Custom Core Software Getting Started Guide

Version 1.0

Custom Core Software Getting Started Guide

Proprietary Notice

Release Information

Version	Date	Changes
Version 1.0	March 11, 2019	• Initial release

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. This document describes the process used to generate new header and linker script files for custom designs.

1.1.1 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.

1.2 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
```

1.2.1 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.

1.2.2 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
> git submodule upate --init --recursive
> cd freedom-devicetree-tools
> 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.

1.2.3 Freedom E SDK

Clone the repository which contains software examples for the SiFive standard cores, and 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 linker script file and the new header file

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

The output linker and header files will now reside in /path/to/freedom-e-sdk/bsp/my-new-core.

1.3 Create and Build the New bsp within Freedom E SDK

Before a project can be built for the new design, a settings.mk file needs to be added. The settings.mk file is used to pass -march and -mabi arguments to the RISC-V GNU Toolchain.

This can be copied from another freedom-e-sdk/bsp example 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 BSP=metal 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.

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

1.3.1 Create New Example Project

Now that we have a custom bsp setup, a new project can be created within freedom-e-sdk using the 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
>             [INCLUDE_METAL_SOURCES=1] [PROGRAM=demo_gpio]
>             [TARGET=freedom-e300-hifive1]:
>     Export 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 \
INCLUDE_METAL_SOURCES=1 standalone
```

The INCLUDE_METAL_SOURCES=1 is optional but it will copy all source code to your new project path. Use this option to enable further development at this new location.

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

```
> make
```

Now the new project can be customized and development can begin here.