Simple Shell for MSP430 UART

Rob Murrer

November 22, 2013

User Manual

The Simple Shell for MSP430 UART (SSMU) provides a serial interface to the Texas Instruments (TI) MSP-EXP430G2 Experimenter Board. This interface allows the memory to be displayed or modified and the Arithmetic Logic Unit's (ALU) addition or subtraction can be executed.

Requirements

- Basic computer knowledge of opening and running programs
- TI MSP-EXP430G2 Experimenter Board (TIEB) with the SSMU program loaded
- Personal Computer with free USB port
- Terminal program with the following configuration: 9600 baud, 8 Data bits, 1 Stop bit, No Parity.
- USB cable with micro-usb connection for connection with the TIEB

Getting Started

1. Connect the TIEB to the computer with the micro-usb cable.
2. Determine which Com port is being used by the TIEB. In MS Windows this is done by going to the Control Panel then to Device Manager and then to Ports. Look for "MSP430 Application UART (COMx) where 'x' would be 1-6 normally.
3. Open up Terminal program and setup a connection to the TIEB with the Com Port determined in previous step and the configuration of 9600 baud, 8 Data bits, 1 Stop bit, No Parity
4. Type `HS FFFF FFFF` the result should be `0000` along with the status bits underneath.
5. The command prompt of `>` should be displayed and is waiting on the next command.

Addition and Subtraction

The SSMU can be used as a simple calculator performing both addition and subtraction. The commands `HA` and `HS` are used for addition and subtraction. These commands take two 4 digit arguments. The arguments must be valid hexadecimal numbers such as `0FEE` or `AADD`.

The results of the addition and subtraction are followed by a line of additional information that can aid in making sense of the answer returned. The flags are as follows:

```
V=x N=x Z=x C=x
```

The value of x can be `0` or `1`. Zero means the flag is not set and One means it is set.

- `V` is for overflow. The sign of the value returned is incorrect. (ie: positive + positive = negative)

- `N` is for negative. The value returned is a negative number.
- `Z` is for zero. The value returned is zero.
- `C` is for carry. During the operation the carry flag was set.

Examples

```
> HA 0000 0001 = 0001
V=0 N=0 Z=0 C=0

> HA FFFF FFFF = FFFE
V=0 N=1 Z=0 C=1

> HS 2525 3535 = EFF0
V=0 N=1 Z=0 C=0

> HS FFFF FFFF = 0000
V=0 N=0 Z=1 C=1
```

Displaying Values in Memory

The SSMU can be used to view the data in memory. The data is displayed 8 words per line and displayed with the address and data along with the ASCII representation–if it is printable.

The command to display memory is `D XXXX YYYY` where `XXXX` is the hexadecimal start address and `YYYY` is the hexadecimal end address. The display is always 8 words wide and the end address is non-inclusive.

Example

```
> D
0200 0280
0200 0203 0202 0910 0204 C4A6 0206 0072 0208 7480 020A 0420 020C 1614 020E 7600 ......r..t.....v
0210 0594 0212 8104 0214 D451 0216 1020 0218 81E1 021A 1B0E 021C B888 021E 0C00 ....Q...........
0220 02B8 0222 0404 0224 0964 0226 6042 0228 0604 022A 8124 022C 4825 022E 2001 ....d.B`..$.%H..
0230 1096 0232 1102 0234 14C4 0236 1000 0238 90A1 023A 2082 023C 86A1 023E 0490 ................
0240 BEFF 0242 FFF7 0244 7E76 0246 BFFF 0248 5BC4 024A 1FF3 024C DD8D 024E 8D5E ....v~...[....^.
0250 7BEB 0252 5ADD 0254 78FC 0256 72BE 0258 3FF7 025A DFFD 025C FDD7 025E 5C9D .{.Z.x.r.?.....\
0260 DFFE 0262 7FFE 0264 7F16 0266 F955 0268 EDFE 026A FEFD 026C 8FF7 026E F5BF ......U.........
0270 FCE7 0272 DFBE 0274 C9FD 0276 FFEF 0278 7E32 027A C7FF 027C FFAA 027E 9DD1 ........2~......
```

Modifying Contents of Memory

The SSMU can be used to modify the data in memory. To enter this mode of operation the user must type `M XXXX` where `XXXX` is the address to be modified. Once the user is in this mode the SSMU will display `AAAA VVVV` where `AAAA` is the address and `VVVV` is the current hexadecimal representation of the data in in that address. The user is now prompted to enter in a new data value to be inserted into that location.

To exit the modify memory mode the user must type enter in a `space`. If the user enters in a valid hex value the SSMU will insert the data and then go to the next word in memory to be modified.

To navigate the memory the user may enter in an `N` or a `P` at any time in the modify memory mode.

Commands

- `space` : exit modify memory mode
- `N` : go backwards to the previous word
- `P` : go forwards to the next word in memory

Examples

```
> M 0300
0300 084A FFFF
0302 4918 FF11
0304 10C0 N
0302 FF11 N
0300 FFFF

> M 0300
0300 FFFF 22P
0302 FF11 33N
0300 FFFF 3333
0302 FF11 N
0300 3333
```

Test procedure

To validate that the Simple Shell for MSP430 UART (SSMU) for the Texas Instruments MSP-EXP430G2 Experimenter Board (TIEB) works correctly the following can be used.

Requirements

- Personal Computer with Microsoft Windows 7 and a free USB port
- Code Composer Studio (CCS) 5.3
- Project files and source for SSMU
- TIEB and micro-usb cable
- Putty 0.63 Terminal Program

Setup

1. Turn on computer and wait for operating system to fully load.
2. Plug USB cable into computer and TIEB.
3. Start the CCS application from the `Start Menu`.
4. From the `File` menu select `Import...`
5. Select the folder that contains the Project files for the SSMU.
6. Select `Code Composer Studio` then `Existing CCS Eclipse Projects`.
7. Browse for project folder and click finish.
8. From the `View` menu select `Project Explorer`.
9. Click the project and it will become the active project.
10. Press the `F11` Key and the project will compile and load.
11. Wait until the compilation is finished and press the `F8` key.
12. Press the `Windows` key and the `Pause/Break` key at the same time.
13. Click on the `Device Manager` link.

14. Select the `Ports` item.
15. Look for the `MSP430 Application UART` and record the Com Port.
16. Start the Putty Application.
17. In the Putty Settings set the `connection type` to `serial`.
18. Enter the `Serial line` as `COMx` where x is the value found in previously.
19. Set `Speed` to `9600`.
20. The defaults in Putty are adequate: `Data bits`: 8, `Stop bit`: 1, `Parity`: None.
21. The SSMU is now ready to be tested.

Basic Addition

Input

```
HA 0300 0500
```

Output

```
> HA 0300 0500 = 0800
V=0 N=0 Z=0 C=0
```

Basic Subtraction

Input

```
HA 0500 0300
```

Output

```
> HS 0500 0300 = 0200
V=0 N=0 Z=0 C=1
```

Zero Flag

Input

```
HS 0300 0300
```

Output

```
> HS 0300 0300 = 0000
V=0 N=0 Z=1 C=1
```

Negative Flag

Input

```
HS 0300 0400
```

Output

```
> HS 0300 0400 = FF00
V=0 N=1 Z=0 C=0
```

Overflow Flag

Input

```
HS 9999 9999
```

Output

```
> HA 9999 9999 = 3332
V=1 N=0 Z=0 C=1
```

Display Memory

Input

```
D 0200 0200
```

Output

```
> D
0200 0200
0200 0000 0202 6548 0204 6C6C 0206 206F 0208 2020 020A 2020 020C 2020 020E 2020 ..Hello......
```

Display 16 Words of Memory

Input

```
D 0200 021F
```

Output

```
> D
0200 021F
0200 0000 0202 6548 0204 6C6C 0206 206F 0208 2020 020A 2020 020C 2020 020E 2020 ..Hello......
0210 07DC 0212 8114 0214 D453 0216 12A0 0218 D1E1 021A 1B0E 021C B8A8 021E 0C00 ....S........
```

Modify Memory Backwards/Forwards Skip

Input

```
M 0208 N P P <space>
```

Output

```
> M 0208
0208 2020 N
0206 206F P
0208 2020 P
020A 2020
>
```

Modify Memory Insert

Input

```
M 0208 2121 <space> D 0200 0200
```

Output

```
> M 0208
0208 2020 2121
020A 2020
> D
0200 0200
0200 0000 0202 6548 0204 6C6C 0206 206F 0208 2020 020A 2020 020C 2020 020E 2020 ..Hello!!....
```

SSMU Source Code

```
; EEL 4742 Project
; Rob Murrer (r2965302)
; 10-26-13
; This program provides a simple shell
; to the MSP430 over the serial port.


            ; set up device with proper header file
            .cdecls C,LIST,"msp430g2553.h"
            .text                       ; Assemble into program memory
            .retain                     ; Override ELF conditionalo linking
            .retainrefs                 ; Retain references to current sect


            ; constants
            .sect   ".const"
prompt      .string "> "
            .byte   0x00                ; null character to terminate string
endline     .byte   0x0D, 0x0A, 0x00    ; carriage return, line feed, null

            ; ram
            .sect   ".sysmem"
mFlag       .word   0x00
hello       .string "Hello         "
```

```
            ; code
            .text
            .global _START

            ; init
START       mov.w   #300h, SP           ; init stack pointer
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL    ; stop watchdog
            call    #Init_UART

            ; main program
cprompt     call    #newLine
            mov.w   #prompt, R6
            call    #printStr
            call    #getCmd
            jmp     cprompt


; gets command and processes it
getCmd      call    #INCHAR_UART
            cmp.b   #'H', R4
            jnz     chkDisplay
            call    #gcAS                ; jump to add/subtract
            jmp     cprompt

chkDisplay  cmp.b   #'D', R4
            jnz     chkModify
            call    #cmdDisplay
            jmp     cprompt

chkModify   cmp.b   #'M', R4
            jnz     chkEnd
            call    #cmdModify
            jmp     cprompt

chkEnd      jmp     getCmd              ; invalid input get new char

gcAS        call    #OUTA_UART
            call    #INCHAR_UART        ; get second character in cmd
            cmp.b   #'A', R4
            jz      gcAdd
            cmp.b   #'S', R4
            jz      gcSub
            jmp     gcAS                ; invalid input get new char

gcAdd       call    #OUTA_UART
            call    #cmdAdd
            ret

gcSub       call    #OUTA_UART
            call    #cmdSub
            ret

; gets two 4 digit hex numbers from uart
; adds them and prints result to uart
```

```
cmdAdd      call    #space
            call    #Hex8In             ; get 2 numbers
            call    #space
            call    #equal
            add.w   R4, R5              ; add the numbers
            mov.w   SR, R6
            call    #space
            call    #Hex4Out            ; print result
            call    #newLine
            call    #printFlags
            ret

; gets two 4 digit hex numbers from uart
; subtracts them and prints result to uart
cmdSub      call    #space
            call    #Hex8In             ; get 2 numbers
            call    #space
            call    #equal
            sub.w   R4, R5              ; subtract the numbers
            mov.w   SR, R6
            call    #space
            call    #Hex4Out            ; print result
            call    #newLine
            call    #printFlags
            ret

cmdModify   call    #OUTA_UART
            call    #space
            call    #Hex4In
cmNL        call    #newLine
            mov.w   R4, R5
            call    #Hex4Out
            call    #space
            mov.w   0(R4), R5           ; move contents of address R4 into R4
            call    #Hex4Out            ; print data
            call    #space
            push    R4                  ; save address
            call    #Hex4Inm            ; get data or command
            cmp.w   #0x00, &mFlag       ; see if it is only a command
            jz      cmModify
            cmp.w   #0x01, &mFlag       ; see if positive skip was pressed
            jz      cmPos
            cmp.w   #0x02, &mFlag       ; see if negative skip was pressed
            jz      cmNeg
            cmp.w   #0x03, &mFlag       ; see if exit was pressed
            jz      cmExit
cmPos       pop     R4                  ; get original address off stack
            incd.w  R4                  ; increase pointer to next word
            jmp     cmNL                ; goto next line and print next word and data
cmNeg       pop     R4                  ; ... same but go backwards in meme
            decd.w  R4
            jmp     cmNL
cmExit      pop     R5                  ; exit modify command
            ret
```

```
cmModify    pop     R5
            mov.w   R4, 0(R5)           ; copy input into memory address
            incd.w  R5                  ; increase pointer to next word
            mov.w   R5, R4              ; move new address to be start address
            jmp     cmNL                ; jump to change next address



; recieves to 4 digit Hex address from uart
; and prints the values of what is in RAM at
; those locations followed by their ascii equivalant
cmdDisplay  call    #OUTA_UART
            call    #newLine
            call    #Hex8In
            push    R4
cdSrt       call    #newLine
            mov.w   #0, R7              ; counter for line
cdHexSrt    push.w  R5                  ; starting address
            call    #Hex4Out            ; print address
            call    #space
            mov.w   0(R5), R5           ; R5 = memory[R5]
            call    #Hex4Out            ; print memory contents
            pop.w   R5                  ; Reset pointer to address
            incd.w  R5                  ; increment pointer to next word
            incd.w  R7                  ; increment counter by 2
            cmp.w   #16, R7             ; R7 == 16?
            jz      cdPAscii            ; start printing ascii
            call    #space
            jmp     cdHexSrt

cdPAscii    call    #space
            sub.w   #16, R5             ; reset pointer to start of row
cdPAscii1   mov.b   0(R5), R4           ; get first character from mem
            cmp.b   #0x21, R4           ; check if unprintable
            jlo     cdPPeriod
            cmp.b   #0x7F, R4
            jhs     cdPPeriod
            call    #OUTA_UART          ; print ascii representation
            jmp     cdPANxt
cdPPeriod   mov.b   #'.', R4
            call    #OUTA_UART
cdPANxt     inc.w   R5                  ; advance pointer
            dec.w   R7                  ; decrement counter
            jz      cdNxtRow            ; done printing row?
            jmp     cdPAscii1           ; print next char

cdNxtRow    pop R4                      ; get end address off stack
            cmp.w   R4, R5              ; compare start address to end address
            jhs     cdEnd               ; if start address is >= end then exit
            push    R4                  ; push end addres back onto stack
            jmp     cdSrt               ; print next line

cdEnd       ret
```

```
; prints the status flags located in R6
printFlags  push    R4
            mov.b   #'V', R4            ; check overflow bit
            call    #OUTA_UART
            call    #equal
            bit.w   #0x100, R6          ; bit test bit 8
            jnz     pfYesOverf          ; if not zero then yes over flow
            mov.b   #'0', R4            ; no overflow
            call    #OUTA_UART
            jmp     pfNegative
pfYesOverf  mov.b   #'1', R4
            call    #OUTA_UART

pfNegative  call    #space
            mov.b   #'N', R4            ; check negative bit
            call    #OUTA_UART
            call    #equal
            bit.w   #0x04, R6           ; bit test bit 2
            jnz     pfYesNeg            ; if not zero then yes negative
            mov.b   #'0', R4            ; not negative
            call    #OUTA_UART
            jmp     pfZero
pfYesNeg    mov.b   #'1', R4
            call    #OUTA_UART

pfZero      call    #space
            mov.b   #'Z', R4            ; check zero bit
            call    #OUTA_UART
            call    #equal
            bit.w   #0x02, R6           ; bit test bit 1
            jnz     pfYesZero           ; if not zero then yes zero
            mov.b   #'0', R4            ; not zero
            call    #OUTA_UART
            jmp     pfCarry
pfYesZero   mov.b   #'1', R4
            call    #OUTA_UART

pfCarry     call    #space
            mov.b   #'C', R4            ; check carry bit
            call    #OUTA_UART
            call    #equal
            bit.w   #0x01, R6           ; bit test bit 0
            jnz     pfYesCarry          ; if not zero then yes carry
            mov.b   #'0', R4            ; no carry
            call    #OUTA_UART
            jmp     pfEnd
pfYesCarry  mov.b   #'1', R4
            call    #OUTA_UART

pfEnd       pop     R4
            ret
```

```
; print new line and carriage return
newLine     push    R6
            mov.w   #endline, R6
            call    #printStr
            pop     R6
            ret

; print = to serial
equal       push.w  R4
            mov.b   #'=', R4
            call    #OUTA_UART
            pop.w   R4
            ret

; print space to serial
space       push.w  R4
            mov.b   #0x20, R4
            call    #OUTA_UART
            pop.w   R4
            ret


; gets 2 separate 4 digit hex values from serial
; returns them in R4, R5
Hex8In      call    #Hex4In             ; get first set of hex digits
            mov.w   R4, R5
            call    #space              ; print space to screen
            call    #Hex4In             ; get second set of hex digits
            ret


; gets 4 valid hex values and put them into R4
Hex4In      push    R5
            call    #Hex2In             ; get two hex values into R4
            rla.w   R4                  ; rotate into upper byte
            rla.w   R4
            rla.w   R4
            rla.w   R4
            rla.w   R4
            rla.w   R4
            rla.w   R4
            rla.w   R4
            mov.w   R4, R5              ; save away first value
            call    #Hex2In             ; get second set of 2 hex values
            add.w   R5, R4              ; combine all four digits into R4
            pop     R5
            ret

; gets 2 valid hex digits from UART and store in R4
Hex2In      push    R5
            call    #Hex1In
```

```
            rla.b   R4                  ; rotate into upper half of lower byte
            rla.b   R4
            rla.b   R4
            rla.b   R4
            mov.b   R4, R5              ; save away first digit
            call    #Hex1In             ; get second value
            add.b   R5, R4              ; combine both digits into R4
            pop     R5
            ret

; gets a valid Hex digit from UART returns in R4
Hex1In      call    #INCHAR_UART            ; get character from input
            cmp.b   #0x30, R4               ; check if lower than 0
            jlo     Hex1In                  ; not valid hex value
            cmp.b   #0x3A, R4               ; check if greater than 9
            jge     CheckLetter
            call    #OUTA_UART
            sub.b   #0x30, R4
            jmp     Hex1InEnd

CheckLetter cmp.b   #0x41, R4               ; check if lower than A
            jlo     Hex1In
            cmp.b   #0x47, R4               ; check if greater than F
            jge     Hex1In
            call    #OUTA_UART
            sub.b   #0x37, R4
Hex1InEnd   ret


; gets 4 valid hex values and put them into R4
; for the modify command
Hex4Inm     push    R5
            call    #Hex2Inm            ; get two hex values into R4
            cmp.w   #0x00, &mFlag
            jz      h4Normal
            pop     R5
            ret
h4Normal    rla.w   R4                  ; rotate into upper byte
            rla.w   R4
            rla.w   R4
            rla.w   R4
            rla.w   R4
            rla.w   R4
            rla.w   R4
            rla.w   R4
            mov.w   R4, R5              ; save away first value
            call    #Hex2Inm            ; get second set of 2 hex values
            cmp.w   #0x00, &mFlag
            jz      h4Normal1
            pop     R5
            ret
h4Normal1   add.w   R5, R4              ; combine all four digits into R4
            pop     R5
            ret
```

```
; gets 2 valid hex digits from UART and store in R4
; for the modify command
Hex2Inm     push    R5
            call    #Hex1Inm
            cmp.w   #0x00, mFlag     ; see if flag has been set
            jz     h2Normal           ; continue on no flag is set
            call    #OUTA_UART          ; print n,p,space
            pop     R5
            ret                         ; return back
h2Normal    rla.b   R4                  ; rotate into upper half of lower byte
            rla.b   R4
            rla.b   R4
            rla.b   R4
            mov.b   R4, R5              ; save away first digit
            call    #Hex1Inm            ; get second value
            cmp.w   #0x00, &mFlag       ; see if flag has been set
            jz     h2Normal1            ; continue on no flag is set
            call    #OUTA_UART          ; print n,p,space
            pop     R5
            ret                         ; return back
h2Normal1   add.b   R5, R4              ; combine both digits into R4
            pop     R5
            ret
; gets a valid Hex digit from UART returns in R4
; or a 'n' 'p' or ' '
Hex1Inm     call    #INCHAR_UART        ; get character from input
            cmp.b   #'P', R4            ; see if special characters are entered
            jz      h1Positive
            cmp.b   #'N', R4
            jz      h1Negative
            cmp.b   #' ', R4
            jz      h1Space
            mov.w   #0x00, &mFlag       ; reset flag to zero
            jmp     h1Normal            ; skip to normal hex entry
h1Positive  mov.w   #0x01, &mFlag       ; set flag to 1
            ret
h1Negative  mov.w   #0x02, &mFlag       ; set flag to 2
            ret
h1Space     mov.w   #0x03, &mFlag       ; set flag to 3
            ret
h1Normal    cmp.b   #0x30, R4           ; check if lower than 0
            jlo     Hex1Inm             ; not valid hex value
            cmp.b   #0x3A, R4           ; check if greater than 9
            jge     CheckLettem
            call    #OUTA_UART
            sub.b   #0x30, R4
            jmp     Hex1InmEnd

CheckLettem cmp.b   #0x41, R4           ; check if lower than A
            jlo     Hex1Inm
            cmp.b   #0x47, R4           ; check if greater than F
            jge     Hex1Inm
            call    #OUTA_UART
```

```
            sub.b   #0x37, R4
Hex1InmEnd  ret

; prints 2 4 digit hex values seperated by space r5 r4
Hex8Out     push    R5
            push    R4
            call    #Hex4Out
            mov.b   #0x20, R4           ; print space
            call    #OUTA_UART
            pop     R4
            mov.w   R4, R5
            call    #Hex4Out
            pop     R5
            ret

; prints 4 hex values located in R5
Hex4Out     push.w  R5
            swpb    R5                  ; swap bytes to print in correct order
            call    #Hex2Out
            pop.w   R5
            call    #Hex2Out
            ret


; prints 2 hex values in the lower bit R5
Hex2Out     push.w  R4
            push.w  R5
            rra.b   R5                  ; clear lower bits
            rra.b   R5                  ; and move upper bits
            rra.b   R5
            rra.b   R5

            and.b   #0x0F, R5           ; clear sign extension
            cmp.b   #0x0A, R5           ; R5 > 10?
            jlo     H2Onum1             ; if (R5 < 10) goto H2Onum1

            add.b   #0x37, R5           ; R5 = R5 + 0x37
            jmp     H2Onext             ; goto next character

H2Onum1     add.b   #0x30, R5           ; R5 = R5 + 0x30

H2Onext     mov.w   R5, R4              ; R4 = R5
            call    #OUTA_UART          ; print character

            pop     R5                  ; restore r5
            push    R5
            ; print second character
            and.b   #0x0F, R5           ; clear upper bits

            cmp.b   #0x0A, R5           ; R5 > 10?
            jlo     H2Onum2             ; if (R5 < 10) goto H2Onum1

            add.b   #0x37, R5           ; R5 = R5 + 0x37
            jmp     H2Oend              ; goto end
```

```
H2Onum2     add.b   #0x30, R5           ; R5 = R5 + 0x30

H2Oend      mov.w   R5, R4              ; R4 = R5
            call    #OUTA_UART          ; print character

            pop     R5
            pop     R4
            ret


printStr    ; prints a string at the address in R6
            push    R6
            push    R4
strL1       mov.b   0(R6), R4           ; move first byte of string to R4
            cmp.w   #0x00, R4           ; at end of string?
            jz      strEnd              ; go print line feed and carriage return

            call    #OUTA_UART          ; print char
            inc     R6                  ; advance pointer to next char
            jmp     strL1               ; loop

strEnd      pop     R4
            pop     R6                  ; leave no trace, return
            ret

OUTA_UART   ; wait for transmit buffer to be empty, then send data into R4
            push    R5
lpa         mov.b   &IFG2, R5
            and.b   #0x02, R5
            cmp.b   #0x00, R5
            jz lpa
            ; send the data to the transmit buffer UCA0TXBUF = A
            mov.b   R4, &UCA0TXBUF
            pop     R5
            ret

INCHAR_UART ; wait until recieve buffer is full, then get data into R4
            push    R5
lpb         mov.b   &IFG2, R5
            and.b   #0x01, R5
            cmp.b   #0x00, R5
            jz lpb
            mov.b   &UCA0RXBUF, R4
            pop     R5
            ret

Init_UART
;-------------------------------------------------------------------
; Initialization code to set up the uart on the experimenter board to 8 data,
; 1 stop, no parity, and 9600 baud, polling operation
;-------------------------------------------------------------------
;-------------------------------------------------------------------
; Set up the MSP430g2553 for a 1 MHZ clock speed
```

```
; For the version 1.5 of the launchpad MSP430g2553
; BCSCTL1=CALBC1_1MHZ;
; DCOCTL=CALDCO_1MHZ;
; CALDCO_1MHZ and CALBC1_1MHZ is the location in the MSP430g2553
; so that the for MSP430 will run at 1 MHZ.
; give in the *.cmd file
; CALDCO_1MHZ        = 0x10FE;
; CALBC1_1MHZ        = 0x10FF;
            mov.b   &CALBC1_1MHZ, &BCSCTL1
            mov.b   &CALDCO_1MHZ, &DCOCTL
;-------------------------------------------------------------------
; Set up the MSP430g2553 for 1.2 for the transmit pin and 1.1 receive pin
; For the version 1.5 of the launchpad MSP430g2553
; Need to connect the UART to port 1.
; 00 = P1SEL, P1sel2 = off, 01 = primary I/O, 10 = Reserved, 11 = secondary I/O for UART
; P1SEL =  0x06;    // transmit and receive to port 1 bits 1 and 2
; P1SEL2 = 0x06;   // transmit and receive to port 1 bits 1 and 2
;-------------------------------------------------------------------
            mov.b   #0x06,&P1SEL
            mov.b   #0x06,&P1SEL2
; Bits p2.4 transmit and p2.5 receive UCA0CTL0=0
; 8 data, no parity 1 stop, uart, async
            mov.b   #0x00,&UCA0CTL0
; (7)=1 (parity), (6)=1 Even, (5)= 0 lsb first,
; (4)= 0 8 data / 1 7 data, (3) 0 1 stop 1 / 2 stop, (2-1) --
; UART mode, (0) 0 = async
; select MLK set to 1 MHZ and put in software reset the UART
; (7-6) 00 UCLK, 01 ACLK (32768 hz), 10 SMCLK, 11 SMCLK
; (0) = 1 reset
; UCA0CTL1= 0x81;
            mov.b   #0x81,&UCA0CTL1
; UCA0BR1=0;
; upper byte of divider clock word
            mov.b   #0x00,&UCA0BR1
; UCA0BR0=68; ;
; clock divide from a MLK of 1 MHZ to a bit clock of 9600 -> 1MHZ /
; 9600 = 104.16 104 =0x68
            mov.b   #0x68,&UCA0BR0
; UCA0BR1:UCA0BR0 two 8 bit reg to from 16 bit clock divider
; for the baud rate
; UCA0MCTL=0x06;
; low frequency mode module 3 modulation pater used for the bit
; clock
            mov.b   #0x06,&UCA0MCTL
; UCA0STAT=0;
; do not loop the transmitter back to the receiver for echoing
            mov.b   #0x00,&UCA0STAT
; (7) = 1 echo back trans to rec
; (6) = 1 framing, (5) = 1 overrun, (4) =1 Parity, (3) = 1 break
; (0) = 2 transmitting or receiving data
;UCA0CTL1=0x80;
; take UART out of reset
            mov.b   #0x80,&UCA0CTL1
;IE2=0;
```

```
; turn transmit interrupts off
            mov.b   #0x00,&IE2
; (0) = 1 receiver buffer Interrupts enabled
; (1) = 1 transmit buffer Interrupts enabled
;-------------------------------------------------------------------
;*******************************************************************
;-------------------------------------------------------------------
; IFG2 register (0) = 1 receiver buffer is full, UCA0RXIFG
; IFG2 register (1) = 1 transmit buffer is empty, UCA0RXIFG
; UCA0RXBUF 8 bit receiver buffer, UCA0TXBUF 8 bit transmit
; buffer
            ret

END
            .sect   ".reset"
            .short  START
            .end
```