CM900 API Summary

CM9 시리즈의 모든 API는 Core library라고 하고 IDE 폴더의 아래 경로에 위치해 있습니다.

₩ROBOTIS₩hardware₩robotis₩core₩robotis₩

여기에 서술된 대부분의 API들은 Leaf Maple보드나 Arduino홈페이지를 활용하시면 더욱 풍부한 설명과 예제를 확인하실 수 있습니다.

CM9 Developer World
http://www.robotsource.org/xe/Circle_CM9_Developer_World

Leaflabs의 Maple Reference
http://leaflabs.com/docs/language-index.html

Arduino Reference
http://arduino.cc/en/Reference/HomePage

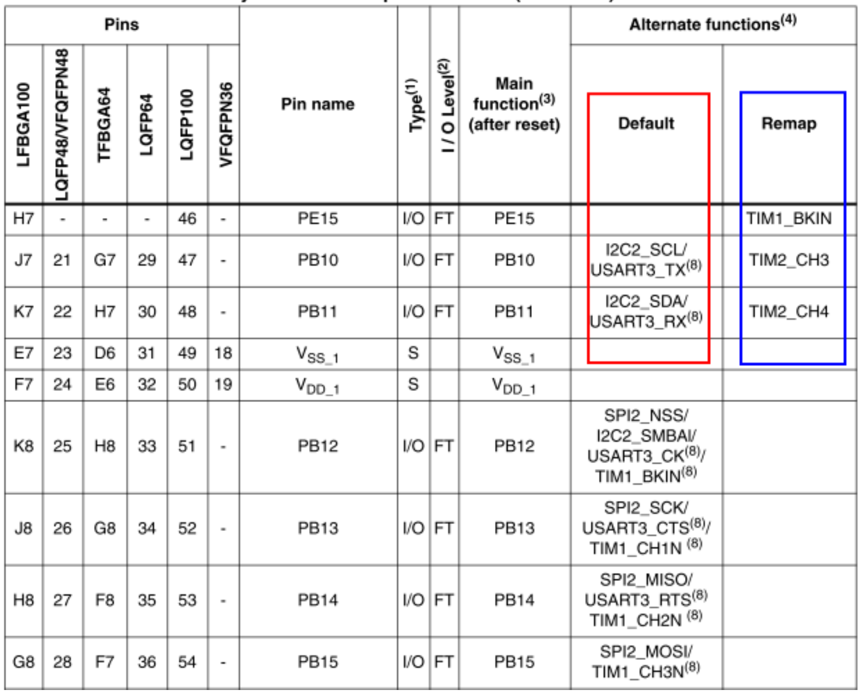
CM-900 Pin Description

또한 각각의 핀에 대한 설정은 아래의 테이블을 보시고 STM32F103C8 에 대한 데이터쉬트를 참고하시면 더욱 이해가 빠를 것입니다.

CM900	STM32F103C8	CM900	STM32F103C8	CM900	STM32F103C8
D0	PA0	D11	PA13	D22	PB8
D1	PA1	D12	PA14	D23	PB9
D2	PA2	D13	PA15	D24	PB10
D3	PA3	D14	PB0	D25	PB11
D4	PA4	D15	PB1	D26	PB12
D5	PA5	D16	PB2	D27	PB13
D6	PA6	D17	PB3	D28	PB14
D7	PA7	D18	PB4	D29	PB15
D8	PA8	D19	PB5	D30	PC14
D9	PA9	D20	PB6	D31	PC15
D10	PA10	D21	PB7		

특히 모든 핀들이 GPIO로 쓸 수 있는 것은 아니기 때문에 Default로 어떤 기능인지 데이터쉬트에서 참고하시고 Remap이 필요하다면 Alternative Function관련 API로 설정을 변경하셔야 합니다.

Table 5. Medium-density STM32F103xx pin definitions (continued)



Pins						Pin name	Type[1]	I / O Level[2]	Main function[3] (after reset)	Alternate functions[4]	
LFBGA100	LQFP48/VFQFPN48	TFBGA64	LQFP64	LQFP100	VFQFPN36					Default	Remap
H7	-	-	-	46	-	PE15	I/O	FT	PE15		TIM1_BKIN
J7	21	G7	29	47	-	PB10	I/O	FT	PB10	I2C2_SCL/ USART3_TX[8]	TIM2_CH3
K7	22	H7	30	48	-	PB11	I/O	FT	PB11	I2C2_SDA/ USART3_RX[8]	TIM2_CH4
E7	23	D6	31	49	18	V_{SS_1}	S		V_{SS_1}		
F7	24	E6	32	50	19	V_{DD_1}	S		V_{DD_1}		
K8	25	H8	33	51	-	PB12	I/O	FT	PB12	SPI2_NSS/ I2C2_SMBAl/ USART3_CK[8]/ TIM1_BKIN[8]	
J8	26	G8	34	52	-	PB13	I/O	FT	PB13	SPI2_SCK/ USART3_CTS[8]/ TIM1_CH1N [8]	
H8	27	F8	35	53	-	PB14	I/O	FT	PB14	SPI2_MISO/ USART3_RTS[8] TIM1_CH2N [8]	
G8	28	F7	36	54	-	PB15	I/O	FT	PB15	SPI2_MOSI/ TIM1_CH3N[8]	

위의 pin definitions에 대한 테이블을 보시면 Default로 된 기능은 다른 설정 없이 쓸 수 있지만 Remap으로 된 기능들은 따로 설정을 해주셔야 쓸 수 있습니다.

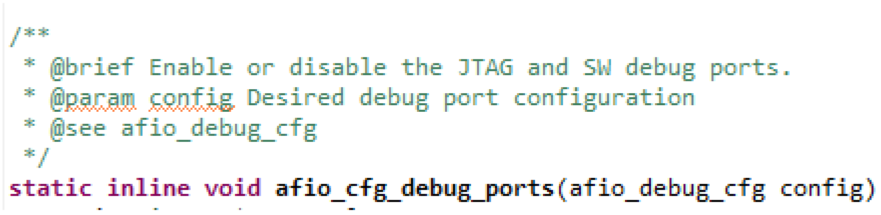
```
void afio_remap(afio_remap_peripheral p);
```

afio_remap_peripheral로 쓸 수 있는 인자는 아래와 같습니다.

```
AFIO_REMAP_ADC2_ETRGREG  = AFIO_MAPR_ADC2_ETRGREG_REMAP, /**<
    ADC 2 external trigger regular conversion remapping */
AFIO_REMAP_ADC2_ETRGINJ  = AFIO_MAPR_ADC2_ETRGINJ_REMAP, /**<
    ADC 2 external trigger injected conversion remapping */
AFIO_REMAP_ADC1_ETRGREG  = AFIO_MAPR_ADC1_ETRGREG_REMAP, /**<
    ADC 1 external trigger regular conversion remapping */
AFIO_REMAP_ADC1_ETRGINJ  = AFIO_MAPR_ADC1_ETRGINJ_REMAP, /**<
    ADC 1 external trigger injected conversion remapping */
AFIO_REMAP_TIM5CH4_I      = AFIO_MAPR_TIM5CH4_IREMAP, /**<
    Timer 5 channel 4 internal remapping */
AFIO_REMAP_PD01           = AFIO_MAPR_PD01_REMAP, /**<
    Port D0/Port D1 mapping on OSC_IN/OSC_OUT */
AFIO_REMAP_CAN_1          = AFIO_MAPR_CAN_REMAP_PB8_PB9, /**<
    CAN alternate function remapping 1 (RX on PB8, TX on PB9) */
AFIO_REMAP_CAN_2          = AFIO_MAPR_CAN_REMAP_PD0_PD1, /**<
    CAN alternate function remapping 2 (RX on PD0, TX on PD1) */
AFIO_REMAP_TIM4           = AFIO_MAPR_TIM4_REMAP, /**<
    Timer 4 remapping */
AFIO_REMAP_TIM3_PARTIAL  = AFIO_MAPR_TIM3_REMAP_PARTIAL, /**<
    Timer 3 partial remapping */
AFIO_REMAP_TIM3_FULL      = AFIO_MAPR_TIM3_REMAP_FULL, /**<
    Timer 3 full remapping */
AFIO_REMAP_TIM2_PARTIAL_1 = AFIO_MAPR_TIM2_REMAP_PA15_PB3_PA2_PA3, /**<
    Timer 2 partial remapping 1 (CH1 and ETR on PA15, CH2 on PB3, CH3
    on PA2, CH4 on PA3) */
AFIO_REMAP_TIM2_PARTIAL_2 = AFIO_MAPR_TIM2_REMAP_PA0_PA1_PB10_PB11, /**<
    Timer 2 partial remapping 2 (CH1 and ETR on PA0, CH2 on PA1, CH3
    on PB10, CH4 on PB11) */
AFIO_REMAP_TIM2_FULL      = AFIO_MAPR_TIM2_REMAP_FULL, /**<
    Timer 2 full remapping */
AFIO_REMAP_USART2         = AFIO_MAPR_USART2_REMAP, /**<
    USART 2 remapping */
AFIO_REMAP_USART1         = AFIO_MAPR_USART1_REMAP, /**<
    USART 1 remapping */
AFIO_REMAP_I2C1           = AFIO_MAPR_I2C1_REMAP, /**<
```

```
     I2C 1 remapping */
  AFIO_REMAP_SPI1          = AFIO_MAPR_SPI1_REMAP, /**<
     SPI 1 remapping */
  AFIO_REMAP_FSMC_NADV      = (AFIO_MAPR2_FSMC_NADV |
                             AFIO_REMAP_USE_MAPR2), /**<
     NADV signal not connected */
  AFIO_REMAP_TIM14         = (AFIO_MAPR2_TIM14_REMAP |
                             AFIO_REMAP_USE_MAPR2), /**<
     Timer 14 remapping */
  AFIO_REMAP_TIM13         = (AFIO_MAPR2_TIM13_REMAP |
                             AFIO_REMAP_USE_MAPR2), /**<
     Timer 13 remapping */
  AFIO_REMAP_TIM11         = (AFIO_MAPR2_TIM11_REMAP |
                             AFIO_REMAP_USE_MAPR2), /**<
     Timer 11 remapping */
  AFIO_REMAP_TIM10         = (AFIO_MAPR2_TIM10_REMAP |
                             AFIO_REMAP_USE_MAPR2), /**<
     Timer 10 remapping */
  AFIO_REMAP_TIM9          = (AFIO_MAPR2_TIM9_REMAP |
                             AFIO_REMAP_USE_MAPR2) /**<
```

디버그 포트관련 Remap 설정은 아래 API로 변경합니다.



```
/**
 * @brief Enable or disable the JTAG and SW debug ports.
 * @param config Desired debug port configuration
 * @see afio_debug_cfg
 */
static inline void afio_cfg_debug_ports(afio_debug_cfg config)
```

afio_debug_cfg로 등록된 인자는 아래와 같습니다.

```
  AFIO_DEBUG_FULL_SWJ = AFIO_MAPR_SWJ_CFG_FULL_SWJ, /**<
                             Full Serial Wire and JTAG debug */
  AFIO_DEBUG_FULL_SWJ_NO_NJRST = AFIO_MAPR_SWJ_CFG_FULL_SWJ_NO_NJRST, /**<
                             Full Serial Wire and JTAG, but no NJTRST. */
  AFIO_DEBUG_SW_ONLY = AFIO_MAPR_SWJ_CFG_NO_JTAG_SW, /**<
                             Serial Wire debug only (JTAG-DP disabled,
                             SW-DP enabled) */
  AFIO_DEBUG_NONE = AFIO_MAPR_SWJ_CFG_NO_JTAG_NO_SW /**<
                             No debug; all JTAG and SW pins are free
                             for use as GPIOs. */
```

단 JTAG Debug 모드를 Disable하면 추후에 JTAG/SWD연결이 안될 수 있으니 다시 사용 하시고 싶다면 반드시 Enable로 해주시면 됩니다.

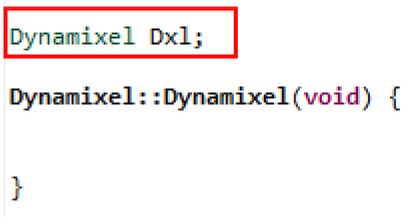
Contents

1. Dynamixel API

다이나믹셀은 Dxl이라는 클래스를 미리 Dxl이라는 인스턴스를 미리 선언해놓았고 아래의 메소드를 활용해서 다이나믹셀을 제어하시면 됩니다.

다이나믹셀 프로토콜과 제품에 대한 설명은 아래의 ROBOTIS e-Manual을 참고하시면 됩니다.

http://support.robotis.com/



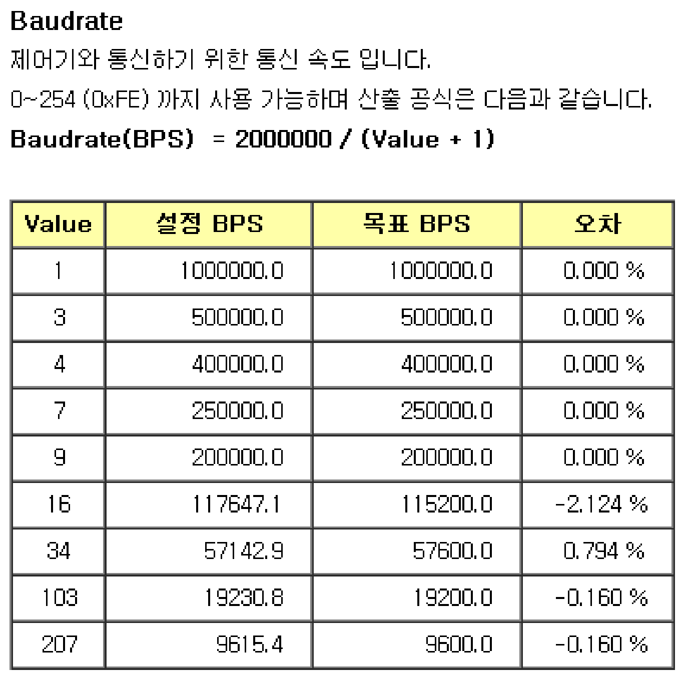
```
Dynamixel Dxl;

Dynamixel::Dynamixel(void) {


}
```

Dynamixel 클래스에 포함된 메소드 리스트

```
/////////// Device control methods /////////////
   void begin(int buad);
```

baud로 쓸 수 있는 value는 아래의 테이블을 보시고 원하시는 baud rate로 초기화하시면 됩니다.

아래는 ROBOTIS e-Manual에서 캡쳐한 화면입니다.



Baudrate

제어기와 통신하기 위한 통신 속도 입니다.

0~254 (0xFE) 까지 사용 가능하며 산출 공식은 다음과 같습니다.

Baudrate(BPS) = 2000000 / (Value + 1)

Value	설정 BPS	목표 BPS	오차
1	1000000.0	1000000.0	0.000 %
3	500000.0	500000.0	0.000 %
4	400000.0	400000.0	0.000 %
7	250000.0	250000.0	0.000 %
9	200000.0	200000.0	0.000 %
16	117647.1	115200.0	-2.124 %
34	57142.9	57600.0	0.794 %
103	19230.8	19200.0	-0.160 %
207	9615.4	9600.0	-0.160 %

```
   void end(void);

   //// High communication methods ////////
   int readByte( int id, int address );
   void writeByte( int id, int address, int value );
   int readWord( int id, int address );
   void writeWord( int id, int address, int value );
```

```
    void ping(int id);
    void reset(int id);
    int getResult(void);
    void setPosition(int ServoID, int Position, int Speed);//Made by Martin S.
Mason(Professor @Mt. San Antonio College)

    /////// Methods for making a packet ////////
    void setTxPacketId( int id );
    void setTxPacketInstruction( int instruction );
    void setTxPacketParameter( int index, int value );
    void setTxPacketLength( int length );
    int getRxPacketParameter( int index );
    int getRxPacketLength(void);
    int getRxPacketError( int errbit );

    ////////// utility methods for value ////////////
    int makeWord( int lowbyte, int highbyte );
    int getLowByte( int word );
    int getHighByte( int word );

    ////////// packet communication methods ///////////////////////
    void txPacket(void);
    void rxPacket(void);
    void txrxPacket(void);
```

아래는 Dynamixel 프로토콜을 쓰고 있는 HaViMo 카메라관련 메소드

```
    /**
     * Wrapper function to begin an image capture with the HaViMo2 camera module.
     * @param id HaViMo2 camera ID (fixed as 100 in HaViMo2 firmware).
     * @see dxl_recover() and havGet()
     */
       void havCap(uint8_t id);
    /**
     * Wrapper function to retrieve an image buffer from a HaViMo2 camera module.
     * @param id HaViMo2 camera ID (fixed as 100 in HaViMo2 firmware).
     * @param hvm2rb Pointer to a user region buffer data type.
     * @see havCap()
     * @return The number of valid regions found in the image.
     */
       uint8_t havGet(uint8_t id, HaViMo2_Region_Buffer_t* hvm2rb);


void setup() {
    //baud로 쓸수 있는 인자는 아래의 리스트를 참고하거나
    //support.robotis.com 의 다이나믹셀 패킷부분을 참고.
    //아래는 1 로 초기화했을 경우 1Mpbs의 속도를 가지고 34 의 경우 57600bps
    Dxl.begin(1);
}

void loop() {
    delay(1000);                // Wait for 1 second (1000 milliseconds)
    Dxl.writeWord(1, 30, 100); //Turn dynamixel ID 1 to position 100
    delay(1000);                // Wait for 1 second (1000 milliseconds)
    Dxl.writeWord(1, 30, 1000);//Turn dynamixel ID 1 to position 1000
```

}

다이나믹셀 제어는 CM-900 같은 제어기에서 다이나믹셀로 Instruction packet이라는 바이너리 형태의 데이터를 보냄으로써 제어가 시작된다. 총 7 가지의 명령이 있고 패킷을 받은 다이나믹셀은 명령을 수행한뒤에 그결과를 status packet으로 리턴한다. 아래와 같이 support.robotis.com을 방문하시면 구입하신 다이나믹셀의 Control table을 확인하실 수 있는데 목표위치를 바꾸고 싶으시면 Goal position이라는 명령의 주소를 확인하고 writeWord()같은 메소드로 value를 보내면 됩니다.
나머지는 ROBOTIS e-Manual 참고하시면 됩니다.
http://support.robotis.com/

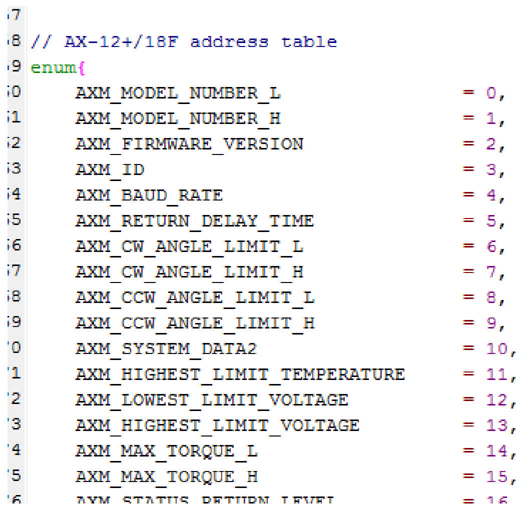
Dxl.writeWord(1, 30, 1000); // 1 번 ID의 다이나믹셀에 1000 의 위치로 이동

30 (0X1E)	Goal Position(L)	목표 위치 값의 하위 바이트	RW
31 (0X1F)	Goal Position(H)	목표 위치 값의 상위 바이트	RW

또한 아래의 라이브러리 폴더의 dynamixel_address_tables.h 파일을 include하시면 다양한 다이나믹셀 시리즈의 컨트롤 테이블이 준비되어 있습니다.

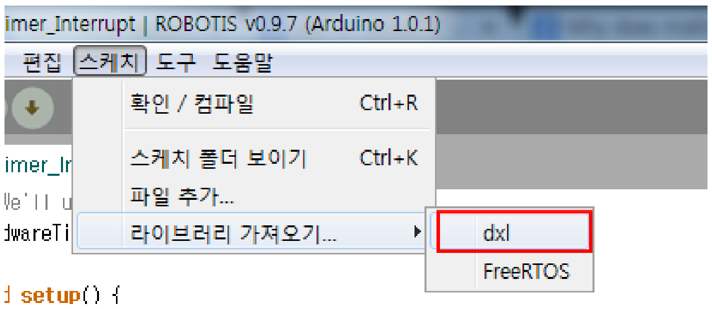
```
\ROBOTIS\libraries\dxl\dynamixel_address_tables.h
```

열어보시면 아래와 같이 다이나믹셀 모델에 대한 컨트롤 테이블이 미리 준비되어 있습니다.



```
// AX-12+/18F address table
enum{
    AXM_MODEL_NUMBER_L               = 0,
    AXM_MODEL_NUMBER_H               = 1,
    AXM_FIRMWARE_VERSION             = 2,
    AXM_ID                           = 3,
    AXM_BAUD_RATE                    = 4,
    AXM_RETURN_DELAY_TIME            = 5,
    AXM_CW_ANGLE_LIMIT_L             = 6,
    AXM_CW_ANGLE_LIMIT_H             = 7,
    AXM_CCW_ANGLE_LIMIT_L            = 8,
    AXM_CCW_ANGLE_LIMIT_H            = 9,
    AXM_SYSTEM_DATA2                 = 10,
    AXM_HIGHEST_LIMIT_TEMPERATURE    = 11,
    AXM_LOWEST_LIMIT_VOLTAGE         = 12,
    AXM_HIGHEST_LIMIT_VOLTAGE        = 13,
    AXM_MAX_TORQUE_L                 = 14,
    AXM_MAX_TORQUE_H                 = 15,
    AXM_STATUS_RETURN_LEVEL          = 16
```

혹은 스케치(Scketch) -> 라이브러리 가져오기(Import Library) -> dxl을 선택하셔도 됩니다.



아래의 Constants들은 따로 define없이 그냥 쓰시면 됩니다. 모두 다이나믹셀 공통 Constants입니다. 여기에 없는 Instruction들은 따로 define해서 쓰시면 됩니다.

```
#define BROADCAST_ID         (254)
/*
 * Instruction command
 * */
#define INST_PING           0x01
#define INST_READ           0x02
#define INST_WRITE          0x03
#define INST_REG_WRITE       0x04
#define INST_ACTION         0x05
#define INST_RESET          0x06
#define INST_DIGITAL_RESET  0x07    //reset과 동일한데 id를 제외하고 나머지는 reset동일
#define INST_SYSTEM_READ    0x0C    //어떤 주소위치든 1 byte 를 무조건 읽을 수 있게. CT
범위를 초과해도 읽음.
#define INST_SYSTEM_WRITE   0x0D    //어떤 주소위치든 1 byte 를 무조건 쓸수 있게. CT 범위를
초과해도 씀.
#define INST_SYNC_WRITE     0x83
#define INST_SYNC_REG_WRITE 0x84    //action을 내려야만 실행. 여러 ID dxl 에 명령 보냄.

/*
 * Index for packet instruction
 * */
#define DEFAULT_BAUDNUMBER   (1)
#define ID                          (2)
#define LENGTH               (3)
#define INSTRUCTION                 (4)
#define ERRBIT               (4)
#define PARAMETER                   (5)


#define MAXNUM_RXPARAM              (60)
#define MAXNUM_TXPARAM              (150)


/*
 * defines for error message
 * */
#define ERRBIT_VOLTAGE              (1)
#define ERRBIT_ANGLE         (2)
#define ERRBIT_OVERHEAT             (4)
#define ERRBIT_RANGE         (8)
#define ERRBIT_CHECKSUM             (16)
#define ERRBIT_OVERLOAD             (32)
#define ERRBIT_INSTRUCTION   (64)

/*
 * defines message of communication
 * */
#define COMM_TXSUCCESS              (0)
#define COMM_RXSUCCESS              (1)
#define COMM_TXFAIL                 (2)
#define COMM_RXFAIL                 (3)
#define COMM_TXERROR         (4)
#define COMM_RXWAITING              (5)
#define COMM_RXTIMEOUT              (6)
#define COMM_RXCORRUPT              (7)
```

2. Zigbee API

```
///////////// device control methods ///////////////////////

int zgbInitialize( int devIndex );
```

Zigbee/bluetooth를 초기화 하는 API, USART2번을 57600bps로 초기화하고 ROBOTIS에서 사용되는 프로토콜을 사용하기 위한 변수, 타이머 장치를 초기화 합니다.

devIndex는 0으로 초기화 하면 되고 아무 숫자나 넣어도 상관없습니다. 원래는 호스트 프로그래밍에서 COM포트 번호를 넣는 인자이나 CM900은 Zigbee포트가 항상 USART2번이므로 지금은 의미가 없습니다.

```
void zgbTerminate(void);
```

프로토콜을 종료, USART2번이나 타이머 장치를 종료함.

```
////////// communication methods //////////////////////
int zgbTxData(int data);
int zgbRxCheck(void);
int zgbRxData(void);
```

```
if(zgbRxCheck() == 1){    //Zigbee/Bluetooth로 데이터가 왔는지 확인
        RcvData = zgbRxData();    //데이터가 왔다면 zgbRxData()로 데이터를 받음
        SerialUSB.print("RcvData = ");
        SerialUSB.println(RcvData);
```

3. Digital I/O

```
/**
 * Configure behavior of a GPIO pin.
 *
 * @param pin Number of pin to configure.
 * @param mode Mode corresponding to desired pin behavior.
 * @see WiringPinMode
 */

void pinMode(uint8 pin, WiringPinMode mode);
```

CM900의 pin을 초기화 할 때 쓰이며, mode로 쓸 수 있는 값은 아래와 같습니다.

출력일 경우 OUTPUT, 외부에 따로 풀업/풀다운을 쓰고 싶다면 OUTPUT_OPEN_DRAIN을하고 입력핀으로 쓰고 싶을 때는 INPUT으로 쓰면 됩니다.

```
    pinMode(0, OUTPUT);
    pinMode(1, OUTPUT_OPEN_DRAIN);
```

PWM출력을 하고 싶으면 mode를 PWM으로 초기화 하면 됩니다. 이렇게 초기화 한 핀은 analogWrite()나 pwmWrite() 함수로 제어할 수 있습니다. 단 PWM출력은 TIMER 기능이 있는 핀만 해당하니 보드에서 TIMER가 되는 핀인지 확인이 필요합니다.

```
pinMode(0, PWM);
```

```
OUTPUT, /**< Basic digital output: when the pin is HIGH, the
          voltage is held at +3.3v (Vcc) and when it is LOW, it
          is pulled down to ground. */

OUTPUT_OPEN_DRAIN, /**< In open drain mode, the pin indicates
                     "low" by accepting current flow to ground
                     and "high" by providing increased
                     impedance. An example use would be to
                     connect a pin to a bus line (which is pulled
                     up to a positive voltage by a separate
                     supply through a large resistor). When the
                     pin is high, not much current flows through
                     to ground and the line stays at positive
                     voltage; when the pin is low, the bus
                     "drains" to ground with a small amount of
                     current constantly flowing through the large
                     resistor from the external supply. In this
                     mode, no current is ever actually sourced
                     from the pin. */

INPUT, /**< Basic digital input. The pin voltage is sampled; when
         it is closer to 3.3v (Vcc) the pin status is high, and
         when it is closer to 0v (ground) it is low. If no
         external circuit is pulling the pin voltage to high or
         low, it will tend to randomly oscillate and be very
         sensitive to noise (e.g., a breath of air across the pin
         might cause the state to flip). */

INPUT_ANALOG, /**< This is a special mode for when the pin will be
                used for analog (not digital) reads.  Enables ADC
                conversion to be performed on the voltage at the
                pin. */

INPUT_PULLUP, /**< The state of the pin in this mode is reported
                the same way as with INPUT, but the pin voltage
                is gently "pulled up" towards +3.3v. This means
                the state will be high unless an external device
                is specifically pulling the pin down to ground,
                in which case the "gentle" pull up will not
                affect the state of the input. */

INPUT_PULLDOWN, /**< The state of the pin in this mode is reported
                  the same way as with INPUT, but the pin voltage
                  is gently "pulled down" towards 0v. This means
                  the state will be low unless an external device
                  is specifically pulling the pin up to 3.3v, in
                  which case the "gentle" pull down will not
                  affect the state of the input. */

INPUT_FLOATING, /**< Synonym for INPUT. */

PWM, /**< This is a special mode for when the pin will be used for
       PWM output (a special case of digital output). */

PWM_OPEN_DRAIN, /**< Like PWM, except that instead of alternating
                  cycles of LOW and HIGH, the voltage on the pin
                  consists of alternating cycles of LOW and
                  floating (disconnected). */
```

디지털 제어 함수는 아래의 주석 부분을 참고 하시면 됩니다.

```
/**
 * Writes a (digital) value to a pin.  The pin must have its
 * mode set to OUTPUT or OUTPUT_OPEN_DRAIN.
 *
 * @param pin Pin to write to.
 * @param value Either LOW (write a 0) or HIGH (write a 1).
 * @see pinMode()
 */
void digitalWrite(uint8 pin, uint8 value);

/**
 * Read a digital value from a pin.  The pin must have its mode set to
 * one of INPUT, INPUT_PULLUP, and INPUT_PULLDOWN.
 *
 * @param pin Pin to read from.
 * @return LOW or HIGH.
 * @see pinMode()
 */
uint32 digitalRead(uint8 pin);

/**
 * Read an analog value from pin.  This function blocks during ADC
 * conversion, and has 12 bits of resolution.  The pin must have its
 * mode set to INPUT_ANALOG.
 *
 * @param pin Pin to read from.
 * @return Converted voltage, in the range 0--4095, (i.e. a 12-bit ADC
 *         conversion).
 * @see pinMode()
 */
uint16 analogRead(uint8 pin);

/**
 * Toggles the digital value at the given pin.
 *
 * The pin must have its mode set to OUTPUT.
 *
 * @param pin the pin to toggle.  If the pin is HIGH, set it LOW.  If
 * it is LOW, set it HIGH.
 *
 * @see pinMode()
 */
void togglePin(uint8 pin);

/**
 * Toggle the LED.
 *
 * If the LED is on, turn it off.  If it is off, turn it on.
 *
 * The LED must its mode set to OUTPUT. This can be accomplished
 * portably over all LeafLabs boards by calling pinMode(BOARD_LED_PIN,
 * OUTPUT) before calling this function.
 *
 * @see pinMode()
 */
static inline void toggleLED() {
    togglePin(BOARD_LED_PIN);
}
```

아래와 같이 blink 예제를 toggleLED()라는 메소드로 쉽게 구현하실 수 있습니다.

```
void loop(){
        toggleLED()
        delay(100);
}
```

아두이노에서 쓰이는 analogWrite()함수와 pwmWrite는 동일 함수입니다.

```
#define analogWrite pwmWrite

/**
 * Set the PWM duty on the given pin.
 *
 * User code is expected to determine and honor the maximum value
 * (based on the configured period).
 *
 * @param pin PWM output pin
 * @param duty_cycle Duty cycle to set.
 */
void pwmWrite(uint8 pin, uint16 duty_cycle);
```

```
int analogPin = 3;     // potentiometer connected to analog pin 3

void setup() {
  pinMode(0, PWM);    // sets the 0 pin as output

  pinMode(analogPin, INPUT_ANALOG); // sets the potentiometer pin as
                                    // analog input
}

void loop() {
  int val = analogRead(analogPin);         // read the input pin

  pwmWrite(0, val * 16);  // analogRead values go from 0
                              // to 4095, pwmWrite values
                              // from 0 to 65535, so scale roughly
}
```

핀 11(PA13), 12(PA14), 13(PA15), 17(PB3), 18(PB4)번에 PWM을 출력하기 위해서는 AFIO기능을 활성화해야 합니다.
여기 핀들은 기본적으로(Default) JTAG와 관련된 핀들이므로
AFIO_DEBUG_NONE으로 비활성화 해야 PWM관련 기능을 활용할 수 있습니다.

afio_cfg_debug_ports(AFIO_DEBUG_NONE);

```
void setup() {
   afio_cfg_debug_ports(AFIO_DEBUG_NONE);
   pinMode(11, PWM);    // sets the 11 pin as PWM

}

void loop() {
…
}
```

4. Ext-Interrupt

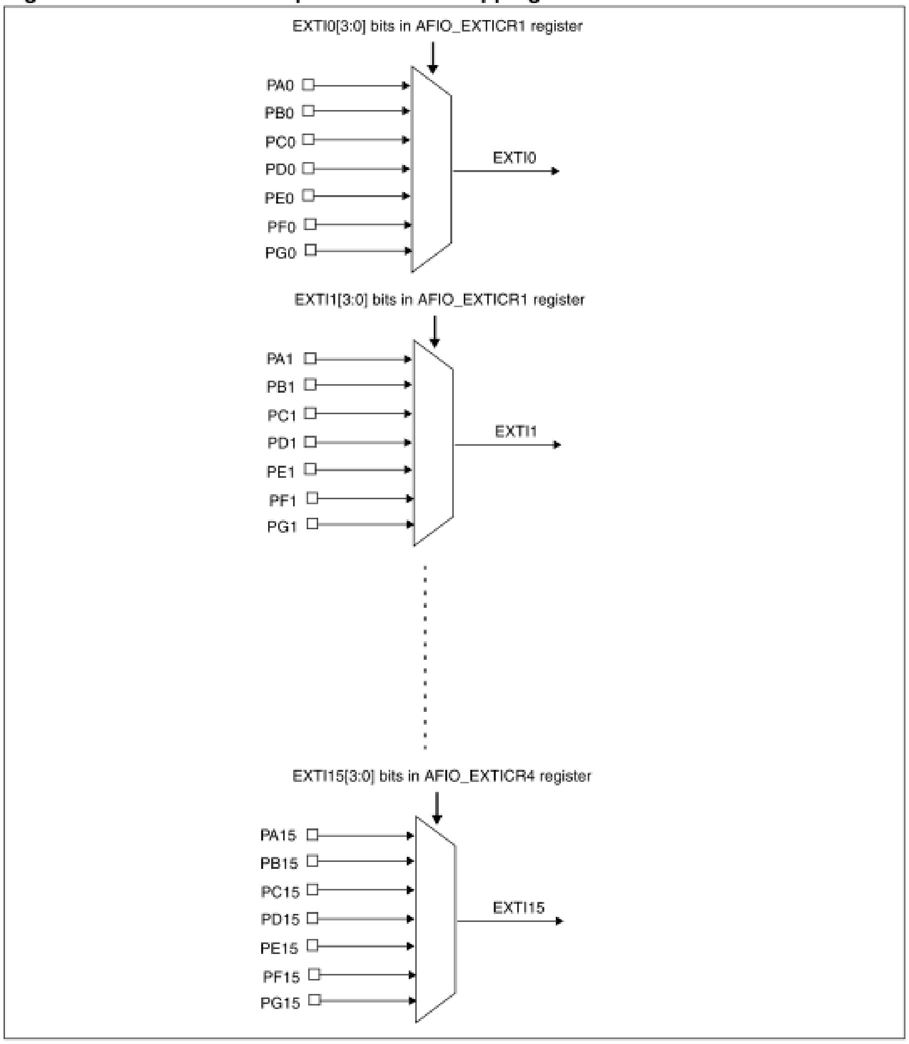
외부의 핀의 변화를 감지해서 인터럽트를 발생시키는 기능입니다. attachInterrupt()를 이용해서 원하는 interrupt 함수와 연결할 수 있습니다. 단, CM-900 의 경우 STM32F103C8 에서 지원하는 16 개의 EXIT Line에 따라 외부 인터럽트를 제어하고 동일한 EXIT Line에 연결된 GPIO끼리는 따로 인터럽트를 함수를 연결할 수 없습니다.

아래의 Figure21 블록다이어그램을 참고하시면 각각의 EXTI Line들에 어떤 GPIO가 연결되어 있는지 확인하고 같은 라인에 붙어 있는 GPIO들은 똑 같은 인터럽트 핸들러가 수행되니 참고하세요.

참고로 STM32 시리즈는 총 20 개의 EXTI Line이 있는데 16 개는 일반 GPIO에 연결되고 나머지 4 개는 아래와 같이 특수한 기능에 External Interrupt를 제공하고 있습니다.

- EXTI line 16 is connected to the PVD output
- EXTI line 17 is connected to the RTC Alarm event
- EXTI line 18 is connected to the USB Wakeup event
- EXTI line 19 is connected to the Ethernet Wakeup event (available only in connectivity line devices)

Figure 21. External interrupt/event GPIO mapping



EXTI0[3:0] bits in AFIO_EXTICR1 register
PA0
PB0
PC0
PD0
PE0
PF0
PG0
EXTI0
EXTI1[3:0] bits in AFIO_EXTICR1 register
PA1
PB1
PC1
PD1
PE1
PF1
PG1
EXTI1
EXTI15[3:0] bits in AFIO_EXTICR4 register
PA15
PB15
PC15
PD15
PE15
PF15
PG15
EXTI15

1. To configure the AFIO_EXTICRx for the mapping of external interrupt/event lines onto GPIOs, the AFIO clock should first be enabled. Refer to *Section 6.3.7: APB2 peripheral clock enable register (RCC_APB2ENR)* for low-, medium- and high-density devices and, to *Section 7.3.7: APB2 peripheral clock enable register (RCC_APB2ENR)* for connectivity line devices.

```
/**
 *  @brief Registers an interrupt handler on a pin.
 *
 *  The interrupt will be triggered on a given transition on the pin,
 *  as specified by the mode parameter.  The handler runs in interrupt
 *  context.  The new handler will replace whatever handler is
 *  currently registered for the pin, if any.
 *
 *  @param pin Maple pin number
 *  @param handler Function to run upon external interrupt trigger.
 *  The handler should take no arguments, and have void return type.
 *  @param mode Type of transition to trigger on, e.g. falling, rising, etc.
 *
 *  @sideeffect Registers a handler
 *  @see detachInterrupt()
 */
void attachInterrupt(uint8 pin, voidFuncPtr handler, ExtIntTriggerMode mode);


ExtIntTriggerMode
    RISING, /**< To trigger an interrupt when the pin transitions LOW
              to HIGH */
    FALLING, /**< To trigger an interrupt when the pin transitions
               HIGH to LOW */
    CHANGE /**< To trigger an interrupt when the pin transitions from
             LOW to HIGH or HIGH to LOW (i.e., when the pin
             changes). */
```

```
volatile int state = LOW; // must declare volatile, since it's
                          // modified within the blink() handler

void setup() {
    pinMode(BOARD_LED_PIN, OUTPUT);
    pinMode(0, INPUT);
    attachInterrupt(0, blink, CHANGE);
}

void loop() {
    digitalWrite(BOARD_LED_PIN, state);
}

void blink() {
    if (state == HIGH) {
        state = LOW;
    } else { // state must be LOW
        state = HIGH;
    }
}
```

```
/**
 * @brief Disable any registered external interrupt.
 * @param pin Maple pin number
 * @sideeffect unregisters external interrupt handler
 * @see attachInterrupt()
 */
void detachInterrupt(uint8 pin);

/**
 * Re-enable interrupts.
 *
 * Call this after noInterrupts() to re-enable interrupt handling,
 * after you have finished with a timing-critical section of code.
 *
 * @see noInterrupts()
 */
static inline void interrupts() {
    nvic_globalirq_enable();
}

/**
 * Disable interrupts.
 *
 * After calling this function, all user-programmable interrupts will
 * be disabled.  You can call this function before a timing-critical
 * section of code, then call interrupts() to re-enable interrupt
 * handling.
 *
 * @see interrupts()
 */
static inline void noInterrupts() {
    nvic_globalirq_disable();
}
```

Debug pin configuration API

```
static inline void disableDebugPorts(void) {
    afio_cfg_debug_ports(AFIO_DEBUG_NONE);
}

/**
 * @brief Enable the JTAG and Serial Wire (SW) debug ports.
 *
 * After you call this function, the JTAG and SW debug pins will no
 * longer be usable as GPIOs.
 *
 * @see disableDebugPorts()
 */
static inline void enableDebugPorts(void) {
    afio_cfg_debug_ports(AFIO_DEBUG_FULL_SWJ);
}
```

5. Serial API

시리얼(USART) 관련 API는 Leaf Maple보드와 아두이노 API와 호환되므로 아래의 사이트에서 튜토리얼을 학습하시면 좋습니다. 참고로 CM-900 은 Leaf Maple보드와 같은 STM32 시리즈를 쓰기 때문에 Leaflabs사의 튜토리얼이 더욱 잘 맞습니다.

Leaflabs.com의 Serial Tutorial.
http://leaflabs.com/docs/lang/api/serial.html#lang-serial
Arduino Serial Tutorial
http://arduino.cc/en/Reference/Serial

CM900 은 총 3 개의 USART포트를 지원하고 각각 Serial1, Serial2, Serial3 로 활용하시면 됩니다.

```
/* Set up/tear down */
    void begin(uint32 baud);
    void end(void);
```

Example:

```
void setup() {
    Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
}

void loop() {}
```

```
    /* I/O */
    uint32 available(void);
    uint8 read(void);
    void flush(void);
    virtual void write(unsigned char);
    using Print::write;
```

write()의 경우 직접 쓰시기 보다는 상속 받은 print(), println()을 쓰시는 것이 더욱 편리합니다.

```
void setup(){
  Serial.begin(9600);
  Serial1.begin(38400);
  Serial2.begin(19200);
  Serial3.begin(4800);

  Serial.println("Hello Computer");
  Serial1.println("Hello Serial 1");
  Serial2.println("Hello Serial 2");
  Serial3.println("Hello Serial 3");
}
```

```
 /*
 * [ROBOTIS]2012-12-13 add to support making user interrupt function
 * @brief Attach/detach an interrupt handler to USART
 */
    void attachInterrupt(voidFuncPtrUart handler);
```

Serial 인터럽트를 연결하는 메소드입니다. Serial은 1byte씩 인터럽트를 처리하므로 반드시 아래와 같은 proto type으로 인터럽트 처리 핸들러를 만들어주시면 됩니다.

```
void interrupt_handler(byte buffer){}
```

```
    void detachInterrupt(void);
```

인터럽트 핸들러와 함수포인터의 연결을 끊습니다.

아래와 같이 구현하시면 됩니다.

```
void setup(){
  //USB Serial initialize
  Serial2.begin(57600);
  //You can attach your serial interrupt
  //or, also detach the interrupt by detachInterrupt(void) method
  Serial2.attachInterrupt(serialInterrupt);

}
//Serial Interrupt type must have the below proto-type
//void interrupt_name (byte variable){}
void serialInterrupt(byte buffer){
  Serial2.print((char)buffer);
}
void loop(){
  toggleLED();
  delay(50);

}
```

```
    /* Pin accessors */
    int txPin(void) { return this->tx_pin; }
    int rxPin(void) { return this->rx_pin; }
```

아래는 Serial2 를 활용한 Echo 예제 입니다.

```
void setup() {
    Serial2.begin(9600);
}

void echoCharacter() {
    // Check to see if we have received any information.  numUnread
    // will hold the number of bytes we've received, but haven't
    // looked at yet.
    int numUnread = Serial2.available();

    // numUnread > 0 means that there are some unread bytes waiting
    if (numUnread > 0) {
        // Read a single byte out:
        byte b = Serial2.read();
        // And then print it back:
        Serial2.print(b);
    }
}

void loop() {
    echoCharacter();
}
```

6. SerialUSB API

아래의 Leaflabs의 튜토리얼을 보시면 CM-900의 SerialUSB 를 쉽게 활용하실 수 있습니다.

http://leaflabs.com/docs/lang/api/serialusb.html#lang-serialusb

```
    void begin(void);
    void end(void);

    uint32 available(void);
    uint32 read(void *buf, uint32 len);
    uint8  read(void);

    void write(uint8);
    void write(const char *str);
    void write(const void*, uint32);
    /**
     * [ROBOTIS]2012-12-14 add to support making user interrupt function
     * */
    void attachInterrupt(voidFuncPtrUsb handler);
```

SerialUSB 인터럽트 핸들러는 반드시 아래와 같은 Proto-type으로 만들어주시고 attachInterrupt()메소드로 연결해주시면 됩니다. buffer는 넘어온 데이터를 가리키는 포인터이고 nCount는 몇 바이트가 넘어왔는지 전달하는 변수입니다.
USB의 경우 한 패킷 당 최대 전송 바이트 수는 64byte이므로 nCount가 64를 넘지 않습니다.

```
void interrupt_handler(byte* buffer, byte nCount){}
```

```
    void detachInterrupt(void);
```

인터럽트 핸들러의 연결을 해제합니다.

```
    uint8 getRTS();
    uint8 getDTR();
    uint8 isConnected(); //연결여부를 확인합니다. 연결되면 1, 아니면 0
    uint8 pending();
```

```

void setup(){
  //USB Serial initialize
  SerialUSB.begin();
  //You can attach your serialUSB interrupt
  //or, also detach the interrupt by detachInterrupt(void) method
  SerialUSB.attachInterrupt(usbInterrupt);
}
//SerialUSB Interrupt type must have the below proto-type
//void interrupt_name (byte* buffer, byte nCount){}
//USB max packet data is maximum 64byte, so nCount can not exceeds 64 bytes
int i=0;
void usbInterrupt(byte* buffer, byte nCount){
  SerialUSB.print("nCount =");SerialUSB.println(nCount);
  for(i=0; i < nCount;i++)
    SerialUSB.print((char)buffer[i]);
  SerialUSB.println("");
}
void loop(){
  toggleLED();
  delay(100);

}
```

7. Timer API

Timer관련 API는 Leaflabs의 HardwareTimer를 그대로 포팅하였으니 아래의 튜토리얼을 활용하십시요
http://leaflabs.com/docs/lang/api/hardwaretimer.html#lang-hardwaretimer

```
/**
     * @brief Construct a new HardwareTimer instance.
     * @param timerNum number of the timer to control.
     */
    HardwareTimer(uint8 timerNum);
```

타이머를 사용하기 위해서는 위의 Constructor를 활용해서 HardwareTimer를 초기화해야 합니다. TimerNum은 사용하고자 하는 타이머의 번호에 해당하고 CM900의 STM32F103C8은 4개의 일반적인 용도의 TIMER를 가지고 있고 각각의 TIMER마다 4개 채널씩 가지고 있습니다.

```
// Use timer 1
HardwareTimer timer(1);

void setup() {
   // Your setup code
}

void loop() {
   // ...
}
```

```
    /**
     * @brief Stop the counter, without affecting its configuration.
     *
     * @see HardwareTimer::resume()
     */
    void pause(void);

    /**
     * @brief Resume a paused timer, without affecting its configuration.
     *
     * The timer will resume counting and firing interrupts as
     * appropriate.
     *
     * Note that there is some function call overhead associated with
     * using this method, so using it in concert with
     * HardwareTimer::pause() is not a robust way to align multiple
     * timers to the same count value.
     *
     * @see HardwareTimer::pause()
     */
    void resume(void);

    /**
     * @brief Get the timer's prescale factor.
```

```
 * @return Timer prescaler, from 1 to 65,536.
 * @see HardwareTimer::setPrescaleFactor()
 */
uint32 getPrescaleFactor();

/**
 * @brief Set the timer's prescale factor.
 *
 * The new value won't take effect until the next time the counter
 * overflows.  You can force the counter to reset using
 * HardwareTimer::refresh().
 *
 * @param factor The new prescale value to set, from 1 to 65,536.
 * @see HardwareTimer::refresh()
 */
void setPrescaleFactor(uint32 factor);

/**
 * @brief Get the timer overflow value.
 * @see HardwareTimer::setOverflow()
 */
uint16 getOverflow();

/**
 * @brief Set the timer overflow (or "reload") value.
 *
 * The new value won't take effect until the next time the counter
 * overflows.  You can force the counter to reset using
 * HardwareTimer::refresh().
 *
 * @param val The new overflow value to set
 * @see HardwareTimer::refresh()
 */
void setOverflow(uint16 val);

/**
 * @brief Get the current timer count.
 *
 * @return The timer's current count value
 */
uint16 getCount(void);

/**
 * @brief Set the current timer count.
 *
 * @param val The new count value to set.  If this value exceeds
 *           the timer's overflow value, it is truncated to the
 *           overflow value.
 */
void setCount(uint16 val);

/**
 * @brief Set the timer's period in microseconds.
 *
 * Configures the prescaler and overflow values to generate a timer
 * reload with a period as close to the given number of
 * microseconds as possible.
 *
 * @param microseconds The desired period of the timer.  This must be
 *                    greater than zero.
 * @return The new overflow value.
 */
```

```
    uint16 setPeriod(uint32 microseconds);

    /**
     * @brief Configure a timer channel's mode.
     * @param channel Timer channel, from 1 to 4
     * @param mode Mode to set
     */
    void setMode(int channel, timer_mode mode);
```

channel은 총 4개가 있으며 아래와 같은 Constant를 쓰면 됩니다.
TIMER_CH1, TIMER_CH2, TIMER_CH3, TIMER_CH4

timer_mode enum

Used to configure the behavior of a timer channel.

Note that not all timers can be configured in every mode.

Values:

- `TIMER_DISABLED` -

  In this mode, the timer stops counting, channel interrupts are detached, and no state changes are output.

- `TIMER_PWM` -

  PWM output mode.

  This is the default mode for pins after initialization.

- `TIMER_OUTPUT_COMPARE` -

  In this mode, the timer counts from 0 to its reload value repeatedly; every time the counter value reaches one of the channel compare values, the corresponding interrupt is fired.

```
    /**
     * @brief Get the compare value for the given channel.
     * @see HardwareTimer::setCompare()
     */
    uint16 getCompare(int channel);

    /**
     * @brief Set the compare value for the given channel.
     *
     * @param channel the channel whose compare to set, from 1 to 4.
     * @param compare The compare value to set.  If greater than this
     *                timer's overflow value, it will be truncated to
     *                the overflow value.
     *
     * @see timer_mode
     * @see HardwareTimer::setMode()
     * @see HardwareTimer::attachInterrupt()
     */
    void setCompare(int channel, uint16 compare);

    /**
     * @brief Attach an interrupt handler to the given channel.
     *
     * This interrupt handler will be called when the timer's counter
     * reaches the given channel compare value.
     *
     * @param channel the channel to attach the ISR to, from 1 to 4.
     * @param handler The ISR to attach to the given channel.
     * @see voidFuncPtr
     */
    void attachInterrupt(int channel, voidFuncPtr handler);

    /**
     * @brief Remove the interrupt handler attached to the given
     *        channel, if any.
     *
     * The handler will no longer be called by this timer.
     *
     * @param channel the channel whose interrupt to detach, from 1 to 4.
     * @see HardwareTimer::attachInterrupt()
     */
    void detachInterrupt(int channel);

    /**
     * @brief Reset the counter, and update the prescaler and overflow
     *        values.
     *
     * This will reset the counter to 0 in upcounting mode (the
     * default).  It will also update the timer's prescaler and
     * overflow, if you have set them up to be changed using
     * HardwareTimer::setPrescaleFactor() or
     * HardwareTimer::setOverflow().
     *
     * @see HardwareTimer::setPrescaleFactor()
     * @see HardwareTimer::setOverflow()
     */
    void refresh(void);
```

```
#define LED_RATE 500000    // in microseconds; should give 0.5Hz toggles

// We'll use timer 2
HardwareTimer timer(2);

void setup() {
    // Set up the LED to blink
    pinMode(BOARD_LED_PIN, OUTPUT);

    // Pause the timer while we're configuring it
    timer.pause();
    // Set up period
    timer.setPeriod(LED_RATE); // in microseconds

    // Set up an interrupt on channel 1
    timer.setMode(TIMER_CH1, TIMER_OUTPUT_COMPARE);
    timer.setCompare(TIMER_CH1, 1);  // Interrupt 1 count after each update
    timer.attachInterrupt(TIMER_CH1, handler_led);

    // Refresh the timer's count, prescale, and overflow
    timer.refresh();

    // Start the timer counting
    timer.resume();
}

void loop() {
    // Nothing! It's all in the handler_led() interrupt:
}

void handler_led(void) {
    toggleLED();
}
```

8. SPI API

SPI역시 Leaflabs사의 HardwareSPI 튜토리얼을 참고하시면 됩니다.
http://leaflabs.com/docs/lang/api/hardwarespi.html#lang-hardwarespi

```
/**
     * @param spiPortNumber Number of the SPI port to manage.
     */
    HardwareSPI(uint32 spiPortNumber);
```

```
// Use SPI port number 1
HardwareSPI spi(1);

void setup() {
   // Your setup code
}

void loop() {
   // Do stuff with SPI
}
```

```
/**
     * @brief Turn on a SPI port and set its GPIO pin modes for use as master.
     *
     * SPI port is enabled in full duplex mode, with software slave management.
     *
     * @param frequency Communication frequency
     * @param bitOrder Either LSBFIRST (little-endian) or MSBFIRST (big-endian)
     * @param mode SPI mode to use, one of SPI_MODE_0, SPI_MODE_1,
     *             SPI_MODE_2, and SPI_MODE_3.
     */
    void begin(SPIFrequency frequency, uint32 bitOrder, uint32 mode);
```

Parameters:	`frequency` - Communication frequency `bitOrder` - Either LSBFIRST (little-endian) or MSBFIRST (big-endian) `mode` - SPI mode to use, one of SPI_MODE_0, SPI_MODE_1, SPI_MODE_2, and SPI_MODE_3.

The speed at which the SPI port communicates is configured using a `SPIFrequency` value:

SPIFrequency enum

Defines the possible SPI communication speeds.

Values:

- `SPI_18MHZ = 0` -

  18 MHz

- `SPI_9MHZ = 1` -

  9 MHz

- `SPI_4_5MHZ = 2` -

  4.5 MHz

- `SPI_2_25MHZ = 3` -

  2.25 MHz

- `SPI_1_125MHZ = 4` -

  1.125 MHz

- `SPI_562_500KHZ = 5` -

  562.500 KHz

- `SPI_281_250KHZ = 6` -

  281.250 KHz

- `SPI_140_625KHZ = 7` -

  140.625 KHz

Note

Due to hardware issues, you can’t use the frequency `SPI_140_625KHz` with SPI port 1.

The “mode” value determines the clock phase and polarity, like so:

spi_mode enum

SPI mode configuration.

Determines a combination of clock polarity (CPOL), which determines idle state of the clock line, and clock phase (CPHA), which determines which clock edge triggers data capture.

Values:

- `SPI_MODE_0` -

  Clock line idles low (0), data capture on first clock transition.

- `SPI_MODE_1` -

  Clock line idles low (0), data capture on second clock transition.

- `SPI_MODE_2` -

  Clock line idles high (1), data capture on first clock transition.

- `SPI_MODE_3` -

  Clock line idles high (1), data capture on second clock transition.

```
// Use SPI port number 1
HardwareSPI spi(1);
```

```
void setup() {
    // Turn on the SPI port
    spi.begin(SPI_18MHZ, MSBFIRST, 0);
}

void loop() {
   // Do stuff with SPI
}
```

```
    /**
     * @brief Equivalent to begin(SPI_1_125MHZ, MSBFIRST, 0).
     */
    void begin(void);

    /**
     * @brief Turn on a SPI port and set its GPIO pin modes for use as a slave.
     *
     * SPI port is enabled in full duplex mode, with software slave management.
     *
     * @param bitOrder Either LSBFIRST (little-endian) or MSBFIRST(big-endian)
     * @param mode SPI mode to use
     */
    void beginSlave(uint32 bitOrder, uint32 mode);

    /**
     * @brief Equivalent to beginSlave(MSBFIRST, 0).
     */
    void beginSlave(void);

    /**
     * @brief Disables the SPI port, but leaves its GPIO pin modes unchanged.
     */
    void end(void);

    /*
     * I/O
     */

    /**
     * @brief Return the next unread byte.
     *
     * If there is no unread byte waiting, this function will block
     * until one is received.
     */
    uint8 read(void);
```

```
// Use SPI port number 1
HardwareSPI spi(1);

void setup() {
    // Turn on the SPI port
    spi.begin(SPI_18MHZ, MSBFIRST, 0);
}

void loop() {
   // Send 245 over SPI, and wait for a response.
   spi.write(245);
   byte response = spi.read();
   // Print out the response received.
   SerialUSB.print("response: ");
   SerialUSB.println(response, DEC);
}
```

```
    /**
     * @brief Read length bytes, storing them into buffer.
     * @param buffer Buffer to store received bytes into.
     * @param length Number of bytes to store in buffer.  This
     *              function will block until the desired number of
     *              bytes have been read.
     */
    void read(uint8 *buffer, uint32 length);

    /**
     * @brief Transmit a byte.
     * @param data Byte to transmit.
     */
    void write(uint8 data);

    /**
     * @brief Transmit multiple bytes.
     * @param buffer Bytes to transmit.
     * @param length Number of bytes in buffer to transmit.
     */
    void write(const uint8 *buffer, uint32 length);

    /**
     * @brief Transmit a byte, then return the next unread byte.
     *
     * This function transmits before receiving.
     *
     * @param data Byte to transmit.
     * @return Next unread byte.
     */
    uint8 transfer(uint8 data);

    /*
     * Pin accessors
     */

    /**
     * @brief Return the number of the MISO (master in, slave out) pin
     */
    uint8 misoPin(void);

    /**
     * @brief Return the number of the MOSI (master out, slave in) pin
     */
    uint8 mosiPin(void);

    /**
     * @brief Return the number of the SCK (serial clock) pin
     */
    uint8 sckPin(void);
    /**
     * @brief Return the number of the NSS (slave select) pin
     */
    uint8 nssPin(void);

    /* -- The following methods are deprecated --------------------------- */

    /**
     * @brief Deprecated.
     *
     * Use HardwareSPI::transfer() instead.
     *
     * @see HardwareSPI::transfer()
     */
    uint8 send(uint8 data);

    /**
     * @brief Deprecated.
     *
     * Use HardwareSPI::write() in combination with
     * HardwareSPI::read() (or HardwareSPI::transfer()) instead.
     *
     * @see HardwareSPI::write()
     * @see HardwareSPI::read()
     * @see HardwareSPI::transfer()
     */
    uint8 send(uint8 *data, uint32 length);

    /**
     * @brief Deprecated.
     *
     * Use HardwareSPI::read() instead.
     *
     * @see HardwareSPI::read()
     */
    uint8 recv(void);
```

9. Time and delay functions

```
/**
 * Returns time (in milliseconds) since the beginning of program
 * execution. On overflow, restarts at 0.
 * @see micros()
 */
static inline uint32 millis(void) ;

/**
 * Returns time (in microseconds) since the beginning of program
 * execution.  On overflow, restarts at 0.
 * @see millis()
 */
static inline uint32 micros(void) ;

/**
 * Delay for at least the given number of milliseconds.
 *
 * Interrupts, etc. may cause the actual number of milliseconds to
 * exceed ms.  However, this function will return no less than ms
 * milliseconds from the time it is called.
 *
 * @param ms the number of milliseconds to delay.
 * @see delayMicroseconds()
 */
void delay(unsigned long ms);

/**
 * Delay for at least the given number of microseconds.
 *
 * Interrupts, etc. may cause the actual number of microseconds to
 * exceed us.  However, this function will return no less than us
 * microseconds from the time it is called.
 *
 * @param us the number of microseconds to delay.
 * @see delay()
 */
void delayMicroseconds(uint32 us);
```

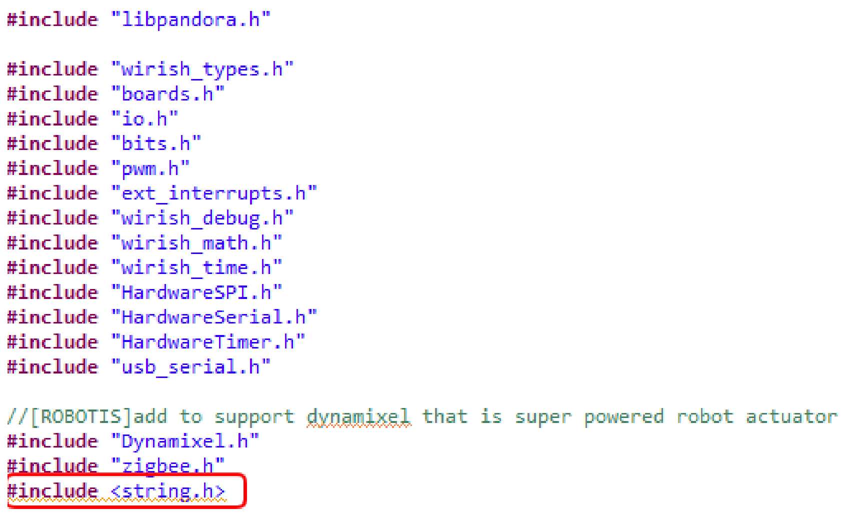
10. Math functions

```
/**
 * @brief Initialize the pseudo-random number generator.
 * @param seed the number used to initialize the seed; cannot be zero.
 */
void randomSeed(unsigned int seed);

/**
 * @brief Generate a pseudo-random number with upper bound.
 * @param max An upper bound on the returned value, exclusive.
 * @return A pseudo-random number in the range [0,max).
 * @see randomSeed()
 */
long random(long max);

/**
 * @brief Generate a pseudo-random number with lower and upper bounds.
 * @param min Lower bound on the returned value, inclusive.
 * @param max Upper bound on the returned value, exclusive.
 * @return A pseudo-random number in the range [min, max).
 * @see randomSeed()
 */
long random(long min, long max);

/**
 * @brief Remap a number from one range to another.
 *
 * That is, a value equal to fromStart gets mapped to toStart, a value
 * of fromEnd to toEnd, and other values are mapped proportionately.
 *
 * Does not constrain value to lie within [fromStart, fromEnd].
 *
 * If a "start" value is larger than its corresponding "end", the
 * ranges are reversed, so map(n, 1, 10, 10, 1) would reverse the
 * range [1,10].
 *
 * Negative numbers may appear as any argument.
 *
 * @param value the value to map.
 * @param fromStart the beginning of the value's current range.
 * @param fromEnd the end of the value's current range.
 * @param toStart the beginning of the value's mapped range.
 * @param toEnd the end of the value's mapped range.
 * @return the mapped value.
 */
static inline long map(long value, long fromStart, long fromEnd,
              long toStart, long toEnd) {
    return (value - fromStart) * (toEnd - toStart) / (fromEnd - fromStart) +
        toStart;
}

#define PI         3.1415926535897932384626433832795
#define HALF_PI     1.5707963267948966192313216916398
#define TWO_PI      6.283185307179586476925286766559
#define DEG_TO_RAD  0.017453292519943295769236907684886
#define RAD_TO_DEG 57.295779513082320876798154814105

#define min(a,b)              ((a)<(b)?(a):(b))
#define max(a,b)              ((a)>(b)?(a):(b))
#define constrain(amt,low,high) ((amt)<(low)?(low):((amt)>(high)?(high):(amt)))

#define round(x)              ((x)>=0?(long)((x)+0.5):(long)((x)-0.5))
#define radians(deg)           ((deg)*DEG_TO_RAD)
#define degrees(rad)           ((rad)*RAD_TO_DEG)
#define sq(x)                 ((x)*(x))
```

11. C/C++ Standard Library

#include <string.h>가 미리 선언되어 있으므로 아래의 String처리 함수를 모두 사용하실 수 있습니다.

C:₩ROBOTIS₩hardware₩robotis₩cores₩robotis₩wirish.h

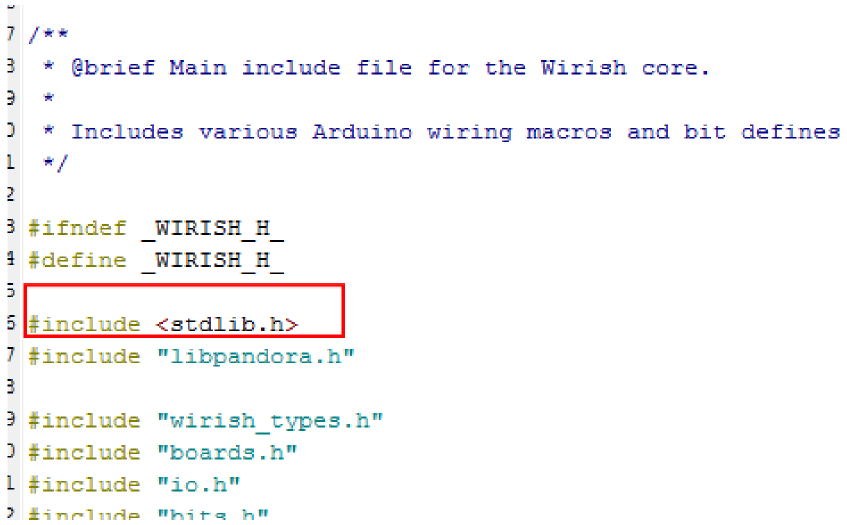


```
#include "libpandora.h"

#include "wirish_types.h"
#include "boards.h"
#include "io.h"
#include "bits.h"
#include "pwm.h"
#include "ext_interrupts.h"
#include "wirish_debug.h"
#include "wirish_math.h"
#include "wirish_time.h"
#include "HardwareSPI.h"
#include "HardwareSerial.h"
#include "HardwareTimer.h"
#include "usb_serial.h"

//[ROBOTIS]add to support dynamixel that is super powered robot actuator
#include "Dynamixel.h"
#include "zigbee.h"
#include <string.h>
```

```
_PTR    _EXFUN(memchr,(const _PTR, int, size_t));
int     _EXFUN(memcmp,(const _PTR, const _PTR, size_t));
_PTR    _EXFUN(memcpy,(_PTR, const _PTR, size_t));
_PTR    _EXFUN(memmove,(_PTR, const _PTR, size_t));
_PTR    _EXFUN(memset,(_PTR, int, size_t));
char    *_EXFUN(strcat,(char *, const char *));
char    *_EXFUN(strchr,(const char *, int));
int      _EXFUN(strcmp,(const char *, const char *));
int      _EXFUN(strcoll,(const char *, const char *));
char    *_EXFUN(strcpy,(char *, const char *));
size_t   _EXFUN(strcspn,(const char *, const char *));
char    *_EXFUN(strerror,(int));
size_t   _EXFUN(strlen,(const char *));
char    *_EXFUN(strncat,(char *, const char *, size_t));
int      _EXFUN(strncmp,(const char *, const char *, size_t));
char    *_EXFUN(strncpy,(char *, const char *, size_t));
char    *_EXFUN(strpbrk,(const char *, const char *));
char    *_EXFUN(strrchr,(const char *, int));
size_t   _EXFUN(strspn,(const char *, const char *));
char    *_EXFUN(strstr,(const char *, const char *));

#ifndef _REENT_ONLY
char    *_EXFUN(strtok,(char *, const char *));
#endif

size_t   _EXFUN(strxfrm,(char *, const char *, size_t));

#if !defined __STRICT_ANSI__ && !defined _AEABI_PORTABLE
char    *_EXFUN(strtok_r,(char *, const char *, char **));

int      _EXFUN(bcmp,(const void *, const void *, size_t));
void     _EXFUN(bcopy,(const void *, void *, size_t));
void     _EXFUN(bzero,(void *, size_t));
int      _EXFUN(ffs,(int));
char    *_EXFUN(index,(const char *, int));
_PTR    _EXFUN(memccpy,(_PTR, const _PTR, int, size_t));
_PTR    _EXFUN(mempcpy,(_PTR, const _PTR, size_t));
_PTR    _EXFUN(memmem, (const _PTR, size_t, const _PTR, size_t));
char    *_EXFUN(rindex,(const char *, int));
char    *_EXFUN(stpcpy,(char *, const char *));
char    *_EXFUN(stpncpy,(char *, const char *, size_t));
int      _EXFUN(strcasecmp,(const char *, const char *));
char    *_EXFUN(strcasestr,(const char *, const char *));
char    *_EXFUN(strdup,(const char *));
char    *_EXFUN(_strdup_r,(struct _reent *, const char *));
char    *_EXFUN(strndup,(const char *, size_t));
char    *_EXFUN(_strndup_r,(struct _reent *, const char *, size_t));
char    *_EXFUN(strerror_r,(int, char *, size_t));
size_t   _EXFUN(strlcat,(char *, const char *, size_t));
size_t   _EXFUN(strlcpy,(char *, const char *, size_t));
```

```
int       _EXFUN(strncasecmp,(const char *, const char *, size_t));
size_t    _EXFUN(strnlen,(const char *, size_t));
char     *_EXFUN(strsep,(char **, const char *));
char     *_EXFUN(strlwr,(char *));
char     *_EXFUN(strupr,(char *));
```

atoi()같은 stdlib.h에 포함된 함수를 쓰고 싶으시면 방금전 wirish.h의 맨 탑에(반드시) #include <stdlib.h>를 추가하시면 됩니다.



```
/**
 * @brief Main include file for the Wirish core.
 *
 * Includes various Arduino wiring macros and bit defines
 */

#ifndef _WIRISH_H_
#define _WIRISH_H_

#include <stdlib.h>
#include "libpandora.h"

#include "wirish_types.h"
#include "boards.h"
#include "io.h"
#include "bits.h"
```

stdlib.h나 stdio.h는 반드시 소스코드의 맨위에 선언하도록 한다.