Goal:

In this project you will create an OS-driven multitasking device than can capture data at precise intervals, buffer the data to EEPROM, and send data over a serial interface to a computer, while playing delightful music and riveting games.

Description:

Overview

In this project you will create an OS-driven multitasking device than can capture data at precise intervals, buffer the data to eeprom, send data over a serial interface to a computer, while playing music and games. The device will boot into a menu 5 commands are available. This menu serves as the command prompt on a traditional linux system. These commands initiate and suspend tasks. You'll need to perform the following steps to complete the project

- Create/add tasks to run provided music and game functions in the OS skeleton code
- Setup ADC and Timer0 to implement precision capture intervals (you already did this in discussion session)
- Implement a circular buffer in EEPROM (may do this in RAM for lesser credit)
- Create Task to poll and store ADC data into the circular buffer
- Create Task to dequeue data from the circular buffer and send data through serial port at a sufficient rate to avoid overflow
- Implement a menu system (MAIN MENU) using the LCD and Joystick with these 5 commands:
    - MUSC - Resumes Music task (stay at prompt, this task runs forever)
    - GAME - Resumes Game task (may need to use a mutex to halt I/O processing by the main prompt while game is running and interfacing with the user)
    - CAPT- Resumes Capture task (stay at prompt. task suspends itself if circular buffer fills)
    - STSD - Resumes Send task (stay at prompt, task runs until suspended)
    - SPSD - Suspends Send task (stay at prompt, suspends send task)
- Implement Task Status Indicator on LEDs (optional, for debugging)

Circular Buffer

When you Implement the circular buffer, use variables to keep track of the write position, read position, and count (number of elements stored). The buffer should store 16bit values. Use a count variable to implement the IsEmpty and is IsFull funcions. The buffer should use all of the EEPROM, but it should only allow count to reach 100. Cycling through the entire EEPROM this way extends its lifetime.

Use a semaphore to avoid the enqueue and dequeue functions from accessing and modifying count variable at the same time.

APPROACH: I suggest first implementing the circular buffer without using EEPROM. Write Enqueue, Dequeue, IsEmpty and IsFull so that the physcial storage medium is abstracted.

Try to store at least 30 points. Assuming you have a good abstraction, you may switch to the EEPROM with minimal code change. Full credit requires use of the EEPROM though.

EEPROM USAGE:

You should use AvrX-provided EEPROM commands:

```
//offset should be an integer between 0 and (size of eeprom in bytes - 1)
AvrXWriteEEProm((void*)(offset)  ,unsigned char ch);
c= AvrXReadEEProm((void*)(offset));
```

SPECIAL EEPROM-PROTECTION NOTE FOR ENQUEUE: To protect the EEPROM from excessive writes because of buggy programs, use a static variable in the Enqueue function to track the number of calls and prevent any actual action after 10000 calls are made.

Task Details:

Main Menu TASK

The main menu command driven task control menu should be implemented using a AVRX tasks and should control other AVRX tasks

Implement a menu system (MAIN MENU) using the LCD and Joystick with these 5 commands:

- MUSC - Resumes Music task (stay at prompt, this task runs forever)
- GAME - Resumes Game task (may need to use a mutex to halt I/O processing by the main prompt while game is running and interfacing with the user)
- CAPT- Resumes Capture task (stay at prompt. task suspends itself if circular buffer fills)
- STSD - Resumes Send task (stay at prompt, task runs until suspended)
- SPSD - Suspends Send task (stay at prompt, suspends send task)

Playing game (slot machine) TASK

Code is provide for the game. You will need to make minimal but masterfully selected changes. When user selects the game command from main menu, we enter the game. User can keep playing game infinitely. While in game, only the center button is being used to play the game. If user decides to go back to the MAIN MENU, he/she will press up on joystick. Going back to main menu from game should save the state of game and resume it if user decides to continue playing game later. To explain it more:

The user enters into game and starts playing the game. After pressing the center button twice (selecting 2 first numbers on slot machine e.g. 8 and 2) he decides to go back to menu. When this happens you have to save the state of the game in a way that when player comes back to the game later on, he/she will continue playing the game as if he/she never left the game. (in our example numbers 8 and 2 will be shown on LCD and he will try his chance for 3rd number).

Capture TASK

If user selects the capture command, the task for capturing the temperature of the surrounding will start and it will continue to execute as long as there is a free room in the circular buffer.In other words, capture task will suspend itself if there is no room in the buffer, and will continue to run forever if it always finds a free room in the buffer. EEPROM is available in ATmega169P. check datasheet for more info.

ADC and Timer0: (you did this in discussion session):

- Setup ADC to Trigger off of Timer0, with a frequency around 30-32 Hz. You need to figure out how to trigger the ADC off of Timer0 and configure Timer0 accordingly. Set the autoconversion enable bit and set ADCSRB appropriately to select the timer signal source (check datasheet).
- You can trigger the ADC off of Timer0 overflow or output compare match. IMPORTANT NOTE: I didn't catch it in the datasheet, but I had to enable the overflow interrupt in Timer0 in order to get the ADC to actually trigger off of the overflow event, and so I provided an empty ISR for that interrupt (though sometimes I used this for debugging). When I instead used the compare-match event, I had to enable that interrupt and I provided an empty interrupt handler for that.
-
- Set the prescalar value, ADPS, to make sure the ADC clock is below the specified 200kHz to achieve full accuracy.
-
- Set the reference to the internal 1.1V using the REFS bits.
-

Send Data TASK

Starting/Resuming Send Data: Enabled by sending STSD. This will start a task which will read the captured data from the buffer (implemented in EEPROM) and will send it via serial port to the PC. On the PC side an application will listen to the port and will capture the data. We will use Matlab, but you can do most testing and developement using the terminal programs

The Sending-Data task will not halt/terminate even if it finds the buffer empty (underflow). It should wait and send new data in the buffer to the PC as it becomes available.

Suspending Send Data: This command will suspend the send-data task if it is running. The side effect of this action will be a full buffer if the Capture Task is still running. A full buffer will cause the Capture Task to be suspended.

Music TASK

Use provide code to play music

Reading EEPROM Data using AVR Studio

Using AVR Programing Tool, you can read the data in EEPROM. This will be helpful especially when you are debugging your code.

This example writes alphabet in EEPROM and reads them back. AVRX_EPROM.c This will be your start point to work with EEPROM. After running the code on Butterfly, if you read the data out of EEPROMyou will see something as follows:

```
:10000000414243444546474849 4A4B4C4D4E4F5068

:10001000515253545556575859 5AFFFFFFFFFFFF8F

:10002000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE0
```

The first 8 digits are address digits. Each line is showing 16 bytes of EEPROM and since EEPROM on ATmega169P is 512byte we have 32 lines

First line has the address of :10000000. If you ignore ":1", it is number zero. Second line by ignoring ":1"is number 16. As you see first line includes bytes for address 0 to 15.

Also you know that each byte is 2 hex numbers. So 41 from first line is the 'A' that the program has written in address zero. The rest should be clear to you no, but ask for assistance as required.

Provided Code

Matlab Code

Example To Open Serial Port and Send Text From Matlab (you don't need to send text, but I thought this may be useful)

```
%% setup serial
 %setup com2 19200 buad, littleEndian, 20kbyes software buffer, 10 sec wait
 %sec wait
 c=serial('com2','Baud',19200,'ByteOrder','littleEndian',...
          'InputBufferSize',20*1024,'Timeout',10);
 fopen(c);
 fprintf(c,'text');
 fclose(c);
```

Example To Recieve and Plot Data

```
%% setup serial
 %setup com2 19200 buad, littleEndian, 20kbyes software buffer, 10 sec wait
 %sec wait
 c=serial('com2','Baud',19200,'ByteOrder','littleEndian',...
          'InputBufferSize',20*1024,'Timeout',10);
 fopen(c);
 %% clear buffer
 fread(c,c.BytesAvailable);
 %% grab up to 1000 uint16 values
 data=fread(c,1000,'uint16')
 %% plot values
 figure;plot(data,'-o');
 fclose(c);
```

How to Cleanup open ports

```
%% find any open (serial) objects and close them
 tobj=instrfind;fclose(tobj);
```

.

Music Code

```
//required software timers
 TimerControlBlock timerNote,timerMusic;

 AVRX_GCC_TASKDEF(taskMusic, 8, 3){
   MusicSetupPort();
   MusicSetupTimer1();

   while(1){
     MusicPlayB();
     MusicPlayA();
     MusicPlayG();
     AvrXDelay(&timerMusic,500);
   }
 }


 //
*****************************************************************************
*
 //  MUSIC FUNCTIONS USING TIMER 1
 //
*****************************************************************************
*

 void MusicSetupPort(){
 #define PORTB5_MASK  0b00100000
     DDRB |= PORTB5_MASK; //enable PORTB5 as output
 }

 void MusicSetNote(int period_us){
     ICR1=period_us/2;
     OCR1A=period_us/4;
     TCNT1=0;
 }

 void MusicRest(){
     OCR1A=0;
 };

 void MusicPlayG(){
    MusicSetNote(2552);
    AvrXDelay(&timerNote, 500);
    MusicRest();
 }

 void MusicPlayA(){
    MusicSetNote(2272);
    AvrXDelay(&timerNote, 500);
    MusicRest();
 }

 void MusicPlayB(){
    MusicSetNote(2024);
    AvrXDelay(&timerNote, 500);
    MusicRest();
 }

 void MusicSetupTimer1() {
     /* Timer/Counter1 Control Register A */
     int value_TCCR1A_COM1A_2b = 2; //clear on match up, set on match down
     int value_TCCR1A_COM1B_2b = 0; //disable
     //this one is spread over both regsters
     int value_TCCR1AB_WGM1_4b = 8; //PWM+frequency control using register
ICR1
     /* Timer/Counter1 Control Register B */
     int value_TCCR1B_ICNC1_1b = 0; //no input capture noise cancel
     int value_TCCR1B_ICES1_1b = 0; //Dont Care since not used
     int value_TCCR1B_CS1_3b = 2; //use clock prescaler 8...see page 131 of
datasheet...1Mhz for 8Mhz clk
     TCCR1A = (value_TCCR1A_COM1A_2b << COM1A0) + (value_TCCR1A_COM1B_2b <<
COM1B0) + ((value_TCCR1AB_WGM1_4b&0x3) << WGM10);
     TCCR1B = (value_TCCR1B_ICNC1_1b << ICNC1) + (value_TCCR1B_ICES1_1b <<
ICES1) + (((value_TCCR1AB_WGM1_4b&0b00001100)>>2) << WGM12) +
(value_TCCR1B_CS1_3b<<CS10);
     /* Timer/Counter1 Input Capture Register */
     //also used for frequency setting
     ICR1 = 0xFF/2; //16 bits //just some initial value
     /* Timer/Counter 1 Interrupt Mask Register */
     int value_TIMSK1_ICIE1_1b = 0;//input capture int. enable
     int value_TIMSK1_OCIE1B_1b = 0;//output compare int. enable
     int value_TIMSK1_OCIE1A_1b = 0;//output compare int. enable
     int value_TIMSK1_TOIE1_1b = 0;//overflow int. enable
     TIMSK1 = (value_TIMSK1_ICIE1_1b <<ICIE1) +
(value_TIMSK1_OCIE1B_1b<<OCIE1B) + (value_TIMSK1_OCIE1A_1b<<OCIE1A) +
(value_TIMSK1_TOIE1_1b<<TOIE1);
 }
```

Game and button code

```
//global variables
 char  g_buttonPressedFlag;
 char g_buttonPressed; //up=4,down=3,left=2,right=1,center=0
 #define BUTTON_CODE_UP 4
 #define BUTTON_CODE_DOWN 3
 #define BUTTON_CODE_CENTER 0

 AVRX_SIGINT(PCINT1_vect) {   //remember this is called on pin change 0->1
and 1->0
   IntProlog(); //interrupt prolog
   //    static uint8_t pinBPrev=1; //trace

   static unsigned char pressedPrev;
   unsigned char pressed = (~PINB & (_BV(7)|_BV(6)|_BV(4))); //leaves a one
on 7,6,4 if they are low because button is pressed

   //capture newly pressed buttons only
   if(pressed &  ~pressedPrev) { //if button pressed and previously it wasn't

     if (g_buttonPressedFlag==0){
       g_buttonPressedFlag=1;
       if (pressed & _BV(7)){
        g_buttonPressed=BUTTON_CODE_DOWN;
       } else if (pressed & _BV(6)){
        g_buttonPressed=BUTTON_CODE_UP;
       }else{// if (pressed & _BV(4)){
        g_buttonPressed=BUTTON_CODE_CENTER;
       }
       PORTB ^= (1<<0); //toggle led on B0
     }

   } else {
     //nothing to do
   }

   pressedPrev = pressed; //save pressed button status

   Epilog();

 }

 //Setting the intrupt handler
 void SetupButtonInterrupts(void){
   PCMSK1  |= (1<<PCINT15)+(1<<PCINT14)+(1<<PCINT12); //Unmask bit for Center
Button and LR on Butterfly, PB4->PCINT12 to allow it to trigger interrupt
flag PCIF1
   EIMSK   = (1<<PCIE1);    //Enable interupt for flag PCIF1
 }


 int HasButtonPressed(){
   return g_buttonPressedFlag;
 }

 int GetButtonPressed(){
   int return_val;
   if (g_buttonPressedFlag==1){
     //save button THEN release hold
     return_val = g_buttonPressed;
     g_buttonPressedFlag=0;
     return return_val;
   }else{
     return -1;
   }

 }

 //setup pins for input
 void SetupJoystickPins(void){
   //set as input and enable pull up resistor for pins 4,6,7 to use the
pushbuttons
   DDRB  &= 0b00101111; //set three pindirections
   PORTB |= 0b11010000; //set three pinpullups
 }

 //required timer
 TimerControlBlock timerGame;

 void MyGame(){
     //static volatile unsigned long ulDummyVariable = 3UL;

     static volatile unsigned char display[3] = {'0','0','0'};

     const int xDelay = 500;

     static enum state_t {open00,open0,open1,roll0,roll123,roll4,win,loss}
state=open00;

     static unsigned char stopCount; //number of "wheels" stoped

     LCD_Init();

     for( ;; )    {
        AvrXDelay(&timerGame, 2); //AvrXYield(); should have worked but did
not compile
        switch (state) {
        case open00: //just an entry point
            state=open0;
            break;
        case open0:  //display play prompt
            LCD_WriteDigit('P',3);
            LCD_WriteDigit('L',4);
            LCD_WriteDigit('Y',5);
            state=open1;
            break;

        case open1: //wait for button press
                     // up causes exit to suspend,
                    // center causes game to start
            if (HasButtonPressed()){
                switch (GetButtonPressed()){
                case BUTTON_CODE_CENTER:
                    state = roll0;
                    break;
                case BUTTON_CODE_UP: //on up suspend and resume in same place
                    return;
                    state=open00;
                    break;
                }
            }else{
                // do nothing if no button pressed
            }
            break;

        case roll0: //roll substate to init stop count
            stopCount = 0;
            state = roll123;
            break;

        case roll123: //roll state, takes three center presses to complete
game
            if (HasButtonPressed()){
                GetButtonPressed();
                stopCount+=1;
                if (stopCount == 3){
                    state = roll4;
                }
            }


            //generate and display roll of 3 random numbers
            if (stopCount < 3){
                display[0]='0' + rand()%5;
                LCD_WriteDigit(display[0],5);
                if (stopCount<2){
                    display[1]='0' + rand()%5;
                    LCD_WriteDigit(display[1],4);
                    if (stopCount<1){
                        display[2]='0' + rand()%5;
                        LCD_WriteDigit(display[2],3);
                    }
                }
            }

            AvrXDelay(&timerGame, 1000); //sets roll speed
            break;


        case roll4: //roll subsate t decide win or loss
            if (display[0]==display[1]&&display[0]==display[2]){
                state = win;
            }else{
                state = loss;
            }
            break;

        case win: // display win and restart
            AvrXDelay(&timerGame, xDelay); LCD_WriteDigit('W',3);
            AvrXDelay(&timerGame, xDelay); LCD_WriteDigit('I',4);
            AvrXDelay(&timerGame, xDelay); LCD_WriteDigit('N',5);
            AvrXDelay(&timerGame, xDelay); LCD_WriteDigit(display[0],3);
            AvrXDelay(&timerGame, xDelay); LCD_WriteDigit(display[1],4);
            AvrXDelay(&timerGame, xDelay); LCD_WriteDigit(display[2],5);
            AvrXDelay(&timerGame, xDelay);
            LCD_WriteDigit(';',3);  //clear
            LCD_WriteDigit('+',4);
            LCD_WriteDigit('5',5);
            //TODO: should show user cash
            state=open0;
            break;

        case loss: //display loss and restarts
            AvrXDelay(&timerGame, xDelay);       LCD_WriteDigit('X',3);
            AvrXDelay(&timerGame, xDelay);        LCD_WriteDigit('X',4);
            AvrXDelay(&timerGame, xDelay);        LCD_WriteDigit('X',5);
            AvrXDelay(&timerGame, xDelay);
LCD_WriteDigit(display[0],3);
            AvrXDelay(&timerGame, xDelay);
LCD_WriteDigit(display[1],4);
            AvrXDelay(&timerGame, xDelay);
LCD_WriteDigit(display[2],5);
            AvrXDelay(&timerGame, xDelay);
            LCD_WriteDigit(';',3);  //clear
            LCD_WriteDigit('+',4);
            LCD_WriteDigit('0',5);
            state=open0;
            break;
        }
     }
 }
```