WRITING AN ARM, RS-232C DRIVER

**FOR**

**AN RC SYSTEMS 8660 VOICE SYNTHESIZER**

Design Project #1 ECE 372 Winter 2019

Josh Pradera

Introduction :

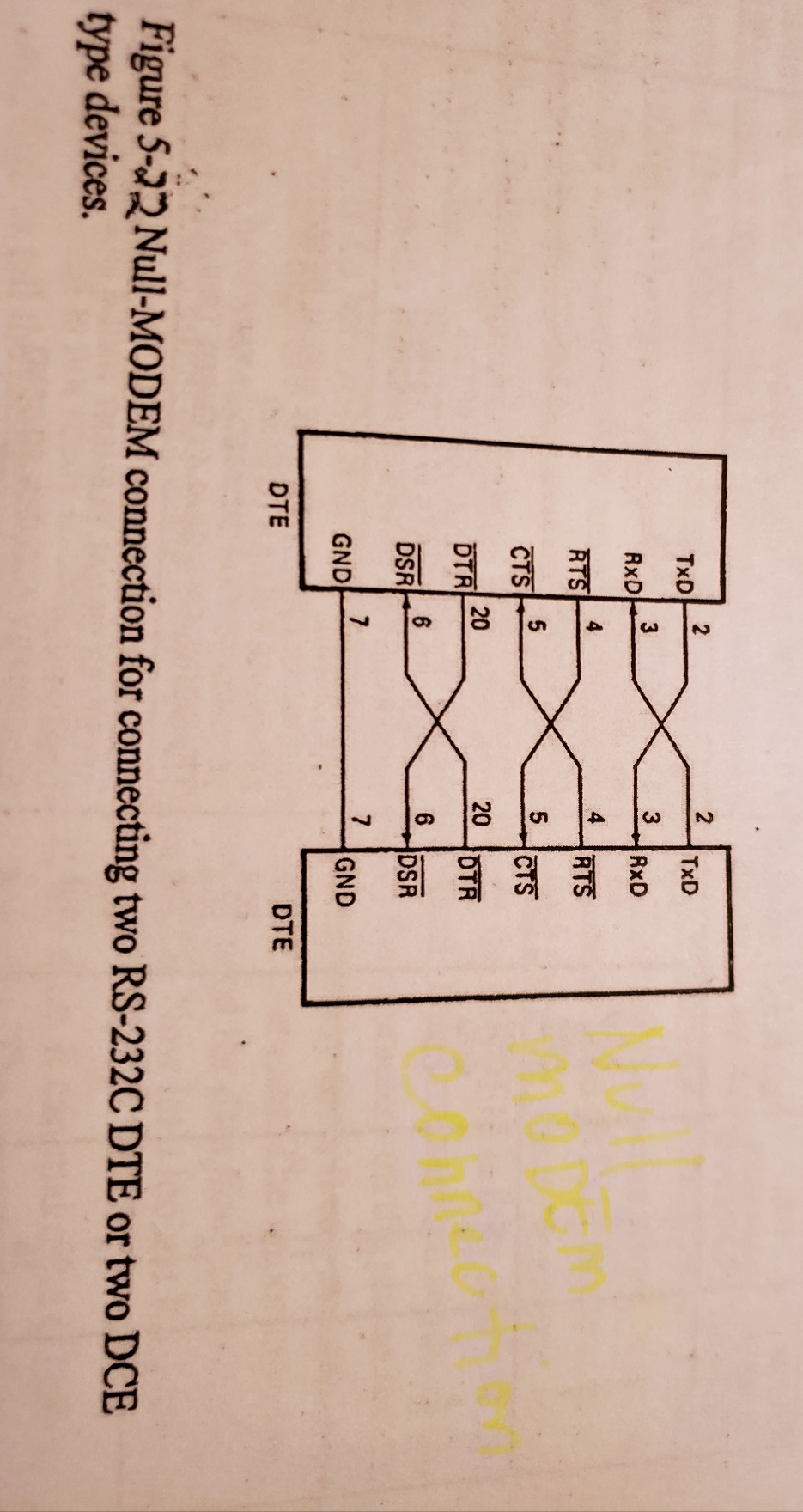
In this lab we learn how to implement chipsets that can convert ASCII text into string spoken words. These chipsets can be connected to embedded systems like the beagle bone black. The ASCII strings are usually sent over an RS-232C or I2C serial cable. To communicate with the RC system 8660 text-to-speech module I will use UART5 device on a Beagle Bone Black. Working with the programmable peripheral device that have multiple interrupt sources and also getting practice using the R3T3SD process to successfully work your way through a new design.

**PROCEDURE:**

1. Thoroughly STUDY the section in Hall Chapter 5 on RS-232C and especially learn the following.
   1. The format in which asynchronous serial data is sent.
      * The start of character its indicated by a line going low for one bit time(the start bit).The bits are then sent one after another starting with the lest significant bit
      * Depending if a parity bit it’s found 1 or 2 stops bits are set (high signal)
   2. The meaning of the term Baud rate and how a Baud rate clock is produced.
      * Baud rate its defined by 1/the time for one bit and it’s the rate at witch the data bits are being sent.
   3. The meaning of the terms DTE and DCE.
      * DCE: Data Communication Equipment
      * DTE: Data Terminal Equipment
   4. 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.
      * The RS-232C signals

|  |  |  |  |
| --- | --- | --- | --- |
| Pin Numbers (9 pin connector) | Common Name | Description | Signal Direction On DCE |
| 1 | CD | Received Line Signal Detector | Out |
| 2 | RxD | Received Data | Out |
| 3 | TxD | Transmitted Data | In |
| 4 | DTR | Secondary Request to Send | In |
| 5 | GND | Signal Ground | - |
| 6 | DSR | Data Set Ready | Out |
| 7 | RTS | Request To Send | In |
| 8 | CTS | Clear To Send | Out |
| 9 |  | Ring indicator | Out |

* 1. The use of RTS# and CTS# for data flow control (handshaking).
     + Handshaking signals are required to make sure the receiver is ready before the sender sends a data word. Most current systems use two of the RS-232C signals.
     + When the terminal has a character ready to send, it will assert the Request to Send (RTS#) signal to the MODEM. If the carrier tone signal from the remote computer, it will assert the Carrier Detect signal to the terminal.
     + If the MODEM its ready to send data, it asserts the Clear To Send (CTS#) signal to the terminal. The terminal then sends data characters to the MODEM and the MODEM sends out the characters on the phone line one after the other.
     + If the MODEM is not ready to send a character at any time it can make the CTS# line to the terminal HIGH to tell the terminal not to send another character.
  2. 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.
     + Connecting DTE to DCE straight through means pin2 on DTE on pin 2 on DCE same with the other pins.
  3. **The Null MODEM connections for DTE to DTE or DCE to DCE.**



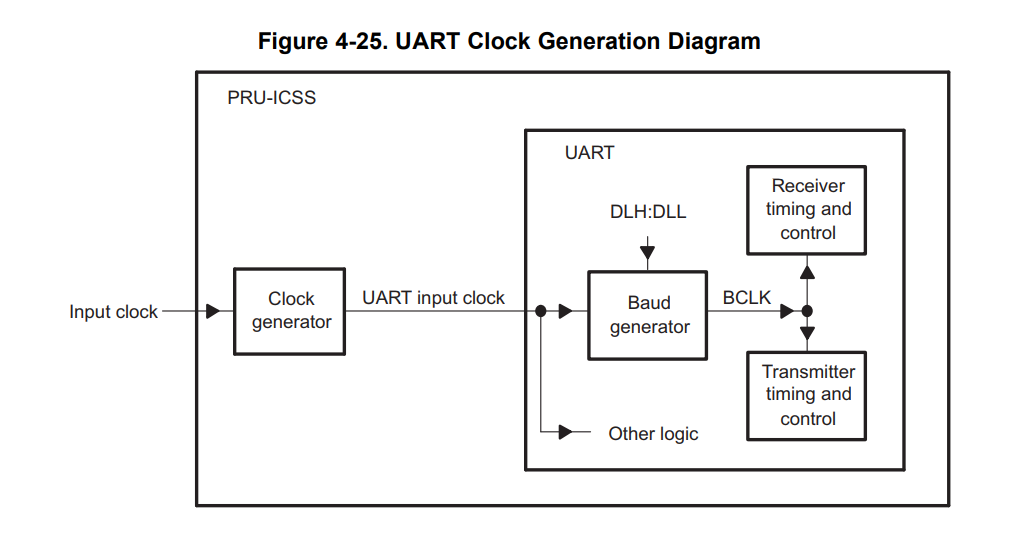
**Double Talk RC8660**

Voice and sound synthesizer integrating a text to speech (TTS) processor, audio recording and play back musical and sinusoidal tone generators, telephone dialer and A/D converter into an easy to use chipset. TTS processor uses an 8-bit bus interface, virtually any ASCII text can be streamed to the RC8660 for automatic conversion into speech by the TTS processor. The output of this device its delivered in both digital PCM audio formats and analog signals.

**UART Universal Asynchronous Receiver/Transmitter**

The UART can be used in alternate FIFO (TL16C550). The advantage of this mode its that will alleviate the processor by relieving the CPU from excessive software overhead by buffering received and transmitted characters. The process of Receiver and Transmitter consist in storing up to 16 bytes including three additional bits of errors status per byte for the inverter FIFO.

The UART includes control capabilities and a processor interrupt system that can be tailored to minimize software management of the communications link.



The UART also includes a baud rate generator capable of dividing the PRU\_ICSS\_UART\_CLK input clock by divisors from 1 to 65535 and producing a 16× reference clock or a 13× reference clock for the internal transmitter and receiver logic.

**PART 1**

Develop a program that sends a basic message (e.g. “testing”) to the RC8660 Evaluation Board on an interrupt basis, when a button connected to GPIO2\_1 is pressed.

1. Set up Stacks as before
2. Set up GPIO2 for falling edge on GPIO2\_1
3. Initialize INTC for GPIO2\_1 interrupt
4. Initialize INTC for UART5 Interrupt
5. Map the UART5 TxD, RxD, CTS and RTS to pins available on the Beagle Bone Black (BBB) P8 connector by changing the mode of the multiplexers that select the signals that go to the pads that are connected to P8 pins signals.

|  |  |  |  |
| --- | --- | --- | --- |
| BBB P8 Pin# | PROC | Name | Mode |
| 37 | U1 | UART5\_TXD | lcd\_data8 Switch to MODE 4 |
| 38 | U2 | UART5\_RXD | lcd\_data9 Switch to MODE 4 |
| 32 | T5 | UART5\_RTSN | lcd\_data15 Switch to MODE 6 |
| 31 | V4 | UART5\_CTSN | lcd\_data14 Switch to MODE 6 |

**Output from TI**

**PinMuxTool devicetree.dtsi**

/\* This file was auto-generated by TI PinMux on 1/12/2019 at 9:05:02 PM. \*/

/\* This file should only be used as a reference. Some pins/peripherals, \*/

/\* depending on your use case, may need additional configuration. \*/

myuart5\_pins\_default: myuart5\_pins\_default {

pinctrl-single,pins = <

AM33XX\_IOPAD(0xc4, PIN\_INPUT | MUX\_MODE4) /\* (U2) lcd\_data9.uart5\_rxd \*/

AM33XX\_IOPAD(0xc0, PIN\_OUTPUT | MUX\_MODE4) /\* (U1) lcd\_data8.uart5\_txd \*/

AM33XX\_IOPAD(0xd8, PIN\_INPUT | MUX\_MODE6) /\* (V4) lcd\_data14.uart5\_ctsn \*/

AM33XX\_IOPAD(0xdc, PIN\_OUTPUT | MUX\_MODE6) /\* (T5) lcd\_data15.uart5\_rtsn \*/

>;

};

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Control Module** base address | BBB PIN 8 # | Offset | Acronym | Mode # |
| 0x44E1\_0000 | 31 | 8D8 | Conf\_lcd\_data14 | Mode 6 |
| 0x44E1\_0000 | 32 | 8DC | Conf\_lcd\_data15 | Mode 6 |
| 0x44E1\_0000 | 37 | 8C0 | Conf\_lcd\_data8 | Mode 4 |
| 0x44E1\_0000 | 38 | 8C4 | Conf\_lcd\_data9 | Mode 4 |

**Turn on USER LED 0**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **GPIO** | **31** | **30** | **29** | **28** | **27** | **26** | **25** | **24** | **23** | **22** | **21** | **20** |
| **Bit** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **HEX** | 0 | | | | 0 | | | | 2 | | | |
| **GPIO** | **19** | **18** | **17** | **16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| **Bit** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **HEX** | 0 | | | | 0 | | | | 0 | | | |
| **GPIO** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Bit** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **HEX** | 0 | | | | 0 | | | |

Hex: 0x00200000

|  |  |
| --- | --- |
| **Enable bit as Output** | **Word to turn USR logic High when output** |
| Pin 21 : 0xFFDFFFFF | 0x00200000 |

**Part\_1**

**Standard Program Structure and Algorithm:**

**Mainline**

* INITIALIZE STACK
* INITIALIZE GPIO
* Initialize memory with length of the message
  + Load a pointer to the counter (CHAR\_C)
  + Get the message length (#MSG\_LENGHT) and store it in memory

**Detect falling edge on GPIO2\_1**

- Load Address of CM\_PER\_GPIO2\_CLKCTRL ( 0x44E000B0 )

- Copy value to turn on the clock ( 0x02 ) and proceed to turn on GPIO2 clock

* Set up GPIO2\_1 for detecting falling edge by storing 0x00000002 to 0x481AC14C (0x481AC000 (GPIO2\_Base) + 0x14C (offset of GPIO\_FALLINGEDGE\_DETECT)
* Set up GPIO2\_1 for IRQ enable by storing 0x00000002 to 0x481AC034 (0x481AC000 (GPIO2\_Base) + 0x34 (offset of GPIO\_IRQSTATUS\_SET0)

**CONTROL MODULE**

**Initialize UART**

* Change pin 37 on P8 to mode 4 by writing #4 to 0x44E108C0 (0x44E10000 (Control Module Base) + 0x8C0 (offset of pin 37))
* Change pin 38 on P8 to mode 4 by writing #4 to 0x44E108C4 (0x44E10000 (Control Module Base) + 0x8C4 (offset of pin 38))
* Change pin 31 on P8 to mode 6 by writing #6 to 0x44E108D8 (0x44E10000 (Control Module Base) + 0x8D8 (offset of pin 31))
* Change pin 32 on P8 to mode 6 by writing #6 to 0x44E108DC (0x44E10000 (Control Module Base) + 0x8DC (offset of pin 32))

**Set up Baud rate for Transmitting**

* Enable DLL and DLH access by writing 0x83 (config mode A) to 0x481AA00C (0x83 to enable Divisor latch and select 8-bit word length, 0x481AA00C = 0x481AA000 (UART Base) + 0x0C (offset of LCR register))
* SET DLL to 156 and DLH to 78 for a 38.4 Baud rate by writing 0x4E to UART5 Base address (0x481AA000)
* Choose UART 16x Mode by writing 0x00 to 0x481AA020 (0x00 to select UART 16x Mode, 0x481AA020 = 0x481AA000 (UART Base) + 0x20 (offset of MDR1 register)
* Write 0x1A to 0x481AA000 (DLL register) and write 0x00 to 0x481AA004 (DLH register)

**Initialize Tx and Rx Registers**

* Clear bit 7 in LCR register to enable Tx and Rx registers
* Enable THR and MSR interrupt by writing 0x0A to 0x481AA004 = 0x481AA000 (UART Base) + 0x04 (offset of IER\_UART register)

**Turn off FIFO**

* Turn off FIFO by writing 0x06 to 0x481AA008 = 0x481AA000 (UART Base) + 0x08 (offset of FIFO Control Register).

**INITIALIZE INT**

* Turn UART clock ON
* Load address of CM\_PER\_UART5-CLKCTRL ( 0x44E00038 )
* Use 0x02 value to turn on clock
* Enable UART5 interrupt by writing 0x4000 to 0x482000A8 (0x4000 for unmasking INT 46, 0x482000A8 = 0x48200000 (INTC\_Base) + 0xA8 (Offset of INTC\_MIR\_CLEAR1).
* Unmask INTC INT 32 for the button with value 0x00000001 and the address of INT\_MIR\_CLEAR\_1 register (0x482000A8)

**ENABLE Processor IRQ in CSPR**

* Copy current value in CPSR into a register
* Clear bit 7 of the current CPSR
* Write the modified result back to CPSR ( 8 bit lowest).
* Infinite loop to wait for an Interrupt

**LOOP:** NOP

B LOOP

**INT\_DIRECTOR:**

* Saved uses register and linked register on Stack
* Load GPIO2\_IRQSTATUS\_0 (0x481AC02C)
* Test bit 1 with value (0x00000002) INT32
* If it’s not from button then check UART (BEQ UART\_CHECK)
* In the case of the button then go BUTTON\_SVC
* Otherwise PASS\_ON (go back to wait loop)
* Check if the interrupt come from UART5 by testing bit 14 (0x00004000) (int number 46) of the current value stored in INTC\_PENDING\_IRQ1 (0x482000B8: 0x48200000 (base address for INTC) + 0xB8 offset for INTC\_PENDING\_IRQ1)
* If the interrupt not come from UART branch to PASS\_ON
* Else (interrupt from UART)
* Load address of IIR\_UART5 read data and test bit 0x01 if its set then

Go to TLKR\_SVC (Handshake for sending character)

* Otherwise return to PASS\_ON

**Button\_SVC**

* The Function of the Button service it’s to enable UAR5 to generate interrupt signals so characters can be sent to the RC 8660 on an interrupt basis.

**Turn off IRQ request from GPIO2\_1**

* Writing 0x00000002 to 0x481AC02C (0x481AC000 (GPIO1\_base) + 0x2C (offset of GPIO1\_IRQ\_STATUS\_SET0))
* Writing 0x1 to 0x48200048 (0x48200000 (INTC\_base) + 0x48 (offset of INTC\_CONTROL))

**Enable THR and MSR interrupt**

* Writing 0x0A to 0x481AA004 = 0x481AA000 (UART Base) + 0x04 (offset of IER\_UART register)
* Restore saved registers and return to wait loop

**TLKR\_SVC procedure**

* Save registers
* Turn off GPIO2\_1 interrupt from button
* Check if **CTS signal** is asserted low and THR is empty.
* Read MSR register at 0x481AA018, check bit 4 (0x10)

If bit 4 (0x10) = 0

{

Check THR status by reading LSR register at 0x481AA014, check bit 5 (0x20)

If THR (LSR bit 5 (0x20) = 0)

{

Restore saved registers and return to mainline

}

Else

{ - Mask THR INT

Disable TX interrupt by writing 0x08 to 0x481AA004 (UART Base) + 0x04 (offset of IER\_UART register) to prevent spinning problem

}

Else

{

Check THR status

Read LSR register at 0x481AA014 and check bit 5

If bit 5 = 0

{

Restore saved registers and return to mainline

}

Else, go to SEND procedure

}

**SEND**

Enable THR and MSR interrupts by writing 0x0A to 0x481AA004 = 0x481AA000 (UART Base) + 0x04 (offset of IER\_UART register)

Get message store at CHAR\_PTR address

Get one character in the message and increment pointer by 1

Store the character in the THR register (0x481AA000)

Get the value store at CHAR\_COUNT address

Decrement the value by 1 and store it back to CHAR\_COUNT

Check if it is the last character by checking the current value after decrement

If the value is less than to zero counter

{

Restore the original message to CHAR\_PTR

Restore the original message’s length to CHAR\_COUNT

Disable THR interrupt by clearing bit 2 of IER\_UART (0x481AA004)

Restore saved registers and return to mainline

}

Else if the value its bigger or equal to zero go back to mainline and continue sending next character)

Restore saved registers and return to mainline.

**Data**

MESSAGE:

@.byte 0xD ( 1st letter must be sent to the speaker)

// .byte 0x01, 0x39, 0x4F ( Command to change the voice, if you want)

.ascii “Test” (Message that the speaker will speak)

.byte 0xD

CHAR\_PTR: .word MESSAGE (Pointer for the message)

CHAR\_COUNT: .word 6 (Pointer for the message’s length counter)

.EQU MESSAGE\_LEN, 6 ( Value to restore the counter)

Project 1 ECE 372

@ Josh Pradera

@ After the button connected to GPIO2\_1 its pushed

@ the launcher orally count in intervals of 1 second

@ from 10 t0 0, then says blast off,

@ and send a signal that turns LED 0 to simulate ignition.

**.text**

**.global** \_start

**.global** INT\_DIRECTOR

**\_start:**

.equ MSG\_LEN, 6

@ Load stack data

LDR R13, =STACK1 @ Point to base of SVC stack

ADD R13, R13, #0x1000 @ Move pointer to top of stack

CPS #0x12 @ Switch to IRQ mode

LDR R13, =STACK2 @ Point to base of IRQ stack

ADD R13, R13, #0x1000 @ Move pointer to top of stack

CPS #0x13 @ Switch back to SVC mode

@ Detect falling edge on GPIO2\_1 and enable to assert POINTERPEND1

MOV R2, #0x02 @ Value to turn on clock

LDR R1, =0x44E000B0 @ Address of CM\_PER\_GPIO2\_CLKCTRL

STRB R2, [R1] @ Turn on GPIO2 clock

LDR R0, =0x481AC000 @ Base address of GPIO2 registers

MOV R2, #0x00000002 @ Value to enable bit 1

STR R2, [R0, #0x14C] @ Turn on bit 1 for GPIO2\_FALLINGDETECT

STR R2, [R0, #0x34] @ Enable GPIO2\_IRQSTATUS\_SET\_0 for bit 1

@ Initialize UART

LDR R1, =0x44E10000 @ Base address of pinout MODE0

MOV R2, #0x04 @ Value to switch to MODE6

STRB R2, [R1, #0x8C0] @ Switch P8-37 to MODE6

STRB R2, [R1, #0x8C4] @ Switch P8-38 to MODE6

MOV R2, #0x06 @ Value to switch to MODE1

STRB R2, [R1, #0x8D8] @ Switch P8-31 to MODE1

STRB R2, [R1, #0x8DC] @ Switch P8-32 to MODE1

MOV R2, #0x02 @ Value to enable UART clock

LDR R1, =0x44E00038 @ Address of CM\_PER\_UART5\_CLKCTRL

STRB R2, [R1] @ Turn on UART5 clock

LDR R1, =0x481AA000 @ Base address of UART5

MOV R2, #0x83 @ Value for UART config mode A

STRB R2, [R1, #0x0C] @ Switch UART5\_LCR to config mode A

MOV R2, #0x4E @ Value to set DLL to 156

STRB R2, [R1] @ DLL value of 0x4E and DHL value of 0x00

@for 38.4 kbps Baud rate

MOV R2, #0x00 @ Value to set MDR to x16

STRB R2, [R1, #0x04] @ Set DLH to 0

STRB R2, [R1, #0x20] @ Switch UART5\_MDR1 to x16 divisor mode

MOV R2, #0x03 @ Value for UART operational mode

STRB R2, [R1, #0x0C] @ Switch UART5\_LCR to operational mod

MOV R2, #0x06 @ Value to disable FIFO

STRB R2, [R1, #0x08] @ Clear and disable UART5 FIFO

@ Initialize INTC

LDR R1, =0x48200000 @ Load base address of INTC

MOV R2, #0x02 @ Value to reset the INTC

STRB R2, [R1, #0x10] @ Write to INTC\_SYSCONFIG to reset

MOV R2, #0x4000 @ Value to unmask INTC INT 46

STR R2, [R1, #0xA8] @ Write to INTC\_MIR\_CLEAR1 to unmask

MOV R2, #0x00000001 @ Value to unmask INTC INT 32, GPIOINT2A

STRB R2, [R1, #0xA8] @ Write to INTC\_MIR\_CLEAR1 to unmask

@ Enable processor IRQ in CSPR

MRS R3, CPSR @ Copy CPSR to R3

BIC R3, #0x80 @ Clear bit 7

MSR CPSR\_c, R3 @ Write back to CPSR

@ Wait for an interrupt

**LOOP:** NOP

B LOOP

**INT\_DIRECTOR:**

STMFD SP!, {R0-R5, LR} @ Push registers onto stack

LDR R0, =0x481AC02C @ Load GPIO2\_IRQSTATUS\_0 register address

LDR R1, [R0]

TST R1, #0x00000002 @ Test bit 1 (INT 32 button)

BEQ UART\_CHECK @ IF is not coming from button, check UART!

BNE BUTTON\_SVC @ IF yes, branch to service the button

B PASS\_ON @ Otherwise PASS\_ON

**UART\_CHECK:**

LDR R0, =0x482000B8 @ Load address of INTC\_PENDING\_IRQ1

LDR R1, [R0] @ Read the data

TST R1, #0x00004000 @ Test bit 14 (INT 46 UART)

BEQ PASS\_ON @ IF is not from UART THEN PASS\_ON

LDR R0, =0x481AA008 @ Otherwise Load address of IIR\_UART5

LDR R1, [R0] @ Read the data

TST R1, #0x01 @ Check bit 0

BEQ TALKER\_SVC @ IF bit 0 its low then branch to TALKER\_SCV

B PASS\_ON @ Otherwise PASS\_ON

**PASS\_ON:**

LDR R0, =0x48200048 @ Address of INTC\_CONTROL

MOV R2, #0x01 @ Value to reset IRQ

STR R2, [R0] @ Reset IRQ and also listen for a new one

LDMFD SP!, {R0-R5, LR} @ Restore registers

SUBS PC, LR, #0x04 @ Pass execution to beginning of LOOP

**BUTTON\_SVC:**

@ Fisrt TURN OFF GPIO2

LDR R0,=0x481AC02C @ Load GPIO2\_IRQSTATUS\_0 register address, text p. 131

MOV R1, #0x00000002 @ Value turns off GPIO2\_1 interrupt request

STR R1, [R0] @ Write to GPIO2\_IRQSTATUS\_0 register

@ Enable UART5

@ Enable THR and MSR interrupt

LDR R2, =0x481AA000 @ Enable Register(IER\_UART)

MOV R3, #0x0A @ Value to tur on bit 1(Enables THR) and bit 3(Enables Modem change interrupt)

STR R3, [R2, #0x04] @ enables UART INT

B PASS\_ON

**TALKER\_SVC:**

LDR R0, =0x481AC02C @ Load address of GPIO2\_IRQSTATUS\_1

MOV R1, #0x00000002 @ Value to turn off GPIO2\_1 interrupt

STR R1, [R0] @ Turn off Button interrupt

LDR R0, =0x481AA000 @ Load UART5 base address

LDR R1, [R0, #0x18] @ Read the MSR

TST R1, #0x10 @ Test bit 4 of the MSR

BEQ MASK\_THR @ IF its Low check if THR needs masking

LDR R1, [R0, #0x14] @ Otherwise read LSR\_UART register

TST R1, #0x20 @ Test bit 5 to see if THR is full or empty

BEQ PASS\_ON @ IF its full bit 5 its indeed low, therefore PASS\_ON

B SEND @ Otherwise branch to send the message

**SEND:**

LDR R1, =CHAR\_PTR @ Load a pointer to char point

LDR R2, [R1] @ Read the string

LDR R3, =CHAR\_COUNT @ Load a pointer to the count store

LDR R4, [R3] @ Read the count value

LDRB R5, [R2], #1 @ Read char value to send

STR R2, [R1] @ Put incremented address back in CHAR\_PTR location

STRB R5, [R0] @ Write char to THR

MOV R2, #0x02 @ Value to turn on UART interrupt

STRB R2, [R0, #0x04] @ Turn on IER\_UART interrupt

SUBS R4, R4, #1 @ Decrement counter by 1

STR R4, [R3] @ Store char counter

BNE PASS\_ON @ IF there are more charaters left go back to PASS\_ON

LDR R2, =MESSAGE @ Otherwise reload the string

STR R2, [R1] @ message back to char pointer

MOV R2, #MSG\_LEN @ Load the original message length

STR R2, [R3] @ Reload length

LDR R0 , = 0x481AA000

MOV R1, #0x00 @ Value to turn off UART interrupt

STRB R1, [R0, #0x04] @ Turn off IER\_UART interrupt

B PASS\_ON

**MASK\_THR:**

LDR R1, [R0, #0x14] @ LSR\_UART register

TST R1, #0x20 @ Test bit 5

BEQ PASS\_ON

@ Disable Disable Tx interrupt by writing 0x08 to 0x481AA004

MOV R1, #0x08 @ Otherwise load value to mask THR interrupt

STRB R1, [R0, #0x04] @ Mask the THR interrupt in IER\_UART

B PASS\_ON @ Return to PASS\_ON

**.data**

**.align** 2

**STACK1:** .rept 1024

**.word** 0x0000

.endr

**STACK2:** .rept 1024

**.word** 0x0000

.endr

**.align** 2

**MESSAGE:**

**.byte** 0xD @ Determine Baud rate and sets internal clock for the frequency needed

**.ascii** "test"

**.byte** 0xD @ Tells the RC8660 to speak the words contained in the ASCII ""

**.align** 2

**CHAR\_PTR:** **.word** MESSAGE

**CHAR\_COUNT:** **.word** 6

.equ MSG\_LEN, 6

.END

**Part\_2**

**Standard Program Structure and Algorithm:**

**Mainline**

* INITIALIZE STACK
* INITIALIZE GPIO1 CLK
  + Load the 0x02 to enable GPIO MODULES
  + Load address 0x44E000AC and write 0x02 to it ( Address of CM\_PER\_GPIO1\_CLKCTRL register)
* @ Not sure if needed ```````````````````````````````````````````````````````````````````````````
* Clear data out from GPIO2\_1 by writing 0x00000002 to GPIO2 address (0x481AC000) + offset (0x190)
* Enable GPIO2\_1 as output by Read, Modify, Write 0xFFFFFFFE to 0x481AC134
* Write 0x1 to Timer4 CFG register at 0x48044010 to reset TIMER4
* Write 0xFFFFC000 to Timer4 TCRR register at 0x4804403C to get 0.5 second
* Write 0XFFFFC000 to Timer4 TLDR register at 0x48044040 to get .5 seconds
* Enable auto reload by writing 0x3 to 0x48044038 (0x3 for enable Auto-reload timer and start the timer, 0x48044038 =0x48044000 (Timer4\_base) + 0x38 (offset of Timer Control Register)).
* Write 0x2 to Timer4 IRQENABLE\_SET register at 0x4804402C

`````````````````````````````````````````````````````````````````````````````````````````````````````````````````````

**Detect falling edge on GPIO2\_1**

* INITIALIZE GPIO2 CLK
  + Copy value to turn on the clock ( 0x02 ) and proceed to turn on GPIO2 clock
  + Load Address of CM\_PER\_GPIO2\_CLKCTRL ( 0x44E000B0 )
* Set up GPIO2\_1 for detecting falling edge by storing 0x00000002 to 0x481AC14C (0x481AC000 (GPIO2\_Base) + 0x14C (offset of GPIO\_FALLINGEDGE\_DETECT)
* Set up GPIO2\_1 for IRQ enable by storing 0x00000002 to 0x481AC034 (0x481AC000 (GPIO2\_Base) + 0x34 (offset of GPIO\_IRQSTATUS\_SET0)

**CONTROL MODULE**

**Initialize UART**

* Change pin 37 on P8 to mode 4 by writing #4 to 0x44E108C0 (0x44E10000 (Control Module Base) + 0x8C0 (offset of pin 37))
* Change pin 38 on P8 to mode 4 by writing #4 to 0x44E108C4 (0x44E10000 (Control Module Base) + 0x8C4 (offset of pin 38))
* Change pin 31 on P8 to mode 6 by writing #6 to 0x44E108D8 (0x44E10000 (Control Module Base) + 0x8D8 (offset of pin 31))
* Change pin 32 on P8 to mode 6 by writing #6 to 0x44E108DC (0x44E10000 (Control Module Base) + 0x8DC (offset of pin 32))

**Set up Baud rate for Transmitting**

* Enable DLL and DLH access by writing 0x83 (config mode A) to 0x481AA00C (0x83 to enable Divisor latch and select 8-bit word length, 0x481AA00C = 0x481AA000 (UART Base) + 0x0C (offset of LCR register))
* SET DLL to 156 and DLH to 78 for a 38.4 Baud rate by writing 0x4E to UART5 Base address (0x481AA000)
* Choose UART 16x Mode by writing 0x00 to 0x481AA020 (0x00 to select UART 16x Mode, 0x481AA020 = 0x481AA000 (UART Base) + 0x20 (offset of MDR1 register)
* Write 0x1A to 0x481AA000 (DLL register) and write 0x00 to 0x481AA004 (DLH register)

**Initialize Tx and Rx Registers**

* Clear bit 7 in LCR register to enable Tx and Rx registers
* Enable THR and MSR interrupt by writing 0x0A to 0x481AA004 = 0x481AA000 (UART Base) + 0x04 (offset of IER\_UART register)

**Turn off FIFO**

* Turn off FIFO by writing 0x06 to 0x481AA008 = 0x481AA000 (UART Base) + 0x00 (offset of FIFO Control Register).
* Sending 0xD on UART THR. This is to avoid the one second delay after pressing button
* Storing 0xD at address 0x481AA000 ( UART Transmit buffer)

**INITIALIZE INT**

* Turn UART clock ON
* Load address of INTC ( 0x48200000 )
* Use 0x02 value to reset the INTC
* Write to INTC\_SYSCONFIG to reset with offset 0x10
* Enable **UART5** interrupt by writing 0x4000 to 0x482000A8 (0x4000 for unmasking INT 46, 0x482000A8 = 0x48200000 (INTC\_Base) + 0xA8 (Offset of INTC\_MIR\_CLEAR1).
* Unmask INTC INT 32 for the **button GPIO2\_1** with value 0x00000001 and the address of INT\_MIR\_CLEAR\_1 register (0x482000A8)
* Unmask INTC INT 92 for **TIMER 4** with value 0x10000000 and the address of INT\_MIR\_CLEAR2 ( 0x482000C8 )

**INITIALIZATION OF TIMER 4**

* Write 0x2 to CM\_PER\_TIMER4\_CLKCTRL at 0x44E00000+0x88
* Set Timer clock frequency multiplexer for 32.768 KHz
* Write 0x02 to PRCMCLKSEL\_TIMER4 register address 0x44E00510
* Initialize timer register for desired count , overflow interrupt generation
* Write 0x1 to Timer4 CFG register at 0x48044010 to reset TIMER4
* Write 0x2 to Timer4 IRQENABLE\_SET register at 0x4804402C
* Write 0XFFFF8000 to Timer4 TLDR register at 0x48044040 to get 1 second
* Write 0xFFFF8000 to Timer4 TCRR register at 0x4804403C to get 1 second

**ENABLE Processor IRQ in CSPR**

* Copy current value in CPSR into a register
* Clear bit 7 of the current CPSR
* Write the modified result back to CPSR ( 8 bit lowest).
* Infinite loop to wait for an Interrupt

**LOOP:** NOP

B LOOP

**INT\_DIRECTOR:**

* Saved uses register and linked register on Stack
* Load GPIO2\_IRQSTATUS\_0 (0x481AC02C)
* Test bit 1 with value (0x00000002) INT32
* If it’s not from button, then check UART (BEQ UART\_CHECK)
* In the case of the button then go BUTTON\_SVC
* Otherwise check if it’s coming from Timer 4 by testing value from address of INTC PENDING IRQ2 register ( 0x482000D8 ) and value 0x4
* If it’s coming from TIMER4, BNE to TIMER\_INT
* Otherwise PASS\_ON (go back to wait loop)

**UART\_CHECK**

* Check if the interrupt come from UART5 by testing bit 14 (0x00004000) (int number 46) of the current value stored in INTC\_PENDING\_IRQ1 (0x482000B8: 0x48200000 (base address for INTC) + 0xB8 offset for INTC\_PENDING\_IRQ1)
* If the interrupt not come from UART branch to PASS\_ON
* Else (interrupt from UART)
* Load address of IIR\_UART5 read data and test bit 0x01 if its set then

Go to TLKR\_SVC (Handshake for sending character)

* Otherwise return to PASS\_ON

**PASS\_ON**

**-** Load Address of INTC\_CONTROL ( 0x48200048 ) and write value 0x01 to reset IRQ

- Restore the registers ( LDMFD SP!, {R0-R5, LR} )

- Pass execution on to wait LOOP for now ( SUBS PC, LR, #4 )

**Button\_SVC**

* The Function of the Button service it’s to enable UAR5 to generate interrupt signals so characters can be sent to the RC 8660 on an interrupt basis.

**Turn off IRQ request from GPIO2\_1**

* Writing 0x00000002 to 0x481AC02C (0x481AC000 (GPIO1\_base) + 0x2C (offset of GPIO1\_IRQ\_STATUS\_SET0))

**Generate new IRQ**

* Writing 0x1 to 0x48200048 (0x48200000 (INTC\_base) + 0x48 (offset of INTC\_CONTROL))

**Enable THR and MSR interrupt**

* Writing 0x0A to 0x481AA004 = 0x481AA000 (UART Base) + 0x04 (offset of IER\_UART register)
* Enable Timer interrupt by writing 0x03 to address 0x48044038
* Turn OFF USR0 led by writing 0x00200000 to 0x4804C(190) Offset of CLEARDATAOUT
* Restore saved registers and return to wait loop

**TLKR\_SVC procedure**

* Save registers
* Turn off GPIO2\_1 interrupt from button
* Check if **CTS signal** is asserted low and THR is empty.
* Read MSR register at 0x481AA018, check bit 4 (0x10)
* If bit 4 (0x10) = 0
* Check THR status by reading LSR register at 0x481AA014, check bit 5 (0x20)
* If THR (LSR bit 5 (0x20) = 0)
  + Restore saved registers and return to mainline
* Else
* Mask THR INT
  + Disable TX interrupt by writing 0x08 to 0x481AA004 (UART Base) + 0x04 (offset of IER\_UART register) to prevent spinning problem

Else

* + Check THR status
  + Read LSR register at 0x481AA014 and check bit 5
  + If bit 5 = 0
    - Restore saved registers and return to mainline
  + Else, go to SEND procedure

**SEND**

* Enable THR and MSR interrupts by writing 0x0A to 0x481AA004 = 0x481AA000 (UART Base) + 0x04 (offset of IER\_UART register)
* Get message store at CHAR\_PTR address
* Get one character in the message and increment pointer by 1
* Compare to check character 0xD
* If Character equals 0xD
* Then go to Check\_TIMER\_interrupt
* Else ( not character 0xD)
* Reload\_timer by Storing 0xFFFF8300 at Timer TCRR register ( make sure 1s delay when it see character 0xD)
* Store the character in the THR register (0x481AA000)
* Get the value store at CHAR\_COUNT address
* Decrement the value by 1 and store it back to CHAR\_COUNT
* Check if it is the last character by checking the current value after decrement
* If the value is less than to zero counter
  + Restore the original message to CHAR\_PTR
  + Restore the original message’s length to CHAR\_COUNT
  + Disable THR interrupt by clearing bit 2 of IER\_UART (0x481AA004)
  + **Turn ON LED** by writing 0x00200000 to 0x4804C194 = 0x4804C000 (GPIO1 Base) + 0x194 (offset of GPIO1\_SET\_DATA\_OUT)
  + Restore saved registers and return to mainline
* Else if the value it bigger or equal to zero go back to mainline and continue sending next character)
* Restore saved registers and return to mainline.

**CHECK\_TIMER**:

* Read INTC\_PENDING\_IRQ2 REGISTER at 0x482000D8, test with 0x4
* If Bit 2 == 0 ( Interrupt does not come from Timer)
  + Restore saved registers
  + Return to mainline
* Else ( Bit 2 ==1)
* Read address of Timer 4 IRQ Status (0x48044028), test with 0x2
* If bit 1 ==0 ( Not overflow)
  + Restore saved registers
  + Return to mainline
* Else ( bit 1==1  Overflow)
  + Go to IRQ\_Timer

**IRQ\_Timer:**

* Turn off IRQ request from overflow
* Writing 0x2 to 0x48044028 (0x48044000 (Timer4\_base) + 0x28 (offset of Timer\_IRQ STATUS))
* **Generate new IRQ generation**

Writing 0x1 to 0x48200048 (0x48200000 (INTC\_base) + 0x48 (offset of INTC\_CONTROL))

* Sending 0xD UART THR
* Storing 0xD at address 0x481AA000 ( UART Transmit buffer)
* Update pointer for next character ( after 0xD)
* Load value from CHAR\_PTR
* Add value with 0x1 and Store back to CHAR\_PTR
* Restore saved registers
* Return to mainline

**Data**

**.data**

**.align** 2

**STACK1:** .rept 1024

**.word** 0x0000

.endr

**STACK2:** .rept 1024

**.word** 0x0000

.endr

**.align** 2

**Data**

**MESSAGE:**

**.ascii** "Ten"

**.byte** 0xD

**.ascii** "Nine"

**.byte** 0xD

**.ascii** "Eight"

**.byte** 0xD

**.ascii** "Seven"

**.byte** 0xD

**.ascii** "Six"

**.byte** 0xD

**.ascii** "Five"

**.byte** 0xD

**.ascii** "Four"

**.byte** 0xD

**.ascii** "Three"

**.byte** 0xD

**.ascii** "Two"

**.byte** 0xD

**.ascii** "One"

**.byte** 0xD

**.ascii** "Zero"

**.byte** 0xD

**.ascii** "blast off"

**CHAR\_PTR:** **.word** MESSAGE

**CHAR\_COUNT:** **.word** 44

.EQU MESSAGE\_LEN,44

High Level Implementation

@ Project 1 part 2ECE 372

@ Josh Pradera

@ After the button connected to GPIO2\_1 its pushed

@ the launcher orally count in intervals of 1 second

@ from 10 t0 0, then says blast off,

@ and send a signal that turns LED 0 to simulate ignition.

**.text**

**.global** \_start

**.global** INT\_DIRECTOR

**\_start:**

.equ MSG\_LEN, 6

@ Load stack data

LDR R13, =STACK1 @ Point to base of SVC stack

ADD R13, R13, #0x1000 @ Move pointer to top of stack

CPS #0x12 @ Switch to IRQ mode

LDR R13, =STACK2 @ Point to base of IRQ stack

ADD R13, R13, #0x1000 @ Move pointer to top of stack

CPS #0x13 @ Switch back to SVC mode

@ Turn On GPIO1 CLK (For User led)

LDR R0,=0x02 @ Value to enable the clock for GPIO MODULES

LDR R1,=0x44E000AC @ Address of CM\_PER\_GPIO1\_CLKCTRL REgister

STR R0,[R1] @ write #02 to register

@ Turn On GPIO2 CLK (For Button)

MOV R2, #0x02 @ Value to turn on clock

LDR R1, =0x44E000B0 @ Address of CM\_PER\_GPIO2\_CLKCTRL

STRB R2, [R1] @ Turn on GPIO2 clock

@ USR LED 0 (GPIO1\_21)

@ Clear USR led0

LDR R0, =0x4804C000 @ Base address of GPIO1 registers

MOV R2, #0x00200000 @ Value to enable bit

STR R2, [R0, #0x190] @ Clear data out

@ Enable GPIO1\_21 as output by read modify write

LDR R0, =0x4804C000 @ Base address for GPIO1 register

ADD R2, R0, #0x0134 @ GPIO\_OE register address

LDR R4, [R2]

MOV R5, #0xFFDFFFFF @ Value to make GPIO1\_21 (USR led 0) as output

AND R4, R5, R4 @ Modify

STR R4, [R2] @ Write to GPIO1 Output enable register

@ Detect falling edge on GPIO2\_1 and enable to assert POINTERPEND1

LDR R0, =0x481AC000 @ Base address of GPIO2 registers

MOV R2, #0x00000002 @ Value to enable bit 1

STR R2, [R0, #0x14C] @ Turn on bit 1 for GPIO2\_FALLINGDETECT

@ Enables GPIO2\_1 as an Interrupt

STR R2, [R0, #0x34] @ Enable GPIO2\_IRQSTATUS\_SET\_0 for bit 1

@ Initialize INTC

LDR R1, =0x48200000 @ Load base address of INTC

MOV R2, #0x02 @ Value to reset the INTC

STRB R2, [R1, #0x10] @ Write to INTC\_SYSCONFIG to reset

@ FOR UART5

MOV R2, #0x4000 @ Value to unmask INTC INT 46 (UART5)

STR R2, [R1, #0xA8] @ Write to INTC\_MIR\_CLEAR1 to unmask

@ FOR BUTTON GPIO2\_1

MOV R2, #0x00000001 @ Value to unmask INTC INT 32, GPIOINT2A (Button GPIO2\_1)

STRB R2, [R1, #0xA8] @ Write to INTC\_MIR\_CLEAR1 to unmask

@ FOR TIMER4

MOV R2, #0x10000000 @ unmask INTC #92, MIR2 bit 28 (TIMER4)

STR R2, [R1, #0xC8] @ Initialize Timer 4

@ Turn on Timer4 clock:

@ Write 0x02 to Base Address of CM\_PER 0x44E00000 base + offset 0x88 for CM\_PER\_TIMER4\_CLKCTRL

MOV R3, #0x2 @ Value to enable TIMER4 CLK

LDR R1,=0x44E00088 @ Address CM\_PER\_TIMER4 CLK

STR R3, [R1] @ Turn on CLK

@Set Timer clock frequency MUX for 32K Hz:

@Write 0x2 to PRCM CLKSEL\_TIMER4 register at address 0x44E00510

LDR R1,=0x44E00510 @ Address of PRCM CLKSEL\_TIMER4 register

STR R3, [R1] @ Select 32K CLK for timer 4, by writing 0x2

@ Initialize Timer 4 registers, with count, overflow interrupt generation

@ Write 0x1 to TIMER4\_CFG at 0x48044010 to reset Timer4

LDR R1,=0x48044000 @ Base address TIMER4 registers

MOV R3, #0x1 @ value to reset TIMER4

STR R3, [R1, #0x10] @ Write to TIMER4 CONFIG register

LDR R1,=0x48044000 @ Base address TIMER4 registers

LDR R3,=0xFFFFC000

STR R3, [R1, #0x3C] @ Write to Timer4 TCRR count register

STR R3, [R1, #0x40] @ Timer4 TLDR load register

@ Enable auto reload by writing 0x3 to 0x48044038

LDR R1,=0x48044000 @ Base address TIMER4 registers

MOV R3, #0x3

STR R3, [R1, #0x38]

@ Enable IRQ overflow interrpt for TIMER4

LDR R1,=0x48044000 @ Base address TIMER4 registers

MOV R3, #0x2 @ Get value to enable Overflow interrupt

STR R3, [R1, #0x2C] @ Write to TIMER4 IRQENABLE\_SET

@ Initialize UART

LDR R1, =0x44E10000 @ Base address of pinout MODE0

MOV R2, #0x04 @ Value to switch to MODE6

STRB R2, [R1, #0x8C0] @ Switch P8-37 to MODE6

STRB R2, [R1, #0x8C4] @ Switch P8-38 to MODE6

MOV R2, #0x06 @ Value to switch to MODE1

STRB R2, [R1, #0x8D8] @ Switch P8-31 to MODE1

STRB R2, [R1, #0x8DC] @ Switch P8-32 to MODE1

MOV R2, #0x02 @ Value to enable UART clock

LDR R1, =0x44E00038 @ Address of CM\_PER\_UART5\_CLKCTRL

STRB R2, [R1] @ Turn on UART5 clock

LDR R1, =0x481AA000 @ Base address of UART5

MOV R2, #0x83 @ Value for UART config mode A

STRB R2, [R1, #0x0C] @ Switch UART5\_LCR to config mode A

MOV R2, #0x4E @ Value to set DLL to 156

STRB R2, [R1] @ DLL value of 0x4E and DHL value of 0x00

@for 38.4 kbps Baud rate

LDR R1, =0x481AA000 @ Base address of UART5

MOV R2, #0x00 @ Value to set MDR to x16

STRB R2, [R1, #0x04] @ Set DLH to 0

STRB R2, [R1, #0x20] @ Switch UART5\_MDR1 to x16 divisor mode

MOV R2, #0x03 @ Value for UART operational mode

STRB R2, [R1, #0x0C] @ Switch UART5\_LCR to operational mode

MOV R2, #0x06 @ Value to disable FIFO

STRB R2, [R1, #0x08] @ Clear and disable UART5 FIFO

@ Sending 0xD on UART THR to avoid one second delay after button push

LDR R1, =0x481AA000 @ Base address of UART5

MOV R2, #0xD

STR R2, [R1]

@ Enable processor IRQ in CSPR

MRS R3, CPSR @ Copy CPSR to R3

BIC R3, #0x80 @ Clear bit 7

MSR CPSR\_c, R3 @ Write back to CPSR

@ Wait for an interrupt

**LOOP:** NOP

B LOOP

**INT\_DIRECTOR:**

STMFD SP!, {R0-R5, LR} @ Push registers onto stack

@@@@ NEW PROCEDURE

@ Check if the interrupt comes from GPIO-MIR\_CLEAR1 by testing bit 0

LDR R0,=0x482000B8 @ Address of INTC\_PENDING\_IRQ1

LDR R1, [R0] @ Read INTC\_PENDING\_IRQ1 register

TST R1, #0x00000001 @ test bit 0 of IRQ 1

@ IF its NOT an INterrupt from GPIO go check if its from UART

BEQ UART\_CHECK @ IF is not coming from button, check UART!

@ But if the interrupt is from GPIO then Check if the button was pushed!

LDR R0,=0x481AC02C @ Load GPIO2\_IRQSTATUS\_0 register address

LDR R1, [R0] @ READ status register, to see if button pushed

TST R1, #0x00000002 @ Check if bit 1 = 1

BNE BUTTON\_SVC @ If bit 1 = 1, then button pushed!

@ At this point its coming from TIMER4

/\*

LDR R0,=0x48200048 @ Else, go back address of INTC\_PENDING\_IRQ2 register TIMER4

MOV R3, #01 @ value to clear bit 0

STR R3, [R0] @ Write to INTC\_CONTROL register.

\*/

**TIMER\_CHECK:**

LDR R3,=0x482000D8 @ Address of INTC PENDING IRQ2 register

LDR R0, [R3] @ Get value

TST R0, #0x10000000 @ Check if interrupt is from TIMER4

BNE TIMER\_INT @ IF INT comes from timer4 branch and continue

**UART\_CHECK:**

LDR R0, =0x482000B8 @ Load address of INTC\_PENDING\_IRQ1

LDR R1, [R0] @ Read the data

TST R1, #0x00004000 @ Test bit 14 (INT 46 UART)

/\*

AND R1, R2 @Mask bit 14

CMP R1, #0x4000

\*/

BEQ PASS\_ON @ IF is not from UART THEN PASS\_ON

LDR R0, =0x481AA008 @ Otherwise Load address of IIR\_UART5

LDR R1, [R0] @ Read the data

TST R1, #0x01 @ Check bit 0

@ TaLKER\_SVC handshake for sending characters

BEQ TALKER\_SVC @ IF bit 0 its low then branch to TALKER\_SCV

B PASS\_ON @ Otherwise PASS\_ON

**PASS\_ON:**

LDR R0, =0x48200048 @ Address of INTC\_CONTROL

MOV R2, #0x01 @ Value to reset IRQ

STR R2, [R0] @ Reset IRQ and also listen for a new one

LDMFD SP!, {R0-R5, LR} @ Restore registers

SUBS PC, LR, #0x04 @ Pass execution to beginning of LOOP

**BUTTON\_SVC:**

@ Fisrt TURN OFF IRQ request from GPIO2\_1

LDR R0,=0x481AC02C @ Load GPIO2\_IRQSTATUS\_0 register address, text p. 131

MOV R1, #0x00000002 @ Value turns off GPIO2\_1 interrupt request

STR R1, [R0] @ Write to GPIO2\_IRQSTATUS\_0 register

/\*

• Generate new IRQ generation

Writing 0x1 to 0x48200048 (0x48200000 (INTC\_base) + 0x48 (offset of INTC\_CONTROL))

@ Generate new IRQ

LDR R3,=0x48200048

MOV R2, #0x1

STR R2, [R3]

\*/

@ Enable THR and MSR interrupt

LDR R2, =0x481AA000 @ Enable Register(IER\_UART)

MOV R3, #0x0A @ Value to tur on bit 1(Enables THR) and bit 3(Enables Modem change interrupt)

STR R3, [R2, #0x04] @ enables UART INT

@ Enable TIMER4 interrupt

MOV R3, #0x03 @ load value of auto reload timer and start

LDR R1,=0x48044038

STR R3, [R1]

@ Turn OFF USER LED0 (GPIO1\_21)

LDR R0,=0x4804C000 @ LOAD ADDRESS OF GPIO1

MOV R1, #0x00200000 @ Get word to turn OFF USER LED0

STR R1,[R0, #0x190] @ Load Address to CLEARDATAOUT

B PASS\_ON

**TALKER\_SVC:**

STMFD SP!, {R0-R5, LR} @ Push registers onto stack

LDR R0, =0x481AC02C @ Load address of GPIO2\_IRQSTATUS\_1

MOV R1, #0x00000002 @ Value to turn off GPIO2\_1 interrupt

STR R1, [R0] @ Turn off Button interrupt

LDR R0, =0x481AA000 @ Load UART5 base address

LDR R1, [R0, #0x18] @ Read the MSR

TST R1, #0x10 @ Test bit 4 of the MSR

BEQ MASK\_THR @ IF its Low check if THR needs masking

LDR R1, [R0, #0x14] @ Otherwise read LSR\_UART register

TST R1, #0x20 @ Test bit 5 to see if THR is full or empty

BEQ PASS\_ON @ IF its full bit 5 its indeed low, therefore PASS\_ON

B SEND @ Otherwise branch to send the message

**SEND:**

@ Enable THR and MSR interrupts @@@@@@@@@@@@@@

LDR R2, =0x481AA000 @ Enable Register(IER\_UART)

MOV R3, #0x0A @ Value to tur on bit 1(Enables THR) and bit 3(Enables Modem change interrupt)

STR R3, [R2, #0x04] @ enables UART INT

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@ Get message store at CHAR\_PTR address

LDR R1, =CHAR\_PTR @ Load a pointer to char point

LDR R2, [R1] @ Read the string

LDR R3, =CHAR\_COUNT @ Load a pointer to the count store

LDR R4, [R3] @ Read the count value

LDRB R5, [R2], #1 @ Read char value to send

@ Compare to check character 0xD

TST R5, #0xD @ test if the first character is 0xD

BNE TIMER\_INT @ If Character equals then go and check Timer interrupt

@ Otherwise NOT 0xD then reload TIMER by storing 0xFFFF8300 at timer TCRR register

STR R2, [R1] @ Put incremented address back in CHAR\_PTR location

STRB R5, [R0] @ Write char to THR

MOV R2, #0x02 @ Value to turn on UART interrupt

STRB R2, [R0, #0x04] @ Turn on IER\_UART interrupt

SUBS R4, R4, #1 @ Decrement counter by 1

STR R4, [R3] @ Store char counter

BNE PASS\_ON @ IF there are more charaters left go back to PASS\_ON

LDR R2, =MESSAGE @ Otherwise reload the string

STR R2, [R1] @ message back to char pointer

MOV R2, #MSG\_LEN @ Load the original message length

STR R2, [R3] @ Reload length

@ Disable THR Interrupt

LDR R0 , = 0x481AA000

MOV R1, #0x00 @ Value to turn off UART interrupt

STRB R1, [R0, #0x04] @ Turn off IER\_UART interrupt

@ Turn On LED

LDR R0,=0x4804C000 @ LOAD ADDRESS OF GPIO1

MOV R1, #0x00200000 @ Get word to turn OFF USER LED0

STR R1,[R0, #0x194] @ Load Address to SETDATAOUT

B PASS\_ON

**MASK\_THR:**

LDR R1, [R0, #0x14] @ LSR\_UART register

TST R1, #0x20 @ Test bit 5

BEQ PASS\_ON

@ Disable Disable Tx interrupt by writing 0x08 to 0x481AA004

MOV R1, #0x08 @ Otherwise load value to mask THR interrupt

STRB R1, [R0, #0x04] @ Mask the THR interrupt in IER\_UART

B PASS\_ON @ Return to PASS\_ON

**TIMER\_INT:**

LDR R3,=0x482000D8 @ Address of INTC PENDING IRQ2 register

LDR R0, [R3] @ Get value

TST R0, #0x10000000 @ Check if interrupt is from TIMER4

@ If not TIMER4 interrupt,Branch to PASS\_ON

BNE PASS\_ON

@ If TIMER4 interrupt check

LDR R3,=0x48044028 @ Address of INTC PENDING IRQ2 register

LDR R0, [R3] @ Get value

TST R0, #0x2 @ Check with bit 1

BEQ PASS\_ON @ IF bit 1 is equal to zero (Not Overflow) PASS\_ON

@ Otherwise bit 1 i equal to 1 and we have overflow

B IRQ\_TIMER

**IRQ\_TIMER:**

@ Turn OFF TIMER4 IRQ request from overflow

LDR R3,=0x48044028 @ Address of INTC PENDING IRQ2 register

MOV R2, #0x2 @ value to turn OFF

STR R2, [R3]

@ Generate new IRQ

LDR R3,=0x48200048

MOV R2, #0x1

STR R2, [R3]

@ SEnding 0xD UART THR

LDR R2, =0x481AA000 @ Enable Register(IER\_UART)

MOV R3, #0xD

STR R3, [R2]

@Update Pointer for next char after 0xD

LDR R1, =CHAR\_PTR @ Load a pointer to char point

LDR R2, [R1] @ Read the string

LDRB R5, [R2], #1 @ Read value and add 1

STR R2, [R5] @ Store back to char pointer

@ Return to mainline

B PASS\_ON

**.data**

**.align** 2

**STACK1:** .rept 1024

**.word** 0x0000

.endr

**STACK2:** .rept 1024

**.word** 0x0000

.endr

**.align** 2

**MESSAGE:**

**.byte** 0xD

**.ascii** "Ten"

**.byte** 0xD

**.ascii** "Nine"

**.byte** 0xD

**.ascii** "Eight"

**.byte** 0xD

**.ascii** "Seven"

**.byte** 0xD

**.ascii** "Six"

**.byte** 0xD

**.ascii** "Five"

**.byte** 0xD

**.ascii** "Four"

**.byte** 0xD

**.ascii** "Three"

**.byte** 0xD

**.ascii** "Two"

**.byte** 0xD

**.ascii** "One"

**.byte** 0xD

**.ascii** "Zero"

**.byte** 0xD

**.ascii** "BLastoff"

**.align** 2

**CHAR\_PTR:** **.word** MESSAGE

**CHAR\_COUNT:** **.word** 44

.EQU MESSAGE\_LEN,44

.END