Sam Caldwell
Rohith Prakash

Lab 1 Report

1.0 OBJECTIVES

The objectives of this lab were to lay a foundation for the operating system that we will write in future labs. A major part of that was familiarize ourselves with the LM3S8962 board, µVision development system and the LM3S8962 Arm Cortex-M3 microcontroller. Additionally, we wrote drivers for important components/peripherals in our future system. The ADC will be used for test threads for our operating system and eventually for sensors for a robot. The UART will be used as an interpreter/shell for our OS, allowing real-time monitoring and control. We also implemented a periodic task mechanism, which is used to keep time for the OS and could be used for other OS tasks, like memory management/garbage collection.

2.0 SOFTWARE DESIGN

included as zip

3.0 MEASUREMENT DATA

Using the Keil debugger, we counted the timer2 ISR to have 72 instructions. Estimating 1 cycle per instruction and a clock period of 20 ns gives approximately 1440 ns or 1.44 µs. We then profiled

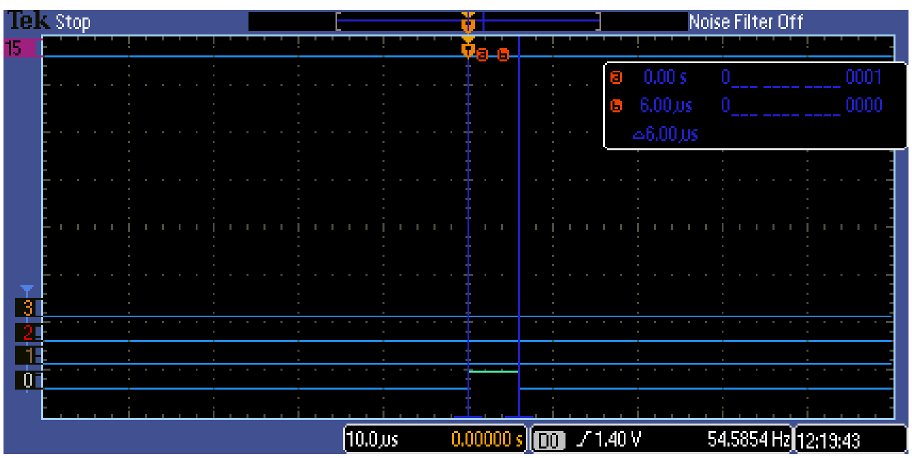

Figure 1. Profile of timer2 ISR

the ISR using a logic analyzer as shown in Figure 1, measuring the ISR to take 6.00 µs.

4.0 ANALYSIS AND DISCUSSION

1) The range of the ADC is 0-3.3V, the resolution is 10 bits, and the precision is 3.223 mV.

2) An ADC conversion can be explicitly triggered by the software, an analog comparator, periodically triggered by one of the general-purpose timers, or by a pulse-width modulator. We chose to use timer-triggered interrupts to minimize jitter.

3) We measured the time to execute the timer2 ISR by setting a GPIO port pin high at the start of the ISR and low at the end and then profiling it with a logic analyzer (Figure 1). The advantages of these types of measurements is that they are very minimally intrusive, but require equipment like a logic analyzer or oscilloscope.

4) The time required to execute one instance of the ISR is 6 microseconds. There are 72 instructions in this ISR plus a function call. This means that each instruction takes approximately 83ns to execute. This is approximately 4 times as long as 1 cycle on a 50MHz clock. This could be due to each instruction requiring multiple cycles due to cache misses in addition to the overhead required for the function call. This time could be reduced by changing the function into a C macro.

5) SysTick has a 24-bit timer, with a maximum range of 0 to $2^{24} = 16777216$. It has a resolution of one clock cycle, and a precision of the reset value the timer was initialized with.

ADC.c

```
#include "ADC.h"
#include "inc/hw_types.h"
#include "driverlib/adc.h"
#include "inc/lm3s8962.h"
#include "debug.h"

#ifndef TRUE
  #define TRUE 1
#endif
#ifndef FALSE
  #define FALSE 0
#endif

#define ADC_NVIC_PRIORITY 3
#define PRESCALE 9 // constant prescale value
#define TIMER_RATE 5000000

static void _ADC_SetTimer0APeriod(unsigned int fs);
static int _ADC_EnableNVICInterrupt(int channelNum);
static int _ADC_DisableNVICInterrupt(int channelNum);
static int _ADC_SetNIVCPriority(int channelNum, unsigned int priority);
static void _ADC_ADC0_Init(void);
static void _ADC_ADC1_Init(void);
static void _ADC_ADC2_Init(void);
static void _ADC_ADC3_Init(void);

long StartCritical (void);    // previous I bit, disable interrupts
void EndCritical(long sr);    // restore I bit to previous value
void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts

// mailbox and flag for each ADC channel
int ADCHasData[4] = {FALSE, FALSE, FALSE, FALSE};
unsigned short ADCMailBox[4];

void ADC_Init(unsigned int fs) {
  volatile unsigned long delay;
  DisableInterrupts();
  SYSCTL_RCGC0_R |= SYSCTL_RCGC0_ADC;       // activate ADC
  SYSCTL_RCGC0_R &= ~SYSCTL_RCGC0_ADCSPD_M; // clear ADC sample speed field
  SYSCTL_RCGC0_R += SYSCTL_RCGC0_ADCSPD500K;// configure for 500K ADC max
sample rate
  delay = SYSCTL_RCGC0_R;                   // allow time to finish
activating
  // map ADC0-3 handlers to port D pins 0-3 for profiling
  #if DEBUG == 1
    SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOB;
    delay = SYSCTL_RCGC2_R;
    GPIO_PORTB_DIR_R |= 0x0F;             // make PB0-3 out
    GPIO_PORTB_DEN_R |= 0x0F;             // enable digital I/O on PB0-3
    GPIO_PORTB_DATA_R &= ~0x0F;           // clear PB0-3
```

```
  #endif
  ADC_TimerInit(fs);
  EnableInterrupts();
}

void ADC_TimerInit(unsigned int fs) {
  volatile unsigned long delay;
  SYSCTL_RCGC1_R |= SYSCTL_RCGC1_TIMER0;    // activate timer0
  delay = SYSCTL_RCGC1_R;                   // allow time to finish
activating
  TIMER0_CTL_R &= ~TIMER_CTL_TAEN;          // disable timer0A during setup
  TIMER0_CTL_R |= TIMER_CTL_TAOTE;          // enable timer0A trigger to
ADC
  TIMER0_CFG_R = TIMER_CFG_16_BIT;          // configure for 16-bit timer
mode
  TIMER0_TAMR_R = TIMER_TAMR_TAMR_PERIOD;   // configure for periodic mode
  TIMER0_TAPR_R = PRESCALE;                 // prescale value for trigger
  _ADC_SetTimer0APeriod(fs);
  TIMER0_IMR_R &= ~TIMER_IMR_TATOIM;        // disable timeout (rollover)
interrupt
  TIMER0_CTL_R |= TIMER_CTL_TAEN;           // enable timer0A 16-b,
periodic, no interrupts
}

// rate is (clock period)*(prescale + 1)(period + 1)
static void _ADC_SetTimer0APeriod(unsigned int fs) {
  unsigned int period = TIMER_RATE / fs;
  TIMER0_TAILR_R = period;                  // start value for trigger
}


// should this take a channel number as the argument?
int ADC_Open(int channelNum) {
  long sr;
  sr = StartCritical();
  switch (channelNum) {
    case 0:
      _ADC_ADC0_Init();
      break;
    case 1:
      _ADC_ADC1_Init();
      break;
    case 2:
      _ADC_ADC2_Init();
      break;
    case 3:
      _ADC_ADC3_Init();
      break;
    default:
      EndCritical(sr);
      return 0;
  }
  EndCritical(sr);
  return 1;
```

```
}

unsigned short ADC_In(unsigned int channelNum) {
  unsigned short data;
  long sr;
  while(!ADCHasData[channelNum]) {
    ;
  }
  sr = StartCritical();
  data = ADCMailBox[channelNum];
  ADCHasData[channelNum] = FALSE;
  EndCritical(sr);
  return data;
}

int ADC_Collect(unsigned int channelNum, unsigned int fs, unsigned short
buffer[], unsigned int numberOfSamples) {
  int i;
  _ADC_SetTimer0APeriod(fs);
  for(i = 0; i < numberOfSamples; i++) {
    buffer[i] = ADC_In(channelNum);
  }
  return 1;
}

static int _ADC_EnableNVICInterrupt(int channelNum) {
  switch (channelNum) {
    case 0:
      NVIC_EN0_R |= NVIC_EN0_INT14;             // enable interrupt 14 in
NVIC (ADC SS 0)
      break;
    case 1:
      NVIC_EN0_R |= NVIC_EN0_INT15;             // enable interrupt 15 in
NVIC (ADC SS 1)
      break;
    case 2:
      NVIC_EN0_R |= NVIC_EN0_INT16;             // enable interrupt 16 in
NVIC (ADC SS 2)
      break;
    case 3:
      NVIC_EN0_R |= NVIC_EN0_INT17;             // enable interrupt 17 in
NVIC (ADC SS 3)
      break;
    default:
      return 0;
  }
  return 1;
}

static int _ADC_DisableNVICInterrupt(int channelNum) {
  // requires privelege mode
  switch (channelNum) {
    case 0:
```

```
      NVIC_DIS0_R |= NVIC_EN0_INT14;             // disable interrupt 14 in
NVIC (ADC SS 0)
      break;
    case 1:
      NVIC_DIS0_R |= NVIC_EN0_INT15;             // disable interrupt 15 in
NVIC (ADC SS 1)
      break;
    case 2:
      NVIC_DIS0_R |= NVIC_EN0_INT16;             // disable interrupt 16 in
NVIC (ADC SS 2)
      break;
    case 3:
      NVIC_DIS0_R |= NVIC_EN0_INT17;             // disable interrupt 17 in
NVIC (ADC SS 3)
      break;
    default:
      return 0;
  }
  return 1;
}

static int _ADC_SetNIVCPriority(int channelNum, unsigned int priority) {
  if (priority > 7) {
    return 0;
  }
  switch (channelNum) {
    case 0:
      NVIC_PRI3_R = (NVIC_PRI3_R&0xFF1FFFFF)|(priority << 21); // bits 21-
23
      break;
    case 1:
      NVIC_PRI3_R = (NVIC_PRI4_R&0x1FFFFFFF)|(priority << 29); // bits 29-
31
      break;
    case 2:
      NVIC_PRI4_R = (NVIC_PRI4_R&0xFFFFFF1F)|(priority << 5); // bits 5-7
      break;
    case 3:
      NVIC_PRI4_R = (NVIC_PRI4_R&0xFFFF1FFF)|(priority << 13); // bits 13-
15
      break;
    default:
      return 0;
  }
  return 1;
}

// default priority 0 (highest)
static void _ADC_ADC0_Init(void) {
  ADC_ACTSS_R &= ~ADC_ACTSS_ASEN0;           // disable sample sequencer 0
  ADC_EMUX_R &= ~ADC_EMUX_EM0_M;             // clear SS0 trigger select
field
  ADC_EMUX_R += ADC_EMUX_EM0_TIMER;          // configure for timer trigger
event
```

```
  ADC_SSMUX0_R &= ~ADC_SSMUX0_MUX0_M;      // clear SS0 1st sample input
select field
                                          // configure for ADC0 as first
sample input
  ADC_SSMUX0_R += (0 << ADC_SSMUX0_MUX3_S);
  ADC_SSCTL0_R = (/*0                      // settings for 1st sample:
                 & */~ADC_SSCTL0_TS3        // read pin specified by
ADC0_SSMUX0_R
                 | ADC_SSCTL0_IE3          // raw interrupt asserted here
                 | ADC_SSCTL0_END3         // sample is end of sequence
                 & ~ADC_SSCTL0_D3);        // differential mode not used

  ADC_IM_R |= ADC_IM_MASK0;               // enable SS0 interrupts
  ADC_ACTSS_R |= ADC_ACTSS_ASEN0;          // enable sample sequencer 0
  _ADC_SetNIVCPriority(0, ADC_NVIC_PRIORITY);
  _ADC_EnableNVICInterrupt(0);
}

// default priority 1
static void _ADC_ADC1_Init(void) {
  ADC_ACTSS_R &= ~ADC_ACTSS_ASEN1;         // disable sample sequencer 1
  ADC_EMUX_R &= ~ADC_EMUX_EM1_M;          // clear SS1 trigger select
field
  ADC_EMUX_R += ADC_EMUX_EM1_TIMER;       // configure for timer trigger
event
  ADC_SSMUX1_R &= ~ADC_SSMUX1_MUX0_M;      // clear SS1 1st sample input
select field
                                          // configure for ADC1 as first
sample input
  ADC_SSMUX1_R += (1 << ADC_SSMUX1_MUX0_S);
  ADC_SSCTL1_R = (/*0                      // settings for 1st sample:
                 &*/ ~ADC_SSCTL1_TS0        // read pin specified by
ADC1_SSMUX0_R
                 | ADC_SSCTL1_IE0          // raw interrupt asserted here
                 | ADC_SSCTL1_END0         // sample is end of sequence
                 & ~ADC_SSCTL1_D0);        // differential mode not used

  ADC_IM_R |= ADC_IM_MASK1;               // enable SS1 interrupts
  ADC_ACTSS_R |= ADC_ACTSS_ASEN1;          // enable sample sequencer 1
  _ADC_SetNIVCPriority(1, ADC_NVIC_PRIORITY);
  _ADC_EnableNVICInterrupt(1);
}

// default priority 2
static void _ADC_ADC2_Init(void) {
  ADC_ACTSS_R &= ~ADC_ACTSS_ASEN2;         // disable sample sequencer 2
  ADC_EMUX_R &= ~ADC_EMUX_EM2_M;          // clear SS2 trigger select
field
  ADC_EMUX_R += ADC_EMUX_EM2_TIMER;       // configure for timer trigger
event
  ADC_SSMUX2_R &= ~ADC_SSMUX2_MUX0_M;      // clear SS2 1st sample input
select field
                                          // configure for ADC2 as first
sample input
```

```
  ADC_SSMUX2_R += (2 << ADC_SSMUX2_MUX0_S);
  ADC_SSCTL2_R = (/*0                            // settings for 1st sample:
                 & */~ADC_SSCTL2_TS0            // read pin specified by
ADC2_SSMUX0_R
                 | ADC_SSCTL2_IE0             // raw interrupt asserted here
                 | ADC_SSCTL2_END0            // sample is end of sequence
                 & ~ADC_SSCTL2_D0);           // differential mode not used

  ADC_IM_R |= ADC_IM_MASK2;                  // enable SS2 interrupts
  ADC_ACTSS_R |= ADC_ACTSS_ASEN2;             // enable sample sequencer 2
  _ADC_SetNIVCPriority(2, ADC_NVIC_PRIORITY);
  _ADC_EnableNVICInterrupt(2);
}

// default priority 3 (lowest)
static void _ADC_ADC3_Init(void) {
  ADC_ACTSS_R &= ~ADC_ACTSS_ASEN3;            // disable sample sequencer 3
  ADC_EMUX_R &= ~ADC_EMUX_EM3_M;             // clear SS3 trigger select
field
  ADC_EMUX_R += ADC_EMUX_EM3_TIMER;          // configure for timer trigger
event
  ADC_SSMUX3_R &= ~ADC_SSMUX3_MUX0_M;        // clear SS3 1st sample input
select field
                                            // configure for ADC3 as first
sample input
  ADC_SSMUX3_R += (3 /*<< ADC_SSMUX3_MUX0_S*/);
  ADC_SSCTL3_R = (/*0                            // settings for 1st sample:
                 &*/ ~ADC_SSCTL3_TS0            // read pin specified by
ADC0_SSMUX3_R
                 | ADC_SSCTL3_IE0             // raw interrupt asserted here
                 | ADC_SSCTL3_END0            // sample is end of sequence
(default setting, hardwired)
                 & ~ADC_SSCTL3_D0);           // differential mode not used

  ADC_IM_R |= ADC_IM_MASK3;                  // enable SS3 interrupts
  ADC_ACTSS_R |= ADC_ACTSS_ASEN3;             // enable sample sequencer 3
  _ADC_SetNIVCPriority(3, ADC_NVIC_PRIORITY);
  _ADC_EnableNVICInterrupt(3);
}

void ADC0_Handler(void) {
  long sr = StartCritical();
  #if DEBUG == 1
    GPIO_PORTB_DATA_R |= 0x01;
  #endif
  ADC_ISC_R |= ADC_ISC_IN0;                // acknowledge ADC sequence 0
completion
  ADCMailBox[0] = ADC_SSFIFO0_R & ADC_SSFIFO0_DATA_M;
  ADCHasData[0] = TRUE;
  #if DEBUG == 1
    GPIO_PORTB_DATA_R &= ~0x01;
  #endif
  EndCritical(sr);
}
```

```
void ADC1_Handler(void) {
  long sr = StartCritical();
  #if DEBUG == 1
    GPIO_PORTB_DATA_R |= 0x02;
  #endif
  ADC_ISC_R |= ADC_ISC_IN1;              // acknowledge ADC sequence 1
completion
  ADCMailBox[1] = ADC_SSFIFO1_R & ADC_SSFIFO1_DATA_M;
  ADCHasData[1] = TRUE;
  #if DEBUG == 1
    GPIO_PORTB_DATA_R &= ~0x02;
  #endif
  EndCritical(sr);
}

void ADC2_Handler(void) {
  long sr = StartCritical();
  #if DEBUG == 1
    GPIO_PORTB_DATA_R |= 0x04;
  #endif
  ADC_ISC_R |= ADC_ISC_IN2;              // acknowledge ADC sequence 2
completion
  ADCMailBox[2] = ADC_SSFIFO2_R & ADC_SSFIFO2_DATA_M;
  ADCHasData[2] = TRUE;
  #if DEBUG == 1
    GPIO_PORTB_DATA_R &= ~0x04;
  #endif
  EndCritical(sr);
}

void ADC3_Handler(void) {
  long sr = StartCritical();
  #if DEBUG == 1
    GPIO_PORTB_DATA_R |= 0x08;
  #endif
  ADC_ISC_R |= ADC_ISC_IN3;              // acknowledge ADC sequence 3
completion
  ADCMailBox[3] = ADC_SSFIFO3_R & ADC_SSFIFO2_DATA_M;
  ADCHasData[3] = TRUE;
  #if DEBUG == 1
    GPIO_PORTB_DATA_R &= ~0x08;
  #endif
  EndCritical(sr);
}
```

ADC.h

```
// ADC.h

// to use the ADC:
// first call ADC_Init(fs) where fs the desired sampling rate in Hertz
// then call ADC_Open(channelNum) for each channel you wish to open
// note that all channels will operate at the same sampling rate.
// to get data, call ADC_In(channelNum) for one result or ADC_Collect() for
more samples.
// ADC_Collect also sets/changes the sampling rate.

void ADC_Init(unsigned int fs);

void ADC_TimerInit(unsigned int fs);

// should this take a channel number as the argument?
int ADC_Open(int channelNum);

unsigned short ADC_In(unsigned int channelNum);

int ADC_Collect(unsigned int channelNum, unsigned int fs,
       unsigned short buffer[], unsigned int numberOfSamples);
```

ADCTestMain.c

```
// main program for testing the ADC driver

#include "ADC.h"

GPIO_PORTF_DATA_R

unsigned short result = 0;
unsigned short buffer[3][64];
int main (void) {
  unsigned int fs = 10000;
  ADC_Init(fs);
  ADC_Open(0);
  ADC_Open(1);
  ADC_Open(2);
  ADC_Open(3);
  result = ADC_In(0);
  ADC_Collect(0, fs, buffer[0], 64);
  ADC_Collect(1, fs, buffer[1], 64);
  ADC_Collect(2, fs, buffer[2], 64);
  ADC_Collect(3, fs, buffer[3], 64);
  while(1) {
    ;
  }
}
```

Debug.h

```
#ifndef __DEBUG_H__
#define __DEBUG_H__

#define DEBUG 1

#endif //__DEBUG_H__
```

Fifo.c

```
// FIFO.c
// Runs on any LM3Sxxx
// Provide functions that initialize a FIFO, put data in, get data out,
// and return the current size.  The file includes a transmit FIFO
// using index implementation and a receive FIFO using pointer
// implementation.  Other index or pointer implementation FIFOs can be
// created using the macros supplied at the end of the file.
// Daniel Valvano
// June 16, 2011

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to the Arm Cortex M3",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2011
   Programs 3.7, 3.8., 3.9 and 3.10 in Section 3.7

 Copyright 2011 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS
SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,
INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

#include "FIFO.h"

// Two-index implementation of the transmit FIFO
// can hold 0 to TXFIFOSIZE elements
#define TXFIFOSIZE 16 // must be a power of 2
#define TXFIFOSUCCESS 1
#define TXFIFOFAIL    0

typedef char txDataType;
unsigned long volatile TxPutI;// put next
unsigned long volatile TxGetI;// get next
txDataType static TxFifo[TXFIFOSIZE];

// initialize index FIFO
void TxFifo_Init(void){ long sr;
  sr = StartCritical(); // make atomic
  TxPutI = TxGetI = 0;  // Empty
  EndCritical(sr);
```

```
}
// add element to end of index FIFO
// return TXFIFOSUCCESS if successful
int TxFifo_Put(txDataType data){
  if((TxPutI-TxGetI) & ~(TXFIFOSIZE-1)){
    return(TXFIFOFAIL); // Failed, fifo full
  }
  TxFifo[TxPutI&(TXFIFOSIZE-1)] = data; // put
  TxPutI++;  // Success, update
  return(TXFIFOSUCCESS);
}
// remove element from front of index FIFO
// return TXFIFOSUCCESS if successful
int TxFifo_Get(txDataType *datapt){
  if(TxPutI == TxGetI ){
    return(TXFIFOFAIL); // Empty if TxPutI=TxGetI
  }
  *datapt = TxFifo[TxGetI&(TXFIFOSIZE-1)];
  TxGetI++;  // Success, update
  return(TXFIFOSUCCESS);
}
// number of elements in index FIFO
// 0 to TXFIFOSIZE-1
unsigned short TxFifo_Size(void){
 return ((unsigned short)(TxPutI-TxGetI));
}

// Two-pointer implementation of the receive FIFO
// can hold 0 to RXFIFOSIZE-1 elements
#define RXFIFOSIZE 10 // can be any size
#define RXFIFOSUCCESS 1
#define RXFIFOFAIL    0

typedef char rxDataType;
rxDataType volatile *RxPutPt; // put next
rxDataType volatile *RxGetPt; // get next
rxDataType static RxFifo[RXFIFOSIZE];

// initialize pointer FIFO
void RxFifo_Init(void){ long sr;
  sr = StartCritical();      // make atomic
  RxPutPt = RxGetPt = &RxFifo[0]; // Empty
  EndCritical(sr);
}
// add element to end of pointer FIFO
// return RXFIFOSUCCESS if successful
int RxFifo_Put(rxDataType data){
  rxDataType volatile *nextPutPt;
  nextPutPt = RxPutPt+1;
```

```
  if(nextPutPt == &RxFifo[RXFIFOSIZE]){
    nextPutPt = &RxFifo[0];  // wrap
  }
  if(nextPutPt == RxGetPt){
    return(RXFIFOFAIL);      // Failed, fifo full
  }
  else{
    *(RxPutPt) = data;       // Put
    RxPutPt = nextPutPt;     // Success, update
    return(RXFIFOSUCCESS);
  }
}
// remove element from front of pointer FIFO
// return RXFIFOSUCCESS if successful
int RxFifo_Get(rxDataType *datapt){
  if(RxPutPt == RxGetPt ){
    return(RXFIFOFAIL);      // Empty if PutPt=GetPt
  }
  *datapt = *(RxGetPt++);
  if(RxGetPt == &RxFifo[RXFIFOSIZE]){
     RxGetPt = &RxFifo[0];   // wrap
  }
  return(RXFIFOSUCCESS);
}
// number of elements in pointer FIFO
// 0 to RXFIFOSIZE-1
unsigned short RxFifo_Size(void){
  if(RxPutPt < RxGetPt){
    return ((unsigned short)(RxPutPt-
RxGetPt+(RXFIFOSIZE*sizeof(rxDataType)))/sizeof(rxDataType));
  }
  return ((unsigned short)(RxPutPt-RxGetPt)/sizeof(rxDataType));
}
```

Fifo.h

```
// FIFO.h
// Runs on any LM3Sxxx
// Provide functions that initialize a FIFO, put data in, get data out,
// and return the current size.  The file includes a transmit FIFO
// using index implementation and a receive FIFO using pointer
// implementation.  Other index or pointer implementation FIFOs can be
// created using the macros supplied at the end of the file.
// Daniel Valvano
// June 16, 2011

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to the Arm Cortex M3",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2011
     Programs 3.7, 3.8., 3.9 and 3.10 in Section 3.7

 Copyright 2011 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS
SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,
INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

#ifndef __FIFO_H__
#define __FIFO_H__

long StartCritical (void);    // previous I bit, disable interrupts
void EndCritical(long sr);    // restore I bit to previous value

// Two-index implementation of the transmit FIFO
// can hold 0 to TXFIFOSIZE elements
#define TXFIFOSIZE 16 // must be a power of 2
#define TXFIFOSUCCESS 1
#define TXFIFOFAIL    0

typedef char txDataType;

// initialize index FIFO
void TxFifo_Init(void);
// add element to end of index FIFO
```

```
// return TXFIFOSUCCESS if successful
int TxFifo_Put(txDataType data);
// remove element from front of index FIFO
// return TXFIFOSUCCESS if successful
int TxFifo_Get(txDataType *datapt);
// number of elements in index FIFO
// 0 to TXFIFOSIZE-1
unsigned short TxFifo_Size(void);

// Two-pointer implementation of the receive FIFO
// can hold 0 to RXFIFOSIZE-1 elements
#define RXFIFOSIZE 10 // can be any size
#define RXFIFOSUCCESS 1
#define RXFIFOFAIL    0

typedef char rxDataType;

// initialize pointer FIFO
void RxFifo_Init(void);
// add element to end of pointer FIFO
// return RXFIFOSUCCESS if successful
int RxFifo_Put(rxDataType data);
// remove element from front of pointer FIFO
// return RXFIFOSUCCESS if successful
int RxFifo_Get(rxDataType *datapt);
// number of elements in pointer FIFO
// 0 to RXFIFOSIZE-1
unsigned short RxFifo_Size(void);

// macro to create an index FIFO
#define AddIndexFifo(NAME,SIZE,TYPE,SUCCESS,FAIL) \
unsigned long volatile NAME ## PutI;    \
unsigned long volatile NAME ## GetI;    \
TYPE static NAME ## Fifo [SIZE];        \
void NAME ## Fifo_Init(void){ long sr;  \
  sr = StartCritical();                 \
  NAME ## PutI = NAME ## GetI = 0;      \
  EndCritical(sr);                      \
}                                       \
int NAME ## Fifo_Put (TYPE data){       \
  if(( NAME ## PutI - NAME ## GetI ) & ~(SIZE-1)){  \
    return(FAIL);      \
  }                    \
  NAME ## Fifo[ NAME ## PutI &(SIZE-1)] = data; \
  NAME ## PutI ## ++;  \
  return(SUCCESS);     \
}                      \
int NAME ## Fifo_Get (TYPE *datapt){  \
  if( NAME ## PutI == NAME ## GetI ){ \
```

```
    return(FAIL);      \
  }                    \
  *datapt = NAME ## Fifo[ NAME ## GetI &(SIZE-1)];  \
  NAME ## GetI ## ++;  \
  return(SUCCESS);     \
}                      \
unsigned short NAME ## Fifo_Size (void){  \
 return ((unsigned short)( NAME ## PutI - NAME ## GetI ));  \
}
// e.g.,
// AddIndexFifo(Tx,32,unsigned char, 1,0)
// SIZE must be a power of two
// creates TxFifo_Init() TxFifo_Get() and TxFifo_Put()

// macro to create a pointer FIFO
#define AddPointerFifo(NAME,SIZE,TYPE,SUCCESS,FAIL) \
TYPE volatile *NAME ## PutPt;    \
TYPE volatile *NAME ## GetPt;    \
TYPE static NAME ## Fifo [SIZE];        \
void NAME ## Fifo_Init(void){ long sr;  \
  sr = StartCritical();                 \
  NAME ## PutPt = NAME ## GetPt = &NAME ## Fifo[0]; \
  EndCritical(sr);                      \
}                                       \
int NAME ## Fifo_Put (TYPE data){       \
  TYPE volatile *nextPutPt;             \
  nextPutPt = NAME ## PutPt + 1;        \
  if(nextPutPt == &NAME ## Fifo[SIZE]){ \
    nextPutPt = &NAME ## Fifo[0];       \
  }                                     \
  if(nextPutPt == NAME ## GetPt ){      \
    return(FAIL);                       \
  }                                     \
  else{                                 \
    *( NAME ## PutPt ) = data;          \
    NAME ## PutPt = nextPutPt;          \
    return(SUCCESS);                    \
  }                                     \
}                                       \
int NAME ## Fifo_Get (TYPE *datapt){    \
  if( NAME ## PutPt == NAME ## GetPt ){ \
    return(FAIL);                       \
  }                                     \
  *datapt = *( NAME ## GetPt ## ++);    \
  if( NAME ## GetPt == &NAME ## Fifo[SIZE]){ \
    NAME ## GetPt = &NAME ## Fifo[0];   \
  }                                     \
  return(SUCCESS);                      \
}                                       \
```

```
unsigned short NAME ## Fifo_Size (void){\
  if( NAME ## PutPt < NAME ## GetPt ){  \
    return ((unsigned short)( NAME ## PutPt - NAME ## GetPt +
(SIZE*sizeof(TYPE)))/sizeof(TYPE)); \
  }                                     \
  return ((unsigned short)( NAME ## PutPt - NAME ## GetPt )/sizeof(TYPE)); \
}
// e.g.,
// AddPointerFifo(Rx,32,unsigned char, 1,0)
// SIZE can be any size
// creates RxFifo_Init() RxFifo_Get() and RxFifo_Put()

#endif //  __FIFO_H__
```

OLEDTestMain.c

```
#include <stdio.h>
#include "hw_types.h"
#include "sysctl.h"
#include "rit128x96x4.h"
#include "OS.h"
#include "UART.h"
#include "shell.h"

void EnableInterrupts(void);

int main(void)
{
	SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_XTAL_8MHZ |
SYSCTL_OSC_MAIN);
	EnableInterrupts();
  OLED_Init(15);
  UART_Init();
	Timer2A_Init();
  SH_Init();
	//OS_Add_Periodic_Thread(&f, 2000, 5);

	/* Loop indefinitely */
  while(1) ;
}
```

OS.c

```
#include "hw_types.h"
#include "lm3s8962.h"
#include "stdlib.h"
#include "OS_Critical.h"
#include "OS.h"
#include "Debug.h"

#if DEBUG == 1
	#include "stdio.h"
	#include "rit128x96x4.h"
#endif

/* Task node for linked list */
typedef struct _OS_Task {
	void (*task)(void);			/* Periodic task to perform */
	unsigned long time,			/* _OS_Task_Time at which to perform */
					priority,		/* Task priority */
					period,			/* Frequency in units of
100ns */
					task_id;		/* Task id */
	struct _OS_Task *next;		/* Pointer to next task to perform*/
} _OS_Task;

/* Increment the OS system time with each call
 * param: none
 * return none
 */
static void _OS_Inc_Time(void);

static void _OS_Update_Root(_OS_Task * temp, _OS_Task * cur_task);

/* Dummy function for profiling the Timer2 ISR */
static void dummy(void);

/* Linked list of tasks */
static _OS_Task* _OS_Root = NULL;

/* Interrupt counter, used to determine when scheduled
   tasks are to be executed				*/
static int _OS_Task_Time = 0;

/* OS system time */
static unsigned int _OS_System_Time;

/* Initialize timer2 for system time */
void Timer2A_Init(void)
```

```
{
	int nop = 5;
  #if DEBUG == 1
    volatile unsigned long delay;
  #endif
	SYSCTL_RCGC1_R |= SYSCTL_RCGC1_TIMER2;		/* Activate timer2A */
	nop *= SYSCTL_RCGC1_TIMER2;								/* Wait 
for clock to activate */
	nop *= SYSCTL_RCGC1_TIMER2;								/* Wait 
for clock to activate */
	TIMER2_CTL_R &= ~0x00000001;							/* Disable 
timer2A during setup */
	TIMER2_CFG_R = 0x00000004;							/* Configure 
for 16-bit timer mode */
	TIMER2_TAMR_R = 0x00000002;							/* Configure 
for periodic mode */
	TIMER2_TAPR_R = 49;
	/* 1us timer2A */
	TIMER2_ICR_R = 0x00000001;							/* Clear 
timer2A timeout flag */
	TIMER2_TAILR_R = 100;
	/* Reload time of 100us */
	TIMER2_IMR_R |= TIMER_IMR_TATOIM;						/* Arm timeout 
interrupt */

  #if DEBUG == 1
    /* setup PB0 to profile the timer2 ISR */
    SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOB;
    delay = SYSCTL_RCGC2_R;
    GPIO_PORTB_DIR_R |= 0x01;
    GPIO_PORTB_DEN_R |= 0x01;
    GPIO_PORTB_DATA_R &= ~0x01;
  #endif

	/* Add system time task */
	OS_Add_Periodic_Thread(&_OS_Inc_Time, 1, 4);
//	OS_Add_Periodic_Thread(&dummy, 100, 1);
}

/* Adds a new task to the Timer2A interrupt thread scheduler
 * param: void (*task)(void), function pointer of task to be called
 * param: unsigned long period, period of task
 * param: unsigned long priority, priority of task
 * return: 0 if task successfully added to thread scheduler,
 						1 if maximum threads already queued.
 */
int OS_Add_Periodic_Thread(void (*task)(void), unsigned long period, unsigned long priority)
{
	/* Declare function as critical */
```

```
	OS_CRITICAL_FUNCTION;

	/* Number of scheduled tasks */
	static int _OS_Num_Tasks = 0;

	/* Allocate variables */
	_OS_Task *new_task;
	int new_priority;

	/* Bounds checking */
	if(_OS_Num_Tasks >= OS_MAX_TASKS)
		return -1;

	/* Allocate space for new task */
	new_task = (_OS_Task*)malloc(sizeof(_OS_Task) + 1);

	/* Set task parameters */
	new_task->task = task;
	new_task->period = period * 10;
	new_task->priority = priority;
	new_task->task_id = _OS_Num_Tasks;
	new_task->time = period * 10;
	new_task->next = NULL;					/* No next task (last
task in list) */

	/* Calculate outside critical section */
	new_priority = ((NVIC_PRI5_R&0x00FFFFFF)
						| (1 << (28 + priority)));

	/* Start critical section */
	OS_ENTER_CRITICAL();

	/* Insert task into position  */
	if(_OS_Root == NULL)
		_OS_Root = new_task;
	else
		new_task->next = _OS_Root, _OS_Root = new_task;
	_OS_Update_Root(_OS_Root, new_task);

	/* Update interrupt values */
	NVIC_PRI5_R = new_priority;

	/* Enable timer and interrupt (in case this is the first task) */
	TIMER2_CTL_R |= 0x00000001;
	NVIC_EN0_R |= NVIC_EN0_INT23;

	/* End critical section */
	OS_EXIT_CRITICAL();
	return _OS_Num_Tasks++;
```

```
}

/* Execute next periodic task if interrupt count and
			task time match. Update task list and next interrupt
			priority if task is executed.
 * param: none
 * return: none
 */
void Timer2A_Handler(void)
{
	/* Declare function as critical */
	OS_CRITICAL_FUNCTION;

	_OS_Task *cur_task = _OS_Root;

  #if DEBUG == 1
    GPIO_PORTB_DATA_R |= 0x01;
  #endif

	/* Acknowledge interrupt */
	TIMER2_ICR_R = TIMER_ICR_TATOCINT;

	/* Update task time */
	_OS_Task_Time++;

	if(_OS_Task_Time <= _OS_Root->time)
		return;

	/* Execute task */
	cur_task->task();

	/* Begin critical section */
	OS_ENTER_CRITICAL();

	/* Update task's time */
	cur_task->time += cur_task->period;

	/* Insert executed task into new position */
	_OS_Update_Root(_OS_Root, cur_task);

	/* Update interrupt priority */
	NVIC_PRI5_R = _OS_Root->priority;

  #if DEBUG == 1
    GPIO_PORTB_DATA_R &= ~0x01;
  #endif

	/* End critical section */
	OS_EXIT_CRITICAL();
```

```
}

static void _OS_Update_Root(_OS_Task *temp, _OS_Task * cur_task)
{
	/* Find new position */
	while(temp->next && temp->next->time < cur_task->time)
		temp = temp->next;

	/* Create links and update OS root if necessary */
	if(temp != cur_task)
	{
		_OS_Root = _OS_Root->next;
		cur_task->next = temp->next;
		temp->next = cur_task;
	}
}

/* Increment OS system time (in milliseconds)
 * param: none
 * return: none
 */
static void _OS_Inc_Time()
{
	#if DEBUG == 1
	char time[20];
	#endif

	_OS_System_Time++;

	#if DEBUG == 1
//	if(_OS_System_Time % 1000 == 0)
//	{
//		sprintf(time, "Time: %d", _OS_System_Time);
//		OLED_Out(BOTTOM, time);
//	}
	#endif
}

/* Return OS system time (in milliseconds)
 * param: none
 * return: OS time in milliseconds
 */
unsigned int OS_MsTime(void)
{
	return _OS_System_Time;
}

/* Clears the OS system time
 * param: none
```

```
 * return: none
 */
void OS_ClearMsTime(void)
{
	_OS_System_Time = 0;
}

/* Dummy function for profiling the Timer2 ISR */
static void dummy(void) {
}
```

OS.h

```
#ifndef __OS_H__
#define __OS_H__

#define OS_MAX_TASKS 10

void Timer2A_Init(void);
int OS_Add_Periodic_Thread(void(*task)(void), unsigned long period, unsigned long priority);
void Timer2A_Handler(void);
unsigned int OS_MsTime(void);
void OS_ClearMsTime(void);
#endif
```

OS_Critical.c

```
#include "OS_Critical.h"

	//Keil uVision assembly code
	__asm
	OS_CPU_SR OS_CPU_SR_Save()
	{
		MRS R0, PRIMASK
		CPSID I
		BX LR
	}

	__asm
	void OS_CPU_SR_Restore(OS_CPU_SR cpu_sr)
	{
		MSR PRIMASK, R0
		BX LR
	}
```

OS_Critical.h

```
#ifndef __OS_CRITICAL_H__
#define __OS_CRITICAL_H__

typedef unsigned int OS_CPU_SR;

#define OS_CRITICAL_FUNCTION OS_CPU_SR cpu_sr
#define OS_ENTER_CRITICAL() { cpu_sr = OS_CPU_SR_Save(); }
#define OS_EXIT_CRITICAL() { OS_CPU_SR_Restore(cpu_sr); }

void OS_CPU_SR_Restore(OS_CPU_SR cps_sr);
OS_CPU_SR OS_CPU_SR_Save(void);

#endif
```

Rit128x96x4.c

```
/* Addition by Sam Caldwell and Rohith Prakash
 * Added OLED functions split the on-board OLED
 * into two logical screens, TOP and BOTTOM.
 */

/*********************************************/
/*                                    */
/*        BEGIN OLED FUNCTIONS          */
/*                                    */
/*********************************************/

#include "driverlib/debug.h"
#include "rit128x96x4.h"
#include "string.h"
#include "OS_Critical.h"

/* 8-bit OLED grayscale value */
static unsigned char _OLED_color = 0;

/* Initializes the OLED devices to be used as a
                    split screen interface
 * param: unsigned char color, grayscale text color value
 * return: none
 */
void OLED_Init(unsigned char color)
{
        RIT128x96x4Init(1000000);
        OLED_Set_Color(color);
}

/* Set the color of the OLED output
 * param: unsigned char color, 8-bit grayscale color
 * return: none
 */
void OLED_Set_Color(unsigned char color)
{
        _OLED_color = color;
}

/* Get the color of the OLED output
 * param: none
 * return: unsigned char _OLED_color, 8-bit grayscale color
 */
unsigned char OLED_Get_Color(void)
{
        return _OLED_color;
}
```

```
/* Allow the use of two (logically) separate screens.
 * This function outputs a given string onto the specified
 * part of the OLED screen
 * param: int device, screen to output onto (TOP or BOTTOM)
 * param: const char * string, character array to be displayed
 * return: none
 */
void OLED_Out(int device, const char * string)
{
		/* Marks function as critical */
		OS_CRITICAL_FUNCTION;

		/* Current line */
		static int line[NUM_DEVICES] = {0};

		/* Screen character contents */
		static char char_buff[NUM_DEVICES][OLED_LINES][OLED_COLUMNS];

		/* Blank line */
		char clear[OLED_COLUMNS + 2] = {0};
		int offset = 0;

		ASSERT(line < 5 && device < 2 && device >= 0);

		/* Enter critical section */
		OS_ENTER_CRITICAL();

		/* Initialize blank line */
		memset(clear, ' ', OLED_COLUMNS + 1);

		/* Write string, wrapping as needed */
		do
		{
			/* Row buffer */
			char cur_row[OLED_COLUMNS + 1] = {0};
			int num = 0, length;

			/* Copy next row, update offset */
			length = OLED_MIN( /* length up until end of string/column */
OLED_MIN(strlen(&string[offset]), OLED_COLUMNS),
									/* location of
'\n' */ _OLED_Find(&string[offset], '\n') + 1 );
			memcpy(cur_row, &string[offset], length);
			offset += length;

			/* Roll screen back when max lines reached */
			line[device] += num;
			if(line[device] >= OLED_LINES)
```

```
            {
                _OLED_Rollback(char_buff[device], device);
                line[device] = OLED_LINES - 1;
            }

            /* Write most recent row */
            strcpy(char_buff[device][line[device]], cur_row);
            _OLED_Message(device, line[device]++, cur_row, OLED_Get_Color());

        } while(string[offset]);

        /* End critical section */
        OS_EXIT_CRITICAL();
}

/* Draws a given string on the screne based on OLED device, line number, and color
 * param: int device, OLED screen to draw to (TOP or BOTTOM)
 * param: unsigned int line, line number to draw string on
 * param: const char * string, pointer to string to be drawn
 * param: unsigned char color, grayscale color value
 * return: none
 */
static void _OLED_Message(int device, unsigned int line, const char * string, unsigned char color)
{
        ASSERT(line < 5 && device < 2 && device >= 0);
        RIT128x96x4StringDraw(string, 0, device * 48 + (line * 8) + OLED_BUFFER, color);
}

/* Searches through string for a character
 * param: const char * string, string to search through
 * param: char c, character to find
 * return: int i, index of c in string
 */
static int _OLED_Find(const char * string, char c)
{
    int i;
    for(i = 0; string[i] != 0 && string[i] != c; i++);
    return i;
}

/*********************************************/
/*                                          */
/*          END OLED FUNCTIONS              */
/*                                          */
/*********************************************/

//*****************************************************************************
//
// rit128x96x4.c - Driver for the RIT 128x96x4 graphical OLED display.
```

```
//*****************************************************************************
//
//! \addtogroup display_api
//! @{
//
//*****************************************************************************

#include "inc/hw_ssi.h"
#include "inc/hw_memmap.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/ssi.h"
#include "driverlib/sysctl.h"

//*****************************************************************************
//
// Macros that define the peripheral, port, and pin used for the OLEDDC
// panel control signal.
//
//*****************************************************************************
#define SYSCTL_PERIPH_GPIO_OLEDDC   SYSCTL_PERIPH_GPIOA
#define GPIO_OLEDDC_BASE            GPIO_PORTA_BASE
#define GPIO_OLEDDC_PIN             GPIO_PIN_6
#define GPIO_OLEDEN_PIN             GPIO_PIN_7

//*****************************************************************************
//
```

```
// Flags to indicate the state of the SSI interface to the display.
//
//*****************************************************************************
static volatile unsigned long g_ulSSIFlags;
#define FLAG_SSI_ENABLED        0
#define FLAG_DC_HIGH            1

//*****************************************************************************
//
// Buffer for storing sequences of command and data for the display.
//
//*****************************************************************************
static unsigned char g_pucBuffer[8];

//*****************************************************************************
//
// Define the SSD1329 128x96x4 Remap Setting(s).  This will be used in
// several places in the code to switch between vertical and horizontal
// address incrementing.  Note that the controller support 128 rows while
// the RIT display only uses 96.
//
// The Remap Command (0xA0) takes one 8-bit parameter.  The parameter is
// defined as follows.
//
// Bit 7: Reserved
// Bit 6: Disable(0)/Enable(1) COM Split Odd Even
//        When enabled, the COM signals are split Odd on one side, even on
//        the other.  Otherwise, they are split 0-63 on one side, 64-127 on
//        the other.
// Bit 5: Reserved
// Bit 4: Disable(0)/Enable(1) COM Remap
//        When Enabled, ROW 0-127 map to COM 127-0 (that is, reverse row order)
// Bit 3: Reserved
// Bit 2: Horizontal(0)/Vertical(1) Address Increment
//        When set, data RAM address will increment along the column rather
//        than along the row.
// Bit 1: Disable(0)/Enable(1) Nibble Remap
//        When enabled, the upper and lower nibbles in the DATA bus for access
//        to the data RAM are swapped.
// Bit 0: Disable(0)/Enable(1) Column Address Remap
//        When enabled, DATA RAM columns 0-63 are remapped to Segment Columns
//        127-0.
//
//*****************************************************************************
#define RIT_INIT_REMAP      0x52 // app note says 0x51
#define RIT_INIT_OFFSET     0x00
static const unsigned char g_pucRIT128x96x4VerticalInc[]   = { 0xA0, 0x56 };
static const unsigned char g_pucRIT128x96x4HorizontalInc[] = { 0xA0, 0x52 };
```

```
//*****************************************************************************
//
// A 5x7 font (in a 6x8 cell, where the sixth column is omitted from this
// table) for displaying text on the OLED display.  The data is organized as
// bytes from the left column to the right column, with each byte containing
// the top row in the LSB and the bottom row in the MSB.
//
// Note:  This is the same font data that is used in the EK-LM3S811
// osram96x16x1 driver.  The single bit-per-pixel is expaned in the StringDraw
// function to the appropriate four bit-per-pixel gray scale format.
//
//*****************************************************************************
static const unsigned char g_pucFont[96][5] =
{
    { 0x00, 0x00, 0x00, 0x00, 0x00 }, // " "
    { 0x00, 0x00, 0x4f, 0x00, 0x00 }, // !
    { 0x00, 0x07, 0x00, 0x07, 0x00 }, // "
    { 0x14, 0x7f, 0x14, 0x7f, 0x14 }, // #
    { 0x24, 0x2a, 0x7f, 0x2a, 0x12 }, // $
    { 0x23, 0x13, 0x08, 0x64, 0x62 }, // %
    { 0x36, 0x49, 0x55, 0x22, 0x50 }, // &
    { 0x00, 0x05, 0x03, 0x00, 0x00 }, // '
    { 0x00, 0x1c, 0x22, 0x41, 0x00 }, // (
    { 0x00, 0x41, 0x22, 0x1c, 0x00 }, // )
    { 0x14, 0x08, 0x3e, 0x08, 0x14 }, // *
    { 0x08, 0x08, 0x3e, 0x08, 0x08 }, // +
    { 0x00, 0x50, 0x30, 0x00, 0x00 }, // ,
    { 0x08, 0x08, 0x08, 0x08, 0x08 }, // -
    { 0x00, 0x60, 0x60, 0x00, 0x00 }, // .
    { 0x20, 0x10, 0x08, 0x04, 0x02 }, // /
    { 0x3e, 0x51, 0x49, 0x45, 0x3e }, // 0
    { 0x00, 0x42, 0x7f, 0x40, 0x00 }, // 1
    { 0x42, 0x61, 0x51, 0x49, 0x46 }, // 2
    { 0x21, 0x41, 0x45, 0x4b, 0x31 }, // 3
    { 0x18, 0x14, 0x12, 0x7f, 0x10 }, // 4
    { 0x27, 0x45, 0x45, 0x45, 0x39 }, // 5
    { 0x3c, 0x4a, 0x49, 0x49, 0x30 }, // 6
    { 0x01, 0x71, 0x09, 0x05, 0x03 }, // 7
    { 0x36, 0x49, 0x49, 0x49, 0x36 }, // 8
    { 0x06, 0x49, 0x49, 0x29, 0x1e }, // 9
    { 0x00, 0x36, 0x36, 0x00, 0x00 }, // :
    { 0x00, 0x56, 0x36, 0x00, 0x00 }, // ;
    { 0x08, 0x14, 0x22, 0x41, 0x00 }, // <
    { 0x14, 0x14, 0x14, 0x14, 0x14 }, // =
    { 0x00, 0x41, 0x22, 0x14, 0x08 }, // >
    { 0x02, 0x01, 0x51, 0x09, 0x06 }, // ?
    { 0x32, 0x49, 0x79, 0x41, 0x3e }, // @
    { 0x7e, 0x11, 0x11, 0x11, 0x7e }, // A
    { 0x7f, 0x49, 0x49, 0x49, 0x36 }, // B
```

```
{ 0x3e, 0x41, 0x41, 0x41, 0x22 }, // C
{ 0x7f, 0x41, 0x41, 0x22, 0x1c }, // D
{ 0x7f, 0x49, 0x49, 0x49, 0x41 }, // E
{ 0x7f, 0x09, 0x09, 0x09, 0x01 }, // F
{ 0x3e, 0x41, 0x49, 0x49, 0x7a }, // G
{ 0x7f, 0x08, 0x08, 0x08, 0x7f }, // H
{ 0x00, 0x41, 0x7f, 0x41, 0x00 }, // I
{ 0x20, 0x40, 0x41, 0x3f, 0x01 }, // J
{ 0x7f, 0x08, 0x14, 0x22, 0x41 }, // K
{ 0x7f, 0x40, 0x40, 0x40, 0x40 }, // L
{ 0x7f, 0x02, 0x0c, 0x02, 0x7f }, // M
{ 0x7f, 0x04, 0x08, 0x10, 0x7f }, // N
{ 0x3e, 0x41, 0x41, 0x41, 0x3e }, // O
{ 0x7f, 0x09, 0x09, 0x09, 0x06 }, // P
{ 0x3e, 0x41, 0x51, 0x21, 0x5e }, // Q
{ 0x7f, 0x09, 0x19, 0x29, 0x46 }, // R
{ 0x46, 0x49, 0x49, 0x49, 0x31 }, // S
{ 0x01, 0x01, 0x7f, 0x01, 0x01 }, // T
{ 0x3f, 0x40, 0x40, 0x40, 0x3f }, // U
{ 0x1f, 0x20, 0x40, 0x20, 0x1f }, // V
{ 0x3f, 0x40, 0x38, 0x40, 0x3f }, // W
{ 0x63, 0x14, 0x08, 0x14, 0x63 }, // X
{ 0x07, 0x08, 0x70, 0x08, 0x07 }, // Y
{ 0x61, 0x51, 0x49, 0x45, 0x43 }, // Z
{ 0x00, 0x7f, 0x41, 0x41, 0x00 }, // [
{ 0x02, 0x04, 0x08, 0x10, 0x20 }, // "\"
{ 0x00, 0x41, 0x41, 0x7f, 0x00 }, // ]
{ 0x04, 0x02, 0x01, 0x02, 0x04 }, // ^
{ 0x40, 0x40, 0x40, 0x40, 0x40 }, // _
{ 0x00, 0x01, 0x02, 0x04, 0x00 }, // `
{ 0x20, 0x54, 0x54, 0x54, 0x78 }, // a
{ 0x7f, 0x48, 0x44, 0x44, 0x38 }, // b
{ 0x38, 0x44, 0x44, 0x44, 0x20 }, // c
{ 0x38, 0x44, 0x44, 0x48, 0x7f }, // d
{ 0x38, 0x54, 0x54, 0x54, 0x18 }, // e
{ 0x08, 0x7e, 0x09, 0x01, 0x02 }, // f
{ 0x0c, 0x52, 0x52, 0x52, 0x3e }, // g
{ 0x7f, 0x08, 0x04, 0x04, 0x78 }, // h
{ 0x00, 0x44, 0x7d, 0x40, 0x00 }, // i
{ 0x20, 0x40, 0x44, 0x3d, 0x00 }, // j
{ 0x7f, 0x10, 0x28, 0x44, 0x00 }, // k
{ 0x00, 0x41, 0x7f, 0x40, 0x00 }, // l
{ 0x7c, 0x04, 0x18, 0x04, 0x78 }, // m
{ 0x7c, 0x08, 0x04, 0x04, 0x78 }, // n
{ 0x38, 0x44, 0x44, 0x44, 0x38 }, // o
{ 0x7c, 0x14, 0x14, 0x14, 0x08 }, // p
{ 0x08, 0x14, 0x14, 0x18, 0x7c }, // q
{ 0x7c, 0x08, 0x04, 0x04, 0x08 }, // r
{ 0x48, 0x54, 0x54, 0x54, 0x20 }, // s
```

```
    { 0x04, 0x3f, 0x44, 0x40, 0x20 }, // t
    { 0x3c, 0x40, 0x40, 0x20, 0x7c }, // u
    { 0x1c, 0x20, 0x40, 0x20, 0x1c }, // v
    { 0x3c, 0x40, 0x30, 0x40, 0x3c }, // w
    { 0x44, 0x28, 0x10, 0x28, 0x44 }, // x
    { 0x0c, 0x50, 0x50, 0x50, 0x3c }, // y
    { 0x44, 0x64, 0x54, 0x4c, 0x44 }, // z
    { 0x00, 0x08, 0x36, 0x41, 0x00 }, // {
    { 0x00, 0x00, 0x7f, 0x00, 0x00 }, // |
    { 0x00, 0x41, 0x36, 0x08, 0x00 }, // }
    { 0x02, 0x01, 0x02, 0x04, 0x02 }, // ~
    { 0x00, 0x00, 0x00, 0x00, 0x00 }
};

//*****************************************************************************
//
// The sequence of commands used to initialize the SSD1329 controller.  Each
// command is described as follows:  there is a byte specifying the number of
// bytes in the command sequence, followed by that many bytes of command data.
// Note:  This initialization sequence is derived from RIT App Note for
// the P14201.  Values used are from the RIT app note, except where noted.
//
//*****************************************************************************
static const unsigned char g_pucRIT128x96x4Init[] =
{
    //
    // Unlock commands
    //
    3, 0xFD, 0x12, 0xe3,

    //
    // Display off
    //
    2, 0xAE, 0xe3,

    //
    // Icon off
    //
    3, 0x94, 0, 0xe3,

    //
    // Multiplex ratio
    //
    3, 0xA8, 95, 0xe3,

    //
    // Contrast
    //
    3, 0x81, 0xb7, 0xe3,
```

```
//
// Pre-charge current
//
3, 0x82, 0x3f, 0xe3,

//
// Display Re-map
//
3, 0xA0, RIT_INIT_REMAP, 0xe3,

//
// Display Start Line
//
3, 0xA1, 0, 0xe3,

//
// Display Offset
//
3, 0xA2, RIT_INIT_OFFSET, 0xe3,

//
// Display Mode Normal
//
2, 0xA4, 0xe3,

//
// Phase Length
//
3, 0xB1, 0x11, 0xe3,

//
// Frame frequency
//
3, 0xB2, 0x23, 0xe3,

//
// Front Clock Divider
//
3, 0xB3, 0xe2, 0xe3,

//
// Set gray scale table.  App note uses default command:
// 2, 0xB7, 0xe3
// This gray scale attempts some gamma correction to reduce the
// the brightness of the low levels.
//
17, 0xB8, 1, 2, 3, 4, 5, 6, 8, 10, 12, 14, 16, 19, 22, 26, 30, 0xe3,
```

```
    //
    // Second pre-charge period. App note uses value 0x04.
    //
    3, 0xBB, 0x01, 0xe3,

    //
    // Pre-charge voltage
    //
    3, 0xBC, 0x3f, 0xe3,

    //
    // Display ON
    //
    2, 0xAF, 0xe3,
};

//*****************************************************************************
//
//! \internal
//!
//! Write a sequence of command bytes to the SSD1329 controller.
//!
//! The data is written in a polled fashion; this function will not return
//! until the entire byte sequence has been written to the controller.
//!
//! \return None.
//
//*****************************************************************************
static void
RITWriteCommand(const unsigned char *pucBuffer, unsigned long ulCount)
{
    //
    // Return if SSI port is not enabled for RIT display.
    //
    if(!HWREGBITW(&g_ulSSIFlags, FLAG_SSI_ENABLED))
    {
        return;
    }

    //
    // See if data mode is enabled.
    //
    if(HWREGBITW(&g_ulSSIFlags, FLAG_DC_HIGH))
    {
        //
        // Wait until the SSI is not busy, meaning that all previous data has
        // been transmitted.
        //
        while(SSIBusy(SSI0_BASE))
```

```
        {
        }

        //
        // Clear the command/control bit to enable command mode.
        //
        GPIOPinWrite(GPIO_OLEDDC_BASE, GPIO_OLEDDC_PIN, 0);
        HWREGBITW(&g_ulSSIFlags, FLAG_DC_HIGH) = 0;
    }

    //
    // Loop while there are more bytes left to be transferred.
    //
    while(ulCount != 0)
    {
        //
        // Write the next byte to the controller.
        //
        SSIDataPut(SSI0_BASE, *pucBuffer++);

        //
        // Decrement the BYTE counter.
        //
        ulCount--;
    }
}

//*****************************************************************************
//
//! \internal
//!
//! Write a sequence of data bytes to the SSD1329 controller.
//!
//! The data is written in a polled fashion; this function will not return
//! until the entire byte sequence has been written to the controller.
//!
//! \return None.
//
//*****************************************************************************
static void
RITWriteData(const unsigned char *pucBuffer, unsigned long ulCount)
{
    //
    // Return if SSI port is not enabled for RIT display.
    //
    if(!HWREGBITW(&g_ulSSIFlags, FLAG_SSI_ENABLED))
    {
        return;
    }
```

```
    //
    // See if command mode is enabled.
    //
    if(!HWREGBITW(&g_ulSSIFlags, FLAG_DC_HIGH))
    {
        //
        // Wait until the SSI is not busy, meaning that all previous commands
        // have been transmitted.
        //
        while(SSIBusy(SSI0_BASE))
        {
        }

        //
        // Set the command/control bit to enable data mode.
        //
        GPIOPinWrite(GPIO_OLEDDC_BASE, GPIO_OLEDDC_PIN, GPIO_OLEDDC_PIN);
        HWREGBITW(&g_ulSSIFlags, FLAG_DC_HIGH) = 1;
    }

    //
    // Loop while there are more bytes left to be transferred.
    //
    while(ulCount != 0)
    {
        //
        // Write the next byte to the controller.
        //
        SSIDataPut(SSI0_BASE, *pucBuffer++);

        //
        // Decrement the BYTE counter.
        //
        ulCount--;
    }
}

//*****************************************************************************
//
//! Clears the OLED display.
//!
//! This function will clear the display RAM.  All pixels in the display will
//! be turned off.
//!
//! \return None.
//
//*****************************************************************************
void
```

```
RIT128x96x4Clear(void)
{
    static const unsigned char pucCommand1[] = { 0x15, 0, 63 };
    static const unsigned char pucCommand2[] = { 0x75, 0, 127 };
    unsigned long ulRow, ulColumn;

    //
    // Clear out the buffer used for sending bytes to the display.
    //
    *(unsigned long *)&g_pucBuffer[0] = 0;
    *(unsigned long *)&g_pucBuffer[4] = 0;

    //
    // Set the window to fill the entire display.
    //
    RITWriteCommand(pucCommand1, sizeof(pucCommand1));
    RITWriteCommand(pucCommand2, sizeof(pucCommand2));
    RITWriteCommand(g_pucRIT128x96x4HorizontalInc,
                    sizeof(g_pucRIT128x96x4HorizontalInc));

    //
    // Loop through the rows
    //
    for(ulRow = 0; ulRow < 96; ulRow++)
    {
        //
        // Loop through the columns.  Each byte is two pixels,
        // and the buffer hold 8 bytes, so 16 pixels are cleared
        // at a time.
        //
        for(ulColumn = 0; ulColumn < 128; ulColumn += 8 * 2)
        {
            //
            // Write 8 clearing bytes to the display, which will
            // clear 16 pixels across.
            //
            RITWriteData(g_pucBuffer, sizeof(g_pucBuffer));
        }
    }
}

//*****************************************************************************
//
//! Displays a string on the OLED display.
//!
//! \param pcStr is a pointer to the string to display.
//! \param ulX is the horizontal position to display the string, specified in
//! columns from the left edge of the display.
//! \param ulY is the vertical position to display the string, specified in
```

```
//! rows from the top edge of the display.
//! \param ucLevel is the 4-bit gray scale value to be used for displayed text.
//!
//! This function will draw a string on the display.  Only the ASCII characters
//! between 32 (space) and 126 (tilde) are supported; other characters will
//! result in random data being draw on the display (based on whatever appears
//! before/after the font in memory).  The font is mono-spaced, so characters
//! such as ``i'' and ``l'' have more white space around them than characters
//! such as ``m'' or ``w''.
//!
//! If the drawing of the string reaches the right edge of the display, no more
//! characters will be drawn.  Therefore, special care is not required to avoid
//! supplying a string that is ``too long'' to display.
//!
//! \note Because the OLED display packs 2 pixels of data in a single byte, the
//! parameter \e ulX must be an even column number (for example, 0, 2, 4, and
//! so on).
//!
//! \return None.
//
//*****************************************************************************
void
RIT128x96x4StringDraw(const char *pcStr, unsigned long ulX,
                      unsigned long ulY, unsigned char ucLevel)
{
    unsigned long ulIdx1, ulIdx2;
    unsigned char ucTemp;

    //
    // Check the arguments.
    //
    ASSERT(ulX < 128);
    ASSERT((ulX & 1) == 0);
    ASSERT(ulY < 96);
    ASSERT(ucLevel < 16);

    //
    // Setup a window starting at the specified column and row, ending
    // at the right edge of the display and 8 rows down (single character row).
    //
    g_pucBuffer[0] = 0x15;
    g_pucBuffer[1] = ulX / 2;
    g_pucBuffer[2] = 63;
    RITWriteCommand(g_pucBuffer, 3);
    g_pucBuffer[0] = 0x75;
    g_pucBuffer[1] = ulY;
    g_pucBuffer[2] = ulY + 7;
    RITWriteCommand(g_pucBuffer, 3);
    RITWriteCommand(g_pucRIT128x96x4VerticalInc,
```

```
            sizeof(g_pucRIT128x96x4VerticalInc));

    //
    // Loop while there are more characters in the string.
    //
    while(*pcStr != 0)
    {
        //
        // Get a working copy of the current character and convert to an
        // index into the character bit-map array.
        //
        ucTemp = *pcStr++ & 0x7f;
        if(ucTemp < ' ')
        {
            ucTemp = 0;
        }
        else
        {
            ucTemp -= ' ';
        }

        //
        // Build and display the character buffer.
        //
        for(ulIdx1 = 0; ulIdx1 < 6; ulIdx1 += 2)
        {
            //
            // Convert two columns of 1-bit font data into a single data
            // byte column of 4-bit font data.
            //
            for(ulIdx2 = 0; ulIdx2 < 8; ulIdx2++)
            {
                g_pucBuffer[ulIdx2] = 0;
                if(g_pucFont[ucTemp][ulIdx1] & (1 << ulIdx2))
                {
                    g_pucBuffer[ulIdx2] = (ucLevel << 4) & 0xf0;
                }
                if((ulIdx1 < 4) &&
                   (g_pucFont[ucTemp][ulIdx1 + 1] & (1 << ulIdx2)))
                {
                    g_pucBuffer[ulIdx2] |= (ucLevel << 0) & 0x0f;
                }
            }

            //
            // Send this byte column to the display.
            //
            RITWriteData(g_pucBuffer, 8);
            ulX += 2;
```

```
            //
            // Return if the right side of the display has been reached.
            //
            if(ulX == 128)
            {
                return;
            }
        }
    }
}

//*****************************************************************************
//
//! Displays an image on the OLED display.
//!
//! \param pucImage is a pointer to the image data.
//! \param ulX is the horizontal position to display this image, specified in
//! columns from the left edge of the display.
//! \param ulY is the vertical position to display this image, specified in
//! rows from the top of the display.
//! \param ulWidth is the width of the image, specified in columns.
//! \param ulHeight is the height of the image, specified in rows.
//!
//! This function will display a bitmap graphic on the display.  Because of the
//! format of the display RAM, the starting column (\e ulX) and the number of
//! columns (\e ulWidth) must be an integer multiple of two.
//!
//! The image data is organized with the first row of image data appearing left
//! to right, followed immediately by the second row of image data.  Each byte
//! contains the data for two columns in the current row, with the leftmost
//! column being contained in bits 7:4 and the rightmost column being contained
//! in bits 3:0.
//!
//! For example, an image six columns wide and seven scan lines tall would
//! be arranged as follows (showing how the twenty one bytes of the image would
//! appear on the display):
//!
//! \verbatim
//!     +-------------------+-------------------+-------------------+
//!     |      Byte 0       |      Byte 1       |      Byte 2       |
//!     +---------+---------+---------+---------+---------+---------+
//!     | 7 6 5 4 | 3 2 1 0 | 7 6 5 4 | 3 2 1 0 | 7 6 5 4 | 3 2 1 0 |
//!     +---------+---------+---------+---------+---------+---------+
//!     |      Byte 3       |      Byte 4       |      Byte 5       |
//!     +---------+---------+---------+---------+---------+---------+
//!     | 7 6 5 4 | 3 2 1 0 | 7 6 5 4 | 3 2 1 0 | 7 6 5 4 | 3 2 1 0 |
//!     +---------+---------+---------+---------+---------+---------+
//!     |      Byte 6       |      Byte 7       |      Byte 8       |
```

```
//!     +---------+---------+---------+---------+---------+---------+
//!     | 7 6 5 4 | 3 2 1 0 | 7 6 5 4 | 3 2 1 0 | 7 6 5 4 | 3 2 1 0 |
//!     +---------+---------+---------+---------+---------+---------+
//!     |      Byte 9       |      Byte 10      |      Byte 11      |
//!     +---------+---------+---------+---------+---------+---------+
//!     | 7 6 5 4 | 3 2 1 0 | 7 6 5 4 | 3 2 1 0 | 7 6 5 4 | 3 2 1 0 |
//!     +---------+---------+---------+---------+---------+---------+
//!     |      Byte 12      |      Byte 13      |      Byte 14      |
//!     +---------+---------+---------+---------+---------+---------+
//!     | 7 6 5 4 | 3 2 1 0 | 7 6 5 4 | 3 2 1 0 | 7 6 5 4 | 3 2 1 0 |
//!     +---------+---------+---------+---------+---------+---------+
//!     |      Byte 15      |      Byte 16      |      Byte 17      |
//!     +---------+---------+---------+---------+---------+---------+
//!     | 7 6 5 4 | 3 2 1 0 | 7 6 5 4 | 3 2 1 0 | 7 6 5 4 | 3 2 1 0 |
//!     +---------+---------+---------+---------+---------+---------+
//!     |      Byte 18      |      Byte 19      |      Byte 20      |
//!     +---------+---------+---------+---------+---------+---------+
//!     | 7 6 5 4 | 3 2 1 0 | 7 6 5 4 | 3 2 1 0 | 7 6 5 4 | 3 2 1 0 |
//!     +---------+---------+---------+---------+---------+---------+
//! \endverbatim
//!
//! \return None.
//
//*****************************************************************************
void
RIT128x96x4ImageDraw(const unsigned char *pucImage, unsigned long ulX,
                     unsigned long ulY, unsigned long ulWidth,
                     unsigned long ulHeight)
{
    //
    // Check the arguments.
    //
    ASSERT(ulX < 128);
    ASSERT((ulX & 1) == 0);
    ASSERT(ulY < 96);
    ASSERT((ulX + ulWidth) <= 128);
    ASSERT((ulY + ulHeight) <= 96);
    ASSERT((ulWidth & 1) == 0);

    //
    // Setup a window starting at the specified column and row, and ending
    // at the column + width and row+height.
    //
    g_pucBuffer[0] = 0x15;
    g_pucBuffer[1] = ulX / 2;
    g_pucBuffer[2] = (ulX + ulWidth - 2) / 2;
    RITWriteCommand(g_pucBuffer, 3);
    g_pucBuffer[0] = 0x75;
    g_pucBuffer[1] = ulY;
```

```
    g_pucBuffer[2] = ulY + ulHeight - 1;
    RITWriteCommand(g_pucBuffer, 3);
    RITWriteCommand(g_pucRIT128x96x4HorizontalInc,
                    sizeof(g_pucRIT128x96x4HorizontalInc));

    //
    // Loop while there are more rows to display.
    //
    while(ulHeight--)
    {
        //
        // Write this row of image data.
        //
        RITWriteData(pucImage, (ulWidth / 2));

        //
        // Advance to the next row of the image.
        //
        pucImage += (ulWidth / 2);
    }
}

//*****************************************************************************
//
//! Enable the SSI component of the OLED display driver.
//!
//! \param ulFrequency specifies the SSI Clock Frequency to be used.
//!
//! This function initializes the SSI interface to the OLED display.
//!
//! \return None.
//
//*****************************************************************************
void
RIT128x96x4Enable(unsigned long ulFrequency)
{
    //
    // Disable the SSI port.
    //
    SSIDisable(SSI0_BASE);

    //
    // Configure the SSI0 port for master mode.
    //
    SSIConfigSetExpClk(SSI0_BASE, SysCtlClockGet(), SSI_FRF_MOTO_MODE_3,
                       SSI_MODE_MASTER, ulFrequency, 8);

    //
    // (Re)Enable SSI control of the FSS pin.
```

```
    //
    GPIOPinTypeSSI(GPIO_PORTA_BASE, GPIO_PIN_3);
    GPIOPadConfigSet(GPIO_PORTA_BASE, GPIO_PIN_3, GPIO_STRENGTH_8MA,
                     GPIO_PIN_TYPE_STD_WPU);

    //
    // Enable the SSI port.
    //
    SSIEnable(SSI0_BASE);

    //
    // Indicate that the RIT driver can use the SSI Port.
    //
    HWREGBITW(&g_ulSSIFlags, FLAG_SSI_ENABLED) = 1;
}

//*****************************************************************************
//
//! Enable the SSI component of the OLED display driver.
//!
//! This function initializes the SSI interface to the OLED display.
//!
//! \return None.
//
//*****************************************************************************
void
RIT128x96x4Disable(void)
{
    unsigned long ulTemp;

    //
    // Indicate that the RIT driver can no longer use the SSI Port.
    //
    HWREGBITW(&g_ulSSIFlags, FLAG_SSI_ENABLED) = 0;

    //
    // Drain the receive fifo.
    //
    while(SSIDataGetNonBlocking(SSI0_BASE, &ulTemp) != 0)
    {
    }

    //
    // Disable the SSI port.
    //
    SSIDisable(SSI0_BASE);

    //
    // Disable SSI control of the FSS pin.
```

```
    //
    GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_3);
    GPIOPadConfigSet(GPIO_PORTA_BASE, GPIO_PIN_3, GPIO_STRENGTH_8MA,
                     GPIO_PIN_TYPE_STD_WPU);
    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3, GPIO_PIN_3);
}

//*****************************************************************************
//
//! Initialize the OLED display.
//!
//! \param ulFrequency specifies the SSI Clock Frequency to be used.
//!
//! This function initializes the SSI interface to the OLED display and
//! configures the SSD1329 controller on the panel.
//!
//! \return None.
//
//*****************************************************************************
void
RIT128x96x4Init(unsigned long ulFrequency)
{
    unsigned long ulIdx;

    //
    // Enable the SSI0 and GPIO port blocks as they are needed by this driver.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIO_OLEDDC);

    //
    // Configure the SSI0CLK and SSIOTX pins for SSI operation.
    //
    GPIOPinTypeSSI(GPIO_PORTA_BASE, GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_5);
    GPIOPadConfigSet(GPIO_PORTA_BASE, GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_5,
                     GPIO_STRENGTH_8MA, GPIO_PIN_TYPE_STD_WPU);

    //
    // Configure the GPIO port pin used as a D/Cn signal for OLED device,
    // and the port pin used to enable power to the OLED panel.
    //
    GPIOPinTypeGPIOOutput(GPIO_OLEDDC_BASE, GPIO_OLEDDC_PIN |
GPIO_OLEDEN_PIN);
    GPIOPadConfigSet(GPIO_OLEDDC_BASE, GPIO_OLEDDC_PIN | GPIO_OLEDEN_PIN,
                     GPIO_STRENGTH_8MA, GPIO_PIN_TYPE_STD);
    GPIOPinWrite(GPIO_OLEDDC_BASE, GPIO_OLEDDC_PIN | GPIO_OLEDEN_PIN,
                 GPIO_OLEDDC_PIN | GPIO_OLEDEN_PIN);
    HWREGBITW(&g_ulSSIFlags, FLAG_DC_HIGH) = 1;
```

```
    //
    // Configure and enable the SSI0 port for master mode.
    //
    RIT128x96x4Enable(ulFrequency);

    //
    // Clear the frame buffer.
    //
    RIT128x96x4Clear();

    //
    // Initialize the SSD1329 controller.  Loop through the initialization
    // sequence array, sending each command "string" to the controller.
    //
    for(ulIdx = 0; ulIdx < sizeof(g_pucRIT128x96x4Init);
        ulIdx += g_pucRIT128x96x4Init[ulIdx] + 1)
    {
        //
        // Send this command.
        //
        RITWriteCommand(g_pucRIT128x96x4Init + ulIdx + 1,
                        g_pucRIT128x96x4Init[ulIdx] - 1);
    }
}

//*****************************************************************************
//
//! Turns on the OLED display.
//!
//! This function will turn on the OLED display, causing it to display the
//! contents of its internal frame buffer.
//!
//! \return None.
//
//*****************************************************************************
void
RIT128x96x4DisplayOn(void)
{
    unsigned long ulIdx;

    //
    // Initialize the SSD1329 controller.  Loop through the initialization
    // sequence array, sending each command "string" to the controller.
    //
    for(ulIdx = 0; ulIdx < sizeof(g_pucRIT128x96x4Init);
        ulIdx += g_pucRIT128x96x4Init[ulIdx] + 1)
    {
        //
```

```
        // Send this command.
        //
        RITWriteCommand(g_pucRIT128x96x4Init + ulIdx + 1,
                        g_pucRIT128x96x4Init[ulIdx] - 1);
    }
}

//*****************************************************************************
//
//! Turns off the OLED display.
//!
//! This function will turn off the OLED display.  This will stop the scanning
//! of the panel and turn off the on-chip DC-DC converter, preventing damage to
//! the panel due to burn-in (it has similar characters to a CRT in this
//! respect).
//!
//! \return None.
//
//*****************************************************************************
void
RIT128x96x4DisplayOff(void)
{
    static const unsigned char pucCommand1[] =
    {
        0xAE, 0xe3
    };

    //
    // Put the display to sleep.
    //
    RITWriteCommand(pucCommand1, sizeof(pucCommand1));
}

//*****************************************************************************
//
// Close the Doxygen group.
//! @}
//
//*****************************************************************************
```

rit128x96x.h

```
//*****************************************************************************
//
// rit128x96x4.h - Prototypes for the driver for the RITEK 128x96x4 graphical
//                 OLED display.
//
// Copyright (c) 2007-2010 Texas Instruments Incorporated.  All rights reserved.
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
// This is part of revision 6075 of the EK-LM3S8962 Firmware Package.
//
//*****************************************************************************

#ifndef __RIT128X96X4_H__
#define __RIT128X96X4_H__

//*****************************************************************************
//
// Prototypes for the driver APIs.
//
//*****************************************************************************

/* OLED defines */
#define OLED_BUFFER 2
#define OLED_LINES 5
#define OLED_COLUMNS 21

/* OLED macros */
#define OLED_MIN(a, b) (a < b ? a : b)
#define _OLED_Rollback(c, loc) {\

	int i;\

	for(i = 0; i < OLED_LINES; i++) {\
```

```
			strcpy(c[i], c[i + 1]);\
			_OLED_Message(loc, i, clear, 0);\
			_OLED_Message(loc, i, c[i], OLED_Get_Color());\
		}\
																}

static enum
{
	TOP, BOTTOM, NUM_DEVICES
} OLED_ENUM;

void OLED_Init(unsigned char color);
void OLED_Set_Color(unsigned char color);
/*inline*/ unsigned char OLED_Get_Color(void);
void OLED_Out(int device, const char * string);
void OLED_Clear(int device);
static void _OLED_Message(int device, unsigned int line, const char *string, unsigned char color);
static int _OLED_Find(const char* string, char c);

extern void RIT128x96x4Clear(void);
extern void RIT128x96x4StringDraw(const char *pcStr,
                                  unsigned long ulX,
                                  unsigned long ulY,
                                  unsigned char ucLevel);
extern void RIT128x96x4ImageDraw(const unsigned char *pucImage,
                                 unsigned long ulX,
                                 unsigned long ulY,
                                 unsigned long ulWidth,
                                 unsigned long ulHeight);
extern void RIT128x96x4Init(unsigned long ulFrequency);
extern void RIT128x96x4Enable(unsigned long ulFrequency);
extern void RIT128x96x4Disable(void);
extern void RIT128x96x4DisplayOn(void);
extern void RIT128x96x4DisplayOff(void);

#endif // __RIT128X96X4_H__
```

Shell.c

```
#include "shell.h"
#include "UART.h"
#include "OS.h"
#include "debug.h"
#include <string.h>
#include <stdlib.h>

#if DEBUG == 1
	#include "rit128x96x4.h"
#endif

typedef struct {
  char name[10], val[25];
	char set;
} _SH_Env_Var;

typedef struct {
	char command[25];
	char arguments[10][25];
} _SH_Command;

static _SH_Env_Var* _SH_setVar(const char* varName, const char* newVal);
static char* _SH_getVar(const char* varName);
static _SH_Env_Var* _SH_findVar(const char* varName);
static int _SH_Execute(_SH_Command command);
static _SH_Command _SH_Create_Command(char* input);
static _SH_Command _SH_Parse_Command(const char* input);

static _SH_Env_Var _SH_Env[SH_NUM_VARS];

/* Retrieves the value of an environment variable.
 * param: const char* varName, name of variable to retrieve
 * return: value of variable, null string if not set
 */
static char* _SH_getVar(const char* varName)
{
  _SH_Env_Var* temp = _SH_findVar(varName);
  if (temp != SH_NULL && temp->set == 1) {
    #if DEBUG == 1
      OLED_Out(TOP, "Got");
      OLED_Out(TOP, temp->name);
      OLED_Out(TOP, "value:");
      OLED_Out(TOP, temp->val);
    #endif
    return temp->val;
  }
  return "Value not set";
```

```
}

/* Sets an environment variable
 * param: const char* varName, variable name to set
 * param: const char* newVal, value of variable
 * return: pointer to environment variable.
                                        if no space to store variable, return SH_ERROR
 */
static _SH_Env_Var* _SH_setVar(const char* name, const char* val)
{
	_SH_Env_Var* var = _SH_findVar(name);
	if(var == SH_NULL)
		return SH_NULL;
	strcpy(var->name, name);
	strcpy(var->val, val);
	var->set = 1;
  #if DEBUG == 1
    OLED_Out(BOTTOM, "Set");
    OLED_Out(BOTTOM, var->name);
    OLED_Out(BOTTOM, "value:");
    OLED_Out(BOTTOM, var->val);
    OLED_Out(BOTTOM, val);
  #endif
	return var;
}

/* Find variable in environment array. If not found, attempt to find space for it.
 * param: const char* varName, variable to find.
 * return: pointer to environment variable
 */
static _SH_Env_Var* _SH_findVar(const char* varName)
{
  int i;
  for(i = 0; i < SH_NUM_VARS; i++)
    if((_SH_Env[i].set == 1) && (strcmp(_SH_Env[i].name, varName) == 0))
      return &_SH_Env[i];
	for(i = 0; i < SH_NUM_VARS; i++)
      if(_SH_Env[i].set == 0)
        return &_SH_Env[i];
  return SH_NULL;
}

void SH_Init(void)
{
	int i;
	for(i = 0; i < SH_NUM_VARS; i++)
		_SH_Env[i].set = 0;

	UART_Init();
```

```
	_SH_setVar(SH_PROMPT_NAME, ">");
	while(1)
	{
		char input[SH_MAX_LENGTH] = {0};
		/* Show prompt */
		UART_OutString(SH_NL), UART_OutString(_SH_getVar(SH_PROMPT_NAME));
		/* Input command */
		UART_InString(input, SH_MAX_LENGTH);
		UART_OutString(SH_NL);
		/* Construct and execute command */
		_SH_Execute(_SH_Parse_Command(input));
    memset(input, 0, SH_MAX_LENGTH);
  }
}

/* Maybe hash the commands so O(1) lookup time */
static int _SH_Execute(_SH_Command command)
{
  int exitCode = 0;
  char buff[2] = {0};
  switch (command.command[0]) {
    case 's':
      exitCode = (_SH_setVar(command.arguments[0], command.arguments[1]) == SH_NULL);
      break;
    case 'e':
      UART_OutString(_SH_getVar(command.arguments[0]));
				exitCode = 0;
      break;
    case 't':
      UART_OutUDec(OS_MsTime());
				exitCode = 0;
      break;
    default:
      UART_OutString(SH_INVALID_CMD);
      UART_OutString(command.command);
      UART_OutString(SH_NL);
				exitCode = 1;
      break;
  }
  buff[0] = exitCode + 0x30;
//  _SH_setVar("?", buff);
	return exitCode;
}

static _SH_Command _SH_Create_Command(char* input)
{
	_SH_Command command;
  char * delims = " ";
	char* pt;
```

```
	//input[strlen(input)-1] = '\0';
	pt = strtok(input, delims);
	strcpy(command.command, pt);
	#if DEBUG == 1
    UART_OutString(command.command);
    UART_OutString(SH_NL);
		OLED_Out(TOP, command.command);
	#endif
	pt = strtok(NULL, delims);
	while(pt != NULL)
	{
		static int i = 0;
		strcpy(command.arguments[i++], pt);
		pt = strtok(NULL, delims);
		#if DEBUG == 1
      UART_OutString(command.arguments[i-1]);
      UART_OutString(SH_NL);
			OLED_Out(TOP, command.arguments[i-1]);
		#endif
	}
	return command;
}

static _SH_Command _SH_Parse_Command(const char* input) {
  _SH_Command command;
  int base, offset, argNum = 0;
  base = offset = 0;
  while(input[offset] != 0 && input[offset++] != ' ' ) {
    ;
  }
  base = offset;
  memcpy(command.command, input, offset);
  command.command[offset] = 0;
  while(input[offset] != 0) {
    while(input[offset] != 0 && input[offset] != ' ')  {
      offset++;
    }
    memcpy(command.arguments[argNum], &input[base], offset - base);
    command.arguments[argNum++][offset - base] = 0;
    UART_OutString(command.arguments[argNum - 1]);
    base = ++offset;
  }
  return command;
}
```

Shell.h

```
#ifndef __SHELL_H__
#define __SHELL_H__

void SH_Init(void);

#define SH_MAX_LENGTH 128
#define SH_NUM_VARS 64
#define SH_NL "\r\n"
#define SH_ERROR -1
#define SH_NULL NULL
#define SH_INVALID_CMD "Not a recognized command" SH_NL
#define SH_PROMPT_NAME "PROMPT"

#endif //__SHELL_H__
```

UART.c

```
// UART.c
// Runs on LM3S1968
// Use UART0 to implement bidirectional data transfer to and from a
// computer running HyperTerminal.  This time, interrupts and FIFOs
// are used.
// Daniel Valvano
// October 9, 2011
// Modified by EE345L students Charlie Gough && Matt Hawk
// Modified by EE345M students Agustinus Darmawan && Mingjie Qiu

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to the Arm Cortex M3",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2011

   Program 5.11 Section 5.6, Program 3.10

 Copyright 2011 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS
SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,
INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

// U0Rx (VCP receive) connected to PA0
// U0Tx (VCP transmit) connected to PA1

#include "FIFO.h"
#include "UART.h"

#include "lm3s8962.h"

void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts
long StartCritical (void);    // previous I bit, disable interrupts
void EndCritical(long sr);    // restore I bit to previous value
void WaitForInterrupt(void);  // low power mode
#define FIFOSIZE   64         // size of the FIFOs (must be power of 2)
#define FIFOSUCCESS 1         // return value on success
#define FIFOFAIL    0         // return value on failure
                              // create index implementation FIFO (see FIFO.h)
```

```
AddIndexFifo(Rx, FIFOSIZE, char, FIFOSUCCESS, FIFOFAIL)
AddIndexFifo(Tx, FIFOSIZE, char, FIFOSUCCESS, FIFOFAIL)

// Initialize UART0
// Baud rate is 115200 bits/sec
void UART_Init(void){
  SYSCTL_RCGC1_R |= SYSCTL_RCGC1_UART0; // activate UART0
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOA; // activate port A
  RxFifo_Init();                        // initialize empty FIFOs
  TxFifo_Init();
  UART0_CTL_R &= ~UART_CTL_UARTEN;      // disable UART
  UART0_IBRD_R = 27;                    // IBRD = int(50,000,000 / (16 * 115,200)) = int(27.1267)
  UART0_FBRD_R = 8;                     // FBRD = int(0.1267 * 64 + 0.5) = 8
                                        // 8 bit word length (no parity bits, one stop bit, FIFOs)
  UART0_LCRH_R = (UART_LCRH_WLEN_8|UART_LCRH_FEN);
  UART0_IFLS_R &= ~0x3F;                // clear TX and RX interrupt FIFO level fields
                                        // configure interrupt for TX FIFO <= 1/8 full
                                        // configure interrupt for RX FIFO >= 1/8 full
  UART0_IFLS_R += (UART_IFLS_TX1_8|UART_IFLS_RX1_8);
                                        // enable TX and RX FIFO interrupts and RX time-out interrupt
  UART0_IM_R |= (UART_IM_RXIM|UART_IM_TXIM|UART_IM_RTIM);
  UART0_CTL_R |= UART_CTL_UARTEN;       // enable UART
  GPIO_PORTA_AFSEL_R |= 0x03;           // enable alt funct on PA1-0
  GPIO_PORTA_DEN_R |= 0x03;             // enable digital I/O on PA1-0
                                        // UART0=priority 2
  NVIC_PRI1_R = (NVIC_PRI1_R&0xFFFF00FF)|0x00004000; // bits 13-15
  NVIC_EN0_R |= NVIC_EN0_INT5;          // enable interrupt 5 in NVIC
  EnableInterrupts();
}
// copy from hardware RX FIFO to software RX FIFO
// stop when hardware RX FIFO is empty or software RX FIFO is full
void static copyHardwareToSoftware(void){
  char letter;
  while(((UART0_FR_R&UART_FR_RXFE) == 0) && (RxFifo_Size() < (FIFOSIZE - 1))){
    letter = UART0_DR_R;
    RxFifo_Put(letter);
  }
}
// copy from software TX FIFO to hardware TX FIFO
// stop when software TX FIFO is empty or hardware TX FIFO is full
void static copySoftwareToHardware(void){
  char letter;
  while(((UART0_FR_R&UART_FR_TXFF) == 0) && (TxFifo_Size() > 0)){
    TxFifo_Get(&letter);
    UART0_DR_R = letter;
  }
}
// input ASCII character from UART
// spin if RxFifo is empty
```

```
unsigned char UART_InChar(void){
  char letter;
  while(RxFifo_Get(&letter) == FIFOFAIL){};
  return(letter);
}
// output ASCII character to UART
// spin if TxFifo is full
void UART_OutChar(unsigned char data){
  while(TxFifo_Put(data) == FIFOFAIL){};
  UART0_IM_R &= ~UART_IM_TXIM;          // disable TX FIFO interrupt
  copySoftwareToHardware();
  UART0_IM_R |= UART_IM_TXIM;           // enable TX FIFO interrupt
}
// at least one of three things has happened:
// hardware TX FIFO goes from 3 to 2 or less items
// hardware RX FIFO goes from 1 to 2 or more items
// UART receiver has timed out
void UART0_Handler(void){
  if(UART0_RIS_R&UART_RIS_TXRIS){       // hardware TX FIFO <= 2 items
    UART0_ICR_R = UART_ICR_TXIC;        // acknowledge TX FIFO
    // copy from software TX FIFO to hardware TX FIFO
    copySoftwareToHardware();
    if(TxFifo_Size() == 0){             // software TX FIFO is empty
      UART0_IM_R &= ~UART_IM_TXIM;      // disable TX FIFO interrupt
    }
  }
  if(UART0_RIS_R&UART_RIS_RXRIS){       // hardware RX FIFO >= 2 items
    UART0_ICR_R = UART_ICR_RXIC;        // acknowledge RX FIFO
    // copy from hardware RX FIFO to software RX FIFO
    copyHardwareToSoftware();
  }
  if(UART0_RIS_R&UART_RIS_RTRIS){       // receiver timed out
    UART0_ICR_R = UART_ICR_RTIC;        // acknowledge receiver time out
    // copy from hardware RX FIFO to software RX FIFO
    copyHardwareToSoftware();
  }
}


//------------UART_OutString------------
// Output String (NULL termination)
// Input: pointer to a NULL-terminated string to be transferred
// Output: none
void UART_OutString(char *pt){
  while(*pt){
    UART_OutChar(*pt);
    pt++;
  }
}
```

```
//------------UART_InUDec------------
// InUDec accepts ASCII input in unsigned decimal format
//     and converts to a 32-bit unsigned number
//     valid range is 0 to 4294967295 (2^32-1)
// Input: none
// Output: 32-bit unsigned number
// If you enter a number above 4294967295, it will return an incorrect value
// Backspace will remove last digit typed
unsigned long UART_InUDec(void){
unsigned long number=0, length=0;
char character;
  character = UART_InChar();
  while(character != CR){ // accepts until <enter> is typed
// The next line checks that the input is a digit, 0-9.
// If the character is not 0-9, it is ignored and not echoed
    if((character>='0') && (character<='9')) {
      number = 10*number+(character-'0');   // this line overflows if above 4294967295
      length++;
      UART_OutChar(character);
    }
// If the input is a backspace, then the return number is
// changed and a backspace is outputted to the screen
    else if((character==BS) && length){
      number /= 10;
      length--;
      UART_OutChar(character);
    }
    character = UART_InChar();
  }
  return number;
}

//-----------------------UART_OutUDec-----------------------
// Output a 32-bit number in unsigned decimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1-10 digits with no space before or after
void UART_OutUDec(unsigned long n){
// This function uses recursion to convert decimal number
//   of unspecified length as an ASCII string
  if(n >= 10){
    UART_OutUDec(n/10);
    n = n%10;
  }
  UART_OutChar(n+'0'); /* n is between 0 and 9 */
}

//---------------------UART_InUHex----------------------------------------
```

```
// Accepts ASCII input in unsigned hexadecimal (base 16) format
// Input: none
// Output: 32-bit unsigned number
// No '$' or '0x' need be entered, just the 1 to 8 hex digits
// It will convert lower case a-f to uppercase A-F
//     and converts to a 16 bit unsigned number
//     value range is 0 to FFFFFFFF
// If you enter a number above FFFFFFFF, it will return an incorrect value
// Backspace will remove last digit typed
unsigned long UART_InUHex(void){
unsigned long number=0, digit, length=0;
char character;
  character = UART_InChar();
  while(character != CR){
    digit = 0x10; // assume bad
    if((character>='0') && (character<='9')){
      digit = character-'0';
    }
    else if((character>='A') && (character<='F')){
      digit = (character-'A')+0xA;
    }
    else if((character>='a') && (character<='f')){
      digit = (character-'a')+0xA;
    }
// If the character is not 0-9 or A-F, it is ignored and not echoed
    if(digit <= 0xF){
      number = number*0x10+digit;
      length++;
      UART_OutChar(character);
    }
// Backspace outputted and return value changed if a backspace is inputted
    else if((character==BS) && length){
      number /= 0x10;
      length--;
      UART_OutChar(character);
    }
    character = UART_InChar();
  }
  return number;
}

//--------------------------UART_OutUHex----------------------------
// Output a 32-bit number in unsigned hexadecimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1 to 8 digits with no space before or after
void UART_OutUHex(unsigned long number){
// This function uses recursion to convert the number of
//   unspecified length as an ASCII string
```

```
  if(number >= 0x10){
    UART_OutUHex(number/0x10);
    UART_OutUHex(number%0x10);
  }
  else{
    if(number < 0xA){
      UART_OutChar(number+'0');
     }
    else{
      UART_OutChar((number-0x0A)+'A');
    }
  }
}

//------------UART_InString------------
// Accepts ASCII characters from the serial port
//    and adds them to a string until <enter> is typed
//    or until max length of the string is reached.
// It echoes each character as it is inputted.
// If a backspace is inputted, the string is modified
//    and the backspace is echoed
// terminates the string with a null character
// uses busy-waiting synchronization on RDRF
// Input: pointer to empty buffer, size of buffer
// Output: Null terminated string
// -- Modified by Agustinus Darmawan + Mingjie Qiu --
void UART_InString(char *bufPt, unsigned short max) {
int length=0;
char character;
  character = UART_InChar();
  while(character != CR){
    if(character == BS){
      if(length){
        bufPt--;
        length--;
        UART_OutChar(BS);
      }
    }
    else if(length < max){
      *bufPt = character;
      bufPt++;
      length++;
      UART_OutChar(character);
    }
    character = UART_InChar();
  }
  *bufPt = 0;
}
```

UART.h

```
// UART2.h
// Runs on LM3S1968
// Use UART0 to implement bidirectional data transfer to and from a
// computer running HyperTerminal.  This time, interrupts and FIFOs
// are used.
// Daniel Valvano
// October 9, 2011

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to the Arm Cortex M3",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2011

 Copyright 2011 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS
SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,
INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

// U0Rx (VCP receive) connected to PA0
// U0Tx (VCP transmit) connected to PA1

// standard ASCII symbols
#define CR   0x0D
#define LF   0x0A
#define BS   0x08
#define ESC  0x1B
#define SP   0x20
#define DEL  0x7F

//------------UART_InChar------------
// Wait for new serial port input
// Initialize the UART for 115,200 baud rate (assuming 50 MHz clock),
// 8 bit word length, no parity bits, one stop bit, FIFOs enabled
// Input: none
// Output: none
void UART_Init(void);

//------------UART_InChar------------
// Wait for new serial port input
```

```
// Input: none
// Output: ASCII code for key typed
unsigned char UART_InChar(void);

//------------UART_OutChar------------
// Output 8-bit to serial port
// Input: letter is an 8-bit ASCII character to be transferred
// Output: none
void UART_OutChar(unsigned char data);

//------------UART_OutString------------
// Output String (NULL termination)
// Input: pointer to a NULL-terminated string to be transferred
// Output: none
void UART_OutString(char *pt);

//------------UART_InUDec------------
// InUDec accepts ASCII input in unsigned decimal format
//     and converts to a 32-bit unsigned number
//     valid range is 0 to 4294967295 (2^32-1)
// Input: none
// Output: 32-bit unsigned number
// If you enter a number above 4294967295, it will return an incorrect value
// Backspace will remove last digit typed
unsigned long UART_InUDec(void);

//-----------------------UART_OutUDec-----------------------
// Output a 32-bit number in unsigned decimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1-10 digits with no space before or after
void UART_OutUDec(unsigned long n);

//---------------------UART_InUHex----------------------------------------
// Accepts ASCII input in unsigned hexadecimal (base 16) format
// Input: none
// Output: 32-bit unsigned number
// No '$' or '0x' need be entered, just the 1 to 8 hex digits
// It will convert lower case a-f to uppercase A-F
//     and converts to a 16 bit unsigned number
//     value range is 0 to FFFFFFFF
// If you enter a number above FFFFFFFF, it will return an incorrect value
// Backspace will remove last digit typed
unsigned long UART_InUHex(void);

//--------------------------UART_OutUHex----------------------------
// Output a 32-bit number in unsigned hexadecimal format
// Input: 32-bit number to be transferred
// Output: none
```

```
// Variable format 1 to 8 digits with no space before or after
void UART_OutUHex(unsigned long number);

//------------UART_InString------------
// Accepts ASCII characters from the serial port
//    and adds them to a string until <enter> is typed
//    or until max length of the string is reached.
// It echoes each character as it is inputted.
// If a backspace is inputted, the string is modified
//    and the backspace is echoed
// terminates the string with a null character
// uses busy-waiting synchronization on RDRF
// Input: pointer to empty buffer, size of buffer
// Output: Null terminated string
// -- Modified by Agustinus Darmawan + Mingjie Qiu --
void UART_InString(char *bufPt, unsigned short max);
```