

Software-Toolchain

1. Software-Toolchain nutzen

Um Programme auf den Softcore-Prozessor mit dem FPGArduino laufen zu lassen, wird dem Compiler ein Linkerskript und den startup-Code in Assembler übergeben.

Linkerskript:

Die Aufgabe eines Linker-Skript ist zu beschreiben wie die Segmente im Speicher abgebildet werden und definiert wo der Programmstart `ENTRY(_start)` ist.

.text – Code
.data – Initialisierte Variablen
.rodata – Read-Only-Variablen

Beispiel: link.ld (von spu32-Softcore):

```
OUTPUT_ARCH( "riscv" )
ENTRY( _start )
SECTIONS
{
    /* text: test code section */
    . = 0x00000000
    .text : { *(.text) }
    /* data: Initialized data segment */
    .data : { *(.data) }
    /* End of uninitialized data segment */
    _end = .;
}
```

crt0:

Die crt0 enthält die Programminitialisierungsfunktionen eines Programms. Dazu gehören Aufgaben wie initialisieren globaler Variablen, Anlegen des Stacks und den Sprung auf main.

Beispiel: crt0.s (von spu32-Softcore)

```
.section .text
.global _start
_start:
    # reset vector at 0x0
    . = 0x0
    j _init

# interrupt handler
.= 0x10
_interrupt:
    # for now just do an endless loop
    j _interrupt

_init:
    # set up stack pointer
    li sp, 4096
    # call main function
    jal ra, main
    # back to start
    j _start
```

2. C-Programm für den SPU32-Softcore compilieren und einbinden

```
led.c:
#include <stdint.h>
#include <stdbool.h>

#define reg_leds (*(volatile uint8_t*)0xFFFFFFFF0)

void main() {
    uint32_t led = 0;

    while (1) {
        reg_leds = (led >> 16) & 0xFF;
        led = led + 1;
    }
}
```

Generieren der Objektdatei:

```
riscv64-unknown-elf-gcc -march=rv32i -mabi=ilp32 -static -nostdlib -fno-builtin-printf -Os -fPIC -c led.c
```

Einbinden von crt0.s und link.ld:

```
riscv64-unknown-elf-gcc -march=rv32i -mno-div -mabi=ilp32 -static -nostdlib -fno-builtin-printf -Os -fPIC -fdata-sections -ffunction-sections -o led.elf crt0.s led.o -Tlink.ld -Xlinker --gc-sections
```

Generieren der bin Datei:

```
riscv64-unknown-elf-objcopy -O binary led.elf led.bin
```

Generieren der dat Datei:

```
hexdump -v -e '1/1 "%02x" "\n"' led.bin > led.dat
```

Anmerkung: -march: Gibt durch Angabe der Architektur vor, welche Befehle und Register zur Verfügung stehen.

-mabi: Gibt mit der ABI die Aufrufkonventionen vor.

Einbinden in den SPU32:

In der *top.v* unter *boards/fpgarduino* wird der Pfad zum Programm angegeben.

```
rom_wb8 #(
    .ROMINITFILE("./software/asm/led.dat")
) rom_inst (
    .CLK_I(clk),
    .STB_I(rom_stb),
    .ADR_I(cpu_adr[8:0]),
    .DAT_I(cpu_dat),
    .DAT_O(rom_dat),
    .ACK_O(rom_ack)
);
```

Abbildung 1: Einbinden der dat Datei in der top.v

Erklärungen zu Argumenten:

- march=rv32im:32, 32-bit general-purpose integer registers + M extension
- mabi=ilp32:int, long, and pointers are all 32-bits long. long long is a 64-bit type, char is 8-bit, and short is 16-bit
- static: Do not link against shared libraries.
- nostdlib, -nostartfiles: Do not use the standard system startup files or libraries when linking.
- fno-builtin-printf: Disables special handling and optimizations of standard C library function printf
- Os: Specifies the level of optimization to use when compiling source files
- fPIC: Generate position-independent code (PIC)
- d: disassemble
- O binary: Write the output file using the object format binary
- x assembler-with-cpp: assembly code contains C directives
- fdata-sections -ffunction-sections: Place each function or data item into its own section in the output file
- Xlinker -gc-sections: Enable garbage collection and remove all unused code

3. Quellen

RISC-V GNU Compiler Toolchain

URL: <https://github.com/riscv/riscv-gnu-toolchain>

The -march, -mabi, and -mtune arguments to RISC-V Compilers URL:

<https://www.sifive.com/blog/all-aboard-part-1-compiler-args>

GCC Command Options URL:

URL: <https://gcc.gnu.org/onlinedocs/gcc/Invoking-GCC.html>

Using LD, the GNU linker - Linker Scripts URL:

https://scgberlin.de/content/media/http/informatik/gcc_docs/ld_3.html

crt0

URL: <https://de.wikipedia.org/wiki/Crt0>

spu32-Softcore crt.s und link.ld URL:

<https://github.com/maikmerten/spu32/blob/master/software/c-firmware/crt0.s>

<https://github.com/maikmerten/spu32/blob/master/software/asm/link.ld>