Custom Core Software Getting Started Guide

Version 1.1

Custom Core Software Getting Started Guide

Proprietary Notice

Release Information

Version	Date	Changes
Version 1.0	March 11, 2019	• Initial release
Version 1.1	June 20, 2019	• Small fixes, graphical detail, code size options

Contents

Chapter 1

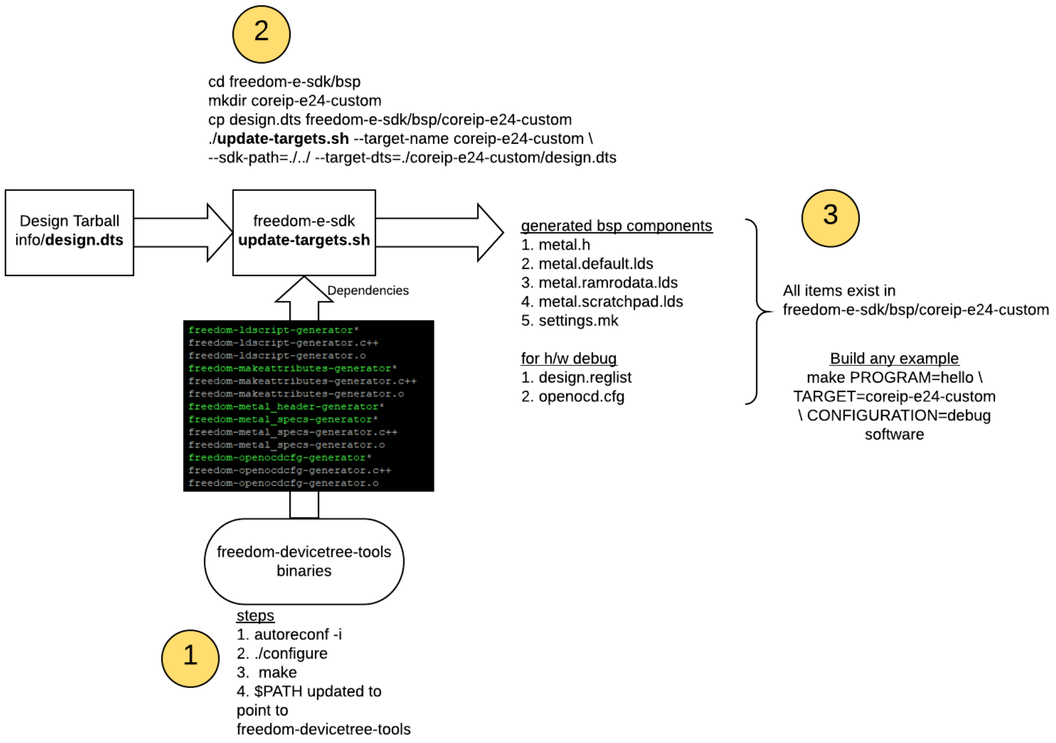
Custom Core Software Development Getting Started Guide

1.1 Introduction

SiFive custom core designs are packaged with a Device Tree Specification (**design.dts**) file that is used to describe the hardware. This file is the foundation for generating the linker script (**metal.lds**) and the main header file (**metal.h**) required for software development.

These components can be found in the **/bsp** path within the github repository freedom-e-sdk which supports SiFive's standard cores. For custom cores, a unique bsp is automatically generated and included in the design package, also known as the *design tarball*. This document describes the manual process used to generate new header and linker script files for custom designs.

1.2 High Level Flow Using E24 Custom Core Example



```
cd freedom-e-sdk/bsp
mkdir coreip-e24-custom
cp design.dts freedom-e-sdk/bsp/coreip-e24-custom
./update-targets.sh --target-name coreip-e24-custom \
--sdk-path=./../ --target-dts=./coreip-e24-custom/design.dts
```

```
freedom-ldscript-generator*
freedom-ldscript-generator.c++
freedom-ldscript-generator.o
freedom-makeattributes-generator*
freedom-makeattributes-generator.c++
freedom-makeattributes-generator.o
freedom-metal_header-generator*
freedom-metal_specs-generator*
freedom-metal_specs-generator.c++
freedom-metal_specs-generator.o
freedom-openocdcfg-generator*
freedom-openocdcfg-generator.c++
freedom-openocdcfg-generator.o
```

generated bsp components
1. metal.h
2. metal.default.lds
3. metal.ramrodata.lds
4. metal.scratchpad.lds
5. settings.mk

for h/w debug
1. design.reglist
2. openocd.cfg

All items exist in freedom-e-sdk/bsp/coreip-e24-custom

Build any example
```
make PROGRAM=hello \
TARGET=coreip-e24-custom
\ CONFIGURATION=debug
software
```

steps
1. autoreconf -i
2. ./configure
3. make
4. $PATH updated to point to freedom-devicetree-tools

2
Design Tarball info/design.dts
freedom-e-sdk update-targets.sh
Dependencies
freedom-devicetree-tools binaries
3
1

1.2.1 Detail

The **.dts** file contains information in text format that describes the hardware in detail, including base address and size for memory and peripherals. There is a specification that describes the format of this file. For more information see devicetree.org. This file is an output from the design environment and one exists for every custom core.

The **.h** file contains macros, include files, and structures which declare all instances of memory mapped features of the device.

The **.lds** file contains formatted, named sections for code and data. This file is used by the tool-chain to link compiled code and data sections to the actual hardware memory map.

1.3 Development Flow

Generating the new linker script and header file requires the design.dts file contained within the design tarball, and utilities within two github repositories.

For this example we will reference a new workspace that contains the design tarball and two github repos.

```
my-new-core-tarball
freedom-devicetree-tools
freedom-e-sdk
```

1.3.1 Design Tarball

Extract the tarball for the custom core. The design.dts file is located here and used to generate the **.lds** and **.h** files.

```
        > mkdir my-new-core-tarball
        > cp my_new_tarball_v0p0.tar.gz /path/to/my-new-core-tarball
        > cd /path/to/my-new-core-tarball
        > tar -xvf tarball_v0p0.tar.gz
```

The design.dts file resides in the /info path. We will reference this file in a later step.

1.3.2 Device Tree Tools

Setup the device tree tools repository which contains C++ utilities to generate the various components needed for software development. Refer to the README for the required packages necessary to use the tools. Also note that a C++ 11 compiler is required to build the binaries.

```
        > git clone https://github.com/sifive/freedom-devicetree-tools.git
        > cd freedom-devicetree-tools
        > git submodule update --init --recursive
        > autoreconf -i
```

This step should show output similar to the following

```
        > configure.ac:23: installing './compile'
        > configure.ac:11: installing './install-sh'
        > configure.ac:11: installing './missing'
        > ...
```

Now, run configure

```
        > ./configure
```

You should see the following steps at this point

```
        > checking for a BSD-compatible install... /usr/bin/install -c
        > checking whether build environment is sane... yes
        > checking for a thread-safe mkdir -p... /bin/mkdir -p
        > checking for gawk... gawk
        > checking whether make sets $(MAKE)... yes
        ...
        > checking for dtc... yes
        > checking that generated files are newer than configure... done
        > configure: creating ./config.status
        > config.status: creating Makefile
        > config.status: executing depfiles commands
```

Now run make

```
> make
```

Now the binaries have been built and the **freedom-devicetree-tools** location needs to be added to your sytem PATH.

1.3.3 Freedom E SDK

In this next step, we clone the repository which contains software examples for the SiFive standard cores. This repository also contains the **update-targets.sh** script that uses the Device Tree Tools binaries we built in the previous step to create the custom header file and linker script file.

```
> git clone --recursive https://github.com/sifive/freedom-e-sdk.git
> cd freedom-e-sdk/bsp
> mkdir my-new-core
```

The design.dts file from the tarball should now be moved into /bsp/my-new-core created in the previous step.

```
> cp /path/to/my-new-core-tarball/info/design.dts \
/path/to/freedom-e-sdk/bsp/my-new-core
```

From the freedom-e-sdk/bsp path, show the arguments for the generation script using the following command

```
> ./update-targets.sh --help
```

Generate the bsp components

```
> ./update-targets.sh --target-name my-new-core \
--sdk-path=./../ --target-dts=./my-new-core/design.dts
```

All of the generated bsp components now reside in /path/to/freedom-e-sdk/bsp/my-new-core.

1.4 Create and Build Example

The bsp also requires a settings.mk file. The settings.mk file is used to pass -march (architecture) and -mabi (application binary interface) information to the RISC-V GNU Toolchain. Earlier versions of freedom-devicetree-tools did not automatically generate a settings.mk file. If one does not exist, copy settings.mk from a different freedom-e-sdk/bsp with similar architecture. Check the file contents to ensure the correct architecture is declared in this file for the new design.

We now have the required software components in freedom-e-sdk/bsp/my-new-core to start compiling examples and creating your new projects.

To build an example project

```
> cd freedom-e-sdk
> make PROGRAM=hello TARGET=my-new-core CONFIGURATION=debug software
```

The output file(s) will reside in **/software/debug** or **/software/release** path depending on the build **CONFIGURATION** selected. The **CONFIGURATION** options include `debug` or `release`.

To clean the build, replace 'software' above with 'clean'. To see all build options, use 'make help'.

1.4.1 Build Options

The `debug` configuration uses level zero optimizations by specifying compiler option **-O0**. This default selection does not optimize for code size. The `debug` configuration additionally uses **-g** to include debug symbols in the .elf file, but this does not change the code size sections. The `release` configuration uses the compiler option **-Os** to produce the smallest code footprint. These optimizations can be changed by modifying freedom-e-sdk/debug.mk or freedom-e-sdk/release.mk files.

Options for Code Size

For example, the **-Os** optimization will optimize for size for a possible reduction in performance. Consider **-O2** or **-O3** based on the application needs. Another popular option for smaller code size is to use nano specs library by specifying **--specs=nano.specs**.

The code size summary below shows how the different options directly affect code size for one of the example programs available in the freedom-e-sdk repository. The code size summary can be shown by using the `riscv64-unknown-elf-size` utility which is part of the prebuilt GCC toolchain available on sifive.com.

Level 0 Optimizations

`RISCV_CFLAGS += -O0`

```
riscv64-unknown-elf-size local-interrupt.elf
   text           data           bss           dec           hex
  65001           8888         23884         97773         17ded
```

Use newlib library

`RISCV_CFLAGS += -O0 --specs=nano.specs`

```
riscv64-unknown-elf-size local-interrupt.elf
   text           data           bss           dec           hex
  21409           6496         26280         54185          d3a9
```

Level 2 Optimizations and use smaller newlib library

`RISCV_CFLAGS += -O2 --specs=nano.specs`

```
riscv64-unknown-elf-size local-interrupt.elf
   text           data           bss           dec           hex
  14069           6504         26272         46845          b6fd
```

Options for Code Location

Multiple linker files exist in each bsp for flexibility in choosing the proper configuration based on application requirements. The following options describe the linker memory map options created for each bsp

- **metal.default.lds** places code and data into SPI flash.
- **metal.ramrodata.lds** places read-only data into RAM for higher performance but executes code by fetching instructions from SPI flash.
- **metal.scratchpad.lds** places all code and data into RAM. This provides the best performance, provided enough memory exists for the targeted application.
- Note: The scratchpad option may not compile successfully for applications which require more memory than available on-target.

The default linker file is **metal.default.lds**. To specify a new linker file on the command line

```
> make PROGRAM=hello TARGET=my-new-core \
CONFIGURATION=debug LINK_TARGET=ramrodata software
```

This option will select **metal.ramrodata.lds** as the linker file.

The default build configuration is **debug** so the code size is not optimized. To configure a build for best code size, specify the **release** configuration

```
> make PROGRAM=hello TARGET=my-new-core \
CONFIGURATION=release software
```

1.4.2 Create New Project Space

Now that we have a custom bsp setup, a separate project can be created using the freedom-e-sdk make process. Here we use the `standalone` option which creates a new project in a location you specify, complete with supporting Makefiles.

```
> cd freedom-e-sdk
> make help
```

A portion of the help menu will display the following

```
> standalone STANDALONE_DEST=/path/to/desired/location
>              [PROGRAM=hello] [TARGET=sifive-hifive1]
>     Exports a program for a single target into a standalone
>     project directory at STANDALONE_DEST.
```

To create a standalone project that uses interrupts for the new bsp

```
> make PROGRAM=local-interrupt TARGET=my-new-core \
CONFIGURATION=debug STANDALONE_DEST=/path/to/my/new/proj standalone
```

Executing this command will copy all source code to your new project path specified by STANDALONE_DEST. This includes all of the freedom-metal API code, the specified example, and all makefile components.

To build your new project, navigate to /path/to/my/new/proj and type

```
> make
```

Simply using `make` will use the default build options, including the default linker file and the optimizations defined in the debug.mk file. Addtional command line options include `LINK_TARGET=ramrodata` or `LINK_TARGET=scratchpad`, and `CONFIGURATION=release`. These are the same command line options described in the previous section.

New source code files (.s, .h, .c) can be added to the **/src** path in this new project space and the will be included automatically by the make files. This new project space can be used for future project development.

1.4.3 Freedom Studio IDE

It is important to note that for users who prefer an integrated IDE for editing, compiling, and debugging, the Eclipse based IDE Freedom Studio is available on SiFive.com. Freedom Studio is packaged with a prebuilt toolchain, OpenOCD/GDB debugger, and a tagged release of the freedom-e-sdk repository, which provides the platform for example code and the flexible freedom-metal API.

Freedom Studio has a new project creation process that leverages the **standalone** make option within freedom-e-sdk, described previously. New project creation within Freedom Studio is as easy as clicking a button. Refer to the Freedom Studio Manual available at https://www.sifive.com/documentation. This is the quickest way to get up an running on SiFive hardware.