43oh

SIMPL - An update on the Tiny Forth-like language

forth simpl msp430g2553

By monsonite,
May 23, 2017 in Projects

Reply to this topic

SIMPL = Serial Interpreted Minimal Programming Language

Hi,

Posted May 23, 2017

It's been about a year since I talked about SIMPL - a tiny language that allows you basic control of a microcontroller using serial commands.

In the 6 months I have coded it up in MSP430 assembly language to make it super compact - and fast - with high level commands taking about 1uS to execute on the virtual machine interpreter.

SIMPL is based on a jump table - so for any single, printable ascii character, the jump table will act on it and execute whatever function you choose to write. This technique gives amazing flexibility, and for under 1K of code it can offer a very powerful user interface to your latest MSP430 project. One example is using the jump table to interpret commands from a text file to control a CNC mill or drill - or even a 3D printer.

The core routines can be applied to any MSP430 - you just have to change the initialisation routines to suit the DCO, GPIO and UART of the specific microcontroller

Just this week, I have got the looping to work, so you can now do things like send square waves to port pins for flashing LEDs and creating musical tones.

With the standard Launchpad (MSP430GR2553) clocked at 16MHz you can send a 1uS pulse to a port pin - just by typing hl (shorthand for high, low) at the serial terminal.

SIMPL is coded up in just 872 bytes of program memory - and can handle up to 96 separate commands.

Commands can be sent to the device from a text file - using teraterm or similar - or just typed manually at the keyboard.

I have put the latest code (some recent changes) on this github gist https://gist.github.com/monsonite/6483a32404c1c53cd8027dd6f9dcea6e

This is a work in progress - and there are still a few bugs - but the basics seem to work OK.

regards

Ken

Here's some more info about the various routines that make up the kernel

textRead 33 Instructions 90 bytes

Receive characters from the serial uart and place them into a buffer in RAM. Two modes of operation are possible, immediate mode, where the characters are executed as instructions directly after a carriage return line feed is received, and compile mode, where the character sequences are preceded by a colon and stored in pre-calculated command buffers in RAM.

number 16 instructions 42 bytes

Number interprets sequences of consecutive digits as a 16-bit integer number and places the value in the register that is the top entry of the stack.

next 5 instructions 12 bytes

Next is the routine that all commands return the program flow back to once they have executed. It fetches the next character instruction from the RAM buffer and through a jump table technique passes program control to the code body that performs the task associated with that instruction. The jump table is used to direct the character to the areas of code that will treat them correctly -

for example on encountering a numerical digit, program control is passed to the number routine, whilst for upper case alphabetical characters, program control is passed to a routine that handles these separately.

jump_table 96 instructions 192 bytes

Primitives 72 instructions 166 bytes

SIMPL uses a collection of 32 instruction primitives from which other instructions can be synthesised. The code body of these primitives is some 100 instructions or so.

Upper (called alpha in V1) 10 instructions 26 bytes

Upper handles the capital letters - as these are user commands, and the user is able to write and store in RAM certain functionality based on these characters. When a capital letter is encountered in the instruction buffer, Upper directs control to the correct command buffer.

Lower 86 bytes

Lower is an area of program that interprets the lower case characters and provides a higher level of program complexity than is achievable form the primitives alone.

printnum 30 instructions 106 bytes

This takes a 16 bit integer number from the stack and prints a string of ascii digit characters to the terminal.

Uart Routines 13 instructions 41 bytes

Low level communication with the uart is by way of the get_c and put_c routines

Initialisation 19 instructions 90 bytes

Here the hardware such as the oscillator, GPIO and uart are initialised for correct operation. This is code specific to whichever microcontroller has been chosen

Interpreter 4 instructions 16 bytes

This is the main routine that runs the SIMPL interpreter combining textRead, next, number and Upper.

Here's the code - as it stands. Code window has mess up the formatting - but it should still cut and paste

```
;-------------------------------------------------------------------------------
; SIMPL - a very small Forth Inspired Extensible Language
; Implementing the Initialisation, TextTead, TextEval and UART routines in MSP430 ass
;
; A Forth-Like Language in under 1024 bytes

; Ken Boak   May 22nd/23rd 2017
; Loops, I/O, Strings and Delays added
; This version 872 bytes
; Instructions take about 1uS cycle  time  - so about 1/16th of clockspeed
;-------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"     ; Include device header file

;-------------------------------------------------------------------------------
            .def    RESET                   ; Export program entry-point to
                                            ; make it known to linker.
;-------------------------------------------------------------------------------
; Variables
;-------------------------------------------------------------------------------
                        .sect "vars"

                        .bss parray, 256


                        .bss x, 2
                        .bss name, 2

;-------------------------------------------------------------------------------
 ; Using the register model of CH Ting's Direct Thread Model of MSP430 eForth
 ; CPU registers

 ; Register Usage

; R0    MSP430 PC Program Counter
; R1    MSP430 SP Stack Pointer
; R2    MSP430 SR Status Register

tos                     .equ R4
stack                   .equ R5
ip                              .equ R6
temp0                   .equ R7                 ; loop start
temp1                   .equ R8                 ; loop counter k
temp2                   .equ R9                 ; millisecond delay
```

```
temp3                   .equ R10        ; microsecond delay
temp4                   .equ R11
instr                   .equ R12
temp5                   .equ R13
temp6                   .equ R14                ;
temp7                   .equ R15            ; Return from alpha  next IP

;-------------------------------------------------------------------------
; Macros

pops                    .macro          ;DROP
                                mov.w @stack +, tos
                                .endm

pushs                   .macro          ;DUP
                                decd.w stack
                                mov.w tos, 0(stack)
                                .endm;

; Constants

$NEXT                   .macro
                                jmp next
;                               mov @ip+, pc ; fetch code address into PC
                                .endm


$NEST                   .macro
                                .align 2
                                call #DOLST ; fetch code address into PC, W = PFA
                                .endm

$CONST                  .macro
                                .align 2
                                call #DOCON ; fetch code address into PC, W = PFA
                                .endm

;-------------------------------------------------------------------------
;; Assembler constants

COMPO           .equ 040H               ;lexicon compile only bit
IMEDD           .equ 080H               ;lexicon immediate bit
MASKK           .equ 07F1FH     ;lexicon bit mask
```

```
CELLL           .equ 2                  ;size of a cell
BASEE           .equ 10                 ;default radix
VOCSS           .equ 8                  ;depth of vocabulary stack
BKSPP           .equ 8                  ;backspace
LF                      .equ 10                 ;line feed
CRR             .equ 13                 ;carriage return
ERR             .equ 27                 ;error escape
TIC             .equ 39                 ;tick
CALLL           .equ 012B0H     ;NOP CALL opcodes
UPP             .equ 200H
DPP             .equ 220H
SPP             .equ 378H               ;data stack
TIBB            .equ 380H               ;terminal input buffer
RPP             .equ 3F8H               ;return stacl
CODEE           .equ 0C000H     ;code dictionary
COLDD           .equ 0FFFEH     ;cold start vector
EM              .equ 0FFFFH     ;top of memory




;-----------------------------------------------------------------------------
            .text                           ; Assemble into program memory.
            .retain                         ; Override ELF conditional linking
                                            ; and retain current section.
            .retainrefs                     ; And retain any sections that have
                                            ; references to current section.
;-----------------------------------------------------------------------------
; This implements the SIMPL interpreter is MSP430 assembly Language


;-----------------------------------------------------------------------------
; textRead
; ----------------------------------------------------------------------------

; Get a character from the UART and store it in the input buffer starting at 0x0200

; Register Usage

; The input buffer - start is at 0x0200, which is pointed to by R14
; R11 is a counter to ensure that we don't exceed 64 characters in input buffer
; R12 receives the character from the uart_get_c routine and puts in the buffer, poin
; R14 is the current character position in the input buffer
```

```
; 33 instructions


textRead:               MOV.W   #0x0200,R14             ; R14 = start of input buffer
                        CLR.B   R11                             ; i = 0

getChar:                CALL    #uart_getc      ; char ch = uart_getc()
                        CMP.B   #0x000d,R12             ; is it carriage retu
                JEQ     textEnd
                        CMP.B   #0x000a,R12     ; Is it  newline?  0a
                JEQ     textEnd
                        CMP.B   #0x0020,R12     ; if (ch >= ' ' && ch <= '~')
                JLO     nonValid
                        CMP.B   #0x007f,R12
                        JHS     nonValid

                        CMP.B   #0x003A,R12             ; is it colon?  3A
                        JNE     notColon

colon:                  ; If the input character is a colon

                        CALL    #uart_getc          ; get the next character
                        MOV.B   R12,R13                     ; move the 1st ch

times_32:               SUB.B   #0x0041,R13                     ; Calculate the desti
                        ADD.W   R13,R13                         ; Double R13
                        ADD.W   R13,R13                         ; Double R13
                        ADD.W   R13,R13                         ; Double R13
                        ADD.W   R13,R13                         ; Double R13
                        ADD.W   R13,R13                         ; Double R13
                        ADD.W   R13,R14                         ; Add (32*R13
                        ADD.W   #0x020,R14                      ; Add to arra
                        MOV.B   R12,0x0000(R14)     ; Store character at RAM

                        JMP     incPointer

notColon:               INC.W   R14                             ; Increment buffer po
                        MOV.B   R12,0xffff(R14)         ; Store character at


incPointer:             INC.B   R11                             ; Increment the input
```

```
nonValid:               CMP.B   #0x003f,R11                     ; If input pointer <6
                JLO     getChar                 ; loop back and get next character

textEnd:                mov.b   #0x00,0x0000(R14)       ; Put a null terminating  (0x
                ;               MOV.B   R11,0x0000(R14)         ; Put a null terminat

                                RET


;-------------------------------------------------------------------------------
; We now come onto the textEval - where based on the value of the character we perfor
; But first we need to determine whether the characers form part of a number - and th
; and put on the stack
;-------------------------------------------------------------------------------

; Register Usage

; ip  - instruction pointer to the current character in the input buffer
; R12   is the accumulator for the number - then stored in location #0x380
; R13   Temporary - use in x10 multipication
; R14


; 16 Instructions



number:                 SUB.W   #0x0030,R12                             ; subtract 0x

number1:                CMP.B   #0x0030,0x0000(ip)              ; >= '0'   Is the nex
                        JLO     endNumber                               ; bre
                        CMP.B   #0x003a,0x0000(ip)              ; <= '9'
                        JHS     endNumber                               ; bre

times_10:
                        ADDC.W  R12,R12                 ; R12 = 2 * R12
                        MOV.W   R12,R13                                 ; R13 = 2
                        ADDC.W  R12,R12                                 ; R12 = 4
                        ADDC.W  R12,R12                                 ; R12 = 8
                        ADDC.W  R13,R12                 ; R12 = 10 x R12

                        MOV.B   @ip+,R14                                ; Increme
```

```
                        SUB.W   #0x0030,R14
                        ADD.W   R14, R12                        ; Add in
                        JMP     number1                             ; pro

endNumber:          MOV.W   R12, tos                        ; Put in tos
                        JMP     next                            ; process









; -------------------------------------------------------------------------------
; next fetches the next ascii character instruction from memory, decodes it into a ju
; found at that code address
; Each executed word jumps back to next
; Numbers are treated differenty - they are enummerated and put onto the stack by the

; Now we need to decode the instructions using a jump table
; Jump table uses 2 bytes per instruction - so  2 x 96  = 192 bytes

next:               MOV.B   @ip+,R12                        ; Get the nex
                        MOV.W   R12,R13                         ; Copy in
                        SUB.w   #0x0020,R13                         ; sub
                        ADD.w   R13,R13                             ; dou
                        add.w   R13,pc                              ; jum

tabstart:           jmp space                   ; SP
                        jmp store                       ; !
                        jmp dup                         ; "
                        jmp lit                         ; #
                        jmp swap                        ; $
                        jmp over                        ; %
                        jmp and                         ; &
                        jmp drop                        ; '
                        jmp left_par            ; (
                        jmp right_par           ; )
                        jmp mult                        ; *
                        jmp add                         ; +
```

```
jmp push                        ; ,
jmp sub                         ; -
jmp pop                         ; .
jmp div                         ; /
jmp number                      ; 0
jmp number                      ; 1
jmp number                      ; 2
jmp number                      ; 3
jmp number                      ; 4
jmp number                      ; 5
jmp number                      ; 6
jmp number                      ; 7
jmp number                      ; 8
jmp number                      ; 9
jmp colon                       ; :
jmp semi                        ; ;
jmp less                        ; <
jmp equal                       ; =
jmp greater                     ; >
jmp     query                   ; ?
jmp fetch                       ; @
jmp alpha                       ; A
jmp alpha                       ; B
jmp alpha                       ; C
jmp alpha                       ; D
jmp alpha                       ; E
jmp alpha                       ; F
jmp alpha                       ; G
jmp alpha                       ; H
jmp alpha                       ; I
jmp alpha                       ; J
jmp alpha                       ; K
jmp alpha                       ; L
jmp alpha                       ; M
jmp alpha                       ; N
jmp alpha                       ; O
jmp alpha                       ; P
jmp alpha                       ; Q
jmp alpha                       ; R
jmp alpha                       ; S
jmp alpha                       ; T
jmp alpha                       ; U
jmp alpha                       ; V
```

```
    jmp alpha                       ; W
    jmp alpha                       ; X
    jmp alpha                       ; Y
    jmp alpha                       ; Z
    jmp     square_left             ; [
    jmp f_slash                     ; \ ;
    jmp square_right        ; ]
    jmp xor                         ; ^
    jmp underscore          ; _
    jmp tick                        ; `
    jmp lower_a                     ; a
    jmp lower_b                     ; b
    jmp lower_c                     ; c
    jmp lower_d                     ; d
    jmp lower_e                     ; e
    jmp lower_f                     ; f
    jmp lower_g                     ; g
    jmp lower_h                     ; h
    jmp lower_i                     ; i
    jmp lower_j                     ; j
    jmp lower_k                     ; k
    jmp lower_l                     ; l
    jmp lower_m                     ; m
    jmp lower_n                     ; n
    jmp lower_o                     ; o
    jmp lower_p                     ; p
    jmp lower_q                     ; q
    jmp lower_r                     ; r
    jmp lower_s                     ; s
    jmp lower_t                     ; t
    jmp lower_u                     ; u
    jmp lower_v                     ; v
    jmp lower_w                     ; w
    jmp lower_x                     ; x
    jmp lower_y                     ; y
    jmp lower_z                     ; z
    jmp     curly_left              ; {
    jmp or                          ; |
    jmp curly_right         ; }
    jmp inv                         ; ~
    jmp delete                      ; del
    jmp textEval_end        ; 0x80 is used as null termin
```

```
;-------------------------------------------------------------------------------

; Handle the alpha and lower case chars

alpha:                  SUB.B   #0x0041,R12                             ; subtract 65
                        MOV.W   R12,R13                                         ; get
                        ADD.W   R13,R13                                         ; Dou
                        ADD.W   R13,R13                                         ; Dou
                        ADD.W   R13,R13                                         ; Dou
                        ADD.W   R13,R13                                         ; Dou
                        ADD.W   R13,R13                                         ; Dou
                        ADD.W   #0x220,R13                                      ; Add
                        MOV.W   ip,R15                                  ; Save th

                        MOV.W   R13,ip                                          ; ins

                        JMP     next                                    ; process



;-------------------------------------------------------------------------------
; Handle the primitive instructions



space:                  pushs                   ; Move a 2nd number onto the stack
                        $NEXT



store:                  mov.w @stack +, 0(tos)
                        pops
                        $NEXT

dup:            pushs
                $NEXT

lit:

                        $NEXT

swap:           mov.w tos, temp0
                mov.w @stack, tos
```

```
                mov.w temp0,0( stack)
                $NEXT

over:                   mov.w tos, temp0
                                mov.w @stack, tos
                                mov.w temp0,0( stack)
                                $NEXT

and:                    and @stack +, tos
                                $NEXT

drop:                   pops
                                $NEXT



left_par:                                                                       ; cod

                                MOV.W   tos,R8                          ; save tos to
                                MOV.W   ip,R7                           ; loop-start
                                JMP     next                            ; get the nex

right_par:                                                                      ; cod

                                ; TST.W   R8                           ; is loop cou
                                ; JEQ     next                      ; terminate loop

                                DEC.W   R8                                      ; dec
                                JEQ     next                      ; terminate loop
                                MOV.W   R7,ip                           ; set instruc
                                JMP     next                            ; go around l


mult:

                                $NEXT


add:                    add @stack +, tos
                                $NEXT
```

```
push:

                        $NEXT


sub:                    sub @stack +, tos
;                       jmp NEGAT

NEGAT:                  inv tos
                        inc tos
                        $NEXT

pop:
                        jmp     printNum                ; go to decimal numbe
                        $NEXT


div:

                        $NEXT

semi:                   ; On encountering a semicolon return program control to the n

                        MOV.W   R15,ip                  ; restore the ip
                        $NEXT


query:                  $NEXT

fetch:                  mov.w @tos, tos
                        $NEXT

square_right:

f_slash:

square_left:

curly_right:

curly_left:
```

```
underscore:                                                    ; Print the e


print_start:
                        MOV.B   @ip+,R12            ; Get the next character
                        CMP.B   #0x005f,R12     ; is it an underscore
                        jeq     print_end
                        CALL    #uart_putc          ; send it to uart
                        jmp     print_start
print_end               call    #crlf               ; line feed at end of text st
                        $NEXT


tick:                   ; tick allows access to the loop counter

                        MOV.W   R8,tos
                        $NEXT




delete:                 $NEXT

or:                             bis @stack +, tos
                                $NEXT

xor:                    xor     @stack +, tos
                                $NEXT

inv:                    inv tos
                                $NEXT

less:                   cmp @stack +, tos
                                jz FALSE
                                jge TRUE
                                jmp FALSE

equal:                  xor @stack +, tos
                                jnz FALSE
                                jmp TRUE

greater:                cmp @stack +, tos
                                jge FALSE
```

```
                                jmp TRUE

FALSE:                  clr tos
                                $NEXT

TRUE:                   mov #0x01, tos
                                $NEXT


;-------------------------------------------------------------------------------
;lower case routines

lower_a:

                                $NEXT

lower_b:

                                $NEXT

lower_c:

                                $NEXT

lower_d:

                                $NEXT

lower_e:

                                $NEXT

lower_f:

                                $NEXT

lower_g:

                                $NEXT

lower_h:
                                MOV.B   #0x0001,&P1OUT
                                $NEXT
```

```
lower_i:

                                $NEXT

lower_j:

                                $NEXT

lower_k:
                                ; k allows access to the loop counter variable stored

                                MOV.W   R8,tos

                                $NEXT

lower_l:
                                MOV.B   #0x0000,&P1OUT

                                $NEXT

lower_m:
                                MOV.W   tos,R10
mS_loop:
                                mov.w   #5232,R9
uS3_loop:               DEC.W   R9
                                JNE             uS3_loop

                                DEC.W   R10
                                JNE             mS_loop
                                $NEXT

lower_n:

                                $NEXT

lower_o:

                                $NEXT

lower_p:
                                JMP printNum
```

```
                        $NEXT

lower_q:

                        $NEXT

lower_r:

                        $NEXT

lower_s:

                        $NEXT

lower_t:

                        $NEXT

lower_u:                                                        ; 3 m

                        MOV.W   tos,R10
uS_loop:
                        DEC.W   R10
                        JNE             uS_loop
                        $NEXT

lower_v:

                        $NEXT

lower_w:

                        $NEXT

lower_x:

                        $NEXT


lower_y:

                        $NEXT
```

```
lower_z:

                                $NEXT
;-----------------------------------------------------------------------------
; User Routines


;-----------------------------------------------------------------------------
printNum:               ; Take the 16 bit value in R4 stack register and print to ter
                                ; do by repeated subtraction of powers of 10
                                ; Uses R10,11,12,13
;-----------------------------------------------------------------------------
                                        MOV.W   #10000,R10              ; R10 used as
        ;                               CLR.W   R12                             ; use
                                        CLR.W   R11                             ; Use
                                        CLR.W   R13
                                        MOV.W   tos,R12                 ; copy the to
                                        CLRC                                    ; cle

sub10K:                         SUB.W   R10,R12
                                    JLO      end10K

add10K:                         ADD.B #1,R11                    ; increments the digi
add_zero:                       ADD.W R10,R13                   ; R13 increases by th
                                        JMP   sub10K

end10K:                         ADD.B  #0x30,R11                ; make it a number
                                        MOV.W  R11,R12
                                        CALL    #uart_putc              ; output char
                                        SUB.W   R13,tos                 ; Decrement t
                                        CLR.W   R11                             ; Use
                                        CLR.W   R13
                                        MOV.W   tos,R12

decimate:                       CMP.W   #10000,R10
                                        JEQ     use1K
                                        CMP.W   #1000,R10
                                        JEQ     use100
                                        CMP.W   #100,R10
                                        JEQ     use10
                                        CMP.W   #10,R10
                                        JEQ     use1
```

```
newline:                MOV.W #0x0A, R12
                                CALL    #uart_putc                      ; out
                                MOV.W #0x0D, R12
                                CALL    #uart_putc                      ; out
                                JMP     next

use1K:                          MOV.W   #1000,R10
                                JMP             sub10K
use100:                         MOV.W   #100,R10
                                JMP             sub10K
use10:                          MOV.W   #10,R10
                                JMP             sub10K
use1:                           MOV.W   #1,R10
                                JMP             sub10K




;-------------------------------------------------------------------------------




;-------------------------------------------------------------------------------
; Uses R12 to send receive chars via the UART at 115200 baud.

uart_getc:              BIT.B   #1,&IFG2                        ; while (!(IFG2&UCA0R
                                JEQ     (uart_getc)
                                MOV.B   &UCA0RXBUF,R12          ; return UCA0RXBUF;
                                RET



uart_putc:              BIT.B   #2,&IFG2                        ; while (!(IFG2&UCA0T
                                JEQ     (uart_putc)
                MOV.B   R12,&UCA0TXBUF          ; UCA0TXBUF = c;    // TX
                                RET


crlf:                   MOV.W #0x0A, R12
```

```
                                CALL    #uart_putc                      ; output CR
                                MOV.W #0x0D, R12
                                CALL    #uart_putc                      ; output LF
                                RET

;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
main:

;-------------------------------------------------------------------------------
RESET:       ;  mov.w   #03E0h,SP              ; Initialize stackpointer

                                mov #RPP, SP                                    ; set
                                mov #SPP, stack
                                clr tos

StopWDT:        mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer WDTCTL  = WDTPW


OSC_GPIO_init:  ; Run the CPU at full 16MHz with 11500baud UART

                                MOV.B   &CALBC1_16MHZ,&BCSCTL1
                                MOV.B   &CALDCO_16MHZ,&DCOCTL

SetupP1:        bis.b   #041h,&P1DIR                                    ;P1.0
                                MOV.B   #0x0000,&P1OUT

uart_init:      MOV.B   #0x0006,&P1SEL
                                MOV.B   #0x0006,&P1SEL2
                                BIS.B   #0x0080,&UCA0CTL1
                                MOV.B   #0x008A,&UCA0BR0
                                CLR.B   &UCA0BR1
                                MOV.B   #2,&UCA0MCTL
                                BIC.B   #1,&UCA0CTL1


                                MOV.W #0x4F, R12
                                CALL    #uart_putc                      ; output "O"
                                MOV.W #0x4B, R12
                                CALL    #uart_putc                      ; output "K"

;-------------------------------------------------------------------------------
```

```
interpreter:
                                call    #textRead
                                MOV.W   #0x0200,ip                              ; set
                                jmp     next

textEval_end:   jmp     interpreter                             ; loop around



; Stack Pointer definition
;-----------------------------------------------------------------------------
            .global __STACK_END
            .sect   .stack

;-----------------------------------------------------------------------------
; Interrupt Vectors
;-----------------------------------------------------------------------------
            .sect   ".reset"                ; MSP430 RESET Vector
            .short  RESET

                        .end
```

Quote

jBrizzle and dubnet 2

Join the conversation

You can post now and register later. If you have an account, sign in now to post with your account.

Reply to this topic...

Share

Followers 0

Go to topic listing