There is a lot of output there but the interesting things to note are:

- **--with-sysroot**=/home/chris/x-tools/arm-cortex_a8-linux- gnueabihf/arm-cortex_a8-linux-gnueabihf/sysroot: This is the default sysroot directory
- **--enable-languages**=c,c++: Using this we have both C and C++ languages enabled
- **--with-arch=armv7-a:** The code is generated using the ARM v7a instruction set
- **--with-cpu=cortex-a8** and --with-tune=cortex-a8: The the code is further tweaked for a Cortex A8 core
- **--with-float=hard**: Generate opcodes for the floating point unit and uses the VFP registers for parameters
- **--enable-threads=posix**: Enable POSIX thread

These are the default settings for the compiler. You can override most of them on the gcc command line so, for example, if you want to compile for a different CPU, you can override the configured setting, –-with-cpu, by adding -mcpu to the command line, as follows:

```
$ arm-cortex_a8-linux-gnueabihf-gcc -mcpu=cortex-a5 helloworld.c -o helloworld
```

You can print out the the range of architecture-specific options available using --target-help, as follows:

```
$ arm-cortex_a8-linux-gnueabihf-gcc --target-help
```

You may be wondering if it matters whether or not you get the exact configuration right at the time you generate the toolchain if you can change it later on, and the answer depends on the way you anticipate using it. If you plan to create a new toolchain for each target, then it makes sense to set everything up at the beginning because it will reduce the risks of getting it wrong later on. Jumping ahead a little to Chapter 6, Selecting a Build System, I call this the Buildroot philosophy. If, on the other hand, you want to build a toolchain that is generic and you are prepared to provide the correct settings when you build for a particular target, then you should make the base toolchain generic, which is the way the Yocto Project handles things. The preceding examples follow the Buildroot philosophy.