Implementing a Hardware Monitor Using the TMS320C40 Analysis Module and JTAG Interface for Performance Measurements in a Multi-DSP System

Authors: R. Ginthor-Kalcsics, H. Eder, G. Straub, Dr. R. Weiss

EFRIE, France

December 1995
SPRA308

TEXAS INSTRUMENTS

TRADEMARKS

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Figures

Tables

Implementing a Hardware Monitor Using the TMS320C40 Analysis Module and JTAG Interface for Performance Measurements in a Multi-DSP System

Abstract

This application report describes the design and implementation of a hardware monitor that provides information from the processor level up to the application level. It uses the on-chip analysis module of the Texas Instruments (TI™) TMS320C40 digital signal processor (DSP) and a boundary-scan technique according to the IEEE 1149.1 JTAG-standard. The monitor can be used for both single processor and multiprocessor systems. There is no limit on the number of processors monitored. An instrumentation of the software running on the DSPs is not required. The monitor influences the application in terms of runtime but does not change the order of events.

This document was an entry in the 1995 DSP Solutions Challenge, an annual contest organized by TI to encourage students from around the world to find innovative ways to use DSPs. For more information on the TI DSP Solutions Challenge, see TI's World Wide Web site at www.ti.com.

Product Support on the World Wide Web

Our World Wide Web site at www.ti.com contains the most up to date product information, revisions, and additions. Users registering with TI&ME can build custom information pages and receive new product updates automatically via email.

Introduction

It is the aim of each software design to develop an algorithm/application that uses the underlying HW-resources in the best way. This is a complex task in case of multiprocessor systems. The distribution of work amongst the processors greatly influences the performance of the application. Therefore, users need tools to obtain information about the system's behavior and the degree of its utilization.

Monitors are such tools that can provide this information. They use different approaches, like hardware, software, and hybrid monitoring.[1] Examples of these approaches are documented.[2,3,4] These monitors usually yield information of the application and operating system level (e.g. process creation). At the processor level, they can monitor the CPU load but not the use of resources like coprocessors or memory. Furthermore, they often demand an additional instrumentation of the application and/or system software. Since multiprocessor systems feature asynchronous concurrent activities and lack central control, the instrumentation can change the order of events. So, the results delivered by the monitor do not correspond to the real behavior.

Motivation

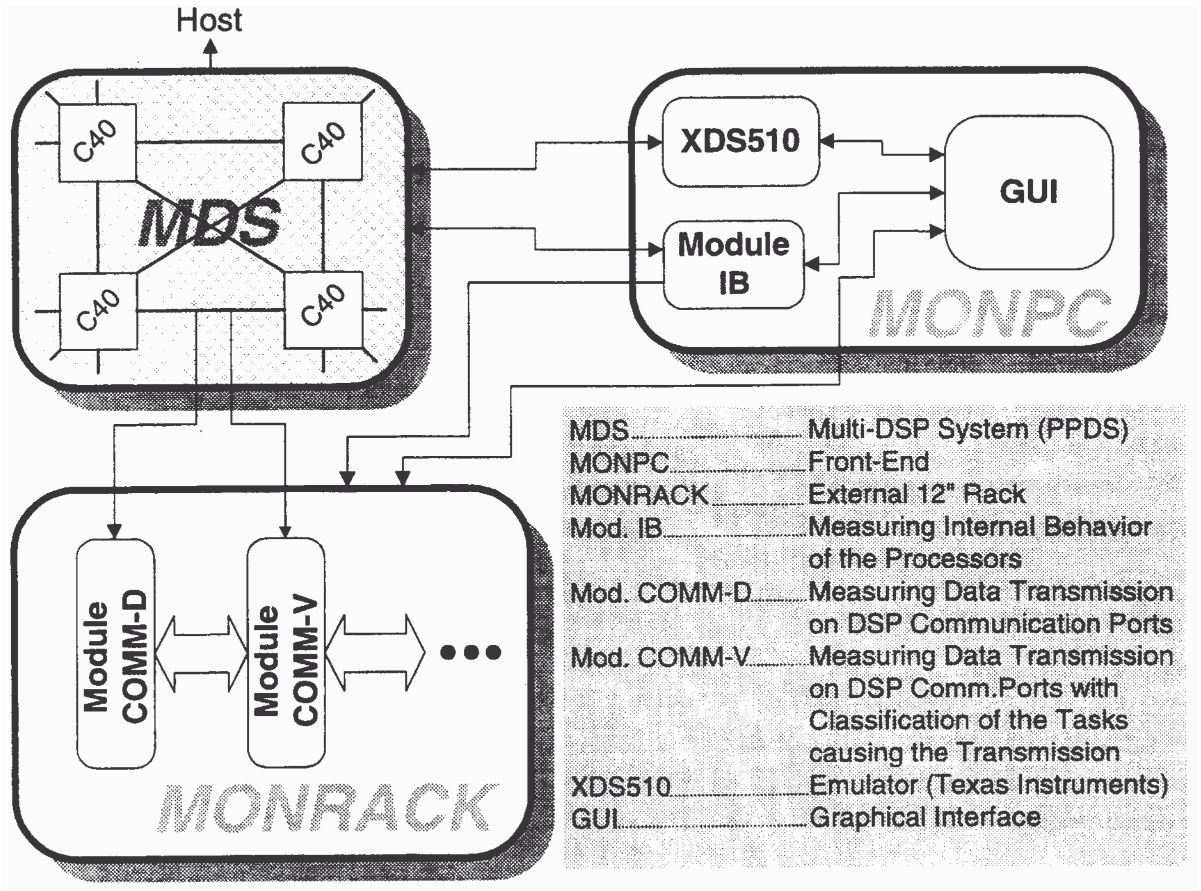
One research area at our institute is the analysis and parallelization of different simulation algorithms.[5] These algorithms are implemented as distributed applications on multi-DSP systems, like the PPDS (Parallel Processing Development System) from Texas Instruments, which contains four C40s. The operating system, VIRTUOSO[6] from Eonic Systems, Inc, has a programming framework based on a virtual single processor model. By this, the same application programming interface (API) is provided across all target platforms from single processor to multiprocessor systems independently of the number of interconnected processors.

The main objects in VIRTUOSO are the tasks as they are the originators of all microkernel services, such as, signaling of semaphores, dynamic allocation of memory, use of mailboxes, protection of resources such as the graphics display. Unfortunately, VIRTUOSO only supports static process (task) mapping, which has to be carried out by the user before runtime.

Beside improvement of the underlying algorithm, the user has two options to increase the performance of his application on the multi-DSP system: variation of the process mapping and memory utilization. A special object file format called the common object file format[7] (COFF), enables the second option. This format encourages modular programming and provides more powerful and flexible methods for managing code segments and system memory. The code and data are divided into blocks, called sections (e.g. text-section, bss-section for uninitialized data, stack, heap). The partitioning into sections has to be done by the assembly language programmer himself or by the compiler in the case of programs written in C.

The user can force the linker to map the sections to certain memory locations. For instance, a section which is frequently accessed could be located in the fast on-chip RAM, whereas another section with fewer accesses could be mapped to slower external memory. Section mapping is a determining factor for performance. For example, by defining user sections and mapping them to the processor's internal RAM the execution time of a petrinet-simulator decreased threefold.[8]

Figure 1. The Modular Monitor System



MDS	Multi-DSP System (PPDS)
MONPC	Front-End
MONRACK	External 12" Rack
Mod. IB	Measuring Internal Behavior of the Processors
Mod. COMM-D	Measuring Data Transmission on DSP Communication Ports
Mod. COMM-V	Measuring Data Transmission on DSP Comm.Ports with Classification of the Tasks causing the Transmission
XDS510	Emulator (Texas Instruments)
GUI	Graphical Interface

Note: Modules COMM-D and COMM-V measure the physical and logical communication respectively. Module IB measures the internal behavior of C40s by using the on-chip analysis module and the JTAG interface.

In order to support performance improvement as mentioned above, we have developed a modular monitor system. The design goals for the monitor were:

- ❑ User transparency (no demand for software instrumentation)
- ❑ On-line data acquisition (early detection of performance bottlenecks)
- ❑ Modularity (measurement of the entire system behavior - from the processor to the application)

Figure 1 shows the modular concept of the monitor. Although the modules COMM-D and COMM-V are not part of the project, they should be mentioned for the sake of completeness. These modules measure the physical and logical° communication load on the C40's communication ports.[9,10]

°We define logical communication as communication which is caused by a particular task.

The module IB provides information related to the utilization of processor resources (memory, DMA coprocessor), to the operating system and the application itself (see Table 1). This module cooperates with the XDS510 emulator from TI and uses the JTAG interface and the on-chip analysis module of the TMS320C40.

The graphical user interface (GUI) allows the user to configure the modules and to control their action. Furthermore, the GUI displays the measured data by providing appropriate views like Kiviat diagrams or Gantt charts.[11,12]

Table 1. Overview of the Information Provided by Module lB

Level	**Information Delivered by Module IB**
Application	Profiling of function calls, variable tracking
Operating System	Workload, state of system objects (semaphores, queues, mailboxes)
Processor (HW)	Memory utilization, DMA

The functionality and implementation of module IB and the graphical user interface will be described in the following chapters.

The Monitor (Module IB)

Analysis Module

The C40's on-chip analysis module allows a higher level of software and hardware debugging capability than simple software breakpoints.[13] It offers the following features:

- ❑ Hardware breakpoints (multiple breakpoints can be selected, e.g. program/data/DMA addresses, interrupts taken, external events on the C40's EMU-pins)
- ❑ Program discontinuity stack
- ❑ Event counting (only one event can be selected) on
  - ■ Program address executed
  - ■ Data address, DMA address executed (with 64K range masking and read/write qualifier)
  - ■ CPU clocks, instructions fetched
  - ■ Interrupts/traps or branches or calls taken; return from interrupt/subroutine/trap

Method of Measurement

The measurement method is event counting[1] thus not producing voluminous event data (in contrast to event tracing which generates time stamped records each time an event occurs). Events are provided by the analysis module (see the *Analysis Module* section). The number of data addresses executed gives information about the access frequency to a certain memory area. This can be a useful hint to instruct the linker to allocate certain sections to other memory areas. The count of DMA addresses executed shows the utilization of the DMA coprocessor. The feature of counting executed program addresses allows on-line function profiling. Note, that only one event can be counted by the analysis module's event counter at a certain time. Several test runs must be performed if information about more than one event is desired.

It is also possible to obtain results concerning the CPU workload. If no other tasks are running or they are all blocked, VIRTUOSO invokes an idle-task which has the lowest priority. The task continuously increments a variable. The location is known before runtime. Because all storage locations can be accessed via the JTAG interface, it is easy to get the value of this variable.

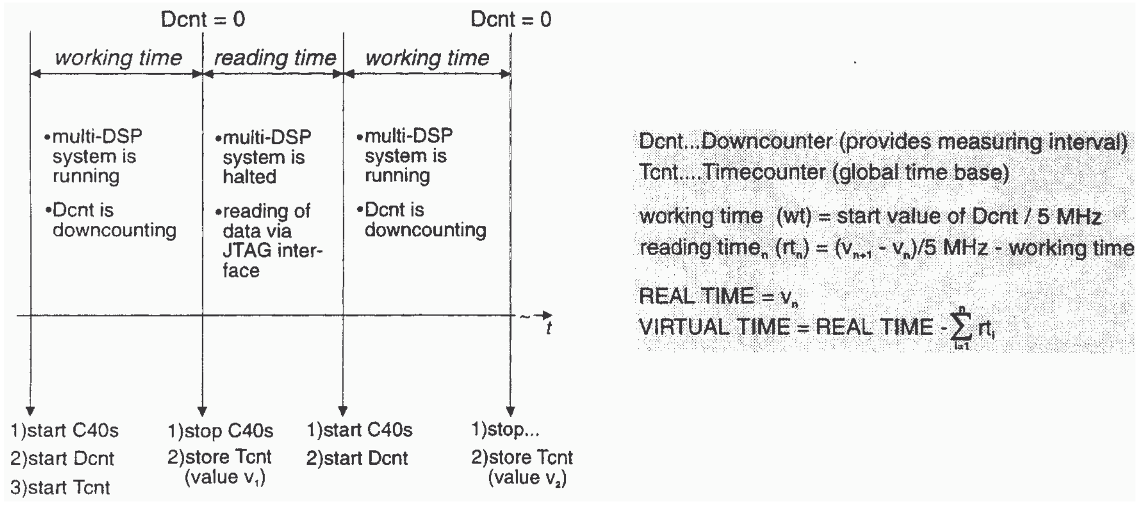
The process of the measurement is as follows. At the beginning the application is started via the XDS510 emulator. A sensor on the module IB detects this and starts a counter. This counter provides the measurement interval. When the time interval has been elapsed (the counter has reached zero), the module simultaneously stops all the C40s on the multi-DSP system via their EMU-pins. Then the value of the analysis module's event counter and/or the content of the desired memory location is read through the JTAG interface (and the XDS510). After the reading is finished, the system is restarted.

This cycle continues until termination of the application or the user aborts. There is a second counter on the module IB that runs the entire time the monitoring lasts. At the end of monitoring, two time values are delivered: the real time, which has been elapsed and a virtual time. The virtual time depicts the time which the application under observation would have consumed without the monitor. The measurement cycle and the calculation of the two time values are illustrated in Figure 2.

Using the JTAG interface and the analysis module for performance measurements offers four advantages:

- ❑ No additional hardware is needed for event counting because this is done by the analysis module
- ❑ No source code instrumentation is necessary
- ❑ No extra interface to the C40 has to be implemented because the measured data can be transmitted via the JTAG interface
- ❑ The number of processors is not restricted because further C40s have only to be "hooked" into the boundary-scan path

Figure 2. Measurement Process and Calculation of "Real" and "Virtual" Time



Dcnt...Downcounter (provides measuring interval)
Tcnt....Timecounter (global time base)

working time (wt) = start value of Dcnt / 5 MHz
reading time$_n$ (rt_n) = (v_{n+1} - v_n)/5 MHz - working time

REAL TIME = v_n
VIRTUAL TIME = REAL TIME - $\sum_{i=1}^{n} rt_i$

The module IB produces a predictable perturbation of the application which is manifested as an increase of program execution time. However, by the global and simultaneous "freezing" of the system the order of events is not changed.

Implementation

Module IB is implemented as a PC plug-in board to ease its cooperation with the XDS510. The logic of the module is implemented by an FPGA (Xilinx) module. Figure 3 shows the block diagram of the module. The module is controlled by reading from or writing to the appropriate status and control registers (see Table 2). The main parts of the module comprises the *Downcounter* and the *Timecounter*. The Downcounter provides the measuring interval. Each time the Downcounter value reaches zero the module halts the C40s by either pulling their EMU-0 or EMU-1 pins low. Then the contents of the analysis module's event counters are transferred via the XDS510 emulator to the PC. (The software for programming the XDS510 is provided by TI's "Emulation Porting Kit".[14], see *The Graphical User Interface (GUI)*).

The time needed for this transfer ranges from 70 – 500 ms. In order to get a reasonable ratio (e.g. 10:1) between the runtime and the reading time, the measuring interval – during this time the C40s are running - has to be in the range of several seconds. On the other hand, the interval has to be short enough to ensure that the 12 bit event counter of the analysis module does not overrun (this happens at most every 164 µs in a TMS320C40 operated at 50 MHz). Therefore the *Downcounter* is realized as a 5 MHz, 25 bit wide counter. By this, the range of the measuring interval is 200ns – 6,7 seconds. After reading of data, the control software on the PC restarts the *Downcounter* and the XDS510 starts the C40s. It is possible to configure the event counters to signal an overrun on the EMU-pins. The module IB can be programmed to stop the *Downcounter* if this happens. This way the user can determine if the measuring interval is too long.

The *Timecounter* represents the global time base. It starts at the beginning and runs nonstop until the end of the measurement. Each time the measuring interval lapses, the value of the *Timecounter* is latched and can be read by the control software. The *Timecounter* runs over a cycle of 6,7 seconds. So, it must be polled cyclically in order not to lose any timing information.

The TAP sensor, which is positioned at the JTAG connector of the PPDS, imitates the TAP controller's state machine. The state "Update-IR" indicates that a new instruction has been shifted into the instruction register of the boundary scan. Because the run command sent by the XDS510 contains several instructions, the FPGA *Instructioncounter* detects the start of the C40s by counting the "Update-IR" states. The C012 link adapter stops the COMM-modules of the monitor system if necessary.

Figure 3. Block Diagram of the Monitor Module

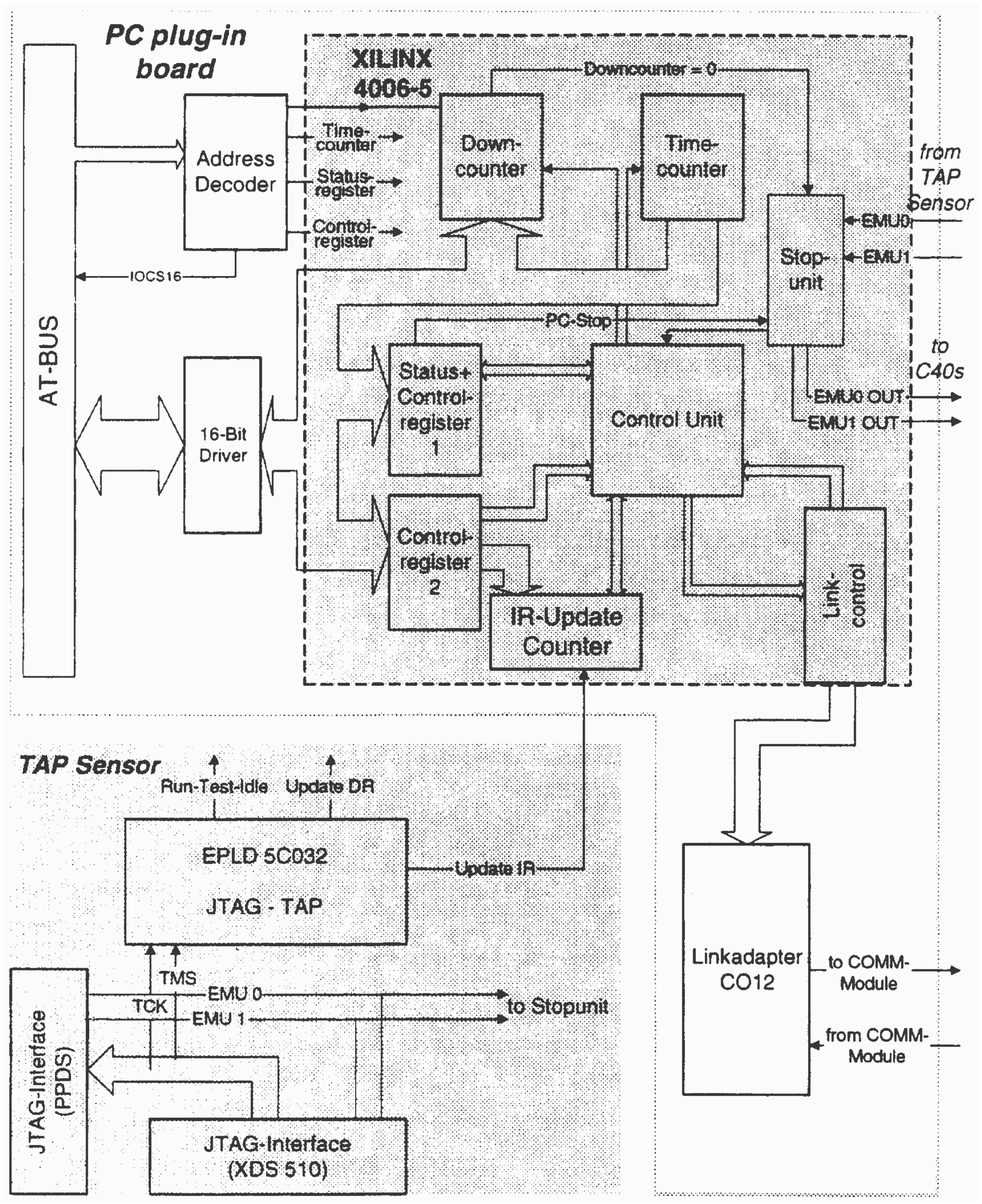


Table 2. Control/Status Register Definitions.

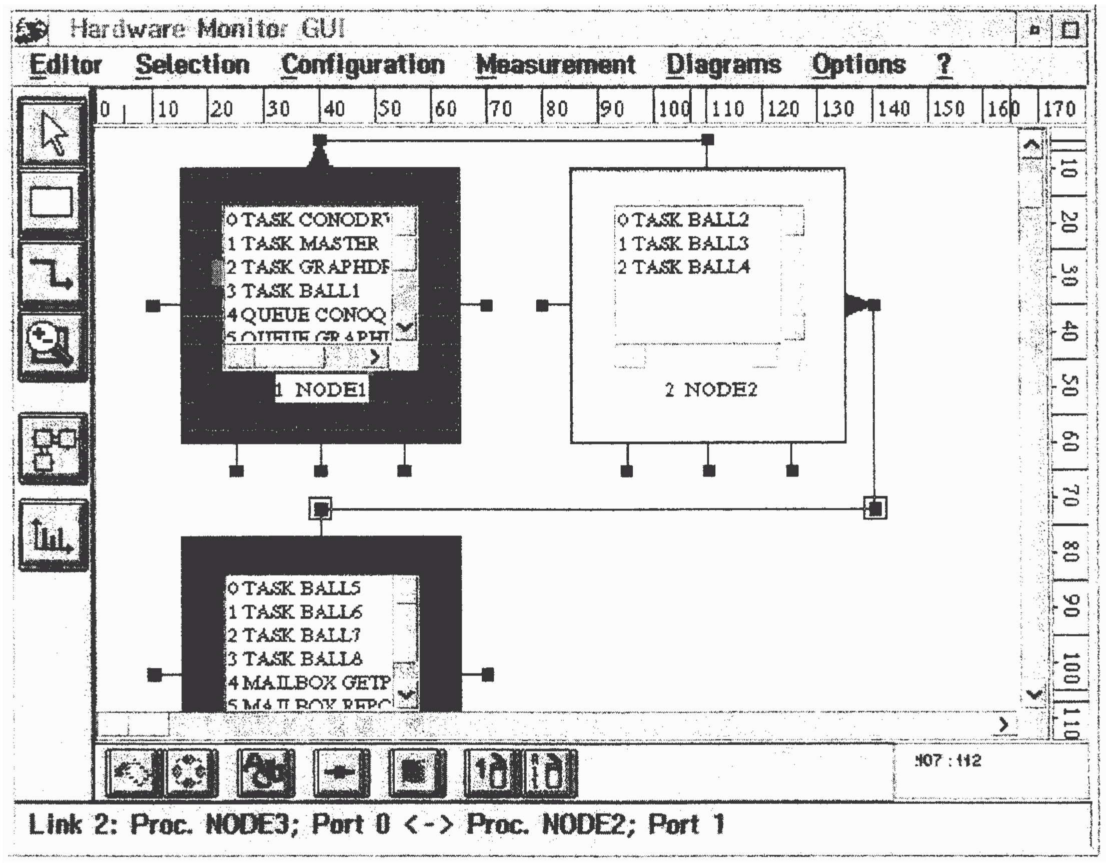
Status Register/ Control Register 1		
Bit Number	**Meaning** (If Logical High)	**r = Read Only** **w = Write Only** **rw = Read/Write**
0	Overrun of Timecounter has occurred	r
1	Access to module IB is possible	r
2	Link adapter C012 is ready	r
3	External stop signal has been received	r
4	Start Downcounter	w
5	Stop Downcounter	w
6	Reset	w
7	Lower bits (0-9) of the 25 bit Down- and Timecounter are accessible	rw
8	Generate a test signal for the instructioncounter	rw
9-11	Not used	-
12-15	ID number	r

Control Register 2		
0	Stop module IB if EMU0 is low	rw
1	Stop module IB if EMU1 is low	rw
2	Stop module IB if EV1IN is low	rw
3	Enable stop of downcounter (bit 5 of control register 1)	rw
4	Enable start detection (by the Instructioncounter)	rw
5	Send a start/stop signal via C012 to the COMM modules	rw
6	If stop (bit 5 of control register 1) then pull down EMU0 for 7 ms	rw
7	If stop (bit 5 of control register 1) then pull down EMU1 for 7 ms	rw
8-15	Number of impulses (Update IR states) of the run command (necessary for the instruction counter)	rw

In Appendix A the schematics of the module IB, the TAP sensor, and the logic implemented in the FPGA are listed.

The Graphical User Interface (GUI)

Figure 4. The GUI



Link 2: Proc. NODE3; Port 0 <-> Proc. NODE2; Port 1

Note: The GUI with an example for a system configuration. The application runs under VIRTUOSO, therefore the processors contain list boxes with the names of the tasks and system objects (queues...). The buttons on the left margin are used for editing (e.g. "Draw processor", "Draw connection", "Zoom"...).

Function

The GUI has to accomplish three functions:

- Graphical representation of the multi-DSP system and the processors in use respectively. That means that the user to graphically edit the arrangement of the processors and the connections of the communication ports (see Figure 4). The edited system description can be saved as a file. If the application runs under VIRTUOSO, it is also possible to automatically load the system description from VIRTUOSO's system files.

- ❑ Selection of the monitor modules (IB, COMM), their configuration (e.g. length of the measuring interval, end condition for measurement), and the control of their actions.
- ❑ Online display of the measured data (see the *Diagrams* section).

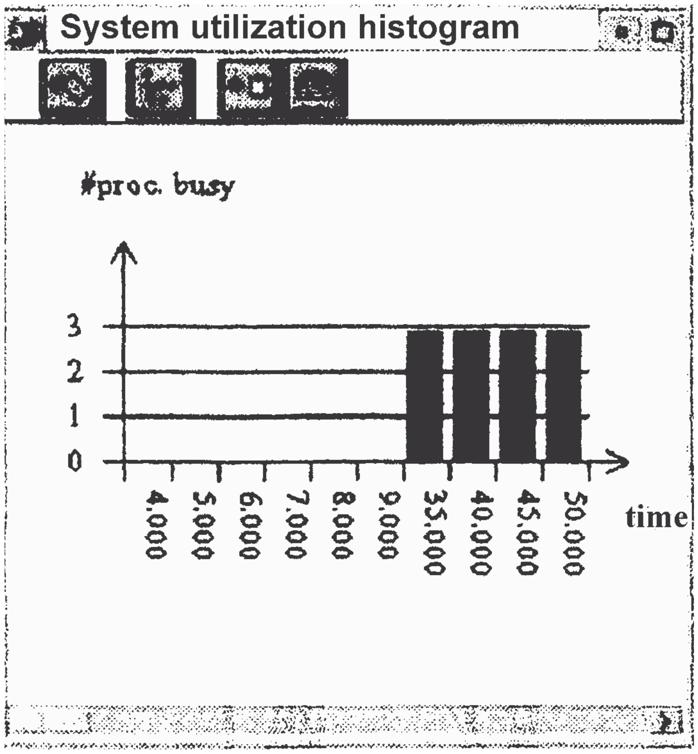
The GUI has been developed with the IBM CSet++ 2.1 and the Starview C++ class library.[15] This class library is available for many platforms (Windows 3.1, Windows NT, Windows 95, OSF/1, OS/2, AIX, Solaris, Macintosh/System 7, and HP-UX). Therefore the application can be easily portable to other systems.

Diagrams

The main function of the GUI is to visualize the performance data (the values of the analysis module's event counters and the variables representing the idle times under VIRTUOSO). For that purpose, the GUI provides several diagrams:

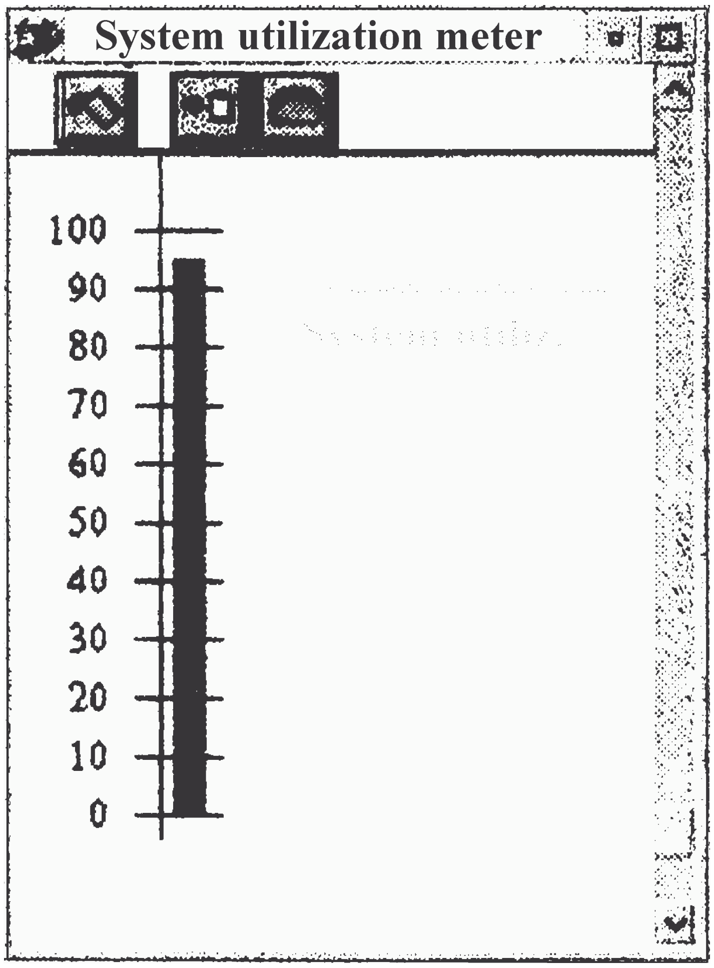
System Utilization

Figure 5. Utilization Histogram



Note: This diagram shows the percental usage for the multi-DSP system.

Figure 6. Utilization Meter

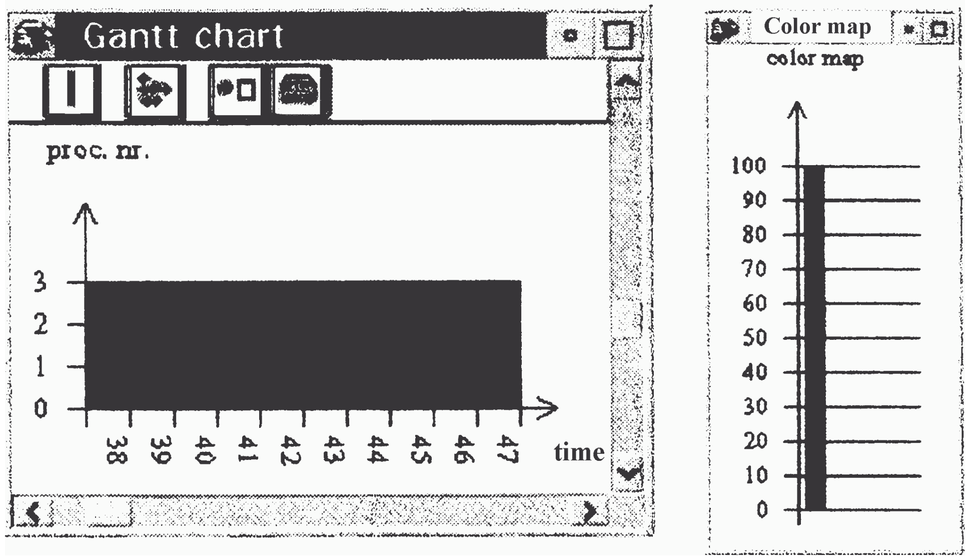


Note: The utilization meter shows the same information as the utilization histogram, but only for a certain point in time (there is no time axis).

Processor Utilization

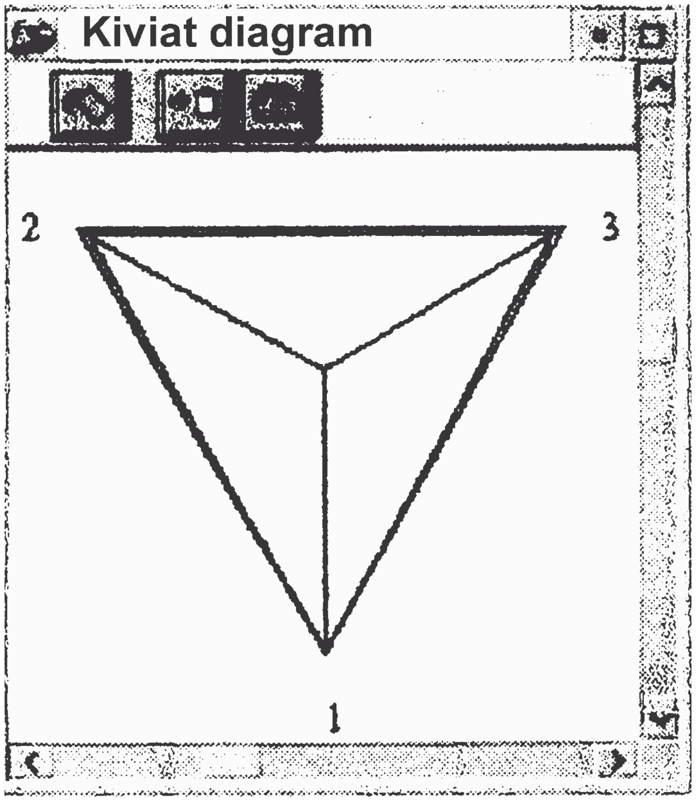
The processor utilization diagrams show the usage of each processor in percent.

Figure 7. Gantt Chart



Note: The color map assigns the colors to the percental usage.

Figure 8. Kiviat Diagram



The utilization diagrams can only be displayed if the application runs under VIRTUOSO.

For the following diagrams the event counter's values from the TMS320C40 analysis modules are used (the user must specify the address ranges to be monitored in advance):

- ❑ DMA Utilization - This diagram shows the number of DMA-accesses (reads and writes with 64K range masking) over the time measurement.
- ❑ Memory Utilization - This diagram shows the number of data addresses executed (read and write with 64K range masking) over the time.
- ❑ Profiling - This diagram shows the number of program addresses executed over the time.

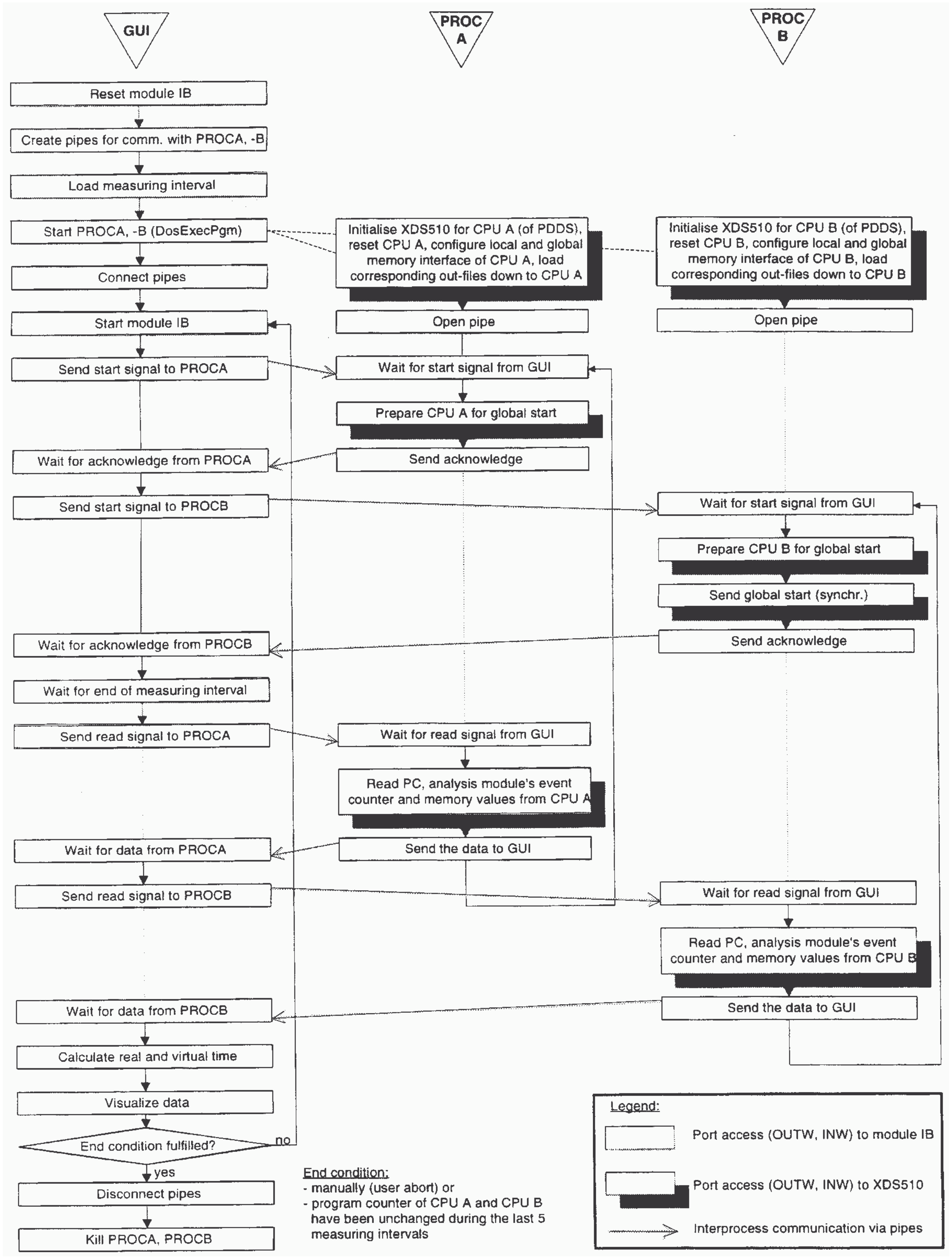
There also exist other diagrams (e.g. system communication load, task communication matrix) which can be viewed if the COMM-modules of the monitor system are used.

All the diagrams can be opened and closed dynamically. Further, it is possible to save the current diagrams to the clipboard or to the hard disk.

Cooperation with the XDS510

The XDS510 is used for the configuration of the analysis modules and the reading of the values from their event counters. The EPK (Emulation Porting Kit) from TI provides the source code (several C functions) for programming the XDS510.[14] These functions use "old" API-calls from OS/2 1.x, written in16 bit code! However, the GUI is a 32-bit application using the OS/2 2.x API calls. So, it can only use the EPK functions if they are in a Dynamic Link Library (DLL). Unfortunately the EPK functions are not reentrant (e.g. TARG_init). Thus, we had to write separate programs (PROCA.EXE - PROCD.EXE for each processor of the PPDS) which configure the various analysis modules and read the data via the JTAG interface. Figure 9 shows the flow of the programs and their interprocess communications assuming that two DSPs should be monitored.

Figure 9. Process Flow of the GUI and the Programs Controlling the XDS510


GUI
PROC A
PROC B
Reset module IB
Create pipes for comm. with PROCA, -B
Load measuring interval
Start PROCA, -B (DosExecPgm)
Connect pipes
Start module IB
Send start signal to PROCA
Wait for acknowledge from PROCA
Send start signal to PROCB
Wait for acknowledge from PROCB
Wait for end of measuring interval
Send read signal to PROCA
Wait for data from PROCA
Send read signal to PROCB
Wait for data from PROCB
Calculate real and virtual time
Visualize data
End condition fulfilled?
no
yes
Disconnect pipes
Kill PROCA, PROCB
Initialise XDS510 for CPU A (of PDDS), reset CPU A, configure local and global memory interface of CPU A, load corresponding out-files down to CPU A
Open pipe
Wait for start signal from GUI
Prepare CPU A for global start
Send acknowledge
Wait for read signal from GUI
Read PC, analysis module's event counter and memory values from CPU A
Send the data to GUI
Initialise XDS510 for CPU B (of PDDS), reset CPU B, configure local and global memory interface of CPU B, load corresponding out-files down to CPU B
Open pipe
Wait for start signal from GUI
Prepare CPU B for global start
Send global start (synchr.)
Send acknowledge
Wait for read signal from GUI
Read PC, analysis module's event counter and memory values from CPU B
Send the data to GUI
End condition:
- manually (user abort) or
- program counter of CPU A and CPU B have been unchanged during the last 5 measuring intervals
Legend:
Port access (OUTW, INW) to module IB
Port access (OUTW, INW) to XDS510
Interprocess communication via pipes

Summary and Conclusion

In the course of this project, we developed a monitor, which measures processor related event data using the analysis module and JTAG capabilities of the C40 respectively. Event data provided is the utilization of memory, CPU, and DMA coprocessors. It is also possible to evaluate performance parameters of the operating system (e.g. workload).

Our approach offers many advantages. No additional hardware is needed for event counting because this is carried out in the the analysis module, no source code instrumentation is necessary, no extra interface to the C40 has to be implemented because the measured data can be transmitted via the JTAG interface, and, finally, the number of processors in the multi-DSP system to be monitored is not limited.

The only influence on the measured system is the increase of execution time, but this perturbation is predictable and does not change the partial order of events.

The monitor is equipped with a graphical user interface which is an OS/2 application and which visualizes the measured data on-line by providing views like Kiviat-diagrams.

Future work will concentrate on experiments, so that we can prove the practicability of the monitor. For example, we must investigate “how to choose" the length of the measuring interval in order to obtain meaningful results without extending the execution time too much. There will also be an emphasis on measuring more operating system related parameters using the event tracing method. VIRTUOSO includes a debug kernel which provides snapshots of the VIRTUOSO objects like queues, mailboxes, and so on. These snapshots are timestamped and can be accessed via the JTAG interface.

References

[1] D. Reed. Experimental Analysis of Parallel Systems: Techniques and Open Problems. In *Proceedings of the 7th Int Conference on Computer Performance Evaluation - Modeling Techniques and Tools,* Vienna, May 1994, pp.25-51

[2] P. Dauphin et al. ZM4/Simple: A General Approach to Performance Evaluation of Distributed Systems. In Casavant and M. Singhal eds., *Readings in Distributed Computing Systems*, IEEE Computer Society Press, Los Alamitos, California, 1994, pp. 286-309

[3] A. Mink et al. Multiprocessor Performance-Measurement Instrumentation. *IEEE Computer*, Sept.1990, pp. 63-75

[4] Ch. Scheidler, L.Schäfers. TRAPPER: A Graphical Programming Environment for Industrial High Performance Applications. In *Proceedings of the 5th International Conference on Parallel Architectures and Languages Europe PARLE '93*, Munich, Germany, June 1993, pp.403-413

[5] M. Platzner, B. Rinner and R. Weiss. A Distributed Computer Architecture for Qualitative Simulation Based on a Multi-DSP and FPGAs. In *Proceedings of the Euromicro Workshop on Parallel and Distributed Processing,* San Remo, Italy, January 1995, pp.311-318

[6] E. Verhulst. Virtuoso: A virtual single processor programming system for distributed real-time applications. In *Microprocessing and Microprogramming,* Euromicro Journal, 1994, pp.103-115

[7] Texas Instruments. *TMS320 Floating-Point DSP Assembly Language Tools User's Guide*; 1991

[8] J. Schinnerl. Portierung des Petrinetz-Simulators auf die Signalprozessoren TMS320C30/C40. *Project Report,* Institute for Technical Informatics, 1992 TU Graz

[9] R. Ginthör-Kalcsics, R. Weiss. Design and FPGA Implementation of a Hardware Monitor for Performance Measurements in Multi-DSP Systems (TMS320C40). In *Proceedings of the International Conference on Signal Processing Applications and Technology ICSPAT '94,* Dallas, Texas, October 1994, pp. 536-541

[10] R. Ginthoer-Kalcsics, K. Laher. Process Mapping in a Multiprocessor System by Means of a Hardware Monitor. In *Proceedings of the ISCA Eighth International Conference on Parallel and Distributed Computing Systems,* Orlando, Florida, September 1995, pp.583-588

[11] P. Kiviat. Software Unit Profiles and Kiviat Figures. In *ACM SIGMETRICS, Performance Evaluation Review,* Vol.2, March 1973, pp.2 - 12

[12] M.T.Heath, J.A.Etheridge. Visualizing Performance of Parallel Programs. *IEEE Software,* Vol.8, May 1991, pp. 29-39

[13] Texas Instruments. *TMS320C4x C Source Debugger User's Guide*; 1992

[14] Texas Instruments. *Emulation Porting Guide TMS320C4x*, 1994

[15] Star Division. *StarView C++ Class Library Vers. 2.0 User Manual*, 1993