C Compiler • Real-Time OS • Simulator • Training • Evaluation Boards

Using and Programming the I²C BUS — Application Note 153

June 8, 2000, Munich, Germany

by Keil Support, Keil Elektronik GmbH support.intl@keil.com ++49 89 456040-0

This Application Note describes programming of the I²C Bus for various devices. Included is the debugging and simulation of I²C Bus Applications with the Keil µVision2 Debugger/Simulator.

Definitions and Documents

The I²C Bus is a two-wire BUS system defined by Philips beginning of the 1980's. The I2C Bus is a bi-directional and designed for simple but efficient control applications. It is widely used in embedded systems to interface a microcontroller with peripherals.

The system is comprised of two lines, SCL (serial clock) and SDA (serial data) that carry information between the IC's connected to them. All devices connected to the bus can be master or slave devices. Each device can be in one of the following modes:

- **Idle Mode**: device is in high-impedance state and waits for data.
- **Master Transmitter Mode:** the device transmits data to a slave receiver.
- **Master Receiver Mode:** the device receives data from a slave transmitter.
- **Slave Receiver Mode:** a number of data bytes are received from a master transmitter.
- **Slave Transmitter Mode:** a number of data bytes are transmitted to a master receiver.

The I²C standard is described in the ***I²C BUS SPECIFICATION*** that is available at the Philips web page or the Keil Development Tools CD-ROM in the folder **Datashts\Philips**.

I²C Concepts

This application note describes the implementation of the I²C bus on 8051, 251 and 166 based devices. The bus can be implemented in several different ways on a device.

- **I²C serial port with hardware implemented Master and Slave functions** (as in Philips 80C552, 558, ect and serveral AtmelWM devices)
- **Combined SPI/I²C interface with hardware implemented Slave and software based Master functions** (as in Analog Devices ADuC812, ADuC824 and several other devices).
- **Single bit hardware for software based Master and Slave support** (as in Philips 8xC75x and Philips LPC series, described in Philips AN422 available from the Philips Web page or the Keil Development Tools CD-ROM folder **AppNotes\Philips**).

- Using the High-Speed Serial Interface of the Infineon 166/ST10 family for simulation of the I2C bus. (Described in the Infineon AppNote AP1626 (RK has it)).
- **Software based simulation of a I2C Bus Master device.** This can be implemented in any 8051 or 166 device be using two un-used I/O pins as SCL and SDA pins. These I/O pins are controlled by software only.

I²C Simulation

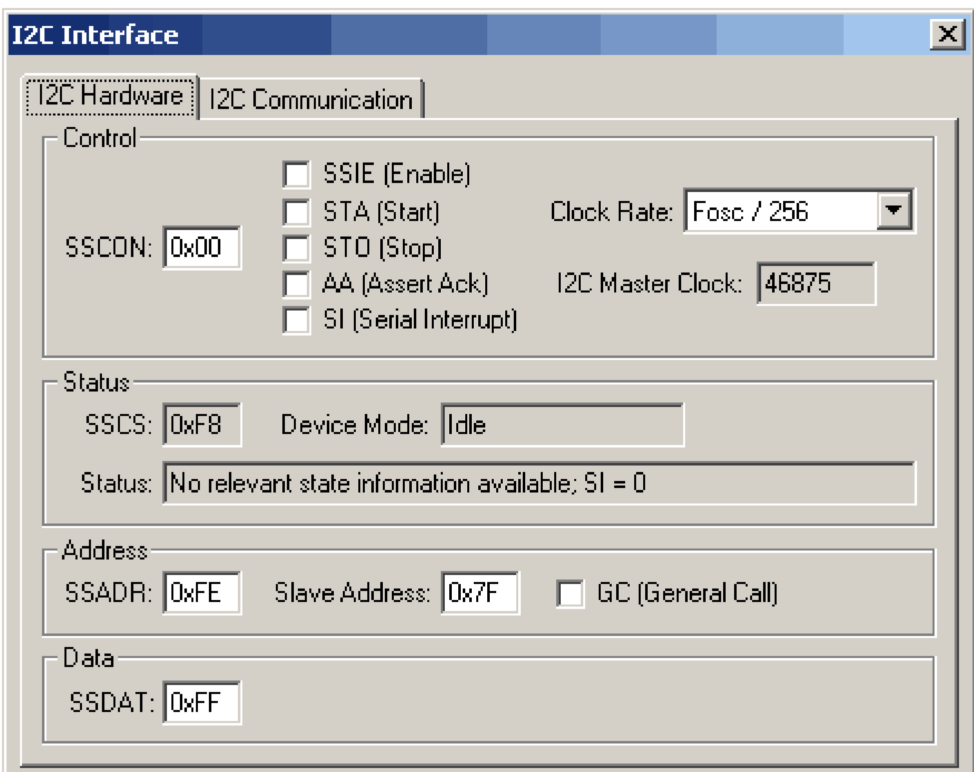
For efficient software testing is it not enough just to simulate the behavior of the I2C bus at bit-level. Instead it must be possible to simulate also bus communication. Therefore µVision allows you:

- to review the bus activities and create data on the I²C bus in the I²C communication dialog.
- virtual simulation registers (VTREG) the can be used to review and enter data to the I²C bus.
- and write debug functions that simulate a device that is connected to the microcontroller. In this way you can simulate your complete application rather than just a small piece of the bus communication.

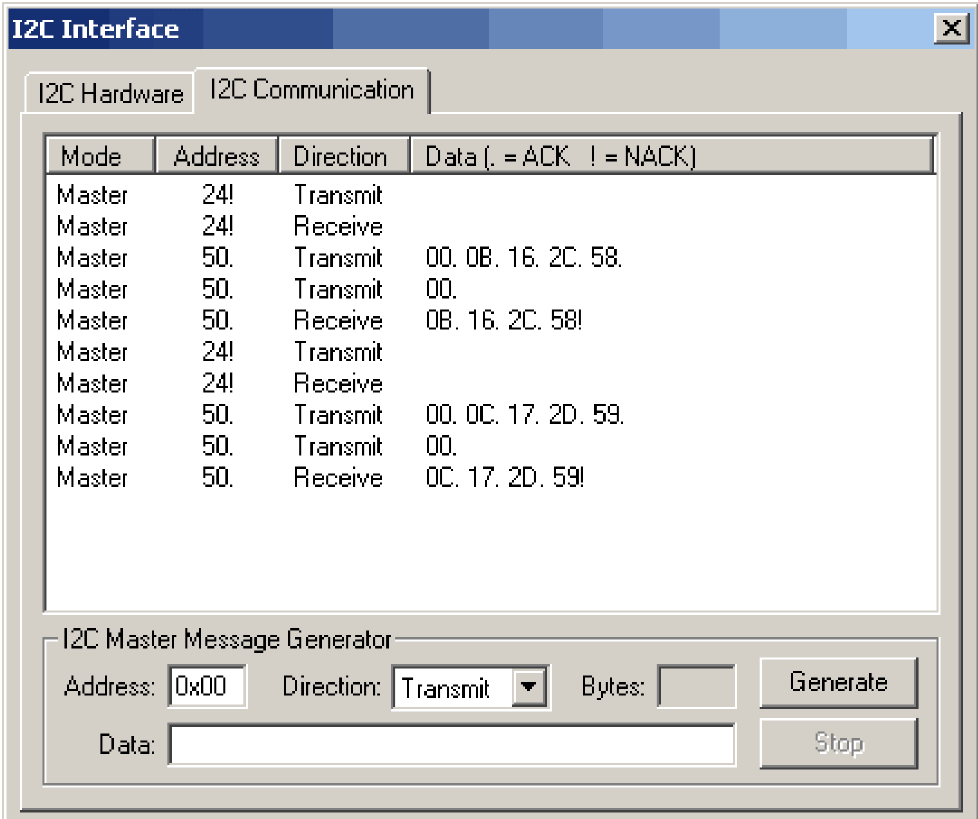
I²C Dialog

µVision2 offers an tabbed dialog that shows you on the first page the controls and status of the I²C interface and on the 2nd page a communication (similar to the CAN communication page see AN147).

I²C Hardware: this page allows you to review and modify the I²C settings trough hardware registers and to show the current I²C Interface status



I^{2}C Communication: this page allows you to review the data communication on the I^{2}C bus and to directly enter data on the I^{2}C bus using the Message Generator



Virtual Simulation Registers (VTREG)

The µVision2 Debugger implements virtual simulation registers (VTREG) that can be used to review the I^{2}C communication on the Debugger command line level or within Debug and Signal functions. The following registers are implemented:

VTREG	Description
I2C_IN	Data sent from the I^{2}C peripherals to the the Microcontroller: Possible values in this register are: 0xFFFF for IDLE or STOP condition 0x0100 for START, initiates SLAVE transmit or receive on the microcontroller; next byte is slave address 0x00 .. 0xFF any address or data byte transfer to the microcontroller. 0xFF00 for ACK 0xFF01 for NACK
I2C_OUT	Data sent from the Microcontroller to the I^{2}C peripherals. Possible values in this register are: 0xFFFF for IDLE or STOP condition 0x0100 for START, initiates MASTER transmit or receive on the microcontroller; next byte is slave address 0x00 .. 0xFF any address or data byte transfer to the I^{2}C peripherals. 0xFF00 for ACK 0xFF01 for NACK
I2C_CLK	Clock Frequency in Slave Mode in Hz, i.e. 100000 for 100KHz transmission

µVision supports only the 8-bit address mode of the I2C bus. The 11-bit address modes are currently not implemented and also not supported by the most microcontroller devices.

Simulating a Device connected to the I²C Bus

With specific signal functions the user can implement hardware components that are connected to the I²C bus. The following example shows a signal function that simulates an I²C Memory (256 bytes) like the Philips PCF8570.

The I²C Memory Slave address is set trough the SADR variable. Example:

```
SADR = 0x3F   // I2C Memory Slave Address
```

The signal function is invoked from the command window as:

```
I2Cmemory()
```

The I²C Memory is mapped to the memory region V:0 .. V:0xFF.

Once the simulator detects a START condition in the I2C_OUT VTReg, the next byte will be interpreted as address byte. This address byte contains the 7-bit Slave address in bits 7 .. 1 and in bit 0 the direction (0 = Write, 1 = Read). If the Slave Memory is addressed the Memory sends an ACK back to the Microcontroller. If the direction bit was "1" (Memory Read) the Microcontroller reads data bytes from the Memory (from the current address which is automatically incremented after each read byte) trough the I2C_IN VTReg. Microcontorller sends an ACK to the Memory after each byte if more data bytes should be read or an NACK if this is the last data byte read. If the direction bit was "0" (Memory Write) the Microcontroller sends first a byte with the new Memory address (Memory must return an ACK) and then sends data bytes which will be written to the Memory (to current address which is auto incremented after each written byte). Memory must return an ACK after each received byte.

```
// Simulation of I2C Memory (Slave): like Philips PCF8570 (256 byte I2C RAM)

MAP V:0,V:0xFF READ WRITE                    // Map User Memory region

DEFINE int SADR                              // Slave Address

signal void I2CMEMORY (void) {
  unsigned long adr;

  adr = V:0;
  while (1) {
    wwatch (I2C_OUT);                        // Wait for data from Microcontroller
    while (I2C_OUT == 0x0100) {              // START detected
      wwatch (I2C_OUT);                      // Wait for data from Microcontroller
      if (I2C_OUT > 0xFF) continue;
      if ((I2C_OUT >> 1) != SADR) continue;  // test if Slave is addressed
      I2C_IN = 0xFF00;                       // ACK to Microcontroller
      if (I2C_OUT & 1) {                     // Slave Read
        while (1) {
          I2C_IN = _RBYTE(adr);              // Read Byte from Memory
          adr++;                             // Increment Address
          wwatch (I2C_OUT);                  // Wait for ACK from Microcontroller
          if (I2C_OUT != 0xFF00) break;
        }
      }
      else {                                 // Slave Write
        wwatch (I2C_OUT);                    // Wait for data from Microcontroller
        if (I2C_OUT > 0xFF) continue;
        adr = I2C_OUT | V:0;                 // Set Memory Address
        I2C_IN = 0xFF00;                     // ACK to Microcontroller
        while (1) {
          wwatch (I2C_OUT);                  // Wait for data from Microcontroller
          if (I2C_OUT > 0xFF) break;
          _WBYTE (adr, I2C_OUT);             // Store Byte in Memory
          adr++;                             // Increment Address
          I2C_IN = 0xFF00;                   // ACK to Microcontroller
        }
      }
    }
  }
}
```

Application Examples

I^2C serial port with hardware implemented Master and Slave functions

Enclosed is an example that shows you how to use the I^2C bus with a Byte orientated Hardware. As example CPU we have used a Philips 8xC591 device.

The µVision2 project "I2CEEPROM" includes a 591 I^2C demonstration of reading and writing to a serial EEPROM (for the Phytec Development Board with 87C591 phyCORE module) and also the signal function that simulates an I^2C EEPROM Memory (4k bytes).

Single-bit Hardware for Software-Based Master and Slave support

Enclosed is an example that shows you how to use the I^2C bus with a Single Bit Hardware. As example CPU we have used a Philips LPC device.

The µVision2 project "Master" includes I^2C Single Master Routines for the 87LPC764 (taken from Philips AN422 - 8XC751 as I^2C Bus Master) and also the signal function that simulates an I^2C Memory (256 bytes).