Report

Laboratory 2
07.05.2018

Maciej Korzeniewski

GPIO

General-purpose input/output (GPIO) is a generic pin on an integrated circuit or computer board whose behavior - including whether it is an input or output pin - is controllable by the user at run time.

Controlling GPIO pins

GPIO pins can be controlled through sysfs interface under the path /sys/class/gpio/.

We can reserve pin using:

$ echo NN > /sys/class/gpio/export

where NN is number of pin to reserve.
In case of successful operation the folder /sys/class/gpio/gpioNN is created.

We can define direction wth:
$ echo xx > /sys/class/gpio/gpioNN/direction
where xx = [in | out | low | high]

or select signal edge with:
$ echo [none | rising| falling| both] > /sys/class/gpio/gpioNN/edge
Those signal edge values are important when using *poll* function.

And finally we can change value of the selected pin:
$ echo [0 | 1] > /sys/class/gpio/gpioNN/value

Contact bounce effect

Whenever we click a button it is in very unstable state for the first 100-200ms. It generates multiple ‘on’ and ‘offs’ through this short amount of time. We need software solution to prevent it.

The simplest solution to prevent contact bounce effect is to wait about 150ms after button has been clicked and only after that delay start to read the value from pins.

Debugging with GDB

The purpose of a debugger such as GDB is to allow you to see what is going on ''inside'' another program while it executes-or what another program was doing at the moment it crashed.

In order to debug our own application, that is executing on Raspberry Pi, from host machine we need to:

1. Install *gdbserver* package in Buildroot
2. Check *Build cross gdb for the host* in Toolchain
3. Check *Build packages with debugging symbol* in Build Options
4. Compile our application with *-g3* flag, which includes extra informations such as macro definitions.
5. Launch *gdbserver* on Raspberry Pi with *gdbserver host:7123 application*
6. Run gdb on host machine for specific system architecture using the very same binary file:
   [...]/buildroot-{ver}/output/host/usr/bin/arm-none-linux-gnueabi-gdb [...]/application

   target remote [raspberry-pi-ip]:7123.

At the very beginning of the execution debugger halts and waits for user interaction.
We can:

- set breakpoint using: *break N*, where N stands for line number,
- continue execution using: *continue* (or c for shortcut)
- step to next line using: *step*
- show code using: *list*

Buildroot package

Since our application is written in compiled language we need to tell buildroot how to build our application.

We need to create package/<manufacturer>/ and place our packages in that directory. Create an overall <manufacturer>.mk file that includes the .mk files of all your packages. Create an overall Config.in file that sources the Config.in files of all your packages. Include this Config.in file from Buildroot’s package/Config.in file.

We can build our application with:

make application rebuild

Solution files:

1. .config
2. build.sh - builds application
3. package - folder containing package files
4. Makefile
5. mytimer.c - application sourcecode

References:

1. https://buildroot.org/downloads/manual/manual.html#writing-rules-config-in
2. https://linux.die.net/man/1/gdb