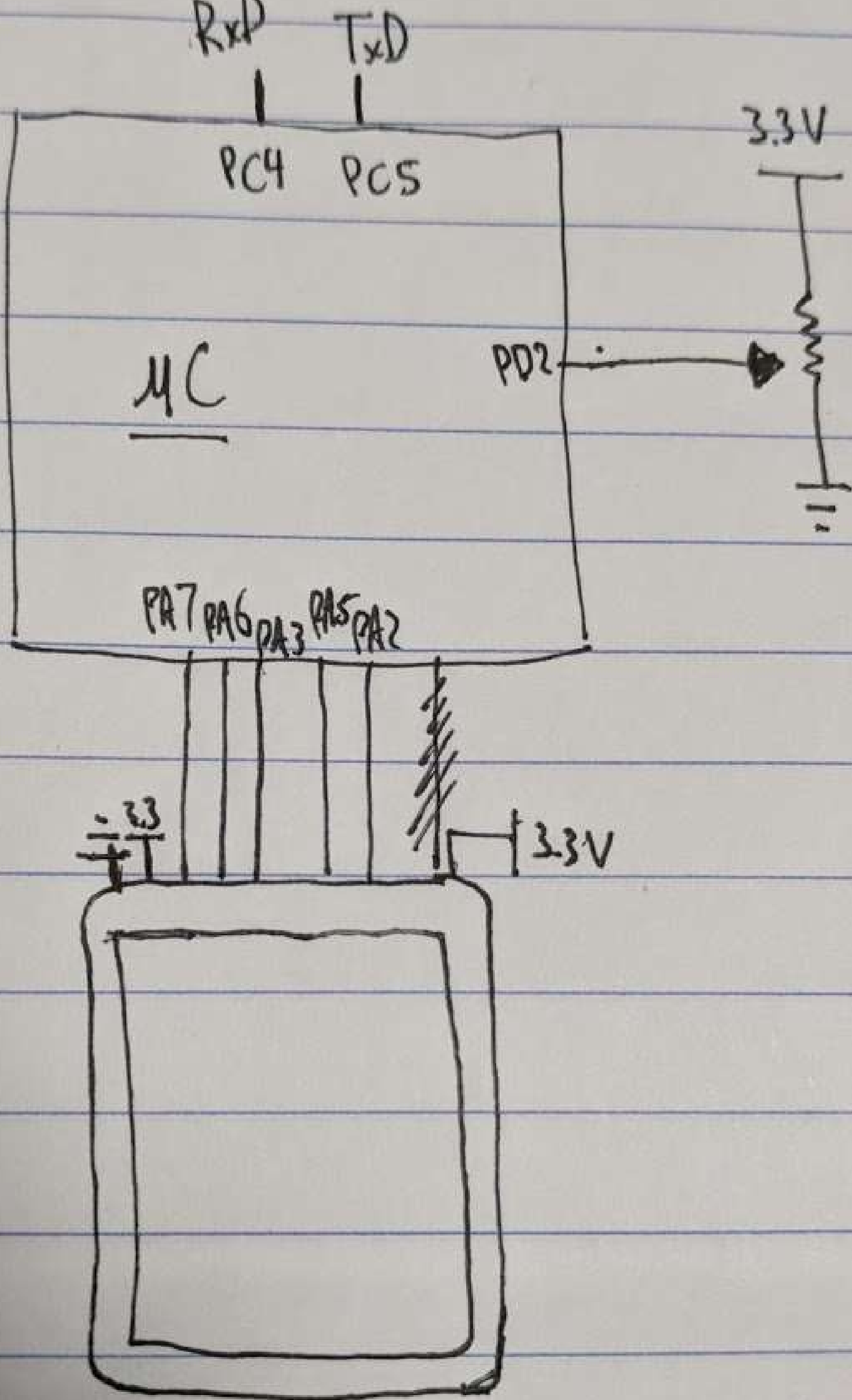

RxD
TxD
PC4
PC5
3.3V
µC
PD2
PA7
PA6
PA3
PA5
PA2
3.3
3.3V

```
// main.c
// Runs on LM4F120 or TM4C123
// Student names: put your names here
// Last modification date: change this to the last modification date or look very silly
// Last Modified: 4/11/2018

// Analog Input connected to PD2=ADC5
// displays on Sitronox ST7735
// PF3, PF2, PF1 are heartbeats
// EE319K Lab 9, use U1Rx connected to PC4 interrupt
// EE319K Lab 9, use U1Tx connected to PC5 busy wait
// EE319K Lab 9 hardware
// System 1        System 2
//   PC4 ----<<----- PC5
//   PC5 ---->>----- PC4
//   Gnd ----------- Gnd

// main1 Understand UART interrupts
// main2 Implement and test the FIFO class on the receiver end
//   import ST7735 code from Lab 7,8
// main3 convert UART0 to UART1, implement busy-wait on transmission
// final main for Lab 9
//   Import SlidePot and ADC code from Lab8.
//   Figure out what to do in UART1_Handler ISR (receive message)
//   Figure out what to do in SysTickHandler (sample, convert, transmit message)
//   Figure out what to do in main (LCD output)

#include <stdint.h>
#include "../tm4c123gh6pm.h"
```

```
#include "PLL.h"
#include "ST7735.h"
#include "PLL.h"
#include "SlidePot.h"
#include "print.h"
#include "UART.h"
#include "FIFO.h"

SlidePot Sensor(2115, 254);
char arr[8];
int TxCounter = 0;

extern "C" void DisableInterrupts(void);
extern "C" void EnableInterrupts(void);
extern "C" void SysTick_Handler(void);
extern Queue RxFifo;

// PF1 should be toggled in UART ISR (receive data)
// PF2 should be toggled in SysTick ISR (60 Hz sampling)
// PF3 should be toggled in main program
#define PF1  (*((volatile uint32_t *)0x40025008))
#define PF2  (*((volatile uint32_t *)0x40025010))
#define PF3  (*((volatile uint32_t *)0x40025020))
#define PF4  (*((volatile uint32_t *)0x40025040))

// **************SysTick_Init*********************
// Initialize Systick periodic interrupts
// Input: interrupt period
//        Units of period are 12.5ns
```

```
//      Maximum is 2^24-1
//      Minimum is determined by length of ISR
// Output: none
void SysTick_Init(unsigned long period){
  NVIC_ST_CTRL_R = 0;              // disable SysTick during setup
  NVIC_ST_RELOAD_R = period;                                    // 60 Hz
  NVIC_ST_CURRENT_R = 0;           // any write to current clears it
  NVIC_ST_CTRL_R = 7;                                                                  // enable
SysTick with core clock
}

// Initialize Port F so PF1, PF2 and PF3 are heartbeats
void PortF_Init(void){
  SYSCTL_RCGCGPIO_R |= 0x20;     // activate port F
  while((SYSCTL_PRGPIO_R&0x20) != 0x20){};
  GPIO_PORTF_DIR_R |=  0x0E;   // output on PF3,2,1 (built-in LED)
  GPIO_PORTF_PUR_R |= 0x10;
  GPIO_PORTF_DEN_R |=  0x1E;   // enable digital I/O on PF
}

// main1 is used to run initial code with UART0 interrupts
// this code should run without changing
// and interact with serial terminal on PC
int main1(void){  // run this program and look at Data
char letter;
  DisableInterrupts();
  PLL_Init(Bus80MHz);     // set system clock to 80 MHz
  PortF_Init();
  UART_Init();   // enable UART
```

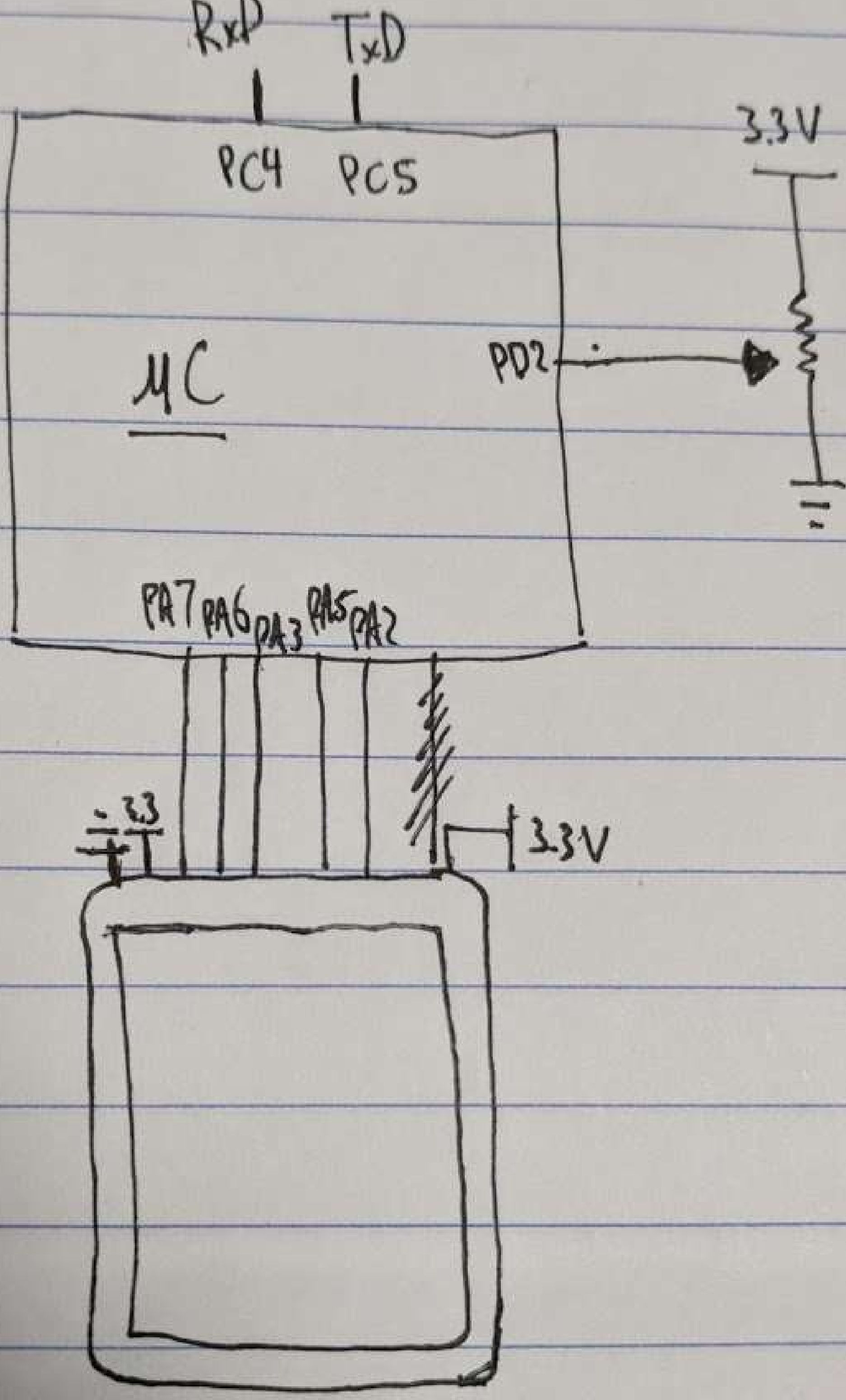
```
  EnableInterrupts();
  UART_OutString((char *)"1234567890\n\r");
  //UART_OutString((char *)"abcdefghij\n\r");
  //UART_OutUDec(12345);
        UART_OutChar(LF);
        UART_OutChar(CR);
  while(1){
    letter = UART_InChar();
    UART_OutChar(letter);
  }
}

// step two, implement the FIFO class
// FIFO.h is prototype
// FIFO.cpp is implementation
// main2 program will test FIFO
Queue FIFO;
int main2(void){
  char data = 0; char out;
  DisableInterrupts();
  PLL_Init(Bus80MHz);     // set system clock to 80 MHz
  ST7735_InitR(INITR_REDTAB);
  PortF_Init();

  while(1){
    int count=0;
    for(int i=0; i<10; i++){
      if(FIFO.Put((data%10) + '0')){
        count++;
```

```
      data++;
     }
    }
    FIFO.Print();
    for(int i=0; i<count; i++){
      FIFO.Get(&out);
    }
    PF2 ^= 0x04;
  }
}

// step 3
// Convert UART0 to UART1
// Move PA1,PA0 to PC5, PC4
// Use your queue class in receiver interrupt
// change receiver interrupt to 1/2 full only
// change transmitter to busy wait
// PF1 toggles in UART ISR
// PF2 toggles in main
int main3(void){
  char OutData = 'A';
  char InData;
  int count = 0;
  uint32_t time=0;
  DisableInterrupts();
  PLL_Init(Bus80MHz);     // set system clock to 80 MHz
  ST7735_InitR(INITR_REDTAB);
  PortF_Init();
  UART_Init();      // enable UART
```

```
  EnableInterrupts();
  while(1){
    time++;
    if((time%100000)==0){
      UART_OutChar(OutData);
      if(OutData == 'Z'){
        OutData = 'A';
      }else{
        OutData++;
      }
    }
    if(UART_InStatus()){
      InData = UART_InChar();
      ST7735_OutChar(InData);
      count++;
      if((count%16)==0){
        ST7735_OutChar('\n');
      }
      PF3 ^= 0x08;
    }
  }
}



// final main program for bidirectional communication
// Sender sends using SysTick Interrupt
// Receiver receives using RX interrupt
int main(void){
  PLL_Init(Bus80MHz);       // Bus clock is 80 MHz
```

```
  ST7735_InitR(INITR_REDTAB);
  ADC_Init();    // initialize to sample ADC
  PortF_Init();
  UART_Init();    // initialize UART
  // you write this

        SysTick_Init(1333333);
        EnableInterrupts();

  while(1){ // runs every 16,67 ms
              char pt;
              if(RxFifo.Get(&pt) && (pt == 0x02)){
                          for(int i = 0; i < 5; i++){
                                while(!RxFifo.Get(&pt));
                                ST7735_OutChar(pt);
                          }
                          ST7735_OutString(" cm");
                          ST7735_SetCursor(0,0);
              }
        }
}

void chop(){
        int distance = Sensor.Distance();
        arr[0] = (0x02);
        arr[1] = (distance/1000 + '0');
        distance %= 1000;
        arr[2] = (0x2E);
        arr[3] = (distance/100 + '0');
```

```
            distance %= 100;
            arr[4] = (distance/10 + '0');
            distance %= 10;
            arr[5] = (distance + '0');
            arr[6] = (0x0D);
            arr[7] = (0x03);
}

void SysTick_Handler(void){ // every 16.67 ms
 //Similar to Lab8 except rather than grab sample,
// form a message, transmit
  PF2 ^= 0x04;  // Heartbeat
  PF2 ^= 0x04;  // Heartbeat
            Sensor.Save(ADC_In());
            chop();
            for(int i = 0; i < 8; i++){
                        UART_OutChar(arr[i]);
            }
            TxCounter++;
  PF2 ^= 0x04;  // Heartbeat
}
```


RxD
TxD
PC4
PC5
3.3V
μC
PD2
PA7
PA6
PA3
PA5
PA2
3.3
3.3V

```
// UART.cpp
// Runs on LM4F120/TM4C123
// This code runs on UART0 with interrupts and a simple FIFO
// EE319K tasks
// 1) Convert to UART1 PC4 PC5
// 2) Implement the FIFO as a class
// 3) Run transmitter with busy-wait synchronization
// 4) Run receiver with 1/2 full FIFO interrupt
// Daniel and Jonathan Valvano
// April 11, 2018


/* This example accompanies the book
  "Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",
  ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2017
  Program 5.11 Section 5.6, Program 3.10

 Copyright 2018 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
// EE319K Lab 9, use U1Rx connected to PC4 interrupt
// EE319K Lab 9, use U1Tx connected to PC5 busy wait
// EE319K Lab 9 hardware
// System 1        System 2
//   PC4 ----<<----- PC5
//   PC5 ---->>----- PC4
//   Gnd ----------- Gnd
#include <stdint.h>
#include "../tm4c123gh6pm.h"

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

#define PF1  (*((volatile uint32_t *)0x40025008))

extern "C" void DisableInterrupts(void); // Disable interrupts
extern "C" long StartCritical (void);    // previous I bit, disable interrupts
extern "C" void EndCritical(long sr);    // restore I bit to previous value
extern "C" void WaitForInterrupt(void);  // low power mode
extern "C" void EnableInterrupts(void);

#define starter 0
// 1 means UART0, TX/RX interrupts, simple FIFO
// 0 means your Lab 9
#if starter
// EE319K - do not use these two FIFOs
```

```
// Two-index implementation of the transmit FIFO
// can hold 0 to TXFIFOSIZE elements
#define NVIC_EN0_INT5    0x00000020  // Interrupt 5 enable
extern "C" void UART0_Handler(void);
#define TXFIFOSIZE 32 // must be a power of 2
#define FIFOSUCCESS 1
#define FIFOFAIL    0

typedef char txDataType;
uint32_t volatile TxPutI;// put next
uint32_t 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+1)&(TXFIFOSIZE-1)) == TxGetI){
    return(FIFOFAIL); // Failed, fifo full
  }
  TxFifo[TxPutI] = data; // put
  TxPutI = (TxPutI+1)&(TXFIFOSIZE-1);  // Success, update
  return(FIFOSUCCESS);
}
```

```
// remove element from front of index FIFO
// return TXFIFOSUCCESS if successful
int TxFifo_Get(txDataType *datapt){
  if(TxPutI == TxGetI){
    return(FIFOFAIL); // Empty if TxPutI=TxGetI
  }
  *datapt = TxFifo[TxGetI];
  TxGetI = (TxGetI+1)&(TXFIFOSIZE-1);  // Success, update
  return(FIFOSUCCESS);
}
// number of elements in index FIFO
// 0 to TXFIFOSIZE-1
uint32_t TxFifo_Size(void){
 return (TxPutI-TxGetI)&(TXFIFOSIZE-1);
}

#define RXFIFOSIZE 32 // must be a power of 2
typedef char rxDataType;
uint32_t volatile RxPutI;// put next
uint32_t volatile RxGetI;// get next
rxDataType static RxFifo[RXFIFOSIZE];
// initialize index FIFO
void RxFifo_Init(void){ long sr;
  sr = StartCritical(); // make atomic
  RxPutI = RxGetI = 0;  // Empty
  EndCritical(sr);
}
// add element to end of index FIFO
// return FIFOSUCCESS if successful
```

```
int RxFifo_Put(rxDataType data){
  if(((RxPutI+1)&(RXFIFOSIZE-1)) == RxGetI){
    return(FIFOFAIL); // Failed, fifo full
  }
  RxFifo[RxPutI] = data; // put
  RxPutI = (RxPutI+1)&(RXFIFOSIZE-1);  // Success, update
  return(FIFOSUCCESS);
}
// remove element from front of index FIFO
// return FIFOSUCCESS if successful
int RxFifo_Get(rxDataType *datapt){
  if(RxPutI == RxGetI){
    return(FIFOFAIL); // Empty if TxPutI=TxGetI
  }
  *datapt = RxFifo[RxGetI];
  RxGetI = (RxGetI+1)&(RXFIFOSIZE-1);  // Success, update
  return(FIFOSUCCESS);
}
// number of elements in index FIFO
// 0 to RXFIFOSIZE-1
uint32_t RxFifo_Size(void){
 return (RxPutI-RxGetI)&(RXFIFOSIZE-1);
}

// Initialize UART0
// Baud rate is 115200 bits/sec
void UART_Init(void){
  SYSCTL_RCGCUART_R |= 0x01;           // activate UART0
  SYSCTL_RCGCGPIO_R |= 0x01;           // activate port A
```

```
  RxFifo_Init();                       // initialize empty FIFOs
  TxFifo_Init();
  UART0_CTL_R &= ~UART_CTL_UARTEN;      // disable UART
  UART0_IBRD_R = 43;                    // IBRD = int(80,000,000 / (16 * 115,200)) = int(43.403)
  UART0_FBRD_R = 26;                    // FBRD = round(0.403 * 64 ) = 26
                                    // 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 |= 0x301;                 // 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
                                    // configure PA1-0 as UART
  GPIO_PORTA_PCTL_R = (GPIO_PORTA_PCTL_R&0xFFFFFF00)+0x00000011;
                                    // 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
}
// 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
char UART_InChar(void){
  char letter;
  while(RxFifo_Get(&letter)){};
  return(letter);
}
bool UART_InStatus(void){
  // true if input data ready
  return RxFifo_Size()>0;
}

// output ASCII character to UART
// spin if TxFifo is full
void UART_OutChar(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){
  PF1  ^= 0x02; // triple toggle debugging
  PF1  ^= 0x02;
  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();
  }
  PF1  ^= 0x02;
```

```
}
// ************Lab 9 TO DO**********

#else
extern "C" void UART1_Handler(void);
#define NVIC_EN0_INT6   0x00000040  // Interrupt 6 enable
Queue RxFifo; // static implementation of class
int RxCounter = 0;
// Initialize UART0
// Baud rate is 115200 bits/sec
// Lab 9
void UART_Init(void){
        volatile int x = 0;
  SYSCTL_RCGCUART_R |= 0x00000002;  // activate UART1
        x++;
        x++;
        x++;
        x++;
  SYSCTL_RCGCGPIO_R |= 0x00000004;  // activate port C
        x++;
        x++;
        x++;
        x++;
  UART1_CTL_R &= ~0x00000001;    // disable UART
  UART1_IBRD_R = 43;     // IBRD = int(80,000,000/(16*115,200)) = int(43.40278)
  UART1_FBRD_R = 26;     // FBRD = round(0.40278 * 64) = 26
  UART1_LCRH_R = 0x00000070;  // 8 bit, no parity bits, one stop, FIFOs
  UART1_CTL_R |= 0x00000301;     // enable UART
```

```
            UART1_IM_R |= 0x10; //enables interrupts for the receiver


            UART1_IFLS_R |= 0x10; //interrupt triggered 1/2 full
            UART1_IFLS_R &= ~0x28;


  GPIO_PORTC_AFSEL_R |= 0x30;    // enable alt funct on PC5-4
  GPIO_PORTC_DEN_R |= 0x30;      // configure PC5-4 as UART1
  GPIO_PORTC_PCTL_R = (GPIO_PORTC_PCTL_R&0xFF00FFFF)+0x00220000;        //check
  GPIO_PORTC_AMSEL_R &= ~0x30;   // disable analog on PC5-4


            NVIC_PRI1_R |= 0x00200000;   // priority 1 -- set bits 21-23
            NVIC_EN0_R |= 0x40;    // enable interrupt 6 in NVIC
}


// input ASCII character from UART
// spin if RxFifo is empty
// Lab 9
char UART_InChar(void){
  while((UART1_FR_R & 0x10) != 0){};
            return UART1_DR_R & 0xFF;
}
// Lab 9
bool UART_InStatus(void){ return(UART1_FR_R & 0x10);}


// output ASCII character to UART
// busy-wait spin if hardware not ready
// Lab 9
// in Lab 9 this will never wait
void UART_OutChar(char data){
```

```
  while((UART1_FR_R & 0x20)) {};
        UART1_DR_R = data;
}
// one thing has happened:
// hardware RX FIFO goes from 7 to 8 items
// Lab 9
void UART1_Handler(void){
  PF1  ^= 0x02; // triple toggle debugging
  PF1  ^= 0x02;

        while((UART1_FR_R & 0x10) == 0){
                RxFifo.Put(UART_InChar());
        }
        RxCounter++;
        UART1_ICR_R |= 0x10;   // this clears bit 4 (RXRIS) in the RIS register
  PF1  ^= 0x02;
}


#endif

//------------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_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(uint32_t 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_InMessage------------
// Accepts ASCII characters from the serial port
//    and adds them to a string until ETX is typed
//    or until max length of the string is reached.
// Input: pointer to empty buffer of 8 characters
// Output: Null terminated string
void UART_InMessage(char *bufPt){
// write this if you want
}
```

```
// FIFO.cpp
// Runs on any microcontroller
// Student names: put your names here
// Last modification date: change this to the last modification date or look very silly
// Last Modified: 4/11/2018

#include <stdint.h>

#include "FIFO.h"
#include "ST7735.h"
#include "print.h"

// A class named Queue that defines a FIFO
Queue::Queue(){
  // Constructor - make FIFO initially empty
  this->PutI=0;this->GetI=0;this->race=0;
}

// To check whether Queue is empty or not
bool Queue::IsEmpty(void){return this->race==0;}

  // To check whether Queue is full or not
bool Queue::IsFull(void){return this->race>=31;}

  // Inserts an element in queue at rear end
bool Queue::Put(char x){
        if(this->IsFull()) return false;
        this->Buf[this->PutI]=x;PutI++; PutI%=32;
        this->race++;
```

```
  return true;
}

  // Removes an element in Queue from front end.
bool Queue::Get(char *pt){
  if(PutI==GetI){return false;}
	*pt=this->Buf[this->GetI];this->GetI++;GetI%=32;
	this->race--;
	return true;
}

  /*
    Printing the elements in queue from front to rear.
    This function is only to test the code.
    This is not a standard function for Queue implementation.
  */
void Queue::Print(void){for(int i=GetI;i<PutI;i++)ST7735_OutChar(this->Buf[i]);}
```