**Final Project Report**

**Unreal Virtual Glove**

**Authors: Sal Cardone and Steve Marshall**

**Course: CET 498**

**Date: December 18, 2015**

**Instructor: Dr. David Broderick**

TABLE OF CONTENTS

1 INTRODUCTION 3

1.1 Abstract 3

2 PROJECT Specification 3

2.1 Project Summary 3

2.2 Specifications 3

2.3 Requirements 4

2.4 Constraints 4

2.5 Methodology 4

3 PARTS list and cost 5

3.1 Parts Table 5

4 IMPLEMENTATION 5

4.1 Experimental Evaluation 5

4.2 Results and Discussion 6

5 CONCLUSION 7

5.1 Overall Summary 7

5.2 Current Limitations 7

5.3 Future Work 7

6 REFERENCES #

7 APPENDIX user guide #

7.1 System Configuration #

7.2 Project Code #

**1 INTRODUCTION**

**1.1 Abstract**

An increasing interest in virtual reality has spurred many advances in technology recently. Virtual technology would also allow for a safe environment to practice lifesaving skills. A surgeon could practice intense surgeries while Firefighters may practice in the virtual environment without the risk of getting hurt by collapsing buildings. The possible uses of virtual technology range from professional training, like a flight simulation for pilots, to entertainment such as games. While the technology has not made its way into many industries, the gaming industry has become a large testing field as gamers are always looking for the next great thing to improve their experience. The Unreal Engine 4 is a game engine with compatibility for the virtual headset, Oculus Rift. We will be creating a system that connects this engine and a Leap Motion Controller to allow a player to interact with the virtual environment using their hands. We have created a glove which utilizes feedback as the user interacts with objects in a virtual environment. This will allow the player to feel the objects and become more immersed in the world as they play.

**2 PROJECT SPECIFICATION**

**2.1 PROJECT SUMMARY**

The objective of the Unreal Leap project is to provide gamers with an experience unlike any other. This project will utilize the Leap Motion controller with the Unreal Engine 4, and will improve the experience with the inclusion of a touch-sensitive glove. Unreal Engine will be programmed to create data points at each finger's location of the player’s hand. These data points will be transferred as input signals to an MSP430 microcontroller when making contact with an object, which will produce outputs to 5 buzzers (one for each finger) attached to the glove. When the player interacts with an object in the virtual environment, they will feel the buzz on the fingers that are touching that object, creating a sense of connection to the environment and by adjusting the intensity of the buzzing sensation a user will become capable of feeling the movement of the object.

**2.2 SPECIFICATIONS**

The Leap Motion Controller has an infrared sensor that has a range between 1 inch to 2 feet. There is a coordinate system involved, where the y axis extends upwards from the device, the x axis runs parallel to the long edge and the z axis runs parallel to the short edge. The z axis has positive values in the direction of the user (away from the green light).

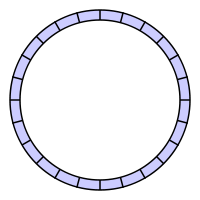
**2.3 REQUIREMENTS**

The main requirements for this project were the installations of Unreal Engine and IAR Embedded Workbench, the purchase of a Leap Motion Controller, an MSP430 and an HC-06 Bluetooth chip, as well as extensive knowledge of assembly language and C++ coding. Coding and testing began immediately following the collection of all necessary requirements.

**2.4 CONSTRAINTS**

As the name implies, the Bluetooth chip requires a Bluetooth connection in order to be able to interact with it. The laptop being used for Unreal Engine did not contain a working Bluetooth adapter, so an external dongle was purchased. The range for Bluetooth is about 33ft, so the user must be within this range in order to see results and the functionality of the equipment. The user must also be within 1 inch to 2 feet of the infrared sensor on the Leap Motion Controller.

**2.5 METHODOLOGY**

The first step is to launch the Unreal Editor software, which is the home of the virtual environment. When the “Play in Editor” button is clicked, the game officially begins. We created a trigger in the shape of a box, which is attached to a cylinder. When this box spawns at the start of the level, code is executed in which the serial com port 10 opens up, and connects to the HC-06 module. Once this connection has been established, data can be sent from the game. A character was created in which the player can control the body movements with the arrow or WASD keys, and the arms using the Leap Controller. The player will begin the game with the virtual glove on their right hand. When the player touches the correct cylinder using their own hand, a packet of bytes is sent through the com port to the HC-06, which is then transferred to the MSP430. The MSP reads this data and sends it to the buzzers that are located at each fingertip. This activates these buzzers and the player can now “feel” the virtual object. When the player moves away from the cylinder, a different set of bytes is sent, which tells the buzzers to deactivate.

To control the buzzers the MSP430 will require a buffer to hold the data sent from the Computer so that it may analyze the data and ensure only valid packets are utilized by the program. For this application a Ring buffer shall be used. When reading from the Rx line, a sub routine will place the latest byte into the tail of the buffer and advance the pointer. The Head will advance as later routines look through the bytes and place 10 bytes into the message buffer after it finds the starting byte allowing the Ring buffer to free up space for more packets of Data.

<https://en.wikipedia.org/wiki/Circular_buffer>

**3 PARTS LIST AND COST**

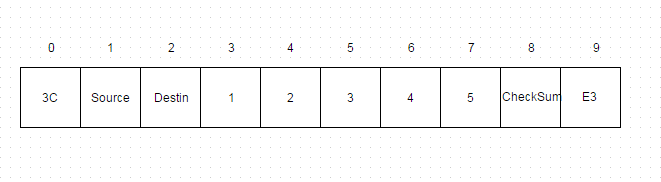
**3.1 PARTS TABLE**

|  |  |  |
| --- | --- | --- |
| Part | Vendor | Price |
| MSP430 | Texas Instruments | $10 |
| Jumper Cables | Amazon | $10 |
| Glove | Target | $5 |
| Phone Wire | Home Depot | $5 |
| Leap Motion | LEAP | $80 |
| Bluetooth adaptor | Amazon | $10 |
| Bluetooth Chip (HC-06) | Amazon | $10 |
| **Total:** |  | **$130** |

**4 IMPLEMENTATION**

**4.1 EXPERIMENTAL EVALUATION**

This project required very extensive testing and debugging as there was a lot of code written, and many connections that were unstable. While testing the data transfer through the com port, data was either not being sent, or not being sent correctly. Bits tend to get lost when they travel, so many changes were made to the MSP code in order to achieve higher accuracy of incoming bits. To improve performance and stability, the device had certain criteria it would check for in data packets before moving the data to a spot readable by the program. It would first read through the bytes stored in the Ring Buffer until a starting bit ( 3C ) was found. The next 9 bits will be added to a message buffer. The program then checks the Destination Address against a constant set within the code. The message is then passed so bits 1-7 of the message can be byte wise summed and checked against the checksum in bit 8 as the final check to measure data consistency.



After the checks the data is at last moved to the “received\_data” array where the PWM will immediately begin outputting the new state.

PWM Outputs are created when the line is given a time frame, in our case 200 ticks of Timer\_A. Starting at tick 0 the line will not be on. When the timer’s tick count exceeds the Value stored in Recieved\_data, the Line will turn on and remain on until the time frame is reset. This creates a waveform where Received Data controls the pulse width. The Code:

|  |
| --- |
| off             mov.w  #0, R15       ;reset timer             mov.B  #255, &P1OUT   ;default off  main:             cmp.w #200, R15             JC off  time    ;Thumb             mov.b #0, R4             cmp.w Recived\_data(R4), R15             ; turn on if Timer passed             JNC time2             BIC.B   #BIT7,&P1OUT            ; Clear(On) P1.7 |

**4.2 RESULTS AND DISCUSSION**

Overall the final outcome of this project turned out almost as expected. We are able to communicate from the Leap Motion Controller to the virtual glove through the Unreal Engine, the Bluetooth chip, and the MSP430. The most challenging part of this was getting all of the equipment to communicate with one another. C++ coding in Unreal Engine is comprised of many new and different functions, so a new language had to be learned. This coding was used to create the trigger box that was attached to the cylinder, and to write a set packet of bytes to the serial port when the player interacts with the cylinder. At the beginning, we decided that we were going to make the game as realistic as possible by adjusting the frequency of the buzzers based on how hard the player is touching or grabbing an object. Unfortunately we were unable to accomplish this task as the steps leading up to that point were much more challenging and took up more time than originally expected. Another issue that we had run into was the fact that the Jenga-style game that was planned was not able to be created as we had wanted. The trigger box that was created calls the opening of the com port, and if there are many boxes all attached to the many cylinders, this call gets interrupted, resulting in data loss or packets not being sent at all. There was no work around for this as far as we were concerned, so the one cylinder with our box, we believe, is enough to show our proof of concept.

**5 CONCLUSION**

**5.1 OVERALL SUMMARY**

The Prototype is able to receive data streamed from the computer and translate it into a sensory response using the buzzers on each finger and a PWM output modulated by the received data. The Unreal Game will turn the glove on when the characters hand makes contact with the test cylinder, and off when the player moves their hand away.

**5.2 CURRENT LIMITATIONS**

One limitation that we are currently facing is the fact that when the player interacts with the object, all of the buzzers, except for the thumb which is not working, activate instead of just the ones on the fingers that are “touching” the object.  Furthermore, the player is unable to feel like they are pressing harder on the object because we could not figure out how to grab the individual strength of each finger being pressed against the object. The player also is unable to play a virtual jenga game with tactical feedback due to the limitation on the number of trigger boxes that the engine can accept. However, there is a game of jenga implemented in the environment that the player can still interact with if he or she chooses to.

**5.3 FUTURE WORK**

Due to all of the current limitations, our game is not in the shape that we want it to be in. There is only one object that can be interacted with as well as the Unreal Engine does not send variable data based on how much contact is made.  In the following months, changes to the game itself to send more varied data for more objects will create the desired environment to show off the capabilities of our prototype.

**6 REFERENCES**

1<https://developer.leapmotion.com/documentation/csharp/devguide/Leap_Overview.html>

2<https://www.youtube.com/watch?v=I39nohn-Fq0>

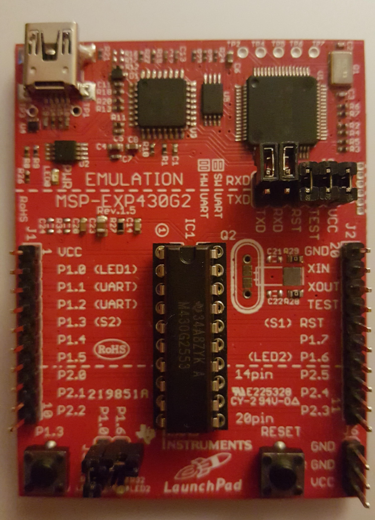
3 C++ in Unreal Engine Tutorials

4 IAR Embedded Workbench guides

**7 APPENDIX USER GUIDE**

**7.1 SYSTEM CONFIGURATION**

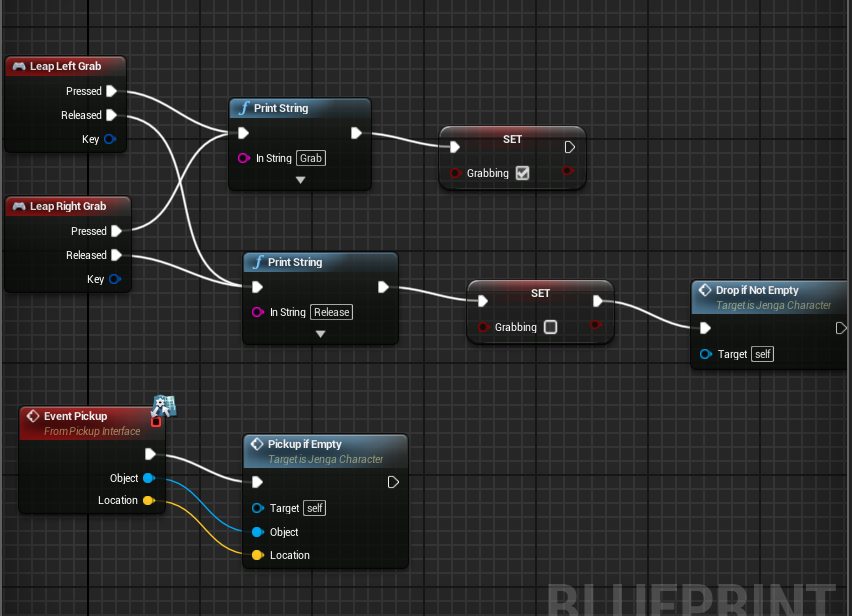
|  |  |
| --- | --- |
| **P1.1** | **Rx** |
| **P1.2** | **Tx** |
| **P1.3** | **Thumb** |
| **P1.4** | **Index finger** |
| **P1.5** | **Middle finger** |
| **P1.6** | **Ring Finger** |
| **P1.7** | **Pinky** |



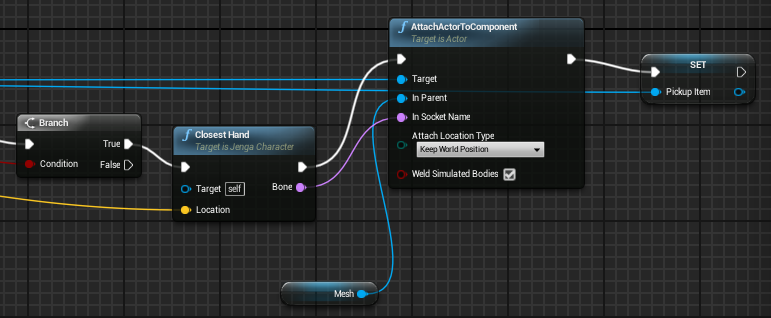
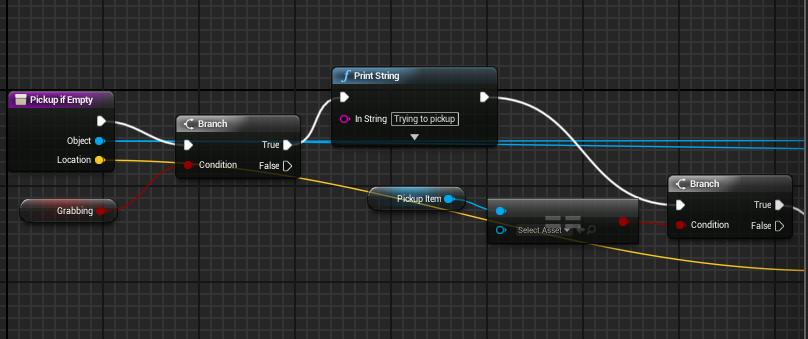
**7.2 PROJECT CODE**

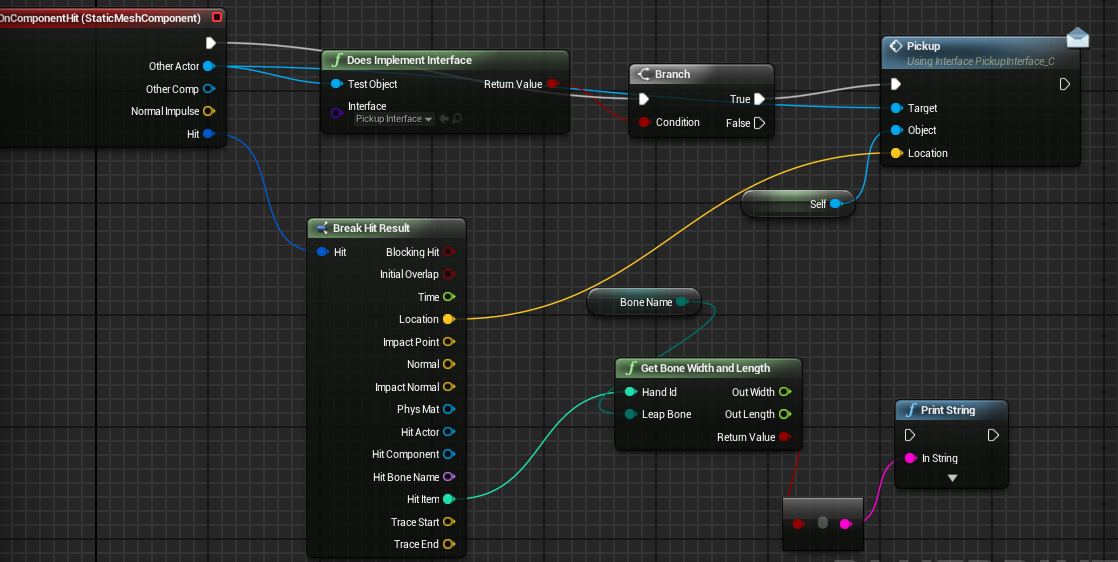
**Blueprints:**

Main Character:



Pickup if Empty Function:



Movable Object:

Creating the trigger box and sending data to the HC-06 on contact

|  |
| --- |
| **MyBox.h**  #pragma once  #include "GameFramework/Actor.h"  #include "MyBox.generated.h"  UCLASS()  class JENGA\_API AMyBox : public AActor  {  GENERATED\_BODY()    public:  AMyBox();  virtual void BeginPlay() override;  virtual void Tick(float DeltaSeconds) override;  UFUNCTION()  void TriggerEnter(class AActor\* OtherActor, class UPrimitiveComponent\* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const FHitResult & SweepResult);  UFUNCTION()  void TriggerExit(class AActor\* OtherActor, class UPrimitiveComponent\* OtherComp, int32 OtherBodyIndex);  }; |

|  |
| --- |
| **MyBox.cpp**  #include "Jenga.h"  #include "MyBox.h"  #include "Header1.h"  #include <windows.h>  #include <stdio.h>  #include <tchar.h>  //#include <winbase.h>  #include "AllowWindowsPlatformTypes.h"  #include "HideWindowsPlatformTypes.h"  HANDLE hSerial;  DCB dcbSerialParams = { 0 };  COMMTIMEOUTS timeouts = { 0 };  AMyBox::AMyBox()  {  UShapeComponent\* Box = CreateDefaultSubobject<UBoxComponent>(TEXT("Box"));  PrimaryActorTick.bCanEverTick = true;  FVector TriggerSize = FVector(2.0f, 2.0f, 2.0f);  Box->bGenerateOverlapEvents = true;  Box->SetRelativeScale3D(TriggerSize);  RootComponent = Box;  Box->OnComponentBeginOverlap.AddDynamic(this, &AMyBox::TriggerEnter);  Box->OnComponentEndOverlap.AddDynamic(this, &AMyBox::TriggerExit);  }  void AMyBox::BeginPlay()  {  Super::BeginPlay();  hSerial = CreateFile(\_T("\\\\.\\COM10"), GENERIC\_READ | GENERIC\_WRITE, 0, 0, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, 0);    dcbSerialParams.DCBlength = sizeof(dcbSerialParams);    dcbSerialParams.BaudRate = CBR\_9600;  dcbSerialParams.ByteSize = 8;  dcbSerialParams.StopBits = ONESTOPBIT;  dcbSerialParams.Parity = NOPARITY;  // Set COM port timeout settings  timeouts.ReadIntervalTimeout = 50;  timeouts.ReadTotalTimeoutConstant = 50;  timeouts.ReadTotalTimeoutMultiplier = 10;  timeouts.WriteTotalTimeoutConstant = 50;  timeouts.WriteTotalTimeoutMultiplier = 10;  }  void AMyBox::Tick(float DeltaTime)  {  Super::Tick(DeltaTime);  }  void AMyBox::TriggerEnter(class AActor\* OtherActor, class UPrimitiveComponent\* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const FHitResult & SweepResult)  {  UE\_LOG(LogTemp, Log, TEXT("TriggerEnter"));  char bytes\_to\_send[10];  bytes\_to\_send[0] = 60;  bytes\_to\_send[1] = 0;  bytes\_to\_send[2] = 30;  bytes\_to\_send[3] = 83;  bytes\_to\_send[4] = 65;  bytes\_to\_send[5] = 76;  bytes\_to\_send[6] = 70;  bytes\_to\_send[7] = 85;  int temp = bytes\_to\_send[1] + bytes\_to\_send[2] + bytes\_to\_send[3] + bytes\_to\_send[4] + bytes\_to\_send[5] + bytes\_to\_send[6] + bytes\_to\_send[7];  bytes\_to\_send[8] = temp % 256;  bytes\_to\_send[9] = 62;    //DWORD bytes\_written, total\_bytes\_written = 0;  for (int i = 0; i < 3; i++)  {  WriteFile(hSerial, bytes\_to\_send, 10, NULL, NULL);  }    }  void AMyBox::TriggerExit(class AActor\* OtherActor, class UPrimitiveComponent\* OtherComp, int32 OtherBodyIndex)  {  UE\_LOG(LogTemp, Log, TEXT("TriggerExit"));  unsigned char bytes\_to\_send[10];  bytes\_to\_send[0] = 60;  bytes\_to\_send[1] = 0;  bytes\_to\_send[2] = 30;  bytes\_to\_send[3] = 205;  bytes\_to\_send[4] = 204;  bytes\_to\_send[5] = 203;  bytes\_to\_send[6] = 202;  bytes\_to\_send[7] = 201;  int temp = bytes\_to\_send[1] + bytes\_to\_send[2] + bytes\_to\_send[3] + bytes\_to\_send[4] + bytes\_to\_send[5] + bytes\_to\_send[6] + bytes\_to\_send[7];  bytes\_to\_send[8] = 21;  bytes\_to\_send[9] = 62;  //DWORD bytes\_written, total\_bytes\_written = 0;  for (int i = 0; i < 3; i++)  {  WriteFile(hSerial, bytes\_to\_send, 10, NULL, NULL);  }  } |

Glove Code

|  |
| --- |
| ;requires ringbuffer.asm  #include "msp430.h"                     ; #define controlled include file         NAME    main                    ; module name         EXTERN  ringbuffer\_flush        ; make the label visible outside this file         EXTERN  ringbuffer\_isempty      ; make the label visible outside this file         EXTERN  ringbuffer\_isfull       ; make the label visible outside this file         EXTERN  ringbuffer\_dequeue      ; make the label visible outside this file         EXTERN  ringbuffer\_enqueue      ; make the label visible outside this file         PUBLIC  main                    ; make the main label visible outside this module         ORG     0x200         tx\_rb:  DS8 34  // Head, Tail, 32bytes of buffer         space:  DS8 14         rx\_rb:  DS8 34  // Head, Tail, 32bytes of buffer           txmessage: DS8 10         rxmessage: DS8 10         recieved\_data: DS8 5         send\_data: DS8 5         destinationadd: DS8 1         arithmeticsum: DS8 1         msgnum: DS8 1         bad\_recieve: DS8 1         my\_address: DS8 1         b\_state:    DS8 1             ORG     0FFFEh         DC16    init                    ; set reset vector to 'init' label         ORG     0FFEEh                  ; USCI RX Vector         DW      USCI\_RX\_ISR         ORG     0FFECh                  ; USCI TX Vector         DW      USCI\_TX\_ISR         ORG     0FFF2h                  ; Timer\_A0 Vector         DW      TA0\_ISR                 ;         RSEG    CSTACK                  ; pre-declaration of segment         RSEG    CODE                    ; place program in 'CODE' segment  init:  mov   #SFE(CSTACK), SP        ; set up stack         mov.b   &CALBC1\_1MHZ, &BCSCTL1  ; Load calibrated DCO 1MHZ pt 1/2         mov.b   &CALDCO\_1MHZ, &DCOCTL   ; Load calibrated DCO 1MHZ pt 2/2           BIS.B   #250, &P1DIR    StopWDT  mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; Disable watchdog timer  SetupC0  mov.w   #CCIE,&CCTL0            ;         mov.w   #100,&CCR0            ;  SetupTA  mov.w   #TASSEL\_2+MC\_1,&TACTL   ;  SetupP1         bis.b   #BIT3, &P1REN           ; Enable pullup resistors for button         bis.b   #BIT0, &P1DIR           ; Set P1.0 as output         bis.b   #BIT1 + BIT2, &P1SEL    ; P1.1 = RXD, P1.2=TXD         bis.b   #BIT1 + BIT2, &P1SEL2   ; P1.1 = RXD, P1.2=TXD  SetupUSCI\_A0         bis.b   #UCSSEL\_2, &UCA0CTL1    ; SMCLK as input clock         mov.b   #104, &UCA0BR0          ; Config baud rate divider 1MHz 9600         mov.b   #0, &UCA0BR1            ; 1MHz 9600         mov.b   #UCBRS0, &UCA0MCTL      ; Modulation UCBRSx = 1         bic.b   #UCSWRST, &UCA0CTL1     ; \*\*Initialize USCI state machine\*\*         bis.b   #UCA0RXIE, &IE2         ; Enable USCI\_A0 RX interrupt         mov.w   #tx\_rb, R12             ; MOV the address of the ring buffer into R12         call    #ringbuffer\_flush         mov.w   #rx\_rb, R12             ; MOV the address of the ring buffer into R12         call    #ringbuffer\_flush         bis.w   #GIE,SR                 ; Enable global interrupts           ;initialize variables here                 mov.w #0, rxmessage  mov.w #0, arithmeticsum  mov.w #0, msgnum  mov.w #30, destinationadd                 mov.w   #0,     R7    varinit:    ;Default Data for test  mov.w   #0,R15                 mov.b   #114, rxmessage(R15)  mov.b #83,  send\_data(R15)                 mov.b   #116, txmessage(R15)                 mov.b #235,  recieved\_data(R15)  inc.w   R15                 mov.b   #120, rxmessage(R15)  mov.b #84,  send\_data(R15)                 mov.b   #120, txmessage(R15)                 mov.b #235,  recieved\_data(R15)  inc.w   R15                 mov.b   #109, rxmessage(R15)  mov.b #69, send\_data(R15)                 mov.b   #109, txmessage(R15)                 mov.b #235,  recieved\_data(R15)  inc.w   R15                 mov.b   #101, rxmessage(R15)  mov.b #86, send\_data(R15)                 mov.b   #101, txmessage(R15)                 mov.b #235,  recieved\_data(R15)                 inc.w   R15                 mov.b   #83, rxmessage(R15)  mov.b #69, send\_data(R15)                 mov.b   #83, txmessage(R15)                 mov.b #235,  recieved\_data(R15)            off:             mov.b  #0, R7       ;reset timer             mov.B  #0, &P1OUT   ;default off  main:             cmp.w #200, R7             JC off  time                                    ;thumbs             mov.w #0 , R4             mov.b recieved\_data(R4), R8             cmp.b recieved\_data(R4), R7             ;switch             JNC time2                   ;don't turn on             BIS.B   #BIT3,&P1OUT        ;turn on  time2:      ;index             mov.w #1 , R4             mov.b recieved\_data(R4), R8             cmp.b recieved\_data(R4), R7             ;             JNC time3             BIS.b  #BIT4, &P1OUT  time3:      ;middle             mov.w #2 , R4             mov.b recieved\_data(R4), R8  ;53             cmp.b recieved\_data(R4), R7             ;             JNC time4             BIS.b  #BIT5, &P1OUT  time4:      ;ring             mov.w #3 , R4             mov.b recieved\_data(R4), R8  ;54             cmp.b recieved\_data(R4), R7             ;             JNC time5             BIS.b  #BIT6, &P1OUT  time5:       ;pinky             mov.w #4 , R4             mov.b recieved\_data(R4), R8             cmp.b recieved\_data(R4), R7             ;             JNC endtime             BIS.b  #BIT7, &P1OUT  endtime:  statuscheck:       call #protocol\_recieve       mov.b #0,R15       cmp.b #250,recieved\_data(R15)       JNE main       inc.w R15       cmp.b #100,recieved\_data(R15)       inc.w R15       JNE main       cmp.b #50,recieved\_data(R15)       inc.w R15       JNE main       cmp.b #150,recieved\_data(R15)       inc.w R15       JNE main       cmp.b #200,recieved\_data(R15)       JNE main       JMP continue  continue:         jmp continue    protocol\_send:         XOR.B  #BIT0, &P1OUT         push R12  push R13  push R9         push R11         mov.w #tx\_rb, R12         mov.w   #0,R9  mov.w   #0,R11           mov.b   #0E3h , txmessage(R11) ;0 byte  add.b   txmessage(R11),R9  add.b #1, R11    mov.b   &msgnum, txmessage(R11) ;1 byte  add.b   txmessage(R11),R9  add.b #1, R11    mov.b   &destinationadd, txmessage(R11) ;2 byte  add.b   txmessage(R11),R9  add.b #1, R11    mov.b #0,R10         mov.b   send\_data(R10), txmessage(R11) ;4 byte  add.b   txmessage(R11),R9  add.b #1, R11    inc.b R10         mov.b   send\_data(R10), txmessage(R11) ;5 byte  Correct  add.b   txmessage(R11),R9  add.b #1, R11    inc.b R10  mov.b   send\_data(R10), txmessage(R11) ;6 byte  add.b   txmessage(R11),R9  add.b #1, R11    inc.b R10         mov.b   send\_data(R10), txmessage(R11) ;7 byte  add.b   txmessage(R11),R9  add.b #1, R11    ;checksum in R9         mov.b   R9, txmessage(R11) ;8 byte  add.b #1, R11    mov.b   #3Eh, txmessage(R11) ;9 byte  add.b #1, R11  mov.w   #0,R11  enq:    add.b   txmessage(R11),R9         mov.w   #tx\_rb, R12               ; MOV the address of the ring buffer into R12         mov.b   txmessage(R11), R13         call    #ringbuffer\_enqueue  add.b #1, R11           cmp.w   #10,R11   ;10th byte is more than needed 9th max         JEQ     enq\_done         JMP enq    enq\_done:         pop R11         pop R9  pop R13  bis.b   #UCA0TXIE, &IE2         ; Enable USCI\_A0 TX interrupt to begin sending data in tx\_rb         pop R12         ret    protocol\_recieve:         push    R12         push    R15         push    R9         mov.b   #0, R15    deqp:         ;mov.w   #rx\_rb, R12         ;call    #ringbuffer\_isempty         ;cmp.b   #0x01, R12         ;JEQ     skipdeqp         mov.w   #rx\_rb, R12               ; MOV the address of the ring buffer into R1         call    #ringbuffer\_dequeue         cmp.w   #60, R12         JEQ     datastartinit         add.w   #1,R15         cmp.w   #10,R15                   ;10th byte is more than needed 9th max         JGE     skipdeqp         JMP     deqp    datastartinit:         mov.b   #0, R5         mov.b   R12,rxmessage(R5)  datastart:         add.w   #1,R5         mov.w   #rx\_rb, R12         call    #ringbuffer\_isempty         cmp.b   #0x01, R12         JEQ     skipdeqp         mov.w   #rx\_rb, R12               ; MOV the address of the ring buffer into R1         call    #ringbuffer\_dequeue         mov.b   R12,rxmessage(R5)         cmp.w   #9,R5  ;10th byte is more than needed 9th max         JGE      deq\_done         JMP datastart  deq\_done:           mov.b  #0,R9         mov.b  #1,R8  ;; re  sum:         add.b rxmessage(R8),R9         add.b #1,R8         cmp.b #8,R8         JC  checksum         JMP  sum    checksum:    mov.b #8,R15         cmp.b rxmessage(R15),R9         JNE bad\_recieve\_func  destination\_add:         mov.w #0,R14         mov.w #2,R15         cmp.b rxmessage(R15),destinationadd         JEQ   good\_recieve  bad\_recieve\_func:         mov.b #1,&bad\_recieve         JMP skipdeqp  good\_recieve:         mov.b #0,&bad\_recieve  translate\_data:  push R14    mov.w #3,R15  ;;re  mov.w #0, R14   ;re         mov.b rxmessage(R15),recieved\_data(R14)  inc.w R15  inc.w R14         mov.b rxmessage(R15),recieved\_data(R14)  inc.w R15  inc.w R14         mov.b rxmessage(R15),recieved\_data(R14)  inc.w R15  inc.w R14         mov.b rxmessage(R15),recieved\_data(R14)          inc.w R15  inc.w R14         mov.b rxmessage(R15),recieved\_data(R14)  pop R14      skipdeqp:         pop R9         pop R15         pop R12         ret          ;-------------------------------------------------------------------------------  USCI\_RX\_ISR;  ;-------------------------------------------------------------------------------                                              mov.w   #tx\_rb, R12               ; MOV the address of the ring buffer into R12         push    R12         push    R13         mov.b   &UCA0RXBUF, R13         mov.w   #rx\_rb, R12  ;sets default start         call    #ringbuffer\_enqueue             pop     R13         pop     R12         reti    ;-------------------------------------------------------------------------------  USCI\_TX\_ISR;  ;-------------------------------------------------------------------------------           push    R12             mov.w   #tx\_rb, R12               ; MOV the address of the ring buffer into R12           call    #ringbuffer\_isempty           cmp.b   #0x01, R12           JNE     deq                     ; if the ring buffer is not empty, jump to deq and send the data           bic.b   #UCA0TXIE, &IE2         ; else - disable USCI\_A0 TX interrupt -> we are done sending data           jmp     skipdeq    deq:           mov.w   #tx\_rb, R12               ; MOV the address of the ring buffer into R12           call    #ringbuffer\_dequeue           mov.b   R12, &UCA0TXBUF  skipdeq:           pop     R12           reti  ;-------------------------------------------------------------------------------  TA0\_ISR;    Toggle P1.0  ;-------------------------------------------------------------------------------             ;time variable                       ;glove timer             inc.b R7             reti                            ;                                             ;           END |

Ring Buffer

|  |
| --- |
| NAME    RING\_BUFF                    ; module name    PUBLIC  ringbuffer\_flush                    ; make the label visible outside this file  PUBLIC  ringbuffer\_isempty                    ; make the label visible outside this file  PUBLIC  ringbuffer\_isfull                    ; make the label visible outside this file  PUBLIC  ringbuffer\_dequeue                    ; make the label visible outside this file  PUBLIC  ringbuffer\_enqueue                    ; make the label visible outside this file  ; Ring Buffer Structure  ; Head (1 byte)  ; Tail (1 byte)  ; array (32 bytes)  RSEG    CODE                    ; place program in 'CODE' segment    ;;;;;;;;;;;;;;;  ringbuffer\_flush:    MOV.B   #0x0, R14    MOV.B   R14, 0x1(R12)    MOV.B   R14, 0(R12)    RET      ;;;;;;;;;;;;;;;  ringbuffer\_isempty:   CMP.B   0x1(R12), 0(R12)   JNE     ??ringbuffer\_isempty\_0   MOV.B   #0x1, R12   RET      ??ringbuffer\_isempty\_0:   MOV.B   #0x0, R12   RET    ;;;;;;;;;;;;;;;;  ringbuffer\_isfull:   MOV.B   @R12, R14   MOV.B   0x1(R12), R15   CMP.B   #0x1f, R15   JNE     ??ringbuffer\_isfull\_0   CMP.B   #0x0, R14   JNE     ??ringbuffer\_isfull\_1   MOV.B   #0x1, R12   RET  ??ringbuffer\_isfull\_0:   MOV.B   R15, R15   ADD.W   #0x1, R15   MOV.B   R14, R14   CMP.W   R14, R15   JNE     ??ringbuffer\_isfull\_1   MOV.B   #0x1, R12   RET  ??ringbuffer\_isfull\_1:   MOV.B   #0x0, R12   RET    ;;;;;;;;;;;;;;;;;;  ; IN  R12 = Address of start of ring buffer structure  ; OUT R12 = Value  ; -- Must call ringbuffer\_isempty before calling this function  ringbuffer\_dequeue:   push    R14   push    R15   MOV.B   @R12, R15             ; Load head data into R15   MOV.B   R15, R15              ;                                        Example at 200h, with head = 1   MOV.W   R12, R14              ; load head addres into R14         :    200h   ADD.W   R15, R14              ; add head data to R14              :    201h   MOV.B   0x2(R14), R14         ; copy data at start of array      ;    203h = array[1] (202h = array[0])   ADD.B   #0x1, 0(R12)          ; increment head   CMP.B   #0x20, 0(R12)         ; test to see if head is outside bounds of array   JNC     ??ringbuffer\_dequeue\_0                                 ; if it is outside of bounds of array   MOV.B   #0x0, 0(R12)          ; "wrap" to 0  ??ringbuffer\_dequeue\_0:   MOV.B   R14, R12              ; copy data to output register R12     pop     R15   pop     R14   RET  ;;;;;;;;;;;;;  ; IN  R12 = Address of start of ring buffer structure  ;     R13 = Data to put in queue  ; OUT R12 = Success = 1, Failure = 0  ringbuffer\_enqueue:   PUSH.W  R10   MOV.W   R12, R10   MOV.W   R10, R12   CALL    #ringbuffer\_isfull   CMP.B   #0x0, R12   JNE     ??ringbuffer\_enqueue\_1   MOV.B   0x1(R10), R15   MOV.B   R15, R15   MOV.W   R10, R14   ADD.W   R15, R14   MOV.B   R13, 0x2(R14)   ADD.B   #0x1, 0x1(R10)   CMP.B   #0x20, 0x1(R10)   JNC     ??ringbuffer\_enqueue\_2   MOV.B   #0x0, 0x1(R10)  ??ringbuffer\_enqueue\_2:   MOV.B   #0x1, R12  ??ringbuffer\_enqueue\_0:   POP.W   R10   RET  ??ringbuffer\_enqueue\_1:   MOV.B   #0x0, R12   JMP     ??ringbuffer\_enqueue\_0         END |