Implementational issues in CACSD

Torp, Steffen; Nørgård, Peter Magnus; Christensen, Anders; Ravn, Ole

Published in:
Proceedings of the IEEE/IFAC Joint Symposium on Computer-Aided Control System Design

Link to article, DOI:
10.1109/CACSD.1994.288882

Publication date:
1994

Document Version
Publisher's PDF, also known as Version of record

Implementational Issues in CACSD

Steffen Torp, Peter Magnus Nørgaard, Anders Christensen and Ole Ravn
Institute of Automatic Control Systems, Technical University of Denmark,
Building 326, DK-2800 Lyngby, Denmark, E-Mail: or@sl.dth.dk

Abstract

The paper describes design considerations for a program for real-time testing of control algorithms in a laboratory environment. The algorithms are developed and tested using simulation in the MATLAB environment. The real-time code is built from the structure of the MATLAB script file using a matrix library with interface functions to MATLAB data files. Three real-time hardware platforms are analysed with respect to deriving a device independent program structure, facilitating portability among the three platforms and supporting portability to new platforms. The three platforms are a Transputer based system, an ADSP21020 based DSP system and a MC 68030 based VME-bus system. The programming language is ANSI C.

Keywords: Real-time systems, implementation, CACSD algorithms, algorithm portability.

1 Introduction

During the last few years, there has been an increased interest in automating the process of implementing digital controllers. The driving forces behind this development has been the wish to have a better consistency between the designed and implemented controllers also this part of the design process has traditionally been rather work intensive. This has led to a number of modules for design software packages. AutoCode for Matrix$_X$ and the C-code generation toolbox for SIMULINK are examples of such products. Other attempts has been made to make a MATLAB to C compiler (Tang et al., 1992) and a SIMNON to MODULA 2 compiler (Dahl, 1991). Such products face problems such as code efficiency, problems with debugging and data logging and a limited number of hardware platforms.

One of the problems of testing complex controllers in a laboratory environment is that the iterative nature of the design process makes it necessary to perform ad hoc modifications and experiments in order to arrive at an optimal control performance. These experiments and changes should be possible to make in a structured way in the real-time code without having to recompile. Another problem is that the real-time version of a specific algorithm is hard to port to another platform, leading to the situation where real-time versions of algorithms are not reused but rewritten, thus increasing the possibility of errors.

Much time could be saved by using a structured methodl which enables porting a controller between a simulation environment and a real-time program. It is our aim to derive such a method and describe the tools needed, which provided with a MATLAB or SIMULINK (Matlab, 1992) description of a plant, enables the user easily to set up the control structure needed and test it by simulations. A real-time program is written, realising any control structure derived in the simulations and saved to a file.

An analysis of three different real-time platforms is the basis of the design of the portable overall structure of the real-time program. The developed programs, consists of a MATLAB script, some additional functions for simulation written in MATLAB, and an ANSI C program for real-time control including a library for matrix calculations, and device support libraries.

Any family of control algorithms could be implemented using the approach presented here. The method, e.g. the MATLAB script and real-time program has been tested using a family of implicit parameter adaptive controllers as an example. However, other controllers with variable or selectable structure could be implemented using the same approach.

The benefits of such an analysis, design and testing procedure eases the design of such controllers. The adaptive controller is a good example of the fact that some controller algorithms are better described my a sequence of commands than by a drawing them in a graphical editor.

2 Analysis of Real-time Platforms

Practical implementation of adaptive control algorithms often dictates the use of existing process control equipment that is not at all compatible with the system on which the software was initially developed and tested. Hence, it is an advantage for the engineer implementing the software, if the device dependent parts of the code, such as analog I/O, inter-task communication and real-

0-7803-1800-5/94/$3.00

time scheduling, is encapsulated in a single software library. To do so, we need to identify all device dependent code needed in the program and specify a common interface to the library, which can be used on all relevant platforms. In this paper, three platforms have been investigated and their advantages and disadvantages, in terms of real-time control, have been analysed. The three platforms are an OS-9 system, a Transputer system and an ADSP21020 based digital signal processor system. None of these platforms comply with the POSIX standard, which is a good attempt to deal with portability in a very general way. Due to the lack of compliance with such standards, we will deal with portability on a lower level, analysing the three platforms and suggesting a common software structure. Further, we will discuss how to design a program that is capable of running on all of the three platforms, only requiring recompilation of the source code, when changing to another platform.

2.1 OS-9 system

OS-9 is a real-time, multitasking operating system from Microware Inc. (OS-9, 1987). It is available for a range of different platforms such as Intel 386/486, Motorola 680X0 and SPARC. In our laboratory OS-9 is running on a MVME147 single-board computer from Motorola hosted in a VME bus system. The computer is based on a MC68030 integer CPU with a MC68882 floating-point co-processor and 4 Mb of dynamic RAM. In addition the board has four RS232 channels for serial communication with terminals etc., a SCSI interface for floppy and hardisk and an Ethernet interface. Mounted on the VME bus is an A/D and D/A converter board from Modular computers Inc. for direct connection to physical processes, figure 1.

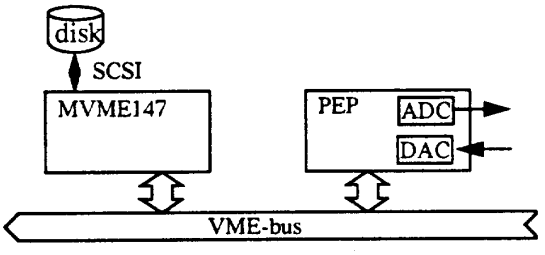

Figure 1. Structure of the OS-9 system.

The OS-9 system itself is an extensive real-time operating system with a pre-emptive scheduling multitasking kernel providing facilities such as semaphores and pipes. Pipes are a sort of RAM files, providing tasks with extensive but easy to use means of inter task communication. OS-9 pipes are quite similar to UNIX pipes, and resembles the so-called channels, which are used in the Transputer environment.

2.2 Transputer system

A network of Transputers is programmed according to the CSP (Communicating Sequential Processes) model (Inmos, 1992). A model, which widely resembles the way OS-9 systems are programmed.

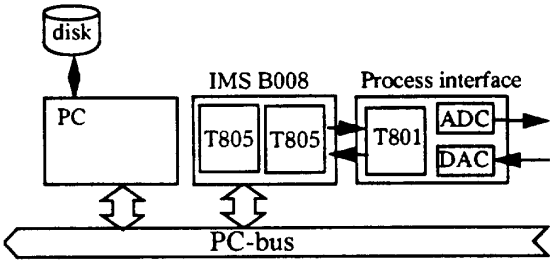

Figure 2. Structure of the Transputer system.

The actual Transputer system, figure 2, is a IMSB008 board from Inmos Inc. It is a plug-in board for a conventional IBM compatible PC. Via links on the board, it is connected to an external transputer box with A/D and D/A converters. The external Transputer box is designed in our laboratory and serves as interface to physical processes. The Transputer in the process interface is equipped with 32 kb RAM apart from the 4 kb on-chip memory. The IMSB008 board contains a DMA based interface between the host PC and the so-called root Transputer on the IMSB008 board. It serves as a motherboard for up to 10 TRAM's (TRAnsputer Module). The Transputer in the first slot (slot 0) of the motherboard will be the root transputer. A TRAM is a small board equipped with a Transputer, some memory and maybe some additional interfaces etc. We use two TRAM's, each having a T805 Transputer and 2 Mb of dynamic RAM. A programmable switch on the board enables the user to setup the network topology. We use this switch to connect the second Transputer with the external process interface.

2.3 DSP system

The DSP system, figure 3, is based on an ADSP21020 chip from Analog Devices Inc. It is designed by Loughborough Sound Images Ltd. as an expansion board for an IBM compatible PC's. The interface to the PC is realised by a dual-port RAM block, which is divided into two blocks of each 160 k words. The two blocks are data and code memory and the wordlengths are 40 and 48

bits. The host PC serves both as a development platform and as a front-end processor handling disk I/O etc. Those parts of a program needing access to disk, keyboard and screen must run on the host PC. The board is supplied with routines in Borland C, which allows access to the dual-port RAM while the DSP is running.

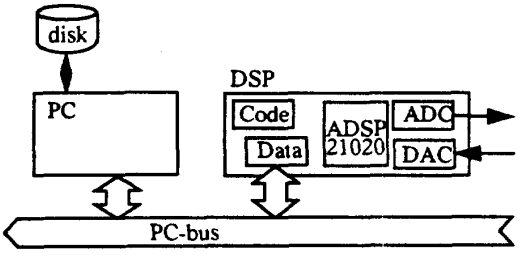

Figure 3. Structure of the DSP system.

The ADSP21020 is an extremely fast floating point processor designed especially for repetitive calculations, f.ex. matrix operations. Instructions in its instruction set, including the floating point operations, are executed within a single clock cycle, in this case 50 ns.

2.4 Comparison

We will now try to compare the three platforms and their applicability to advanced control. Such a comparison can not possibly pose full justice to each individual system, since they all have features that are hardly comparable to the others. Below (Table 1) we give a schematic overview of some of the most comparable features and then we will include the individual facilities in a discussion of each system and its applicability to adaptive control.

TABLE 1. Features of the real time platforms.

	MVME 147 and OS-9	Transputer and PC	ADSP21020 and PC
Operating system	OS-9	Host-PC: DOS	Host-PC: DOS
Real-time kernel	OS-9	In Hardware	None
Multi-tasking	OS-9	In Microcode	None
Software environment	ANSI C, C++	ANSI C, C++ and Occam	Host-PC: Borland C DSP: DSP C (ANSI)
Processor(s)	MC68030 MC68882	2 T805 1 T801	80486 ADSP21020
Memory	4 Mb * 8 bit	2 * 2 Mb * 8 bit	160 k * 40 bit 160 k * 48 bit
A/D channels	16	8	4
D/A channels	4	8	2
Interrupts	Yes	Yes	No
Disk and Terminals	Yes	From root CPU	No
Performance	Medium	Good	Very good

Notice in particular that the DSP system suffers from a lack of system software to support real-time programs and multi-tasking. Timers must be programmed in assembler using interrupts.

Analysing the platforms the minimum requirements that must be satisfied was identified. As it was decided to use C, an ANSI compliant C compiler must be available for the systems. The platform must of course have access to analog input and output devices and a timer for synchronized execution of real-time programs. Furthermore, an operating system supporting multitasking and some basic form of inter task communication is necessary. The DSP system does not support any multitasking facilities, but its PC front-end enables us to run two tasks on the system, one on the PC and one on the DSP. The task running on the PC has of course full access to disk and other I/O. This task must deliver all data from disk to the DSP task in the proper format through a sort of pipe or channel emulated in software via the dual-port RAM. The situation is quite similar to the Transputer system, where the tasks needing access to disk and other I/O must be running on the root Transputer. Apart from this, the Transputer systems has practically all the facilities you will find in an advanced real-time operating system and it is all in hardware or microcode. The limited amount of memory in the external Transputer process interface suggests an additional task running in the interface box, handling sampling of the analog signals only. On the OS-9 system there is no limitations of any such kind. Due to the simple single processor structure and the extensive facilities, any task may gain access to peripheral devices.

3 Structure of the software system

The software, which we have developed is mainly intended as a research tool for investigating adaptive control systems in practice. The purpose was to ease the process of porting a control algorithm to a real-time platform and test it on a physical object. As we mentioned earlier, the real-time software works in close connection with a simulation environment for MATLAB. SIMULINK is used for simulation of continuous-time plant dynamics.

To implement this in a software package, we chose the over-all structure of the simulation and the real-time program to be identical. This has several additional advantages, for instance that all data through-out the program are comparable, which can be very useful in a debugging phase. Furthermore, the two programs share the same user interface, which makes it easier to switch between them.

Software for real-time control must take into account the computational performance of the code. An issue, that is not equally important in software, which is only intended for simulations. In order to ease the process of moving e.g. a parameter estimation algorithm from MATLAB to C, a library of matrix functions performing the simpler MATLAB operations was made. In MATLAB, the programmer has little or no control of memory allocation. Variables are allocated and deallocated in each assignment. Performing a simple operation such as A = A + B; implies allocation of memory for a new copy of A and deallocation of the old one. In algorithms for real-time execution, the programmer would want to control exactly when memory is allocated and deallocated, since memory allocation is often a quite time-consuming task involving calls to the operating system. Furthermore the allocation and deallocation at each sampling interval can segment memory, making it impossible to get a contingent block. By providing two simple functions for allocation and deallocation in the matrix library, we have enabled a programmer to extract all allocation of memory to those parts of the program, which are not sensitive to computational delays. Even in the case of local variables in functions this can be done by allocating the memory during the first run of the function and then storing the pointer to the memory location in a static variable. This just requires, that the function only can be used once every sample as e.g. in a parameter estimation algorithm, because local variables are now fixed to a certain size.

As a bridge between the real-time matrix library and MATLAB, two functions are included for loading and saving matrices in files with MATLAB format. These functions are used intensively to construct the interface between the simulation and the real-time environment.

In a laboratory environment, just as in industrial ones, process control hardware from many different manufacturers are often used. Hence the need to bring a MATLAB adaptive controller to several different platforms is obvious. We have designed our software to match the three different platforms previously described in this paper.

The obvious modularity of the adaptive controller suggests a task structure dividing the code into the tasks 'Identification', 'Control', 'Design', as proposed in (Henningsen et al., 1991). This issue becomes even more interesting in the case where more Transputers are available, which allows the programmer to distribute the different modules of the controller on to their own Transputer such as proposed by (Fortuna et al., 1989). Due to the missing multitasking facilities of the DSP system and a wish to concentrate on the problems of multi platform software development, we chose a simple two task structure. (Three in the case of the Transputer system).

The two tasks are what we call a regulator task and an administrator task. Table 2 shows what they do.

TABLE 2. Task division

	Administrator task	**Regulator task**
Characteristics	Low priority Foreground task User interface	High priority Background task System interface
Tasks	Allocating resources Get data from disk Download data and maybe code to regulator User communication Activate & start regulator Save data on disk	Receive data Read from ADC Compute control action Write to DAC Parameter Estimation Controller design Data-acquisition

This structure has the advantage of being directly implementable on all of the three platforms. Although the Transputer system needs an extra task to handle the sampling of process data, this is not considered a part of the main program.The main program of each task depends on two important library files, figure 4. That is the matrix library, which we have mentioned earlier, and a library called `sysdep.c`, which encapsulates all of the system dependent code. Both libraries are needed in both tasks of the program to enable them to handle matrix data and provide them with a common interface to system specific functions. The `sysdep` library contains the only parts of the source code, that need to be modified, when porting the program to another platform. We have chosen to have the MATLAB file interface functions in the matrix library, even though they are not completely portable. We find that they are more related to the functions in this library and that the portability problems in the MATLAB functions deals with data portability more than code portability. When using the matrix library in the `regulator` task the MATLAB interface functions can be excluded from compilation by setting a switch in the makefile. Similar switches selects what part of the `sysdep` is compiled.

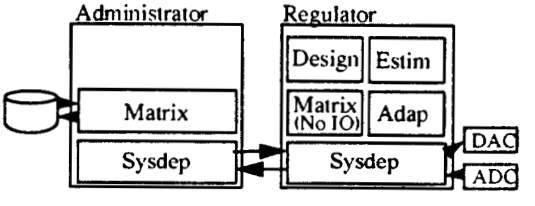

Figure 4. Tasks and libraries.

In the regulator task, three other function libraries are used. They all contain functions used to form the adaptive controller. The `estim` library contains different parameter estimation algorithms, the `design` library contains controller design algorithms and the `adap` library contains miscellaneous functions mainly related to the implementation of continuous time adaptive algorithms.

4 Extending to new algorithms

Using the matrix library, we have made it a far simpler task to port an algorithm from MATLAB to C, but we have lost some of the simplicity. In a C program, variables still have to be declared, allocated and initialized before use and deallocated again after use. This requires four new statements, which was not in the MATLAB code. Since the structure of our two programs are identical, the coding of these statements is trivial, and can be automated. By stating all MATLAB variables used in the real-time program in a special file called `matlab.var`, the statements needed can be generated by the C preprocessor using file inclusion and macro expansion.

The described procedure reduces the amount of coding to one line in one file and to the best of our knowledge reduces the probability of errors equally. All variables stated in the file will now automatically be initialized with the data from a file called `datain.mat`, that the simulation program generates on request.

A quite similar method is used to include new estimator and controller design algorithms. When a new algorithm has been programmed and tested in MATLAB and should be implemented on the real-time system, the algorithm is first coded in ANSI C using the matrix library, then its name is added to a list in the file `estim.lst`. This automatically makes the algorithm available in both environments and when it is selected for simulation, it will also be used for real-time control. As an example of how to add an estimation algorithm to the program:

```
void esti( matrix *phi, matrix *theta, double
ey ); {
   static matrix *k;
   /* If first call, allocate memory    */
   if (first_last == FIRST )
         k = mmake(unknowns,1);
   /* Here is the estimation algorithm */
   /* if last call, deallocate          */
   if (first_last == LAST ) mfree(k);
}
```

The new estimation algorithm must be included in the `estim.c` file and the function header must comply with the format shown in the example. Of course other function identifiers may be chosen as long as they do not conflict with other identifiers in the program. To limit the number of memory allocations for local variables, these are declared as static. During the first run of the functions the necessary memory is allocated using the `mmake` function from the matrix library. Usually the size of the local variables will depend on the number of parameters to be estimated. This number is specified in the global variable `unknowns`, which is transferred form the MATLAB file. The actual algorithm for updating the parameter vector is placed in between memory allocation and deallocation. When the program is closing down and the function is called for the last time, all of the memory must be deallocated. If the algorithm needs variables from the MATLAB file, other than those already provided, these must be added at the end of the `matlab.var` file:

```
VAR( New_var, matrix *, Matrix, global )
```

At last the new function is made available to the program by adding its name at the end of the `estim.lst` file:

```
EstimatorFcn(esti)
```

Now, the function can be selected in the simulation environment and used both for simulation and real-time control.

5 Extending to new platforms.

When porting the software to a new platform, the functions in the `sysdep` file must be rewritten. The file contains the definition of a general communication channel and the functions needed to use this channel. The functions highly resemble those supported by the Transputer C compiler, but they can furthermore be used directly for sending and receiving matrices. The function call is the same on all the platforms, and the communication channel is quite general and could be used in other programs as well as ours. In addition to the communication functions, the `sysdep` library contains some simple functions for setting up sampling time and for analog I/O, which are specific to our program. The following example shows a function for receiving a matrix over a channel and how the preprocessor can be used to select what part of the source code is compiled:

```
matrix *ChanInMatrix( Chan_type in_chan )
{
   int rows, cols, mn;
   matrix *ptm;
   /* Read number of rows and columns */
#ifdef OS9   /* OS-9 code */
   read( in_chan, (char *)&rows, sizeof(int) );
   read( in_chan, (char *)&cols, sizeof(int) );
#endif

#ifdef IMS   /* Transputer code */
   rows = ChanInInt( in_chan );
   cols = ChanInInt( in_chan );
#endif
```

```
   mn  = rows*cols; /* Number of matrix ele-
ments */
   ptm = mmake( rows, cols );  /* Memory allo-
cation */
   /* Recieve data */
#ifdef OS9
    read( in_chan, (char *)ptm->mat[0],
mn*sizeof(double) );
#endif

#ifdef IMS
    ChanIn( in_chan, (void *)ptm->mat[0],
mn*sizeof(double) );
#endif
   return ptm;
}
```

The symbols used to select the proper parts of the code for compilation are defined in the make-file for the different platforms. For each platform, a slightly different make-file is used, which handles the special compilers and other tools needed on that specific platform. When the function above is used in one of the tasks, the function call is the same, no matter what platform the program is running on, but the underlying communication may take place in very different ways.

So far we have considered *code* portability only. We have chosen the conditional compilation approach as a way of dealing with code portability. This approach is not applicable for *data* portability. In our case, the MATLAB file used as input to the real-time program could have been generated on several different platforms. The real-time program must at any platform be ready to accept data from any MATLAB platform. Thus, the program must contain code for importing the different MATLAB data file formats. The difference in data formats originates in the type of CPU of the workstation platforms on which MATLAB is running. PC's using Intel processors are storing the single bytes of double precision floats in reverse order of Apollo and HP workstations using Motorola processors. This problem has already been considered in MATLAB, and the data files contains a type flag, that can be used to detect the type of platform a file was generated. We use this flag to detect if any kind of data conversion is needed.

6 Conclusion.

We have successfully developed a real-time adaptive control program, which is algorithm-portable among three different hardware platforms: a Transputer system, an OS-9 system and a DSP system. It has all device dependent code isolated in a single library and has a simple and general task structure. Hence, it should be easy to port to other platforms as well. In the paper we have given examples of how the portability can be achieved and expanded. It has been demonstrated how some of the flexibility of MATLAB can be brought to a real-time platform.

The example used in this paper is an adaptive control algorithm, however the approach presented should provide the good features of portability, expandability etc. when used with other types of control algorithms as well. The flexibility of the system with respect to the ease of changing features in the algorithm (both structure and parameters) has proven useful in the experimental verification of adaptive control algorithms in a laboratory environment. The software relies completely on the presence of MATLAB. A command interpreter allowing on-line changes in controller parameters and structure has been planned, but not yet implemented.

References

Dahl, O. (1991). An interactive environment for real time implementation of control systems. In Barker, H., editor, *Preprints of the IFAC Symposium on CADCS*, pages 518–523, Swansea, UK. IFAC.

Fortuna, L., Gallo, A., Muscato, G., and Nunnari, G. (1989). Implementation of a self-tuning regulator by using a transputer network. In *Proc. IFAC Symposium on Low Cost Automation*, pages 33–38, Milan, Italy. IFAC.

Henningsen, A., Laursen, S., and Ravn, O. (1991). Real-time adaptive control using CPAS on a MC68010-based process computer. In *Preprints of the IEE International Conference on Control*, pages 370–375, Edinburgh, UK. IFAC.

Inmos (1992). *ANSI C Tool-set Users Manual*. Inmos Ltd., Bristol, UK.

Matlab (1992). *MATLAB Reference Guide*. The MathWorks Inc., Mass. USA.

OS-9 (1987). *Using Professional OS-9*. Microware Inc., Iowa, USA.

Tang, R., Jalel, N. A., Mirzai, A. R., and Leigh, J. R. (1992). Identification and modelling of fermentation process using matlab: A case study. In *Modelling and Control of Biotechnical Processes*, pages 331–334, Colorado, USA. IFAC.