An Architecture for High Performance Control Using Digital Signal Processor Chips

Stefano Battilotti and Giovanni Ulivi

Increasingly demanding industrial applications require fast and cheap computing tools for real-time control. For example, high performance industrial robotics would benefit from fast and cheap computing, but fast structures are generally expensive and dedicated to particular tasks [1,2,3]. Another example which requires fast computing is the control of electrical transients of ac motors, where complex numerical computations must be carried out within times like 1 ms [4]. A recent paper described implementation of a self-tuning controller which uses a digital signal processing chip for rapid calculations [5].

For many control applications there is a natural hierarchical structure so that algorithms devoted to simple tasks are placed at the lowest level (for example, controlling a robot axis or determining the switching time of a static converter), whereas complex tasks are at the high levels, forming a pyramidal control structure which reflects the multilevel structure described by Mesarovic [6]. In such a structure, tasks are usually repetitive and involve rapid manipulation of data, directly derived from the measurements of sensors, while high-level tasks are done over larger time intervals with more complex algorithms.

This paper describes a computing structure taking into account the hierarchical considerations above. The architecture consists of a high level general purpose computer (HOST) and up to eight digital signal processors (DSPs) which can be interfaced with the controlled plant(s).

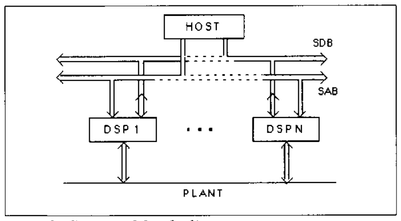

Fig. 1. System block diagram.

The authors are with Universita di Roma "La Sapienza," Dipartimento di Informatica e Sistemistica, v. udossiana 18, 00184 Rome, Italy. An earlier version of this paper was presented at the 1989 IEEE Industrial Electronics Conference, Philadelphia, PA, November 6-10, 1989.

The high-level computer is either a work station or an advanced personal computer with sufficient memory space (RAM and mass memory), equipped with peripherals for implementation of user-friendly interface and with the ability to communicate with other computers, perhaps in a local network. The synchronization and the real-time communications between the HOST and a DSP are implemented by the two memory bank alternatively switched between the HOST and the DSP. A complete transparency and a minimum overhead result for the tasks running on the DSP.

Design Outline

The DSP chip can rapidly and efficiently manipulate numerical data. For this application the Texas Instrument 320XX DSPs chips have been used, since they show a high compatibility among the components of the family. Moreover, derived processors are available (the so-called Digital Signal Controllers) which can ease in some cases the interface toward specific industrial processes [4]. However, the proposed architecture could be implemented also by means of other kind of DSPs, with the proper circuital modifications.

Two data paths collect most of the information. The first is between the low level processor and the plant. This path is involved with frequent transactions and computations, concerning small amounts of data, and determines the useful bandwidth of the control system at the actuator level, so its speed is of paramount importance.

The second data path is between the low-level processor and the HOST computer. It collects other types of data such as reference values, controller parameters and quantities derived from manipulating the measures coming from the plant. These data can be transferred in bursts with a time interval between transactions much longer than those at the first level (ten times is a typical value). The transfer speed during each burst is that allowed by the HOST and is much lower than that the frequency the DSP can achieve.

Communications are based on two random access (RAM) memory banks for each DSP card alternatively switched between the DSP and the HOST so as to get complete transparency. This technique is conceptually very clear, its implementation is simple and allows the maximum decoupling between the two computers.

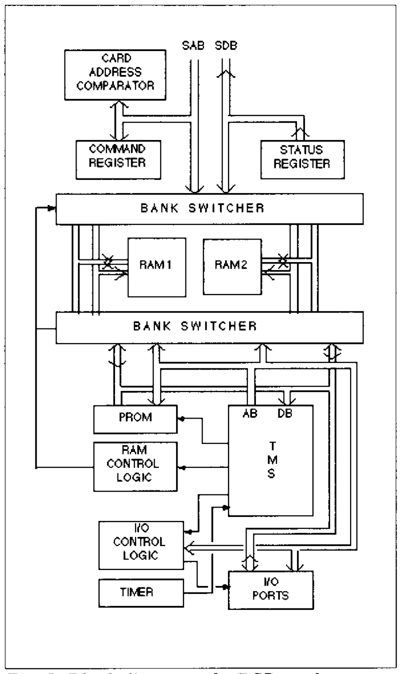

Fig. 2. Block diagram of a DSP card.

The DSP cards master the bank switching but do not participate to the data transfer, so their operating speed is practically unaffected by the communications.

The synchronization between the two levels results for each DSP card from the handshake relative to the memory bank switching. Such mechanism is started by the HOST, which sends a suitable message to the DSP as soon as new data have been computed and then waits for the answer. This request is detected by the DSP, which, at a suitable instant, switches the two banks and sends an acknowledge to the HOST. In this way the data are exchanged between the processes and the HOST can synchronize its activities.

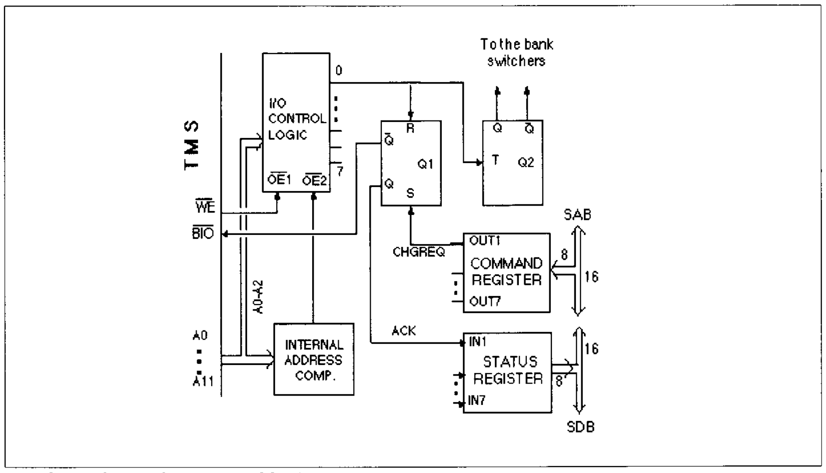

Fig. 3. Bank switching control logic.

System Block Diagram

The overall block diagram of the system is given in Fig. 1, where the two system buses, connecting the HOST and the DSP cards, are shown. The first one is used for commands and addresses (SAB), the other one for data (SDB).

Figure 2 shows the internal structure of a DSP card. It is worth noting that, although not explicitly pointed out in the figure, the DSP has two separate memory spaces, one for the program code and the other for the data. The data transfer between the spaces is accomplished by the highly efficient instructions TBLR and TBLW. In the figure the two memory banks and a PROM, accessible only from the DSP, are shown. They are connected as part of the program memory of the DSP.

This choice allows to easily download the program code into the DSP. Indeed, beside the data exchanged with the HOST-P, the banks hold the TMS program sections most subject to be changed plus a reserved parameter area. This allows some parametric program sections to be changed "dynamically" (e.g., part of the control algorithm) by rewriting them starting from a suitable address.

On the other hand, the PROM holds the permanent program sections, as for example an initialization program and the synchronization and handling routines.

The two memory banks (2 kwords which can be extended up to 4 kwords) are alternatively connected to the DSP or the HOST. The commutations are implemented by two groups of 3-state buffers (Bank Switchers) driven by the RAM Control Logic.

The PROM and the two banks give a Y-shape to the program memory space. The bifurcation address can be selected by jumpers.

Strictly connected to the memory is the RAM Control Logic, which implement also the synchronization mechanism.

Flip-flops necessary to card operations are driven by the DSP by addressing them by means of I/O instructions. The I/O Control Logic provides for that and moreover for enabling the ports which allow the system to be connected to the controlled plant.

A suitable register allows the HOST to carry out a set of operations in the DSP cards (e.g., bank read/write operations) while a status register supplies the relevant information about its operations. They are connected to most of the other blocks. Moreover, a Card Address Comparator allows the HOST to address a specific DSP card. The comparator and the two registers are directly connected to the system buses.

All the system timings derive from an onboard timer, which provides the interrupt requests at each sampling instants to synchronize the DSP. It is presettable over an interval ranging from 100 μs to nearly 1 s.

Data Communication and Interprocessor Synchronization

As already mentioned, data communication and synchronization are both implemented by the bank switching mechanism. The data are contemporarily exchanged between the two computers simply by switching the RAM banks containing them. In this way the low-level processor has a minimum overhead and also the synchronization proceeds from the low to the high levels, as requested. Figure 3 shows the logic which implements the switching of the memory banks and supplies the relative acknowledge when the request is honored. It contains the relevant part of the RAM Control Logic. The switching mechanism is ruled by a semaphore which represent the number of sampling instants left to the next bank switching and is decremented at every sampling instant by the DSP. At each bank switching, the value of the semaphore is initialized by the HOST. Therefore the switching instant is determined by the low level processor under the HOST control.

The toggle flip-flop $Q2$ stores the status of the connections of the RAM banks.

The switching procedure is started by the HOST-P, which sends a request along the SAB. This is stored into the set-reset flip-flop $Q1$, whose output is connected to the polling line BIO of the DSP. The DSP-P, when enabled by the semaphore, honors the request by switching the banks and signaling the operation by the ACK line.

The lack of a wait line in the simplest members of the TMS family (TMS32010) has determined also the I/O design. In fact, it was not possible to use sophisticated programmable ports but it was necessary to resort to standard TTL logic to build two external I/O buses (address and data) to communicate with the transducers installed on the plant.

Card Control Operations

Card control operations are carried out by means of commands which the HOST-P can address to each card. The most important commands are listed in Table I. Some of them (e.g., memory read) naturally refer to single cards while others can be required to be contemporarily executed by a group of cards (e.g., run). Therefore a mechanism has been implemented based on two phases: during the first, the command is stored in a dedicated register (Command Register, see Fig. 4), during the second all the stored commands are executed. Obviously this mechanism requires to store a do-nothing (idle) command in the cards which should remain unaffected.

Table I
List of Main Commands

Memory Bank write
Memory bank read
Latch card status
Card status read
Bank switch request
Idle
Run*
Reset*
Forced bank switching*

*Commands used only during code download

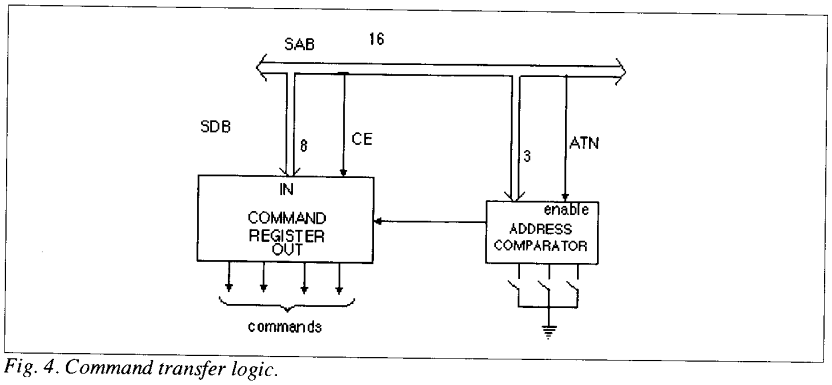

Fig. 4. Command transfer logic.

All the commands are sent by the HOST along the SAB (normally used for memory addresses) together with the card address.

It is to be noted that, with the used memory bank sizes, 12 bit long addresses are required. This value has been increased to 14 to take into account future expansions. Therefore, if a standard 16-bit port is used in the HOST, two bits remain to implement the command mechanism. One of them (ATN) is used to distinguish between commands and memory addresses. When it is active, each card compares its address with that present on the SAB; if they are equal, the Command Register is enabled to latch the command. In this way the proper commands are stored in the required cards. Afterwards, the HOST-P can activate the commands on all the card of the system using the other bit (CE).

It is worth noting that highly repetitive commands such as memory read and memory write require only one command transfer: once the command is latched, it can be activated each time by asserting the CE line.

```
{DSP loop}
Repeat
 If semaphore = 0 then
  If switch_request Then switch_banks;
 semaphore = semaphore - 1;
 If semaphore < 0 Then error = true;
 read_data_from_bank;
 low_level_conrol;
 write_data_to_bank;
Forever

{HOST loop}
Repeat
 read_data_from_bank;
 high_level_control;
 write_data_to_bank;
 send_switch_request;
 Repeat
  wait
 Until switch_acknowledge
Forever
```

Fig. 5. Iterative parts of the software.

The drawbacks of this approach are the slowing of the information flow along the bus and the more complicated communication software in the HOST; anyway such drawbacks are completely compatible with the purpose of the design, taking into account the low-rate communication of the HOST along the bus with respect to the very high operative speed of the DSPs which control a real-time process.

Moreover, this technique allows to get rid of the problem coming from the cable length and to adopt every needed communication rate without slowing down the DSP operations.

The Management Software

As most of the communication mechanisms heavily rely upon hardware resources, the system software turns out to be very simple. It mainly consists of the permanent DSP code and the procedures implementing the various operations which the HOST can perform on the DSP card. The procedures are written mostly in a high-level language (Pascal was used) with some small parts in assembler to improve the overall efficiency. The assembler parts are transparent to the user as they are embedded in Pascal procedures so to make the programming as clear and easy as possible.

The HOST is a general purpose work-station which can run cross-compilers or cross-assemblers. Thus there is the possibility of writing (maybe compiling) and assembling the TMS programs on the HOST computer itself. The obtained machine code is afterwards down-loaded into the DSP cards. Moreover the general purpose HOST can be used to perform off-line computations. For example, in a robotics application it can compute the trajectory values and moreover it can perform a performance evaluation.

The DSP synchronization software consists in a few tens of assembler lines which provides for decrementing and checking the semaphore and switching the memory banks. Other utility functions are provided as for example standard I/O routines and data transferring between the DSP program memory and data memory.

A typical way the system works is the following. At the beginning the TMS code is down-loaded from the memory-mass of the HOST into the two RAM banks while the DSP is kept reset; afterwards the system is started. The HOST-P works under a mixed interrupt/polling environment. First an interrupt wakes up the HOST-P; immediately it asserts a switching request and then it waits for the acknowledge from the DSP-P.

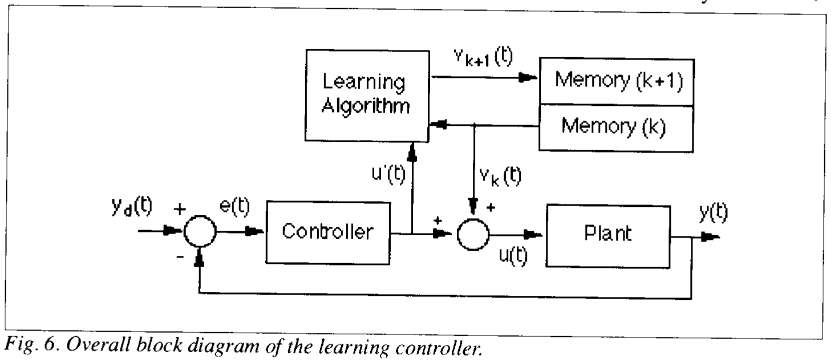

Fig. 6. Overall block diagram of the learning controller.

On the other side the DSP-P tests the semaphore at each sampling time (given by the on-board timer); if not enabled then it skips the switching part of the code and starts another control loop. Otherwise, it switches the banks, signaling the acknowledge to the HOST-P which thus can go on reading the data stored by the DSP-P from the new available bank.

The HOST-P, after computing and storing the data to be used by the DSP-P in the next cycle, can go in an idle state, thus making the HOST CPU available for lower-priority tasks, such as giving information about the plant status to the operator, if requested, until another interrupt is generated.

The repetitive part of the processes above described is shown in Fig. 5, using pseudo Pascal-code.

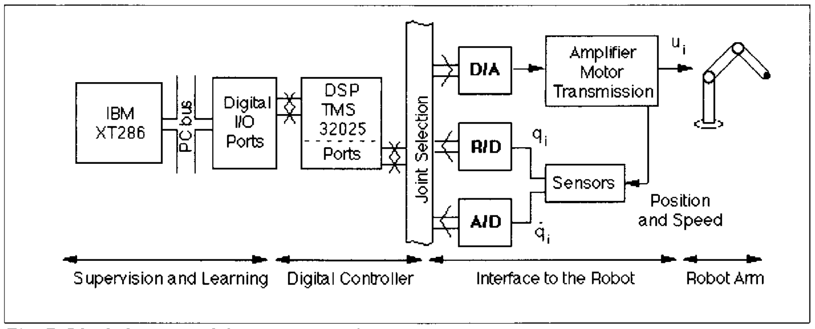

Fig. 7. Block diagram of the experimental environment.

Further Developments

The described system has been tested successfully and it is currently used in our robotics laboratory. The first version has been implemented using a TMS32010 which allows only a very small memory to be used. Later on the same card a TMS32025 has been installed by a piggy-back into the socket of the first processor. In this way the C-compiler available for this CPU can be used.

Currently a new card based on this processor is being designed, allowing room for a larger memory (up to 32 kwords).

To reduce the component number, fast PLA will replace several standard TTL chips. Moreover, since during the tests it was checked that the processor timings allow the bank switching during consecutive instruction fetches, the PROM will not be used so as to make the circuitry simpler.

Experimental Results

An interesting application of the proposed computer architecture has been the implementation of a learning controller for a three-degree-of-freedom robot arm [7]. Iterative learning is a control technique where the required input to a given system is built iteratively from successive experiments and the performances on the repetitive tasks are asymptotically improved from one iteration to the other. This technique works even with limited *a priori* knowledge of the system.

Under suitable assumptions, the learning schemes will converge and acquire from the trials the information needed for the successful completion of the task.

The control scheme (drawn in Fig. 6 for one joint) is constituted by a standard feedback loop in which a feedforward signal ($v_k(t)$), stored a in memory (Memory (k)), is injected. The next feedforward signal ($v_{k+1}(t)$) is derived from the current one and the output of the controller $u'(t)$. The filters contained in the learning algorithm block ensure the convergence of the iterative procedure. The asymptotic value of the memory is such to zero the error $e(t)$ in a prescribed band of frequency. Clearly, the involved signal processing needs not to be carried out in real time and, on the other hand, the two vectors containing the memory data are rather long, depending on the duration of the task.

Therefore, these processings have been devoted to the HOST, while the standard feedback loops have been implemented on the DSP.

The experimental system is constituted by a prototype manipulator and the multimicro structure. Its block diagram is given in Fig. 7. As the HOST computer, an XT286 has been used, equipped with a Burr Brown Carrier which provides the lines to SAB and SDB buses. A single DSP card has been used, carrying a TMS 32025.

All the robot joints are revolute and are actuated by dc motors through harmonic drives with transmission ratios equal to 160. Motors are powered by current amplifiers, whose reference values are provided by 12-bit D/A converters.

Each joint is equipped with a resolver and a dc tachometer for velocity feedback. The analog outputs of both sensors are converted into digital values with a resolution of 16-bit/2π and, respectively, 11-bit/(rad/s).

Besides computing the new feedforward signal, the XT286 provides the system mass memory and the graphic user interfaces. All the functions are programmed in Pascal; in particular, the trajectory generation is done at this level. It also provides a programming environment for the TMS 32025, including an editor and the *C* cross-compiler to generate the executable codes.

The closed loop linear control (proportional plus derivative) is performed by the DSP with a sampling time of 400 μs. The most relevant system variables (among the others, position and velocity errors and the output of the linear controller) are stored every other sampling instant in a local buffer and transferred to the XT286 every 20 ms, the sampling time of this computer.

These data are progressively saved in a large buffer in the PC RAM. At the same rate, the proper segment of the time-varying reference signals $y_d(t)$ and $v_k(t)$ is downloaded. At the end of each trial, the XT286 buffer is processed off-line and the new learning output $v_{k+1}(t)$ is computed.

Conclusions

A multimicro DSP-based architecture for control applications has been designed and implemented showing high performances and modularity. To obtain these results, a hierarchical structure has been preferred and the use of real-time operating system was avoided. Instead, the system relies upon simple, still efficient, hardware resources. Two levels can be seen, the higher one consists of a high performance work-station, the lower one of up to eight DSP. Each DSP card contains two RAM banks alternatively switched between the DSP and the work-station which controls the whole system. The synchronization between the two levels is obtained by exploiting a RAM bank switching mechanism requested by the HOST and honored by the DSP. The system can be easily programmed and it is currently used in our robotics laboratory.

References

[1] J. Ish-Shalom and P. Kazanzides, "SPARTA: Multiple signal processors for high-performance robotic control," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 1988.

[2] Y. Wang and S. E. Butner, "A new architecture for robot control," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 1987.

[3] S. S. Leung and M. A. Shanblatt, "Computer architecture design for robotics," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 1988.

[4] A. Bellini, G. Figalli, and G. Ulivi, "A microcomputer-based optimal control system to reduce the effects of parametric variations and speed measurement errors in induction motor drives," *IEEE Trans. Ind. App.*, Vol. IA-22, no.1, 1986.

[5] K. H. Gurubasavaraj, "Implementation of a self-tuning controller using digital signal processor chips," *IEEE Control Syst. Mag.*, vol. 9, June 1989.

[6] W. Findeisen *et al.*, "Control and coordination in hierarchical systems," IIASA Wiley and Sons, 1980.

[7] A. De Luca, G. Paesano, and G. Ulivi, "A frequency-domain approach to learning control: Implementation for a robot manipulator," presented at 4th IEEE Int. Symp. Intelligent Control, Sept. 1989.