

FP-GAME

Build from Source Guide

This guide details the process of building the .img file used for FP-GAME using the build-script included in the FP-GAME source repo. Additionally, instructions on how to flash your SD card, taken from FP-GAME Getting Started document are repeated at the end of this document for convenience.

Document Overview

[Download FP-GAME Source Repository](#)

[Synthesize FP-GAME FPGA](#)

[Install Toolchain](#)

[Build Linux zImage](#)

[Build Device Tree Blob](#)

[Build FP-GAME Kernel Modules](#)

[Download Terasic Console Image](#)

[Run Build-Script](#)

[Build User Libraries](#)

Download FP-GAME Source Repository

There is a good chance that if you are reading this document, you have also downloaded the FP-GAME User Repository (<https://github.com/FP-GAME/fpgame-usr>). This repo only contains files that the common user needs to run FP-GAME and build/play their own games.

If, however, you wish to modify or explore the FP-GAME hardware, kernel module, or user-library implementations, then you must download the FP-GAME Source Repository:

Run `git clone https://github.com/FP-GAME/fpgame-src.git`

Synthesize FP-Game FPGA

Note, if you have not already done so, please install Quartus. A link to Quartus download can be found below. We have used Quartus Prime Version 20.1.1 Lite Edition.

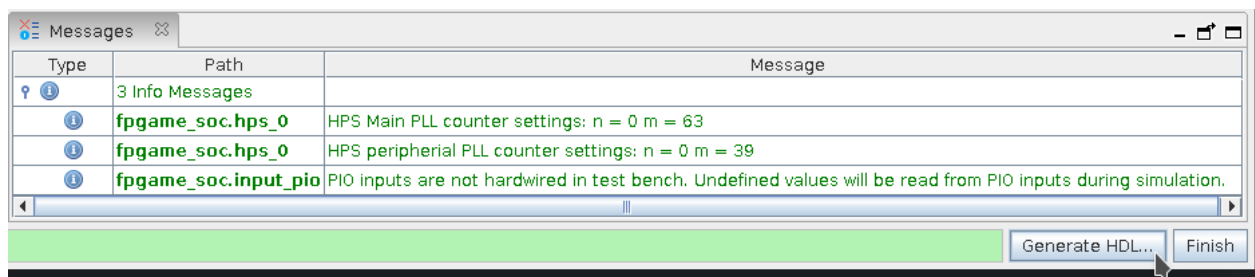
<https://fpgasoftware.intel.com/?edition=lite>

In the FP-Game source repo, `cd fpgame` and run `quartus fpgame &` to open the FP-Game hardware Quartus project.



First, we want to generate all of the IP files from Platform Designer. To do this, click the Platform Designer icon, or in the top menu, select “Tools/Platform Designer”.

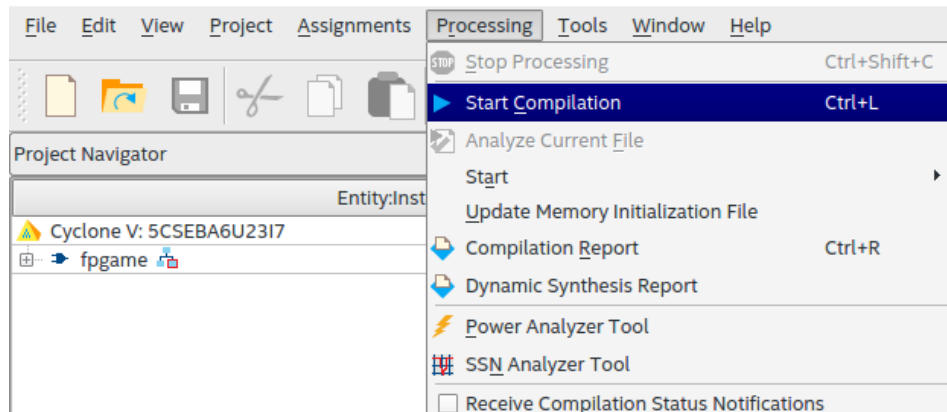
A file dialog should open. Select `fpgame_soc.qsys`.



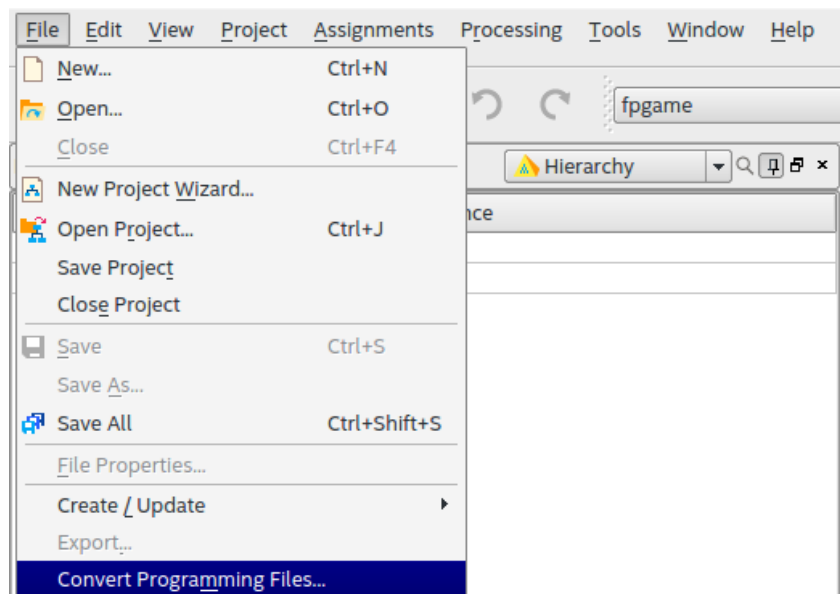
At the bottom of the newly opened window, click “Generate HDL...”.

A new window will open with some settings. Use the default settings and click “Generate”.

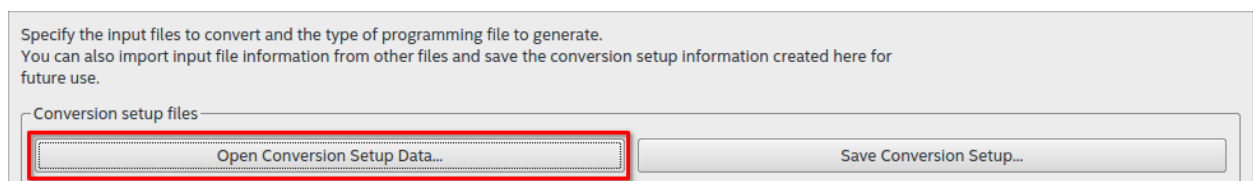
When Platform Designer finishes generating HDL files, close out the Platform Designer windows and return to the main Quartus view.



In the Top Menu, go to “Processing/Start Compilation” or press CTRL-L. This will begin the compilation process. This will take 15 mins or more on good hardware. Expect warnings, but no errors.

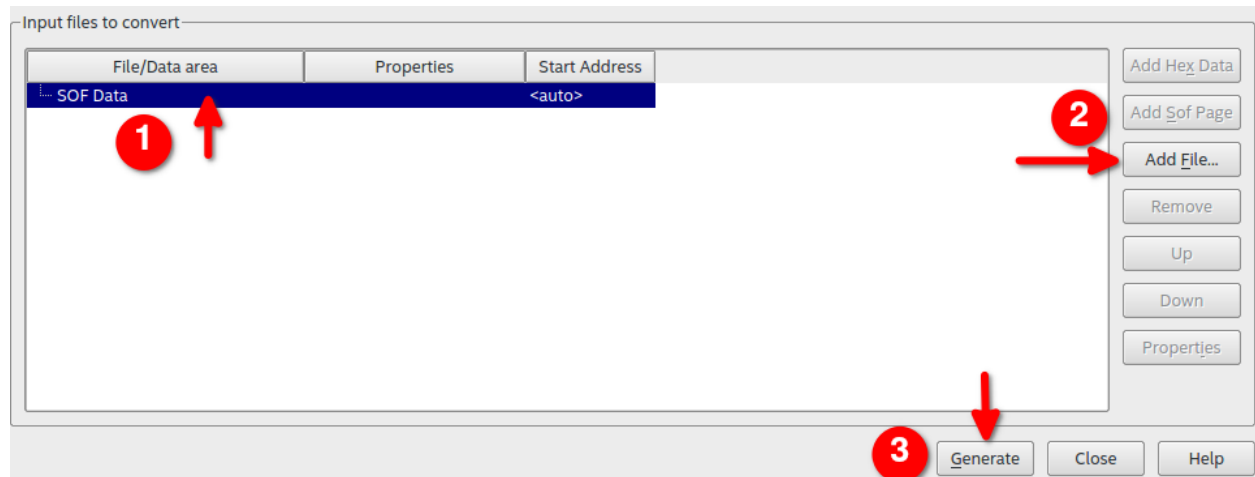


Once the compilation process is finished, go to “File/Convert Programming Files”.



In the newly opened window, click “Open Conversion Setup Data”

In the fpgame folder, we have included an “output_file.cof” file to be used as the Conversion Setup File. Choose this file.



Next, under “SOF Data” in the Convert Programming File window:

1. Click on the “SOF Data”
2. Click on “Add File”. Choose the “fpgame.sof” file under the fpgame folder. This file should have been generated in the compilation step from earlier.
3. Click on “Generate”.

Congratulations! The .rbf (bitstream) file used for programming the FPGA has been generated. These instructions must be repeated whenever the FP-Game hardware configuration has been modified.

Install Toolchain

A copy of these instructions are also available in the user repo (game_development_guide.pdf).

FP-GAME uses a prebuilt arm-linux-none-gnueabihf toolchain from ARM. The toolchain can be found at the following link:

<https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-a/downloads>

Download the “AArch32 target with hard float (arm-linux-none-gnueabihf)” .tar.xz file. Our download was titled: “gcc-arm-10.2-2020.11-x86_64-arm-none-linux-gnueabihf.tar.xz”. However, you may be presented with a different version depending on the time of reading. Download the version available to you. Note, the remainder of these instructions will assume the version listed above, so be sure to change any references to the toolchain to your specific version in the commands you run.

We recommend keeping a folder to contain your separate toolchains. For the remainder of these instructions, we will assume that you have a folder `~/toolchains` to store the extracted toolchain folders. If you want to create one, run `mkdir ~/toolchains` and follow the instructions below to extract your newly downloaded toolchain:

```
cd <path to your downloads folder>
tar -xvf gcc-arm-10.2-2020.11-x86_64-arm-none-linux-gnueabihf.tar.xz \
-C ~/toolchains
```

To enable Makefiles to see your new toolchain, you should add the following to your `.bashrc` (or `.zshrc`, or whatever you use):

```
# Toolchain for Cyclone V SoC (FP-GAME)
export FPGAME_TC="$HOME/toolchains/\
gcc-arm-10.2-2020.11-x86_64-arm-none-linux-gnueabihf"
export PATH="$PATH:$FPGAME_TC/bin"
```

Ensure that you `source ~/.bashrc` (or `.zshrc`) or restart your terminal session so that you get the path changes and environment variables.

With this done, you are ready to continue the build-from-source process.

Build Linux zImage

Prerequisites:

- You must have the toolchain installed (see [Install Toolchain](#)).

Before attempting to run any Makefiles for building the Linux Kernel, run the commands:

```
export ARCH=arm
export CROSS_COMPILE="$FPGAME_TC/bin/arm-none-linux-gnueabihf-"
```

Now ensure you are in the root directory of the FP-GAME source repository. We will download the linux-socfpga kernel source and checkout the version we use for FP-GAME:

```
git clone https://github.com/altera-opensource/linux-socfpga
cd linux_socfpga
git checkout rel_socfpga-5.9_21.03.02_pr
```

Warning, you must clone it in the root directory of the FP-GAME source repository, or else the symlinks in Library/ and Kernel/ will not work.

Inside the linux_socfpga folder:

```
make socfpga_defconfig
make menuconfig
```

A GUI will open up. Follow these steps to configure the kernel to build:

1. Disable “General Setup”->“Automatically append version information to version string”.

```
[ ] Compile also drivers which will not load
[ ] Local version - append to kernel release
[*] Automatically append version information to the version string
[ ] Build ID Salt
    Kernel compression mode (Gzip) --->
[ ] Default init path
((none)) Default hostname
[*] Support for paging of anonymous memory (swap)
[*] System V IPC
[ ] POSIX Message Queues
[ ] General notification queue
[*] Enable process_vm_readv/writev syscalls
[ ] uselib syscall
[ ] Auditing support
    IRQ subsystem --->
    Timers subsystem --->
    Preemption Model (No Forced Preemption (Server)) --->
    CPU/Task time and stats accounting --->
[*] CPU isolation
    RCU Subsystem --->
<*> Kernel .config support
[*] Enable access to .config through /proc/config.gz
< > Enable kernel headers through /sys/kernel/kheaders.tar.xz
(14) Kernel log buffer size (16 => 64KB, 17 => 128KB)
(12) CPU kernel log buffer size contribution (13 => 8 KB, 17 => 128KB)
(13) Temporary per-CPU printk log buffer size (12 => 4KB, 13 => 8KB)
Scheduler features ----
[*] Control Group support --->
[*] Namespaces support --->
[ ] Checkpoint/restore support
[ ] Automatic process group scheduling
[ ] Enable deprecated sysfs features to support old userspace tools
v(+)
```

<Select> <Exit> <Help> <Save> <Load>

2. Go back to the starting menu by double pressing the “ESC” key.
3. Next go to “Enable the block layer” and enable
 - a. “Block layer SG support v4”,
 - b. “Block layer SG support v4 helper lib”,
 - c. and “Block layer data integrity support”.



```
--- Enable the block layer
-*- Block layer SG support v4
[*] Block layer SG support v4 helper lib
[*] Block layer data integrity support
[ ] Zoned block device support
[ ] Block device command line partition parser
[ ] Enable support for block device writeback throttling
[*] Block layer debugging information in debugfs
[ ] Logic for interfacing with Opal enabled SEDs
[ ] Enable inline encryption support in block layer
Partition Types --->
```

<=select> < Exit > < Help > < Save > < Load >

4. Save the config as .config and exit.

Now, compile the kernel with the following command. Note that there is a space between "LOCALVERSION=" and "zImage"

```
make ARCH=arm LOCALVERSION= zImage
```

This will take a while. Once it is done, congratulations! You have built the kernel zImage. We recommend that while you are in the linux-socfpga folder you also follow the instructions below build the device tree blob.

Build Device Tree Blob

Prerequisites:

- You must have already built the Linux zImage (see [Build Linux zImage](#)).
- You must have already installed the toolchain (see [Install Toolchain](#)).

In the downloaded linux-socfpga subfolder of the FP-GAME source repository, run the following commands:

```
export ARCH=arm
export CROSS_COMPILE="$FPGAME_TC/bin/arm-none-linux-gnueabihf-"
ln -s ../../../../../../Kernel/kern/dts/fpgame_int.dts arch/arm/boot/dts
make fpgame_int.dtb
```

Congratulations, you've built the device tree blob for FP-GAME.

Build FP-GAME Kernel Modules

Prerequisites:

- You must have already built the Linux zImage (see [Build Linux zImage](#)).
- You must have already installed the toolchain (see [Install Toolchain](#)).

In the root directory of the FP-GAME source repository, `cd Kernel`.

In order to get the kernel modules to compile, we must first run:

```
ln -s ../../../../Kernel/kern/inc/fp-game ../linux-socfpga/include/linux/
```

Ensure you have run the following command before attempting to build the Kernel modules:




```
export ARCH=arm  
export CROSS_COMPILE="$FPGAME_TC/bin/arm-none-linux-gnueabihf-"
```

Simply run `make` and you are done! Congratulations, the kernel modules for FP-GAME have been built.

Download Terasic Console Image

Download the Linux Console (kernel X.X) file from the following link:

<https://www.terasic.com.tw/cgi-bin/page/archive.pl?No=1046&PartNo=4>

Title	Version	Size(KB)	Date Added	Download
Linux Console (kernel 4.5)	1.3		2018-03-15	
Linux Xfce Desktop (kernel 4.1.33-Itsi-altera)	1.0		2017-04-11	
Linux LXDE Desktop (kernel 4.5)	1.1		2017-04-10	

At the time of writing, the Linux Console download version was 1.3, kernel version 4.5, as shown above.

Once downloaded, copy the de10_nano_linux_console.img file to the img_src/ directory under the FP-Game source repository root folder.

Run Build-Script

To ensure this step runs successfully, you need to have the following files generated and located in the specified directories under the FP-Game source repo:

File/Source	Location
de10_nano_linux_console.img	img_src/
soc_system.rbf	fpga/
zImage	linux-socfpga/arch/arm/boot/
fpga_int.dtb	linux-socfpga/arch/arm/boot/dts/
apu.ko	Kernel/apu/
con.ko	Kernel/con/
ppu.ko	Kernel/ppu

Note, building the user libraries is not necessary for this step.

In the root of the FP-Game Source repo, run `sudo ./build-script.sh <.img mount point>`, specifying a mount point, like “/mnt/imgmnt” as the place to temporarily mount the image file.

If this completes successfully, you can copy the newly modified de10_nano_linux_console.img file in img_src to your SD Card.

To do this, you can use:

```
sudo dd bs=1M if=img_src/de10_nano_linux_console.img of=/dev/<YOUR SD CARD>
```

BE VERY CAREFUL NOT TO OVERWRITE YOUR HARD-DRIVES.

Find <YOUR SD CARD> by running `lsblk` with and without your SD card plugged in, and seeing which device is added and removed.

Build User Libraries

Prerequisites:

- You must have already installed the toolchain (see [Install Toolchain](#)).
- If you want to build documentation, install doxygen (optional).

This step is not mandatory for building the .img file for flashing to the DE10-Nano's SD card. If you made changes to the FP-Game user library and would like to use them in your own game, then follow these steps.

In the root folder of the FP-Game source repository, `cd Library`.

Simply run `make` to generate the "usr" folder.

Copy the "usr" folder in the "Library" directory to your own game project. The Makefile and config we provide in the FP-Game user repository searches this folder for the FP-Game library.

If you want to build the documentation as well, run `doxygen Doxyfile` and copy the resulting files in docs/html to wherever you want to place the new documentation.

Congratulations, you have successfully built the FP-Game user library.