Linux GPIO Framework

General Purpose Input/Output Library (gpiolib)

Bill Gatliff
`bgat@billgatliff.com`

Freelance Embedded Systems Developer

Overview

Roadmap:

- What is “gpio”?
- The `sysfs` interface
- Asserting GPIO outputs, reading inputs
- Examples

What is “GPIO”?

General-Purpose I/O:

- Software-controlled digital signal
- A.k.a. “GPIO”

Manifestations:

- Dedicated SOC registers
- Standalone chips

What is “GPIO”?

Types:

- Inputs, outputs
- Input/output
- Push-pull, open drain (a.k.a. “wire-OR”)

Some can assert interrupts:

- Functionality is very platform-specific

What is “GPIO”?

SOC registers:

- 8, 16, or 32 bits common
- MMIO, typically
- Atomic access

Standalone “expansion” chips:

- I2C, SPI, USB, ...
- Require serial communications
- Implementations usually not spinlock-compatible

Why a common framework?

Platform-specific, but:

- Externally-visible API can be standardized
- Implementation can offload some details

Benefits:

- Stability
- Portability

What does it cost?

Virtually nothing!

- Inline functions
- No redundant code
- (Sysfs interface requires a system call)

As good, probably better than hand coding!

How does it work?

Two interfaces:

- Sysfs, for user applications
- Kernel data structures, APIs

Transition phase:

- Macros, etc. that mimic the API
- (A permanent solution for some implementations)

How does it work?

Sysfs API:

- Application request an “export”
- Read, write attributes to manipulate pin state

Limitations:

- Minimal concurrency protection
- (Not all hardware requires it)

How does it work?

Kernel API:

- Common API to kernel users
- `gpio_direction_output()`, `gpio_get_value()`, etc.
- (We'll come back to this later)

```
#include <linux/gpio.h>
```

Caveat

Currently not “pluggable”:

- Platform integrator defines pin-to-integer mapping
- Drivers must Just Know the pin number

Not a significant shortcoming:

- GPIO is usually platform-specific anyway

Sysfs API

```
$ **ls -CF /sys/class/gpio**
export      gpio35/  gpiochip160/ gpiochip32/
gpiochip64/ gpio100/ gpiochip128/ gpiochip96/
unexport
```

Sysfs API

How to use it:

- Request an “export”
- Set pin direction
- Assert, request pin value
- When done, “unexport” the pin

Sysfs “attributes”:

- Interfaces to similarly-named kernel objects

Sysfs API

Request an “export”:

- Ask the API to grant access to the pin
- Attributes appear in pin-labeled subdirectory

```
$ echo 35 > export
$ ls -CF gpio35
direction  power/  subsystem@  uevent  value
```

Sysfs API

Assess pin configuration:

- “Is it an input, or an output?”

```
$ cat gpio35/direction
in
```

Sysfs API

Configure as an output:

- ... and specify the value (important!)

```
$ echo low > gpio35/direction
$ echo high > gpio35/direction
```

Sysfs API

Configure as an input:

- ... and read back the value

```
$ echo in > gpio35/direction
$ cat gpio35/value
0
```

Sysfs API

“Unexport” the pin:

- Frees it for use by kernel, others

```
$ echo 35 > unexport
```

Recap

The GPIO Sysfs interface:

- Read, write attributes to control pins
- Must request an “export” first

Advantages and disadvantages:

- Common API
- Possible concurrency issues

Linux GPIO Framework

General Purpose Input/Output Library (gpiolib)

Bill Gatliff
bgat@billgatliff.com

Freelance Embedded Systems Developer