MutekH tutorial for SoCLib platform

This guide explain how to run MutekH on a customizable SoCLib hardware simulator. This is allow easy experimentation with advanced multi-processor programming. You should read the QuickStartSoclib guide instead if you do not plan to model your own hardware platform and are just interested in embedded software development.

You are **highly encouraged** to first follow the MutekH as Unix process quick start guide which introduces more basic concepts.

MutekH for SoCLib can be compiled for Mips, Arm, PowerPC processors. Other processors are available with different platforms.

In order to begin easy, we'll use only the Arm-v6k ISS model available in SoCLib in this tutorial. Changing the Arm to a Mips or a PowerPC is left as an exercise to the reader.

The SoCLib platform

The MutekH kernel source code is fully configurable and can be tweaked to adapt hardware platform and application needs. Configuration is handled by a dedicated tool which check dependencies and other relationships between the large set of available configuration tokens.

The example below explains how to setup a SoCLib hardware simulator with 4 RISC processors.

Getting started

Of course, you need a working SoCLib install. SoCLib installation is explained here: ?soclib:InstallationNotes

Moreover, you'll need the MutekH source tree and its prerequisites. See InstallationNotes

The MutekH part

Getting the sources

Even if it is available in newer revisions, this tutorial has been tested and is expected to work well at revision 1489, please try with this one if you have troubles with the last revision.

```
svn co -r 1489 https://www.mutekh.org/svn/mutekh/trunk/mutekh
```

Getting the cross-compilers

You can rely on the `tools/crossgen.mk` script which comes along with MutekH to build some GNU toolchains or download a precompiled toolchain. See BuildingExamples page.

Writing the example source code

Note: This example is available directly from examples/hello directory in source tree.

- Writing the source code in `hello.c`

```
#include <pthread.h>

pthread_mutex_t m;
pthread_t a, b;

void *f(void *param)
{
  while (1)
    {
      pthread_mutex_lock(&m);
      printf("(%i) %s", cpu_id(), param);
      pthread_mutex_unlock(&m);
      pthread_yield();
    }
}
int main()
{
  pthread_mutex_init(&m, NULL);
  pthread_create(&a, NULL, f, "Hello ");
  pthread_create(&b, NULL, f, "World\n");
}
```

- Writing the `Makefile`

```
objs = hello.o
```

Writing the MutekH configuration

The MutekH configuration for the hello application is in the examples/hello/config file.

This file only holds information about the application (here a simple pthread application) and relies upon files in the examples/common directory for the platform definitions.

Have a look to the BuildSystem page for more information about configuration system and configuration file format.

The ?MutekH API reference manual describes all available configuration tokens.

Platform description

The MutekH software uses hardware enumeration to get details about available hardware in the platform, so the `CONFIG_ARCH_DEVICE_TREE` token is defined in the examples/common/platforms-soclib.conf configuration file. It will let the kernel get the platform layout description from a FlattenedDeviceTree which will be built into the kernel.

The build system also compiles the correct FlattenedDeviceTree from the platform name, see examples/common/Makefile.

The used FlattenedDeviceTree source file are in examples/common/: pf_soclib_tutorial_arm.dts.

Configuring and compiling the application along with MutekH

Change to the MutekH source directory. We are going to build a MutekH Kernel with our hello application inside. We need to pass the following variables to the build system:

`CONF`
: MutekH configuration file name

`BUILD`
: MutekH build option list (target architecture, cpu type, ?)

```
.../mutekh/ $ make CONF=examples/hello/config BUILD=soclib-arm:pf-tutorial

?

   COPY      ldscript
   LD out    hello-soclib-arm.out
BUILD DIR    .../mutekh/obj-hello-soclib-arm
KERNEL       hello-soclib-arm.out
.../mutekh/ $
```

Compiling the the simulator

Now we can compile the hardware simulator which will simulate the platform we are running the kernel on. Go to the platform directory, in SoCLib source tree, and run make:

```
? $ cd soclib/soclib/platform/topcells/caba-vgmn-mutekh_soclib_tutorial
caba-vgmn-mutekh_soclib_tutorial $ make
mkisofs -f -R -V volname -o block0.iso iso_contents
Total translation table size: 0
Total rockridge attributes bytes: 2901
Total directory bytes: 16384
Path table size(bytes): 126
Max brk space used 0
195 extents written (0 MB)
soclib-cc -P -p platform_desc  -o system.x
[-----------------------------------------=-----------------------------------------------------=] 0
caba-vgmn-mutekh_soclib_tutorial $
```

Now you have a simulator binary named `system.x`:

```
caba-vgmn-mutekh_soclib_tutorial $ ls
Makefile  README  arch.fig  block0.iso  default_caba:-0x7c4e9665_38f32183__top.o  iso_contents
caba-vgmn-mutekh_soclib_tutorial $
```

Execution

Simply run the simulator passing it the MutekH binary kenrel file:

```
$ ./system.x path/to/mutekh/hello-soclib-arm.out
```

You may want to refer to other articles and documents available from the main page to go further with MutekH.

The ?SoCLib home page provides a livecd image with more advanced examples ready to compile and run. These examples are using older MutekH revisions though.

Other more advanced topics and guides are available from the Main page.