CS 310 Raspberry Pi GPIO Lab

Spring 2021

February 2, 2021

1 Intro

In this lab, you're going to blink an LED using code that runs on the bare metal (no OS) on your Pi. The green ACT LED on your Raspberry Pi is controlled by a general-purpose input/output pin (GPIO) on the Pi's CPU. GPIO pins can be controlled by software to have either a high voltage (usually 3.3V) or low voltage (0V) by writing a 1 or a 0 to a special register. The Raspberry Pi has many GPIOs connected to the 40-pin rectangular header at the top of the board[1].

1.1 Overview of this Project

You will use your OS template repository as a base for this project. You will add an extra assembly file called `src/led.s` which will contain a few assembly-language functions to set up and blink the green ACT LED on you board.

Function Name	Description
`led_init`	Configures the GPIO42 pin to act as an output.
`led_on`	Turns the LED on by writing a 1 to GPIO pin 42.
`led_off`	Turns the ACT LED off by writint a 0 to GPIO pin 42.

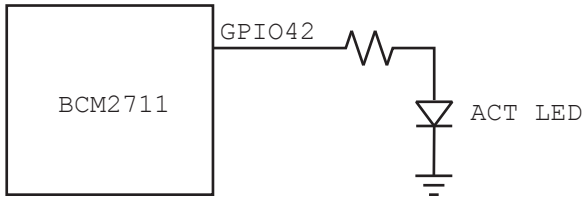
1.2 Controlling the ACT LED

1.2.1 Initialization

Each GPIO pin on the BCM2711 has several possible functions. Depending on the pin, it can be a GPIO, a SPI (for connecting to low-speed peripherals), a UART (for terminal I/O) a PCM (for audio I/O), etc. This is called GPIO multiplexing, and it allows us to have a lot of flexibility in what devices we connect to various pins on the processor. Before we can control the state of the GPIO in software, we need to configure GPIO 42 it to be a GPIO output. Detailed instructions on how to do that are below.

1.2.2 Turning the LED On and Off

The figure below shows how the ACT LED is connected to the CPU on the Pi 4 (different versions of the Pi use other GPIO pins for the ACT led). We can turn the light on by setting GPIO 42 to a logical 1, which will cause the processor to output 3.3V on the GPIO 42 pin, powering on the ACT LED. Four internal registers in the BCM2711 CPU control the state of the GPIO lines: `GPIOSET0` and `GPIOSET1` turn the GPIO pins on and `GPIOCLR0` and `GPIOCLR1` turn the GPIO pins off. Each of these registers is 32 bits wide. Since we have more than 32 GPIO pins on the BCM2711, we need two registers to set (turn on) the GPIO pins and two registers to clear (turn off) the GPIOs. The ACT LED is connected to GPIO 42, so we will need to write to `GPIOSET1` to turn it on and `GPIOCLR1` to turn it off.



[1] See `https://pinout.xyz` for a list.

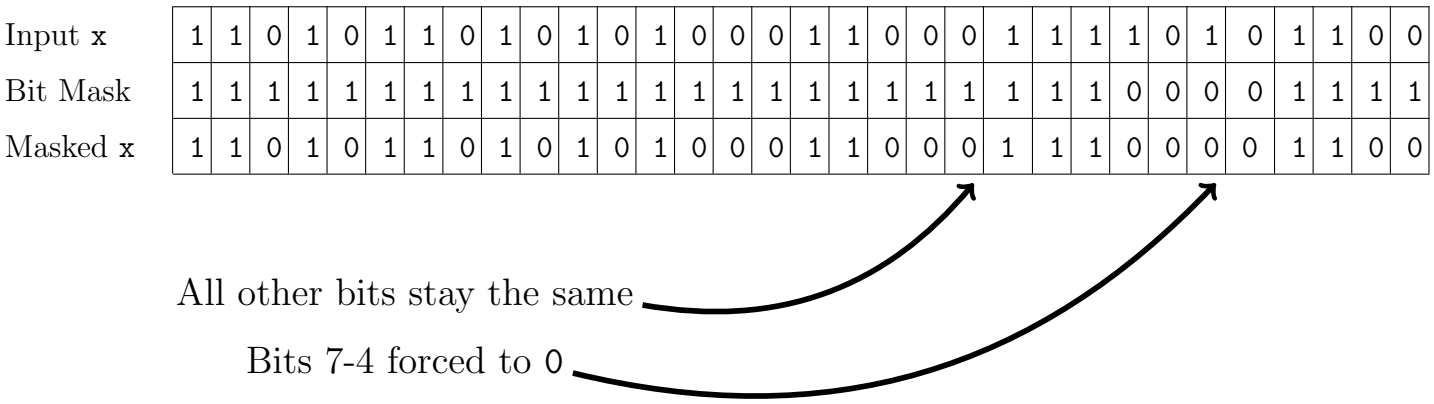
2 Skeleton for your LED Driver

```
1  void led_init() {
2      return;
3  }
4
5  void led_on() {
6
7  }
8
9  void led_off() {
10
11 }
```

The above skeleton code is basically just some function definitions for your driver. You're going to have to implement these functions (guidance below) and call them from `kernel_main`. You'll also need to write a `delay` function which waits for a while between turning the LED on and off. `kernel_main` will call `led_init` once and then run in an infintie loop, alternatively calling `led_on`, then `delay`, then `led_off`, then `delay` forever.

2.1 Initializing the GPIO

Bit Masks A bit mask is a number that you logically AND with another number in order to force some bit positions to zero. For example, let's say we have a 32-bit number in a variable `x`, and we want to force bits 7-4 to zero. We can use the bit mask `0xffffff0f` to force those bits to zero: no matter what the value of `x`, when we logical AND with our bit mask, bits 7-4 will always be zero.



Writing to a Specific Memory Address in C Usually we—the programmers—are not in charge of choosing the addresses where our variables live. Choosing variable addresses is normally a function of the compiler of the OS. But if we want to write a value to a specific address in memory, we can do it by (i) initializing a pointer to the address and (ii) dereferencing the pointer to read or write to its address.

For example, say we want to write one byte to address `0xDEADBEEF`. Since we want to write one byte, we will use the `char` datatype which refers to a single byte. We can initialize a character pointer with the address:

```
char *ptr = 0xDEADBEEF;  // point to specific location in memory
*ptr = 0x33;             // write the value 0x33 to the location in memory pointed to by ptr
```

There are other datatypes in C that can refer to larger variables in memory. The table below has a list of commonly used integer data types in C along with their sizes.

Type Name	Size
char	1 byte
short	2 bytes
int	4 or 8 bytes
long	4 or 8 bytes
uint8_t	1 byte
uint16_t	2 bytes
uint32_t	4 bytes
uint64_t	8 bytes

Setting the GPIO Function Register with a Bit Mask To initialize GPIO 42, we need to set its function using a GPIO Function Select Register (GPFSELx[2]). The GPFSELx registers can be accessed at specially assigned locations in memory:

Register Name	Address
GPFSEL0	0xFE200000
GPFSEL1	0xFE200004
GPFSEL2	0xFE200008
GPFSEL3	0xFE20000C
GPFSEL4	0xFE200010
GPSET0	0xFE20001C
GPSET1	0xFE200020
GPCLR0	0xFE200028
GPCLR1	0xFE20002C

Bits 8, 7, and 6 control the function of GPIO 42. They must be set to 001 to configure GPIO 42 as a general purpose output. But we can't just write the value 0x00000040 (which has the value 001 in bit positions 8-6) because it would zero out the other bits in the register. Instead, use a bitmask to zero out bits 8-6 of the register and then logical OR with 0x00000040.

2.2 Turning the ACT LED On and Off

You can use the same procedure to turn GPIO42 on and off by writing to GPSET1 and GPCLR1 which respectively set and clear the value on GPIO outputs. The GPSET0 register can be used to turn on GPIOs 0-31 by writing 1 to any of the bits in that 32-bit register. For example, if we want to turn on GPIO 3, we need to construct a number that has a 1 in bit position 3: 0x00000004. We can then write that value to the GPSET0 register:

```
uint32_t gpset0 = 0xFE20001C;
*gpset0 = 0x00000004;
```

But this procedure won't exactly work because we need to set and clear the value of GPIO42, which is not controlled by GPSET0—it's controlled by GPSET1 which controls the output values of GPIOs 32-63. We can use GPCLR1 to clear the value of GPIO 42.

[2]See the BCM2711 ARM Peripherals Manual: https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/rpi_DATA_2711_1p0.pdf