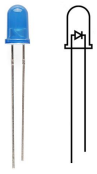
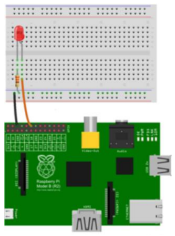
COMPUTER SYSTEM

DR. NGUYEN DANG KHOA

FACULTY OF ELECTRICAL AND ELECTRONIC ENGINEERING

LED control





Raspberry Pi 3 GPIO Header

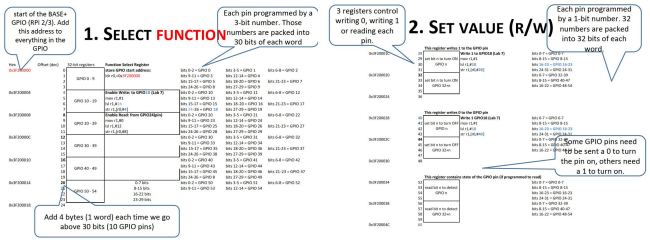
Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I²C)	DC Power 5v	04
05	GPIO03 (SCL1 , I²C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)	(I²C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

Rev. 2 29/02/2016 www.element14.com/RaspberryPi

https://www.youtube.com/watch?v=Rd9kvVs1ISQ

Sample Programming: TURN ON/OFF LED

Map memory and register



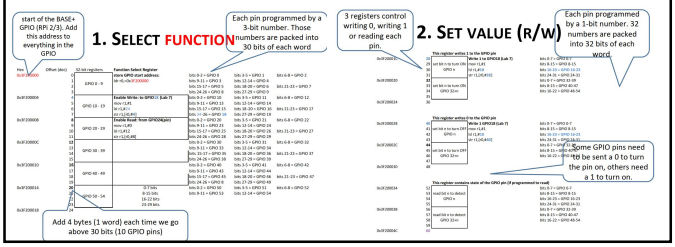
Sample Programming: TURN ON/OFF LED

How to use?

Like other microchip Arduino, PIC, AVR... We need to have 2 steps

Step 1: Configuration for GPIO in input/output mode ➔ *1. Select function*

Step 2: Set value ON/OFF for GPIO ➔ *2. Set value (R/W)*

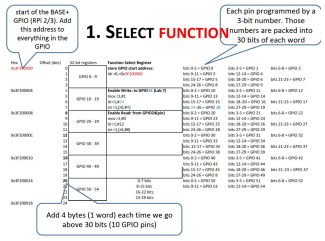


Sample Programming: TURN ON/OFF LED

How to use?

Like other microchip Arduino, PIC, AVR... We need to have 2 steps

Step 1: Configuration for GPIO in input/output mode ➔ *1. Select function*



From 0x3F200000 to 0x3F200018

➔ *Present for GPIO0 to GPIO54*

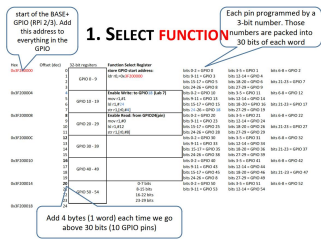
➔ *How to config?*

Sample Programming: TURN ON/OFF LED

How to use?

Like other microchip Arduino, PIC, AVR... We need to have 2 steps

Step 1: Configuration for GPIO in input/output mode ➔ *1. Select function*



For each GPIO, 3 bits to config is shown

Bit pattern	Pin Function
000	The pin is an input
001	The pin is an output
100	The pin does alternate function 0
101	The pin does alternate function 1
110	The pin does alternate function 2
111	The pin does alternate function 3
011	The pin does alternate function 4
010	The pin does alternate function 5

We will consider with 000 and 001

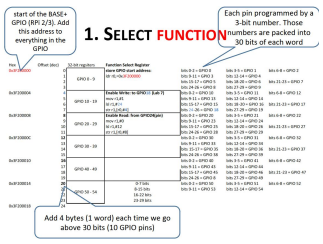
You could see in the 1. Select function ➔ GPIO0 = bit 0-2 ➔ 3 bits...

Sample Programming: TURN ON/OFF LED

How to use?

Like other microchip Arduino, PIC, AVR... We need to have 2 steps

Step 1: Configuration for GPIO in input/output mode ➔ *1. Select function*



For example: Defining GPIO35 is output

- *GPIO35 is 0x3F20000C to 0x3F200010 (or offset 12)*

➔ *32 bit*

31	30	....	17	16	15	...	...	1	0

- *GPIO35 is bit 15-17* ➔ *Set output configuration*

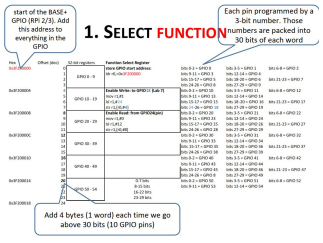
0	0	0	0	0	1	...	...	0	0

Sample Programming: TURN ON/OFF LED

How to use?

Like other microchip Arduino, PIC, AVR... We need to have 2 steps

Step 1: Configuration for GPIO in input/output mode ➔ *1. Select function*



For example: Defining GPIO35 is output

- *GPIO35 is bit 15-17* ➔ *Set output configuration*

0	0	0	0	0	1	...	...	0	0

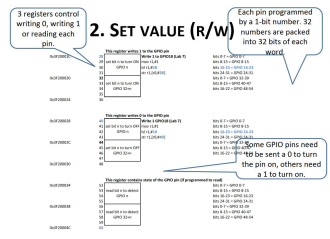
```
BASE    =    $3F000000
GPIO_OFFSET=$200000
mov     r0,BASE
orr     r0,GPIO_OFFSET
mov     r1,#1                ; R1=1
lsl     r1,#15               ; shift 15 times
str     r1,[r0,#12]          ; [R0+12]=R1
```

Sample Programming: TURN ON/OFF LED

How to use?

Like other microchip Arduino, PIC, AVR... We need to have 2 steps

Step 2: Set value ON/OFF for GPIO ➔ *2. Set value (R/W)*



For example: Defining GPIO35 is output

- *GPIO35 is 0x3F200020 to 0x3F200024 (or offset 32)*

Bit 3th

7	6	5	4	3	2	1	0
GPIO39	GPIO38	GPIO37	GPIO36	GPIO35	GPIO34	GPIO33	GPIO32
0	0	0	0	1	0	0	0

➔ *For set ON*

- *GPIO35 is 0x3F20002C to 0x3F200030 (or offset 44)*

Bit 3th

➔ *For set OFF*

Sample Programming: TURN ON/OFF LED

How to use?

Like other microchip Arduino, PIC, AVR... We need to have 2 steps

Step 2: Set value ON/OFF for GPIO ➔ *2. Set value (R/W)*

```
mov    r1,#1          ; R1=1
lsl    r1,#3          ; 3 times
str    r1,[r0,#32]    ; [R0+32]=R1 ➔ ON

mov    r1,#1          ; R1=1
lsl    r1,#3          ; 3 times
str    r1,[r0,#44]    ; [R0+44]=R1 ➔ OFF
```

For example: Defining GPIO35 is output

- *GPIO35 is 0x3F200020 to 0x3F200024 (or offset 32)*

Bit 3th

7	6	5	4	3	2	1	0
GPIO39	GPIO38	GPIO37	GPIO36	GPIO35	GPIO34	GPIO33	GPIO32
0	0	0	0	1	0	0	0

➔ *For set ON*

- *GPIO35 is 0x3F20002C to 0x3F200030 (or offset 44)*

Bit 3th

➔ *For set OFF*

Sample Programming: TURN ON/OFF LED

Full program for GPIO18

```
macro delay {

local .wait
mov r2,#0x3F0000

.wait:
    sub r2,#1
    cmp r2,#0
    bne .wait
}

BASE    =    $3F000000
GPIO_OFFSET=$200000
mov    r0,BASE
orr    r0,GPIO_OFFSET
mov    r1,#1
lsl    r1,#24
str    r1,[r0,#4]  ; finished select GPIO18
```

```
loop$:

mov    r1,#1
lsl    r1,#18
str    r1,[r0,#28]   ; 28=LED ON; 40=LED OFF
delay


mov    r1,#1
lsl    r1,#18
str    r1,[r0,#40]   ; 28=LED ON; 40=LED OFF
delay

b      loop$
```

Sample Programming: TURN ON/OFF LED

Example: Blink LED in GPIO9 and GPIO18 (15 mins)

Sample Programming: TURN ON/OFF LED

Example: Blink LED in GPIO9 and GPIO18 (15 mins)

```
macro delay {

local .wait
mov r2,#0x3F0000

.wait:
    sub r2,#1
    cmp r2,#0
    bne .wait
}

BASE    =       $3F000000
GPIO_OFFSET=$200000
mov     r0,BASE
orr     r0,GPIO_OFFSET
mov     r1,#1
lsl     r1,#24
str     r1,[r0,#4]  ; finished select GPIO18

        mov     r1,#1
        lsl     r1,#27
        str     r1,[r0,#0]  ; finished select GPIO9

loop$:

        mov     r1,#1
        lsl     r1,#18
        mov  r3, #9
        orr  r1,r3
        str     r1,[r0,#28]   ; ON
        delay

        mov     r1,#1
        lsl     r1,#18
        mov  r1, #1
        lsl  r3,#9
        orr  r1,r3

        str     r1,[r0,#40]   ; OFF
        delay
        b       loop$
```

```
loop$:

mov     r1,#1
lsl     r1,#18
mov r3,#1
lsl  r3, #9
orr  r1,r3
str     r1,[r0,#28]   ; ON
delay

str     r1,[r0,#40]   ; OFF
delay
b       loop$
```

Sample Programming: TURN ON/OFF LED

Example: Blink LED in GPIO5 and GPIO6 (5 mins)

Sample Programming: TURN ON/OFF LED

How to use?

Like other microchip Arduino, PIC, AVR... We need to have 2 steps

Step 2: Set value ON/OFF for GPIO ➔ *2. Set value (R/W)*

```
mov     r1,#1           ; R1=1
lsl     r1,#3           ; 3 times
str     r1,[r0,#32]     ; [R0+32]=R1 ➔ ON

mov     r1,#1           ; R1=1
lsl     r1,#3           ; 3 times
str     r1,[r0,#44]     ; [R0+44]=R1 ➔ OFF
```

Need the delay time among 2 states

Because it is very fast

For example: Defining GPIO35 is output

- *GPIO35 is 0x3F200020 to 0x3F200024 (or offset 32)*

Bit 3th

7	6	5	4	3	2	1	0
GPIO39	GPIO38	GPIO37	GPIO36	GPIO35	GPIO34	GPIO33	GPIO32
0	0	0	0	1	0	0	0

➔ *For set ON*

- *GPIO35 is 0x3F20002C to 0x3F200030 (or offset 44)*

Bit 3th

➔ *For set OFF*

Sample Programming: TURN ON/OFF LED

Full program for GPIO18

```
macro delay {

local .wait
mov r2,#0x3F0000

.wait:
    sub r2,#1
    cmp r2,#0
    bne .wait
}

BASE    =    $3F000000
GPIO_OFFSET=$200000
mov    r0,BASE
orr    r0,GPIO_OFFSET
mov    r1,#1
lsl    r1,#24
str    r1,[r0,#4]  ; finished select GPIO18

loop$:

mov    r1,#1
lsl    r1,#18
str    r1,[r0,#28]   ; 28=LED ON; 40=LED OFF
delay


mov    r1,#1
lsl    r1,#18
str    r1,[r0,#40]   ; 28=LED ON; 40=LED OFF
delay

b      loop$
```

Sample Programming: TURN ON/OFF LED

Consider the delay function/delay code

```
; delay function
macro delay {

local .wait
mov r2,#0x3F0000

.wait:
    sub r2,#1
    cmp r2,#0
    bne .wait
}

;delay code (no function)
mov r2,#0x3F0000

.wait:
    sub r2,#1
    cmp r2,#0
    bne .wait
```

Sample Programming: TURN ON/OFF LED

```
; first method
loop$:

mov    r1,#1
lsl    r1,#18
str    r1,[r0,#28]   ; 28=LED ON; 40=LED OFF
delay


mov    r1,#1
lsl    r1,#18
str    r1,[r0,#40]   ; 28=LED ON; 40=LED OFF
delay

b      loop$
```

```
; second method
loop$:
mov    r1,#1
lsl    r1,#18
str    r1,[r0,#28]   ; 28=LED ON; 40=LED OFF
mov r2,#0x3F0000

.wait:
    sub r2,#1
    cmp r2,#0
    bne .wait
mov    r1,#1
lsl    r1,#18
str    r1,[r0,#40]   ; 28=LED ON; 40=LED OFF
mov r2,#0x3F0000

.wait:
    sub r2,#1
    cmp r2,#0
    bne .wait
b      loop$
```

Sample Programming: TURN ON/OFF LED

How to code the delay function/delay code

```
macro delay {

local .wait
mov r2,#0x3F0000 ; [r2]= 0x3F0000

.wait:
    sub r2,#1       ; [r2]=[r2]-1
    cmp r2,#0       ; compare [r2] and zero
    bne .wait       ; if not equal then goto .wait
}
```

NEW COMMAND

For details

- **Command: cmp**

➔ *Compare a register with a value (register...), and store result to Application Program Status Register (APSR)*
Example: cmp r1, #20

- **Command: b**

➔ *unconditional branch*
Example: b loop$ ➔ goto loop$ label

- **Command: beq**

➔ *Branch if equal*
Example:

```
cmp r1, r2
beq loop$  ➔ goto loop if r1==r2. (Access APSR to get result of cmp)
```

- **Command: bge**

➔ *Branch if greater equal*
Example:

```
cmp r1, r2
bge loop$  ➔ goto loop if r1>=r2. (Access APSR to get result of cmp)
```

NEW COMMAND

For details

- **Command: bgt**

➔ *Branch if greater than*
Example:

```
cmp r1, r2
bgt loop$  ➔ goto loop if r1>r2. (Access APSR to get result of cmp)
```

- **Command: ble**

➔ *Branch if lesser equal than*
Example:

```
cmp r1, r2
ble loop$  ➔ goto loop if r1<=r2. (Access APSR to get result of cmp)
```

- **Command: blt**

➔ *Branch if less than*
Example:

```
cmp r1, r2
blt loop$  ➔ goto loop if r1<r2. (Access APSR to get result of cmp)
```

NEW COMMAND

Suffix	Flags	Meaning
EQ	Z set	Equal
NE	Z clear	Not equal
CS or HS	C set	Higher or same (unsigned >=)
CC or LO	C clear	Lower (unsigned <)
MI	N set	Negative
PL	N clear	Positive or zero
VS	V set	Overflow
VC	V clear	No overflow
HI	C set and Z clear	Higher (unsigned >)
LS	C clear or Z set	Lower or same (unsigned <=)
GE	N and V the same	Signed >=
LT	N and V differ	Signed <
GT	Z clear, N and V the same	Signed >
LE	Z set, N and V differ	Signed <=

NEW COMMAND

For example

```
mov r2,#0
loop1:
        add r2,#1       ;1,2,3,4,5,6,7,8,9,10
        cmp r2,#10
bne loop1               ;this counts to 10
```

NEW COMMAND

```
; second method
loop$:
mov    r1,#1
lsl    r1,#18
str    r1,[r0,#28]   ; 28=LED ON; 40=LED OFF
mov r2,#0x3F0000

.wait:
    sub r2,#1
    cmp r2,#0
    bne .wait
mov    r1,#1
lsl    r1,#18
str    r1,[r0,#40]   ; 28=LED ON; 40=LED OFF
mov r2,#0x3F0000

.wait:
    sub r2,#1
    cmp r2,#0
    bne .wait
b     loop$
```

➔ .wait ➔ delay time ➔ get 100% of CPU

➔ *Not accuracy*

➔ ➔ How to replace?

TIMER

(Remind)

5. LDR

ldr r0, [r1]

➔ Load r0 with the value pointed to by r1; Herein, the r0 and r1 are the 32bit register (4 bytes)

6. LDRD

ldrd r6, r7, [r3]

➔ [r3] points to a address 64bit (8 bytes) ➔ r6= 4 lower bytes and r7= 4 higher bytes

7. Sub

sub r8, r6, r5

➔ r8 = r6-r5

TIMER

- RPi has a dedicated timer register:
  - Independent of clock speed.
  - Housed inside the same chip as the ARM CPU, the GPIO, GPU, RAM and most other things.
  - This chip is called the SoC (System on a Chip).
- The Timer registers start at BASE address + 0x3000
  - Timer counts 1 microsecond intervals (10^{-6} second)

TIMER REGISTER

Byte offset (from BASE)	Size / Bytes	Name	Description	Read or Write
0x3000	4	Control / Status	Register used to control and clear timer channel comparator matches.	RW
0x3004	8	Counter	A counter that increments at 1MHz.	R
0x300C	4	Compare 0	0th Comparison register.	RW
0x3010	4	Compare 1	1st Comparison register.	RW
0x3014	4	Compare 2	2nd Comparison register.	RW
0x3018	4	Compare 3	3rd Comparison register.	RW

- *We consider timer configuration in the address 0x3004 which 64 bits (8 bytes)*
- *The frequency is 1MHz ➔ $t=1/f = 1/(1*10^6)=1$(microsecond)*

TIMER REGISTER

- *ARM (Raspberry) uses timer with 16 bit and prescale 1:8*

➔ 8.2^{16}= *524288 (microsecond)*
= *0.524 (second)*

➔ *hexa: $80000*

https://www.st.com/resource/en/datasheet/st72215g2.pdf

https://www.studica.com/blog/raspberry-pi-timer-embedded-environments

11.2 16-BIT TIMER

11.2.1 Introduction

The timer consists of a 16-bit free-running counter driven by a programmable prescaler.

It may be used for a variety of purposes, including measuring the pulse lengths of up to two input signals (*input capture*) or generating up to two output waveforms (*output compare* and *PWM*).

Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the CPU clock prescaler.

Some ST7 devices have two on-chip 16-bit timers. They are completely independent, and do not share any resources. They are synchronized after a MCU reset as long as the timer clock frequencies are not modified.

This description covers one or two 16-bit timers. In ST7 devices with two timers, register names are prefixed with TA (Timer A) or TB (Timer B).

11.2.2 Main Features

- Programmable prescaler: f_{CPU} divided by 2, 4 or 8.
- Overflow status flag and maskable interrupt
- External clock input (must be at least 4 times slower than the CPU clock speed) with the choice of active edge
- Output compare functions with:
  - 2 dedicated 16-bit registers
  - 2 dedicated programmable signals
  - 2 dedicated status flags
  - 1 dedicated maskable interrupt
- Input capture functions with:
  - 2 dedicated 16-bit registers
  - 2 dedicated active edge selection signals
  - 2 dedicated status flags
  - 1 dedicated maskable interrupt
- Pulse Width Modulation mode (PWM)

When reading an input signal on a non-bonded pin, the value will always be '1'.

11.2.3 Functional Description

11.2.3.1 Counter

The main block of the Programmable Timer is a 16-bit free running upcounter and its associated 16-bit registers. The 16-bit registers are made up of two 8-bit registers called high and low.

Counter Register (CR)
- Counter High Register (CHR) is the most significant byte (MS Byte).
- Counter Low Register (CLR) is the least significant byte (LS Byte).

Alternate Counter Register (ACR)
- Alternate Counter High Register (ACHR) is the most significant byte (MS Byte).
- Alternate Counter Low Register (ACLR) is the least significant byte (LS Byte).

These two read-only 16-bit registers contain the same value but with the difference that reading the ACLR register does not clear the TOF bit (Timer overflow flag), located in the Status register (SR). (See note at the end of paragraph titled 16-bit read sequence).

Writing in the CLR register or ACLR register resets the free running counter to the FFFCh value.

Both counters have a reset value of FFFCh (this is the only value which is reloaded in the 16-bit timer). The reset value of both counters is also FFFCh in One Pulse mode and PWM mode.

The timer clock depends on the clock control bits of the CR2 register, as illustrated in Table 1. The value in the counter register repeats every 131072, 262144 or 524288 CPU clock cycles depending on the CC[1:0] bits.

The timer frequency can be $f_{CPU}/2$, $f_{CPU}/4$, $f_{CPU}/8$ or an external frequency.

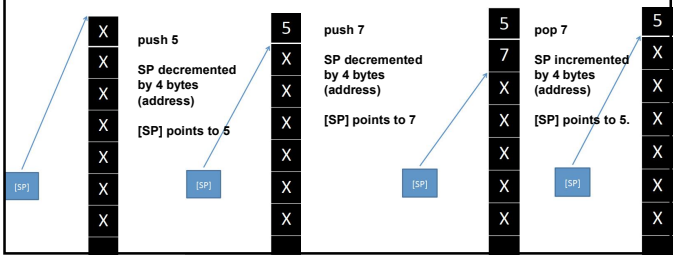
HOW TO USE THE TIMER?

```
BASE = $3F000000
TIMER_OFFSET = $3000
mov r3,BASE
orr r3,TIMER_OFFSET          ;store base address of timer (r3)

mov r4,$80000                ;store delay (r4)  0.524 second

ldrd r6,r7,[r3,#4]
mov r5,r6                    ;mov starttime (r5)(=currenttime (r6))
timerloop:
        ldrd r6,r7,[r3,#4]          ;read currenttime (r6)
        sub r8,r6,r5                ;remainingtime (8)= currenttime (r6) - starttime (r5)
        cmp r8,r4                   ;compare remainingtime (r8), delay (r4)
bls timerloop                ;loop if LE (reaminingtime <= delay)
```

STACK

PUSH, POP

[SP] points to start of stack



STACK - ASM

Using keyword: push and pop

Example:

```
push{r2}
push{r5}
push{r6,r7}
pop{r2}
pop{r7}
```