Freescale Semiconductor
Application Note

Document Number: AN4241
Rev. 0, Feb 2011

MPC5668x Family Demonstration Lab Training

Example Projects for MPC5668G/E 32-bit Microcontroller

**by: Steven McLaughlin and Martin Vaupel
Microcontroller Solutions Group
East Kilbride
Scotland**

Contents

1 Introduction

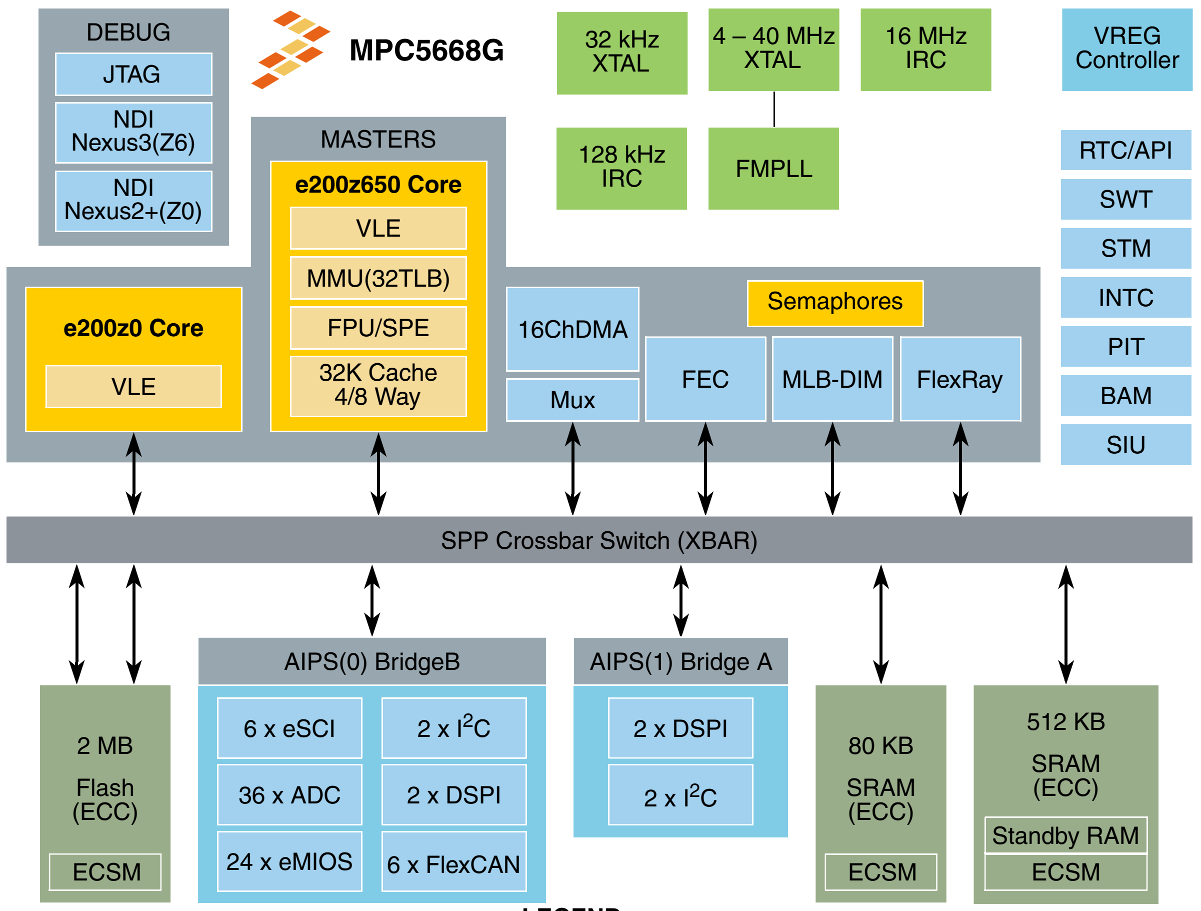
This tutorial application note has been developed to increase the user's knowledge of the MPC5668x 32-bit microcontroller. It contains a series of code examples, written in C. The sample projects have been written with the Green Hills compiler and have been developed on the MPC5668EVB evaluation board.

The setup for the MPC5668EVB should be followed as per the MPC5668EVBUMD guide. Any deviation from this will be outlined within the specific sections.

2 MPC5668x background

The MPC5668G/E 32-bit microcontroller is built on the Power Architecture embedded category and represents an automotive-focused product designed to address gateway and body controller applications.

The device features a dual-core system using the e200z6 and e200z0 cores, capable of execution speeds of up to 128 MHz and 64 MHz respectively. Flash capacity is up to 2 MB and RAM up to 592 KB, both featuring ECC. Being aimed for body and gateway, the MCU features an array of communication modules, including CAN, LIN, FlexRay, MOST, and Ethernet.



LEGEND

ADC	– Analog to Digital Converter
BAM	– Boot Assist Module
DSPI	– Serial Peripherals Interface
ECC	– Error Correction Code
ECSM	– Error Correction Status Module
eMIOS	– Timed Input Output
eDMA	– Enhanced Direct Memory Access controller
eSCI	– Serial Communications Interface
FEC	– Fast Ethernet Controller
FlexCAN	– Controller Area Network controller
FlexRay™	– FlexRay Bus Controller
FMPLL	– Frequency Modulated Phase Locked Loop
I^2C	– Inter IC Controller
INTC	– Interrupt Controller
JTAG	– Joint Test Action Group interface
MLB-DIM	– Media Local Bus Device Interface Module
NDI	– Nexus Debug Interface
PIT	– Periodic Interrupt Timer
RTC	– Real Time Clock
SIU	– System Integration
STM	– System Timer Module
SWT	– Software Watchdog Timer
VREG	– Voltage Regulator

Figure 1. MPC5668G block diagram

Figure 1 illustrates the majority of the modules available on MPC5668x. The reference manual should be consulted for an in-depth description of each one.

2.1 Device initialization

The initialization is common to all the code examples as each uses the z6 core. The z6 core is capable of supporting VLE (variable length instruction) or BOOKE instruction sets, whereas the z0 core only supports VLE encoding.

Another point to note about the cores is that z6 core accesses are executed through the memory management unit (MMU), whereas z0 core accesses are not. The MMU is responsible for address translation, mapping logical addresses to physical addresses. Assembly software must be used to execute translation lookaside buffers (TLB) instructions to initialize the

MMU. The TLB is accessed directly through several MMU assist (MAS) registers where software can read and write MAS registers, using move to SPR (mtspr) and move from SPR (mfspr) instructions. A TLB write entry (tlbwe) instruction writes the MAS register values to a TLB entry.

NOTE

The compiler cannot generate TLB initialization code — the user must do this. This has already been done for each of the code examples, but the user should be aware of this point for application development.

The boot assist module (BAM) is responsible for determining whether to initiate the internal flash boot mode or a serial boot from SCI or CAN. For these examples, make sure the boot configuration jumper (J69) on the EVB is set high (internal flash). The BAM will search for a valid reset configuration half-word (RCHW) during the assertion of reset. The BAM, which is 4 KB of ROM, contains information for configuring the z6 MMU to cover all available MCU address space — internal flash, peripheral bridge, and SRAM — without address translation. Hence the requirement for user intervention for TLB initialization.

2.2 Loading the projects

Within the Green Hills code examples, the TLB entries can be found in the MPC5668.cmm file. For the purposes of these examples, no further TLB entries are required. The code examples are programmed to execute from RAM, unless specified otherwise.

If you are using Trace 32 debugger, the steps given here are required to load a project with these examples:

1. Go to File → Edit Batch File.

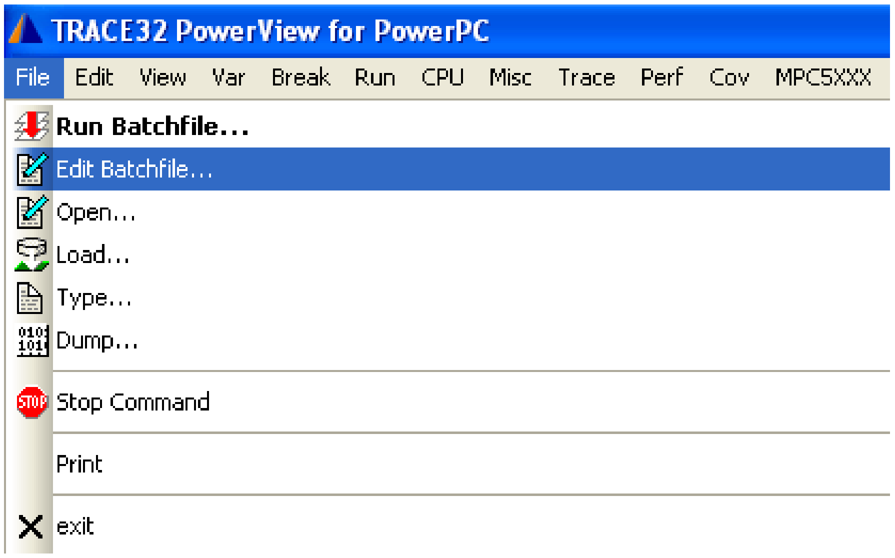


Figure 2. Loading script file with Trace32 debugger

2. Select the appropriate folder, *_GHS, and select MPC5668.cmm (all projects have this name).

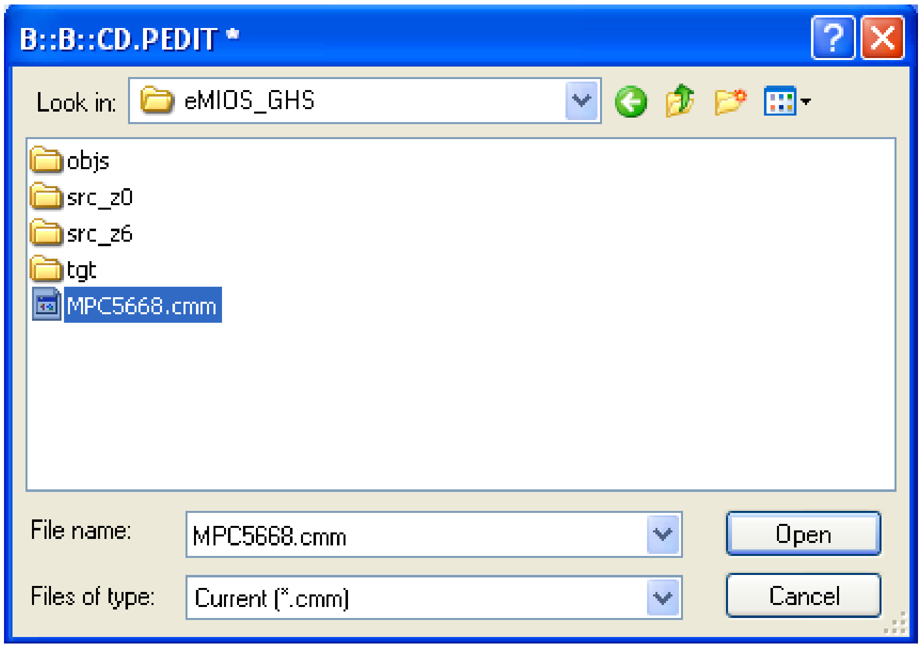
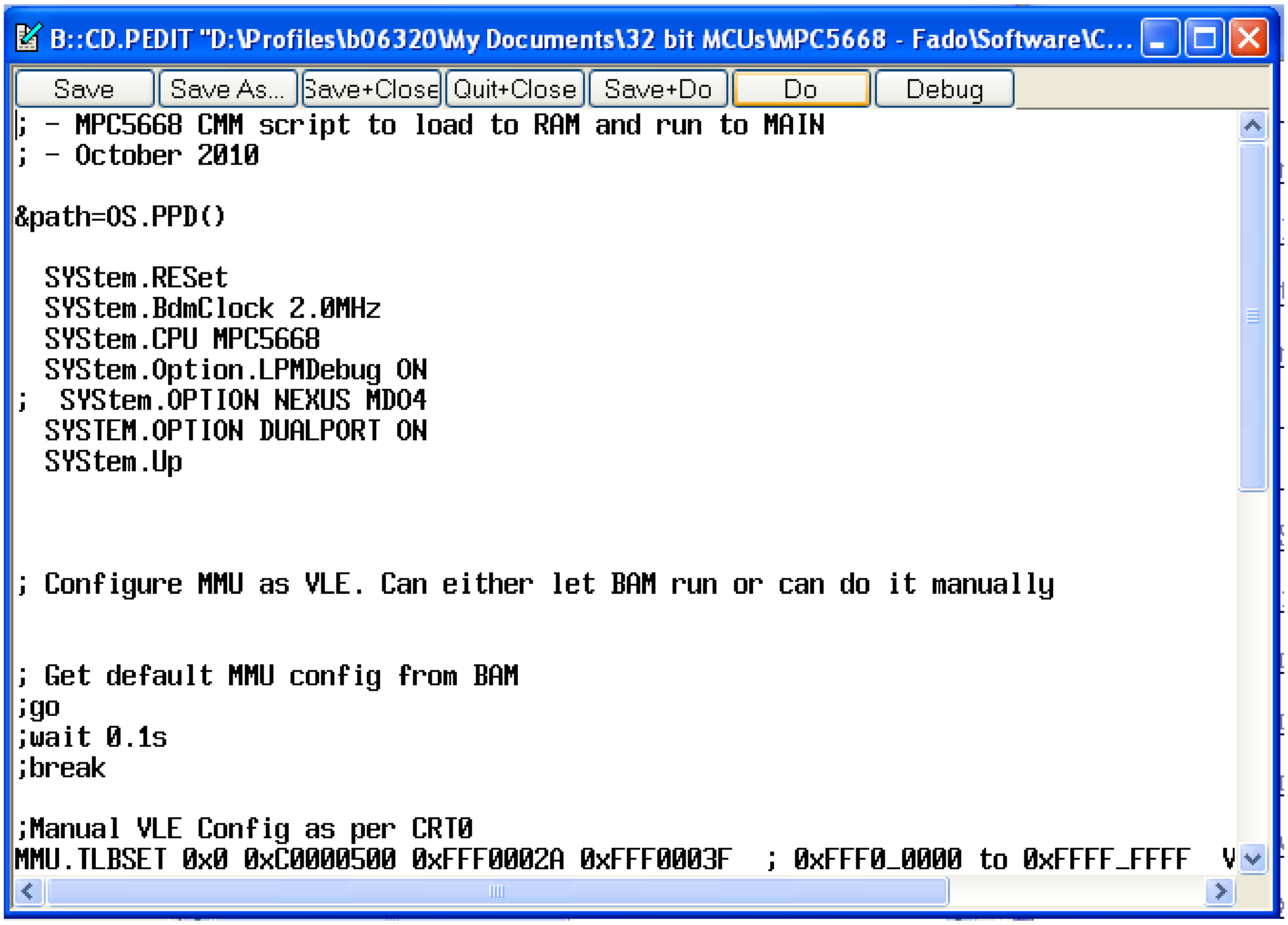


Figure 3. MPC5668 script file (all projects have this name)

3. Click on ‘Do’ within the window which has appeared:


B::CD.PEDIT "D:\Profiles\b06320\My Documents\32 bit MCUs\MPC5668 - Fado\Software\C...
Save
Save As...
Save+Close
Quit+Close
Save+Do
Do
Debug

```
; - MPC5668 CMM script to load to RAM and run to MAIN
; - October 2010

&path=OS.PPD()

  SYStem.RESet
  SYStem.BdmClock 2.0MHz
  SYStem.CPU MPC5668
  SYStem.Option.LPMDebug ON
;  SYStem.OPTION NEXUS MDO4
  SYSTEM.OPTION DUALPORT ON
  SYStem.Up



; Configure MMU as VLE. Can either let BAM run or can do it manually


; Get default MMU config from BAM
;go
;wait 0.1s
;break

;Manual VLE Config as per CRT0
MMU.TLBSET 0x0 0xC0000500 0xFFF0002A 0xFFF0003F  ; 0xFFF0_0000 to 0xFFFF_FFFF  V
```

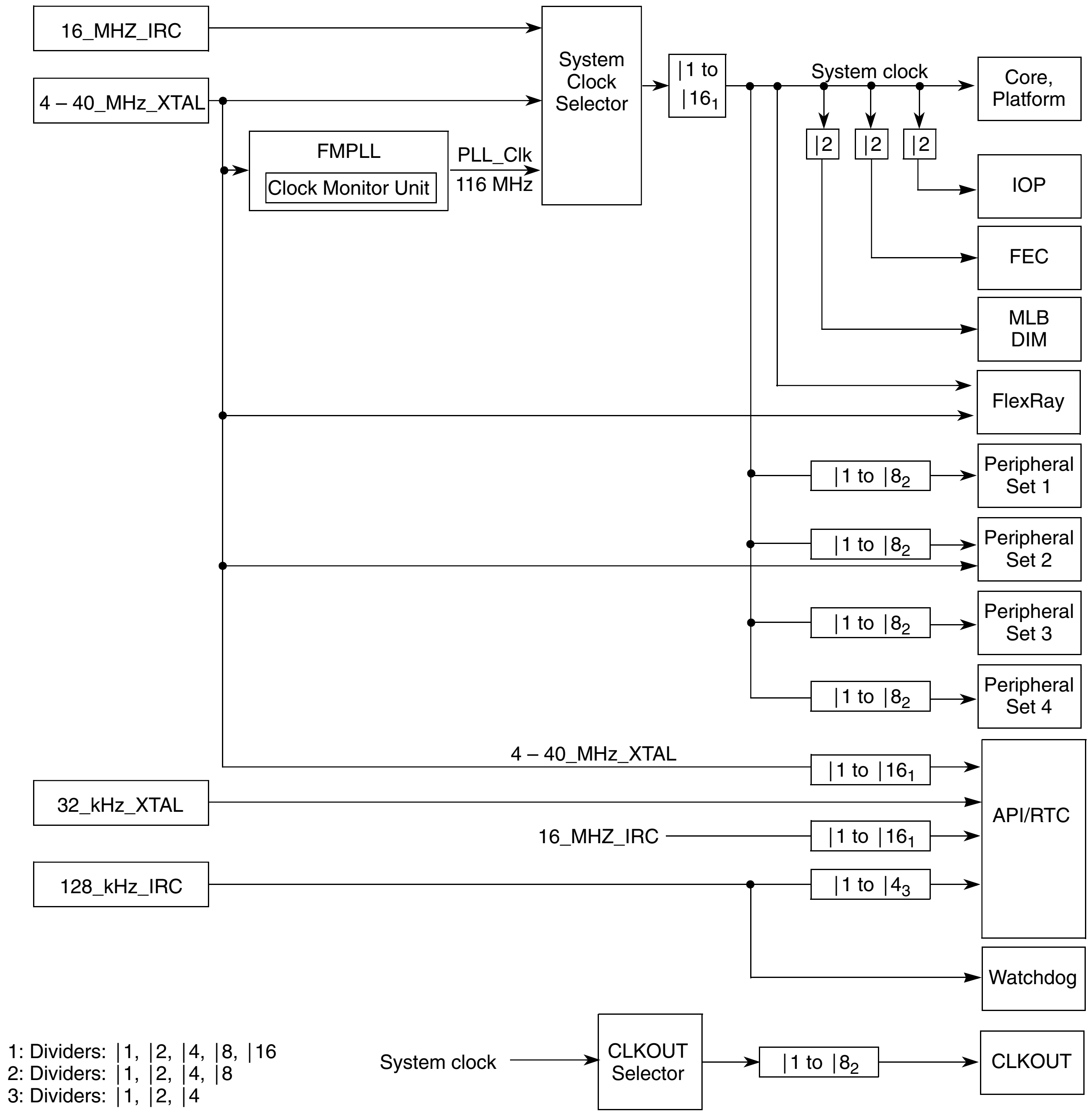
Figure 4. MPC5668 CMM script

The script has been set up to execute directly to the "main" function of the project.

3 MPC5668 system clocks

MPC5668 has a variety of system clocks that serve different application purposes. The source of the clock can come from the following:

- High frequency crystal oscillator in range 4–40 MHz
- Fast on-chip 16 MHz RC
- Slow on-chip 128 kHz RC
- External 32 kHz crystal oscillator
- PLL supporting system clock 40–116 MHz



The example includes the high frequency oscillator, fast IRC, and PLL as the system clock sources. The slow IRC is commonly used for wakeup from low power modes and for the watchdog timer.

3.1 Description

Task

The example code demonstrates the 16 MHz fast IRC, external high frequency 40 MHz crystal, and PLL as the main system clock sources. The clock source can be viewed at the CLKOUT pin (PK9) and PE0 has been enabled to toggle within a while loop. This LED will blink faster with increasing system clock speed.

Hardware setup

The MPC5668EVB should be set up as follows:

1. As per MPC5668EVBUMD, make sure the external 40 MHz crystal is responsible for clocking the MCU.
2. Connect a user switch at P16 to PE0.
3. Connect a user LED at P15 to PE1.
4. Connect a scope probe to PK9. Make sure it is properly grounded.

Exercise

The code should be loaded to the device and by default the 16 MHz IRC is used as the system clock. The user can alter the clock source simply by pressing a switch connected to PE0.

1. 16 MHz IRC is used as the default clock and can be viewed at PK9 on an oscilloscope. Note that the device's individual value is determined during one of the silicon test steps and is stored on the device's test flash block. Until the flash is up and stable, the IRC runs at its untrimmed frequency.
2. On pressing switch 1 (PE0) the program will activate an external 40 MHz crystal as the clock source. The LED will blink quicker and PK9 will show a 40 MHz waveform.
3. On pressing switch 1 again, the system clock will utilize the 16 MHz IRC with the FMPLL as the clock source. Again, the LED will blink quicker and PK9 will show the clock waveform.

3.2 Program setup

The SIU system clock register (SIU_SYSCLK) is responsible for selecting the system clock. It also has responsibility for choosing the system clock divider as well as low power peripheral clock divide values.

The PLL is capable of generating VCO frequencies in the range of 192–600 MHz from an input clock range 4–40 MHz. The ESYNCR1 and ESYNCR2 registers have responsibility for selecting the appropriate divider values for the PLL as well as other advanced bits to control lock behavior.

3.3 Sample code

```
/*************************************************************************/
/* FILE NAME: main.c                              COPYRIGHT (c) Freescale 2010 */
/*                                                         All Rights Reserved */
/* */
/* MPC5668 Example Projects Suite */
/* 1.0 System Clocks Examples */
/* */
/* Revision History */
/* Rev: 1.0 Author:Steven McLaughlin  DATE: 23/07/2010  */
/* */
/* Example demostrates the main running system clocks of MPC5668 and the user*/
/* can toggle through them pressing PE0 and view the toggling LED on PE1 and */
/* attached a scope to CLKOUT on PK9 to view waveform */
/*************************************************************************/
#include "mpc5668.h"

uint32_t count;

void PLLinit(void);
/*********************************** main ************************************/
void main (void)
{
uint32_t sys_clk = 0;
/*----------------------ENABLE CLKOUT------------------------------------*/
SIU.PCR[153].R = 0x060C; /* CLKOUT - PK9 */
SIU.ECCR.B.ECDF = 0;      /* CLKOUT - Set to 0 for sys freq. */
SIU.ECCR.B.ECEN = 1; /* enable CLKOUT */

/*-------------------LED and Switch port init----------------------------*/
SIU.PCR[64].R=0x0100;/* IBE on PE0 - switch connection */
SIU.PCR[65].R=0x0200;/* OBE on PE1 - LED connection */


  while(1)
    {
if (SIU.GPDI[64].R == 1)                 /* SW1 to select system clock on PE1 */
    {
        sys_clk=sys_clk++;
        if (sys_clk == 3)
{
sys_clk=0;
}
}
switch (sys_clk)
{
case(0):
SIU.SYSCLK.B.SYSCLKSEL = 0x0; /* 16MHz IRC as system clock */
break;

case(1):
SIU.SYSCLK.B.SYSCLKSEL = 0x1; /* 40MHz external crystal as system clock */
break;

case(2):
PLLinit();/* 116MHz PLL from IRC */
break;
}

for (count=0; count<100000; count++);/* loops to toggle PE1 */
SIU.GPDO[65].R = 0x01;
for (count=0; count<100000; count++);
SIU.GPDO[65].R = 0x00;
    }

}/* end of main() */

void PLLinit(void)
{
int ERFD = 3;
int EPREDIV = 9;
int EMFD = 112;

/* Select crystal as CLK Source */
SIU.SYSCLK.B.SYSCLKSEL = 0x1;

/* Configure PLL CTRL Regs */
FMPLL.ESYNCR1.B.CLKCFG = 0x7;
FMPLL.ESYNCR2.B.ERFD = ERFD+2;
FMPLL.ESYNCR1.B.EPREDIV = EPREDIV;
FMPLL.ESYNCR1.B.EMFD = EMFD;

/* Wait for PLL to lock */
while(FMPLL.SYNSR.B.LOCK != 1);

/* Change pll up to final freq */
FMPLL.ESYNCR2.B.ERFD = ERFD;

/* Switch from IRC to PLL */
SIU.SYSCLK.B.SYSCLKSEL = 0x2;
SIU.ECCR.B.ECDF = 4;      /* Must have CLKOUT/4 for full speed PLL (116MHz) */

} /* End of PLLinit */
```

4 eMIOS — Modulus up-counter, OPWM, IPWM

The eMIOS200 provides modes to create or measure timed signal events. MPC5668x has a single eMIOS200 module that implements 16-bit counters. MPC5668G implements twenty-four unified channels and MPC5668E implements thirty-two unified channels in the same manner.

4.1 Description

Task

This example demonstrates the eMIOS200 module by utilizing the OPWM and IPWM functions. A 1 kHz pulse train is generated and is output on PG14 (eMIOS ch1). The pulse width (500 µs) is read by PG15 (eMIOS ch0). PE1 is connected to an LED to notify when the correct pulse width has been measured.

Hardware setup

The MPC5668EVB should be set up as follows:

1. As per MPC5668EVBUMD, make sure the external 40 MHz crystal is responsible for clocking the MCU.
2. Connect pins PG14 and PG15.
3. Place a scope probe on (or between PG14 and) PG15 to detect the 1 kHz signal.
4. If desired, the CLKOUT on PK9 has been enabled to check that the system clock being used is the external 40 MHz crystal.

Exercise

The code should be loaded to the device. Run the project and notice that the LED connected to PE1 should blink. This means that a valid input capture match has occurred and the pulse width of the signal being measured on this channel is the same as the signal being generated on the output channel. Essentially, this is demonstrating the output and input capabilities of the eMIOS module.

If the connection between the output pulse and input measurement is removed, you can see that the LED will stop blinking. This is because the measured pulse width at eMIOS ch0 does not match the generated pulse at eMIOS ch1 — therefore the code has been violated and will not commence execution.

4.2 Program setup

The eMIOS clock has been set at 1 MHz via the EMIOS_MCR register and eMIOS ch23 has been set as a global counter bus up-counter for 1000 eMIOS clocks. This assists in synchronising the time base for eMIOS channels 0 and 1.

eMIOS ch1 is configured in "output pulse width modulation" via the EMIOS_CCR register (mode bits). The leading edge has been set at counter = 500 and falling edge set at counter = 999, and a 1 kHz pulse train is generated at the GPIO (PG14). The pulse width information is written to the internal registers A and B (EMIOS_CADR[n] / EMIOS_CBDR[n]) of the channel.

The pulse train is input into PG15, eMISO ch0, which is configured as "input pulse width measurement." The pulse width information is passed to the EMIOS_CBDR[n] register where it is read and compared to what has been produced from eMIOS ch1. A simple software loop then determines whether this is valid or not.

4.3 Sample code

```
/****************************************************************************/
/* FILE NAME: main.c                            COPYRIGHT (c) Freescale 2010 */
/*                                                       All Rights Reserved */
/*   */
/* MPC5668 Example Projects Suite */
/* eMIOS Examples - Derived from AN2865 */
/* */
/* Revision History */
/* Rev: 1.0 Author:Steven McLaughlin  DATE: 20/07/2010  */
/* */
/* Example demostrates the eMIOS200 module by utilising the OPWM and IPWM    */
/* functions. A 1kHz pulse is generated and output on PG14 (eMIOS ch1) and   */
/* the pulse width (500us) is read on PG15 (eMIOS ch0). PE1 is connected     */
/* to an LED to notify when the correct pulse width has been measured.       */
/****************************************************************************/

/****************************** Includes **************************************/
#include "mpc5668.h"
/****************************** Global variables ******************************/
uint32_t pulse_width, count;
/****************************** Function Prototypes ***************************/
void initEMIOSch0(void);
void initEMIOSch1(void);
void initEMIOSch23(void);
void initEMIOS(void);

/*********************************** Main *************************************/
void main (void)
{
SIU.SYSCLK.B.SYSCLKSEL = 0x1;/* Select crystal as CLK Source - 40MHz */

/* --------------------ENABLE CLKOUT------------------------------------ */
SIU.PCR[153].R = 0x060C; /* CLKOUT - PK9 */
SIU.ECCR.B.ECDF = 0;     /* CLKOUT = Set to 0 for sys freq. */
SIU.ECCR.B.ECEN = 1; /* enable CLKOUT */
/* ------------------LED and Switch port init----------------------- */
SIU.PCR[65].R=0x0200;/* OBE on PE1 - LED connection */
SIU.GPDO[65].R = 0x01;

initEMIOS();
initEMIOSch0();
initEMIOSch1();
initEMIOSch23();

while (1)
{
while (!EMIOS.CH[0].CSR.B.FLAG);/* Wait for flag to be set */
pulse_width = EMIOS.CH[0].CBDR.R;/* read pulse width from register B */

if (pulse_width = EMIOS.CH[1].CADR.R)   /* Compare measured width at ch1 with original */
{
for (count=0; count<100000; count++);/* loops to toggle PE1, pulse width is correct */
SIU.GPDO[65].R = 0x01;
for (count=0; count<100000; count++);
SIU.GPDO[65].R = 0x00;
}

/* Clear flags and overruns */
EMIOS.CH[0].CSR.B.FLAG = 1;
EMIOS.CH[1].CSR.B.FLAG = 1;
EMIOS.CH[0].CSR.B.OVR = 1;
EMIOS.CH[1].CSR.B.OVR = 1;
}
}/* end of main() */

/**************************** Functions******************************************/

void initEMIOSch23(void)         /* EMIOS CH 23: Modulus Up Counter */
{
EMIOS.CH[23].CADR.R = 999;       /* Period will be 999+1 = 1000 clocks (1 msec) */
EMIOS.CH[23].CCR.B.MODE = 0x50;  /* Modulus Counter Buffered (MCB) */
EMIOS.CH[23].CCR.B.BSL = 0x3;    /* Use internal counter */
EMIOS.CH[23].CCR.B.UCPRE=0;    /* Set channel prescaler to divide by 1 */
EMIOS.CH[23].CCR.B.FREN = 1;    /* Freeze channel counting when in debug mode */
EMIOS.CH[23].CCR.B.UCPREN = 1;   /* Enable prescaler; uses default divide by 1 */
}

void initEMIOSch0(void) /* EMIOS CH 0: Input Pulse Width Measurement (IPWM) */
{
EMIOS.CH[0].CCR.B.BSL = 0x0; /* Use counter bus A (default) */
EMIOS.CH[0].CCR.B.EDPOL = 1;   /* Polarity-leading edge sets output/trailing clears */
EMIOS.CH[0].CCR.B.EDSEL = 1;   /* internal counter trigger by both edges */
EMIOS.CH[0].CCR.B.MODE = 0x04;   /* Input pulse width measurement */
SIU.PCR[111].R = 0x0500;          /* Initialize pad for eMIOS chan. 0 input */
}

void initEMIOSch1(void)     /* EMIOS CH 1: Output Pulse Width Modulation (OPWMB) */
{
  EMIOS.CH[1].CADR.R = 500;        /* Leading edge when channel counter bus=500 */
  EMIOS.CH[1].CBDR.R = 999;        /* Trailing edge when channel's counter bus=999 */
  EMIOS.CH[1].CCR.B.BSL = 0x0;   /* Use counter bus A (default) */
  EMIOS.CH[1].CCR.B.EDPOL = 1;   /* Polarity-leading edge sets output/trailing clears */
  EMIOS.CH[1].CCR.B.MODE = 0x60;   /* MPC551x, MPC563x: Mode is OPWM Buffered */
  SIU.PCR[110].R = 0x0600;          /* Initialize pad for eMIOS chan. 1 output */
}

void initEMIOS(void)
{
  EMIOS.MCR.B.GPRE= 39;     /* Divide 40 MHz sysclk by 39+1 = 40 for 1MHz eMIOS clk */
  EMIOS.MCR.B.GPREN = 1;/* Enable eMIOS clock */
  EMIOS.MCR.B.GTBE = 1;/* Enable global time base */
  EMIOS.MCR.B.FRZ = 1;/* Enable stopping channels when in debug mode */
}
```

5 Low power operation

MPC5668E/G supports two modes of operation — run and sleep. Sleep mode is essential to reduce the power consumption of the application by removing power from areas of the device, to eliminate static leakage. The CRP module remains powered to monitor wakeup events. The table below provides a summary of both sleep mode and run mode. Application note AN4150 has been written to specifically address the larger subject of low power operation on MPC5668.

Table 1. Run and sleep mode summary

Features	Run mode	Sleep mode
Standard cell logic (z6, z0, DMA, peripherals, etc.)	Powered, running	Powered down
Flash	Power from reset optionally disabled	Powered down

Table continues on the next page...

Table 1. Run and sleep mode summary (continued)

Features	Run mode	Sleep mode
SRAM	All SRAM powered	Optional amounts of SRAM powered: 0: All RAM powered 1: First 32 KB powered 2: First 64 KB powered 3: First 128 KB powered
Output pins	Active	Disabled
Input pins	Active	Active for wakeup
RTC, API	Optional	Optional

5.1 Description

Task

The demonstration code executes from RAM and waking up to RAM after sleep mode. Fast recovery has been enabled on wake up, which is provided by an edge transition on an external pin.

Hardware setup

The MPC5668EVB should be set up as follows:

1. As per MPC5668EVBUMD, make sure the external 40 MHz crystal is responsible for clocking the MCU.
2. Connect PB0 to a user LED.
3. Connect PA0 to a user switch.
4. Connect PB4 to a user switch.

Exercise

Run the project and notice that the LED connected to PB0 will blink. Currently the device is in RUN mode and the software is looping within "while(1)," toggling the port PB0. To proceed to sleep mode the user must activate port PA0. At this point the device will enter sleep mode and the LED will stop toggling. Asserting PB4 will wake the device from sleep mode to run mode and the LED connected to PB0 will blink again.

An ammeter can be placed in series with the VRC header (J46) on the EVB to observe the current consumption dropping as expected when going into sleep mode.

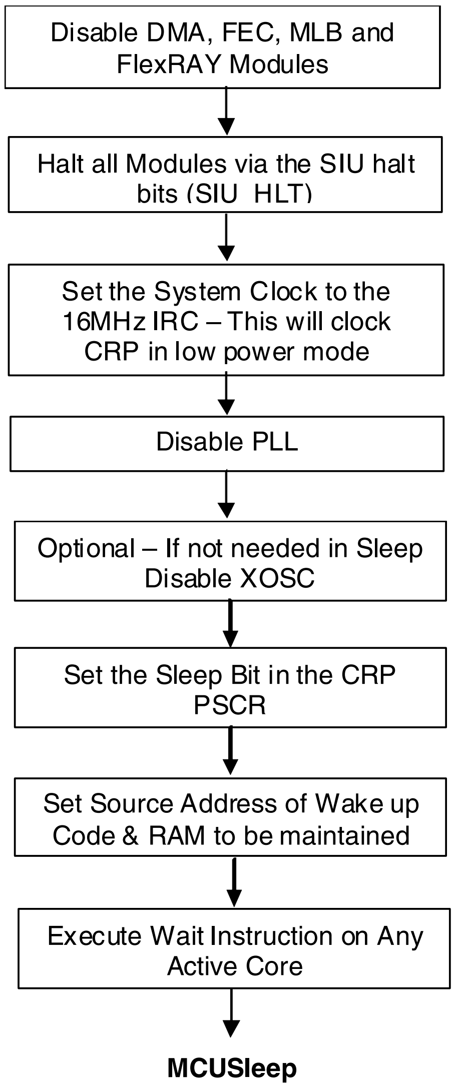
5.2 Program setup

The CRP module supports sleep mode, which is the MPC5668x low-power mode of operation. The primary function of the clock, reset, and power (CRP) module is to maintain all the control logic that requires power when other portions of the MCU are powered down in sleep mode. The CRP manages entry into, operation during, and exit from, sleep mode.

The CRP consists of the following:

- Input isolation block
- RTC/API
- Wakeup and power status block
- Clock and reset control block
- Low-power state machine
- Bus interface unit

The software executes a series of events to enter sleep mode as outlined in this flow diagram:


Disable DMA, FEC, MLB and FlexRAY Modules
Halt all Modules via the SIU halt bits (SIU_HLT)
Set the System Clock to the 16MHz IRC – This will clock CRP in low power mode
Disable PLL
Optional – If not needed in Sleep Disable XOSC
Set the Sleep Bit in the CRP PSCR
Set Source Address of Wake up Code & RAM to be maintained
Execute Wait Instruction on Any Active Core
MCUSleep

The software example uses the pin wakeup functionality on PB4, which is programmed by the CRP_PWKENL register. Rising, falling, or both edge transitions can be programmed to accomplish this (this example demonstrates rising edge).

The code example continuously loops between run and sleep modes. On waking up, the sleep and wakeup flags are cleared so that subsequent sleep requests are acknowledged.

5.3 Sample code

```
/*****************************************************************************/
/* FILE NAME: main.c                               COPYRIGHT (c) Freescale 2010 */
/*                                                          All Rights Reserved */
/*   */
/* MPC5668 Example Projects Suite */
/* Low Power Mode - Derived from AN4150 */
/* */
/* Revision History */
/* Rev: 1.0 Author:Steven McLaughlin  DATE: 23/07/2010  */
/* */
/* Example demonstrates low power sleep mode entry and exit from a pin.     */
/* PA0 is used for sleep entry and PB4 is used to exit sleep mode */
/*************************************************************************/

/****************************** Includes *************************************/
#include "mpc5668.h"
#include "typedefs.h"

/******************************* Global variables ******************************/
int PIN_NUMBER_SLEEP = 16;
int CLOCK_FREQ = 16000000;

/******************************* Function Prototypes ***************************/
extern void execute_wait(void);
void enter_sleep_mode (void);
void led_on(int time_on, int pin_number);
void led_off(int time_off, int pin_number);
void setup_pin(void);

/********************************** Main ***************************************/
void main(void)
{
setup_pin();
CRP.PWKENL.R = 0x00000002; /* Enable Positive Edge on PB4 as Wake */

while(1)
{
led_off(100,16);
led_on(100,16);

if (SIU.GPDI[0].R == 1) /* PA0 = 1 go to sleep mode */
{
enter_sleep_mode();
CRP.PSCR.B.SLEEPF = 0x1;/* Clear sleep flags to allow pads to operate */
CRP.PWKSRCF.R = 0x00000002;/* Clear the wake up flag */
setup_pin();/* Reconfig Pins - SIU Reset after Sleep */
}
}
}

/******************************* Enter Sleep Mode ******************************/
void enter_sleep_mode (void)
{

    CRP.PSCR.B.SLEEP = 1;                        /* Confirm Sleep Mode Required */
    CRP.Z6VEC.B.Z6VECB = 0x40000;                      /* Write Z6VEC register */
    CRP.Z6VEC.B.VLE = 0x0;                          /* set for non-VLE Wake code */

    SIU.HLT0.R = 0x037FFF3D;         /* Halt all modules except Z6 and Z0 */
    SIU.HLT1.R = 0x18000F3C;            /* Halt all modules except Z6 and Z0 */

    while((SIU.HLTACK0.R != 0x037FFF3D) && (SIU.HLTACK1.R != 0x18000F3C)); /* Wait for Halt
to Complete */

    CRP.PSCR.B.RAMSEL = 0x3;              /* All RAM maintained in SLEEP */
    CRP.RECPTR.B.FASTREC = 0x1;

    CRP.Z0VEC.B.Z0RST = 1;                             /* Put Z0 core into reset */
    SIU.SYSCLK.B.SYSCLKSEL = 0;             /* Switch system clock to 16MHz IRC */
    FMPLL.ESYNCR1.B.CLKCFG = 0;                        /* Disable PLL */
    CRP.PSCR.B.WKCLKSEL = 0x1;               /* 16MHz IRC Used for wake edge */

    SIU.PCR[16].R = 0x3;             /* Pull up enabled due to connected LED */

    execute_wait(); /* save data to stack and enter sleep mode */
}

/*************************************************************************/
/* FUNCTION: led_on                                                     */
/* PURPOSE : Drives the output pin to 0 (LED on) for a time defined         */
/*   by the parameter "time_on".      */
```

```
/*****************************************************************************/
void led_on(int time_on, int pin_number)
{
int x = 0;

SIU.GPDO[pin_number].R = 0x0;
time_on = (CLOCK_FREQ * time_on)/10000;

for(x=0; x<time_on; x++)
{
asm("nop");
}

time_on = 0;
}

/*****************************************************************************/
/* FUNCTION : led_off                                                        */
/* PURPOSE  : Drives the output pin to 1 (LED off) for a time defined        */
/*  by the parameter "time_off".                      */
/*****************************************************************************/
void led_off(int time_off, int pin_number)
{
int x = 0;

SIU.GPDO[pin_number].R = 0x1;
time_off = (CLOCK_FREQ * time_off)/10000;

for(x=0; x<time_off; x++)
{
asm("nop");
}
time_off = 0;
}

/*****************************************************************************/
/* FUNCTION     : setup_pin                                                  */
/* PURPOSE      : Configures the PCR settings for the pins used in this     */
/*                test.                                                      */
/*****************************************************************************/
void setup_pin()
{
 SIU.PCR[0].R = 0x0102;/* PA0 Input for Sleep Entry */
 SIU.PCR[16].R = 0x303;/* PB0 For signal LED's */
 SIU.PCR[20].R = 0x0100; /* PB4 Wake pin IBE = 1 */
}
```

6 Shifting DSPI data with eDMA

The eDMA provides an efficient method for moving data with minimal CPU intervention. The MPC5668x also contains an eDMA_Mux, which allows software selection of the channel source for the eDMA as well as specifying the triggering method to commence data transfer.

This example demonstrates the movement of data from SPI transmit and receive data buffers to and from RAM locations using the eDMA.

6.1 Description

Task

Only one board is required for this project. Two DSPI modules have been configured for SPI operation: DSPI_A has been configured as the master and DSPI_B as the slave. Channel 0 of the eDMA has been configured to move data from the RAM buffer to the DSPI_A transmit buffer register. With a physical connection between DSPI_A and DSPI_B, eDMA channel 1 has been configured to move data from the DSPI_B receive buffer register to the final RAM location. An LED is used to indicate when the data transfer has been completed.

Hardware setup

The MPC5668EVB should be set up as follows:

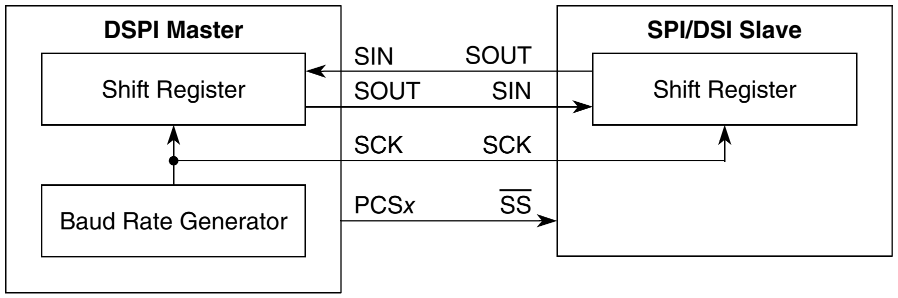


Figure 5. DSPI connections for SPI transfer

1. As per MPC5668EVBUMD, make sure the external 40 MHz crystal is responsible for clocking the MCU.
2. Connect PB0 to a user LED.
3. Connect PF0 (SCK_A) and PF4 (SCK_B).
4. Connect PF1 (SOUT_A) and PF6 (SIN_B).
5. Connect PF2 (SIN_A) and PF5 (SOUT_B).
6. Connect PF3 (PCS_A) and PF7 (PCS_B).
7. Connect a scope probe to PK9. Make sure it is properly grounded.
8. Optionally, scope probes can be connected between the above connections to check the signals.

Exercise

Running this project will not in itself demonstrate fully what has happened. However, with the connections in place, one can observe that the LED connected to PB0 will toggle. This indicates that a successful transfer of data has occurred between the two DSPI modules via the eDMA. By scoping the ports a 30 MHz SCK can be viewed on PF0 and data on PF1.

6.2 Program setup

NOTE

The Tx_buffer array was set up as a bitwise OR: Tx_buffer[i] = 0x00010000|i. The 0x00010000 setting was to allow the use of CS0 as indicated in DSPI.R.PUSHR.

The program writes an array of ten variables, Tx_buffer[i], to RAM, starting at address 0x40000028–0x4000004C.

Channel 0 on the eDMA sets up the transfer control descriptor (TCD). Within this, the source address is set to the Tx_buffer and the destination address is set at 0xFFF9_0034 which is the DSPI_A.PUSHR register. It is possible to see the Tx_buffer data being written to the PUSHR register. The PUSHR register provides a means to write to the TX FIFO (DSPI_TXFRn).

A physical connection is made between the ports of the DSPI_A and DSPI_B modules as shown in Figure 5 and the subsequent description. This allows the data to be moved between the DSPI shift registers.

Channel 1 TCD sets up its source address as DSPI_B.POPR register at 0xFFF9_4038. Watching this address within the memory window shows the continued writing of the data from DSPI_A. DSPI_B.POPR provides a means to read the RX FIFO located at 0xFFF9_407C/0xFFF9_407C 80/0xFFF9_407C 84/0xFFF9_407C 88. Subsequently, the eDMA TCD[1] destination address is configured at 0x40000050–0x40000074 in RAM. Therefore the original data from the Tx_buffer array is now stored in the Rx_buffer array.

Finally, the LED connected to PB0 is used to indicate if the data from Tx_buffer equates to Rx_buffer.

6.3 Sample code

```
/********************************************************************************/
/* FILE NAME: main.c                                COPYRIGHT (c) Freescale 2010 */
/*                                                          All Rights Reserved */
/*   */
/*MPC5668 Example Projects Suite*/
/*eDMA_SPI Example*/
/**/
/*Revision History*/
/*Rev: 1.0Author:Steven McLaughlinDATE: 08/10/2010*/
/* */
/* eDMA channel 0 used to take data from RAM Tx_buffer and place it in the  */
/* tranmit FIFO of DSPI_A as required. Channel 1 used to move received data  */
/* from DSPI_B receive FIFO to Rx_buffer in RAM                              */
/*                                                                            */
/* After transfers are complete, the receive bufer is tested to ensure*/
/* success and an LED indicates this. Number of errors are stored in Errcount*/
/********************************************************************************/

/****************************** Includes ***************************************/
#include "mpc5668.h"

/****************************** Global variables ********************************/
uint32_t Errcount, count; /* stores number of errors */
uint32_t RFOFCount, TFUFCount;
vuint32_t Tx_buffer[10], Rx_buffer[10];

/******************************* Function Prototypes ****************************/
void PLLinit(void);                    /* PLL Initialisation */
void eDMAinit(void);
void DSPIinit(void);

/*********************************** Main ***************************************/
void main()
{
int i;/* local variables */
Errcount = 0;/* assigning values to globals */
count = 0;/* assigning values to globals */
SIU.PCR[16].R=0x0200;/* PB0 to output */

    for(i=0;i<10;i++)
{
Tx_buffer[i] = 0x00010000|i; /* use CS0, data = 0-9 */
Rx_buffer[i] = 0;
}

PLLinit();/* Init PLL to 66Mhz from an 40MHz XOSC */
eDMAinit();/* setup the TCD registers */
DSPIinit();/* configure DSPI*/

/* exercise the SPI */
    DSPI_A.MCR.B.HALT = 0x0;/* exit HALT mode in master */
DSPI_A.MCR.B.MDIS = 0x0;/* enable module in master*/
for(i=0;i<0xFFF;i++);     /* delay to ensure master is fully active before slave */
DSPI_B.MCR.B.HALT = 0x0;/* exit HALT mode in slave */
DSPI_B.MCR.B.MDIS = 0x0;/* enable module in slave  */

DMAMUX.CHCONFIG[0].R = 0x92; /* setup DMA MUX */
    DMAMUX.CHCONFIG[1].R = 0x95; /* setup DMA MUX */

EDMA.SERQR.R = 0x00; /* enable hardware interrupt for channel 0 */
EDMA.SERQR.R = 0x01; /* enable hardware interrupt for channel 1 */
```

```
for(i=0;i<0xFFFF;i++);/* pause to allow transfers to take place */

for(i=0;i<10;i++)/* test to ensure success */
{
if(Rx_buffer[i] == Tx_buffer[i])
Errcount++;
}

if(Errcount==0)
{
while(1)/* pass :) */
{
for (count=0; count<100000; count++);
SIU.GPDO[16].R = 0x01;
for (count=0; count<100000; count++);
SIU.GPDO[16].R = 0x00;
}
}
else
{
while(1);/* fail :( */
}
} /* END of Main */

void PLLinit(void)
{
int ERFD = 3;
int EPREDIV = 9;
int EMFD = 112;

SIU.SYSCLK.B.SYSCLKSEL = 0x0;/* Select IRC as CLK Source */

/* Configure PLL CTRL Regs */
FMPLL.ESYNCR1.B.CLKCFG = 0x7;
FMPLL.ESYNCR2.B.ERFD = ERFD+2;
FMPLL.ESYNCR1.B.EPREDIV = EPREDIV;
FMPLL.ESYNCR1.B.EMFD = EMFD;

while(FMPLL.SYNSR.B.LOCK != 1);/* Wait for PLL to lock */

FMPLL.ESYNCR2.B.ERFD = ERFD;/* Change pll up to final freq */
SIU.SYSCLK.B.SYSCLKSEL = 0x2;/* Switch from IRC to PLL */

SIU.PCR[153].R = 0x060C; /* clkout - K9 */
SIU.ECCR.B.ECDF = 0x1;     /* CLKOUT = sys freq / 2. Set to 0 for sys freq. */
SIU.ECCR.B.ECEN = 1; /* enable CLKOUT */
} /* End of PLLinit */

void eDMAinit(void)
{
/* channel 0 used to move data from RAM to DSPI_A transfer buffer */
EDMA.TCD[0].SADDR = (vuint32_t)Tx_buffer; /* source address of data */
EDMA.TCD[0].SMOD = 0;/* source address modulo disabled */
EDMA.TCD[0].SSIZE = 0x2;/* 32-bit source size */
EDMA.TCD[0].DMOD = 0;/* Destination address modulo disabled */
EDMA.TCD[0].DSIZE = 0x2;/* 32-bit destination size */
EDMA.TCD[0].SOFF = 0x4;/* source address offset = 4 (32 bit) */
EDMA.TCD[0].NBYTES = 0x4;/* transfer 4 bytes per channel activation */
EDMA.TCD[0].SLAST = -40;/* Restore Source address by -40 -> 10x32-bits */
EDMA.TCD[0].DADDR = (vuint32_t)&DSPI_A.PUSHR.R;/* destination address (DSPI_A.PUSHR) */
EDMA.TCD[0].CITERE_LINK = 0;/* disabled chanel2chanel linking */
EDMA.TCD[0].CITER = 0xA;/* current major iteration */
EDMA.TCD[0].DOFF = 0;/* destination address offset = 0 */
EDMA.TCD[0].DLAST_SGA = 0;/* restore destination address by 0 */
EDMA.TCD[0].BITERE_LINK = 0;/* disabled chanel2chanel linking */
EDMA.TCD[0].BITER = 0xA;/* starting major iteration count */
EDMA.TCD[0].BWC = 0;/* no bandwidth control */
EDMA.TCD[0].MAJORLINKCH = 0;/* disabled chanel2chanel linking */
EDMA.TCD[0].DONE = 0;
EDMA.TCD[0].ACTIVE = 0;
EDMA.TCD[0].MAJORE_LINK = 0;
EDMA.TCD[0].E_SG = 0;
EDMA.TCD[0].D_REQ = 0;
EDMA.TCD[0].INT_HALF = 0;
EDMA.TCD[0].INT_MAJ = 0;
EDMA.TCD[0].START = 0;

/* channel 1 used to move data from DSPI_B receive buffer to RAM */
EDMA.TCD[1].SADDR = (vuint32_t)&DSPI_B.POPR.R;/* source address of data */
EDMA.TCD[1].SMOD = 0;/* source address modulo disabled */
EDMA.TCD[1].SSIZE = 0x2;/* 32-bit source size */
EDMA.TCD[1].DMOD = 0;/* Destination address modulo disabled*/
EDMA.TCD[1].DSIZE = 0x2;/* 32-bit destination size */
EDMA.TCD[1].SOFF = 0;/* source address offset = 0 */
EDMA.TCD[1].NBYTES = 0x4;/* transfer 4 bytes per channel activation */
EDMA.TCD[1].SLAST = 0;/* Restore Source address by 0 */
EDMA.TCD[1].DADDR = (vuint32_t)Rx_buffer;/* destination address (DSPI_A.PUSHR) */
EDMA.TCD[1].CITERE_LINK = 0;/* disabled chanel2chanel linking */
EDMA.TCD[1].CITER = 0xA;/* current major iteration*/
EDMA.TCD[1].DOFF = 0x4;/* destination address offset = 4 (32-bit) */
EDMA.TCD[1].DLAST_SGA = -40;/* restore destination address by -40  -> 10x32-bits */
EDMA.TCD[1].BITERE_LINK = 0;/* disabled chanel2chanel linking */
EDMA.TCD[1].BITER = 0xA;/* starting major iteration count */
EDMA.TCD[1].BWC = 0;/* no bandwidth control */
EDMA.TCD[1].MAJORLINKCH = 0;/* disabled chanel2chanel linking */
EDMA.TCD[1].DONE = 0;
EDMA.TCD[1].ACTIVE = 0;
EDMA.TCD[1].MAJORE_LINK = 0;
EDMA.TCD[1].E_SG = 0;
EDMA.TCD[1].D_REQ = 0;
EDMA.TCD[1].INT_HALF = 0;
EDMA.TCD[1].INT_MAJ = 0;
EDMA.TCD[1].START = 0;
}

void DSPIinit(void)
{
/* setup DSPI_A as master */
SIU.PCR[81].R = 0x60C;          /* Configure pad PF1 for primary func: TxDA, output, fastest
slew rate */
SIU.PCR[82].R = 0x500;      /* Configure pad PF2 for primary func: RxDA, input  */
SIU.PCR[84].R = 0x60C;      /* Configure pad PF4 for primary func: TxDA, output (chip select
1) , fastest slew rate */
SIU.PCR[83].R = 0x60C;      /* Configure pad PF3 for primary func: TxDA, output (chip select
0) , fastest slew rate */
SIU.PCR[80].R = 0x60C;      /* Configure pad PF0 for primary func: TxDA, output (clock) ,
fastest slew rate */
SIU.PCR[95].R = 0xA0C;/* Configure pad PF15 for secondary func: TxDA, output (chip select
5) , fastest slew rate */
SIU.PCR[96].R = 0xA0C;/* Configure pad PG0 for secondary func: TxDA, output (chip select
4) , fastest slew rate */
SIU.PCR[154].R = 0xE0C;/* Configure pad PK10 for secondary func: TxDA, output (chip select
3) , fastest slew rate */

/* setup DSPI_B as slave */
SIU.PCR[85].R = 0x60C;      /* Configure pad PF5 for primary func: TxDA, output, fastest
slew rate */
SIU.PCR[146].R = 0x900;      /* Configure pad PK2 for primary func: RxDA, input  */
SIU.PCR[87].R = 0x500;      /* Configure pad PF7 for primary func: TxDA, input (chip select
0) */
SIU.PCR[84].R = 0x50C;      /* Configure pad PK0 for primary func: TxDA, input  (clock) */
SIU.PCR[86].R = 0x500;      /* Configure pad PF5 SIN_B */

/* setup DSPI modules */
DSPI_A.MCR.B.MSTR = 0x1;  /* select master mode */
DSPI_A.MCR.B.DCONF = 0x00;  /* select SPI mode */
DSPI_A.DSICR.B.DSICTAS = 0x000;
DSPI_A.MCR.B.PCSIS0 = 1; /* set inactive CS0 state to high */
DSPI_A.CTAR[0].R = 0x78000000;/* frame size 16bits, speed: 16.67MHz */
```

```
DSPI_B.MCR.B.MSTR = 0x0;   /* select slave mode */
DSPI_B.MCR.B.DCONF = 0x00;   /* select SPI mode */
DSPI_B.DSICR.B.DSICTAS = 0x001;
DSPI_B.MCR.B.PCSIS0 = 1;   /* set inactive CS0 state to high (required for slave module) */
DSPI_B.CTAR[1].R = 0x78000000;/* frame size 16bits*/

/* setup eDMA "interrupts" */
DSPI_A.RSER.B.TFFF_RE = 1; /* enable transmit fifo fill request interrupt */
DSPI_A.RSER.B.TFFF_DIRS = 1;/* DMA interrupt will be generated */
DSPI_B.RSER.B.RFDF_RE = 1;/* enable receiver not empty interrupt */
DSPI_B.RSER.B.RFDF_DIRS = 1;/* DMA interrupt will be generated */

DSPI_A.RSER.B.TCF_RE = 1; /* enable transmit fifo fill request interrupt */
DSPI_A.RSER.B.EOQF_RE = 1; /* enable transmit fifo fill request interrupt */

DSPI_B.RSER.B.TCF_RE = 1; /* enable transmit fifo fill request interrupt */
DSPI_B.RSER.B.RFOF_RE = 1; /* enable transmit fifo fill request interrupt */
}
```

7 Using the ADC to configure the eMIOS

The ADC module contains advanced features for normal, injected, and triggered injected conversion, along with offset refresh control, and supports the interface to the Cross Triggering Unit (CTU). The ADC contains user-configurable sampling and conversion times with a clock prescaler unit, to generate the ADC clock from the clock provided to the ADC digital interface. The ADC contains analog watchdogs for comparing values of the converted data against user programmed thresholds, plus interrupt generation based on threshold violation by the converted data.

7.1 Description

Task

The example code uses the output of the potentiometer as the input to the ADC. The potentiometer is connected to the analogue input AN0. The ADC converts this signal and the conversion result is written to the duty cycle register of the according eMIOS register which is normalized to the maximum output of the ADC.

Hardware setup

The MPC5668EVB should be set up as follows:

1. As per MPC5668EVBUMD, make sure the external 40 MHz crystal is responsible for clocking the MCU.
2. Set jumper J86 to connect the potentiometer to AN0.
3. Connect LED1 to PG14.
4. Connect a scope probe to PG14 to view the PWM on a scope.

Exercise

The code should be loaded to the device. It will start the conversions and the PWM output automatically.

Twisting the potentiometer will vary the input to the ADC and hence the PWM output. The LED connected to PG14 will vary in brightness as the PWM duty is altered.

7.2 Program setup

The basic elements of this setup:

- ADC is enabled.
- Mode is set to scan mode.
- Conversion start is software-triggered.

- Channel 0 is enabled for ADC conversion.
- Pad 0 is set to first alternate function (analog input) through its SIU_PCR.

In addition:
- The eMIOS is initialized with 1 MHz clock cycle.
- The eMIOS clock and global time base are enabled.
- Freezing the channel in debug mode is enabled.
- Output Pulse Width Modulation (OPWMB) is realized via eMIOS channel 1.
- Falling edge is set to 1023, the maximum value that can come out of the ADC.
- The leading edge is variable and set in the main function according to the ADC output. The bigger the ADC output the higher the value for the leading edge, and the smaller the PWM's duty cycle.

The conversion result is read from the ADC_ PRECDATAREG[0] and will be between 0x80000 and 0x803FF. The 'adc_result' will be between 0 and 0x3FF (or 1023 in decimal numbers).

7.3 Sample code

```
/*************************************************************************/
/* FILE NAME: main.c                              COPYRIGHT (c) Freescale 2010 */
/*                                                       All Rights Reserved */
/*         */
/* MPC5668 Example Projects Suite      */
/* ADC/PWM Examples      */
/*         */
/* Revision History          */
/*      Rev: 1.0 Author:Martin VaupelDATE: 11/10/2010      */
/*      */
/* Example demonstrates the use of ADC and the eMIOS of MPC5668. The user    */
/* can influence the brightness of an LED using the potentimeter as an     */
/* input to the ADC. Its result changes the PWMs duty cycle affecting the    */
/* LEDs brightness.      */
/*     */
/*************************************************************************/

/**************************** Includes *************************************/
#include "mpc5668.h"
/**************************** Global variables *****************************/
uint32_t adc_result;
/**************************** Function Prototypes **************************/
void initADC(void);
void initEMIOS(void);
void initEMIOSch1(void);
void initEMIOSch23(void);


/********************************* Main *************************************/
void main (void)
{
SIU.SYSCLK.B.SYSCLKSEL = 0x1;/* Select crystal as CLK Source - 40MHz */

initADC();
initEMIOS();
initEMIOSch1();
initEMIOSch23();

    ADC.MCR.B.NSTART = 1; /* start conversion */

while (1)
{
adc_result = ADC.PRECDATAREG[0].R - 0x80000;/* read conversion */
EMIOS.CH[1].CADR.R = adc_result; /* will be 0 to 1023 */
    EMIOS.CH[1].CSR.B.FLAG = 1;/* clear flag */
    EMIOS.CH[1].CSR.B.OVR = 1;/* clear overrun */
}
}/* end of main() */

/**************************** Functions*****************************************/

void initADC(void)
{
    ADC.MCR.B.PWDN = 0;/* ADC enable */
    ADC.MCR.B.MODE = 1; /* scan mode */
    ADC.MCR.B.TRGEN = 0;/* start by SW */
    ADC.NCMR0.B.CH0 = 1;/* mask enable for channel0 */
    SIU.PCR[0].B.PA=1; /* alt func on PA0 = AN0 - analog input from pot */
}

void initEMIOS(void)
{
EMIOS.MCR.B.GPRE= 39;   /* Divide 40 MHz sysclk by 39+1 = 40 for 1MHz eMIOS clk */
EMIOS.MCR.B.GPREN = 1;/* Enable eMIOS clock */
EMIOS.MCR.B.GTBE = 1;/* Enable global time base */
EMIOS.MCR.B.FRZ = 1;/* Enable stopping channels when in debug mode */
}

void initEMIOSch1(void)     /* EMIOS CH 1: Output Pulse Width Modulation (OPWMB) */
{
EMIOS.CH[1].CBDR.R = 1023;    /* Trailing edge when channel's counter bus=999 */
EMIOS.CH[1].CCR.B.BSL = 0x0;/* Use counter bus A (default) */
EMIOS.CH[1].CCR.B.EDPOL = 1;/* Polarity-leading edge sets output/trailing clears */
EMIOS.CH[1].CCR.B.MODE = 0x60;/* Output Pulse Width Modulation Buffered (flag on B1 match) */
SIU.PCR[110].R = 0x0600;      /* Initialize pad for eMIOS chan. 1 output -> PG14 */
}

void initEMIOSch23(void)      /* EMIOS CH 23: Modulus Up Counter */
{
EMIOS.CH[23].CADR.R = 1023;   /* Period will be 999+1 = 1000 clocks (1 msec) */
EMIOS.CH[23].CCR.B.MODE = 0x50;/* Modulus Counter Buffered (MCB) */
EMIOS.CH[23].CCR.B.BSL = 0x3;/* Use internal counter */
EMIOS.CH[23].CCR.B.UCPRE=0;   /* Set channel prescaler to divide by 1 */
EMIOS.CH[23].CCR.B.FREN = 1; /* Freeze channel counting when in debug mode */
EMIOS.CH[23].CCR.B.UCPREN = 1; /* Enable prescaler; uses default divide by 1 */
}
```

8 Using the I^{2}C module

The inter-integrated circuit (I^{2}C™) bus is a two-wire bidirectional serial bus that provides a simple and efficient method of data exchange between devices. It minimizes the external connections to devices and does not require an external address decoder.

The interface is designed to operate as fast as 100 kbps with maximum bus loading and timing. The device is capable of operating at higher baud rates, up to a maximum of module clock/20, with reduced bus loading. The MPC5668x provides four functionally identical I^{2}C modules.

8.1 Description

Task

The example code demonstrates the use of the I^{2}C module by showing how data is sent from module A (master) to module B (slave). The user will vary the blinking frequency of an LED using a push button to increase an internal counter up to three. The counter variable is permanently sent from I^{2}C module A to I^{2}C module B and eventually used to vary the frequency of the blinking LED.

Hardware setup

The MPC5668EVB should be set up as follows:

1. As per MPC5668EVBUMD, make sure the external 40 MHz crystal is responsible for clocking the MCU.
2. Connect LED1 to PE1.
3. Connect SW1 to PE0.
4. Make sure jumpers J38 and J57 are not connected/off.
5. Connect PE14 to PG4 (clock line) and PE15 to PG5.[1]
6. Connect a scope probe to PE14 to view the I2C clock signal, and to PE15 to view the I^{2}C data signal.

NOTE

The quality of the clock signal can be improved by using external pullups on pins PE14 and PG4.

Exercise

The code should be loaded to the device. It will start I^{2}C communication and LED blinking immediately.

Twisting the potentiometer will vary the input to the ADC and hence the PWM output. The LED connected to PG14 will vary in brightness as the PWM duty is altered.

8.2 Program setup

The program starts with the initialization of the PLL and I^{2}C modules. The transmission frequency is set through module A’s IBFG register. Its value and parameters should be chosen to ensure communication at 10 kHz.

The slave address (module B) needs to be set. A master address is not necessary as communication happens in just one direction. The control registers of both modules will be initialized by setting their values to zero: both modules are now enabled. Pads PE14 (SCL_A), PE15 (SDA_A), PG4 (SCL_B), and PG5 (SDA_B) are configured to I^{2}C functions and necessary parameters.

Before starting I^{2}C communication the bus will first be checked to be ready (bus idles high).

1. When ready, a start signal (falling edge on the data line) is generated by setting module A to master and transmit mode. This will also set the busy flag (IBB) which will not be reset until the stop signal.
2. After successful generation of the start signal the slave address will be transmitted. The completion of a transmission is indicated by the IBIF flag.
3. A dummy read on the slave module will ensure that the module is ready to receive the next (data) byte.
4. Then data is transmitted and successful transmission checked.
5. A stop signal is generated by setting module A back to slave and receive mode. This will also reset the busy flag.
6. Finally the received data is read from module B’s data register (IBDR).

8.3 Sample code

```
/*****************************************************************************/
/* FILE NAME: main.c                            COPYRIGHT (c) Freescale 2010 */
/*                                                       All Rights Reserved */
/*        */
/* MPC5668 Example Projects Suite      */
/* I2C Examples              */
/*      */
/* Revision History     */
/*      Rev: 1.0 Author:Martin VaupelDATE: 12/10/2010     */
/*      */
/* Example demonstrates the use of the I2C module. The user will vary the    */
```

1. Make sure connections are of the same length.

```
/* blink frequency of an LED using a push button to increase an internal     */
/* counter up to 3. The counter variable is permanently send from I2C module */
/* A to I2C module B end eventually used to vary the frequency of the     */
/* blinking LED.     */
/*         */
/*****************************************************************************/

/****************************** Includes ************************************/
#include "mpc5668.h"
/****************************** Global variables ******************************/
uint32_t a, b, i, counter;
/****************************** Function Prototypes *************************/
void initI2C(void);
int transmitI2C (int);
void PLLinit(void);

/* I2C parameters */
#define slave_address  (0x6E)

/********************************* Main ************************************/
void main (void)
{
PLLinit();            /* Init PLL to 128Mhz w 40MHz XTAL */
initI2C();

counter= 1;
b = 100000;

/* LED and Switch port init */
SIU.PCR[64].R=0x0100;/* IBE on PE0 - switch connection */
SIU.PCR[65].R=0x0200;/* OBE on PE1 - LED connection */

/* wait a little bit*/
i = 10000;
while(i--);

while(1)
{
if (SIU.GPDI[64].R == 1)    /* used to increase internal counter via PE0 */
{
counter++;
if (counter > 3)
{
counter=1;
}
}

counter=transmitI2C(counter);/* send and receive counter via I2C */

for (a=0; a<b*counter; a++);/* loops to toggle PE1 */
SIU.GPDO[65].R = 1;
for (a=0; a<b*counter; a++);
SIU.GPDO[65].R = 0x00;
}
}/* end of main() */

//*************************** Functions*****************************************

void initI2C(void)
{
    /* Set transmission frequency 0x9C = 100kHz -based on 128Mhz Fsys */
    I2C_A.IBFD.R = 0x9C;

    /* Set module I2C addresses */
    I2C_B.IBAD.R = slave_address;

    /* Configure modules to idle (but active) state */
    I2C_A.IBCR.R = 0;
    I2C_B.IBCR.R = 0;
    /* Configure pads */
    SIU.PCR[78].R = 0x0733;/* PE14: Alt.F.: 1 -> SCL_A, OBE, IBE, ODE, HYS, WPE, WPS */
    SIU.PCR[79].R = 0x0733;     /* PE15: SDA_A */
    SIU.PCR[100].R = 0x0933;/* PG4: Alt.F.: 2 -> SCL_B, OBE, IBE, ODE, HYS, WPE, WPS */
    SIU.PCR[101].R = 0x0933;    /* PG5: SDA_B */
}

int transmitI2C (int data_send)
{
int data_rec;
I2C_A.IBSR.R |= 0x02;/* reset interrupt flag */

/* Make sure bus is idle -> IBB must be cleared*/
while (I2C_A.IBSR.B.IBB);

/* generate start condition by setting transmit and master */
I2C_A.IBCR.R = 0x30;

/* Wait for IBSR.IBB (bus busy) to be set */
while (!(I2C_A.IBSR.B.IBB));

/* Put target address into IBDR */
I2C_A.IBDR.R = slave_address;

/* Wait for address transfer to complete */
while (!(I2C_A.IBSR.B.IBIF));
I2C_A.IBSR.R |= 0x02;

/* Dummy read of IBDR to signal the module is ready for the next byte */
data_rec = I2C_B.IBDR.R;

/* Send data */
I2C_A.IBDR.R = data_send;

/* Wait for address transfer to complete */
while (!(I2C_A.IBSR.B.IBIF));
I2C_B.IBSR.R |= 0x02;

/* Restore module A to its idle (but active) state */
I2C_A.IBCR.R = 0;        /* stop transmission */

/* Wait for IBSR.IBB (bus busy) to be set */
while (!(I2C_A.IBSR.B.IBB));

data_rec = I2C_B.IBDR.R;

return data_rec;
}

void PLLinit(void)
{
int ERFD = 3;
int EPREDIV = 9;
int EMFD = 112;

/* Select IRC as CLK Source */
SIU.SYSCLK.B.SYSCLKSEL = 0x0;

/* Configure PLL CTRL Regs */
FMPLL.ESYNCR1.B.CLKCFG = 0x7;
FMPLL.ESYNCR2.B.ERFD = ERFD+2;
FMPLL.ESYNCR1.B.EPREDIV = EPREDIV;
FMPLL.ESYNCR1.B.EMFD = EMFD;

/* Wait for PLL to lock */
while(FMPLL.SYNSR.B.LOCK != 1);

/* Change pll up to final freq */
FMPLL.ESYNCR2.B.ERFD = ERFD;
```

```
/* Switch from IRC to PLL */
SIU.SYSCLK.B.SYSCLKSEL = 0x2;

} /* End of PLLinit */
```

9 Basic FlexCAN operation

The MPC5668x device features up to six FlexCAN modules which implement the CAN protocol according to Bosch Specification 2.0B and ISO standard 11898. The CAN protocol is used as a serial data bus, meeting the specifications of the field: real-time processing, reliable operation in electrically noisy environments, cost effectiveness, and required bandwidth.

9.1 Description

Task

The example code demonstrates FlexCAN usage by transmitting a string from one FlexCAN module to another. FlexCAN_A is initialized as the transmitter and FlexCAN_C is initialized as the receiver node. The bit rate has been set at 500 kHz using the 40 MHz external crystal on the EVB.

The MPC5668x EVBs contain two physical CAN transceivers which are aligned for use on CAN _A and CAN_B. This example uses two can nodes (CAN_A and CAN_C) through one transceiver (U5) as shown in Figure 6.

NOTE

It is the user’s responsibility to ensure bit timing is compliant with the CAN standard.

Hardware setup

The MPC5668EVB should be set up as follows:

1. As per MPC5668EVBUMD, make sure the external 40 MHz crystal is responsible for clocking the MCU.
2. Connect PD0 to PD4.
3. Connect PD1 to PD5. (Make sure the PD0/PD4 and the PD1/PD5 connections are of the same length.)
4. Connect PE1 to an LED.
5. Make sure jumper J30 is connected: pin 1 to pin 2, pin 3 to pin 4.
6. Make sure jumper J31 is connected: pin 1 to pin 2, pin 3 to pin 4.
7. Make sure jumper J21 is connected: pin 1 to pin 2, pin 3 to pin 4 and 5 to 6.
8. Optionally, connect a scope probe to PD0 and PD1, to view the data and ID being sent over the CAN bus.

NOTE

An external pullup resistor of 4.7 KΩ can be used on the CAN_Tx connection on PD0/PD4.

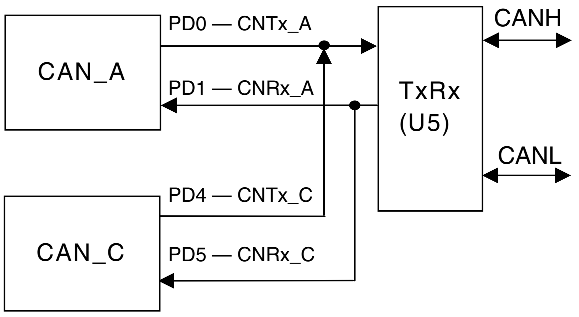


Figure 6. Sample CAN module hardware setup

Exercise

The code should be loaded to the device, and when it is run it will send the data immediately. An LED connected to PE1 will blink to show the functions for initializing and transmitting/receiving of data has been successful. Message buffer 4 in CAN_C can be viewed within the debugging environment and the sent data can be viewed.

9.2 Program setup

The program begins by initializing both CAN_A and CAN_C. CAN_C message buffer 4 has been set-up to receive the data and the buffer is set to ‘Rx Empty’ thereby awaiting the data. CAN_A message buffer 0 is passed a string of ‘hello’ from the array TxData within the TransmitMsg() function.

The message buffer is activated for transmission and the information is passed into the CAN_C message buffer. The function RecieveMsg() awaits the flag IFLAG_BUF04I and reads CODE, ID, LENGTH, and TIMESTAMP information from the message buffer, confirming correct transmission.

By scoping PD0 and PD1 it is possible to view the waveforms sent and received, and the data 48 65 6C 6C 6F can be viewed within CAN_A message buffer 0 followed by CAN_C message buffer 4.

9.3 Sample code

```
/****************************************************************************/
/* FILE NAME: main.c                            COPYRIGHT (c) Freescale 2010 */
/*                                                       All Rights Reserved */
/*       */
/* MPC5668 Example Projects Suite      */
/* FlexCAN Example - Re-use and derived from AN2865   */
/*              */
/*     */
/* Revision History     */
/*      Rev: 1.0 Author:Steven McLaughlinDATE: 19/10/2010     */
/*      */
/*This code example demostrates a CAN frame being transmitted from CAN_A */
/* and recieved at CAN_C on the device. Physical connections should be made */
/*between PD0 and PD4 (CAN Tx) and PD1 and PD5 (CAN Rx). A connection   */
/*between PE1 and a LED should be made. The user must also enable the   */
/*  CAN_A transciever on the EVB */
/*  Note the header file has been adjusted to recieved byte data on the CAN  */
/*message buffers  */
/****************************************************************************/

/****************************** Includes ************************************/
#include "mpc5668.h"
/***************************** Global variables ******************************/
uint32_t RxCODE;               /* Received message buffer code */
uint32_t RxID;                 /* Received message ID */
uint32_t RxLENGTH;             /* Recieved message number of data bytes */
uint8_t  RxDATA[8];            /* Received message data string*/
uint32_t RxTIMESTAMP;          /* Received message time */
/****************************** Function Prototypes **************************/

void initCAN_C (void) {
  uint8_t   i;

  CAN_C.MCR.R = 0x5000003F;       /* Put in Freeze Mode & enable all 64 message buffers */
  CAN_C.CTRL.R = 0x04DB0006;        /* Configure for 40MHz OSC, 500KHz bit time */
  for (i=0; i<64; i++) {
    CAN_C.BUF[i].CS.B.CODE = 0;   /* Inactivate all message buffers */
  }
  CAN_C.BUF[4].CS.B.IDE = 0;      /* MB 4 will look for a standard ID */
  CAN_C.BUF[4].ID.B.STD_ID = 555; /* MB 4 will look for ID = 555 */
  CAN_C.BUF[4].CS.B.CODE = 4;     /* MB 4 set to RX EMPTY */
  CAN_C.RXGMASK.R = 0x1FFFFFFF;   /* Global acceptance mask */
  SIU.PCR[52].R = 0x0620;          /* Configure pad as CNTXC, open drain */
  SIU.PCR[53].R = 0x0500;          /* Configure pad as CNRXC */
  CAN_C.MCR.R = 0x0000003F;        /* Negate FlexCAN C halt state for  64 MB */
}

void initCAN_A (void) {
  uint8_t   i;

  CAN_A.MCR.R = 0x5000003F;        /* Put in Freeze Mode & enable all 64 message buffers */
  CAN_A.CTRL.R = 0x04DB0006;         /* Configure for 40MHz OSC, 500KHz bit time */
  for (i=0; i<64; i++) {
    CAN_A.BUF[i].CS.B.CODE = 0;    /* Inactivate all message buffers */
  }
  CAN_A.BUF[0].CS.B.CODE = 8;      /* Message Buffer 0 set to TX INACTIVE */
  SIU.PCR[48].R = 0x0620;          /* Configure pad as CNTXA, open drain */
  SIU.PCR[49].R = 0x0500;          /* Configure pad as CNRXA */
  CAN_A.MCR.R = 0x0000003F;        /* Negate FlexCAN A halt state for 64 MB */
}

void TransmitMsg (void) {
  uint8_ti;
/* Assumption:  Message buffer CODE is INACTIVE */
  const uint8_t TxData[] = {"Hello"};  /* Transmit string*/
  CAN_A.BUF[0].CS.B.IDE = 0;           /* Use standard ID length */
  CAN_A.BUF[0].ID.B.STD_ID = 555;      /* Transmit ID is 555 */
  CAN_A.BUF[0].CS.B.RTR = 0;           /* Data frame, not remote Tx request frame */
  CAN_A.BUF[0].CS.B.LENGTH = sizeof(TxData) -1 ; /* # bytes to transmit w/o null */
  for (i=0; i<sizeof(TxData); i++) {
    CAN_A.BUF[0].DATA.B[i] = TxData[i];      /* Data to be transmitted */
  }
  CAN_A.BUF[0].CS.B.SRR = 1;           /* Tx frame (not req'd for standard frame)*/
  CAN_A.BUF[0].CS.B.CODE =0xC;         /* Activate msg. buf. to transmit data frame */
}

void RecieveMsg (void) {
  uint8_t j;
  uint32_t dummy;
  //while (!CAN_C.IFLAG1.B.BUF04I) {};  /* MPC551x: Wait for CAN C MB 4 flag */
  RxCODE   = CAN_C.BUF[4].CS.B.CODE;      /* Read CODE, ID, LENGTH, DATA, TIMESTAMP */
  RxID     = CAN_C.BUF[4].ID.B.STD_ID;
  RxLENGTH = CAN_C.BUF[4].CS.B.LENGTH;
  for (j=0; j<RxLENGTH; j++) {
    RxDATA[j] = CAN_C.BUF[4].DATA.B[j];
  }
  RxTIMESTAMP = CAN_C.BUF[4].CS.B.TIMESTAMP;
  dummy = CAN_C.TIMER.R;                /* Read TIMER to unlock message buffers */
  CAN_C.IFLAG1.R = 0x00000010;          /* MPC551x: Clear CAN C MB 4 flag */
}

void main(void) {
  volatile uint32_t count = 0;
  SIU.PCR[65].R=0x0200;/* OBE on PE1 - LED connection */
  initCAN_C();              /* Initialize FLEXCAN C & one of its buffers for receive*/
  initCAN_A();              /* Initialize FlexCAN A & one of its buffers for transmit*/
  TransmitMsg();            /* Transmit one message from a FlexCAN A buffer */
  RecieveMsg();             /* Wait for the message to be recieved at FlexCAN C */

  while (1) {
    for (count=0; count<100000; count++);/* loops to toggle PE1*/
SIU.GPDO[65].R = 0x01;
for (count=0; count<100000; count++);
SIU.GPDO[65].R = 0x00;

  }
}
```

10 Interrupt controller and PIT

The interrupt controller (INTC) schedules interrupt requests (IRQs) from software and internal peripherals to the e200z6 and e200z0 cores. The INTC provides interrupt prioritization and pre-emption, interrupt masking, interrupt priority elevation, and protocol support. The INTC supports 316 interrupt requests.

The Periodic Interrupt Timer (PIT) is an array of timers that can be used to initiate interrupts and trigger DMA channels.

10.1 Description

Task

This example demonstrates the functionality of the periodic interrupt timer (PIT) and the interrupt controller (INTC). While LED1 is toggled in the main loop a periodic interrupt will occur. In its ISR, LED2 is toggled 5 times while LED1 is frozen.

Hardware setup

The MPC5668EVB should be set up as follows:

1. As per MPC5668EVBUMD, make sure the external 40 MHz crystal is responsible for clocking the MCU.
2. Connect LED1 to PE0.
3. Connect LED2 to PE1.

Exercise

After the code is loaded to the device it will start toggling LED1 immediately. Approximately every five seconds the PIT will generate an interrupt, which is followed by LED2 flashing five times while LED1 is frozen.

10.2 Program setup

After selecting the crystal to run the system clock at 40 MHz and selecting the output ports for the LEDs the program starts with the intialization of the interrupts: ‘xcptn_xmpl()’. This function is in file ‘MPC56xx_55xx_Dual_PRC0_Interrupt_Init.c’ and will call three other functions.

The first one, e200zX_Interrupt_Setup(), initializes core IVPR and IVOR registers. This function is in the file ‘Vector_SW_VLE – Z3_Z4_Z6_Z7.s’. However, the IVOR4 handler which handles external interrupt requests (for example from peripherals) is programmed in file ‘IVOR4_SW_Handler_OldVLE.s’.

The second one, InitINTC(), initializes the interrupt controller by selecting software mode for processer 0 (Z6), choosing the default vector table offset, and setting the vector table base address. The actual vector table with the ISR entries is programmed in the file ‘IntclsrVectors.c’ with the entry ‘PIT1_ISR’ at position 149, as that is the vector number of PIT channel 1. The ISRs are programmed in the file ‘ISR_Routines_Dual.c’.

The third one, Init_INTC_Priority(), also in the file ‘ISR_Routines_Dual.c’, initializes the interrupt priorities.

In the next step INTC and core interrupts are enabled by ‘enableIrq()’ from file MPC56xx_55xx_Dual_PRC0_Interrupt_Init.c’.

Then the PIT is initialized: The value 0x03FFFFFF is loaded to the LDVAL register. The module and the timer are activated, interrupts are disabled, and the interrupt flag is cleared. The interrupt flag will be set when the counter comes down to zero, where it will start over at the initial value.

After the initializations the program enters an infinite loop where it does nothing but toggle PE0. Every five seconds the counter comes down to zero and an interrupt is requested by PIT1. The program will then execute the PIT1_ISR interrupt service routine and toggle PE1 five times before it returns to the main loop.

10.3 Sample code

```
/*****************************************************************************/
/* FILE NAME: main.c                            COPYRIGHT (c) Freescale 2010 */
/*                                                       All Rights Reserved */
/*        */
/* MPC5668 Example Projects Suite      */
/* PIT Examples               */
/*     */
/* Revision History     */
/*      Rev: 1.0 Author:Martin VaupelDATE: 19/10/2010     */
/*      */
/* */
/* Example demonstrates the functionality of the periodic interrup timer */
/* (PIT) and the interrupt controller (INTC). While LED1 is toggled in the */
/* main loop a periodic interrupt will occur. In its ISR LED2 is toggled */
/* 5 times while LED1 is frozen.  */
/*****************************************************************************/
#include "mpc5668.h"

  uint32_t count;

void PITinit(void);

/********************************** main ************************************/
void main (void)
{
SIU.SYSCLK.B.SYSCLKSEL = 0x1;/* Select crystal as CLK Source - 40MHz */

SIU.PCR[64].R=0x0200;/* OBE on PE0 - LED connection */
SIU.PCR[65].R=0x0200;/* OBE on PE1 - LED connection */

xcptn_xmpl();/* Initialize IVPR, IVOR, INTC and priorities */
enableIrq();/* Enable Interrupts in INTC and core */
PITinit();/* Initialize PIT */

while(1)
{
for (count=0; count<100000; count++);/* loops to toggle PE0 */
SIU.GPDO[64].R = 0x01;
for (count=0; count<100000; count++);
SIU.GPDO[64].R = 0x00;
    }
}/* end of main() */

void PITinit(void)/* Initializes PIT channel 1 */
{
PIT.MCR.B.MDIS = 0; /* enable PIT */
PIT.MCR.B.FRZ = 1; /* freeze timer in debug mode */

PIT.LDVAL1.R = 0x03FFFFFF;  /* load start value for timer1 */
/* will generate interrupt every 5 secs */

PIT.TCTRL1.B.TIE = 1; /* enable timer1 interrupts */
PIT.TCTRL1.B.TEN = 1; /* timer1 active */

PIT.TFLG1.B.TIF = 1;/* clear interrupt flag */
}
```

11 Enhanced Serial Communication Interface (eSCI), LIN mode

The eSCI allows asynchronous serial communications with peripheral devices and other CPUs. The eSCI has special features that allow the eSCI to operate as a LIN bus master, complying with the LIN 1.3, 2.0, 2.1, and SAE J2602 specification.

11.1 Description

Task

The project is based on Freescale application note AN2865, "MPC5500 and MPC5600 Simple Cookbook," example 15, "eSCI: LIN Transmit."The example demonstrates the use of the eSCI module in LIN mode. The string 'Hello' will be transmitted via LIN. The communication can be observed by connecting a scope to the transmit pin.

Hardware setup

The MPC5668EVB should be set up as follows:
1. As per MPC5668EVBUMD, make sure the external 40 MHz crystal is responsible for clocking the MCU.
2. Connect PD12 to PD13.
3. Connect a scope probe to PD12 to view the LIN transmit signal.

Exercise

After the code is loaded to the device it will start LIN transmission immediately. The voltage course of the transmitted LIN signal can be viewed on the scope.

11.2 Program setup

The program starts with setting up the PLL to clock the system with 64 MHz. The eSCI module A is initialized to operate in LIN mode and subsequent registers are set to appropriate values. The baud rate is set to approximately 10417 through the BRR register. Transmitter and receiver are enabled to ensure communication.

For the LIN transmission an array of data is defined including overhead information (the first three bytes in the LTR register) and actual data. After checking for the transmit ready flag (TXRDY) LIN data bytes are written to the LIN Transmission Register (LTR) and shifted by eight bits to align to the 16-bit register.

11.3 Sample code

```
/**************************************************************************/
/* FILE NAME: main.c                            COPYRIGHT (c) Freescale 2010 */
/*                                                       All Rights Reserved */
/*        */
/* MPC5668 Example Projects Suite      */
/* LIN Examples              */
/*     */
/* Revision History     */
/*      Rev: 1.0 Author:Martin VaupelDATE: 16/11/2010 */
/*      */
/* The project is based on AN2865 (MPC5500 & MPC5600 Simple Cookbook), */
/* example 15 eSCI: LIN Transmit. */
/* Example demonstrates the use of the eSCI module in LIN mode. The string */
/* 'Hello' will be transmitted via LIN. The communication can be observed by */
/* connecting a scope to the transmit pin.     */
/*        */
/*****************************************************************************/

/******************************* Includes *************************************/
#include "mpc5668.h"
/******************** Global variables and constants ************************/
const uint8_t FrameSpecAndData[] = {0x35,0x08,0xD0,'H','e','l','l','o',' ',' ',' '};

void initSysclk(void) {             /* Initialize PLL and sysclk to 64 MHz (set EMFD to 112
for 128MHz)*/

int ERFD = 3;
int EPREDIV = 9;
int EMFD = 48;

/* Select IRC as CLK Source */
SIU.SYSCLK.B.SYSCLKSEL = 0x0;

/* Configure PLL CTRL Regs */
FMPLL.ESYNCR1.B.CLKCFG = 0x7;
FMPLL.ESYNCR2.B.ERFD = ERFD+2;
FMPLL.ESYNCR1.B.EPREDIV = EPREDIV;
FMPLL.ESYNCR1.B.EMFD = EMFD;

/* Wait for PLL to lock */
while(FMPLL.SYNSR.B.LOCK != 1);

/* Change pll up to final freq */
FMPLL.ESYNCR2.B.ERFD = ERFD;

/* Switch from IRC to PLL */
SIU.SYSCLK.B.SYSCLKSEL = 0x2;

SIU.PCR[153].R = 0x060C; /* clkout - K9 */
SIU.ECCR.B.ECDF = 0x1;     /* CLKOUT = sys freq / 2. Set to 0 for sys freq.*/
SIU.ECCR.B.ECEN = 1; /* enable CLKOUT */
}

void initESCI_A (void) {
ESCI_A.CR2.R = 0x6240;/* Module is enabled, 13 bit break, stop on errors */
ESCI_A.CR1.R = 0x000C;  /* Tx and Rx enabled */
ESCI_A.BRR.R = 0x0180;/* 10417 baud, 8 bits, no parity */
ESCI_A.LCR1.R = 0x0100;  /* eSCI put in LIN mode */
SIU.PCR[60].R = 0x0400;     /* Configure pad for primary func: TxDA PD12*/
SIU.PCR[61].R = 0x0400;   /* Configure pad for primary func: RxDA PD13*/
}

void TransmitData (void) {
uint8_tj;                                      /* Dummy variable */
for (j=0; j< sizeof (FrameSpecAndData); j++) {     /* Loop for character string */
while (ESCI_A.IFSR2.B.TXRDY == 0) {};          /* Wait for LIN transmit ready = 1 */
ESCI_A.IFSR2.R = 0x00004000;                   /* Clear TXRDY flag */
ESCI_A.LTR.R = FrameSpecAndData[j] << 8;       /* Write byte to LIN Trans Reg. */
}
}

void main(void) {
initSysclk();        /* Set sysclk = 64MHz running from PLL */
initESCI_A();        /* Enable Tx for 10417 baud, 8 bits, no parity */
TransmitData();      /* Transmit string of characters on eSCI A */
while (1) {}         /* Wait forever */
}
```

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4241
Rev. 0, Feb 2011