MIDAS: An inexpensive, real-time, scalable, network-based Data Acquisition System

Antonio Barbancho and José Antonio Pérez

Departamento de Tecnología Electrónica. Universidad de Sevilla. Escuela Universitaria Politécnica de Sevilla. C/ Virgen de África, 7. 41011 Sevilla. España. Tlf: (34) 5 455 28 37. Fax: (34) 5 455 28 33. Email ayboc@cica.es

Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla. Facultad de Informática y Estadística. Avenida de la Reina Mercedes, s/n. 41012 Sevilla. España. Tlf: (34) 5 455 27 67. Email: jperez@lsi.us.es.

Abstract: At this paper, we present MIDAS. MIDAS is a low-cost, low-power, distributed data acquisition system. MIDAS is based on a bus-like network and a multitasking real-time master. Sensors are connected as slaves and are fully controlled by the network master. Since some prototypes have already been built, we expose practical results on costs and performance obtained.

Résumé: Dans cette communication, nous présentons MIDAS. MIDAS c'est un système distribué d'acquisition de données à bas consommation d'énergie et bas coût. MIDAS utilise une architecture de bus, avec protocole maître-esclave et logiciel multitache temps réel. Les capteurs sont connectés comme esclaves et sont contrôlés par le maître du réseau. Des prototypes ont déjà étés construits, et ici nous présentons des résultats pratiques avec des considérations sur les coûts et les performances obtenues.

1. Introduction

Data acquisition systems are becoming more and more sophisticated, having to fulfil harder requirements about power consumption, cost, scalability and flexibility.

A modern general-purpose data acquisition system should be able to:

- manage a large number of sensors;
- manage very different nature sensors;
- manage long distance located sensors;
- make easy installation and maintenance tasks;
- operate in stand-alone environment, battery-powered;
- store large amount of data;
- perform real time actuation;

all of this, of course, at the lowest cost.

In this paper, we describe MIDAS, a MultIpurpose Data Acquisition System, which complies with all these requirements. MIDAS is a network-based, real-time, low-power, low-cost system. Furthermore, MIDAS is an open system, which can grow up as much as needed. Its most relevant features are:

- Up to 256 elements (sensors, actuators or any other device) can be addressed simultaneously.
- All devices are plugged into an 8-wire bus-based network. So only eight parallel wires are needed to power and control all devices, making easy installation and maintenance.
- Low supply current. This enables to power it from a battery pack with solar panel.
- All sensors have a smart interface, allowing uniform treatment. This simplifies management of different nature sensors.
- All devices are plug and play. Warm plugging is allowed.
- Real time Operating System.
- Flexible programming.

MIDAS is not a commercial system, but it has been designed to be used as a prototype for several commercial systems actually under development.

In section 2 we describe the architectural basics of MIDAS. Section 3 shows hardware and software implementation issues, and section 4 discusses some experimental results obtained from first prototypes. Finally, conclusion and actual investigation lines are presented in section 5.

2. System architecture

MIDAS is based on a real-time bus-like network. Devices connected to this network can be classified as masters and slaves. Only one master is currently allowed in the network. The master device configures and assigns network addresses to slaves. The master acts as datalogger, governing the system operation (see Fig. 1).

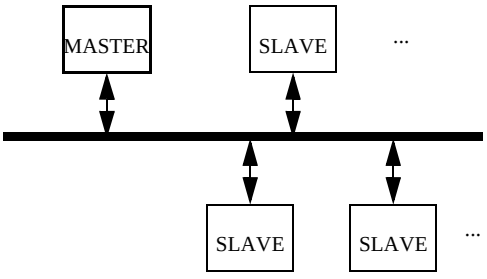

Fig. 1: System architecture

All other devices in the network (intelligent sensors, data storage systems, signal conditioning, user interfaces, and so on) are slaves. All slaves must show a standard interface to the network. If needed, any traditional sensor can be connected to the network through an interface module, allowing low-cost transition from traditional systems.

2.1. Network Architecture

MIDAS network model is based on a simplified OSI model [2] and lacks of network to presentation layers. This means, it only has physical, data link and application layers.

The Physical Layer is concerned with transmitting raw bits over the communication channel [3]. In order to make a low cost design, our design is based on RS-485 and SPI (Serial Peripheral Interface) standards. The communication channel is a bus that carries three signal lines (CLK for clock, DATA for data and SS for slave selection) and two power lines (V+ and ground). Since RS-485 protocol is used, each line is composed of two wires (balanced transmission). This makes possible communication along medium distances (4000 feet) without line repeaters. The communication is synchronous and leaded by the master device. Transmission speed is controlled by CLK and may be up to 10 Mbps. Although RS-485 specifies that up to 32 loads[1] on the bus, our design uses 1/8-unit-load transceivers, allowing 256 devices on the bus.

The main task of the Data Link Layer is to take raw transmission facility and transform it into a line that appears free of transmission errors to application layer [3]. It is also responsible for the identification of the slave. As all devices share the bus, there must be a unique network address for each device. A detailed description of the protocol of this layer would exceed this paper purposes.

The Application Layer is a collection of small protocols providing specific solutions. Examples are file transfer, remote operation and slave management. The later deals with the automatic identification and configuration (Plug'n'Play) and network address assignation.

2.2. Master Architecture

The master is responsible to perform the following tasks:

- Manage the application level network protocol, as described in section 2.1.
- Perform the acquisition process. This is done by executing an user program, as we will describe later.

From the hardware point of view, the master architecture consists on a CPU microcontroller, a network interface, a serial RS-232 interface, a real-time clock and programmable NMI generator, a ROM memory containing embedded code and a nonvolatile SRAM memory bank, as described in Fig. 2.

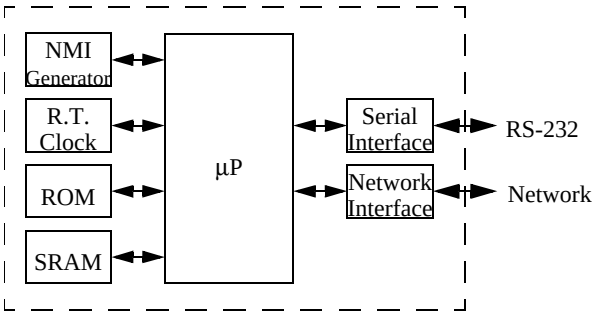

Fig. 2: Master architecture

The purpose of the RAM bank is for device internal use or for data storage. Data storage may be increased via an external device connected to the network, allowing massive storage.

The serial RS-232 interface is conceived for facilitating downloading and uploading programs, debugging and frequent tasks like data retrieving or system monitoring. Although a specialized device connected to the network can be used for this purpose, having this facility integrated into the master has two main advantages:

- The master is self-contained. No other device is needed to control the system. You do not need an additional device to perform elementary tasks. This reduces the cost of a minimum system.
- It reduces data traffic in the network when performing such tasks, reducing congestion risk.

Furthermore, while developing MIDAS first prototypes, this serial interface was very useful for debugging and profiling tasks.

From the software point of view, the master contains four independent modules:

- Real-time kernel, which implements multitasking and real-time scheduling issues. This kernel serves as basic support for all other software facilities, as described in section 3.2.
- Network protocols for physical and data-link layers, and master management protocol for application layer.
- User's interface for RS-232 serial link. Simple Xon-Xoff protocol is used to manage communication.
- User's program interpreting machine. The master executes an user-supplied program which executes on an emulated virtual data acquisition machine.

Software aspects are detailed in section 3.2.

1. A load is the input impedance of a transceiver, and is 12KΩ in the standard.

2.3. Slave Architecture

Since the master has not input/output capability (except for the network and serial links), it has to reside on the sensor end. Therefore slave architecture is very application dependent. Nevertheless, it always must include an interface part to the network. This interface has been designed to be simple, allowing low cost and low power.

3. Implementation issues

3.1. Hardware issues

Two types of cables can be present on a bus: 10-wire IDC ribbon cable for short distances, and 4 twisted pair 24 AWG cable for long distances and special protection. A special connection module has been designed to connect both.

The master device is based on the Hitachi H8/500 microcontroller family. It was selected because of its 16 bits nature, low power, rich set of peripherals and low cost. The first prototypes use the H8/520, a small 64 pin PLCC housed, 32 Kb OTP EPROM and 1 Kb RAM microcontroller. A pair of USART on-chip peripherals allows connection to an RS-232 port and to the network.

A bank of SRAM is also available on the master, allowing up to 512 Kb of local, fast, nonvolatile memory. A lithium battery must be internally or externally connected to power SRAM and real- time clock in the absence of main power.

In order to increase the number of devices that can be present on the bus, 1/8-unit-load RS-485 transceivers are used, allowing up to 256 devices. The MAX3082 was selected due their low power, low cost and fail-safe features. This make possible turning off all drivers to save power when the bus is idle. When a receiver 'sees' a floating line, puts its output in a known state (high).

The whole master is enclosed in a small DIN rail mount enclosure.

Slave architecture is application dependent as noted in section 2.3. The network interface and other common tasks rely on a Zilog Z8 microcontroller [5]. A Z86E06 device was chosen due to its very low cost, very low power, low pin count and SPI capabilities. A unique feature of this device is its ability to act as SPI slave in a compare mode. In this mode, the microntroller is in standby mode, drawing a few microamperes. The SPI peripheral is controlled externally by the master. A transaction begins with the assertion of SS line. Then eight bits are received via the DATA line. This byte is compared with a special purpose register. If the data does not match, the Z8 ignores all data until SS line is reset. If the data does much, the microcontroller leaves standby and continues the transaction.

3.2. Software issues

Since MIDAS complexity mainly relies on the master, this was the main challenge when dealing with software design. A great software engineering effort was taken in order to produce a flexible, reusable, easy to maintain and extend embedded code. As advanced in section 2.2, the master software architecture consists on four slightly coupled modules:

- Real-time kernel, which implements multitasking and real-time scheduling issues.
- Network protocols.
- User's interface for RS-232 serial link.
- User's program interpreting machine.

In order to reduce coupling as much as possible among these modules, every module presents a well-defined standard interface consisting on a set of abstract data types (ADT) and a set of operations that must be taken on them. Every module (network, serial interface, etc.) is modeled as one or more ADTs, and all functionality is modeled as operations on these ADTs. This makes possible to fully upgrade a module without side-effects. The description of this standard interface would exceed this paper purposes.

The greatest deal from the software point of view was the Real-Time kernel. Let us recall that one of our main goals was to reduce power consumption as much as possible. For this purpose, an Hitachi H8/520 Microcontroller was chosen, as described in section 3.1. The kernel must be able to schedule all pending tasks, but, if a “long” period of idle time is detected until the next task activation, the kernel must stand-by the CPU.

This is a hard problem, since the CPU wake-up may take up to 10 milliseconds (depending on CPU clock speed), and CPU can not response for external events except that for NMI during stand-by mode. For this purpose, a real-time operating system was developed for its use in embedded Hitachi's H8-500 family applications: LOPORTOS, which stands for LOw POwer Real Time Operating System.

This operating system has microkernel based architecture [4] and is structured into 3 modules:

- task management
- memory management
- Interprocess communication management.

The memory management is very simple, due to the small amount of memory needed to manage (typically, from 1K to 4K). Memory is managed as a free blocks [7] linked list. Because hardware has no support for it, no garbage collector can be implemented.

The interprocess communication management is also very simple. A single message queue is assigned to every process, identifying it by its process identification. In the other hand, message can be sent to a queue in a synchronous (the sender is locked until the message is read) or asynchronous mode.

The real-time feature relies on the tasking system. This module assigns tasks priorities according to Rate Monotonic Analysis principles, assigning higher priorities to the tasks that serve events with higher occurrence frequency. Interrupts are treated as tasks: every task in the system must declare which interrupts it manages, and when an interrupt occurs, the kernel catches it, identifies the task that manages it, and sends it a message identify-

ing the interrupt, waking it up for execution if the task is currently idle.

The low power consumption feature is also managed by the tasking system. The CPU can program a NMI generator that supplies a fixed but programmable sequence of non maskable interrupts which can wake up CPU from a stand-by state. Every process has a message queue, and express its idle states trying to read a message from its message queue. Every process has a (dynamic) priority associated, calculated from it allowable maximum time response.When a process tries to read a message from its queue and there is no readable message there, the process is marked as “idle”. When the CPU detects that every process in the system is in a idle state, it checks whether it can program the NMI generator to wake up CPU before next process deadline, taking in account the CPU wake-up time. If checked as possible, the NMI generator is programmed and stand-by mode is entered. This keeps the CPU as much time as possible in stand-by mode, meeting all processes deadlines.

4. Results

Some MIDAS prototypes has been built and tested to now. The unitary manufacturing costs estimated for medium-large production series are:

MIDAS MASTER: 50$
MIDAS User Interface (LCD display and numeric keypad): 35$
MIDAS Main Power Supply and Smart battery charger: 25$
MIDAS Wind vane sensor: 100$(anodized aluminum) 25$(PCB)
MIDAS Anemometer: 100$(anodized aluminum) 25$(PCB)
MIDAS Temperature Sensor: 35$
MIDAS 32MB Massive storage module: 300$
MIDAS 16-Channel 16-Bit analog input-module 35$
MIDAS 32-Channel digital input-output module: 25$

The average power consumption of this whole system, working as a typical weather-station application, resulted to be under 5mA.

5. Conclusions and future work

A low cost, low power data acquisition system has been presented. The results we have obtained from the first prototype show that having multiple low power smart sensors connected through a bus network to a master device, on which relies data acquisition tasks, is a feasible way to reduce cost, power consumption and system complexity, increasing in the other hand reliability, flexibility, scalability and robustness.

Future work consists on adding new features like object-oriented master programming, multimaster network system, and multiple logical networks coexistence on physical network.

6. References

[1] Ramón Pallàs Areny. “Sensores y Acondicionadores de señal”. Ed. Marcombo, 1.994. ISBN: 84-267-0989-3.

[2] Day and Zimmermann, “The OSI Reference Model”

[3] Andrew S. Tanenbaum. “Computer Networks”. 3rd edition. Ed. Prentice-Hall, 1996. ISBN 0-13-349945-6.

[4] Andrew S. Tanenbaum et al. “Operating Systems: Design And Implementation” 2nd edition. Ed. Prentice-Hall, 1997. ISBN 0-13-638677-6.

[5] “Discrete Z8 Microcontrollers. Product Specifications Databooks”. Zilog, Inc.

[6] “H8/520 Hardware Reference Guide”. Hitachi

[7] Willian Stallings. “Operating Systems”, 2nd edition. Ed. Prentice-Hall, 1997. ISBN 0-13-180977-6.

MIDAS: An inexpensive, real-time, scalable, network-based Data Acquisition System

Antonio Barbancho and José Antonio Pérez

Departamento de Tecnología Electrónica. Universidad de Sevilla. Escuela Universitaria Politécnica de Sevilla. C/ Virgen de África, 7. 41011 Sevilla. España. Tlf: (34) 5 455 28 37. Fax: (34) 5 455 28 33. Email ayboc@cica.es

Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla. Facultad de Informática y Estadística. Avenida de la Reina Mercedes, s/n. 41012 Sevilla. España. Tlf: (34) 5 455 27 67. Email: jperez@lsi.us.es.

Abstract: At this paper, we present MIDAS. MIDAS is a low-cost, low-power, distributed data acquisition system. MIDAS is based on a bus-like network and a multitasking real-time master. Sensors are connected as slaves and are fully controlled by the network master. Since some prototypes have already been built, we expose practical results on costs and performance obtained.

Résumé: Dans cette communication, nous présentons MIDAS. MIDAS c'est un système distribué d'acquisition de données à bas consommation d'énergie et bas coût. MIDAS utilise une architecture de bus, avec protocole maître-esclave et logiciel multitache temps réel. Les capteurs sont connectés comme esclaves et sont contrôlés par le maître du réseau. Des prototypes ont déjà étés construits, et ici nous présentons des résultats pratiques avec des considérations sur les coûts et les performances obtenues.

1. Introduction

Data acquisition systems are becoming more and more sophisticated, having to fulfil harder requirements about power consumption, cost, scalability and flexibility.

A modern general-purpose data acquisition system should be able to:

- manage a large number of sensors;
- manage very different nature sensors;
- manage long distance located sensors;
- make easy installation and maintenance tasks;
- operate in stand-alone environment, battery-powered;
- store large amount of data;
- perform real time actuation;

all of this, of course, at the lowest cost.

In this paper, we describe MIDAS, a MultIpurpose Data Acquisition System, which complies with all these requirements. MIDAS is a network-based, real-time, low-power, low-cost system. Furthermore, MIDAS is an open system, which can grow up as much as needed. Its most relevant features are:

- Up to 256 elements (sensors, actuators or any other device) can be addressed simultaneously.
- All devices are plugged into an 8-wire bus-based network. So only eight parallel wires are needed to power and control all devices, making easy installation and maintenance.
- Low supply current. This enables to power it from a battery pack with solar panel.
- All sensors have a smart interface, allowing uniform treatment. This simplifies management of different nature sensors.
- All devices are plug and play. Warm plugging is allowed.
- Real time Operating System.
- Flexible programming.

MIDAS is not a commercial system, but it has been designed to be used as a prototype for several commercial systems actually under development.

In section 2 we describe the architectural basics of MIDAS. Section 3 shows hardware and software implementation issues, and section 4 discusses some experimental results obtained from first prototypes. Finally, conclusion and actual investigation lines are presented in section 5.

2. System architecture

MIDAS is based on a real-time bus-like network. Devices connected to this network can be classified as masters and slaves. Only one master is currently allowed in the network. The master device configures and assigns network addresses to slaves. The master acts as datalogger, governing the system operation (see Fig. 1).

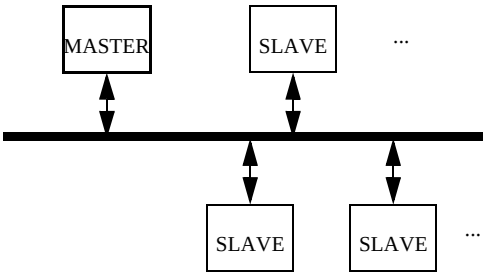

Fig. 1: System architecture

All other devices in the network (intelligent sensors, data storage systems, signal conditioning, user interfaces, and so on) are slaves. All slaves must show a standard interface to the network. If needed, any traditional sensor can be connected to the network through an interface module, allowing low-cost transition from traditional systems.

2.1. Network Architecture

MIDAS network model is based on a simplified OSI model [2] and lacks of network to presentation layers. This means, it only has physical, data link and application layers.

The Physical Layer is concerned with transmitting raw bits over the communication channel [3]. In order to make a low cost design, our design is based on RS-485 and SPI (Serial Peripheral Interface) standards. The communication channel is a bus that carries three signal lines (CLK for clock, DATA for data and SS for slave selection) and two power lines (V+ and ground). Since RS-485 protocol is used, each line is composed of two wires (balanced transmission). This makes possible communication along medium distances (4000 feet) without line repeaters. The communication is synchronous and leaded by the master device. Transmission speed is controlled by CLK and may be up to 10 Mbps. Although RS-485 specifies that up to 32 loads[1] on the bus, our design uses 1/8-unit-load transceivers, allowing 256 devices on the bus.

The main task of the Data Link Layer is to take raw transmission facility and transform it into a line that appears free of transmission errors to application layer [3]. It is also responsible for the identification of the slave. As all devices share the bus, there must be a unique network address for each device. A detailed description of the protocol of this layer would exceed this paper purposes.

The Application Layer is a collection of small protocols providing specific solutions. Examples are file transfer, remote operation and slave management. The later deals with the automatic identification and configuration (Plug'n'Play) and network address assignation.

2.2. Master Architecture

The master is responsible to perform the following tasks:

- Manage the application level network protocol, as described in section 2.1.
- Perform the acquisition process. This is done by executing an user program, as we will describe later.

From the hardware point of view, the master architecture consists on a CPU microcontroller, a network interface, a serial RS-232 interface, a real-time clock and programmable NMI generator, a ROM memory containing embedded code and a nonvolatile SRAM memory bank, as described in Fig. 2.

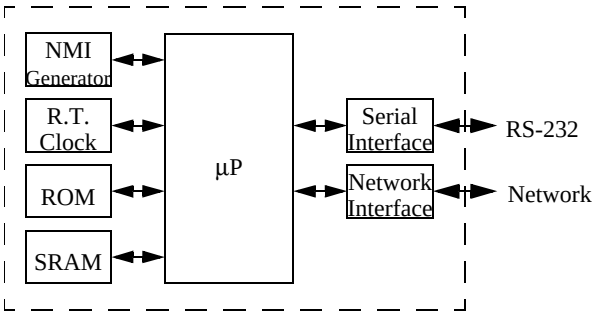

Fig. 2: Master architecture

The purpose of the RAM bank is for device internal use or for data storage. Data storage may be increased via an external device connected to the network, allowing massive storage.

The serial RS-232 interface is conceived for facilitating downloading and uploading programs, debugging and frequent tasks like data retrieving or system monitoring. Although a specialized device connected to the network can be used for this purpose, having this facility integrated into the master has two main advantages:

- The master is self-contained. No other device is needed to control the system. You do not need an additional device to perform elementary tasks. This reduces the cost of a minimum system.
- It reduces data traffic in the network when performing such tasks, reducing congestion risk.

Furthermore, while developing MIDAS first prototypes, this serial interface was very useful for debugging and profiling tasks.

From the software point of view, the master contains four independent modules:

- Real-time kernel, which implements multitasking and real-time scheduling issues. This kernel serves as basic support for all other software facilities, as described in section 3.2.
- Network protocols for physical and data-link layers, and master management protocol for application layer.
- User's interface for RS-232 serial link. Simple Xon-Xoff protocol is used to manage communication.
- User's program interpreting machine. The master executes an user-supplied program which executes on an emulated virtual data acquisition machine.

Software aspects are detailed in section 3.2.

1. A load is the input impedance of a transceiver, and is 12KΩ in the standard.

2.3. Slave Architecture

Since the master has not input/output capability (except for the network and serial links), it has to reside on the sensor end. Therefore slave architecture is very application dependent. Nevertheless, it always must include an interface part to the network. This interface has been designed to be simple, allowing low cost and low power.

3. Implementation issues

3.1. Hardware issues

Two types of cables can be present on a bus: 10-wire IDC ribbon cable for short distances, and 4 twisted pair 24 AWG cable for long distances and special protection. A special connection module has been designed to connect both.

The master device is based on the Hitachi H8/500 microcontroller family. It was selected because of its 16 bits nature, low power, rich set of peripherals and low cost. The first prototypes use the H8/520, a small 64 pin PLCC housed, 32 Kb OTP EPROM and 1 Kb RAM microcontroller. A pair of USART on-chip peripherals allows connection to an RS-232 port and to the network.

A bank of SRAM is also available on the master, allowing up to 512 Kb of local, fast, nonvolatile memory. A lithium battery must be internally or externally connected to power SRAM and real- time clock in the absence of main power.

In order to increase the number of devices that can be present on the bus, 1/8-unit-load RS-485 transceivers are used, allowing up to 256 devices. The MAX3082 was selected due their low power, low cost and fail-safe features. This make possible turning off all drivers to save power when the bus is idle. When a receiver 'sees' a floating line, puts its output in a known state (high).

The whole master is enclosed in a small DIN rail mount enclosure.

Slave architecture is application dependent as noted in section 2.3. The network interface and other common tasks rely on a Zilog Z8 microcontroller [5]. A Z86E06 device was chosen due to its very low cost, very low power, low pin count and SPI capabilities. A unique feature of this device is its ability to act as SPI slave in a compare mode. In this mode, the microntroller is in standby mode, drawing a few microamperes. The SPI peripheral is controlled externally by the master. A transaction begins with the assertion of SS line. Then eight bits are received via the DATA line. This byte is compared with a special purpose register. If the data does not match, the Z8 ignores all data until SS line is reset. If the data does much, the microcontroller leaves standby and continues the transaction.

3.2. Software issues

Since MIDAS complexity mainly relies on the master, this was the main challenge when dealing with software design. A great software engineering effort was taken in order to produce a flexible, reusable, easy to maintain and extend embedded code. As advanced in section 2.2, the master software architecture consists on four slightly coupled modules:

- Real-time kernel, which implements multitasking and real-time scheduling issues.
- Network protocols.
- User's interface for RS-232 serial link.
- User's program interpreting machine.

In order to reduce coupling as much as possible among these modules, every module presents a well-defined standard interface consisting on a set of abstract data types (ADT) and a set of operations that must be taken on them. Every module (network, serial interface, etc.) is modeled as one or more ADTs, and all functionality is modeled as operations on these ADTs. This makes possible to fully upgrade a module without side-effects. The description of this standard interface would exceed this paper purposes.

The greatest deal from the software point of view was the Real-Time kernel. Let us recall that one of our main goals was to reduce power consumption as much as possible. For this purpose, an Hitachi H8/520 Microcontroller was chosen, as described in section 3.1. The kernel must be able to schedule all pending tasks, but, if a “long” period of idle time is detected until the next task activation, the kernel must stand-by the CPU.

This is a hard problem, since the CPU wake-up may take up to 10 milliseconds (depending on CPU clock speed), and CPU can not response for external events except that for NMI during stand-by mode. For this purpose, a real-time operating system was developed for its use in embedded Hitachi's H8-500 family applications: LOPORTOS, which stands for LOw POwer Real Time Operating System.

This operating system has microkernel based architecture [4] and is structured into 3 modules:

- task management
- memory management
- Interprocess communication management.

The memory management is very simple, due to the small amount of memory needed to manage (typically, from 1K to 4K). Memory is managed as a free blocks [7] linked list. Because hardware has no support for it, no garbage collector can be implemented.

The interprocess communication management is also very simple. A single message queue is assigned to every process, identifying it by its process identification. In the other hand, message can be sent to a queue in a synchronous (the sender is locked until the message is read) or asynchronous mode.

The real-time feature relies on the tasking system. This module assigns tasks priorities according to Rate Monotonic Analysis principles, assigning higher priorities to the tasks that serve events with higher occurrence frequency. Interrupts are treated as tasks: every task in the system must declare which interrupts it manages, and when an interrupt occurs, the kernel catches it, identifies the task that manages it, and sends it a message identify-

ing the interrupt, waking it up for execution if the task is currently idle.

The low power consumption feature is also managed by the tasking system. The CPU can program a NMI generator that supplies a fixed but programmable sequence of non maskable interrupts which can wake up CPU from a stand-by state. Every process has a message queue, and express its idle states trying to read a message from its message queue. Every process has a (dynamic) priority associated, calculated from it allowable maximum time response.When a process tries to read a message from its queue and there is no readable message there, the process is marked as “idle”. When the CPU detects that every process in the system is in a idle state, it checks whether it can program the NMI generator to wake up CPU before next process deadline, taking in account the CPU wake-up time. If checked as possible, the NMI generator is programmed and stand-by mode is entered. This keeps the CPU as much time as possible in stand-by mode, meeting all processes deadlines.

4. Results

Some MIDAS prototypes has been built and tested to now. The unitary manufacturing costs estimated for medium-large production series are:

MIDAS MASTER: 50$
MIDAS User Interface (LCD display and numeric keypad): 35$
MIDAS Main Power Supply and Smart battery charger: 25$
MIDAS Wind vane sensor: 100$(anodized aluminum) 25$(PCB)
MIDAS Anemometer: 100$(anodized aluminum) 25$(PCB)
MIDAS Temperature Sensor: 35$
MIDAS 32MB Massive storage module: 300$
MIDAS 16-Channel 16-Bit analog input-module 35$
MIDAS 32-Channel digital input-output module: 25$

The average power consumption of this whole system, working as a typical weather-station application, resulted to be under 5mA.

5. Conclusions and future work

A low cost, low power data acquisition system has been presented. The results we have obtained from the first prototype show that having multiple low power smart sensors connected through a bus network to a master device, on which relies data acquisition tasks, is a feasible way to reduce cost, power consumption and system complexity, increasing in the other hand reliability, flexibility, scalability and robustness.

Future work consists on adding new features like object-oriented master programming, multimaster network system, and multiple logical networks coexistence on physical network.

6. References

[1] Ramón Pallàs Areny. “Sensores y Acondicionadores de señal”. Ed. Marcombo, 1.994. ISBN: 84-267-0989-3.

[2] Day and Zimmermann, “The OSI Reference Model”

[3] Andrew S. Tanenbaum. “Computer Networks”. 3rd edition. Ed. Prentice-Hall, 1996. ISBN 0-13-349945-6.

[4] Andrew S. Tanenbaum et al. “Operating Systems: Design And Implementation” 2nd edition. Ed. Prentice-Hall, 1997. ISBN 0-13-638677-6.

[5] “Discrete Z8 Microcontrollers. Product Specifications Databooks”. Zilog, Inc.

[6] “H8/520 Hardware Reference Guide”. Hitachi

[7] Willian Stallings. “Operating Systems”, 2nd edition. Ed. Prentice-Hall, 1997. ISBN 0-13-180977-6.

MIDAS: An inexpensive, real-time, scalable, network-based Data Acquisition System

Antonio Barbancho and José Antonio Pérez

Departamento de Tecnología Electrónica. Universidad de Sevilla. Escuela Universitaria Politécnica de Sevilla. C/ Virgen de África, 7. 41011 Sevilla. España. Tlf: (34) 5 455 28 37. Fax: (34) 5 455 28 33. Email ayboc@cica.es

Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla. Facultad de Informática y Estadística. Avenida de la Reina Mercedes, s/n. 41012 Sevilla. España. Tlf: (34) 5 455 27 67. Email: jperez@lsi.us.es.

Abstract: At this paper, we present MIDAS. MIDAS is a low-cost, low-power, distributed data acquisition system. MIDAS is based on a bus-like network and a multitasking real-time master. Sensors are connected as slaves and are fully controlled by the network master. Since some prototypes have already been built, we expose practical results on costs and performance obtained.

Résumé: Dans cette communication, nous présentons MIDAS. MIDAS c'est un système distribué d'acquisition de données à bas consommation d'énergie et bas coût. MIDAS utilise une architecture de bus, avec protocole maître-esclave et logiciel multitache temps réel. Les capteurs sont connectés comme esclaves et sont contrôlés par le maître du réseau. Des prototypes ont déjà étés construits, et ici nous présentons des résultats pratiques avec des considérations sur les coûts et les performances obtenues.

1. Introduction

Data acquisition systems are becoming more and more sophisticated, having to fulfil harder requirements about power consumption, cost, scalability and flexibility.

A modern general-purpose data acquisition system should be able to:

- manage a large number of sensors;
- manage very different nature sensors;
- manage long distance located sensors;
- make easy installation and maintenance tasks;
- operate in stand-alone environment, battery-powered;
- store large amount of data;
- perform real time actuation;

all of this, of course, at the lowest cost.

In this paper, we describe MIDAS, a MultIpurpose Data Acquisition System, which complies with all these requirements. MIDAS is a network-based, real-time, low-power, low-cost system. Furthermore, MIDAS is an open system, which can grow up as much as needed. Its most relevant features are:

- Up to 256 elements (sensors, actuators or any other device) can be addressed simultaneously.
- All devices are plugged into an 8-wire bus-based network. So only eight parallel wires are needed to power and control all devices, making easy installation and maintenance.
- Low supply current. This enables to power it from a battery pack with solar panel.
- All sensors have a smart interface, allowing uniform treatment. This simplifies management of different nature sensors.
- All devices are plug and play. Warm plugging is allowed.
- Real time Operating System.
- Flexible programming.

MIDAS is not a commercial system, but it has been designed to be used as a prototype for several commercial systems actually under development.

In section 2 we describe the architectural basics of MIDAS. Section 3 shows hardware and software implementation issues, and section 4 discusses some experimental results obtained from first prototypes. Finally, conclusion and actual investigation lines are presented in section 5.

2. System architecture

MIDAS is based on a real-time bus-like network. Devices connected to this network can be classified as masters and slaves. Only one master is currently allowed in the network. The master device configures and assigns network addresses to slaves. The master acts as datalogger, governing the system operation (see Fig. 1).

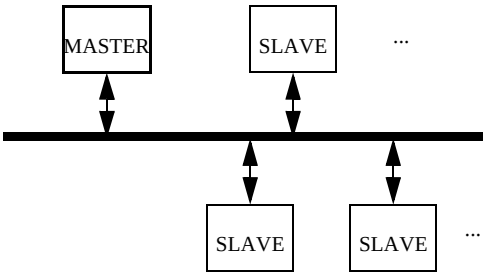

Fig. 1: System architecture

All other devices in the network (intelligent sensors, data storage systems, signal conditioning, user interfaces, and so on) are slaves. All slaves must show a standard interface to the network. If needed, any traditional sensor can be connected to the network through an interface module, allowing low-cost transition from traditional systems.

2.1. Network Architecture

MIDAS network model is based on a simplified OSI model [2] and lacks of network to presentation layers. This means, it only has physical, data link and application layers.

The Physical Layer is concerned with transmitting raw bits over the communication channel [3]. In order to make a low cost design, our design is based on RS-485 and SPI (Serial Peripheral Interface) standards. The communication channel is a bus that carries three signal lines (CLK for clock, DATA for data and SS for slave selection) and two power lines (V+ and ground). Since RS-485 protocol is used, each line is composed of two wires (balanced transmission). This makes possible communication along medium distances (4000 feet) without line repeaters. The communication is synchronous and leaded by the master device. Transmission speed is controlled by CLK and may be up to 10 Mbps. Although RS-485 specifies that up to 32 loads[1] on the bus, our design uses 1/8-unit-load transceivers, allowing 256 devices on the bus.

The main task of the Data Link Layer is to take raw transmission facility and transform it into a line that appears free of transmission errors to application layer [3]. It is also responsible for the identification of the slave. As all devices share the bus, there must be a unique network address for each device. A detailed description of the protocol of this layer would exceed this paper purposes.

The Application Layer is a collection of small protocols providing specific solutions. Examples are file transfer, remote operation and slave management. The later deals with the automatic identification and configuration (Plug'n'Play) and network address assignation.

2.2. Master Architecture

The master is responsible to perform the following tasks:

- Manage the application level network protocol, as described in section 2.1.
- Perform the acquisition process. This is done by executing an user program, as we will describe later.

From the hardware point of view, the master architecture consists on a CPU microcontroller, a network interface, a serial RS-232 interface, a real-time clock and programmable NMI generator, a ROM memory containing embedded code and a nonvolatile SRAM memory bank, as described in Fig. 2.

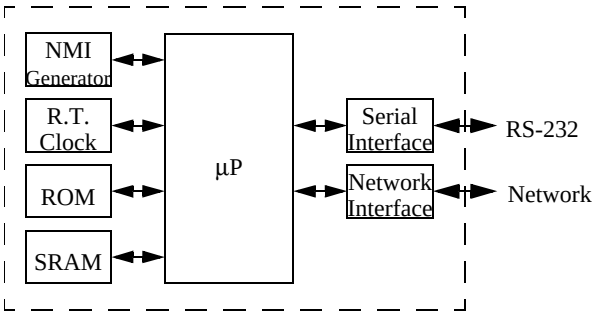

Fig. 2: Master architecture

The purpose of the RAM bank is for device internal use or for data storage. Data storage may be increased via an external device connected to the network, allowing massive storage.

The serial RS-232 interface is conceived for facilitating downloading and uploading programs, debugging and frequent tasks like data retrieving or system monitoring. Although a specialized device connected to the network can be used for this purpose, having this facility integrated into the master has two main advantages:

- The master is self-contained. No other device is needed to control the system. You do not need an additional device to perform elementary tasks. This reduces the cost of a minimum system.
- It reduces data traffic in the network when performing such tasks, reducing congestion risk.

Furthermore, while developing MIDAS first prototypes, this serial interface was very useful for debugging and profiling tasks.

From the software point of view, the master contains four independent modules:

- Real-time kernel, which implements multitasking and real-time scheduling issues. This kernel serves as basic support for all other software facilities, as described in section 3.2.
- Network protocols for physical and data-link layers, and master management protocol for application layer.
- User's interface for RS-232 serial link. Simple Xon-Xoff protocol is used to manage communication.
- User's program interpreting machine. The master executes an user-supplied program which executes on an emulated virtual data acquisition machine.

Software aspects are detailed in section 3.2.

1. A load is the input impedance of a transceiver, and is 12KΩ in the standard.

2.3. Slave Architecture

Since the master has not input/output capability (except for the network and serial links), it has to reside on the sensor end. Therefore slave architecture is very application dependent. Nevertheless, it always must include an interface part to the network. This interface has been designed to be simple, allowing low cost and low power.

3. Implementation issues

3.1. Hardware issues

Two types of cables can be present on a bus: 10-wire IDC ribbon cable for short distances, and 4 twisted pair 24 AWG cable for long distances and special protection. A special connection module has been designed to connect both.

The master device is based on the Hitachi H8/500 microcontroller family. It was selected because of its 16 bits nature, low power, rich set of peripherals and low cost. The first prototypes use the H8/520, a small 64 pin PLCC housed, 32 Kb OTP EPROM and 1 Kb RAM microcontroller. A pair of USART on-chip peripherals allows connection to an RS-232 port and to the network.

A bank of SRAM is also available on the master, allowing up to 512 Kb of local, fast, nonvolatile memory. A lithium battery must be internally or externally connected to power SRAM and real- time clock in the absence of main power.

In order to increase the number of devices that can be present on the bus, 1/8-unit-load RS-485 transceivers are used, allowing up to 256 devices. The MAX3082 was selected due their low power, low cost and fail-safe features. This make possible turning off all drivers to save power when the bus is idle. When a receiver 'sees' a floating line, puts its output in a known state (high).

The whole master is enclosed in a small DIN rail mount enclosure.

Slave architecture is application dependent as noted in section 2.3. The network interface and other common tasks rely on a Zilog Z8 microcontroller [5]. A Z86E06 device was chosen due to its very low cost, very low power, low pin count and SPI capabilities. A unique feature of this device is its ability to act as SPI slave in a compare mode. In this mode, the microntroller is in standby mode, drawing a few microamperes. The SPI peripheral is controlled externally by the master. A transaction begins with the assertion of SS line. Then eight bits are received via the DATA line. This byte is compared with a special purpose register. If the data does not match, the Z8 ignores all data until SS line is reset. If the data does much, the microcontroller leaves standby and continues the transaction.

3.2. Software issues

Since MIDAS complexity mainly relies on the master, this was the main challenge when dealing with software design. A great software engineering effort was taken in order to produce a flexible, reusable, easy to maintain and extend embedded code. As advanced in section 2.2, the master software architecture consists on four slightly coupled modules:

- Real-time kernel, which implements multitasking and real-time scheduling issues.
- Network protocols.
- User's interface for RS-232 serial link.
- User's program interpreting machine.

In order to reduce coupling as much as possible among these modules, every module presents a well-defined standard interface consisting on a set of abstract data types (ADT) and a set of operations that must be taken on them. Every module (network, serial interface, etc.) is modeled as one or more ADTs, and all functionality is modeled as operations on these ADTs. This makes possible to fully upgrade a module without side-effects. The description of this standard interface would exceed this paper purposes.

The greatest deal from the software point of view was the Real-Time kernel. Let us recall that one of our main goals was to reduce power consumption as much as possible. For this purpose, an Hitachi H8/520 Microcontroller was chosen, as described in section 3.1. The kernel must be able to schedule all pending tasks, but, if a “long” period of idle time is detected until the next task activation, the kernel must stand-by the CPU.

This is a hard problem, since the CPU wake-up may take up to 10 milliseconds (depending on CPU clock speed), and CPU can not response for external events except that for NMI during stand-by mode. For this purpose, a real-time operating system was developed for its use in embedded Hitachi's H8-500 family applications: LOPORTOS, which stands for LOw POwer Real Time Operating System.

This operating system has microkernel based architecture [4] and is structured into 3 modules:

- task management
- memory management
- Interprocess communication management.

The memory management is very simple, due to the small amount of memory needed to manage (typically, from 1K to 4K). Memory is managed as a free blocks [7] linked list. Because hardware has no support for it, no garbage collector can be implemented.

The interprocess communication management is also very simple. A single message queue is assigned to every process, identifying it by its process identification. In the other hand, message can be sent to a queue in a synchronous (the sender is locked until the message is read) or asynchronous mode.

The real-time feature relies on the tasking system. This module assigns tasks priorities according to Rate Monotonic Analysis principles, assigning higher priorities to the tasks that serve events with higher occurrence frequency. Interrupts are treated as tasks: every task in the system must declare which interrupts it manages, and when an interrupt occurs, the kernel catches it, identifies the task that manages it, and sends it a message identify-

ing the interrupt, waking it up for execution if the task is currently idle.

The low power consumption feature is also managed by the tasking system. The CPU can program a NMI generator that supplies a fixed but programmable sequence of non maskable interrupts which can wake up CPU from a stand-by state. Every process has a message queue, and express its idle states trying to read a message from its message queue. Every process has a (dynamic) priority associated, calculated from it allowable maximum time response.When a process tries to read a message from its queue and there is no readable message there, the process is marked as “idle”. When the CPU detects that every process in the system is in a idle state, it checks whether it can program the NMI generator to wake up CPU before next process deadline, taking in account the CPU wake-up time. If checked as possible, the NMI generator is programmed and stand-by mode is entered. This keeps the CPU as much time as possible in stand-by mode, meeting all processes deadlines.

4. Results

Some MIDAS prototypes has been built and tested to now. The unitary manufacturing costs estimated for medium-large production series are:

MIDAS MASTER: 50$
MIDAS User Interface (LCD display and numeric keypad): 35$
MIDAS Main Power Supply and Smart battery charger: 25$
MIDAS Wind vane sensor: 100$(anodized aluminum) 25$(PCB)
MIDAS Anemometer: 100$(anodized aluminum) 25$(PCB)
MIDAS Temperature Sensor: 35$
MIDAS 32MB Massive storage module: 300$
MIDAS 16-Channel 16-Bit analog input-module 35$
MIDAS 32-Channel digital input-output module: 25$

The average power consumption of this whole system, working as a typical weather-station application, resulted to be under 5mA.

5. Conclusions and future work

A low cost, low power data acquisition system has been presented. The results we have obtained from the first prototype show that having multiple low power smart sensors connected through a bus network to a master device, on which relies data acquisition tasks, is a feasible way to reduce cost, power consumption and system complexity, increasing in the other hand reliability, flexibility, scalability and robustness.

Future work consists on adding new features like object-oriented master programming, multimaster network system, and multiple logical networks coexistence on physical network.

6. References

[1] Ramón Pallàs Areny. “Sensores y Acondicionadores de señal”. Ed. Marcombo, 1.994. ISBN: 84-267-0989-3.

[2] Day and Zimmermann, “The OSI Reference Model”

[3] Andrew S. Tanenbaum. “Computer Networks”. 3rd edition. Ed. Prentice-Hall, 1996. ISBN 0-13-349945-6.

[4] Andrew S. Tanenbaum et al. “Operating Systems: Design And Implementation” 2nd edition. Ed. Prentice-Hall, 1997. ISBN 0-13-638677-6.

[5] “Discrete Z8 Microcontrollers. Product Specifications Databooks”. Zilog, Inc.

[6] “H8/520 Hardware Reference Guide”. Hitachi

[7] Willian Stallings. “Operating Systems”, 2nd edition. Ed. Prentice-Hall, 1997. ISBN 0-13-180977-6.