An host-target environment for real time image processing

M. Pizzocaro
Electronic laboratory - Blaise Pascal University
U.R.A. 830 - C.N.R.S.

Electronic laboratory - Blaise Pascal University
63177 AUBIERE - FRANCE
Email: pizzocaro@lesmti.univ-bpclermont.fr

ABSTRACT

The development of a real time image processing on a specific architecture is always restricting for the user who must master all the elementary mechanisms of the machine. Such specialised machines are not really designed to be used by inexperienced users. In order to build an application, we need to have a good knowledge of specific processors, and to resolve inherent timing problems when using the real time. The gap between the transformation concept applied to the images and the very low programming access of the specific processors is total for the user. So, we have designed an open bus platform associated with development tools in order to resolve this problem which, on a conceptual level, separates the global functions applied to the images from the hardware system which processes them.

1. Introduction

The open bus platform combines two systems : a real time VME target linked to a Unix workstation by a high speed VMV bus. The target consists of a CPU and an adaptable set of video boards joined together by an auxiliary ECL bus according to a pipeline structure. This bus allows a data flow of 25 images 512x512 per second from a video camera or a video memory to be transfered. The Unix workstation holds development tools for the generation of the target application and complementary off-line processing in synchronization with it. To design the development tools, the smartest solution would be a definition of image processing orientated language. However, in an adaptable context this would not allow us to keep the versatility of the target, since possibility of adding or substracting video boards to process deal with specific applications would be subject to an updating of the language used. So, to keep this versatility through the interface, we have associated a modulable library according to the video boards with a sequencing language which has been defined using the tool YACC -Yet Another Compiler Compiler- generator of syntactical analysers. In this context, the user will define his application in terms of global functions, without working on a low programming level, which would be inefficient. The development tools allow him to adapt the target topology with respects to a given class of applications.

In this paper, we present, first the architecture of this open bus system, secondly the versatile environment based upon the definition of a sequencing language.

2. System Architecture

In order to achieve this platform, we have chosen a cross environment that links a Unix workstation containing the development tools to a real time target system fixed to a VME bus. This type of system is host/target orientated. There is a development environment and an execution environment -Fig. 1-. The two systems are coupled firmly by a common memory area. This link is carried out by the VMV bus as it is called throughout this document [1] which operates with the full 32 bit data and 32 bit address range.

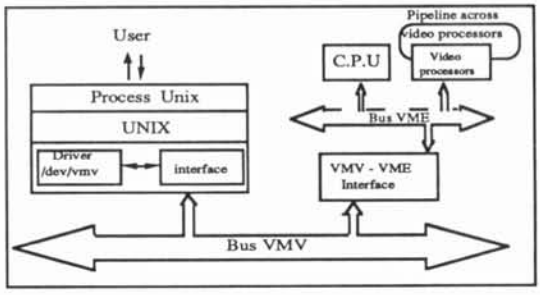

Fig.1 System overview

2.1 The target

The definition of Realtime systems is important. The IEEE Committee producing the POSIX 1003.4 standard propose the following definition :

"Realtime in operating systems : the ability of the operating system to provide a required level of service in a bounded reponse time".

To achieve this, the system must have low overhead. This leads to the requirement that the systems are memory resident with no paging or swapping to slow down the systen.

In our case, real time image processing means processing a data flow of 25 images 8x512x512 per second. In spite of the high bandwidth of the VME bus it is necessary in multiprocessor environments to process data flow on an auxiliary bus. So the target structure can be reduced to three sub-units :

- Specific video boards : a set of processors [2] connected to a synchronous ECL bus according to a pipeline

structure. The exact nature of the processors depends on the application to be carried out. A number of them are however essential, such as those used for the functions of digital coding, reconstruction from a digital form, or image memory. Other modules will be used for low level operations, typically convolution, filtering and middle level operations as characteristic extraction or edge chaining. This set of modules works according to recirculation techniques. The chain of cascaded modules recirculates its output back into its input.

- The management system : it is made up of a conventional central processing unit. It contains the real time operating system. It manages the application which loads the registers of the video boards after each frame passage in the pipeline. The later is specified by a VME interruption.

- The communications system : it includes a module which links the VME bus and the VMV bus [1]. This module is slave VMV and master VME. It shares the VME bus with the CPU and translates the VMV cycles into VME cycles in the target.

2.2 The host computer

The host computer is a Unix machine that combines a bus system with a compatible bus PC-AT. This open structure contains the interface [3] between bus PC-AT and bus VMV. This interface has a memory window of 64 Ko through which, after mapping, a Unix process can reach the VME address space of the target via memory reads and writes.

3. Development and execution environment

The software interface has been designed for building off-line, the real time application interactively. The user acts with a mouse to choose, among a database, processings and places associated data structures into a ressource file. This ressource file is eventually translated to the target where the real time kernel manages the application. Such interface means that a lot of its code must be devoted to handling interactions between the user and the core of the application.

The most important feature of the interface is its flexibility to a wide range of target configurations, i.e according to the active video boards. This interface is composed of two layers. The low level layer is part of operating systems to control peripheral devices such as the communication system and specific video boards. The high level layer handles interactions between the user and the core of the application. This layer, called the user interface, offers specific ressources with respect to the target configuration.

3.1 Device handling

The low level layer includes device drivers for the host and the target computers. It allows to read and write datas to and from the devices.

In the Unix workstation, the device driver allows the window interface to be mapped into the virtual address space of the application running under Unix -Fig. 2-. This device driver controls notably the I/O boundaries for the common memory area and avoids incorrect read-write cycles with devastating results on the target. Moreover, a fault handler is integrated to trap the bus timeouts which result from the passage of the asynchronous VME context to the synchronous host context. Thus the Unix system does not abort the tasks.

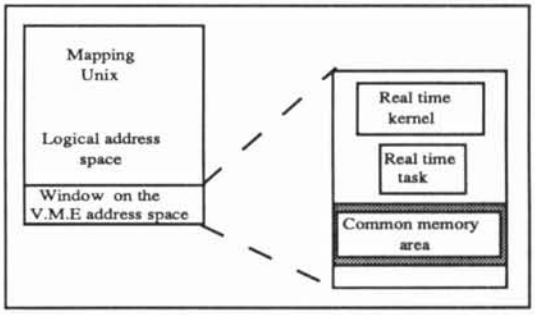

Fig.2 V.M.E acces throught the window interface

The target system works under a real time executive. After each pass of frame in the pipeline, a device driver traps the VME interruptions sent out by the video board which is the master of the auxiliary ECL bus. The processing time of the interruptions must not exceed 2.68 ms, the latent period between two frame passages in the pipeline -Fig. 3-. Only one I/O request will be made in order to load the video board registers and thus to avoid the reaction time of the core processing the request.

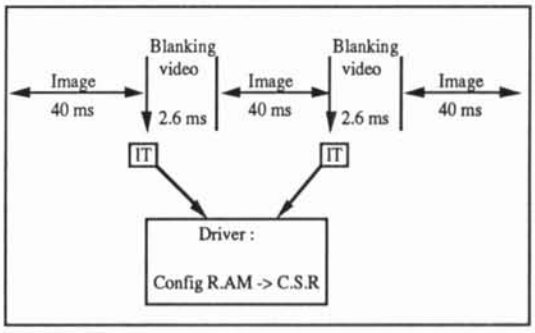

Fig.3 Timing of interruptions

3.2 Development toolkit

The user works only on the workstation side which runs the user interface. In order to design this interface, the smartest solution would be a definition of image processing orientated language. However, in an adaptable context this would not allow us to keep the versatility of the target, since possibility of adding or substracting video boards to process deal with specific applications would be subject to an updating of the language used. So, to keep this versatility through the interface, each video board structure is individually described by a database. A set of board configurations called functions, for standard processing,

are supplied. This environment is flexible. The user can modify it by discarding or by adding new functions for which he has to define their formal parameters and fix those which he hopes to hide.

To build an application, we use the basic concept according to which an application on pipeline architecture includes several elementary processings associated in iteration structures. A sequencing language allows us to specify their chain shape.

An elementary processing is the transformation applied to the image during its passage in the pipeline. Each processing draws together the functions executed by each active video board. The user chooses these functions by their validations in the database and places them into the ressource file. The general shape of the application will comprise a set of elementary processings.

We note "blk" -Fig. 4- the data structure associated to an elementary processing.

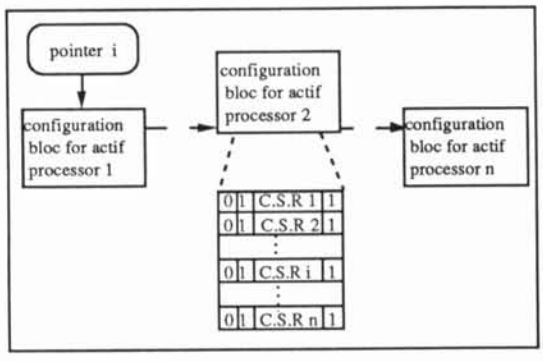

Fig.4 Data structure associated to an elementary processing

If the application requires n different processings, we will use an array of structures blk[n]. We obtain, according to a "c" declaration, for this array with a target including four video boards :

struct { csr1 board_1 ; csr2 board_2 ; csr3 board_3 ; csr4 board_4 } blk ;

array of struct : blk [n] ;

Each blk[i] structure contains the board C.S.R definitions and is defined according to a shape :

struct { char csr [8] } csr1 ;

Those structures, placed in the ressource file, are handled in background and hidden from the user. He works only with an object oriented interface which manages icons.

From this array of structures, it is easy, with a sequencing language to specify the processing chain. This language uses the index "i" of the element blk[i] joined to an iteration number.

This formulation is very concise to specify complex chains from a collection of basic configurations.

Two examples show this formulation

$$\left[[1]*2\,,\,[2]*4\,!\,,\,[3\text{ -> }5]\;*2\right]*1$$

$$\left[[1]*2\,,\,[2\text{->}5]\;*3!\,,\left[\,[6]*1\,,\,[7\text{->}9]\;*2!\right]\;*3\right]*2$$

We find the elementary processings with their iteration numbers : '*n' , a short notation to specify a continuaty of processings : ' ->', breakpoints for a synchronisation with an UNIX task : '!'. This synchronisation is carried out by using a mailbox situated in the VME address space.

The first expression developped gives us :

1,1,2,2,2,2!,3,4,5,3,4,5 ;

3.3 Language and grammar

The coherence of such specific expression given by the user, must be checked.

Phrase-Structure grammar, defined by a set of rewriting rules is used to check the coherence. A grammar is a set of rules which allow to build an infinity of language sentences ; in reverse, this set of grammatical rules allows us to check if a sentence belong to it.

We define an analyser to check the conformity of the expression to the grammar rules.

The grammatical concepts were formalized and developped by Chomsky [4]. He divided the Phrase-Structure grammars into four types according to the forms of the rewriting rules or productions. These types depend on restrictions of the sentence productions.

In this classification, we use type-2 grammars named context free grammars [5]. They allow to define recursive rules what is very useful in programming languages and are not context sensitive.

To define the rewriting rules, we use the "Backus et Naur Form" or B.N.F :

The chain expression shapes to the productions :

```
< expr > ::=   [ expr ] |
               < expr>,<expr> |
               < expr>*<number> |
               [<number arrow number>] |
               [ number]

< number >  ::=   <digit><number> | <digit>

< digit >   ::=   0|1|2|3|4|5|6|7|8|9

< arrow>    ::=   ->
```

The nonterminal vocabulary describes the language but does not appear in the sentence language. It is surrounded by symbols : "<" , ">".

The terminal vocabulary appears explicitly in the sentence language.

The symbol "::= " means "can be rewritten as".

The hierarchical structure of the chain expression

appears through the recursive rewritting rules.

To check the expression coherence, we decompose the analysis. We distinguish two phases which divide the chain expression in intermediate representation :

. The scanner or logical analyser. The input stream is read from left to right and partitioned into basic items called tokens.
. The parser or syntactical analyser. It assigns structure to the resulting pieces according to the rewritting rules.

The scanner allows the lexical units for the expression to be identified and to be moved towards the syntactical analyser. If The expression does not fit the grammatical rules of the language an error is detected.

The syntactical analyser was created by YACC [6] - Yet Another Compiler Compiler- from grammar production rules. A pushdown automation is defined as the recognizer for this type of context free language.

Additional disambiguating rules are necessary because input can be structured in two different ways like this one :

< expr>,<expr>,<expr>

The grammar rule :

< expr > ::= < expr>,<expr>

does not completely specify how complex inputs should be structured. YACC [6] allows to define disambiguating rules to resolve the parsing conflicts. In our case we specify symbol "," as a right associative operator.

3.4 The target load

If no error is detected, a configuration file is generated on the host computer and then loaded into the target. This file holds the validation mask of the active target boards, their configurations and the chain expression.

A loader running on the target, relocates the data structure in the V.M.E address space. This loader tranfers datas from the common memory area which contains the file configuration to the R.AM system .

For a real time task, it is necessary to convert those datas in a adaptable form. The validation mask of the boards is used to create the real time I/O buffer. The buffer shape is identical to the data chain structure of elementary processings. So at each interruption only one I/O request is needed and so avoiding latency of the real time kernel to serve the request.

Between the interruptions, this buffer will be initialized with a new configuration. This initialization must be done according to the expression of the processing chain. This expression cannot be processed in real time. It is necessary to convert it in a tree.

For example, the following expression becomes a binary tree structure as shown in Fig. 5.

[[1,2] * 2 , 3 , [4 ->6] * 4] * 5

At each terminal node of the tree a elementary processing is associated. Each node, root of a sub-tree, holds the iteration factor of the underlying structure.

This representation will give us a very condensed data structure, independently of the iteration number of elementary processings.

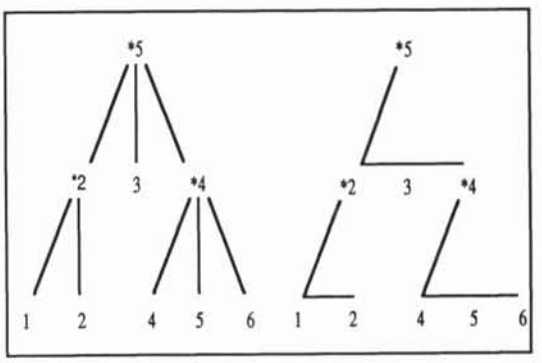

Fig.5 Tree and binary tree

The real time execution is now ready. The tree processing is carried out from terminal node to root node. The shifting is done from left to right and repeated according to the iteration factor contained in the sub-tree root.

4. Conclusion

The development of this system is carried out for five Datacube video boards [2]. The workstation is an HP 9000 - 433 machine. This development platform can be an efficient tool when defining real time image processing on complex systems. The user can rapidly understand how this interface works. One needs no special training in order to appreciate the hardware and software mechanisms of the system.

5. References

[1] VMVbus Vertical Bus System VBR 8212/VBE 8213. Technical Report - 1989 -
Creative Electronic System. Switzerland.

[2] Datacube, The Max Video Family Image Processors, Technical notes 1989.

[3] Vbat 8218 VMV Interface for IBM - AT
Technical Report - 1989 -
Creative Electronic System . Switzerland.

[4] N. Chomsky Three models for the description of language. IEEE Trans. Information Theory IT-2, 113-124 (1956).

[5] A. Aho, R. Sethi, J. Ullman : " compilers " Addison-Wesley Publishing Company. - 1989 -

[6] Stephen C. Johnson : " YACC - Yet Another Compiler Compiler " ; pp 1-29, 1984 ; Bell Laboratories, Murray Hill, New Jersey