


ScholarWorks at WMU An Implementation of Controller Area Network Bus Analyzer Using Microblaze and Petalinux

An Truong Kien

Related papers

[Download a PDF Pack](#) of the best related papers 



8-2013

An Implementation of Controller Area Network Bus Analyzer Using Microblaze and Petalinux

Tung-Hsun Tsou

Western Michigan University, gotolafa@gmail.com

Follow this and additional works at: http://scholarworks.wmich.edu/masters_theses



Part of the [Computer Sciences Commons](#)

Recommended Citation

Tsou, Tung-Hsun, "An Implementation of Controller Area Network Bus Analyzer Using Microblaze and Petalinux" (2013). *Master's Theses*. Paper 174.

This Masters Thesis-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Master's Theses by an authorized administrator of ScholarWorks at WMU. For more information, please contact maira.bundza@wmich.edu.



AN IMPLEMENTATION OF CONTROLLER AREA NETWORK BUS
ANALYZER USING MICROBLAZE AND PETALINUX

by

Tung-Hsun Tsou

A thesis submitted to Graduate College
in partial fulfillment of the requirements
for the degree of Master of Science in Engineering (Electrical)
Electrical and Computer Engineering
Western Michigan University
August 2013

Thesis Committee:

Janos L. Grantner, Ph.D., Chair
Massood Zandi Atashbar, Ph.D.
Bradley J. Bazuin, Ph.D.

AN IMPLEMENTATION OF CONTROLLER AREA NETWORK BUS ANALYZER USING MICROBLAZE AND PETALINUX

Tung-Hsun Tsou, M.S.

Western Michigan University, 2013

This paper presents a controller area network (CAN) monitor system created in a Field-Programmable Gate Array (FPGA) board, which is Xilinx SP605. The goals of this research are to let the system demonstrate a reliable CAN bus monitor system, and to show the Xilinx MicroBlaze and PetaLinux design flow. This system can be used to observe the CAN bus messages by means of C program and embedded Linux environment.

A Xilinx MicroBlaze soft processor is used to read CAN information from the external CAN bus controller, Microchip MCP2515. Two implementations, a stand-alone system and a Linux system, are built to control the bus controller and display the result of CAN messages on terminal. Meanwhile, PetaLinux, a specialized Linux distribution, is used to run the entire monitor system. A Linux kernel driver for this CAN bus controller is established using the PetaLinux design tool. Moreover, a CAN bus testing environment is built by using TI Stellaris LaunchPad microcontroller to emulate a car control system.

Copyright by
Tung-Hsun Tsou
2013

ACKNOWLEDGMENTS

I would like to begin by acknowledging my most sincere gratitude to my advisor, Dr. Janos L. Grantner, who gave me all the support to help me finish my thesis.

I would also like to acknowledge my thesis committee, Dr. Massood Zandi Atashbar and Dr. Bradley J. Bazuin, who took time to help me finish this thesis with their kind advice. Also, I give my wholehearted thanks to Dr. Mike Tarn and Dr. Pairin Katerattanakul who gave me the opportunity to work in Computer Information Systems so that I could have enough funding to cover my living expenses while doing my research.

I would like to thank my friends in WMU and the Taiwanese Student Association here, especially Lalith P. Narasimhan. I think I could not have finished my thesis without all my friends. A Taiwanese writer said “There are too many people to acknowledge, so we should just thank God.”

Last but not least, I want to thank Dad and Mom for their love and support. Without them, I would not have come to this world and come to WMU. Thank my girlfriend, Pei-Chan. You encourage me and give me the energy to keep working.

Tung-Hsun Tsou

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	ii
LIST OF TABLE S.....	vi
LIST OF FIGURES.....	vii
CHAPTER	
I. INTRODUCTION	1
Background.....	1
Literature Review	1
System Architecture.....	2
Summary	4
II. CAN BUS	5
CAN Frame.....	5
CAN Physical Layer	9
III. HARDWARE	10
Xilinx SP605 Evaluation Board.....	10
CAN Transceiver and Controller Module on SP605.....	11
Ti Stellaris Launchpad	13
Transceiver Extention.....	14
Microchip MCP2515	15
Microchip MCP2551	16
System ACE.....	17

Table of Contents-Continued

CHAPTER	
MicroBlaze.....	18
Interrupt	19
IV. EMBEDDED LINUX.....	22
Petalinux Software Development Kit (SDK)	22
First Stage Bootloader (FS-Boot)	22
Device Tree and Device Tree Generator	22
Platform Device and Platform Driver.....	23
Linux Module	24
V. DESIGN AND IMPLEMENTATION	26
CAN Testing Bench.....	26
Xilinx SDK- C Program	27
PetaLinux Module.....	31
VI. CONCLUSION.....	34
REFERENCES	36
APPENDICES	41
A. CAN_01.c	41
B. CAN_02.c	46
C. CAN_03.c	50
D. MCP2515.h	53
E. SPI_driver.h	59

Table of Contents-Continued

APPENDICES

F. SPI.h	61
G. SPI.c	62
H. INTC_driver.h.....	67
I. CAN.c	69
J. Xilinx.dts.....	71
K. SPI_driver_linux.h.....	79
L. MCP2515_fifo.c.....	81
M. Unfinished Network CAN driver.....	89

LIST OF TABLES

4.1	Device Tree Code	23
4.2	Platform Driver Structure	24
4.3	Linux Module.....	24
5.1.	Messages	27
6.1.	Comparison	34

LIST OF FIGURES

1.1. System Diagram.....	3
1.2. System Architecture.....	3
2.1. Data Frame and RTR Frame.....	6
2.2. Error Frame.....	7
2.3. Overload Frame.....	8
2.4. CAN Connection.....	9
2.5. CAN_H and CAN_L.....	9
3.1. Xilinx SP605 and FMC XM105.....	10
3.2. CAN Transceiver and Controller Module.....	11
3.3. Schematic.....	12
3.4. Schematic.....	12
3.5. Schematic.....	12
3.6. TI Stellaris Launchpad.....	13
3.7. Test Node Module.....	14
3.8. Schematic.....	14
3.9. Schematic.....	15
3.10. MCP2515.....	15
3.11. MCP2515 Block Diagram.....	16
3.12. MCP2551.....	16
3.13. System ACE Configuration.....	17

List of Figures – continued

3.14. MicroBlaze Core Block Diagram	18
3.15. Interrupt Controller	19
3.16. AXI SPI.....	20
3.17. IRQ Pin	20
3.18. UCF.....	21
5.1. Continuous Burst	26
5.2. Discontinuous Burst.....	26
5.3. 64 Frames Circular Queue	28
5.4. Software Flow	29
5.5. Reset and Initial Setup For MCP2515	30
5.6. Read Rx Buffer on MCP2515.....	30
5.7. Result	31
5.8. PetaLinux Power On	32
5.9. Linux Module.....	33

CHAPTER I

INTRODUCTION

Background

Due to the advanced semiconductor technology, Field Programmable Gate Array (FPGA) contains more and more logic gates nowadays. We can even implement a microprocessor into a single FPGA chip by using hardware description language, such as VHDL and Verilog. Therefore, FPGA manufacturers and OpenCores community provide well written soft-core processor for designers to reduce development time. Xilinx is one of the FPGA manufacturers that support complete soft processor design flow for academic research and industrial innovation.

This thesis creates a Controller Area Network (CAN) bus analyzer based on Xilinx Spartan-6 FPGA development board and Xilinx soft CPU MicroBlaze. We use Xilinx Platform Studio to build the MicroBlaze system. In addition, a stand-alone C code driver and an Embedded Linux driver are implemented for CAN monitor. Xilinx Software Development Kit is used for C code. On the other hand, PetaLinux SDK is used for Embedded Linux driver.

Literature Review

Controller Area Network, CAN, was first introduced by Bosch in the early 1980s [1]. It solved complicated car problem caused by the fact that devices did not exist independently. The entire device, comprising engine ignition, anti-braking system, and Electronic Stability Control, makes car electronic devices complex. CAN has been

employed in trains, boats, and many other systems due to its reliability. As a result, CAN has become an international standard, i.e., ISO 11898 and ISO 11519.

After 2008, all the cars sold in the U.S. were required to be equipped with CAN bus [2]. CAN is not only used in car industry but also in academic areas. For example, the project of Sunseeker Solar car at Western Michigan University [3] uses CAN to control car devices. Moreover, CAN bus has also been implemented on FPGAs for different monitor applications [4], [5], and [6].

System Architecture

A high-speed CAN Transceiver is used to connect the physical CAN bus and CAN protocol controller. Due to the high license fee for internal MicroBlaze CAN bus controller, an external CAN controller with Serial Peripheral Interface (SPI) is used as a substitute to reduce the total cost of the system. The CAN bus data will be transferred into SPI format and sent to the FPGA. MicroBlaze and other peripherals are implemented into the Xilinx FPGA board. Besides, SPI controller will be implemented to control data from external peripherals and transfer the data to the MicroBlaze. By using PetaLinux and C language application, we can analyze and manage the CAN bus data.

We build a CAN bus test environment by using three TI Stellaris Launchpad. One board broadcasts CAN messages with different device IDs and the others react to the messages with corresponding IDs. The MicroBlaze system listens to all the messages on the bus and shows the result through UART-USB port to the computer screen. Figure 1.1 shows the system diagram and Figure 1.2 shows the actual implementation.

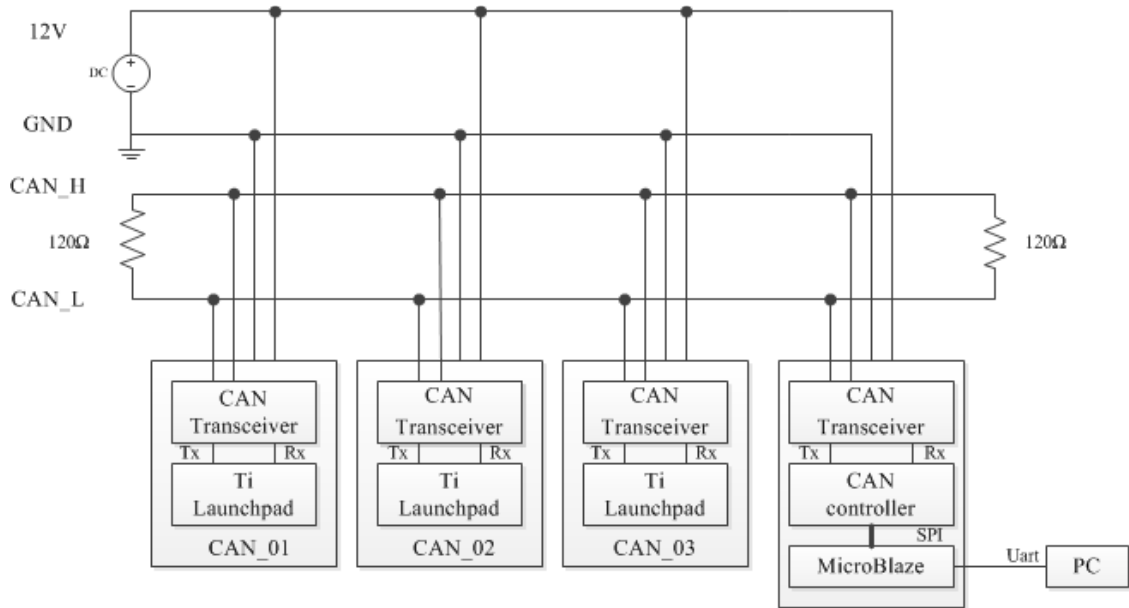


Figure 1.1 System Diagram

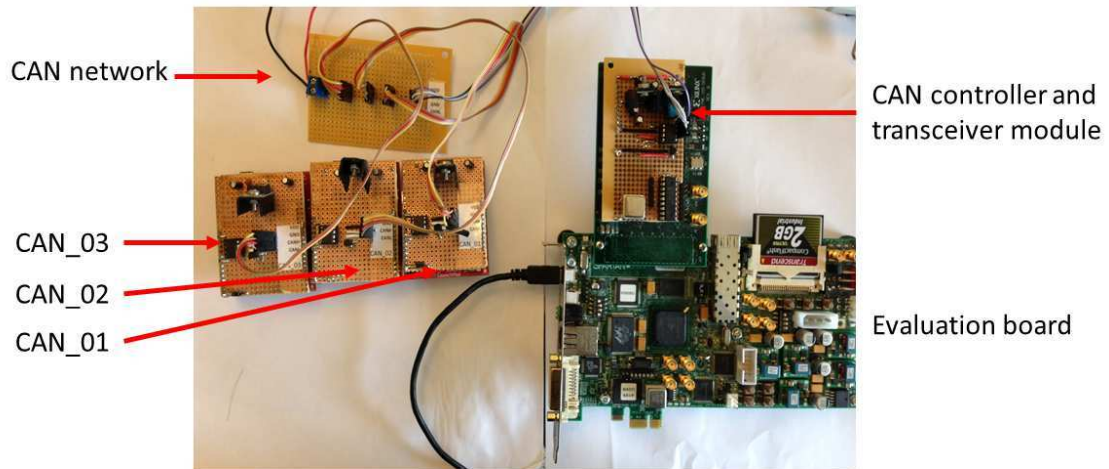


Figure 1.2. System Architecture

Summary

For this thesis, Chapter II is a brief discussion about the CAN bus. Chapter III will talk about hardware implementation. In Chapter IV, we will discuss a lot of the software and Embedded Linux. Chapter V shows the actual implementation and Chapter VI is the conclusion.

CHAPTER II

CAN BUS

CAN bus is defined in Bosch CAN specification [7] as a multi-cast communication protocol, and its advantages are as follows.

1. CAN is a multi-master broadcast system. It means that any node on the bus can communicate with any other node.
2. CAN transmit bitrate can run up to 1Mbit/s.
3. Any new node can be added into the bus without changing the original hardware.
4. It provides error checking to prevent bus faults.
5. Differential CAN signal provides high noise reduction.

Due to the advantages of CAN, the protocol is well used in industry. Also, many microcontrollers have been built in CAN bus for this provides a cheaper product solution. Instances of this use; include Freescale HCS12, Ti ARM cortex M4, and Microchip PIC controllers.

CAN Frame

There are four CAN frame types:

1. Data Frame: it is transmitted from a transmitter to receivers.
2. Remote Frame (RTR): a node can request a Data frame with the same Identifier.
3. Error Frame: it is transmitted by any node when it detects a bus error.
4. Overload Frame: it is used to give extra delay for Data and Remote Frame.

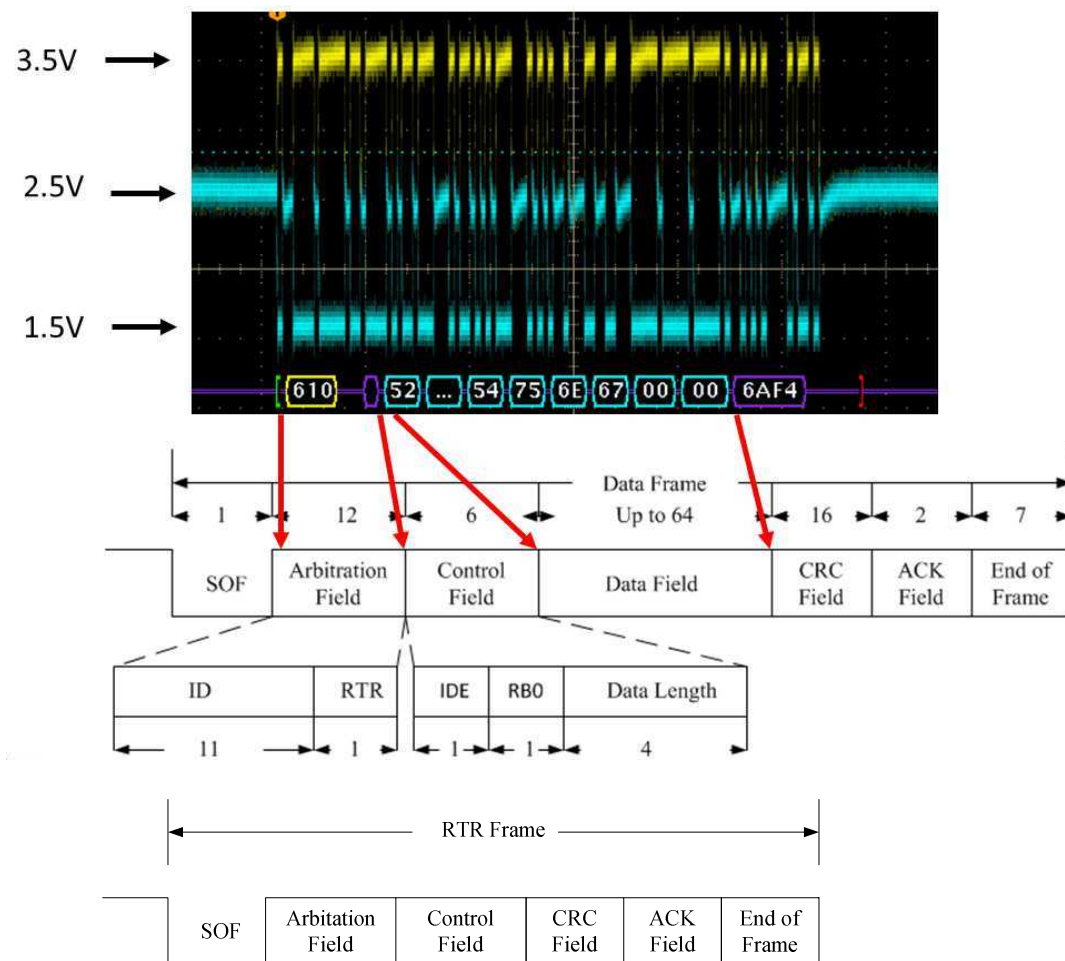


Figure 2.1. Data Frame and RTR Frame

Figure 2.1 shows a Data Frame and RTR frame, including Start of Frame, Arbitration Field, Control Field, Data Field, CRC Field, ACK Field and End of Frame.

Details are as follows:

1. SOF (Start of Frame): It is a “0” that indicates a start of a transmission.
2. Arbitration Field:
 - a. Identifier: Base ID is 11 bits and Extended ID is 29 bits.
 - b. RTR bit: In Data frame, RTR bit is “0”. On the other hand, it is “1” in RTR frame.

3. Control Field:

- a. IDE (Identifier Extension): This bit determines identifier as Base ID or Extended ID.
- b. R0,R1: reserved bits
- c. DLD (Data Length Code): it is used to determine the Data length.

4. Data Field: up to 8 bytes data field.

5. CRC Field (Cyclic Redundancy Check): to check the data correction.

6. ACK Field (Acknowledgement Field): to determine whether the message is received or not. If data is received, this bit will be pulled up high.

7. EOF (End of Frame): “0” shows the end of the message.

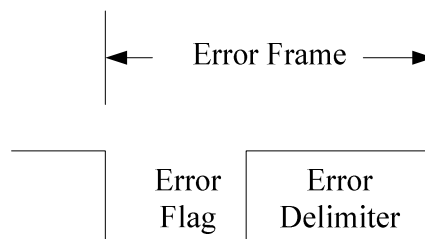


Figure 2.2. Error Frame

Figure 2.2 shows the Error Frame. The Error Frame contains two parts:

- 1. Error Flag: when a node detects an error condition, it generates up to 12 bits “0” for the Error Flag.
- 2. Error Delimiter: 8 bits “1” terminate the Error Frame.

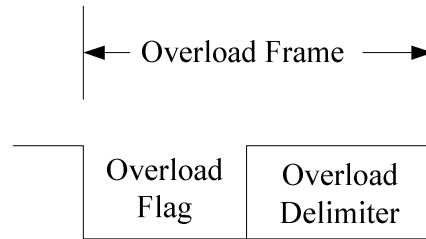


Figure 2.3. Overload Frame

Figure 2.3 is the Overload Frame. It will be generated by the receiving node to force more delay between Data Frames.

CAN Physical Layer

Figure 2.4 shows a CAN connection with four nodes. CAN High (CAN_H) and CAN Low (CAN_L) require two 120Ω terminal resistors. For each node, a transmitter is required to convert the CAN signal into digital Rx and Tx signal for the node controller. Also, according to the CAN specification, a maximum oscillator tolerance is 1.58%. It is important to pick an oscillator with frequency stability and accuracy.

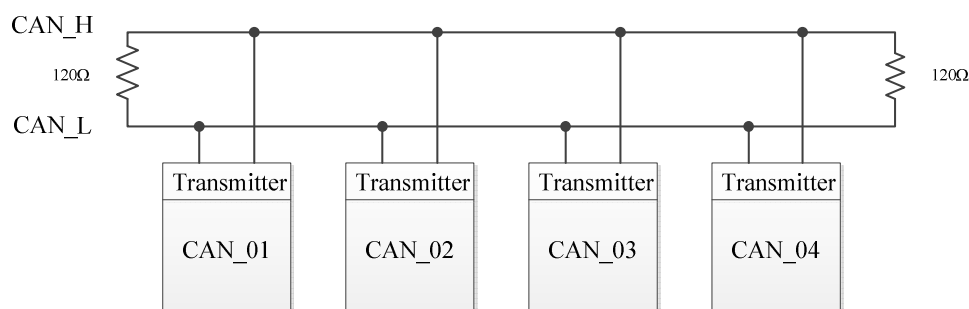


Figure 2.4. CAN Connection

Moreover, CAN_H and CAN_L are differential signals. When two signals are at 2.5V, it means recessive, which is logic 0. When CAN_H goes to 3.5V and CAN_L goes to 1.5V, it means dominant, which is logic 1.

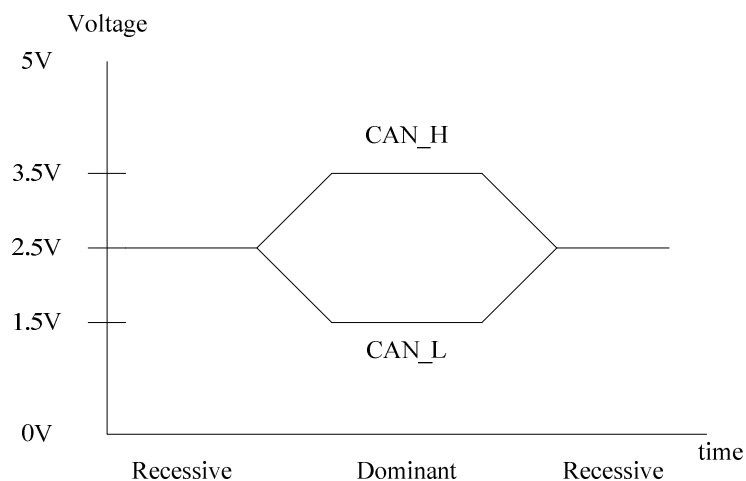


Figure 2.5 . CAN_H and CAN_L

CHAPTER III

HARDWARE

Xilinx SP605 Evaluation Board

Xilinx SP605 evaluation board provides an environment for FPGA designs. We use a Spartan®-6 XC6SLX45T-3FGG484 FPGA in the evaluation board. It contains nice features such as 128 MB DDR3 memories, DVI transmitter, UART, System ACE, and FPGA Mezzanine Connector (FMC) [8]. Designers can add more features through the FMC to extend the board's functionality. In this design, a Xilinx FMC XM105 Debug Card [9] is used to extend the pin accessibility. Figure 3.1 shows the combination of SP605 and FMC XM105.

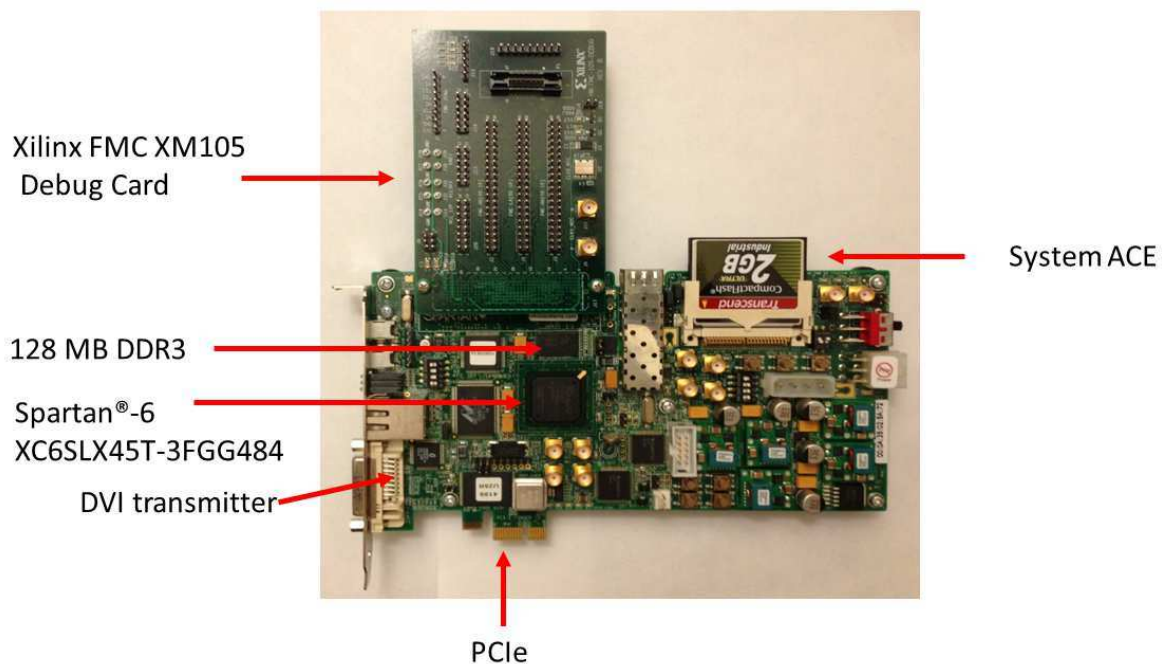


Figure 3.1. Xilinx SP605 and FMC XM105

Based on the board schematics [10] and board block diagram [8], we can see that all four FPGA banks are only powered up to 2.5V. This limits input and output logic level for the board.

CAN Transceiver and Controller Module on SP605

Due to the low voltage input output for the SP605, we need to give lower power supply for MCP2515 to meet the required V_{OH} and V_{IH} . We use one MCP2515 CAN controller, MCP2551 CAN transceiver, Ti 2425c Voltage References, one 7805 and one LM317 voltage regulator to give 5V to the transceiver and 3.3V to the CAN controller. Figure 3.2 is the extend board for controller and transceiver. Figure 3.3, Figure 3.3, and Figure 3.4 show the schematics for the extend board.

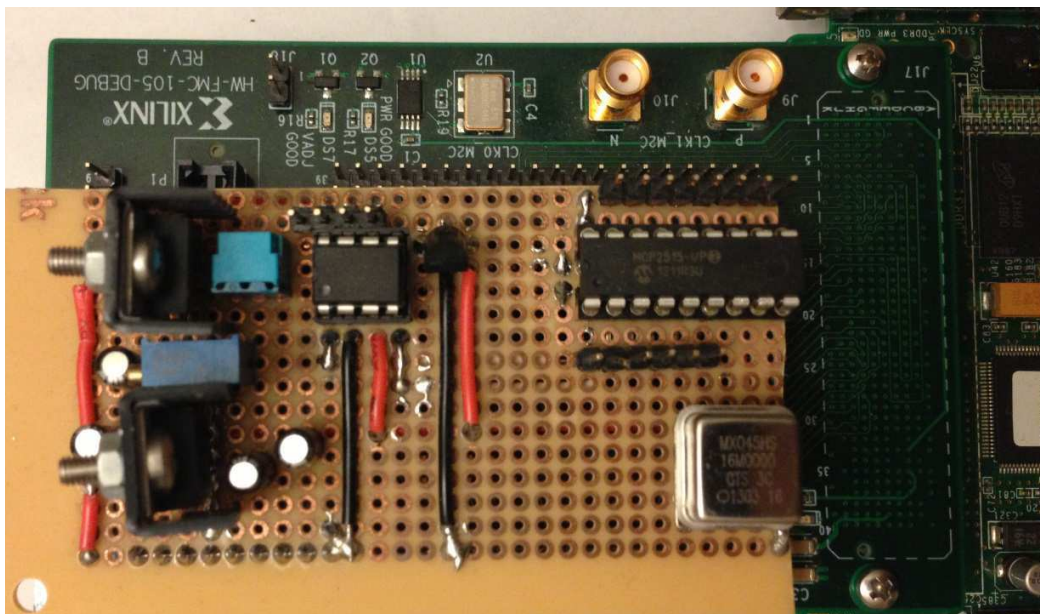


Figure 3.2 CAN Transceiver and Controller Module

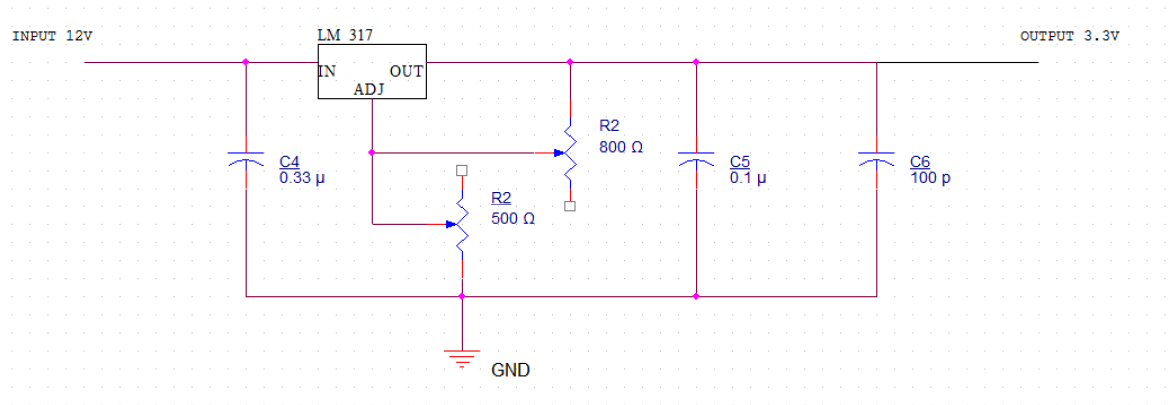


Figure 3.3. Schematic

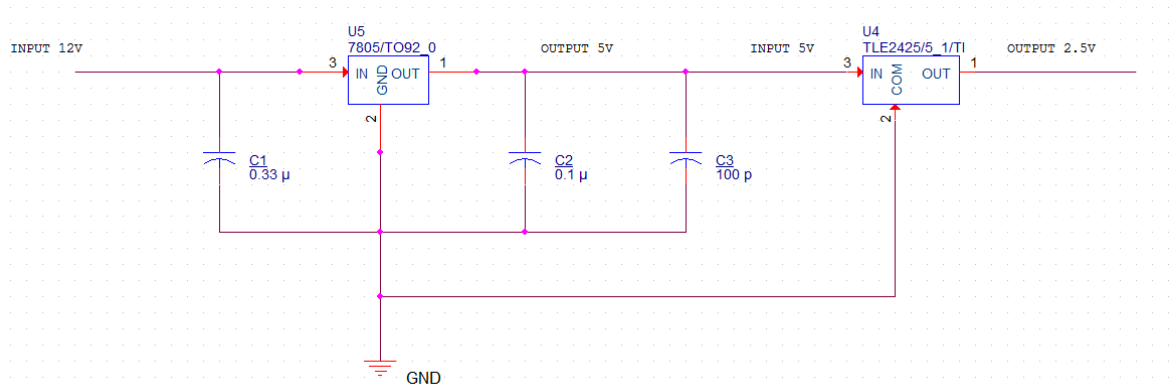


Figure 3.4. Schematic

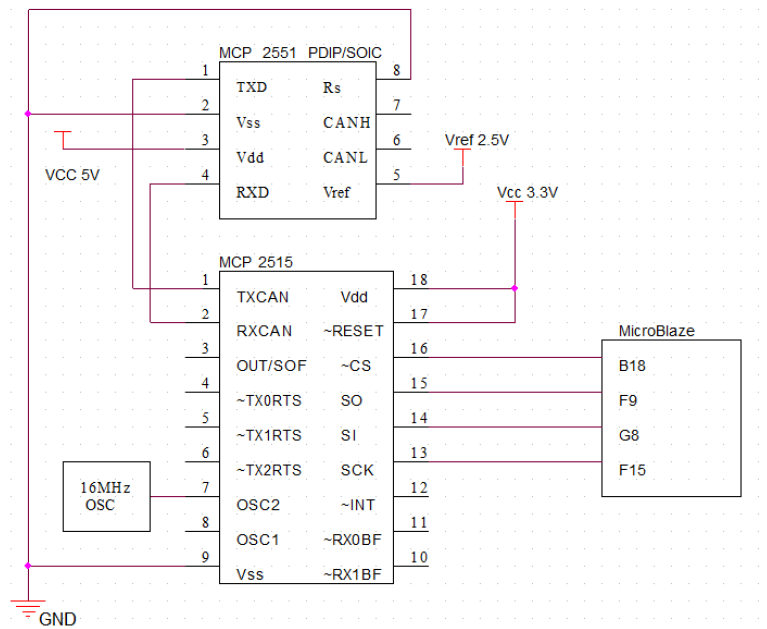


Figure 3.5. Schematic

TI Stellaris Launchpad

TI Stellaris Launchpad is an ideal board to setup a test bench. The advantages are as follows:

1. ARM Cortex M4 can run up to 80MHz.
2. It has a built-in In-Circuit Debug Interface (ICDI) with USB.
3. TI provides free Stellaris Ware built-in functions and free design software Code Composer Studio™, which can help designers to setup a system quickly.
4. It has built-in CAN, I2C, SPI controllers that can be used for most microcontroller applications.
5. It costs only \$12.99.

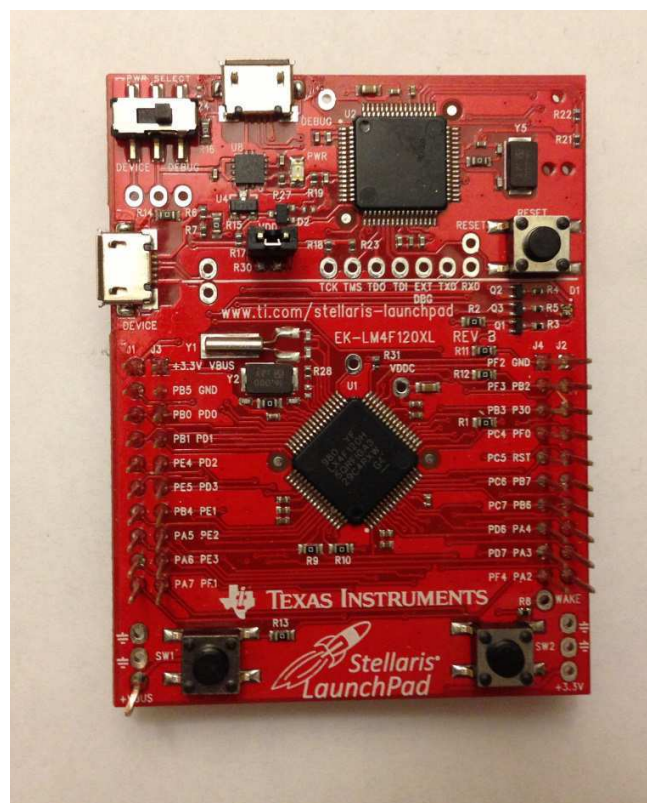


Figure 3.6. TI Stellaris Launchpad

Transceiver Extension

Figure 3.7 shows one of the test Nodes for the CAN network. One node uses a TI Stellaris Launchpad, MCP2551 CAN transceiver, Ti 2425c Voltage References, and 7805 voltage regulators. Figure 3.8 and Figure 3.9 show the schematic of the extend module. This module can be plugged on TI Stellaris Launchpad.

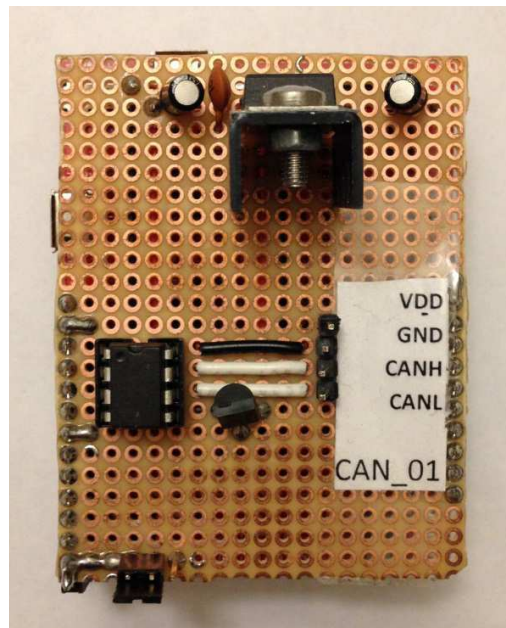


Figure 3.7. Test Node Module

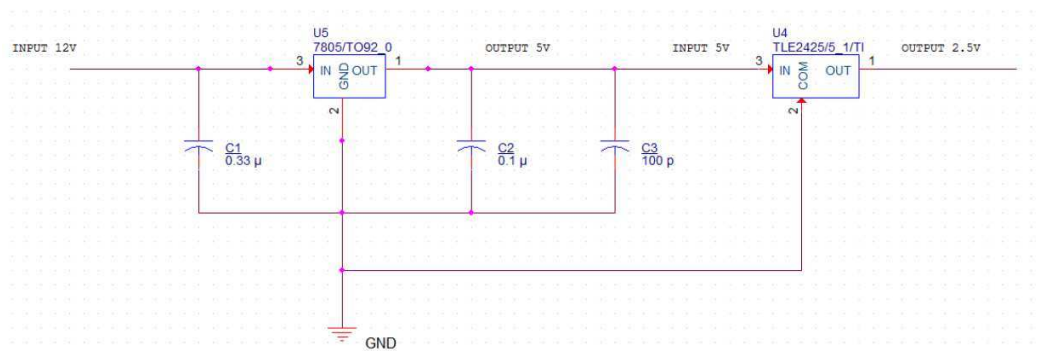


Figure 3.8. Schematic

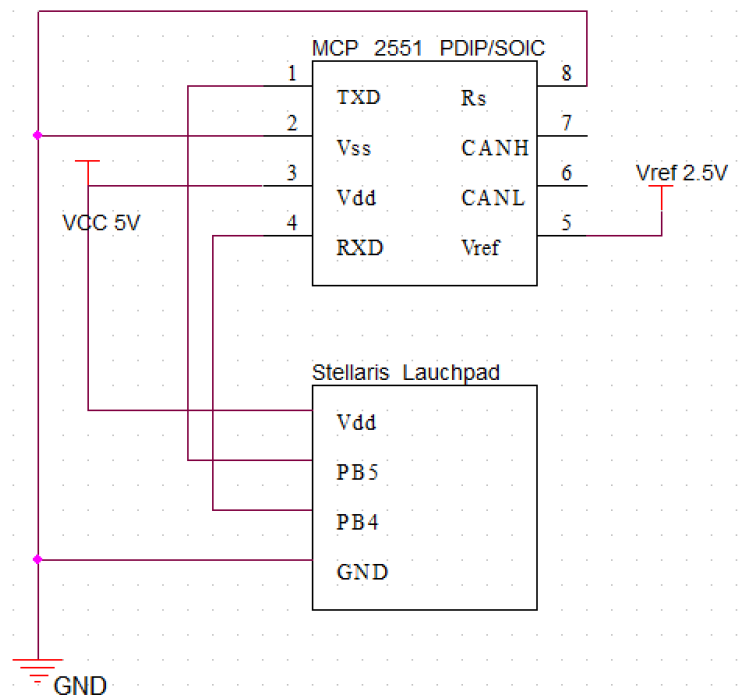


Figure 3.9. Schematic

Microchip MCP2515

MCP2515 [11] is a stand-alone CAN controller. It supports CANV2.0B at 1Mb/s and high-speed SPI transfer.

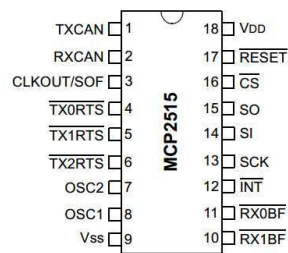


Figure 3.10. MCP2515

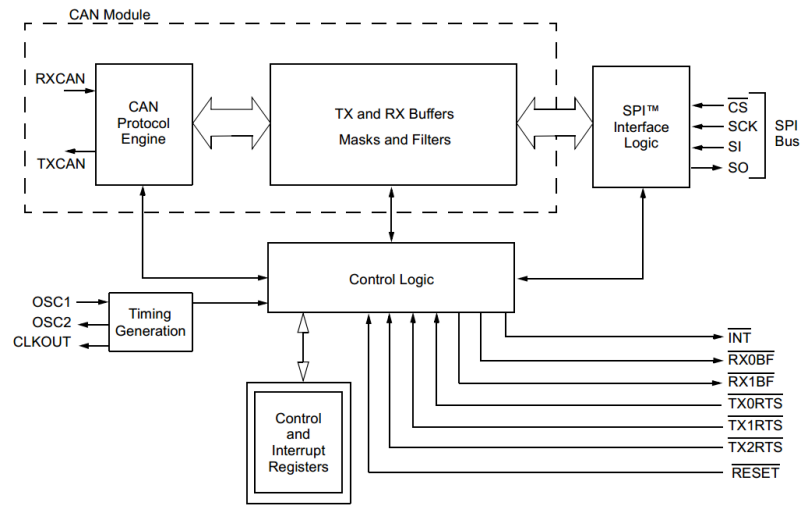


Figure 3.11. MCP2515 Block Diagram

Microchip MCP2551

MCP2551 [12] is a high-speed transceiver. It supports 1Mb/s CAN bus operation. Also, it is suitable for 12V and 24V CAN system.

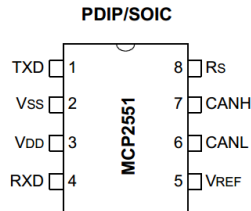


Figure 3.12. MCP2551

System ACE

Xilinx developed the System Advanced Configuration Environment (System ACE), providing a way to combine hardware bit file and software elf file for Xilinx products [13]. It supports up to 8 different configurations. To initialize the System, we need to set up the DIP switch M0 and M1 to “0” and “1” and ACE mode switch to 0001 to read the first configuration.

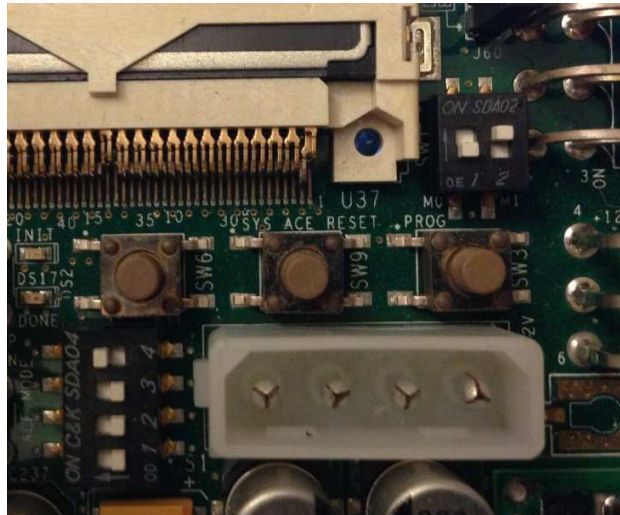


Figure 3.13. System ACE Configuration

MicroBlaze

Xilinx provides a 32 bit reduced instruction set computer (RISC) soft processor which is called MicroBlaze [14]. It is an Intellectual Property (IP) that we can use to design a configurable and flexible system. The MicroBlaze is included with Xilinx ISE Design Suite: Embedded Edition and System Edition. Figure 3.14 shows the block diagram of the MicroBlaze [14].

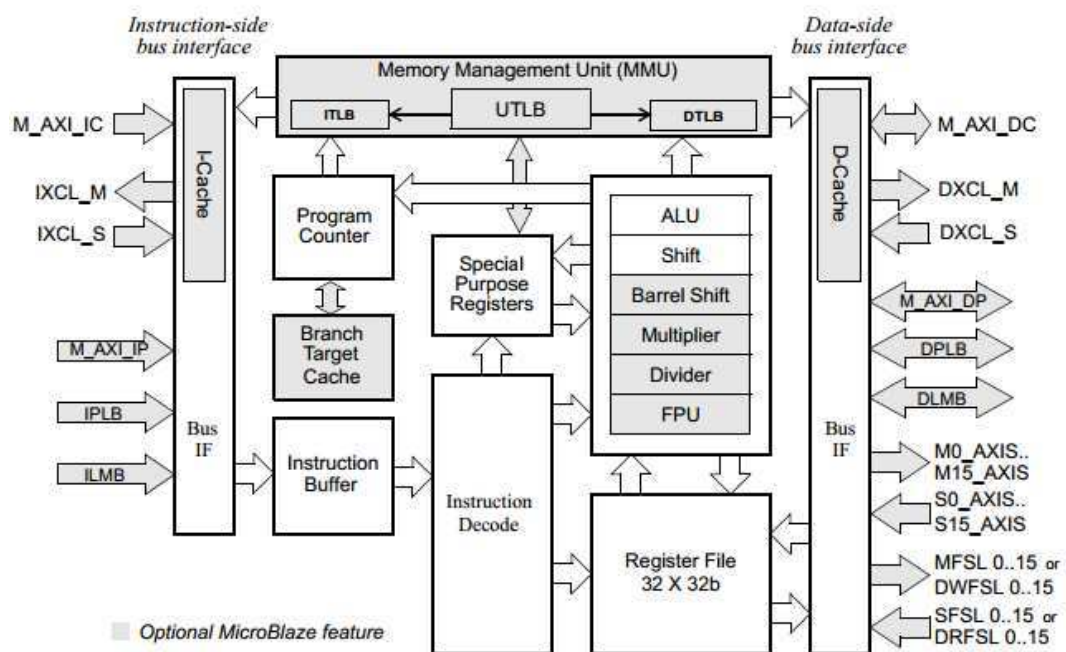


Figure 3.14. MicroBlaze Core Block Diagram

To run Embedded Linux on MicroBlaze, Memory Management Unit (MMU) is required. Also, Petalogix suggests that we should enable at least the barrel shifter feature to improve performance [15].

Interrupt

The interrupt can expand the interrupt input for the MicroBlaze [16]. There is only one interrupt port for the MicroBlaze. By adding the interrupt controller, more than one interrupt can be used in the system. Figure 3.15 shows two ways to connect MicroBlaze interrupt port. Interrupt priority can be set in the Xilinx Platform Studio (XPS) when we configure the hardware.

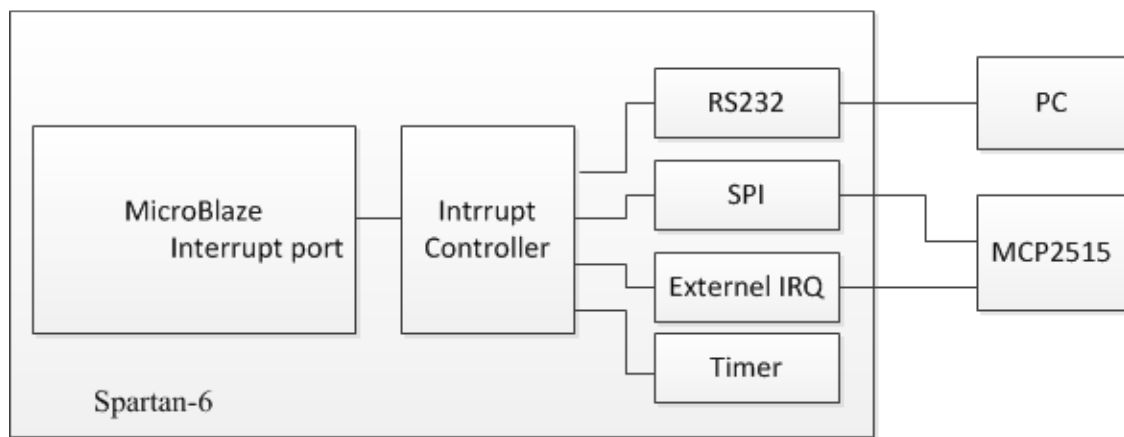


Figure 3.15. Interrupt Controller

Xilinx provides software APIs to handle and manage the interrupt. However, it is difficult and takes a long time and to debug. According to Seshia, the ISR can be done by the following code [17].

```
void myISR(void) __attribute__((interrupt_handler));
```

Also, we need to manually change the MDF file to configure external interrupt (EINT) pin instead of building in GUI interface [18]. Chapter V will show how to set up the EINT pin.

In XPS, we can follow [15] to create a basic AXI bus system by using Base System Builder (BSB) wizard. The following steps show how to create a MicroBlaze.

1. Once we finish the BSB wizard, in the IP catalog, we can add the "AXI SPI Interface" to our system. Advanced eXtensible Interface (AXI) is a substitute for the Processor Local Bus (PLB). It is part of the ARM Advanced Microcontroller Bus Architecture (AMBA) specification [19].

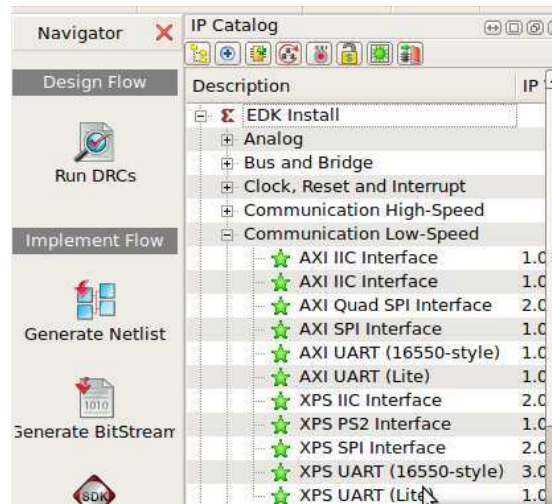


Figure 3.16. AXI SPI

2. For the external IRQ pin, we need to manually add "PORT EINT1_pin = EINT1, DIR = I, SIGIS = INTERRUPT, SENSITIVITY = EDGE_FALLING" in the MHS file [20].

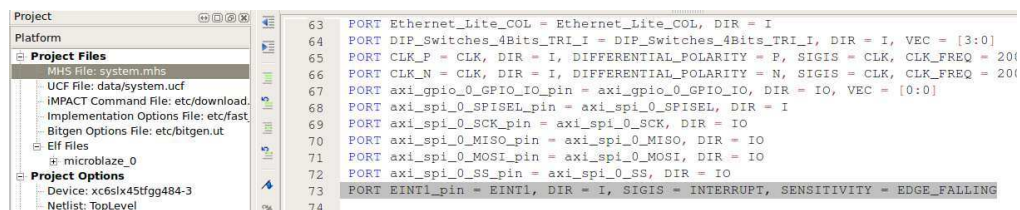


Figure 3.17. IRQ Pin

3. We need to edit the .ucf for connecting IP pin to the FPGA physical pin.

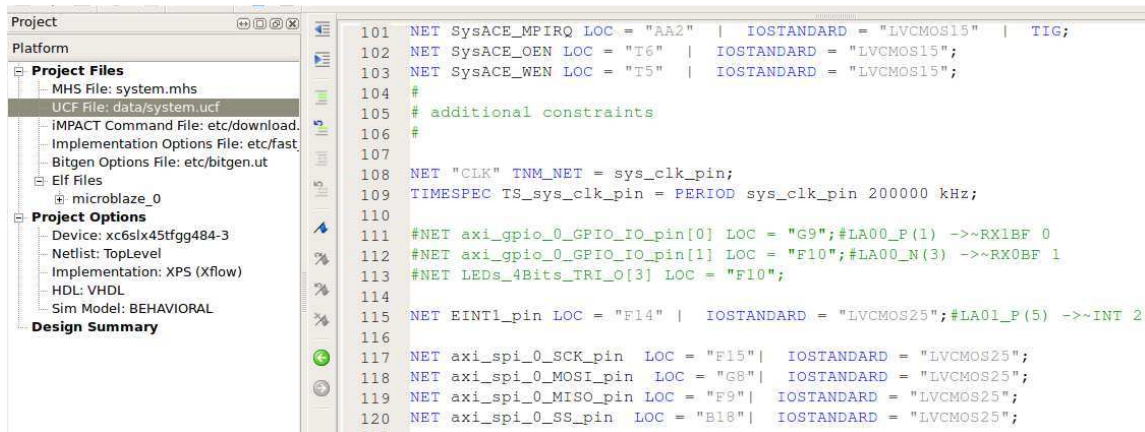


Figure 3.18. UCF

4. Click Export Design. This will export the hardware to the SDK. Choose "Export and Launch SDK" and now we have a processor that can be loaded into the FPGA.

CHAPTER IV

EMBEDDED LINUX

PetaLinux Software Development Kit (SDK)

PetaLinux is a specialized Linux distribution that is designed for Xilinx products, such as MicroBlaze and Zynq ARM board [21]. In Aug. 2012, Xilinx acquired PetaLinux's founding company, PetaLogix, [22] to reinforce their embedded Linux product support capability.

PetaLinux SDK provides not only a customized Linux distribution but also tools to simplify design steps. The following sections will talk about essential components for PetaLinux SDK and embedded Linux concepts.

First Stage Bootloader (FS-Boot)

When MicroBlaze system starts running, it executes the code from memory 0x00000000, which usually is in MicroBlaze BRAM [23]. PetaLinux SDK provides the First Stage Bootloader (FS-Boot) which is small enough to be stored in the BRAM. Once we turn on the board, FS-Boot will lead the primary boot-loader and Linux image to run the system.

Device Tree and Device Tree Generator

To define hardware parameters, Xilinx SDK generates "xparameters.h" for the user. This header file can be used to define peripheral address for C program. However, it is inconvenient to use this for Linux kernel because we need to recompile the kernel once we change the hardware [24]. Device tree is a good way to maintain the kernel. It inherits

the property of IBM Open Firmware [25]. Table 4.1 shows how to define a node in the device tree.

Table 4.1. Device Tree Code

```
MCP2515_kernel_module_instance: MCP2515_kernel_module@40600000 {
    compatible = "petalogix,MCP2515_fifo";
    reg = <0x40600000 0x10000>;
    interrupt-parent = <&microblaze_0_intc>;
    interrupts = < 1 2 >;
};
```

The first line defines the instance name and address. The “compatible” has to match the kernel module driver. Moreover, the “reg” means the address and the size of this node. Last but not least, “Interrupt-parent” points to the address of interrupt controller, and the interrupt priority “1” and “2” means falling edge.

Platform Device and Platform Driver

PetaLinux SDK provides device module template for devices and it is based on the platform driver. The driver drives a platform device on a pseudo-bus which simplifies the complexity of the interface [26]. The template includes the function of registering the device into the kernel. Beginners can easily create a working driver by using the template. Table 4.2 shows a structure for the platform driver that is used to pass the driver information kernel.

Table 4.2. Platform Driver Structure

```
static struct platform_driver MCP2515_driver = {

    .driver = {

        .name = DRIVER_NAME,

        .owner = THIS_MODULE,

        .of_match_table = MCP2515_of_match,

    },

    .probe = MCP2515_probe,

    .remove = __devexit_p(MCP2515_remove),

};
```

The “.name” and the “.owner” are predefined names. “.of_match_table” needs to match the device tree’s “compatible” part so that the kernel can know the driver responds to the node. In addition, “.probe” includes essential functions such as requiring memory region for the device and registering irq. In contract, “.remove” will release the acquired resources back to the kernel.

Linux Module

For Linux, we need to make a module to mount a driver into the kernel [27].

Table 4.3 shows the module initial and module exit that function for the platform driver.

Table 4.3. Linux Module

```
static int __init MCP2515_init(void)

{
```

Table 4.4.-continued

```
printk(KERN_ALERT "Hello module world.\n");  
  
return platform_driver_register(&MCP2515_driver);  
  
}  
  
static void __exit MCP2515_exit(void)  
{  
  
    platform_driver_unregister(&MCP2515_driver);  
  
    printk(KERN_ALERT "Goodbye module world.\n");  
  
}  
  
module_init(MCP2515_init);  
  
module_exit(MCP2515_exit);
```

When we operate a Linux system, we can use “modprobe” and “rmmod” to register a module. Type "modprobe + name" and we will executes the “module_init” function. On the other hand, “rmmod+ name” trigger the “module_exit”.

Another important thing is that the module is running in the kernel-space instead of user-space which means we have the accessibility to the hardware. Also, we need to run the interrupt service routine in the kernel module, not a user application. Finally, it is easy for the user to kill a frozen application, but the kernel driver can cause kernel panic which causes a system break-down.

CHAPTER V

DESIGN AND IMPLEMENTATION

CAN Testing Bench

Three CAN nodes are setup for the testing bench. For CAN_01 node, it can broadcast message with different IDs to change the LCD color on CAN_02 and CAN_03 every 187ms for 12 continuously burst messages or every 396ms for 12 burst with around 18ms delay.

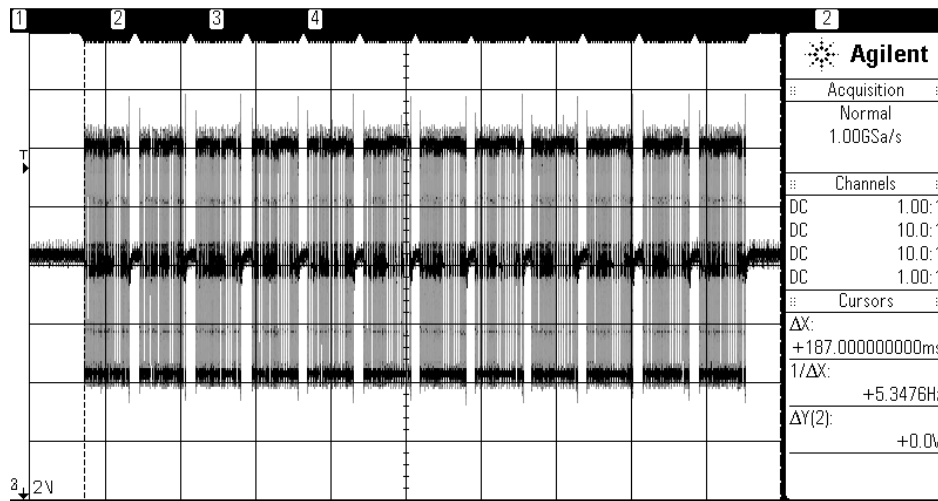


Figure 5.1. Continuous Burst

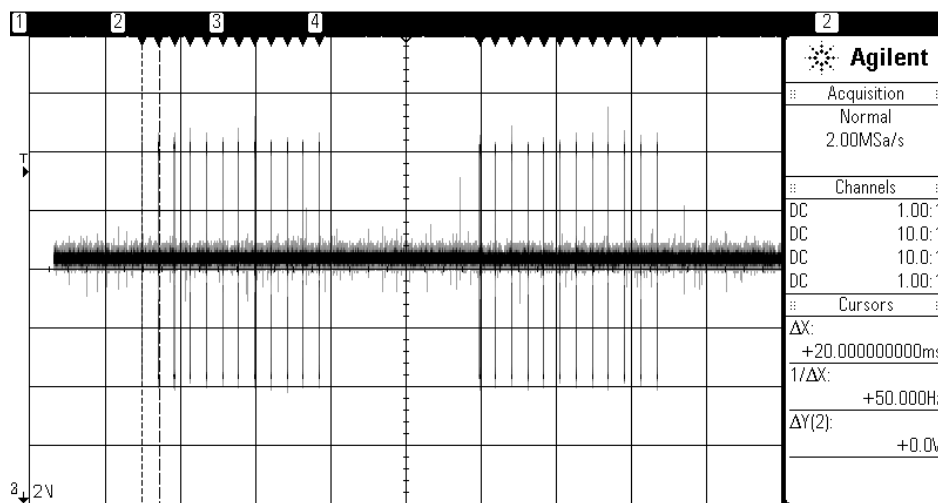


Figure 5.2. Discontinuous burst

Appendix A, B, and C are the codes for the three nodes. It can be run on Ti Code Composer Studio™ with Stellaris ware. The broadcast message shows in Table 5.1.

Table 5.1. Messages

ID	Message
610	R:Tung
620	G:Hsun
611	B:Tsou
621	B:WMU
610	G:2013
620	R:M6
610	B:M7
620	B:M8
611	R:M9
621	R:M10
610	G:M11
610	R:M12

Xilinx SDK-C Program

To create a new C program, we can click “File-> New-> Xilinx C program ->empty_application “in SDK. This creates one file folder and one board support package. The board support package provides drivers for Xilinx IP core. However, it is not easy to track and debug by using their driver. Based on the each IP documents, we create their header file and write the driver for it. Appendix D to I are required to run the MCP2515. They are SPI drivers, MCP2515 driver, and header files.

For the driver, we create a circular queue to prevent data loss. When the interrupt happens and device reads the data, write index will plus one. When the system exits the interrupt, it will read the buffer data and “read index” will plus one. It stops printing message until “write index” equal to “read index”, which means queue is empty.

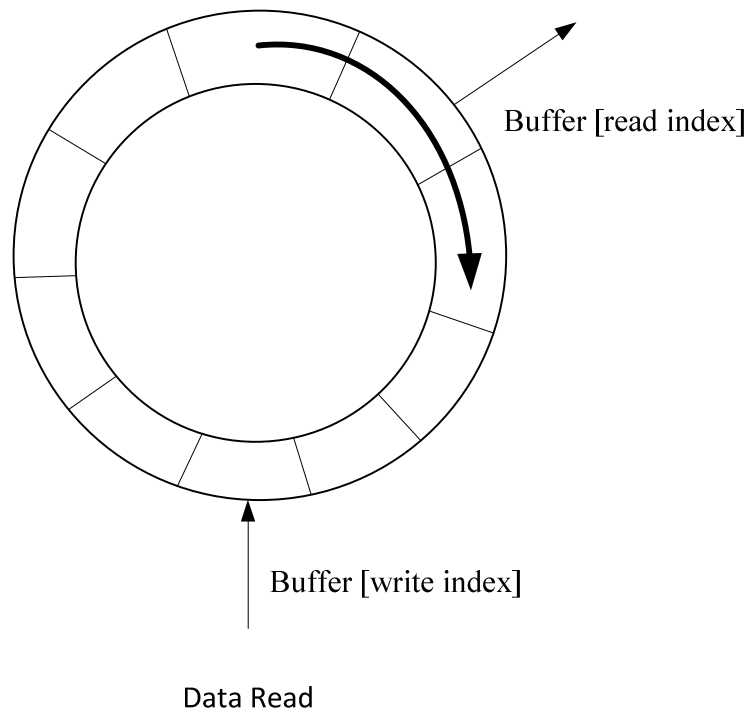


Figure 5.3. 64 Frames Circular Queue

Figure 5.4 shows the C program flow. And Figure 5.5 shows the initiation for the MCP2515.

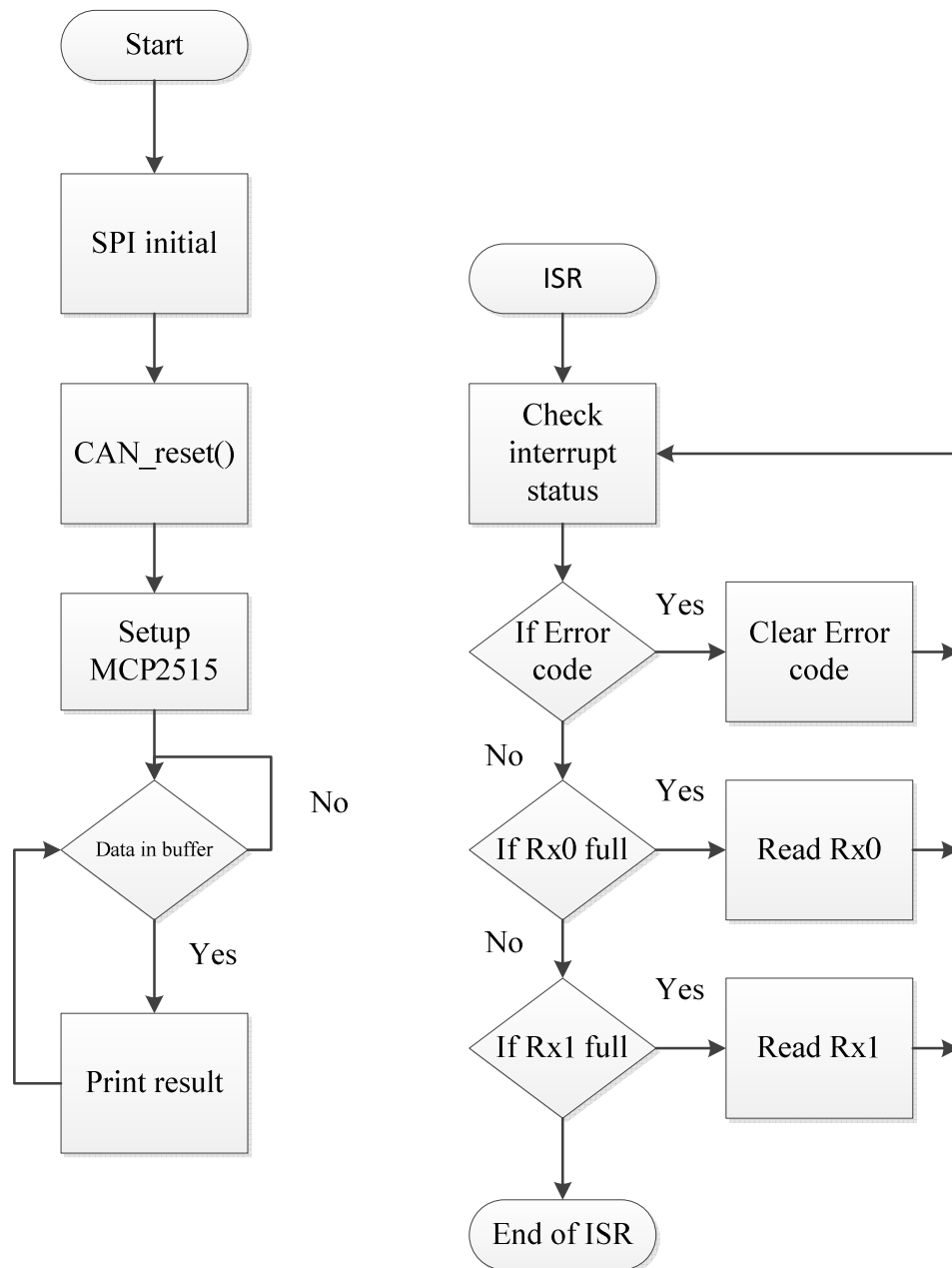


Figure 5.4. Software Flow

Figure 5.5 shows the reset of MCP2515 and configure parts with SPI. Figure 5.6 shows how SPI reads data from MCP2515.

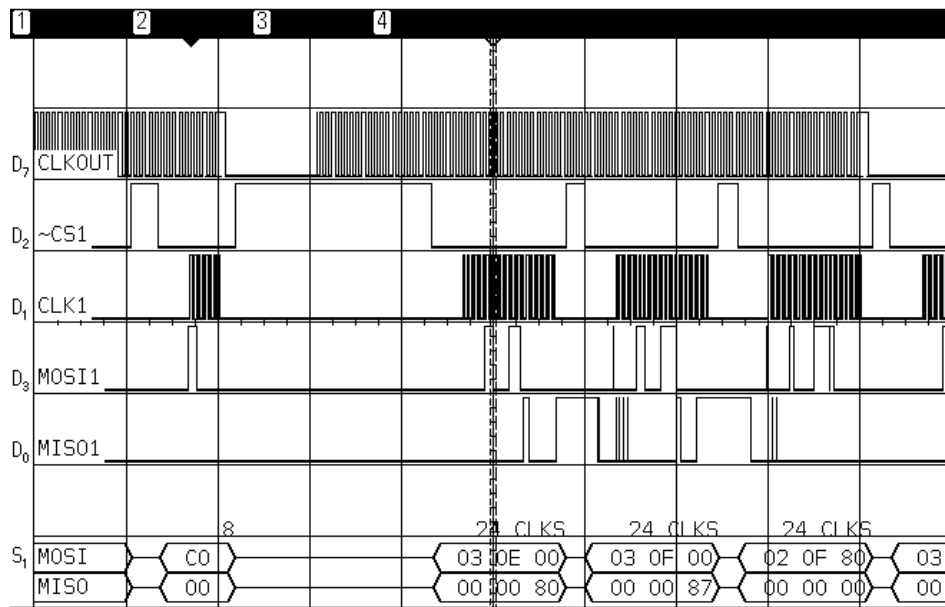


Figure 5.5. Reset and Initial Setup for MCP2515



Figure 5.6. Read Rx Buffer on MCP2515

Figure 5.7 is the result for stand-alone C program and verifies it by using Tektronix MSO 4034.

Stand alone C program

Tektronix MSO4034

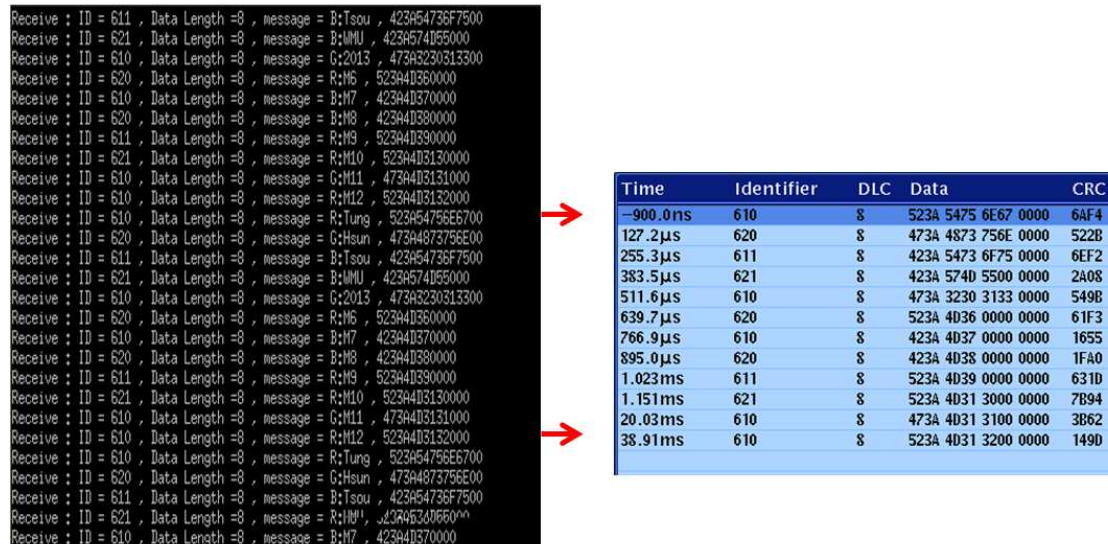


Figure 5.7. Result

PetaLinux Module

PetaLinux script shows how to use PetaLinux SDK to configure a kernel image [15].

After we compile the kernel image, we can use terminal command to create System ACE and place it in to the CF card. The file path might vary depending on where you place the file.

```
xmd -tcl genace.tcl -jprog -target mdm -hw /usr2/petalinux-v12.12-final-
full/hardware/user-
platforms/SP605_0312_AXI/workspace/SP605_0312_AXI_hw_platform/download.bit -
elf /tftpboot/image.elf -board SP605 -ace Ace.ace
```

Figure 5.8 shows the power on screen for the PetaLinux. The default account and password is “root” and “root”. After login into the Linux we can use “modprobe MCP2515_fifo” to load the module that we just wrote and remove it by using “rmmod MCP2515_fifo”.

```
Mounting devpts:
Mounting all filesystem
Setting hostname:
Bringing up network interfaces:
ip: cannot find device "lo"
ip: socket(AF_INET6): Address family not supported by protocol
ip: socket(AF_INET6): Address family not supported by protocol
Starting portmap:
Starting uWeb server:

Welcome to
PetaLinux
on CAN_0412

petalinux <error> (httpd): Failed to create server socket: Address family not supported by protocol

CAN_0412 login: █
```

Figure 5.8. Petalinux Power On

Figure 5.9 shows the module execution.

```
Receive : ID = 610 , Data Length =8 , message = R:Tung , 523a54756e6700
Receive : ID = 620 , Data Length =8 , message = G:Hsun , 473a4873756e00
Receive : ID = 611 , Data Length =8 , message = B:Tsou , 423a54736f7500
Receive : ID = 621 , Data Length =8 , message = B:WMU , 423a574d55000
Receive : ID = 610 , Data Length =8 , message = G:2013 , 473a3230313300
Receive : ID = 620 , Data Length =8 , message = R:M6 , 523a4d360000
Receive : ID = 610 , Data Length =8 , message = B:M7 , 423a4d370000
Receive : ID = 620 , Data Length =8 , message = B:M8 , 423a4d380000
Receive : ID = 611 , Data Length =8 , message = R:M9 , 523a4d390000
Receive : ID = 621 , Data Length =8 , message = R:M10 , 523a4d3130000
Receive : ID = 610 , Data Length =8 , message = G:M11 , 473a4d3131000
Receive : ID = 610 , Data Length =8 , message = R:M12 , 523a4d3132000
Receive : ID = 610 , Data Length =8 , message = R:Tung , 523a54756e6700
Receive : ID = 620 , Data Length =8 , message = G:Hsun , 473a4873756e00
Receive : ID = 611 , Data Length =8 , message = B:Tsou , 423a54736f7500
Receive : ID = 621 , Data Length =8 , message = B:WMU , 423a574d55000
Receive : ID = 610 , Data Length =8 , message = G:2013 , 473a3230313300
Receive : ID = 620 , Data Length =8 , message = R:M6 , 523a4d360000
Receive : ID = 610 , Data Length =8 , message = B:M7 , 423a4d370000
Receive : ID = 620 , Data Length =8 , message = B:M8 , 423a4d380000

~ # rmmod MCP2515_fifo
Goodbye module world.
~ #
```

Figure 5.9. Linux Module

CHAPTER VI

CONCLUSION

This thesis completes a MicroBlaze-based CAN bus analyzer and implements the code in to PetaLinux. In the experiment, we finish a system design flow and run the system on the development board.

The research uses Xilinx XPS to create MicroBlaze system, and uses Xilinx SDK to test C code and test PetaLinux driver. Overall, the thesis runs all Xilinx embedded Linux tools. In the thesis, the accomplishments are as follows.

1. Implement CAN Node in a different platform.
2. Run successfully Linux on the FPGA board and control the external protocol controller.
3. Figure out and solve the problems of running Linux on FPGA.
4. Verify the Xilinx design tool flow.

Table 6.1 is the comparison between stand-alone C and Linux driver code. As a result, a stand-alone C code can run faster than the Linux driver with running operation system.

Table 6.1. Comparison

	Stand-alone C	Linux driver
CAN accepted speed	1Mbit/s	1Mbit/s
Message interval	15 μ sec	19 m sec

More studies can be done in the future that relate to my present research, such as embedded Linux system with Ethernet communication, MicroBlaze with customized peripheral and driver, and socketCAN applications. Details are as follows:

1. Embedded Linux system with Ethernet communication: The MicroBlaze and Linux could control Ethernet IP to reduce the lower level Ethernet protocol control.
2. MicroBlaze with customized peripheral: we could write our own hardware by using VHDL or Verilog to connect to AXI bus and write a driver for it.
3. SocketCAN application: socketCAN implements CAN driver to a network driver in Linux. Raspberry PI community provides working module and application for the SocketCAN. It would be interesting to implement CAN in a higher level communication field.

REFERENCES

- [1] "CAN in Automation," [Online]. Available: <http://www.can-cia.de/index.php?id=161>. [Accessed 1 July 2013].
- [2] "ObdDiag.net," [Online]. Available: <http://www.obddiag.net/adapter.html>. [Accessed 1 July 2013].
- [3] B. J. Bazuin, "Dr. Bradley J. Bazuin," [Online]. Available: http://homepages.wmich.edu/~bazuinb/Research/SS_Electrical.pdf. [Accessed 1 July 2013].
- [4] L. García, G. Alejandra, H. Moreno and G. Jaquenod, "Remote Logic Analyzer Implemented on FPGA," in *Argentine School of Micro-Nanoelectronics, Technology and Applications EAMTA 2008*, 2008.
- [5] H. Kashif, G. Bahig and S. Hammad, "CAN Bus Analyzer and Emulator," in *Design and Test Workshop (IDT)*, Alfaisal Hall, Saudi Arabia, 2009.
- [6] M. Mostafa, M. Shalan and S. Hammad, "FPGA-Based Low-level CAN Protocol Testing," in *The 6th International Workshop on System on Chip for Real Time Applications*, Cairo, Egypt, 2006.
- [7] Bosch, "Bosch CAN specification," [Online]. Available: <http://esd.cs.ucr.edu/webres/can20.pdf>. [Accessed 1 July 2013].

- [8] Xilinx, "UG526 SP605 Hardware User Guide," 24 Sept 2012. [Online]. Available:
http://www.xilinx.com/support/documentation/boards_and_kits/ug526.pdf.
[Accessed 1 July 2013].
- [9] Xilinx, "FMC XM105 Debug," 16 June 2011. [Online]. Available:
http://www.xilinx.com/support/documentation/boards_and_kits/ug537.pdf.
[Accessed 1 July 2013].
- [10] Xilinx, "Xilinx XTP067 - SP605 Schematics," [Online]. Available:
http://www.xilinx.com/support/documentation/boards_and_kits/xtp067_sp605_schematics.pdf. [Accessed 1 July 2013].
- [11] Microchip, "DS21801G Stand-Alone CAN Controller with SPI Interface," 2012. [Online]. Available:
<http://ww1.microchip.com/downloads/en/DeviceDoc/21801G.pdf>. [Accessed 1 July 2013].
- [12] Microchip, "DS21667D High-Speed CAN Transceiver," 2010. [Online]. Available: <http://ww1.microchip.com/downloads/en/devicedoc/21667d.pdf>.
[Accessed 1 July 2013].
- [13] Xilinx, "Xilinx DS080 System ACE CompactFlash Solution, Data Sheet," 1 Oct 2008. [Online]. Available:
http://www.xilinx.com/support/documentation/data_sheets/ds080.pdf.
[Accessed 1 July 2013].
- [14] Xilinx, "UG081 MicroBlaze Processor Reference Guide (v14.1)," 24 April

2012. [Online]. Available:
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/mb_ref_guide.pdf. [Accessed 1 July 2013].
- [15] PetaLogix, "Board Bring Up with PetaLinux SDK," 12 Dec 2012. [Online]. Available: <http://www.xilinx.com/publications/products/petalinux/petalinux-board-bringup-guide.pdf>. [Accessed 1 July 2013].
- [16] Xilinx, "DS572 LogiCORE IP XPS Interrupt Controller (v2.01a)," 29 April 2010. [Online]. Available:
http://www.xilinx.com/support/documentation/ip_documentation/xps_intc.pdf. [Accessed 1 July 2013].
- [17] E. A. L. a. S. A. Seshia, "Introduction to Embedded Systems," 2012. [Online]. Available:
http://chess.eecs.berkeley.edu/eecs149/documentation/th_ublaze_interrupts.pdf. [Accessed 01 12 2012].
- [18] P. Glover, "Using and Creating Interrupt-Based," 11 January 2005. [Online]. Available:
http://www.xilinx.com/support/documentation/application_notes/xapp778.pdf. [Accessed 1 July 2013].
- [19] ARM, "AMBA®4 AXI4™, AXI4-Lite™, and AXI4-Stream™ Protocol Assertions User Guide," 23 July 2012. [Online]. Available:
http://infocenter.arm.com/help/topic/com.arm.doc.dui0534b/DUI0534B_amba_4_axi4_protocol_assertions_ug.pdf. [Accessed 1 July 2013].

- [20] P. Glover, "Using and Creating Interrupt-Based," 11 Jan 2005. [Online]. Available:
http://www.xilinx.com/support/documentation/application_notes/xapp778.pdf.
[Accessed 1 7 2013].
- [21] "Supported FPGA and CPU families," PetaLogix, 2009. [Online]. Available:
<http://www.petalogix.com/about/supported-fpga-and-cpu-families>. [Accessed
1 July 2013].
- [22] S. JOSE, "Xilinx Acquires Embedded Linux Solutions Provider PetaLogix,"
Xilinx, 28 Aug 2012. [Online]. Available: <http://press.xilinx.com/2012-08-28-Xilinx-Acquires-Embedded-Linux-Solutions-Provider-PetaLogix>. [Accessed 1
July 2013].
- [23] PetaLogix, "PetaLinux Bootloader Solutions," PetaLogix, [Online]. Available:
<http://www.petalogix.com/resources/documentation/petalinux/userguide/Bootloaders>. [Accessed 1 7 2013].
- [24] G. Likely and J. Boyer, "A Symphony of Flavours: Using the device tree to
describe embedded," 23 July 2008. [Online]. Available:
<https://www.kernel.org/doc/ols/2008/ols2008v2-pages-27-38.pdf>. [Accessed 1
July 2013].
- [25] D. Gibson and B. Herrenschmidt, "Device trees every where," 13 Feb 2006.
[Online]. Available: <http://ozlabs.org/~dgibson/papers/dtc-paper.pdf>.
[Accessed 1 July 2013].
- [26] "Platform Devices and Drivers," [Online]. Available:

<https://www.kernel.org/doc/Documentation/driver-model/platform.txt>.

[Accessed 1 July 2013].

- [27] J. Corbet, A. Rubini and G. Kroah, Linux Device Drivers, Third Edition, O'Reilly, 2005.

APPENDIX

A. CAN_01.C

```

/*****
* File: CAN_01.c
* Created on: Feb 22, 2013
* Author: Tung-Hsun Tsou(Tony)
* Email: gotolafa@gmail.com
* Version: v1.0 Feb 22 First release
* Ref: 1.Datasheet
*   Datasheet-LM4F120H5QR.pdf
*   2.Getting Started with the Stellaris LaunchPad...
*   http://software-dl.ti.com/trainingTTO/trainingTTO_public_sw
*   /GSW-Stellaris-LaunchPad/StellarisLaunchPadWorkbook.pdf
*   3.StellarisR Peripheral Driver Library user's guide
*   SW-DRL-UG-9453.pdf
*   4.Build in example
*   StellarisWare/examples/can
*   5.CCSv5_Tips_&_Tricks
*   http://downloads.ti.com/dsps/dsps_public_sw/sdo_ccstudio/presentations
*   /CCSv5_Tips_&_Tricks.pdf
* Copyright(c)Tung-Hsun Tsou 2013 All rights reserved.
*****/
/*****
* Don't forget include StellarisWare in CCS.
* 1. Install StellarisWare.
* 2. In CCS, Project -> Properties -> Build -> Include Options -> Add dir...
*   -> C:\StellarisWare\
* 3. Also the linker file
*   -> C:\StellarisWare\driverlib\ccs-cm4f\Debug\driverlib-cm4f.lib
* 4.In Code Composer 5.2 and Earlier:
*   In Code Composer, in the Project Explorer, right-click on your project and
*   select Properties.
*   On the left, click Build and then the Steps tab. Paste the following
*   commands
*   into the Postbuild steps Command box:
*   "${CCS_INSTALL_ROOT}/utils/tiobj2bin/tiobj2bin"
*   "${BuildArtifactFileName}" "${BuildArtifactFileName}.bin"
*   "${CG_TOOL_ROOT}/bin/ofd470" "${CG_TOOL_ROOT}/bin/hex470"
*   "${CCS_INSTALL_ROOT}/utils/tiobj2bin/mkhex4bin"
*****/
#define PART_LM4F120H5QR
#include "inc/lm4f120h5qr.h"
#include "inc/hw_types.h" /* Macros for hardware access */
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h" /* system clock */
#include "driverlib/pin_map.h"

```

```

#include "driverlib/gpio.h"
#include "driverlib/can.h"
// #include "CAN_driver.h"
// #include "driverlib/interrupt.h"
#include "inc/hw_ints.h"
#define CANLinux
/*****
*
*****
*/
int test=0xFF;
int LED = 0x08;
tCANMsgObject MsgObjectRx;
tCANMsgObject MsgObjectTx;
// unsigned char BufferIn[8];
// unsigned char BufferOut[8];
unsigned char CANErrFlag,CANRxFlag1;
unsigned char RxMsgCount;
unsigned char CANMsgCount;
unsigned char TestCount;
unsigned int Delay_time1=500000; //12.5ns(clock)*3=37.5ns
unsigned int Delay_time2=50000; //12.5ns(clock)*3=37.5ns
#define LED_RED 0x2
#define LED_BLUE 0x4
#define LED_GREEN 0x8
/*****
*
*****
*/
void main(void)
{
    /* Initial */
    //IntMasterDisable(); //turn off all interrupt
    tCANBitClkParms CANBitClk = {4,3,2,1};
    unsigned char MsgData1[8] = {"R:Tung"};
    unsigned char MsgData2[8] = {"G:Hsun"};
    unsigned char MsgData3[8] = {"B:Tsou"};
    unsigned char MsgData4[8] = {"B:WMU"};
    unsigned char MsgData5[8] = {"G:2013"};
    unsigned char MsgData6[8] = {"R:M6"};
    unsigned char MsgData7[8] = {"B:M7"};
    unsigned char MsgData8[8] = {"B:M8"};
    unsigned char MsgData9[8] = {"R:M9"};
    unsigned char MsgData10[8] = {"R:M10"};
    unsigned char MsgData11[8] = {"G:M11"};
    unsigned char MsgData12[8] = {"R:M12"};

    SysCtlClockSet(SYSCTL_SYSDIV_25|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|
        SYSCTL_XTAL_16MHZ); // PLL(400MHz)/2.5/2=80Mhz (Ref.3 P350)

    /* CAN Initial */
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    GPIOPinConfigure(GPIO_PB5_CAN0TX);
    GPIOPinConfigure(GPIO_PB4_CAN0RX);
    GPIOPinTypeCAN( GPIO_PORTB_BASE,GPIO_PIN_4|GPIO_PIN_5); //CAN0Tx CAN0Rx
    SysCtlPeripheralEnable(SYSCTL_PERIPH_CAN0); // enable the peripheral

    CANInit(CAN0_BASE); // (Ref.1 P999)

```

```

//CAN0_CTL_R |= (CAN_CTL_CCE | CAN_CTL_TEST);
CANBitRateSet(CAN0_BASE, 8000000, 1000000); //(Ref.3 P56)
CANBitTimingSet(CAN0_BASE, &CANBitClk);
//CAN0_TST_R = CAN_TST_LBACK ;

//CANIntEnable(CAN0_BASE, CAN_INT_MASTER | CAN_INT_ERROR | CAN_INT_STATUS);
//IntEnable(INT_CAN0);
CANEnable(CAN0_BASE);

/* Configure and start transmit of message object */
MsgObjectTx.ulMsgID = 0x610;
MsgObjectTx.ulMsgIDMask = 0x00;
MsgObjectTx.ulFlags = MSG_OBJ_TX_INT_ENABLE;
MsgObjectTx.ulMsgLen = sizeof(MsgData1);
MsgObjectTx.pucMsgData = MsgData1;

/* Configure and start receive of message object */
//MsgObjectRx.ulMsgID = 0x611;           // CAN msg ID
//MsgObjectRx.ulMsgIDMask = 0x5F17;     // mask, all 20 bits must match
//MsgObjectRx.ulFlags = MSG_OBJ_RX_INT_ENABLE; //| MSG_OBJ_USE_ID_FILTER;
//MsgObjectRx.ulMsgLen = 8;             // allow up to 8 bytes

/* GPIO Initial */
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1);
//IntMasterEnable(); //enable interrupt

while (1)
{

    MsgObjectTx.ulMsgID = 0x610;
    MsgObjectTx.pucMsgData = MsgData1; //RED
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
        SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x620;
    MsgObjectTx.pucMsgData = MsgData2; //GREEN
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
        SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x611;
    MsgObjectTx.pucMsgData = MsgData3; //BLUE
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
        SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x621;
    MsgObjectTx.pucMsgData = MsgData4; //BLUE
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux

```

```

    SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x610;
    MsgObjectTx.pucMsgData = MsgData5;//GREEN
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
    SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x620;//RED
    MsgObjectTx.pucMsgData = MsgData6;
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
    SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x610;
    MsgObjectTx.pucMsgData = MsgData7; //RED
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
    SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x620;
    MsgObjectTx.pucMsgData = MsgData8;//GREEN
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
    SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x611;
    MsgObjectTx.pucMsgData = MsgData9;//BLUE
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
    SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x621;
    MsgObjectTx.pucMsgData = MsgData10;//BLUE
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
    SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x610;
    MsgObjectTx.pucMsgData = MsgData11; //RED
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
    SysCtlDelay(Delay_time1);

    MsgObjectTx.ulMsgID = 0x610;
    MsgObjectTx.pucMsgData = MsgData12; //RED
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
    SysCtlDelay(Delay_time1);
    if (LED==0x0E)
        LED = 0x00;

```



```
else
    LED = 0x0E;
    GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1, LED);
}
```

B. CAN_02.C

```

/*****
* File: CAN_01.c
* Created on: Feb 22, 2013
* Author: Tung-Hsun Tsou(Tony)
* Email: gotolafa@gmail.com
* Version: v1.0 Feb 22 First release
* Ref: 1.Datasheet
*   Datasheet-LM4F120H5QR.pdf
*   2.Getting Started with the Stellaris LaunchPad...
*   http://software-dl.ti.com/trainingTTO/trainingTTO_public_sw
*   /GSW-Stellaris-LaunchPad/StellarisLaunchPadWorkbook.pdf
*   3.StellarisR Peripheral Driver Library user's guide
*   SW-DRL-UG-9453.pdf
*   4.Build in example
*   StellarisWare/examples/can
*   5.CCSv5_Tips_&_Tricks
*   http://downloads.ti.com/dsps/dsps_public_sw/sdo_ccstudio/presentations
*   /CCSv5_Tips_&_Tricks.pdf
* Copyright(c)Tung-Hsun Tsou 2013 All rights reserved.
*****/
/*****
* Don't forget include StellarisWare in CCS.
* 1. Install StellarisWare.
* 2. In CCS, Project -> Properties -> Build -> Include Options -> Add dir...
*   -> C:\StellarisWare\
* 3. Also the linker file
*   -> C:\StellarisWare\driverlib\ccs-cm4f\Debug\driverlib-cm4f.lib
* 4.In Code Composer 5.2 and Earlier:
*   In Code Composer, in the Project Explorer, right-click on your project and
*   select Properties.
*   On the left, click Build and then the Steps tab. Paste the following
*   commands
*   into the Postbuild steps Command box:
*   "${CCS_INSTALL_ROOT}/utils/tiobj2bin/tiobj2bin"
*   "${BuildArtifactFileName}" "${BuildArtifactFileName}.bin"
*   "${CG_TOOL_ROOT}/bin/ofd470" "${CG_TOOL_ROOT}/bin/hex470"
*   "${CCS_INSTALL_ROOT}/utils/tiobj2bin/mkhex4bin"
*****/
#define PART_LM4F120H5QR
#include "inc/lm4f120h5qr.h"
#include "inc/hw_types.h" /* Macros for hardware access */
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h" /* system clock */
#include "driverlib/pin_map.h"
#include "driverlib/gpio.h"
#include "driverlib/can.h"
// #include "CAN_driver.h"
// #include "driverlib/interrupt.h"
#include "inc/hw_ints.h"
#define CANLinux
/*****
*
*****/
```

```

int test=0xFF;
int LED = 0x08;
tCANMsgObject MsgObjectRx;
tCANMsgObject MsgObjectTx;
//unsigned char BufferIn[8];
//unsigned char BufferOut[8];
unsigned char CANErrFlag,CANRxFlag1;
unsigned char RxMsgCount;
unsigned char CANMsgCount;
unsigned char TestCount;
unsigned int Delay_time1=500000; //12.5ns(clock)*3=37.5ns
unsigned int Delay_time2=50000; //12.5ns(clock)*3=37.5ns
#define LED_RED 0x2
#define LED_BLUE 0x4
#define LED_GREEN 0x8
/*****
*
*****/
void main(void)
{
    /* Initial */
    //IntMasterDisable(); //turn off all interrupt
    tCANBitClkParms CANBitClk = {4,3,2,1};
    unsigned char MsgData1[8] = {"R:Tung"};
    unsigned char MsgData2[8] = {"G:Hsun"};
    unsigned char MsgData3[8] = {"B:Tsou"};
    unsigned char MsgData4[8] = {"B:WMU"};
    unsigned char MsgData5[8] = {"G:2013"};
    unsigned char MsgData6[8] = {"R:M6"};
    unsigned char MsgData7[8] = {"B:M7"};
    unsigned char MsgData8[8] = {"B:M8"};
    unsigned char MsgData9[8] = {"R:M9"};
    unsigned char MsgData10[8] = {"R:M10"};
    unsigned char MsgData11[8] = {"G:M11"};
    unsigned char MsgData12[8] = {"R:M12"};

    SysCtlClockSet(SYSCTL_SYSDIV_25|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|
        SYSCTL_XTAL_16MHZ); // PLL(400MHz)/2.5/2=80Mhz (Ref.3 P350)

    /* CAN Initial */
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    GPIOPinConfigure(GPIO_PB5_CAN0TX);
    GPIOPinConfigure(GPIO_PB4_CAN0RX);
    GPIOPinTypeCAN( GPIO_PORTB_BASE,GPIO_PIN_4|GPIO_PIN_5); //CAN0Tx CAN0Rx
    SysCtlPeripheralEnable(SYSCTL_PERIPH_CAN0); // enable the peripheral

    CANInit(CAN0_BASE); // (Ref.1 P999)
    //CAN0_CTL_R |= (CAN_CTL_CCE | CAN_CTL_TEST);
    CANBitRateSet(CAN0_BASE, 8000000, 1000000); //(Ref.3 P56)
    CANBitTimingSet(CAN0_BASE, &CANBitClk);
    //CAN0_TST_R = CAN_TST_LBACK ;

    //CANIntEnable(CAN0_BASE, CAN_INT_MASTER | CAN_INT_ERROR | CAN_INT_STATUS);
    //IntEnable(INT_CAN0);
    CANEnable(CAN0_BASE);

```

```

/* Configure and start transmit of message object */
MsgObjectTx.ulMsgID = 0x610;
MsgObjectTx.ulMsgIDMask = 0x00;
MsgObjectTx.ulFlags = MSG_OBJ_TX_INT_ENABLE;
MsgObjectTx.ulMsgLen = sizeof(MsgData1);
MsgObjectTx.pucMsgData = MsgData1;

/* Configure and start receive of message object */
//MsgObjectRx.ulMsgID = 0x611;           // CAN msg ID
//MsgObjectRx.ulMsgIDMask = 0x5F17;      // mask, all 20 bits must match
//MsgObjectRx.ulFlags = MSG_OBJ_RX_INT_ENABLE; //| MSG_OBJ_USE_ID_FILTER;
//MsgObjectRx.ulMsgLen = 8;              // allow up to 8 bytes

/* GPIO Initial */
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1);
//IntMasterEnable(); //enable interrupt

while (1)
{
    MsgObjectTx.ulMsgID = 0x610;
    MsgObjectTx.pucMsgData = MsgData1; //RED
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
    SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x620;
    MsgObjectTx.pucMsgData = MsgData2; //GREEN
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
    SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x611;
    MsgObjectTx.pucMsgData = MsgData3; //BLUE
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
    SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x621;
    MsgObjectTx.pucMsgData = MsgData4; //BLUE
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
    SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x610;
    MsgObjectTx.pucMsgData = MsgData5; //GREEN
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
    SysCtlDelay(Delay_time2);
#endif
}

```

```

    MsgObjectTx.ulMsgID = 0x620;//RED
    MsgObjectTx.pucMsgData = MsgData6;
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
    SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x610;
    MsgObjectTx.pucMsgData = MsgData7; //RED
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
    SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x620;
    MsgObjectTx.pucMsgData = MsgData8;//GREEN
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
    SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x611;
    MsgObjectTx.pucMsgData = MsgData9;//BLUE
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
    SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x621;
    MsgObjectTx.pucMsgData = MsgData10;//BLUE
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
#ifdef CANLinux
    SysCtlDelay(Delay_time2);
#endif

    MsgObjectTx.ulMsgID = 0x610;
    MsgObjectTx.pucMsgData = MsgData11; //RED
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
    SysCtlDelay(Delay_time1);

    MsgObjectTx.ulMsgID = 0x610;
    MsgObjectTx.pucMsgData = MsgData12; //RED
    CANMessageSet(CAN0_BASE, 1, &MsgObjectTx, MSG_OBJ_TYPE_TX);
    SysCtlDelay(Delay_time1);
    if (LED==0x0E)
        LED = 0x00;
    else
        LED = 0x0E;
    GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1, LED);

}
}

```

C. CAN_03.C

```

/*****
* File: CAN_03.c
* Created on: Feb 22, 2013
* Author: Tung-Hsun Tsou(Tony)
* Email: gotolafa@gmail.com
* Version: v1.0 Feb 22 First release
* Ref: 1.Datasheet
*   Datasheet-LM4F120H5QR.pdf
*   2.Getting Started with the Stellaris LaunchPad...
*   http://software-dl.ti.com/trainingTTO/trainingTTO_public_sw
*   /GSW-Stellaris-LaunchPad/StellarisLaunchPadWorkbook.pdf
*   3.StellarisR Peripheral Driver Library user's guide
*   SW-DRL-UG-9453.pdf
*   4.Build in example
*   StellarisWare/examples/can
*   5.CCSv5_Tips_&_Tricks
*   http://downloads.ti.com/dsps/dsps_public_sw/sdo_ccstudio/presentations
*   /CCSv5_Tips_&_Tricks.pdf
* Copyright(c)Tung-Hsun Tsou 2013 All rights reserved.
*****/
/*****
* Don't forget include StellarisWare in CCS.
* 1. Install StellarisWare.
* 2. In CCS, Project -> Properties -> Build -> Include Options -> Add dir...
*   -> C:/StellarisWare/
* 3. Also the linker file
*   -> C:/StellarisWare/driverlib/ccs-cm4f/Debug/driverlib-cm4f.lib
*****/
#define PART_LM4F120H5QR
#include "inc/lm4f120h5qr.h"
#include "inc/hw_types.h" /* Macros for hardware access */
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h" /* system clock */
#include "driverlib/pin_map.h"
#include "driverlib/gpio.h"
#include "driverlib/can.h"
// #include "CAN_driver.h"
#include "driverlib/interrupt.h"
#include "inc/hw_ints.h"
/*****
*
*****/
int test=0xFF;
int LED = 0x08;
tCANMsgObject MsgObjectRx;
tCANMsgObject MsgObjectTx;
unsigned char BufferIn[8];
// unsigned char BufferOut[8];
unsigned char CANErrFlag,CANRxFlag1,CANRxFlag2;
unsigned char RxMsgCount;
unsigned char TxMsgCount;
unsigned char TestCount_main1,TestCount_main2,TestCount_main_interrupt;
unsigned long CAN_Status;

```

```

#define LED_RED 0x2
#define LED_BLUE 0x4
#define LED_GREEN 0x8
#define Meg_RED 0x52
#define Meg_BLUE 0x47
#define Meg_GREEN 0x42
/*****/
void CANIntHandler(void)
{

    CAN_Status = CANIntStatus(CAN0_BASE, CAN_INT_STS_CAUSE);
    if ( CAN_Status == 1)
    {
        CANIntClear(CAN0_BASE, 1);
        //RxMsgCount++;
        CANRxFlag1 = 1;
    }
    else if (CAN_Status == 2)
    {
        CANIntClear(CAN0_BASE, 2);
        //RxMsgCount++;
        CANRxFlag2 = 1;
    }
    else
    {
        LED = LED_RED | LED_BLUE | LED_GREEN;
    }
}

void main(void)
{
    IntMasterDisable(); //turn off all interrupt
    /* Initial */
    tCANBitClkParms CANBitClk = {4,3,2,1};
    SysCtlClockSet(SYSCTL_SYSDIV_25|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|
        SYSCTL_XTAL_16MHZ); /* PLL(400MHz)/2.5/2=80Mhz (Ref.3 P350) */

    /* CAN Initial */
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    GPIOPinConfigure(GPIO_PB5_CAN0TX);
    GPIOPinConfigure(GPIO_PB4_CAN0RX);
    GPIOPinTypeCAN( GPIO_PORTB_BASE,GPIO_PIN_4|GPIO_PIN_5); /*CAN0Tx CAN0Rx*/
    SysCtlPeripheralEnable(SYSCTL_PERIPH_CAN0); /* enable the peripheral */

    CANInit(CAN0_BASE); /* (Ref.1 P999) */

    CANBitRateSet(CAN0_BASE, 8000000, 1000000); /*(Ref.3 P56)*/
    CANBitTimingSet(CAN0_BASE, &CANBitClk);

    CANIntEnable(CAN0_BASE, CAN_INT_MASTER | CAN_INT_ERROR); //CAN_INT_STATUS
    IntEnable(INT_CAN0);
    CANEnable(CAN0_BASE);

    /* Configure and start transmit of message object */
    //MsgObjectTx.ulMsgID = 0x610;
    //MsgObjectTx.ulMsgIDMask = 0x00;
    //MsgObjectTx.ulFlags = MSG_OBJ_TX_INT_ENABLE;

```

```

//MsgObjectTx.ulMsgLen = sizeof(MsgData);
//MsgObjectTx.pucMsgData = MsgData;

/* Configure and start receive of message object */
MsgObjectRx.ulMsgID = 0x620;          // CAN msg ID
MsgObjectRx.ulMsgIDMask = 0x620;     // mask, all 20 bits must match
MsgObjectRx.ulFlags = MSG_OBJ_RX_INT_ENABLE; //| MSG_OBJ_USE_ID_FILTER;
MsgObjectRx.ulMsgLen = 8;            // allow up to 8 bytes
CANMessageSet(CAN0_BASE, 1, &MsgObjectRx, MSG_OBJ_TYPE_RX);
MsgObjectRx.ulMsgID = 0x621;          // CAN msg ID
MsgObjectRx.ulMsgIDMask = 0x621;     // mask, all 20 bits must match
MsgObjectRx.ulFlags = MSG_OBJ_RX_INT_ENABLE; //| MSG_OBJ_USE_ID_FILTER;
MsgObjectRx.ulMsgLen = 8;            // allow up to 8 bytes
CANMessageSet(CAN0_BASE, 2, &MsgObjectRx, MSG_OBJ_TYPE_RX);

/* GPIO Initial */
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1);
IntMasterEnable(); //enable interrupt

while (1)
{
    if (CANRxFlag1 == 1)
    {
        MsgObjectRx.pucMsgData = BufferIn;
        CANMessageGet(CAN0_BASE, 1, &MsgObjectRx, 0);
        CANRxFlag1 = 0;
        TestCount_main1++;
    }
    else if (CANRxFlag2 == 1)
    {
        MsgObjectRx.pucMsgData = BufferIn;
        CANMessageGet(CAN0_BASE, 2, &MsgObjectRx, 0);
        CANRxFlag2 = 0;
        TestCount_main2++;
    }

    if(BufferIn[0] == Meg_RED)
        LED = LED_RED;
    else if (BufferIn[0] == Meg_GREEN)
        LED = LED_GREEN;
    else if (BufferIn[0] == Meg_BLUE)
        LED = LED_BLUE;
    else
        LED = LED_RED | LED_GREEN | LED_BLUE;
    GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1, LED);
}
}

```


D. MCP2515.H

```
/* **** */
* File: MCP2515.h
* Created on: Oct 23, 2012
* Author: Tung-Hsun Tsou(Tony)
* Email: gotolafa@gmail.com
* Version: v1.00
* Reference: Microchip DS21801G MCP2515
* www.microchip.com/downloads/en/DeviceDoc/21801G.pdf
* Copyright(c)Tung-Hsun Tsou 2012 All rights reserved.
/* **** */

#ifndef MCP2515_H_
#define MCP2515_H_

/* **** */
* Message Status
/* **** */
#define CAN_MERR      0xFF80
#define CAN_WAKE      0xFF40
#define CAN_ERROR     0xFF20
#define CAN_RTR       0xFF10
#define CAN_RECEIVE    0xFF01

/* **** */
* Message Transmission (P17)
/* **** */

/* TXBnCTRL-Transmit buffer n control register (ADDRESS:30h,40h,50h) */
#define MCP2515_TXB0CTRL 0x30
#define MCP2515_TXB1CTRL 0x40
#define MCP2515_TXB2CTRL 0x50

/* TXRTSCTRL-TXnRTS Pin control and status register */
#define MCP2515_TXRTSCTRL 0x0D

/* TXBnSIDH-Transmit buffer n standard identifier high (ADDRESS:31h,41h,51h) */
#define MCP2515_TXB0SIDH 0x31
#define MCP2515_TXB1SIDH 0x41
#define MCP2515_TXB2SIDH 0x51

/* TXBnSIDL-Transmit buffer n standard identifier low (ADDRESS:32h,42h,52h) */
#define MCP2515_TXB0SIDL 0x32
#define MCP2515_TXB1SIDL 0x42
#define MCP2515_TXB2SIDL 0x52

/* TXBnEID8-Transmit buffer n extended identifier high (ADDRESS:33h,43h,53h) */
#define MCP2515_TXB0EID8 0x33
#define MCP2515_TXB1EID8 0x43
#define MCP2515_TXB2EID8 0x53

/* TXBnEID0-Transmit buffer n extended identifier low (ADDRESS:34h,44h,54h) */
#define MCP2515_TXB0EID0 0x34
#define MCP2515_TXB1EID0 0x44
```

```

#define MCP2515_TXB2EID0 0x54

/* TXBnDLC-Transmit buffer n data length code (ADDRESS:35h,45h,55h) */
#define MCP2515_TXB0DLC 0x35
#define MCP2515_TXB1DLC 0x45
#define MCP2515_TXB2DLC 0x55

/* TXBnDm-Transmit buffer n data byte m (ADDRESS:36h-3Dh,46h-4Dh,56h-5Dh) */

#define MCP2515_TXB0D0 0x36
#define MCP2515_TXB0D1 0x37
#define MCP2515_TXB0D2 0x38
#define MCP2515_TXB0D3 0x39
#define MCP2515_TXB0D4 0x3A
#define MCP2515_TXB0D5 0x3B
#define MCP2515_TXB0D6 0x3C
#define MCP2515_TXB0D7 0x3D

#define MCP2515_TXB1D0 0x46
#define MCP2515_TXB1D1 0x47
#define MCP2515_TXB1D2 0x48
#define MCP2515_TXB1D3 0x49
#define MCP2515_TXB1D4 0x4A
#define MCP2515_TXB1D5 0x4B
#define MCP2515_TXB1D6 0x4C
#define MCP2515_TXB1D7 0x4D

#define MCP2515_TXB2D0 0x56
#define MCP2515_TXB2D1 0x57
#define MCP2515_TXB2D2 0x58
#define MCP2515_TXB2D3 0x59
#define MCP2515_TXB2D4 0x5A
#define MCP2515_TXB2D5 0x5B
#define MCP2515_TXB2D6 0x5C
#define MCP2515_TXB2D7 0x5D

/*****
* Message Reception (P23)
*****/

/* RXB0CTRL-Receive buffer 0 control (ADDRESS:60h) */
#define MCP2515_RXB0CTRL 0x60
/* RXB1CTRL-Receive buffer 1 control (ADDRESS:70h) */
#define MCP2515_RXB1CTRL 0x70

#define RXM1 0x40
#define RXM0 0x20
#define RXRTR 0x08
#define BUKT 0x04
#define BUKT1 0x02
#define FILHIT 0x01

/* BFPCTRL-RXnBF pin control and status (ADDRESS:0Ch) */
#define MCP2515_BFPCTRL 0x0C

#define B1BFS 0x20

```

```

#define B0BFS          0x10
#define B1BFE          0x08
#define B0BFE          0x04
#define B1BFM          0x02
#define B0BFM          0x01
/* RXBnSIDH-Receive buffer n standard identifier high (ADDRESS:61h,71h) */
#define MCP2515_RXB0SIDH  0x61
#define MCP2515_RXB1SIDH  0x71

/* RXBnSIDL-Receive buffer n standard identifier low (ADDRESS:62h,72h) */
#define MCP2515_RXB0SIDL  0x62
#define MCP2515_RXB1SIDL  0x72

/* RXBnEID8- Receive buffer n extended identifier high (ADDRESS:63h,73h) */
#define MCP2515_RXB0EID8  0x63
#define MCP2515_RXB1EID8  0x73

/* RXBnEID0- Receive buffer n extended identifier low (ADDRESS:64h,74h) */
#define MCP2515_RXB0EID0  0x64
#define MCP2515_RXB1EID0  0x74

/* RXBnDLC-Receive buffer n data length code (ADDRESS:65h,75h) */
#define MCP2515_RXB0DLC  0x65
#define MCP2515_RXB1DLC  0x75

#define RTR              0x40

/* RXBnDM-Receive buffer n data byte M (ADDRESS:66h-6Dh,76h-7Dh) */
#define MCP2515_RXB0D0    0x66
#define MCP2515_RXB0D1    0x67
#define MCP2515_RXB0D2    0x68
#define MCP2515_RXB0D3    0x69
#define MCP2515_RXB0D4    0x6A
#define MCP2515_RXB0D5    0x6B
#define MCP2515_RXB0D6    0x6C
#define MCP2515_RXB0D7    0x6D

#define MCP2515_RXB1D0    0x76
#define MCP2515_RXB1D1    0x77
#define MCP2515_RXB1D2    0x78
#define MCP2515_RXB1D3    0x79
#define MCP2515_RXB1D4    0x7A
#define MCP2515_RXB1D5    0x7B
#define MCP2515_RXB1D6    0x7C
#define MCP2515_RXB1D7    0x7D

/*****
* Message Acceptance Filters and Masks (P32)
*****/

/* RXFnSIDH-Filter n standard identifier high
(ADDRESS:00h,04h,08h,10h,14h,18h) */
#define MCP2515_RXF0SIDH  0x00
#define MCP2515_RXF1SIDH  0x04
#define MCP2515_RXF2SIDH  0x08
#define MCP2515_RXF3SIDH  0x10
#define MCP2515_RXF4SIDH  0x14

```

```

#define MCP2515_RXF5SIDH    0x18

/* RXFnSIDH-Filter n standard identifier low
  (ADDRESS:01h,05h,09h,11h,15h,19h) */
#define MCP2515_RXF0SIDL    0x01
#define MCP2515_RXF1SIDL    0x05
#define MCP2515_RXF2SIDL    0x09
#define MCP2515_RXF3SIDL    0x11
#define MCP2515_RXF4SIDL    0x15
#define MCP2515_RXF5SIDL    0x19

/* RXBnEID8-filter n extended identifier high
  (ADDRESS:02h,06h,0Ah,12h,16h,1Ah) */
#define MCP2515_RXF0EID8    0x02
#define MCP2515_RXF1EID8    0x06
#define MCP2515_RXF2EID8    0x0A
#define MCP2515_RXF3EID8    0x12
#define MCP2515_RXF4EID8    0x16
#define MCP2515_RXF5EID8    0x1A

/* RXBnEID0-filter n extended identifier low
  (ADDRESS:03h,07h,0Bh,13h,17h,1Bh) */
#define MCP2515_RXF0EID0    0x03
#define MCP2515_RXF1EID0    0x07
#define MCP2515_RXF2EID0    0x0B
#define MCP2515_RXF3EID0    0x13
#define MCP2515_RXF4EID0    0x17
#define MCP2515_RXF5EID0    0x1B

/* RXMnSIDH-Mask n standard identifier high (ADDRESS:20h,24h) */
#define MCP2515_RXM0SIDH    0x20
#define MCP2515_RXM1SIDH    0x24

/* RXMnSIDL-Mask n standard identifier low (ADDRESS:21h,25h) */
#define MCP2515_RXM0SIDL    0x21
#define MCP2515_RXM1SIDL    0x25

/* RXMnEID8-Mask n extended identifier high (ADDRESS:22h,26h) */
#define MCP2515_RXM0EID8    0x22
#define MCP2515_RXM1EID8    0x26

/* RXMnEID0-Mask n extended identifier low (ADDRESS:23h,27h) */
#define MCP2515_RXM0EID0    0x23
#define MCP2515_RXM1EID0    0x27

/* CNF1-configuration 1 (ADDRESS:2Ah) */
#define MCP2515_CNF1        0x2A

#define SJW0                0x40
#define SJW1                0x80

#define BRP5                0x20
#define BRP4                0x10
#define BRP3                0x08
#define BRP2                0x04
#define BRP1                0x02

```

```

#define BRP0          0x01
/* CNF2-configuration 2 (ADDRESS:29h) */
#define MCP2515_CNF2    0x29

#define BTLMODE        0x80
#define SAM            0x40
#define PRSEG12        0x20
#define PRSEG11        0x10
#define PRSEG10        0x08
#define PRSEG2         0x24
#define PRSEG1         0x02
#define PRSEG0         0x01

/* CNF3-configuration 3 (ADDRESS:28h) */
#define MCP2515_CNF3    0x28

#define SOF_ENABLE     0x80
#define WAKFIL_ENABLE  0x40
#define PHSEG22        0x04
#define PHSEG21        0x02
#define PHSEG20        0x01

/*****
 * Error detection(P47)
 *****/
/* TEC-transmit error counter (ADDRESS:1Ch) */
#define MCP2515_TEC     0x1C
/* REC-receiver error counter (ADDRESS:1Dh) */
#define MCP2515_REC     0x1D
/* EFLG-error flag (ADDRESS:2Dh) */
#define MCP2515_EFLG    0x2D

/*****
 * Interrupts (P51)
 *****/

#define MCP2515_CANINTE    0x2B // CANINTE-interrupt enable
#define MCP2515_TX_INT    0x1C // Enable all transmit interrupts
#define MCP2515_TX01_INT  0x0C // Enable TXB0 and TXB1 interrupts
#define MCP2515_RX_INT    0x03 // Enable receive interrupts
#define MCP2515_NO_INT    0x00 // Disable all interrupts
#define MCP2515_CANINTF    0x2C // CANNINTF-interrupt flag

#define MCP2515_MERR      0x80
#define MCP2515_WAKIF     0x40
#define MCP2515_ERRIF     0x20
#define MCP2515_TX2IF     0x10
#define MCP2515_TX1IF     0x08
#define MCP2515_TX0IF     0x04
#define MCP2515_RX1IF     0x02
#define MCP2515_RX0IF     0x01

/*****
 * Modes of operation (P66)
 *****/
/* CANCTRL-CAN control register(ADDRESS:XfH) */

```

```

#define MCP2515_CANCTRL    0x0F

#define MODE_NORMAL        0x00
#define MODE_SLEEP         0x20
#define MODE_LOOPBACK      0x40
#define MODE_LISTENONLY    0x60
#define MODE_CONFIG        0x80
#define MODE_POWERUP       0xE0
#define MODE_MASK          0xE0

#define ABAT_TX            0x10
#define MODE_OSM           0x08
#define CLKOUT_ENABLE      0x04
#define CLKOUT_DISABLE     0x00
#define CLKOUT_PS1         0x00
#define CLKOUT_PS2         0x01
#define CLKOUT_PS4         0x02
#define CLKOUT_PS8         0x03

/* CANSTAT-CAN status register(ADDRESS:XEH) */
#define MCP2515_CANSTAT    0x0E

/*****
 * SPI interface (P66)
 *****/
/* Define SPI Instruction Set */
#define MCP2515_RESET      0xC0 // reset
#define MCP2515_READ       0x03 // Read
#define MCP2515_READ_RXB0SIDH 0x90 // Read Rx0 buffer_standard id high
#define MCP2515_READ_RXB0D0 0x92 // receive buffer 0 data byte 0
#define MCP2515_READ_RXB1SIDH 0x94 // Read Rx1 buffer
#define MCP2515_READ_RXB1D0 0x96 // receive buffer 1 data byte 0
#define MCP2515_WRITE      0x02 // write
#define MCP2515_LOAD_TX0   0x40 // load TX0 Buffer
#define MCP2515_LOAD_TX1   0x42 // load TX1 Buffer
#define MCP2515_LOAD_TX2   0x44 // load TX2 Buffer
#define MCP2515_RTS_TX0    0x81 // RTS(message request to send)
#define MCP2515_RTS_TX1    0x82 // RTS(message request to send)
#define MCP2515_RTS_TX2    0x84 // RTS(message request to send)
#define MCP2515_RTS_ALL    0x87 // RTS(message request to send)
#define MCP2515_READ_STATUS 0xA0 // Read status
#define MCP2515_RX_STATUS  0xB0 // Rx status
#define MCP2515_BITMODIFY   0x05 // bit modify

#endif /* MCP2515_H */

```

E. SPI_DRIVER.H

```

/*****
* File: SPI_driver.h
* Created on: Sep 26, 2012
* Author: Tung-Hsun Tsou(Tony)
* Email: gotolafa@gmail.com
* Version: v1.00
* Reference: DS570 LogiCORE IP XPS Serial Peripheral Interface (SPI) (v2.02a)
*           www.xilinx.com/support/documentation/ip_documentation/xps_spi.pdf
* Copyright(c)Tung-Hsun Tsou 2012 All rights reserved.
*****/

#ifndef SPI_DRIVER_H_
#define SPI_DRIVER_H_

#include "xparameters.h"
/*****
* CANCTRL- CAN CONTROL REGISTER(xFh)
*
* bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0
* PEQOP2 PEQOP1 PEQOP0 ABAT OSM CLKEN CLKPRE1 CLKPRE0
*
*PEQOP<2:0> 000 Normal Operation mode,
*           001 Sleep mode,
*           010 Loopback mode,
*           011 Listen-Only,
*           100 Configuration mode
* CLKPRE<1:0> 00 System clock/1,
*           01 System clock/2,
*           10 System clock/4,
*           11 System clock/8
*****/
#define CANCTRL    0x0F
#define REQOP2     0x80 // Request operation mode bits
#define REQOP1     0x40 // Request operation mode bits
#define REQOP0     0x20 // Request operation mode bits
#define ABAT       0x10 // Abort All Pending Transmissions bit
#define OSM        0x08 // One-Shot mode
#define CLKEN      0x04 // CLKOUT PIN Enable bit
#define CLKPRE1    0x02 // CLKOUT Pin Prescaler bits
#define CLKPRE0    0x01 // CLKOUT Pin Prescaler bits

/*****
* CANSTATL- CAN STATUS REGISTER(xEh)
*****/
#define CANSTAT     0x0E
// Define AXI Bass Address
#define SPI_BASEADDR XPAR_AXI_SPI_0_BASEADDR

/*****
* Control register define
*****/
#define XSPI_CR (* (volatile unsigned short *) (SPI_BASEADDR|0x060))
/* control register */
#define XSPI_CR_LOOP_MODE          0x00000001 // Loop mode

```

```

#define XSPI_CR_ENABLE_MODE          0x00000002 // Enable system
#define XSPI_CR_MASTER_MODE          0x00000004 // Master mode
#define XSPI_CR_CPLO_MODE            0x00000008 // CPLO
#define XSPI_CR_CPHA_MODE            0x00000010 // CPHA
#define XSPI_CR_TxFIFO_Reset          0x00000020 // TxFIFO_Reset
#define XSPI_CR_RxFIFO_Reset          0x00000040 // RxFIFO_Reset
#define XSPI_CR_Manual_SS_MODE        0x00000080 // Manual Slave Select
                                     // Assertion Enable
#define XSPI_CR_MASTER_TRANS_INHIBIT_MODE 0x00000100
#define XSPI_CR_LSB_FIRST              0x00000200

/*****
 * Status register
 *****/
#define XSPI_SR (* (volatile unsigned char *) (SPI_BASEADDR|0x64))
/* status register */
#define XSPI_SR_RX_EMPTY 0x00000001 // Receive Full
#define XSPI_SR_RX_FULL 0x00000002 // Receive Full
#define XSPI_SR_TX_EMPTY 0x00000004 // Transmit Empty
#define XSPI_SR_TX_FULL 0x00000008 // Transmit Full

/*****
 * Data transmit
 *****/
#define XSPI_DTR (* (volatile unsigned char *) (SPI_BASEADDR|0x68))
/* Data transmit */

/*****
 * Data receive
 *****/
#define XSPI_DRR (* (volatile unsigned char *) (SPI_BASEADDR|0x6C))
/* Data receive */

/*****
 * Slave Select Register
 *****/
#define XSPI_SSR (* (volatile unsigned char *) ((SPI_BASEADDR|0x70)))

/*****
 * SPI Transmit FIFO Occupancy Register
 *****/
#define XSPI_TX_FIFO_OCY (* (volatile unsigned char *) ((SPI_BASEADDR|0x74)))

/*****
 * SPI Receive FIFO Occupancy Register
 *****/
#define XSPI_RX_FIFO_OCY (* (volatile unsigned char *) ((SPI_BASEADDR|0x78)))
#endif /* SPI_DRIVER_H_ */

```


F. SPL.H

```

/*****
* File: SPL.h
* Created on: Oct 16, 2012
* Author: Tung-Hsun Tsou(Tony)
* Email: gotolafa@gmail.com
* Version: v1.00
* Reference: DS570 LogiCORE IP XPS Serial Peripheral Interface (SPI) (v2.02a)
*           www.xilinx.com/support/documentation/ip_documentation/xps_spi.pdf
* Copyright(c)Tung-Hsun Tsou 2012 All rights reserved.
*****/

#ifndef SPI_H_
#define SPI_H_

typedef struct can_frame
{
    unsigned int status;
    unsigned int can_id;
    unsigned int can_dlc;
    unsigned char data[8];
} can_type;

extern can_type can[64];

void SPI_exchange(int *ptr,int count);
int SPI_exchange_one(void);

inline void CAN_init(void);
inline void CAN_Reset();
inline int CAN_ReadInst( int address);
inline void CAN_WriteInst(int address, int Data);
inline void CAN_BitModifyInst(int address,int Mask,int Data);
inline void CAN_RTSInst(int address,int Mask,int Data);

inline void CAN_Read_RX_Buffer(int address);

inline void ERROR_frame(int err_status,int index);

inline void Data_frame_Read(int RX,int index);

#endif /* SPI_H_ */
```

G. SPI.C

```

/*****
* File: SPI.c
* Created on: Sep 26, 2012
* Author: Tung-Hsun Tsou(Tony)
* Email: gotolafa@gmail.com
* Version: v1.0 First time can read...
*          v1.2 polling
* Reference: DS570 LogiCORE IP XPS Serial Peripheral Interface (SPI) (v2.02a)
*            www.xilinx.com/support/documentation/ip_documentation/xps_spi.pdf
* Copyright(c)Tung-Hsun Tsou 2012 All rights reserved.
*****/

#include "stdio.h"
#include "SPI_driver.h"
#include "SPI.h"
#include "MCP2515.h"

unsigned char Buffer_R;
unsigned char read_value[13];
int i;
#define chip_select XSPI_SSR=0x00
#define chip_deselect XSPI_SSR=0xFF

can_type can[64];
/*****
* Write Functions
*****/
void SPI_exchange(int *ptr,int count)
{
    while ((XSPI_SR & XSPI_RX_FIFO_OCY)==count-1);
    for(i=0;i<count;i++)
        *ptr++ = XSPI_DRR;
}
int SPI_exchange_one(void)
{
    while (XSPI_SR & XSPI_SR_RX_EMPTY);
    return XSPI_DRR;
}

/*****
* SPI initial
*****/
inline void CAN_init(void)
{
    int Control=0;
    //master mode & manual Slave
    Control = XSPI_CR;
    Control |=((XSPI_CR_MASTER_MODE)|(XSPI_CR_Manual_SS_MODE));
    XSPI_CR = Control;
    //Slave Select flag
    XSPI_SSR=0xFF;
    /* asserted when core configured in slave mode and selected */
    /* by external SPI master. */
    //CPLO&CPHA

```

```

Control &=~XSPI_CR_CPLO_MODE;
Control &=~XSPI_CR_CPHA_MODE ;
//enable device
Control = XSPI_CR;
Control &=~XSPI_CR_MASTER_TRANS_INHIBIT_MODE;
Control |= XSPI_CR_ENABLE_MODE;
XSPI_CR = Control;
//empty Transmit and Receive FIFO
Control = XSPI_CR;
Control &=~XSPI_CR_TxFIFO_Reset;
Control &=~XSPI_CR_RxFIFO_Reset;
XSPI_CR = Control;
XSPI_SSR=0xFF;
}
/*****
* Reset Instruction
*****/
inline void CAN_Reset()
{
    chip_select;
    XSPI_DTR = MCP2515_RESET;
    SPI_exchange_one();
    chip_deselect;
}
/*****
* Read Instruction
*****/
inline int CAN_ReadInst(address)
{
    chip_select;
    XSPI_DTR = MCP2515_READ; //put data to the transmitter
    XSPI_DTR = address; //put data to the transmitter
    XSPI_DTR = 0xFF; //put data to the transmitter
    while ((XSPI_SR & XSPI_RX_FIFO_OCY)==2);
    XSPI_DRR;
    XSPI_DRR;
    Buffer_R = XSPI_DRR;
    chip_deselect;
    return Buffer_R;
}
/*****
* Read Rx Buffer Instruction
*****/
inline void CAN_Read_RX_Buffer(address)
{
    chip_select;
    XSPI_DTR = address;
    XSPI_DTR = 0x00;
    SPI_exchange(Buffer_R,2);
    chip_deselect;
}
/*****
* Write Instruction
*****/
inline void CAN_WriteInst(address,Data)
{

```

```

chip_select;
XSPI_DTR = MCP2515_WRITE;
XSPI_DTR = address;
XSPI_DTR = Data;
while ((XSPI_SR & XSPI_RX_FIFO_OCY)==2);
XSPI_CR|= XSPI_CR_RxFIFO_Reset;
chip_deselect;
}
/*****
* Bit Modify Instruction
*****/
inline void CAN_BitModifyInst(address,Mask,Data)
{
chip_select;
XSPI_DTR = MCP2515_BITMODIFY;
XSPI_DTR = address;
XSPI_DTR = Mask;
XSPI_DTR = Data;
while ((XSPI_SR & XSPI_RX_FIFO_OCY)==3);
XSPI_CR|= XSPI_CR_RxFIFO_Reset;
chip_deselect;
}
/*****
* Request-To-Send Instruction
*****/
inline void CAN_RTSInst(address,Mask,Data)
{
chip_select;
XSPI_DTR = MCP2515_BITMODIFY;
XSPI_DTR = address;
XSPI_DTR = Mask;
XSPI_DTR = Data;
while ((XSPI_SR & XSPI_RX_FIFO_OCY)==3);
XSPI_CR|= XSPI_CR_RxFIFO_Reset;
chip_deselect;
}
/*****
* Rx Status Instruction
*****/
inline void CAN_RX_Status()
{
chip_select;
XSPI_DTR = MCP2515_READ_STATUS;
XSPI_DTR = 0xFF;
while ((XSPI_SR & XSPI_RX_FIFO_OCY)==1);
XSPI_CR|= XSPI_CR_RxFIFO_Reset;
chip_deselect;
}
/*****
* ERROR frame
*****/
inline void ERROR_frame(int err_status,int index)
{
chip_select;
can[index].status = CAN_ERROR;
XSPI_DTR = MCP2515_READ;

```

```

XSPI_DTR = MCP2515_EFLG;
XSPI_DTR = 0xFF;
XSPI_DTR = MCP2515_READ;
XSPI_DTR = MCP2515_TEC;
XSPI_DTR = 0xFF;
XSPI_DTR = 0xFF;
while ((XSPI_SR & XSPI_RX_FIFO_OCY) == 6);
XSPI_DRR;
XSPI_DRR;
can[index].data[0] = XSPI_DRR; //EFLAG
XSPI_DRR;
XSPI_DRR;
can[index].data[1] = XSPI_DRR; //Transmit error counter
can[index].data[2] = XSPI_DRR; //Receive Error Counter
CAN_BitModifyInst (MCP2515_CANINTF, MCP2515_ERRIF, 0x00);
chip_deselect;
}
/*****
* Standard data frame Read instruction
*****/
inline void Data_frame_Read(int RX,int index)
{
    XSPI_CR|= XSPI_CR_RxFIFO_Reset;
    chip_select;
    XSPI_DTR = RX;
    XSPI_DTR = 0xFF; //read rx buffer instruction
    XSPI_DTR = 0xFF;
    XSPI_DTR = 0xFF;
    XSPI_DTR = 0xFF;
    XSPI_DTR = 0xFF;
    XSPI_DTR = 0xFF;
    XSPI_DTR = 0xFF;
    XSPI_DTR = 0xFF;
    XSPI_DTR = 0xFF;
    XSPI_DTR = 0xFF;
    XSPI_DTR = 0xFF;
    XSPI_DTR = 0xFF;
    while ((XSPI_SR & XSPI_RX_FIFO_OCY) == 12);
    XSPI_DRR;
    can[index].can_id = XSPI_DRR; //62
    can[index].can_id = can[index].can_id << 3;
    can[index].can_id = can[index].can_id | XSPI_DRR >> 5;
    XSPI_DRR; //dummy read
    XSPI_DRR; //dummy read
    can[index].can_dlc = XSPI_DRR;
    if (can[index].can_dlc & RTR)
    {
        can[index].status = CAN_RTR;
    }
    else
    {
        can[index].status = CAN_RECEIVE;
    }
    can[index].data[0] = XSPI_DRR;
    can[index].data[1] = XSPI_DRR;

```

```
can[index].data[2] = XSPI_DRR;  
can[index].data[3] = XSPI_DRR;  
can[index].data[4] = XSPI_DRR;  
can[index].data[5] = XSPI_DRR;  
can[index].data[6] = XSPI_DRR;  
can[index].data[7] = XSPI_DRR;  
chip_deselect;  
}
```

H. INTC_DRIVER.H

```

/*****
* File: INTC_driver.h
* Created on: Nov 7, 2012
* Author: Tung-Hsun Tsou(Tony)
* Email: gotolafa@gmail.com
* Version: v1.0
* Reference: Xilinx DS572 LogiCORE IP XPS Interrupt Controller (v2.01a)
*           www.xilinx.com/support/documentation/ip_documentation/xps_intc.pdf
* Copyright(c)Tung-Hsun Tsou 2012 All rights reserved.
*****/

#ifndef INTC_DRIVER_H_
#define INTC_DRIVER_H_

#define INTC_BASEADDR    XPAR_MICROBLAZE_0_INTC_BASEADDR

/*****
* Interrupt Status Register(ISR)(P13)
*****/
#define XINTC_ISR (* (volatile unsigned long *) (INTC_BASEADDR + 0x00))

/*****
* Interrupt Pending Register(IPR)(P14)
*****/
#define XINTC_IPR (* (volatile unsigned long *) (INTC_BASEADDR + 0x04))

/*****
* Interrupt Enable Register(IER)(P15)
*****/
#define XINTC_IER (* (volatile unsigned long *) (INTC_BASEADDR + 0x08))

/*****
* Interrupt Acknowledge Register(IAR)(P16)
*****/
#define XINTC_IAR (* (volatile unsigned long *) (INTC_BASEADDR + 0x0C))

/*****
* Interrupt Vector Register(IVR)(P19)
*****/
#define XINTC_IVR (* (volatile unsigned long *) (INTC_BASEADDR + 0x18))

/*****
* Master Enable Register(MER)(P20)
*****/
#define XINTC_MER (* (volatile unsigned long *) (INTC_BASEADDR + 0x1C))
#define ME        0x01
#define HIE        0x02

/*****
* Interrupt Mode Register(IMR)(P20)
*****/
#define XINTC_IMR (* (volatile unsigned long *) (INTC_BASEADDR + 0x20))

/*****/

```

```
* Interrupt Vector address Register(IVAR)(P21)
*****/
#define XINTC_IVAR0 (* (volatile unsigned long *) (INTC_BASEADDR + 0x100))

#endif /* INTC_DRIVER_H_ */
```


I. CAN.C

```

/*****
* File: CAN.c
* Created on: Oct 12, 2012
* Author: Tung-Hsun Tsou(Tony)
* Email: gotolafa@gmail.com
* Version: v1.0 First time can read...
*         v1.2 polling
*         v2.0 external interrupt
* Reference: Microchip DS21801D MCP2515
*           www.microchip.com/downloads/en/devicedoc/21801d.pdf
* Copyright(c)Tung-Hsun Tsou 2012 All rights reserved.
*****/

#include "SPI_driver.h"
#include "SPI.h"
#include "MCP2515.h"
#include "INTC_driver.h"
#include "xparameters.h"
#include "mb_interface.h"
#include "stdio.h"

#define EINT1_INTC XPAR_SYSTEM_EINT1_PIN_0_MASK
unsigned int intc_status;
int error_count = 0;
int can_w_index = 0;
int can_r_index = 0;

void myISR(void) __attribute__((interrupt_handler));
void myISR(void)
{
    if(XINTC_IPR & EINT1_INTC)
    {
        XSPI_CR |= XSPI_CR_RxFIFO_Reset;
        while(1)
        {
            intc_status = CAN_ReadInst(MCP2515_CANINTF);

            if (intc_status & MCP2515_ERRIF)
            {
                can[1].status = CAN_ERROR;
                ERROR_frame(intc_status,can_w_index);
                error_count++;
            }
            else if(intc_status & MCP2515_RX0IF)
                Data_frame_Read(MCP2515_READ_RXB0SIDH,can_w_index);
            else if(intc_status & MCP2515_RX1IF)
                Data_frame_Read(MCP2515_READ_RXB1SIDH,can_w_index);
            else
                break;
            can_w_index = (can_w_index+1)%64;
            XSPI_CR |= XSPI_CR_RxFIFO_Reset;
        }
    }
}
```

```

XINTC_IAR=XINTC_IPR;
}

int main (void)
{
    microblaze_disable_interrupts();
    CAN_init ();
    CAN_Reset ();

    CAN_WriteInst (MCP2515_CNF1, SJW0);// length=2*Tq Tq=2/Fosc
    CAN_WriteInst (MCP2515_CNF2, BTLMODE | SAM | PRSEG10 | PRSEG0);//2
    CAN_WriteInst (MCP2515_CNF3, SOF_ENABLE | WAKFIL_ENABLE | PHSEG21);//1
    CAN_WriteInst (MCP2515_CANINTE, MCP2515_RX_INT);/* Rx interrupt */
    CAN_WriteInst (MCP2515_BFPCTRL, B0BFE | B1BFE );/* enable RX0BF RX1BF*/
    CAN_BitModifyInst (MCP2515_CANCTRL, MODE_MASK, MODE_LISTENONLY);

    XINTC_IER = EINT1_INTC;//interrupt enable
    XINTC_MER = ME | HIE;

    microblaze_enable_interrupts();
    while (1)
    {
        if(can_w_index != can_r_index)
        {
            if (can[can_r_index].status == CAN_ERROR)
            {
                xil_printf("ERROR : EFLAG = %d, TEC= %d , REC = %d \r\n",
                    can[can_r_index].data[0], can[can_r_index].data[1],
                    can[can_r_index].data[2]);
            }
            else if (can[can_r_index].status == CAN_RECEIVE)
            {
                xil_printf("Receive : ID = %x , Data Length =%x , "
                    "message = %c%c%c%c%c%c%c%c , %x%x%x%x%x%x%x%x\r\n",
                    can[can_r_index].can_id, (can[can_r_index].can_dlc & 0x0F),
                    can[can_r_index].data[0], can[can_r_index].data[1] ,
                    can[can_r_index].data[2] , can[can_r_index].data[3] ,
                    can[can_r_index].data[4] , can[can_r_index].data[5] ,
                    can[can_r_index].data[6] , can[can_r_index].data[7],
                    can[can_r_index].data[0] , can[can_r_index].data[1] ,
                    can[can_r_index].data[2] , can[can_r_index].data[3] ,
                    can[can_r_index].data[4] , can[can_r_index].data[5] ,
                    can[can_r_index].data[6] , can[can_r_index].data[7] );
            }
            else if (can[can_r_index].status == CAN_RTR)
                xil_printf("RTR : ID = %x", can[can_r_index].can_id );
            else
                asm("nop");
            can_r_index = (can_r_index+1)%64;
        }
    }
    return 0;
}

```

J. XILINX.DTS

```
/*
* Device Tree Generator version: 1.1
*
* (C) Copyright 2007-2012 Xilinx, Inc.
* (C) Copyright 2007-2012 Michal Simek
* (C) Copyright 2007-2012 PetaLogix Qld Pty Ltd
*
* Michal SIMEK <monstr@monstr.eu>
*
* This program is free software; you can redistribute it and/or
* modify it under the terms of the GNU General Public License as
* published by the Free Software Foundation; either version 2 of
* the License, or (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston,
* MA 02111-1307 USA
*
* CAUTION: This file is automatically generated by libgen.
* Version: Xilinx EDK 14.1 EDK_P.15xf
* Today is: Tuesday, the 18 of June, 2013; 22:44:35
*
* XPS project directory: SP605_0312_AXI
*/

/dts-v1/;
/ {
    #address-cells = <1>;
    #size-cells = <1>;
    compatible = "xlnx,microblaze";
    model = "SP605_0312_AXI";
    MCB_DDR3: memory@a8000000 {
        device_type = "memory";
        reg = <0xa8000000 0x8000000 >;
    };
    aliases {
        ethernet0 = &Ethernet_Lite;
        serial0 = &RS232_Uart_1;
    };
    chosen {
        bootargs = "console=ttyUL0,115200";
        linux,stdout-path = "/axi@1/serial@40a00000";
    };
    cpus {
        #address-cells = <1>;
        #cpus = <0x1>;
        #size-cells = <0>;
```

```

microblaze_0: cpu@0 {
    clock-frequency = <100000000>;
    compatible = "xlnx,microblaze-8.30.a";
    d-cache-baseaddr = <0xa8000000>;
    d-cache-highaddr = <0xffffffff>;
    d-cache-line-size = <0x10>;
    d-cache-size = <0x2000>;
    device_type = "cpu";
    i-cache-baseaddr = <0xa8000000>;
    i-cache-highaddr = <0xffffffff>;
    i-cache-line-size = <0x10>;
    i-cache-size = <0x2000>;
    model = "microblaze,8.30.a";
    reg = <0>;
    timebase-frequency = <100000000>;
    xlnx,addr-tag-bits = <0xe>;
    xlnx,allow-dcache-wr = <0x1>;
    xlnx,allow-icache-wr = <0x1>;
    xlnx,area-optimized = <0x0>;
    xlnx,avoid-primitives = <0x0>;
    xlnx,branch-target-cache-size = <0x0>;
    xlnx,cache-byte-size = <0x2000>;
    xlnx,d-axi = <0x1>;
    xlnx,d-lmb = <0x1>;
    xlnx,d-plb = <0x0>;
    xlnx,data-size = <0x20>;
    xlnx,dcache-addr-tag = <0xe>;
    xlnx,dcache-always-used = <0x1>;
    xlnx,dcache-byte-size = <0x2000>;
    xlnx,dcache-data-width = <0x0>;
    xlnx,dcache-force-tag-lutram = <0x0>;
    xlnx,dcache-interface = <0x0>;
    xlnx,dcache-line-len = <0x4>;
    xlnx,dcache-use-fsl = <0x0>;
    xlnx,dcache-use-writeback = <0x0>;
    xlnx,dcache-victims = <0x0>;
    xlnx,debug-enabled = <0x1>;
    xlnx,div-zero-exception = <0x0>;
    xlnx,dynamic-bus-sizing = <0x1>;
    xlnx,ecc-use-ce-exception = <0x0>;
    xlnx,edge-is-positive = <0x1>;
    xlnx,endianness = <0x1>;
    xlnx,family = "spartan6";
    xlnx,fault-tolerant = <0x0>;
    xlnx,fpu-exception = <0x0>;
    xlnx,freq = <0x5f5e100>;
    xlnx,fsl-data-size = <0x20>;
    xlnx,fsl-exception = <0x0>;
    xlnx,fsl-links = <0x0>;
    xlnx,i-axi = <0x1>;
    xlnx,i-lmb = <0x1>;
    xlnx,i-plb = <0x0>;
    xlnx,icache-always-used = <0x1>;
    xlnx,icache-data-width = <0x0>;
    xlnx,icache-force-tag-lutram = <0x0>;
    xlnx,icache-interface = <0x0>;

```

```

xlnx,icache-line-len = <0x4>;
xlnx,icache-streams = <0x0>;
xlnx,icache-use-fsl = <0x0>;
xlnx,icache-victims = <0x0>;
xlnx,ill-opcode-exception = <0x1>;
xlnx,instance = "microblaze_0";
xlnx,interconnect = <0x2>;
xlnx,interrupt-is-edge = <0x0>;
xlnx,lockstep-slave = <0x0>;
xlnx,mmu-dtlb-size = <0x4>;
xlnx,mmu-itlb-size = <0x2>;
xlnx,mmu-privileged-instr = <0x0>;
xlnx,mmu-tlb-access = <0x3>;
xlnx,mmu-zones = <0x2>;
xlnx,number-of-pc-brk = <0x1>;
xlnx,number-of-rd-addr-brk = <0x0>;
xlnx,number-of-wr-addr-brk = <0x0>;
xlnx,opcode-0x0-illegal = <0x1>;
xlnx,optimization = <0x0>;
xlnx,pc-width = <0x20>;
xlnx,pvr = <0x0>;
xlnx,pvr-user1 = <0x0>;
xlnx,pvr-user2 = <0x0>;
xlnx,reset-msr = <0x0>;
xlnx,sco = <0x0>;
xlnx,stream-interconnect = <0x0>;
xlnx,unaligned-exceptions = <0x1>;
xlnx,use-barrel = <0x1>;
xlnx,use-branch-target-cache = <0x0>;
xlnx,use-dcache = <0x1>;
xlnx,use-div = <0x1>;
xlnx,use-ext-brk = <0x1>;
xlnx,use-ext-nm-brk = <0x1>;
xlnx,use-extended-fsl-instr = <0x0>;
xlnx,use-fpu = <0x0>;
xlnx,use-hw-mul = <0x1>;
xlnx,use-icache = <0x1>;
xlnx,use-interrupt = <0x1>;
xlnx,use-mmu = <0x3>;
xlnx,use-msr-instr = <0x1>;
xlnx,use-pcmp-instr = <0x1>;
xlnx,use-reorder-instr = <0x1>;
xlnx,use-stack-protection = <0x0>;
    };
};
axi4_0: axi@0 {
    #address-cells = <1>;
    #size-cells = <1>;
    compatible = "xlnx,axi-interconnect-1.06.a", "simple-bus";
    ranges ;
};
axi4lite_0: axi@1 {
    #address-cells = <1>;
    #size-cells = <1>;
    compatible = "xlnx,axi-interconnect-1.06.a", "simple-bus";
    ranges ;
};

```

```

DIP_Switches_4Bits: gpio@40040000 {
    #gpio-cells = <2>;
    compatible = "xlnx,axi-gpio-1.01.b", "xlnx,xps-gpio-1.00.a";
    gpio-controller ;
    reg = < 0x40040000 0x10000 >;
    xlnx,all-inputs = <0x1>;
    xlnx,all-inputs-2 = <0x0>;
    xlnx,dout-default = <0x0>;
    xlnx,dout-default-2 = <0x0>;
    xlnx,family = "spartan6";
    xlnx,gpio-width = <0x4>;
    xlnx,gpio2-width = <0x20>;
    xlnx,instance = "DIP_Switches_4Bits";
    xlnx,interrupt-present = <0x0>;
    xlnx,is-dual = <0x0>;
    xlnx,tri-default = <0xffffffff>;
    xlnx,tri-default-2 = <0xffffffff>;
} ;

Ethernet_Lite: ethernet@40e00000 {
    compatible = "xlnx,axi-ethernetlite-1.01.b", "xlnx,xps-ethernetlite-1.00.a";
    device_type = "network";
    local-mac-address = [ 00 0a 35 00 00 00 ];
    reg = < 0x40e00000 0x10000 >;
    xlnx,duplex = <0x1>;
    xlnx,family = "spartan6";
    xlnx,include-global-buffers = <0x0>;
    xlnx,include-internal-loopback = <0x0>;
    xlnx,include-mdio = <0x1>;
    xlnx,include-phy-constraints = <0x1>;
    xlnx,instance = "Ethernet_Lite";
    xlnx,rx-ping-pong = <0x0>;
    xlnx,s-axi-id-width = <0x1>;
    xlnx,s-axi-supports-narrow-burst = <0x0>;
    xlnx,tx-ping-pong = <0x0>;
} ;

LEDs_4Bits: gpio@40020000 {
    #gpio-cells = <2>;
    compatible = "xlnx,axi-gpio-1.01.b", "xlnx,xps-gpio-1.00.a";
    gpio-controller ;
    reg = < 0x40020000 0x10000 >;
    xlnx,all-inputs = <0x0>;
    xlnx,all-inputs-2 = <0x0>;
    xlnx,dout-default = <0x0>;
    xlnx,dout-default-2 = <0x0>;
    xlnx,family = "spartan6";
    xlnx,gpio-width = <0x4>;
    xlnx,gpio2-width = <0x20>;
    xlnx,instance = "LEDs_4Bits";
    xlnx,interrupt-present = <0x1>;
    xlnx,is-dual = <0x0>;
    xlnx,tri-default = <0xffffffff>;
    xlnx,tri-default-2 = <0xffffffff>;
} ;

Linear_Flash: flash@42000000 {
    bank-width = <2>;
    compatible = "xlnx,axi-emc-1.03.a", "cfi-flash";

```

```

reg = < 0x42000000 0x2000000 >;
xlnx,axi-clk-period-ps = <0x4e20>;
xlnx,family = "spartan6";
xlnx,include-datawidth-matching-0 = <0x1>;
xlnx,include-datawidth-matching-1 = <0x0>;
xlnx,include-datawidth-matching-2 = <0x0>;
xlnx,include-datawidth-matching-3 = <0x0>;
xlnx,include-negedge-ioregs = <0x0>;
xlnx,instance = "Linear_Flash";
xlnx,lflash-period-ps = <0x4e20>;
xlnx,linear-flash-sync-burst = <0x0>;
xlnx,max-mem-width = <0x10>;
xlnx,mem0-type = <0x2>;
xlnx,mem0-width = <0x10>;
xlnx,mem1-type = <0x0>;
xlnx,mem1-width = <0x20>;
xlnx,mem2-type = <0x0>;
xlnx,mem2-width = <0x20>;
xlnx,mem3-type = <0x0>;
xlnx,mem3-width = <0x20>;
xlnx,num-banks-mem = <0x1>;
xlnx,parity-type-mem-0 = <0x0>;
xlnx,parity-type-mem-1 = <0x0>;
xlnx,parity-type-mem-2 = <0x0>;
xlnx,parity-type-mem-3 = <0x0>;
xlnx,s-axi-en-reg = <0x0>;
xlnx,s-axi-mem-addr-width = <0x20>;
xlnx,s-axi-mem-data-width = <0x20>;
xlnx,s-axi-mem-id-width = <0x1>;
xlnx,s-axi-mem-protocol = "AXI4LITE";
xlnx,s-axi-reg-addr-width = <0x20>;
xlnx,s-axi-reg-data-width = <0x20>;
xlnx,s-axi-reg-protocol = "axi4";
xlnx,synch-pipedelay-0 = <0x2>;
xlnx,synch-pipedelay-1 = <0x2>;
xlnx,synch-pipedelay-2 = <0x2>;
xlnx,synch-pipedelay-3 = <0x2>;
xlnx,tavdv-ps-mem-0 = <0x1fbd0>;
xlnx,tavdv-ps-mem-1 = <0x3a98>;
xlnx,tavdv-ps-mem-2 = <0x3a98>;
xlnx,tavdv-ps-mem-3 = <0x3a98>;
xlnx,tcedv-ps-mem-0 = <0x1fbd0>;
xlnx,tcedv-ps-mem-1 = <0x3a98>;
xlnx,tcedv-ps-mem-2 = <0x3a98>;
xlnx,tcedv-ps-mem-3 = <0x3a98>;
xlnx,thzce-ps-mem-0 = <0x88b8>;
xlnx,thzce-ps-mem-1 = <0x1b58>;
xlnx,thzce-ps-mem-2 = <0x1b58>;
xlnx,thzce-ps-mem-3 = <0x1b58>;
xlnx,thzoe-ps-mem-0 = <0x1b58>;
xlnx,thzoe-ps-mem-1 = <0x1b58>;
xlnx,thzoe-ps-mem-2 = <0x1b58>;
xlnx,thzoe-ps-mem-3 = <0x1b58>;
xlnx,tlzwe-ps-mem-0 = <0x88b8>;
xlnx,tlzwe-ps-mem-1 = <0x0>;
xlnx,tlzwe-ps-mem-2 = <0x0>;

```

```

xlnx,tlzwe-ps-mem-3 = <0x0>;
xlnx,tpacc-ps-flash-0 = <0x61a8>;
xlnx,tpacc-ps-flash-1 = <0x61a8>;
xlnx,tpacc-ps-flash-2 = <0x61a8>;
xlnx,tpacc-ps-flash-3 = <0x61a8>;
xlnx,twc-ps-mem-0 = <0x11170>;
xlnx,twc-ps-mem-1 = <0x3a98>;
xlnx,twc-ps-mem-2 = <0x3a98>;
xlnx,twc-ps-mem-3 = <0x3a98>;
xlnx,twp-ps-mem-0 = <0x11170>;
xlnx,twp-ps-mem-1 = <0x2ee0>;
xlnx,twp-ps-mem-2 = <0x2ee0>;
xlnx,twp-ps-mem-3 = <0x2ee0>;
xlnx,twph-ps-mem-0 = <0x2ee0>;
xlnx,twph-ps-mem-1 = <0x2ee0>;
xlnx,twph-ps-mem-2 = <0x2ee0>;
xlnx,twph-ps-mem-3 = <0x2ee0>;
xlnx,wr-rec-time-mem-0 = <0x186a0>;
xlnx,wr-rec-time-mem-1 = <0x186a0>;
xlnx,wr-rec-time-mem-2 = <0x186a0>;
xlnx,wr-rec-time-mem-3 = <0x186a0>;
};
Push_Buttons_4Bits: gpio@40000000 {
    #gpio-cells = <2>;
    compatible = "xlnx,axi-gpio-1.01.b", "xlnx,xps-gpio-1.00.a";
    gpio-controller;
    reg = < 0x40000000 0x10000 >;
    xlnx,all-inputs = <0x1>;
    xlnx,all-inputs-2 = <0x0>;
    xlnx,dout-default = <0x0>;
    xlnx,dout-default-2 = <0x0>;
    xlnx,family = "spartan6";
    xlnx,gpio-width = <0x4>;
    xlnx,gpio2-width = <0x20>;
    xlnx,instance = "Push_Buttons_4Bits";
    xlnx,interrupt-present = <0x1>;
    xlnx,is-dual = <0x0>;
    xlnx,tri-default = <0xffffffff>;
    xlnx,tri-default-2 = <0xffffffff>;
};
RS232_Uart_1: serial@40a00000 {
    clock-frequency = <50000000>;
    compatible = "xlnx,axi-uartlite-1.02.a", "xlnx,xps-uartlite-1.00.a";
    current-speed = <115200>;
    device_type = "serial";
    interrupt-parent = <&microblaze_0_intc>;
    interrupts = < 3 0 >;
    port-number = <0>;
    reg = < 0x40a00000 0x10000 >;
    xlnx,baudrate = <0x1c200>;
    xlnx,data-bits = <0x8>;
    xlnx,family = "spartan6";
    xlnx,instance = "RS232_Uart_1";
    xlnx,odd-parity = <0x1>;
    xlnx,use-parity = <0x0>;
};

```



```

SysACE_CompactFlash: sysace@41800000 {
    8-bit ;
    compatible = "xlnx,axi-sysace-1.01.a", "xlnx,xps-sysace-1.00.a";
    port-number = <0>;
    reg = < 0x41800000 0x10000 >;
    xlnx,family = "spartan6";
    xlnx,instance = "SysACE_CompactFlash";
    xlnx,mem-width = <0x8>;
};

axi_gpio_0: gpio@40080000 {
    #gpio-cells = <2>;
    compatible = "xlnx,axi-gpio-1.01.b", "xlnx,xps-gpio-1.00.a";
    gpio-controller ;
    reg = < 0x40080000 0x10000 >;
    xlnx,all-inputs = <0x1>;
    xlnx,all-inputs-2 = <0x0>;
    xlnx,dout-default = <0x0>;
    xlnx,dout-default-2 = <0x0>;
    xlnx,family = "spartan6";
    xlnx,gpio-width = <0x1>;
    xlnx,gpio2-width = <0x20>;
    xlnx,instance = "axi_gpio_0";
    xlnx,interrupt-present = <0x1>;
    xlnx,is-dual = <0x0>;
    xlnx,tri-default = <0xffffffff>;
    xlnx,tri-default-2 = <0xffffffff>;
};

MCP2515_kernel_module_instance: MCP2515_kernel_module@40600000 {
    compatible = "petalogix,MCP2515_fifo";
    reg = <0x40600000 0x10000>;
    interrupt-parent = <&microblaze_0_intc>;
    interrupts = < 1 2 >;
};

axi_timer_0: system-timer@41c00000 {
    clock-frequency = <50000000>;
    compatible = "xlnx,axi-timer-1.03.a", "xlnx,xps-timer-1.00.a";
    interrupt-parent = <&microblaze_0_intc>;
    interrupts = < 0 2 >;
    reg = < 0x41c00000 0x10000 >;
    xlnx,count-width = <0x20>;
    xlnx,family = "spartan6";
    xlnx,gen0-assert = <0x1>;
    xlnx,gen1-assert = <0x1>;
    xlnx,instance = "axi_timer_0";
    xlnx,one-timer-only = <0x0>;
    xlnx,trig0-assert = <0x1>;
    xlnx,trig1-assert = <0x1>;
};

debug_module: serial@41400000 {
    compatible = "xlnx,mdm-2.00.b", "xlnx,xps-uartlite-1.00.a";
    reg = < 0x41400000 0x10000 >;
    xlnx,family = "spartan6";
    xlnx,interconnect = <0x2>;
    xlnx,jtag-chain = <0x2>;
    xlnx,mb-dbg-ports = <0x1>;
    xlnx,use-uart = <0x1>;
};

```

```

    };
    microblaze_0_intc: interrupt-controller@41200000 {
        #interrupt-cells = <0x2>;
        compatible = "xlnx,axi-intc-1.02.a", "xlnx,xps-intc-1.00.a";
        interrupt-controller;
        reg = < 0x41200000 0x10000 >;
        xlnx,kind-of-intr = <0xa>;
        xlnx,num-intr-inputs = <0x4>;
    };
};

```

K. SPI_DRIVER_LINUX.H

```

/*****
* File: SPI_driver_linux.h
* Created on: Sep 26, 2012
* Author: Tung-Hsun Tsou(Tony)
* Email: gotolafa@gmail.com
* Version: v1.00
*          v2.00 Linux version
* Reference: DS742 LogiCORE IP AXI Serial Peripheral Interface (v1.01a)
*            www.xilinx.com/support/documentation/ip_documentation/axi_spi_ds742.pdf
* Copyright(c)Tung-Hsun Tsou 2012 All rights reserved.
*****/

#ifndef SPI_DRIVER_H_
#define SPI_DRIVER_H_

#define SPI_MAP_SIZE 0x10000

/*****
* CANCTRL- CAN CONTROL REGISTER(xFh)
*
* bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0
* PEQOP2 PEQOP1 PEQOP0 ABAT OSM CLKEN CLKPRE1 CLKPRE0
*
*PEQOP<2:0> 000 Normal Operation mode,
*           001 Sleep mode,
*           010 Loopback mode,
*           011 Listen-Only,
*           100 Configuration mode
* CLKPRE<1:0> 00 System clock/1,
*            01 System clock/2,
*            10 System clock/4,
*            11 System clock/8
*****/
#define CANCTRL 0x0F
#define REQOP2 0x80 /* Request operation mode bits */
#define REQOP1 0x40 /* Request operation mode bits */
#define REQOP0 0x20 /* Request operation mode bits */
#define ABAT 0x10 /* Abort All Pending Transmissions bit */
#define OSM 0x08 /* One-Shot mode */
#define CLKEN 0x04 /* CLKOUT PIN Enable bit */
#define CLKPRE1 0x02 /* CLKOUT Pin Prescaler bits */
#define CLKPRE0 0x01 /* CLKOUT Pin Prescaler bits */

/*****
* CANSTATL- CAN STATUS REGISTER(xEh)
*****/
#define CANSTAT 0x0E
/* Define AXI Bass Address */
#define SPI_BASEADDR XPAR_AXI_SPI_0_BASEADDR

/*****
* Control register define
*****/
#define XSPI_CR 0x060

```

```

/* control register */
#define XSPI_CR_LOOP_MODE          0x00000001 /* Loop mode */
#define XSPI_CR_ENABLE_MODE        0x00000002 /* Enable system */
#define XSPI_CR_MASTER_MODE        0x00000004 /* Master mode */
#define XSPI_CR_CPLO_MODE          0x00000008 /* CPLO */
#define XSPI_CR_CPHA_MODE          0x00000010 /* CPHA */
#define XSPI_CR_TxFIFO_Reset       0x00000020 /* TxFIFO_Reset */
#define XSPI_CR_RxFIFO_Reset       0x00000040 /* RxFIFO_Reset */
#define XSPI_CR_Manual_SS_MODE      0x00000080 /* Manual Slave Select
                                     Assertion Enable */
#define XSPI_CR_MASTER_TRANS_INHIBIT_MODE 0x00000100
#define XSPI_CR_LSB_FIRST          0x00000200

/*****
 * Status register
 *****/
#define XSPI_SR                    0x64
/* status register */
#define XSPI_SR_RX_EMPTY 0x00000001 /* Receive Full */
#define XSPI_SR_RX_FULL  0x00000002 /* Receive Full */
#define XSPI_SR_TX_EMPTY 0x00000004 /* Transmit Empty */
#define XSPI_SR_TX_FULL  0x00000008 /* Transmit Full */

/*****
 * Data transmit
 *****/
#define XSPI_DTR                    0x68
/* Data transmit */

/*****
 * Data receive
 *****/
#define XSPI_DRR                    0x6C
/* Data receive */

/*****
 * Slave Select Register
 *****/
#define XSPI_SSR                    0x70

/*****
 * SPI Transmit FIFO Occupancy Register
 *****/
#define XSPI_TX_FIFO_OCY           0x74

/*****
 * SPI Receive FIFO Occupancy Register
 *****/
#define XSPI_RX_FIFO_OCY           0x78

#endif /* SPI_DRIVER_H */

```

L. MCP2515_FIFO.C

```

/*****
* File: MCP2515_fifo.c
* Created on: May 20, 2013
* Author: Tung-Hsun Tsou(Tony)
* Email: tung-hsun.tsou@wmich.edu
* Version: v1.0 First time can read...
*         v1.5 readable...
*
* Ref: 1.Microchip DS21801D MCP2515
*       www.microchip.com/downloads/en/devicedoc/21801d.pdf
*       2.DS570 LogiCORE IP XPS Serial Peripheral Interface (SPI) (v2.02a)
*       www.xilinx.com/support/documentation/ip_documentation/xps_spi.pdf
*       3.Embedded Linux Primer Second Edition
*       Christopher Hallinan Char 8
*       4.Linux Device Drivers 3e
*       Char 2,17
*       5.Platform Devices and Drivers
*       petalinux-v12.12-final-full/software/linux-2.6.x/Documentation/
*       driver-model/platform.txt
*       6. kernel/Documentation/driver-model/driver.txt
*       7. Linux can driver
*       /linux-2.6.x/drivers/net/can
*****/

#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/interrupt.h>
#include <linux/can.h>
#include <linux/can/core.h>
#include <linux/can/dev.h>
#include <linux/netdevice.h>

#include <linux/of_address.h>
#include <linux/of_device.h>
#include <linux/of_platform.h>
#include <linux/platform_device.h>

#include "MCP2515.h"
#include "SPI_driver_linux.h"

int SPI_base = 0;
unsigned char Buffer_R[13];
unsigned char read_value[13];
int Control=0;
unsigned int intc_status;
/* Standard module information, edit as appropriate */
MODULE_LICENSE("GPL");
MODULE_AUTHOR
("John Williams - PetaLogix Qld Pty Ltd <john.williams@petalogix.com>");
MODULE_AUTHOR
("Tung-Hsun Tsou - Western Michigan University <tung-hsun.tsou@wmich.edu>");

```

```
("MCP2515 - loadable module template generated by petalinux-new-module");
```

```
//define spi device
struct MCP2515_spi
```

```

/*****
* SPI initial
*****/

```

```

/*****
* Write Functions
*****/

```

```
#define chip_deselect iowrite32be(0xFF,SPI_base + XSPI_SSR)
```

```

void SPI_Write_once(int Buffer_W)//FIFO is enable in SPI controller
{
    iowrite32be(Buffer_W,SPI_base + XSPI_DTR);//put data to the transmitter
    while (!(ioread32(SPI_base +XSPI_SR_RX_EMPTY) == 0));
    ioread32(SPI_base + XSPI_DRR);
}
/*****
* Reset Instruction
*****/
void SPI_Reset(void)
{
    chip_select;
    SPI_Write_once(MCP2515_RESET);
    chip_deselect;
}
/*****
* Read Instruction
*****/
int SPI_ReadInst(int address)
{
    chip_select;
    iowrite32be(MCP2515_READ,SPI_base + XSPI_DTR);
    iowrite32be(address,SPI_base + XSPI_DTR);
    iowrite32be(0xFF,SPI_base + XSPI_DTR);
    while (!(ioread32(SPI_base +XSPI_RX_FIFO_OCY) == 2));
    ioread32(SPI_base + XSPI_DRR);
    ioread32(SPI_base + XSPI_DRR);
    Buffer_R[3] = ioread32(SPI_base + XSPI_DRR);
    chip_deselect;
    return Buffer_R[3];
}
/*****
* Write Instruction
*****/
void SPI_WriteInst(int address,int data)//FIFO is enable in SPI controller
{
    chip_select;
    iowrite32be(MCP2515_WRITE,SPI_base + XSPI_DTR);
    iowrite32be(address,SPI_base + XSPI_DTR);
    iowrite32be(data,SPI_base + XSPI_DTR);
    while (!(ioread32(SPI_base +XSPI_RX_FIFO_OCY) == 2));//occupancy minus one
    Control = ioread32(SPI_base +XSPI_CR);
    Control |= XSPI_CR_RxFIFO_Reset;
    iowrite32be(Control,SPI_base + XSPI_CR);
    chip_deselect;
}
/*****
* Bit Modify Instruction
*****/
void SPI_BitModifyInst(int address,int Mask,int Data)
{
    chip_select;
    iowrite32be(MCP2515_BITMODIFY,SPI_base + XSPI_DTR);
    iowrite32be(address,SPI_base + XSPI_DTR);
    iowrite32be(Mask,SPI_base + XSPI_DTR);

```

```

iowrite32be(Data,SPI_base + XSPI_DTR);
while (!(ioread32(SPI_base +XSPI_RX_FIFO_OCY) == 3));//occupancy minus one
Control = ioread32(SPI_base +XSPI_CR);
Control |= XSPI_CR_RxFIFO_Reset;
iowrite32be(Control,SPI_base + XSPI_CR);
chip_deselect;
}
/*****
* ERROR frame
*****/
void ERROR_frame(int err_status,int index)
{
    struct can_frame *can;
    chip_select;
    can->can_id = can->can_id & CAN_EFF_FLAG; //error flag
    iowrite32be(MCP2515_READ,SPI_base + XSPI_DTR);
    iowrite32be(MCP2515_EFLG,SPI_base + XSPI_DTR);
    iowrite32be(0xFF,SPI_base + XSPI_DTR);
    iowrite32be(MCP2515_READ,SPI_base + XSPI_DTR);
    iowrite32be(MCP2515_TEC,SPI_base + XSPI_DTR);
    iowrite32be(0xFF,SPI_base + XSPI_DTR);
    iowrite32be(0xFF,SPI_base + XSPI_DTR);

    while (!(ioread32(SPI_base +XSPI_RX_FIFO_OCY) == 6));

    ioread32(SPI_base + XSPI_DRR);
    ioread32(SPI_base + XSPI_DRR);
    ioread32(SPI_base + XSPI_DRR); //EFLAG counter
    ioread32(SPI_base + XSPI_DRR);
    ioread32(SPI_base + XSPI_DRR);
    ioread32(SPI_base + XSPI_DRR); //Transmit error counter
    ioread32(SPI_base + XSPI_DRR); //Receive Error Counter
    SPI_BitModifyInst (MCP2515_CANINTF, MCP2515_ERRIF, 0x00);
    chip_deselect;
}
/*****
* Standard data frame Read instruction
*****/
void Data_frame_Read(struct MCP2515_priv *priv ,int RX)
{
    //struct sk_buff *skb;
    struct can_frame *can;
    u8 buf[14];

    chip_select;
    iowrite32be(RX,SPI_base + XSPI_DTR);
    iowrite32be(0xFF,SPI_base + XSPI_DTR);//1
    iowrite32be(0xFF,SPI_base + XSPI_DTR);//2
    iowrite32be(0xFF,SPI_base + XSPI_DTR);//3
    iowrite32be(0xFF,SPI_base + XSPI_DTR);//4
    iowrite32be(0xFF,SPI_base + XSPI_DTR);//5
    iowrite32be(0xFF,SPI_base + XSPI_DTR);//6
    iowrite32be(0xFF,SPI_base + XSPI_DTR);//7
    iowrite32be(0xFF,SPI_base + XSPI_DTR);//8
    iowrite32be(0xFF,SPI_base + XSPI_DTR);//9
    iowrite32be(0xFF,SPI_base + XSPI_DTR);//10

```



```

iowrite32be(0xFF,SPI_base + XSPI_DTR);//11
iowrite32be(0xFF,SPI_base + XSPI_DTR);//12
iowrite32be(0xFF,SPI_base + XSPI_DTR);//13

while (!(ioread32(SPI_base +XSPI_RX_FIFO_OCY) == 13));
ioread32(SPI_base + XSPI_DRR);
can->can_id = ioread32(SPI_base + XSPI_DRR);
can->can_id = can->can_id << 3;
can->can_id = can->can_id | ioread32(SPI_base + XSPI_DRR) >> 5;

ioread32(SPI_base + XSPI_DRR);//3
ioread32(SPI_base + XSPI_DRR);//4
//can->can_id[29] = 0; //error flag clear
can->can_dlc = ioread32(SPI_base + XSPI_DRR);

can->data[0] = ioread32(SPI_base + XSPI_DRR);
can->data[1] = ioread32(SPI_base + XSPI_DRR);
can->data[2] = ioread32(SPI_base + XSPI_DRR);
can->data[3] = ioread32(SPI_base + XSPI_DRR);
can->data[4] = ioread32(SPI_base + XSPI_DRR);
can->data[5] = ioread32(SPI_base + XSPI_DRR);
can->data[6] = ioread32(SPI_base + XSPI_DRR);
can->data[7] = ioread32(SPI_base + XSPI_DRR);

//empty Transmit and Receive FIFO
Control = ioread32(SPI_base +XSPI_CR);
Control &=~(XSPI_CR_TxFIFO_Reset|XSPI_CR_RxFIFO_Reset);
iowrite32be(Control,SPI_base + XSPI_CR);
chip_deselect;
printk(KERN_ALERT "Receive : ID = %x , Data Length =%x , "
        "message = %c%c%c%c%c%c%c%c , %x%x%x%x%x%x%x%x\r\n",
        can->can_id,(can->can_dlc & 0x0F),
        can->data[0],can->data[1] ,
        can->data[2],can->data[3] ,
        can->data[4],can->data[5] ,
        can->data[6],can->data[7],
        can->data[0],can->data[1] ,
        can->data[2],can->data[3] ,
        can->data[4],can->data[5] ,
        can->data[6],can->data[7] );
}

void SPI_init_set(void)
{
    SPI_WriteInst (MCP2515_CNF1, SJW0);// length=2*Tq Tq=2/Fosc
    SPI_WriteInst (MCP2515_CNF2, BTLMODE | SAM | PRSEG10 | PRSEG0);//2
    SPI_WriteInst (MCP2515_CNF3, SOF_ENABLE | WAKFIL_ENABLE | PHSEG21);//1
    SPI_WriteInst (MCP2515_CANINTE, MCP2515_RX_INT);/* Rx interrupt */
    SPI_WriteInst (MCP2515_BFPCTRL, B0BFE | B1BFE );/* enable RX0BF RX1BF*/
    SPI_BitModifyInst (MCP2515_CANCTRL, MODE_MASK, MODE_LISTENONLY);
}

static irqreturn_t MCP2515_irq(int irq, void *dev_id)
{
    struct MCP2515_priv *priv = dev_id;
    struct MCP2515_spi *spi = priv->spi;

```

```

unsigned char INT_flag = 0;
Control = ioread32(SPI_base +XSPI_CR);
Control |= XSPI_CR_RxFIFO_Reset;
iowrite32be(Control,SPI_base + XSPI_CR);
while(!INT_flag)
{
    intc_status = SPI_ReadInst(MCP2515_CANINTF);
    if (intc_status & MCP2515_ERRIF)
        SPI_BitModifyInst (MCP2515_CANINTF, MCP2515_ERRIF, 0x00);
    else if(intc_status & MCP2515_RX0IF)
        Data_frame_Read(dev_id,MCP2515_READ_RXB0SIDH);
    else if(intc_status & MCP2515_RX1IF)
        Data_frame_Read(dev_id,MCP2515_READ_RXB1SIDH);
    else
        INT_flag = 1;
}
INT_flag = 0;

return IRQ_HANDLED;
}

static int __devinit MCP2515_probe(struct platform_device *pdev)
{
    struct resource *r_irq; /* Interrupt resources */
    struct resource *r_mem; /* IO mem resources */
    struct device *dev = &pdev->dev;
    struct MCP2515_spi *spi = NULL;
    struct net_device *net;

    int rc = 0,ret = 0;

    dev_info(dev, "Device Tree Probing\n");

    /* Get iospace for the device */
    r_mem = platform_get_resource(pdev, IORESOURCE_MEM, 0);
    if (!r_mem)
    {
        dev_err(dev, "invalid address\n");
        return -ENODEV;
    }

    spi = (struct MCP2515_spi *) kmalloc(sizeof(struct MCP2515_spi),
        GFP_KERNEL);
    if (!spi)
    {
        dev_err(dev, "Could not allocate MCP2515 device\n");
        return -ENOMEM;
    }

    dev_set_drvdata(dev, spi);

    spi->mem_start = r_mem->start;
    spi->mem_end = r_mem->end;

    if (!request_mem_region(spi->mem_start,spi->mem_end - spi->mem_start + 1,

```

```

        DRIVER_NAME))
    {
        dev_err(dev, "Couldn't lock memory region at %p\n",
            (void *)spi->mem_start);
        rc = -EBUSY;
        goto error1;
    }

    spi->base_addr = ioremap(spi->mem_start, spi->mem_end - spi->mem_start + 1);
    if (!spi->base_addr)
    {
        dev_err(dev, "MCP2515: Could not allocate iomem\n");
        rc = -EIO;
        goto error2;
    }
    SPI_base = spi->base_addr;
    SPI_init();
    SPI_Reset();
    SPI_init_set();
    printk(KERN_ALERT "base address : %x", spi->base_addr);

    /* Get IRQ for the device */
    r_irq = platform_get_resource(pdev, IORESOURCE_IRQ, 0);
    if (!r_irq)
    {
        dev_info(dev, "no IRQ found\n");
        dev_info(dev, "MCP2515 at 0x%08x mapped to 0x%08x\n",
            (unsigned int __force)spi->mem_start,
            (unsigned int __force)spi->base_addr);
        return 0;
    }
    spi->irq = r_irq->start;

    rc = request_irq(spi->irq, &MCP2515_irq, 0, DRIVER_NAME, spi);
    if (rc)
    {
        dev_err(dev, "testmodule: Could not allocate interrupt %d.\n", spi->irq);
        goto error3;
    }

    dev_info(dev, "MCP2515 at 0x%08x mapped to 0x%08x, irq=%d\n",
        (unsigned int __force)spi->mem_start,
        (unsigned int __force)spi->base_addr, spi->irq);

    return 0;
error3:
    free_irq(spi->irq, spi);
error2:
    release_mem_region(spi->mem_start, spi->mem_end - spi->mem_start + 1);
error1:
    kfree(spi);
    dev_set_drvdata(dev, NULL);
error_alloc:
    printk(KERN_ALERT "probe failed\n");
    return rc;

```

```

}

static int __devexit MCP2515_remove(struct platform_device *pdev)
{
    struct device *dev = &pdev->dev;
    struct MCP2515_spi *spi = dev_get_drvdata(dev);
    free_irq(spi->irq, spi);
    release_mem_region(spi->mem_start, spi->mem_end - spi->mem_start + 1);
    kfree(spi);
    dev_set_drvdata(dev, NULL);
    return 0;
}

#ifdef CONFIG_OF
static struct of_device_id MCP2515_of_match[] __devinitdata =
{
    { .compatible = "petalogix,MCP2515_fifo", },
    { /* end of list */ },
};
MODULE_DEVICE_TABLE(of, MCP2515_of_match);
#else
# define MCP2515_of_match
#endif

static struct platform_driver MCP2515_driver = {
    .driver = {
        .name = DRIVER_NAME,
        .owner = THIS_MODULE,
        .of_match_table = MCP2515_of_match,
    },
    .probe = MCP2515_probe,
    .remove = __devexit_p(MCP2515_remove),
};

static int __init MCP2515_init(void)
{
    printk(KERN_ALERT "Hello module world.\n");
    return platform_driver_register(&MCP2515_driver);
}

static void __exit MCP2515_exit(void)
{
    platform_driver_unregister(&MCP2515_driver);
    printk(KERN_ALERT "Goodbye module world.\n");
}

module_init(MCP2515_init);
module_exit(MCP2515_exit);

```

M. UNFINISHED NETWORK CAN DRIVER

```

/*****
* File: MCP2515.c
* Created on: May 20, 2013
* Author: Tung-Hsun Tsou(Tony)
* Email: tung-hsun.tsou@wmich.edu
* Version: v1.0 First time can read...
*         v1.5 readable...
*
* Reference: 1.Microchip DS21801D MCP2515
*            www.microchip.com/downloads/en/devicedoc/21801d.pdf
*            2.DS570 LogiCORE IP XPS Serial Peripheral Interface (SPI) (v2.02a)
*            www.xilinx.com/support/documentation/ip_documentation/xps_spi.pdf
*            3.Embedded Linux Primer Second Edition
*            Christopher Hallinan Char 8
*            4.Linux Device Drivers 3e
*            Char 2,17
*            5.Platform Devices and Drivers
*            petalinux-v12.12-final-full/software/linux-2.6.x/Documentation/
*            driver-model/platform.txt
*            6. kernel/Documentation/driver-model/driver.txt
*****/
/*
* CAN bus driver for Microchip 251x CAN Controller with SPI Interface
*
* MCP2510 support and bug fixes by Christian Pellegrin
* <chripell@evolware.org>
*
* Copyright 2009 Christian Pellegrin EVOL S.r.l.
*
* Copyright 2007 Raymarine UK, Ltd. All Rights Reserved.
* Written under contract by:
*   Chris Elston, Katalix Systems, Ltd.
*
* Based on Microchip mcp251x CAN controller driver written by
* David Vrabel, Copyright 2006 Arcom Control Systems Ltd.
*
* Based on CAN bus driver for the CCAN controller written by
* - Sascha Hauer, Marc Kleine-Budde, Pengutronix
* - Simon Kallweit, intefo AG
* Copyright 2007
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the version 2 of the GNU General Public License
* as published by the Free Software Foundation
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

```

```

*
*
*
* Your platform definition file should specify something like:
*
* static struct mcp251x_platform_data mcp251x_info = {
*     .oscillator_frequency = 8000000,
*     .board_specific_setup = &mcp251x_setup,
*     .power_enable = mcp251x_power_enable,
*     .transceiver_enable = NULL,
* };
*
* static struct spi_board_info spi_board_info[] = {
*     {
*         .modalias = "mcp2510",
*         // or "mcp2515" depending on your controller
*         .platform_data = &mcp251x_info,
*         .irq = IRQ_EINT13,
*         .max_speed_hz = 2*1000*1000,
*         .chip_select = 2,
*     },
* };
*
* Please see mcp251x.h for a description of the fields in
* struct mcp251x_platform_data.
*
*/
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/interrupt.h>
#include <linux/can.h>
#include <linux/can/core.h>
#include <linux/can/dev.h>
#include <linux/netdevice.h>

#include <linux/of_address.h>
#include <linux/of_device.h>
#include <linux/of_platform.h>
#include <linux/platform_device.h>

#include "MCP2515.h"
#include "SPI_driver_linux.h"

int SPI_base = 0;
unsigned char Buffer_R[13];
unsigned char read_value[13];
int Control=0;
unsigned int intc_status;
/* Standard module information, edit as appropriate */
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Chris Elston <celston@katalix.com>, "
              "Christian Pellegrin <chripell@evolware.org>");
MODULE_AUTHOR

```

```

    ("John Williams - PetaLogix Qld Pty Ltd <john.williams@petalogix.com>");
MODULE_AUTHOR
    ("Tung-Hsun Tsou - Western Michigan University <tung-hsun.tsou@wmich.edu>");
MODULE_DESCRIPTION
    ("mcp2515_can - loadable module template generated by petalinux-new-module");

#define DRIVER_NAME "mcp2515_can"
#define DEVICE_NAME "mcp2515"
/*
 * Buffer size required for the largest SPI transfer (i.e., reading a
 * frame)
 */
#define CAN_FRAME_MAX_DATA_LEN 8
#define SPI_TRANSFER_BUF_LEN    (6 + CAN_FRAME_MAX_DATA_LEN)
#define CAN_FRAME_MAX_BITS      128

#define TX_ECHO_SKB_MAX    1

//define device
struct spi_mcp2515 {
    int irq;
    unsigned long mem_start;
    unsigned long mem_end;
    void __iomem *base_addr;
};

static const struct can_bittiming_const mcp2515_bittiming_const = {
    .name = DEVICE_NAME,
    .tseg1_min = 3,
    .tseg1_max = 16,
    .tseg2_min = 2,
    .tseg2_max = 8,
    .sjw_max = 4,
    .brp_min = 1,
    .brp_max = 64,
    .brp_inc = 1,
};

struct mcp2515_priv {
    struct can_priv    can;
    struct net_device *net;
    struct platform_device *pdev;

    struct mutex mcp_lock; /* SPI device lock */

    u8 spi_tx_buf[16];
    u8 spi_rx_buf[16];

    struct sk_buff *tx_skb;
    int tx_len;

    struct workqueue_struct *wq;
    struct work_struct tx_work;
    struct work_struct restart_work;

    int force_quit;

```

```

        int after_suspend;
        int restart_tx;

};

static void mcp2515_clean(struct net_device *net)
{
    struct mcp2515_priv *priv = netdev_priv(net);

    if (priv->tx_skb || priv->tx_len)
        net->stats.tx_errors++;
    if (priv->tx_skb)
        dev_kfree_skb(priv->tx_skb);
    if (priv->tx_len)
        can_free_echo_skb(priv->net, 0);
    priv->tx_skb = NULL;
    priv->tx_len = 0;
}

#define chip_select iowrite32be(0x00,SPI_base + XSPI_SSR)

#define chip_deselect iowrite32be(0xFF,SPI_base + XSPI_SSR)

/*****
 * SPI initial
 *****/
static void SPI_init(void)
{
    printk(KERN_ALERT "SPI_base address : %x.\n",SPI_base);
    int Control = 0;
    //master mode & manual Slave
    Control = ioread32(SPI_base + XSPI_CR);
    Control |=((XSPI_CR_MASTER_MODE)|(XSPI_CR_Manual_SS_MODE));
    iowrite32be(Control,SPI_base+XSPI_CR);
    //Slave Select flag
    iowrite32be(0xFF,SPI_base + XSPI_SSR);
    // asserted when core configured in slave mode and selected
    // by external SPI master.
    //CPLO&CPHA
    Control &=~(XSPI_CR_CPLO_MODE|XSPI_CR_CPHA_MODE);
    //enable device
    Control &=~XSPI_CR_MASTER_TRANS_INHIBIT_MODE;
    Control |= XSPI_CR_ENABLE_MODE;
    iowrite32be(Control,SPI_base + XSPI_CR);
    //empty Transmit and Receive FIFO
    Control = ioread32(SPI_base +XSPI_CR);
    Control &=~(XSPI_CR_TxFIFO_Reset|XSPI_CR_RxFIFO_Reset);
    iowrite32be(Control,SPI_base + XSPI_CR);
    iowrite32be(0xFF,SPI_base + XSPI_SSR);
    printk(KERN_ALERT "End SPI_init");
}

static int mcp2515_spi_trans_8(struct platform_device *pdev, int len)
{
    struct mcp2515_priv *priv = dev_get_drvdata(&pdev->dev);

```



```

        chip_select;
        iowrite32be(priv->spi_tx_buf[0],SPI_base + XSPI_DTR);
        iowrite32be(priv->spi_tx_buf[1],SPI_base + XSPI_DTR);
        iowrite32be(priv->spi_tx_buf[2],SPI_base + XSPI_DTR);
        iowrite32be(priv->spi_tx_buf[3],SPI_base + XSPI_DTR);
        iowrite32be(priv->spi_tx_buf[4],SPI_base + XSPI_DTR);
        iowrite32be(priv->spi_tx_buf[5],SPI_base + XSPI_DTR);
        iowrite32be(priv->spi_tx_buf[6],SPI_base + XSPI_DTR);
        iowrite32be(priv->spi_tx_buf[7],SPI_base + XSPI_DTR);

        while (!(ioread32(SPI_base +XSPI_RX_FIFO_OCY) == 7));

        priv->spi_rx_buf[0] = ioread32(SPI_base + XSPI_DRR);
        priv->spi_rx_buf[1] = ioread32(SPI_base + XSPI_DRR);
        priv->spi_rx_buf[2] = ioread32(SPI_base + XSPI_DRR);
        priv->spi_rx_buf[3] = ioread32(SPI_base + XSPI_DRR);
        priv->spi_rx_buf[4] = ioread32(SPI_base + XSPI_DRR);
        priv->spi_rx_buf[5] = ioread32(SPI_base + XSPI_DRR);
        priv->spi_rx_buf[6] = ioread32(SPI_base + XSPI_DRR);
        priv->spi_rx_buf[7] = ioread32(SPI_base + XSPI_DRR);
        chip_deselect;
    }

static int mcp2515_spi_trans_1(struct platform_device *pdev, int len)
{
    struct mcp2515_priv *priv = dev_get_drvdata(&pdev->dev);

    chip_select;
    iowrite32be(priv->spi_tx_buf[0],SPI_base + XSPI_DTR);

    while (!(ioread32(SPI_base +XSPI_SR_RX_EMPTY) == 0));

    priv->spi_rx_buf[0] = ioread32(SPI_base + XSPI_DRR);
    chip_deselect;
}

/*****
* Read Instruction
*****/
static u8 mcp2515_read_reg(struct platform_device *pdev, uint8_t reg)
{
    struct mcp2515_priv *priv = dev_get_drvdata(&pdev->dev);
    u8 val = 0;

    chip_select;
    iowrite32be(INSTRUCTION_READ,SPI_base + XSPI_DTR);
    iowrite32be(reg,SPI_base + XSPI_DTR);
    iowrite32be(0xFF,SPI_base + XSPI_DTR);
    while (!(ioread32(SPI_base +XSPI_RX_FIFO_OCY) == 2));
    ioread32(SPI_base + XSPI_DRR);
    ioread32(SPI_base + XSPI_DRR);
    Buffer_R[3] = ioread32(SPI_base + XSPI_DRR);
    chip_deselect;

    return Buffer_R[3];
}

```

```

static void mcp2515_read_2regs(struct platform_device *pdev, uint8_t reg,
                             uint8_t *v1, uint8_t *v2)
{
    chip_select;
    iowrite32be(INSTRUCTION_READ,SPI_base + XSPI_DTR);
    iowrite32be(reg,SPI_base + XSPI_DTR);
    iowrite32be(0xFF,SPI_base + XSPI_DTR);
    iowrite32be(0xFF,SPI_base + XSPI_DTR);
    while (!(ioread32(SPI_base +XSPI_RX_FIFO_OCY) == 3));
    ioread32(SPI_base + XSPI_DRR);
    ioread32(SPI_base + XSPI_DRR);
    Buffer_R[2] = ioread32(SPI_base + XSPI_DRR);
    Buffer_R[3] = ioread32(SPI_base + XSPI_DRR);
    chip_deselect;

    *v1 = Buffer_R[2];
    *v2 = Buffer_R[3];
}

/*****
* Write Functions
*****/

static void mcp2515_write_reg(struct platform_device *pdev, u8 reg, uint8_t val)
{
    chip_select;
    iowrite32be(INSTRUCTION_WRITE,SPI_base + XSPI_DTR);
    iowrite32be(reg,SPI_base + XSPI_DTR);
    iowrite32be(val,SPI_base + XSPI_DTR);
    while (!(ioread32(SPI_base +XSPI_RX_FIFO_OCY) == 2));//occupancy minus one
    Control = ioread32(SPI_base +XSPI_CR);
    Control |= XSPI_CR_RxFIFO_Reset;
    iowrite32be(Control,SPI_base + XSPI_CR);
    chip_deselect;
}

static void mcp2515_write_bits(struct platform_device *pdev, u8 reg,
                              u8 mask, uint8_t val)
{
    chip_select;
    iowrite32be(INSTRUCTION_BIT_MODIFY,SPI_base + XSPI_DTR);
    iowrite32be(reg,SPI_base + XSPI_DTR);
    iowrite32be(mask,SPI_base + XSPI_DTR);
    iowrite32be(val,SPI_base + XSPI_DTR);

    while (!(ioread32(SPI_base +XSPI_RX_FIFO_OCY) == 3));//occupancy minus one
    Control = ioread32(SPI_base +XSPI_CR);
    Control |= XSPI_CR_RxFIFO_Reset;
    iowrite32be(Control,SPI_base + XSPI_CR);
    chip_deselect;
}

static void mcp2515_hw_tx_frame(struct platform_device *pdev, u8 *buf,
                                int len, int tx_buf_idx)

```

```

{
    struct mcp2515_priv *priv = dev_get_drvdata(&pdev->dev);

    memcpy(priv->spi_tx_buf, buf, TXBDAT_OFF + len);
    mcp2515_spi_trans_8(pdev, TXBDAT_OFF + len);
}

static void mcp2515_hw_tx(struct platform_device *pdev, struct can_frame *frame,
                        int tx_buf_idx)
{
    struct mcp2515_priv *priv = dev_get_drvdata(&pdev->dev);
    u32 sid, eid, exide, rtr;
    u8 buf[SPI_TRANSFER_BUF_LEN];

    exide = (frame->can_id & CAN_EFF_FLAG) ? 1 : 0; /* Extended ID Enable */
    if (exide)
        sid = (frame->can_id & CAN_EFF_MASK) >> 18;
    else
        sid = frame->can_id & CAN_SFF_MASK; /* Standard ID */
    eid = frame->can_id & CAN_EFF_MASK; /* Extended ID */
    rtr = (frame->can_id & CAN_RTR_FLAG) ? 1 : 0; /* Remote transmission */

    buf[TXBCTRL_OFF] = INSTRUCTION_LOAD_TXB(tx_buf_idx);
    buf[TXBSIDH_OFF] = sid >> SIDH_SHIFT;
    buf[TXBSIDL_OFF] = ((sid & SIDL_SID_MASK) << SIDL_SID_SHIFT) |
        (exide << SIDL_EXIDE_SHIFT) |
        ((eid >> SIDL_EID_SHIFT) & SIDL_EID_MASK);
    buf[TXBEID8_OFF] = GET_BYTE(eid, 1);
    buf[TXBEID0_OFF] = GET_BYTE(eid, 0);
    buf[TXBDLC_OFF] = (rtr << DLC_RTR_SHIFT) | frame->can_dlc;
    memcpy(buf + TXBDAT_OFF, frame->data, frame->can_dlc);
    mcp2515_hw_tx_frame(pdev, buf, frame->can_dlc, tx_buf_idx);

    /* use INSTRspiUNCTION_RTS, to avoid "repeated frame problem" */
    priv->spi_tx_buf[0] = INSTRUCTION_RTS(1 << tx_buf_idx);
    mcp2515_spi_trans_1(priv->pdev, 1);
}

/*****
 * Standard data frame Read instruction
 *****/
static void mcp2515_hw_rx_frame(struct platform_device *pdev, u8 *buf,
                              int buf_idx)
{
    struct mcp2515_priv *priv = dev_get_drvdata(&pdev->dev);

    chip_select;
    iowrite32be(INSTRUCTION_READ_RXB(buf_idx), SPI_base + XSPI_DTR); //0
    iowrite32be(0xFF, SPI_base + XSPI_DTR); //1
    iowrite32be(0xFF, SPI_base + XSPI_DTR); //2
    iowrite32be(0xFF, SPI_base + XSPI_DTR); //3
    iowrite32be(0xFF, SPI_base + XSPI_DTR); //4
    iowrite32be(0xFF, SPI_base + XSPI_DTR); //5
    iowrite32be(0xFF, SPI_base + XSPI_DTR); //6
    iowrite32be(0xFF, SPI_base + XSPI_DTR); //7

```

```

iowrite32be(0xFF,SPI_base + XSPI_DTR);//8
iowrite32be(0xFF,SPI_base + XSPI_DTR);//9
iowrite32be(0xFF,SPI_base + XSPI_DTR);//10
iowrite32be(0xFF,SPI_base + XSPI_DTR);//11
iowrite32be(0xFF,SPI_base + XSPI_DTR);//12
iowrite32be(0xFF,SPI_base + XSPI_DTR);//13

while (!(ioread32(SPI_base +XSPI_RX_FIFO_OCY) == 13));
buf[0] = ioread32(SPI_base + XSPI_DRR);
buf[1] = ioread32(SPI_base + XSPI_DRR);
buf[2] = ioread32(SPI_base + XSPI_DRR);
buf[3] = ioread32(SPI_base + XSPI_DRR);
buf[4] = ioread32(SPI_base + XSPI_DRR);
buf[5] = ioread32(SPI_base + XSPI_DRR);
buf[6] = ioread32(SPI_base + XSPI_DRR);
buf[7] = ioread32(SPI_base + XSPI_DRR);
buf[8] = ioread32(SPI_base + XSPI_DRR);
buf[9] = ioread32(SPI_base + XSPI_DRR);
buf[10] = ioread32(SPI_base + XSPI_DRR);
buf[11] = ioread32(SPI_base + XSPI_DRR);
buf[12] = ioread32(SPI_base + XSPI_DRR);
buf[13] = ioread32(SPI_base + XSPI_DRR);

memcpy(buf, priv->spi_rx_buf, SPI_TRANSFER_BUF_LEN);
//empty Transmit and Receive FIFO
Control = ioread32(SPI_base +XSPI_CR);
Control &=~(XSPI_CR_TxFIFO_Reset|XSPI_CR_RxFIFO_Reset);
iowrite32be(Control,SPI_base + XSPI_CR);
chip_deselect;
}

static void mcp2515_hw_rx(struct platform_device *pdev, int buf_idx)
{
    printk("mcp2515_hw_rx_alloc_can_skb1\n");
    struct mcp2515_priv *priv = dev_get_drvdata(&pdev->dev);
    printk("mcp2515_hw_rx_alloc_can_skb2\n");
    struct sk_buff *skb;
    printk("mcp2515_hw_rx_alloc_can_skb3\n");

    struct can_frame *frame;
    printk("mcp2515_hw_rx_alloc_can_skb4\n");

    u8 buf[SPI_TRANSFER_BUF_LEN];
    printk("mcp2515_hw_rx_alloc_can_skb\n");

    skb = alloc_can_skb(priv->net, &frame);
    if (!skb) {
        dev_err(&pdev->dev, "cannot allocate RX skb\n");
        priv->net->stats.rx_dropped++;
        return;
    }
    printk("mcp2515_hw_rx_frame\n");
    mcp2515_hw_rx_frame(pdev, buf, buf_idx);
    printk("mcp2515_hw_rx_frame2\n");
    if (buf[RXBSIDL_OFF] & RXBSIDL_IDE) {
        /* Extended ID format */

```

```

        frame->can_id = CAN_EFF_FLAG;
        frame->can_id |=
            /* Extended ID part */
            SET_BYTE(buf[RXBSIDL_OFF] & RXBSIDL_EID, 2) |
            SET_BYTE(buf[RXBEID8_OFF], 1) |
            SET_BYTE(buf[RXBEID0_OFF], 0) |
            /* Standard ID part */
            (((buf[RXBSIDH_OFF] << RXBSIDH_SHIFT) |
              (buf[RXBSIDL_OFF] >> RXBSIDL_SHIFT)) << 18);
        /* Remote transmission request */
        if (buf[RXBDLC_OFF] & RXBDLC_RTR)
            frame->can_id |= CAN_RTR_FLAG;
    } else {
        /* Standard ID format */
        frame->can_id =
            (buf[RXBSIDH_OFF] << RXBSIDH_SHIFT) |
            (buf[RXBSIDL_OFF] >> RXBSIDL_SHIFT);
        if (buf[RXBSIDL_OFF] & RXBSIDL_SRR)
            frame->can_id |= CAN_RTR_FLAG;
    }
    /* Data length */
    frame->can_dlc = get_can_dlc(buf[RXBDLC_OFF] & RXBDLC_LEN_MASK);
    memcpy(frame->data, buf + RXBDAT_OFF, frame->can_dlc);

    priv->net->stats.rx_packets++;
    priv->net->stats.rx_bytes += frame->can_dlc;
    netif_rx_ni(skb);
}

static void mcp2515_hw_sleep(struct platform_device *pdev)
{
    mcp2515_write_reg(pdev, CANCTRL, CANCTRL_REQOP_SLEEP);
}

static netdev_tx_t mcp2515_hard_start_xmit(struct sk_buff *skb,
                                           struct net_device *net)
{
    struct mcp2515_priv *priv = netdev_priv(net);
    struct platform_device *pdev = priv->pdev;

    if (priv->tx_skb || priv->tx_len) {
        dev_warn(&pdev->dev, "hard_xmit called while tx busy\n");
        return NETDEV_TX_BUSY;
    }

    if (can_dropped_invalid_skb(net, skb))
        return NETDEV_TX_OK;

    netif_stop_queue(net);
    priv->tx_skb = skb;
    queue_work(priv->wq, &priv->tx_work);

    return NETDEV_TX_OK;
}

static int mcp2515_do_set_mode(struct net_device *net, enum can_mode mode)

```

```

{
    struct mcp2515_priv *priv = netdev_priv(net);

    switch (mode) {
    case CAN_MODE_START:
        mcp2515_clean(net);
        /* We have to delay work since SPI I/O may sleep */
        priv->can.state = CAN_STATE_ERROR_ACTIVE;
        priv->restart_tx = 1;
        if (priv->can.restart_ms == 0)
            priv->after_suspend = AFTER_SUSPEND_RESTART;
        queue_work(priv->wq, &priv->restart_work);
        break;
    default:
        return -EOPNOTSUPP;
    }

    return 0;
}

static int mcp2515_set_normal_mode(struct platform_device *pdev)
{
    struct mcp2515_priv *priv = dev_get_drvdata(&pdev->dev);
    unsigned long timeout;

    /* Enable interrupts */
    mcp2515_write_reg(pdev, CANINTE,
                      CANINTE_ERRIE | CANINTE_TX2IE | CANINTE_TX1IE |
                      CANINTE_TX0IE | CANINTE_RX1IE | CANINTE_RX0IE);

    if (priv->can.ctrlmode & CAN_CTRLMODE_LOOPBACK) {
        /* Put device into loopback mode */
        mcp2515_write_reg(pdev, CANCTRL, CANCTRL_REQOP_LOOPBACK);
    } else if (priv->can.ctrlmode & CAN_CTRLMODE_LISTENONLY) {
        /* Put device into listen-only mode */
        mcp2515_write_reg(pdev, CANCTRL, CANCTRL_REQOP_LISTEN_ONLY);
    } else {
        /* Put device into normal mode */
        mcp2515_write_reg(pdev, CANCTRL, CANCTRL_REQOP_NORMAL);

        /* Wait for the device to enter normal mode */
        timeout = jiffies + HZ;
        while (mcp2515_read_reg(pdev, CANSTAT) & CANCTRL_REQOP_MASK) {
            schedule();
            if (time_after(jiffies, timeout)) {
                dev_err(&pdev->dev, "MCP251x didn't"
                        " enter in normal mode\n");
                return -EBUSY;
            }
        }
    }
    priv->can.state = CAN_STATE_ERROR_ACTIVE;
    return 0;
}

static int mcp2515_do_set_bittiming(struct net_device *net)

```

```

{
    struct mcp2515_priv *priv = netdev_priv(net);
    struct can_bittiming *bt = &priv->can.bittiming;
    struct platform_device *pdev = priv->pdev;

    mcp2515_write_reg(pdev, CNF1, ((bt->sjw - 1) << CNF1_SJW_SHIFT) |
        (bt->brp - 1));
    mcp2515_write_reg(pdev, CNF2, CNF2_BTLMODE |
        (priv->can.ctrlmode & CAN_CTRLMODE_3_SAMPLES ?
        CNF2_SAM : 0) |
        ((bt->phase_seg1 - 1) << CNF2_PS1_SHIFT) |
        (bt->prop_seg - 1));
    mcp2515_write_bits(pdev, CNF3, CNF3_PHSEG2_MASK,
        (bt->phase_seg2 - 1));
    dev_info(&pdev->dev, "CNF: 0x%02x 0x%02x 0x%02x\n",
        mcp2515_read_reg(pdev, CNF1),
        mcp2515_read_reg(pdev, CNF2),
        mcp2515_read_reg(pdev, CNF3));

    return 0;
}

static int mcp2515_setup(struct net_device *net, struct mcp2515_priv *priv,
    struct platform_device *pdev)
{
    mcp2515_do_set_bittiming(net);

    mcp2515_write_reg(pdev, RXBCTRL(0),
        RXBCTRL_BUKT | RXBCTRL_RXM0 | RXBCTRL_RXM1);
    mcp2515_write_reg(pdev, RXBCTRL(1),
        RXBCTRL_RXM0 | RXBCTRL_RXM1);

    return 0;
}

static SPI_Reset(void)
{
    chip_select;
    iowrite32be(MCP2515_RESET, SPI_base + XSPI_DTR); //put data to the transmitter
    while (!(ioread32(SPI_base + XSPI_SR_RX_EMPTY) == 0));
    ioread32(SPI_base + XSPI_DRR);
    chip_deselect;
}

static int mcp2515_hw_reset(struct platform_device *pdev)
{
    struct mcp2515_priv *priv = dev_get_drvdata(&pdev->dev);
    int ret;
    unsigned long timeout;

    SPI_Reset();

    /* Wait for reset to finish */
    timeout = jiffies + HZ;
    mdelay(10);
    while ((mcp2515_read_reg(pdev, CANSTAT) & CANCTRL_REQOP_MASK)
        != CANCTRL_REQOP_CONF) {

```

```

        schedule();
        if (time_after(jiffies, timeout)) {
            dev_err(&pdev->dev, "MCP2515 didn't"
                    " enter in conf mode after reset\n");
            return -EBUSY;
        }
    }
    return 0;
}

static int mcp2515_hw_probe(struct platform_device *pdev)
{
    int st1, st2;

    mcp2515_hw_reset(pdev);

    /*
     * Please note that these are "magic values" based on after
     * reset defaults taken from data sheet which allows us to see
     * if we really have a chip on the bus (we avoid common all
     * zeroes or all ones situations)
     */
    st1 = mcp2515_read_reg(pdev, CANSTAT) & 0xEE;
    st2 = mcp2515_read_reg(pdev, CANCTRL) & 0x17;

    dev_dbg(&pdev->dev, "CANSTAT 0x%02x CANCTRL 0x%02x\n", st1, st2);

    /* Check for power up default values */
    return (st1 == 0x80 && st2 == 0x07) ? 1 : 0;
}

static void mcp2515_open_clean(struct net_device *net)
{
    struct mcp2515_priv *priv = netdev_priv(net);
    struct platform_device *pdev = priv->pdev;

    mcp2515_hw_sleep(pdev);

    close_candev(net);
}

static int mcp2515_stop(struct net_device *net)
{
    struct mcp2515_priv *priv = netdev_priv(net);
    struct platform_device *pdev = priv->pdev;

    close_candev(net);

    priv->force_quit = 1;
    destroy_workqueue(priv->wq);
    priv->wq = NULL;

    //mutex_lock(&priv->mcp_lock);

    /* Disable and clear pending interrupts */

```



```

mcp2515_write_reg(pdev, CANINTE, 0x00);
mcp2515_write_reg(pdev, CANINTF, 0x00);

mcp2515_write_reg(pdev, TXBCTRL(0), 0);
mcp2515_clean(net);

mcp2515_hw_sleep(pdev);

priv->can.state = CAN_STATE_STOPPED;

//mutex_unlock(&priv->mcp_lock);

return 0;
}

static void mcp2515_error_skb(struct net_device *net, int can_id, int data1)
{
    struct sk_buff *skb;
    struct can_frame *frame;

    skb = alloc_can_err_skb(net, &frame);
    if (skb) {
        frame->can_id |= can_id;
        frame->data[1] = data1;
        netif_rx_ni(skb);
    } else {
        netdev_err(net, "cannot allocate error skb\n");
    }
}

static void mcp2515_tx_work_handler(struct work_struct *ws)
{
    struct mcp2515_priv *priv = container_of(ws, struct mcp2515_priv,
                                              tx_work);

    struct platform_device *pdev = priv->pdev;
    struct net_device *net = priv->net;
    struct can_frame *frame;

    //mutex_lock(&priv->mcp_lock);
    if (priv->tx_skb) {
        if (priv->can.state == CAN_STATE_BUS_OFF) {
            mcp2515_clean(net);
        } else {
            frame = (struct can_frame *)priv->tx_skb->data;

            if (frame->can_dlc > CAN_FRAME_MAX_DATA_LEN)
                frame->can_dlc = CAN_FRAME_MAX_DATA_LEN;
            mcp2515_hw_tx(pdev, frame, 0);
            priv->tx_len = 1 + frame->can_dlc;
            can_put_echo_skb(priv->tx_skb, net, 0);
            priv->tx_skb = NULL;
        }
    }
    //mutex_unlock(&priv->mcp_lock);
}

```

```

static void mcp2515_restart_work_handler(struct work_struct *ws)
{
    struct mcp2515_priv *priv = container_of(ws, struct mcp2515_priv,
                                              restart_work);

    struct platform_device *pdev = priv->pdev;
    struct net_device *net = priv->net;

    //mutex_lock(&priv->mcp_lock);
    if (priv->after_suspend) {
        mdelay(10);
        mcp2515_hw_reset(pdev);
        mcp2515_setup(net, priv, pdev);
        if (priv->after_suspend & AFTER_SUSPEND_RESTART) {
            mcp2515_set_normal_mode(pdev);
        } else if (priv->after_suspend & AFTER_SUSPEND_UP) {
            netif_device_attach(net);
            mcp2515_clean(net);
            mcp2515_set_normal_mode(pdev);
            netif_wake_queue(net);
        } else {
            mcp2515_hw_sleep(pdev);
        }
        priv->after_suspend = 0;
        priv->force_quit = 0;
    }

    if (priv->restart_tx) {
        priv->restart_tx = 0;
        mcp2515_write_reg(pdev, TXBCTRL(0), 0);
        mcp2515_clean(net);
        netif_wake_queue(net);
        mcp2515_error_skb(net, CAN_ERR_RESTARTED, 0);
    }
    //mutex_unlock(&priv->mcp_lock);
}

static irqreturn_t mcp2515_can_irq(int irq, void *dev_id)
{
    struct mcp2515_priv *priv = dev_id;
    struct platform_device *pdev = priv->pdev;
    struct net_device *net = priv->net;

    //mutex_lock(&priv->mcp_lock);
    priv->force_quit = 0;
    while (!priv->force_quit) {
        enum can_state new_state;
        u8 intf, eflag;
        u8 clear_intf = 0;
        int can_id = 0, data1 = 0;

        mcp2515_read_2regs(pdev, CANINTF, &intf, &eflag);
        printk("intf: eflag:\n");
        /* mask out flags we don't care about */
        intf &= CANINTF_RX | CANINTF_TX | CANINTF_ERR;
        printk("masked\n");
        /* receive buffer 0 */
    }
    //mutex_unlock(&priv->mcp_lock);
}

```

```

if (intf & CANINTF_RX0IF) {
    printk("intf & CANINTF_RX0IF\n");
    mcp2515_hw_rx(pdev, 0);
}

/* receive buffer 1 */
if (intf & CANINTF_RX1IF) {
    mcp2515_hw_rx(pdev, 1);
}

/* any error or tx interrupt we need to clear? */
if (intf & (CANINTF_ERR | CANINTF_TX))
    clear_intf |= intf & (CANINTF_ERR | CANINTF_TX);
if (clear_intf)
    mcp2515_write_bits(pdev, CANINTF, clear_intf, 0x00);

if (eflag)
    mcp2515_write_bits(pdev, EFLG, eflag, 0x00);

/* Update can state */
if (eflag & EFLG_TXBO) {
    new_state = CAN_STATE_BUS_OFF;
    can_id |= CAN_ERR_BUSOFF;
} else if (eflag & EFLG_TXEP) {
    new_state = CAN_STATE_ERROR_PASSIVE;
    can_id |= CAN_ERR_CRTL;
    data1 |= CAN_ERR_CRTL_TX_PASSIVE;
} else if (eflag & EFLG_RXEP) {
    new_state = CAN_STATE_ERROR_PASSIVE;
    can_id |= CAN_ERR_CRTL;
    data1 |= CAN_ERR_CRTL_RX_PASSIVE;
} else if (eflag & EFLG_TXWAR) {
    new_state = CAN_STATE_ERROR_WARNING;
    can_id |= CAN_ERR_CRTL;
    data1 |= CAN_ERR_CRTL_TX_WARNING;
} else if (eflag & EFLG_RXWAR) {
    new_state = CAN_STATE_ERROR_WARNING;
    can_id |= CAN_ERR_CRTL;
    data1 |= CAN_ERR_CRTL_RX_WARNING;
} else {
    new_state = CAN_STATE_ERROR_ACTIVE;
}

/* Update can state statistics */
switch (priv->can.state) {
case CAN_STATE_ERROR_ACTIVE:
    if (new_state >= CAN_STATE_ERROR_WARNING &&
        new_state <= CAN_STATE_BUS_OFF)
        priv->can.can_stats.error_warning++;
case CAN_STATE_ERROR_WARNING: /* fallthrough */
    if (new_state >= CAN_STATE_ERROR_PASSIVE &&
        new_state <= CAN_STATE_BUS_OFF)
        priv->can.can_stats.error_passive++;
    break;
default:
    break;
}

```

```

    }
    priv->can.state = new_state;

    if (intf & CANINTF_ERRIF) {
        /* Handle overflow counters */
        if (eflag & (EFLG_RX0OVR | EFLG_RX1OVR)) {
            if (eflag & EFLG_RX0OVR) {
                net->stats.rx_over_errors++;
                net->stats.rx_errors++;
            }
            if (eflag & EFLG_RX1OVR) {
                net->stats.rx_over_errors++;
                net->stats.rx_errors++;
            }
            can_id |= CAN_ERR_CRTL;
            data1 |= CAN_ERR_CRTL_RX_OVERFLOW;
        }
        mcp2515_error_skb(net, can_id, data1);
    }

    if (priv->can.state == CAN_STATE_BUS_OFF) {
        if (priv->can.restart_ms == 0) {
            priv->force_quit = 1;
            can_bus_off(net);
            mcp2515_hw_sleep(pdev);
            break;
        }
    }

    if (intf == 0)
        break;

    if (intf & CANINTF_TX) {
        net->stats.tx_packets++;
        net->stats.tx_bytes += priv->tx_len - 1;
        if (priv->tx_len) {
            can_get_echo_skb(net, 0);
            priv->tx_len = 0;
        }
        netif_wake_queue(net);
    }

}
//mutex_unlock(&priv->mcp_lock);
return IRQ_HANDLED;
}

void SPI_init_set(struct platform_device *pdev)
{
    mcp2515_write_reg(pdev, MCP2515_CNF1, SJW0); // length=2*Tq Tq=2/Fosc
    mcp2515_write_reg(pdev, MCP2515_CNF2, BTLMODE | SAM | PRSEG10 | PRSEG0); //2
    mcp2515_write_reg(pdev, MCP2515_CNF3, SOF_ENABLE | WAKFIL_ENABLE |
PHSEG21); //1
    mcp2515_write_reg(pdev, MCP2515_CANINTE, MCP2515_RX_INT); /* Rx interrupt */
    mcp2515_write_reg(pdev, MCP2515_BFPCTRL, B0BFE | B1BFE); /* enable RX0BF RX1BF */
    mcp2515_write_bits(pdev, MCP2515_CANCTRL, MODE_MASK, MODE_LISTENONLY);

```

```

}
/*
static irqreturn_t mcp2515_can_irq(int irq, void *lp)
{
    struct mcp2515_priv *priv = lp;
    struct platform_device *pdev = priv->spi;
    struct net_device *net = priv->net;

    unsigned char INT_flag = 0;
    Control = ioread32(SPI_base +XSPI_CR);
    Control |= XSPI_CR_RxFIFO_Reset;
    iowrite32be(Control,SPI_base + XSPI_CR);
    while(!INT_flag)
    {
        intc_status = SPI_ReadInst(MCP2515_CANINTF);

        if (intc_status & MCP2515_ERRIF)
        {
            SPI_BitModifyInst (MCP2515_CANINTF, MCP2515_ERRIF, 0x00);
        }
        else if(intc_status & MCP2515_RX0IF)
        {
            Data_frame_Read(pdev,MCP2515_READ_RXB0SIDH);
        }
        else if(intc_status & MCP2515_RX1IF)
        {
            Data_frame_Read(pdev,MCP2515_READ_RXB1SIDH);
        }
        else
        {
            INT_flag = 1;
        }
    }
    INT_flag = 0;

    return IRQ_HANDLED;
}
*/
static int mcp2515_open(struct net_device *net)
{
    struct mcp2515_priv *priv = netdev_priv(net);
    struct platform_device *pdev = &priv->pdev;
    int ret;

    ret = open_candev(net);
    if (ret) {
        dev_err(&priv->pdev, "unable to set initial baudrate!\n");
        return ret;
    }

    //mutex_lock(&priv->mcp_lock);

    priv->force_quit = 0;
    priv->tx_skb = NULL;
    priv->tx_len = 0;

```

```

priv->wq = create_freeable_workqueue("mcp2515_wq");
//INIT_WORK(&priv->tx_work, mcp2515_tx_work_handler);
//INIT_WORK(&priv->restart_work, mcp2515_restart_work_handler);

SPI_Reset();
SPI_init_set(pdev);
netif_wake_queue(net);

open_unlock:
    //mutex_unlock(&priv->mcp_lock);
    return ret;
}

static const struct net_device_ops mcp2515_netdev_ops = {
    .ndo_open = mcp2515_open,
    .ndo_stop = mcp2515_stop,
    .ndo_start_xmit = mcp2515_hard_start_xmit,
};

static int __devinit mcp2515_can_probe(struct platform_device *pdev)
{
    struct resource *r_irq; /* Interrupt resources */
    struct resource *r_mem; /* IO mem resources */
    struct device *dev = &pdev->dev;
    struct spi_mcp2515 *lp = NULL;
    struct net_device *net;
    struct mcp2515_priv *priv;

    int rc = 0, ret = 0;

    dev_info(dev, "Device Tree Probing\n");

    /* Get iospace for the device */
    r_mem = platform_get_resource(pdev, IORESOURCE_MEM, 0);
    if (!r_mem) {
        dev_err(dev, "invalid address\n");
        return -ENODEV;
    }

    lp = (struct spi_mcp2515 *) kmalloc(sizeof(struct spi_mcp2515),
        GFP_KERNEL);
    if (!lp) {
        dev_err(dev, "Could not allocate mcp2515_can device\n");
        return -ENOMEM;
    }

    dev_set_drvdata(dev, lp);

    lp->mem_start = r_mem->start;
    lp->mem_end = r_mem->end;

    if (!request_mem_region(lp->mem_start,
        lp->mem_end - lp->mem_start + 1,
        DRIVER_NAME)) {

```

```

        dev_err(dev, "Couldn't lock memory region at %p\n",
                (void *)lp->mem_start);
        rc = -EBUSY;
        goto error1;
    }

    lp->base_addr = ioremap(lp->mem_start, lp->mem_end - lp->mem_start + 1);
    if (!lp->base_addr) {
        dev_err(dev, "mcp2515_can: Could not allocate iomem\n");
        rc = -EIO;
        goto error2;
    }
    SPI_base = lp->base_addr;
    SPI_init();
    SPI_Reset();
    SPI_init_set(pdev);
    printk(KERN_ALERT "base address : %x\n", lp->base_addr);

    net = alloc_candev(sizeof(struct mcp2515_priv), 1);
    if (!net) {
        ret = -ENOMEM;
        goto error_alloc;
    }
    printk("alloc_candev\n");
    net->netdev_ops = &mcp2515_netdev_ops;
    net->flags |= IFF_ECHO;
    printk("net->flags\n");
    //mutex_init(&priv->mcp_lock);
    printk("mutex_init\n");
    SET_NETDEV_DEV(net, dev);
    printk("Dev\n");

    /* Get IRQ for the device */
    r_irq = platform_get_resource(pdev, IORESOURCE_IRQ, 0);
    if (!r_irq) {
        dev_info(dev, "no IRQ found\n");
        dev_info(dev, "mcp2515_can at 0x%08x mapped to 0x%08x\n",
                (unsigned int __force)lp->mem_start,
                (unsigned int __force)lp->base_addr);
        return 0;
    }
    lp->irq = r_irq->start;
    printk("irq1");
    rc = request_irq(lp->irq, &mcp2515_can_irq, 0, DRIVER_NAME, lp);
    if (rc) {
        dev_err(dev, "testmodule: Could not allocate interrupt %d.\n",
                lp->irq);
        goto error3;
    }

    dev_info(dev, "mcp2515_can at 0x%08x mapped to 0x%08x, irq=%d\n",
            (unsigned int __force)lp->mem_start,
            (unsigned int __force)lp->base_addr,
            lp->irq);
    printk("irq2");
    ret = register_candev(net);

```

```

        if (!ret) {
            dev_info(dev, "register_candev probed\n");
            return ret;
        }
        printk("register_candev");
        return 0;
error3:
    free_irq(lp->irq, lp);
error2:
    release_mem_region(lp->mem_start, lp->mem_end - lp->mem_start + 1);
error1:
    kfree(lp);
    dev_set_drvdata(dev, NULL);
error_alloc:
    printk(KERN_ALERT "probe failed\n");
    return rc;
}

static int __devexit mcp2515_can_remove(struct platform_device *pdev)
{
    struct device *dev = &pdev->dev;
    struct spi_mcp2515 *lp = dev_get_drvdata(dev);
    struct mcp2515_priv *priv = dev_get_drvdata(&pdev->dev);
    struct net_device *net = priv->net;

    unregister_candev(net);
    printk("unregister_candev");
    free_candev(net);
    printk("free_candev");
    free_irq(lp->irq, lp);
    printk("free_irq");
    release_mem_region(lp->mem_start, lp->mem_end - lp->mem_start + 1);
    printk("release_mem_region");
    kfree(lp);
    printk("kfree");
    dev_set_drvdata(dev, NULL);
    printk("dev_set_drvdata");
    return 0;
}

#ifdef CONFIG_OF
static struct of_device_id mcp2515_can_of_match[] __devinitdata = {
    { .compatible = "petalogix,mcp2515_can", },
    { /* end of list */ },
};
MODULE_DEVICE_TABLE(of, mcp2515_can_of_match);
#else
# define mcp2515_can_of_match
#endif

static struct platform_driver mcp2515_can_driver = {
    .driver = {
        .name = DRIVER_NAME,
        .owner = THIS_MODULE,
        .of_match_table = mcp2515_can_of_match,
    },
};

```



```

        },
        .probe          = mcp2515_can_probe,
        .remove         = __devexit_p(mcp2515_can_remove),
};

static int __init mcp2515_can_init(void)
{
    printk(KERN_ALERT "Hello module world.\n");
    return platform_driver_register(&mcp2515_can_driver);
}

static void __exit mcp2515_can_exit(void)
{
    platform_driver_unregister(&mcp2515_can_driver);
    printk(KERN_ALERT "Goodbye module world.\n");
}

module_init(mcp2515_can_init);
module_exit(mcp2515_can_exit);

```