```
/*
 *     SocialLedge.com - Copyright (C) 2013
 *
 *     This file is part of free software framework for embedded processors.
 *     You can use it and/or distribute it as long as this copyright header
 *     remains unmodified.  The code is free for personal use and requires
 *     permission to use in a commercial product.
 *
 *      THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS,
  IMPLIED
 *      OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 *      MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS
  SOFTWARE.
 *      I SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
  OR
 *      CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 *     You can reach the author of this software at :
 *          p r e e t . w i k i @ g m a i l . c o m
 */

#include "i2c2.hpp"
#include "LPC17xx.h"



/**
 * IRQ Handler needs to be enclosed in extern "C" because this is C++ file, and
 * we don't want C++ to "mangle" this function name.
 * This ISR Function need needs to be named precisely to override "WEAK" ISR
 * handler defined at startup.cpp
 */
extern "C"
{
    void I2C2_IRQHandler()
    {
        I2C2::getInstance().handleInterrupt();
    }
}



bool I2C2::init(unsigned int speedInKhz)
{
    /**
     * Before I2C is initialized, check to be sure that the I2C wires are logic
      "1"
     * means that they are pulled high, otherwise there may be a short circuit.
     *
     * I2C2 is on P0.10, and P0.11
     */
```

```
    const uint32_t i2c_pin_mask = ( (1<<10) | (1<<11) );
    const bool i2c_wires_are_pulled_high = (i2c_pin_mask == (LPC_GPIO0->FIOPIN
     & i2c_pin_mask) );

    LPC_PINCON->PINMODE0 &= ~(0xF << 20); // Both pins with Pull-Up Enabled
    LPC_PINCON->PINMODE0 |=  (0xA << 20); // Disable both pull-up and pull-down

    // Enable Open-drain for I2C2 on pins P0.10 and P0.11
    LPC_PINCON->PINMODE_OD0 |= i2c_pin_mask;

    LPC_PINCON->PINSEL0 &= ~(0xF << 20);  // Clear
    LPC_PINCON->PINSEL0 |=  (0xA << 20);  // Enable I2C Pins: SDA, SCL

    lpc_pclk(pclk_i2c2, clkdiv_8);
    const uint32_t pclk = sys_get_cpu_clock() / 8;

    /**
     * I2C wires should be pulled high for normal operation, so if they are,
      initialize I2C
     * otherwise disable operations on I2C since I2C has a likely hardware BUS
      fault such as:
     *  - I2C SDA/SCL with no pull-up
     *  - I2C SDA/SCL shorted to ground
     */
    if (i2c_wires_are_pulled_high) {
        return I2C_Base::init(pclk, speedInKhz);
    }
    else {
        disableOperation();
        return false;
    }
}

__attribute__ ((weak)) bool I2C2::initSlave(const uint8_t slaveAddr, volatile
 uint8_t * buffer, uint32_t buffer_size)
{
  //Initialize slave
  const uint8_t slave_init_magic = 0x44;

  // Set the slave address
  LPC_I2C2->I2ADR0  = slaveAddr;
  LPC_I2C2->I2CONSET= slave_init_magic;

  this->slave_buffer = buffer;
  this->slave_buffer_size = buffer_size;
  this->offset = 0;

  return true;
}

I2C2::I2C2() : I2C_Base((LPC_I2C_TypeDef*) LPC_I2C2_BASE)
```

```
{

}
```