# Tiger (Bosch)

Introduction to the Bosch project

# Background

- Small embedded CPU called 'Cronus' used for sensors
- We're replacing their existing tools
- They have provided a reference simulator, but basically nothing else

# Status (1)

- We have a basic tool chain: a compiler/assembler, a linker, a debugger, a debug server and a simulator
- The tools are functional, but are missing a lot of features

# Status (2)

- We have started running tests, and now we need to triage the failures.

```
gcc summary:

# of expected passes            26087
# of unexpected failures        6140
# of unresolved testcases       2331
# of unsupported tests          6415
```

# Tool chain components

| Component | Command | Description |
|---|---|---|
| Clang/LLVM | `bsc-elf-clang` | Compiler and assembler |
| GNU ld | `bsc-elf-ld` | Linker |
| GDB | `bsc-elf-gdb` | Debugger (gdb) |
| Embdebug | `embdebug` | Debug server |
| Embdebug LLVM ISS target | | LLVM-based sim target for Embdebug |
| LLVM-ISS | `bsc-run` | Standalone LLVM-based simulator |

Note: **bsc** (**B**osch **S**ensor **C**ore) is the family of CPUs, one of which is 'cronus'. **elf** is the format of the object files.

# Project organization

There's a Tiger team on the Embecosm Mattermost (https://mattermost.embecosm.com/tiger-internal) that you should have been invited to, if anything in these slides doesn't make sense or you need help then one of us should normally be able to answer questions on there.

# Getting started

- Project directory structure
- Cloning the respositories
- Checking out branches
- Building the tool chain
- Building a program with the tool chain
- Running a program on the LLVM-ISS
- Running a program from the debugger

# Getting started - Project directory structure

Create a directory for the project to hold all of the repositories.

All of the git repositories that make up the project will be cloned into subdirectories in this main project directory.

# Getting started - Cloning the repositories (1)

The code is hosted on https://tiger.git.embecosm.com, you should (now) have an account on this server.

The next slide has the repositories that you will need to clone, where you need to clone them to, and the branch you will need to check out. Commands to do this will follow.

# Getting started - Cloning the repositories (2)

| Repository | Subdirectory | Branch |
|---|---|---|
| toolchain | toolchain/ | master |
| binutils-gdb | binutils/ | tiger-2.34-binutils |
| binutils-gdb | gdb/ | tiger-9.2-gdb |
| llvm-project | llvm-project/ | tiger-11.0.0-llvm |
| newlib | newlib/ | tiger-3.3.0-newlib |
| embdebug | embdebug/ | master |
| bsc-embdebug-target | bsc-embdebug-target/ | master |
| bsc-embdebug-sysc-target | bsc-embdebug-sysc-target/ | master |
| gcc-for-llvm-testing | gcc-for-llvm-testing/ | llvm-testing |

# Getting started - Cloning the repositories (3)

You can clone with HTTPS using the following command (this will require a username and password):

```
git clone https://tiger.git.embecosm.com/tiger/<repository name>.git <destination directory>
```

Or you can clone with SSH (this requires you to first upload the public half of your SSH key to your profile on the git server):

```
git clone ssh://git@tiger.git.embecosm.com:2231/tiger/<repository name>.git <destination directory>
```

# Getting started - Checkout out branches

Once the repositories have been cloned you can check out the appropriate branch by changing directory to the subdirectory and then doing a `git checkout`:

```
cd <subdirectory>
git checkout <branch>
```

# Getting started - Building the tool chain

Once all of the components of the tool chain are cloned, and the appropriate branches have been checked out, we need to build the tool chain. There is a script to acheive this in the `toolchain` directory.

```
cd toolchain
./build-toolchain.sh --debug --build-newlib --build-compiler-rt --build-embdebug
```

This will build all of the commands listed earlier (hopefully)

# Getting started - Building a program with the tool chain (1)

Once the tool chain is built, the commands for the components can be found in the directory `<project>/install/bin/`, where `<project>` is the top level directory containing all of the project components.

# Getting started - Building a program with the tool chain (2)

In order to use any of the commands we need to put them on our path:

```
PATH=<project>/install/bin:$PATH
```

Now you should be able to run the command:

```
bsc-elf-clang --version
```

And get some output.

# Getting started - Building a program with the tool chain (3)

We should be able to build a small program with the tool chain now. Here's the simplest C program I can think of:

```
int main(void) {
  return 0;
}
```

Put this in a temporary file `test.c`, and then run the following command to build it:

```
bsc-elf-clang -o test test.c
```

This will produce the file `test` which is an executable which can be run on the Cronus CPU.

# Getting started - Running a program on the LLVM-ISS

The executable file `test` cannot be run on the host, so we must run it under the simulator.

The command for the standalone LLVM-based simulator is `bsc-run`, and we can run our executable as follows:

```
bsc-run test
```

This loads the executable into the simulator and then starts executing, however we don't have any way of starting or stopping the simulator, or asking it what it's doing.

# Getting started - Running a program from the debugger (1)

If we want to be able to debut the program that is running we need to run through the debugger, gdb.

```
bsc-elf-gdb test
```

This loads up the debugger, and gives it an executable file. We now have an interactive prompt where we can provide commands to the debugger:

```
(gdb) <commands go here>
```

# Getting started - Running a program from the debugger (2)

Here's a basic set of commands which we can use to execute our program against the LLVM-based simulator *through the debugger*.

```
(gdb) target remote | embdebug --soname bsc --stdin
(gdb) load
(gdb) break main
(gdb) continue
```