Progress Report: Feasibility Study for a New Robot Electronic System Architecture

TAM492 - Spring 2007
Research Report

Hsiang Wei LEE
hl336@cornell.edu

Current Address
120 Valentine Place Apt3008
Ithaca, NY 14850, USA

Permanent Address
9 Upper Bukit Timah View #08-04
Singapore 588136, Singapore

Contents **Page**

1. Introduction

For this semester, the goal of my project was to conduct a feasibility study for the next robot that is next built in the lab. In particular, this feasibility study seeks to examine that proposed electronic system architecture of the robot, learn how to used the various electronic components, as well as to test electronic components that will be used to implement the electronic system architecture. This new system is essentially about having in place a "network-based" model for controlling the robot.

2. Motivation

The motivation for studying the feasibility of the new system is twofold. Firstly, we hope to be able to implement a modular system of design for the electronic components. From our previous experiences of designing the Cornell Ranger as well as the Marathon Walker Bot (MWB), where the electronic components were integrated in a less modular manner, the programmers often had difficulty finding bugs in the code when a programming error was suspected. A less modular system also leads to a small group doing most of the code due to the laborious process of integrating various portions of the code. With a more modular system design, however, debugging code and testing programs can be simplified as we are able to test individual components and more people can participate in programming due to the very fact that it is more modular in nature. The second motivation of studying this new system is that we hope to find a solution to the massive wiring issues that plague not only the Cornell Ranger but also the MWB. Figure 1 and Figure 2 shows the wiring problem that has plagued the two robots.

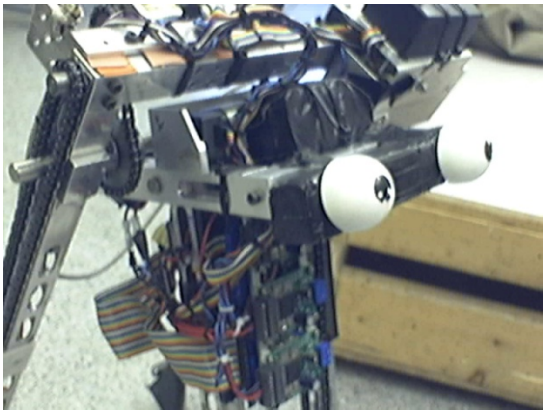

Figure 1: Wiring Problems of the MWB

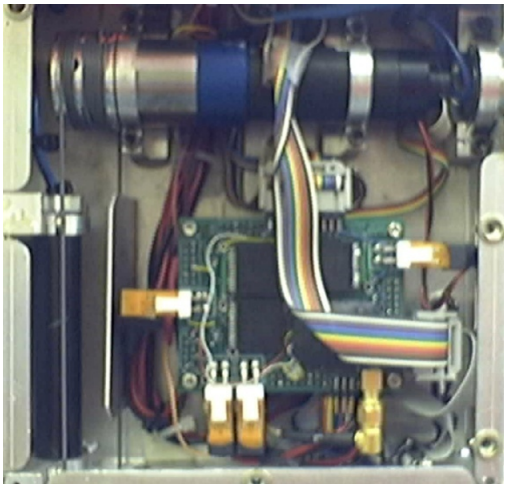

Figure 2: Wiring Problems of the Cornell Ranger

When a robot is in its design phase, it is usually the case that we do not foresee all the components that are needed to make it walk. As such, when the printed circuit board (PCB) is designed and manufactured, it is rarely self sufficient and additional components almost definitely have to be added onto the existing circuitry. With the addition of components outside of those already on the PCB, it is necessary to connect voltage and ground lines as well as data cables from the microcontroller to the added on component. This results in not only an unsightly nest of wires, but also the uncertainty of having a bad connection whenever something fails to work, instead of knowing that it I a software problem.

3. Network-Based Model

The network-based model that is being studied for the next robot proposes using multiple microcontrollers to control various parts of the robot. The model consists of the high level portion, communications level portion and the low level portion. The 3 main portions, as seen in Figure 3, are the High Level portion, the Communications Level portion and the Low Level portion.

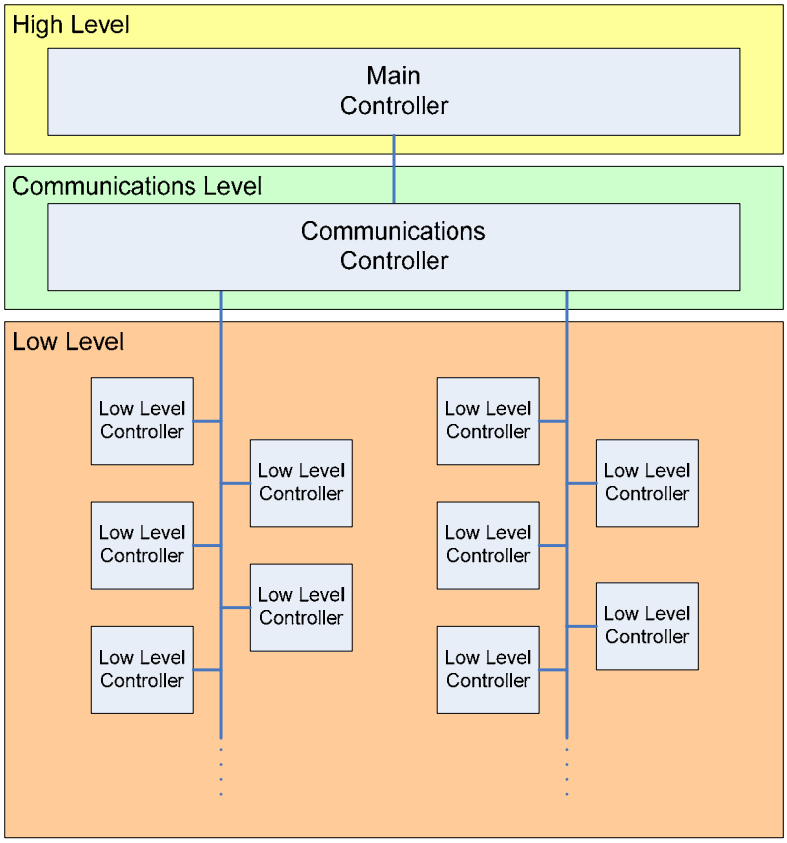

Figure 3: Overview of the proposed electronic system architecture

3.1 High Level Portion

The high level portion consists of the main controller which will be in charge of the controlling the robot on the high level. The code is specific to providing instructions to how the robot should move. Ideally, the code on the main microcontroller is similar to a human-readable list of instructions of how to get a robot to walk. The main controller communicates with the

communications controller constantly so as to ensure that the information it has about the robot is up to date.

3.2 Communications Level Portion

The main purpose of the communications level is to act as a bridge between the main controller in the High Level and the low level microcontrollers in the Low Level. It is responsible for the data flow between the two other levels. It acts very much like a router, collecting and redistributing information between all the microcontrollers in the system

3.3 Low Level Portion

In the low level portion, it consists of a number of low level microcontrollers. Each of these microcontrollers are be responsible for controlling a subset of the sensors and motors based either on function or location. Connected via a network with the communications controller, the multiple low level controllers send information collected via the sensors and receive instructions of what to carry actions to execute, to and from the communications microcontroller.

4. Equipment

4.1 Main Controller

Figure 4: Functional Board of the LPC3180

For the main controller, the current microcontroller that is being tested and evaluated is the LPC3180 as shown above in Figure 4. In a strict sense, the LPC3180 is more of a microprocessor than a microcontroller. Unlike a microcontroller which has its RAM and Flash memory integrated into the chip, the LPC3180 has its RAM and Flash memory external to the chip. The LPC3180 has several following key advantages:

- Operates with speeds up to 208Mhz
- High performance with low power dissipation (Full operating power consumption of 0.4mA/MHz)
- Exclusive hardware vector floating point unit
- Flexible power management for peak performance
- Large Memory Space (32MB RAM and 32MB Flash)
- General purpose DMA controller that can be used with the SD card and SPI interfaces, as well as for memory-to-memory transfers

The LPC3180 was primarily designed as a power embedded systems microcontroller and it is currently used in the following areas:

- Industrial
- Medical
- Peripheral control: printers, scanners, POS
- Medical devices
- GPS, motors, security devices, servo loops
- Network control
- Embedded Linux

4.2 Communications Controller

For the commutations controller, the current microcontroller that is being tested and evaluated is the LPC2368 as shown below on the evaluation board in Figure 5. The LPC2368, compared to the main controller, has significantly less processing power. As its main role is to handle communications from various parts of the robot, the LPC2368 has processing speeds up to 72MHz, with 58kB of RAM and 512kB of FLASH.

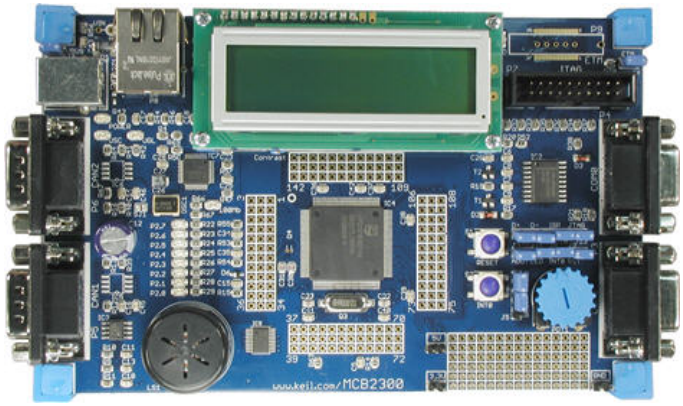

Figure 5: LPC2368 on a Keil Evaluation Board

The LPC2368 seems to be designed largely for communication purposes, with the ability to support multiple simultaneous communications operations. Its key communication features include:

- Ethernet 10/100 MAC with DMA
- USB 2.0 full-speed device with PHY and DMA
- CAN 2.0B with two channels
- General-purpose DMA controller
- I^2S, three I^2C, three SPI/SSP, and four UARTs

4.3 Low Level Controller

For the low level controller, the current microcontroller that is being tested and evaluated is the LPC2129 as shown below on the board by New Micros in Figure 6. The LPC2129 is a comparatively small and low powered processor which is widely used in the industrial control, automotive and medical industries. The LPC2129 is mainly targeted at general purpose applications, making it suitable as a low level controller which potentially has to interface with a wide range of peripherals. The key features of the LPC2129 include:

- Multiple 32-bit Timers
- 4 channel 10-bit Analog to Digital Converter
- 2 CAN bus channels
- Pulse Width Modulation channels
- 46 GPIO lines

Figure 6: LPC2129 on a board by New Micros

5. Communications Protocols

To allow for the main controller to have a good estimate of what state the robot is in and to instruct the various low level controllers to carry out various functions, channels of communication has to be set up. In specific, communication channels have to be set up between the main controller and the communications controller as well as between the communications controller and the low level controllers. One important consideration when choosing the appropriate protocols is that the protocol should be able to support continuously transferring a significant amount of data in a fast enough speed.

5.1 Direct Memory Access (DMA)

Direct memory access (DMA) is a feature that allows certain hardware subsystems within the computer to access system memory for reading and writing independently of the main core of the processor. DMA channels have the ability to transfer data to and from devices with much less overhead than microcontrollers without a DMA channel.

Without DMA, the main core of the processor typically has to be occupied for the entire time it's performing a transfer, while with DMA, the main core initiates the transfer, do other operations while the transfer is in progress, and receive an interrupt from the DMA controller once the operation has been done. This is especially useful for our purposes as it reduces the load on the core of the main controller and free up more resources for the needs of the robot. Figure 7 below shows the differences between data transfers that use DMA and those that do not, for data that resides in Memory to the Serial Peripheral Interface (SPI)

For the purposes of this robot, the data transfer between the main controller and the communications controller will be using the Serial Peripheral Interface (SPI) protocol and it will utilize DMA and implemented on the two respective controllers.

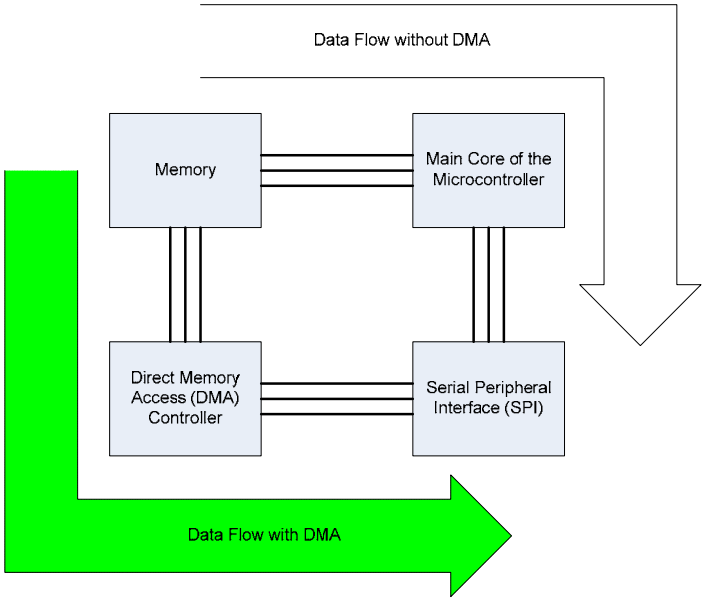

Figure 7: Data Flow between DMA enabled and DMA disabled transfers

5.2 Controller Area Network Bus (CAN)

CAN (also referred to as CANbus or CAN bus) is a network used in many every-day products consisting of multiple microcontrollers that need to communicate with each other. The

Controller Area Network is a broadcast, differential serial bus standard and it is specifically designed for operation in electromagnetically noisy environments. CAN also features an automatic 'arbitration free' transmission. A CAN message transmitted with highest priority will 'win' the arbitration, and the node transmitting the lower priority message will sense this and back off and wait. In addition, CAN provide:

- A multi-master hierarchy, which allows building intelligent and redundant systems. If one network node is defect the network is still able to operate.
- Broadcast communication. A sender of information transmits to all devices on the bus. All receiving devices read the message and then decide if it is relevant to them. This guarantees data integrity as all devices in the system use the same information.
- Sophisticated error detecting mechanisms and re-transmission of faulty messages. This also guarantees data integrity.

Given the above mentioned properties, such a network is suitable for use to transfer data and information between the communications microcontroller and the numerous low level controllers.

6. Getting Started

6.1 Equipment and Programs

6.1.1 IAR Embedded Workbench

IAR Embedded Workbench is the program that the lab is currently using to evaluate and program the various microcontrollers. To obtain the software to start programming the microcontrollers, proceed to http://supp.iar.com/Download/SW/?item=EWARM-KS32 to obtain the *KickStart version of IAR Embedded Workbench for ARM*. The KickStart version of the software has a code size limit of 32kB, but for our purposes it is more than sufficient.

Following the link, you will be prompted to submit some information. IAR Systems will proceed to email you a link to obtain the license number, license key as well as to download the software itself.

If any problems are encountered in the download process, proceed to http://supp.iar.com/Contact/ to request assistance from IAR Systems.

6.1.2 J-Link

The J-Link is a device that connects to your computer via USB and it is used by IAR Embedded Workbench to program the microcontroller via JTAG. To install the drivers to your computer, proceed to http://www.iar.com/jlinkarm and the drivers can be downloaded from the "product updates" in the "downloads" column. Once the driver has been installed, you will just need to plug it into your computer and to the microcontroller via JTAG and you are able to program the microcontroller

6.2 Evaluation Boards

6.2.1 LPC2368 Keil Evaluation Board

For the LPC268 Keil evaluation board, the key components of the evaluation board are listed in Figure 8 below. The evaluation board is powered via a USB cable to your computer and it is programmed via JTAG using the J-Link.

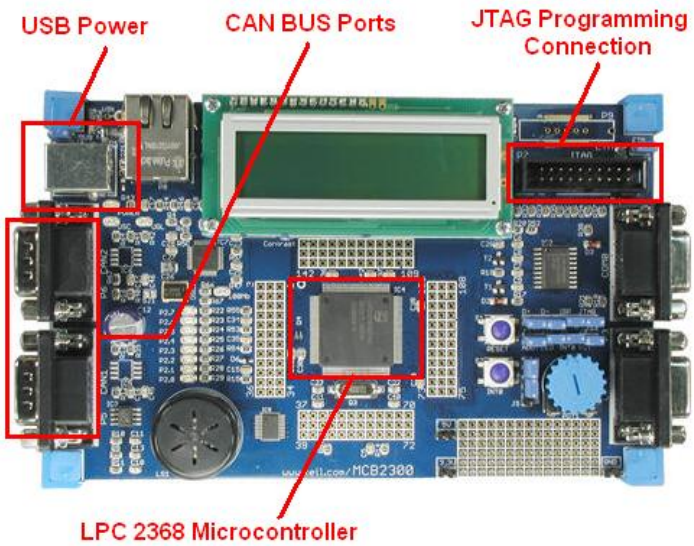

Figure 8: Key Components of the LPC2368 Evaluation Board

6.2.2 LPC3180 Phytec Evaluation Board

For the LPC3180 Phytec evaluation board, the key components of the evaluation board are listed in Figure 9 below. The evaluation board is powered via a 5 Volt DC adapter and it is programmed via JTAG using the J-Link.

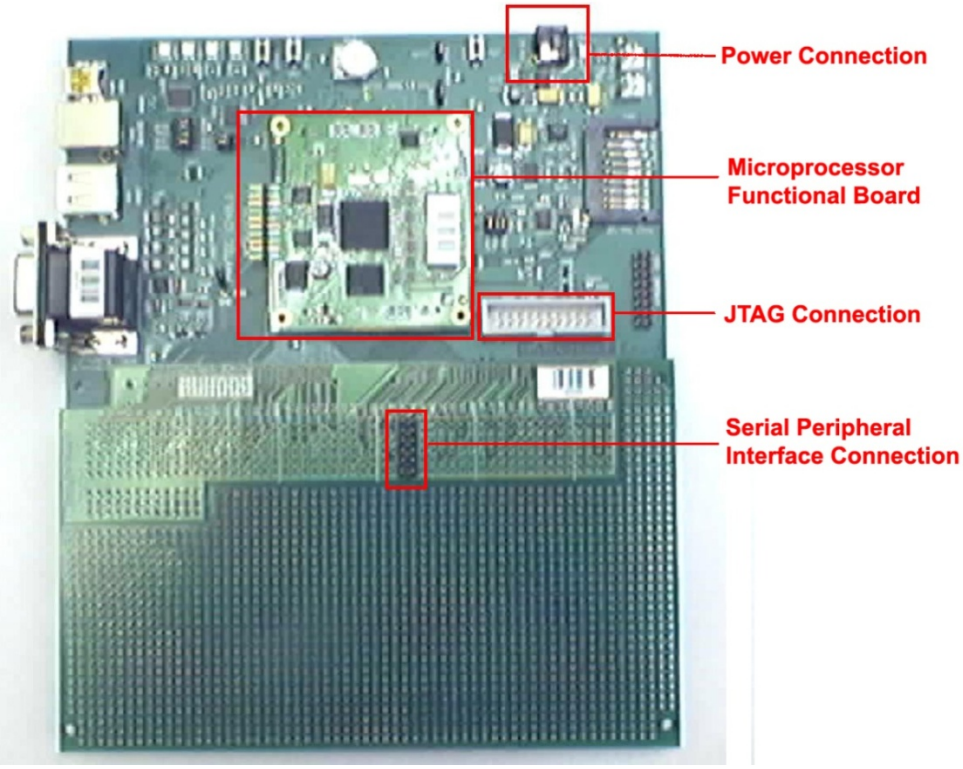

Figure 9: Key Components of the LPC3180 Evaluation Board

6.3. Coding

The program structure in the example code differs significantly between the LPC2368 and LPC3180. Hence, care has to be taken to prevent the confusion between the two different implementation styles in C.

6.3.1 LPC3180

Using the Example Code

The example code for the LPC3180 can be obtained from the following web address: http://www.phytec.com/zip/Demos_PCM-031_I.zip. After unzipping the file to the folder of your choice, open the project in IAR Embedded Workbench by double clicking on GettingStarted.ewp in the "Gettingstarted" folder or opening it through IAR workbench itself.

Updating the IAR Workbench Software

There is an update available for the IAR Workbench software that is needed to program the LPC3180. In the CD that is provided in this report, in the software folder, using from the file:

FlashPhytecLPC3180 – Software Update.zip

Unpack the files in the .zip folder into your Embedded Workbench ARM (EWARM) installation directory. This is necessary as there is a minor bug in some of the files that is provided by the installation software.

Programming to FLASH

The example code at the current point of time is configured to program to RAM. However to program to FLASH, you have to switch configurations by changing the configurations under the workspace sub window, to "LPC3180 Ext NAND Flash", as shown in Figure 10

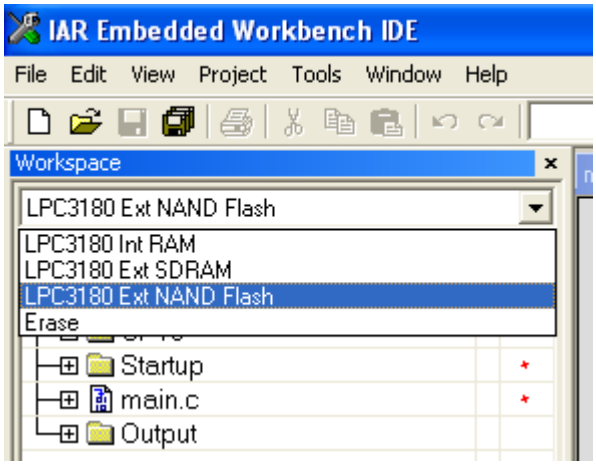

Figure 10: Changing the Configuration to Program to Flash

In a case were the project does not have to configurations as described above, proceed with the following steps to setup a new configuration.

1. Proceed to : Project > Edit Configurations > New...

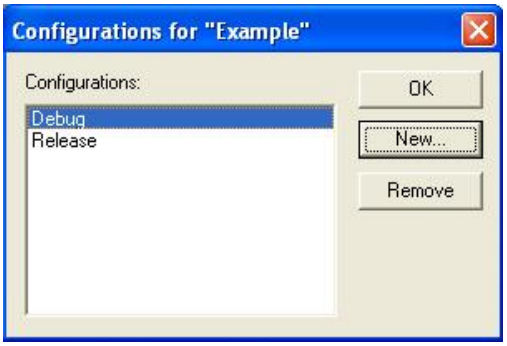

Figure 11 : Step1

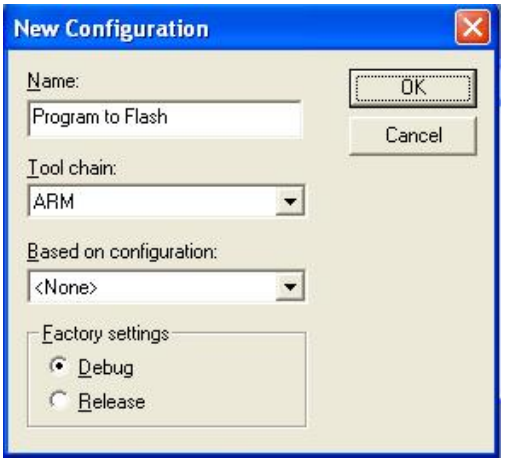

Figure 12 : Step 2

2. Name the configuration, i.e. “Flash” and set it be base on “<None>” and click “OK”
3. Proceed to : Project> Options while the Program to Flash configuration is selected under the workspace sub window.
4. Under General Options, in the Target tab, (see Figure 13)
    a. set the device to LPC3180
    b. set the FPU to VFP9-S
    c. set Processor Mode to “Arm mode

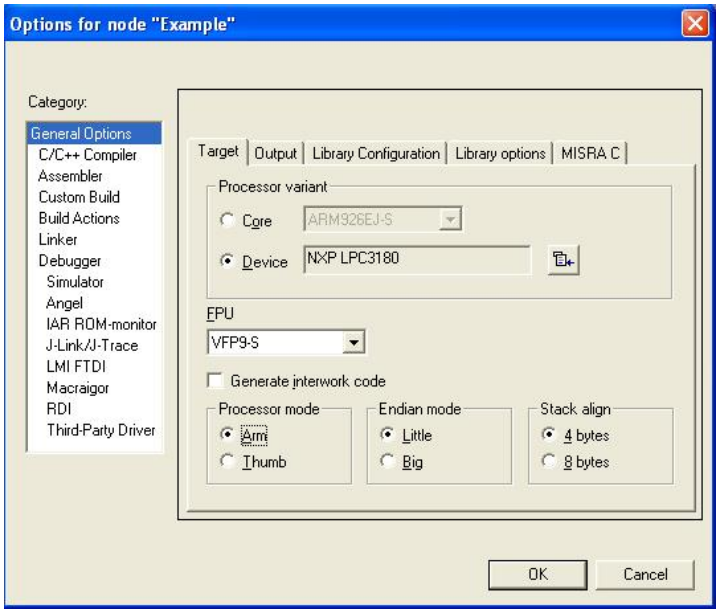

Figure 13: General Options (Step 4)

5. Under the “Linker” option, in the “Output” tab, make sure that the following options are checked:

    a. “Debug information for C-SPY”

    b. “With runtime modules”

    c. “With I/O emulation modules”

    d. “Allow C-SPY specific extra output file”

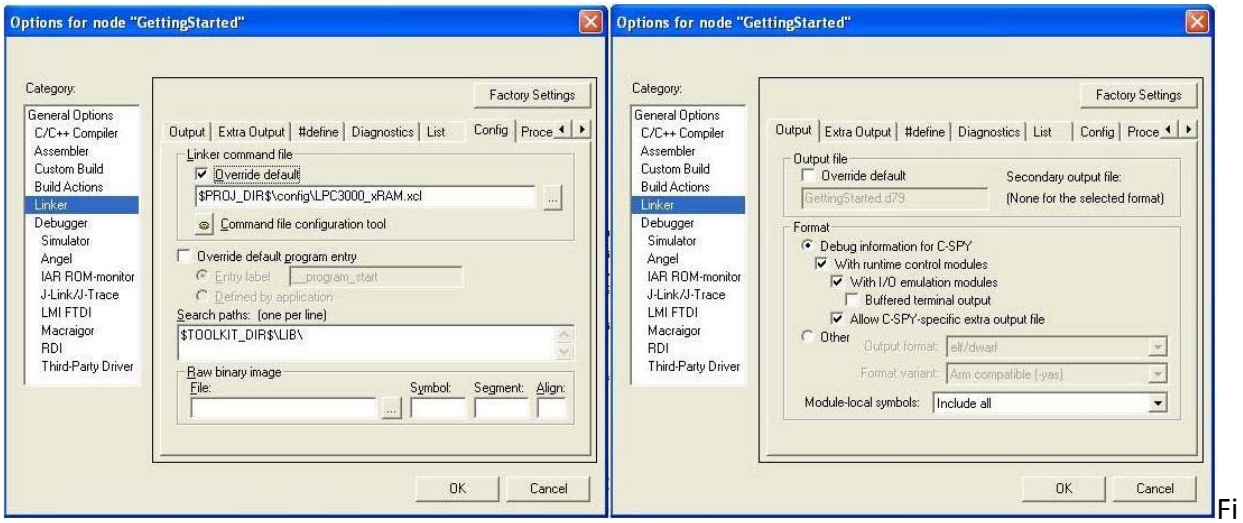

Figure 14: Steps 5 and 6

6. Under the “Linker” option, in the “Config” tab, override the default linker file and set it to the appropriate .XCL file. For LPC3180, it is in the “config” folder in the GettingStarted directory.

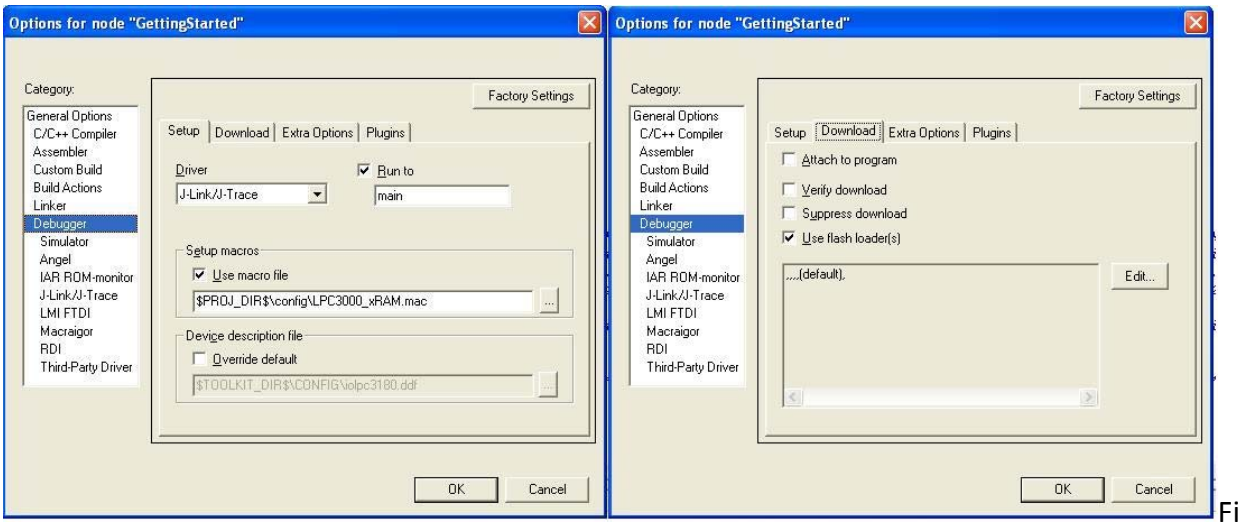

Figure 15 : Steps 7 and 8

7. In the “Debugger” option, under the Setup tab, make sure that the driver is set to “J Link/J Trace” and the appropriate macro file is chosen.

8. In the “Debugger” option, under the Download tab, check the “Use flash loader(s)” option

Coding Conventions

The key features in the coding conventions that is unique to the LPC3180 is the way registers and the bits in the respective registers named and referenced. The naming of the respective bits in a register is user-defined in the macro file, “iolpc3180.h”. They are named and referenced to follow the naming conventions as specified in the user manual.

For example, if we wanted to access a hypothetical “Register A” and we like to set the first bit to zero, the following code is written:

```
REGISTERA_bit.BIT0      = 0;
```

Also, if there is a field in the register that spans more than 1 bit, the above convention can be used to set that particular field, as long as the fields are defined in the macro file. Alternatively, if there is a need to assign bits to the whole register in a single line, the following code can also be written, assuming "Register A" is a 32-bit register:

```
REGISTERA                    = 0x89ABCDEF;
```

6.3.2 LPC2368

Using the Example Code

The example code for the LPC2368 can be obtained from the following web address: http://www.indyelectronics.com/code.lpc23xx.peripherals.iar.zip. After unzipping the file to the folder of your choice, open the project in IAR Embedded Workbench by double clicking on the .ewp file or opening it through IAR workbench itself.

In the example code provided in the file above comes with multiple projects which test and use the various aspects of the LPC2368. To choose the particular project, right click on the particular project and click on "Set as Active", as shown in Figure 16.

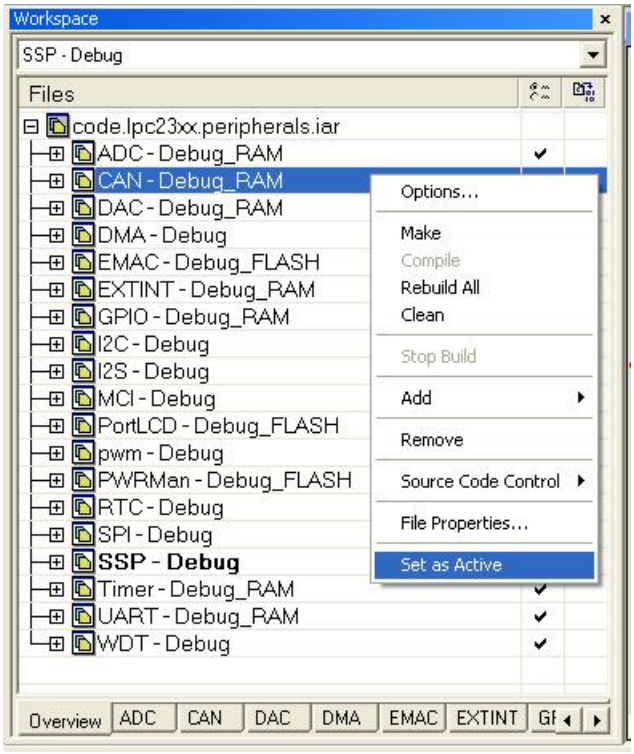

Figure 16: Choosing a project in the example code

To just browse through the other projects without selecting them, click on the tabs at the bottom of the workspace. Do note that you can, as mentioned previously with the LPC3180, set the configurations for this project. The example code, however, has a bug in its configuration files, hence, we are unable to program to Flash at the moment.

Coding Conventions

The coding convention of the files on the LPC2368 was originally different as compared to the files for the LPC3180. However, the code was revised and rewritten so that both the microcontrollers share the same coding conventions.

7. Preliminary Results

7.1 Standardizing Coding Conventions for LPC2368 and LPC3180

The main motivation for standardizing the coding conventions is that the styles of the example code differ greatly between that provided with the LPC3180 and the LPC2368. I decided to rewrite the LPC2368 to match that of the LPC3180 as the coding convention of the LPC3180 was more user-friendly and it supported the code convention of LPC2368 as well. To achieve this, I had to study the structure of the macro files that is used by the LPC3180 example and replicate it with the LPC2368.

To replicate the similar code structure, it requires defining the registers and their respective bits in the macro file. First, it is necessary to define the contents in the register – what bit fields the register contains, and secondly, the register needs to be defined and linked with the memory address the values reside in.

Table 25. PLL Control register (PLLCON - address 0xE01F C080) bit description

Bit	Symbol	Description	Reset value
0	PLLE	PLL Enable. When one, and after a valid PLL feed, this bit will activate the PLL and allow it to lock to the requested frequency. See PLLSTAT register, Table 4–27.	0
1	PLLC	PLL Connect. Having both PLLC and PLLE set to one followed by a valid PLL feed sequence, the PLL becomes the clock source for the CPU, as well as the USB subsystem and. Otherwise, the clock selected by the Clock Source Selection Multiplexer is used directly by the LPC2300. See PLLSTAT register, Table 4–27.	0
7:2	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Figure 17:

For example, given the register in Figure 17 above, as obtained from the datasheet, we need to make the following definitions.

Defining the contents in the register:

```
/*PLL Control register*/
typedef struct{
__REG8 PLLE    : 1;
__REG8 PLLC    : 1;
__REG8         : 6;
} __pllcon_bits;
```

Defining the register itself and setting the address that it resides in memory

```
__IO_REG8_BIT(PLLCFG,               0xE01FC084,__READ_WRITE,__pllcfg_bits);
```

The definitions of the registers are contained in iolpc2368.h and it can be found as Appendix X. At the current point of time, not all the registers are defined in the macro file. Only those pertaining to the System Control Block, Clocking and Power Control, Vectored Interrupt Controller, Pin Connect Block, SSP Interface and the DMA Interface are implemented in the macro file.

7.2 Controller Area Network (CAN) Self Test on LPC2368

The Controller Area Network (CAN) Self Test was carried out using the example code provided. The purpose of this test was to try out the CAN protocol and to attempt to determine the data transfer limits with respect to the reliability of the protocol. The CAN protocol however, has not been tested with multiple microcontrollers as there is currently only the LPC2368 that has the CAN protocol. The LPC2129 could not be used at the current point of time due to faulty products provided by the manufacturer.

In getting the CAN protocol to work, it was found that the example code provided had a bug and it, causing the example code not to work. The bug was found and the code with the bug removed is contained in “Canbus.zip”. The main file, cantest.c, which contains the updated changes, can be found in Appendix X.

7.3 Serial Peripheral Interface (SPI) and DMA Self Test on LPC2368

7.3.1 Getting SPI to Work

For getting the SPI protocol to work, the following registers have to be initialized.

- Power Control for Peripherals Register (PCONP)
- Pin Function Select Register (PSELn)
- Serial Peripheral Interface Clock Prescale Register (SSPnCPSR)
- Serial Peripheral Interface Control Register 0 and 1 (SSPnCR0 and SSPnCR1)

After setting up the registers as mentioned above, there is a need to clear the receive buffers so as to ensure no garbage data remains in the buffers. The initialization sequence code can be seen in Appendix X.

7.3.2 Determining Possible Speeds for SPI

The transfer speeds for the SPI are dependent on the following 3 values:

1. Peripheral Clock Rate (PCLK)
2. SSP Clock Prescaler (SSPnCPSR)
3. Serial Clock Rate (SCR)

And the actual rate is determined by the following relation:

$$SPI\ Transfer\ Rate = \frac{PCLK}{SSPnCPSR * (SCR + 1)}$$

According to the datasheet the maximum SPI Transfer Rate is 6MHz. With:

$$PCLK = 72MHz$$
$$SSPnCPSR = 12$$
$$SCR = 0$$

Since the PCLK is running at its maximum speed, the only way to increase the transfer rate is to decrease the SSPnCPSR. Testing the SPI connection by lowering this value, I found that the value of SSPnCPSR can be lowered to 6 and it still functions reliably. This implies that the maximum possible SPI Transfer Rate is 12MHz.

7.3.3 Enabling Direct Memory Access (DMA) with SPI

The setting up of the DMA on the LPC2368 was significantly more time consuming and tricky due to the poor documentation provided by the user manual.

The first important point that should be noted is that for the LPC2368, the DMA source and destination registers can only be pointing to locations in the USB RAM. Setting the pointers to other memory locations will not work.

The second important point is that the Source Burst and Destination Burst Sizes in the SPI Channel Control Register have to be set to 4. Attempting to set up the DMA transfers, at least for the self test, with other values will cause the DMA to transmit and receive erroneous data.

The initialization routines for the DMA Controller on the LPC2368 is attached as appendix X

7.4 SPI and DMA Self Test on LPC3180

7.4.1 Getting SPI to Work

For getting the SPI protocol to work on the LPC3180, the following registers have to be initialized.

- SPI Block Control register (SPI_CTRL)
- SPIn Global Control register (SPIn_GLOBAL)
- SPIn Control register (SPIn_CON)
- SPIn Frame Count register (SPIn_FRM)
- Various Interrupt Registers

As the SPI interface on the LPC3180 has in built buffers – 56 entries for receive, 8 entries for transmit, to enable the buffers, the respective bits have to be set in the SPIn Control Register.

After setting up the registers as mentioned above, to send data via the SPI, pass the data into the SPIn Data Buffer register (SPIn_DAT).

```
//Send Data via SPI1
SPI1_DAT=0x01;
```

And to receive data from SPI, just read the data from the Data Buffer Register

```
//Receive Data from SPI1
temp = SPI1_DAT;
```

7.4.1 Setting up DMA on LPC3180

Setting up the DMA Self Test on the LPC3180, the sequence of setting the registers are paramount in getting the DMA to work.

Setting the appropriate transfer size and enabling the Transfer Counter (TC) interrupt, it is possible to let the DMA operate in the background until the given number of transfers is completed. After which, the DMA can be reinitialized to repeat the same transfer of data.

The DMA initialization sequence for the LPC3180 can be found in Appendix X

7.5 SPI Communication between LPC2368 and LPC3180

After getting the LPC2368 and LPC3180 each to communicate with itself, the next step would naturally be getting both of them to communicate with each other.

7.5.1 Timing Problem

One of the issues that had to be considered in getting the two microcontrollers to communicate is the difference in speeds between them.

The LPC2368 has a 12MHz crystal as its main oscillator and its CPU speed can be set based on the following calculations, with a maximum of 72MHz:

$$CPU\ Clock = \frac{FCCO}{CLKSEL}$$

$$where\ FCCO - \frac{2 * M * F_{IN}}{N}$$

F_{IN} has to be in the range of 32kHz to 50Mhz and FCCO in the range of 260MHz and 290MHz. (Note: the range of values for FCCO differs from that mentioned in the user manual and the change can be found in the erratum issued by NXP)

The LPC3180 however, uses a 13MHz crystal as its main oscillator and there are multiple modes of setting the CPU Clock. However, in order to not deviate too far away from the maximum possible CPU Clock of 208MHz, the CPU Clock is a multiple of 13MHz

In order to fit both the microcontrollers such that they have a common rate of transfer over the SPI, it was decided to lower the CPU Clock for the LPC2368 from 72MHz to 65MHz, thereby allowing the SPI transfer rate to be that of 6.5MHz. The clock initialization settings for the LPC2368 and LPC3180 can be found in Appendix X and X respectively.

7.5.1 Compatibility Problem

While attempting to get both microcontrollers to communicate via SPI with each other, I realized that there were key differences between their implementation resulting in two main compatibility problems.

The first compatibility problem is the design of SPI interface of the two microcontrollers. The LPC2368 has a SPI interface consisting of 4 wires:

1. Serial Clock
2. Slave Select
3. Data In (Master Out Slave In)
4. Data Out (Master In Slave Out)

However the LPC3180 only has the following 3 wires, lacking the Slave Select:

1. Serial Clock
2. Data In (Master In Slave Out)
3. Data Out (Master Out Slave In)

The Slave Select on the SPI is a standard line where it is necessary to indicate to the communicating devices, the intended recipient of the transferred data. The function of the Slave Select can be further explained in Figure X below:

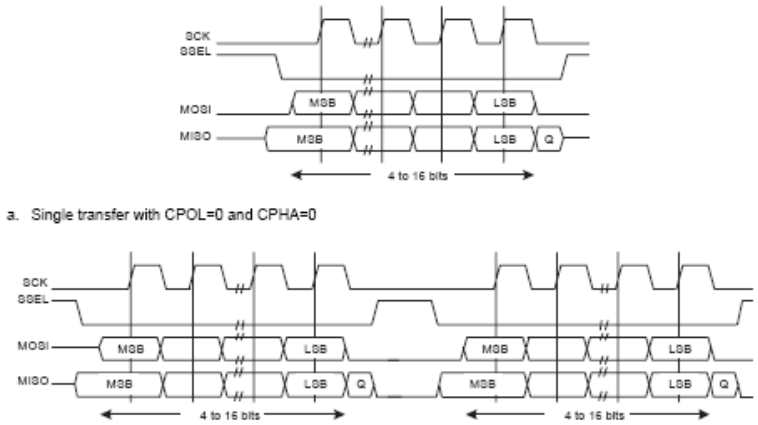

Figure X

This problem is currently being worked on by using a series of inverting buffers to generate the Slave Select signals, based on the serial clock signal.

8. References

CAN in Automation (CiA). (2004, May 26). *Controller Area Network*. Retrieved April 23, 2007, from CAN in Automation (CiA): http://www.can-cia.org/can/

Embedded Microprocessor Benchmark Corporation. (2006, November 8). *EEMBC® Scores for NXP's ARM7-Based LPC2129* . Retrieved April 23, 2007, from Embedded Microprocessor Benchmark Corporation: http://www.eembc.org/Press/PressRelease/061108.htm

Future Electronics. (2006, June). *90nm ARM9™-based Microcontroller with Hardware Floating Point and USB OTG*. Retrieved April 23, 2007, from Future Electronics Technology Magazine: http://www.future-mag.com/0610/061032.asp

NXP. (2005, September 25). *NXP Semiconductors unveils industry's first ARM7 microcontrollers with dual high-speed buses*. Retrieved April 23, 2007, from NXP Semiconductors: http://www.standardics.nxp.com/news/lpc23xx.lpc24xx/~LPC2368/

9 Appendices

```
/***************************************************************************
 **
 **    This file defines the Special Function Registers for
 **    NXP LPC2368
 **
 **    Done By : Hsiang Wei LEE
 **              Cornell University
 **              Biorobotics and Locomotion Lab
 **
 **    Last Updated : 3 May 2007
 ***************************************************************************/

#ifndef __IOLPC2368_H
#define __IOLPC2368_H

#if (((__TID__ >> 8) & 0x7F) != 0x4F)     /* 0x4F = 79 dec */
#error This file should only be compiled by ARM IAR compiler and assembler
#endif

#include "io_macros.h"

/***************************************************************************
**    LPC3180 SPECIAL FUNCTION REGISTERS
***************************************************************************/

/* C-compiler specific declarations  ***********************************/

#ifdef __IAR_SYSTEMS_ICC__

/* Reset Source Identification Register */
typedef struct{
__REG8 POR       : 1;
__REG8 EXTR      : 1;
__REG8 WDTR      : 1;
__REG8 BODR      : 1;
__REG8           : 4;
} __rsir_bits;

/* External Interupt Resgister */
typedef struct{
__REG8 EINT0     : 1;
__REG8 EINT1     : 1;
__REG8 EINT2     : 1;
__REG8 EINT3     : 1;
__REG8           : 4;
} __extint_bits;

/* External Interupt Resgister */
typedef struct{
__REG8 EXTMODE0  : 1;
__REG8 EXTMODE1  : 1;
__REG8 EXTMODE2  : 1;
__REG8 EXTMODE3  : 1;
__REG8           : 4;
} __extmode_bits;

/* External Interupt Polarity Resgister */
typedef struct{
__REG8 EXTPOLAR0 : 1;
__REG8 EXTPOLAR1 : 1;
__REG8 EXTPOLAR2 : 1;
__REG8 EXTPOLAR3 : 1;
__REG8           : 4;
} __extpolar_bits;

/* System Control and Status Resgister */
typedef struct{
__REG32 GPIOM    : 1;
__REG32 EMC      : 1;
__REG32          : 1;
__REG32 MCIPWR   : 1;
__REG32 MCIRANGE : 1;
__REG32 OSCEN    : 1;
__REG32 OSCSTAT  : 1;
__REG32          : 25;
} __scs_bits;

/*AHB Arbiter Configuration register 1*/
typedef struct{
__REG32 SCHEDULIZER    : 1;
__REG32 BREAK_BURST    : 2;
__REG32 QUANTUM_TYPE   : 1;
```

```
__REG32 QUANTUM_SIZE   : 4;
__REG32 DEFAULT_MASTER : 4;
__REG32 EP1            : 4;
__REG32 EP2            : 4;
__REG32 EP3            : 4;
__REG32 EP4            : 4;
__REG32                : 4;
} __ahbcfg1_bits;

/*AHB Arbiter Configuration register 1*/
typedef struct{
__REG32 SCHEDULIZER    : 1;
__REG32 BREAK_BURST    : 2;
__REG32 QUANTUM_TYPE   : 1;
__REG32 QUANTUM_SIZE   : 4;
__REG32 DEFAULT_MASTER : 4;
__REG32 EP1            : 4;
__REG32 EP2            : 4;
__REG32                :12;
} __ahbcfg2_bits;

/*Clock Source Select register*/
typedef struct{
__REG8 CLKSRC  : 2;
__REG8         : 6;
} __clksrcsel_bits;

/*PLL Control register*/
typedef struct{
__REG8 PLLE    : 1;
__REG8 PLLC    : 1;
__REG8         : 6;
} __pllcon_bits;

/*PLL Configuration register*/
typedef struct{
__REG32 MSEL    : 15;
__REG32         : 1;
__REG32 NSEL    : 8;
__REG32         : 8;
} __pllcfg_bits;

/*PLL Status register*/
typedef struct{
__REG32 MSEL    : 15;
__REG32         : 1;
__REG32 NSEL    : 8;
__REG32 PLLE    : 1;
__REG32 PLLC    : 1;
__REG32 PLOCK   : 1;
__REG32         : 5;
} __pllstat_bits;

/*USB Clock Configuration register*/
typedef struct{
__REG8 USBSEL  : 4;
__REG8         : 4;
} __usbclkcfg_bits;

/*Peripheral Clock Selection register 0*/
typedef struct{
__REG32 PCLK_WDT    : 2;
__REG32 PCLK_TIMER0 : 2;
__REG32 PCLK_TIMER1 : 2;
__REG32 PCLK_UART0  : 2;
__REG32 PCLK_UART1  : 2;
__REG32 PCLK_PWM0   : 2;
__REG32 PCLK_PWM1   : 2;
__REG32 PCLK_I2C0   : 2;
__REG32 PCLK_SPI    : 2;
__REG32 PCLK_RTC    : 2;
__REG32 PCLK_SSP1   : 2;
__REG32 PCLK_DAC    : 2;
__REG32 PCLK_ADC    : 2;
__REG32 PCLK_CAN1   : 2;
__REG32 PCLK_CAN2   : 2;
__REG32 PCLK_ACF    : 2;
} __pclksel0_bits;

/*Peripheral Clock Selection register 1*/
typedef struct{
__REG32 PCLK_BAT_RAM: 2;
```

```
__REG32 PCLK_GPIO   : 2;
__REG32 PCLK_PCB    : 2;
__REG32 PCLK_I2C1   : 2;
__REG32             : 2;
__REG32 PCLK_SSP0   : 2;
__REG32 PCLK_TIMER2 : 2;
__REG32 PCLK_TIMER3 : 2;
__REG32 PCLK_UART2  : 2;
__REG32 PCLK_UART3  : 2;
__REG32 PCLK_I2C2   : 2;
__REG32 PCLK_I2S    : 2;
__REG32 PCLK_MCI    : 2;
__REG32             : 2;
__REG32 PCLK_SYSCON : 2;
__REG32             : 2;
} __pclksel1_bits;

/*Interrupt Wakeup Register*/
typedef struct{
__REG16 EXTWAKE0     : 1;
__REG16 EXTWAKE1     : 1;
__REG16 EXTWAKE2     : 1;
__REG16 EXTWAKE3     : 1;
__REG16 ETHWAKE      : 1;
__REG16 USBWAKE      : 1;
__REG16 CANWAKE      : 1;
__REG16 GPIOWAKE     : 1;
__REG16              : 6;
__REG16 BODWAKE      : 1;
__REG16 RTCWAKE      : 1;
} __intwake_bits;

/*Power Mode Control register*/
typedef struct{
__REG8 PM0          : 1;
__REG8 PM1          : 1;
__REG8 BODPDM       : 1;
__REG8 BOGD         : 1;
__REG8 BORD         : 1;
__REG8              : 2;
__REG8 PM2          : 1;
} __pcon_bits;

/*Power Control for Peripherals register*/
typedef struct{
__REG32             : 1;
__REG32 PCTIM0      : 1;
__REG32 PCTIM1      : 1;
__REG32 PCUART0     : 1;
__REG32 PCUART1     : 1;
__REG32 PCPWM0      : 1;
__REG32 PCPWM1      : 1;
__REG32 PCI2C0      : 1;
__REG32 PCSPI       : 1;
__REG32 PCRTC       : 1;
__REG32 PCSSP1      : 1;
__REG32 PCEMC       : 1;
__REG32 PCAD        : 1;
__REG32 PCAN1       : 1;
__REG32 PCAN2       : 1;
__REG32 PCI2C1      : 4;
__REG32             : 1;
__REG32 PCSSP0      : 1;
__REG32 PCTIM2      : 1;
__REG32 PCTIM3      : 1;
__REG32 PCUART2     : 1;
__REG32 PCUART3     : 1;
__REG32 PCI2C2      : 1;
__REG32 PCI2S       : 1;
__REG32 PCSDC       : 1;
__REG32 PCGPDMA     : 1;
__REG32 PCENET      : 1;
__REG32 PCUSB       : 1;
} __pconp_bits;

/*Vector Interupts*/
typedef struct{
__REG32  INT0       : 1;
__REG32  INT1       : 1;
__REG32  INT2       : 1;
__REG32  INT3       : 1;
__REG32  INT4       : 1;
```

```
__REG32  INT5       : 1;
__REG32  INT6       : 1;
__REG32  INT7       : 1;
__REG32  INT8       : 1;
__REG32  INT9       : 1;
__REG32  INT10      : 1;
__REG32  INT11      : 1;
__REG32  INT12      : 1;
__REG32  INT13      : 1;
__REG32  INT14      : 1;
__REG32  INT15      : 1;
__REG32  INT16      : 1;
__REG32  INT17      : 1;
__REG32  INT18      : 1;
__REG32  INT19      : 1;
__REG32  INT20      : 1;
__REG32  INT21      : 1;
__REG32  INT22      : 1;
__REG32  INT23      : 1;
__REG32  INT24      : 1;
__REG32  INT25      : 1;
__REG32  INT26      : 1;
__REG32  INT27      : 1;
__REG32  INT28      : 1;
__REG32  INT29      : 1;
__REG32  INT30      : 1;
__REG32  INT31      : 1;
} __vic_bits;

/*VIC SW Priority Register*/
typedef struct{
__REG32 PriorityMask: 16;
__REG32             : 16;
} __vicswpriority_bits;

/*VIC Protection Register*/
typedef struct{
__REG32 VIC_access  : 1;
__REG32             : 31;
} __vicprotection_bits;
```

```
/*Pin Select Register 0*/
typedef struct{
__REG32 P0          : 2;
__REG32 P1          : 2;
__REG32 P2          : 2;
__REG32 P3          : 2;
__REG32 P4          : 2;
__REG32 P5          : 2;
__REG32 P6          : 2;
__REG32 P7          : 2;
__REG32 P8          : 2;
__REG32 P9          : 2;
__REG32 P10         : 2;
__REG32 P11         : 2;
__REG32 P12         : 2;
__REG32 P13         : 2;
__REG32 P14         : 2;
__REG32 P15         : 2;
} __pinselA_bits;

/*Pin Select Register 1*/
typedef struct{
__REG32 P16         : 2;
__REG32 P17         : 2;
__REG32 P18         : 2;
__REG32 P19         : 2;
__REG32 P20         : 2;
__REG32 P21         : 2;
__REG32 P22         : 2;
__REG32 P23         : 2;
__REG32 P24         : 2;
__REG32 P25         : 2;
__REG32 P26         : 2;
__REG32 P27         : 2;
__REG32 P28         : 2;
__REG32 P29         : 2;
__REG32 P30         : 2;
__REG32 P31         : 2;
} __pinselB_bits;

/*Pin Select Register 10*/
```

```
typedef struct{
__REG32              : 3;
__REG32 TRACE        : 1;
__REG32              : 28;
}__pinsel10_bits;

/*Pin Mode Register */
typedef struct{
__REG32 P00MODE      : 2;
__REG32              : 28;
__REG32 P15MODE      : 2;
}__pinmodeA_bits;
typedef struct{
__REG32 P16MODE      : 2;
__REG32              : 28;
__REG32 P31MODE      : 2;
}__pinmodeB_bits;
typedef struct{
__REG32 P16MODE      : 2;
__REG32              : 28;
__REG32 P26MODE      : 2;
}__pinmodeC_bits;


/*SSPn Control Register 0*/
typedef struct{
__REG16 DSS          : 4;
__REG16 FRF          : 2;
__REG16 SPO          : 1;
__REG16 SPH          : 1;
__REG16 SCR          : 8;
}__sspcr0_bits;

/*SSPn Control Register 1*/
typedef struct{
__REG8 LBM          : 1;
__REG8 SSE          : 1;
__REG8 MS           : 1;
__REG8 SOD          : 1;
__REG8              : 4;
}__sspcr1_bits;

/*SSPn Status Register*/
typedef struct{
__REG8 TFE          : 1;
__REG8 TNF          : 1;
__REG8 RNE          : 1;
__REG8 RNF          : 1;
__REG8 BSY          : 1;
__REG8              : 3;
}__sspstat_bits;

/*SSPn Interrupt  register*/
typedef struct{
__REG8 ROR         : 1;
__REG8 RT         : 1;
__REG8 RX          : 1;
__REG8 TX          : 1;
__REG8               : 4;
}__sspir_bits;

/*SSPn Interrupt Clear  register*/
typedef struct{
__REG8 RORIC         : 1;
__REG8 RTIC         : 1;
__REG8               : 5;
}__sspicr_bits;

/*SSPn DMAControl  register*/
typedef struct{
__REG16 RXDMAE       : 1;
__REG16 TXDMAE         : 1;
__REG16               : 14;
}__sspdma_bits;

#endif    /* __IAR_SYSTEMS_ICC__ */

/* Declarations common to compiler and assembler **************************/
/***************************************************************************
 System Control Block
 ***************************************************************************/
//Reset, reset source identification
```

```
__IO_REG8_BIT(RSIR,               0xE01FC180,__READ_WRITE,__rsir_bits);
//External Interrupts
__IO_REG8_BIT(EXTINT,             0xE01FC140,__READ_WRITE,__extint_bits);
__IO_REG8_BIT(EXTMODE,            0xE01FC148,__READ_WRITE,__extmode_bits);
__IO_REG8_BIT(EXTPOLAR,           0xE01FC14C,__READ_WRITE,__extpolar_bits);
__IO_REG16_BIT(INTWAKE,           0xE01FC144,__READ_WRITE,__intwake_bits);
//Code Security Protection Register
__IO_REG32(CPSR,                   0xE01FC184,__READ_WRITE);
//AHB configuration
__IO_REG32_BIT(AHBCFG1,            0xE01FC188,__READ_WRITE,__ahbcfg1_bits);
__IO_REG32_BIT(AHBCFG2,            0xE01FC18C,__READ_WRITE,__ahbcfg2_bits);
//System Controls and Status
__IO_REG32_BIT(SCS,                0xE01FC1A0,__READ_WRITE,__scs_bits);
/******************************************************************************
Clocking and Power Control
 ******************************************************************************/
//Clock source selection
__IO_REG8_BIT(CLKSRCSEL,           0xE01FC10C,__READ_WRITE,__clksrcsel_bits);
//Phase Locked Loop (PLL)
__IO_REG8_BIT(PLLCON,              0xE01FC080,__READ_WRITE,__pllcon_bits);
__IO_REG8_BIT(PLLCFG,              0xE01FC084,__READ_WRITE,__pllcfg_bits);
__IO_REG32_BIT(PLLSTAT,            0xE01FC088,__READ      ,__pllstat_bits);
__IO_REG8(PLLFEED,                 0xE01FC08C,__WRITE);
//Clock Divider
__IO_REG8(CCLKCFG,                 0xE01FC104,__READ_WRITE);
__IO_REG8_BIT(USBCLKCFG,           0xE01FC108,__READ_WRITE,__usbclkcfg_bits);
__IO_REG32_BIT(PCLKSEL0,           0xE01FC1A8,__READ_WRITE,__pclksel0_bits);
__IO_REG32_BIT(PCLKSEL1,           0xE01FC1AC,__READ_WRITE,__pclksel1_bits);
//Power Control
__IO_REG8_BIT(PCON,                0xE01FC0C0,__READ_WRITE,__pcon_bits);
__IO_REG32_BIT(PCONP,              0xE01FC0C4,__READ_WRITE,__pconp_bits);
/******************************************************************************
External Memory Controller
 ******************************************************************************/
//Not Implemented Yet
/******************************************************************************
Memory Acceleration Module (MAM)
 ******************************************************************************/
//Not Implemented Yet
/******************************************************************************
Vectored Interrupt Controller (VIC)
 ******************************************************************************/
__IO_REG32_BIT(VICIRQStatus,        0xFFFFF000,__READ,__vic_bits);
__IO_REG32_BIT(VICFIQStatus,        0xFFFFF004,__READ,__vic_bits);
__IO_REG32_BIT(VICRawIntr,          0xFFFFF008,__READ,__vic_bits);
__IO_REG32_BIT(VICIntSelect,        0xFFFFF00C,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICIntEnable,        0xFFFFF010,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICIntEnClear,       0xFFFFF014,__WRITE,__vic_bits);
__IO_REG32_BIT(VICSoftInt,          0xFFFFF018,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICSoftIntClear,     0xFFFFF01C,__WRITE,__vic_bits);
__IO_REG32_BIT(VICProtection,     0xFFFFF020,__READ_WRITE,__vicprotection_bits);
__IO_REG32_BIT(VICSWPriorityMask,0xFFFFF024,__READ_WRITE,__vicswpriority_bits);
__IO_REG32_BIT(VICVectAddr0,        0xFFFFF100,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr1,        0xFFFFF104,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr2,        0xFFFFF108,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr3,        0xFFFFF10C,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr4,        0xFFFFF110,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr5,        0xFFFFF114,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr6,        0xFFFFF118,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr7,        0xFFFFF11C,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr8,        0xFFFFF120,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr9,        0xFFFFF124,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr10,       0xFFFFF128,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr11,       0xFFFFF12C,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr12,       0xFFFFF130,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr13,       0xFFFFF134,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr14,       0xFFFFF138,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr15,       0xFFFFF13C,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr16,       0xFFFFF140,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr17,       0xFFFFF144,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr18,       0xFFFFF148,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr19,       0xFFFFF14C,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr20,       0xFFFFF150,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr21,       0xFFFFF154,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr22,       0xFFFFF158,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr23,       0xFFFFF15C,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr24,       0xFFFFF160,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr25,       0xFFFFF164,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr26,       0xFFFFF168,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr27,       0xFFFFF16C,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr28,       0xFFFFF170,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr29,       0xFFFFF174,__READ_WRITE,__vic_bits);
```

```
__IO_REG32_BIT(VICVectAddr30,       0xFFFFF178,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr31,       0xFFFFF17C,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority0,    0xFFFFF200,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority1,    0xFFFFF204,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority2,    0xFFFFF208,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority3,    0xFFFFF20C,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority4,    0xFFFFF210,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority5,    0xFFFFF214,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority6,    0xFFFFF218,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority7,    0xFFFFF21C,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority8,    0xFFFFF220,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority9,    0xFFFFF224,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority10,   0xFFFFF228,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority11,   0xFFFFF22C,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority12,   0xFFFFF230,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority13,   0xFFFFF234,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority14,   0xFFFFF238,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority15,   0xFFFFF23C,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority16,   0xFFFFF240,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority17,   0xFFFFF244,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority18,   0xFFFFF248,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority19,   0xFFFFF24C,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority20,   0xFFFFF250,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority21,   0xFFFFF254,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority22,   0xFFFFF258,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority23,   0xFFFFF25C,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority24,   0xFFFFF260,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority25,   0xFFFFF264,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority26,   0xFFFFF268,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority27,   0xFFFFF26C,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority28,   0xFFFFF270,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority29,   0xFFFFF274,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority30,   0xFFFFF278,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectPriority31,   0xFFFFF27C,__READ_WRITE,__vic_bits);
__IO_REG32_BIT(VICVectAddr,         0xFFFFFF00,__READ_WRITE,__vic_bits);
/***************************************************************************
Pin connect block
 ***************************************************************************/
__IO_REG32_BIT(PINSEL0,             0xE002C000,__READ_WRITE,__pinselA_bits);
__IO_REG32_BIT(PINSEL1,             0xE002C004,__READ_WRITE,__pinselB_bits);
__IO_REG32_BIT(PINSEL2,             0xE002C008,__READ_WRITE,__pinselA_bits);
__IO_REG32_BIT(PINSEL3,             0xE002C00C,__READ_WRITE,__pinselB_bits);
__IO_REG32_BIT(PINSEL4,             0xE002C010,__READ_WRITE,__pinselA_bits);
__IO_REG32_BIT(PINSEL5,             0xE002C014,__READ_WRITE,__pinselB_bits);
__IO_REG32_BIT(PINSEL6,             0xE002C018,__READ_WRITE,__pinselA_bits);
__IO_REG32_BIT(PINSEL7,             0xE002C01C,__READ_WRITE,__pinselB_bits);
__IO_REG32_BIT(PINSEL8,             0xE002C020,__READ_WRITE,__pinselA_bits);
__IO_REG32_BIT(PINSEL9,             0xE002C024,__READ_WRITE,__pinselB_bits);
__IO_REG32_BIT(PINSEL10,            0xE002C028,__READ_WRITE,__pinsel10_bits);
__IO_REG32_BIT(PINMODE0,            0xE002C040,__READ_WRITE,__pinmodeA_bits);
__IO_REG32_BIT(PINMODE1,            0xE002C044,__READ_WRITE,__pinmodeC_bits);
__IO_REG32_BIT(PINMODE2,            0xE002C048,__READ_WRITE,__pinmodeA_bits);
__IO_REG32_BIT(PINMODE3,            0xE002C04C,__READ_WRITE,__pinmodeB_bits);
__IO_REG32_BIT(PINMODE4,            0xE002C050,__READ_WRITE,__pinmodeA_bits);
__IO_REG32_BIT(PINMODE5,            0xE002C054,__READ_WRITE,__pinmodeB_bits);
__IO_REG32_BIT(PINMODE6,            0xE002C058,__READ_WRITE,__pinmodeA_bits);
__IO_REG32_BIT(PINMODE7,            0xE002C05C,__READ_WRITE,__pinmodeB_bits);
__IO_REG32_BIT(PINMODE8,            0xE002C060,__READ_WRITE,__pinmodeA_bits);
__IO_REG32_BIT(PINMODE9,            0xE002C064,__READ_WRITE,__pinmodeC_bits);
/***************************************************************************
General Purpose Input/Output ports (GPIO)
 ***************************************************************************/
//Not Implemented Yet
/***************************************************************************
Ethernet controller
 ***************************************************************************/
//Not Implemented Yet
/***************************************************************************
CAN controller
 ***************************************************************************/
//Not Implemented Yet
/***************************************************************************
USB device controller
 ***************************************************************************/
//Not Implemented Yet
/***************************************************************************
Universal Asynchronous Receiver Transmitter 0,2,3
 ***************************************************************************/
//Not Implemented Yet
/***************************************************************************
Universal Asynchronous Receiver Transmitter 1
 ***************************************************************************/
```

```
//Not Implemented Yet
/*****************************************************************************
SPI interface SPI0
 *****************************************************************************/
//Not Implemented Yet
/*****************************************************************************
SSP Interface
 *****************************************************************************/
__IO_REG16_BIT(SSP0CR0,             0xE0068000,__READ_WRITE,__sspcr0_bits);
__IO_REG8_BIT(SSP0CR1,              0xE0068004,__READ_WRITE,__sspcr1_bits);
__IO_REG16(SSP0DR,                  0xE0068008,__READ_WRITE);
__IO_REG8_BIT(SSP0STAT,             0xE006800C,__READ,__sspstat_bits);
__IO_REG8(SSP0CPSR,                 0xE0068010,__READ_WRITE);
__IO_REG8_BIT(SSP0IMSC,             0xE0068014,__READ_WRITE,__sspir_bits);
__IO_REG8_BIT(SSP0RIS,              0xE0068018,__READ_WRITE,__sspir_bits);
__IO_REG8_BIT(SSP0MIS,              0xE006801C,__READ_WRITE,__sspir_bits);
__IO_REG8_BIT(SSP0ICR,              0xE0068020,__READ_WRITE,__sspicr_bits);
__IO_REG16_BIT(SSP0DMACR,           0xE0068024,__READ_WRITE,__sspdma_bits);
__IO_REG16_BIT(SSP1CR0,             0xE0030000,__READ_WRITE,__sspcr0_bits);
__IO_REG8_BIT(SSP1CR1,              0xE0030004,__READ_WRITE,__sspcr1_bits);
__IO_REG16(SSP1DR,                  0xE0030008,__READ_WRITE);
__IO_REG8_BIT(SSP1STAT,             0xE003000C,__READ,__sspstat_bits);
__IO_REG8(SSP1CPSR,                 0xE0030010,__READ_WRITE);
__IO_REG8_BIT(SSP1IMSC,             0xE0030014,__READ_WRITE,__sspir_bits);
__IO_REG8_BIT(SSP1RIS,              0xE0030018,__READ_WRITE,__sspir_bits);
__IO_REG8_BIT(SSP1MIS,              0xE003001C,__READ_WRITE,__sspir_bits);
__IO_REG8_BIT(SSP1ICR,              0xE0030020,__READ_WRITE,__sspicr_bits);
__IO_REG16_BIT(SSP1DMACR,           0xE0030024,__READ_WRITE,__sspdma_bits);
/*****************************************************************************
SD_MMC card interface
 *****************************************************************************/
//Not Implemented Yet
/*****************************************************************************
I2C interfaces I2C0, I2C1, I2C2
 *****************************************************************************/
//Not Implemented Yet
/*****************************************************************************
I2S
 *****************************************************************************/
//Not Implemented Yet
/*****************************************************************************
Timer0, 1, 2, 3
 *****************************************************************************/
//Not Implemented Yet
/*****************************************************************************
Watchdog Timer (WDT)
 *****************************************************************************/
//Not Implemented Yet
/*****************************************************************************
Pulse Width Modulator (PWM)
 *****************************************************************************/
//Not Implemented Yet
/*****************************************************************************
Analog-to-Digital Converter (ADC)
 *****************************************************************************/
//Not Implemented Yet
/*****************************************************************************
Digital-to-Analog Converter (DAC)
 *****************************************************************************/
//Not Implemented Yet

/*****************************************************************************
Real Time Clock (RTC) and battery RAM
 *****************************************************************************/
//Not Implemented Yet
/*****************************************************************************
General Purpose DMA controller (GPDMA)
 *****************************************************************************/
//Not Implemented Yet
/*****************************************************************************
 **  Assembler-specific declarations
 *****************************************************************************/

#ifdef __IAR_SYSTEMS_ASM__
#endif    /* __IAR_SYSTEMS_ASM__ */
/*****************************************************************************
 **
 **  Interrupt vector table
 **
 *****************************************************************************/
#endif    /* __IOLPC3180_H */
```

```
/*****************************************************************************
**   SPI0 Initialize
*****************************************************************************/
void SPI0Init(void){  //Transmit TX
    PCONP |= (1 << 21);
    /* Pin Select for MISO0, MOSI0, SSEL0, SCK0 */
    PINSEL0 |= 0x80000000;
    PINSEL1 |= 0x0000002A;

    /* Set DSS data to 8-bit, Frame format SPI, CPOL = 0, CPHA = 0, and SCR is 15 */
    SSP0CR0  = 0x0087;                      //Resets CR0 Register


    /* SSPCPSR clock prescale register, master mode, minimum divisor is 0x02 */
    SSP0CPSR = 10;



    /* Device select as master, SSP Enabled, loopback operational mode */
    SSP0CR1 = 0x02;
    //while(SSP0CR1 &= 0x02);

    for ( i = 0; i < 20; i++ )
    {
              Dummy = SSP0DR;            /* clear the RxFIFO */
    }
    SSP0DMACR=0x02; //TX Only


}
```

```
/*****************************************************************************
** DMAInit
*****************************************************************************/
void DMA_Init()
{
     //Power Up GPDMA
     PCONP |= (1<<29);


     //Clear Interupts
     GPDMA_INT_TCCLR = 0x03;
     GPDMA_INT_ERR_CLR = 0x03;


     //DMA Channel 0
     GPDMA_CH0_SRC =  DMA_SRC;
     GPDMA_CH0_DEST = DMA_SSP0DR;


     GPDMA_CH0_CTRL = (3) | (0x01 << 12) | (0x01 << 15)
                          | (1 << 26) | 0x80000000;
     GPDMA_CONFIG = 0x01;  // Enable DMA channels, little endian
       while ( !(GPDMA_CONFIG & 0x01) );

     GPDMA_CH0_CFG |= 0x08001 | (0x00 << 6) | (0x01 << 11);




     //DMA Channel 1
     GPDMA_CH1_SRC = DMA_SSP1DR;
     GPDMA_CH1_DEST = DMA_DST;

     GPDMA_CH1_CTRL = (3) | (0x01 << 12) | (0x01 << 15)
                          | (1 << 27) | 0x80000000;
     GPDMA_CONFIG = 0x01;  // Enable DMA channels, little endian
       while ( !(GPDMA_CONFIG & 0x01) );
     GPDMA_CH1_CFG |= 0x08001 | (0x03 << 1) | (0x02 << 11);

}
```

```
/*****************************************************************************
*
*   cantest.c:  CAN test module file for NXP LPC23xx Family Microprocessors
*
*   Copyright(C) 2006, NXP Semiconductor
*   All rights reserved.
*
*   History
*   2006.09.13  ver 1.00    Prelimnary version, first Release
*
*   Revised by: Sam LEE
*               Cornell University
*               Biorobotics and Locomotion Lab
*               3 May 2007
*****************************************************************************/
#include "LPC23xx.h"                /* LPC23xx definitions */
#include "type.h"
#include "irq.h"
#include "target.h"
#include "can.h"
#include "fio.h"

CAN_MSG MsgBuf_TX1, MsgBuf_TX2; // TX and RX Buffers for CAN message
CAN_MSG MsgBuf_RX1, MsgBuf_RX2; // TX and RX Buffers for CAN message

volatile DWORD CAN1RxDone, CAN2RxDone;


int main( void )
{
  unsigned int track = 0x01,tt,counter,temp=0x00ff;
  CAN_Init( BITRATE100K28_8MHZ );


  // Initialize MsgBuf
  MsgBuf_TX1.Frame = 0x80080000; // 29-bit, no RTR, DLC is 8 bytes
  MsgBuf_TX1.MsgID = 0x00012345; // CAN ID
  MsgBuf_TX1.DataA = 0x3C3C3C3C;
  MsgBuf_TX1.DataB = 0xC3C3C3C3;

  MsgBuf_RX2.Frame = 0x0;
  MsgBuf_RX2.MsgID = 0x0;
  MsgBuf_RX2.DataA = 0x0;
  MsgBuf_RX2.DataB = 0x0;
  CAN_SetACCF( ACCF_BYPASS );

  IENABLE;  //This Line Added to the Example Code to Make it Work

  while(1){

    while ( !(CAN1GSR & (1 << 3))==(1 << 3) );

    while ( CAN1_SendMessage( &MsgBuf_TX1 ) == FALSE );

    if ( CAN2RxDone == TRUE )
    {
      CAN2RxDone = FALSE;
```

```
      if ( MsgBuf_RX2.Frame & (1 << 10) ) /* by pass mode */
      {
        MsgBuf_RX2.Frame &= ~(1 << 10 );
      }
      if ( ( MsgBuf_TX1.Frame != MsgBuf_RX2.Frame ) ||
          ( MsgBuf_TX1.MsgID != MsgBuf_RX2.MsgID ) ||
            ( MsgBuf_TX1.DataA != MsgBuf_RX2.DataA ) ||
              ( MsgBuf_TX1.DataB != MsgBuf_RX2.DataB ) )
      {
        while ( 1 ){
          for(tt=0;tt<8;tt++){
            FIO2CLR = 0x000000FF;
            FIO2SET = track;
            for(counter=0;counter<1000000;counter++);
            FIO2CLR = track;
            track= track<<1;
          }
          track=0x01;
        }
      }
      // Everything is correct, reset buffer
      MsgBuf_RX2.Frame = 0x0;
      MsgBuf_RX2.MsgID = 0x0;
      MsgBuf_RX2.DataA = 0x0;
      MsgBuf_RX2.DataB = 0x0;
      temp=0x0f;
    } // Message on CAN 2 received
  }

}
/*********************************************************************************
**
**                              End Of File
*********************************************************************************/
```

```
/*****************************************************************************
**
** Function name:       ConfigurePLL
**
** Descriptions:        Configure PLL switching to main OSC instead of IRC
**                              at power up and wake up from power down.
**                              This routine is used in TargetResetInit()
**                              and those examples using power down and
**                              wake up such as USB suspend to resume,
**                              ethernet WOL and power management example
**
*****************************************************************************
*/
__arm void ConfigurePLL ( void )
{
    DWORD MValue, NValue;

    if ( PLLSTAT & (1 << 25) )
    {
            PLLCON = 1;       /* Enable PLL, disconnected */
            PLLFEED = 0xaa;
            PLLFEED = 0x55;
    }

    PLLCON = 0;                     /* Disable PLL, disconnected */
    PLLFEED = 0xaa;
    PLLFEED = 0x55;

    SCS |= 0x20;              /* Enable main OSC */
    while( !(SCS & 0x40) );           /* Wait until main OSC is usable */

    CLKSRCSEL = 0x1;          /* select main OSC, 12MHz, as the PLL clock
source */

    PLLCFG = PLL_MValue | (PLL_NValue << 16);
    PLLFEED = 0xaa;
    PLLFEED = 0x55;

    PLLCON = 1;                     /* Enable PLL, disconnected */
    PLLFEED = 0xaa;
    PLLFEED = 0x55;

    CCLKCFG = CCLKDivValue;           /* Set clock divider */
    USBCLKCFG = USBCLKDivValue;           /* usbclk = 288 MHz/6 = 48 MHz */


    while ( ((PLLSTAT & (1 << 26)) == 0) );     /* Check lock bit status */


    PLLCON = 3;                     /* enable and connect */
    PLLFEED = 0xaa;
    PLLFEED = 0x55;

    while ( ((PLLSTAT & (1 << 25)) == 0) );     /* Check connect bit status
*/
    return;
}
```

```
/*****************************************************************************
* CLOCK INITIALIZATION ******************************************************
*****************************************************************************/
void ClockInit (void)
{
  // Set Clk dividers
  HCLKDIV_CTRL_bit.HCLK = 2-1;        // 1/2  Pll_clk_out
  HCLKDIV_CTRL_bit.PERIPH_CLK = 16-1; // 1/16 Pll_clk_out
  // PLL Init - OSC * 16 = 208MHz
  HCLKPLL_CTRL_bit.BYPASS   = 0;      // OSC connected to PLL input
  HCLKPLL_CTRL_bit.DIRECT   = 1;      // CCO connected to PLL_Clk output
  HCLKPLL_CTRL_bit.FEEDBACK = 0;      // CC0 connected to N divider input
  HCLKPLL_CTRL_bit.N = 0;             // set divide
  HCLKPLL_CTRL_bit.M = 16-1;          // set multiplier
  HCLKPLL_CTRL_bit.POWER_DOWN = 1;    // Enable Pll
  // Wait until PLL lock
  while(!HCLKPLL_CTRL_bit.PLL_LOCK);
  // Connect Pll_clk_out
  PWR_CTRL_bit.RUN_MODE = 1;
}
```

```
/*****************************************************************************
* SPI1 INITIALIZATION ********************************************************
*****************************************************************************/
void SPI1Init(void){

     START_ER_PIN_bit.SPI1_DATIN = 1; //Power up Pin

     //Set SPI Control Register
     SPI_CTRL_bit.SPI1_CLK_ENA   = 1;
     SPI_CTRL_bit.SPI1_PIN_SEL   = 1;
     SPI_CTRL_bit.SPI1_CLK_OUT   = 1;
     SPI_CTRL_bit.SPI1_DATIO     = 1;

     SPI1_GLOBAL              = 0;       // disable SPI1

     SPI1_FRM                 = 1;       //1 Frame per transmission

     //SPI Control Register
     SPI1_CON                 = 0;       // reset SPI1_CON register
     //SPI1_CON_bit.unidir      = 1;     // Unidirectional Pins
     SPI1_CON_bit.rxtx        = 1;       // transmit
     SPI1_CON_bit.thr         = 1;       // FIFO treshold enabled
     SPI1_CON_bit.shift_off   = 0;       // Enable clock generation
     SPI1_CON_bit.bitnum      = 7;       // 8bits to be tx or rx
     SPI1_CON_bit.ms          = 1;       // SPI operating as a master
     SPI1_CON_bit.rate        = SPI_RATE;// SPI transfer rate

     //SPI Interrupt Enable Register
     SPI1_IER_bit.inteot      = 1;       // End of Transfer Int Enabled
     SPI1_IER_bit.intthr      = 1;       // Threshold Int Enabled

     //Enable Interrupts on Sub Interrupt Controller 1
     SIC1_APR_bit.SPI1_INT    = 1;       // Int on high or rising edge
     SIC1_ATR_bit.SPI1_INT    = 0;       // Interrupt is level sensitive
     SIC1_ITR_bit.SPI1_INT    = 0;       // Interrupt is routed to the
     SIC1_ER_bit.SPI1_INT     = 1;       // SPI1 Interupt Enable

     SPI1_GLOBAL_bit.enable   = 1;       // Enable SPI1
     while (!(SPI1_GLOBAL_bit.enable));
}
```

```
/*****************************************************************************
* DMA INITIALIZATION ********************************************************
*****************************************************************************/
void DMAInit(void){
     DMACLK_CTRL_bit.DMA_CLK_ENA    = 1; //All Clocks to DMA enabled



     //Clear Channel Error Interrupts
     DMACIntErrClr_bit.DMA_CH0      = 1;
     DMACIntErrClr_bit.DMA_CH1      = 1;
     DMACIntTCClear_bit.DMA_CH0     = 1;
     DMACIntTCClear_bit.DMA_CH1     = 1;

     //Source and Destination Registers
     DMACC0SrcAddr                  = (long)&tx[0];
     DMACC0DestAddr                 = (volatile)&SPI1_DAT;
     DMACC1SrcAddr                  = (volatile)&SPI2_DAT;
     DMACC1DestAddr                 = (long)&rx[0];

     //Disable Linked List Item
     DMACC0LLI                      = 0;
     DMACC1LLI                      = 0;

     //DMA Ch0 Control Register
     DMACC0Control_bit.I            = 1;  //Terminal Count Interupt disabled
     DMACC0Control_bit.Prot3        = 1;  //Access Cacheable
     DMACC0Control_bit.Prot2        = 1;  //Access Bufferable
     DMACC0Control_bit.Prot1        = 1;  //Access Privileged Mode
     DMACC0Control_bit.DI           = 0;  //Destination Address not
incremented
     DMACC0Control_bit.SI           = 1;  //Source Address incremented
     DMACC0Control_bit.D            = 0;  //AHB Master 0 for destination tfr
     DMACC0Control_bit.S            = 0;  //AHB Master 0 for source tfr
     DMACC0Control_bit.DWidth       = 0;  //8bits
     DMACC0Control_bit.SWidth       = 0;  //8bits
     DMACC0Control_bit.DBSize       = 0;  //Burst Size 1
     DMACC0Control_bit.SBSize       = 0;  //Burst Size 1
     DMACC0Control_bit.TransferSize = 3; //TransferSize

     //DMA Ch1 Control Register
     DMACC1Control_bit.I            = 0;  //Terminal Count Interupt disabled
     DMACC1Control_bit.Prot3        = 1;  //Access Cacheable
     DMACC1Control_bit.Prot2        = 1;  //Access Bufferable
     DMACC1Control_bit.Prot1        = 1;  //Access Privileged Mode
     DMACC1Control_bit.DI           = 1;  //Destination Address incremented
     DMACC1Control_bit.SI           = 0;  //Source Address not incremented
     DMACC1Control_bit.D            = 0;  //AHB Master 0 for destination tfr
     DMACC1Control_bit.S            = 0;  //AHB Master 0 for source tfr
     DMACC1Control_bit.DWidth       = 0;  //8bits
     DMACC1Control_bit.SWidth       = 0;  //8bits
     DMACC1Control_bit.DBSize       = 0;  //Burst Size 1
     DMACC1Control_bit.SBSize       = 0;  //Burst Size 1
     DMACC1Control_bit.TransferSize = 3; //TransferSize


     //DMA Ch0 Configuration Register
```

```
    DMACC0Config_bit.H             = 0;  //Enable DMA requests
    DMACC0Config_bit.ITC           = 0;  //Disable Terminal Count Int.
    DMACC0Config_bit.IE            = 0;  //Disable Error Int.
    DMACC0Config_bit.FlowCntrl     = 1;  //Memory to Peripheral
    DMACC0Config_bit.DestPeripheral= 11; //Destination Peropheral
    DMACC0Config_bit.SrcPeripheral = 1;  //Source Peripheral
    DMACC0Config_bit.E             = 1;  //Enable Ch0

    //DMA Ch1 Configuration Register
    DMACC1Config_bit.H             = 0;  //Enable DMA requests
    DMACC1Config_bit.ITC           = 0;  //Disable Terminal Count Int.
    DMACC1Config_bit.IE            = 0;  //Disable Error Int.
    DMACC1Config_bit.FlowCntrl     = 2;  //Peripheral to Memory
    DMACC1Config_bit.DestPeripheral= 1;  //Destination Peropheral
    DMACC1Config_bit.SrcPeripheral = 3;  //Source Peripheral
    DMACC1Config_bit.E              = 1; //Enable Ch1

    DMACConfig_bit.E               = 1; //DMA Controller Enable
}
```

```
/*****************************************************************************
* INCLUDE FILES ***********************************************************
*****************************************************************************/
#include <inarm.h>
#include <iolpc3180.h>
#include "arm926ej_cp15_drv.h"
#include "ttbl.h"

/*****************************************************************************
* DEFINITIONS *************************************************************
*****************************************************************************/
#define OSC         (13000000UL)        // System OSC 13MHz
#define CORE_CLK    (OSC*16)            // ARM_CLK 208MHz
#define AHB_CLK     (CORE_CLK/2)        // HCLK 104MHz
#define PER_CLK     (CORE_CLK/16)       // PER_CLK 13MHz
#define RTC_CLK     (32768UL)           // RTC_CLK

#define LED_D400    (1UL << 2)
#define LED_D401    (1UL << 3)
#define LED_D402    (1UL << 7)
#define LED_D403    (1UL << 6)
#define button      (1UL << 7)


#define SPI_RATE    0x07                // 6.5Mhz
/*****************************************************************************
* VARIABLES ***************************************************************
*****************************************************************************/
unsigned char ledstatus=0;
unsigned int test,time,blah0,blah1;
unsigned char tx[60];
unsigned char rx[60];


unsigned char i;
unsigned char temp=0;

/*****************************************************************************
* MS INTERUPT HANDLER *****************************************************
*****************************************************************************/
void mstimerInterupt(void){
  MSTIM_INT_bit.MATCH0_INT=1;           //Clears interupt flag
    if(ledstatus==0){
      PIO_OUTP_SET = LED_D400 | LED_D401 | LED_D402 | LED_D403;
      ledstatus=1;
    }
    else{
      PIO_OUTP_CLR = LED_D400 | LED_D401 | LED_D402 | LED_D403;
      ledstatus=0;
    }
}
/*****************************************************************************
* IRQ HANDLER *************************************************************
*****************************************************************************/
__irq __arm void irq_handler (void){

  if(MIC_SR_bit.MSTIMER_INT){
   mstimerInterupt();
  }
  if(SIC1_SR_bit.SPI1_INT){
  }
}
```

```
/*****************************************************************************
* CLOCK INITIALIZATION ******************************************************
*****************************************************************************/
void ClockInit (void)
{
  // Set Clk dividers
  HCLKDIV_CTRL_bit.HCLK = 2-1;        // 1/2  Pll_clk_out
  HCLKDIV_CTRL_bit.PERIPH_CLK = 16-1; // 1/16 Pll_clk_out
  // PLL Init - OSC * 16 = 208MHz
  HCLKPLL_CTRL_bit.BYPASS   = 0;      // OSC connected to PLL input
  HCLKPLL_CTRL_bit.DIRECT   = 1;      // CCO connected to PLL_Clk output
  HCLKPLL_CTRL_bit.FEEDBACK = 0;      // CC0 connected to N divider input
  HCLKPLL_CTRL_bit.N = 0;             // set divide
  HCLKPLL_CTRL_bit.M = 16-1;          // set multiplier
  HCLKPLL_CTRL_bit.POWER_DOWN = 1;    // Enable Pll
  // Wait until PLL lock
  while(!HCLKPLL_CTRL_bit.PLL_LOCK);
  // Connect Pll_clk_out
  PWR_CTRL_bit.RUN_MODE = 1;
}

/*****************************************************************************
* MSTIMER INITIALIZATION ****************************************************
*****************************************************************************/
void mstimerInit(void){
  MSTIM_CTRL_bit.COUNT_ENAB     = 0;  // Stop counting
  MSTIM_CTRL_bit.PAUSE_EN       = 0;  //runs in debug mode

  //Match0
  MSTIM_MCTRL_bit.MR0_INT       = 1;  //Enable Interupt Status Generation Match0
  MSTIM_MCTRL_bit.RESET_COUNT0  = 1;  //Enable Reset of Timer Counter on Match0
  MSTIM_MCTRL_bit.STOP_COUNT0   = 0;  //Disable Stop Functionality on Match0

  MSTIM_CTRL_bit.RESET_COUNT = 1;     // Reset the counter
  while(MSTIM_COUNTER);
  MSTIM_CTRL_bit.RESET_COUNT = 0;     // release reset of the counter

  MSTIM_MATCH0  = 32768;              //Match Value for Match0

  MIC_APR_bit.MSTIMER_INT    = 1;     // Int generated on a high signal or rising edge
  MIC_ATR_bit.MSTIMER_INT    = 0;     // Interrupt is level sensitive
  MIC_ITR_bit.MSTIMER_INT    = 0;     // Int routed to the IRQ
  MIC_ER_bit.MSTIMER_INT     = 1;     // Enable Milisecond timer interrupts
  MSTIM_CTRL_bit.COUNT_ENAB  = 1;     // Enable counting
}
/*****************************************************************************
* SPI1 INITIALIZATION *******************************************************
*****************************************************************************/
void SPI1Init(void){

     START_ER_PIN_bit.SPI1_DATIN = 1;   //Power up Pin

     //Set SPI Control Register
     SPI_CTRL_bit.SPI1_CLK_ENA   = 1;
     SPI_CTRL_bit.SPI1_PIN_SEL   = 1;
     SPI_CTRL_bit.SPI1_CLK_OUT   = 1;
     SPI_CTRL_bit.SPI1_DATIO     = 1;

     SPI1_GLOBAL              = 0;      // disable SPI1

     SPI1_FRM                 = 1;      //1 Frame per transmission
```

```
     //SPI Control Register
     SPI1_CON                 = 0;            // reset SPI1_CON register
     //SPI1_CON_bit.unidir      = 1;          // Unidirectional Pins
     SPI1_CON_bit.rxtx        = 1;            // transmit
     SPI1_CON_bit.thr         = 1;            // FIFO treshold enabled
     SPI1_CON_bit.shift_off   = 0;            // Enable clock generation
     SPI1_CON_bit.bitnum      = 7;            // 8bits to be tx or rx
     SPI1_CON_bit.ms          = 1;            // SPI operating as a master
     SPI1_CON_bit.rate        = SPI_RATE;     // SPI transfer rate

     //SPI Interupt Enable Register
     SPI1_IER_bit.inteot      = 1;            // End of Transfer Int Enabled
     SPI1_IER_bit.intthr      = 1;            // Threshold Int Enabled

     //Enable Interupts on Sub Interrupt Controller 1
     SIC1_APR_bit.SPI1_INT    = 1;      // Int generated on a high or rising edge
     SIC1_ATR_bit.SPI1_INT    = 0;      // Interrupt is level sensitive
     SIC1_ITR_bit.SPI1_INT    = 0;      // The interrupt is routed to the IRQ
     SIC1_ER_bit.SPI1_INT     = 1;      // SPI1 Interupt Enable

     SPI1_GLOBAL_bit.enable   = 1;      // Enable SPI2
     while (!(SPI1_GLOBAL_bit.enable));
}
/*********************************************************************
* SPI2 INITIALIZATION ************************************************
*********************************************************************/
void SPI2Init(void){
     START_ER_PIN_bit.SPI2_DATIN = 1; //Power up Pin

     //Set SPI Control Register
     SPI_CTRL_bit.SPI2_CLK_ENA   = 1;
     SPI_CTRL_bit.SPI2_PIN_SEL   = 1;
     SPI_CTRL_bit.SPI2_CLK_OUT   = 1;
     SPI_CTRL_bit.SPI2_DATIO     = 1;

     SPI2_GLOBAL              = 0;            // disable SPI2

     SPI2_FRM                 = 1;            //1 Frame per transmission


     //SPI Control Register
     SPI2_CON                 = 0;            // reset SPI2_CON register
     //SPI2_CON_bit.unidir      = 1;          // Unidirectional Pins
     SPI2_CON_bit.rxtx        = 0;            // receive
     SPI2_CON_bit.thr         = 1;            // FIFO treshold enabled
     SPI2_CON_bit.shift_off   = 1;            // Disable clock generation
     SPI2_CON_bit.bitnum      = 7;            // 8bits to be tx or rx
     SPI2_CON_bit.ms          = 0;            // SPI operating as a master
     SPI2_CON_bit.rate        = SPI_RATE;      // SPI transfer rate

     //SPI Interupt Enable Register
     SPI2_IER_bit.inteot      = 1;      // End of Transfer Int Enabled
     SPI2_IER_bit.intthr      = 1;      // Threshold Int Enabled

     //Enable Interupts on Sub Interrupt Controller 1
     SIC1_APR_bit.SPI2_INT    = 1;      // Int generated on a high or rising edge
     SIC1_ATR_bit.SPI2_INT    = 0;      // Interrupt is level sensitive
     SIC1_ITR_bit.SPI2_INT    = 0;      // The interrupt is routed to the IRQ
     SIC1_ER_bit.SPI2_INT     = 1;      // SPI2 Interupt Enable
```

```
     SPI2_GLOBAL_bit.enable   = 1;      //Enable SPI2
     while (!(SPI2_GLOBAL_bit.enable));
}

/*************************************************************************
* DMA INITIALIZATION ****************************************************
*************************************************************************/
void DMAInit(void){
     DMACLK_CTRL_bit.DMA_CLK_ENA    = 1; //All Clocks to DMA enabled



     //Clear Channel Error Interrupts
     DMACIntErrClr_bit.DMA_CH0      = 1;
     DMACIntErrClr_bit.DMA_CH1      = 1;
     DMACIntTCClear_bit.DMA_CH0     = 1;
     DMACIntTCClear_bit.DMA_CH1     = 1;

     //Source and Destination Registers
     DMACC0SrcAddr                  = (long)&tx[0];
     DMACC0DestAddr                 = (volatile)&SPI1_DAT;
     DMACC1SrcAddr                  = (volatile)&SPI2_DAT;
     DMACC1DestAddr                 = (long)&rx[0];

     //Disable Linked List Item
     DMACC0LLI                      = 0;
     DMACC1LLI                      = 0;

     //DMA Ch0 Control Register
     DMACC0Control_bit.I            = 1;  //Terminal Count Interupt disabled
     DMACC0Control_bit.Prot3        = 1;  //Access Cacheable
     DMACC0Control_bit.Prot2        = 1;  //Access Bufferable
     DMACC0Control_bit.Prot1        = 1;  //Access Privileged Mode
     DMACC0Control_bit.DI           = 0;  //Destination Address not incremented
     DMACC0Control_bit.SI           = 1;  //Source Address incremented
     DMACC0Control_bit.D            = 0;  //AHB Master 0 for destination tfr
     DMACC0Control_bit.S            = 0;  //AHB Master 0 for source tfr
     DMACC0Control_bit.DWidth       = 0;  //8bits
     DMACC0Control_bit.SWidth       = 0;  //8bits
     DMACC0Control_bit.DBSize       = 0;  //Burst Size 1
     DMACC0Control_bit.SBSize       = 0;  //Burst Size 1
     DMACC0Control_bit.TransferSize = 3; //TransferSize

     //DMA Ch1 Control Register
     DMACC1Control_bit.I            = 0;  //Terminal Count Interupt disabled
     DMACC1Control_bit.Prot3        = 1;  //Access Cacheable
     DMACC1Control_bit.Prot2        = 1;  //Access Bufferable
     DMACC1Control_bit.Prot1        = 1;  //Access Privileged Mode
     DMACC1Control_bit.DI           = 1;  //Destination Address incremented
     DMACC1Control_bit.SI           = 0;  //Source Address not incremented
     DMACC1Control_bit.D            = 0;  //AHB Master 0 for destination tfr
     DMACC1Control_bit.S            = 0;  //AHB Master 0 for source tfr
     DMACC1Control_bit.DWidth       = 0;  //8bits
     DMACC1Control_bit.SWidth       = 0;  //8bits
     DMACC1Control_bit.DBSize       = 0;  //Burst Size 1
     DMACC1Control_bit.SBSize       = 0;  //Burst Size 1
     DMACC1Control_bit.TransferSize = 3; //TransferSize


     //DMA Ch0 Configuration Register
     DMACC0Config_bit.H             = 0;  //Enable DMA requests
     DMACC0Config_bit.ITC           = 0;  //Disable Terminal Count Int.
     DMACC0Config_bit.IE            = 0;  //Disable Error Int.
```

```
    DMACC0Config_bit.FlowCntrl     = 1;  //Memory to Peripheral
    DMACC0Config_bit.DestPeripheral= 11; //Destination Peropheral
    DMACC0Config_bit.SrcPeripheral = 1;  //Source Peripheral
    DMACC0Config_bit.E             = 1;  //Enable Ch0

    //DMA Ch1 Configuration Register
    DMACC1Config_bit.H             = 0;  //Enable DMA requests
    DMACC1Config_bit.ITC           = 0;  //Disable Terminal Count Int.
    DMACC1Config_bit.IE            = 0;  //Disable Error Int.
    DMACC1Config_bit.FlowCntrl     = 2;  //Peripheral to Memory
    DMACC1Config_bit.DestPeripheral= 1;  //Destination Peropheral
    DMACC1Config_bit.SrcPeripheral = 3;  //Source Peripheral
    DMACC1Config_bit.E              = 1; //Enable Ch1

    DMACConfig_bit.E               = 1; //DMA Controller Enable
}
/*****************************************************************************
* INITIALIZATION ROUTINE ****************************************************
*****************************************************************************/
void initialize(void){

  unsigned char *ptrtx = &tx[0];
  char i=0;
  for(i=0;i<60;i++){
     *ptrtx++ = 0x55;
   }


  // Disable all interrupts
  MIC_ER  = 0;
  SIC1_ER = 0;
  SIC2_ER = 0;

  PIO_OUTP_CLR = LED_D400 | LED_D401 | LED_D402 | LED_D403;

  TIMCLK_CTRL_bit.WDT_CLK_ENA = 0; // disable watchdog
  ClockInit();
  mstimerInit();

  SPI1Init();
  SPI2Init();
  DMAInit();


  PIO_OUTP_SET = LED_D400 | LED_D401 | LED_D402 | LED_D403;

}

/*****************************************************************************
* MAIN LOOP *****************************************************************
*****************************************************************************/
void main(void)
{
  initialize();
  __enable_interrupt();


  while(1)
  {

  };

} // main(void)
```