MCU_Peripheral_function

노트북:	MCU		
만든 날짜:	2018-07-13 오전 4:25	**수정한 날짜:**	2018-07-30 오전 11:35
작성자:	정상요		

MCU_Peripheral_모든 함수

1. GIO - 전압을 발생시키는 Peripheral

1. `void gioInit(void);` //반드시 사용 GIO 사용전 반드시 사용
2. `void gioSetDirection(gioPORT_t *port, uint32 dir);` //사용할 필요 없음. HCG에서 설정가능하므로 사용빈도 낮음. 해당 pin을 output 또는 input으로 설정.
   (i.e. gioSetDirection(gioPORTA, 0xfaU : gioPORTA의 1번, 3번, 5번, 6번, 7번, 8번 pin을 출력으로 사용)
3. **`void gioSetBit(gioPORT_t *port, uint32 bit, uint32 value);`** // gioSetPort 둘중 하나 선택적으로 반드시사용 해당 pin을 logic high로 set해줌.
   (i.e. gioSetBit(gioPORTA, 3, 1) : gioPORTA의 3번 pin logic high로 set)
4. **`void gioSetPort(gioPORT_t *port, uint32 value);`** // gioSetBit 둘 중 하나선택적으로 반드시사용 gioSetBit와 동일한 기능이지만 PORT단위로 set해줌.
   (i.e. gioSetPort(gioPORTA, 0xf7U) : gioPORTA의 0번, 1번, 2번 pin logic hish로 set)
5. `uint32 gioGetBit(gioPORT_t *port, uint32 bit);` //현재 logic high로 set되어있는 bit의 return값을 1
   (i.e. gioGetBit(gioPORTB, 3) : gioPORTB의 3번 pin이 logic high면 return 값이 1, logic low면 return 값이 0)
6. `uint32 gioGetPort(gioPORT_t *port);` //현재 켜져있는 bit의 retrun 값을 1, 위 함수와 차이는 위에것은 bit단위, 이 함수는 port단위
   (i.e. gioGetPort(gioPORTA) : gioPORTA에서 logic high로 set되어있는 bit만 return 값이 1이 나온다. 만약 3번, 5번 bit만 logic high로 set 되어있으면 return 값이 0x28 이 나온다.)
7. **`void gioToggleBit(gioPORT_t *port, uint32 bit);`** //현재 logic high로 set되어있는 bit를 logic low로 set시키고, 현재 logic low로 set 되어있는 bit를 logic high로 set시킴.
   (i.e. gioToggleBit(gioPORTA, 3) : 만약 gioPORTA의 3번 pin에 LED를 연결하면 일정한 주기로 깜빡깜빡 거린다.)
8. **`void gioEnableNotification(gioPORT_t *port, uint32 bit);`** // 원하는 pin이 interrupt가 가능하게 해줌.
   (i.e. gioEnableNotification(gioPORTA, 3) : gioPORTA의 3번 pin이 interrupt를 받을 수 있다.
9. **`void gioDisableNotification(gioPORT_t *port, uint32 bit);`** // 원하는 pin이 interrupt가 불가하게 해줌.
   (i.e. gioDisableNotification(gioPORTA, 3) : gioPORTA의 3번 pin이 interrup를 받을 수 없게 만든다.
10. **`void gioNotification(gioPORT_t *port, uint32 bit);`** // interrupt가 발생시 원하는 동작을 구현.
    (i.e void gioNotification(gioPORT_t *port, uint32 bit) : 그대로 함수를 입력하면 위에서 gioEnableNotification에서 작성한 인자가 들어가서 동작한다.)
11. **`void gioGetConfigValue(gio_config_reg_t *config_reg, config_value_type_t type);`** // Configuration register들의 initial value 또는 current value를 config_reg가 가리키는 구조체에 저장시킨다.

2. RTI - 일정한 주기로 Interrupt를 발생시키는 Peripheral(Interrupt 사용시 반드시 _enable_IRQ_interrupt_(); 작성)

1. **`void rtiInit(void);`** // 반드시 사용 RTI 사용전 반드시 사용
2. **`void rtiStartCounter(rtiBASE_t *rtiREG,uint32 counter);`** // 반드시 사용. HCG에서 설정해준 Period에 따라서 Counter block이 동작하게 해준다.
   (i.e. rtiStartCounter(rtiREG1, 0) : rtI module의 Counter0의 Counter 값을 증가시켜 HCG에서 설정해준 주기에 따라 interrupt를 발생시킨다.)
3. **`void rtiStopCounter(rtiBASE_t *rtiREG,uint32 counter);`** // rtiStartCounter에서 동작시켜준 Counter를 멈추게 해준다.
   (i.e. rtiStopCounter(rtiREG1, 0) : rti의 Counter0이 더 이상 동작되지 않는다.)
4. **`uint32 rtiResetCounter(rtiBASE_t *rtiREG,uint32 counter);`** // 선택한 rti module의 원하는 Counter Block을 Reset해준다.(이 함수 사용전 rtiStopCounter를 통하여 reset시킬 Counter block을 stop시켜놓아야한다.)
   (i.e. rtiResetCounter(rtiREG1, 0) : CNT값을 reset시킨다. rtiStopCuunter실행 전 CNT의 값이 Actual Period값에 의하여 계속 증가하고 있었다. rtiStopCounter로 CNT가 멈췄고 특정한 값을 가지고 있었을텐데 이를 초기화시켜준 것이다.)
5. **`void rtiSetPeriod(rtiBASE_t *rtiREG,uint32 compare, uint32 period);`** // 선택한 Compare의 Period를 변경시켜준다.

(i.e. rtiSetPeriod(rtiREG1, 0, 10000000) : Compare0의 period를 10,000,000으로 set해준다. 실제 주기값을 알고싶으면 `rtiREG1->CMP[0U].UDCPx = 9375U;` 값과 비례식을 세워본다.)

6. `uint32 rtiGetPeriod(rtiBASE_t *rtiREG,uint32 compare);` // 선택한 Compare의 Period 값을 return한다.
(i.e. rtiGetPeriod(rtiREG1, 0) : Compare0의 period값을 return한다.)
7. `uint32 rtiGetCurrentTick(rtiBASE_t *rtiREG,uint32 compare);` // 선택한 Compare의 현재 tick 값을 return 한다.
8. `void rtiEnableNotification(rtiBASE_t *rtiREG,uint32 notification);` // RTI module의 선택한 notification이 사용하게 만든다.
(i.e. rtiEnableNotification(rtiREG1, 1) : Compare0의 Interrupt를 활성화시켜준다. notification 1: compare0, notification 2: compare1, notification 4: compare 3, notification 8: compare 4)
9. `void rtiDisableNotification(rtiBASE_t *rtiREG,uint32 notification);` // RTI module의 선택한 notification이 사용불가하게 만든다.
(i.e. rtiDisableNotification(rtiREG1, 4) : Compare3의 Interrupt를 비활성화시켜준다.)
10. `void rtiNotification(rtiBASE_t *rtiREG,uint32 notification); //Interrupt를 받았을때 발생할 동작을 작성하면 된다.`
(i.e rtiNotification(rtiBASE_t *rtiREG, uint32 notification){gioToggleBit(gioPORTA, 3);} : Interrupt발생시마다 gioToggleBit가 동작된다.)

RTI 사용시 필요한 내용
RTI사용시 필요한 2가지 개념이 존재한다. CNT(Counter)와 CMP(Compare)의 관계를 알아야한다. Actual Period가 1번 지날때 CNT는 1씩 증가한다. 1씩 증가하다가 설정해놓은 CMP값과 같아질 경우에 Interrupt가 발생한다. CNT값과 CMP값이 같아질때만 Interrupt가 발생하게 설정해주는 것은 Notification에 1을 넣어 INTFLAG값을 0으로 설정해주는 것이다. 그렇지 않으면 CNT와 CMP값이 달라도 Interrup가 발생할 수도있다.
rtiResetCounter는 CNT의 값을 다시 0으로 만들어주는 역할을 한다.
rtiSetPeriod에서 Period값은 rtiInit에서 같은 register를 통하여 값을 비교하여 설정해주면 된다.
rtiEnableNotification에서 Compare0의 Interrupt값을 활성화시켜준다.
rtiStartCounter는 Counter값을 증가시켜주는 역할을 한다.

3. PWM

1. `void etpwmInit(void); //` 반드시 사용 PWM사용전 반드시 사용
2. `void etpwmStartTBCLK(void);` // gpio pin에 etPWM을 연결한다. (Pin Muxing을 가능하게 해주는 함수.) 모든 eTPWMx module(etPWM1 ~ etPWM7)들에 맞춰놓은 Time-Base clock을 동작시킨다.
3. `void etpwmStopTBCLK(void);` // gpio pin에 etPWM의 연결을 해제한다. 모든 etPWMx module(etPWM1 ~ etPWM7)들에 맞춰놓은 Time-Base clock을 멈춘다.
4. `void etpwmSetClkDiv(etpwmBASE_t *etpwm, etpwmClkDiv_t clkdiv, etpwmHspClkDiv_t hspclkdiv);` // TimeBase Clock 과 High Speed time base clock divider를 set 해준다. (TBCLK = VCLK3 / (HSPCLKDIV * CLKDIV))
5. `void etpwmSetTimebasePeriod(etpwmBASE_t *etpwm, uint16 period);` // timebase counter의 period를 set해준다.
(i.e. etpwmSetTimebasePeriod(etpwmREG1, 1249) : etpwmREG1의 CompareA와 CompareB의 주기를 1249로 맞춰준다. 실제 Period값을 알고싶으면 `etpwmREG1->TBPRD = 1249U;` 값과 비례식을 세워본다.)
6. `void etpwmSetCount(etpwmBASE_t *etpwm, uint16 count);` // timebase counter를 set해준다.
7. `void etpwmDisableTimebasePeriodShadowMode(etpwmBASE_t *etpwm);` //
8. `void etpwmEnableTimebasePeriodShadowMode(etpwmBASE_t *etpwm);` //
9. `void etpwmEnableCounterLoadOnSync(etpwmBASE_t *etpwm, uint16 phase, uint16 direction);` //
10. `void etpwmDisableCounterLoadOnSync(etpwmBASE_t *etpwm);` //
11. `void etpwmSetSyncOut(etpwmBASE_t *etpwm, etpwmSyncOut_t syncOutSrc);` //
12. `void etpwmSetCounterMode(etpwmBASE_t *etpwm, etpwmCounterMode_t countermode);` //
13. `void etpwmTriggerSWSync(etpwmBASE_t *etpwm);` //
14. `void etpwmSetRunMode(etpwmBASE_t *etpwm, etpwmRunMode_t runmode);` //
15. `void etpwmSetCmpA(etpwmBASE_t *etpwm, uint16 value);` // CompareA의 Duty를 정하는 함수.
(i.e. etpwmSetCmpA(etpwmREG1, t * 10) : etpwmREG1의 CompareA Duty를 t * 10으로 설정해준다. 실제 Duty값을 알고싶으면 `etpwmREG1->CMPA = 625U;` 값과 비례식을 세워본다.)
16. `void etpwmSetCmpB(etpwmBASE_t *etpwm, uint16 value);` // CompareB의 Duty를 정하는 함수.
(i.e. etpwmSetCmpB(etpwmREG2, t * 10) : etpwmREG2의 CompareB Duty를 t * 10으로 설정해준다. 실제 Duty값을 알고싶으면 `etpwmREG1->CMPB = 625U;` 값과 비례식을 세워본다.)
17. `void etpwmEnableInterrupt(etpwmBASE_t *etpwm, etpwmEventSrc_t eventsource, etpwmEventPeriod_t eventperiod); //`
18. `void etpwmDisableInterrupt(etpwmBASE_t *etpwm); //`

4. UART

1. `void sciInit(void);` // 반드시 사용 UART 사용전 반드시 사용
2. `void sciSetFunctional(sciBASE_t *sci, uint32 port);` // sciPIN을 UART로 사용할지 gio로 사용할지 정하는 함수. Parameter를 sciREG로 받은 경우, port = 6이면 transmit(3번째자리), recieve(2번째자리) 둘 다 사용. Parameter를 sciPORT로 받은 경우, port가 0이면 gioPIN으로 사용.
   (i.e. sciSetFunctional(sciREG1, 6) : sci 1 module의 송, 수신을 전부 사용하겠다. // sciSetFunctional(sciPORT1, 0) : sci1을 gio처럼 사용하겠다.)
3. `void sciSetBaudrate(sciBASE_t *sci, uint32 baud);` // runtime시에 SCI badurate을 변경시킨다. 연결시키는 Module에 맞춰서 baud값을 변경해준다. (참고. baudrate : 1초당 보내는 데이터수)
4. `uint32 sciIsTxReady(sciBASE_t *sci);` // Tx buffer ready flag 가 set되어있는지 확인한다. flag가 set되어있으면 return 값이 0이 되고, 그렇지 않으면 flag값 자체가 나온다.
   (i.e. sciIsTxReady(sciREG1) : 입력을 받으면 0이 return 된다.)
5. `void sciSendByte(sciBASE_t *sci, uint8 byte);` // Byte단위로 송신하는 함수. Polling mode에서 single byte를 보내고, byte를 보내기 전까지 Transmit buffer가 비어있을때까지 routine에서 기다린다. 대기를 피하려면 sciTxReady를 사용한다.
   (i.e. sciSendByte(sciREG1, msg) : sci1 module을 이용하여 msg를 보낸다.)
6. `void sciSend(sciBASE_t *sci, uint32 length, uint8 * data);` // data가 가리키는 data block을 length byte만큼 송신한다. Interrupt가 활성화된 경우 Interrupt mode로 data를 송신한다. 그렇지 않은 경우에는 Polling mode가 사용된다.
   (i.e. sciSend(sciREG1, 8, data) : data가 가리키는 data block(i.e 배열)의 8byte만큼을 송신한다.)
7. `uint32 sciIsRxReady(sciBASE_t *sci);` // Rx buffer full flag가 set되어있는지 확인한다. set되어있지 않으면 return 값이 0이 나온다. 그렇지 않으면 flag값 자체가 return 된다.
   (i.e. sciIsRxReady(sciREG1) : 입력을 받으면 0이 된다.)
8. `uint32 sciIsIdleDetected(sciBASE_t *sci);` // SCI Idle flag가 set 되어있는를 확인한다. flag가 set되어있지 않다면 return 값이 0이 나온다. 그렇지 않으면 Idle flag값 자체가 return 된다.
9. `uint32 sciRxError(sciBASE_t *sci);` // Rx framing, overrun, parity error flag를 return 한다. return하기전 error flag를 모두 clear해준다.
10. `uint32 sciReceiveByte(sciBASE_t *sci);` // Byte단위로 수신하는 함수. Polling mode에서 single byte를 받는다. receive buffer에 byte가 없으면 routine은 byte를 받을때 까지 기다린다. 대기를 피하려면 sciIsRxReady를 사용한다.
   (i.e. (uint8)sciReceiveByte(sciREG1) : 8bit(char형) 만을 읽어본다.)
11. `void sciReceive(sciBASE_t *sci, uint32 length, uint8 * data);` // length byte의 block을 받아서 data가 가리키는 data buffer에 저장한다. Interrupt가 활성화된 경우 Interrupt mode로 data를 수신한다. 그렇지 않은 경우에는 Polling mode가 사용된다.
   (i.e. sciReceive(sciREG1, 8, data) : data가 가리키는 data block(i.e. 배열)에 8byte만큼을 수신한다.)
12. `void sciEnableNotification(sciBASE_t *sci, uint32 flags);` //
13. `void sciDisableNotification(sciBASE_t *sci, uint32 flags);` //
14. `void sciEnableLoopback(sciBASE_t *sci, loopBackType_t Loopbacktype);` //
15. `void sciDisableLoopback(sciBASE_t *sci);` //
16. `void sciEnterResetState(sciBASE_t *sci);` //
17. `void sciExitResetState(sciBASE_t *sci);` //
18. `void sci1GetConfigValue(sci_config_reg_t *config_reg, config_value_type_t type);` //
19. `void sciNotification(sciBASE_t *sci, uint32 flags);` //

5. I2C

1. `void i2cInit(void);` // 반드시 사용 I2C사용전 반드시 사용
2. `void i2cSetOwnAdd(i2cBASE_t *i2c, uint32 oadd);` // master 주소를 set해주는 함수.
   (i.e. i2cSetOwnAdd(i2cREG1, ADDRESS) : module i2cREG1의 master 주소(ADDRESS)를 set해준다. master가 1일 경우는 필요가 없다.)
3. `void i2cSetSlaveAdd(i2cBASE_t *i2c, uint32 sadd);` // Slave 주소를 set 해주는 함수.
   (i.e. i2cSetSlaveAdd(i2cREG2, 0x68) : Slave 주소를 0x68로 set해준다. 참고. 0x68은 MPU6050 datasheet에서 찾을 수 있는 고유 주소, Register - Who AM I를 찾은 후 register에 해당하는 주소를 찾으면 된다.)
4. `void i2cSetBaudrate(i2cBASE_t *i2c, uint32 baud);` // Baudrate를 변경해주는 함수.
   (i.e. i2cSetBaudrate(i2cREG1, num) : i2cREG1의 clock rate를 num값을 통하여 변경해준다. 실제 Clock rate를 알고싶으면 `i2cREG1->CKH = 5U;` 값과 비교하여 계산한다.)
5. `uint32 i2cIsTxReady(i2cBASE_t *i2c);` // 송신준비가 되어있는지 확인하는 함수. 준비가 되어있지않으면 TxReadyBuffer flag가 0이고, 준비가 되어있으면 1이다.
   (i.e. i2cIsTxReady(i2cREG1) : Moduel i2cREG1이 송신준비가 되어있다면 return 값이 1, 아닐경우에는 자기자신의 Tx flag를 return 한다.

6. `void i2cSendByte(i2cBASE_t *i2c, uint8 byte);` // Single Byte단위로 송신해주는 함수. TxReadyBuffer가 비어있을때까지 routine에서 대기를 한다. 대기모드에 빠지지 않기위하여 사용전 i2cIsTxReady를 통하여 Buffer가 비어있는지 아닌지 확인을 한다.
   (i.e. i2cSendByte(i2cREG1, msg + 1) : i2cREG1에서 출력하고자 하는 메시지(msg + 1)를 송신한다.)
7. `void i2cSend(i2cBASE_t *i2c, uint32 length, uint8 * data);` // Data 포인터가 가리키는 data buffer에서 length byte만큼 data block을 송신해주는 함수.
   (i.e. i2cSend(i2cREG1, 8, address) : address가 가리키는 data buffer에서 8byte크기만큼을 송신해준다.)
8. `uint32 i2cIsRxReady(i2cBASE_t *i2c);` // 수신준비가 되어있는지 확인하는 함수. 수신받을 준비가 되어있으면 RxReadyBuffer flag가 0이고, 준비가 되어있지 않으면 1이다.
   (i.e. i2cIsRxReady(i2cREG1) : Module i2cREG1이 수신준비가 되어있다면 return 값이 0, 준비 되어있지 않다면 return 값이 1로 출력된다. while문을 활용해서 사용하면 된다.)
9. `uint32 i2cIsStopDetected(i2cBASE_t *i2c);` // Stop Condition Detected를 활용하여 정지상태가 되어있는지 확인하는 함수. 1을 return하면 정상적으로 송수신가능, 0을 return하면 Stop 되어있는 상태
   (i.e. while(i2cIsStopDetected(i2cREG1) == 0) : 정상적으로 송수신 가능한 경우 1을 return하므로 while문을 빠져나와 다음 함수 또는 파일등을 진행한다.)
10. `void i2cClearSCD(i2cBASE_t *i2c);` // Stop Condition Detected flag를 clear해주는 함수.
   (i.e. i2cClearSCD(i2cREG1) : Module i2cREG1의 Stop Condition Detected flag를 clear시켜준다.)
11. `uint32 i2cRxError(i2cBASE_t *i2c);` // Error flag를 return 해주는 함수.
   (i.e. i2cRxError(i2cREG1) : Module i2cREG1의 Error가 존재하는 확인한다. Master가 2개 이상일 경우, Arbitration가 활성화 되어야 하지만 제대로 활성화가 되지 않는 경우 이 함수를 통하여 알려준다. 모든 송수신에서 Acknowledgement(ACK)를 통하여 1bit를 보내주는데 그러지 않는 경우 이 함수를 통하여 알려준다.)
12. `uint8 i2cReceiveByte(i2cBASE_t *i2c);` // Single Byte단위로 수신된 byte를 return 해주는 함수. Rx Buffer에 Byte가 없으면 수신될때가지 routine에서 무한대기한다. 그러므로 i2cIsRxReady함수를 통하여 Buffer가 차있는지 확인한다.)
   (i.e. msg = i2cReceiveByte(i2cREG1) : msg 메모리에 수신한 byte를 입력해준다.)
13. `void i2cReceive(i2cBASE_t *i2c, uint32 length, uint8 * data);` // data 포인터가 가리키는 data buffer에 length byte 길이만큼 data block을 넣어서 저장해주는 함수.
   (i.e. i2cReceive(i2cREG1, 8, address) : address가 가리키는 data buffer에 8byte만큼을 수신하여 저장한다.)
14. `void i2cEnableNotification(i2cBASE_t *i2c, uint32 flags);` // Interrupt를 활성화 시키는 함수.
   (i.e. i2cEnableNotification(i2cREG1, i2c_TX_INT) : interrupt가 활성화되어 i2cNotification 함수가 동작한다.)
15. `void i2cDisableNotification(i2cBASE_t *i2c, uint32 flags);` // Interrupt를 비활성화 시키는 함수.
   (i.e. i2cDisableNotification(i2cREG1, i2c_TX_INT) : Module i2cREG1의 interrupt를 비활성화시킨다.)
16. `void i2cSetStart(i2cBASE_t *i2c);` // Master mode에서만 사용가능하다. Data전송을 시작함을 알려주는 함수. Start bit를 생성하여 set해주는 함수.
   (i.e. i2cSetStart(i2cSetStart(i2cREG1) : Module i2cREG1이 송수신이 가능해진다.)
17. `void i2cSetStop(i2cBASE_t *i2c);` // Master mode에서만 사용가능하다. Data전송을 끝남을 알려주는 함수. Stop bit를 생성하여 set해주는 함수.
   (i.e. i2cSetStop(i2cREG1) : Module i2cREG1이 송수신이 끝난다.)
18. `void i2cSetCount(i2cBASE_t *i2c ,uint32 cnt);` // Master mode에서만 사용가능하다. 송신할 Bit를 Count를 해서 Count만큼 보내고 Stop Condition Deteceted를 생성한다.
   (i.e. i2cSetCount(i2cREG1, 2) : 2개의 bit를 보내고 Stop Condition Detected가 생성이 된다.)
19. `void i2cEnableLoopback(i2cBASE_t *i2c);` // Self test를 하기위하여 Loop back(like echo mode) mode를 허용해주는 함수.(Loop back : RX TX 2개 구성하기 힘들거나, 하나의 MCU내부에서 송수신이 제대로 작동되나 확인해보기 위한 mode)
   (i.e. i2cEnableLoopback(i2cREG1) : Module i2cREG1이 Loopback을 사용할 수 있다.)
20. `void i2cDisableLoopback(i2cBASE_t *i2c);` // Loopback mode를 비활성화시키는 함수.
   (i.e. i2cDisableLoopback(i2cREG1) : Module i2cREG1의 Loopback mode를 비활성화시킨다.)
21. `void i2cSetMode(i2cBASE_t *i2c, uint32 mode);` // Master mode나 Slave mode를 set 시켜주는 함수.
   (i.e. i2cSetMode(i2cREG1, I2C_MASTER) : Module i2cREG1을 Master mode로 set해준다.)
22. `void i2cSetDirection(i2cBASE_t *i2c, uint32 dir);` // 송수신을 결정해주는 함수.
   (i2cSetDirection(i2cREG1, I2C_TRANSMITTER) : Module i2cREG1을 송신으로 사용하겠다.)
23. `bool i2cIsMasterReady(i2cBASE_t *i2c);` // Stop Condition Detected가 생성되었는지 해제되어있는지 확인해주는 함수. 설정되어있으면 MST(Master bit)가 0가 return, 해제되어있으면 1이 return된다.
   (i.e. i2cIsMasterReady(i2cREG1) : Module i2cREG1의 Stop Condition Detected가 생성되있는지 확인한다.)
24. `bool i2cIsBusBusy(i2cBASE_t *i2c);` // Bus 사용 flag의 상태를 return 해주는 함수. 설정이 되어있으면 1을 return, 해제되어있으면 0을 return.
   (i.e. i2cIsBusBusy(i2cREG1) : Module i2cREG1의 Bus가 사용되면 1을 return, 사용이 안되면 0을 return된다.)
25. `void i2c1GetConfigValue(i2c_config_reg_t *config_reg, config_value_type_t type);` // i2cREG1의 초기값 혹은 현재 value를 불러오는 함수.

26. `void i2c2GetConfigValue(i2c_config_reg_t *config_reg, config_value_type_t type);` // i2cREG2의 초기값 혹은 현재 value를 불러오는 함수.
27. `void i2cNotification(i2cBASE_t *i2c, uint32 flags);` // Interrup handler for user mode.

Master : SCL을 통하여 CLK을 주는 역할(이따금씩 Multi Master라 하여 N : N 통신도 가능하다. 하지만 주로 1:N 통신)

6. ADC

1. `void adcInit(void);` // 반드시 사용 ADC사용전 반드시 사용
2. `void adcStartConversion(adcBASE_t *adc, uint32 group);` // 디지털 신호로 변환을 시작해주는 함수.
3. `void adcStopConversion(adcBASE_t *adc, uint32 group);` // 디지털 신호로의 변환을 끝내는 함수.
4. `void adcResetFiFo(adcBASE_t *adc, uint32 group);` // FiFo의 읽기 및 쓰기 포인터를 재설정하는 함수.
5. `uint32 adcGetData(adcBASE_t *adc, uint32 group, adcData_t *data);` // 변환된 데이터를 가져오는 함수.
6. `uint32 adcIsFifoFull(adcBASE_t *adc, uint32 group);` // FiFo 버퍼 상태를 확인하는 함수.
7. `uint32 adcIsConversionComplete(adcBASE_t *adc, uint32 group);` // 변환이 완료되었는지 확인하는 함수.
8. `void adcEnableNotification(adcBASE_t *adc, uint32 group);` // Notification 함수를 활성화시키는 함수.
9. `void adcDisableNotification(adcBASE_t *adc, uint32 group);` // Notification 함수를 비활성화시키는 함수.
10. `void adcCalibration(adcBASE_t *adc);` // ADC 모듈 교정 함수.
11. `uint32 adcMidPointCalibration(adcBASE_t *adc);` // ADC 모듈 교정 함수(중간지점사용).
12. `void adcSetEVTPin(adcBASE_t *adc, uint32 value);` // 출력 핀으로 구성된 ADC EVT 핀을 설정하는 함수.
13. `uint32 adcGetEVTPin(adcBASE_t *adc);` // ADC EVT 핀의 값을 반환하는 함수
14. `void adc1GetConfigValue(adc_config_reg_t *config_reg, config_value_type_t type);` // 초기값 또는 현재값을 복사하는 함수.
15, `void adc2GetConfigValue(adc_config_reg_t *config_reg, config_value_type_t type);` // 초기값 또는 현재값을 복사하는 함수.
16. `void adcNotification(adcBASE_t *adc, uint32 group);` // Notification 함수 활성시 동작함수.

7. HET - 정한별

8. CAN - 장성환

9. FREE RTOS - 김형주

MCU_Peripheral_필수함수

5. I2C

1. `void i2cInit(void);` // 반드시 사용 I2C사용전 반드시 사용.
3. `void i2cSetSlaveAdd(i2cBASE_t *i2c, uint32 sadd);` // Slave 주소를 set 해주는 함수.
   (i.e. i2cSetSlaveAdd(i2cREG2, 0x68) : Slave 주소를 0x68로 set해준다. 참고. 0x68은 MPU6050 datasheet에서 찾을 수 있는 고유 주소, Register - Who AM I를 찾은 후 register에 해당하는 주소를 찾으면 된다.)
16. `void i2cSetStart(i2cBASE_t *i2c);` // Master mode에서만 사용가능하다. Data전송을 시작함을 알려주는 함수. Start bit를 생성하여 set해주는 함수.
   (i.e. i2cSetStart(i2cSetStart(i2cREG1) : Module i2cREG1이 송수신이 가능해진다.)
17. `void i2cSetStop(i2cBASE_t *i2c);` // Master mode에서만 사용가능하다. Data전송을 끝남을 알려주는 함수. Stop bit를 생성하여 set해주는 함수.
   (i.e. i2cSetStop(i2cREG1) : Module i2cREG1이 송수신이 끝난다.)
18. `void i2cSetCount(i2cBASE_t *i2c ,uint32 cnt);` // Master mode에서만 사용가능하다. 송신할 Bit를 Count를 해서 Count만큼 보내고 Stop Condition Deteceted를 생성한다.
   (i.e. i2cSetCount(i2cREG1, 2) : 2개의 bit를 보내고 Stop Condition Detected가 생성이 된다.)