```python
#!/usr/bin/python
import re
import smbus2 as smbus

# ===========================================================================
# Raspi_I2C Class
# ===========================================================================

class Raspi_I2C(object):

  @staticmethod
  def getPiRevision():
    "Gets the version number of the Raspberry Pi board"
    # Revision list available at: http://elinux.org/RPi_HardwareHistory#Board_Revision_History
    try:
      with open('/proc/cpuinfo', 'r') as infile:
        for line in infile:
          # Match a line of the form "Revision : 0002" while ignoring extra
          # info in front of the revsion (like 1000 when the Pi was over-volted).
          match = re.match('Revision\s+:\s+.*(\w{4})$', line)
          if match and match.group(1) in ['0000', '0002', '0003']:
            # Return revision 1 if revision ends with 0000, 0002 or 0003.
            return 1
          elif match:
            # Assume revision 2 if revision ends with any other 4 chars.
            return 2
        # Couldn't find the revision, assume revision 0 like older code for compatibility.
        return 0
    except:
      return 0

  @staticmethod
  def getPiI2CBusNumber():
    # Gets the I2C bus number /dev/i2c#
    return 1 if Raspi_I2C.getPiRevision() > 1 else 0

  def __init__(self, address, busnum=-1, debug=False):
    self.address = address
    # By default, the correct I2C bus is auto-detected using /proc/cpuinfo
    # Alternatively, you can hard-code the bus version below:
    # self.bus = smbus.SMBus(0); # Force I2C0 (early 256MB Pi's)
    # self.bus = smbus.SMBus(1); # Force I2C1 (512MB Pi's)
    self.bus = smbus.SMBus(busnum if busnum >= 0 else Raspi_I2C.getPiI2CBusNumber())
    self.debug = debug

  def reverseByteOrder(self, data):
    "Reverses the byte order of an int (16-bit) or long (32-bit) value"
    # Courtesy Vishal Sapre
    byteCount = len(hex(data)[2:].replace('L','')[::2])
    val       = 0
    for i in range(byteCount):
      val    = (val << 8) | (data & 0xff)
      data >>= 8
    return val

  def errMsg(self):
    print( "Error accessing 0x%02X: Check your I2C address" % self.address)
    return -1

  def write8(self, reg, value):
    "Writes an 8-bit value to the specified register/address"
    try:
      self.bus.write_byte_data(self.address, reg, value)
      if self.debug:
        print( "I2C: Wrote 0x%02X to register 0x%02X" % (value, reg))
    except IOError as err:
      return self.errMsg()

  def write16(self, reg, value):
    "Writes a 16-bit value to the specified register/address pair"
    try:
      self.bus.write_word_data(self.address, reg, value)
      if self.debug:
        print ("I2C: Wrote 0x%02X to register pair 0x%02X,0x%02X" %
         (value, reg, reg+1))
    except IOError as err:
      return self.errMsg()

  def writeRaw8(self, value):
    "Writes an 8-bit value on the bus"
    try:
      self.bus.write_byte(self.address, value)
      if self.debug:
        print ("I2C: Wrote 0x%02X" % value)
    except IOError as err:
      return self.errMsg()

  def writeList(self, reg, list):
    "Writes an array of bytes using I2C format"
    try:
      if self.debug:
        print("I2C: Writing list to register 0x%02X:" % reg)
        print(list)
      self.bus.write_i2c_block_data(self.address, reg, list)
    except IOError as err:
      return self.errMsg()

  def readList(self, reg, length):
    "Read a list of bytes from the I2C device"
    try:
      results = self.bus.read_i2c_block_data(self.address, reg, length)
      if self.debug:
        print ("I2C: Device 0x%02X returned the following from reg 0x%02X" %
         (self.address, reg))
        print (results)
      return results
    except IOError as err:
      return self.errMsg()

  def readU8(self, reg):
    "Read an unsigned byte from the I2C device"
    try:
      result = self.bus.read_byte_data(self.address, reg)
      if self.debug:
        print ("I2C: Device 0x%02X returned 0x%02X from reg 0x%02X" %
         (self.address, result & 0xFF, reg))
      return result
    except IOError as err:
      return self.errMsg()

  def readS8(self, reg):
    "Reads a signed byte from the I2C device"
    try:
      result = self.bus.read_byte_data(self.address, reg)
      if result > 127: result -= 256
      if self.debug:
        print ("I2C: Device 0x%02X returned 0x%02X from reg 0x%02X" %
         (self.address, result & 0xFF, reg))
      return result
    except IOError as err:
      return self.errMsg()

```

```python
    def readU16(self, reg, little_endian=True):
      "Reads an unsigned 16-bit value from the I2C device"
      try:
        result = self.bus.read_word_data(self.address,reg)
        # Swap bytes if using big endian because read_word_data assumes little
        # endian on ARM (little endian) systems.
        if not little_endian:
          result = ((result << 8) & 0xFF00) + (result >> 8)
        if (self.debug):
          print ("I2C: Device 0x%02X returned 0x%04X from reg 0x%02X" % (self.address,
          result & 0xFFFF, reg) )
        return result
      except IOError as err:
        return self.errMsg()

    def readS16(self, reg, little_endian=True):
      "Reads a signed 16-bit value from the I2C device"
      try:
        result = self.readU16(reg,little_endian)
        if result > 32767: result -= 65536
        return result
      except IOError as err:
        return self.errMsg()

  if __name__ == '__main__':
    try:
      bus = Raspi_I2C(address=0)
      print("Default I2C bus is accessible")
    except:
      print("Error accessing default I2C bus")
```