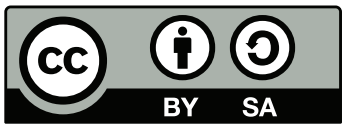
GPIO, SPI and I^{2}C from Userspace, the True Linux Way

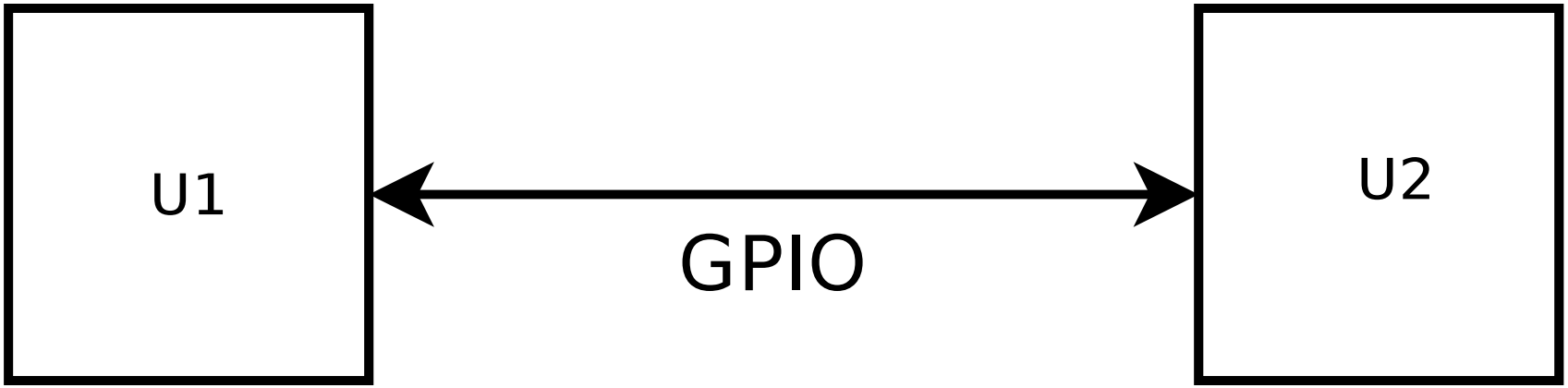
Baruch Siach
baruch@tkos.co.il

Tk Open Systems

June 27, 2011



What is General Purpose Input/Output (GPIO)?

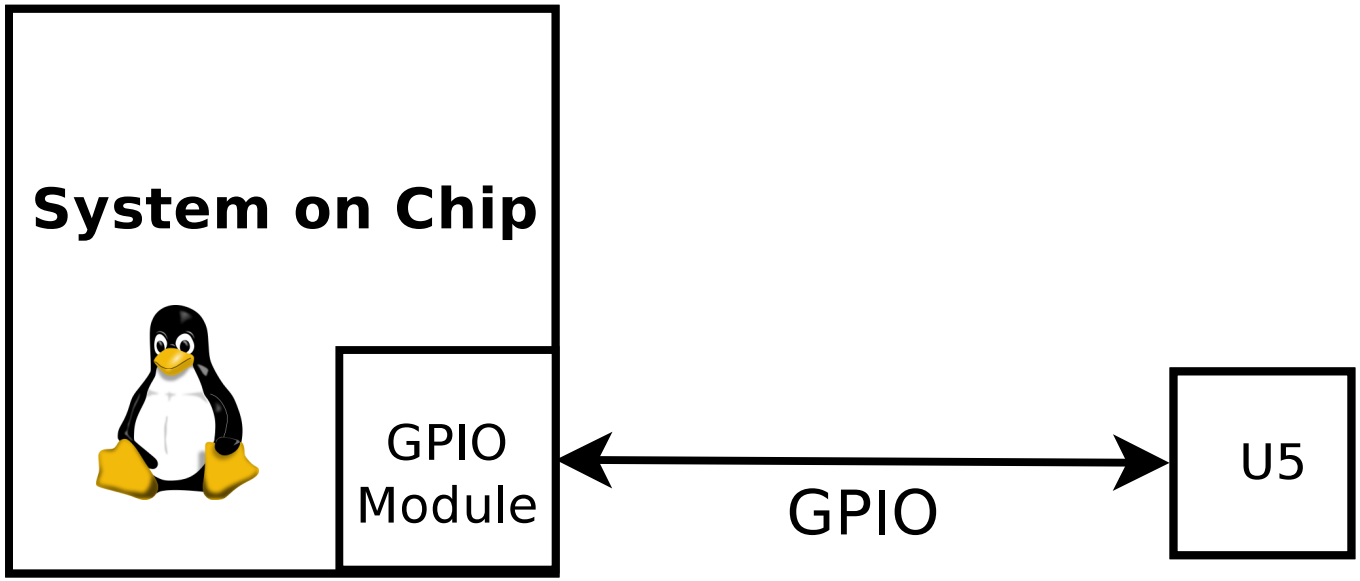


- Pin connection two electronic components (chips)
- Voltage is held at one of two level to indicate 1 or 0 logic
- Controlled by one chip, sensed by the other
- Usually grouped in banks

GPIO Chip Capabilities

- Per GPIO pin configuration
- Configure pin direction mode to input or output
- Input mode:
  - Sense the logic level
  - Interrupt source (asynchronous notification)
- Output mode:
  - Set voltage logic level to 0 or 1

Linux GPIO Userspace Interface Overview



- Documented in `Documentation/gpio.txt`
- Part of the gpiolib framework, originally by David Brownell (R.I.P.)
- Since kernel version 2.6.27
- GPIO ID numbers are set by the platform code (under `arch/`)
- GPIO drivers are under `drivers/gpio/` (e.g. `pl061.c`)

Linux GPIO Userspace Interface Details

GPIO control interface is via sysfs under `/sys/class/gpio`, and includes the following control files:

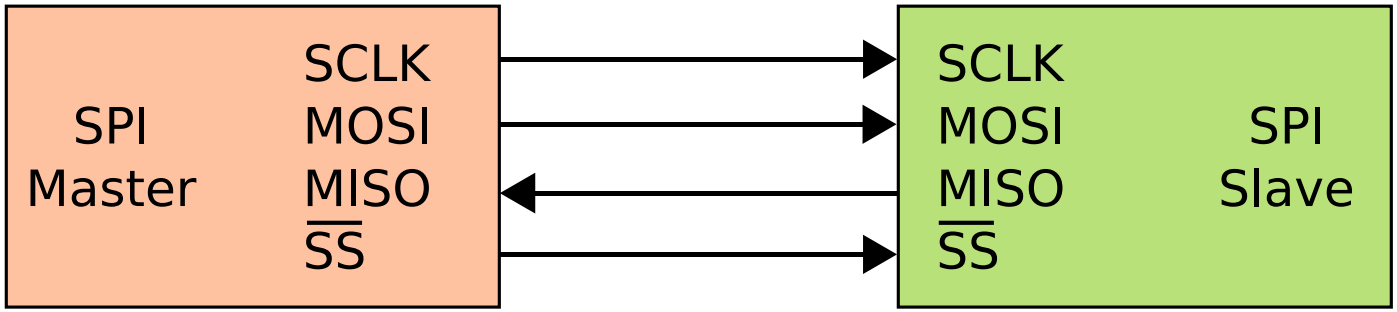
`export` Make a specific GPIO pin available for userspace control. Write the pin number N (e.g. "55", ASCII); the gpioN directory should appear.

`gpioN/direction` Write "in" or "out" to set pin direction. Write "high" or "low" to set direction to output, with initial value, atomically.

`gpioN/value` Read the current pin status in input. For output, write "0" or "1" to set the pin status. To get change notification (interrupt) lseek() to end of file, and either `poll()` for `POLLPRI` and `POLLERR`, or select() with the file descriptor in `exceptfds`.

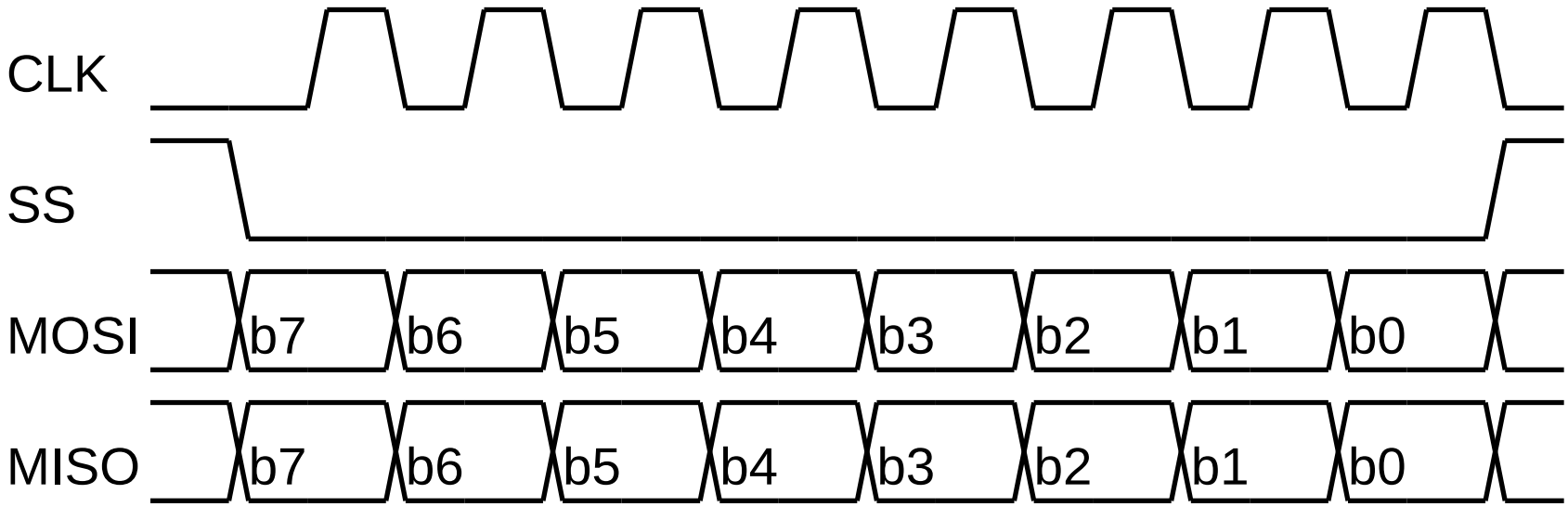
`gpioN/edge` Write "none", "rising", "falling", or "both" to select the signal that makes `poll()` return.

What is Serial Peripheral Interface (SPI)?



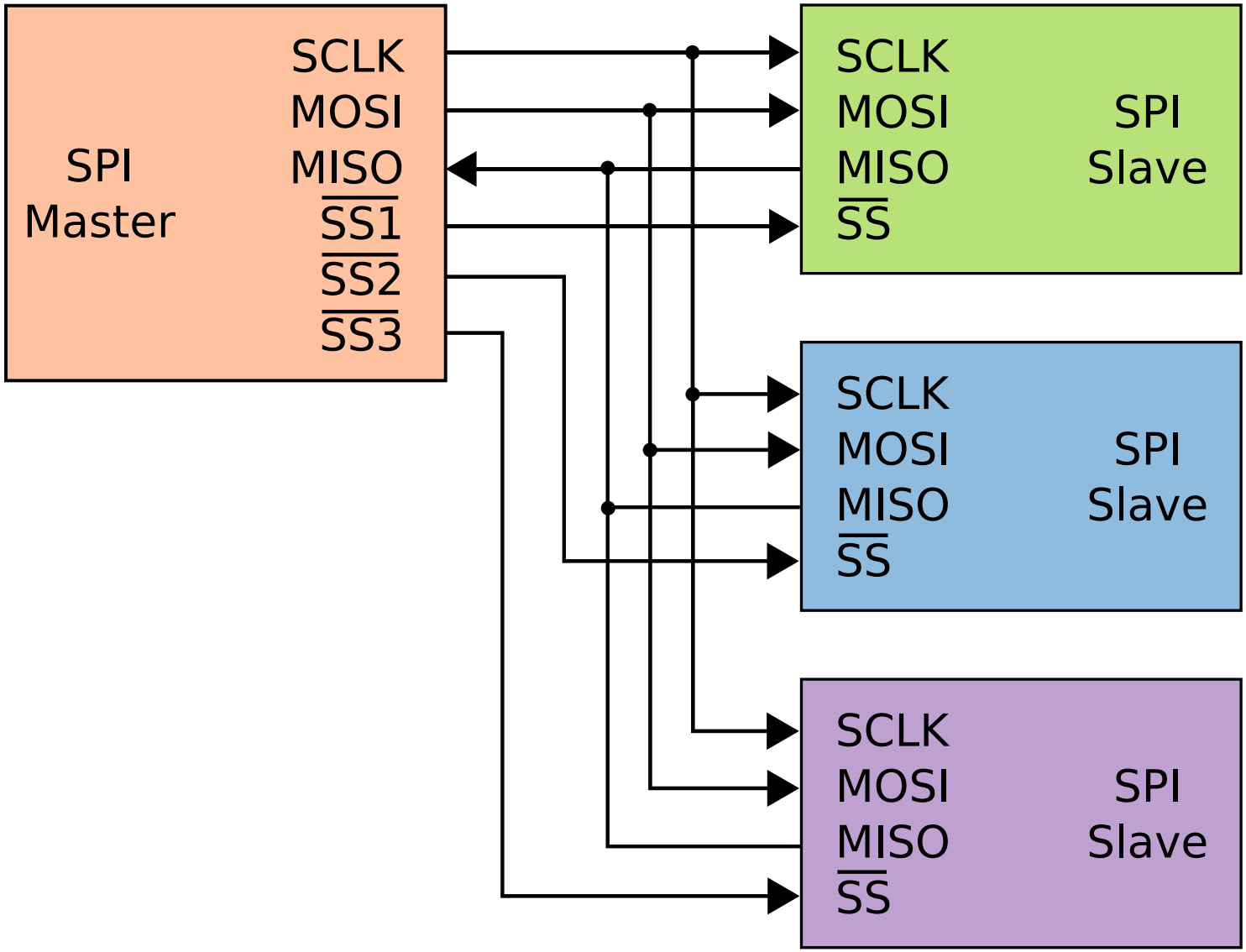
- Synchronous serial digital data link by Motorola
- SPI master controls
  - CLK: synchronization clock
  - SS or CS: slave select or chip-select; when active the slave is allows to talk
  - MOSI: master out, slave in; carries data from master to slave
- SPI slave controls
  - MISO: master in, slave out; carries data from slave to master

SPI Timing Diagram



- SS signals the transaction boundaries
- Data is valid at the rising/first clock edge (SPI mode 0)
- Bytes are sent MSB first

Multiple SPI Slaves Diagram



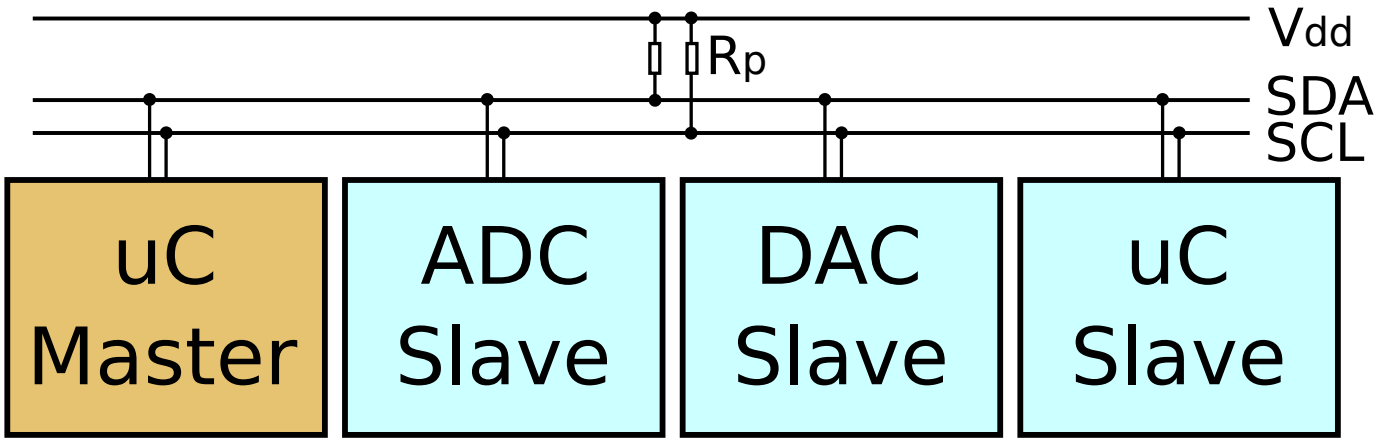
Userspace Interface for Basic SPI Transfer

- Documented in `Documentation/spi/spidev`
- Uses the SPI kernel framework, also by David Brownell
- Since kernel version 2.6.27
- SPI master drivers are under `drivers/spi/` (e.g. `spi_gpio.c`)
- Kernel platform code registers the "`spidev`" platform device
- Creates character device nodes at `/dev/spidevB.C` where:
  - B is the SPI bus (master) number
  - C is the chip-select number of specific SPI slave
- `read()` for read only SPI transaction, with a single chip-select activation
- `write()` for write only SPI transaction, with a single chip-select activation

Userspace Interace for Full Duplex SPI Transfer

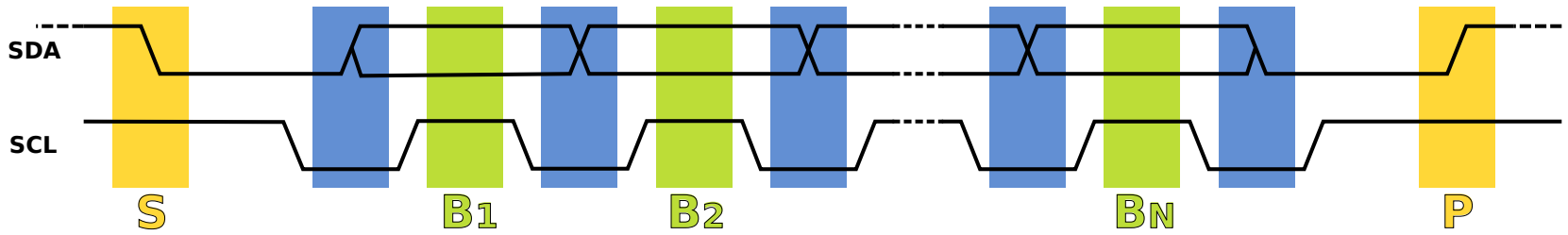
```
1   #include <linux/types.h>
2   #include <linux/spi/spidev.h>
3
4   #define RX_LEN          32
5
6   struct spi_ioc_transfer xfer[2];
7   unsigned char           buf[RX_LEN];
8
9   memset(xfer, 0, sizeof xfer);
10  memset(buf, 0, sizeof buf);
11
12  buf[0] = 0xaa;
13  xfer[0].tx_buf = (unsigned long) buf;
14  xfer[0].len = 1;
15
16  xfer[1].rx_buf = (unsigned long) buf;
17  xfer[1].len = RX_LEN;
18
19  ioctl(fd, SPI_IOC_MESSAGE(2), xfer);
```

What is Inter-Integrated Circuit (I^2C) and System Message Bus (SMBus)?



- Two wires are controlled by the master and slaves according to the protocol:
  - SCL: Serial Clock
  - SDA: Serial Data
- Each slave has a 7 bit address
- The 7 MSB of the first transmitted byte are slave address
- The LSB of this byte indicates read (1), or write (0)
- SMBus is a subset of I^2C, with a stricter protocol definition

I²C Timing Diagram



- Transfer starts with START bit (S), SDA pulled low, while SCL stays high
- Data bit transfered at SCL rise
- Transfer end with a STOP bit (P), SDA rises, while SCL stays high

Userspace Interface for I²C / SMBus

- Documented in `Documentation/i2c/dev-interface`
- Since the beginning of kernel 2.4 era
- I²C master drivers are under `drives/i2c/busses/` (e.g. `i2c-designware.c`)
- Load the `i2c-dev` kernel module to create device nodes
- Each I²C master gets a character device node at `/dev/i2c-N`, where N is the master ID number
- `read()` and `write()` can do single direction transfer, but their use is rare
- `ioctl()` does combined transfers (read and write in one transfer)
- For some convenient `ioctl()` wrappers include the `i2c-dev.h` file from i2c-tools

Userspace Interface for I²C / SMBus, Example

```c
/* NOTE: header from i2c-tools, not the kernel */
#include "i2c-dev.h"

int val;
uint8_t val8    = 0xaa;
uint16_t val16  = 5555;

/* set slave address to 0x44 */
ioctl(fd, I2C_SLAVE, 0x44);

/* read byte (8 bit) value from register 0x0a */
val = i2c_smbus_read_byte_data(fd, 0x0a);

/* read word (16 bit) value from register 0x1a */
val = i2c_smbus_read_word_data(fd, 0x1a);

/* write byte (8 bit) value in register 0x2a */
val = i2c_smbus_write_byte_data(fd, 0x0a, val8);

/* write word (16 bit) value in register 0x3a */
val = i2c_smbus_write_word_data(fd, 0x1a, val16);
```