A Dynamically Reconfigurable CAN System

Pedro Fonseca[1], Fernando Santos[2], Alexandre Mota, José Alberto Fonseca[*]

Abstract

Code for embedded systems is usually developed on a host system and then downloaded to the target. Several solutions exist to do this; most of them require plugging some hardware on the target system and connecting it to the host. Although adequate for mono-processor systems, they become ackward for distributed systems, mainly for two reasons. Firstly, several targets are required to be connected simultaneously to a single host. Secondly, targets may not be physically close to the host.

One solution is to use the existing network to provide communication between host and targets. We have developed a system based on this solution over a CAN network. The stations are able to receive the code, store it and start execution, one of the stations being the gateway to the host. Our system allows the user to control the whole distributed system, downloading code to any of the stations and starting and stopping its execution. This is provided at low cost, with no additional hardware required.

1. Introduction

Embedded systems development is a multidisciplinary trade. The boundary between hardware and software is vague. In some cases, both hardware and software evolve simultaneously through several development stages. As embedded applications strive for hiding the computer inside them, user interface is primitive (if exists…) and debugging facilities are close to non-existent. The features of these systems make embedded systems development a distinct discipline on its own.

Several tools exist for the development of embedded systems. Most of these tools are oriented towards the case of stand-alone, single processor systems. This is a rather simple environment when compared to distributed embedded systems, where many sites, running possibly different versions of the program and physically distant, must co-operate to achieve a common goal. During the development phase, successive versions of the software must be loaded on multiple sites. It is not impossible that the same modification has to be performed on all sites at same time, thus requiring the replacement of the software in every one. This has to be performed in a co-ordinate fashion, guaranteeing that at any time, the versions that run on the different sites are compatible.

We present a solution to assist the development of embedded distributed systems, allowing the user to control the whole system, downloading code to any of the stations and starting and stopping its execution. This is provided at low cost,

[1] Pedro Fonseca was partially supported by Portuguese Government grant PRAXIS XXI BD/3729/94.
[2] Fernando Santos was supported by Portuguese Government grant PRAXIS XXI BM/12654/97.
[*] Sistemas Electrónicos Distribuídos, Departamento de Electrónica, Universidade de Aveiro , 3810 AVEIRO, Portugal
Tel. +351 34 370984 Fax +351 34 381128 Email: pf@ua.pt URL:http://www.ua.pt/det/sed

with no additional hardware required. In section 2 we present the problems of developing embedded systems and, in section 3, the particular case of distributed embedded systems. In section 4 we present our proposal, which is detailed in section 5. Proposals for further work are presented on section 6 and we conclude on section 7.

2. The development of embedded systems

Embedded applications software is, at the present time, mostly developed on a “host" system, such as a PC or a workstation, which is able to provide an user interface that does not exist on the target system. The host computer runs a cross-compiler and a linker, that generate code to be executed on the target system.

The code generated within the host system must be run on the target system (the embedded system). Only this allows the code to be tested on real (or close to real) conditions. Several solutions exist to perform this task. In this section, we will review the most commonly available [1]:

- EPROM programming
- EPROM emulation
- In-circuit emulation
- On-board monitor
- Code loader

EPROM programming is the "real-thing". It uses all the hardware required for a stand-alone system and it can be used with any processor that fetches instructions from external memory. This solution can become extremely tedious and testing facilities are obviously minimal. The use of external logic analysers is possible, but rarely an user-friendly choice.

Another possibility is the use of EPROM emulators. The native code can be downloaded to an EPROM emulator, which behaves like the code memory to the target system. It is targeted to microprocessors that read the code from external memory. It may have memory access timing problems. It requires one EPROM emulator for each target system. Debugging capabilities are also minimal but testing program changes is much easier and quicker.

When looking for a good insight at the target system behaviour, In-Circuit Emulators (ICE) can be used. With an ICE the target system microprocessor is replaced by specific hardware, that emulates it on real time. The cost of this solution is usually high, namely when different microprocessors are to be emulated.

The other two solutions use small resident programs that communicate with the host (usually through a serial interface). The On-board monitor allows the user to download and debug code, as well as verify the contents of the processor registers and memory locations. It can provide a good insight into the target behaviour at a reasonable cost. As processor registers must be accessed (*e.g.*, to provide step-by-step execution), the portability of a monitor is very small.

A code loader does nothing but wait for the host to download the code to the target system. Once this is done, it will wait for a signal to start the execution of the downloaded code. It usually provides better portability than monitors, at the cost of reduced target system insight and of some additional hardware.

3. Embedded distributed systems development

The solutions that have been presented for embedded systems development become awkward for the particular case of distributed systems. This is due mainly for two reasons:

- the multiplicity of the targets; and
- their distance from the host.

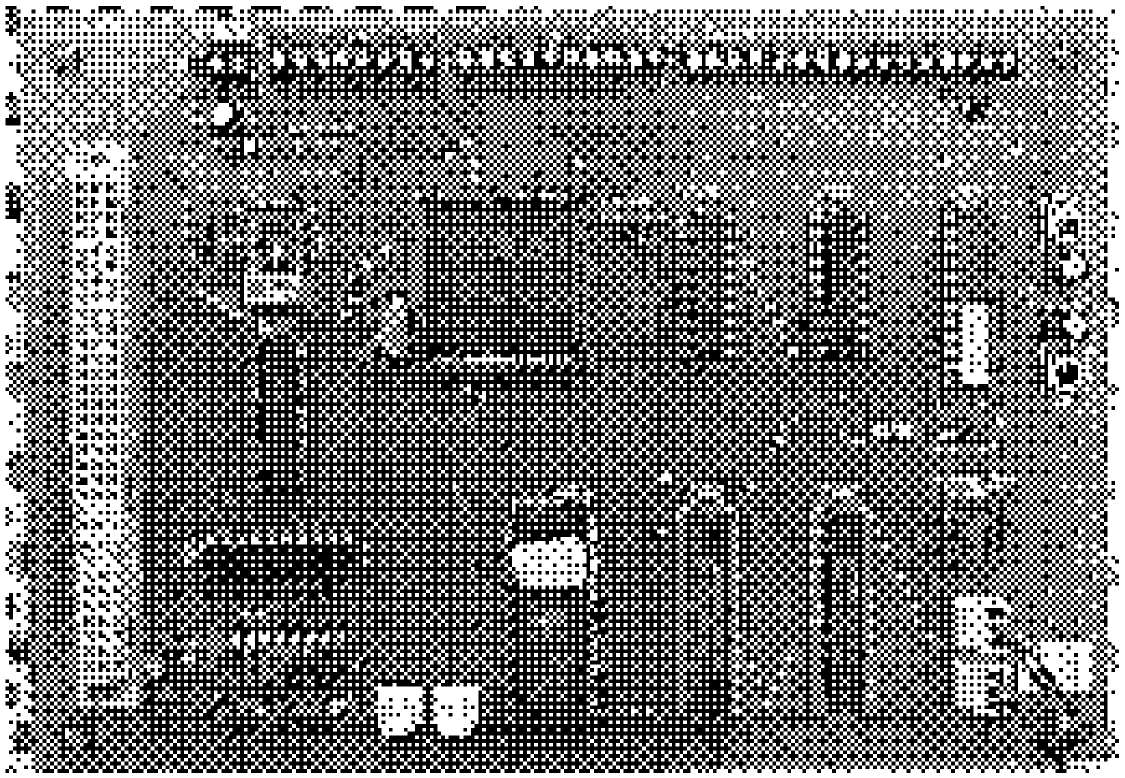

Figure 1 - CANivete board

The first point concerns the number of sites where the code must run (either by installing an ROM with the programme or by connecting these sites to the host). Most of the methods we have presented are now unsuitable. Handling EPROMs with continuously changing versions of the software is already difficult for a single site; in a distributed system, with changes required in several sites, things get even worst. Emulators (EPROM emulators and ICEs) become expensive solutions, as each site requires one. Still, we have the problem of connecting several emulators to a single host, which may not be always possible.

The physical distance from the targets to the host can make things even more difficult. The targets may not be accessible in order to replace the EPROMs at each new test version; emulators require plugging some hardware in and connecting it (by means of a cable, or a radio link, ...) to the host.

We must therefore look for a solution that allows testing software that evolves continuously, in a multiplicity of sites. The main specifications for our system are:

- being able to download code from one host to multiple targets;
- allowing targets to be physically distant from the host.

Other characteristics may be envisaged as desirable, such as: testing the target systems (specially in situations where the targets are not easily accessible), controlling the execution of the program from the host (namely, starting and stopping the program), ...

Distributed systems are built on top of a network. This network can be used to provide the communication between the targets and the host. The advantages of this approach are:

- no extra hardware is required (no plug-ins);
- the communication between targets and host is provided by an existing infrastructure.

We are then left with two possible approaches: monitors and code-loaders. Our solution uses a code-loader over a

CAN network. The advantages envisaged in this solution were:

1. simplicity (code loaders are simpler to develop than monitors and the time to develop a working prototype would be shortened); and
2. the possibility of having a system that would execute code that was directly "prommable" (code to run in monitor systems must usually be relocated, in order to leave room for the monitor itself, that continues to reside in program memory).

Incidentally, note that a code loader can load a monitor; one solution does not preclude the other.

4. CANivete: a CAN based solution for distributed systems development

The CANivete board (fig. 1) is a fundamental part of a system aimed at providing some help to the designer of embedded distributed systems. Each CANivete board is a node on a CAN network, that is capable of receiving the program to execute from the CAN interface and start and stop its execution. In this way, a large number of nodes of an embedded distributed system (30 in the current version) can be connected to a single host machine, such as a PC or a workstation. The CANivete boards are also referred to as the “targets”.

The board has two modes, “download” and “run”. After power on, the board is in download mode. In this mode, every node is waiting to receive the program code from the CAN network. After the code is received, the board prepares itself to change to run mode, which will happen after a system Reset.

The CANivete system is currently being used as a development tool within the *Sistemas Electrónicos Distribuídos* ("Electronic Distributed Systems") group.

5. System description

A possible physical system configuration for a CANivete system is shown in figure 2.

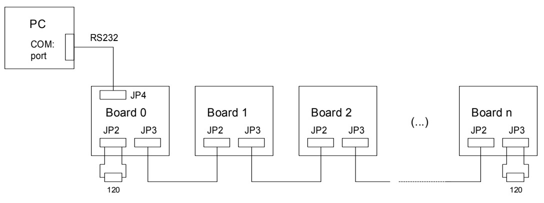

Figure 2 - System architecture

The PC (the host) contains the user interface software. Normally, the development system would be located at the PC too.

Each CANivete board bears an ID number, which must be unique within the system. Board no. 0, the gateway board, provides the communication between the host and the remaining boards. The ID numbers of the boards do not need to be consecutive. The requirements concerning board IDs is that board 0 must be connected to the PC, and the ID numbers of all other boards must be unique.

A minimal system would consist of the host, the gateway board and one remote board. (An "hyper-minimal" system would consist of the PC and the gateway, but no use could be done of the CAN network).

5.1. Hardware

The CANivete board is based on the 80C592 micro-controller, from Philips ([5]). This is a variant of the 80C51, to which a CAN controller (amongst other things) was added. Like all the members of his family, this controller uses separate memories for program and data. Program memory is a read-only memory (there are no instructions for writing in the program memory space).

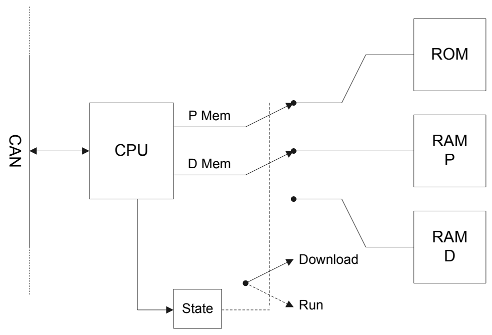

Figure 3 - CANivete working principle.

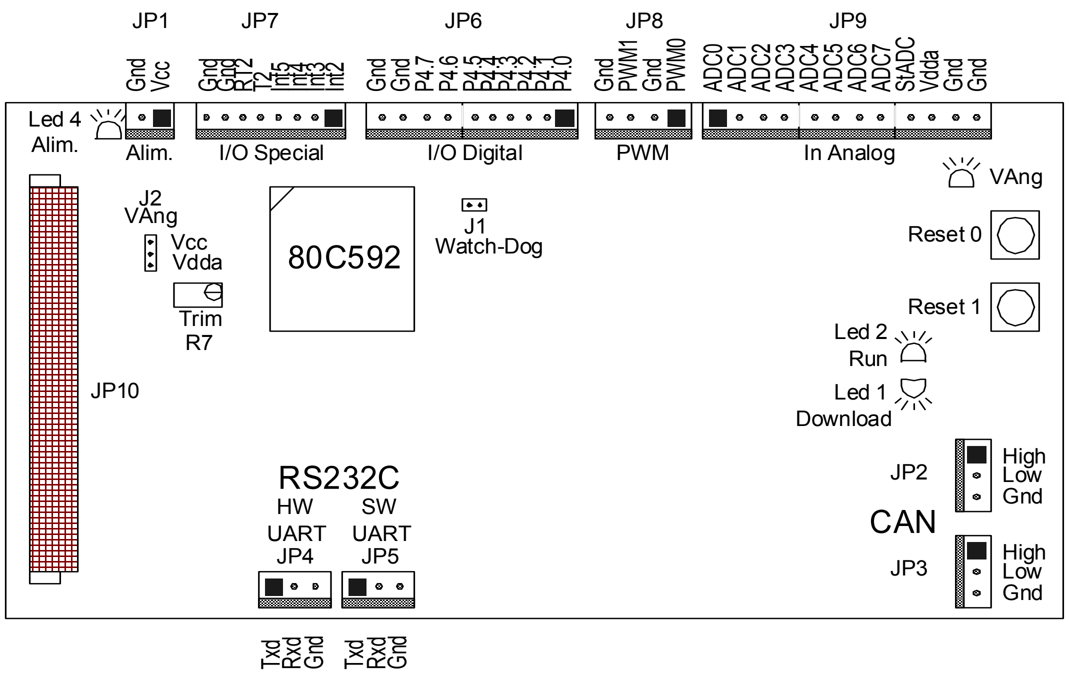

Figure 4 - CANivete board layout

The working principle of the CANivete board is depicted in fig. 3. ROM is the memory containing the CANivete target board software and RAM_P and RAM_D are two 32K bytes RAMs. In "Download" mode, the controller sees the ROM as program memory and RAM_P as data memory; after RESET, it will start executing the program stored in ROM. This program tells it to wait for the program to be downloaded through the CAN network and to store it in the data memory (RAM_P). After the code has been received, the board prepares itself to change to "Run" mode, which will happen after the next Reset. In "Run" mode, the controller sees RAM_P as program memory and RAM_D as data memory; it will then execute the program that was previously stored in RAM_P.

This arrangement allows the system to be able to remotely download code and start its execution, using software that is directly "prommable". This has two advantages. First, prommable code is the standard way of generating code in any compiler targeted for embedded systems. Second, the transition from the development platform to the working system (where the code is stored in a ROM) is straightforward.

5.2. Address space

From the users point of view, the system contains 32k bytes of program memory and 32k bytes of data memory. A range of 32k bytes on the upper half of the addressable data space is reserved for peripherals (Table 1). The address space for peripherals consists of 128 identical blocks of 256 addresses.

Addr. range	Code	Data
0000-7FFFh	Program (32k)	Data (32k)
8000-FFFFn	(Not available)	Peripherals (256)

Table 1: Memory usage

5.3. Interface

The CANivete board contains a wide set of input and output signals, both digital and analogue. These are provided to the user in several connectors (Fig. 4) and they include general purpose I/O, digital and analogue, special function I/O (interrupts, …) and communication ports (CAN and RS-232). A detailed description can be found in [4].

5.4. Software

PCLoad program is the user interface at the host. In the current version, PCLoad allows downloading the code to the targets. Commands for starting the program at the target are not yet available, as the target currently reconfigures itself to start executing the code received after a successful download.

PCLoad uses one of the serial ports as the default communication port. PCLoad uses the following syntax:

```
PCLoad progfile Station_ID
[COM_Port]
```

progfile is the file with the code to run on the target, in Intel HEX format, either with or without the .hex extension. Station_ID is the ID number of the target where the program is to run. COM_Port is an optional argument stating the serial com port that will be used to communicate with station 0. If omitted, the default port will be used.

While downloading, PCLoad displays the size of the program to download and the total number of bytes downloaded. PCLoad should terminate with the number of downloaded bytes equal to program size and a message stating the time in seconds the program ran. By this time, the target that received the code is prepared to change to RUN mode.

The boards software consists on two different programs:

- the gateway program;
- the remote boards program.

The gateway program runs on board no 0. This is the board connected to the host, providing the interface to the CANivete system.

Remote boards run a different program. This program can be generated by the `CANguru` utility (CANivete generator utility for remote units). CANguru automatically generates the code to program the remote board EPROM, given its ID number. The syntax is:

```
CANguru board_id
```

This will generate the code for the remote board and store in a file called CAN030*xx*.HEX, where *xx*=board_id. board_id must be an integer between 0 and 29.

5.5. Message protocol

Downloading the code to the target means sending through the CAN network a stream that can contain as much as 32k bytes. Except for very small programs (that will probably have no other use that performing simple tests), the size of most of the programs to download will be on the range of hundreds of bytes and over. The short size of a CAN message (8 bytes) and the associated overhead can reduce efficiency in the use of the existing bandwidth.

In this way, we decided to use as much as possible the useful data space in a CAN message during downloading phase. There are currently three kind of messages: CONTROL, DATA and ACKNOWLEDGEMENT messages. CONTROL messages are used to: signal start and stop of code download, start and stop code execution, enquiring about node status,... DATA messages are used to convey up to 8 bytes of code. ACKNOWLEDGEMENT messages are used to confirm the correct reception of a set of messages.

Code download starts with a CONTROL message signalling "start of transmission" and indicating the program size. The code is then sent in DATA messages, each message containing 8 bytes of program code. The last message may have less than 8 data bytes, when the code size is not a multiple of 8. Download ends with a CONTROL message signalling "end of transmission" and sending a checksum. The remote node should check both the code size and the checksum and respond with an ACKNOWLEDGEMENT message, in case of success. The identification of the remote node and of the message type is contained in the CAN message ID field.

6. Further work

The CANivete board is yet at an early stage of its development. The main track for evolution lies currently on the boards software. The next version of the board will include remote control of program execution, allowing , for instance, quasi-simultaneous starting of program execution in the different sites. This can be particularly useful in experiments carried on in distributed systems, such as the study of clock synchronization algorithms, which has been one of the reasons for the development of the system ([2,3]).

Developing a monitor will provide a higher degree of insight, which will be useful for teaching applications as well as for research and development.

Another track to follow is the integration of real-time kernels on the boards software, allowing the interaction with the remote boards to be done at the task level (with hard-real time constraints) and not at the program level. This real-time kernel would be in charge of managing the execution of tasks within the node and of interfacing with the user trough the network, receiving the tasks' code and starting and stopping its execution.

7. Conclusions

We have presented a working solution to assist the development of distributed embedded systems. Our solution allows circumventing the major problems faced in this task - namely the multiplicity of targets and its possible distance from the host - with no additional hardware required and using the existing communication infrastructure.

Our proposal comprises the targets hardware and software and the host interface. The CANivete board is the kernel of our system. Each board is a node on a CAN network, that is capable of receiving the code to execute from the CAN interface and start and stop its execution. The host interface is responsible for downloading the code and controlling its execution at the users request.

The code to be run at the targets can be generated by any compiler and the transition from the development system to a working prototype is straightforward.

The CANivete system provides the developer of embedded distributed systems with a testing platform at low cost and ease of use. The system is also adequate for teaching environments.

References

[1] Fonseca, J.A., Mota, A., Santos, F, Fonseca, P, Azevedo, J.L., Cura, J.L., "Affordable tools for teaching embedded systems"*, International Conference on Electronics, Circuits and Systems*, Lisbon, Sept. 1998.

[2] Fonseca, P., Mammeri Z., "A Framework for the Analysis of Non-Deterministic Clock Synchronization Algorithms", *WDAG'96 - 10th Intern. Workshop on Distributed Algorithms*, Lecture Notes in Computer Science 1151, Springer-Verlag, Oct. 1996

[3] Fonseca, P., Mammeri Z., Fonseca, J.A., "Can we trust Internet for distributed systems development? A case study on clock synchronization algorithms", *INDC'98 - 7th IFIP/ICCC Conference on Information Networks and Data Communications*, Aveiro, Portugal, June 1998.

[4] Fonseca, P., Santos, F., Mota, A., Fonseca, J.A., *User's Manual to the CANivete Board*, Universidade de Aveiro, Aveiro, Portugal, Febr. 1998

[5] Philips, *80C51 Based Microcontrollers*, IC-20, 1997.