EmbeddedSystemsBuildScripts

Release v0.5

Embedded Systems Department University Duisburg-Essen

Jun 19, 2019

FOR USERS

A collection of Bazel build scripts adding support for avr-gcc and unit testing with the Unity framework.

CHAPTER

ONE

AVRTOOLCHAIN

The AvrToolchain repository is an external dependency that is generated automatically by a `repository_rule` implemented in `@EmbeddedSystemsBuildScripts//AvrToolchain:avr.bzl`. It provides `cc_toolchains` for compiling code with the avr-gcc compiler, for different mcus. Most of the time you will want to enable the `--compile_mode=optimization` flag that already contains gcc flags we found useful for reducing code size.

1.1 Instantiate the AvrToolchain Repository

To depend on the `EmbeddedSystemsBuildScripts` add this to your `WORKSPACE` file:

```
load("@bazel_tools//tools/build_defs/repo:http.bzl", "http_archive")

http_archive(
  name = "EmbeddedSystemsBuildScripts",
  strip-prefix = "EmbeddedSystemsBuildScripts-{version}",
  urls = ["https://github.com/es-ude/EmbeddedSystemsBuildScripts/archive/{version}.
↪tar.gz"]
)
```

replace `{version}` with the actual version you want to use. Or use:

```
http_archive(
  name = "EmbeddedSystemsBuildScripts",
  strip-prefix = "EmbeddedSystemsBuildScripts-master",
  urls = ["https://github.com/es-ude/EmbeddedSystemsBuildScripts/archive/master.tar.gz
↪"]
)
```

to depend on the current master branch. Now you can call the repository rule, that will create the necessary avr toolchains and platforms. Add:

```
load("@EmbeddedSystemsBuildScripts//AvrToolchain:avr.bzl", "avr_toolchain")

avr_toolchain()
```

to the `WORKSPACE` file. The `http_archive` rule has to be called before loading the `create_avr()` function.

1.2 On Platforms and Constraints

Our code has to be deployable on a range of 8-bit AVR platforms as well as the host platforms (this is where your bazel instance runs). Bazel's platforms and constraints mechanics allow to make build decisions depend on different constraints. The user can then specify a set of specific constraints to apply to the current build process with the help of the `platform` rule.

Constraints are basically just typed enumerations and platforms specify a set of constraints. The type of a `constraint_value` is called `constraint_setting`. For every `platform` at most one `constraint_value` for each `constraint_setting` may be specified (ie. your platform may not have arm and x64_86 as cpu architecture).

The scripts provided by us already take different constraints into account. This allows us to write scripts that will produce correct results without knowing the exact platform you want to build for.

We already ship some platform definitions for platforms that we use internally. You can see a list of these definitions by running:

```
$ bazel query `(kind:platform, @AvrToolchain//platforms:*)`
```

To compile for one of these platforms use e.g.:

```
$ bazel build //:myTarget --platforms @AvrToolchain//platforms:Motherboard
```

1.2.1 How to define your own platforms

To define your own avr based platform you will need to specify at least the mcu. Run:

```
bazel query 'kind(constraint_value, @AvrToolchain//platforms/mcu:*)'
```

to retrieve a list of available mcus. Additionally there is the `@AvrToolchain//platforms:avr_common` platform that serves as a parent for all other avr based platforms. E.g. a new platform definition could look like this:

```
platform(
  name = "MyPlatform",
  constraint_values = [
    "@AvrToolchain//platforms/mcu:atmega16",
    "@AvrToolchain//platforms/cpu_frequency:8mhz",
  ],
  parents = ["@AvrToolchain//platforms:avr_common"],
)
```

To see a list of available constraint settings run:

```
$ bazel query 'kind(constraint_setting, @AvrToolchain//platforms/...)'
```

and to see a list of available values for the setting `<my_setting>` you can run:

```
$ bazel query 'attr(constraint_setting, <my_setting>, @AvrToolchain//...)'
```