- **날　　짜** : 2013년 12월 26일
- **제　　목** : JukeBlox SPI 통신 기초
- **문서번호** : KR_ES_0180
- **작 성 자** : 이현명 ( nathan.lee@microchip.com , FAE )

## < Engineering Issue 내용 >

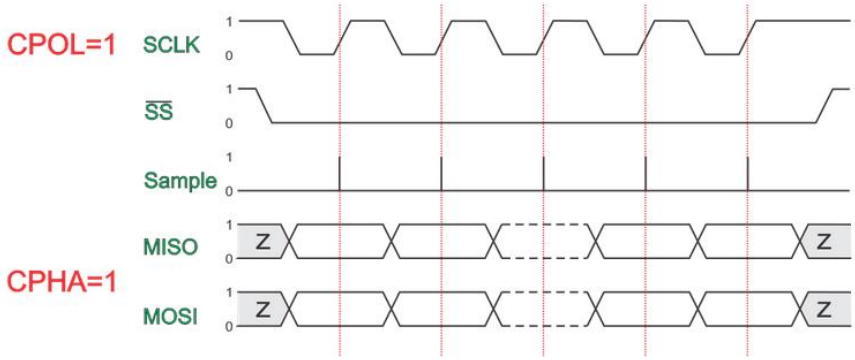
**1. 관련 Devices :** All

**2. 개요**

- eDMP 모드의 JukeBlox 를 외부 MCU 가 SPI 통신으로 제어할때 필요한 사항에 대해서 기술한다.

**3. JukeBlox SPI 사양**

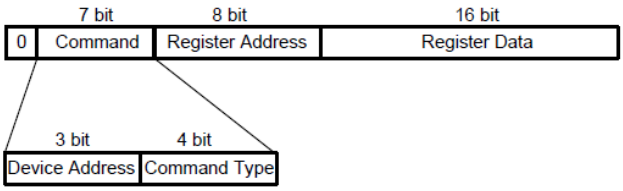
- JukeBlox 는 SPI Slave 로 동작하며 최대 2MHz (Typical 1MHz) 를 지원한다.
- JukeBlox 는 CPOL=1 (Low active clock), CPHA=1 (Sampling at rising edge) 을 지원한다.



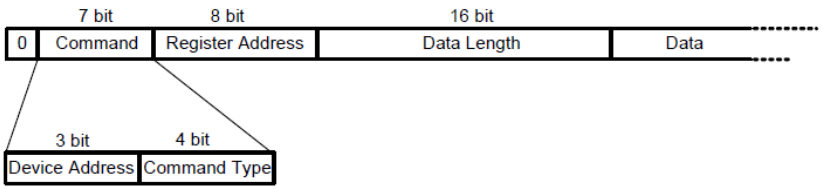

- SPI_REQ 출력 (High active) 라인으로 외부 MCU 에게 읽어갈 데이터가 있음을 알려주며, 읽어갈 메시지가 남아있는 경우 SPI_REQ 라인은 계속 High 로 유지된다.

**3.1.** SPI 데이터 포멧

- 고정 길이 메시지 (4 Byte)




- 가변 길이 메시지



- Command Type (4bit) 의 정의는 다음과 같다

| Value | Command Type | Description |
|---|---|---|
| 0 | Write Register | 고정길이 (4Byte) 쓰기 명령 |
| 1 | Read Register | 고정길이 (4Byte) 읽기 명령 |
| 2 | Multiple Write Register | 가변길이 쓰기 명령 |
| 3 | Multiple Read Register | 가변길이 읽기 명령 |
| 4 | Read All Registers | 모든 레지스터 읽기 명령 |
| 5-13 | Reserved | |
| 14 | XON | N/A (UART 데이터 포멧용) |
| 15 | XOFF | N/A (UART 데이터 포멧용) |

### 3.2. SPI 통신 플로우

SPI_REQ 가 HIGH 상태이면 외부 MCU 에서는 REG_00 읽기 동작을 수행 (SPI_WRITE 0x01, 0x00, 0x00, 0x00) 해야 되고 NOP (0xff) 가 아닌 데이터가 읽혀 질때까지 폴링을 수행한다 (SPI_WRITE 0xff). 이후에 들어오는 메시지는 고정 4Byte 메시지 또는 가변형 메시지이므로 다시 NOP (0xff) 가 읽혀지기 전까지 계속 폴링을 수행하면서(SPI_WRITE 0xff) 버퍼에 쌓아둔다. 버퍼에 있는 Register Data 영역을 파싱해서 어떤 레지스터를 다음에 읽어야 되는지 파악한다. 기본적으로 다음과 같은 모양이 된다.

```
#define SPI_NOP 0xff
#define SPI_REG_00 0x00

unsigned char spi_read_buffer[1024];
unsigned char MSG_READ_REG_00[] = {0x01, 0x00, 0x00, 0x00}

if(SPI_REQ == HIGH){

  switch(read_state_machine){

    case SEND_READ_REG_00:
      spi_write(&MSG_READ_REG_00[0], sizeof(MSG_READ_REG_00));
      spi_read_idx = 0;
      read_state_machine = POLL_READ_NOP;
      break;

    case POLL_READ_NOP:
      rdata = spi_read(SPI_NOP);
      if(rdata == SPI_NOP){/*No valid data yet, put delay and timer*/}
```

```
            else{
                switch(rdata&0x7F){
                    case 0x01:
                    case 0x03:
                        spi_read_buffer[spi_read_idx++] = rdata;
                        read_state_machine = RECV_READ_REG_00;
                        break;
                    default: break; //Do something else
                }
            }
            break;

        case RECV_READ_REG_00:
            spi_read_buffer[spi_read_idx] = spi_read(SPI_NOP);
            if(spi_read_buffer[spi_read_idx] == SPI_NOP){
                //end of one REG0 read message
                spi_reg_00_parser(&spi_read_buffer[0], spi_read_idx);
                read_state = SEND_READ_REG_00;
                break;
            }
            if(spi_read_idx < MAX_SPI_READ_BUF){spi_read_idx++;}
            break;
    }
}
```