WinCE-based Embedded System for Control of an Industrial Screw Machine

António H. J. Moreira*, Jaime Fonseca+, Adriano Tavaresº
Industrial Electronics Department, 4800-058 Azurém, Guimarães - Portugal.
a42974@alunos.uminho.pt*, jaime@dei.uminho.pt+, atavares@dei.uminho.ptº

***Abstract* - Nowadays, industrial systems frequently require the control of some industrial process and monitoring of relevant data about the process, using a friendly visual environment. Normally, is used a PLC (Programmable Logical Controller) to control the process and assure that the timing requirements (deadlines) are satisfied and a PC to monitor the data. However, the implementation of such solution presents the following drawbacks to the system programmer: (1) he or she needs to know the communication protocol between the two platforms - PLC and the PC; (2) he or she needs to learn two different programming languages - the low level PLC language and a high level PC language. On the other hand, in some cases, the reserved space to control the systems is reduced, making the implementation of such solution very hard. This paper presents an approach based on an embedded PC with real-time processing capability and data monitoring facility. The proposed system runs the Windows CE operating system and allows all software development in C/C++, using the Microsoft Visual Studio environment. The system was tested on an industrial screw machine for PCBs.**

I. INTRODUCTION

Traditional control and monitoring systems, Fig. 1, uses a PLC for control and a PC for HMI (Human Machine Interface), making them very hard to implement, since it's necessary to learn different programming languages and hardware platforms [1]. These systems are widely spread in control and monitoring tasks because of the improvements in PC performance, making this kind of approach very cost-effective [2].

Nowadays, the former approach continues to be valid, but is not so cost-effective, as the system development time is a little bit high, due to the use of two different programming languages.

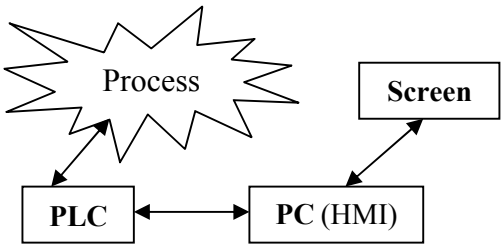

Fig. 1 Control and monitoring system using PLC and PC

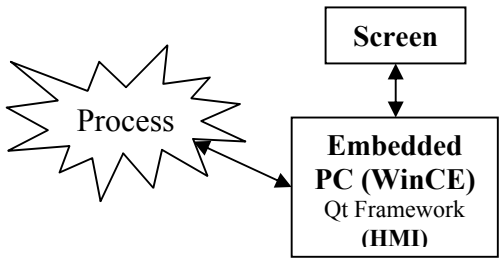

Fig. 2 Control and monitoring system using an embedded system

To make a new product more cost-effective, the embedded PC approach, Fig. 2, with real-time capability is used to overcome the problems of development time and costly control system. With the growing adoption of Windows CE in industrial environment, due to its real-time and user-friendly interface capability arose new opportunities to reduce the system development time and consequently the system cost.

Since minimal system requirements are needed to run Windows CE and with characteristics such as: instant power-up, Low ISR (Interrupt Service Routine), 255 levels of priority and low thread latency, it becomes perfect to almost every industrial process. The use of the priority levels in a preemptive multitasking system, requires a careful assignment to the different threads to avoid application deadlock, as the scheduler gives the processor to the thread with highest priority[3][4]. Also, deadlines missing due to priority inversion must be avoided by careful uses of synchronization objects, like mutex, semaphore, critical section [6].

The choice of Windows CE for this application was natural as its real-time capabilities cover all the needs of nearly 95% of all industrial process, given that no more than 100us of IST (Interrupt Service Thread) latencies is perfect. In fact, for most part of applications this reduced latency is unneeded [5]. To make even a more complete embedded system, the C++ API isn't the easiest way to design a user-friendly interface, since it can be very painful and time consuming. Therefore, it was chosen, the C++ for critical control tasks and Qt framework for monitoring the relevant data (non critical task). In doing so, it's possible to develop a real-time

application with the flexibility of Qt framework and using only one high level language.

This paper is organized in tree major sections: (II) the application timing requirements, (III) the system description in terms of hardware and software and (IV) the final results.

II. Application Timing Requirements

As said before, industrial cells are normally controlled by a PLC and data monitoring performed by a PC. In this work the goal is controlling an automatic screwing machine, constituted by three axes and an electric screw. Each axis is controlled by its own controller. The major requirements are concerned with safety protocols, the emergency stops and fast responsiveness of the system. The user interaction with the safety systems is very critical because the system needs to react as fast as possible but the point is: how fast needs it to be (TABLE 1)?

TABLE 1
LATENCY TIMES FOR DIFFERENT APPLICATIONS

5ms	2.5ms	1ms	0.5ms	200us	100us	50us	20us
			Mobile Phones				
Consumer Electronics							
			Telecom/Datacom				
						Automotive	
		Industrial Automation					

Depends on the application and as TABLE 1 indicates, the worst latency time admitted for industrial automation is between 100us and 1ms, but on automotive applications is only 50us. Approximately 95% of the industrial working cells do not process simultaneous signals because the machines work in a sequential way, i.e., the system only, in few occasions, has to handle more than one signal simultaneously.

Only one deadline related to the safety equipment (emergency procedure) must be met, as it is desirable that when an emergency occurs the system stops and reacts as fast as possible. For the designed system running Windows CE that deadline is approximately 1ms as shown in Fig. 3.

Another requirement was the use of Qt framework for a rapid development of a friendly monitoring interface. This framework is easy to learn and use and also offers several ready-to-use controls. Signals and slots make it one of the most powerful graphical/visual frameworks on the market. The designed user interface shows on screen some information related to the screwing status of each screw.

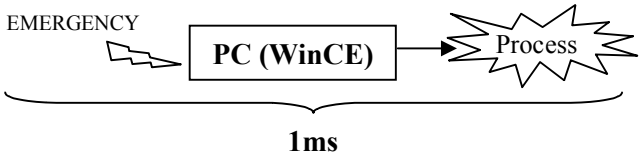

Fig. 3 Latency defined as objective for this application

III. The WinCE-based embedded system

Fig. 4 Orchid and Colibri board from Toradex

The hardware used on this application is an embedded system running Windows CE 5.0 from Toradex Company, Fig. 4. The system is composed by the Orchid board and a 520MHz Colibri XScale PXA270 processor. The main features of this system are: 14 GPIO (General Proposal I/O), 4 analog inputs with a resolution of 10 bits, 4 PWM outputs, I2C, SPI, RS232, Audio In/Out, VGA, LCD, SD/MMC, USB and Ethernet. The board can be supply from 7V to 24V.

This board was chosen since it has the flexibility to be directly connected to the industrial power supplies without any kind of adaptation, providing also all the peripherals connection needed for this and future projects.

The digital interface between the Orchid board and the industrial equipment used in this application presented the following problems:

- The digital inputs/outputs in the industrial equipment require a 24V supply voltage and the internal GPIOs of the Orchid board require a maximum 3.3V supply voltage.
- The number of the inputs/outputs of the Orchid board is not enough for implement all requirements of the screw machine.
- The inputs/outputs of the Orchid board don't have electric isolation.

To guarantee digital interface compatibility between the screw machine and the Orchid board, some external boards with opto-isolating digital inputs/outputs were designed. These boards are connected to the industrial equipment by 24V and to the Orchid board using the I2C protocol. The I2C baudrate offers by the PXA270 is 400kbps that is enough for most of the industrial applications.

1. Hardware Architecture

The external boards use the MicroChip MCP23017 IC that offers 16bit output/input signals over I2C connection. Each board can only be used as inputs or outputs and all 16 pins in the input and output boards were isolated with the PS2502-4 opto-coupler. Additionally, the input board requires extra protection provided by a SIOV Metal Oxide Varistor.

A. *I2C Input Board*

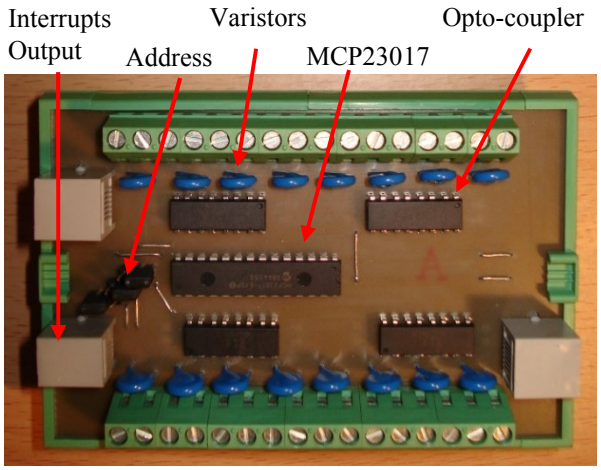

Fig. 5 External I2C INPUT board

The external I2C INPUT board, Fig. 5, has 16 inputs, all of them independent, but if necessary, the inputs could be made common in four groups of four inputs each by shunting the four common inputs in the solder side of the board.

B. *I2C Output Board*

The external I2C OUTPUT board, Fig. 6, has 16 outputs, all of them independent and like the input board four common outputs could be shunt in the solder side of the board, making four common groups of outputs. With this system it's possible, in the same I2C bus, to connect eight boards ($2^3 = 8$) of any type (inputs or outputs), achieving at most 128 I/O.

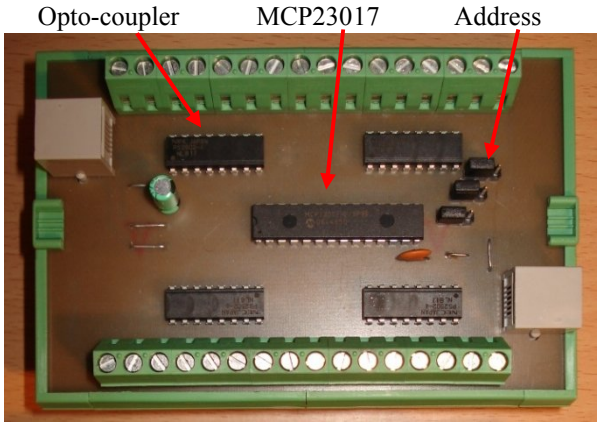

Fig. 6 External I2C OUTPUT board

The Fig. 7 shows the screwing machine were the embedded system was tested.

C. *Screwing machine*

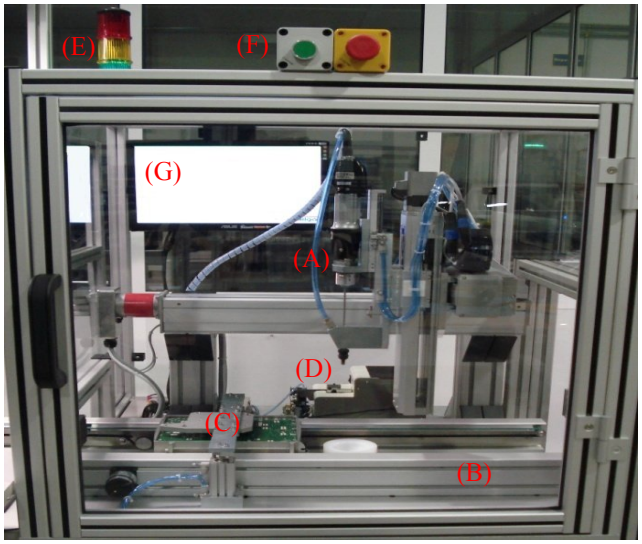

Fig. 7 Industrial working cell – Automatic Screwing

This machine is constituted by an electric screw machine (A) with a controller connected to the external output board to start or reset the screwing. The indicator signal for the end of the screw process is connected to the external input board. Three independent axes, each one with its controller (AEB-Robotis RBT5), are connected to the output and input boards. Two belt conveyors with two DC motors (B), one centering plate that centers the PCB and holds it in place(C).

The centering plate is composed by three pneumatic actuators, where one of them is used to stop the chassis in the center of the machine. After the centering process another pneumatic device pushes the screw machine down. All the electric valves (centering plate, pneumatic table, stopper) and the motor drivers are connected to the output board and a Quicher Screw Feeder (D) is used to feed the electric screw. Compressed air is used to send the screw to the hole when it is detected by one of magnetic sensor in the tube. This sensor and the valve are connected to the input board and output board, respectively. Safety lights (E) are used to warn the user about the process and the two buttons (F), one to start the process and the other one for emergency purposes. The door also actuates as an emergency when it opens. The screen (G) is used to show information related to the current process.

The Toradex board and the extension boards are powered by one 24V power supply and the axes controllers by one 28V power supply.

The next diagram, Fig. 8, shows a simplified view of the connections between the different system modules.

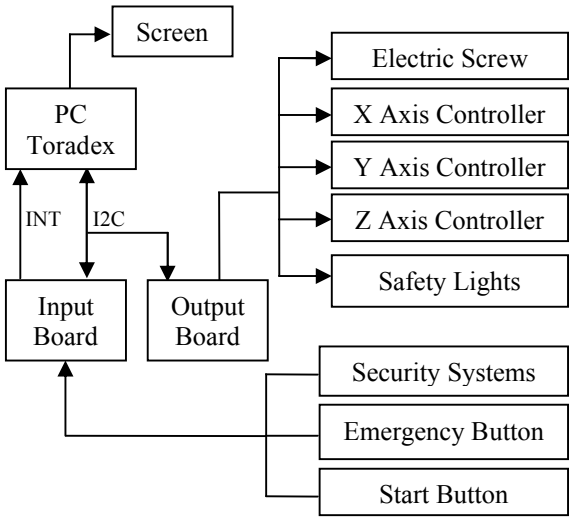

Fig. 8 Connections Diagram

The INT signal has to be connected to two inputs in the Orchid board because it is needed to signal the board of any changes in the inputs of the external I2C input board.

2. Software Architecture

The software application has to be very robust because the machine must work during 24 hours, 7 days a week. The following class diagram, Fig. 9, presents the architecture of the implemented application.

The screwing machine required a sequential execution, making it easy to change, if necessary, and the control process core was implemented as a state machine.

At the startup, an APP object is created to initialize the Qt platform. Then the visual environment object is created and all needed interrupt requests registered. This GUI object communicates with the StateMachine object to carry out the monitoring process.

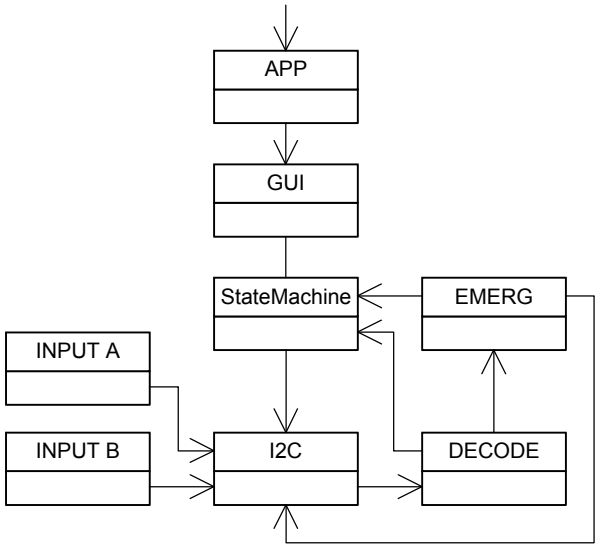

Fig. 9 UML class Diagram for the screwing machine

The GUI object also initializes all the others object such as EMERG, DECODE, I2C, INPUT A, INPUT B and even the StateMachine one.

The INPUT A and B classes configure one Orchid board I/O to be attached to the ISR (Interrupt Service Routine) corresponding to that I/O and associates it to one IST (Interrupt Service Thread) that handles the code execution.

```
while(1)
{
        // Wait for Event (Interrupt)
        if(WaitForSingleObject(hEvent,INFINITE)        ==
        WAIT_OBJECT_0)
        {
                sendSIGNAL();
                InterruptDoneCompat(dwSysIntr);
        }
}
```

The Above code segment shows how to use the *WaitForSingleObject(hEvent,INFINITE)* to wait for an interrupt in one I/O that is registered in *hEvent*, that later can be released by:

```
InterruptDisableCompat(dwSysIntr);
ReleaseSysIntr(dwSysIntr);
```

Upon the occurrence of an interrupt, a signal is sent to the I2C object and it will retrieve 8-bits data that is sent to the DECODE object to analyze if the signal has its value changed. The DECODE object signal can be sent to the EMERG object or to the StateMachine object.

If the input related to the emergency procedure has changed value, the EMERG object is then signaled, starting the highest priority thread to reset the state machine and change the values at the output board. Then the state machine starts from a well-known state and continues working.

The StateMachine class implements a switch statement and each case statement is divided in two parts: an action and a condition. The action part implements the needed actions on the external output board and also on the data to be send to the screen. The condition part implements the needed conditions to transit to the next state.

```
void EXEC_Programa::run()
{
        while(1)
        {
                switch(m_STATE)
                {
                case 0:     STAGE0();
                            STAGE0W();
                            break;
                case 1:     STAGE1();
                            STAGE1W();
                            break;
```

```
                    …
                                }
                Sleep(1);
            }
    }
```

The *Sleep(1)* statement stops the thread for 1 ms, allowing the execution of other lower priority threads.

Qt framework was chosen to design the visual environment since it is easy to model interactions between objects (controls) through its signals and slots mechanism. The signals and slots mechanism is type safe, which means the signature of a signal must match the signature of the receiving slot, otherwise it won't work. Example:

```
public slots:
    void setValue(int value);
signals:
    void valueChanged(int newValue);

QObject::connect(&a, SIGNAL(valueChanged(int)),
                 &b, SLOT(setValue(int)));
```

This mechanism was used to set the communication between the different objects (INPUT A/B, I2C, GUI, etc).

The Fig. 10 shows the interface with the operator. (A) shows the information about the screwing process: Nr (index number of the screw), torque, angle and status (the final result of the screwing process OK or NOK) implemented with the QListWidget class, (B) is the LogOff button implemented with QPushButton class, (C) indicates information about the user: name, local, bar code, program, all of them implemented with the QLineEdit/QLabel classes and (D) on the top indicates the state of the machine, showing messages like: Put Piece, Emergency, Initialization needed, Centering PCB, PCB – OK, PCB – ERROR and others.

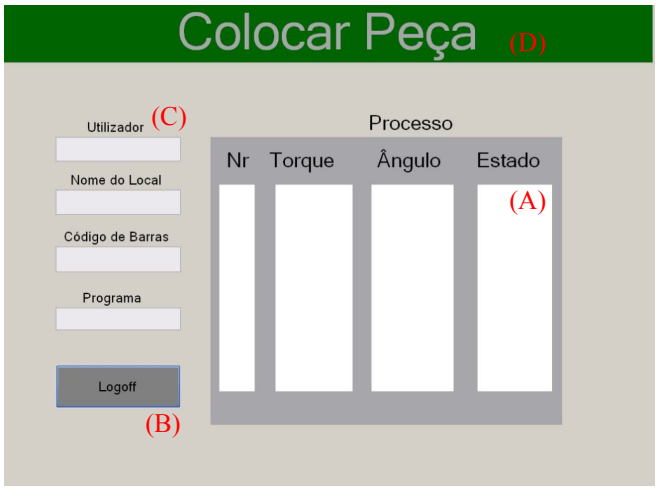

Fig. 10 Qt environment created for the screwing application

IV. Results

During the tests running the screwing machine, some communication times over I^2C were measured: (1) to change the values in the output board and (2) to read values of the input board. In the Fig. 11 it's shown the time needed to send 3 bytes (Address, Register and Data) over I^2C with ACK (Acknowledgment).

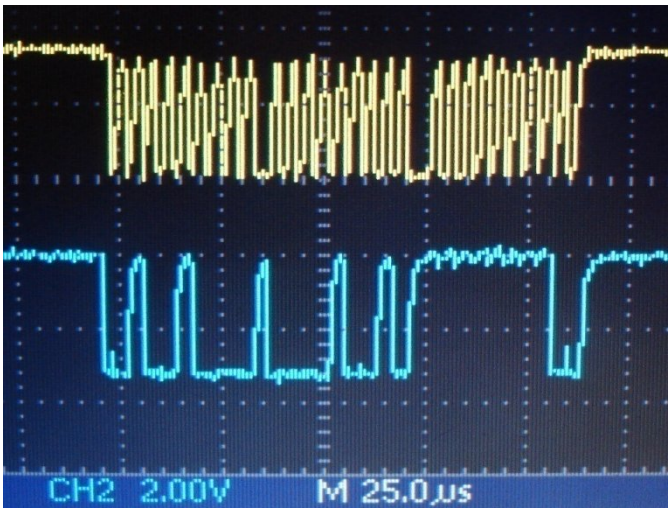

Fig. 11 I2C communication with the output board

Sending these three bytes, over I^2C with 400kbs requires approximately 120us, Fig. 12. The upper signal, Fig. 11 represents the SCL (clock line) and the down signal represents the SDA (data line) with the three data bytes.

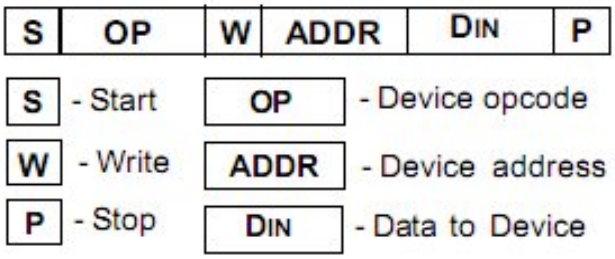

Fig. 12 I^2C Protocol to write on output board

But how much time is needed to achieve some reaction on the output board if one emergency occurs in the input board? To answer this question, the interrupt time of the board was measured, Fig. 13. A 10 kHz square wave was applied in one pin configured as input and the signal replicated into another pin configured as output. The time variations between the input and output signals were measured.

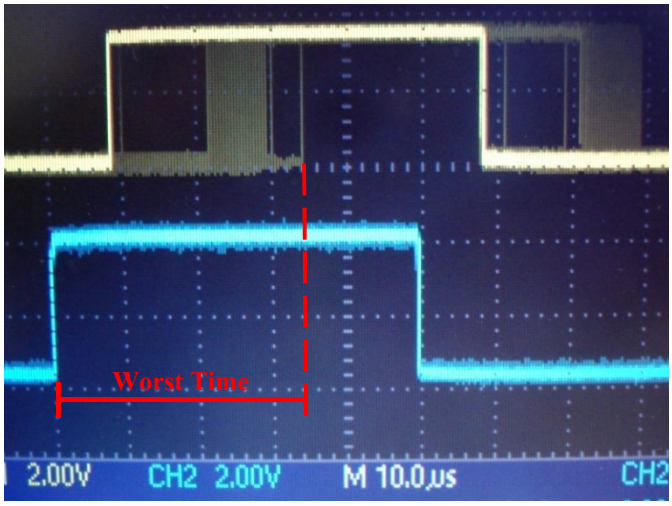

Fig. 13 Interruption time

This time depends on the operating system (Windows CE) as also on the system load, but as it can be seen, during this test which replicates only a square wave with higher priority and applying different loads during time, the higher worst time was only 35us and the minimum worst time less than 10us. This time variation could be unacceptable in some application, but it meets the requirements of the application described here – remember that the worst case time for the emergency was previously defined as 1ms.

The I²C and interrupt time present a worst case time of approximately 155us that was not included in the processing time. The time required to read the input board is the time needed to send at 400 kbs the following frame: 2 bytes indicating the registry, one start bit and 1 byte of opcode (address) and receive 1 byte, Fig. 14.

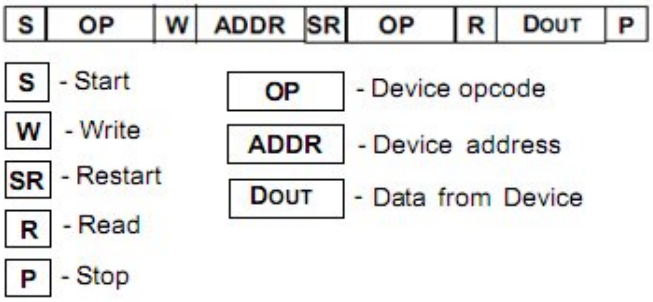

Fig. 14 I2C Protocol to read from the input board

Now the total time needed to acquire one interrupt, read the input board and put some value on the output board, is calculated as:

$$\textit{Interruption time} + \textit{Read Time} + \textit{Write Time} \quad (1)$$
$$(35us + 160us + 120us) = 315us$$

The final time does not take into account the processing time, but since the threads that handle the emergency interrupt have the highest priority, it satisfies the requirements of the designed application. Duplicating the calculated time, only for safety reasons and due to the unknown processing time (because the unpredictable load that the system may experiment), the total time is 630us, which is still less than the 1ms previously pointed as the emergency procedure deadline.

V. Conclusions

In this paper is described an application based in a low cost embedded system from Toradex as an alternative to a system based on PLC and PC. I/O expansion boards with 16 inputs or outputs opto-isolated for interface with the industrial equipment (controllers, pneumatic valves, etc) and capacity to communicate over I²C bus were developed. This system was tested on an industrial screwing machine to screw PCBs.

The principal advantages of this system are:

- Cheaper solution than the PLC+PC approach.
- Only needs to learn one programming language, in this case C++ with Qt Framework, which reduces the developing time.
- It's used a high level language on contrary to the PLC language.
- This system can meet the time requirements of most industrial applications, namely for this type of working cells.
- It's possible to develop different I²C based modules to achieve the needs of other applications, becoming a very flexible and expansible system.
- The system presents different connections, including integration with small touch screens that makes it suitable to panel control.
- The networking integration is easy because the board has an Ethernet port.

Presently, the prototype cell is working almost during one month, 12 hours a day. No reliability problem was registered, so far, during the working time.

References

[1] Jianmin Duan, F. L. (2008). "Research of Key Technologies in a Windows CE-based Monitoring and Control System". Industrial Electronics and Applications, ICIEA 2008 , 494-496.

[2] Ogawa, M., & Henmi, Y. (2006). "Recent Developments on PC+PLC based Control Systems for Beer Brewery Process Automation Applications". SICE-ICASE, 2006. International Joint Conference , 1053-1056.

[3] Jacqueson, N. (1999). "Windows CE for Industrial Computing". Real-Time Magazine, 76-80.

[4] Kawakami, I., Nimura, Y., & Hamada, K. (2000). "Real-time extension for Windows NT/CE used for control systems". SICE 2000. Proceedings of the 39th SICE Annual Conference. , 319-324.

[5] Hall, M. (2005). "Windows CE 5.0 for real-time systems". Embedded Computing Design.

[6] Http://msdn.microsoft.com/en-us/library/aa450594.aspx