REFLECTIVE REPORT

Embedded System

YI JIE LIM 20104720

1. Table of Contents

2. OBJECTIVES

The focus of the Body Control Module (BCM) that is developed in this case studies are to manage different vehicle sensors and actuators (except for vehicle controls) particularly those of the Heating, Ventilation and Air Conditioning (HVAC) system. The project covers embedded system design issues including low power modes, fault tolerance and timing analysis by Infineon XMC1400 ARM microcontroller, together with CAN (Controller Area Network) vehicle networking. A first BCM implementation controls the blower motor fan speed based on settings from the HVAC panel received on CAN bus and generates a control strategy that causes the fan output to change. Added features of advanced HVAC include regulating air temperature with hot and cold air mixing and with air flow using motorized flaps. Infineon DAVE for coding, μC/Probe for monitoring, and simulated testing on a CAN loopback mode are used as development tools before being integrated with a real CAN network or Defender Rig, offering a practical, hands-on approach to modern automotive control systems.

2.1. CASE STUDY 1

In this case study, a design and implementation of a Body Control Module (BCM) for the HVAC system on the Land Rover Defender is done using the Infineon XMC1400 microcontroller. To do so, the project attempted to drive blower motor speed and temperature regulation via a polling loop structure and to analyze the resulting system performance. According to those who assigned us, the key tasks were to configure CCU4 timers, implement DIGITAL_IO for GPIO control and interpreting CAN messages. Real time monitoring was done with on board LEDs displayed the blower speed (0-7).

Performance analysis involved measuring the Worst Case Execution Time (WCET) of key functions, calculating CPU load (total: A Nyquist Frequency of 13.88 Hz was determined (95%) to ensure that CAN message sampling was accurate, as well as determining if the CAN sample received was within the CAN Constraints (95%). CAN loopback mode was tested for simulation and integration with real world tools such as CANoe and the Defender rig.

Democratic debugging and loop structuring were used to tackle challenges such as optimizing high CPU load and verifying CCU4 configurations.

The proficiency in real time system design, debugging and control was demonstrated in this work, which bodes well for future enhancements like interrupt based message handling and more advanced HVAC related capabilities.

2.2. CASE STUDY 2

In this case study, power consumption analysis is discussed as well as the efficiency optimization of the XMC4200 microcontroller. Differentiation between system performance and device power usage in different circumstances: TestMode and ProductionMode with and without usage of WFI (Wait-For-Interrupt). Power consumption was measured in terms of active current and CPU load at various configurations when considering how external factors such as clock speeds and fan control affect power consumption. Calculations were performed through the use of the XMC1400 datasheet and confirmed that the current readings provided by the microcontroller at 5V reference voltage meet the systems' requirements.

Finally, there were also coding changes to manage peripherals like turning the brightness of the LED by regulating the fan speed. Additional interrupt handling mechanisms were introduced so that the chip wouldn’t run away forever, as they improved energy efficiency. SysTimer, Interrupt, and PWM apps were incorporated as critical tools to add to the capabilities and response of the microcontroller under varying operating conditions. These configurations ensured a good trade off between system performance and power consumption.

We then used detailed analysis to understand differences in the current draw and execution time for the different tested scenarios. The knowledge in this thesis helps to optimize the usage of energy in embedded systems without impacting the operation reliability. WFI and proper interrupt management examples make the case study demonstrate that the techniques applied on a microcontroller design should be efficient.

2.3. CASE STUDY 3

In this case study a Real Time Operating System (RTOS) was deployed to manage the Body Control Module (BCM) for the HVAC system of the Land Rover Defender. Motive for switching to an RTOS driven approach (FreeRTOS) based from earlier superloop based design to promote effective task management and performance optimization was the prime goal. The BlowerCtrl and BlowerTest functions were converted into RTOS threads which will run round robin at equal priority. The existence of this shift shows the advantages of multitasking and inter thread communication in the embedded systems.

Configuring the CMSIS-RTOS app to include FreeRTOS and a timer tick for runtime statistics was one of the key tasks. Using osDelay() to introduce thread delays of 100ms for BlowerCtrl and 200ms for BlowerTest, task execution occurred at a Nyquist frequency, further reducing CPU load. As with the other study, it emphasized RTOS debugging using tools such as the Segger Ozone debugger and the FreeRTOS runtime statistics to calculate CPU utilization. Through optimization it was observed that the idle tasks consumed most of the CPU time and the system efficiency is improved.

The use of the RTOS for embedded systems in this case study was illustrated as a practical way to achieve more improved task scheduling, reduced CPU load, and better modularity. The potential for further system enhancements was enabled by implementing RTOS concepts like threads, delays, and runtime analysis enabling the BCM to become a more robust and scalable design.

2.4. CASE STUDY 4

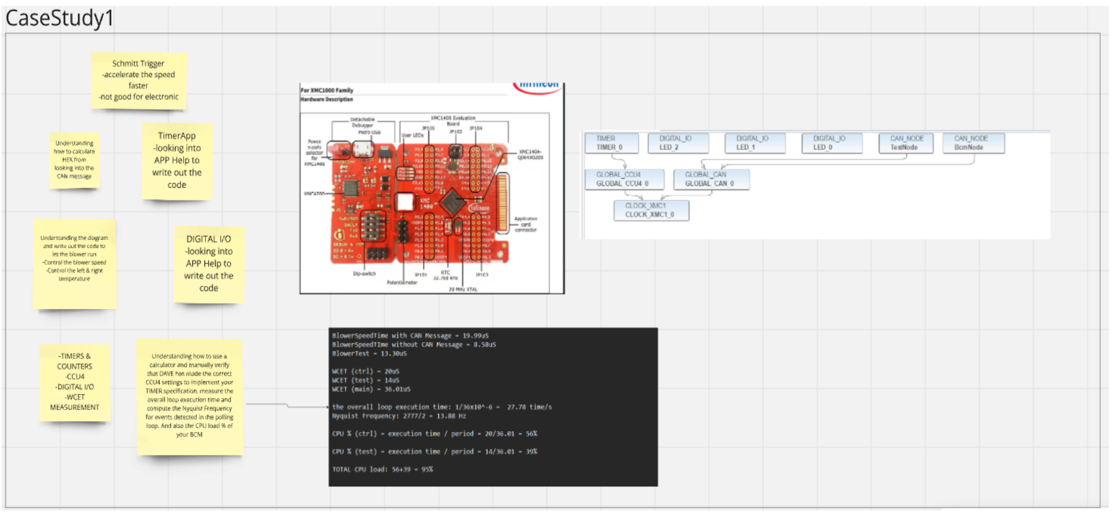
The objective of this case study was to improve the RTOS based Body Control Module (BCM) for the HVAC system by adding inter task communication and task scheduling improvements. We were interested in using Rate Monotonic Scheduling (RMS) that schedules tasks according to their periods so that the deadlines are met and the system runs efficiently within real time boundaries. This case study builds upon the work done in Case Study 3 and attempts to modularize tasks and communicate between them with message passing, which lowers CPU load and improves task management process.

Understanding one of the key improvements was switching from a polling method to an ISR for detecting CAN message, thus improving the efficiency and responsiveness of the system. We modularized the BlowerCtrl task away from the CAN node and introduced a message queue to pass HVAC settings back and forth between the CAN ISR and BlowerCtrl. By allowing efficient communication between tasks, the BlowerCtrl task was only allowed to update the system if new data became available, resulting in less processing of data when no new data is available. Its short period allowed the BlowerCtrl task to preempt other tasks when it wanted, which the system was designed so the BlowerCtrl task had the highest priority based on that.

However, for system performance, only Rate Monotonic Scheduling (RMS) was used such that a tasks rate represents its task priority where the task with the shortest period has the highest task priority. The total CPU utilization was calculated to be only 0.0245%, which indicates that the total processor resources were used efficiently to meet the system's real time deadlines. A Watchdog Timer (WDT) was added to further improve the reliability of the system so that errors or system lockups would cause the system to reset itself, preventing runaway code or otherwise unresponsive behaviour. Resolving this problem relied on a combination of modular task management, priority scheduling, and reliable communication methods to create a robust RTOS based design that sacrificed nothing in terms of efficiency for reliability, capable of performing tasks withing strict real time constraints with minimal resource usage.

3. CONCLUSION

3.1. CASE STUDY 1


CaseStudy1

Schmitt Trigger
-accelerate the speed faster
-not good for electronic

Understanding how to calculate HEX from looking into the CAN message

TimerApp
-looking into APP Help to write out the code

Understanding the diagram and write out the code to let the blower run
-Control the blower speed
-Control the left & right temperature

DIGITAL I/O
-looking into APP Help to write out the code

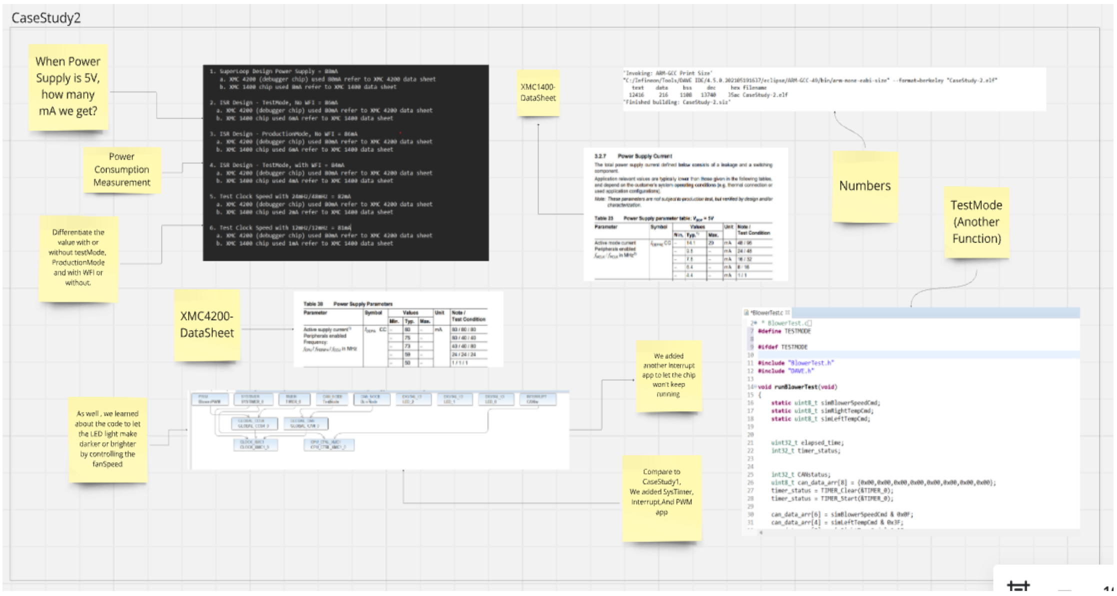
-TIMERS & COUNTERS
-CCU4
-DIGITAL I/O
-WCET
MEASUREMENT

Understanding how to use a calculator and manually verify that DAVE has made the correct CCU4 settings to implement your TIMER specification, measure the overall loop execution time and compute the Nyquist Frequency for events detected in the polling loop. And also the CPU load % of your BCM

```
BlowerSpeedTime with CAN Message - 19.99uS
BlowerSpeedTIme without CAN Message - 8.58uS
BlowerTest - 13.30uS

WCET (ctrl) - 20uS
WCET (test) - 14uS
WCET (main) - 36.01uS

the overall loop execution time: 1/36x10^-6 -  27.78 time/s
Nyquist Frequency: 2777/2 - 13.88 Hz

CPU % (ctrl) - execution time / period - 20/36.01 - 56%

CPU % (test) - execution time / period - 14/36.01 - 39%

TOTAL CPU load: 56+39 - 95%
```

This case study gave me an amazing insight on microcontroller programming and system design. I learnt how to configure and run the digital I/O, timers and counters (CCU4), also decoded the CAN messages and integrate the CAN nodes for real time communication using APP Help tools. Through analyzing an XMC1400 evaluation board I became more acquainted with the hardware architecture and components (LEDs and potentiometers) and learned how to translate system diagrams into functional code. Furthermore, system performance was measured by means of CPU load, Nyquist frequency and, Worst Case Execution Time (WCET), enabling me to optimize resource utilization and execution efficiency. This case study illustrated the importance of theoretical concepts to be applied well to design of efficient embedded systems.

3.2. CASE STUDY 2


CaseStudy2

When Power Supply is 5V, how many mA we get?

Power Consumption Measurement

Differentiate the value with or without testMode, ProductionMode and with WFI or without.

```
1. SuperLoop Design Power Supply = 88mA
   a. XMC 4200 (debugger chip) used 80mA refer to XMC 4200 data sheet
   b. XMC 1400 chip used 8mA refer to XMC 1400 data sheet

2. ISR Design - TestMode, No WFI = 86mA
  a. XMC 4200 (debugger chip) used 80mA refer to XMC 4200 data sheet
  b. XMC 1400 chip used 6mA refer to XMC 1400 data sheet

3. ISR Design - ProductionMode, No WFI = 86mA
  a. XMC 4200 (debugger chip) used 80mA refer to XMC 4200 data sheet
  b. XMC 1400 chip used 6mA refer to XMC 1400 data sheet

4. ISR Design - TestMode, with WFI = 84mA
  a. XMC 4200 (debugger chip) used 80mA refer to XMC 4200 data sheet
  b. XMC 1400 chip used 4mA refer to XMC 1400 data sheet

5. Test Clock Speed with 24mHz/48mHz = 82mA
  a. XMC 4200 (debugger chip) used 80mA refer to XMC 4200 data sheet
  b. XMC 1400 chip used 2mA refer to XMC 1400 data sheet

6. Test Clock Speed with 12mHz/12mHz = 81mA
  a. XMC 4200 (debugger chip) used 80mA refer to XMC 4200 data sheet
  b. XMC 1400 chip used 1mA refer to XMC 1400 data sheet
```

XMC1400-DataSheet

```
'Invoking: ARM-GCC Print Size'
"C:/Infineon/Tools/DAVE IDE/4.5.0.202105191637/eclipse/ARM-GCC-49/bin/arm-none-eabi-size" --format=berkeley "CaseStudy-2.elf"
   text    data     bss     dec     hex filename
  12416     216    1108   13740    35ac CaseStudy-2.elf
'Finished building: CaseStudy-2.siz'
```

3.2.7 Power Supply Current

The total power supply current defined below consists of a leakage and a switching component.

Application relevant values are typically lower than those given in the following tables, and depend on the customer's system operating conditions (e.g. thermal connection or used application configurations).

Note: These parameters are not subject to production test, but verified by design and/or characterization.

Table 23 Power Supply parameter table; V_{DDP} = 5V

Parameter	Symbol		Values			Unit	Note / Test Condition
			Min.	Typ.[1]	Max.		
Active mode current Peripherals enabled f_{MCLK} / f_{PCLK} in MHz[2]	I_{DDPAE}	CC	–	14.1	20	mA	48 / 96
			–	9.8	–	mA	24 / 48
			–	7.8	–	mA	16 / 32
			–	6.4	–	mA	8 / 16
			–	4.4	–	mA	1 / 1

Numbers

TestMode (Another Function)

XMC4200-DataSheet

Table 30 Power Supply Parameters

Parameter	Symbol		Values			Unit	Note / Test Condition
			Min.	Typ.	Max.		
Active supply current[1] Peripherals enabled Frequency: f_{CPU} / f_{PERIPH} / f_{CCU} in MHz	I_{DDPA}	CC	–	80	–	mA	80 / 80 / 80
			–	75	–		80 / 40 / 40
			–	73	–		40 / 40 / 80
			–	59	–		24 / 24 / 24
			–	50	–		1 / 1 / 1

As well , we learned about the code to let the LED light make darker or brighter by controlling the fanSpeed

We added another Interrupt app to let the chip won't keep running

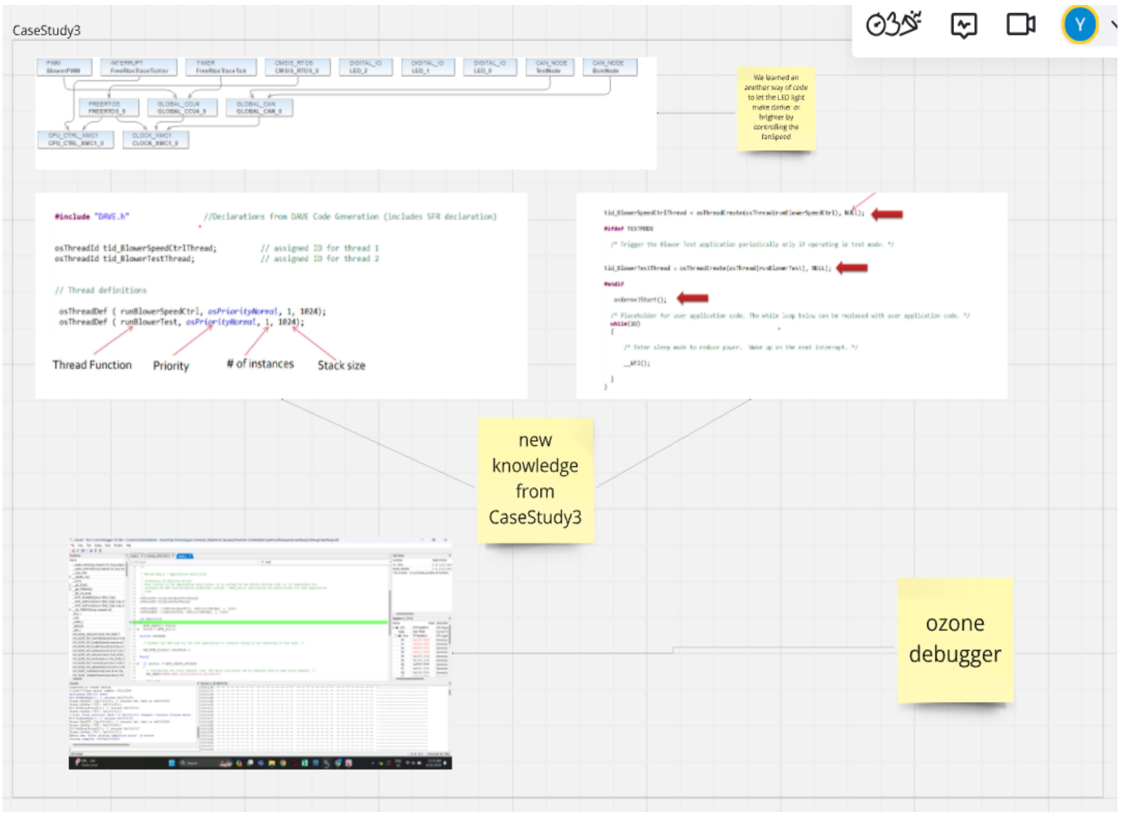
Compare to CaseStudy1, We added SysTimer, Interrupt,And PWM app

*BlowerTest.c

```c
 * BlowerTest.c
#define TESTMODE

#ifdef TESTMODE

#include "BlowerTest.h"
#include "DAVE.h"

void runBlowerTest(void)
{
    static uint8_t simBlowerSpeedCmd;
    static uint8_t simRightTempCmd;
    static uint8_t simLeftTempCmd;


    uint32_t elapsed_time;
    int32_t timer_status;


    int32_t CANstatus;
    uint8_t can_data_arr[8] = {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
    timer_status = TIMER_Clear(&TIMER_0);
    timer_status = TIMER_Start(&TIMER_0);

    can_data_arr[6] = simBlowerSpeedCmd & 0x0F;
    can_data_arr[4] = simLeftTempCmd & 0x3F;
```

By going through this case study, I learnt the mechanism of power consumption measurement by examining how different parameter configurations came into place (TestMode, ProductionMode) and how it is interpreted in the XMC4200 datasheet and XMC1400 datasheet. I became able to distinguish the power supply values across these modes, as well as learn what interrupt and SysTimer apps can do to improve chip performance by preventing continuous operation. I also created code that controls fan speed, along with LED brightness, since it does it using PWM, it helps to handle hardware properly. One of the contributions of this study is the point that datasheet specifications need to be integrated with tangible, coding, and system optimization in order to achieve efficient embedded systems design.

3.3. CASE STUDY 3

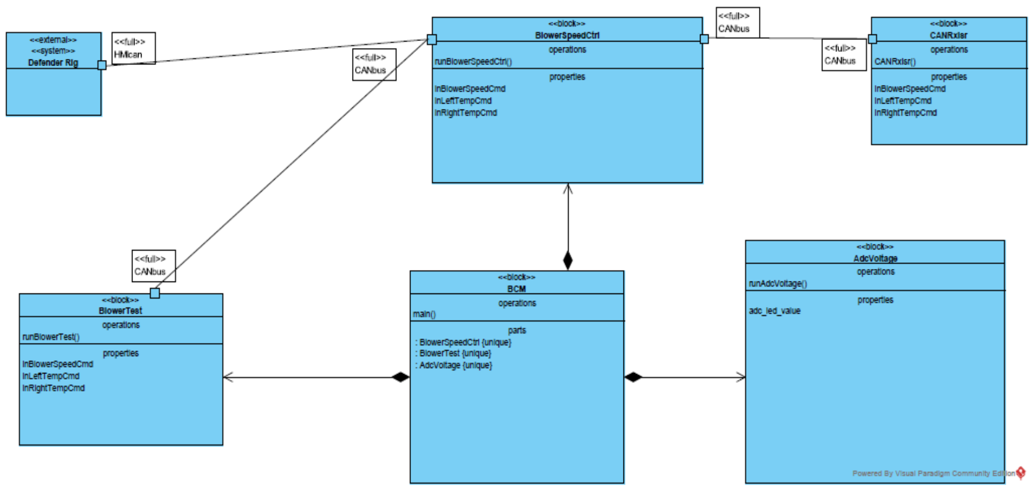


```
#include "DAVE.h"                //Declarations from DAVE Code Generation (includes SFR declaration)

osThreadId tid_BlowerSpeedCtrlThread;          // assigned ID for thread 1
osThreadId tid_BlowerTestThread;               // assigned ID for thread 2

// Thread definitions

 osThreadDef ( runBlowerSpeedCtrl, osPriorityNormal, 1, 1024);
 osThreadDef ( runBlowerTest, osPriorityNormal, 1, 1024);
```

```
tid_BlowerSpeedCtrlThread = osThreadCreate(osThread(runBlowerSpeedCtrl), NULL);
#ifdef TESTMODE
  /* Trigger the Blower Test application periodically only if operating in test mode. */

tid_BlowerTestThread = osThreadCreate(osThread(runBlowerTest), NULL);
#endif
   osKernelStart();
  /* Placeholder for user application code. The while loop below can be replaced with user application code. */
  while(1U)
  {
      /* Enter sleep mode to reduce power.  Wake up on the next interrupt. */
      __WFI();
  }
}
```

In Case Study 3, I learned more about what thread creation and management within an embedded system look in FreeRTOS with its priorities, stack sizes allocation, as well as function definitions. I used FreeRTOS threads to schedule tasks such as blower speed control and to run test functions, to maintain the efficiency of multithreading and real time scheduling. I also explored using the Ozone Debugger to debug and analyze the system behavior so that it would not only help me identify problems, but solve it fast. Another approach to changing the brightness of an LED is manipulating fan speed, which extended my understanding of hardware software integration. This case study served to delve in to FreeRTOS based real time operating systems, efficient embedded system design and debugging tools.

3.4. CASE STUDY 4



After completed this task and I learned about the significance of modular design and how an RTOS helps ensure good task management and communication within embedded systems. All the tasks I created had different priorities, bugs with creating message queues which can be used for safe transfer of data between an ISR and tasks, and I gained hands of experience with polling hardware peripherals such as ADCs using the FreeRTOS framework. I knew how to develop and debug the tasks in a real time system using Ozone debugger to monitor how much CPU the task is consuming, how the task is running and what is its state. I also learned how to introduce new features, like an ADC voltage and LED control, to an existing system without disturbing higher priority tasks showing the versatility and scalability of RTOS based designs. This task provided a good opportunity to part the importance of testing and validation of the system and by means of tools such as CANoe also to ensure CAN communication and hardware interaction can be trusted. In general, I found that this was a good focused task that helped me better understand the real time embedded systems as well as the value of structured and multi-tasking designs.