Embedded Programming using the GNU Toolchain

Vijay Kumar B <vijaykumar@zilogic.com>

1. Scenario: Add Data in Registers

Listing 1. Assembly Source

```
        .thumb
        .syntax unified

sp:     .word 0x200
reset:  .word start+1

start:
        mov r0, #4
        mov r1, #5
        add r2, r1, r0

stop:   b stop
```

Listing 2. Commands

```
$ arm-none-eabi-as -mcpu=cortex-m3 add.s -o add.o
$ arm-none-eabi-ld -Ttext=0x0 -o add.elf add.o
$ arm-none-eabi-objcopy -O binary add.elf add.bin
```

Listing 3. Qemu Monitor Commands

```
$ qemu-system-arm -M lm3s811evb -kernel add.bin -monitor stdio
info registers
xp /10i 0x9
```

2. Scenario: Add Data in Memory

Listing 4. Assembly Source

```
        .syntax unified
        .thumb

        .data

num1:   .word 0x10
num2:   .word 0x20
result: .word 0x0
```

```
        .text

sp:     .word 0x200
reset:  .word start+1

start:
        ldr r0, =sdata          @ Load the address of sdata
        ldr r1, =edata          @ Load the address of edata
        ldr r2, =etext          @ Load the address of etext

copy:   ldrb r3, [r2]           @ Load the value from Flash
        strb r3, [r0]           @ Store the value in RAM
        add  r2, r2, #1         @ Increment Flash pointer
        add  r0, r0, #1         @ Increment RAM pointer
        cmp  r0, r1             @ Check if end of data
        bne  copy               @ Branch if not end of data

        ldr r0, =num1           @ Load the address of num1
        ldr r1, [r0]            @ Load the value in num1

        ldr r0, =num2           @ Load the address of num2
        ldr r2, [r0]            @ Load the value in num2

        add r3, r1, r2          @ Add num1 and num2

        ldr r0, =result         @ Load the address of result
        str r3, [r0]            @ Store the value in result

stop:   b stop
```

Listing 5. Linker Script

```
MEMORY
{
    FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 0x00010000
    SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 0x00002000
}

SECTIONS {
        .text : {
                * (.text);
                etext = .;
        } > FLASH

        .data : {
                sdata = .;
                * (.data);
                edata = .;
        } > SRAM AT> FLASH
}
```

Listing 6. Commands

```
$ arm-none-eabi-as -mcpu=cortex-m3 add-ram.s -o add-ram.o
$ arm-none-eabi-ld -T add-ram.lds -o add-ram.elf add-ram.o
$ arm-none-eabi-objcopy -O binary add-ram.elf add-ram.bin
```

Listing 7. Qemu Monitor Commands

```
$ qemu-system-arm -M lm3s811evb -kernel add-ram.bin -monitor stdio
xp /1xw 0x20000000
xp /1xw 0x20000004
xp /1xw 0x20000008
```

3. Scenario: Add Data in Memory from C

Listing 8. Assembly Source

```
char stack[1024];

extern char sdata;
extern char edata;
extern char etext;

extern char sbss;
extern char ebss;

static char *bssp = (char *)0xDEADBEEF;

void start()
{
        char *from, *to;

        /* Initialize .data */
        from = &etext;
        to = &sdata;

        while (to != &edata) {
                *to++ = *from++;
        }

        /* Clear .bss */
        bssp = &sbss;
        while (bssp != &ebss) {
                *bssp++ = 0;
        }

        main();

        while (1);
}

__attribute__ ((section(".vectors")))
void *vectors[] = {
        stack + sizeof(stack),
        start,
};

int num1 = 0x40;
int num2 = 0x50;
int result = 0x0;

int main()
{
        result = num1 + num2;
        return 0;
}
```

Listing 9. Linker Script

```
MEMORY
{
    FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 0x00010000
    SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 0x00002000
}

SECTIONS {
        .text : {
                * (.vectors);
                * (.text);
                etext = .;
        } > FLASH

        .data : {
                sdata = .;
                * (.data);
                edata = .;
        } > SRAM AT> FLASH

        .bss : {
                sbss = .;
                * (.bss);
                ebss = .;
        } > SRAM

        .rodata : {
                * (.rodata);
        } > FLASH
}
```

Listing 10. Commands

```
$ arm-none-eabi-gcc -mthumb -mcpu=cortex-m3 -c cadd.c -o cadd.o
$ arm-none-eabi-ld -T cadd.lds -o cadd.elf cadd.o
$ arm-none-eabi-objcopy -O binary cadd.elf cadd.bin
```

4. Further Reading

- Embedded Programming with the GNU Toolchain - http://www.bravegnu.org/gnu-eprog/
- GNU Assembler Manual
- GNU Linker Manual

5. Related Links

- GNU Toolchain - http://www.codesourcery.com/gnu_toolchains/arm
- Qemu: http://www.nongnu.org/qemu/download.html