```
//
// How to access GPIO registers from C-code on the Raspberry-Pi
// Example program
// 15-January-2012
// Dom and Gert
// Revised: 15-Feb-2013

// Access from ARM Running Linux

#define BCM2708_PERI_BASE 0x20000000

#define GPIO_BASE (BCM2708_PERI_BASE + 0x200000) / GPIO controller /

#include

#include

#include

#include

#include

#define PAGE_SIZE (4*1024)

#define BLOCK_SIZE (4*1024)

int mem_fd;
void *gpio_map;

// I/O access
volatile unsigned *gpio;

// GPIO setup macros. Always use INP_GPIO(x) before using OUT_GPIO(x) or SET_GPIO_ALT(x,y)

#define INP_GPIO(g) (gpio+((g)/10)) &= ~(7<<(((g)%10)3))

#define OUT_GPIO(g) (gpio+((g)/10)) |= (1<<(((g)%10)3))

#define SET_GPIO_ALT(g,a) (gpio+(((g)/10))) |= (((a)<=3?(a)+4:(a)==4?3:2)<<(((g)%10)3))

#define GPIO_SET *(gpio+7) // sets bits which are 1 ignores bits which are 0

#define GPIO_CLR *(gpio+10) // clears bits which are 1 ignores bits which are 0

#define GET_GPIO(g) (*(gpio+13)&(1<<g)) // 0 if LOW, (1<<g) if HIGH

#define GPIO_PULL *(gpio+37) // Pull up/pull down
```

```
#define GPIO_PULLCLK0 *(gpio+38) // Pull up/pull down clock

void setup_io();

void printButton(int g)
{
if (GET_GPIO(g)) // !=0 <-> bit is 1 <- port is HIGH=3.3V
printf("Button pressed!\n");
else // port is LOW=0V
printf("Button released!\n");
}

int main(int argc, char **argv)
{
int g,rep;

// Set up gpi pointer for direct register access
setup_io();

// Switch GPIO 7..11 to output mode

/**\
```

- You are about to change the GPIO settings of your computer. *
- Mess this up and it will stop working! *
- It might be a good idea to 'sync' before running this program *

- so at least you still have your code changes written to the SD-card!
  ***/

```
// Set GPIO pins 7-11 to output
for (g=7; g<=11; g++)
{
INP_GPIO(g); // must use INP_GPIO before we can use OUT_GPIO
OUT_GPIO(g);
}

for (rep=0; rep<10; rep++)
{
for (g=7; g<=11; g++)
{
GPIO_SET = 1<<g;
sleep(1);
}
for (g=7; g<=11; g++)
```

```
    {
    GPIO_CLR = 1<<g;
    sleep(1);
    }
    }

    return 0;

} // main

//
// Set up a memory regions to access GPIO
//
void setup_io()
{
/ open /dev/mem /
if ((mem_fd = open("/dev/mem", O_RDWR|O_SYNC) ) < 0) {
printf("can't open /dev/mem \n");
exit(-1);
}

/ mmap GPIO /
gpio_map = mmap(
NULL, //Any adddress in our space will do
BLOCK_SIZE, //Map length
PROT_READ|PROT_WRITE,// Enable reading & writting to mapped memory
MAP_SHARED, //Shared with other processes
mem_fd, //File to map
GPIO_BASE //Offset to GPIO peripheral
);

close(mem_fd); //No need to keep mem_fd open after mmap

if (gpio_map == MAP_FAILED) {
printf("mmap error %d\n", (int)gpio_map);//errno also set!
exit(-1);
}

// Always use volatile pointer!
gpio = (volatile unsigned *)gpio_map;

} // setup_io
```