WRITING AN ARM, RS-232C DRIVER
FOR
AN RC SYSTEMS 8660 VOICE SYNTHESIZER

Design Project #1 ECE 372 Winter 2018
Copyright
Douglas V. Hall
Portland, OR
January 2018

INTRODUCTION:

Specialized processors that convert an ASCII text string to spoken words are now available as chipsets or as stand-alone modules. The chipsets can be connected directly on the microprocessor buses in an embedded system and with that connection the ASCII string can be written directly to the chipset. For the stand-alone modules, the ASCII strings are usually sent over an RS-232C or I2C serial cable. For communicating with the RC Systems 8660 Text-to-speech module we have chosen for this exercise, you will use the UART2 device on a BeagleBone Black board that has an adapter to interface with the RS232C signal levels.

Other than to have a little fun with a microprocessor, the major goals of this exercise are to give you some practice working with a programmable peripheral device that has multiple interrupt sources and some more practice using the **R3T3SD** process to successfully work your way through a new design project that is assigned to you on the job. The lesson here is that, if you work through each step carefully, rather than rushing to the end, you should not find this exercise difficult. You have already done many of the pieces needed for this project and just have to put them together and add a few new parts. The main new parts are interrupt-driven communication with a UART and mapping processor signals to available pins.

NOTE: You MUST do all work by yourself with no help from anyone except the Instructor and the TA. Please do NOT contact companies for assistance. You are not a paying customer and therefore not entitled to ask them to solve your problems.

NOTE: YOU MUST KEEP AN AS-YOU-GO LOG OF THE ACTUAL STEPS YOU TOOK, YOUR DATA FINDINGS AT EACH STEP, PROBLEMS ENCOUNTERED, HOW YOU SOLVED THESE PROBLEMS, AND HOW YOU TESTED YOUR PROGRAM.

REFERENCES

Hall Text Chapters 4 and Chapter 5; The BeagleBone Black System Reference Manual on http://elinux.org/Beagleboard:BeagleBoneBlack; and the TI Sitara AM 335X manual at http://www.ti.com/lit/ug/spruh73p/spruh73p.pdf.

PROJECT DESCRIPTION:

You have just accepted a design position with a medical company that makes products which have speech synthesis capability. The particular product you are going to be working on is part of a blood pressure measuring instrument. Your task is to upgrade a section that will speak a blood pressure message such as “Your blood pressure is 120 over 70” and then a pulse message such as “Your pulse is 54.”, when a button connected to GPIO1_30 is pushed. The message will be repeated, if the button is pushed again.

Assume that the engineer that you are replacing was in the middle of moving the program from an older Arcom Viper processor to a a Beagle Bone Black Processor that is used in the new model, when he decided to leave the company. Your specific task is to rewrite the RS-232C speech synthesizer program section so that it will work correctly with UART2 on a Beagle BoneBlack board, instead of with the Bluetooth UART on the old Arcom Viper board. The documentation he left behind included the partially converted and untested program attached here and the accompanying discussion in Hall Chapter 5. The relevant data sheets were somehow lost when someone quickly moved into the window office that had been occupied by the engineer who left, so you will have to obtain these as needed from the company webpages. **Do NOT contact company!** You are not a paying customer and therefore not entitled to contact the company. Everything you need is in the on-line data sheets, the text, or the old program attached.

Your overall project steps are as follows:

1. 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.

2. For the second part of the project, you will 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.

3. Optionally, for extra credit, make the synthesizer do something impressive such as play music, sing, speak a foreign language, etc.

SUGGESTED PROCEDURE

1. Thoroughly STUDY the section in Hall Chapter 5 on RS-232C and especially learn the following.
   a. The format in which asynchronous serial data is sent.
   b. The meaning of the term Baud rate and how a Baud rate clock is produced.
   c. The meaning of the terms DTE and DCE.
   d. For the 9-pin connector, the name of each signal, the direction for each signal on DTE and on DCE, and the function of each signal.
   e. The use of RTS# and CTS# for data flow control (handshaking).
   f. The straight through connections for DTE to DCE, such as the connection from a PC to a MODEM. Note that the B3 Board is configured as DTE and the RC 8660 board is configured as DCE, so a straight-through cable is required for this application.
   g. The Null MODEM connections for DTE to DTE or DCE to DCE.
   h. How to interact with a UART. Note that the UART on the B3 board has the same basic architecture as the one in the text but just has an additional Infrared interface capability.
   i.
2. To get an overview of the RC 8660 Speech Processor chip features and interfaces go to the http://www.rcsys.com/Downloads/rc8660.pdf. Read the first few pages to give you an overview of the chip functions and how the chips are connected in a system. Summarize your main findings in your log but don't be upset if you don't understand all the details at this point. Note especially how the Auto-Baud Rate Detection system works and the timing for it.

3. Next study the UART section of the Sitara AM335X Technical Reference Manual at http://www.ti.com/lit/ug/spruh73p/spruh73p.pdf and specifically look at UART2 and the UART registers. With the help of your experience in ECE 371 make a high level list of the tasks required to set up for the button interrupt as before.

4. Next think about initializing the UART First step is to figure out how to turn on the UART2 clock.

5. Next step is to map the UART2 TxD, RxD, CTS and RTS signals to pins on the B3 board's P8 and P9 connectors. As an example, here's how you would do it if you were using UART5.
   When used as a Linux machine, the Multiplexers are initialized in MODE0 and the LCD signals are connected to pins as shown for the P8 connector in the table below and in Table 10 in the BeagleBone Black System Reference Manual. These signals are used to drive the HDMI framer. Since we are not using the HDMI, we can switch the multiplexers so the UART5 signals go to pins as shown in the P8 PIN and NAME columns in the table. As you can see, UART5_CTSN (Clear to send) can be connected to pin 31 on P8, instead of the lcd_14 signal by programming the pin MUX to do so. Likewise, find the pin for the UART5_ RTSN and the corresponding lcd number that you use to identify the multiplexer that you need to specify the needed mode for these signals. Also, find lhe lcd you use to switch pins 37 and 38 to UART5_TXD and UART5_RXD connections.

P8 PIN	PROC	NAME	MODE0 Assignments
27	U5	GPIO2_22	lcd_vsync
28	V5	GPIO2_24	lcd_pclk
29	R5	GPIO2_23	lcd_hsync
30	R6	GPIO2_25	lcd_ac_bias_en
31	V4	UART5_CTSN	lcd_data14 Switch to MODE 6
32	T5	UART5_RTSN	lcd_data15 Switch to MODE 6
33	V3	UART4_RTSN	lcd_data13
34	U4	UART3_RTSN	lcd_data11
35	V2	UART4_CTSN	lcd_data12
36	U3	UART3_CTSN	lcd_data10
37	U1	UART5_TXD	lcd_data8 Switch to MODE 4
38	U2	UART5_RXD	lcd_data9 Switch to MODE 4
39	T3	GPIO2_12	lcd_data6
40	T4	GPIO2_13	lcd_data7
41	T1	GPIO2_10	lcd_data4
42	T2	GPIO2_11	lcd_data5
43	R3	GPIO2_8	lcd_data2
44	R4	GPIO2_9	lcd_data3
45	R1	GPIO2_6	lcd_data0
46	R2	GPIO2_7	lcd_data1

The registers you use to change the pin MUX mappings are in the Control Module. Go to the L4_WKUP Peripheral Memory Map in the ARM Cortex-A8 Memory Map to find the base address for the Control Module. Then go to this module to find the registers and offsets of the Control Module registers. The registers you need to change the mode of a pin MUX are identified by the LCD name or by the gpmc name in the MODE0 column. For a start, find the Control Module register offset for the lcd_data14 register that controls the MUX that you program to connect UART5_CTSN to P8 pin 31. Pop up the register for this and find the bits you use to set the desired mode. The register format is the same for all of the many mapping registers, so they just show it once for all of them. For UART5 you wold need to map the 4 signals as shown above and in the text. You then write the steps required to change the pin MUXes to get the desired signals to the desired P8/P9 pins. Note that some of the signals are inputs and some are outputs, so you have to put the appropriate value in bit 5 for each.

You are using UART2 for this project so using this example and the Tables 12 and 13 in thee BeagleBoneBlack manual you can figure out and write the steps necessary to map the UART2 signals to pins on the P8 and P9 connectors. Specifically, you want UART2_TXD on pin 21 of P9, UART2_RXD on pin 22 of P9, UART2_CTSN on pin37 of P8, and UART2_RTSN on pin 38 of P8.

6. Complete the high level task list for the first version of the program, including the handshake steps shown in the flowchart shown in Hall Figure 5-24 and mentally work your way though it to make sure you have everything set up, enabled, etc. The attached program is for a different processor but from it you can learn a lot about working with the UART, since the UART2 follows the same standard architecture as the one in the program. Remember, that you don't want to enable the system to send a Baud detection character or message until the button is pushed. Also, remember to reset and disable the FIFO, since you are not using it. After the message is fully sent, make sure to get everything set up for another send.

7. Once you get the high level task list done, carefully write the low level task list and then write the commented assembly language program for it.

8. Test and debug your program in steps. First step is to set a breakpoint and test that the button detect section and initialization section are working. Then check if execution gets to the talker section. Then see if it reads a character correctly from the character array. Then check if it writes a character to the Talker. Finally see if it sends a simple message to the Talker. Then see if it correctly speaks a simple message of your choice in the default voice each time the button is pushed. Then modify the program, so that it sends the message in some voice other than the default, to the RC 8660 when the button is pushed.

9. When the program works correctly, demonstrate its operation to a TA or me and have your commented source file signed off. Save this version for turn in and make a copy of the source file to use for developing the final program. A little celebration dancing and shouting is appropriate at this point. (Maybe have the Talker congratulate you.)

10. Modify your overall high level algorithm so the program speaks the messages when the button is pushed, waits 10 seconds, and speaks he messages again, if the button is not pushed again. 11. Build the new program using the "Fast is Slow" rule and when the new program works correctly, demonstrate it to a TA or me for a signature.

Optional Extra Credit Fun

The RC 8660 Chipset is capable of many audio functions in addition to doing text to speech conversion.

1. Study the data sheet for the RC8660.
2. Figure out how to implement some advanced feature(s) of the RC 8660 chipset.
3. Create a new version of your program to implement one or more of these features.
4. When you have implemented and documented some feature(s) that will impress me, demonstrate your success to me or TA for the signoff. Note that extra credit given based on "wow factor" and documentation. Have fun.

DELIVERABLES

Your documentation for the project should include:

A. A detailed design log that clearly shows all the development steps you took and the results at each step.
B. Clearly written high level and low level algorithms for the basic Button-Driven Talker.
C. Clearly written high level and low level algorithms for the alker Program with timed repeat.
D. Fully documented source files for the two programs signed by the TA to verify that your programs work and meet specifications.
E. A signed statement that you developed and wrote this program by yourself with no help from anyone except the instructor and/or the T.A. and that you did not give any help to

anyone else. (Any evidence of joint work will result in project grades of zeros for all parties involved.)

GRADING KEY FOR ECE372 DESIGN PROJECT I D.V. HALL WINTER 2016

	POSSIBLE SCORE	
WORKING PROGRAM (TA questions answered correctly)	**50**	_______
ALOGRITHMS (CLEAR AND COMPLETE)	**20**	_______
LOG (DETAILED AND "AS YOU GO")	**30**	_______
TOTAL	**100**	_______
BONUS MAX (Play music, songs, speaking another language, MP3 player, etc.)	**20**	_______

PROGRAM EXAMPLE FROM OLDER PROCESSOR

```
@  PROGRAM TO INTERFACE RC SYSTEMS 8660 SPEECH SYNTHESIS BOARD
@ WITH ZEUS BOARD USING RS-232C COM 2 PORT ON AN INTERRPT BASIS
@ DRAFT STARTED BY DOUG HALL, FALL 2010.
@ (INCOMPLETE conversion from Viper Talker Program
@  NOT completed and OBVIOUSLY NOT tested on Zeus board yet.)

.text
.global _start
_start:

@ INITIALIZE GPIO 77 AS LOW OUTPUT, GPIO75 FOR INPUT AND RISING EDGE DETECT
     LDR   R0, =0x40E00000 @  LOAD BASE FOR GPIO REGISTERS
     ADD   R1, R0, #0x14   @  ADD OFFSET OF GPDR2 TO BASE R1 = GPDR2
     ADD   R2, R0, #0x20   @  ADD OFFSET OF GPSR2 TO BASE, R2 = GPSR2
     ADD   R3, R0, #0x2C   @  ADD OFFSET OF GPCR2 TO BASE, R3 = GPCR2
     LDR   R4, =0x00002000 @  WORD TO CLEAR BIT 77,  SIGN OFF WHEN OUTPUT
     STR   R4, [R3]        @  WRITE TO GPCR2
     LDR   R6, [R1]        @  READ GPDR2 TO GET CURRENT VALUE
     ORR   R6, R6, R4      @ (MODIFY) SET BIT 13 TO MAKE GPIO 77  OUTPUT
     BIC   R6, R6, #0x800  @ CLEAR BIT 11 TO MAKE GPIO 75 INPUT FOR SURE
     STR   R6, [R1]        @  WRITE WORD BACK TO GPDR2
     ADD   R4, R0, #0x38       @ LOAD ADDRESS OF GRER2 REGISTER
     LDR   R1, [R4]        @ READ CURRENT VALUE OF REGISTER
     MOV   R2, #0x800      @ LOAD MASK TO SET BIT 11
     ORR   R1, R1, R2      @ SET BIT11
     STR   R1, [R4]        @ WRITE WORD BACK TO GRER2 REGISTER
@ INITIALIZE GPIO <115> AS RISING EDGE INTERUPT INPUT FOR COM2 UART RC8660
```

```
     ADD  R1, R0, #0x130        @ POINT TO GRER3 REGISTER
     LDR  R4, [R1]         @ READ GRER3 REGISTER
     ORR  R4,R4,#0x00080000 @ BIT 19=1 TO ENABL GPIO115,RISING EDGE DETECT
     STR  R4, [R1]         @ WRITE BACK TO GRER3
@ INITIALIZE UART
@ SET DLAB BIT IN LINE CONTROL REGISTER TO ACCESS BAUD RATE DIVISOR
     LDR  R0, =0x10800006  @ POINT TO  UART LINE CONTROL REGISTER
     MOV  R1, #0x83        @ VALUE FOR DIVISOR ENABLE =1, 8 BITS, NO
                           @ PARITY, 1 STOP BIT
     STRB R1, [R0]         @ WRITE TO LINE CONTROL REGISTER
@ LOAD DIVISOR VALUE TO GIVE 38.4K BITS/SEC
     LDR  R0, =0x10800000  @ POINTER TO DIVISOR LOW REGISTER
     MOV  R1, #0x18        @ #0x18 DIVISOR FOR 38.4 KBITS/SEC
     STRB R1, [R0]         @ WRITE TO DIVISOR LOW REGISTER
     MOV  R1, #0x0         @ VALUE FOR DIVISOR HIGH REGISTER
     STRB R1, [R0,#2]      @ PRE-INDEX TO WRITE TO DIVISOR HIGH REGISTER
@ TOGGLE DLAB BIT BACK TO 0 TO GIVE ACCESS TO Tx REGISTER AND Rx REGISTER
     LDR  R0, =0x10800006  @ POINT TO BLUETOOTH UART LINE CONTROL REGISTER
     MOV  R1, #0x03             @ VALUE FOR DIVISOR ENABLE =0, 8 BITS, NO
                           @    PARITY, 1 STOP BIT

     STRB R1, [R0]         @ WRITE TO LINE CONTROL REGISTER

@ ENABLE TX INTERRUPT AND ENABLE MODEM STATUS CHANGE INTERRUPT
     LDR  R0, =0x10800002  @ POINTER TO INTERUPT ENABLE REGISTER (IER)
     MOV  R1, #0x0A        @  BIT 3 = MODEM STATUS INTERRUPT, BIT 1 = Tx
INTERRUPT ENABLE
     STRB R1, [R0]         @ WRITE TO INTERUPT ENABLE REGISTER
@ CLEAR FIFO AND TURN OFF FIFO MODE
     LDR  R0, =0x10800004  @ POINTER TO FIFO CONTROL REGISTER (FCR)
     MOV  R1, #0x00        @ VALUE TO DISABLE FIFO AND CLEAR FIFO
     STRB R1, [R0]         @ WRITE TO FCR
@ HOOK IRQ PROCEDURE ADDRESS AND INSTALL OUR INT_HANDLER ADDRESS
     MOV  R1, #0x18        @ LOAD IRQ INTERRUPT VECTOR ADDRESS 0x18
     LDR  R2, [R1]   @ READ INSTR FROM INTERRUPT VECTOR TABLE AT 0x18
     LDR  R3, =0xFFF       @ CONSTRUCT MASK
     AND  R2, R2,R3        @ MASK ALL BUT OFFSET PART OF INSTRUCTION
     ADD  R2, R2,#0x20     @ ABSOLUTE ADDRESS OF IRQ PROC IN LITERAL POOL
     LDR  R3,[R2]          @ READ BTLDR IRQ ADDRESS FROM LITERAL POOL
     STR  R3, BTLDR_IRQ_ADDRESS      @ SAVE BTLDR IRQ ADDRESS FOR LATER
     LDR  R0,=IRQ_DIRECTOR @ LOAD  ADDRESS OF OUR INTERRUPT DIRECTOR
     STR  R0,[R2]@ STORE THIS ADDRESS IN LITERAL POOL WHERE LDR PC GOES
@ INITIALIZE INTERRUPT CONTROLLER FOR BUTTON AND UART ON IP<10>
@ NOTE: DEFAULT VALUE OF IRQ FOR ICLR BIT 10 IS DESIRED VALUE, SO NO SEND
@ NOTE: DEFAULT VALUE OF DIM BIT IN ICCR IS DESIRED VALUE, SO NO WORD SENT
     LDR  R0, =0x40D00004  @ LOAD ADDRESS OF PXA270 INTERRUPT MASK (ICMR)
     LDR  R1, [R0]         @ READ CURRENT VALUE OF REGISTER
     ORR  R1, R1, #0x00000400   @ SET BIT 10 TO UNMASK IP10
     STR  R1, [R0]         @ WRITE WORD BACK TO ICMR REGISTER
@ MAKE SURE IRQ INTERRUPT ON PROCESSOR ENABLED BY CLEARING BIT 7 IN CPSR
     MRS  R3, CPSR         @ COPY CPSR TO R3
     BIC  R3, R3, #0x80         @ CLEAR BIT 7 (IRQ ENABLE BIT)
     MSR  CPSR_c, R3       @ WRITE BACK TO LOWEST 8 BITS OF CPSR
```

```
@ MAIN PROGRAM WAIT LOOP
LOOP:    NOP                @ WAIT FOR INTERRUPT HERE (SIMULATE MAINLINE
     B    LOOP              @       PROGRAM EXECUTION)

IRQ_DIRECTOR:               @ CHAINS INTERRUPT PROCEDURES
     STMFD SP!, {R0-R1,LR}  @  SAVE REGISTERS TO BE USED IN PROCEDURE ON
STACK
     LDR  R0, =0x40D00000  @ POINT AT IRQ PENDING REGISTER (ICIP)
     LDR  R1, [R0]                    @ READ ICIP REGISTER
     TST  R1, #0x400        @ CHECK IF GPIO 119:2 IRQ INTERRUPT ON IP<10>
     BEQ  PASSON            @ NO, MUST BE OTHER IRQ, PASS ON TO SYSTEM
PROGRAM
     LDR  R0, =0x40300148  @ LOAD ADDRESS OF GEDR3 REGISTER
     LDR  R1, [R0]          @ READ GEDR3 REGISTER
     TST  R1, #0x00080000  @ CHECK FOR UART INTERRUPT ON BIT 19
     BNE  TLKR_SVC          @ YES, GO SEND CHARACTER. NO, CHECK FOR BUTTON
     LDR  R0, =0x40E00050  @ LOAD GEDR2 REGISTER ADDRESS TO CHECK IF GPIO
75
     LDR  R1, [R0]          @ READ GPIO EDGE DETECT REGISTER (GEDR2) VALUE
     TST  R1, #0x800        @ CHECK IF BIT 11 = 1 (GPIO75 EDGE DETECTED)
     BNE  BUTTON_SVC        @ YES, MUST BE BUTTON PRESS
                            @ GO SERVICE - RETURN TO WAIT LOOP FROM SVC
PASSON:    LDMFD SP!, {R0-R1,LR} @ NO, MUST BE OTHER GPIO 119:2 IRQ,
                                 @ RESTORE REGISTERS
     LDR  PC, BTLDR_IRQ_ADDRESS @ GO TO BOOTLOADER IRQ SERVICE PROCEDURE.
                      @ BOOTLOADER WILL USE RESTORED LR TO RETURN TO
                      @         MAINLINE LOOP WHEN DONE

BUTTON_SVC:
     MOV  R1, #0x800 @ VALUE TO CLEAR BIT 11 IN GEDR2 REGISTER
                      @ THIS WILL ALSO RESET BIT 10 IN ICPR AND ICIP
                      @ IF NO OTHER GPIO 119-2 INTERRUPTS
     STR  R1, [R0]   @ WRITE TO GPIO GEDR2 REGISTER, NO RMW NEEDED
@ INITIALIZE THE MODEM CONTROL REGISTER (MCR) TO ENABLE UART INTERRUPT AND
@ ASSERT RTS# TO SEND MESSAGE TO TALKER
     LDR  R0, =0x10800008  @ POINTER TO MODEM CONTROL REGISTER (MCR)
     MOV  R1, #0x0A         @ ENABLE UART INTERRUPT (NO R-M-W IN THIS CASE)
     STRB R1, [R0]          @ WRITE BACK TO MCR
     LDMFD SP!, {R0-R1,LR}  @ RESTORE REGISTERS, INCLUDING RETURN ADDRESS
     SUBS PC, LR, #4        @ RETURN FROM INTERRUPT (TO WAIT LOOP)


@ COULD HAVE GOTTEN HERE BY EITHER CTS# CHANGE OR THR EMPTY
@ NEED TO CHECK CTS# LOW AND THR EMPTY BEFORE SEND.
@ NOTE:  CTS INTERRUPT ONLY TRIGGERED ON CHANGE IN CTS# NOT CTS# ASSERTED

TLKR_SVC:
     STMFD SP!,{R2-R5}           @ SAVE ADDITIONAL REGISTERS
     LDR  R0, =0x1080000C        @ POINT TO MODEM STATUS REGISTER (MSR)
     LDRB R3, [R0]    @ READ MSR (RESETS MODEM STATUS CHANGE INTERRUPT BIT
     TST  R3, #0x10   @ CHECK IF CTS# IS CURRENTLY ASSERTED (MSR BIT 4)
     BEQ  NOCTS             @ IF NO, THEN GO CHECK IF THR CTS# ASERTED
     LDR  R0, =0x1080000A  @ POINT TO LINE STATUS REGISTER (LSR)
     LDRB R1, [R0]          @ READ LSR (DOES NOT CLEAR INTERRUPT)
```

```
      TST   R1, #0x20       @ CHECK IF THR-READY IS ASSERTED
      BEQ   GOBCK           @ IF NO, THEN EXIT AND WAIT FOR THR-READY
      B     SEND  @ ELSE YES, BOTH ARE ASSERTED, SEND CHARACTER
NOCTS:
      LDR   R0, =0x1080000A @  POINT TO LINE STATUS REGISTER (LSR)
      LDRB  R1, [R0]        @ READ LSR (DOES NOT CLEAR INTERRUPT)
      TST   R1, #0x20  @ CTS NOT ASSERTED, CHECK IF THR-READY IS ASSERTED
      BEQ   GOBCK      @ IF NO, THEN EXIT.  NEITHER CTS OR THR ARE ASSERTED
                       @ MUST HAVE BEEN OTHER MSR INTERRUPT SOURCE
                       @ ELSE NO CTS# BUT THR IS ASSERTED,
                       @ DISABLE INTERRUPT ON THR TO PREVENT SPINNING WHILE
                            @ WAITING FOR CTS#
      LDR   R4, =0x10800002 @ LOAD POINTER TO UART INTERUPT ENABLE REGISTER
      MOV   R5, #0x08  @  DISABLE BIT 1 = Tx INTERRUPT ENABLE (MASK THR)
      STRB  R5, [R4]        @ WRITE TO INTERUPT ENABLE REGISTER
      B     GOBCK           @ EXIT TO WAIT FOR CTS# INTERRUPT

@ UNMASK THR, SEND CHARACTER, IF END OF MESSAGE RESET CHAR COUNT
                                 @ AND DISABLE UART INTERRUPT
SEND: LDR   R4, =0x10800004 @ LOAD POINTER TO INTERRUPT ENABLE REGISTER
      MOV   R5, #0x0A  @ BIT 3 = MODEM STATUS, BIT 1 = Tx INTERRUPT ENABLE
      STRB  R5, [R4]   @ WRITE TO INTERUPT ENABLE REGISTER
      LDR   R0, =CHAR_PTR   @ SEND CHARACTER, R0 = ADDRESS OF POINTER STORE
      LDR   R1, [R0]   @ R1 = ADDRESS OF DESIRED CHARACTER IN TEXT STRING
      LDR   R2, =CHAR_COUNT @ R2 = ADDRESS OF COUNT STORE LOCATION
      LDR   R3, [R2]        @ GET CURRENT CHARACTER COUNT VALUE
      LDRB  R4, [R1], #1    @ READ CHAR TO SEND FROM STRING,INC PTR IN R1
      STR   R1, [R0]   @ PUT INCREMENTED ADDRESS BACK IN CHAR_PTR LOCATION
      LDR   R5, =0x10800000      @ POINT AT UART TRANSMIT BUFFER
      STRB  R4, [R5]   @ WRITE CHARACTER TO TRANSMIT BUFFER
                       @ CLEARS TDRQ INTERRUPT SOURCE UNTIL THR EMPTY AGAIN)
      SUBS  R3, R3, #1      @ DECREMENT CHARACTER COUNTER BY 1
      STR   R3, [R2]        @ STORE CHARACTER VALUE COUNTER BACK IN MEMORY
      BPL   GOBCK           @ GREATER THAN OR EQUAL ZERO, MORE CHARACTERS
      LDR   R3, =MESSAGE    @ DONE, RELOAD. GET ADDRESS OF START OF STRING
      STR   R3, [R0]   @ WRITE IN CHAR POINTER STORE LOCATION IN MEMORY
      MOV   R3, #MESSAGE_LEN@ LOAD ORIGINAL NUMBER OF CHAR IN STRING AGAIN
      STR   R3, [R2]        @ WRITE BACK TO MEMORY FOR NEXT MESSAGE SEND
      LDR   R0, =0x40D00004 @ LOAD ADDRESS OF MODEM CONTROL REGISTER (MCR)
      LDRB  R1, [R0]        @ READ CURRENT VALUE OF REGISTER
      BIC   R1, R1, #0x08   @ CLEAR BIT 3 TO DISABLE UART INTERRUPTS
      STRB  R1, [R0]        @ WRITE BYTE  BACK TO ICMR REGISTER
GOBCK:      LDMFD SP!,{R2-R5}    @ RESTORE ADDITIONAL REGISTERS
      LDMFD SP!,{R0-R1,LR}  @ RESTORE ORIGINAL REGISTERS, RETURN ADDRESS
      SUBS  PC, LR, #4      @ RETURN FROM INTERRUPT (TO WAIT LOOP)

BTLDR_IRQ_ADDRESS:    .word  0        @ SPACE TO STORE BOOTLOADER IRQ
ADDRESS

.data
MESSAGE:  .byte  0x0D
.ascii "Take me to your leader"
.byte  0x0D
.align 2
```

```
CHAR_PTR: .word  MESSAGE    @ POINTER TO NEXT CHARACTER TO SEND
CHAR_COUNT:  .word 24       @ COUNTER FOR NUMBER OF CHARACTERS TO SEND
                            @ NUMBER OF CHARACTERS COUNTS X-1 DOWN TO 0
.end
```