RYAN BENTZ

ECE 372

DESIGN PROJECT 1

02/09/18

Ryan Bentz
ECE 372
Project 1

List of Deliverables

1. **Part I**
   - **A. Problem Description**
   - **B. High-Level Algorithm**
   - **C. Low-Level Algorithm**
   - **D. Project Code**

2. **Part II**
   - **E. Problem Description**
   - **F. High-Level Algorithm**
   - **G. Low-Level Algorithm**
   - **H. Project Code**

3. **Appendix**
   - **I. Supporting Documentation**
   - **J. Design Log**

1. Part I

A. Problem Description

Develop a program that sends a basic message to the RC8660 Evaluation Board on an interrupt basis, when a button connected to GPIO1_30 is pressed. Note that this is the same as the button service program that you implemented in ECE 371 Design Project #2 except that you have to add the code to initialize the processor as needed for the UART and send a message to the RC8860 following the basic flow shown in Hall Figure 5-24. As a later part of this step, you will send control codes to the synthesizer to change the voice used by the synthesizer. You may like the Darth Vader voice, for example. The 8660 Speech boards have a headphone jack in which you can plug your headphones to hear the message. For the first version, the messages should be resent each time the button is pushed.

B. High-Level Algorithm

MAIN:
initialize stacks for IRQ and supervisor modes
initialize the stack frames
initialize clocks to the peripherals
delay and wait for peripherals to be ready
initialize the UART
initialize the RC8660
initialize the GPIO
initialize the interrupt controller
enable interrupts in the processor

REPEAT
    Do nothing
UNTIL forever

INT DIRECTOR:
IF GPIO was source of interrupt
    IF button was source of interrupt
        go to button service routine
    ELSE
        exit ISR
IF UART was source of interrupt
    go to UART service routine
ELSE
    exit ISR
re-enable interrupts
RETURN to main

BUTTON SERVICE ROUTINE:
enable UART interrupts in interrupt controller
reset the GPIO interrupt request
reset IRQ interrupts in interrupt controller
exit ISR

UART SERVICE ROUTINE:
push the SPSR on to the stack
IF CTS was source of interrupt
    IF THR was source of interrupt
        Go to UART TRANSMIT
    ELSE
        disable CTS interrupt
IF THR was source of interrupt
    mask THR interrupt
reset interrupt controller IRQ requests
re-enable IRQ interrupts in the processor
return to main

UART TRANSMIT:
load the base address of message string
load current character index
get the next character
send the character to the UART
increment the index
update the character index
IF index == max characters
    disable the UART interrupts
    reset the current character index
    update the character index
ELSE
    return to main

DELAY:
Wait for second
Return to main

C. Low-Level Algorithm

*** Hook the IRQ interrupt with the custom procedure**

MAIN:

initialize stacks for IRQ and supervisor modes	
initialize clocks to the peripherals	
- enable GPIO clock	*Write 0x02 to CM_PER.GPIO1*
- enable the UART clock	*Write 0x02 to CM_PER.UART2*
delay and wait for peripherals to be ready	
initialize the UART	
- disable the UART to access DLL and DLH	*Write 0x07 to UART2.MDR1*
- switch to register configuration mode B	*Write 0x00BF to UART2.LCR*
- enable access to the IER UART register	*Write 0x10 to UART2.EFR*
- switch to register operational mode	*Write 0x00 to UART2.LCR*
- clear the IER register to disable the UART interrupts	*Write 0xFFEF to UART.IER*
- switch to register configuration mode B	*Write 0x00BF to UART2.LCR*
- load the baud rate values for DLL and DLH	*Write 0x004E to UART2.DLL*
	Write 0x0000 to UART2.DLH
- switch to register operational mode	*Write 0x0000 to UART2.LCR*
- load the new interrupt configuration	*Write 0x0082 to UART2.IER*
- switch to configuration mode B	*Write 0x00BF to UART2.LCR*
- restore the EFR ENHANCED_EN bit	*Write saved EN to UART2.EFR*
- load the new protocol formatting	*Write 0xFF03 to UART2.LCR*
- load the new UART mode	*Write 0xFF00 to UART2.MDR1*

- enable the UART
initialize the RC8660
- set the baud rate by sending the baud rate character

initialize GPIO pin for external interrupt
- configure pin as falling edge detect *Write 0x40000000 to GPIO1.FALLING_DETECT*
- initialize external interrupts for GPIO pin *Write 0x40000000 to GPIO1.IRQSTATUS_SET0*
- configure the UARD TXD pin *Write 0x01 to CONTROL.SPI0_D0*
- configure the UART CTS pin *Write 0x01 to CONTROL.LCD_DATA8*
enable interrupts in interrupt controller
- enable GPIO interrupts *Write 0x04 to INTC_MIR_CLEAR*
enable interrupts in CPSR

REPEAT
    Do nothing
UNTIL forever

INTERRUPT SERVICE ROUTINE:
push registers on stack
IF GPIO was source of interrupt *Read INTC_PENDING_IRQ3.[3]*
    IF button was source of interrupt *Read GPIO1_IRQSTATUS_0.[30]*
        go to button service routine
    ELSE
        re-enable IRQs on processor
        pop registers off stack
        exit ISR
ELSE
    IF UART was source of interrupt *Read INTC_PENDING_IRQ2.[10]*
        go to UART service routine
    ELSE
        re-enable IRQs on processor
        pop registers off stack
        exit ISR

BUTTON SERVICE ROUTINE:
push saved program status register on stack
mask lower priority interrupts
initialize interrupt controller for UART *Write 0x400 to INTC_MIR_CLEAR3.[10]*
reset the GPIO interrupt request *Write 0x40000000 to GPIO1_IRQSTATUS_0.[30]*
reset the interrupt controller *Write 0x01 to INTC_CONTROL.[1]*
pop the saved program status register
RETURN to main

UART SERVICE ROUTINE:

push saved program status register on stack
check if CTS but not THR was source of interrupt

IF CTS but not THR was source of interrupt	*Read 0x0020 from UART2.IIR*
reset the interrupt controller IRQs	*Write 0x01 to INTC_CONTROL*
re-enable IRQ interrupts in the processor	
pop saved program status register off the stack	
RETURN to main	
IF THR but not CTS was source of interrupt	*Read 0x0002 from UART2.IIR*
mask THR interrupt	*Write 0xFFFD to UART2.IER*
reset the interrupt controller IRQs	*Write 0x01 to INTC_CONTROL*
re-enable IRQ interrupts in the processor	
pop saved program status register off the stack	
RETURN to main	
IF THR and CTS was source of interrupt	
go to UART TRANSMIT	

UART TRANSMIT ROUTINE:

push registers on to the stack
load the message base address and character index offset
get the value of the next character
send the character
increment the character index

IF the index == message max	
reset the counter	
disable the UART interrupts in the interrupt controller	*Write 0x0400 to INTC_MIR_SET2*
ELSE	
RETURN to main	

DELAY:

Load counter to count for one second
WHILE counter is not equal to zero
    Decrement counter
Return to main

D. Project Code

See attached.

Part II

E. Problem Description

Modify the program so that when the button is pushed, it speaks the messages, waits 10 seconds, and then repeats the message, if the button is not pushed again.

F. High-Level Algorithm

MAIN:

```
initialize stacks for IRQ and supervisor modes
initialize the stack frames
initialize clocks to the peripherals
delay and wait for peripherals to be ready
initialize the UART
initialize the RC8660
initialize the GPIO
initialize the timer
initialize the interrupt controller
enable interrupts in the processor
REPEAT
        Do nothing
UNTIL forever
```

INT DIRECTOR:

```
IF GPIO was source of interrupt
        IF button was source of interrupt
                go to button service routine
        ELSE
                exit ISR
IF UART was source of interrupt
        go to UART service routine
ELSE
        exit ISR
IF TIMER was source of interrupt
        go to timer service routine
ELSE
        exit ISR
re-enable interrupts
RETURN to main
```


Ryan Bentz
ECE 372
Project 1

TIMER SERVICE ROUTINE:
initialize UART interrupts
reset timer IRQ flags
reset interrupt controller
re-enable interrupts in the processor

BUTTON SERVICE ROUTINE:
set the timer counter value
IF timer is enabled
        disable the timer
ELSE
        enable the timer
reset the GPIO interrupt request
reset IRQ interrupts in interrupt controller
exit ISR

UART SERVICE ROUTINE:
push the saved program status register on stack
IF CTS was source of interrupt
        IF THR was source of interrupt
                Go to UART TRANSMIT
        ELSE
                disable CTS interrupt
IF THR was source of interrupt
        mask THR interrupt
pop saved program status register off stack
reset interrupt controller IRQ requests
re-enable IRQ interrupts in the processor
return to main

UART TRANSMIT:
load the base address of message string and current character index
get the next character
send the character to the UART
increment the index
update the character index
IF index == max characters
        disable the UART interrupts
        reset the current character index
        update the character index
ELSE
        return to main

DELAY:
Wait for second
Return to main

G. Low-Level Algorithm

*** Hook the IRQ interrupt with the custom procedure**

MAIN:
initialize stacks for IRQ and supervisor modes
initialize clocks to the peripherals

- enable GPIO clock	*Write 0x02 to CM_PER.GPIO1*
- enable the UART clock	*Write 0x02 to CM_PER.UART2*
- enable 32.768kHz clock for timer	*Write 0x02 to CM_DLL.Timer3*
- enable the TIMER clock	*Write 0x02 to CM_PER.Timer3*

delay and wait for peripherals to be ready

initialize the UART

- disable the UART to access DLL and DLH	*Write 0x07 to UART2.MDR1*
- switch to register configuration mode B	*Write 0x00BF to UART2.LCR*
- enable access to the IER UART register	*Write 0x10 to UART2.EFR*
- switch to register operational mode	*Write 0x00 to UART2.LCR*
- clear the IER register to disable the UART interrupts	*Write 0xFFEF to UART.IER*
- switch to register configuration mode B	*Write 0x00BF to UART2.LCR*
- load the baud rate values for DLL and DLH	*Write 0x004E to UART2.DLL*
	Write 0x0000 to UART2.DLH
- switch to register operational mode	*Write 0x0000 to UART2.LCR*
- load the new interrupt configuration	*Write 0x0082 to UART2.IER*
- switch to configuration mode B	*Write 0x00BF to UART2.LCR*
- restore the EFR ENHANCED_EN bit	*Write saved EN to UART2.EFR*
- load the new protocol formatting	*Write 0xFF03 to UART2.LCR*
- load the new UART mode	*Write 0xFF00 to UART2.MDR1*
- enable the UART	

initialize the RC8660
- set the baud rate by sending the baud rate character

initialize GPIO pin for external interrupt

- configure pin as falling edge detect	*Write 0x40000000 to GPIO1.FALLING_DETECT*
- initialize external interrupts for GPIO pin	*Write 0x40000000 to GPIO1.IRQSTATUS_SET0*
- configure the UARD TXD pin	*Write 0x01 to CONTROL.SPI0_D0*
- configure the UART CTS pin	*Write 0x01 to CONTROL.LCD_DATA8*

initialize the timer
- enable auto-reload for timer overflow *Write 0x02 to TIM3.CNTL*

- set the counter value *Write 0xFFFAFFFF to TIM3.LDR*
Write 0xFFFAFFFF to TIM3.CNTR
- enable the timer IRQ overflow *Write 0x02 to TIM3_IRQ_SET*
enable interrupts in interrupt controller
- enable GPIO interrupts *Write 0x04 to INTC_MIR_CLEAR*
enable interrupts in CPSR
REPEAT
    Do nothing
UNTIL forever

INTERRUPT SERVICE ROUTINE:
push registers on stack
IF GPIO was source of interrupt *Read INTC_PENDING_IRQ3.[3]*
    IF button was source of interrupt *Read GPIO1_IRQSTATUS_0.[30]*
        go to button service routine
    ELSE
        re-enable IRQs on processor
        pop registers off stack
        exit ISR
IF UART was source of interrupt *Read INTC_PENDING_IRQ2.[10]*
    go to UART service routine
ELSE
    re-enable IRQs on processor
    pop registers off stack
    exit ISR
IF TIMER was source of interrupt *Read INTC_PENDING_IRQ2[6]*
    go to timer service routine
ELSE
    re-enable IRQs on processor
    pop registers off stack
    exit ISR
re-enable interrupts
RETURN to main

TIMER SERVICE ROUTINE:
initialize UART interrupts *Write 0x0400 to INTC_MIR_CLEAR2*
reset timer IRQ flags *Write 0x2 to TIM3_IRQ_STATUS*
reset interrupt controller *Write 0x1 to INTC_CONTROL*
re-enable interrupts in the processor

BUTTON SERVICE ROUTINE:
push saved program status register on stack
mask lower priority interrupts
set the counter value *Write 0xFFFAFFFF to TIM3.LDR*
Write 0xFFFAFFFF to TIM3.CNTR
enable/disable the timer *Read 0x01 from TIM3.CNTRL*
IF timer is enabled
    disable the timer *Write 0x03 to TIM3.CNTRL*
ELSE
    enable the timer *Write 0x02 to TIM3.CNTRL*
reset the GPIO interrupt request *Write 0x40000000 to GPIO1_IRQSTATUS_0.[30]*
reset the interrupt controller *Write 0x01 to INTC_CONTROL.[1]*
pop the saved program status register
RETURN to main

UART SERVICE ROUTINE:
push saved program status register on stack
check if CTS but not THR was source of interrupt
IF CTS but not THR was source of interrupt *Read 0x0020 from UART2.IIR*
    reset the interrupt controller IRQs *Write 0x01 to INTC_CONTROL*
    re-enable IRQ interrupts in the processor
    pop saved program status register off the stack
    RETURN to main
IF THR but not CTS was source of interrupt *Read 0x0002 from UART2.IIR*
    mask THR interrupt *Write 0xFFFD to UART2.IER*
    reset the interrupt controller IRQs *Write 0x01 to INTC_CONTROL*
    re-enable IRQ interrupts in the processor
    pop saved program status register off the stack
    RETURN to main
IF THR and CTS was source of interrupt
    go to UART TRANSMIT

UART TRANSMIT ROUTINE:
push registers on to the stack
load the message base address and character index offset
get the value of the next character
send the character
increment the character index
IF the index == message max
    reset the counter
    disable the UART interrupts in the interrupt controller *Write 0x0400 to INTC_MIR_SET2*
ELSE
    RETURN to main

DELAY:

Load counter to count for one second
WHILE counter is not equal to zero
    Decrement counter
Return to main

H. Project Code

See attached.

Appendix

I. Supporting Documentation

RC8660

- Auto baud rate detection with 0x0D character.
- Acceptable baud rates: 300 – 115200 kbps.
- Requires command character at the end of every message string. RC8660 will not start talking until it receives a 0x0D or 0x00 character.
- Serial Protocol: 8 data bits (LSB first), 1 or more stop bits, no parity.
- 16-byte buffer. CTS may be checked every 8 bytes with no data loss.

UART2 - BeagleBone – RC8660 Connections

RC8660	BB3	AM3358	Description	Mode
TXD	P9:21	B17	SPI0_D0	1
RXD	P9:22	A17	SPI0_SCLK	1
CTSN	P8:37	U1	LCD_DATA8	6
RTSN	P8:38	U2	LCD_DATA9	6

Peripheral Clocks

Peripheral Clocks

REGISTER	ADDRESS/OFFSET	DESCRIPTION
CM_DLL	0x44E0 050C	Write 0x2 to bits [0:2] to enable peripheral
CM_PER.GPIO1	0x44E0 00AC	Write 0x2 to bits [0:2] to enable peripheral
CM_PER.UART2	0x44E0 0070	Write 0x2 to bits [0:2] to enable peripheral
CM_PER.TIMER3	0x44E0 0084	Write 0x2 to bits [0:2] to enable peripheral

Interrupt Connections

Interrupt	Number	Register	Bit	Read/Write Value
GPIO1	98	MIR 3	3	0x0000 0004
UART2	74	MIR 2	10	0x0000 0400
TIMER3	69	MIR 2	5	0x0000 0020

UART Registers

UART

REGISTER	ADDRESS/OFFSET	DESCRIPTION
UART.MDR1	0x4802 4020	Write 0x07 to bits [0:2] to put the UART in disabled state
UART.MDR1	0x4802 4020	Write 0x00 to bits [0:2] to put the UART in 16x mode
UART.LCR	0x4802 400C	Write 0x00BF to bits [0:15] to switch to configuration mode B
UART.LCR	0x4802 400C	Write 0x0000 to bits [0:7] to switch to operational mode
UART.EFR	0x4802 4008	Write 0x10 to bits [0:7] to enable Enhanced features. EN = bit 5
UART.IER	0x4802 4004	Write 0xEF to bits [0:7] to put the UART in sleep mode. Write a 0 to bit 4
UART.IER	0x4802 4004	Write 0x82 to bits [0:7] to enable CTS and THR interrupts. CTS = bit 7, THR = bit 1
UART.DLL	0x4802 4000	Write 0x004E to bits [0:7] per datasheet for 38.4 kbps
UART.DLH	0x4802 4004	Write 0x0000 to bits [0:7] per datasheet for 38.4 kbps

UART Baud Rate

Since the RC8660 has a wide range of acceptable baud rates and can auto-detect the baud rate. We choose a value of 38.4 kbps. The reference manual indicates the DLL and DLH registers are used to compute the baud rate. The DLL, DLH values should be 0x00, 0x4E respectively.

Timer Settings

The timer rate equation is as follows:

(FFFF FFFFh – TLDR + 1) × timer Clock period × Clock Divider (PS)

With timer Clock period = 1/ timer Clock frequency and PS = 2(PTV + 1).

As an example, if we consider a timer clock input of 32 kHz, with a PRE field equal to 0, the timer output period is:

Table 20-8. Value and Corresponding Interrupt Period

TLDR	Interrupt period
0000 0000h	37 h
FFFF 0000h	2 s
FFFF FFF0h	500 us
FFFF FFFEh	62.5 us

Figure 5 – Timer calculation discussion from technical reference manual

Calculating the timer load register value for a 1 second clock period is as follows:

$$TLDR = FFFFFFFFh - \left(\frac{50000h}{01h}\right) - 1 = FFFAFFFFh$$

Per the example in the datasheet, a clock source with 32kHz clock and no prescaler division has a 2s interrupt with a TLDR value of FFFF0000h. FFFFh converts to 65535_{10} which is exactly two times the clock source 32768_{10}. Therefore, if we use a timer period value of 8000h, we get the word to write for a 1 second delay in TLDR. To get a 10s timer, we multiply that value by 10 and get a hex value of 0x50000.

Timer Registers

TIMER 3

REGISTER	ADDRESS/OFFSET	DESCRIPTION
TIM.CNTRL	0x4804 2038	Write 0x2 to bits [0:2] to enable auto-reload
TIM.COUNT	0x4804 203C	Write 0xFFFFAFFF to bits [0:31] to eanble a 10s timer period
TIM.LDR	0x4804 2040	Write 0xFFFFAFFF to bits [0:31] to eanble a 10s timer period
TIM.IRQ_SET	0x4804 202C	Write 0x2 to bits [0:2] to enable timer overflow IRQ

GPIO Registers

GPIO

REGISTER	ADDRESS/OFFSET	DESCRIPTION
FALLING_DETECT	0x4804 C14C	Write 0x40000000 to bits [0:31] to enable falling edge detect on pin 30
IRQ_STATUS_SET0	0x4804 C034	Write 0x40000000 to bits [0:31] to enable IRQ interrupts on pin 30
CONTROL.SPI0_D0	0x44E1 0954	Write 0x1 to bits [0:7] to put the pin in mode 1 for TXD pin
CONTROL.LCD_DATA8	0x44E1 08C0	Write 0x6 to bits [0:7] to put the pin in mode 6 for CTS pin

J. <u>Design Log</u>

1/13/18:

- Read the instructions from Dr. Hall for the assignment to get an understanding of what needs to be done.
- Read the RC8660 manual to understand the serial interface and made notes about the baud rate and pin connections.
- Printed out the UART section of the AM3358 manual and skimmed through it to get a feel for what is inside.
- Began gathering notes of relevant information in order to complete the project.

1/14/18:

- Studied the RS-232 section of Hall ECE 371 textbook to get an understanding of RS-232 handshaking, signals, and DTE/DCE descriptions.
- Began gathering information needed to initialize the UART. I need to know what the RC8660 pins are mapped to on the B3 board. Went down to campus to verify the connections and they are all connected to UART4 instead of UART2. Sent Dr. Hall an e-mail confirming the connections needed to be changed. I noticed in the assignment text that Hall says what the pin to header mapping is supposed to be so I will use that for the mode settings for now until I hear otherwise.

1/19/18:

- Wrote the high-level and low-level algorithms for part 1. I am using the UART initialization procedure in the reference manual to setup the UART.

1/20/18:

- Wrote out the code for the entire program based on the algorithm.
- I used the UART initialization procedure as detailed in the reference manual.
- I realized that I would need to update the algorithm to include some kind of procedure to handle the UART transmit.
- Created a special procedure for UART transmit because it could be called from either the button service or from the UART interrupt.

1/22/18

- Reviewed Hall Chapters 1-5 notes about ARM programming to make sure that I was refreshed on what to do and the rules of ARM programming.
- I decided that for this specific project the interrupts do not need to be re-entrant. For part 1, the button can never interrupt the UART unless it somehow managed to get pressed twice in an extremely small amount of time. For part 2, the timer period is so long that the message has already been sent long before the timer interrupt occurs so there never will be a conflict. Same with the button. In order to test it, you would need to press the button at exactly the right time before it starts talking because it does not start talking until it has received all the characters.

1/24/18

- Began debugging and testing the program. First thing I noticed is that I forgot to change the GPIO pin modes for the UART pins.
- Struggling with the flow of checking CTS and THR. I still don't understand why we care about if THR is empty or not. All that should matter is if CTS is low or not. There should never be a condition where CTS is low but the THR is not. This would mean that the UART has data to send but the RC8660 is ready for new data. This should not be possible.

- I realized that my original algorithm to transfer control of the UART to start sending characters from the button service procedure was not correct. Instead, when the button is pressed, it should enable the interrupts for the UART and then the interrupts will handle the transmission process logic.
- Next part to figure out was how to keep track of what characters to send. I decided I will have a location in memory that counts down from the total number of characters to zero. When the counter gets to zero, we have sent all the characters. We can also use the counter as the offset to send to THR knowing which byte to send.
- UART is working. Debugging issue with not returning. I had the wrong return instruction from the branch and link
- Current problem: the UART is sending a continuous stream of data. It is sending the correct amount because it is the same 5 characters over and over. Except only two of the characters are correct. It is sending 0x4F, 0x4C, 0x00, 0x00, 0x00.
- I realized that there is a continuous loop and the counter never resets. I should finish the actual writing of the code before testing. I need to also implement some code to disable the interrupts and reset the UART when the counter reaches zero.

1/26/18:

- I brought in my logic analyzer and fixed the issues with the continuous loop. I am sending the right characters at the correct baud rate but I am not hearing anything on the output.
- Having problems with the baud rate. For some reason, the RC8660 is not responding to my character strings. I figured out from further reading of the datasheet that I needed to send a specific character in order to let the RC8660 figure out what the baud rate is.
- I am still not hearing anything on the output. I went back through the code and checked the logic analyzer and it turns out that I had the wrong values in DLL and DLH for setting the baud rate.
- I can now hear a sample output from the RC8660 and everything works great.

1/29/18:

- I realize that my algorithm is not correct for the UART. The button service does not need to call UART transmit itself. If we just unmask the UART interrupts, the UART interrupt will handle the transmit process.
- Everything works much better now and part 1 is essentially complete.

2/4/18:

- Working on part 2. I re-used the code for the timer from last year's project and I didn't have any problems getting everything working fine. Project is essentially finished.

2/6/18:

- Initially, I had set up my algorithm such that in order to transmit the characters, I start with the base address of the message and then add the index to create the offset to the specific character. The index starts at max and then counts down to zero. I did this because its more efficient to count down to zero and compare for zero and for a nonzero number. But, this means that I have to enter the text in the message backwards (which is not good). So, I have come up with a new scheme where the message is created in correct order and the counter counts up to max instead of down to zero.

- I was having problems getting the new code to work and finally figured out that when I initially created the algorithm, I had done the final write to memory of the new value for the index later in the algorithm. This was probably not good practice and I changed it so the write to memory happens during the process of incrementing the counter and determining if the counter needs to be reset or not.