HARDWARE DESIGN AND PROTOCOL SPECIFICATION FOR THE CONTROL AND COMMUNICATION WITHIN A MECHATRONIC SYSTEM

André Luiz de Freitas Francisco[1], Achim Rettberg[2] and Andreas Hennig[2]
[1]University of Paderborn – MLaP, Pohlweg 98, D-33098 Paderborn, Germany,
Tel:. +49 5251 605576, Fax: + 49 5251 605579,
Email: Andre.Francisco@MLaP.de;
[2]University of Paderborn – C-LAB, Fuerstenallee 11, D-33102 Paderborn, Germany,
Tel:. +49 5251 606110, Fax: + 49 5251 606065,
Email: Achim.Rettberg@c-lab.de, Andreas.Hennig@c-lab.de

Abstract: This paper describes a communication system for the test track that will be used to develop the components of a new train concept, the RailCab. The demand for flexible hardware architecture and optimized communication channels motivated this project. The aim of the network is to control frequency converters for linear motor sections placed along the track. The vehicles that are moving along the track at a given time will issue the commands to each motor. In order to optimize the communication functions, new hardware components and a new protocol were designed, so that modules accessing network are able to do this in a more direct way.

Key words: Mechatronic systems, hardware design, protocol specification, control systems, communication systems.

1. INTRODUCTION

Stand-alone systems are increasingly becoming obsolete. For some applications, they are considered useless in comparison to the distributed ones, the latter being able to use all the features that are present on each element of a network. Distributed systems are especially useful in constructing complex systems that comprise different function modules; they offer the possibility of sharing resources and can be used together nodes that can cooperate to accomplish a certain task.

Although the number of communication interfaces available on the market has largely increased in the past decades, there are still new applications which make specific demands that cannot be completely fulfilled by the available standards in terms of cost/benefit or even particular technical characteristics [9]. The special case to be described in this work requires a protocol that has to guarantee very short latency times (some microseconds at most) but can sometimes dispense extremely high bandwidths as the packets transmitted are relative small and the number of nodes is restricted.

The case study presented here is the RailCab [1] test track. The RailCab, developed at the University of Paderborn, is an innovative variation of conventional trains and based on individual, autonomous shuttles that can be ordered by a client to transport him/her directly from one city to another. The service being personalized, there is no need for the passenger to change trains or wait for fixed departures: the shuttle just sets off at the defined time and location and travels to its destination non-stop. This technology is based on linear motors which are used to move and brake the shuttles and can be installed on existing railroad tracks. The whole track is made up of a succession of linear-motor sections which can be controlled individually. For the shuttle to run smoothly and at a defined velocity, a linear motor has to be synchronized to its respective neighbors as the vehicle is passing over them. However, not only the motors have to be synchronized but their current must also be controlled by a local ECU. To provide these features, a networking system is required that can be optimized to this application. The aim of this paper is to propose a concept to be used with the first prototypes on the test track and to present the basic protocol and hardware platform required to accomplish this task.

2. TEST TRACK

The RailCab test track was built up at the University of Paderborn on a scale of 1:2.5 and has a total length of about 530 meters. It is made up of the rail track itself and 83 linear-motor sections, each one controlled by a servo device; all of these are distributed to 4 stations along the track. Additionally, there is a control room where the track functions are coordinated and monitored. Figure 1 represents the track.

The servos, or frequency converters, are made up of a processing unit and power electronics. The processing unit is the component where the current controller for the servo runs. This unit also provides communication interfaces (CAN-bus and RS-485) to other equipments. At present, the entire network operates using the CAN-bus interface which allows an external system to set the reference values for the motor frequency, amplitude, and phase. However, the RS-485 (SSI) interface is faster and provides features that surpass those of the CAN-bus. Extra features include, e.g., the possibility to let the controllers run externally, using only the servo sensors for reading signals and writing values for high voltages to registers in the frequency converter that is then in charge to generate the power outputs. Due to its advantages, the RS-485 servo interface was chosen for the network described in this paper.

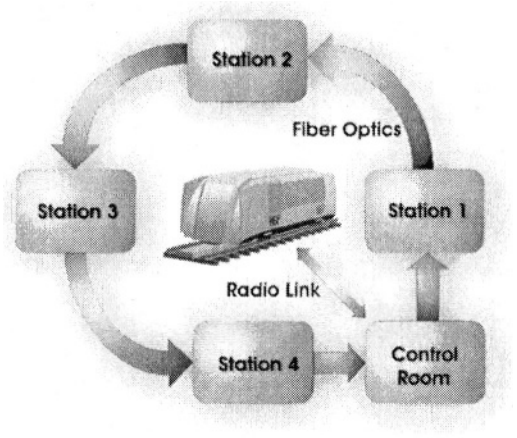

Figure 1. RailCab test track

Networking between the control room and the servo stations is done via a fiber optic (see [6], [7]) channel where all five nodes are interconnected by a point-to-point link that represents a ring topology (fig. 1).

So far two communication interfaces that are going to be used for the actual implementation were described: RS-485 for the servos and fiber optics between the stations. For these links to communicate with one another, additional interfaces and hardware are required and will be described in the following sections.

3. NETWORK FUNCTIONALITIES

The proposed network is intended to gradually provide new functionalities according to the following three steps:

1. Establishment of a networking system that, from a remote operation point, will be able to distribute reference values to all linear motors on the track. These reference values are set by a pre-defined unit located at the control room. At this stage the timing requirements are not as strict

as those for the following two steps once the current controllers run on the servos themselves.

2. Disabling of the internal current controllers of the servos and implement them using an external ECU. This will make possible the tests of new control strategies for the linear motors - one of the tasks of the RailCab development team. To implement this second stage the Motorola PowerPC MPC555 microcontroller will be used. A set of four boards, each containing one of these processors, is going to be connected to the network, initially in a remote position, to control the entire track. Some prior tests have shown that implementation of the current controllers at sampling rates of 8 kHz requires almost the entire microcontroller capacity; as a result, the four boards can simultaneously control up to 4 linear-motor sections. Although there are many more sections to control, four boards should be sufficient because initially there will be simultaneously no more than two shuttles on the track, and each one of them requires a maximum of only two sections operating at a given time (considering that the vehicle is passing from one section to the next). This illustrates a scenario in which the processing power is scheduled dynamically to the active motors, in an "on demand" manner. The immediate advantage is that the number of hardware elements needed to implement the controllers is drastically reduced; on the other hand, though, the communication requirements will increase significantly because the controllers are in a remote position and have to use the network to transfer the servo's input/output data.
3. The third step is meant to minimize the communication drawbacks by distributing the controllers evenly along the track, thus reducing demands on the network bandwidth and creating segments that can make safety or operability decisions autonomously if, for example, the connection to the respective neighbors is interrupted. Another possibility here is to build the ECUs using FPGAs as processing elements. Such devices are appropriate for fast and parallel computations and could therefore be used to implement the current controllers.

Now that the basic idea for the network has been expounded, we can go on to propose appropriate hardware devices.

4. HARDWARE PLATFORM

The hardware described here is based on the Rabbit system [2], [8]. The Rabbit platform is intended for the rapid prototyping of distributed mechatronic systems and made up of three main boards: one with an MPC555 PowerPC, another with a Xilinx Virtex-E FPGA, and the last including a FireWire (IEEE 1394) communication interface. The new processing hardware developed is to be used as an ECU for the current controllers and makes use of the concept of microcontroller/FPGA integration. Although the FireWire module is no longer available on the new hardware, there are also some new communication interfaces, such as 4 LVDS (low-voltage differential signaling) [10] ports and a 10/100 Mbps Ethernet port. Additional features include the possibility for the microcontroller to program the FPGA or its Prom, as well as the FPGA to write the MPC555 memories. This is especially helpful if some reprogramming tasks must be performed remotely (some dozens of meters away), which is the case for the test track. The diagram of this board is shown in figure 2:

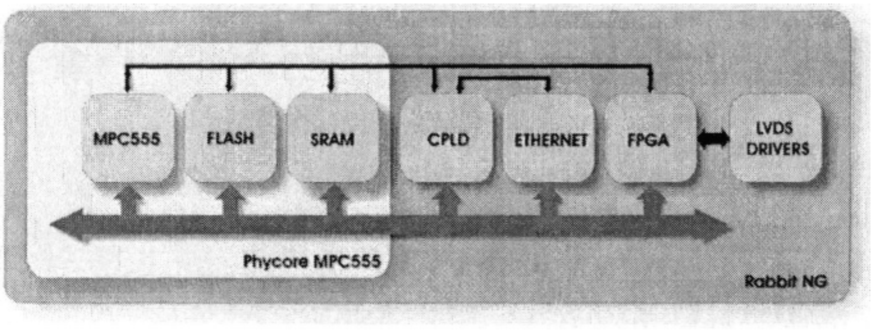

Figure 2. Rabbit NG platform

Communication among all the boards takes place mainly via the LVDS interface. This serial standard offers high signaling rates, but is restricted to just about 40 Mbps for each pair of wires on the boards for this system. Each one of the so-called LVDS ports available on the board contain actually four differential pairs; thus the total throughput for each of the four ports is about 160 Mbps. The reason for using this standard is that it is a merely physical one, i.e., the entire protocol that is being used over it can be implemented to met the requirements of specific applications. As the network proposed in this paper basically defines a new protocol, the choice of the LVDS standard is justified. All LVDS interfaces are connected to the FPGA, where the entire protocol is actually implemented. The programming language chosen for this purpose was VHDL.

As stated above, communication infrastructure between the stations on the track uses fiber optics only. In order to transmit the data of the LVDS interface transparently via optical fibers an adapter board was also built up. With this solution, it will not matter to the user if electrical or optical cables are used to connect two boards. Fiber optics is a good solution for the link between the stations because the involved distances are rather high for electrical cables and also due to the fact that the linear motors' electromagnetic fields may cause interferences on electrical cables.

At this point, there is still the need to connect the motor servos to the network. Because the servos have their own communication standard, a third board (called SSI) must also be developed. This board operates as a bridge between the LVDS (network) and RS-485 (servo) interfaces. Once again an FPGA is used as the processing element for this task.

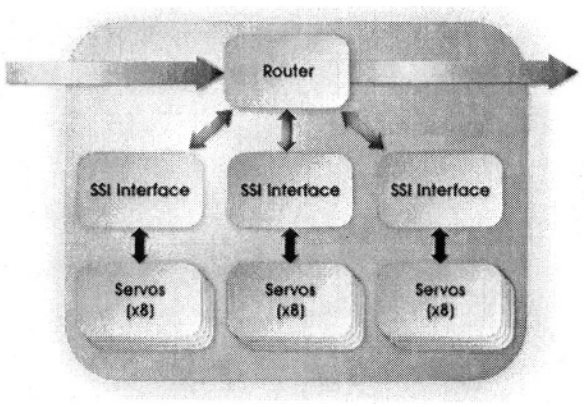

Figure 3. Architecture of a station

Now that we have described the boards, we need an architecture to combine them. The architecture of the four servo stations is shown in figure 3. For these stations, each SSI board has connections for up to 8 servos and a LVDS link to a router (fig. 3). The router is actually the MPC555 board with a LVDS-to-fiber optics, but is represented as a single block for simplification.

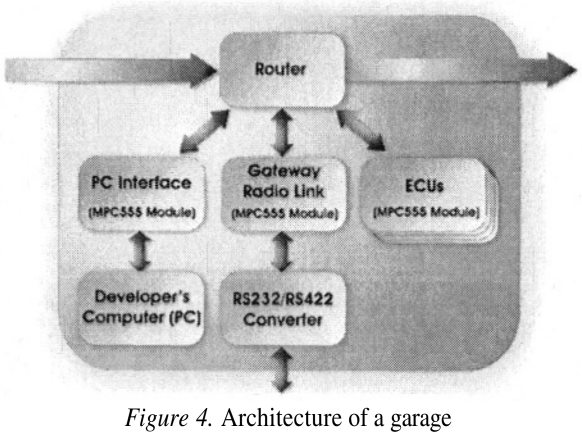
Router
PC Interface (MPC555 Module)
Gateway Radio Link (MPC555 Module)
ECUs (MPC555 Module)
Developer's Computer (PC)
RS232/RS422 Converter

Figure 4. Architecture of a garage

The architecture for the control room is different (figure 4) because its functions differ from those of the servo stations. The basic difference is that the ECUs are for the first development phase centralized in the control room. Additionally, one processing board operates as a gateway for a radio link between the test track and the shuttles and another one is used as an interface to a developer's PC via Ethernet communication.

5. PROTOCOL SPECIFICATION

All the messages in the proposed network fit into one packet. A message is sent only once and will not be repeated even if an error is detected. A node address serves as a unique identifier for each component on a network. Thus, as this project has many similarities with standard networks, it resembles sometimes the Internet Protocol Addressing [3] strategies. The messages that pass through the network can have different purposes. For this reason, four different types of messages were defined: emergency, synchronization, data, and maintenance. The priority scheme is very simple: it considers just the type of message and the order of arrival. As regards priority, the order of the messages from the highest to the lowest is: emergency, synchronization, data, and maintenance.

There are many techniques to detect garbled messages. Three of them are the most common ones used in data communications: Checksum, Error Detection, and Error Correction. Check Sum is a method that uses module summation to detect errors in a stream of data. The problem is that the probability of not identifying an error is equal to the sum of all bits in the message. Error correction can be implemented in two different ways: error correction by itself, in the form of Forward Error Correction (FEC), and strategies such as Automatic Repeat Request (ARQ) that function in combination with a retransmission of corrupted data. Although the first is a good technique, error detection is generally preferred. The main reason is that the number of overhead bits required to implement error detection is much smaller than the number of bits needed for correction [4]. The CRC is a very powerful error detection technique used to obtain data reliability and is easy to implement. In order to employ this technique, the transmitter appends an extra n-bit sequence to every frame. In the present project, it was chosen the 16 bit standard polynomial, the "CRC-16".

To receive data from the memory and send it to the network, it is necessary to modify the data into the correct data format. Another task that has to be implemented is retrieval of the line status, which is used by the memory controller. The main modules specified to fulfil those tasks are the following: *Receiver, Transceiver, Manager, Switching Matrix,* and *Registers* (see figure 5).

The *Receiver* receives the data stream and store 16 bits segments in a FIFO. Furthermore, it has a ready/acknowledge handshaking mechanism with the Switch Manager. When the *Receiver* sets "ready", the priority of the current block is available at the outputs. When a CRC error is detected, the packet is ignored, the FIFO is emptied and the *Receiver* waits for a new packet.

The *Transceiver* transmits the contents of the four message types according to the given priority, if messages are available. It is ensured that the message with the highest priority is transmitted first. It is not necessary to calculate the CRC because it is already calculated by the manager.

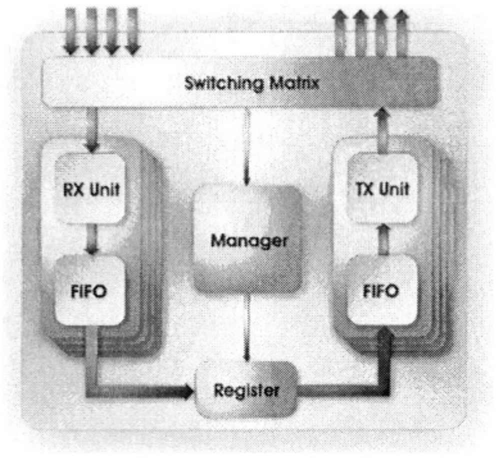

Figure 5. Modules of the protocol implementation

The *Switching Matrix* is necessary to speed communication up. Once the destination address of an incoming packet has been received, and the respective output channel is idle, the data stream is directly sent to the destination.

The *Manager* receives requests from the *Receiver* (RX) and the *Switching Matrix* and then directs the received packet to the transmit memory of the *Transceiver* via an internal bus. The *Registers* are necessary to store the packets internally.

5.1 Memory Management

The RAM memory available in FPGAs for data storage is limited (e.g., Spartan-IIE™ FPGAs range from 32 to 288 kbits) [5]. For this reason, it is necessary to manage the memory to get the most advantage of each free bit space. Many generic algorithms could be implemented to this purpose. Most of them are based in linked queues, binary trees, or even mapping tables that contain information about each space in the memory.

The memory management of this project should be simple. Besides choosing the best algorithm to fit the project needs, it was necessary to think about logical space constraints in FPGAs. Each *Receiver* has only one FIFO to store the incoming messages. A *Transceiver* needs four FIFOs, one for each message type. For the *Transceiver,* a method based on queues was created, considering all the constraints and features of the specific protocol and available hardware configuration. This method uses the memory and takes care of the performance of the concurrent running processes.

Figure 6 illustrates the way the memory management works within the *Transceiver.* For each type of message, one FIFO was created for each transmission channel. The horizontal lines in figure 6 represent the FIFOs, considering the whole illustration as a matrix. The memory manager uses the next available space on the FIFO to store the message. To specify the destination, the *Manager* reads the destination address from the incoming packets and directs it to the appropriated output FIFO. A transmission channel pools the FIFOs, starting from the highest to the lowest priority. After the message is send it is popped from the FIFO.

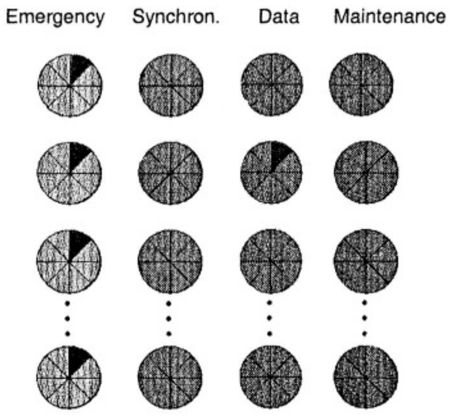

Figure 6. Memory management scheme

5.2 Implementation

All the VHDL code was developed using the Xilinx™ ISE 6.2 software. To simulate these designs Synopsys™ and ModelSim™ were employed. The MPC555 software was developed using Metrowerks™ Codewarrior 6.5 compiler.

6. CONCLUSION AND FUTURE WORK

The present paper briefly described the RailCab project. It also presented the hardware, including the protocol to control the RailCab test track. In combination with the hardware boards, it was possible to specify a software protocol and a management of data passing through a network. All these features were implemented by means of VHDL, with access to the higher-level function enabled by a microcontroller programmed in C. The standards used in the physical layer were LVDS, Fiber Optics, and RS-485.

The approach presented has supplied all the bases for the specifications. After putting the communication system in operation, the experience gathered testing mechatronic functions can be used as a feedback in view of the design of a system for a 1:1 realization, which surely has many more requirements (e.g., safety and extensibility). In addition, the flexibility of the hardware allows scenarios including different topologies and protocols, which are naturally important when implementing new approaches and evaluating trade-offs.

REFERENCES

[1] University of Paderborn. “*Neue Bahntechnik Paderborn* ”. 2002.

[2] Zanella, M., et. al. “*RABBIT: A Modular Rapid-Prototyping Platform for Distributed Mechatronic Systems*”. In: Proc. of the 14th Symposium on Integrated Circuits and Systems Design; Brasília, Brazil, 2001.

[3] Black, Uyless. “*Data Networks*”. Prentice-Hall, NJ, 1989.

[4] Williams, Ross N. “*A Painless Guide to CRC Error Detection Algorithms*”. *Rocksoft Technical Report,* 1993.

[5] Xilinx Inc. “*Spartan-II 2.5V FPGA Family Manual*”. San José, CA, 2000.

[6] Radiant Communications Corporation. “*Design Guide – Fiber Optic Basics*”. South Plainfield, NJ, 2002.

[7] Agilent Technologies. “*Fiber Optic Components Cookbook*”. Palo Alto, CA, 2002.

[8] De Freitas Francisco, André Luiz. “*Realization of controllers for mechatronic systems using FPGAs*”. Bachelor-Thesis, University Paderborn/MLaP, 2001.

[9] Helmers, Scott A. “*Data Communications: A Beginner’s Guide to Concepts and Technology*”. Prentice-Hall, NJ, 1989.

[10] National Semiconductor. “*LVDS Owner’s Manual*”. Santa Clara, CA, 2000.