computing@computingonline.net
www.computingonline.net

ISSN 1727-6209
International Scientific
Journal of Computing

IEEE 1451.2-BASED SENSOR SYSTEM WITH JAVA-TEDS SOFTWARE TOOL

Silvano R. Rossi [1], Alexandre C. Rodrigues da Silva [2], Tércio A. dos Santos Filho [2]

[1] Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina,
srossi@fio.unicen.edu.ar, http://www.fio.unicen.edu.ar/investigacion/intelymec/index.html
[2] Universidade Estadual Paulista, Brazil,
acrsilva@dee.feis.unesp.br, http://www.dee.feis.unesp.br

Abstract: *This work presents the implementation of a microcontroller-based Smart Transducer Interface Module based on the IEEE 1451.2 standard and a Java-TEDS software tool development to generate the electronic data for each transducer channel implemented in the smart module. The module, with two transducer channels was implemented with a PIC16F876A® microcontroller and programmed in C language. A software support resource has been developed in order to generate the data for the Transducer Electronic Data Sheet descriptive memory. This software resource is fully based on Java language. When generated, the TEDS data blocks can be stored in the program memory module of the microcontroller. Methodology and results are presented and discussed.*

Keywords: *Microcontroller, Transducer, Interface, IEEE1451, STIM, TEDS.*

1. INTRODUCTION

In the last years, several concepts associated with computer networks were progressively gaining ground in measurement and control systems. Nowadays, many technologies closely related to Internet, such as WWW, Java, TCP/IP protocols and Ethernet, play a significant role in the instrumentation arena [1], [2].

Within this context, distributed measurement and control systems are composed of a set of sensors and actuators interconnected via a control network, for monitoring and controlling the physical variables involved, for instance, in an industrial process. Thus, networked smart transducers facilitate the reliability and performance of the system.

There are many interconnection technologies such as sensorbus, devicebus, and fieldbus available in different network levels [3], [4]. From the beginning of 1980s until now there were many normalization efforts; nevertheless, it is still evident the lack of a universally accepted standard to simplify the implementation of networked smart transducers [5]. Probably one of the most important efforts in this respect is the advent of the IEEE 1451 Smart Transducers Interface Standards, toward the end of 1990s. The IEEE 1451 specification can be applied in order to address the interfacing problematic [6], [7]. By using IEEE 1451.1 and IEEE 1451.2 standards it is possible to implement a network node comprised by a Network Capable Application Processor (NCAP) based on an object model of a networked smart transducer, and a Smart Transducer Interface Module (STIM) containing up to 255 transducer channels, each with its own channel Transducer Electronic Data Sheet (TEDS) stored in a nonvolatile memory. Two objectives are possible to attain with this approach: a) transducer-to-network interoperability, and b) plug and play operation mode at the transducer level.

Pioneer researchers in demonstrative IEEE 1451.2 applications have used an on-chip acquisition system based on 8051 compatible MCU [8]. These solutions have also been used in the last years [9-11]. There are similar options for implementing STIMs, for instance, by using a Rabbit 3000® microprocessor [12]. An attractive alternative for STIM implementation is the utilization of microcontrollers, due to its low relative cost and the availability of different embedded peripherals.

There are many microcontroller-based IEEE 1451 applications. A Complete IEEE 1451.1-1451.2 node was implemented for a CAN network, with a STIM based on a Phillips 87C752® microcontroller [13]. A STIM implementation with a mid-range microcontroller is also possible [14]. An IEEE 1451-based smart module for in-vehicle networking systems was implemented in [15], using a PIC16F877® microcontroller.

In accordance with previous approaches related to the current work, a minimal STIM based on a low-cost MC68HC908QY4® microcontroller is also possible, in this case, using a device with a small quantity of I/O pins [16].

Another interesting option is the utilization of Programmable Logic Devices (PLD). Through this technology it is possible to perform parallel processing; nevertheless, PLDs can become an expensive alternative when they are compared with microcontroller-based applications. There are IEEE 1451-related experimental works based on programmable logic, using PLDs of different manufacturers and developed through hardware description languages [17-19], multi-core [20], and soft-core technologies [21].

In this work, an IEEE 1451-based sensor system with a STIM module based on the PIC16F876A® microcontroller is presented. This application was created because of two reasons: a) to test an additional novel software resource fully based on Java and developed with the aim of easily generating the transducer electronic data, specified by the standard, and b) to test the 10-wire TII functionality in accordance with IEEE 1451.2 defined in 1997.

Although the initial intention of the NIST-IEEE was to create a universal interface among transducers and network nodes, its acceptance in the industrial context is still incipient. This fact can be attributed to the 10-wire TII, which has been considered to be complex by several users and sensor manufacturers. Some effort was made to search for methods of simplifying and checking the mentioned interface to customize it to others, for instance, RS232 and USB, which use fewer wires [22-24]. This is particularly interesting to support the IEEE 1451.0-2007 Standard [25]. Nevertheless, the authors of this work believe that is necessary to continue working with the TII definition, because it was initially created to implement a reliable connection-oriented communication protocol.

Once generated by means of Java-TEDS tool, TEDS information can be stored in the program memory of the implemented module. Furthermore another software tool based on Java2D was developed to verify the STIM functionality and the protocol communication over the TII interface. To achieve this goal, the parallel port of a conventional PC was used.

The rest of the paper is structured as follows; section 2 introduces a brief commentary about IEEE 1451.2 standard; section 3 focuses on the proposed system development; section 4 and 5 present the STIM implementation and the Java-TEDS tool respectively, and section 6 describes the simulation and experimental results. The conclusions are presented in section 7.

2. IEEE 1451.2

A networked smart sensor system can be partitioned into two parts using the IEEE 1451.1 and IEEE 1451.2 smart transducer interface standards, as can be seen in Fig. 1. The first one is the Network Capable Application Processor (NCAP) that performs the major data processing and control actions, establishing the link between the transducer application and the network, through an object model of a networked smart transducer defined by IEEE 1451.1. Moreover, it is necessary to obtain the data from the transducers in order to carry out the data processing. This second part is defined by IEEE 1451.2 standard, introducing the concept of Smart Transducer Interface Module.

The STIM performs the data acquisition for each transducer channel and the control of the Transducer Independent Interface (TII) in order to communicate with the NCAP. Besides, according to the IEEE 1451.2 standard, the STIM must be capable of containing some essential characteristics such as auto-identification of the transducer channels, and hot-swap capability. The auto-identification is achieved through the Transducer Electronic Data Sheet (TEDS) blocks.

TEDS formats are stored in a non volatile memory that are part of the STIM to achieve network auto-identification and plug and play operation mode of the transducer application.

The STIM can contain up to 255 different transducer channels on account of the 8 bits data processing. Analogue signals from the transducers are converted to digital format by means of an analog-to-digital converter. The ADC and signal conditioning circuits are also part of the STIM. The communication over the TII is based on the standardized protocol defined by IEEE 1451.2.

The TII is composed of ten lines with the following functions: a) address and data transport, data transport framing signals and acknowledge, and clock (DIN, DOUT, NIOE, NACK, DCLK), b) triggering (NTRIG), c) request service by the STIM from the NCAP (NINT), and e) power supply and support (POWER, COMMON and NSDET).

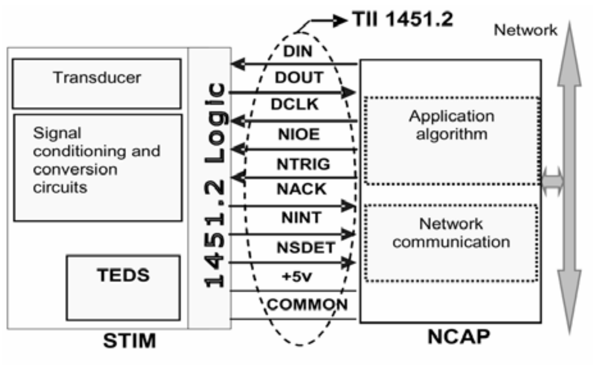


Fig. 1 – STIM-NCAP connection

3. DEVELOPMENT

Before the STIM hardware implementation based on a microcontroller it is useful to gather the following information for storing the TEDS: a) number of transducer channels to be implemented, b) transducer channel type, c) physical properties to be measured, and d) other relevant information, e.g., the acquisition time.

Thus, the data is coded in electronic format in different TEDS blocks for each channel considered into the STIM, and one Meta-TEDS structure considering the STIM as a whole. Channel-TEDS and META-TEDS are the two mandatory TEDS formats according to IEEE 1451.2.

The STIM hardware requires basically a synchronous serial port, an ADC, and a non-volatile storage [26].

In this work a microcontroller-based STIM with two transducer channels was used. One of them contains a general-purpose integrated temperature sensor model LM35, and the other channel contains a circuit that sets in motion a cooler.

4. STIM IMPLEMENTATION

The utilization of a microcontroller to implement the STIM is an attractive alternative due to its relative low cost and the availability of embedded peripherals as ADCs and high performance memories. A PIC16F876A® microcontroller from Microchip, with 28 pines, Harvard architecture and a Reduced Instruction Set Computer [27], was chosen for the STIM implementation.

This device has 368-B of RAM memory, 256-B of EEPROM and 8-kB of program memory. Besides, it is possible to work with clock frequencies of 20MHz. Thus, it is a suitable option for implementing the functionality of the STIM, to easily designate the pines for the I/O TII assignments, and to store the TEDS information.

In agreement with the standard, the mandatory TEDS formats require 96-B for each channel-TEDS block and 80-B for the Meta-TEDS. Fig. 2 shows the TII lines mapped into the chosen microcontroller.

On the other hand, the microcontroller is composed of an embedded 10-bits ADC, which is very useful for the current application, since it eliminates the requirement of an external circuit for such a purpose. In addition, the module is capable to interrupt the CPU, enabling the STIM software optimization, because polling algorithms to control the module are unnecessary.

The STIM software is based on the approach presented in [8] and was conceptually designed according to the diagram depicted in Fig. 3 that shows the software blocks to implement the module.

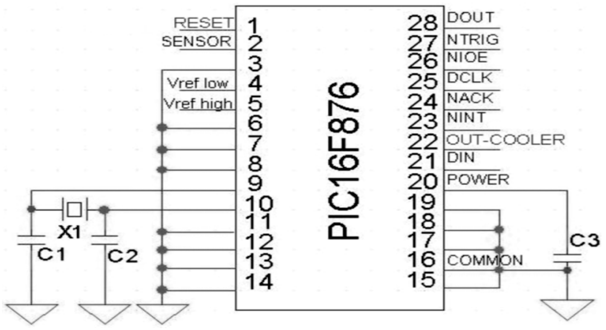


Fig. 2 – STIM hardware

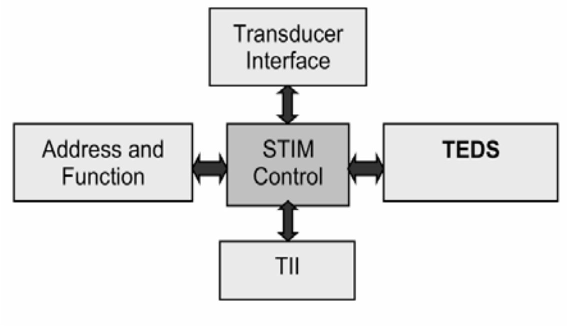


Fig. 3 – Conceptual view of the STIM software

A central routine was included to manage the program flow, the memory for storing the TEDS formats, the interface with the transducers, the control of the TII, and the address and function blocks. In order to introduce more flexibility in the code, C language was chosen for programming the microcontroller.

In terms of STIM functionality, the software can be considered to be divided in modules, each of which can be thought as a state machine. Fig. 4a shows a diagram to illustrate the above-mentioned concept. Microcontroller's ports and interrupts are configured during the initialization stage. The NCAP establishes the handshake with the STIM, thus, the NIOE line is asserted, the variables associated with the STIM and the rest of its state machines are initialized, A/D module configuration is completed, and NACK line is asserted. Subsequently it is waited until the NCAP negates NIOE line. In this situation the STIM negates NACK in order to complete the handshake process. Afterward, the STIM is ready to start the communication, performing an infinite loop as shown in Fig. 4b.

Fig. 5 depicts a simplified diagram of the trigger state machine, which serves as controller of other machines. The initial state for the trigger machine is inactive, waiting for a trigger event performed by the NCAP or the end of a data transfer between NCAP and STIM.

The trigger process begins when the NCAP asserts the NTRIG line at low logic level. In this manner, the STIM starts the data acquisition and the machine assumes the triggered state. While the state machine is in this situation, A/D conversion is performed at the same time or the output that drives the cooler is updated. If the NCAP negates NTRIG before finishing the analog to digital conversion, the STIM passes to the quiescent state. Afterwards this conversion NACK is enabled and the machine goes on to the trigger acknowledged state. Here it is waited that the NCAP negates the NTRIG signal to go on to the quiescent state, in order to carry out the data transport.

The acquisition machine performs the A/D conversion for all the different channels. The hardware of the microcontroller indicates the end of acquisition through interruptions.

The acquisition machine obtains the data necessary to update actuator outputs and dispatch data for using in other parts of the program, by means of a data buffer. Afterwards the sensor data reading process, buffer must be updated.

The function state machine interacts with the data transport machine, indicating whether corresponds a data input or output, how many data must be processed, and wherefrom to obtain them or where to send them. Initially the function machine indicates to the transport machine that should receive the bytes and should store them in a variable that contains the functional address normalized by IEEE 1451.2. Subsequently checks that this one is a valid address, then to continue with the corresponding reading or writing process.

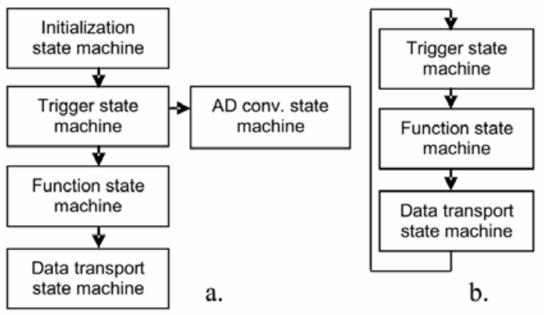


Fig. 4 – STIM functionality

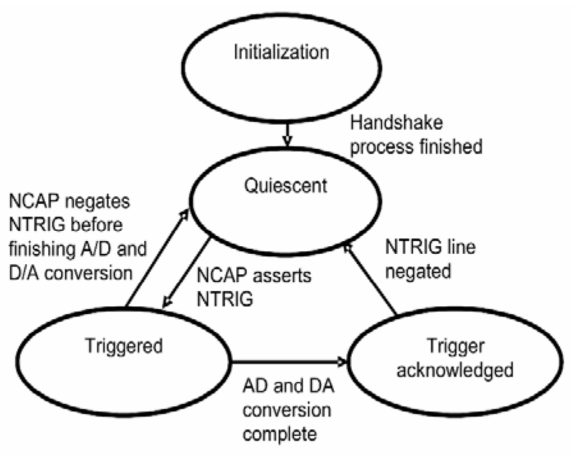

Initialization
Handshake process finished
Quiescent
NCAP negates NTRIG before finishing A/D and D/A conversion
NCAP asserts NTRIG
NTRIG line negated
Triggered
Trigger acknowledged
AD and DA conversion complete

Fig. 5 – Trigger state machine

In the case of the actuator channel a resolution of one byte is used, since this is equivalent to the switched off or switched on state of the cooler attached to the mentioned channel. If the requested operation is a TEDS's reading, the information of the length of every TEDS data block is available, facilitating the data transport through the TII.

The data transport machine receives or sends each of the bytes in a serial synchronous fashion, carrying out the data reception in the positive-going edge of the clock and the sending in the negative-going edge. Finally, the data transport machine delimits each of the 8-bit frames, inverting the NACK signal for every transferred byte. If the NCAP tries to triggers the STIM or aborts the data transfer by negating NIOE line during the transmission, any data transfer is cancelled and then, a hardware error flag is established in the standard status register of the STIM.

Fig. 6 shows the STIM prototype implemented in laboratory.

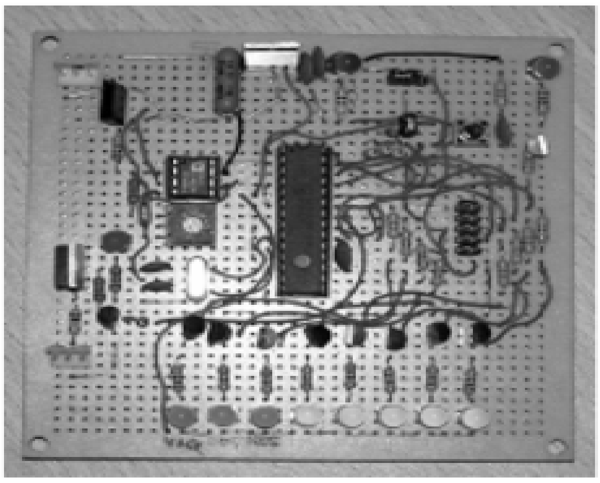


Fig. 6 – STIM prototype

5. Java-TEDS TOOL

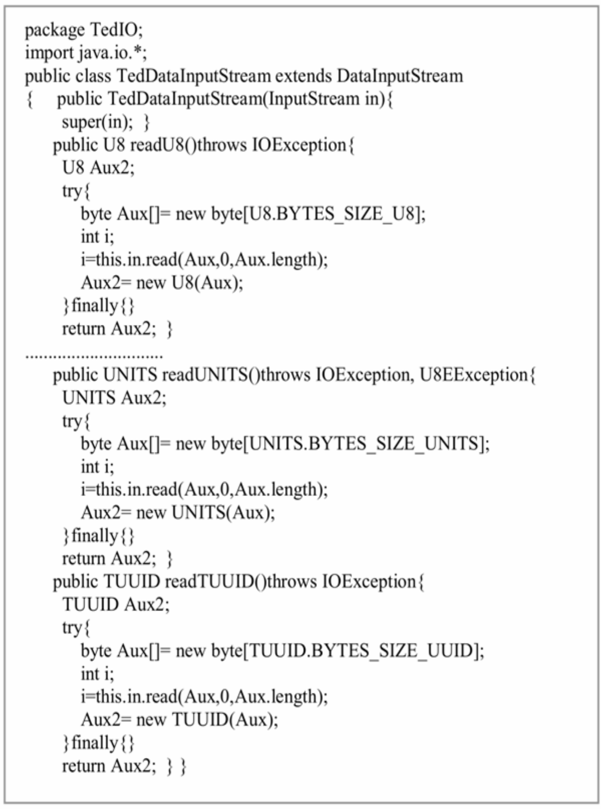
Java-TEDS is a software tool developed to easily implement the TEDS data blocks in microcontroller-based STIMs that use C programming. By using a friendly graphic interface introduced by the above-mentioned resource it is possible to create and modify the TEDS blocks for its subsequent utilization in the STIM.

This resource is broken down in three parts: a) *TedIO*, b) *TEDS*, and c) *GraphicInterface*.

TedIO package contains all the data types defined by the standard such as: U8, U16, U32, LANG, STRING, UUID (*Universal Unique Identifier*), and UNITS, among others. These classes implement an interface named *ToByte* that allows converting the values into binary format. There is also an interface U8E that represents enumeration and constant types, defined by the standard too. *TedIO* package also performs binary data input and output, using *TedOutputStream* and *TedInputStream* classes. These classes extend from *OutputStream* and *InputStream* from *java.io*, that belongs to JSDK (*Java Standard Development Kit*) and they are used in all the input and output data. On the other hand, *FormatCOutputStream* class was created for writing the content of the TEDS in the format used by C to define the byte matrix, this is, writing all the bytes of the structure between {} and separated between commas. The classes *TedChecksum* and *GarbageCollector* also were created. The first one is used by a specialized *OutputStream* to perform the TEDS checksum. The second one is an *OutputStream* in which everything that enter is discarded, because to write in a file for calculating the checksum is unnecessary.

For the sake of simplicity only a few part of the Java code is depicted in Fig. 7, due to the extensive size of the classes diagram. The figure shows a part of the code related to *TedDataInputStream*, which extends of *DataInputStream*.

TEDS formats are saved and managed by the second package, named *TEDS*, using the data formats implemented in *TedIO*, as well as the classes for data input and output, and for the checksum. The structure is composed of a class named *GroupTEDS* that contains: a) a *metaTeds* class representing Meta-TEDS, and b) a *channelTeds* class representing the Channel-TEDS data block, both defined by the IEEE 1451.2 standard.



```
package TedIO;
import java.io.*;
public class TedDataInputStream extends DataInputStream
{    public TedDataInputStream(InputStream in){
       super(in);  }
     public U8 readU8()throws IOException{
       U8 Aux2;
       try{
          byte Aux[]= new byte[U8.BYTES_SIZE_U8];
          int i;
          i=this.in.read(Aux,0,Aux.length);
          Aux2= new U8(Aux);
       }finally{}
       return Aux2;  }
..............................
     public UNITS readUNITS()throws IOException, U8EException{
       UNITS Aux2;
       try{
          byte Aux[]= new byte[UNITS.BYTES_SIZE_UNITS];
          int i;
          i=this.in.read(Aux,0,Aux.length);
          Aux2= new UNITS(Aux);
       }finally{}
       return Aux2;  }
     public TUUID readTUUID()throws IOException{
       TUUID Aux2;
       try{
          byte Aux[]= new byte[TUUID.BYTES_SIZE_UUID];
          int i;
          i=this.in.read(Aux,0,Aux.length);
          Aux2= new TUUID(Aux);
       }finally{}
       return Aux2;  } }
```

Fig. 7 – Java code example. *TedDataInputStream* class

It is important to notice that an object *GroupTEDS* can contain only one object *metaTeds* and more than one object *channelTeds*. Each of the classes *metaTeds* and *channelTeds* are composed of classes that represent the sub-blocks of the Meta-TEDS and Channel-TEDS data blocks, respectively. At the same time these ones are composed of classes that represent the primitive data types defined by IEEE 1451.2.

The *GraphicInterface* package is based on *java.swing*, and constitutes the graphic interface of the TEDS package, allowing users to browse each data block in a simple way, to make modifications, and even to save and open these TEDS in different formats. This resource is comprised by a class *gWindow*, containing the main sub-window and the most important menus to work with the TEDS. At the same time, this one contains a class *GroupTeds* and a class that represents the visualization of the *metaTeds* and *channelTeds*.

A software tool that tries to simulate the basic actions of a NCAP has been created for testing the STIM, allowing users to verify the TII functionality and communication protocols, by means of the utilization of the parallel port of a conventional PC. Java2D was used for developing the test tool, which allowed for the creation of timing diagrams to show the behaviour of signals over the TII.

Parport package was used for the managing of the parallel port. Parport is a Java class for reading and writing bytes to and from the parallel port and can be installed on Windows and Linux platforms, enabling the communication with the parallel port using Java [28].

Java's multithreading processing capability was exploited, using a thread for managing the parallel port and a thread for the graphic interface.

6. RESULTS

STIM functionality was verified through the test program created with such a purpose. With this resource it is possible to control a circuit connected to the PC's parallel port, where the TII physical interface is connected to the STIM. The results are given in qualitative way, using the above-mentioned software tool.

The first case, depicted in Fig. 8, corresponds to the handshake between STIM and NCAP, via TII interface obtained by using the software tool. The initial state for all triggering, read and write frame protocols is with the NTRIG, NACK, and NIOE lines negated, therefore after STIM detection the NTRIG, NACK and NIOE lines are at high logic level. The NCAP puts the NIOE line at low logic level and the STIM replies starting the initialization process, and then negating NIOE line. In such moment the NINT line is enabled too, since a STIM operational bit is set, remembering the fact that the bits of the standard interrupt mask are all one when the STIM is initialized. The second case in Fig. 8 corresponds to a trigger event after finishing the handshake. The NCAP asserts the NTRIG line, and the STIM asserts NACK, after finishing the A/D conversion. Afterwards, the NCAP negates the NTRIG and the STIM negates the NACK, according to the triggering protocol, defined in IEEE 1451.2. From that moment on, when the NCAP asserts NIOE line, the serial data is transferred via DIN or DOUT and controlled by the DCLK line.

Fig. 9 shows a data reading related to transducer channel #1 that contains an environmental temperature sensor. As can be seen, the code 128_{10} (10000000_2) that corresponds to read transducer data functional address is sent from NCAP to STIM, addressed to the channel 1 (00000001_2). The code representing the environmental temperature is obtained via DOUT line and NACK signal delimits every byte transferred. After finishing the data transmission, the NCAP negates NIOE line and the NACK goes on to zero logic level.

Fig. 10 shows a data reading associated with the global standard status register. The code 130_{10} (10000010_2) that corresponds to read global standard status functional address is sent from NCAP to STIM, addressed to the channel zero.

The value of the standard status register obtained through DOUT line is 0103_{16}, signalling: a) The STIM is operative, b) the trigger was acknowledged by the STIM, and c) there is a global service request.

The implementation under test is presented in Fig 11, showing the STIM connected to an interface circuitry with the PC's parallel, via TII, that allows using the Java-TEDS software.

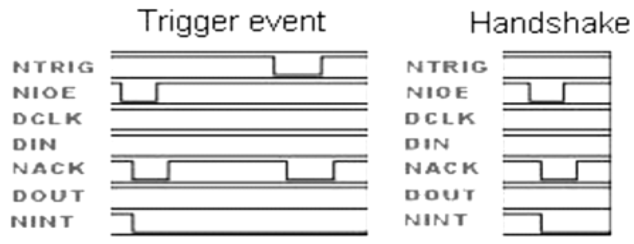


Fig. 8 – Handshake process and trigger event

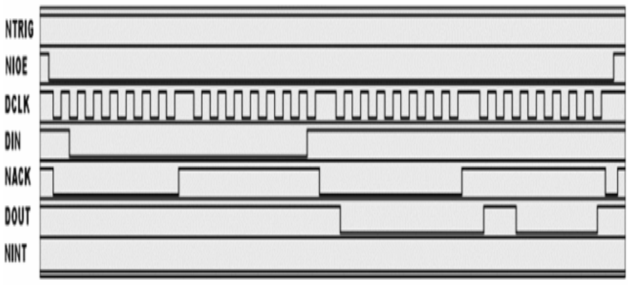


Fig. 9 – Reading data of transducer channel #1

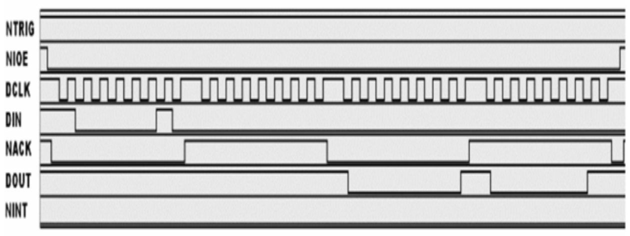


Fig. 10 – Reading data of standard status register

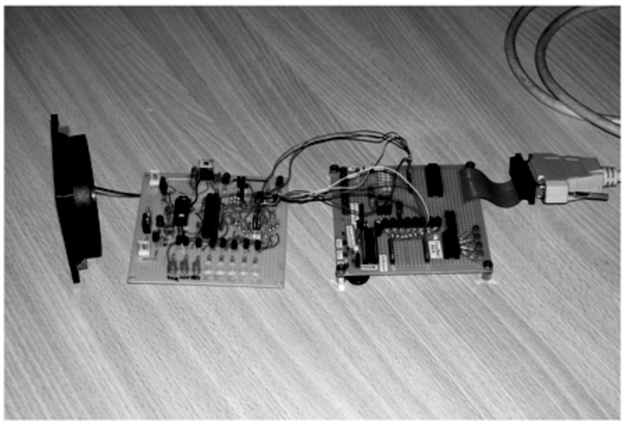
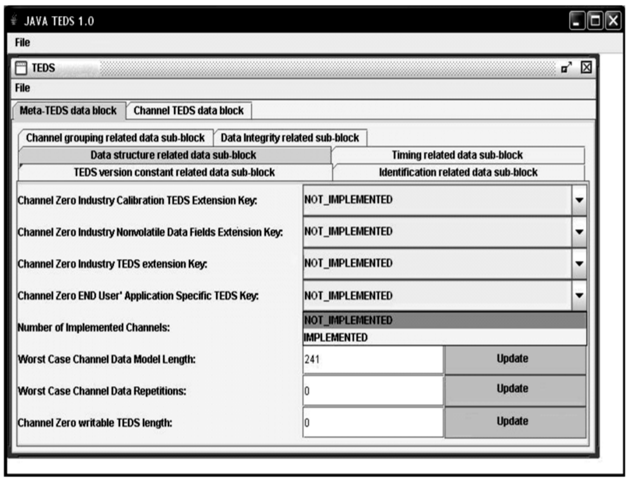


Fig. 11 – Implementation under test

Java-TEDS is a software tool that allows the creation and modification of the TEDS fields necessary for an IEEE 1451.2-based implementation. Fig. 12 shows the graphic interface presented by the program. The above-mentioned sub-window is broken down into several tabs, one for Meta-TEDS data block and one more for every implemented Channel-TEDS data block.


JAVA TEDS 1.0
File
TEDS
File
Meta-TEDS data block
Channel TEDS data block
Channel grouping related data sub-block
Data Integrity related sub-block
Data structure related data sub-block
Timing related data sub-block
TEDS version constant related data sub-block
Identification related data sub-block
Update

Channel Zero Industry Calibration TEDS Extension Key: NOT_IMPLEMENTED
Channel Zero Industry Nonvolatile Data Fields Extension Key: NOT_IMPLEMENTED
Channel Zero Industry TEDS extension Key: NOT_IMPLEMENTED
Channel Zero END User' Application Specific TEDS Key: NOT_IMPLEMENTED
Number of Implemented Channels: NOT_IMPLEMENTED / IMPLEMENTED
Worst Case Channel Data Model Length: 241
Worst Case Channel Data Repetitions: 0
Channel Zero writable TEDS length: 0

Fig. 12 – Java-TEDS tool

7. CONCLUSION

An IEEE 1451.2-based sensor system with Java-TEDS software tool has been presented. The STIM was implemented with a low-cost microcontroller and programmed in C language. The system implementation needed the managing of different technologies and was broken down into three stages: a) creation of the Java-TEDS resource in order to easily work with TEDS formats, b) developing of the microcontroller-based STIM, and c) creation of a test tool simulating an NCAP for checking the system functionality over the TII.

By means of the created resources, the employment of a general-purpose microcontroller appears as an attractive option for STIM's implementation, facilitating the use of the TII interface defined by IEEE 1451.2 – 1997.

8. REFERENCES

[1] K. B. Lee and R. D. Schneeman. Distributed measurement and control based on the IEEE 1451 Smart Transducer Interface Standards, *Trans. Instrumentation and Measurement* 49 (3) (2000). pp. 621-627.

[2] M. Sveda and R. Vrba. Sensor networking. *Proceedings of IEEE International Conf. and Workshop on Engineering of Computer Based Systems (ECBS 2001)*, Washington, 17-20 April 2001, pp. 262-268.

[3] K. Lee. Sensor networking and interface standardization. *Proceedings of IEEE Instrumentation and Measurement Technology Conference*, Budapest, 21-23 May 2001, vol. 1, pp. 147-152.

[4] M. Felser and T. Sauter. The fieldbus war: history or short break between battles? *Proceedings of 4th IEEE International Workshop on Factory Communication Systems,* Sweden, 28-30 Aug. 2002, pp. 73-80.

[5] D. Wobschall. IEEE 1451 – a universal transducer protocol standard. *Proceedings of 42th IEEE Annual Systems Readiness Technology Conference (Autotestcon 2007),* Baltimore, 17-20 September 2007, pp. 359-363.

[6] IEEE Std 1451.2. *IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Transducer to Microprocessor Communication Protocols and Transducer Data Sheet (TEDS) Formats*. IEEE Standards Board. USA, 1997, pp. 1-114.

[7] IEEE Std 1451.1. *IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Network Capable Application Processor (NCAP) Information Model*. IEEE Standards Board. USA, 1999, pp. 1-341.

[8] P. Conway, D. Heffernan, B. O'Mara, P. Burton, T. Miao. IEEE 1451.2: an interpretation and example implementation. *Proceedings of IEEE Instrumentation and Measurement Technology Conference,* Baltimore, 1-4 May 2000, vol. 2, pp. 535-540.

[9] D. Wobschall and W. S. Poh. A smart RTD temperature sensor with a prototype IEEE 1451.2 internet interface implementation. *Proceedings of ISA/IEEE-Sensor for Industry Conf.*, New Orleans, 27-29 January 2004, pp. 183-186.

[10] S. Guanming, S. Aiguo and H. Weiyi. A new distributed measurement architecture for applications of networked smart sensors. *Proceedings of IEEE Instrumentation and Measurement Technology Conference,* Ottawa, 17-19 May 2005, vol. 3, pp. 2131-2135.

[11] M. Sveda and R. Vrba. Embedded systems with IEEE 1451.1 on internet. *Proceedings of 3th International Conf. on Information and Communications,* Cairo, 5-6 December 2005, pp. 539-550.

[12] R. Wall and A. Ekpruke. Developing an IEEE 1451.2 compliant sensor for real-time distributed measurement and control in autonomous log skidder. *Proceedings of the 29th Industrial Electronics Society Conference,* Virginia, 2003, vol. 3, pp. 2482-2487.

[13] L. Cámara, O. Ruiz, J. Samitier. Complete IEEE 1451 node, STIM and NCAP, implemented for a CAN network. *Proceedings of IEEE Instrumentation and Measurement Technology Conference,* Baltimore, 1-4 May 2000, vol. 2, pp. 541-545.

[14] R. L. Fischer and J. Burch. *The PICmicro® MCU as an IEEE 1451.2 compatible Smart Transducer Interface Module (STIM)*. Microchip Technology Inc., AN214. USA, 2000, pp. 1-62.

[15] K. C. Lee, M. H. Kin, H. H. Lee. IEEE-1451-based smart module for in-vehicle networking systems of intelligent vehicles, *Trans. Industrial Electronics* 51 (6) (2004). p. 1150-1158.

[16] E. M. Martins, S. R. Rossi, A. A. de Carvalho, A. C. R. da Silva. Implementação de um módulo de interface para transdutores inteligentes utilizando microcontrolador. *Proceedings of "XV Congresso Brasileiro de Automática (XV CBA)",* Salvador-Brazil, 3-6 October 2006, pp. 1656-1661.

[17] A. Castro, T. Riesgo, E. de la Torre, J. Uceda. Custom hardware IEEE 1451.2 implementation for smart transducers. *Proceedings of 28th Industrial Electronics Conference (IECON 2002)*, Sevilla, June 2002, pp. 2752-2757.

[18] P. Ferrari, A. Flammini, D. Marioli, A. Taroni. VHDL implementation of a IEEE 1451.2 smart sensor. *Proceedings of IEEE Instrumentation and Measurement Technology Conference,* Vail, 20-22 May 2003, pp. 716-719.

[19] S. R. Rossi, E. D. Moreno, A. A. Carvalho, A. C. R. da Silva, E. A. Batista, T. A. Prado, T. A. Santos Filho. A VHDL-based protocol controller for NCAP processors, *Journal*

Computer Standards & Interfaces 31 (2) (2009). p. 515-522.

[20] C. Girerd, S. Gardien, J. Burch, S. Katsanevas, J. Marteau. Ethernet network-based DAQ and smart sensors for the OPERA long-baseline neuterino experiment. *Proceedings of Nuclear Science Symposium Conference*, Lyon, 15-20 October 2000, pp. 111-115.

[21] H. Cheng and H. Qin. A design of IEEE 1451.2 compliant smart sensor based on the Nios soft-core processor. *Proceedings of International Conference on Vehicular Electronics and Safety*, Xi'an, 14-16 October 2005, pp.193-198.

[22] R. N. Johnson and S. P. Woods. Proposed enhancement to the IEEE 1451.2 standard for smart transducers. *Sens. Mag.* 18 (9) (2001). p. 74-87.

[23] L. Bissi, A. Scorzoni, P. Placidi, L. Marrocchi, M. Bennati, S. Zampolli, L. Masini, I. Elmi, G. C. Cardinali. A low-cost distributed measurement system based on gas smart sensors for environmental monitoring. *Proceedings of International Conference on Sensing Technology*, Palmerston North, 2005, pp. 301-306.

[24] H. M. Ramos, P. M. Ramos, and P. Paces. Development of a IEEE 1451 standard compliant smart transducer network with time synchronization protocol. *Proceedings of IEEE Instrum. Meas. Technol. Conf.*, May 2007, pp. 1-6.

[25] E. Y. Song and K. B. Lee. Sensor network based on IEEE 1451.0 and IEEE p1451.2-RS232. *Proceedings of IEEE International Instrum. Meas. Technol. Conf.*, Victoria, Vancouver Island, 12-15 May 2008, pp. 1-6.

[26] R. D. Smith. *Building IEEE 1451.2 Smart Transducer Interface Modules (STIMs).* Telemonitopr Inc., Columbia, pp. 1-11.

[27] PIC16F87X. *Data Sheet*. Microchip Technology Inc., USA, 2001. pp. 1-218.

[28] J. G. del Cid Portillo. Parallel Printer Access Through Java. Available: http://www.geocities.com/Juanga69/parport/index.html.

***Silvano R. Rossi** received the B.Sc. degree in electromechanical engineering from the Universidad Nacional del Centro de la Provincia de Buenos Aires (UNCPBA), Olavarria, Argentine, and the Ph.D. degree in electrical engineering from the São Paulo State University (UNESP), Ilha Solteira, Brazil, in 1999 and 2005, respectively.*

He is currently involved with instrumentation systems and IEEE 1451 standard applications. He is a Professor with the Department of ElectroMechanical Engineering, UNCPBA.

His research interests include instrumentation systems and measurements, smart transducer networks, autonomous vehicles, and digital systems.

***Alexandre C. Rodrigues da Silva** received the B.Sc. degree in electrical engineering from the University of Mogi das Cruzes, Mogi das Cruzes, Brazil, in 1984, the M.Sc. and Ph.D. degrees in electrical engineering from the University of Campinas, Campinas, Brazil, in 1989 and 2003, respectively, and the degree of Free Lecture from the São Paulo State University (UNESP), Ilha Solteira, Brazil, in 2003. In 2007, he developed postgraduate stage at the University of Limerick, Limerick, Ireland.*

He is currently involved with synthesis tools for mixed-signal circuits, embedded systems, and IEEE 1451 standard applications. He is an Associate Professor and a Researcher with the Department of Electrical Engineering, College of Engineering (FEIS), UNESP.

His research interests include synthesis of mixed-signal circuits, embedded systems, smart transducer networks, HDLs, and Petri Net.

***Tércio A. dos Santos Filho** received the degree in computer science in 2004 from the University of Rio Verde, Rio Verde, Brazil, and the M.Sc. degree in electrical engineering in 2007 from the São Paulo State University (UNESP), Ilha Solteira, Brazil, where he is currently working toward the Ph.D. degree in the area of electronic instrumentation and control.*

His research interests include operational systems and computer networks, embedded systems, and sensor networks.