# Sample program

#include "MPC5604B\_M07N.h"

void TransmitCharacter(uint8\_t ch);

/\* main.c - eMIOS OPWM example \*/

/\* Description: eMIOS example using Modulus Counter and OPWM modes \*/

/\* Rev 1.0 Sept 9 2004 S.Mihalik \*/

/\* Rev 1.1 April 13 2006 S.M.- corrected GPRE to be div by 12 instead of 13\*/

/\* Rev 1.2 June 26 1006 S.M. - updated comments & made i volatile uint32\_t \*/

/\* Rev 1.3 July 19 2007 SM- Changes for MPC551x, 50 MHz sysclk, Mod Ctr data value\*/

/\* Rev 1.4 Aug 10 2007 SM - Changed to use sysclk of 64 MHz \*/

/\* Rev 1.5 Jun 04 2008 SM - initSysclk changed for MPC5633M support \*/

/\* Rev 1.6 May 22 2009 SM - modified for MPC56xxB/S \*/

/\* Rev 1.7 Jun 24 2008 SM - simplified code \*/

/\* Rev 1.8 Mar 14 2010 SM - modified initModesAndClock, updated header file \*/

/\* Copyright Freescale Semiconductor, Inc. 2004–2010 All rights reserved. \*/

vuint32\_t i,j; /\* Dummy idle counter \*/

volatile uint8\_t Result[128]; /\* Read converstion result from ADC input ANS0 \*/

volatile uint32\_t dly,lly,chw,adcdata,curdata;

int16\_t posr,posl;

float posp,posi,poserr,posd,last\_poserr[10];

uint8\_t rx\_data[4],pt;

float kp=0.2,kd=0.4,ki=0.05,p;

//uint8\_t TransData[10]; /\* Transmit string & CR\*/

int16\_t ke=1,pospwm;

void printserialhex(uint16\_t innum) {

uint16\_t j1,in;

uint8\_t p1,p2;

in = innum;

j1 = (in & 0x0f);

if (j1 > 9) p1 = (uint8\_t)(j1 + 0x41 - 10);

else p1 = (uint8\_t)(j1 +0x30);

j1 = (in & 0xf0) >> 4;

if (j1 > 9) p2 = (uint8\_t)(j1 +0x41 - 10);

else p2 = (uint8\_t)(j1 +0x30);

TransmitCharacter(p2);

TransmitCharacter(p1);

}

void printserialsingned(uint16\_t innum) {

uint16\_t j1,k1,l1,m1,in;

uint8\_t p1,p2,p3,p4,p5;

if(innum < 0x8000) {

in = innum;

TransmitCharacter('+');

}

else {

in = (uint16\_t)(~innum);

//in = 0x7fff - in;

TransmitCharacter('-');

}

j1 = (in / 10);

p1 = (uint8\_t)(in - j1\*10 +0x30);

k1 = (j1 / 10);

p2 = (uint8\_t)(j1 - k1\*10 +0x30);

l1 = (k1 / 10);

p3 = (uint8\_t)(k1 - l1\*10 +0x30);

m1 = (l1 / 10);

p4 = (uint8\_t)(l1 - m1\*10 +0x30);

p5 = (uint8\_t)m1 +0x30;

TransmitCharacter(p5);

TransmitCharacter(p4);

TransmitCharacter(p3);

TransmitCharacter(p2);

TransmitCharacter(p1);

TransmitCharacter(0x09);

}

void printlistall(void) {

TransmitCharacter(0x0a);

TransmitCharacter(0x0d);

for(pt=0;pt<120;pt++){

//pt++;

//pt++;

printserialhex(Result[pt]);

//printserial(list[pt]);

}

TransmitCharacter(0x0a);

TransmitCharacter(0x0d);

}

void init\_LinFLEX\_0\_UART (void)

{

/\* enter INIT mode \*/

LINFLEX\_0.LINCR1.R = 0x0081; /\* SLEEP=0, INIT=1 \*/

/\* wait for the INIT mode \*/

while (0x1000 != (LINFLEX\_0.LINSR.R & 0xF000)) {}

/\* configure pads \*/

SIU.PCR[18].R = 0x0604; /\* Configure pad PB2 for AF1 func: LIN0TX \*/

SIU.PCR[19].R = 0x0100; /\* Configure pad PB3 for LIN0RX \*/

/\* configure for UART mode \*/

LINFLEX\_0.UARTCR.R = 0x0001; /\* set the UART bit first to be able to write the other bits \*/

LINFLEX\_0.UARTCR.R = 0x0033; /\* 8bit data, no parity, Tx and Rx enabled, UART mode \*/

/\* Transmit buffer size = 1 (TDFL = 0 \*/

/\* Receive buffer size = 1 (RDFL = 0) \*/

/\* configure baudrate 115200 \*/

/\* assuming 64 MHz peripheral set 1 clock \*/

LINFLEX\_0.LINFBRR.R = 12;

LINFLEX\_0.LINIBRR.R = 34;

/\* enter NORMAL mode \*/

LINFLEX\_0.LINCR1.R = 0x0080; /\* INIT=0 \*/

}

void TransmitCharacter(uint8\_t ch)

{

LINFLEX\_0.BDRL.B.DATA0 = ch; /\* write character to transmit buffer \*/

while (1 != LINFLEX\_0.UARTSR.B.DTF) {} /\* Wait for data transmission completed flag \*/

LINFLEX\_0.UARTSR.R = 0x0002; /\* clear the DTF flag and not the other flags \*/

}

void TransmitData (char TransData[])

{

uint8\_t j,k; /\* Dummy variable \*/

k = strlen (TransData);

for (j=0; j< k; j++)

{ /\* Loop for character string \*/

TransmitCharacter(TransData[j]); /\* Transmit a byte \*/

}

}

/\* This functions polls UART receive buffer. when it is full, it moves received data from the buffer to the memory \*/

uint8\_t ReadData (void)

{

uint8\_t ch;

/\* wait for DRF \*/

while (1 != LINFLEX\_0.UARTSR.B.DRF) {} /\* Wait for data reception completed flag \*/

/\* wait for RMB \*/

while (1 != LINFLEX\_0.UARTSR.B.RMB) {} /\* Wait for Release Message Buffer \*/

/\* get the data \*/

ch = (uint8\_t)LINFLEX\_0.BDRM.B.DATA4;

/\* clear the DRF and RMB flags by writing 1 to them \*/

LINFLEX\_0.UARTSR.R = 0x0204;

return ch;

}

void initModesAndClock(void) {

ME.MER.R = 0x0000001D; /\* Enable DRUN, RUN0, SAFE, RESET modes \*/

/\* Initialize PLL before turning it on: \*/

/\* Use 1 of the next 2 lines depending on crystal frequency: \*/

CGM.FMPLL\_CR.R = 0x02400100; /\* 8 MHz xtal: Set PLL0 to 64 MHz \*/

/\*CGM.FMPLL\_R = 0x12400100;\*/ /\* 40 MHz xtal: Set PLL0 to 64 MHz \*/

ME.RUN[0].R = 0x001F0074; /\* RUN0 cfg: 16MHzIRCON,OSC0ON,PLL0ON,syclk=PLL \*/

//ME.RUNPC[0].R = 0x00000010; /\* Peri. Cfg. 0 settings: only run in RUN0 mode \*/

/\* Use the next lines as needed for MPC56xxB/S: \*/

//ME.PCTL[48].R = 0x0000; /\* MPC56xxB LINFlex0: select ME.RUNPC[0] \*/

//ME.PCTL[68].R = 0x0000; /\* MPC56xxB/S SIUL: select ME.RUNPC[0] \*/

ME.RUNPC[1].R = 0x00000010; /\* Peri. Cfg. 1 settings: only run in RUN0 mode \*/

ME.PCTL[32].R = 0x01; /\* MPC56xxB ADC 0: select ME.RUNPC[1] \*/

ME.PCTL[57].R = 0x01; /\* MPC56xxB CTUL: select ME.RUNPC[1] \*/

ME.PCTL[48].R = 0x01; /\* MPC56xxB/P/S LINFlex 0: select ME.RUNPC[1] \*/

ME.PCTL[68].R = 0x01; /\* MPC56xxB/S SIUL: select ME.RUNPC[1] \*/

ME.PCTL[72].R = 0x01; /\* MPC56xxB/S EMIOS 0: select ME.RUNPC[1] \*/

/\* Mode Transition to enter RUN0 mode: \*/

ME.MCTL.R = 0x40005AF0; /\* Enter RUN0 Mode & Key \*/

ME.MCTL.R = 0x4000A50F; /\* Enter RUN0 Mode & Inverted Key \*/

while (ME.GS.B.S\_MTRANS) {} /\* Wait for mode transition to complete \*/

/\* Note: could wait here using timer and/or I\_TC IRQ \*/

while(ME.GS.B.S\_CURRENTMODE != 4) {}/\* Verify RUN0 is the current mode \*/

//while (ME.IS.B.I\_MTC != 1) {} /\* Wait for mode transition to complete \*/

//ME.IS.R = 0x00000001; /\* Clear Transition flag \*/

}

void initPeriClkGen(void) {

CGM.SC\_DC[0].R = 0x80; /\* MPC56xxB/S: Enable peri set 1 sysclk divided by 1 \*/

CGM.SC\_DC[2].R = 0x80; /\* MPC56xxB: Enable peri set 3 sysclk divided by 1\*/

}

void disableWatchdog(void) {

SWT.SR.R = 0x0000c520; /\* Write keys to clear soft lock bit \*/

SWT.SR.R = 0x0000d928;

SWT.CR.R = 0x8000010A; /\* Clear watchdog enable (WEN) \*/

}

void initPads (void) {

SIU.PCR[2].R = 0x0503; /\* MPC56xxB: Initialize PA[2] as eMIOS[2] input \*/

SIU.PCR[3].R = 0x0600; /\* MPC56xxB: Initialize PA[3] as eMIOS[3] output \*/

SIU.PCR[20].R = 0x2000; /\* MPC56xxB: Initialize PB[4] as ANP0 \*/

SIU.PCR[21].R = 0x2000; /\* MPC56xxB: Initialize PB[5] as ANP1 \*/

SIU.PCR[22].R = 0x2000; /\* MPC56xxB: Initialize PB[6] as ANP2 \*/

}

void initADC(void) {

//ADC.MCR.R = 0x20020000; /\* Initialize ADC scan mode\*/

ADC.MCR.R = 0x00000000; /\* Initialize ADC one shot mode\*/

ADC.NCMR[0].R = 0x00000007; /\* Select ANP1:2 inputs for normal conversion \*/

ADC.CTR[0].R = 0x00008606; /\* Conversion times for 32MHz ADClock \*/

}

void initCTU(void) {

CTU.EVTCFGR[2].R = 0x00008000; /\* Config event on eMIOS Ch 2 to trig ANP[0] \*/

}

// INITIALIZE THE EMIOS MODULE A (0)

void initEMIOS\_0(void) {

EMIOS\_0.MCR.B.GPRE= 63; /\* Divide 64 MHz sysclk by 63+1 = 64 for 1MHz eMIOS clk\*/

EMIOS\_0.MCR.B.GPREN = 1; /\* Enable eMIOS clock \*/

EMIOS\_0.MCR.B.GTBE = 1; /\* Enable global time base \*/

EMIOS\_0.MCR.B.FRZ = 1; /\* Enable stopping channels when in debug mode \*/

}

//SINGLE ACTION INPUT CAPTURE

void initEMIOS\_0ch3(void) {

EMIOS\_0.CH[3].CADR.R = 250; /\* Ch 3: Match "A" is 250 \*/

EMIOS\_0.CH[3].CBDR.R = 500; /\* Ch 3: Match "B" is 500 \*/

EMIOS\_0.CH[3].CCR.R= 0x000000E0; /\* Ch 3: Mode is OPWMB, time base = ch 23 \*/

EMIOS\_0.CH[2].CCR.R= 0x01020082; /\* Ch 2: Mode is SAIC, time base = ch 23 \*/

}

// FEEDS THE COUNTER\_BUS\_B

void initEMIOS\_0ch0(void) { /\* EMIOS 0 CH 0: Modulus Up Counter \*/

EMIOS\_0.CH[0].CADR.R = 19999; /\* Period will be 19999+1 = 20000 clocks (20 msec)\*/

EMIOS\_0.CH[0].CCR.B.MODE = 0x50; /\* Modulus Counter Buffered (MCB) \*/

EMIOS\_0.CH[0].CCR.B.BSL = 0x3; /\* Use internal counter \*/

EMIOS\_0.CH[0].CCR.B.UCPRE=0; /\* Set channel prescaler to divide by 1 \*/

EMIOS\_0.CH[0].CCR.B.UCPEN = 1; /\* Enable prescaler; uses default divide by 1\*/

EMIOS\_0.CH[0].CCR.B.FREN = 1; /\* Freeze channel counting when in debug mode\*/

}

// INITIALIZE THE COUNTER\_BUS\_A

void initEMIOS\_0ch23(void) { /\* EMIOS 0 CH 23: Modulus Up Counter \*/

EMIOS\_0.CH[23].CADR.R = 999; /\* Period will be 999+1 = 1000 clocks (1 msec)\*/

EMIOS\_0.CH[23].CCR.B.MODE = 0x50; /\* Modulus Counter Buffered (MCB) \*/

EMIOS\_0.CH[23].CCR.B.BSL = 0x3; /\* Use internal counter \*/

EMIOS\_0.CH[23].CCR.B.UCPRE=0; /\* Set channel prescaler to divide by 1 \*/

EMIOS\_0.CH[23].CCR.B.UCPEN = 1; /\* Enable prescaler; uses default divide by 1\*/

EMIOS\_0.CH[23].CCR.B.FREN = 1; /\* Freeze channel counting when in debug mode\*/

}

// USES COUNTER BUS B

void initEMIOS\_0ch4(void) { /\* EMIOS 0 CH 4: Output Pulse Width Modulation\*/

EMIOS\_0.CH[4].CADR.R = 0; /\* Leading edge when channel counter bus=0\*/

EMIOS\_0.CH[4].CBDR.R = 1500; /\* Trailing edge when channel counter bus=1400 Middle, 1650 Right Max, 1150 Left Max\*/

EMIOS\_0.CH[4].CCR.B.BSL = 0x01; /\* Use counter bus B \*/

EMIOS\_0.CH[4].CCR.B.EDPOL = 1; /\* Polarity-leading edge sets output \*/

EMIOS\_0.CH[4].CCR.B.MODE = 0x60; /\* Mode is OPWM Buffered \*/

SIU.PCR[28].R = 0x0600; /\* MPC56xxS: Assign EMIOS\_0 ch 6 to pad \*/

}

// BOTH USES COUNTER BUS A

void initEMIOS\_0ch6(void) { /\* EMIOS 0 CH 6: Output Pulse Width Modulation\*/

EMIOS\_0.CH[6].CADR.R = 500; /\* Leading edge when channel counter bus=0\*/

EMIOS\_0.CH[6].CBDR.R = 850; /\* Trailing edge when channel counter bus=500\*/

EMIOS\_0.CH[6].CCR.B.BSL = 0x0; /\* Use counter bus A (default) \*/

EMIOS\_0.CH[6].CCR.B.EDPOL = 1; /\* Polarity-leading edge sets output \*/

EMIOS\_0.CH[6].CCR.B.MODE = 0x60; /\* Mode is OPWM Buffered \*/

SIU.PCR[30].R = 0x0600; /\* MPC56xxS: Assign EMIOS\_0 ch 6 to pad \*/

}

void initEMIOS\_0ch7(void) { /\* EMIOS 0 CH 7: Output Pulse Width Modulation\*/

EMIOS\_0.CH[7].CADR.R = 0; /\* Leading edge when channel counter bus=0\*/

EMIOS\_0.CH[7].CBDR.R = 350; /\* Trailing edge when channel's counter bus=999\*/

EMIOS\_0.CH[7].CCR.B.BSL = 0x0; /\* Use counter bus A (default) \*/

EMIOS\_0.CH[7].CCR.B.EDPOL = 1; /\* Polarity-leading edge sets output\*/

EMIOS\_0.CH[7].CCR.B.MODE = 0x60; /\* Mode is OPWM Buffered \*/

SIU.PCR[31].R = 0x0600; /\* MPC56xxS: Assign EMIOS\_0 ch 7 to pad \*/

}

void Delay(void){

for(dly=0;dly<250;dly++);

}

void Delaylong(void){

for(dly=0;dly<20000;dly++);

}

void Delaylonglong(void){

for(lly=0;lly<1;lly++) Delaylong();

}

void Delaycamera(void){

for(lly=0;lly<10;lly++) Delaylong();

}

void Delaywait(void){

for(lly=0;lly<500;lly++) Delaylong();

}

void Delayled(void){

for(lly=0;lly<500;lly++) Delaylong();

}

void LED(void)

{

SIU.PCR[68].R = 0x0200; /\* Program the drive enable pin of LED1 (PE4) as output\*/

SIU.PCR[69].R = 0x0200; /\* Program the drive enable pin of LED2 (PE5) as output\*/

SIU.PCR[70].R = 0x0200; /\* Program the drive enable pin of LED3 (PE6) as output\*/

SIU.PCR[71].R = 0x0200; /\* Program the drive enable pin of LED4 (PE7) as output\*/

TransmitData("\*\*\*\*Led Test\*\*\*\*\n\r");

TransmitData("All Led ON\n\r");

Delayled();

SIU.PGPDO[2].R |= 0x0f000000; /\* Disable LEDs\*/

SIU.PGPDO[2].R &= 0x07000000; /\* Enable LED1\*/

TransmitData("Led 1 ON\n\r");

Delayled();

SIU.PGPDO[2].R |= 0x08000000; /\* Disable LED1\*/

SIU.PGPDO[2].R &= 0x0b000000; /\* Enable LED2\*/

TransmitData("Led 2 ON\n\r");

Delayled();

SIU.PGPDO[2].R |= 0x04000000; /\* Disable LED2\*/

SIU.PGPDO[2].R &= 0x0d000000; /\* Enable LED3\*/

TransmitData("Led 3 ON\n\r");

Delayled();

SIU.PGPDO[2].R |= 0x02000000; /\* Disable LED3\*/

SIU.PGPDO[2].R &= 0x0e000000; /\* Enable LED4\*/

TransmitData("Led 4 ON\n\r");

Delayled();

SIU.PGPDO[2].R |= 0x01000000; /\* Disable LED4\*/

}

void SWITCH(void)

{

SIU.PCR[64].R = 0x0100; /\* Program the drive enable pin of S1 (PE0) as input\*/

SIU.PCR[65].R = 0x0100; /\* Program the drive enable pin of S2 (PE1) as input\*/

SIU.PCR[66].R = 0x0100; /\* Program the drive enable pin of S3 (PE2) as input\*/

SIU.PCR[67].R = 0x0100; /\* Program the drive enable pin of S4 (PE3) as input\*/

TransmitData("\*\*\*\*Switch Test\*\*\*\*\n\r");

TransmitData("Press S1 Switch\n\r");

while((SIU.PGPDI[2].R & 0x80000000) == 0x80000000); /\*Wait until S1 switch is pressed\*/

TransmitData("Switch S1 Pressed \n\r");

TransmitData("Press S2 Switch\n\r");

while((SIU.PGPDI[2].R & 0x40000000) == 0x40000000); /\*Wait until S2 switch is pressed\*/

TransmitData("Switch S2 Pressed \n\r");

TransmitData("Press S3 Switch\n\r");

while((SIU.PGPDI[2].R & 0x20000000) == 0x20000000); /\*Wait until S3 switch is pressed\*/

TransmitData("Switch S3 Pressed \n\r");

TransmitData("Press S4 Switch\n\r");

while((SIU.PGPDI[2].R & 0x10000000) == 0x10000000); /\*Wait until S4 switch is pressed\*/

TransmitData("Switch S4 Pressed \n\r");

}

void SERVO(void)

{

TransmitData("\*\*\*\*Steering Servo Test\*\*\*\*\n\r");

EMIOS\_0.CH[4].CBDR.R = 1500; /\* 1500 Middle \*/

TransmitData("Middle\n\r");

Delaywait();

EMIOS\_0.CH[4].CBDR.R = 1750; /\* 1750 Right Max,\*/

TransmitData("Right\n\r");

Delaywait();

EMIOS\_0.CH[4].CBDR.R = 1250; /\* 1250 Left Max\*/

TransmitData("Left\n\r");

Delaywait();

EMIOS\_0.CH[4].CBDR.R = 1500; /\* 1500 Middle \*/

}

void MOTOR\_LEFT(void)

{

TransmitData("\*\*\*\*Left Drive Motor Test\*\*\*\*\n\r");

SIU.PCR[16].R = 0x0200; /\* Program the drive enable pin of Left Motor as output\*/

SIU.PGPDO[0].R = 0x00008000; /\* Enable Left the motors \*/

Delaywait();

SIU.PGPDO[0].R = 0x00000000; /\* Disable Left the motors \*/

}

void MOTOR\_RIGHT(void)

{

TransmitData("\*\*\*\*Right Drive Motor Test\*\*\*\*\n\r");

SIU.PCR[17].R = 0x0200; /\* Program the drive enable pin of Right Motor as output\*/

SIU.PGPDO[0].R = 0x00004000; /\* Enable Right the motors \*/

Delaywait();

SIU.PGPDO[0].R = 0x00000000; /\* Disable Right the motors \*/

}

void CAMERA(void)

{

TransmitData("\*\*\*\*Line Sensor Test\*\*\*\*\n\r");

SIU.PCR[27].R = 0x0200; /\* Program the Sensor read start pin as output\*/

SIU.PCR[29].R = 0x0200; /\* Program the Sensor Clock pin as output\*/

for(j=0;j<2;j++)

//for(;;)

{

SIU.PCR[27].R = 0x0200; /\* Program the Sensor read start pin as output\*/

SIU.PCR[29].R = 0x0200; /\* Program the Sensor Clock pin as output\*/

SIU.PGPDO[0].R &= ~0x00000014; /\* All port line low \*/

SIU.PGPDO[0].R |= 0x00000010; /\* Sensor read start High \*/

Delay();

SIU.PGPDO[0].R |= 0x00000004; /\* Sensor Clock High \*/

Delay();

SIU.PGPDO[0].R &= ~0x00000010; /\* Sensor read start Low \*/

Delay();

SIU.PGPDO[0].R &= ~0x00000004; /\* Sensor Clock Low \*/

Delay();

for (i=0;i<128;i++)

{

Delay();

SIU.PGPDO[0].R |= 0x00000004; /\* Sensor Clock High \*/

ADC.MCR.B.NSTART=1; /\* Trigger normal conversions for ADC0 \*/

while (ADC.MCR.B.NSTART == 1) {};

adcdata = ADC.CDR[0].B.CDATA;

Delay();

SIU.PGPDO[0].R &= ~0x00000004; /\* Sensor Clock Low \*/

Result[i] = (uint8\_t)(adcdata >> 2);

}

Delaycamera();

//printlistall();

}

printlistall();

}

void RIGHT\_MOTOR\_CURRENT(void)

{

TransmitData("\*\*\*\*Right Motor Current\*\*\*\*\n\r");

SIU.PGPDO[0].R = 0x00004000; /\* Enable Right the motors \*/

Delaywait();

for (i=0;i <10;i++)

{

ADC.MCR.B.NSTART=1; /\* Trigger normal conversions for ADC0 \*/

while (ADC.MSR.B.NSTART == 1) {};

curdata = ADC.CDR[2].B.CDATA;

printserialsingned(curdata);

}

SIU.PGPDO[0].R = 0x00000000; /\* Disable Right the motors \*/

}

void LEFT\_MOTOR\_CURRENT(void)

{

TransmitData("\*\*\*\*Left Motor Current\*\*\*\*\n\r");

SIU.PGPDO[0].R = 0x00008000; /\* Enable Right the motors \*/

Delaywait();

for (i=0;i <10;i++)

{

ADC.MCR.B.NSTART=1; /\* Trigger normal conversions for ADC0 \*/

while (ADC.MSR.B.NSTART == 1) {};

curdata = ADC.CDR[1].B.CDATA;

printserialsingned(curdata);

}

SIU.PGPDO[0].R = 0x00000000; /\* Disable Right the motors \*/

}

void main (void) {

volatile uint32\_t i = 0; /\* Dummy idle counter \*/

uint8\_t option;

initModesAndClock(); /\* Initialize mode entries and system clock \*/

initPeriClkGen(); /\* Initialize peripheral clock generation for DSPIs \*/

disableWatchdog(); /\* Disable watchdog \*/

initPads(); /\* Initialize pads used in example \*/

initADC(); /\* Init. ADC for normal conversions but don't start yet\*/

initCTU(); /\* Configure desired CTU event(s) \*/

initEMIOS\_0(); /\* Initialize eMIOS channels as counter, SAIC, OPWM \*/

initEMIOS\_0ch3(); /\* Initialize eMIOS 0 channel 3 as OPWM and channel 2 as SAIC\*/

initEMIOS\_0ch0(); /\* Initialize eMIOS 0 channel 0 as modulus counter\*/

initEMIOS\_0ch23(); /\* Initialize eMIOS 0 channel 23 as modulus counter\*/

initEMIOS\_0ch4(); \* Initialize eMIOS 0 channel 0 as OPWM, ch 4 as time base \*/

initEMIOS\_0ch6(); /\* Initialize eMIOS 0 channel 0 as OPWM, ch 6 as time base \*/

initEMIOS\_0ch7(); /\* Initialize eMIOS 0 channel 1 as OPWM, ch 7 as time base \*/

init\_LinFLEX\_0\_UART();

SIU.PCR[17].R = 0x0200; /\* Program the drive enable pin of Right Motor as output\*/

SIU.PCR[16].R = 0x0200; /\* Program the drive enable pin of Left Motor as output\*/

SIU.PGPDO[0].R = 0x00000000; /\* Disable the motors \*/

/\* Loop forever \*/

for (;;)

{

TransmitData("\n\r\*\*The Freescale Cup\*\*");

TransmitData("\n\r\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

TransmitData("\n\r1.Led\n\r");

TransmitData("2.Switch\n\r");

TransmitData("3.Servo\n\r");

TransmitData("4.Motor Left\n\r");

TransmitData("5.Motor Right\n\r");

TransmitData("6.Camera\n\r");

TransmitData("7.Left Motor Current\n\r");

TransmitData("8.Right Motor Current");

TransmitData("\n\r\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

option = ReadData();

# JUMPERS





