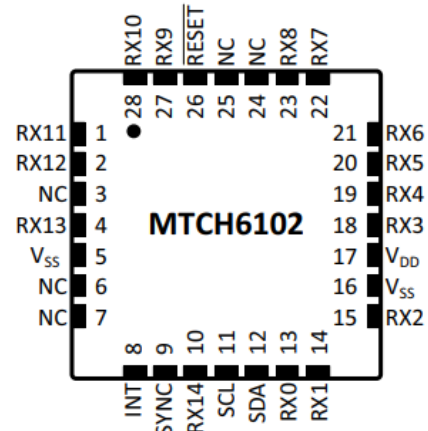
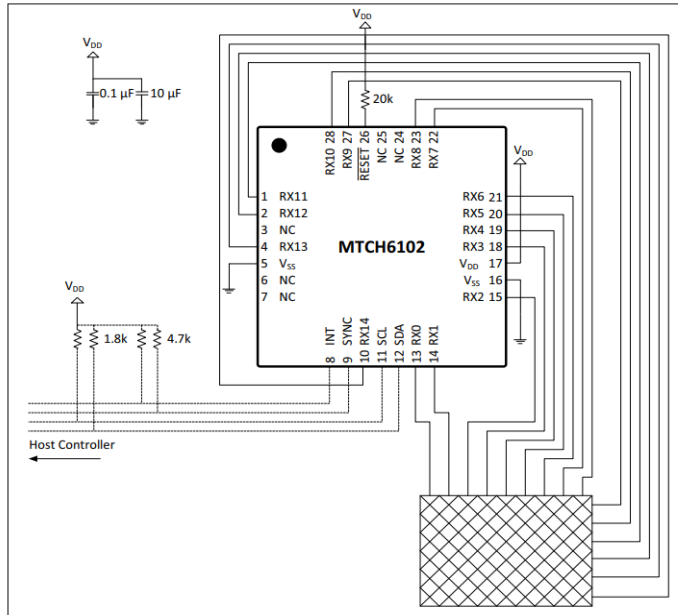


1. H/W LAYOUT

FIGURE 4-1: TYPICAL APPLICATION CIRCUIT

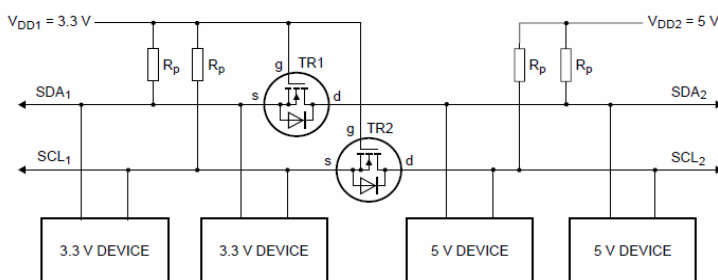


- Capacitive 방식
- Active시 12uA 이하 전류 소모
- Decoupling cap 104 Ceramic 사용
- Bulk cap 4.7 ~ 47uF 전해 사용

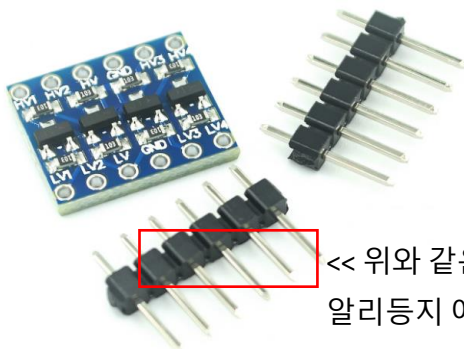
- Sync 와 INT신호 중요 외부 인터럽트 연결

2. Voltage Level

- 사용시 Datasheet를 확인하여 Host(MCU) Slave(Sensor) 전압 레벨을 맞춰줘야 한다.
- 모든 I2C 통신핀은 O.C혹은 O.D로 구성 돼 있다. 따라서 PULL UP저항을 사용하여야 한다.



이때 MTCH6102 : 1.8V ~ 3.5V
사용이 가능하기때문에 MCU가
5V 인 경우 좌측과 같은 레벨 시프트
회로를 사용하여야 한다.

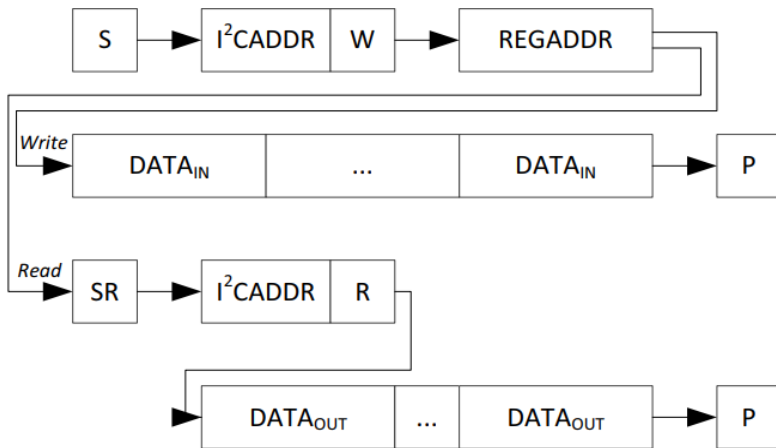


<< 위와 같은 회로 구성 된 모듈을
알리등지에서 쉽게 구매 할 수 있다.

| V_{IH} | $0.7 V_{CC}$ | $V_{CC} + 0.5$ | V |
|----------|--------------|----------------|---|
|----------|--------------|----------------|---|

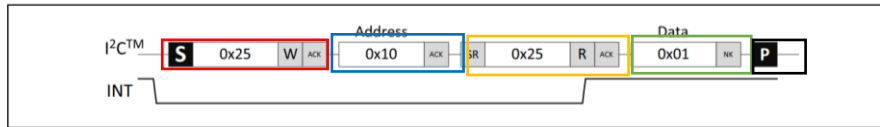
MCU 전압레벨 5V 사용시 $5 * 0.7 = 3.5V$
즉 3.3V를 HIGH신호로 인식 못할 수 있음

3. I2C 통신



Read를 할때에도 먼저 Write를 한
이후 Read해야한다. 주의 해야함

FIGURE 5-3: EXAMPLE I²C™ READ TRANSACTION



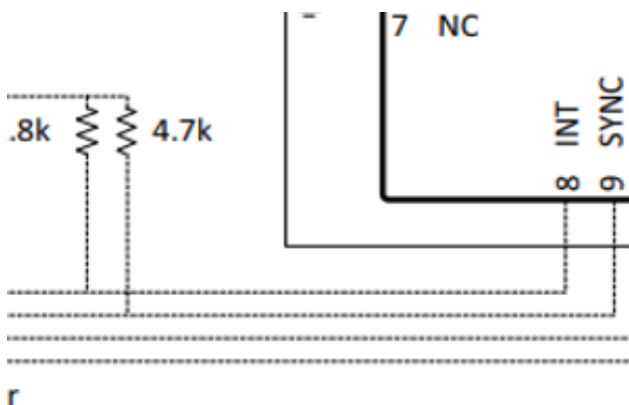
```
uint8_t mtch6102_read_8bit(uint8_t reg_address)
{
    uint8_t reg_status;

    i2c_start(((uint8_t)TOUCHPAD_DEVICE_SLAVE_ADDR<<1)|I2C_WRITE);
    i2c_write(reg_address); // TOUCHPAD_REG_TOUCH_STATE
    i2c_rep_start(((uint8_t)TOUCHPAD_DEVICE_SLAVE_ADDR<<1)|I2C_READ);

    reg_status = i2c_readNak();
    i2c_stop();

    return reg_status;
}
```

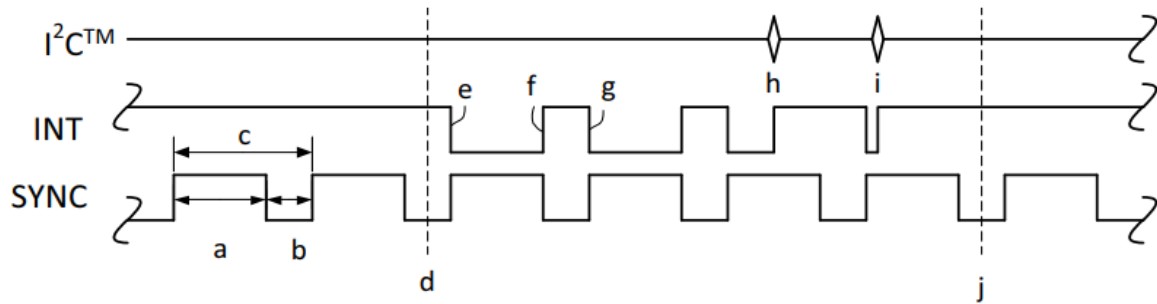
4. INT핀과 SYNC핀



INT : Active Low
SYNC : Active High

INT핀은 일반적인 인터럽트 요청핀으로
새로운 touch가 감지되면 LOW 상태가 된다.

SYNC핀은 MTCH6102가 DECODING과 SLEEP
상태를 오가는데 이를 알려준다.



- 1) MTCH6102 는 ACTIVE(DECODING)과 SLEEP상태를 왔다갔다 한다.
- 2) 위에서 보면 SYNC가 HIGH일때가 ACTIVE상태이며 LOW일때 SLEEP상태이다.
- 3) d시점에서 터치가 감지되며 INT또한 SYNC에 따라서 SLEEP과 ACTIVE상태를 왔다 갔다 하게 된다.

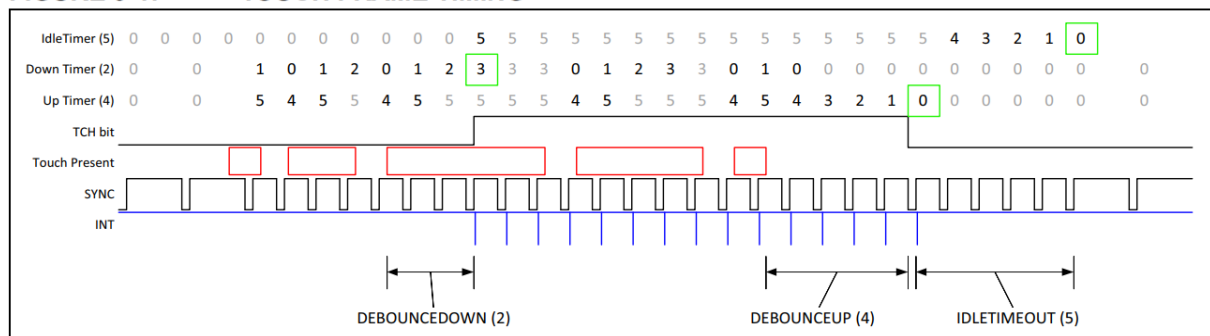
참고! 아래 FIGURE 5-4 에 기재 돼 있는 정보는 오기입 된것으로 예상됨. (a,b 바꿈)

| | |
|---|---------------------|
| a | Controller sleeping |
| b | Controller decoding |

MTCH6102 provides an active-high sync signal that correlates with the current touch frame status. The SYNC pin is low while the device is sleeping (between frames) and high while touch sensing/decoding is occurring. A common use of this pin includes a host that makes use of data on every frame (such as raw-acquisition data), for host-side decoding (see Figure 5-4).

5. Touch Framing

FIGURE 9-1: TOUCH FRAME TIMING



터치 입력 또한 스위치처럼 채터링이 발생한다.

위에서 보듯이 **터치가 발생했다!** 라고 판명하기까지 DEBOUNCE DOWN 이 결정 지으며 **터치가 없어졌다!** 라고 판명하기 까지 DEBOUNCE UP이 결정 짓는다. 이는 해당 레지스터에 값을 써져 조절 가능하다.

6. 핵심 레지스터

REGISTER 10-1: TOUCHSTATE: CURRENT TOUCH STATE REGISTER

| | | | | | | | |
|------------|-------|-------|-------|-----|-------|-------|-------|
| R/W-0 | R/W-0 | R/W-0 | R/W-0 | U-x | R/W-0 | R/W-0 | R/W-0 |
| FRAME<3:0> | | | | — | LRG | GES | TCH |
| bit 7 | | | | | | | bit 0 |

Legend:

| | | | |
|------------------|----------------------|-----------------------|---|
| R = Readable bit | '1' = Bit is set | x = Bit is unknown | -n = Value after initialization (default) |
| W = Writable bit | '0' = Bit is cleared | U = Unimplemented bit | q = Conditional |

| | |
|---------|---|
| bit 7-4 | FRAME<3:0> : Increments on Every Touch Frame |
| bit 3 | Unimplemented : Read as '0' |
| bit 2 | LRG : Large Activation is Present |
| bit 1 | GES : Gesture is Present |
| bit 0 | TCH : Touch is Present |

터치 판별의 **핵심 레지스터!** 이 레지스터에서 touch 유무, 제스처(스вай프, 더블클릭 등) 판별 한다.

TABLE 10-1: SUMMARY OF REGISTERS ASSOCIATED WITH TOUCH DATA

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---------|------------|--------------|-------|-------|-------|-------------|-------|-------|-------|
| 0x10 | TOUCHSTATE | FRAME<3:0> | | | | — | LRG | GES | TCH |
| 0x11 | TOUCHX | TOUCHX<11:4> | | | | | | | |
| 0x12 | TOUCHY | TOUCHY<11:4> | | | | | | | |
| 0x13 | TOUCHLSB | TOUCHX<3:0> | | | | TOUCHY<3:0> | | | |

핵심 레지스터3! 이 레지스터에서 touch x좌표,touch y좌표를 파악한다.

이때 x좌표 y좌표는 세가지 레지스터로 쪼개져 있으니 이를 비트 시프트하여 취합 할 것!

```
mtch6102_pos cur_pos;
```

```
cur_pos.lsb = mtch6102_read_8bit(TOUCHPAD_REG_TOUCH_LSB);
_delay_us(10);
cur_pos.x_11_4 = mtch6102_read_8bit(TOUCHPAD_REG_TOUCH_X);
_delay_us(10);
cur_pos.y_11_4 = mtch6102_read_8bit(TOUCHPAD_REG_TOUCH_Y);
_delay_us(10);
cur_pos.xpos = ((cur_pos.x_11_4)<<4)|((cur_pos.lsb>>4)&0x0f);
cur_pos.ypos = ((cur_pos.y_11_4)<<4)|(cur_pos.lsb&0x0f);

printf("cur_pos.xpos : %d, cur_pos.ypos : %d\r",cur_pos.xpos,cur_pos.ypos);
```

1. 목 적

- RTU Modbus Protocol 포맷을 활용한 신뢰성 있는 데이터 송수신

2. 구 성

<키보드 입력>

| ADDRESS | Function | KEY_DATA | CRC1 |
|---------|----------|----------|-------|
| 0x01 | 0x01 | 6Byte | 2Byte |

<마우스 입력>

| ADDRESS | Function | MOUSE_DATA | CRC1 |
|---------|----------|------------|-------|
| 0x01 | 0x02 | 6Byte | 2Byte |

<파이프 어드레스 지정>

| ADDRESS | Function | PIPE_ADDRESS | DUMMY_DATA | CRC1 |
|---------|----------|--------------|------------|-------|
| 0x01 | 0x03 | 5Byte | 1Byte | 2Byte |

※CRC는 CHIP 자체적으로 검증하므로 생략함

<KEY_DATA>

| DATA1 | DATA2 | DATA3 | DATA4 | DATA5 | DATA6 |
|-------|-------|-------|-------|-------|-------|
| KEY1 | KEY2 | KEY3 | KEY4 | KEY5 | KEY6 |

<MOUSE_DATA>

| DATA1 | DATA2 | DATA3 | DATA4 | DATA5 | DATA6 |
|-------|-------|--------|-------|-------|-------|
| LEFT | RIGHT | DOUBLE | X POS | Y POS | wheel |

<PIPE_ADDRESS>

| DATA1 | DATA2 | DATA3 | DATA4 | DATA5 | DUMMY_DATA |
|--------|--------|--------|--------|--------|------------|
| RANDOM | RANDOM | RANDOM | RANDOM | RANDOM | DUMMY |








3. KEY_DATA, MOUSE_DATA 값

- Device Class Definition for HID 1.11 참조
- <https://www.usb.org/hid>

■ 일정 진행상황

2021-01-14 강경수

1. 전체 일정

| 년도 | 2020년 | | 2021년 | | | | | | | |
|----|--|---|-------|---|---|---|----|---|---|---|
| 월 | 12월 | | 1월 | | | | 2월 | | | |
| 주차 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 |
| 항목 |  H/W 사양 최종결정 및 PCB ARTWORK | | | | | | | | | |
| |  EVALUTAION BOARD상에서 F/W 작업 | | | | | | | | | |
| |  PCB 발주 및 수령 | | | | | | | | | |
| |  납땜 및 H/W 디버깅 | | | | | | | | | |
| | TARGET BOARD상에서 F/W 테스트  | | | | | | | | | |
| | H/W F/W 최종 검토  | | | | | | | | | |
| | 발표 준비  | | | | | | | | | |

2. H/W 진행 현황

- 최종설계 완료

3. F/W 진행 현황

- USB 통신작업 완료

- KEY MATRIX 완료

- NRF24L01 완료

- MTCH6102 완료

4. PCB작업

- 오늘 마무리