Lab 3 Report - Serial

Dr. Ryan Gerdes

ECE 3710

By: Jonathan Tousley

Partner: Aaron Kunz

**Overview:**

The microcontroller was configured to send data to and from the PC using UART via RS-232. Upon input from the microcontroller or the PC, a message is sent to the PC to display. For hardware, this project requires

- A 5V voltage source (DC)

- A UART to RS-232 module

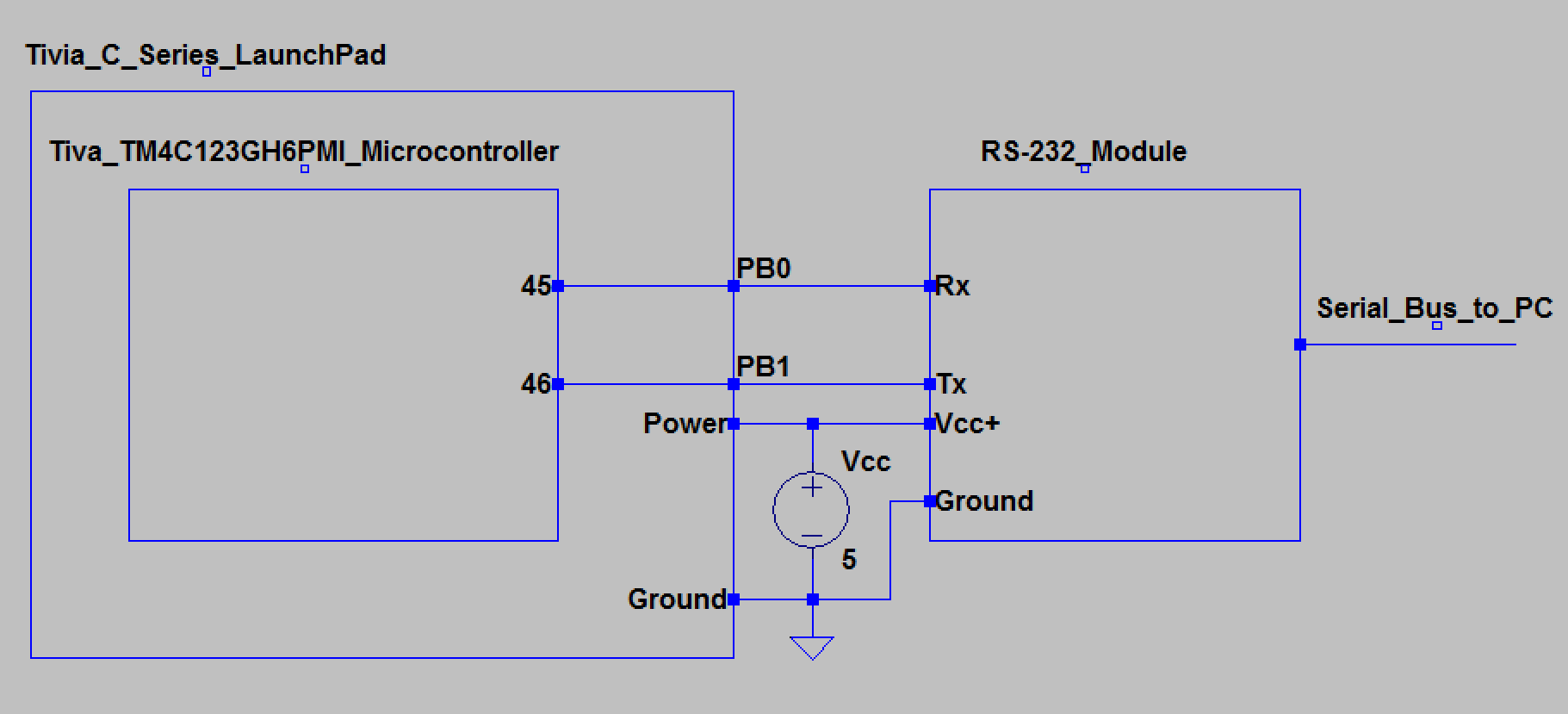
- A serial cable

- The microcontroller

- A 3.3V voltage source (optional, but recommended)

**Hardware Details:**

Using UART module 1 and the alternative function of GPIO pins PB[0:1], the microcontroller is connected to the RS-232 module. The microcontroller is powered by the 5V voltage source and the RS-232 module is powered by the 3.3V voltage source (alternatively, this could be powered from the microcontroller or the same 5V source). The RS-232 module is in turn connected to the PC via a serial cable. See schematic below:



**Software Details:**

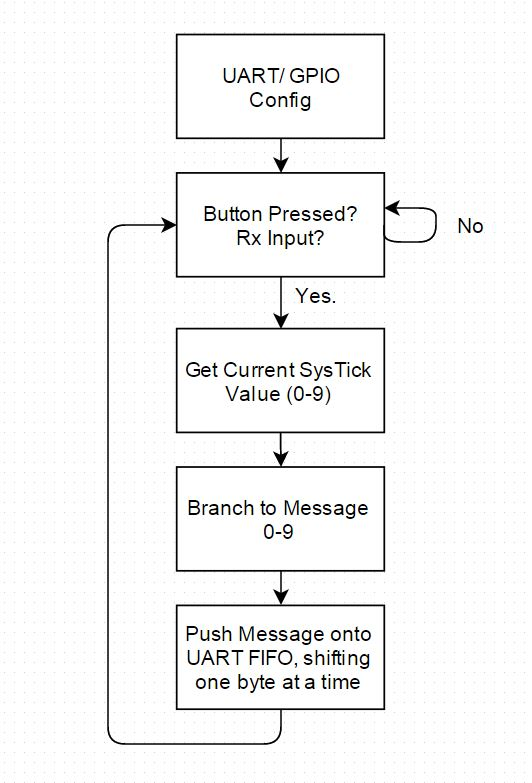
The microcontroller is configured to await input from the user in either of two forms, a button on the microcontroller (PF0) or data from the PC via the UART module (Rx – PB0). Upon either input, the microcontroller sends one of ten messages to the PC. To prevent multiple messages being sent upon a single input, the microcontroller first checks to see if the input is default, and then awaits the user. This was a very concise and efficient method. The not concise part was sending ten different hexadecimal strings, and resulted in the program being the absolute maximum size allowed.

*Randomness:*

To arbitrarily choose between any of ten strings, Systick was utilized to always be running and to count down from ten. When a value was needed, Systick was sampled and that value determined which string to send to the PC.

*Switch Statements:*

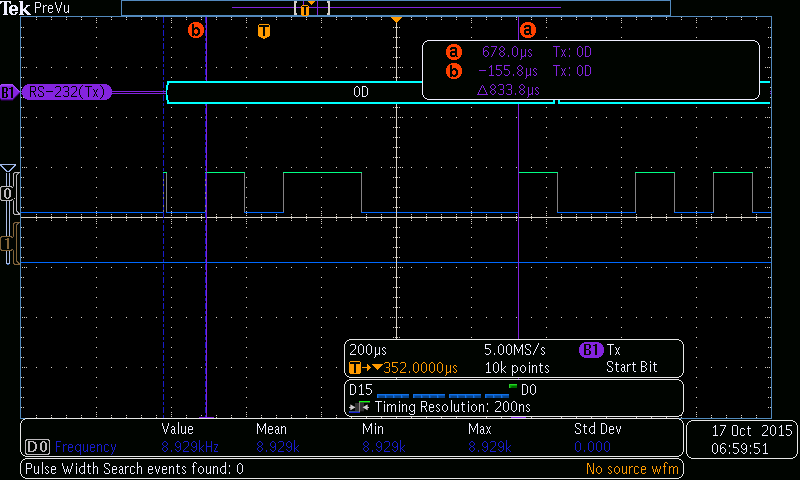
Gathering a random value is one thing, it is quite another to execute entirely different code based on that value. Rather than a very long if-else statement, an effective jump table was used to decide where to go. To accomplish this, the Program Counter was read, and added to the Systick value multiplied by 2 (the length of each instruction, in this case), with a little bit accounted for offsets and the like. This value was then loaded into the link register, and then a simple “BX LR” (return to value in link register) accomplished a jump to a list of branches to appropriate functions. A misuse of the link register to be sure, but it worked perfectly. Code is given at the end. Here is a flowchart of the software logic:

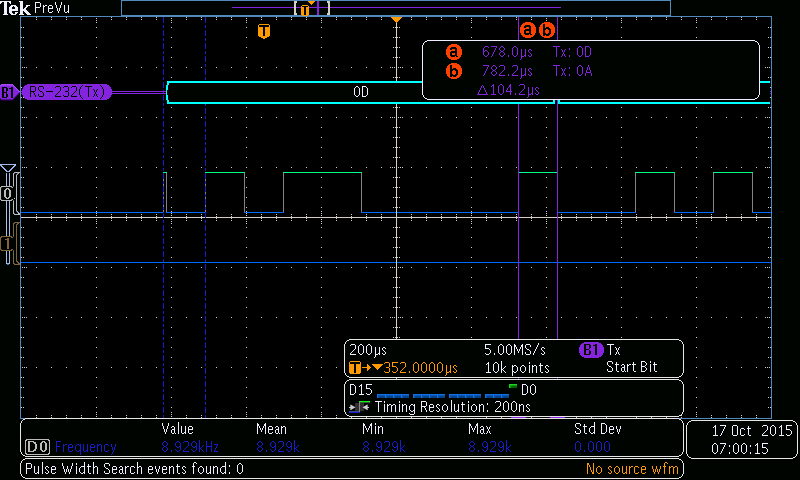


**UART:**

The UART module was configured to use both trnasmit and receive simultaneously using the FIFO buffer. For a 9600 baud rate, the BRDI was set to 104 (16MHz / (16 \* 9600) = 104.166667) and the BRDF was set to 11 (.16667 \* 64 + .5 = 11.16667). Using this configuration with GPIO port B, the data was sent and received to and from the PC. Screenshots from the logic analyzer are given below:

This screenshot confirms the length of one byte of data – 833.8 microseconds:



This screenshot confirms the length of the start bit – 104.2 microseconds:

**Messages:**

Depending on the random value generated, one of ten messages is sent to the PC. The possible messages are given below:

-Nope 

-You are doomed

-Concentrate you fool

-What a rubbish question

-Only in your dreams

-Yes now leave me alone

-Heh you wish

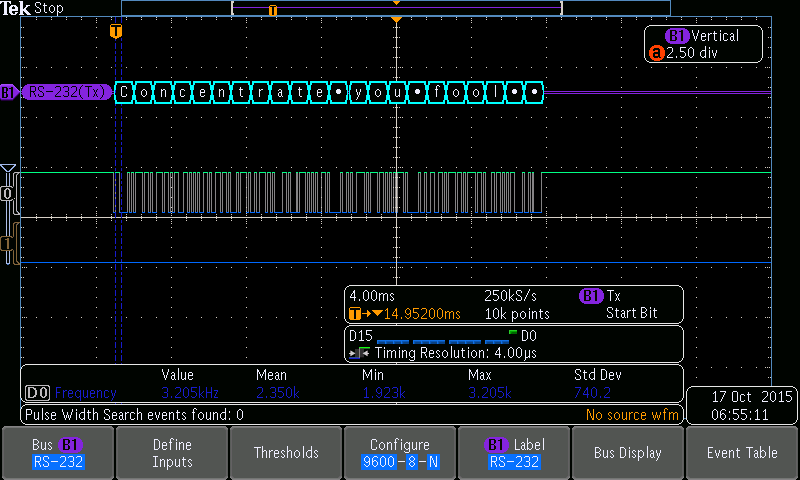
-Oh yeah that will happen

-Stop bothering me

-Not if you were the last person on earth

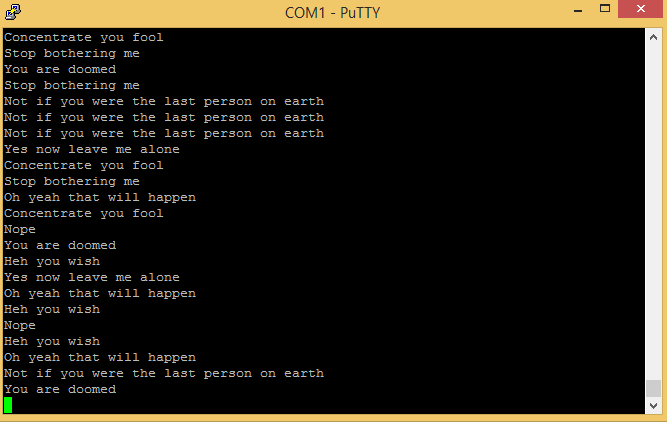
Although the program is fully capable of generating any of these ten strings as verified by the logic analyzer, only the one given below is provided for this report.

This screenshot verifies the output of the data as being one of the messages:

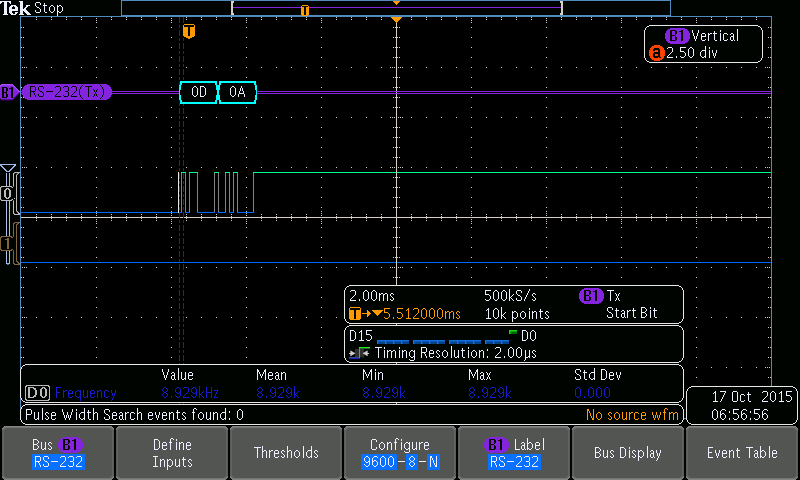


**User Input:**

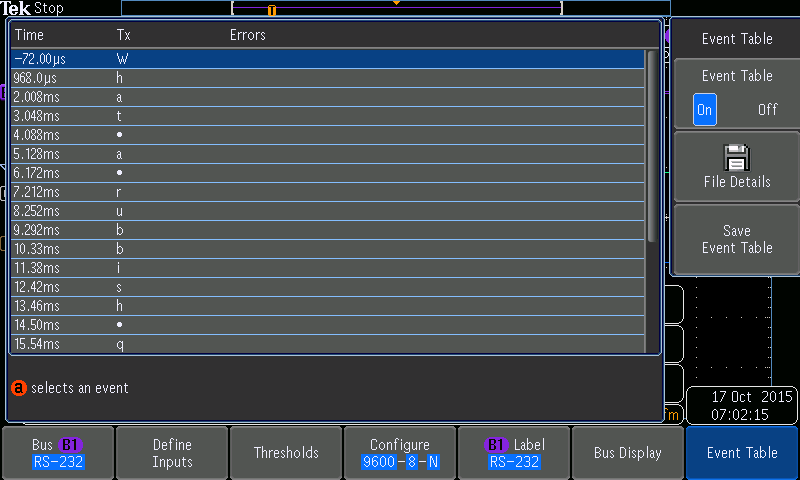
The microcontroller reliably responded to both types of user input and sent appropriate messages to the screen. Given restrictions on the software of the PC used to interface with the microcontroller, the user input does not appear on the screen, only the messages. Below is a screenshot of the PC displaying the appropriate messages.



Also, the requirements dictated that a carriage return and a new line be transmitted at the beginning of the program, so here is a screenshot of that happening:



And here is a screenshot of the event table after a message was sent:



The rest of this document only contains code. Here it is:

; main.s

THUMB

AREA |.text|, CODE, READONLY, ALIGN=2

EXPORT Start

; timer vars

UART1 EQU 0x4000D000 ;UART1 base

UART2 EQU 0x4000E000 ;UART2 base

UART3 EQU 0x4000F000 ; UART3 base

PA EQU 0x40004000 ;GPIO Port A (APB): 0x4000.4000

PB EQU 0x40005000 ;GPIO Port B (APB): 0x4000.5000

PCBASE EQU 0x40006000 ;GPIO Port C (APB): 0x4000.6000

PD EQU 0x40007000 ;GPIO Port D (APB): 0x4000.7000

PE EQU 0x40024000 ;GPIO Port E (APB): 0x4002.4000

PF EQU 0x40025000 ; GPIO Port F (APB): 0x4002.5000

;RCGC2 EQU 0x400FE608 ; GPIO clock

ST EQU 0xE000E000 ; systick timer base address

TM0 EQU 0x40030000 ; GPTM0

TM1 EQU 0x40031000 ; GPTM1

TM2 EQU 0x40032000 ; GPTM2

RCGC EQU 0x400FE000 ; Timer clock

Start

ldr R1,=RCGC

; 1. enable clock: uart then port

mov R0,#0x2

str R0,[R1,#0x618] ;uart1 ;Note 0x104 is for Legacy suppory only 618

MOV R0, #0x22

str R0,[R1,#0x608] ;portb, portF ;Note, 0x108 for Legacy support only 608

nop

nop

; 2. PD enable alt. func. and pin

ldr R1,=PB

; MOV32 R0, #0x4C4F434B ; GPIO Unlock code.

; STR R0, [R1,#0x520] ; unlock GPIOD\_LOCK

; MOV R0, #0x80

; STR R0, [R1,#0x524] ; GPIOCR unlock pin PD7

mov R0,#0xFF

str R0,[R1,#0x420] ;AFSEL

str R0,[R1,#0x51C] ;DEN

STR R0, [R1, #0x514] ;pull down PD[6:7]

MOV R0, #0x11

STR R0, [R1, #0x52C] ; GPIOPCTL in conjunction with GPIOAFSEL offset 0x52C

; STR R0, [R1, #0x400] ; PD[6:7] OUTPUT

;Config port F for sw2

LDR R1, =PF

MOV32 R0, #0x4C4F434B ; GPIO Unlock code.

STR R0, [R1,#0x520] ; unlock GPIOF\_LOCK

MOV R0, #0x00 ; set PF as input

STR R0, [R1, #0x400]

MOV R0, #0x01

STR R0, [R1,#0x524] ; GPIOCR unlock pin PF0

MOV R0, #0x1 ; set pull up PF0

STR R0, [R1, #0x510]

; enable PF0

STR R0, [R1, #0x51C] ; GPIODEN

; 3. disable uart1

ldr R1,=UART1

mov R0,#0x0

str R0,[R1,#0x30]

; 4. set baudrate divisor

; BRD = 16e6/(16\*9600)= 104.16

; integer portion: int(104.16)=104

mov R0,#0x68

str R0,[R1,#0x24]

; fractional portion: int(0.16\*2^6+0.5)=11

mov R0,#0xB

str R0,[R1,#0x28]

; 5. set serial parameters: No pairity, enable fifo, WL = 1 byte

mov R0,#0x70 ;0b01110000

str R0,[R1,#0x2C]

; 6. enable tx, rx, and uart

mov R0,#0x301 ;0b0110000001

str R0,[R1,#0x30]

;three delays before any uart registers accessed

MOV R0, #0x0d0a

BL writeRZeroToUARTTwoByte

LDR R1, =ST

;disable systick

MOV R2, #0x1 ; disable SYSTICK

STR R2, [R1, #0x10] ; SYSTICK Control STCTRL offset 0x010

;enable systick, load 9

MOV R0, #0x9

STR R0, [R1, #0x14] ; SYSTICK Reload STRELOAD

;Start SysTick

MOV R0, #0x5

STR R0, [R1, #0x10] ; SYSTICK Control STCTRL offset 0x010

;Note: offset 0x18 holds SysTick current value

Begin

LDR R1, =PF

LDR R0, [R1, #0x3FC]

CMP R0, #0x0

BEQ Begin

LDR R1,=UART1

LDR R0,[R1,#0x018] ; Read UART flag register

ANDS R0,#0x10

BEQ Begin ; If receiver is NOT empty, Send

delay1

LDR R1, =PF

;Check for sw2 press (check PF0)

LDR R0,[R1,#0x3FC] ; If sw2 pressed, R0 == 0

CMP R0,#0x0

BEQ Send ; If Button pressed, Send

;Check for request from Computer (check Rx of UART1, PB1)

LDR R1,=UART1

LDR R0,[R1,#0x018] ; Read UART flag register

ANDS R0,#0x10

BNE delay1; If receiver is NOT empty, Send

Send

LDR R1,=ST

LDR R0,[R1,#0x18] ; Read SysTick (a 0-9 value)

LDR R1, =UART1

BL waitLoop

MOV R4, #0x2

MUL R6, R4, R0

MOV R4, PC

ADD R4, R6, R4

ADD R2, R4, #0x9

MOV LR, R2

BX LR

B message0

B message1

B message2

B message3

B message4

B message5

B message6

B message7

B message8

B message9

message0

MOV32 R0, #0x4e6f7065

BL writeRZeroToUART

MOV R0, #0x0d0a

BL writeRZeroToUARTTwoByte

B Begin

message1

MOV32 R0, #0x596f7520

BL writeRZeroToUART

MOV32 R0, #0x61726520

BL writeRZeroToUART

MOV32 R0, #0x646f6f6d

BL writeRZeroToUART

MOV32 R0, #0x65640d0a

BL writeRZeroToUART

B Begin

message2

MOV32 R0, #0x436f6e63

BL writeRZeroToUART

MOV32 R0, #0x656e7472

BL writeRZeroToUART

MOV32 R0, #0x61746520

BL writeRZeroToUART

MOV32 R0, #0x796f7520

BL writeRZeroToUART

BL waitLoop

MOV32 R0, #0x666f6f6c

BL writeRZeroToUART

MOV R0, #0x0d0a

BL writeRZeroToUARTTwoByte

B Begin

message3

MOV32 R0, #0x57686174

BL writeRZeroToUART

MOV32 R0, #0x20612072

BL writeRZeroToUART

MOV32 R0, #0x75626269

BL writeRZeroToUART

MOV32 R0, #0x73682071

BL writeRZeroToUART

BL waitLoop

MOV32 R0, #0x75657374

BL writeRZeroToUART

MOV32 R0, #0x696f6e0d

BL writeRZeroToUART

MOV R0, #0x0a

STR R0, [R1]

B Begin

message4

MOV32 R0, #0x4f6e6c79

BL writeRZeroToUART

MOV32 R0, #0x20696e20

BL writeRZeroToUART

MOV32 R0, #0x796f7572

BL writeRZeroToUART

MOV32 R0, #0x20647265

BL writeRZeroToUART

BL waitLoop

MOV32 R0, #0x616d730d

BL writeRZeroToUART

MOV R0, #0x0a

STR R0, [R1]

B Begin

message5

MOV32 R0, #0x59657320

BL writeRZeroToUART

MOV32 R0, #0x6e6f7720

BL writeRZeroToUART

MOV32 R0, #0x6c656176

BL writeRZeroToUART

MOV32 R0, #0x65206d65

BL writeRZeroToUART

BL waitLoop

MOV32 R0, #0x20616c6f

BL writeRZeroToUART

MOV32 R0, #0x6e650d0a

BL writeRZeroToUART

B Begin

message6

MOV32 R0, #0x48656820

BL writeRZeroToUART

MOV32 R0, #0x796f7520

BL writeRZeroToUART

MOV32 R0, #0x77697368

BL writeRZeroToUART

MOV R0, #0x0d0a

BL writeRZeroToUARTTwoByte

B Begin

message7

MOV32 R0, #0x4f682079

BL writeRZeroToUART

MOV32 R0, #0x65616820

BL writeRZeroToUART

MOV32 R0, #0x74686174

BL writeRZeroToUART

MOV32 R0, #0x2077696c

BL writeRZeroToUART

BL waitLoop

MOV32 R0, #0x6c206861

BL writeRZeroToUART

MOV32 R0, #0x7070656e

BL writeRZeroToUART

MOV R0, #0x0d0a

BL writeRZeroToUARTTwoByte

B Begin

message8

MOV32 R0, #0x53746f70

BL writeRZeroToUART

MOV32 R0, #0x20626f74

BL writeRZeroToUART

MOV32 R0, #0x68657269

BL writeRZeroToUART

MOV32 R0, #0x6e67206d

BL writeRZeroToUART

BL waitLoop

MOV R0, #0x65

STR R0, [R1]

MOV R0, #0x0d0a

ROR R0, #8

STR R0, [R1]

ROR R0, #24

STR R0, [R1]

B Begin

message9

MOV32 R0, #0x4e6f7420

BL writeRZeroToUART

MOV32 R0, #0x69662079

BL writeRZeroToUART

MOV32 R0, #0x6f752077

BL writeRZeroToUART

MOV32 R0, #0x65726520

BL writeRZeroToUART

BL waitLoop

MOV32 R0, #0x74686520

BL writeRZeroToUART

MOV32 R0, #0x6c617374

BL writeRZeroToUART

MOV32 R0, #0x20706572

BL writeRZeroToUART

BL waitLoop

MOV32 R0, #0x736f6e20

BL writeRZeroToUART

MOV32 R0, #0x6f6e2065

BL writeRZeroToUART

MOV32 R0, #0x61727468

BL writeRZeroToUART

MOV32 R0, #0x0d0a

BL writeRZeroToUARTTwoByte

B Begin

writeRZeroToUARTTwoByte

ROR R0, #8

STR R0, [R1]

ROR R0, #24

STR R0, [R1]

BX LR

; assumes R1 is set to base address of UART

writeRZeroToUART

ROR R0, #24

STR R0, [R1]

ROR R0, #24

STR R0, [R1]

ROR R0, #24

STR R0, [R1]

ROR R0, #24

STR R0, [R1]

BX LR

waitLoop

ldr R2,[R1,#0x18]

ands R2,#0x80 ;0b100000 (set Z=1 if result is 0)

BEQ waitLoop

BX LR

ALIGN

END