4. 시스템 구현

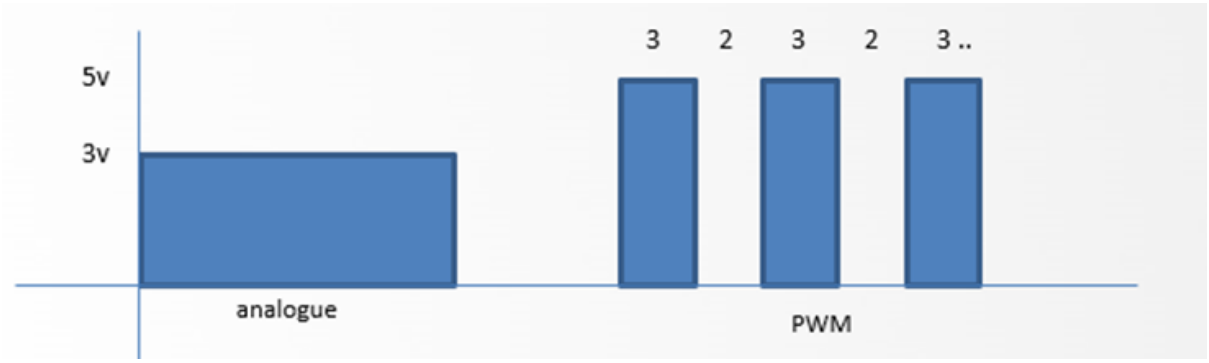
1)디바이스 드라이버

a) 버튼 드라이버

버튼 드라이버는 실습 시간에 진행했던 것과 거의 유사한 코드를 사용하였습니다. 차이점이 있다면 핀 번호가 바뀐 정도입니다. GPSEL을 Input 모드로 설정하기위해서 해당하는 비트를 '000'으로 고쳐 주어야 하고, GPLEV을 통해서 값을 읽습니다. 실습 시간에 자세히 강의하셨던 내용이기 때문에 큰 설명 없이 넘어가도록 하겠습니다.

b) Servo Motor 드라이버

servo Motor는 PWM의 output의 밀도에 따라 모터의 각도를 조절합니다. 이를 창문에 결합시킨다면 창문의 개폐를 할 수 있습니다.



위의 그림은 pwm의 아웃풋이 어떻게 진행되는지를 보여주는 그림입니다. 왼쪽과 오른쪽의 전압의 양은 같습니다. 하지만 출력방식이 다르게 진행됩니다. 아날로그를 균일하게 3V로 출력하는 것

과 PWM방식의 5V를 간헐적으로 출력하는 것입니다.

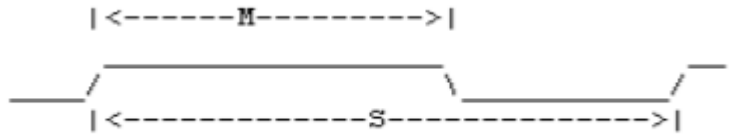
PWM은 위의 사진처럼 전압을 보냈다 안보냈다를 반복합니다. PWM의 전압을 보내는 방식에는 여러 방식이 있습니다. 대표적으로 N/M방식과 M/S방식입니다.

Bad	0	0	0	0	1	1	1	1	0	0	0	0
Fair	0	0	1	1	0	0	1	1	0	0	1	1
Good	0	1	0	1	0	1	0	1	0	1	0	1

위의 사진은 N/M방식을 보여주는 것입니다. duty cycle동안 serial channel을 따라서 전송됩니다. N/M모드는 M이라는 cycle동안 N개의 output 값이 1이 되어야 합니다. 위에 사진을 살펴보시면 8개가 전송될 때는 반드시 1이 4개가 되는 것을 확인할 수 있습니다. 위의 N/M은 4/8로 볼 수 있습니다. 특정 cycle 동안 output값의 1의 개수가 정해져 있으므로 위와 같이 출력의 경우의 수가 다양한 것을 확인할 수 있습니다.

```
1. Set context = 0
2. context = context + N
3.  if (context >= M)
        context = context – M
        send 1
    else
```

위의 그림은 N/M모드로 전송 시 1과 0의 출력을 교차로 보내기 위해 고안된 알고리즘입니다. 이 알고리즘을 사용하면 'Good' 상태로 N/M모드 output출력을 할 수 있습니다.



위의 사진은 PWM의 M/S모드 사용 그림입니다. M은 data값이며 S는 range입니다. M/S모드는 고주파 변조가 필요하지 않은 경우와 negative 효과가 있는 경우에 바람 직 합니다. 또한 데이터 레지스터가 사용되거나 버퍼가 사용되고 비어 있지 않은 한 채널은 출력을 계속 보냅니다.

	PWM0	PWM1
GPIO 12	Alt Fun 0	-
GPIO 13	-	Alt Fun 0
GPIO 18	Alt Fun 5	-
GPIO 19	-	Alt Fun 5
GPIO 40	Alt Fun 0	-
GPIO 41	-	Alt Fun 0
GPIO 45	-	Alt Fun 0
GPIO 52	Alt Fun 1	-
GPIO 53	-	Alt Fun 1

PWM channel은 위의 그림과 같습니다. 저희는 GPIO 18과 GPIO 12를 둘 다 사용해 보았습니다.

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

```
1    #define GPIO_BASE_ADDR 0x3F200000
2    #define GPFSEL1 0X04
```

Bit(s)	Field Name	Description	Type	Reset
31-30	---	Reserved	R	0
29-27	FSEL19	FSEL19 - Function Select 19 000 = GPIO Pin 19 is an input 001 = GPIO Pin 19 is an output 100 = GPIO Pin 19 takes alternate function 0 101 = GPIO Pin 19 takes alternate function 1 110 = GPIO Pin 19 takes alternate function 2 111 = GPIO Pin 19 takes alternate function 3 011 = GPIO Pin 19 takes alternate function 4 010 = GPIO Pin 19 takes alternate function 5	R/W	0
26-24	FSEL18	FSEL18 - Function Select 18	R/W	0
23-21	FSEL17	FSEL17 - Function Select 17	R/W	0
20-18	FSEL16	FSEL16 - Function Select 16	R/W	0
17-15	FSEL15	FSEL15 - Function Select 15	R/W	0
14-12	FSEL14	FSEL14 - Function Select 14	R/W	0
11-9	FSEL13	FSEL13 - Function Select 13	R/W	0
8-6	FSEL12	FSEL12 - Function Select 12	R/W	0
5-3	FSEL11	FSEL11 - Function Select 11	R/W	0
2-0	FSEL10	FSEL10 - Function Select 10	R/W	0

Table 6-3 – GPIO Alternate function select register 1

GPIO 12와 GPIO 18은 GPIO Alternate function select register 1입니다. PWM 채널에 맞추어 GPIO 12는 ALT function select 12에서 alternate function 0인 100을 선택해야 합니다. GPIO 18은 ALT function select 18에서 alternate function 0인 010을 선택해야 합니다. 아래의 코드가 저희 디바이스 코드에서 작성된 부분입니다.

```
*gpsel1 &= ~(1<<1);    //GPIO 12
*gpsel1 |= (1<<8);
*gpsel1 &= ~(1<<1);


*gpsel1 &= ~(1<<1);    //GPIO 18
*gpsel1 |= (1<<25);
*gpsel1 &= ~(1<<1);
```

PWM Address Map			
Address Offset	**Register Name**	**Description**	**Size**
0x0	CTL	PWM Control	32
0x4	STA	PWM Status	32
0x8	DMAC	PWM DMA Configuration	32
0x10	RNG1	PWM Channel 1 Range	32
0x14	DAT1	PWM Channel 1 Data	32
0x18	FIF1	PWM FIFO Input	32
0x20	RNG2	PWM Channel 2 Range	32
0x24	DAT2	PWM Channel 2 Data	32

위의 그림은 PWM의 레지스터입니다. 위의 그림에서 저희가 사용하는 부분은 CTL, RNG1, DAT1 입니다. CTL은 PWM Control과 관련이 있습니다. PWM의 환경을 설정하는 레지스터입니다. DAT1은 PWM channel 1 data입니다. 저희가 channel1만을 사용했기 때문에 DAT1을 사용했습니다. RNG1은 PWM channel 1 range입니다.

7	MSEN1	Channel 1 M/S Enable 0: PWM algorithm is used 1: M/S transmission is used.	RW	0x0
6	CLRF1	Clear Fifo 1: Clears FIFO 0: Has no effect This is a single shot operation. This bit always reads 0	RO	0x0
5	USEF1	Channel 1 Use Fifo 0: Data register is transmitted 1: Fifo is used for transmission	RW	0x0
4	POLA1	Channel 1 Polarity 0 : 0=low 1=high 1: 1=low 0=high	RW	0x0
3	SBIT1	Channel 1 Silence Bit Defines the state of the output when no transmission takes place	RW	0x0
2	RPTL1	Channel 1 Repeat Last Data 0: Transmission interrupts when FIFO is empty 1: Last data in FIFO is transmitted repetedly until FIFO is not empty	RW	0x0
1	MODE1	Channel 1 Mode 0: PWM mode 1: Serialiser mode	RW	0x0
0	PWEN1	Channel 1 Enable 0: Channel is disabled 1: Channel is enabled	RW	0x0

위의 사진은 PWM CTL 레지스터와 관련된 그림입니다. 왼쪽에 보이는 숫자는 CTL 레지스터의 bit 위치입니다. 위 bit의 값을 변경하여 원하는 PWM CTL 모드를 설정합니다. 저희는 위의 그림에서 PWEN1, MODE1, MSEN1을 다루었습니다.

PWEN1는 일치하는 채널을 enable 혹은 disable합니다. 이를 통해서 원하는 채널은 끄고 켤 수 있습니다. bit를 1로 세팅하면 channel과 transmitter state machine을 활성화시킵니다.

MODE1은 작동의 방식을 결정하는 bit입니다. 0으로 세팅하는 것은 PWM mode를 뜻합니다. 1은 Serial 모드를 선택하는 것입니다. 저희는 PWM 모드를 사용하기 때문에 위 bit를 0으로 설정하였습니다.

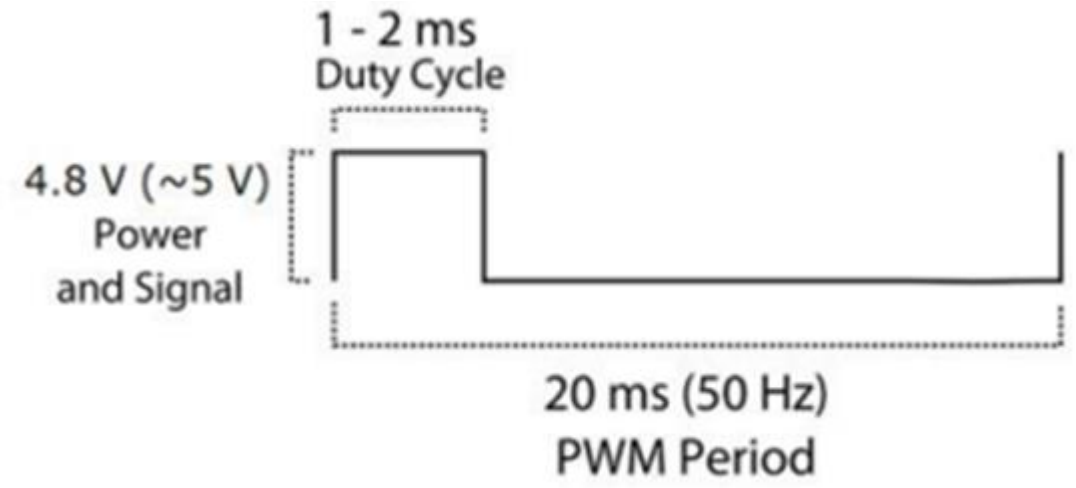
MSEN1 M/S모드와 N/M모드를 결정하는데 사용됩니다. 저희는 M/S모드를 사용하였으므로 위 bit를 1로 바꾸었습니다.

31:0	PWM_RNGi	Channel i Range	RW	0x20

31:0	PWM_DATi	Channel i Data	RW	0x0

다음은 RNG와 DAT값을 변경해야 했습니다. 이를 변경하는 이유는 RNG와 DAT의 비율을 통해서 PWM의 모터의 각도가 정해지기 때문입니다. 실습시간에 주어진 코드를 바탕으로 진행한다면 RNG는 320으로 정해져 있습니다. 아래의 그림을 보시면 PWM period를 RNG로 Duty Cycle을 DAT로 해석하면 됩니다. 즉 data는 16~32 사이의 값을 집어넣으면 됩니다. 16은 -90도이고 32는 +90도입니다.

Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is middle,
is all the way to the right, "-90" (~1ms pulse) is all the way to the left.




```
*gpsel1 &= ~(1<<1);
*gpsel1 |= (1<<25);
*gpsel1 &= ~(1<<1);
```

```
*pwmctl |= (1);     //PWEN      1
*pwmctl &= ~(1<<1);   //MODE1     0
*pwmctl |= (1<<7);   //MSEN1     MS  1

*pwmrng1 = 320;     //RANGE     320
*pwmdat1|= (1<<5);   //DAT     32   (1/10)
```

위의 그림에서 보시면 PWM period는 20ms(50Hz)로 되어있는 것을 확인할 수 있습니다. 위의 그림과 같은 형태로 진행하기 위해선 clk_pwm에 의해서 정의되어 있는 100MHz를 조절해야 합니

다. 이는 clock manager를 통해 진행됩니다. BCM 2837을 보시면 CPRMAN이라는 어휘를 볼 수 있는데 clk과 관련된 것입니다. 100MHz는 특정 div을 통해서 줄여져 50hz가 되어야 합니다. 아래의 코드가 그 과정입니다.

```
int init_pwm(void) {
   int pwm_ctrl = *pwmctl;
   *pwmctl = 0; // store PWM control and stop PWM
   msleep(10);//
   *clkctl = BCM_PASSWORD | (0x01 << 5); // stop PWM Clock
   msleep(10);//

   int idiv = (int)(19200000.0f / 16000.0f); // Oscilloscope to 16kHz
   *clkdiv = BCM_PASSWORD | (idiv << 12); // integer part of divisior register
   *clkctl = BCM_PASSWORD | (0x11); //set source to oscilloscope & enable PWM CLK

   *pwmctl = pwm_ctrl; // restore PWM control and enable PWM
}
```