***EC-ENG 231 (Spring 2024)***

# ***Beaglebone GPIO Programming:***
# Pulse Width Modulation (PWM)

Fatima Anwar

fanwar@umass.edu

UMass Amherst

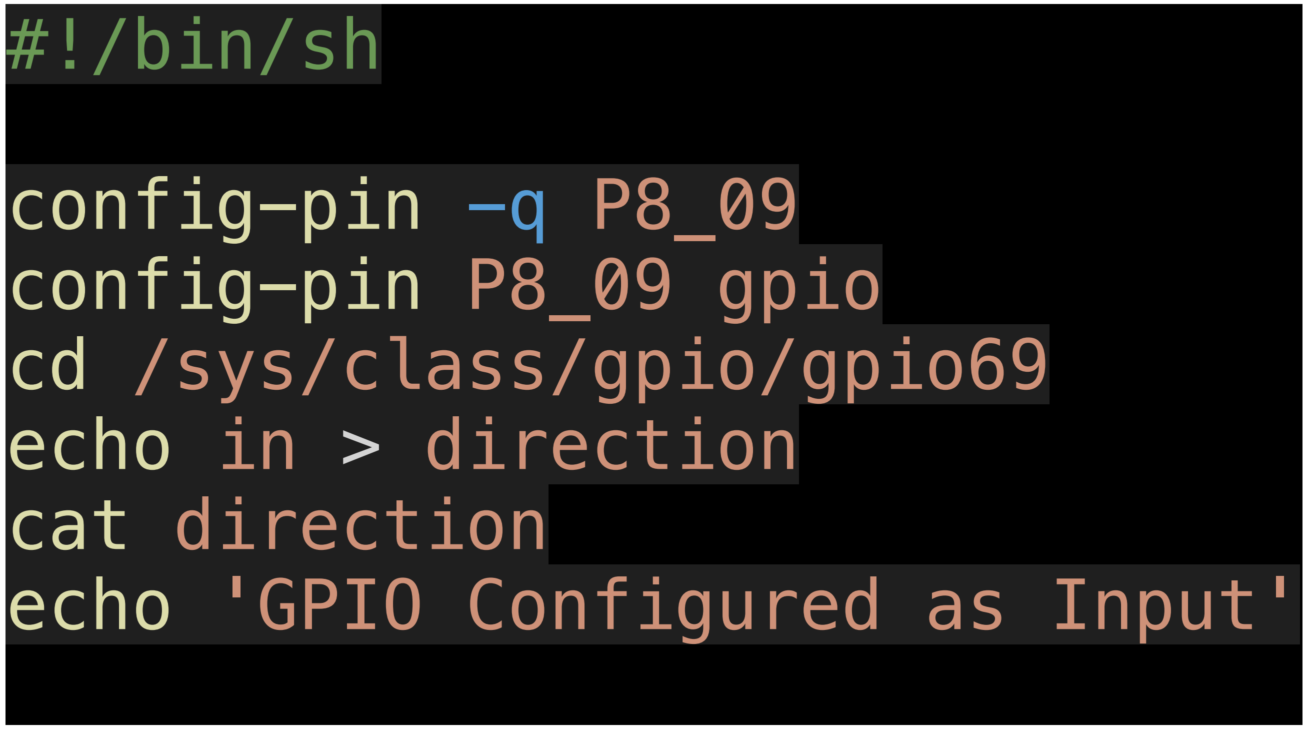
# How to program GPIOs in command line

- Configure P8_9 as input, which corresponds to **gpio69**, (verify from pin header)

| | |
|---|---|
| `cd /sys/class/gpio/` | [ Enter GPIO directory] |
| `ls` | [Check if gpio69 is already exposed] |
| `echo gpio69 > export` | [ Expose P8_9 to the user application] |
| | [Only if gpio is not exported] |
| `cd /sys/class/gpio/gpio69` | [ Enter directory corresponding to P8_9 ] |
| `echo in > direction` | [ Set the gpio as an input pin ] |
| `cat direction` | [Prints the value of direction: “in” corresponds..<br>… to input, and “out” corresponds to output pin] |

# GPIO in Shell script

- Write a shell script that configures a pin to gpio mode and makes it an input pin

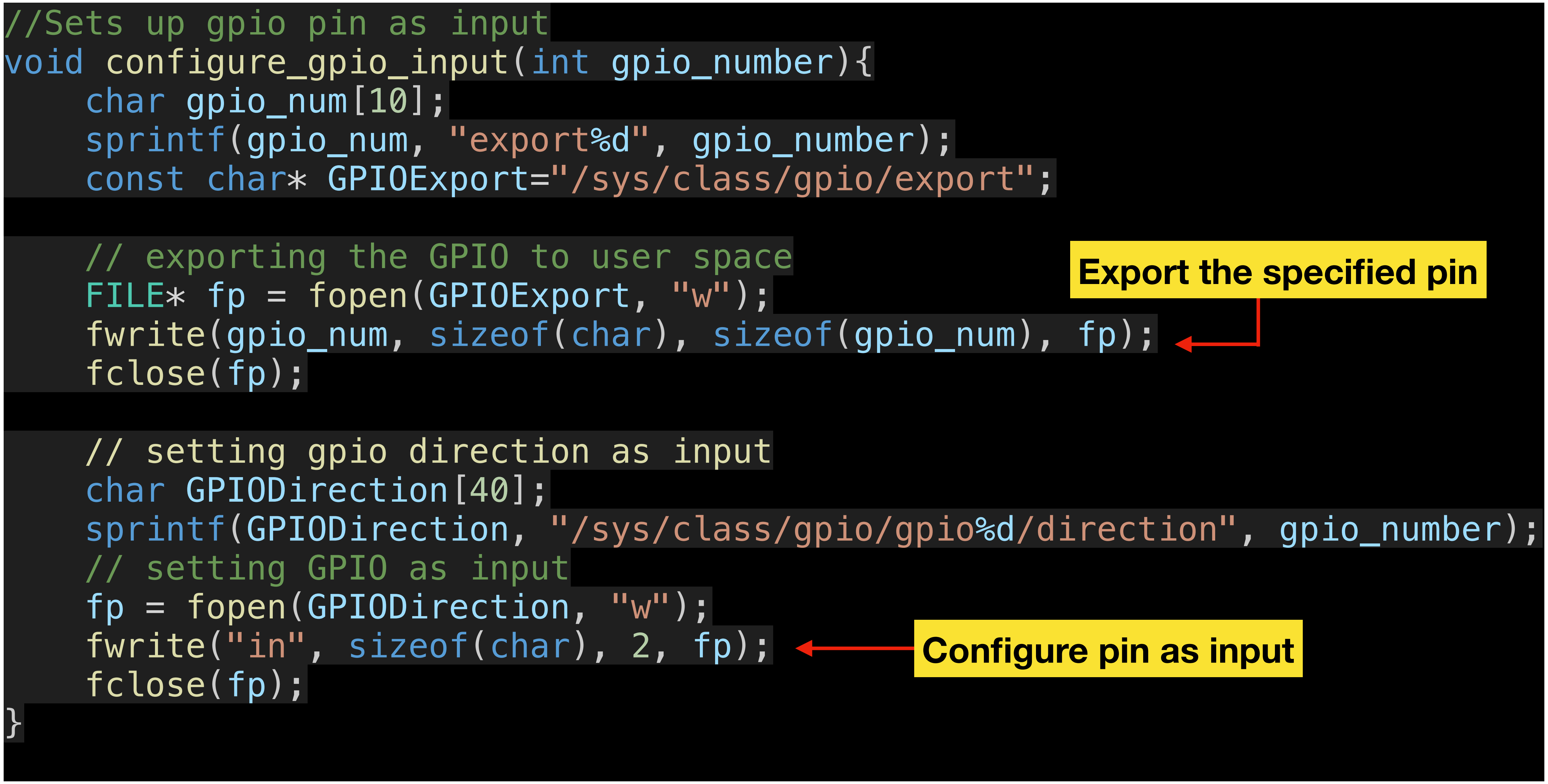


```
#!/bin/sh

config-pin -q P8_09
config-pin P8_09 gpio
cd /sys/class/gpio/gpio69
echo in > direction
cat direction
echo 'GPIO Configured as Input'
```

**NOTE: Don’t forget to check the pin mode. It should be GPIO**

# GPIO Programming in C

- Export the GPIO pin and configure it as an input



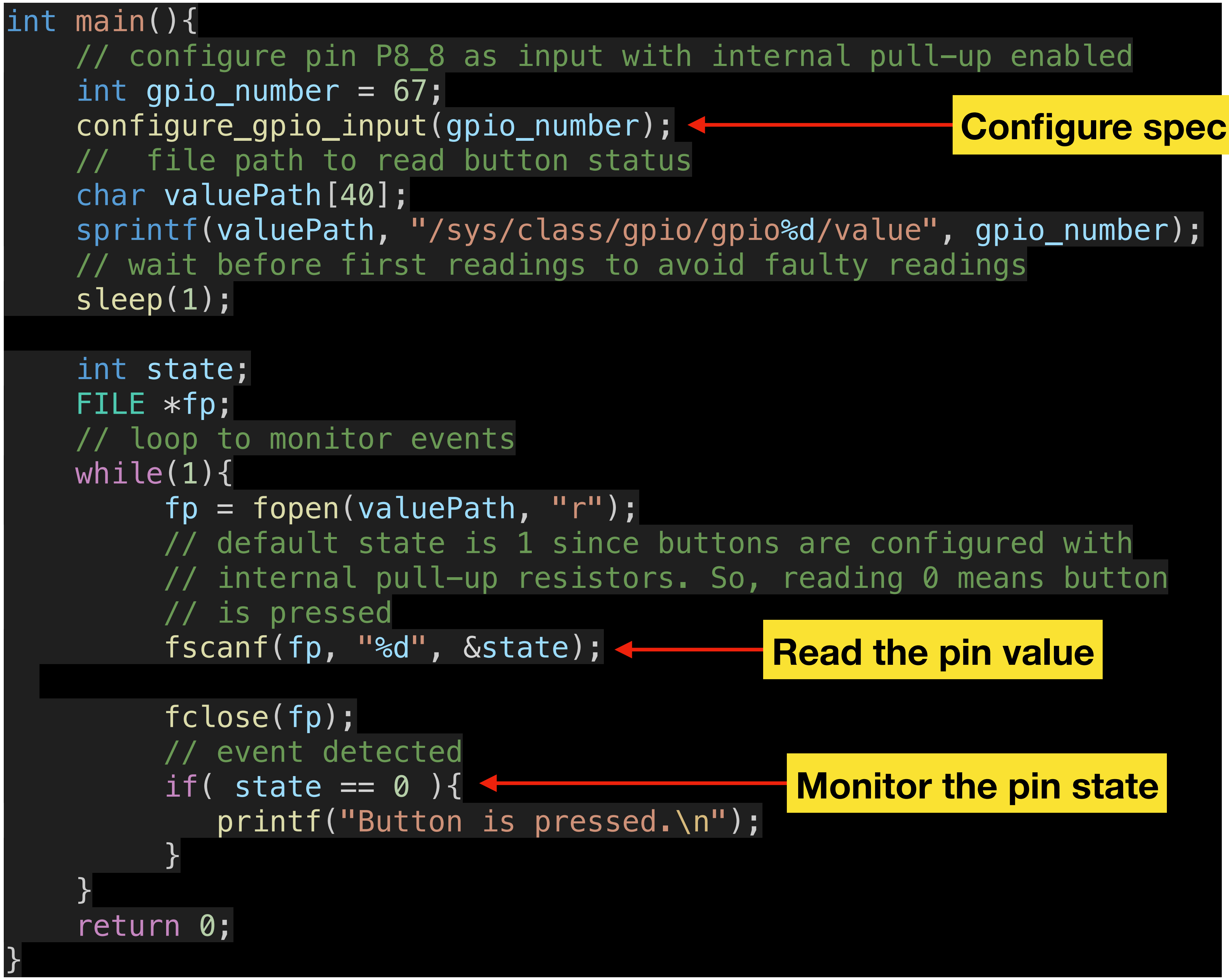
```c
//Sets up gpio pin as input
void configure_gpio_input(int gpio_number){
    char gpio_num[10];
    sprintf(gpio_num, "export%d", gpio_number);
    const char* GPIOExport="/sys/class/gpio/export";

    // exporting the GPIO to user space
    FILE* fp = fopen(GPIOExport, "w");
    fwrite(gpio_num, sizeof(char), sizeof(gpio_num), fp);
    fclose(fp);

    // setting gpio direction as input
    char GPIODirection[40];
    sprintf(GPIODirection, "/sys/class/gpio/gpio%d/direction", gpio_number);
    // setting GPIO as input
    fp = fopen(GPIODirection, "w");
    fwrite("in", sizeof(char), 2, fp);
    fclose(fp);
}
```

**Export the specified pin**

**Configure pin as input**

# GPIO Programming in C

- Read the GPIO Value in an infinite loop
- Check if a button attached to the GPIO is pressed or not

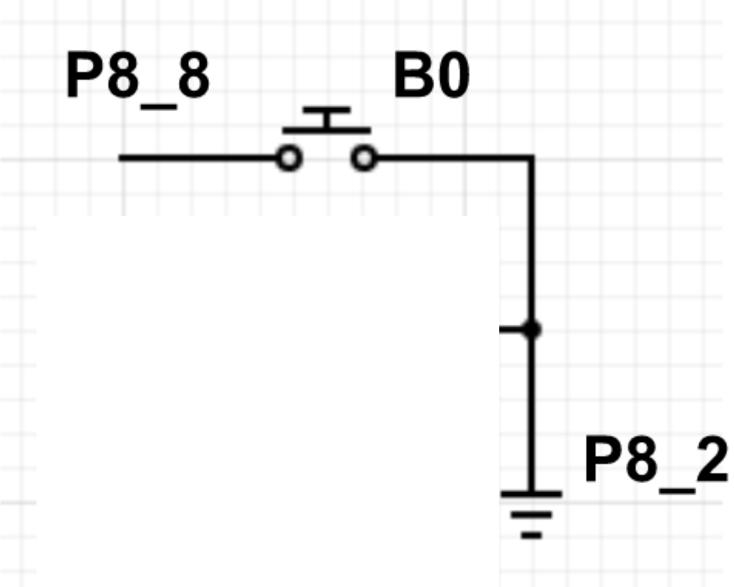


```
int main(){
    // configure pin P8_8 as input with internal pull-up enabled
    int gpio_number = 67;
    configure_gpio_input(gpio_number);
    //  file path to read button status
    char valuePath[40];
    sprintf(valuePath, "/sys/class/gpio/gpio%d/value", gpio_number);
    // wait before first readings to avoid faulty readings
    sleep(1);

    int state;
    FILE *fp;
    // loop to monitor events
    while(1){
         fp = fopen(valuePath, "r");
         // default state is 1 since buttons are configured with
         // internal pull-up resistors. So, reading 0 means button
         // is pressed
         fscanf(fp, "%d", &state);

         fclose(fp);
         // event detected
         if( state == 0 ){
            printf("Button is pressed.\n");
         }
    }
    return 0;
}
```

Configure specified pin as input first

Read the pin value

Monitor the pin state



# LED Programming

- LEDs are represented as files at: /sys/class/leds

```
cd /sys/class/leds
ls -l
```

- Navigate to following folder to reach a particular LED and use some commands:

```
cd /sys/class/leds/beaglebone\:green\:usrx
```
[replace x with 0,1,2,3]

- We can find out the current status of the LED by following command:

```
more trigger
```
[Currently trigger is set up as a “heartbeat”]

- We can turn this pattern off by:

```
echo none > trigger
```
[The LED will stop flashing]

- Once the trigger is off, we can turn on the LED using the brightness setting:

```
echo 1 > brightness
```

- We can turn off the LED using the brightness setting:

```
echo 0 > brightness
```

- We can set it back to the way it was before we started:

```
echo heartbeat > trigger
```

# LED Programming - Todo

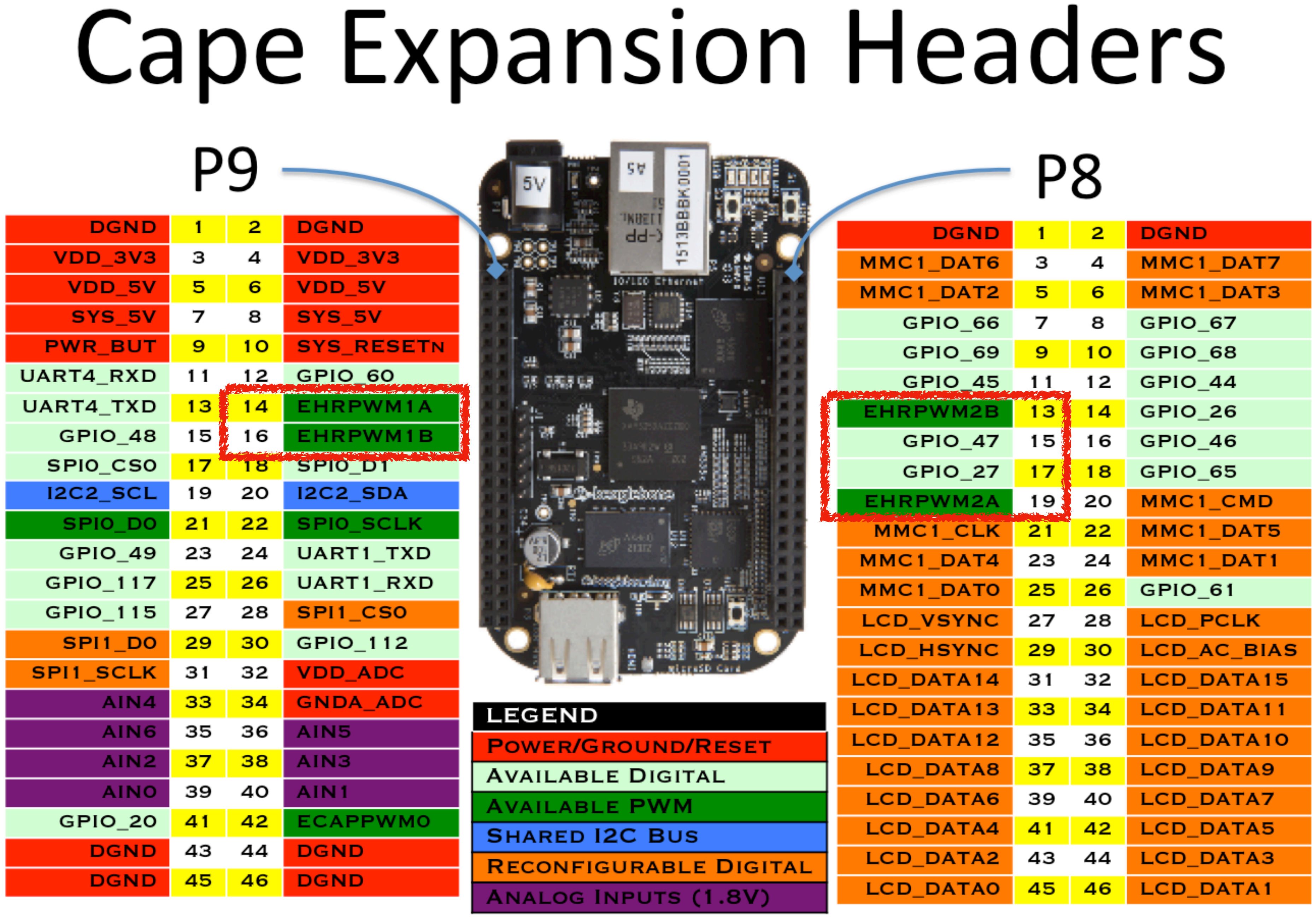
- Write a shellscript that toggles one of the Beaglebone’s LEDs at the rate of 1Hz for a duration of 60 seconds

# Pulse Width Modulation (PWM) Programming

# Pulse Width Modulation

- Example:
  - Consider a 3.3 volt signal, which is turning on and off with a frequency of 50 Hz ( period of 0.02 seconds, or 20 milliseconds)
  - If during that 20 millisecond period, the signal was “High” for 10 milliseconds, and “Low” for 10 milliseconds, the signal would act like a 1.65 volt analog signal
  - The output voltage could be considered the rail voltage (3.3 volts) multiplied by the percentage of time the signal is high (duty cycle).

# Beaglebone Header - default configuration



## Cape Expansion Headers

P9

| | | | |
|---|---|---|---|
| DGND | 1 | 2 | DGND |
| VDD_3V3 | 3 | 4 | VDD_3V3 |
| VDD_5V | 5 | 6 | VDD_5V |
| SYS_5V | 7 | 8 | SYS_5V |
| PWR_BUT | 9 | 10 | SYS_RESETN |
| UART4_RXD | 11 | 12 | GPIO_60 |
| UART4_TXD | 13 | 14 | EHRPWM1A |
| GPIO_48 | 15 | 16 | EHRPWM1B |
| SPI0_CS0 | 17 | 18 | SPI0_D1 |
| I2C2_SCL | 19 | 20 | I2C2_SDA |
| SPI0_D0 | 21 | 22 | SPI0_SCLK |
| GPIO_49 | 23 | 24 | UART1_TXD |
| GPIO_117 | 25 | 26 | UART1_RXD |
| GPIO_115 | 27 | 28 | SPI1_CS0 |
| SPI1_D0 | 29 | 30 | GPIO_112 |
| SPI1_SCLK | 31 | 32 | VDD_ADC |
| AIN4 | 33 | 34 | GNDA_ADC |
| AIN6 | 35 | 36 | AIN5 |
| AIN2 | 37 | 38 | AIN3 |
| AIN0 | 39 | 40 | AIN1 |
| GPIO_20 | 41 | 42 | ECAPPWM0 |
| DGND | 43 | 44 | DGND |
| DGND | 45 | 46 | DGND |

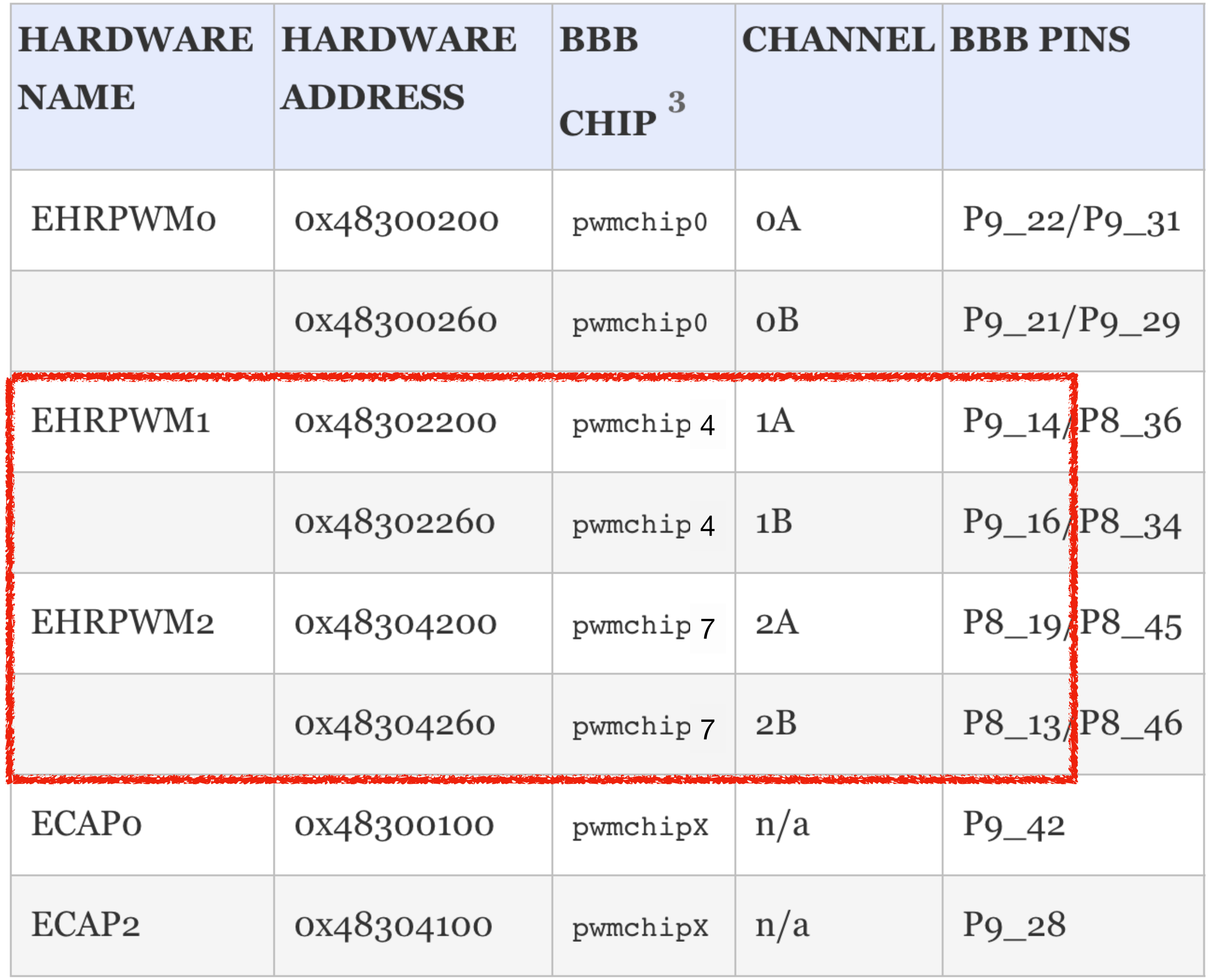
P8

| | | | |
|---|---|---|---|
| DGND | 1 | 2 | DGND |
| MMC1_DAT6 | 3 | 4 | MMC1_DAT7 |
| MMC1_DAT2 | 5 | 6 | MMC1_DAT3 |
| GPIO_66 | 7 | 8 | GPIO_67 |
| GPIO_69 | 9 | 10 | GPIO_68 |
| GPIO_45 | 11 | 12 | GPIO_44 |
| EHRPWM2B | 13 | 14 | GPIO_26 |
| GPIO_47 | 15 | 16 | GPIO_46 |
| GPIO_27 | 17 | 18 | GPIO_65 |
| EHRPWM2A | 19 | 20 | MMC1_CMD |
| MMC1_CLK | 21 | 22 | MMC1_DAT5 |
| MMC1_DAT4 | 23 | 24 | MMC1_DAT1 |
| MMC1_DAT0 | 25 | 26 | GPIO_61 |
| LCD_VSYNC | 27 | 28 | LCD_PCLK |
| LCD_HSYNC | 29 | 30 | LCD_AC_BIAS |
| LCD_DATA14 | 31 | 32 | LCD_DATA15 |
| LCD_DATA13 | 33 | 34 | LCD_DATA11 |
| LCD_DATA12 | 35 | 36 | LCD_DATA10 |
| LCD_DATA8 | 37 | 38 | LCD_DATA9 |
| LCD_DATA6 | 39 | 40 | LCD_DATA7 |
| LCD_DATA4 | 41 | 42 | LCD_DATA5 |
| LCD_DATA2 | 43 | 44 | LCD_DATA3 |
| LCD_DATA0 | 45 | 46 | LCD_DATA1 |

| LEGEND |
|---|
| POWER/GROUND/RESET |
| AVAILABLE DIGITAL |
| AVAILABLE PWM |
| SHARED I2C BUS |
| RECONFIGURABLE DIGITAL |
| ANALOG INPUTS (1.8V) |

# Pulse Width Modulation (PWM)

- Generation of control signals for motors and certain types of actuators (e.g. LEDs, buzzers etc.)
- Eight PWM outputs on the AM335x,
  - three eHRPWM modules (two outputs each), and two eCAP modules



| HARDWARE NAME | HARDWARE ADDRESS | BBB CHIP[3] | CHANNEL | BBB PINS |
|---|---|---|---|---|
| EHRPWM0 | 0x48300200 | pwmchip0 | 0A | P9_22/P9_31 |
|  | 0x48300260 | pwmchip0 | 0B | P9_21/P9_29 |
| EHRPWM1 | 0x48302200 | pwmchip 4 | 1A | P9_14/P8_36 |
|  | 0x48302260 | pwmchip 4 | 1B | P9_16/P8_34 |
| EHRPWM2 | 0x48304200 | pwmchip 7 | 2A | P8_19/P8_45 |
|  | 0x48304260 | pwmchip 7 | 2B | P8_13/P8_46 |
| ECAP0 | 0x48300100 | pwmchipX | n/a | P9_42 |
| ECAP2 | 0x48304100 | pwmchipX | n/a | P9_28 |

# PWM Filesystem Programming

- Configure PWM device using the following commands:

```
cd /sys/class/pwm/
```

[Move to PWM directory]

```
cd pwmchip7
```

[Enter pwm device corresponding to EHRPWM2]

```
echo 1 > export
```

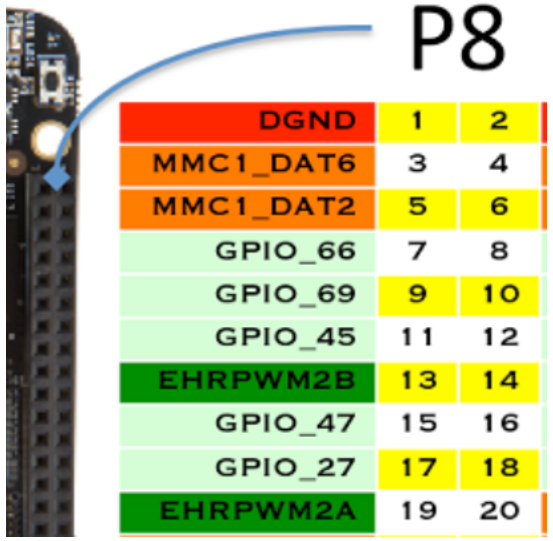
[Configure **channel B** of **EHRPWM2**, which corresponds to pin..

.. **P8_13**]

** Skip the steps above if there is already a folder named “pwm-7:1” inside pwmchip7

- You will find the directory “`pwm-7:1`” inside `pwmchip7`

```
/sys/class/pwm/pwmchip7/pwm-7:1/
```



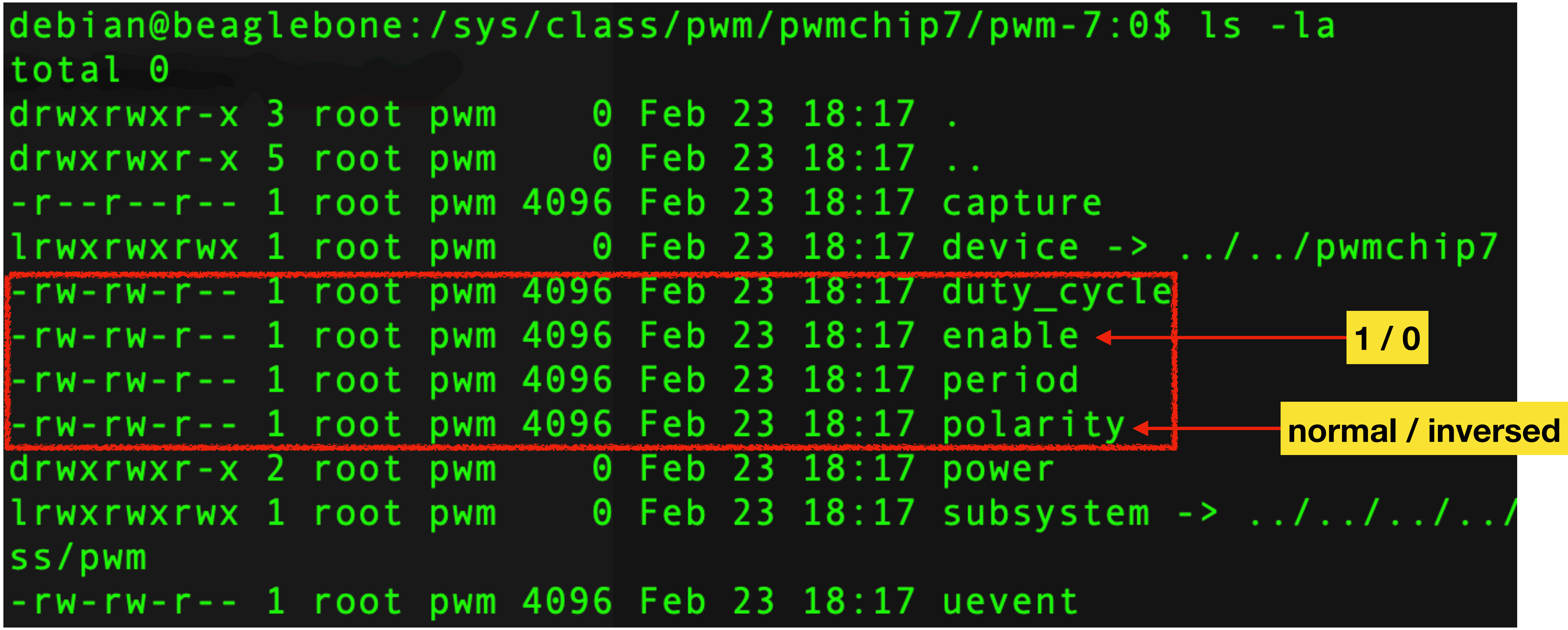
P8

| | | |
|---|---|---|
| DGND | 1 | 2 |
| MMC1_DAT6 | 3 | 4 |
| MMC1_DAT2 | 5 | 6 |
| GPIO_66 | 7 | 8 |
| GPIO_69 | 9 | 10 |
| GPIO_45 | 11 | 12 |
| EHRPWM2B | 13 | 14 |
| GPIO_47 | 15 | 16 |
| GPIO_27 | 17 | 18 |
| EHRPWM2A | 19 | 20 |

# PWM Filesystem Programming

- Check other PWM interfaces
  - **channel A** of **EHRPWM2** which is pin **P8_19** will create a directory “`pwm-7:0`”

`/sys/class/pwm/pwmchip7/pwm-7:0/`



```
debian@beaglebone:/sys/class/pwm/pwmchip7/pwm-7:0$ ls -la
total 0
drwxrwxr-x 3 root pwm    0 Feb 23 18:17 .
drwxrwxr-x 5 root pwm    0 Feb 23 18:17 ..
-r--r--r-- 1 root pwm 4096 Feb 23 18:17 capture
lrwxrwxrwx 1 root pwm    0 Feb 23 18:17 device -> ../../pwmchip7
-rw-rw-r-- 1 root pwm 4096 Feb 23 18:17 duty_cycle
-rw-rw-r-- 1 root pwm 4096 Feb 23 18:17 enable
-rw-rw-r-- 1 root pwm 4096 Feb 23 18:17 period
-rw-rw-r-- 1 root pwm 4096 Feb 23 18:17 polarity
drwxrwxr-x 2 root pwm    0 Feb 23 18:17 power
lrwxrwxrwx 1 root pwm    0 Feb 23 18:17 subsystem -> ../../../../
ss/pwm
-rw-rw-r-- 1 root pwm 4096 Feb 23 18:17 uevent
```

1 / 0

normal / inversed

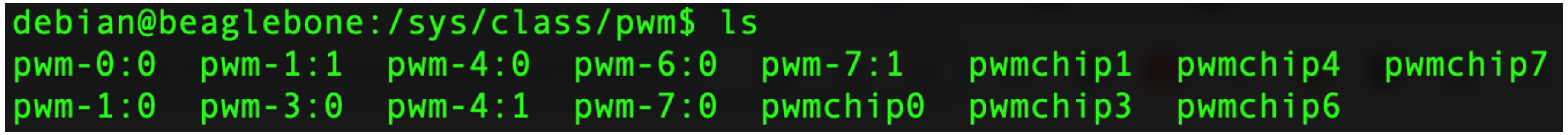
# PWM Filesystem Programming

- Configure the PWM devices and enable it using the following commands:

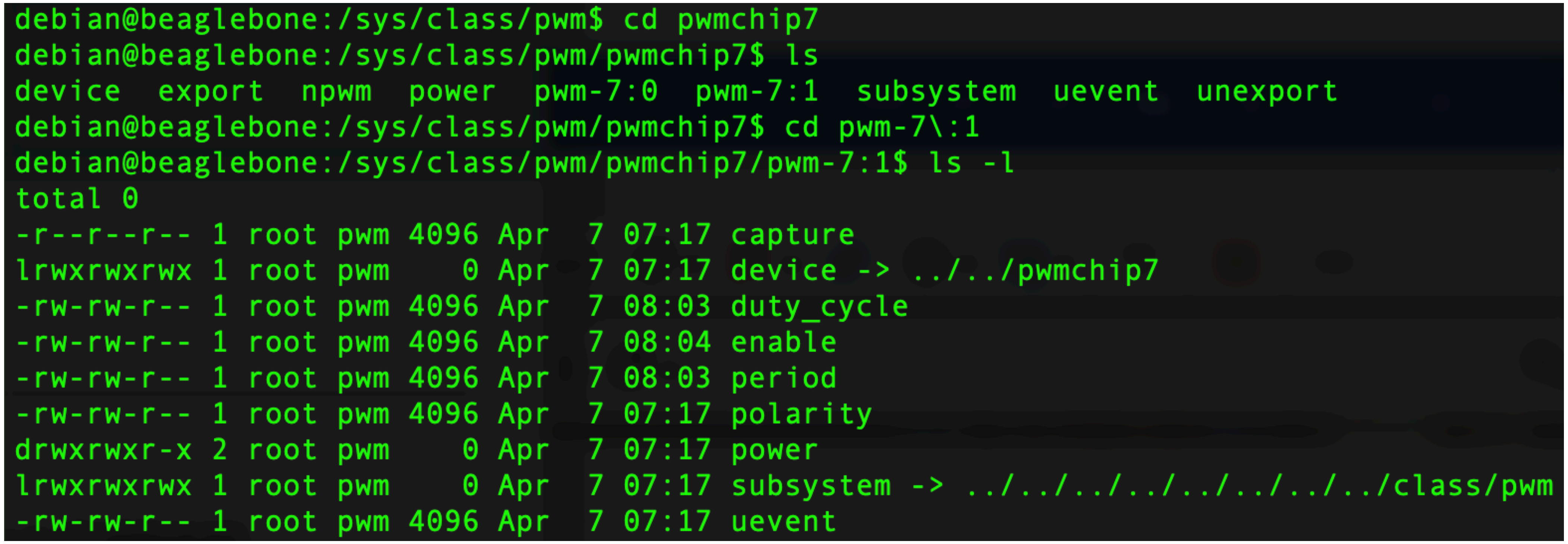
| | |
|---|---|
| `cd pwm-7:1` | [Enter newly created directory to enable PWM and..<br>.. set it's period and duty cycle] |
| `echo 1000000000 > period` | [Setting PWM period to be 1 second = $10^9$ nanosec] |
| `echo 250000000 > duty_cycle` | [Setting duty cycle to be 0.25 second] |
| `echo 1 > enable` | [Enable the PWM] |
| `echo 0 > enable` | [Disables PWM on the pin] |
| `cat polarity` | [Check if  the polarity is normal or inversed] |

# PWM Filesystem Programming

- /sys/class/pwm



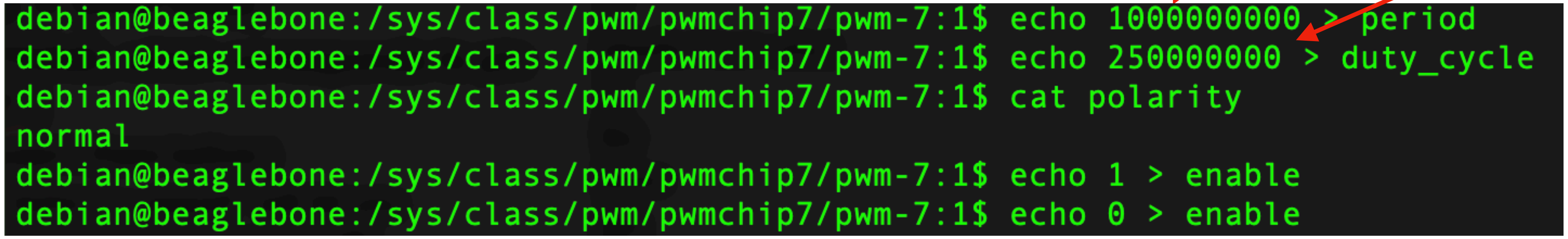
```
debian@beaglebone:/sys/class/pwm$ ls
pwm-0:0  pwm-1:1  pwm-4:0  pwm-6:0  pwm-7:1   pwmchip1  pwmchip4  pwmchip7
pwm-1:0  pwm-3:0  pwm-4:1  pwm-7:0  pwmchip0  pwmchip3  pwmchip6
```



```
debian@beaglebone:/sys/class/pwm$ cd pwmchip7
debian@beaglebone:/sys/class/pwm/pwmchip7$ ls
device  export  npwm  power  pwm-7:0  pwm-7:1  subsystem  uevent  unexport
debian@beaglebone:/sys/class/pwm/pwmchip7$ cd pwm-7\:1
debian@beaglebone:/sys/class/pwm/pwmchip7/pwm-7:1$ ls -l
total 0
-r--r--r-- 1 root pwm 4096 Apr  7 07:17 capture
lrwxrwxrwx 1 root pwm    0 Apr  7 07:17 device -> ../../pwmchip7
-rw-rw-r-- 1 root pwm 4096 Apr  7 08:03 duty_cycle
-rw-rw-r-- 1 root pwm 4096 Apr  7 08:04 enable
-rw-rw-r-- 1 root pwm 4096 Apr  7 08:03 period
-rw-rw-r-- 1 root pwm 4096 Apr  7 07:17 polarity
drwxrwxr-x 2 root pwm    0 Apr  7 07:17 power
lrwxrwxrwx 1 root pwm    0 Apr  7 07:17 subsystem -> ../../../../../../../class/pwm
-rw-rw-r-- 1 root pwm 4096 Apr  7 07:17 uevent
```

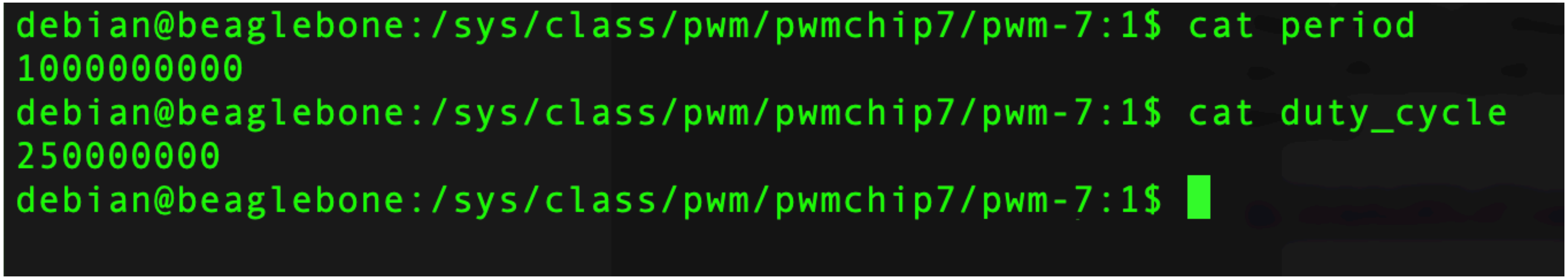
1 second

0.25 second



```
debian@beaglebone:/sys/class/pwm/pwmchip7/pwm-7:1$ echo 1000000000 > period
debian@beaglebone:/sys/class/pwm/pwmchip7/pwm-7:1$ echo 250000000 > duty_cycle
debian@beaglebone:/sys/class/pwm/pwmchip7/pwm-7:1$ cat polarity
normal
debian@beaglebone:/sys/class/pwm/pwmchip7/pwm-7:1$ echo 1 > enable
debian@beaglebone:/sys/class/pwm/pwmchip7/pwm-7:1$ echo 0 > enable
```

# Important!



```
debian@beaglebone:/sys/class/pwm/pwmchip7/pwm-7:1$ cat period
1000000000
debian@beaglebone:/sys/class/pwm/pwmchip7/pwm-7:1$ cat duty_cycle
250000000
debian@beaglebone:/sys/class/pwm/pwmchip7/pwm-7:1$ 
```

NOTE:

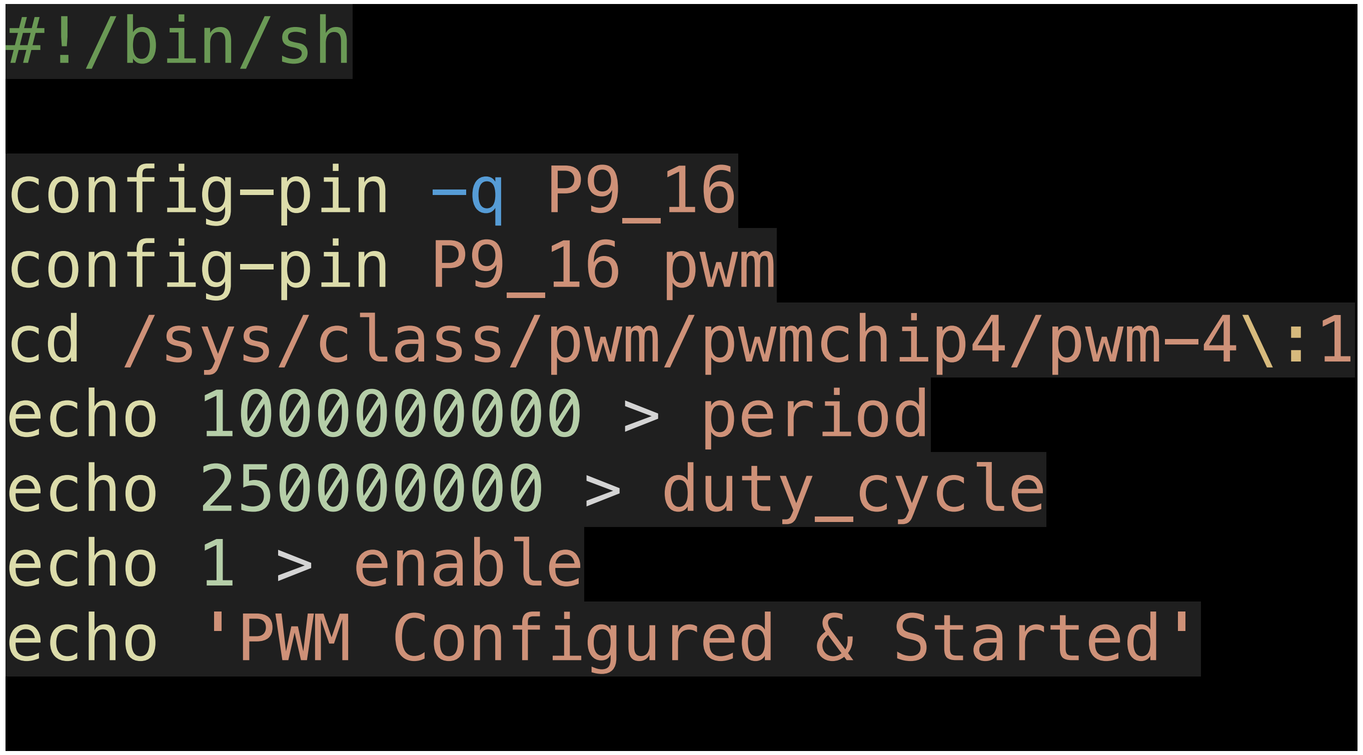
If the default period and duty_cycle values exist, and if

**new_period_value < old_duty_cycle_value**,

configure the duty_cycle before the PWM period

# PWM using Shell script

- Write a shell script that configures a pin to generate a PWM of a specific period and duty cycle

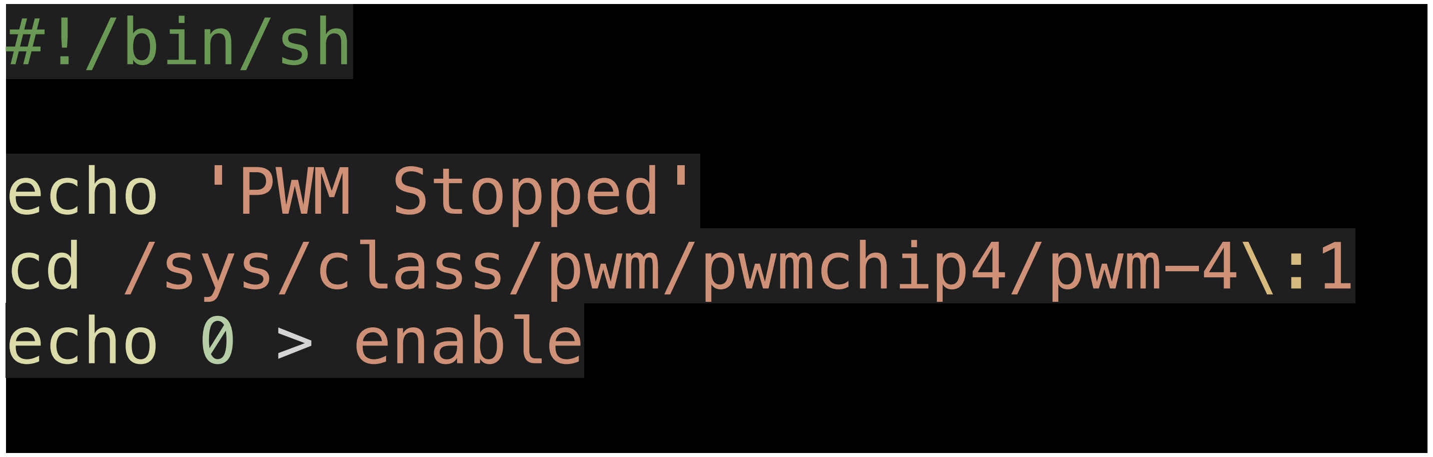


```
#!/bin/sh

config-pin -q P9_16
config-pin P9_16 pwm
cd /sys/class/pwm/pwmchip4/pwm-4\:1
echo 1000000000 > period
echo 250000000 > duty_cycle
echo 1 > enable
echo 'PWM Configured & Started'
```

**NOTE: Don’t forget to configure the pin mode to PWM**

# PWM using Shell script

- Write a shell script that stops the pwm generation



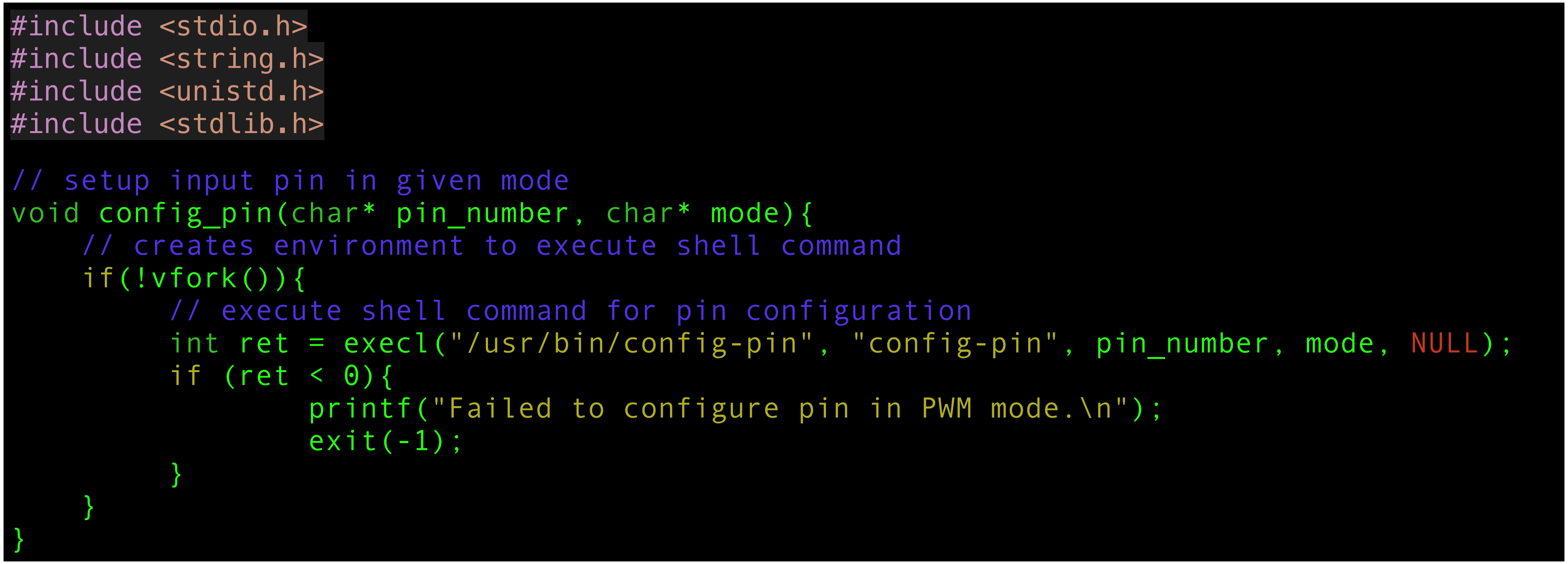
```
#!/bin/sh

echo 'PWM Stopped'
cd /sys/class/pwm/pwmchip4/pwm-4\:1
echo 0 > enable
```

**NOTE:** ALWAYS REMEMBER TO STOP THE PWM

# Actuation Signal Generation
# PWM Generation in C

# PWM Mode Configuration

Modular functions for different PWM parameter setting



```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

// setup input pin in given mode
void config_pin(char* pin_number, char* mode){
    // creates environment to execute shell command
    if(!vfork()){
         // execute shell command for pin configuration
         int ret = execl("/usr/bin/config-pin", "config-pin", pin_number, mode, NULL);
         if (ret < 0){
                 printf("Failed to configure pin in PWM mode.\n");
                 exit(-1);
         }
    }
}
```

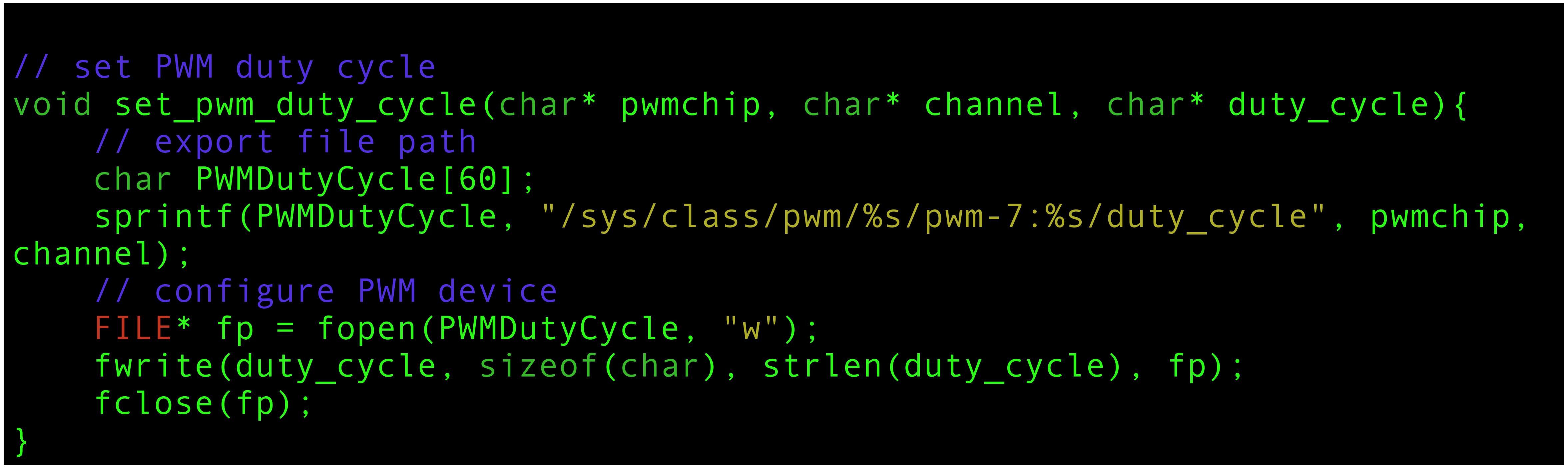
- This function invokes config-pin utility in C code to change default pin functionality to PWM

>> config-pin -q P8_13

>> config-pin P8_13 pwm

# PWM Generation - Set Duty Cycle

Modular functions for different PWM parameter setting



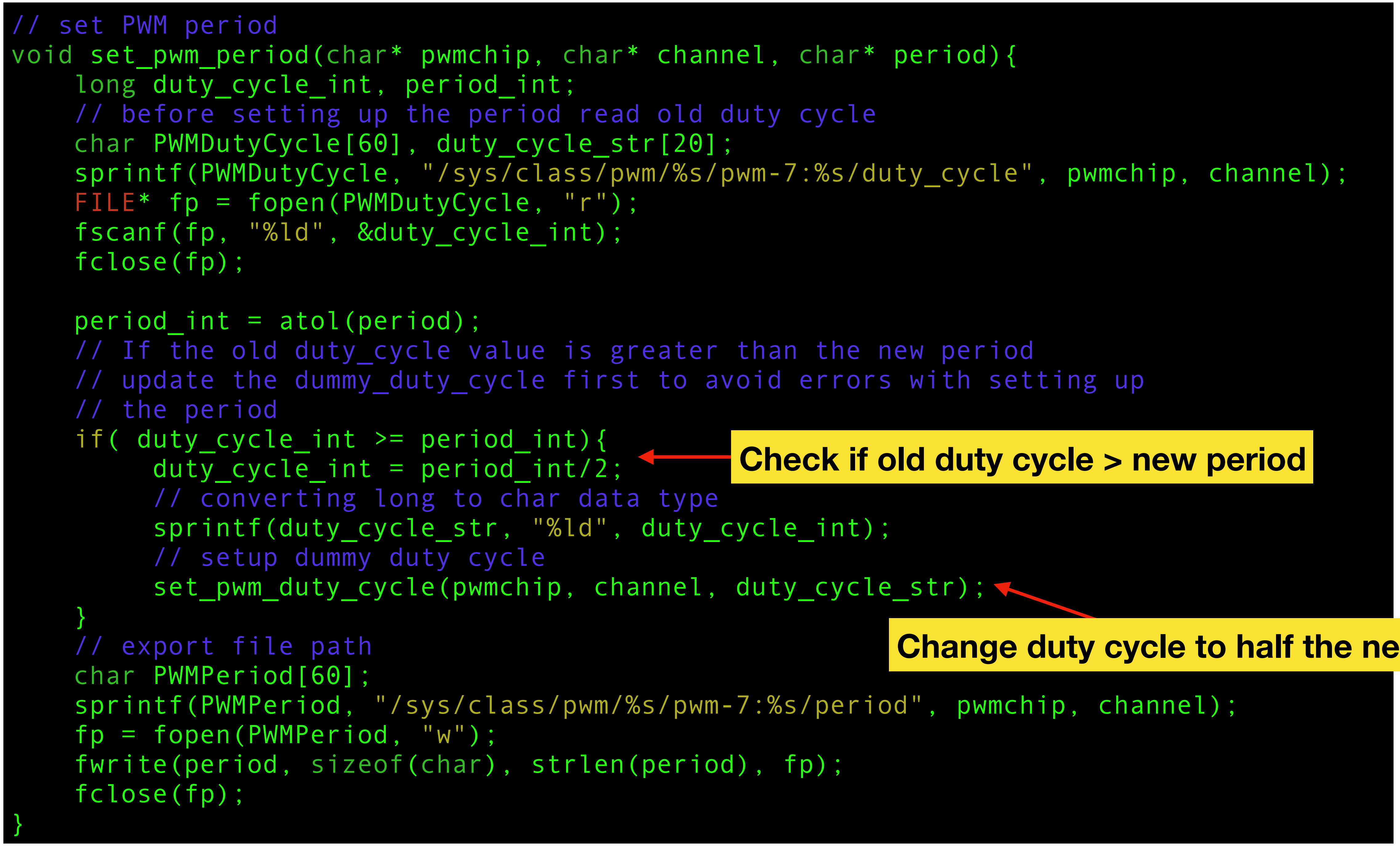
```
// set PWM duty cycle
void set_pwm_duty_cycle(char* pwmchip, char* channel, char* duty_cycle){
    // export file path
    char PWMDutyCycle[60];
    sprintf(PWMDutyCycle, "/sys/class/pwm/%s/pwm-7:%s/duty_cycle", pwmchip,
channel);
    // configure PWM device
    FILE* fp = fopen(PWMDutyCycle, "w");
    fwrite(duty_cycle, sizeof(char), strlen(duty_cycle), fp);
    fclose(fp);
}
```

- Open the duty cycle file at the exact PWM chip path,

`/sys/class/pwm/%s/pwm-7:%s/duty_cycle`

- Write the duty cycle value to the duty cycle file

# PWM Generation - Set Period



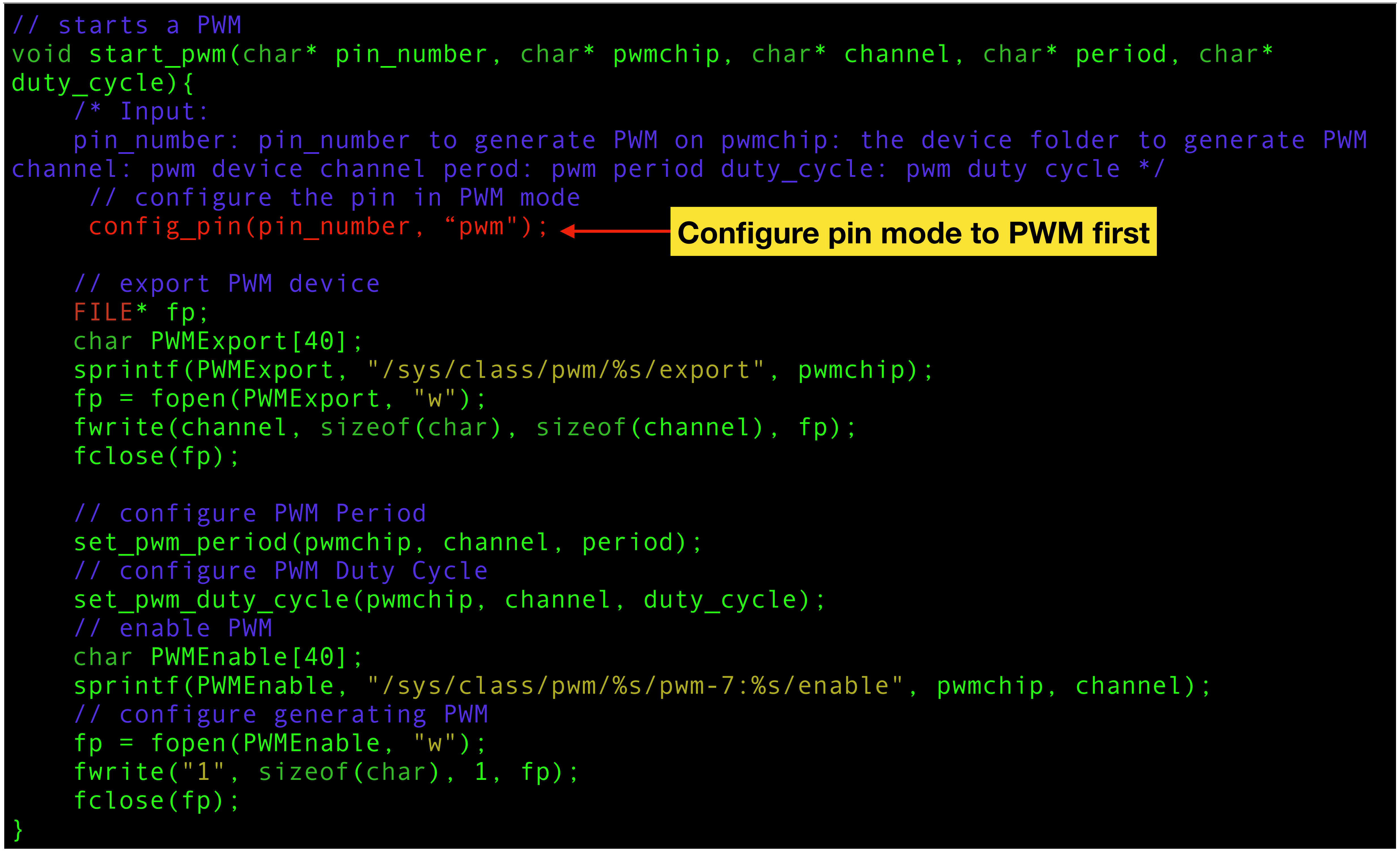
```c
// set PWM period
void set_pwm_period(char* pwmchip, char* channel, char* period){
    long duty_cycle_int, period_int;
    // before setting up the period read old duty cycle
    char PWMDutyCycle[60], duty_cycle_str[20];
    sprintf(PWMDutyCycle, "/sys/class/pwm/%s/pwm-7:%s/duty_cycle", pwmchip, channel);
    FILE* fp = fopen(PWMDutyCycle, "r");
    fscanf(fp, "%ld", &duty_cycle_int);
    fclose(fp);

    period_int = atol(period);
    // If the old duty_cycle value is greater than the new period
    // update the dummy_duty_cycle first to avoid errors with setting up
    // the period
    if( duty_cycle_int >= period_int){
        duty_cycle_int = period_int/2;
        // converting long to char data type
        sprintf(duty_cycle_str, "%ld", duty_cycle_int);
        // setup dummy duty cycle
        set_pwm_duty_cycle(pwmchip, channel, duty_cycle_str);
    }
    // export file path
    char PWMPeriod[60];
    sprintf(PWMPeriod, "/sys/class/pwm/%s/pwm-7:%s/period", pwmchip, channel);
    fp = fopen(PWMPeriod, "w");
    fwrite(period, sizeof(char), strlen(period), fp);
    fclose(fp);
}
```

**Check if old duty cycle > new period**

**Change duty cycle to half the new period**

- The above duty cycle condition should be checked before setting up a new period

# PWM Generation - Start PWM

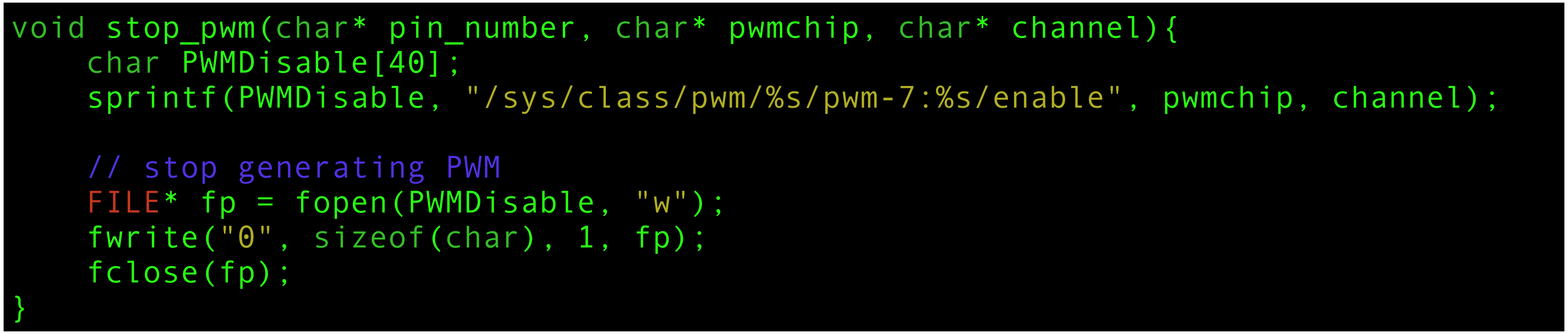


```
// starts a PWM
void start_pwm(char* pin_number, char* pwmchip, char* channel, char* period, char*
duty_cycle){
    /* Input:
    pin_number: pin_number to generate PWM on pwmchip: the device folder to generate PWM
channel: pwm device channel perod: pwm period duty_cycle: pwm duty cycle */
     // configure the pin in PWM mode
     config_pin(pin_number, “pwm");

    // export PWM device
    FILE* fp;
    char PWMExport[40];
    sprintf(PWMExport, "/sys/class/pwm/%s/export", pwmchip);
    fp = fopen(PWMExport, "w");
    fwrite(channel, sizeof(char), sizeof(channel), fp);
    fclose(fp);

    // configure PWM Period
    set_pwm_period(pwmchip, channel, period);
    // configure PWM Duty Cycle
    set_pwm_duty_cycle(pwmchip, channel, duty_cycle);
    // enable PWM
    char PWMEnable[40];
    sprintf(PWMEnable, "/sys/class/pwm/%s/pwm-7:%s/enable", pwmchip, channel);
    // configure generating PWM
    fp = fopen(PWMEnable, "w");
    fwrite("1", sizeof(char), 1, fp);
    fclose(fp);
}
```

**Configure pin mode to PWM first**

Input arguments are used to set pwm at a particular pin and channel with desired period and duty cycle

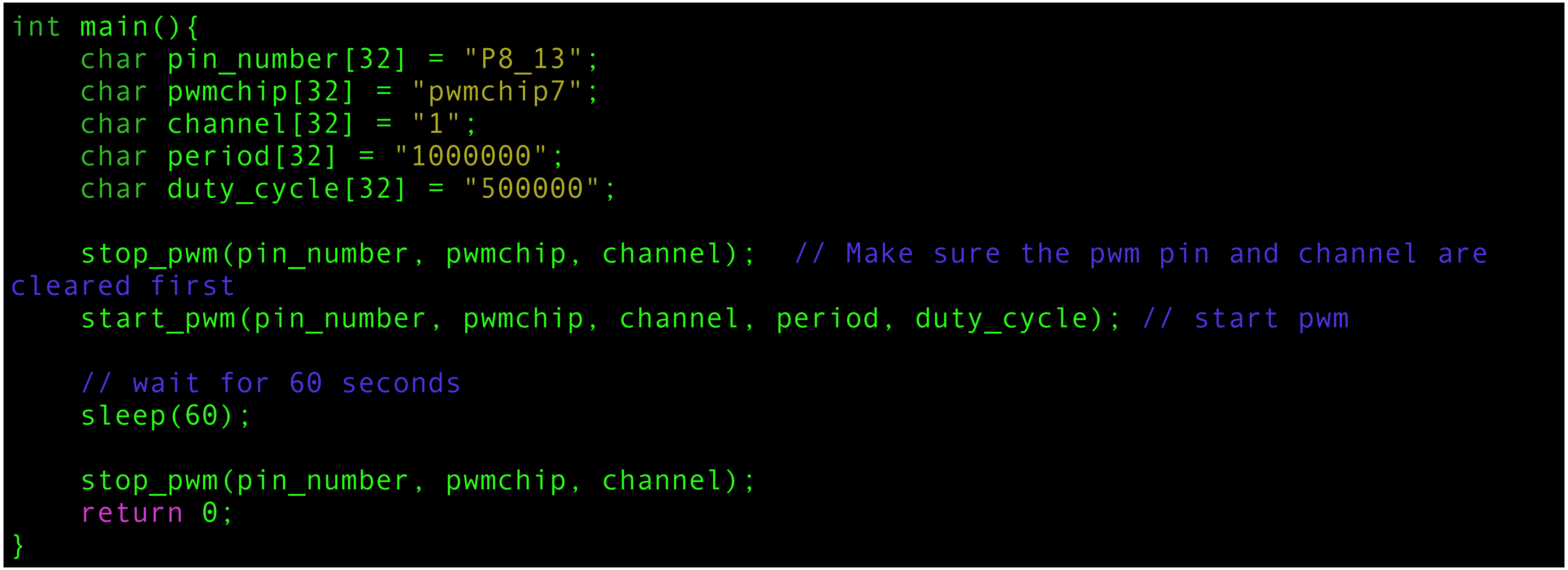
# PWM Generation - Stop PWM



```
void stop_pwm(char* pin_number, char* pwmchip, char* channel){
    char PWMDisable[40];
    sprintf(PWMDisable, "/sys/class/pwm/%s/pwm-7:%s/enable", pwmchip, channel);

    // stop generating PWM
    FILE* fp = fopen(PWMDisable, "w");
    fwrite("0", sizeof(char), 1, fp);
    fclose(fp);
}
```

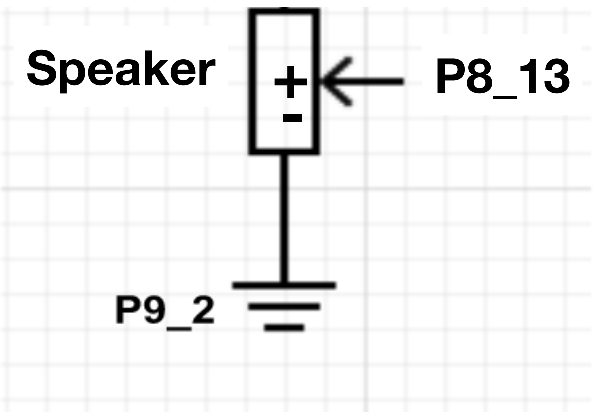
- A good practice to clear PWM state before attempting to start a new one
- Always stop PWM after completion to avoid file system issues

# PWM Generation in C code



```
int main(){
    char pin_number[32] = "P8_13";
    char pwmchip[32] = "pwmchip7";
    char channel[32] = "1";
    char period[32] = "1000000";
    char duty_cycle[32] = "500000";

    stop_pwm(pin_number, pwmchip, channel);  // Make sure the pwm pin and channel are
cleared first
    start_pwm(pin_number, pwmchip, channel, period, duty_cycle); // start pwm

    // wait for 60 seconds
    sleep(60);

    stop_pwm(pin_number, pwmchip, channel);
    return 0;
}
```

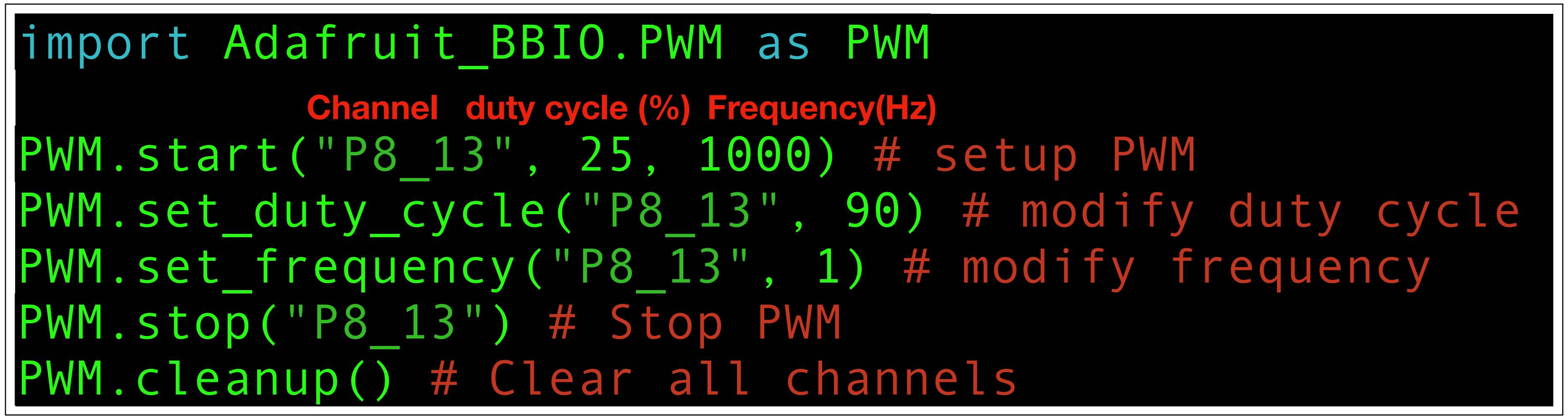
- A good practice to clear PWM state before attempting to start a new one
- Stop PWM after completion to avoid lingering state



**Optional** **Requires Internet on BBB**

# PWM Generation using Python

- Create a new file “pwm_test.py”, and copy the following code,



```
import Adafruit_BBIO.PWM as PWM
```

Channel duty cycle (%) Frequency(Hz)

```
PWM.start("P8_13", 25, 1000) # setup PWM
PWM.set_duty_cycle("P8_13", 90) # modify duty cycle
PWM.set_frequency("P8_13", 1) # modify frequency
PWM.stop("P8_13") # Stop PWM
PWM.cleanup() # Clear all channels
```

- Save and compile your python code. Then run it. PWM on the pin can be visualized using a logic analyzer

***Install PWM & GPIO Libraries**:*
*https://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black/installation-on-ubuntu*

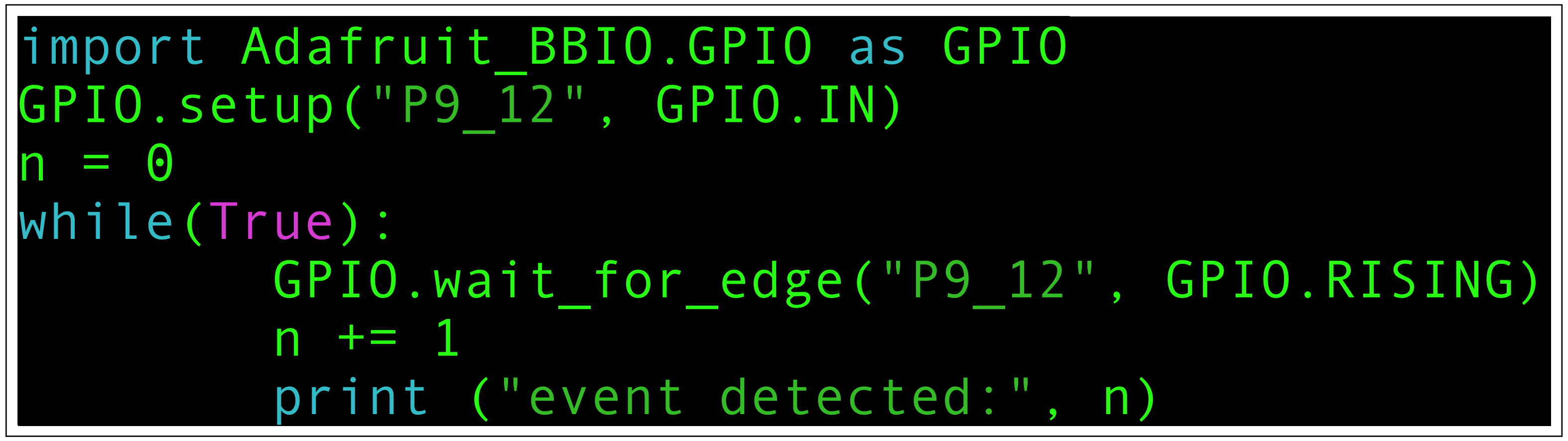
How to test that PWM is working?

**Optional**

**Requires Internet on BBB**

# PWM Testing using Python

- Use logic analyzer to validate pin, PWM frequency and duty cycle
- Alternatively,
  - Hook the PWM pin to a GPIO using a jumper wire
  - Install Adafruit_BBIO.GPIO library on your device and run the following python code,

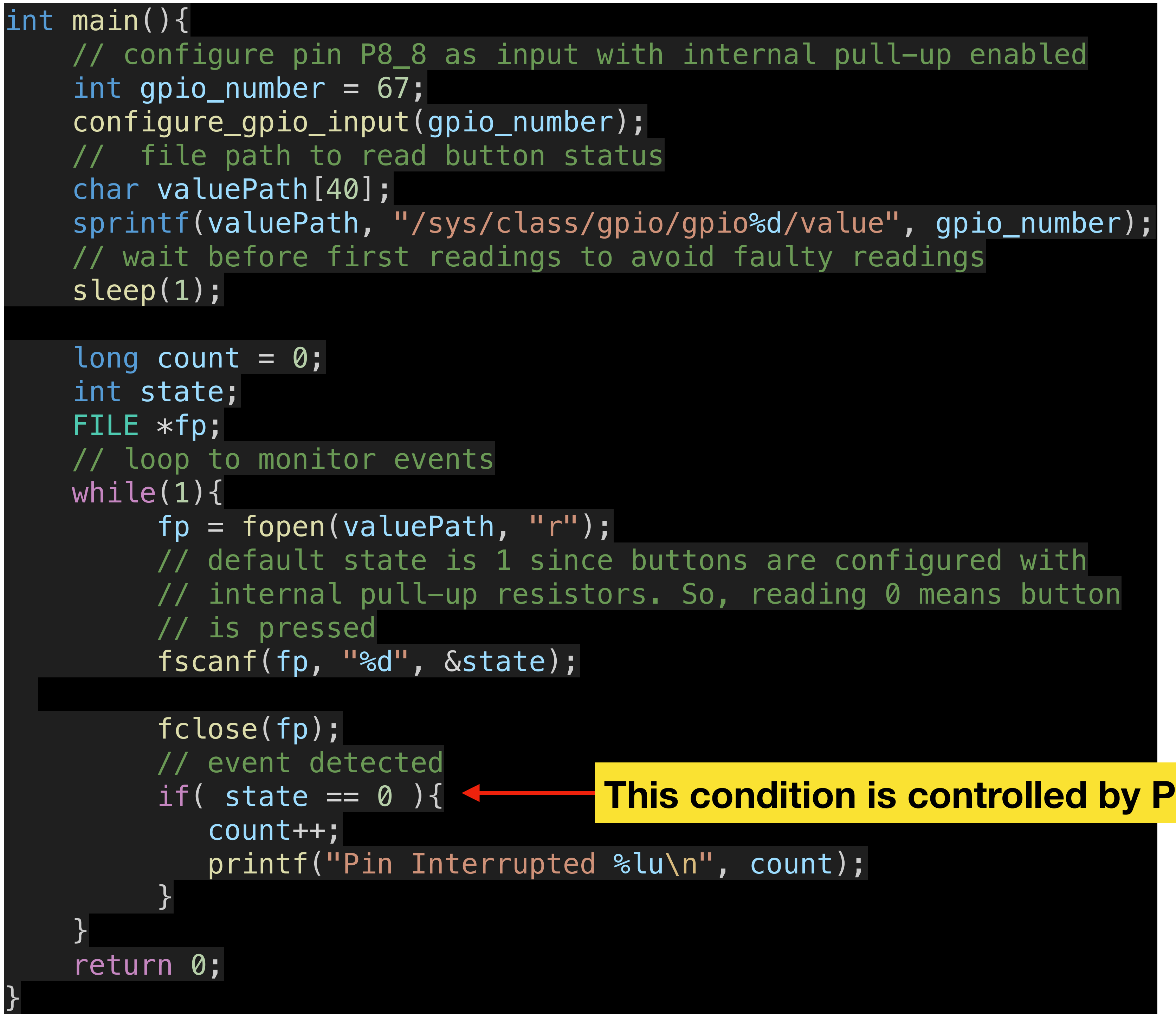


```
import Adafruit_BBIO.GPIO as GPIO
GPIO.setup("P9_12", GPIO.IN)
n = 0
while(True):
        GPIO.wait_for_edge("P9_12", GPIO.RISING)
        n += 1
        print ("event detected:", n)
```

- PWM will trigger the GPIO pin and “event detected” statement is printed

# PWM Testing using C

- Attach the PWM pin to a GPIO using a jumper wire
- PWM will trigger the GPIO pin and desired statement is printed



```c
int main(){
    // configure pin P8_8 as input with internal pull-up enabled
    int gpio_number = 67;
    configure_gpio_input(gpio_number);
    //  file path to read button status
    char valuePath[40];
    sprintf(valuePath, "/sys/class/gpio/gpio%d/value", gpio_number);
    // wait before first readings to avoid faulty readings
    sleep(1);

    long count = 0;
    int state;
    FILE *fp;
    // loop to monitor events
    while(1){
        fp = fopen(valuePath, "r");
        // default state is 1 since buttons are configured with
        // internal pull-up resistors. So, reading 0 means button
        // is pressed
        fscanf(fp, "%d", &state);

        fclose(fp);
        // event detected
        if( state == 0 ){
            count++;
            printf("Pin Interrupted %lu\n", count);
        }
    }
    return 0;
}
```

**This condition is controlled by PWM duty cycle**

# Reading

- Practice all these concepts/commands on your shells
- Book: Read Chapter 6: Interfacing to Beagleboard Inputs/Outputs
  - Go through only relevant topics