실습 5

Minsoo Ryu

Operating Systems and Distributed Computing Lab.
Hanyang University

msryu@hanyang.ac.kr

목차

❒ 실습

- **Programming Ultrasonic Distance Sensor**
- **Programming Motor with SW PWM**
- **Programming Motor with HW PWM**
- **Programming LED via Memory Mapping**

실습

1. Programming Ultrasonic Sensor (1)

❒ 구현 목표

- 초음파 거리 센서를 이용하여 거리를 측정하고, 거리가 **10cm** 미만으로 측정되면 **LED**를 점등하고 그렇지 않으면 **LED**를 멸등하는 프로그램을 작성한다

❒ 구현 조건

- **WiringPi**가 제공하는 아래의 **5**개 함수를 사용한다

```
int wiringPiSetupGpio(void)
void pinMode (int pin, int mode)
void digitalWrite (int pin, int value)
void delayMicroseconds (unsigned int howLong)
int gettimeofday(struct timeval *tv, struct timezone *tz)
```

- **GPIO 23**과 **GPIO 24**를 초음파 센서에 사용한다 **(BCM numbering)**
  - Trig → GPIO 23, Echo → GPIO 24
  - Vcc → 5.0V

1. Programming Ultrasonic Sensor (2)

❒ 구현 조건 (계속)

- 초음파 센서를 이용하여 거리를 측정하여 리턴하는 아래 함수를 작성한다

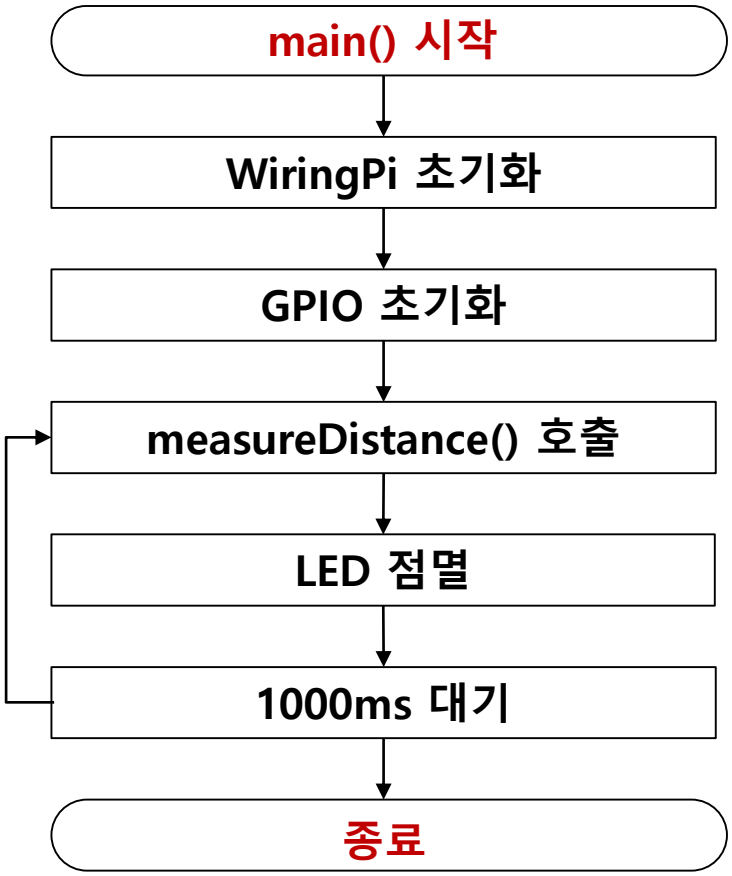
```
float measureDistance(void)
```

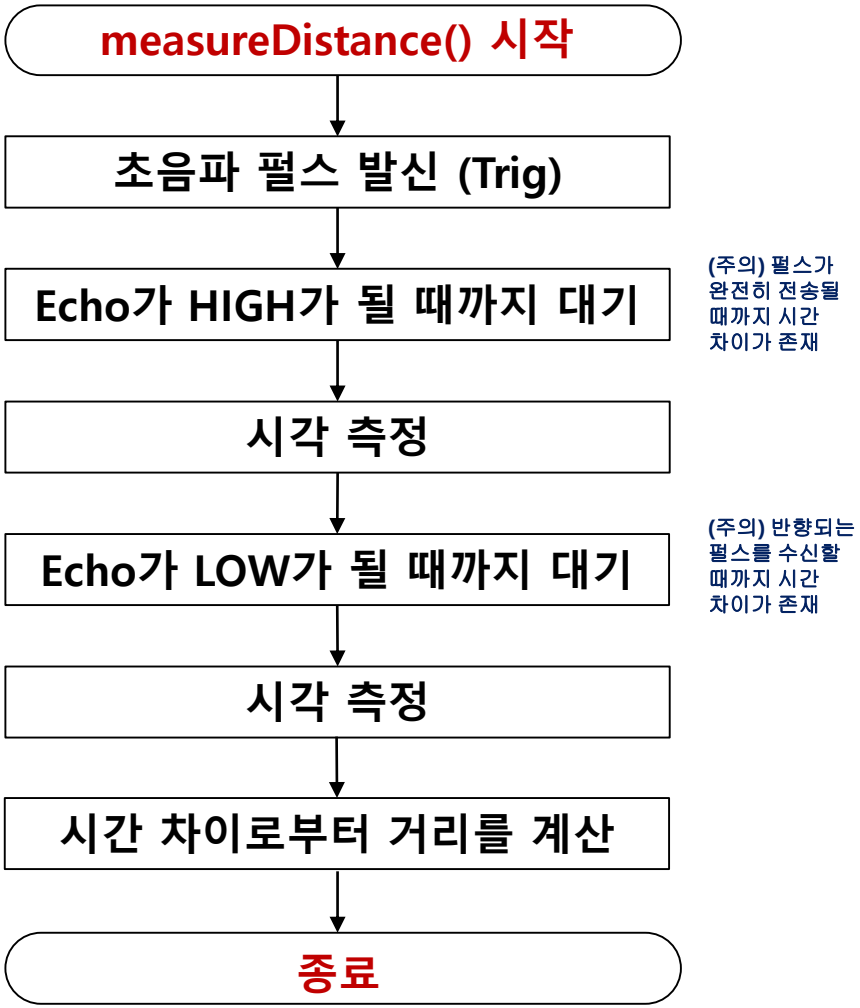
- 리턴값의 단위는 **cm**

❒ 제공 자료

- **measureDistance()** 함수가 삭제된 **"ultrasonic.src"** 소스 파일
- **(주의) LED**가 **GPIO 17**에 연결된 상태를 가정하여 **10cm** 이내 사물이 인식되면 **LED**를 점등하도록 작성되어 있음

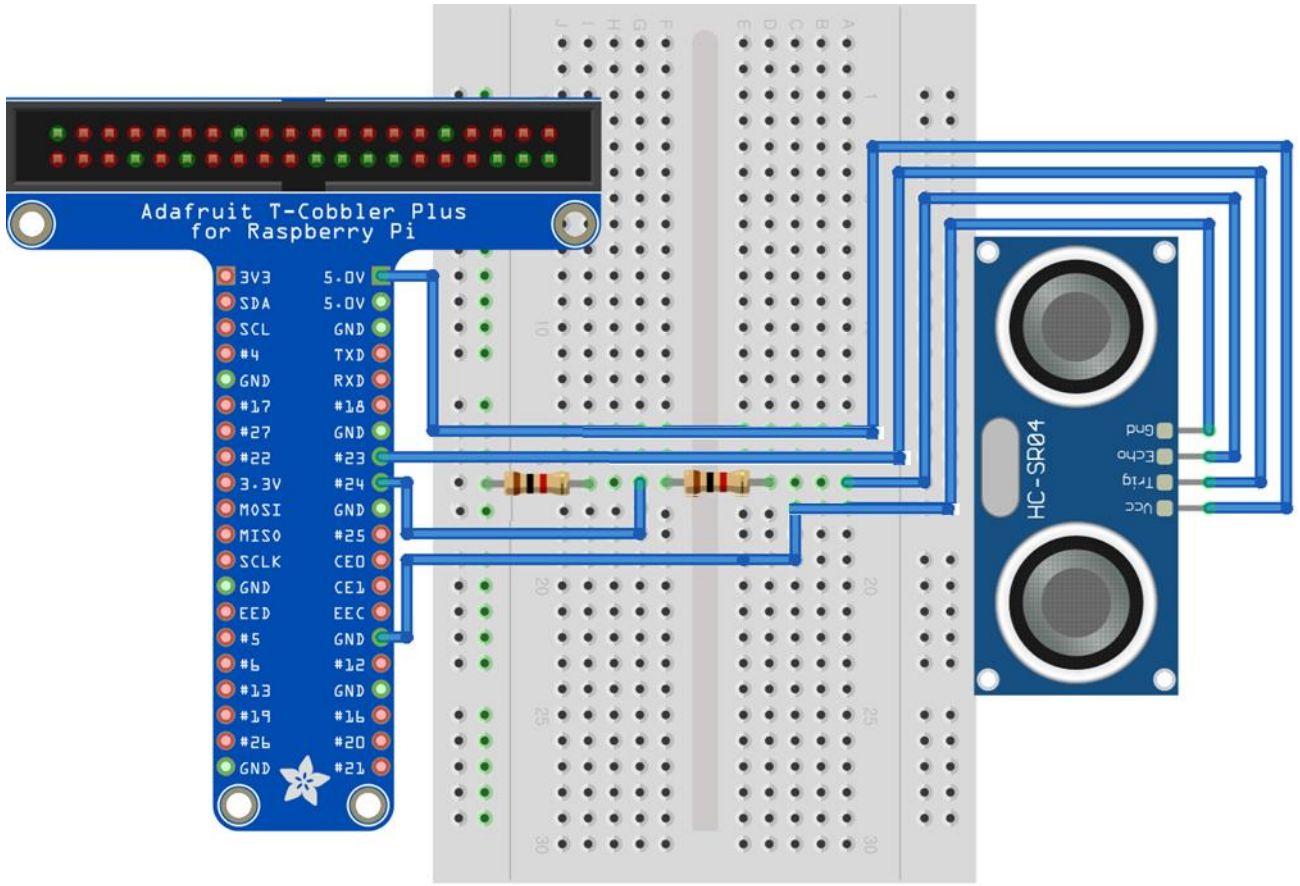
1. Programming Ultrasonic Sensor (3)





1. Programming Ultrasonic Sensor (4)

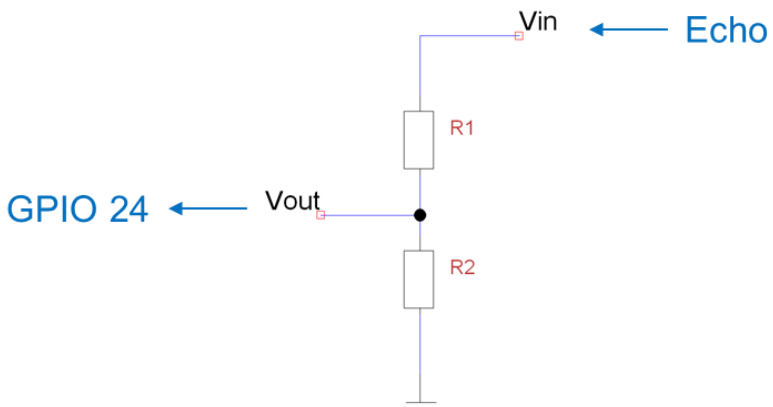
❒ 초음파 센서 연결



1. Programming Ultrasonic Sensor (5)

❒ 참고

- 사용될 초음파 센서 **HC-SR04**는 **5V**로 동작
- **Raspberry Pi**의 **GPIO**는 **3.3V**로 구동하므로 **Echo**가 연결될 **GPIO 24**에 대해 전압분배**(voltage division)** 필요
  - $\frac{3.3}{5} = \frac{V_{out}}{V_{in}} = \frac{R2}{R1+R2}$
  - $R1 = 1000$이라면 $R2 = 1941$
- 앞의 연결도에서는 $R1 = 1000$, $R2 = 1000$로 사용하였으나 동작에는 지장이 없음



1. Programming Ultrasonic Sensor (6)

❒ 참고: 시간 측정을 위해 아래 함수를 사용

- **int gettimeofday(struct timeval *tv, struct timezone *tz)**
- **1970-01-01 00:00:00**을 기준으로 현재까지 경과된 **second** 개수와 **microsecond** 개수를 제공

```
struct timeval {
    time_t      tv_sec;          /* seconds */
    suseconds_t tv_usec;    /* microseconds */
};
```

```
struct timezone {
    int tz_minuteswest; /* minutes west of Greenwich */
    int tz_dsttime;         /* type of DST correction */
};
```

- 실습에서는 **“timeval”**의 **“tv_sec”**와 **“tv_usec”**을 활용하여 아래와 같이 현재 시각을 **microsecond** 단위로 측정할 수 있음

```
struct timeval tv1;
long time1;

gettimeofday(&tv1, NULL);
time1 = tv1.tv_sec * 1000000 + tv1.tv_usec;
```

2. Programming Motor with SW PWM (1)

❒ 구현 목표

- **Software PWM**을 이용하여 사용자의 입력에 따라 서보 모터의 동작을 제어하는 프로그램을 구현한다 **(동작 구간: 0° ~ 180°)**

❒ 구현 조건

- 아래의 함수를 사용한다

```
int wiringPiSetupGpio(void)
void pinMode (int pin, int mode)
int softPwmCreate (int pin, int initialValue, int pwmRange)
void softPwmWrite (int pin, int value)
```

- **SG90** 모터의 **PWM** 입력에 **GPIO 18**을 사용한다 **(BCM numbering)**

2. Programming Motor with SW PWM (2)

❒ 구현 조건 (계속)

- 화면으로 모터의 각도 위치(**0° ~ 180°**)를 입력받아 해당 위치로 모터를 회전시키는 아래의 함수를 구현한다

void moveMotor(int degree)

❒ 제공 자료

- **“moveMotor()”** 구현이 삭제된 **servomotor_sw.src** 소스 파일

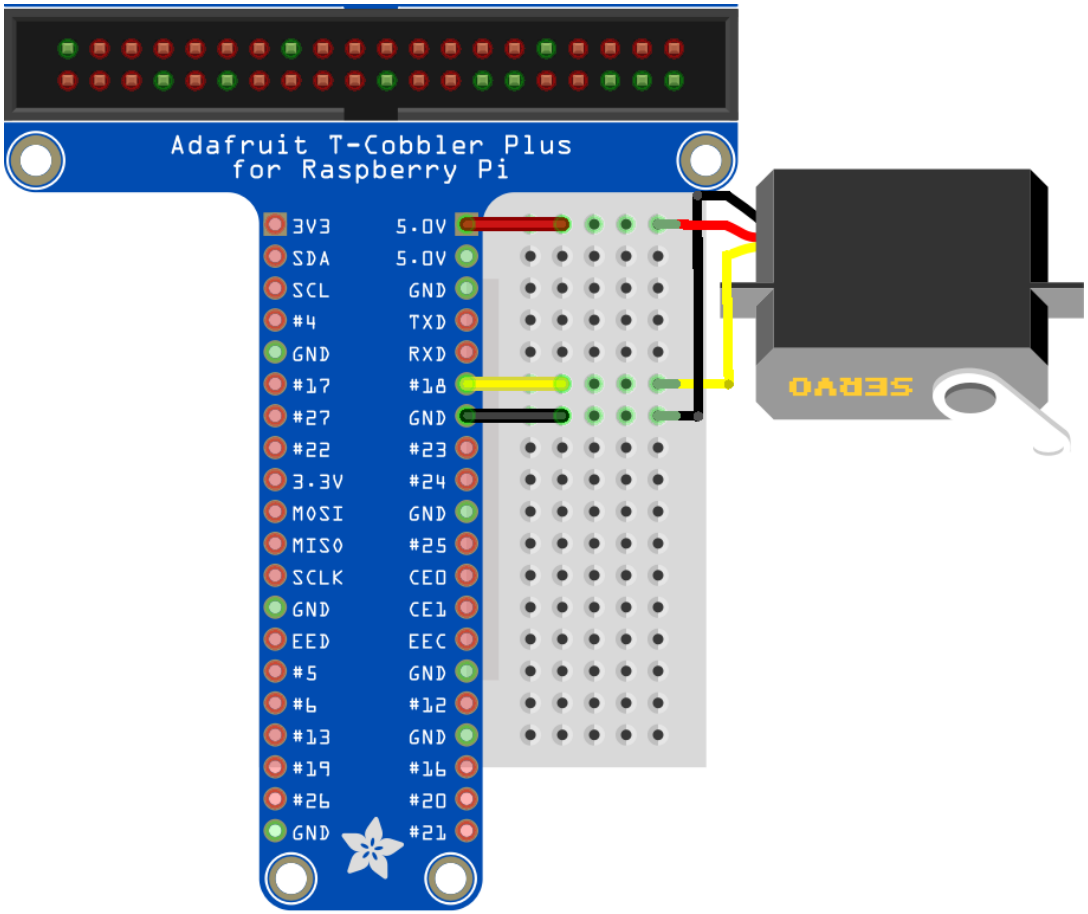
❒ 화면 출력 예시

```
msryu@rpi:~/Lab-RPi/52_Motor_SW_PWM $ ./executable
Enter the position of motor (in degree): 90
dutyCycle = 7.5%
pulseWidth = 15

Enter the position of motor (in degree): 180
dutyCycle = 10.0%
pulseWidth = 20
```

2. Programming Motor with SW PWM (3)

❒ 서보 모터 연결



2. Programming Motor with SW PWM (4)

❒ 참고

- **SG90 specification**
  - Duty Cycle → Degree: (5% ~ 10%) → (0° ~ 180°)
- **Pulse Width = Duty Cycle * (PWM Range/100)**
  - PWM Range = # of clock ticks in a period

3. Programming Motor with HW PWM (1)

- 구현 목표
  - **Hardware PWM**을 이용하여 사용자의 입력에 따라 서보 모터의 동작을 제어하는 프로그램을 구현한다 **(**동작 구간**: 0° ~ 180°)**

- 구현 조건
  - 아래의 함수를 사용한다

```
int wiringPiSetupGpio(void)
void pinMode (int pin, int mode)
void pwmSetMode (int mode)
void pwmSetClock (int divisor)
void pwmSetRange (unsigned int range)
void pwmWrite (int pin, int value)
```

  - 이전과 동일하게 **SG90** 모터의 **PWM** 입력에 **GPIO 18**을 사용한다 **(BCM numbering)**
    - (참고) GPIO 18은 alternate function으로 PWM 기능을 제공

3. Programming Motor with HW PWM (2)

❒ 구현 조건 (계속)

- 화면으로 모터의 각도 위치(**0° ~ 180°**)를 입력받아 해당 위치로 모터를 회전시키는 아래의 함수를 구현한다

void moveMotor(int degree)

- **PWM** 초기화 부분을 작성한다

❒ 제공 자료

- **"moveMotor()"** 구현 및 **PWM** 초기화 부분이 삭제된 **servomotor_hw.src** 소스 파일

❒ 화면 출력 예시

```
Enter the position of motor (in degree): 90
dutyCycle = 7.5%
pulseWidth = 15

Enter the position of motor (in degree): 180
dutyCycle = 10.0%
pulseWidth = 20
```

3. Programming Motor with HW PWM (3)

❒ 참고

- **void pinMode (int pin, int mode);**
  - GPIO 핀이 alternate function으로 PWM 기능을 제공한다면 “mode”는 “PWM_OUTPUT”으로 설정
- **void pwmSetMode (int mode)**
  - “mode”는 PWM_MODE_BAL 또는 PWM_MODE_MS
- **void pwmSetClock (int divisor)**
  - “divisor”는 PWM clock 값을 지정하며, 분주값(divisor)을 의미
- **void pwmSetRange (unsigned int range)**
  - “range”는 PWM range 레지스터 값을 지정하며, “range” 값의 의미는 # of clock ticks (in a period)
- **void pwmWrite (int pin, int value)**
  - “int”는 GPIO 핀 번호 (= 18)
  - “value”는 0 ~ 1024의 값으로 PWM의 pulse width를 지정 (# of ‘on’s)
  - (주의) HW PWM을 사용할 때는 softPwmWrite() 함수 대신 pwmWrite() 함수를 사용해야 됨

4. Programming LED via Mem. Map. (1)

- 구현 목표
  - 메모리 매핑을 이용하여 **LED**를 **1**초 간격으로 점등과 멸등을 번갈아 실행하는 프로그램을 구현한다

- 구현 조건
  - **WiringPi**를 사용하지 않고 아래 함수만을 사용한다

```
int open(const char *pathname, int flags)
void *mmap (void *addr, size_t length, int prot, int flags, int fd, off_t offset)
unsigned int sleep(unsigned int seconds)
```

  - **LED**를 **GPIO 17**을 통해 제어한다

- 제공 자료
  - **GPIO**를 제어하는 부분만 생략된 소스 코드 **“led_mmap.src”**
  - 제공된 코드에서 **while**문 내부의 공백을 완성하면 됨
    - 제공된 코드의 코멘트 설명을 참고할 것

4. Programming LED via Mem. Map. (2)

❒ 참고

- **int open(const char *pathname, int flags);**
  - #include <fcntl.h> 필요
  - 다음과 같이 호출하면 메모리에 매핑할 GPIO 메모리의 파일 디스크립터를 얻을 수 있음

```
int gpio_fd = open("/dev/gpiomem", O_RDWR);
```

- **void *mmap (void *addr, size_t length, int prot, int flags, int fd, off_t offset);**
  - #include <sys/mman.h> 필요
  - 다음과 같이 호출하면 메모리에 매핑할 GPIO 메모리의 파일 디스크립터를 얻을 수 있음

```
unsigned int *gpio = (unsigned int *)mmap(NULL, 4096,
      PROT_READ|PROT_WRITE, MAP_SHARED, gpio_fd, GPIO_BASE);
```

  - 리턴 값 “gpio”는 메모리에 매핑된 GPIO 메모리의 시작 주소

- **unsigned int sleep(unsigned int seconds);**
  - #include <unistd.h> 필요
  - 다음과 같이 호출하면 1초간 sleep

```
sleep(1);
```

4. Programming LED via Mem. Map. (3)

❒ 참고: BCM2835 GPIO register offsets

Address	Field Name	Description	Size	Read/ Write
0x 7E20 0000	GPFSEL0	GPIO Function Select 0	32	R/W
0x 7E20 0000	GPFSEL0	GPIO Function Select 0	32	R/W
0x 7E20 0004	GPFSEL1	GPIO Function Select 1	32	R/W
0x 7E20 0008	GPFSEL2	GPIO Function Select 2	32	R/W
0x 7E20 000C	GPFSEL3	GPIO Function Select 3	32	R/W
0x 7E20 0010	GPFSEL4	GPIO Function Select 4	32	R/W
0x 7E20 0014	GPFSEL5	GPIO Function Select 5	32	R/W
0x 7E20 0018	-	Reserved	-	-
0x 7E20 001C	GPSET0	GPIO Pin Output Set 0	32	W
0x 7E20 0020	GPSET1	GPIO Pin Output Set 1	32	W
0x 7E20 0024	-	Reserved	-	-
0x 7E20 0028	GPCLR0	GPIO Pin Output Clear 0	32	W
0x 7E20 002C	GPCLR1	GPIO Pin Output Clear 1	32	W
0x 7E20 0030	-	Reserved	-	-
0x 7E20 0034	GPLEV0	GPIO Pin Level 0	32	R
0x 7E20 0038	GPLEV1	GPIO Pin Level 1	32	R
0x 7E20 003C	-	Reserved	-	-
0x 7E20 0040	GPEDS0	GPIO Pin Event Detect Status 0	32	R/W
0x 7E20 0044	GPEDS1	GPIO Pin Event Detect Status 1	32	R/W
0x 7E20 0048	-	Reserved	-	-
0x 7E20 004C	GPREN0	GPIO Pin Rising Edge Detect Enable 0	32	R/W
0x 7E20 0050	GPREN1	GPIO Pin Rising Edge Detect Enable 1	32	R/W
0x 7E20 0054	-	Reserved	-	-
0x 7E20 0058	GPFEN0	GPIO Pin Falling Edge Detect Enable 0	32	R/W
0x 7E20 005C	GPFEN1	GPIO Pin Falling Edge Detect Enable 1	32	R/W

Address	Field Name	Description	Size	Read/ Write
0x 7E20 0060	-	Reserved	-	-
0x 7E20 0064	GPHEN0	GPIO Pin High Detect Enable 0	32	R/W
0x 7E20 0068	GPHEN1	GPIO Pin High Detect Enable 1	32	R/W
0x 7E20 006C	-	Reserved	-	-
0x 7E20 0070	GPLEN0	GPIO Pin Low Detect Enable 0	32	R/W
0x 7E20 0074	GPLEN1	GPIO Pin Low Detect Enable 1	32	R/W
0x 7E20 0078	-	Reserved	-	-
0x 7E20 007C	GPAREN0	GPIO Pin Async. Rising Edge Detect 0	32	R/W
0x 7E20 0080	GPAREN1	GPIO Pin Async. Rising Edge Detect 1	32	R/W
0x 7E20 0084	-	Reserved	-	-
0x 7E20 0088	GPAFEN0	GPIO Pin Async. Falling Edge Detect 0	32	R/W
0x 7E20 008C	GPAFEN1	GPIO Pin Async. Falling Edge Detect 1	32	R/W
0x 7E20 0090	-	Reserved	-	-
0x 7E20 0094	GPPUD	GPIO Pin Pull-up/down Enable	32	R/W
0x 7E20 0098	GPPUDCLK0	GPIO Pin Pull-up/down Enable Clock 0	32	R/W
0x 7E20 009C	GPPUDCLK1	GPIO Pin Pull-up/down Enable Clock 1	32	R/W
0x 7E20 00A0	-	Reserved	-	-
0x 7E20 00B0	-	Test	4	R/W

4. Programming LED via Mem. Map. (4)

❒ 참고: GPSET0 ~ GPSET1 fields

GPSET0

Bit(s)	Field Name	Description	Type	Reset
31-0	SETn (n=0..31)	0 = No effect 1 = Set GPIO pin *n*	R/W	0

GPSET1

Bit(s)	Field Name	Description	Type	Reset
31-22	-	Reserved	R	0
21-0	SETn (n=32..53)	0 = No effect 1 = Set GPIO pin *n*.	R/W	0

❒ 참고: GPCLR0 ~ GPCLR1 fields

GPCLR0

Bit(s)	Field Name	Description	Type	Reset
31-0	CLRn (n=0..31)	0 = No effect 1 = Clear GPIO pin *n*	R/W	0

GPCLR1

Bit(s)	Field Name	Description	Type	Reset
31-22	-	Reserved	R	0
21-0	CLRn (n=32..53)	0 = No effect 1 = Set GPIO pin *n*	R/W	0

4. Programming LED via Mem. Map. (5)

❒ 참고

- **BCM 2835 매뉴얼 기준 GPIO 주소 매핑**
  - Raspberry Pi에서 GPIO base의 bus address는 0x7E200000
  - ARM CPU에서 바라보는 GPIO base의 address는 0xF2200000
- **mmap() 함수는 GPIO base의 address를 user space의 특정 위치로 매핑**
  - mmap() 함수의 리턴값을 출력해보면 0x7FA67CC000 등의 값을 확인할 수 있음 (64-bit CPU에서는 40-bit address space를 사용하는 것도 함께 확인할 수 있음)

thank you!