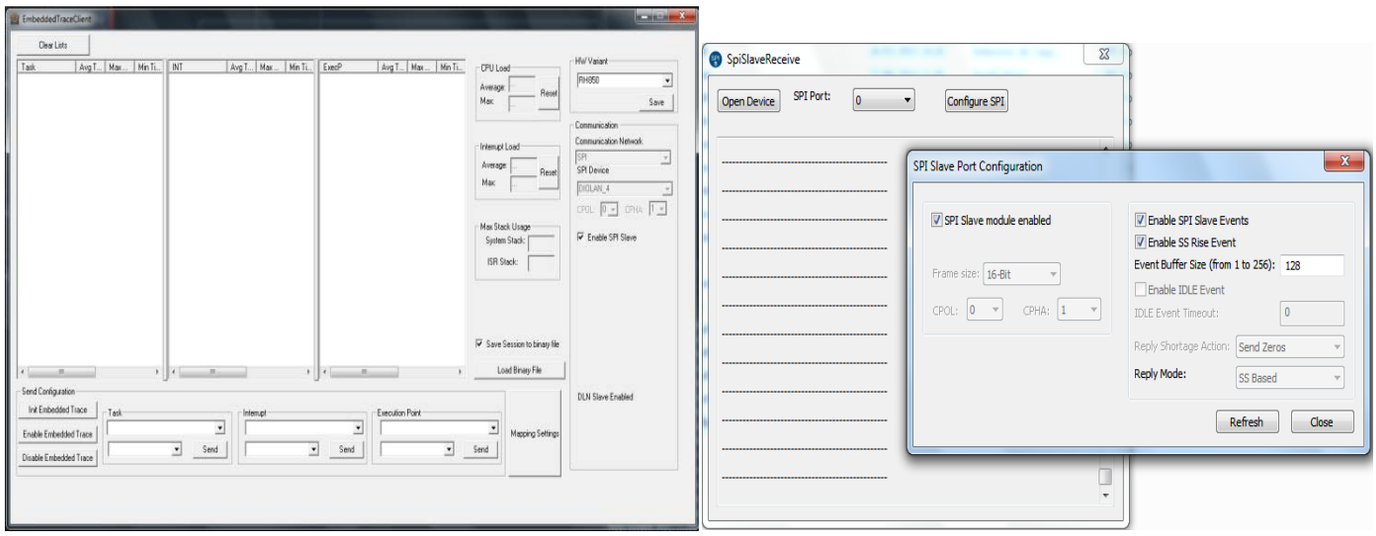
FIGURE—15 DEBUGGING FEATURE - EMBEDDED TRACE TOOL BUGS

In order to validate the output logs, an additional tool a '**logic analyzer**' was utilized. This scope can capture signals and these signals can be later interpreted as the messages.

FIGURE—16 DEBUGGING FEATURE - SALEAE LOGIC ANALYSER OUTPUT

Once the Logs were obtained in high power, the logging mechanism in Low power was implemented.

PRIOR TO THE FEATURE IMPLEMENTATION [VIEW IN LOW POWER MODE]



FIGURE—17 ON LEFT- EMBEDDED TRACE TOOL ON RIGHT – DIOLAN BOARD INTERFACE

For which debug module APIs were utilized. As discussed earlier, the aim of this logging feature was:

1. To clearly identify what were the LP task being executed
2. The time duration for which each LP task runs
3. To identify the resources being used in LP
4. To identify the interrupts received in LP
5. The state changes observed during the transitions from HP –to- LP and vice versa

Events to be Tracked	Implementation
Task Execution Time and Ids	Debug APIs: 1. DbgF_PreTaskHook 2. DbgF_PostTaskHook Were introduced before and after scheduling the LP task, Additionally task Ids had to be tweaked as the OS configuration files did not support LP

Resource Utilization	DBGF_EXECPOINT_TEST_8B_START(Lpm_Resource) // for activation of resource
	DBGF_EXECPOINT_TEST_8B_STOP(Lpm_Resource) // for deactivation of resource

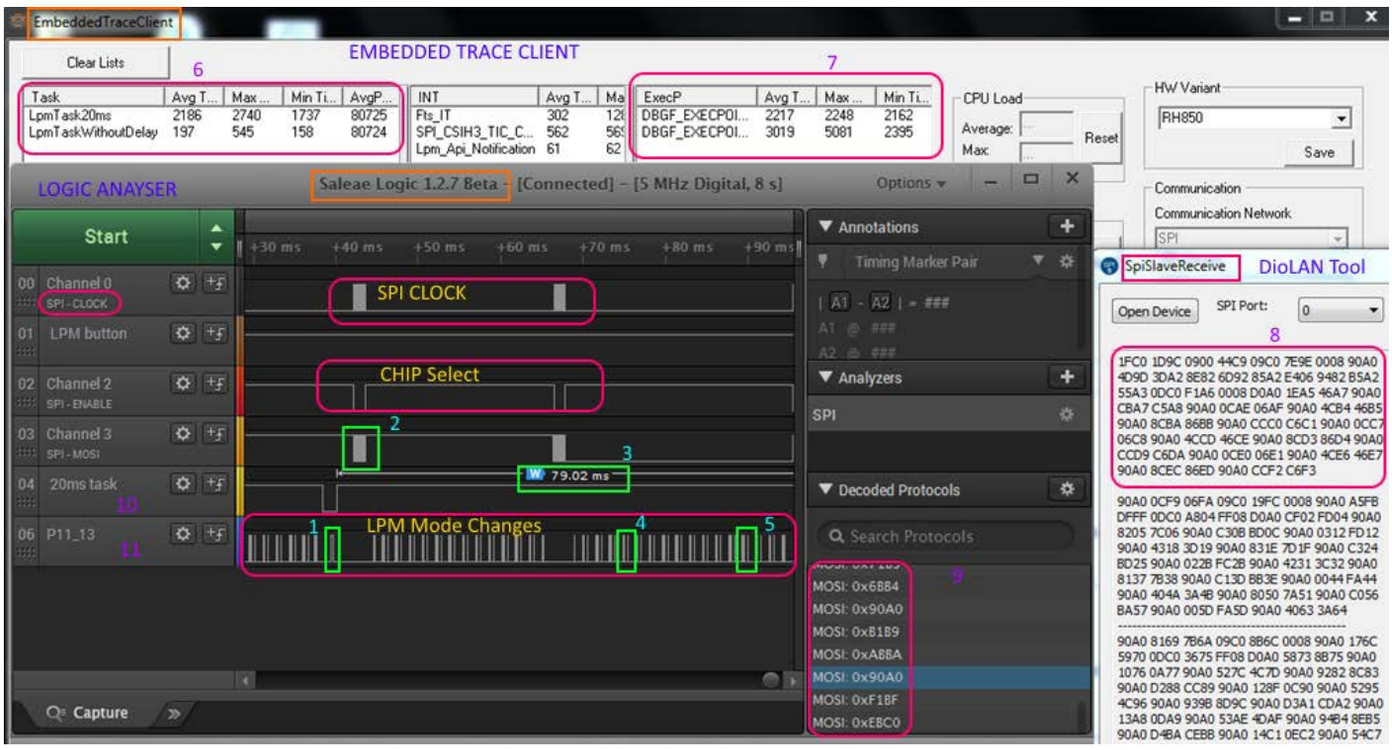
Interrupt Execution	Debug APIs: 1. DbgF_PreISRHook 2. DbgF_PostISRHook For tracking interrupts being executed

TABLE—5 DEBUGGING FEATURE – API ADDED

Functions were identified to add appropriate APIs, but even if the logs were added the output was not visible in LP. After further debugging and analysis it was realized that the debug module APIs did not support in low power mode. Hence the next task was to extend the debug module to facilitate using its API. It was also realized that the OS configuration files were to be modified as the LP task were not added in the OS configuration files. (**Note**: This made sense as in LP, the OS is not used, but the Embedded Trace Tool took the OS configuration files as its input to interpret the logs), the tool was tweaked to parse the MOSI frames received as task.

To ensure that the system does not enter hardware low power in between the ongoing logging activity, the SPI channel was requested as a resource by adding an LPM_Resource_User for requesting the SPI channel and this LPM_Resource_User was released once the logging activity was completed.

The below figure combines the output observed on the three different tools 'The Embedded Trace Client View', 'The Logic Analyzer View' and 'The DioLAN Board View' to be more optimized.



FIGURE—18 DEBUGGING FEATURE - FINAL OUTPUT

Reference for the numbers indicated in the above figure:

[1] - P11_13 Refers to the Port used for the output, it signals are captured from The `LPM_SetMode()` function that is responsible for mode / state changes in Low Power.

`Here at Instance 1 the Logical Low Power Mode is entered`

[2] - Channel 3 – SPI-MOSI refers to the signals received at the Master Out Slave In Port (`MOSI`)

`The SPI Frames are received from the Microcontroller board at the DioLAN end`

[3] - Refers to the LPM Period set for this Test

`The signal was introduced where the code enters just once in a single low power period`

[4] - Double glitch signal has been added in **LPM_SetMode()** function when the check for entry into Hardware Low Power is true

[5] - Fast Timer Task activities wake up the microcontroller from Hardware low power mode. The signal was added in **LPM_SetMode()** under the case where **Run_Eco Mode** is true.

[6] - Embedded Trace Client logs. The SPI Frames were interpreted to refer to the appropriate events. The time stamp indicates:

- The Average time to complete the task
- The Maximum time taken for task completion
- The Minimum time taken for the task to complete and
- The period between which the task/event is entered

[7] - The Debug Module APIs used to trace the entry and exit into a task or service routine

[8] - The MOSI Frames received at the DioLAN Board Interface

[9] - The MOSI Frames received by the Logical Analyzer Scope

[10] - 20ms task – signal refers to the Period Task in Low Power mode. This signal was added to monitor the entry into Run Eco Mode

Logs similar to as shown below are obtained:

Time (in CPU Ticks)	Event Occurred (Task / ISR)	Event Details	
32426	SPI_CSIH3_TIC_CAT2_ISR	START	
32929	SPI_CSIH3_TIC_CAT2_ISR	STOP	INT:0
33538	Fts_IT	START	
33789	Fts_IT	STOP	INT:0
34549	Lpm_Api_Notification	START	
34611	Lpm_Api_Notification	STOP	INT:0
34744	Fts_IT	START	
34995	Fts_IT	STOP	INT:0
35195	DBGF_EXECPOINT_TEST_32B	PARAM:640108	START
35233	LpmTask_LpmTask20ms	START	
35787	Fts_IT	START	
36038	Fts_IT	STOP	INT:0
36827	Fts_IT	START	
37078	Fts_IT	STOP	INT:0
37706	DBGF_EXECPOINT_TEST_32B	PARAM:640108	STOP
37738	LpmTask_LpmTask20ms	STOP	INT:502
37789	LpmTask_LpmTaskWithoutDelay	START	
37869	Fts_IT	START	
38120	Fts_IT	STOP	INT:0
38329	LpmTask_LpmTaskWithoutDelay	STOP	INT:251

TABLE—6 DEBUGGING FEATURE - FINAL OUTPUT LOGGINGS

3.5 IMPROVEMENTS

- With this feature, the team can log using Logic Analyzer and Embedded Trace client and DioLan tool; it will be a good idea to use python script to compare the outputs that are received. This can help for the calibration of the timers.
- Enhancements on Embedded Trace tool to include LPM Task
- Test with smallest LPM Period (currently the test were done on LPM Period = 80ms)
- There were events of Fast Timer Task that woke up the microcontroller the Core team shall invested the reason for continuous use of this task
- Test in DeepStop* Mode with different timers enabled (along with inaccurate timers enabled), as it for such features there would be need to calibrate/ re-calculate the time before logging.

❖ The microcontroller has many timers that refer to different oscillators for synchronizing its time. Different projects use different configurations and it is necessary to test a feature in all possible project configurations.

4 DUAL CORE PROTOTYPE DEVELOPMENT

4.1 TECHNICAL CONTEXT

The growing number of customer functions and the increased function complexity has lead to, up to 80 ECUs in the car. As a result of it was decided to "**merge**" ECUs together needing an increasing performance of the microcontroller.

The most optimized way to increase CPU performance with current technologies is to use **multi-core architecture**.

- It optimizes price
- It optimizes power consumption

CPU **performance increases** with the **quantity of parallelized code.** But an overhead is introduced for communication between the cores, care should be taken such that the communication mechanism must not create a bigger CPU load than the gain brought by the second core.

Hence a prototype shall be developed to realize all the constraints and challenges that might be faced during dual core product development for Body control unit.

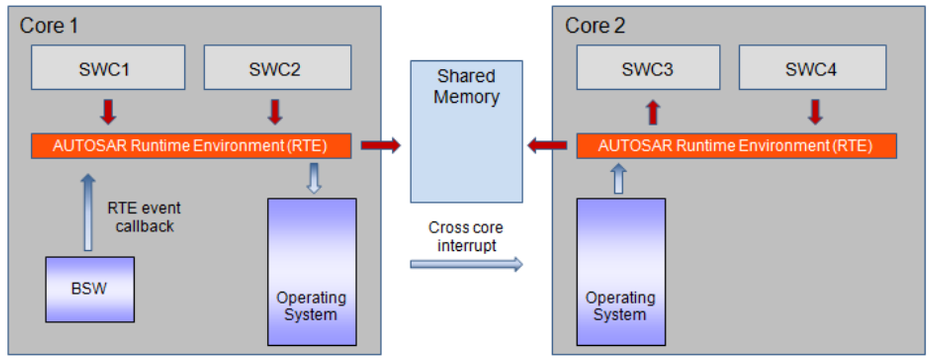
As an initially start, a prototype design for executing the two cores in High Power was already available, my task was to implement safe, synchronously entry and exit of the two cores into LPM. The existing prototype was developed on basis of following specifications / design constraints:

- F1H has one source clock and one Stand-by Controller (STBC) that is assigned to the Master core; hence if the Master core stops its operation (enters HW LP) the slave core automatically stops (as it is triggered by the same source clock and operated by the same stand-by controller).
- There is only one OS Counter available on the Master Core, this specify that only Master Core can initiate or trigger task to be initiated. This also indicates that the slave core shall have no information that a task is about to be triggered on its core
- Application software shall be distributed on both cores
- ECUM – ECU Manager shall be available only on one core (i.e. Master Core)
- A shared RTE shall be used
- The Basic software shall be present on one of the core (i.e. on Master Core)

Note:

1. The OS counter refers to the timer that ticks based on accurate oscillators and is responsible for triggering an alarm or initiating the task on the core
2. Stand-by controller is responsible for the mode changes in the core. All HW wake-up events are located on the stand-by controller. It also controls the switching on and off of the ISO* area
3. When core 1 decides to enter LP it will stop the source clock (basically writes into the register controlling the oscillators) and then indicates the STBC that it is ready to enter Low Power. The STBC then check for STOP / DEEPSTOP* option and appropriately perform clock gating or shutting down of the ISO area.

Communication between both cores shall look similar to as shown in the below figure



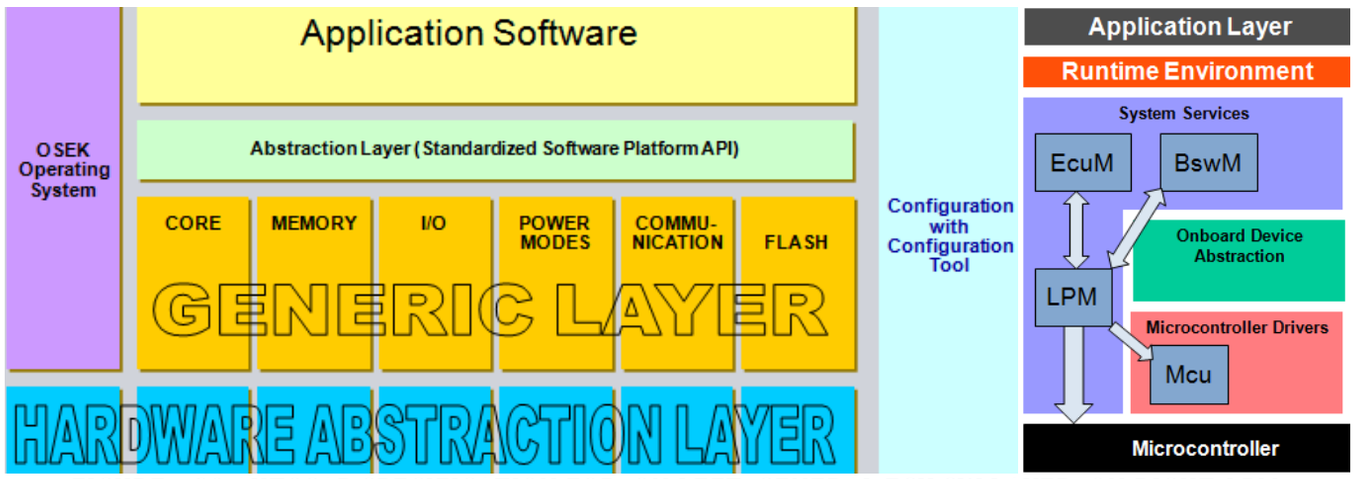
FIGURE—19 DUAL CORE PROTOTYPE - COMMUNICATION MECHANISIM

4.1.1 POWER MODE FURTHER IN DETAIL

In section 2.1 the low power concept has been described but how does it work in such implementation scenario and what is the software architecture.

From the earlier chapters and the sections 1.2 it is understood that the Basic software is a layer just above the underlining hardware unit. It forms an interface layer (**GENERIC LAYER**) between the Application Layer and the Microcontroller. During High power many application tasks are scheduled and when these application tasks need to interact with the microcontroller or vice versa, it calls the appropriate APIs of the basic software.

The LPM belongs to the System Stack and is a part of basic Software and constantly interacts with the EcuM and MCU Modules.



FIGURE—20 AUTOSAR SPECIFICATION FOR: ON LEFT–GENERAL ECU SW LAYER; ON RIGHT-LPM

EcuM is the State Machine module that reflects the current ECU state. All application tasks are users of the ECUM (when an application task is executed it runs as a user of ECU), hence the EcuM can monitor the tasks of all applications and switch between different states of the state machine.

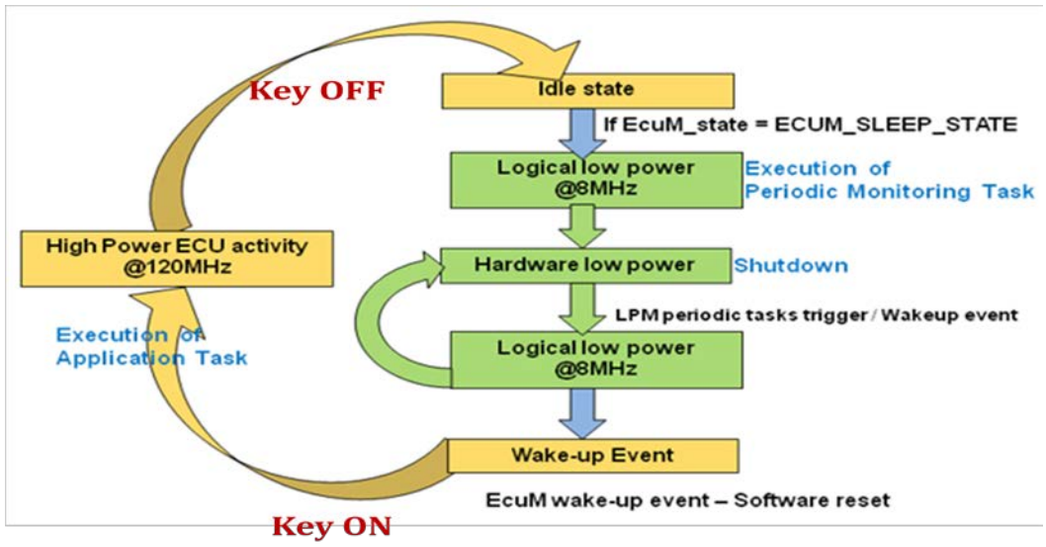
When the vehicle is switched off, the execution of the application tasks are completed and then the EcuM state is set to SLEEP mode. On this event the basic software triggers an indication to enter low power mode in order to reduce the current drain.

Hence the LPM module has constant interaction with the EcuM in-order to monitor the ECUs state and with the MCU in-order to monitor the wake-up indications and for adapting the clock and frequency of the unit to the power mode.

Once the application tasks have finished their execution and the core is not running any other high power task, the scheduler triggers the Idle Task (this task is an auto run task, with low priority).

The system remains in idle loop unless there is an EcuM state change. At this point if there is no other task running and the ECU enters into a SLEEP STATE.

If the ECU state is SLEEP the system initiates the process to enter Low Power, here the OS is stopped to be clocked and **'LPM_Scheduler'** takes over. The system remains in Low Power until a valid ECU wake-up event is received. The below figure show the transition activities between High Power and Low Power.



FIGURE—21 SEQUENCE TO ENTER AND EXIT LOW POWER MODE

The Dual Core system software shall be based on the single core, this is done to increase reusability and to reduce cost. Hence similar concept shall be implemented on both the cores. But it shall be ensured that both cores are synchronized.

Please Note: *The Power modes are explained in the section 8 – Appendix – B

4.2 TASK BREAKDOWN

As discussed in previous sections to achieve this goal the user stories outlined in section 2.2 was further sub-grouped into smaller tasks:

Task	Description
1	Analyze the requirements and assumptions outlined for dual core operation
2	Evaluate the existing design and propose a design to efficiently enter and exit low power mode.
3	Set-up the RH850 / F1H dual core environment and execute using Lauterbach
4	Set-up a debug environment using Lauterbach, (Modify the Lauterbach scripts to successfully execute the code)
5	Develop the proposed design to enter and exit low power mode
6	Test the software for all scenarios of possible failure

TABLE—7 DUAL CORE PROTOTYPE - TASK BREAKDOWN

This helped in monitoring each task status outlined above and in-turn helped in being on track with respect to the goals and the deadlines.

4.3 INITIAL GROUNDWORK AND SET-UP

As described in the section 2.2, this part of the internship was oriented towards evaluating different design approaches wherein the two cores can be synchronized to enter and exit low power mode.

When working in dual core approach, there exists a need to evaluate several communication mechanisms between the two cores to allow the low power sequence, including synchronization points and activity status sharing.

In RH850 F1H, there are two CPUs, which execute different codes at different times. Hence they may get wake up events from different sources and they may reach the IDLE state at different times. This is the idea of having two independent CPU cores.

Additionally, in Low Power Mode the device does not have many tasks to be performed, due to this it shall be sufficient to have one core operate in Low Power and the other core can be in Stop or Halt mode until a wake-up factor is received to return to High Power Mode. For dual-core systems - synchronization mechanism must be established during the mode transitions in order to ensure coherent operation on both the cores.

The STOP triggered by CORE 1 will also stop CORE 2 and nearly all the peripherals immediately at the same time. Due to this reason we need to make sure that both cores do not have any event executing and are in Idle Task before entering Low Power.

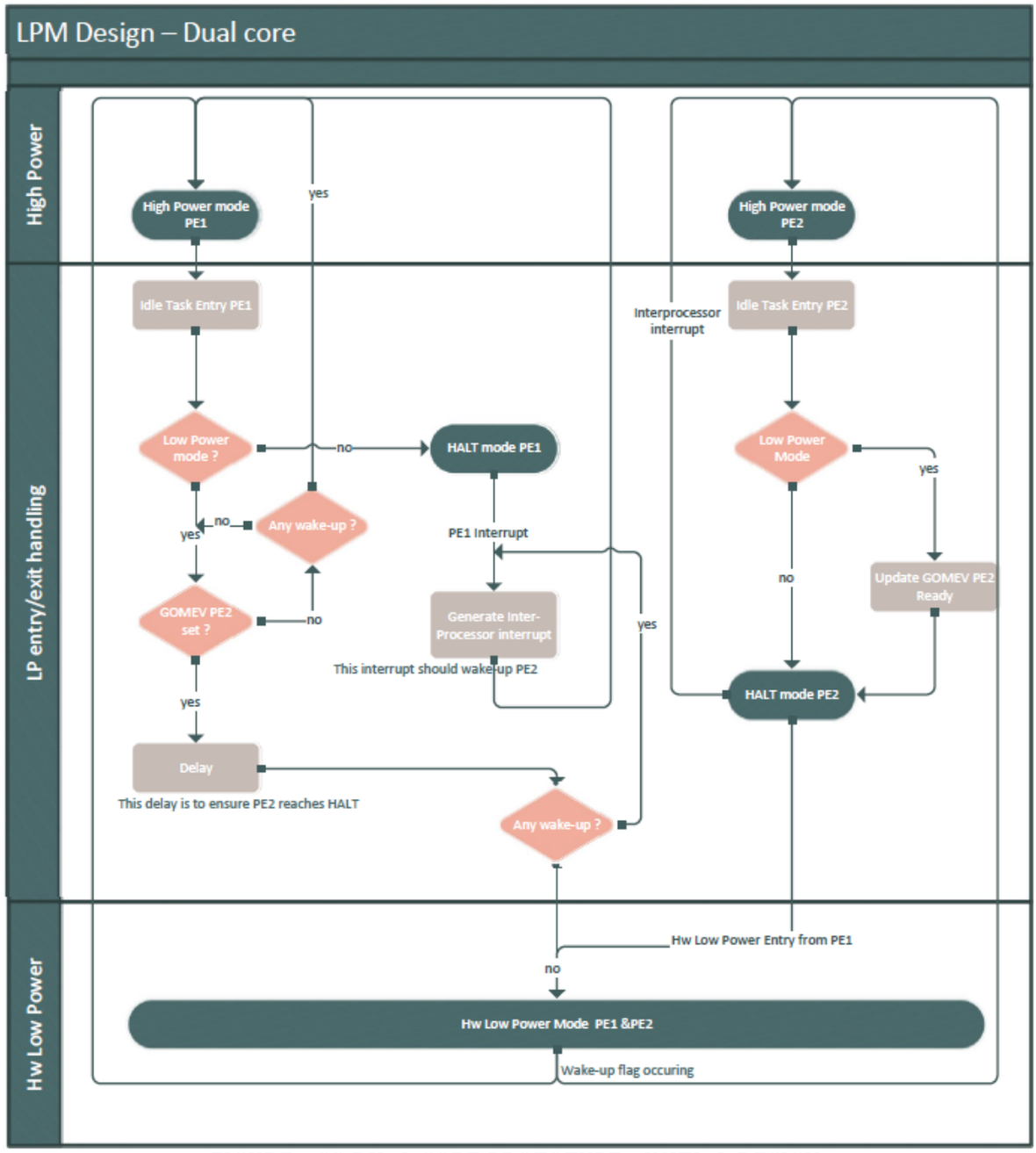
Note: In case the project is configured for Deepstop* Mode – The complete microcontroller is restarted again.

4.3.1 MISSING LINK IN THE INITIAL DESIGN

The below figure indicates the initial design that had been outlined in the project. This design was an initial guideline but had few details missing:

- How and when shall Core 2 indicate that it is not available to enter Low Power
- There is no need for Core 1 to be in halt mode
- How and when are the readyToEnterIntoLowPower flags reset. (In the below diagram this flag is indicated as G0MEV. Refer to section 4.6.2.1 for detail description)
- There is no core synchronization indicated between entering Logical Low Power Mode from High Power Mode
- An important step was missed where the two cores shall enter Idle Task at different instants of time.

Hence though the initial concept seemed to be promising, but it required further refining.



FIGURE—22 DUAL CORE PROTOTYPE - INITIAL DESIGN

4.4 DESIGN IMPROVEMENTS BROUGHT IN

As explained above during the transition from High Power to Low Power the clock frequency shall also be changed. Before CORE 1 switches the clock from the 120MHz to 8MHz frequency, CORE 1 has to make sure that no High Power task except Idle Task are being executed on CORE 2 (i.e. CORE 2 shall currently be executing the Idle Task), else the result of switching the clock in between the execution of a High Power Task would lead to undesirable results. For which, a co-operative power mode handling mechanism as described in the following chapters shall be adapted.

After a wake event the device has to switch from Low Power to High Power Mode. During the transition from Low Power mode to High Power, in case Stop Mode is configured: CORE 2 shall be switched back from STOP mode to HALT mode.

4.4.1 GUIDELINES ESTABLISHED

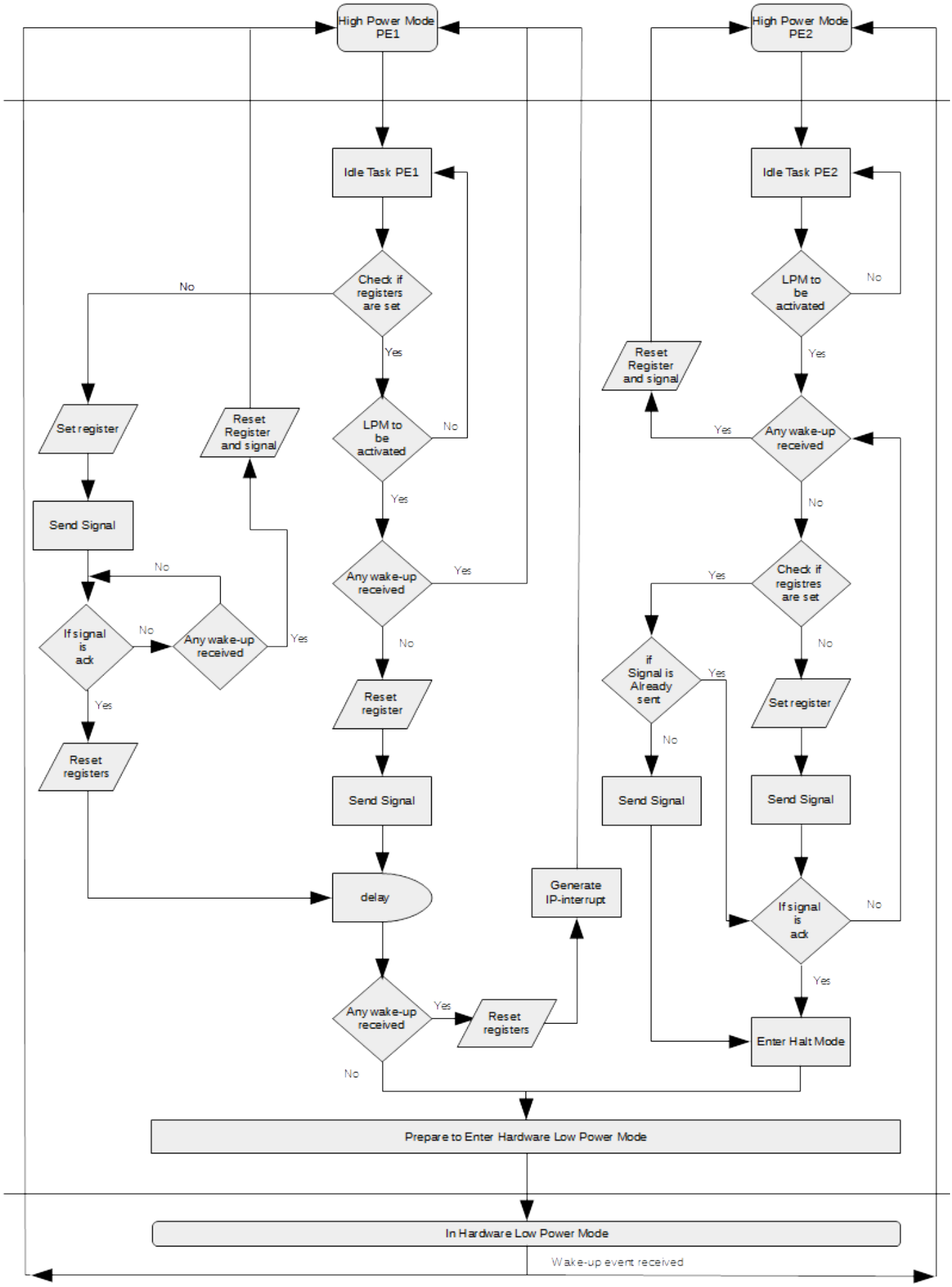
Sl. Number	GUIDELINES for the Design
1	There is only one ECU Manager, which represents Core 1 status
2	Core 1 and Core 2 enter Idle task independent of each other
3	Core 1 and Core 2 both should be in Idle task to enter LP
4	Core 2 shall indicate it's availability status to Core 1 in-order to enter LP
5	Core 1 schedules / triggers all tasks, even the ones that run in Core 2 environment
6	At any given point Core 2 is unaware that a task has been started on its core
7	Core 1 shall wake up Core 2 in case a wakeup event is received
8	Core 2 shall not handle any interrupt, all interrupts shall be handled by Core 1'
9	Core 1 shall handle entering and exiting LPM
10	Core 2 shall remain in Halt mode in Logical LP and enter hardware low power on Core 1's indication (basically all operations are just stopped)
11	All test should be performed in STOP mode, Deepstop* mode is yet to be tested

TABLE—8 DUAL CORE PROTOTYPE - GUIDELINES OUTLINED FOR THE DESIGN

The first design addressed some of the aspects that have been described above but during the Design Reviews the drawbacks in this approach was pointed out.

4.4.2 DRAWBACKS IN FIRST APPROACH

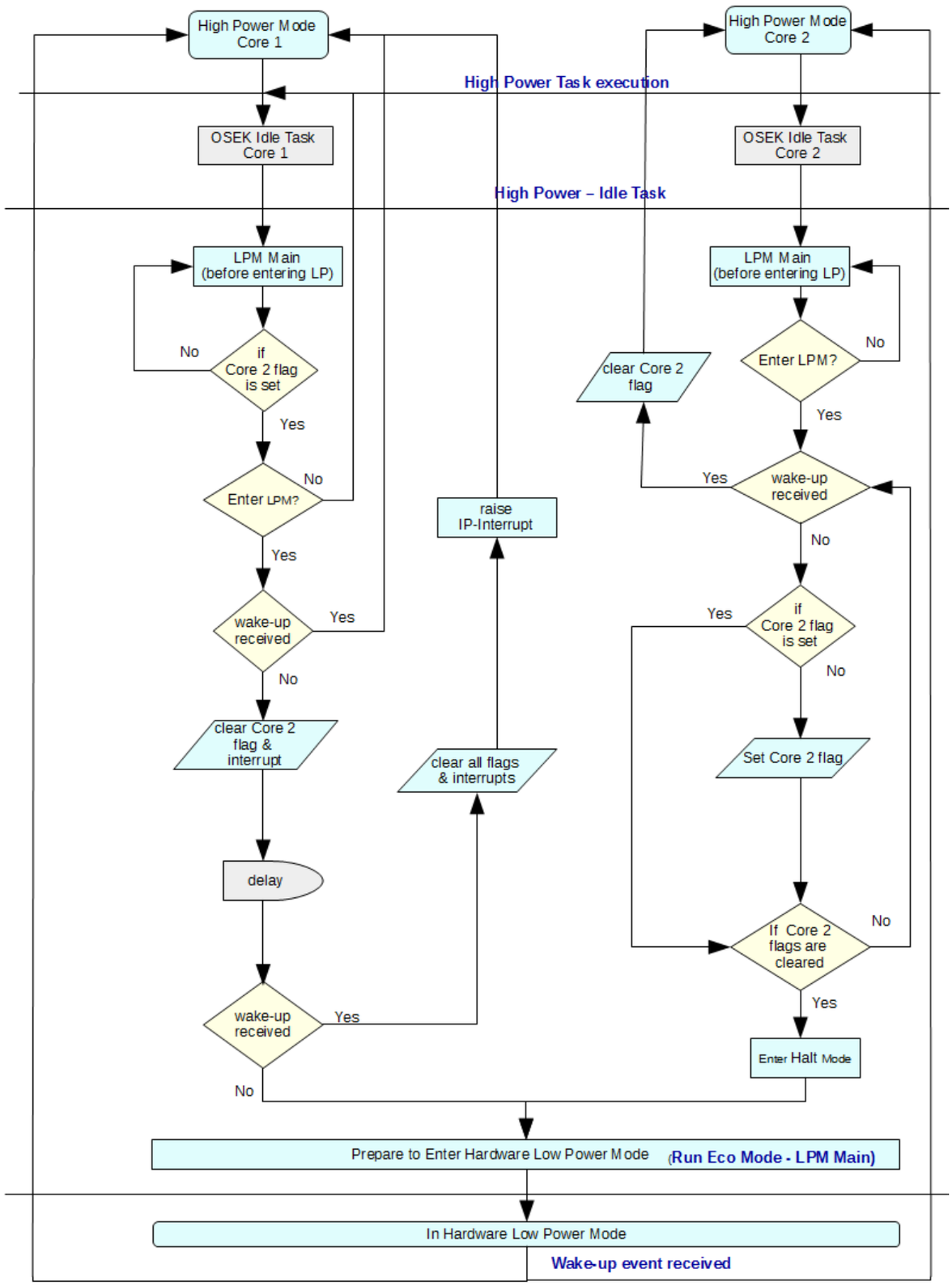
- It is a very complex design approach for embedded system implementation as the application itself is very complex a simpler approach is recommended
- It was realized that since both Cores enter EcuM State of SLEEP together, the cores will be synchronized before entering Idle Task. The Application tasks shall be synchronized before entering EcuM as SLEEP.
- It was confirmed that in Low Power Mode, Core 2 shall not have any task running, so there is no need to synchronize the two cores in Logical Low Power.
- A signaling and acknowledgement mechanism is not required as it makes the code complex, instead mutually exclusive registers can be used as flags to indicate if the core was ready to enter Low Power or not



FIGURE—23 DUAL CORE PROTOTYPE - FIRST DESIGN APPROACH

4.4.3 IMPROVEMENTS NEEDED IN THE SECOND APPROACH

The drawbacks were discussed and a new approach was presented as shown below:



FIGURE—24 DUAL CORE PROTOTYPE – SECOND DESIGN APPROACH

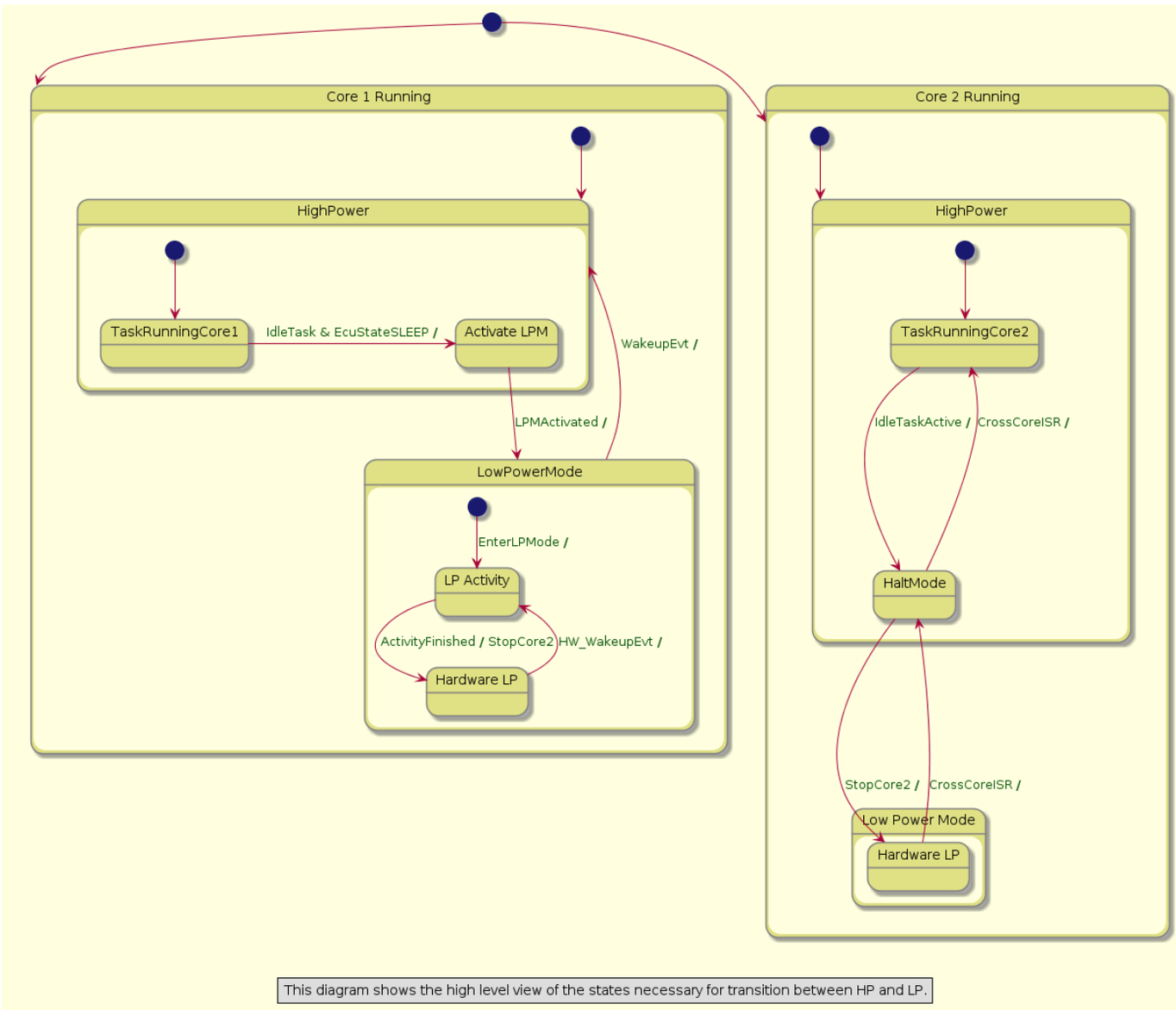
This design suited the project requirements but required further refinement such as there was no need of interrupts to be raised once the Flag is set. (The Core 1 need not immediately respond to Core 2's request). Core 1 can finish its execution and when it enters Idle Loop can refer to Core 2's status.

4.4.4 FINAL APPROVED DESIGN

Hence a simpler approach was adapted where in the following scenario with step by step indication was realized

1. Core 1 and Core 2 are executing High Power Application Task
2. Core 2 enters Idle task, it sets the readyToEnterIntoLowPower flag and waits for an event
3. If Core 2 becomes busy it shall reset the readyToEnterIntoLowPower flag
4. Core 1 enters Idle task and verifies Core 2 readyToEnterIntoLowPower flag status, if the flag is not set the Core 1 remains in the for Loop, and continues to check for the Core 2 readyToEnterIntoLowPower flag to be set or for its own High Power task to be triggered before performing any state change

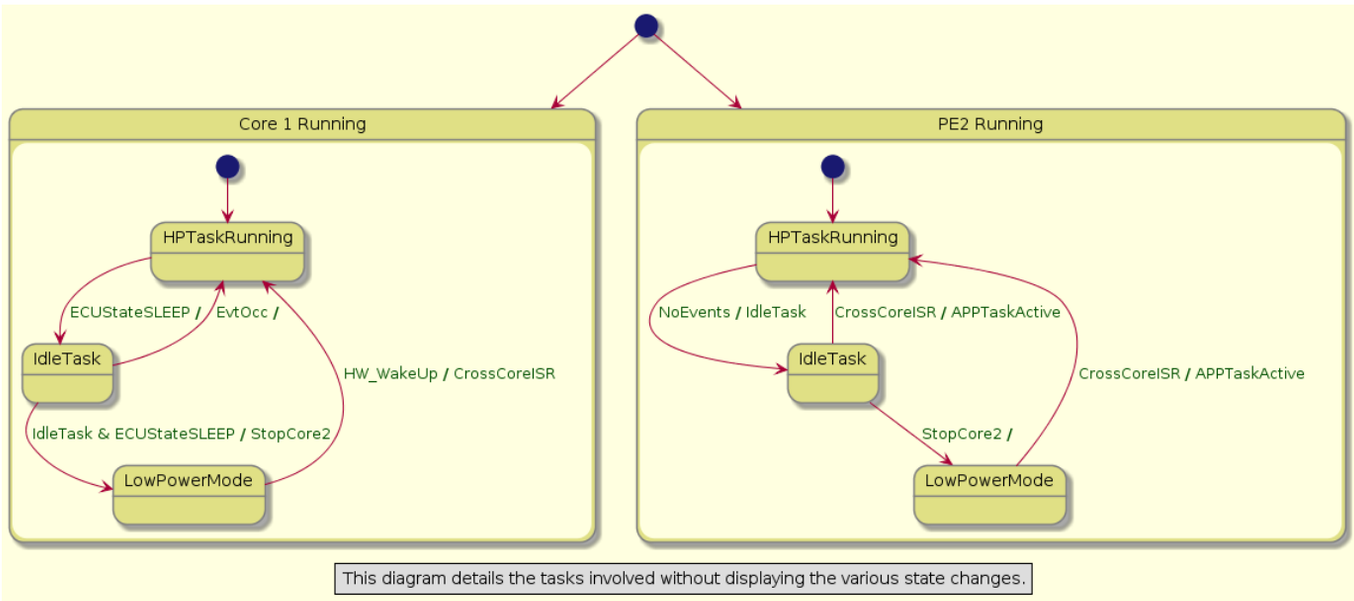
With the above guidelines and aspects in mind the below design aspect was finalized on



FIGURE—25 DUAL CORE PROTOTYPE - STATE DIAGRAM – POWER MODES TRANSITIONS

The following synchronization mechanism has been realized:

1. Core 2 shall set a flag once its Idle Task is executed and enters `HALT mode`
2. When Core 1 enters its Idle Task and the `EcuMState` is `ECUM_STATE_SLEEP`, It checks if Core 2 flag is set
3. If both cores are in idle task and Low power Mode is configured, Core 1 initiates a transition to Low Power
4. In `logical Low Power` – Core 2 shall remain in Halt mode and will have no task to perform where as Core 1 shall have minimum LP activity that is triggered by the `LPM_Scheduler`
5. Once the low power activity is completed in Core 1, the two cores enter hardware low power



FIGURE—26 DUAL CORE PROTOTYPE - STATE DIAGRAM – SYNCHRONIZATION DURING LP TRANSITION

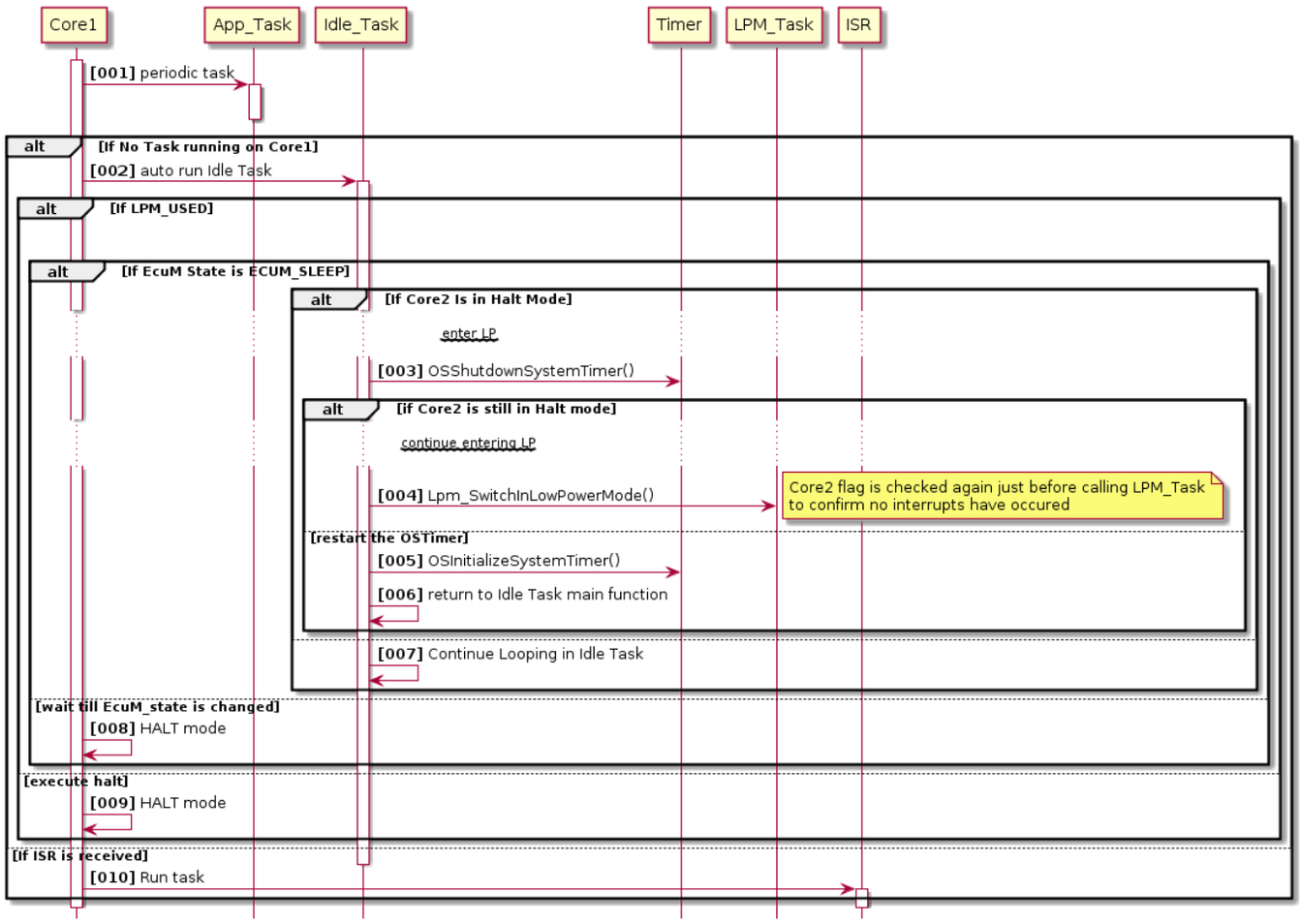
Sequence on CORE 1:

```
❖ CORE 1 Enters the Idle Task For_Loop [for( ; ; )]
  ➢ LPM_configured == TRUE [Yes then]
    ▪ If ECUM_STATE == SLEEP [Yes then]
      • if MULTICORE Environment
        ♦ If Idle Task is activated on CORE 2 – [Yes then]
          ➢ Disable the system timer so the OS high power tasks are no longer activated
          ➢ Call the specific service from LPM to enter the LP mode
          ➢ Check if idle task is still running on CORE 2 (done to be sure that no
            interrupts have occurred)
            ▪ If yes then
              • Activate LPM Task
            ▪ Else
              • Restart OS timer
              • Exit shutdown function
        ♦ Else
          ➢ Continue in the For_Loop of the CORE 1 Idle Task
    ▪ Else
      • Enter Halt Mode
  ➢ Else // do nothing... LPM is not entered
❖ End of For_Loop
```

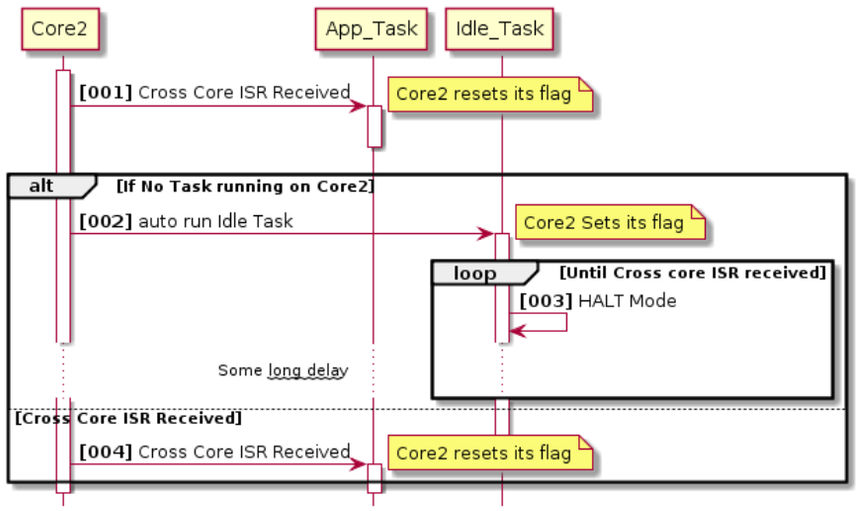
Sequence on CORE 2:

- CORE 2 enters idle task
- Set flag to indicate CORE 2 is ready for Low Power transition
- Activate halt mode
- If a Cross Core ISR is received reset the flag and execute the High Power task

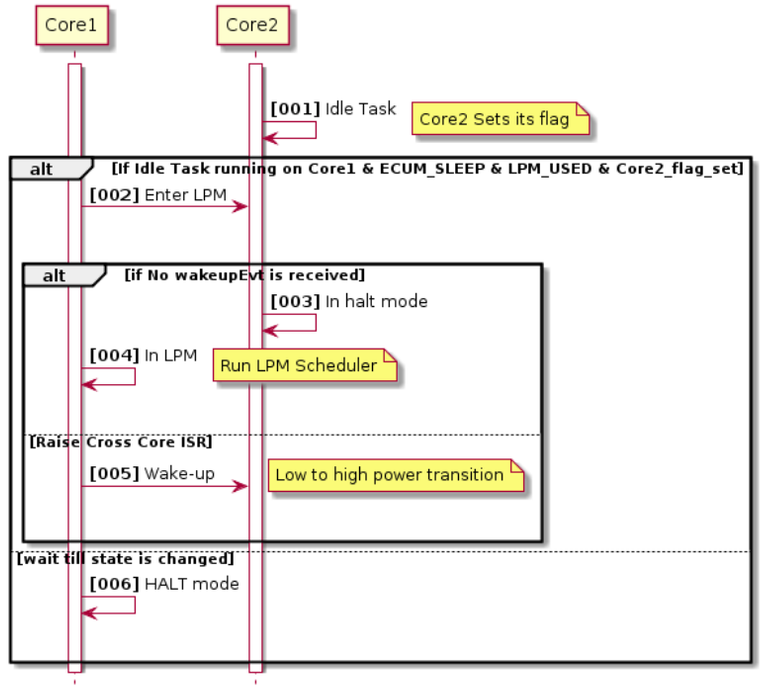
In case there shall be a high power activity triggered on CORE 2. The flag shall be reset and CORE 2 performs the high power task. This reset is performed in the `CrossCoreISR1` function that is called once CORE 2 receives an interrupt from CORE 1 for scheduling a task.



FIGURE—27 DUAL CORE PROTOTYPE - SEQUENCE DIAGRAM – TRANSITION [HP TO LP] – CORE 1



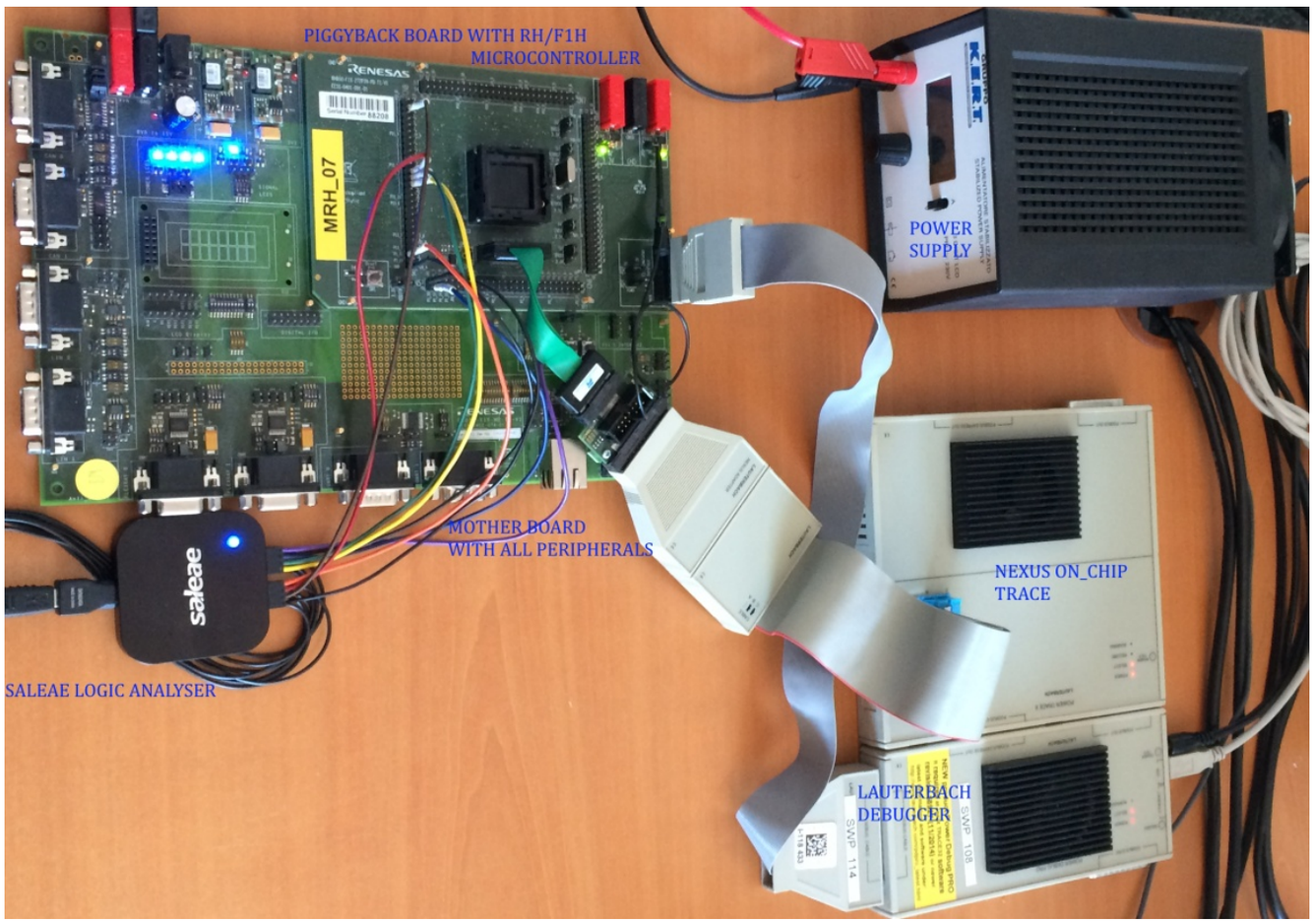
FIGURE—28 DUAL CORE PROTOTYPE - SEQUENCE DIAGRAM – TRANSITION [HP TO LP] – CORE 2



FIGURE—29 DUAL CORE PROTOTYPE - SEQUENCE DIAGRAM – COMMUNICATION BETWEEN CORES

The above design was finalized and was implemented and tested thoroughly to identify any other failure scenarios.

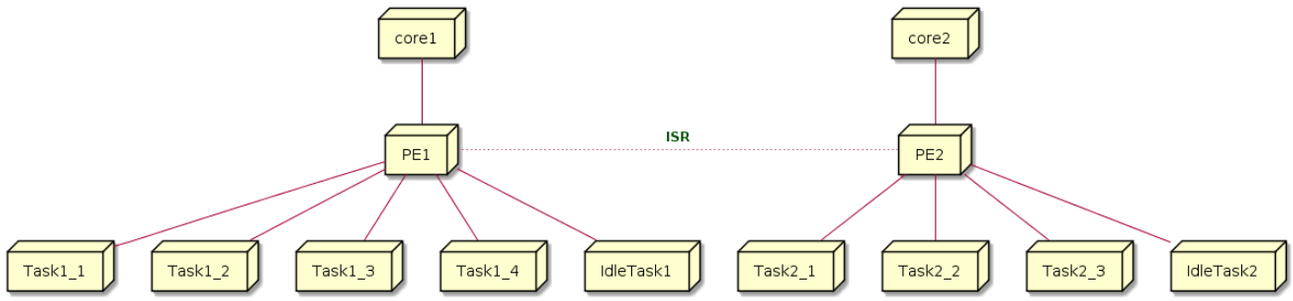
4.5 HARDWARE SETUP



FIGURE—30 DUAL CORE PROTOTYPE - SETUP FOR RH850 / F1H

4.6 IMPLEMENTATION

Each core is assigned one application / process (PE) and the tasks are assigned to these processes. The tasks can either be periodic or async task (Application task or basic software task or os task). And accordingly the task period is set or appropriate alarms are set to trigger async task. `The timer unit available on Master Core shall trigger all tasks (i.e. for both the Cores)`. Additionally, each application / process shall have a low priority, auto triggered task – 'the Idle Task' that will run in case no other High Power Task is scheduled.



FIGURE—31 DUAL CORE PROTOTYPE - DEPLOYMENT DIAGRAM

The project was configured similar to the framework shown above using tools like CESSAR-CT and SPARC (Continental Internal Tools). Where the application tasks and basic software tasks are assigned on Core 1 along with one auto-run idle task. And a 15 ms periodic task was added on Core 2 along with an auto-run Core 2 idle task.
Additional configurations were done to start the cores synchronously but that are not discussed in this report as the strategy to synchronize the core was already present.

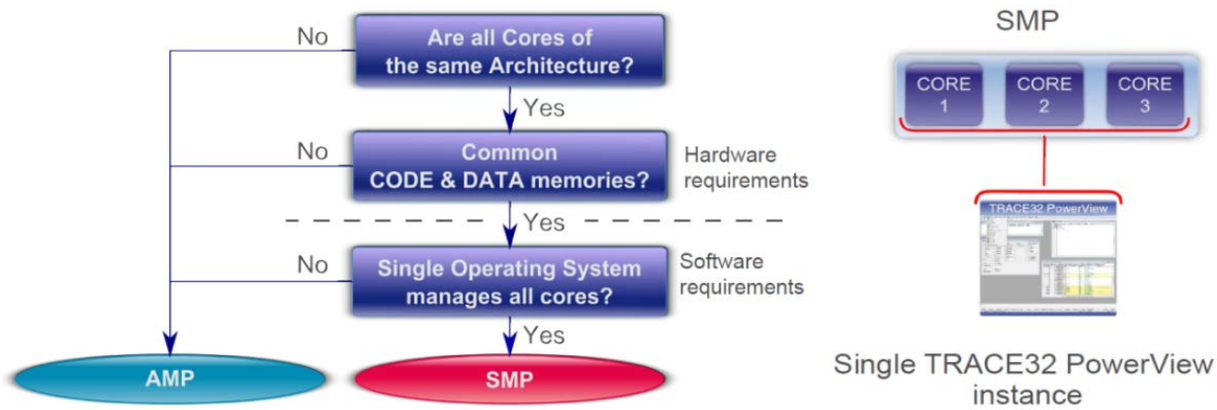
4.6.1 LAUTERBACH DEBUGGING

<u>Lauterbach Debugging for Multi-core environment</u>

[**SMP** – Symmetric Multiprocessing vs **AMP** – Asymmetric Multiprocessing]

Static SMP has been used in this prototype development. The following points justify this choice.

- F1H core board is a homogeneous system – the cores have the same architecture
- One operating system manages both the cores
- The operating system assigns task to the cores, in this prototype development the tasks are already configured before compilation, their period and alarms are already configured. Hence a static configuration is done.
- Cores share code, data and peripherals (UMA: Uniform Memory Access)



FIGURE—32 DUAL CORE PROTOTYPE - STATIC SMP LAUTERBACH DEBUGGING

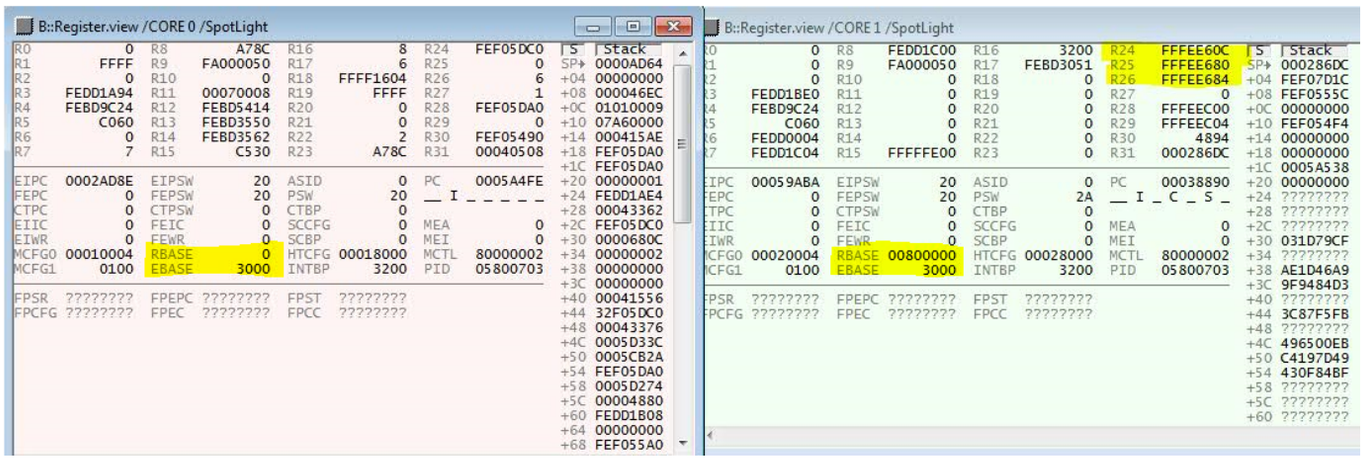
Next the code that was built needed to be flashed on to the microcontroller and tested for which Lauterbach was used. (Lauterbach is a powerful emulator that facilitates debugging of the device and supports dual core launch).

First and the foremost a decision had to be made for using appropriate debugging mode (as discussed in the above section 4.6.1).

Initially the scripting language was learnt and then the existing scripts were modified to suit the current project configuration: for example initially,

There were issues such as the Reset vector did not point to the starting address of both cores. Hence even if it was successful to see both the cores getting launched on to the debugger windows, the code could not be executed. Also it was observed that the stack was getting cleared and the registers that stored the address were also getting cleared

Note: The Reset Vector is used to indicate the starting address when there is a reset for both the cores. This is important as the two cores in the current configuration use the same code flash area. Please refer to section 7.1.1.2.8 of the Appendix to see further details on this.



FIGURE—33 DUAL CORE PROTOTYPE - LAUTERBACH DEBUGGING ISSUES

For this the scripts were modified to include dual core set up and to suit the SMP multi core environment.

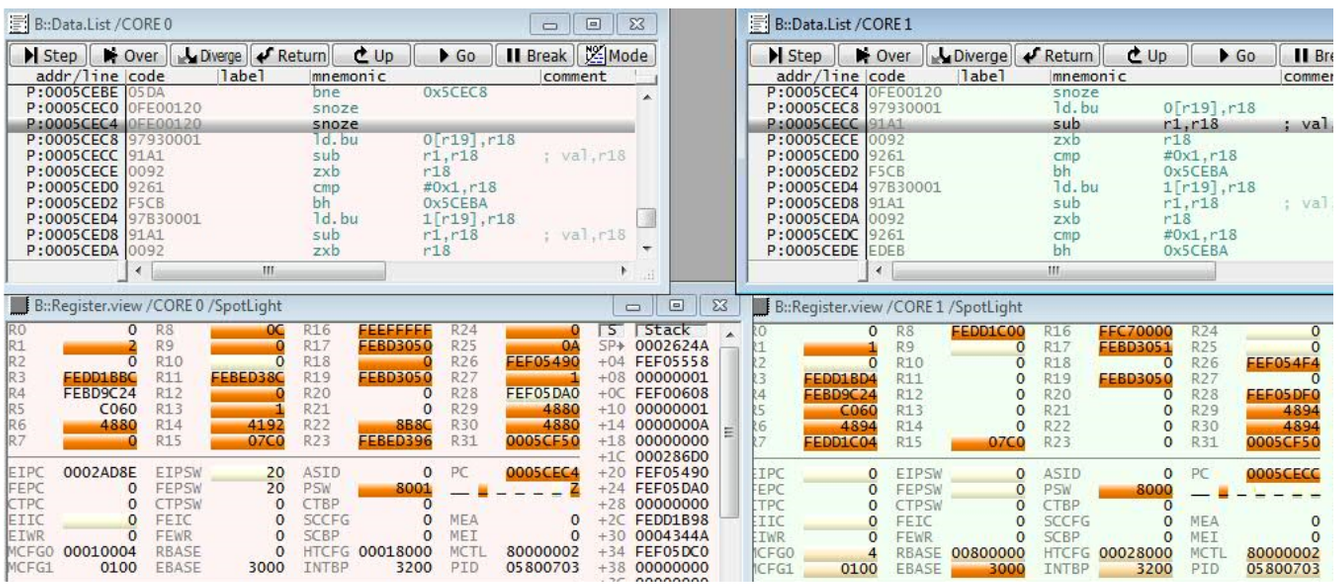
```
SYStem.CONFIG.CoreNumber 2.
SYStem.CONFIG.CORE 1. 1.
; mount CORE1 and CORE2 to a SMP system:
CORE.ASSIGN 1. 2.
SYStem.Up

; FLASHSETUP
 FLASH.RESet
 FLASH.Create 1. 0x00000000--0x0000FFFF 0x2000 TARGET Long
 FLASH.Create 1. 0x00010000--0x003FFFFF 0x8000 TARGET Long
 FLASH.Create 1. 0x00800000--0x009FFFFF 0x8000 TARGET Long
 FLASH.Create 3. 0x01000000--0x01007FFF 0x8000 TARGET Long
 FLASH.Create 2. 0xFF200000--0xFF20FFFF 0x0040 TARGET Long
 FLASH.TARGET 0xfebf8000 0xfebf6000  0x1000 ~~/rh850/flash/long/rh850f1x.bin /STACKSIZE 0x180

 CORE 1  ; switch GUI to second core of SMP system
 Break   ; stop program execution
```

And the Reset script was similarly modified as indicated below:

```
SYSTEM.DOWN
SYSTEM.RESet
SYStem.CPU R7F701515    ;F1H
CORE.ASSIGN 1. 2.
SYSTEM.UP
 IF ANALYZER()
 (
 trace.reset
 iprobe.RESET
 )
  CORE 1   ; switch GUI to second core of SMP system
  Break    ; stop program execution
```



FIGURE—34 DUAL CORE PROTOTYPE - LAUTERBACH DEBUGGING WORKING ENVIRONMENT

4.6.2 DESIGN CHOICES MADE FOR IMPLEMENTATION

4.6.2.1 EXCLUSIVE REGISTER CONTROL

These register supports exclusive control for variables shared between the two cores (common resources). (MEV: Mutual Exclusion Variable Register). These registers are exclusive facility provided only in this **'Renesas' (RH850/F1H)** microcontroller configuration.

Core1 and Core2 each have an independent access path for the G0MEVm registers. Therefore, when Core1 and Core2 each access different G0MEVm registers, they do not need to wait for access. When they access the same G0MEVm register, however, they might have to wait for the register access. G0MEV registers are used for synchronization purposes and are particularly important when the two cores access the same register and both the cores can modify the same register (when both cores writes into the same register)

In the initial design both the cores had write access to the same G0MEV register, so we had to utilize G0MEVs, but in the final design only Core2 modifies the G0MEV; therefore shared variables can also be used. Below are our reasons for which we are still using G0MEVs.

- ❑ The access to G0MEV registers is faster than shared variables
- ❑ It is more secured
- ❑ Usage of shared variables add in the memory
- ❑ To facilitate further design changes (facilitate extendibility)

Note: G0MEV registers are used as flags by Core2 to indicate Core1 of its availability.

4.6.2.2 SYNCHRONIZATION BETWEEN THE TWO CORES TO ENTER LPM

From the earlier chapters it is understood that as Core 2 is the slave core and has no module or means to realize that a Core 2 task is about to be triggered, Or If it is no more going to remain in Idle Loop. It is important to find a way that Core 1 can evaluate Core 2's status.
For establishing synchronization between the two cores a number of approaches were evaluated and tested, some important ones are mentioned below:

1. CORE1 could check for CORE2's status i.e. if Idle Loop is being currently executed on Core 2
   - `// Get the CORE ID using OS APIs`
   - `// GET all the task assigned to this Core`
   - `// loop through the array to get the Core 2 Idle_task_Id`
   - `// get the current status of the task`
   - `// If TaskState == running then enter LPM else do nothing`

   The following are the drawbacks observed
   - As per AUTOSAR specification the task are configurable with user defined task name and the task Id shall be based on the order of configuration. This indicates that neither the task name nor the task Id is constant, so there would be a need to either add a constraint on defining the task in the configuration tool or the complete loop of task should be iterated to check if the Idle Task is currently being executed

   - This check had to been done in Idle task which comprises of a 'for loop' and CORE1 would loop through this Idle task infinitely while waiting for the OS scheduler event. Hence it would be very expensive to loop through the entire list of configured task on CORE2 to identify idle task and then check its status.

2. Instead a flagging mechanism could be developed wherein Core 2 indicates' Core 1 that it is free by setting a readyToEnterIntoLowPower flag – this is accomplished by setting a G0MEV register to 1 once Core 2 enters its idle loop. (as described in the previous chapters)

With this solution care should be taken to reset the readyToEnterIntoLowPower flag (G0MEV) when Core 2 is no more available to enter Low Power.

So the next question to be addressed is - Where the flag should be reset?

4.6.2.3 RESET OF THE 'readyToEnterIntoLowPower ' FLAG

Due to the design constraints in the project there exists only one Os Counter that can be utilized to trigger task on both the cores. This timer is allocated onto the Master Core – Core 1.

Hence Core1 shall trigger all tasks even on Core2 using a Cross Core ISR. That is to say Core2 shall not have the information if a task is about to be executed on it.

But with respect to the design, Core 2 should be able to reset its flag when it shall not be available to enter Low Power (as it has received a Cross core ISR to execute an Application Task).

To solve the above scenario the solution adapted was - this Flag was reset in the `CrossCoreCallback` API. This API is called by the core 2 to acknowledge that the ISR has been received.

Solution 1. Use `Os_Cbk_CrosscoreISRStart()` to reset the flag

- This seemed to be a good candidate as it is an OS call back function and it's the first function executed on Core 2 when it receives a Cross Core ISR, but during the Design and Implementation review it was confirmed that this function shall not be present in future implementation.

Solution 2. Reset could be performed in `PreTaskHook()`

`PreTaskHook()` is a debug API called when a task is triggered, used to monitor the time taken for a task to execute.

- This function is used by both CORE1 and CORE2 and it is entered when any task is initiated. It is a very generic function and lot of check would have to be done such as
  - If multicore
  - If CORE2
  - If task to be run is **not Idle task**
    - Then reset the flag
  - Else do nothing
- As discussed in the previous section - all task are highly configurable and to get the ID of Idle task or name of Idle task constraints must be defined in the configuration tool
- More over this part of the code is not configured for production units. Used only for debug / development propose

Solution 3. CORE1 triggers all CORE2 tasks; it has the information that CORE2 would no longer be available. Hence a reset of CORE2 flag can be done at CORE1 level. Probably during raiseCrossCoreInterrupt() that triggers the interrupts on CORE2 for executing a task.

This is not a good approach as CORE2 should be able to handle its flag, the idea of one core setting the flag and the other core resetting is not safe approach.

Moreover, in case `raiseCrossCoreInterrupt()` would have been chosen to reset Core2's readyToEnterIntoLowPower flag, then consider the case where:

- CORE2 is in idle task and it sets the flag to free
- CORE1 has to schedule CORE2 task so it raises a cross core interrupt and just before raising the interrupt, CORE1 resets the readyToEnterIntoLowPower flag.
- CORE2 is in Idle loop that executes this loop every 32 clock cycle and since CORE2 does not have any indication that it would be come busy and sets the readyToEnterIntoLowPower flag back to free. As it had not yet received the cross core interrupt.

Solution 4. Reset readyToEnterIntoLowPower flag when every high power task (other than idle task) is executed on Core 2 – simple but not a clean approach as in case we have 20 task configured on CORE2 the first statement in all 20 task would be to reset the readyToEnterIntoLowPower flag.

Solution 5. At last a wrapper function was written that would be called just before the actual OS `crossCoreISR1()` function is called.

Note: The `crossCoreISR1()` is an OS API that dispatches the task to be executed CORE2 once a `CrossCoreISR` is received. The readyToEnterIntoLowPower flag could be reset at this location.

```
Call the Wrapper function (_wrap_Os_CrossCoreISR1())

{

if LPM is configured

   Reset readyToEnterIntoLowPower flag (G0MEV2 = 0)

//Call the actual API for the task of Core 2 to be executed (__real_Os_CrossCoreISR1())

}
```

The linker files were modified to that the wrapper function is called before the actual function. This method is just a tweak applied as there were many constraints of the underlining software. And as this was just a prototype design this was accepted by the architects.

4.6.2.4 PROBLEMS DISCOVERED DURING IMPLEMENTATION

The above chapters defined the approach that would be followed to design the prototype and some design choices made when the detail design or low level design was done. (As to – How to indicate the Master core that core 2 is ready or not, Where to add the setting and re-setting of the readyToEnterIntoLowPower flags etc)

But when the actual implementation was done and test was performed a loop hole in the final detail design was observed, discussed and pointed out during the review meeting with the architects.

Scenario:

5. CORE1 and CORE2 are executing High Power Task
6. CORE2 enters Idle task it sets the flag and waits for an event
7. CORE2 becomes busy and reset the flag (in the cross core ISR)
8. CORE1 enters Idle task and verifies CORE2 status and waits for CORE2 to become idle or for a High Power task before performing any state change

For establishing this scenario the following code snippet was introduced in CORE1.

```
// CORE1 Idle Task loop

{

…….        If EcuM State is Sleep ……….. // step 1

          if (CORE2 is in Idle task)

……. // PE 2 is in Idle state ::: enter LP ……....  // step 2

    // Stop OS counter ……..  // step 3

    // Enter LPM ……..  // step 4
}
```

The drawbacks observed in this mechanism are as follows:

Now, consider a case wherein just at the execution of (step 3), if a new CORE2 interrupt is received then CORE2 would no longer be in idle task and CORE1 would enter Low Power.

To handle this scenario the below design aspects were evaluated.

Solution 1. To disable all cross core interrupts when CORE1 enters its idle task Main function –

- CORE1 is in its Idle loop
- `EcuM_State` is `Sleep` ….
- Disable all Cross Core Interrupts `// it will block any new task to be triggered on CORE2`
- Wait for CORE2 to be in Idle state `// check for the CORE2 flag – if the flag has been set`
- If Yes
  - Then enter LPM
- Else continue to wait
- In case CORE2 take a long time to finish and in the mean time CORE1 receives a high power event. CORE1 has to re-enable all Cross Core Interrupts before exiting the idle loop else CORE2 will remain in idle.

Drawback: Managing this scenario is not that easy. The code would be too complex.

Solution 2. Shut the `OS` System Timer before waiting for CORE2 to enter Halt Mode.

- CORE1 is in its Idle Loop
- `EcuM_State` is `Sleep`
- `ShutOSSystemTimer // it will block any new activity both on CORE1 and CORE2`
- Wait for CORE2 to finish its current task
- Enter LPM

This design option was more secured and it's an easy solution.

Drawback: But in case CORE2 has chain task (one task call another task and so on), it would block CORE1 for a very long time and that is not very feasible. Blocking Master core's execution in HP in-order only enter LP might lead to poor design choice as it depends on how the tasks on CORE2 would be configured. In case, CORE2 task are configured to have long execution time with pre-emption and chain task option is enabled then this could lead to the blockage of Master Core for undesirable time.

Alternate Solution: We could add a timeout mechanism such that, in case CORE2 does not become free in xxx ms then the CORE1 clock is initiated again. This was acceptable.

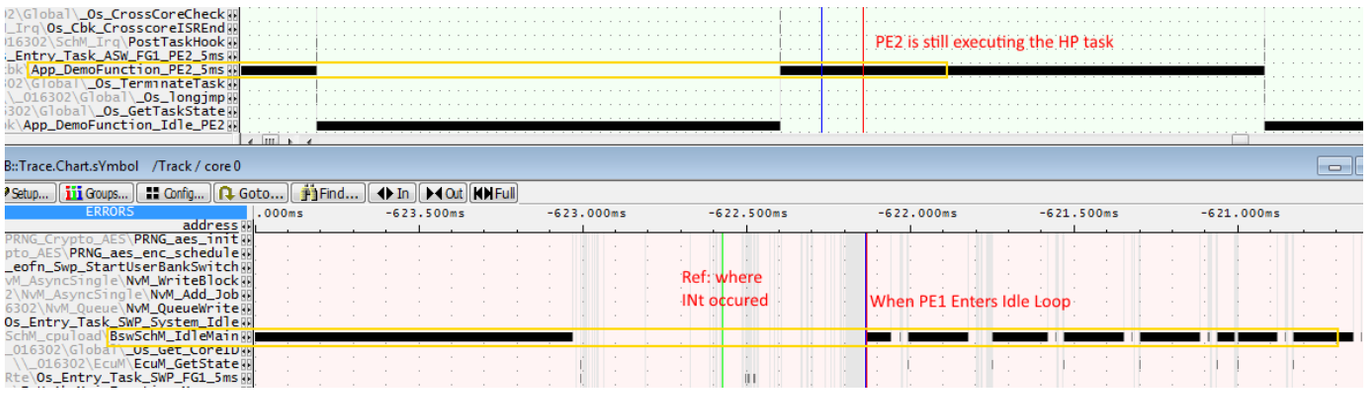
Drawback: This is not the very simplest approach and the architect's advice to look for a simpler approach.

Solution 3. (**FINAL SOLUTION**) Re-check the state of CORE2 just before executing Low Power task `LP_Task` (this task is responsible for starting the Low Power Scheduler). The check added was - In case Core 2 is no more available do not enter LP and continue to wait in Core 1 Idle loop.

4.7 RESULTS FROM SIMULATION

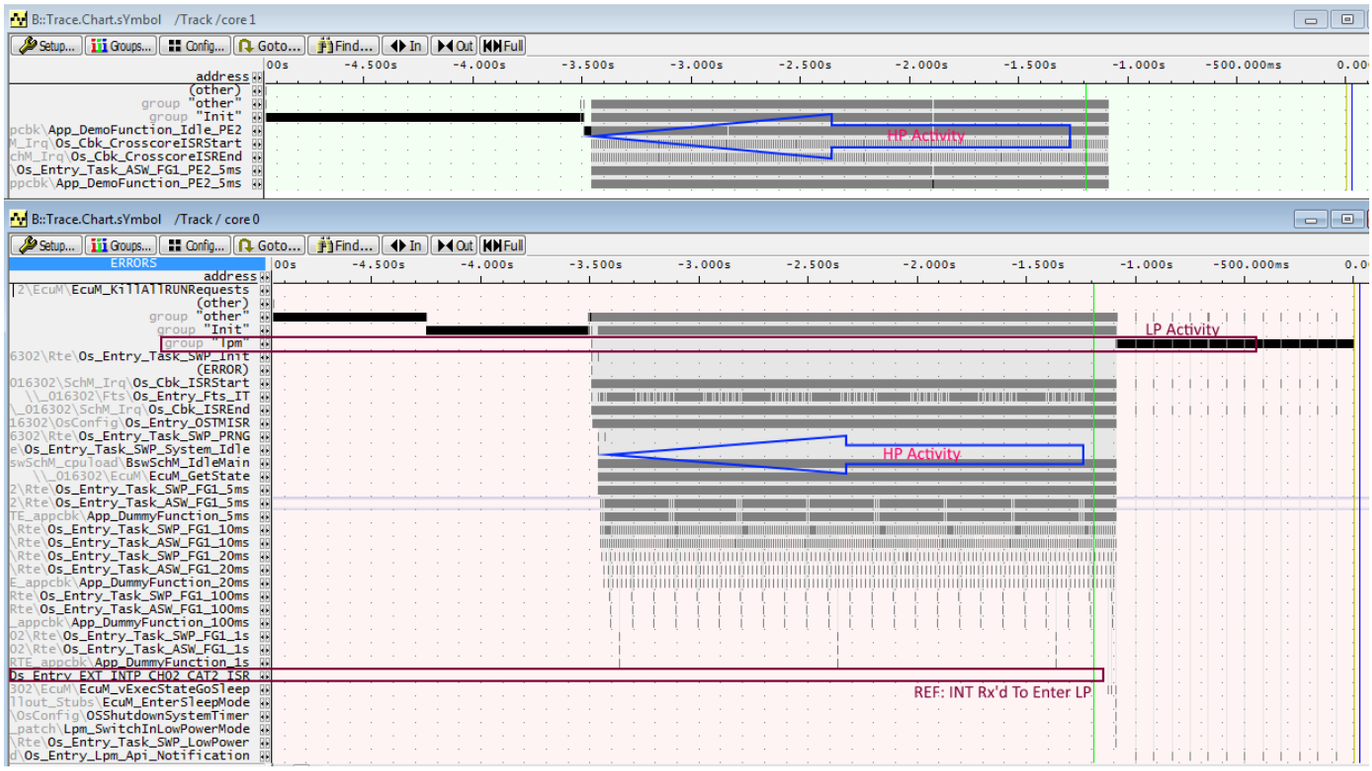
For Testing the Design: A push button was configured to trigger an interrupt for Core1. This Interrupt '**EXT_INTP_CH02_CAT2_ISR**' has an entry point to the test function `Lpm_Entry()`, where in the function `EcuMKillAllRUNRequests()` is executed. Once this function is executed, all the High Power Task except for the idle task are terminated and the idle task is executed. After checking all peripheral state the EcuM sets the ECU state to SLEEP. At this state if Core 2 is also executing the Idle loop then the microcontroller enters Low power mode else it remains in High Power.

The below Screenshot shows core1 entering Idle loop after it received the Interrupt. After some instance of time Core 2 also enters its Idle Loop.



FIGURE—35 DUAL CORE PROTOTYPE - PROCESS TO ENTER LPM - TOP CORE2, BOTTOM CORE1

- The below Screenshot intends to display an overview of the scenario explain above. The green trace marker line is where the Interrupt is received. Before that we can see high power task being executed on both cores.
- Once all the conditions are satisfied to enter Low Power, Core2 remains in Halt mode and on Core1 we can view Low Power mode activity.

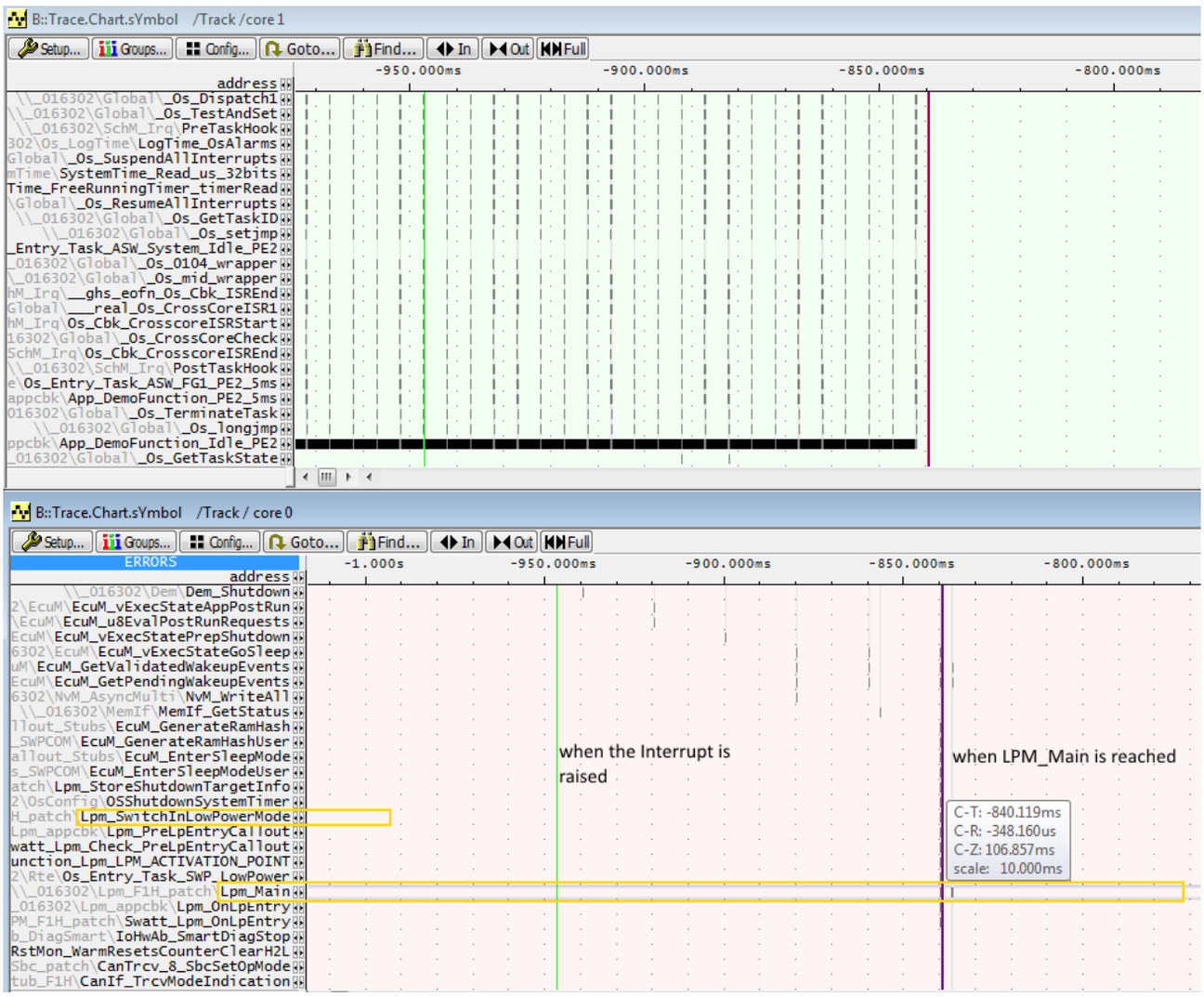


FIGURE—36 DUAL CORE PROTOTYPE - OVERVIEW OF HP AND LP - TOP CORE2, BOTTOM CORE1

In the below case, when the interrupt is received CORE2 was executing HP task (that is not idle task). By the time CORE2 enters Idle loop, the 5ms high power task is scheduled on CORE1. Hence CORE 1 is no more available to enter Low Power Mode.

FIGURE—37 DUAL CORE PROTOTYPE - TASK SCHEDULING – TOP CORE2, BOTTOM CORE1

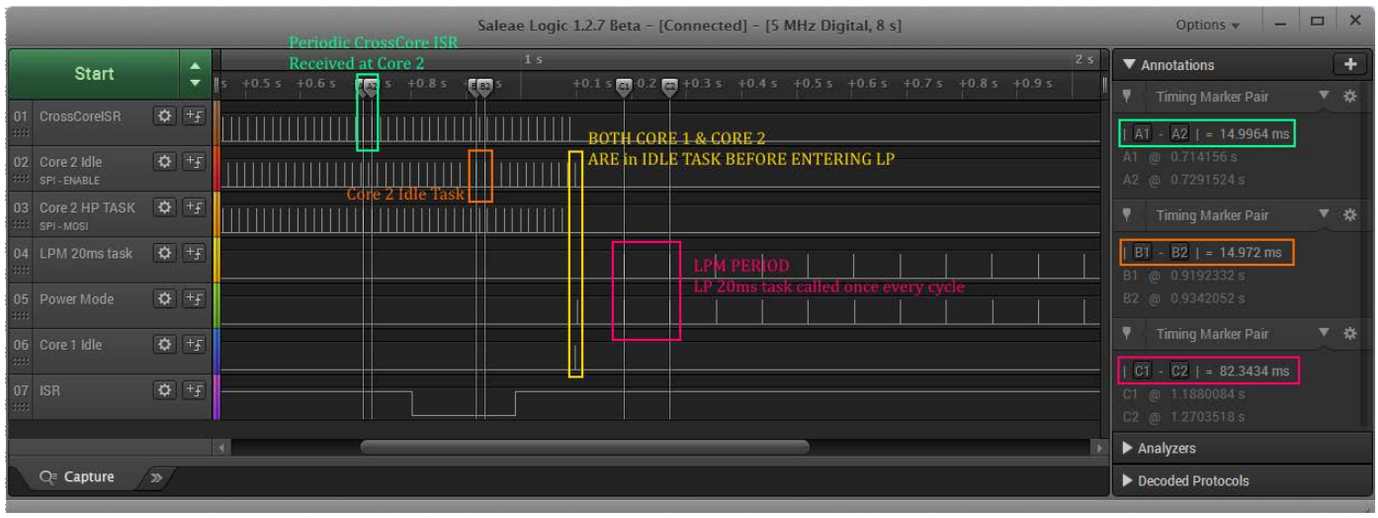
The below figure shows, it took 100ms to enter Low Power mode once the interrupt was raised.



FIGURE—38 DUAL CORE PROTOTYPE - TIME TAKEN TO ENTER LOW POWER

The below output is obtained from the logic analyzer. It can be verified from this screenshot that both the Cores were executing Idle Task when the low power mode became active. Similar results were obtained by testing various scenarios. Various scenarios that have been tested are:

1. Core 1 is in Idle Task but Core 2 is executing High Power Application Task
2. Core 2 is in Idle Task but Core 1 is executing High Power Application task
3. Core1 and Core 2 are executing Idle task
4. Core 1 and Core 2 are executing High Power Task (that is not Idle Task)



FIGURE—39 DUAL CORE PROTOTYPE - OUTPUT FROM SALEAE LOGIC ANALYSER

4.8 IMPROVEMENTS

Though these test confirmed that the prototype code is working, there are some important scenarios that we cannot test and need to be analyzed using special tool or methods. We have indentified such scenarios but only a theoretical analysis is currently done to confirm that this prototype code is accepted.

Scenario 1.

- Core 1 and Core 2 are in idle loop
- Entry to Low Power is initiated
- But just before entering Low Power mode Core 2 receives a CorssCoreISR and schedules a high power task. The readyToEnterIntoLowPower flag is reset
- Core1 returns to the Idle loop
- Core 1 receives and interrupts, gets out of the loop and executes the high power task
- Once the HP task is completed, Core1 returns to LP. What should happen?

This scenario has been taken care in the development and the solution adapted is explained in the above section but it cannot be tested. Such scenarios have to be analyzed using theoretical use cases.

- All test have been performed in STOP mode of LP, In future the same test have to be performed in Deepstop* mode
- There ongoing reviews and analysis being done for the different design aspects that can be visited for this prototype development and with the changing requirements on the product the design would have to be adapted. For example
  - Design aspect would change in case mode management strategies would be allocated onto both cores. (If the Ecu state manager is available on both cores)
  - Design aspect would also change in case Low power activities would have to be performed on both the cores

5 EVALUATION AND ASSESSMENT

The power of engineering is immense "Embedded Engineering can make machines move, communicate to each other (IOT) and be intelligent". With the right knowledge and the right platform we can make better, comfortable, communicable lives. As technology advances our lives are getting busier and complicated day by day, with the right engineering we can bring about changes where at any time and any place we can ensure that our lives and the lives of our loved ones are safe. With this in mind I joined the course offered in **Embedded Systems** in, **Institut supérieur de l'aéronautique et de l'espace**' and **'École nationale supérieure d'électronique, d'électrotechnique, d'informatique, d'hydraulique et des télécommunications'**. I am glad that I got the platform to widen my prospective of Embedded System, I came across concepts and domains which I was either unaware of or did not have much knowledge about.

Additionally, there are many areas in '**Continental**' which have very interesting work being done. There are **Tech Days** organized wherein the domain experts, students, engineers show case their work and share the advancements made in their domains. There are many training sessions, knowledge sharing sessions that are conducted for everyone in the team to provide them the correct base for starting their work or to broaden the skill base in '**Continental**'. This helped me to explore advancements done in Power Train division (engine system Management), learn new emulators like E1, Lauterbach, understand the concept of AUTOSAR and its necessity in body and security division, I got to learn topics related to project and also had a chance to work on the hardware, which is in itself a unique experience. I got to work on microcontroller, to learn how to interpret data-sheets, I got to work on latest technologies like Dual Core, had brainstorming productive review sessions with not only my team here in Toulouse but also across Europe. I had the opportunity to have the design reviewed and to listen to ideas from engineers around the world.

Apart from the technical and project related learning I had the opportunity to observe and absorb the work culture of '**Continental**'. The '**Software Platform Team**' has a unique closeness; there are many activities and sharing among the team members that keeps a lively environment. Before assigning a work the person's interest area and capability are judged among other aspects and I am glad to have a chance to learn and perform in interesting projects.

6 CONCLUSION

As a part of studies I did my Internship In '**Continental**' and as indicated in the above chapters the objective of this Internship was divided into 2 main sectors:

DEBUGGING FEATURE TO BE INTRODUCED IN LOW POWER MODE: I was able to successfully get the debugging feature working for Low Power Mode which will help the software platform team to validate the LPM feature and to monitor the state changes that occur in Low Power in an representative environment (without any additional debugger or tweaks to be done on the software or to the environment).

As a part of delivery the code developed for Low Power module with all the logging statements, the code changes done for the debug module to extend the functionality to include debugging feature in low power mode, the configuration files and a user guide explaining the hardware set-up, configuration required, the difficulties faced and the code changes done with output observed and lastly mentioning the improvements required in the current code based were delivered successfully.

DUAL CORE PROTOTYPE DEVELOPMENT (SYNCHRONIZATION TO BE DONE TO ENTER AND EXIT LPM): For the second Part of my Internship, I was able to present an acceptable design and a prototype for the dual core microcontroller, which was well tested for entering and exiting the Low Power Mode. An optional part of this development was also to implement the logging feature in low power mode, but as this was of a lower priority and the task in the development of the prototype evolved and different means to test the software were implemented, and due to the lack of time to accommodate this feature, it has not been implemented.

As a part of a delivery, the code changes done, the configuration files changed, the user guide explaining the hardware set-up and the changes done in the configuration file and additionally a detail design document explaining the complete design development for this prototype, the design choices made the different test implemented, the challenges and improvements that could be brought about were successfully delivered.

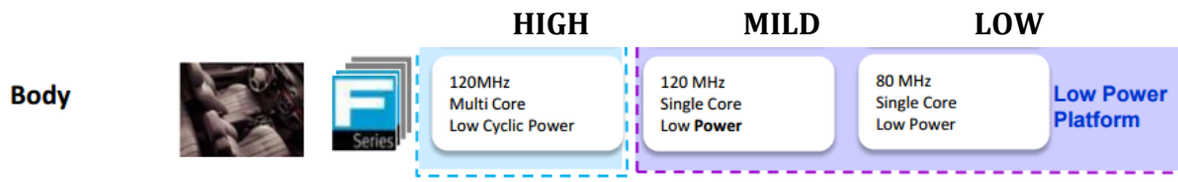
As discussed above as the customer requirements evolve and for different products different strategies would have to be applied to ensure the concept is well adapted. Hence, there are various ongoing investigations and improvements evolving in this development area.

It was a challenging subject and I glad to have been a part of it.

7 APPENDIX – A

7.1 MICROCONTROLLER PRESENTATION

As discussed earlier body control unit is rapidly expanding and it is the most demanding sector in automotive industry. To cope up with the user requirements very efficient microcontroller should be utilize to control and monitor the central body control unit. As '**Renesas**' microcontrollers are fast, highly reliable, low in cost, and they incorporate the latest processor technology, which enables integration of large-capacity flash memory, they are used in a wide array of applications, including demanding fields requiring high quality and high reliability, such as the body control unit in the automotive industry. Moreover they provide a wide range of products that can satisfy the various choices in the market for different end products.
'RH850' Family is utilized for the development of Basic Software in '**Continental**'.



FIGURE—40 RH850 40NM MCU SERIES FOR VEHICLE CONTROL

G3M - Provides real-time supports, improved data processing performance, high performance and Low Power facility
G3K - Small core, provides good performance and Low Power facility

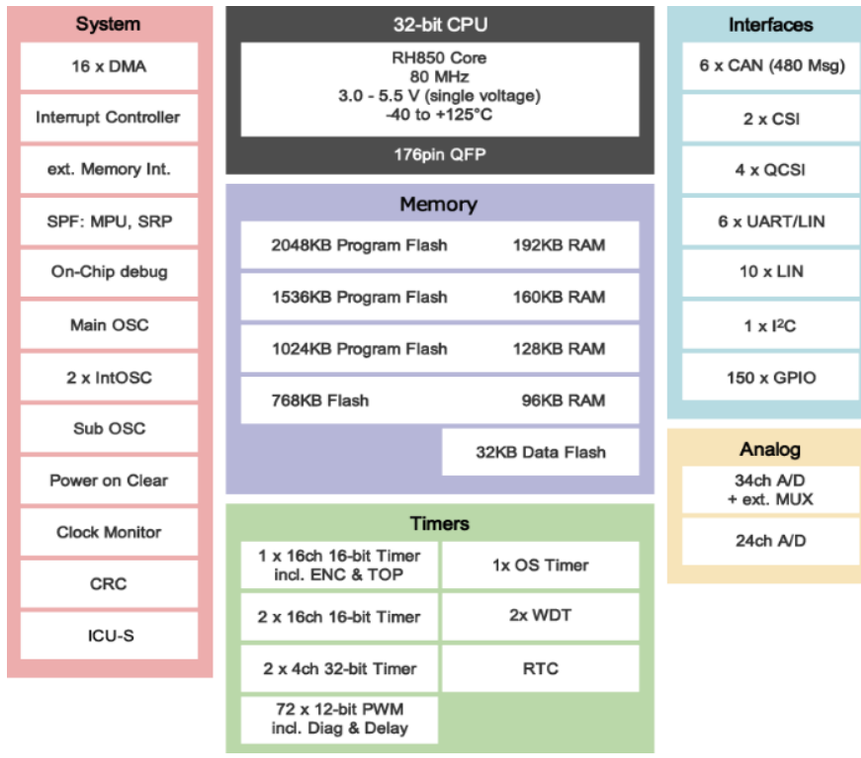
7.1.1 RH850 MICROCONTROLLERS

- The CPU has a Harvard architecture and uses a RISC based instruction set
- The CPU subsystem is located on the Isolated-Area (ISO)* and is supplied with CPUCLK clock
- Places the CPU core that executed the HALT instruction in the HALT state. Therefore, the CPU stops operating, until the release of the halt occurs either by and occurrence of an Interrupt or reset.
- When the SNOOZE instruction is executed, the thread that was stopped by the SNOOZE instruction starts operating again after 32 clock cycles have elapsed
- Low-Power Sampler (LPS) – It is a hardware design component introduced by '**Renesas**' which can poll for analog / digital inputs and use the A/D convertor without using the CPU.
  This sampler is used in case wake-up factors (when in Stop* / Deepstop* mode) need to be monitored by the hardware unit instead of the software unit.
  Note: The choice between using this hardware component over software component to monitor and poll for the wake-up factors depends on specific project requirements.
  For e.g. in a case where in the wake-up factors need to be monitored frequently (let's say every 10 ms) then instead of using a software component (i.e. like the LPM scheduler) the Low Power Sampler is used.

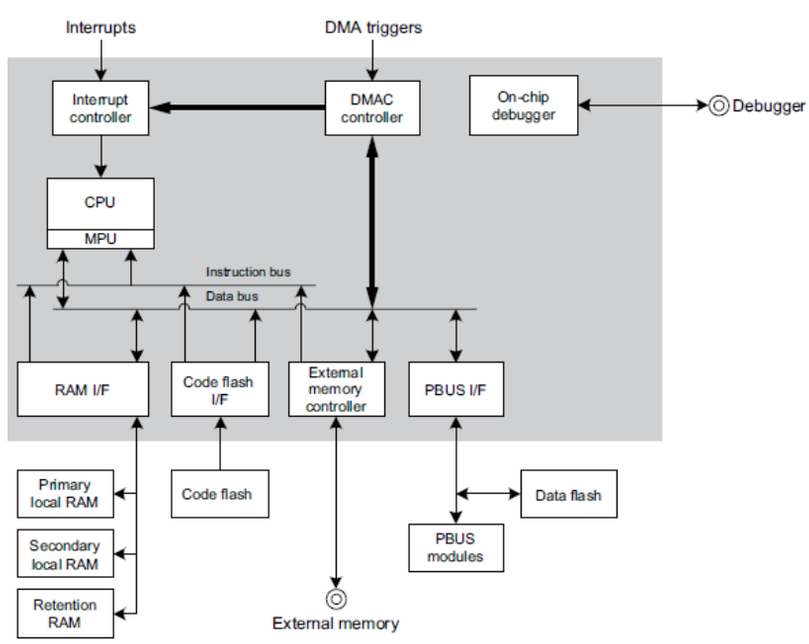
The LPS used in combination with TAUJ (the clock used for periodic wake-up of LPS) and STBC (The standby Controller that shall be used to raise an interrupt in case of a wake-up) and the A/D convertors to evaluate the voltage levels.

7.1.1.1 F1L -176 PIN MICROCONTROLLER

In the below section, I have described the components that have to be understood to develop and investigate the internship topics explained above. These sections have been taken from the microcontroller data-sheets, user guides and user manuals available online or in 'Continental' data base.



FIGURE—41 [F1L – 176 PIN] MICROCONTROLLER



FIGURE—42 RH850 / F1L SUB-CPU COMPONENT

The RH850/F1L is a 32-bit single-chip microcontroller with a G3K core. The RH850/F1L supports single-thread processing only. Multiprocessing function are not supported by the RH850/F1L.

7.1.1.1.1 PRIMARARY LOCAL RAM AREA

Values are not retained in this area in DeepSTOP* mode or after a reset.

7.1.1.1.2 SECONDARARY LOCAL RAM AREA

Values are not retained in this area in DeepSTOP* mode or after a reset.

7.1.1.1.3 RETENTION RAM AREA

Values are retained in this area in DeepSTOP* mode and after a reset.

7.1.1.1.4 EDGE / LEVEL DETECTION

External interrupts can be specified to be generated when a rising edge, falling edge, rising or falling edge, low level, or high level is detected at an external interrupt pin.

7.1.1.1.5 INTERRUPT CONTROLLER REGISTERS

Registers is assigned to each Enable Interrupt (EI) level mask able interrupt (EIINT) channel and is used to set the conditions for controlling that channel.

7.1.1.1.6 CLOCK GENERATION CIRCUITS

Five clock oscillators are provided: Four clock oscillators are located on the Always-On-Area (AWO)* and PLL* is located on the Isolated-Area (ISO)*.

7.1.1.1.7 CLOCKED SERIAL INTERFACE H - (CSIH)

(Used in case of SPI* Communication) - The CSIH uses three signals for communication:

- Transmission clock CSIHTSCK (output in master mode, input in slave mode)
- Data output signal CSIHTSO
- Data input signal CSIHTSI
- CSIHTCSS [7:0]: Chip select signal
- Data transmission is bit-wise serial, and performed synchronously with the transmission clock

Unit Interrupt Signal	Outline	Interrupt Number	DMA Trigger Number
CSIH3			
INTCSIHTIC	Communication status interrupt	1, 150	41 (channels 0 to 7)
INTCSIHTIR	Receive status interrupt	5, 151	42 (channels 0 to 7)
INTCSIHTIRE	Communication error interrupt	6, 152	—
INTCSIHTIJC	Job completion interrupt	7, 153	43 (channels 0 to 7)

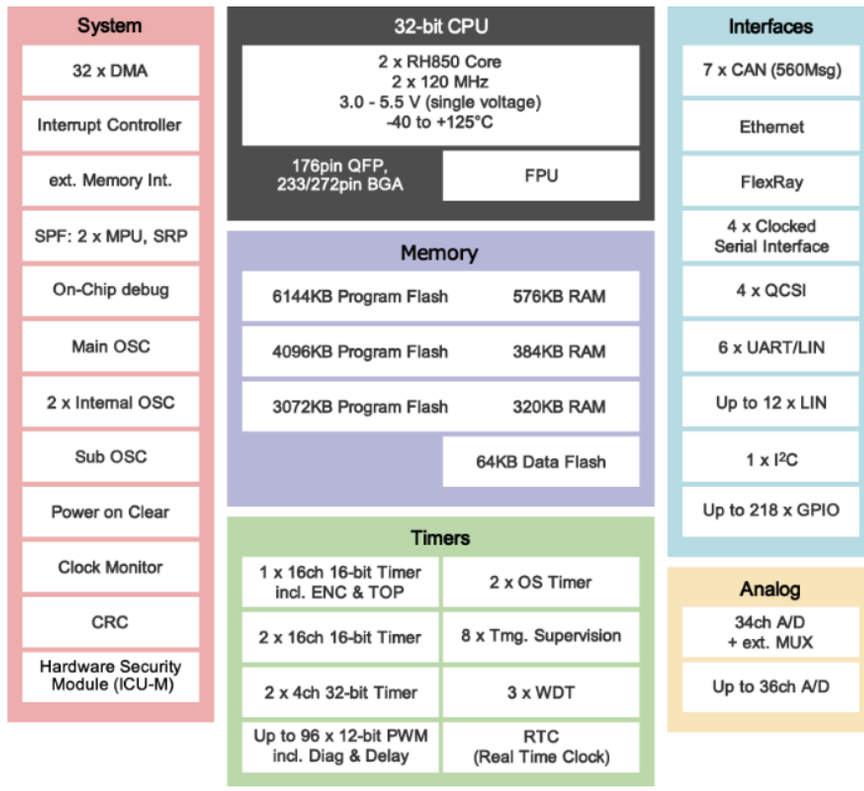
Unit Signal Name	Outline	Alternative Port Pin Signal Name
CSIH3		
CSIHTSCK	Serial clock signal	CSIH3SC*2
CSIHTSI	Serial data input signal	CSIH3SI
$\overline{\text{CSIHTSSI}}$	Slave select input signal	$\overline{\text{CSIH3SSI}}$
CSIHTRYI	Ready/busy input signal	CSIH3RYI
CSIHTSO	Serial data output signal	CSIH3SO*2
CSIHTRYO	Ready/busy output signal	CSIH3RYO
CSIHTCSS[3:0]*1	Chip select signal	CSIH3CSS[3:0]*1

FIGURE—43 PORT FOR SPI COMMUNICATION

Function	Alternative Name	Port Name	Device - 176 Pins	Base Address Name	Base Address
CSIH	CSIH3SC	P11_7	Activated	<CSIH3_base>	FFD8 6000H
	CSIHTSO	P11_6			
	CSIH3CSS1	P8_8			
	CSIH3SI	P11_5			

TABLE—9 SPI PORT CONFIGURATION

7.1.1.2 F1H -272 PIN MICROCONTROLLER



FIGURE—44 RH850 [F1H – 272 PIN] MICROCONTROLLER

RH850 / F1H is a 32-bit single-chip microcontroller with two G3M cores. The main features of the F1H are low power consumption, high computational processing power and a wide variety of internal peripheral functions. There is a Low Power Sampler (LPS) that can poll signals input to the analog and digital input pins without using the CPU, when in DeepSTOP mode, in which the power supply to the most circuits of the microcontroller can be turned off.

7.1.1.2.1 CPU1 (CORE 1)

The Boot loader determines which of the two cores shall be used as a Main CPU. For the purposes of this document the term CORE 1 shall be used to refer Master Core or the Main CPU.

7.1.1.2.2 CPU2 (CORE 2)

The second core shall be used as a slave core in order to enhance the performance of the microcontroller.

7.1.1.2.3 LOCAL RAM

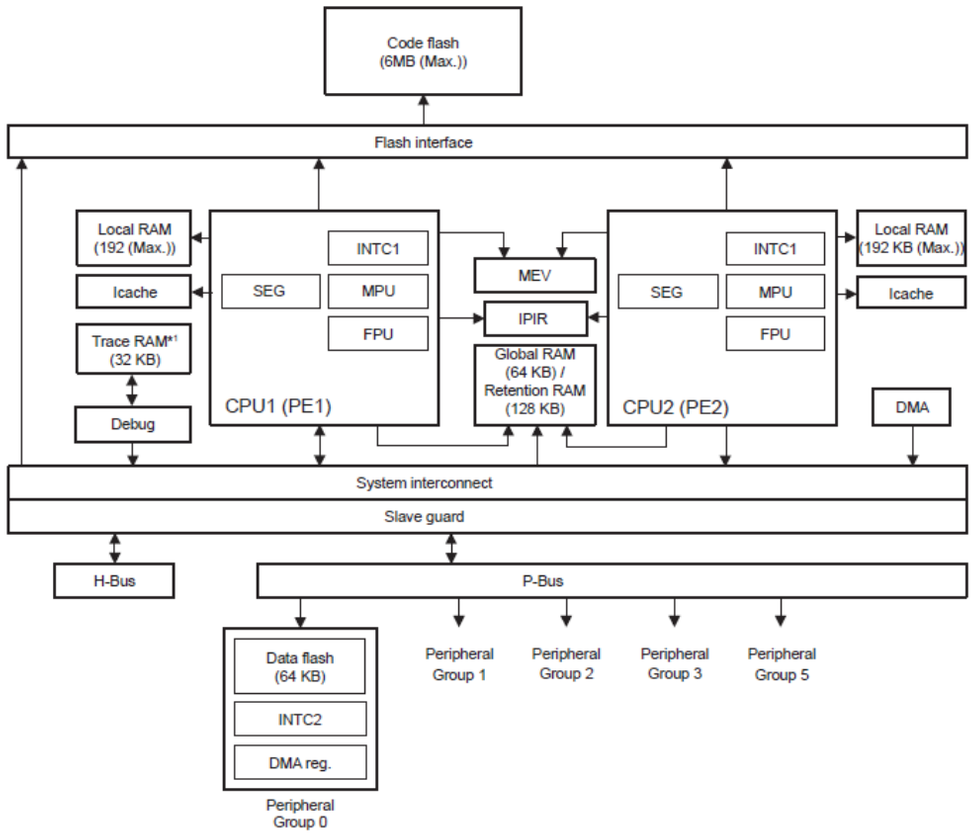
Each CPU shall have a high-speed local RAM*.

7.1.1.2.4 CODE FLASH

The code flash memory is included for program storage. CPU1 and CPU2 share the code flash and they are connected with each other via the flash interface.

7.1.1.2.5 INTC1, INTC2

INTC1* is an interrupt controller exclusive to each CPU*. INTC2 is a common interrupt controller that CPU1 and CPU2 share. The PE to which an interrupt request is bound can be specified by a register setting.



FIGURE—45 BLOCK DIAGRAM OF RH850 / F1H

7.1.1.2.6 GLOBAL RAM / RETENTION RAM

The Global RAM shall be used for data sharing among the PEs via DMAC*. A part of the Global RAM area shall be used as a Retention RAM (this area of RAM is always powered on, hence the data stored in this area shall be retained). The Global RAM is divided into Bank A and Bank B. Different banks can be accessed concurrently.

7.1.1.2.7 SLAVE GUARD

The slave guard is a function to restrict access from other cores and consists of the following guard structures:

- **PE Guard (PEG)** - The PE guard is a function to prevent unauthorized access to the resources in the PE from an external master. After reset is released, access from other than the own PE is prohibited
- **Global RAM Guard (GRG)** - The global RAM guard is a function to prevent unauthorized access to the global RAM and retention RAM from an external master. The global RAM is in the unprotected state (accessible from all bus masters) after reset is released
- **Peripheral Guard (PBG, HBG)** - The peripheral guard is a function to prevent unauthorized access to peripheral modules by an external master. Peripheral devices, with the exception of certain groups, are in the unprotected state (accessible from all bus masters) after reset is released

7.1.1.2.8 BASIC SYSTEM REGISTERS

The basic system registers are used to control CPU status and to retain exception information.

- **RBASE** - 'Reset vector BASE address' - This register indicates the reset vector address when there is a reset
- **EBASE** - 'Exception handler vector address' - This register indicates the exception handler vector address
- **PID Processor ID** - The PID* register retains a processor identifier that is unique to the CPU. The PID register is a read-only register

7.1.1.2.9 INTER-PROCESSOR INTERRUPTS

Four registers (IPIR_CHn) are provided for communicating four channels of interrupts between the CPUs (PEs). These registers are located in the CPU Peripheral of each PE. Each PE has the IPIR_CH0 to IPIR_CH3 registers and cannot access the registers in other PEs.

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	—	—	—	—	—	—	—	—	—	—	PE2	PE1
reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R/W	R/W

Bit Position	Bit Name	Function
31 to 2	Reserved	When read, the value after reset is returned. When writing, write the value after reset.
1	PE2	Inter-PE Interrupt Request to PE2 Writing 1 to this bit enables an interrupt request to PE2. This bit is automatically cleared to 0 when the interrupt request has been notified. 0: Inter-PE interrupt request output is not specified or an interrupt request is not being output. 1: Interrupt request output is specified or an interrupt request is being output.
0	PE1	Inter-PE Interrupt Request to PE1 Writing 1 to this bit enables an interrupt request to PE1. This bit is automatically cleared to 0 when the interrupt request has been notified. 0: Inter-PE interrupt request output is not specified or an interrupt request is not being output. 1: Interrupt request output is specified or an interrupt request is being output.

FIGURE—46 IPIR_CHN — INTER-PE INTERRUPT REGISTER N (N = 0 TO 3) CONTENTS

8 APPENDIX – B

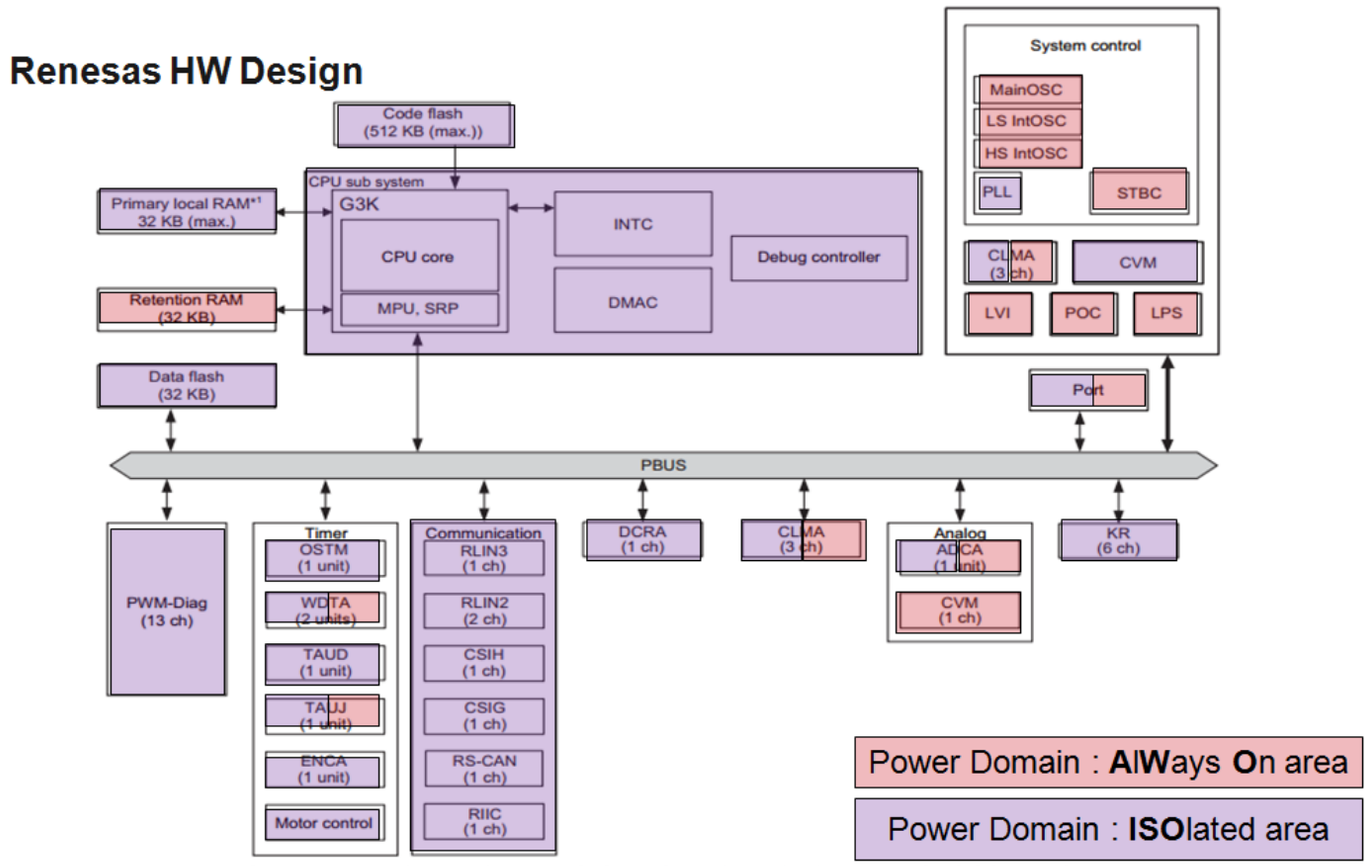
8.1 ADVANCED POWER DOWN CONCEPTS

The advanced power down concepts from '**Renesas**' includes the following strategies:

1. **Module clock Gating** – Cutting supply of clocks in the unit, done specially in STOP mode of Low Power to reduce the current drain through the microcontroller
2. **Module Power Gating** – Powering off the ISO* (isolated area) – This is done in DeepStop* mode of Low Power – Drastically reduces the current through the microcontroller
3. **Run Eco / Logical Low Power mode** – Wherein the software runs at a very low frequency compared to that in full power / RUN Mode. (here the transistor switching frequency drastically reduces hence lesser switching losses)
4. Using Hardware units such as Low Power Sampler instead of Software further reduces the need to keep the software running at all times hence reduces the current

The internal circuits are separated into two independent power domains, the **Always-On** area (AWO) and the **Isolated** area (ISO).

- The power supply of the **Always-On** area is always powered-on, in all operating modes and stand-by modes
- The power supply of the **Isolated area** can be turned off to reduce the overall power consumption depending on the types of stand-by mode



FIGURE—47 HARDWARE DESIGN FOR F1L – 176 PIN MICROCONTROLLER

Always-On-Area and Isolated-Area refer to the power supply domains. Always-On-Area (AWO) is an always-on power supply, and the Isolated_Area (ISO) is a power supply that is switched on or off according to the operation mode.

Note: To explain further,

1. In deepstop mode the ISO area is switched-off hence all the settings and data of the components in ISO area will be lost
2. The INTC – interrupt controller of the system is available in ISO area in deepstop mode will be shutdown and to enable the unit to respond to wakeup factors in deepstop mode and LPS and a Standby Controller (STBC) is used
3. The LPS shall monitor for the wakeup factors as explained above. And once an wake-up event is recognized, LPS indicated the STBC to raise the appropriate interrupt for the wake-up to be acknowledged

As described in the previous chapters the two main modes supported by the microcontroller are:

- **HIGH Power Mode (RUN MODE):** Here the ECU State Manager [ECUM] manages the ECU transitions {START, RUN, SHUTDOWN, WAKE-UP, SLEEP}.
- **LOW Power Mode**: This Mode occurs when ECU goes to SLEEP State.

Additionally, there shall be two possible modes in Low Power as mentioned above:

1. **Stop Mode** – (module clock gating) - Wherein, the clocks of the components that are not required are stopped. In stop mode the configuration and data of the units are preserved, hence when the vehicle switches to high power mode. The components resume their operations from where it had been paused. In this mode the system continues its operation from where it had been paused. The supply to RAM is retained and the microcontroller is retained, hence OS_Context and data are not lost – On transition from LP to HP the program continues from where it had been stopped

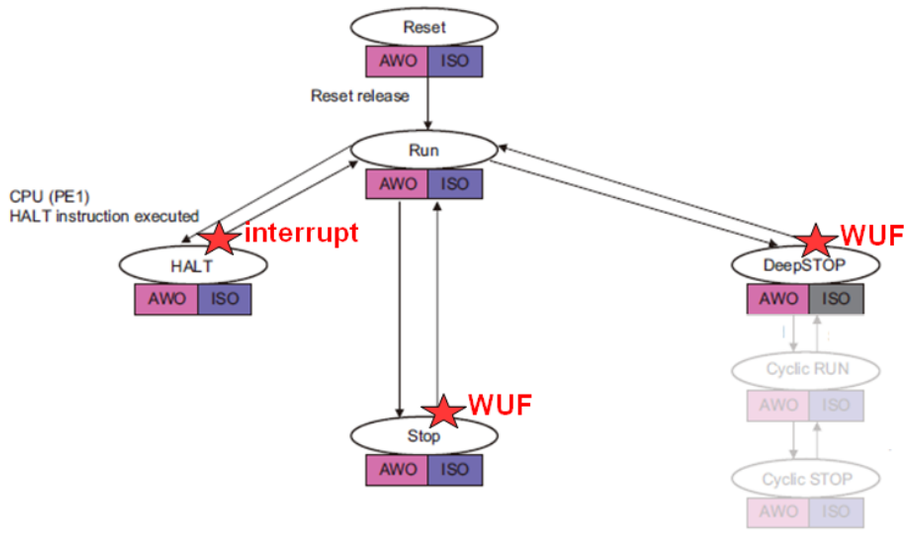
2. **Deepstop Mode** – (module power gating) - In this mode, the components that belong to the isolated area (ISO) are switched off. This mode offers maximum reduction in power consumption. The RAM (except for Retention RAM) is located in the isolated area and shall also be switched off, hence the components whose configuration and data are stored in RAM (except for Retention RAM) shall be lost and due to this the system would have to be restarted when the vehicle switches to high power mode. The supply to RAM except to the Retention RAM is cutoff and the complete ISO* area is switched-Off, hence all data including the Os_Context shall be lost – On transition from LP to HP a clean startup / re-initialization needs to be done
   Note: When going back to High Power from DEEPSTOP mode, a software reset will be executed so EcuM states will be switched back to EcuM STARTUP state

The below table list the various modes the microcontroller can be in:

Operation	Description	Mode Transition trigger
RUN	All function are operational	Engine Power On
RUN ECO	LP Scheduling activity	Activity performed by LP_Task
HALT	CPU operation stopped, all clocks operation except clock for CPU core, all areas under power	HALT Instruction
STOP	Several clock supplies are stopped	Engine Power Off - Register change
DEEPSTOP	Power supply of ISO are is turned off	Engine Power Off - Register change

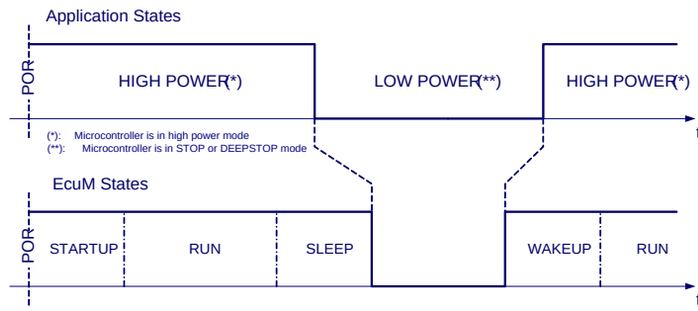
TABLE—10 POWER MODES HANDLED BY THE MICROCONTROLLER

The figure below shows transition between RUN mode and stand-by mode. Wherein, The CPU clock is set to a clock other than the PLL. The CPU starts the operation from the reset vector address on the code flash / on the retention RAM. To return from DeepSTOP mode to RUN mode, ISO peripheral function registers must be set again.



FIGURE—48 TRANSITION TO STAND-BY MODE

Note: BCM* shall remain power supplied at all times as it is located in the main ECU in the Body Control Unit.



FIGURE—49 TIMING DIAGRAM – POWER MODES

9 APPENDIX – C

9.1 INTERESTING TECHNICAL ASPECTS LEARNT

- The LPM main function shall be located in an OS task defined with the **highest priority** at the **highest foreground** level in the application

But how are the high power task executed if the Low power task has the Highest priority

In Bsw – The task are grouped together in something called as foreground groups.
There are basically 3 foreground groups

- Foreground Group 1: task priorities – [99 –to- 199] – Periodic task that are executed in RUN / RUN Eco mode fall into this group.
- Foreground Group 2: task priorities – [200 –to- 240] – LPM Task and ASW task / Asynchronous task (that requires initiation) are executed in this group.
- Foreground Group 3: task priorities – [<99] – Low priority task for example the Idle task are run in this group.

In High Power Mode Foreground 1 is executed and when EcuM_State == SLEEP foreground 3 Idle task is executed and if Low Power is activated for the microcontroller the LPM_Task from foreground 2 shall be executed.

- The basic software contains many task as described above, the application task, OS task, basic software task. These tasks can be periodic or asynchronous (triggered on receiving an event).

There are options that allow executing the asynchronous task with a time delay. That is to say that - When an asynchronous task is triggered due to an event, the task could have a time delay associated with it, hence after the counter elapses the task is executed.

In the LP context when the transition from HP to LP is taking place the OS timer is stopped to avoid any such asynchronous task with time delays to be triggered. Under such scenarios these task are set as pending in LPM mode.

- Wake-up Interrupts and Interrupts with no wake-up capabilities:

Interrupts are very common in embedded context. Here the Interrupts are divided under the context of wake-up capability:

  - Wake-up able interrupts are Interrupts that can trigger the microcontroller to wake up from Hardware Low Power Mode
  - EcuM Wake-up able events that triggers a change in the EcuM so that Low Power mode is existed
  - Interrupts with no wake-up capability

- As we have discussed in the previous section, the Ecus have Autosar layered architecture, according to which it has an application layer and then the basic software layer and then a hardware abstraction Layer and finally the Microcontroller.
  - When the user stops the vehicle (i.e. a keys off operation is performed)
  - The ECU switches to battery mode

- The execution of currently running application task are completed
- No new tasks are initiated. (as the EcuM state is no more equal to RUN)
- A timer is set after which the vehicle is expected to enter LPM
- When the all application task have been terminated, the ECU State is switched to SLEEP MODE
- The Basic Software Idle task is executed
- And If LPM is configured for the ECU
- The Low Power Mode is entered

Please Note: There are many ECUs in the vehicle and all of them contain Basic Software Layer, but not all are configured to operate in LP, most ECUs are stopped when the vehicle is halted. But some of them like the BCM and Gateway are configured to operate in LP for monitoring purposes (as explained in the above sections)

- The Current drain through Deepstop mode is considerably reduced hence saving a lot of battery power, which ensures longer battery life

Note: Current drain – Refers to the current received by a load from a voltage source.

- Run Current linked to Core and technology implementation

	Run Mode		Deep Stop Current	
Single Core 80MHz	Typical 0,3 mA/MHz => 25mA	Maximum 0,75 mA/MHz => 60mA	Typical => 35µA	Maximum => 350µA
Dual Core 2 x 120MHz	Typical 0,4 mA/MHz => 96mA	Maximum 0,75 mA/MHz => 170mA	Typical => 55µA	Maximum => 540µA

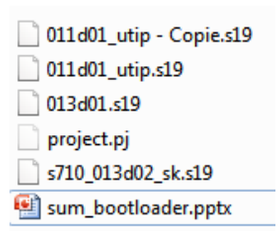
FIGURE—50 CURRRENT DRAIN OFFERED BY RENESAS MICROCONTROLLERS

BootLoader / FBL – Flash Boot Loader

In '**Continental**', Software Platform, most of the Boot loader projects has the FBL with two main purposes:

1. EOL – End Of Line Test – used for hardware Testing. To verify if all the microcontroller components are working as expected, if the components are initialized correctly, for example: the communication lines like (the CAN/ LIN/SPI etc) are checked to ensure correct operation or to test PWM signal, to measure ADC level, to verify pin toggling
2. For updating the Application Software

In the software platform team of body and security, when the Software is compiled, 4 main binary files are generated. For instance let's take an example of Application build; the output folder will contain 4 main binary files as shown below



FIGURE—51 BINARY FILES GENERATED ON COMPILATION

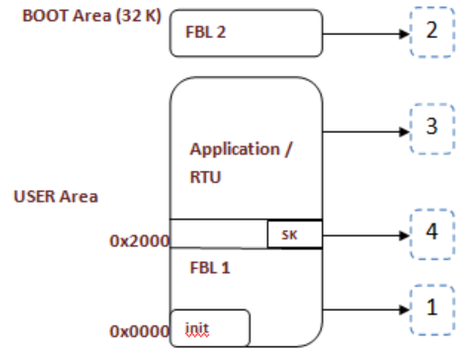
A Security Key is a kind Validation Flag used to indicate the correctness of the build / application. The software is loaded and if the CRC and other conditions are ok the validation key is set to true.
The **.abs** file is used for debugging purposes as it contains the **code and symbols**

- The 013512.s19 file contains a dummy code at address 0x0000 that will jump to the required address in the Application code.
- The s710_013512.s19 file contains application code without security key.
- The s710_013512_sk.s19 file contains the application code with security key.
- Similarly in the FBL side: Utip is just an indication used to suggest that this code had been modified using Utip Tool (to include additional information such as CRC, security key etc)

- The s710_013d02_sk.s19 file indicates that this build can be used when the application code with the security key shall be flashed.

Now the microcontroller has the below configuration:

1. FBL – part 1 – start address is 0x0000 mainly used on every reset - has very less functionality - points to the location of FBL 2
2. FBL – part2 – present in the boot location. Carry outs the boot loader functionality and if the application is valid will jump to the application code address
3. Application code / RTU
4. Security Key



FIGURE—52 STRUCTURALISATION OF FLASH BOOTLOADER

There are 2 ways of flashing software

1. Software flashed during production (for e.g using JTag). Here the complete code vase including FBL, Application etc are all included in one big xxxx.s19 and flashed. After which the EOL test is done to verify the product is working as expected, then the product is deployed into the car
2. Application update through FBL
   Software is flashed through CAN – (mostly done during the life time of the device after production)
   - The Application is indicated that a re-flashing will occur
   - Go to FBL mode
   - SA – Security Access [algorithm to check if update is performed by valid source]
   - Erase old application
   - Write new application
   - Validate application using CRC, if valid application is flashed a security key is generated and written in a memory location in the application software else the security key area is left blank or with wrong data.
   - Reset
   - If a valid security key is found then the device boots up by launching application code or else remains in boot loader mode

Note 1: The Application flashed during development. During this time if one is an application code developer the boot loader is not really required for testing of application code so the .s19 with dummy boot loader is flashed that just basically jumps to the address of the application code.

Note 2: When one opens the .s19 in a notepad the address can be read. And in multi tool (debugging tool) we can enable assembly code by selecting assembly option use memory option to view the memory information and the instructions stored in there.

10 BIBLOGRAPHGY

Site for Principal of Continental Automotive:

http://www.continental-automotive.com/www/automotive_de_en

Site for Principal of Continental Automotive / Interior:

http://www.continental-automotive.com/www/automotive_de_en/general/interior/

Site for Principal of Continental Automotive / Interior / Body & Security:

http://www.continental-automotive.com/www/automotive_de_en/general/interior/business_units/bu_body_security_en.html

Specification of RH850 microcontroller and details of series F1L and F1H:

http://am.renesas.com/products/mpumcu/rh850/rh850f1x/index.jsp

Site for Diolan board specification

https://diolan.com/dln-4m

Other project related documents

DualCore_SPI_Debug_User_Guide.doc
DualCore_Prototype.docx
SRS_SDD_PowermodeHandler_Honda_MY20_PCU_PatHaYaMu.doc