IMPLEMENTING THE I²C COMMUNICATION PROTOCOL IN LABVIEW

E. Lunca, A. Salceanu, M. Cretu

Faculty of Electrical Engineering, Bd. Dimitrie Mangeron 53, 700050, Iasi, Romania,
+40.232.237.627/1246, elunca@ee.tuiasi.ro

Abstract- The I²C (Inter-Integrated Circuit) Bus is a two-wire, low to medium speed, communication bus developed by Philips Semiconductors in the early 1980s to reduce the manufacturing costs of electronic products. This paper discusses how to implement the I²C communication protocol using National Instruments LabVIEW, the most popular programming environment for measurement and control applications. By using a simple interface circuitry for PC's parallel port, a set of specific VIs (virtual instruments) has been created to enable testing of various devices or quickly developing PC-based data logging and recording solutions.

I. Introduction

Today, the I²C can be found in a wide variety of applications including microcontrollers, LCD, memory, keyboards, PCs, cell phones, TVs, etc. The I²C bus requires only two bidirectional signals: a serial data line (SDA) and a serial clock line (SCL). Each device connected to the bus is software-addressable by a unique address, used to identify the device in communication, and simple master-slave relationships exist at all time on the bus. The protocol comprises a set of conditions to establish or terminate communication, a designation to read or write and the capability to address devices with an expanded address scheme. Further information regarding the I²C protocol can be found in [1].

II. Hardware requirements

For implementing the I²C communication protocol in LabVIEW, the only hardware required to be used in conjunction with the specific VIs consists of a 2-wire parallel port adapter developed around the 74LS05 IC, which contains six inverters with open-collector outputs. It should be noted that a 74HC05 hex inverter with open-drain outputs will also work; several parallel port interfacing solutions based on these ICs are described in [2], [3] and [4].

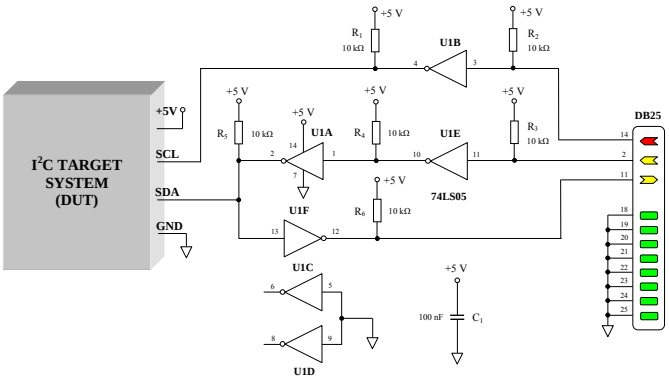

Figure 1. Simplified 2-wire adapter for PC's parallel port

Because the I²C interface uses only two signals, serial clock (SCL) and serial data (SDA), the bidirectional communication between the PC and DUT (device under test) is achieved using three pins of the parallel port. The SCL signal is entered only with respect to the DUT and is generated using the pin 14 of the parallel port DB-25 connector (Auto Linefeed), configured as an output. On the other hand, the SDA signal is bidirectional, being implemented using one dedicated input (pin 11 of the parallel port DB-25 connector) and one dedicated output (D0, pin 2 of the parallel port DB-25 connector). It should be noted that before reading any data from the DUT, the developed LabVIEW software releases the SDA line by forcing D0 high.

Depending on the application characteristics, an enhanced interface can be implemented by adding supplementary protection for the interface circuitry as well as the parallel port itself, but such designs will also require more ICs like circuit-protectors or DC-DC regulators.

III. Designing and using the I²C communication software

The communication between the PC (master device) and connected I²C target system (slave device) is achieved by only three bits residing in the three 8-bit registers of the parallel port, LPT1. The default base address for LPT1 is typically 378h, also corresponding to the Data Register (Byte), which contains the D0 bit, responsible for controlling the pin 2 and therefore the SDA input data for the target system. Complementary, the SDA output data are retrieved at pin 11 by reading the bit 7 (Busy) in the Status Byte, existing at the location base address+1. Finally, the bit 1 in the Control Byte, at location base address+2, is responsible for generating the SCL signal which is only entered with respect to the target system.

For implementing the I²C protocol in LabVIEW, a collection of high-level functions that conform to the main events on the bus has been created. The I²C LabVIEW library in Figure 2 includes the following VIs (virtual instruments):

- *I²C Start.vi* is used to generate the start condition, which is necessary prior to any transaction on the bus. This VI defaults to the LPT address of 378h, so you can let the VI's input unwired. However, if you determine that the software is unable to communicate with the I²C target hardware, then another address should be specified at the LPT Port input. The reference output of this function can be passed to other VIs in the I²C library.
- *I²C Send.vi* serves for transmitting messages to the I²C slave device. Typically, the input of this function is an 8-bit array, so it can be used, for example, to send a 7-bit address followed by the read/write bit, which is added into the array as the LSB. However, the function input can be resized to transmit even a single bit such as an acknowledge or NOT acknowledge.
  When using 10-bit addressing schemes, two *I²C Send* functions must be used because the 10-bit addresses are split in two 8-bit packets. The first byte will contain five default bits, 11110, two bits of the 10-bit address and the read/write bit.
- *I²C Read ACK.vi* is used for reading the acknowledge bit issued by the addressed device.
- *I²C Read 1 Byte.vi* reads a byte of serial data from the addressed device, MSB first. When calling this function, its output must be passed to the *I²C Send* VI, which will be used for sending either an acknowledge or NOT acknowledge bit.
- *I²C Read 2 Bytes.vi* reads two bytes of serial data from the addressed device. The function automatically sends ACK and NACK after receiving MSB data byte and LSB data byte, respectively.
- *I²C Stop.vi* is used to generate the stop condition, which is necessary after each data transfer.
- Other VIs can be developed and added into the I²C library in order to satisfy a specific requirement. For example, the first three functions in Figure 2 were implemented to convert the two's complement output code of an analog-to-digital converter to a decimal value.

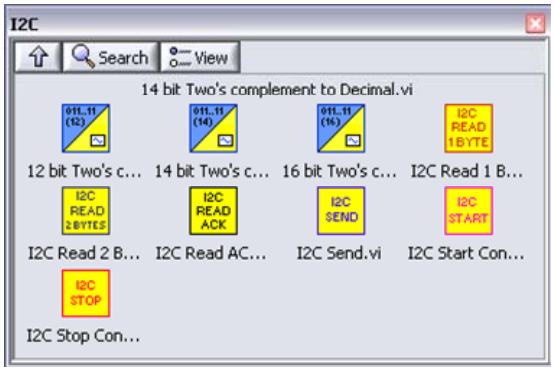

Figure 2. I²C LabVIEW library

All I²C functions in the library were developed using the LabVIEW *Out Port* and *In Port* VIs that directly write / read a data byte to / from the parallel port's hardware registers. For example, to force SDA line high, it is necessary to write the value 00000001 to the base address.

Figure 3 shows the block diagram of the *I²C Read 2 Bytes* function, which uses two for loops for reading two data bytes. As depicted in figure below, using a *Number to Boolean Array* function, the LPT Status Byte, provided by the *In Port* VI at its output, is converted to a Boolean array of 8 elements, in which the element 0 corresponds to the least significant bit. The *Index Array* VI returns only the element corresponding to the bit of interest in the Status Byte (bit 7, MSB) and then, after the last iteration of the for loop, at the output tunnel on the loop border, a new Boolean array is obtained, its elements corresponding to the MSB data byte from the I²C device. Finally, a *Boolean Array to Number* VI converts the new Boolean array to a 32-bit unsigned integer by interpreting the array as the two's complement of the binary representation of an integer, with the first element of the array being the least significant bit. For this reason, the correct reception of the MSB data byte is ensured by a prior *Reverse 1D Array* function, which reverses the order of the elements in array. Obviously, the LSB data byte from the I²C device is received in the same way.

The *I²C Read 2 Bytes* VI will automatically transmit an acknowledge bit after the first byte read and a not acknowledge bit after the second byte read, accordingly to the I²C protocol specifications. It is very important to point out that, in order for the addressed device to be able to communicate back to the application software, this function will force D0 high before any reading, allowing the I²C device to toggle the SDA line.

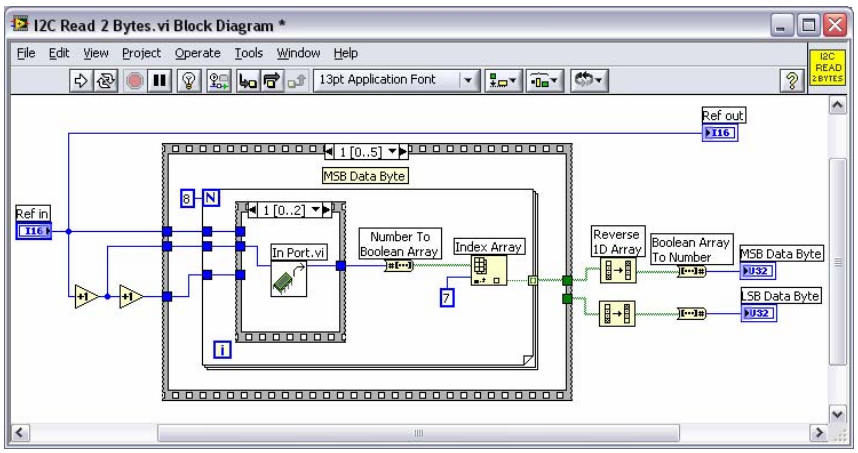

Figure 3. The block diagram of the *I²C Read 2 Bytes* VI

Figure 4 shows an interfacing example with MAX128, a 12-bit Data Acquisition System from Dallas Semiconductor. Basically, a complete I²C communication flow is based on a sequence structure with two frames, first of them corresponding to the write cycle and the second to the read cycle.

The first frame begins with a start condition. Then, using an *I²C Send* function, the device is addressed with its 7-bit address, followed by a write bit (*R/W=0*). After the reception of the acknowledge bit issued by the MAX128, with a *I²C Read ACK* function, a new *I²C Send* VI is used for transmitting the input control (setup) byte, which consists of a START bit (first logic "1" received after acknowledge of a write bit), three bits for selecting the desired "ON" channel, two bits for range and polarity selection and two bits for setting the power-down mode. Finally, the first frame ends with the reception of a new acknowledgement from the device and the generation of a stop condition.

In conformity with the I²C protocol, the second frame, not shown in Figure 4, begins again with a start condition, followed by the 7-bit address and a read bit (*R/W=1*). After receiving the acknowledgement from the MAX128, the MSB data byte (D11-D4, MSB first) and LSB data byte (D3-D0 and four zeros) are read using the *I²C Read 2 Bytes* VI. Obviously, the second frame ends with a stop condition, bringing the clock and data lines high.

Because the output data coding for the MAX128 is binary in unipolar mode and two's complement binary in bipolar mode, in the last case we use a specific VI to convert this code to a decimal value. Finally, depending on the selected range, the analog voltage to be measured is obtained by multiplying the resulted signed word integer by the value of an LSB.

The front panel of the virtual instrument controls the setup byte on the fly through the *Channel* and *Range* controllers. It also displays the voltage and output code of the input signal(s), depending on the configuration of the MAX128. When the VI runs continuously, the software automatically reads the data acquisition system every 100 ms, but this time can be modified.

Because in this example the MAX128 operates with the internal +4.096 voltage reference, each of the eight analog input channels can be independently programmed to one of four ranges: 0 to +4.096 V,

0 to +2.048 V, ±4.096 V and ±2.048 V.

The I²C bus speed expected when developing LabVIEW applications is depended on the speed of the PC that is executing the software, but the today's computer systems can ensure satisfactory communication. On the other hand, additional problems can arise, when using Windows, the programs driving the parallel port may be periodically interrupted.

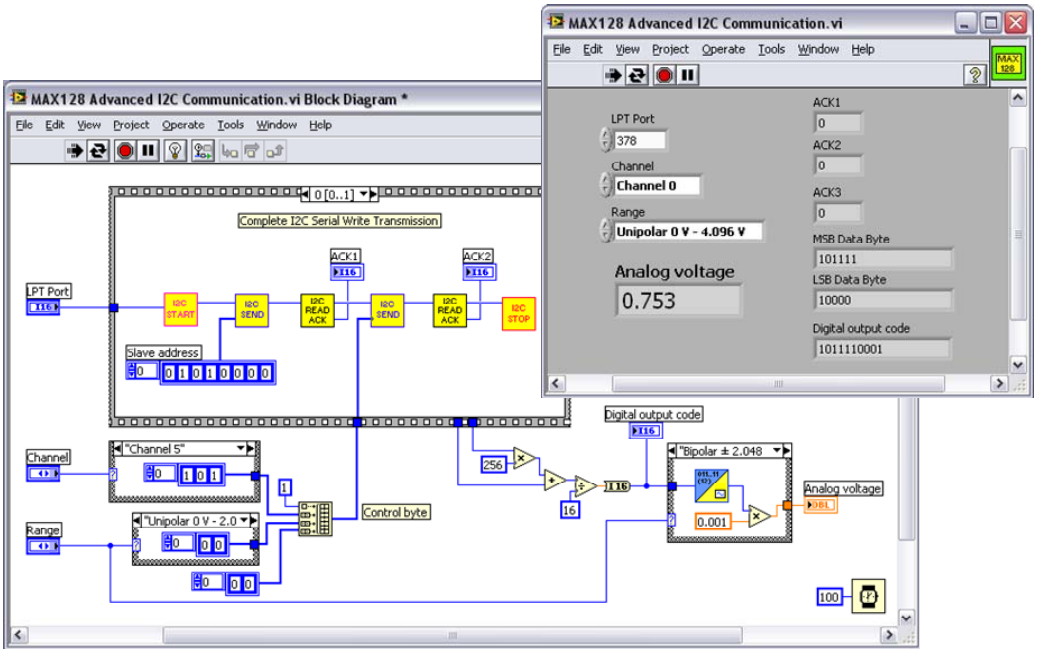

Figure 4. Advanced I²C communication with MAX128 Data Acquisition System

At this time, a few types of parallel port's operation modes exist (SPP, EPP, ECP, etc.). For this reason, the I²C protocol has been implemented in order to avoid the user's involvement with BIOS utility. Unlike the proposed parallel port solution, a possible RS-232 implementation has the major disadvantage of requiring a microcontroller to translate the RS-232 data format to the I²C protocol.

IV. Conclusions

Although there are many connecting interfaces and the parallel port has its own drawbacks, it still remains a simple solution that allows implementing popular digital protocols such as I²C. The hardware and software we proposed can be successfully used to quickly develop PC-based measurement and data logging applications including digital sensors, ADCs, microcontrollers etc., also ensuring further access to the powerful LabVIEW features.

Acknowledgements

This work was financed and developed in the framework of the RTD National Programme CEEX, financed by Romanian Authority for Scientific Research, the grants P-CONFORM CEEX-M4-3885, "Laboratory for Immunity Tests at Electrostatic Discharges" and P-INT-VIZ CEEX-M3-12335, "Support for the Integration of Romanian Research in the Field of Electromagnetic Pollution".

References

[1] Philips Semiconductor, *The I²C-Bus Specification*, version 2.1, January 2000.

[2] Electronic Design, *PCs Provide the Key to Economical yet Effective Automated Test Systems*, February 19, 2001.

[3] Dallas Semiconductor, *How to Use a PC's Parallel Port to Communicate with 2-Wire Devices*, Application Note 3230, May 2004.

[4] Dallas Semiconductor, *Automatic Test Equipment on a Budget*, Application Note 761, June 2001.