Pointers (2G)

Young Won Lim
4/10/18

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Young Won Lim
4/10/18

Based on Embedded Software in C for an ARM Cortex M
http://users.ece.utexas.edu/~valvano/Volume1/

Pointer Declarations

```
short *pt1;                 /* define pt1, declare as a pointer to a 16-bit integer */
char *pt2;                  /* define pt2, declare as a pointer to an 8-bit character */
unsigned short data,*pt3;   /* define data and pt3,
                               declare data as an unsigned 16-bit integer and
                               declare pt3 as a pointer to a 16-bit unsigned integer */
long *pt4;                  /* define pt4, declare as a pointer to a 32-bit integer */
extern short *pt5;          /* declare pt5 as a pointer to an integer */
```

Pointer Referencing

```
long *pt;              // pointer to 32-bit data
long data;             // 32-bit
long buffer[4];        // array of 4 32-bit numbers

int main(void){
  pt = &buffer[1];
  *pt = 1234;          // 0x04D2
  data = *pt;
  return 1;
}
```

address	data	contents
0x20000000	0x00000000	pt
0x20000004	0x00000000	data
0x20000008	0x00000000	buffer[0]
0x2000000C	0x00000000	buffer[1]
0x20000010	0x00000000	buffer[2]
0x20000014	0x00000000	buffer[3]

Address of buffer[1]

The C code pt=&buffer[1]; set the pt to point to buffer[1]
The expression &buffer[1] returns 0x2000000C.

address	data	contents
0x20000000	0x2000000C	pt
0x20000004	0x00000000	data
0x20000008	0x00000000	buffer[0]
0x2000000C	0x00000000	buffer[1]
0x20000010	0x00000000	buffer[2]
0x20000014	0x00000000	buffer[3]

Dereference *pt : write

(*pt)=0x04D2; will store 0x04D2 into the place pointed to by pt.
it stores 0x04D2 into buffer[1].
the *pt means "the 32-bit signed integer at 0x2000000C"
(*pt)=0x04D2; sets buffer[1] to 0x04D2.

address	data	contents
0x20000000	0x2000000C	pt
0x20000004	0x00000000	data
0x20000008	0x00000000	buffer[0]
0x2000000C	0x000004D2	buffer[1]
0x20000010	0x00000000	buffer[2]
0x20000014	0x00000000	buffer[3]

Dereference *pt : read

data=(*pt); read memory from address pointed to by pointer pt
into the place pointed to by data.
stores 0x04D2 into data
data=(*pt); sets data to 0x04D2
copies the 32-bit information from buffer[1] into data.

address	data	contents
0x20000000	0x2000000C	pt
0x20000004	0x000004D2	data
0x20000008	0x00000000	buffer[0]
0x2000000C	0x000004D2	buffer[1]
0x20000010	0x00000000	buffer[2]
0x20000014	0x00000000	buffer[3]

Assembly listing

```
    48:         pt = &buffer[1];
0x000003C4      4806            LDR         r0,[pc,#24]  ; @0x000003E0      // &buffer[1]
0x000003C6      4907            LDR         r1,[pc,#28]  ; @0x000003E4      // pt
0x000003C8      6008            STR         r0,[r1,#0x00]

    49:         *pt = 1234;
0x000003CA      F24040D2        MOVW        r0,#0x4D2
0x000003CE      6809            LDR         r1,[r1,#0x00]
0x000003D0      6008            STR         r0,[r1,#0x00]

    50:         data = *pt;
0x000003D2      4804            LDR         r0,[pc,#16]  ; @0x000003E4
0x000003D4      6800            LDR         r0,[r0,#0x00]
0x000003D6      6800            LDR         r0,[r0,#0x00]
0x000003D8      4903            LDR         r1,[pc,#12]  ; @0x000003E8
0x000003DA      6008            STR         r0,[r1,#0x00]

    51:         return 1;
0x000003DC      2001            MOVS        r0,#0x01

    52: }
0x000003DE      4770            BX          lr
```

Types of Memory and Usages

Memory	When power is removed	Ability to Read/Write	Program cycles
RAM	volatile	random and fast access	infinite
battery-backed RAM	nonvolatile	random and fast access	infinite
EEPROM	nonvolatile	easily reprogrammed	***
Flash	nonvolatile	easily reprogrammed	***
OTP PROM	nonvolatile	can be easily programmed	once
ROM	nonvolatile	programmed at the factory	once

Pointer Comparison

```
short *pt1;                    /* define 16-bit integer pointer */
short *pt2[10];                /* define ten 16-bit integer pointers */

short done(void) {             /* returns true if pt1 is higher than pt2[5] */
    if (pt1 > pt2[5]) return(1);
    return(0);
}
```

FIFO (1)

```
#define FifoSize 10                  /* Pointer implementation of the FIFO */
char *PUTPT;                         /* Number of 8 bit data in the Fifo */
char *GETPT;                         /* Pointer of where to put next */
                                     /* Pointer of where to get next */
                                     /* FIFO is empty if PUTPT=GETPT */
                                     /* FIFO is full if PUTPT+1=GETPT */
char Fifo[FifoSize];                 /* The statically allocated fifo data */

void InitFifo(void) {
   PUTPT=GETPT=&Fifo[0];             /* Empty when PUTPT=GETPT */
}
```

FIFO (2)

```
int PutFifo (char data) {
   char *Ppt;                                   /* Temporary put pointer */

   Ppt=PUTPT;                                   /* Copy of put pointer */
   *(Ppt++)=data;                               /* Try to put data into fifo */
   if (Ppt == &Fifo[FifoSize]) Ppt = &Fifo[0];  /* Wrap */
   if (Ppt == GETPT ){
      return(0);}                               /* Failed, fifo was full */
   else{
      PUTPT=Ppt;
      return(-1);                               /* Successful */
   }
}
```

FIFO (3)

```
int GetFifo (char *datapt) {
    if (PUTPT== GETPT){
        return(0);}                                  /* Empty if PUTPT=GETPT */
    else{
        *datapt=*(GETPT++);
        if (GETPT == &Fifo[FifoSize])
            GETPT = &Fifo[0];
        return(-1);
    }
}
```

IO Port Access

```
#define NVIC_ST_CTRL_R          (*((volatile unsigned long *) 0xE000E010))
#define NVIC_ST_RELOAD_R        (*((volatile unsigned long *) 0xE000E014))
#define NVIC_ST_CURRENT_R       (*((volatile unsigned long *) 0xE000E018))
#define GPIO_PORTF_DATA_R       (*((volatile unsigned long *) 0x400253FC))

// The delay parameter is in units of the 80 MHz core clock. (12.5 ns)

void SysTick_Wait(unsigned long delay) {
  unsigned long data;

  NVIC_ST_RELOAD_R = delay-1;                    // number of counts to wait
  NVIC_ST_CURRENT_R = 0;                         // any value written to CURRENT clears
  data = GPIO_PORTF_DATA_R;
  while ((NVIC_ST_CTRL_R & 0x00010000)==0) { }   // wait for count flag
}
```

IO Port Access

```
#define PA5   (*((volatile unsigned long *) 0x40004080))

void wait(void){
  while ((PA5 & 0x20)==0) { };
}

void wait2(void){
  while ((*((volatile unsigned long *) 0x40004080)) & 0x20)==0) { };
}

void wait3(void){
  volatile unsigned long *pt;

  pt = ((volatile unsigned long *) 0x40004080);
  while (((*pt) & 0x20)==0) { };
}
```

References

[1] Essential C, Nick Parlante
[2] Efficient C Programming, Mark A. Weiss
[3] C A Reference Manual, Samuel P. Harbison & Guy L. Steele Jr.
[4] C Language Express, I. K. Chun
[5] “A Whirlwind Tutorial on Creating Really Teensy ELF Executables for Linux“
http://cseweb.ucsd.edu/~ricko/CSE131/teensyELF.htm
[6] http://en.wikipedia.org
[7] http://www.muppetlabs.com/~breadbox/software/tiny/teensy.html
[8] http://csapp.cs.cmu.edu/public/ch7-preview.pdf