Chapter 5

The master-checker mode

In order to assure a high availability of a computer system, error detection and correction are the most crucial factors. For instance, if checkpointing is applied to assure a system recov ery after faults, the effectiveness assumes a reliable error detection in order to restart the sys tem from a really fault free state. Accordingly, the main requirements againts run-time checks are a high fault coverage and low error latency at the price of an affordable hardware overhead.

In systems, where the CPU type applied supports the master-checker mode, a very high level of availability can be reached, by simply duplicating the processor. Another main advantage of this principle is, that it ptrovides a fully hidden protection, i.e. any software can run on the master-checker based configuration without any modifications (naturally with the exception of the error exception handlers).

5.1 The master-checker principle

Multiplicated modular redundancy (n-MR) is a well known principle in fault tolerant system design. When the function to be protected is executed simultaneously on multiple units having identical inputs, as far the system is fault-free, each unit generates the same output. In the case of a fault in any unit a mismatch appears on the system outputs.

The system setup (Fig. 1.) is based upon duplication in the basic master-checker mode .

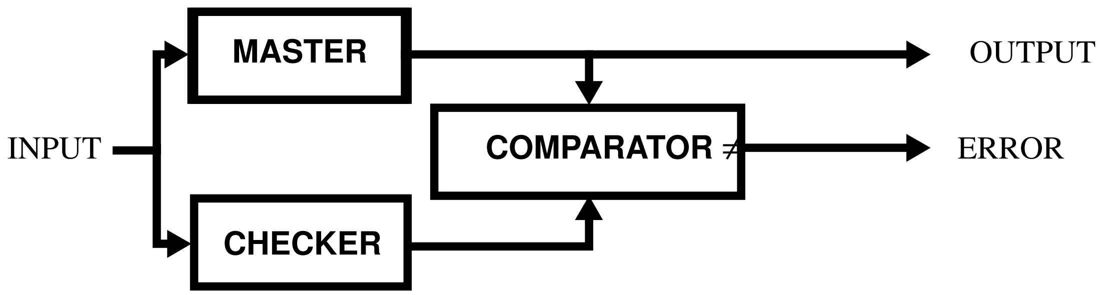


Fig. 1. Master-checker mode

Both of the master and the checker units share identical inputs and run lock-step. The master generates the output signals of the system. The checker unit performs the same operation as the master and the outputs of both units are compared. In the case of a mismatch an error is detected and signalled to the "outer world". It should be pointed out, that the application of a combinatorial comparator assumes a strict synchronism of both units.

As in all of the duplicated systems, this approach supports only fault detection. In the case of a discrepancy it can be not decided, which unit produced a faulty output, correspondingly neither unit level diagnostics, nor error correction can be performed.

Another restriction on the fault detecting capabilities results from the processing of a shared input by both processors. In the case of an arbitrary error in the input both units generate identical and faulty outputs without triggering any error alarm.

5.1.1 Circuit support of the master-checker mode

The setup of master-checker mode is supported by chip manufacturers, like Intel in the Pentium , or Motorola in the MC88000 family by the integration of the comparator on the CPU chip. This way the hardware overhead required for the master checker setup is reduced to the duplication of the CPU chip (set).

The internal and external states of an active output signal are compared by means of the integrated comparator circuitry (Fig. 3.) . If a mismatch is detected, than an error impulse is gen-

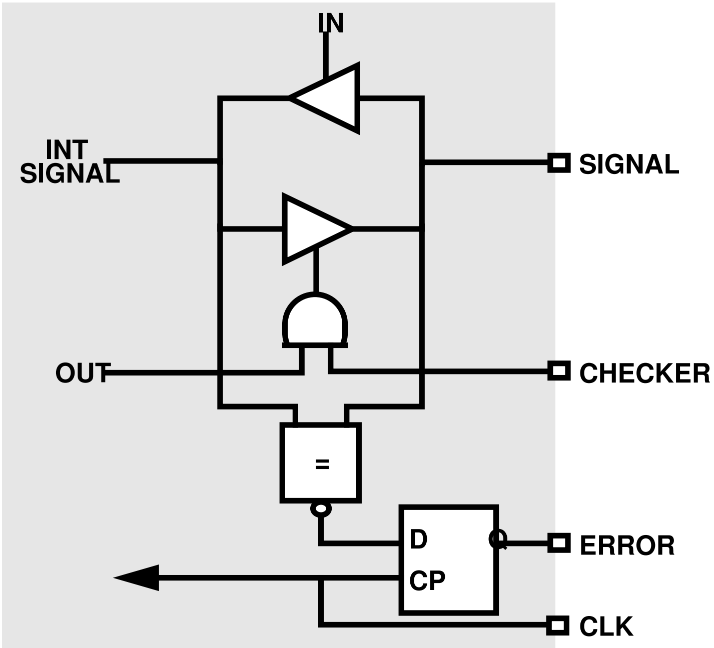


Fig. 2. Master-checker circuitry

erated by the sampling flip-flop. (In the real circuit implementation the errors on the different outputs are summed up by a logic OR operation and the chip has a common ERROR output). The comparator checks the equality of the internal and external signal values, if the output is is active (e.g data lines during write operations). This way a msimatch between the driving and driven signal values triggers an alarm, e.g. indicating a bus stuck-at fault.

The input and output pins of the master and checker circuits are pairwise interconnected in master-checker mode(Fig. 4.). An active level on the master-checker mode select control pin of the checker circuit disables its output drivers, correspondingly the comparator checks the matching of the internal signal value generated by the checker and the external one driven by the master. In the case of a mismatch an error will be signalled.

5.1.2 Implementation issues

We will use the Motorola MC88k processor as illustration, in order to show the interference of CPU architecture design principles with the master-checker mode. MC88k is of a Harvard architecture having two internal communication buses: (Fig. 5.),

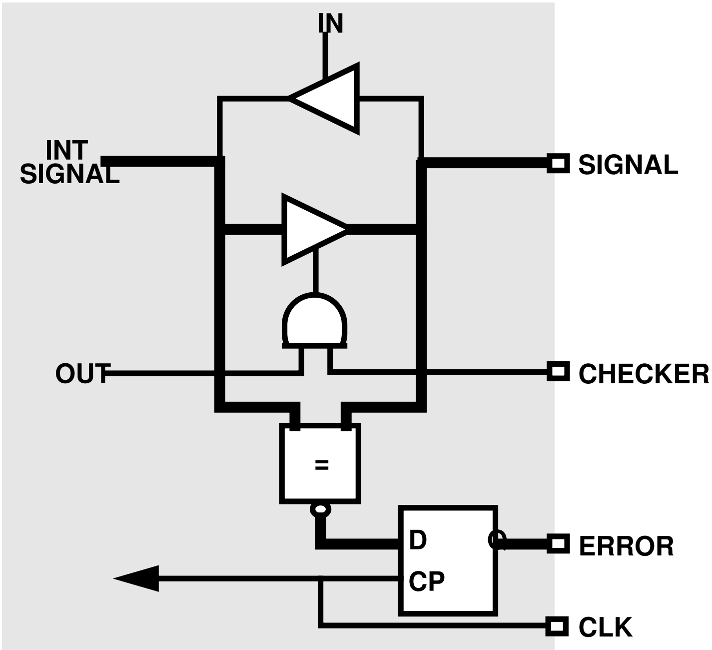


Fig. 3. Master-checker circuitry

- Separate P buses code and data transfer between the CPU and MMU chips;
- M bus as the external interface of the processor, connecting the MMU chips and the external memory and peripherals.

The chip set supports the setting up of the master-checker mode both on the M and on the P bus interfaces. Using P bus master-checker mode, the operation of the CPU will be checked. M master-checker assures a global supervising of the entire processor (CPU+MMUs) operation during the communication with the external world. The implementation of the P bus and combined P and M bus master-checker setups requires a special processor board design allowing a local duplication and interconnection of the chips. On the other hand, this pin-by-pin interconnection can be implemented on the backplane in M bus MC mode, if the processor board has the unbuffered M bus pins of the CPU chip, as bus interface.

The original configuration (Fig. 5. a) of a two independent processor structure can be transformed by setting up master-checker mode on the M-bus (Fig. 5. b) emulating an error detecting single processor configuration retaining a full software compatibility . CPU and MMU error signals are processed by an external error logic.

Master-checker mode assumes a fully clock-synchronous running of both CPUs. As input signals are sampled by the same clock in both of the logic in the master and checker CPUs (Fig. 4. , signal setup time violations can result in a mismatch in the internal states of the master and the checker circuits producing false error reports.

Accordingly, a strict satisfaction of the prescribed setup time specification is required prohibiting asynchronous input signals. E.g. the interrupt input in the normal configuration can be activated at an arbitrary time, a setup time violation results only in a single clock phase uncertainty in triggering the interrupt handler. However in master-checker mode the same skew may result in an asynchronism between master and checker, producing a sequence of error signals.

Note, that the required identity of the outputs of the master and checker implies a need for a deterministic program run, for instance necessitating a proper initialization of the entire internal CPU/MMU storage area in order to avoid false alarms, due to mismatches between uninitialized storage locations in the master and checker.

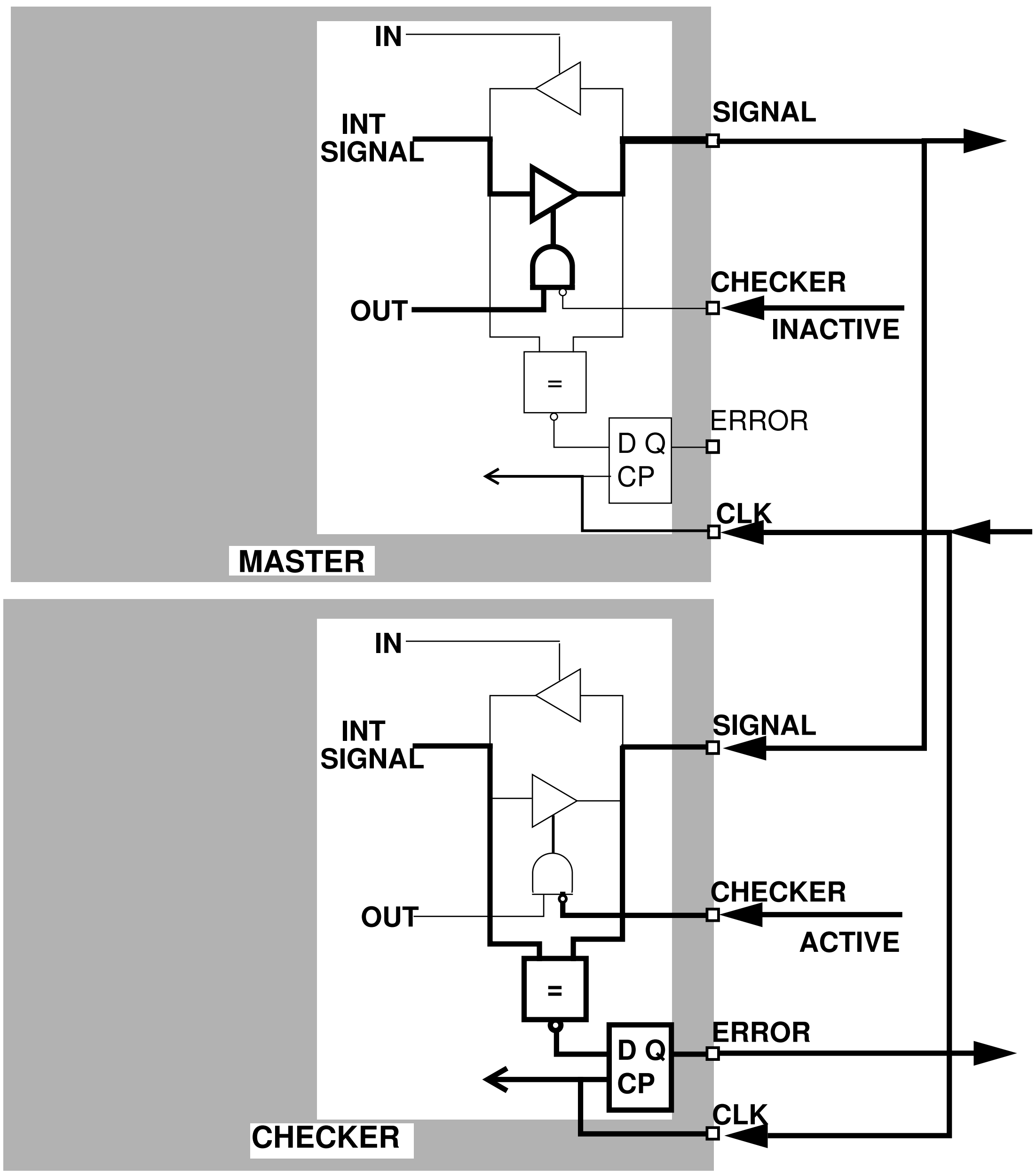


Fig. 4. Master-checker mode setup

5.1.3 Error handling

Two groups of errors can be distinguished upon the duration of the error alarm signal. :

- *Fatal errors,* where the two processors were falling out of synchronism and a continuous burst-like error signal stream is generated, for instance originating in faults during MMU accesses.
- *Short transients* of a duration of typically a single cycle resulting from transient errors due to noise effects on the bus; or a single manifestation of a fault in some internal state of the master or the checker (cache, register mismatch e.t.c).

An error alarm longer than a single bus cycle, or repeated burst-like alarms are handled as fatal errors. The potential actions undertaken by the error logic is as follows:

- In the case of a *fatal error*, no more protection is assured by the master-checker mode, as the system is from the beginning of the error burst unsupervised. The only way to regain

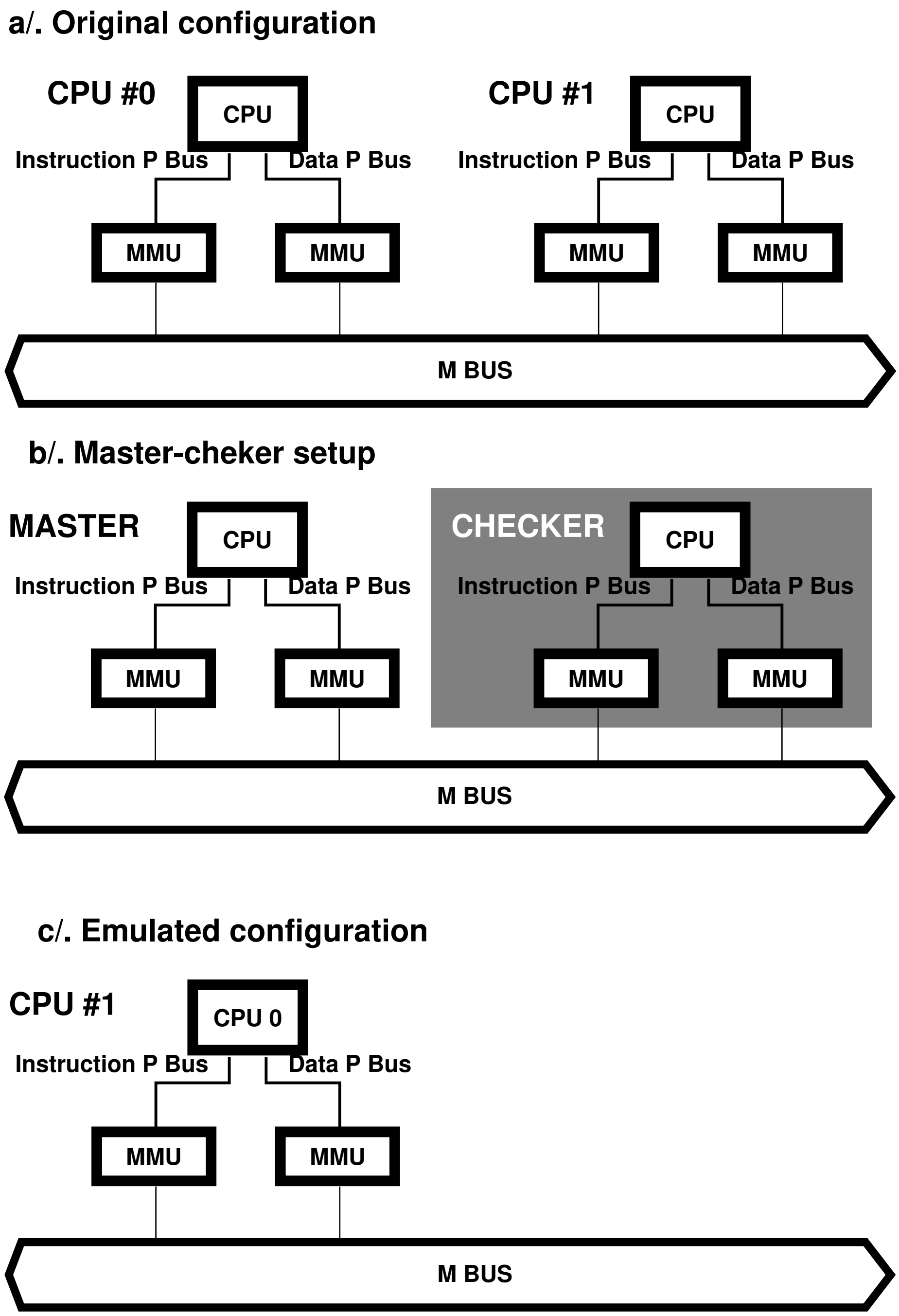


Fig. 5. Master-checker setup

resynchronization is a hard reset, after which a system restart using the last saved checkpoint data set can be initiated.

- If a *transient error* was detected, there are essentially two alternate actions to be undertaken depending on the origin of the error estimated by the follow-up diagnostics phase:
  - If the error resulted from a *mismatch* in an internal storage location of the master and the checker respectively, than no error correction can be performed and the same must be applied as for a *fatal error*.
  - If the error is generated during a *bus transfer* between cache and memory, there are total of three copies of the data, sufficient to perform an *error correction*, if detailed information on the faulty bus transaction is available. It should be pointed out, that the computing time overhead related to the error correction is by some orders of magnitude less,

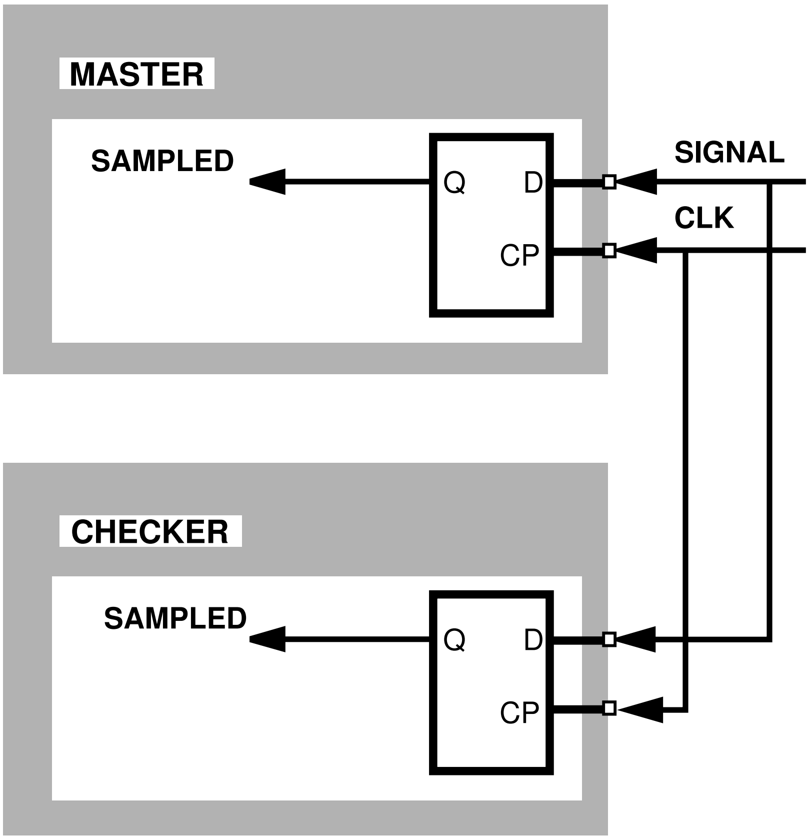


Fig. 6. sampling of the input signals

than that related to a system restart or recovery.

In order to have a starting point for the diagnostics and a potential system recovery for correctable transient errors, the error logic has to store the address and control information of the first transaction marked as faulty. The proposed error detection sequence for transient errors is as follows:

- If a transient error is detected, than an interrupt and/or bus error signal is generated. In the error logic the values of the address and control signals during the faulty M bus transfer are stored and an "Error processing pending" flag is raised.
- The interrupt or error exception handler subroutine performs a full cache flush and writes all of the internal register contents (CPU, MMU) onto the M bus.
  - If there was any mismatch in the contents of the internal storage of the master and slave processor, this will be manifested during this phase. As result of the checking of the full internal state of both processors, a repeated error signal will be generated during the bus cycle writing out the mismatching data. This results in an "error overrun", i.e. the error logic will initiate the same interaction as for fatal errors.
  - If all data in the internal storage locations of both processors are identical, than the error is localized as a faulty bus transaction. In this case the processor can decide on retrying this operation by analysing the stored "snapshot" of the faulty bus cycle.

5.2 Fault coverage

The duplicated processor configuration in master-checker mode can be considered as practically fail safe from the users point of view, with the exception of a few of non-duplicated bus arbitration functions. (In a current paper on fault injection experiments an significantly greater error coverage, than 90% is reported for duplicated systems [6]. , even taking masked errors into account).

The most significant limitation in the error detecting capabilities of the master/checker mode originates in unprotected common resources located outside of the processing kernel. E.g. the MMU automatically downloads logical-physical memory address translation tables from the main memory. If some data in this table are in semantic error undetected by syntactic checks, than the faulty memory mapping can result in very hard to diagnose errors, as both processors use the same set of faulty address translation data.

Accordingly, the fault coverage of the master-checker mode from the general point of view can be summarized as follows:

- Practically complete for internal errors in the processors
- External (or common-mode) errors will be undetected.

5.2.1 Explicitly observed signals

Master-checker mode evaluates only the following active output signals during data transfer on the external bus interface (marked by a dotting in Fig. 7.). :

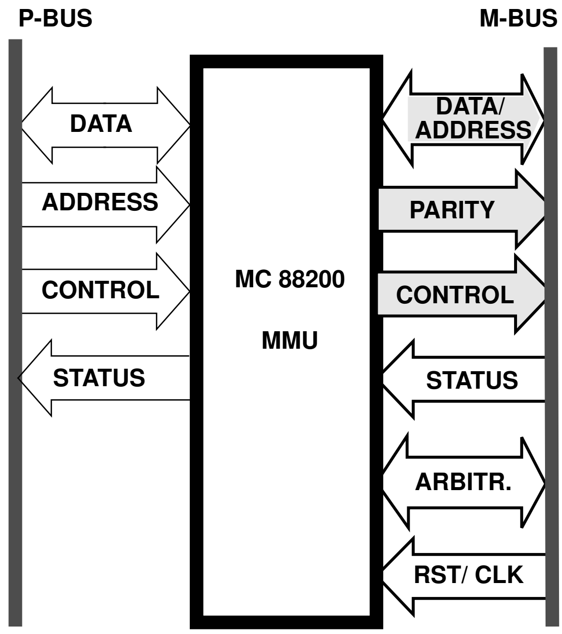


Fig. 7. Checked signals

- The address and control signals during memory reads;
- Address, data, and control lines during memory writes.

Only indirect observability is assured for the input (e.g. interrupt, bus arbitration) signals of the CPU, based upon their interaction with the the program and data flow.

Additionally to common mode errors remaining undected, the master-checker setup can detect only faults observable directly on the CPU pins. If the bus signals are separated from the CPU pins by bus drivers due to DC and capacitive load reasons, the bus belongs to the "external world" and errors in the bus transfer to the system elements remain undetected.

5.3 Error latency

The master-checker setup sooner or later detects any internal processor error leading to a mismatch in the results, however latency hardly depence on the location, where the master checker is applied. On the P bus every cache/memory cycle is checked, but on the system bus only those directed to the main memory.

If aiming only at performance maximization, such a cache architecture and update policy guaranteeing a high cache hit rate are to be selected, preferably limiting memory copyback until the very end of the calculation. As an error corrupting data in the cache will be detected at the first write onto the M bus, *in the worst case error latency can reach the full computing time*!

On the other hand a long error latency means ahigh probability of error propagation, thus the lack of error confinement in time. Long error propagation chains prohibit an effective fault diagnosis, degrading recovery to a symptomatic treatment only

5.3.1 Cache-memory update policy and its impact on the checking

A typical MMU chip supports the following *cache-memory update policies* (Fig. 8.)

- *Cache inhibit*, in which case all read and write operations result in a memory access.
- *Writethrough mode,* in which case the read operations act on the data in the cache, but modified data are immediately rewritten into the memory.
- *Copyback* mode, in which case all read and write operations that hit in the cache affect only the data in it, with the exception of swaps after a cache miss, or by a software-initiated cache flush.

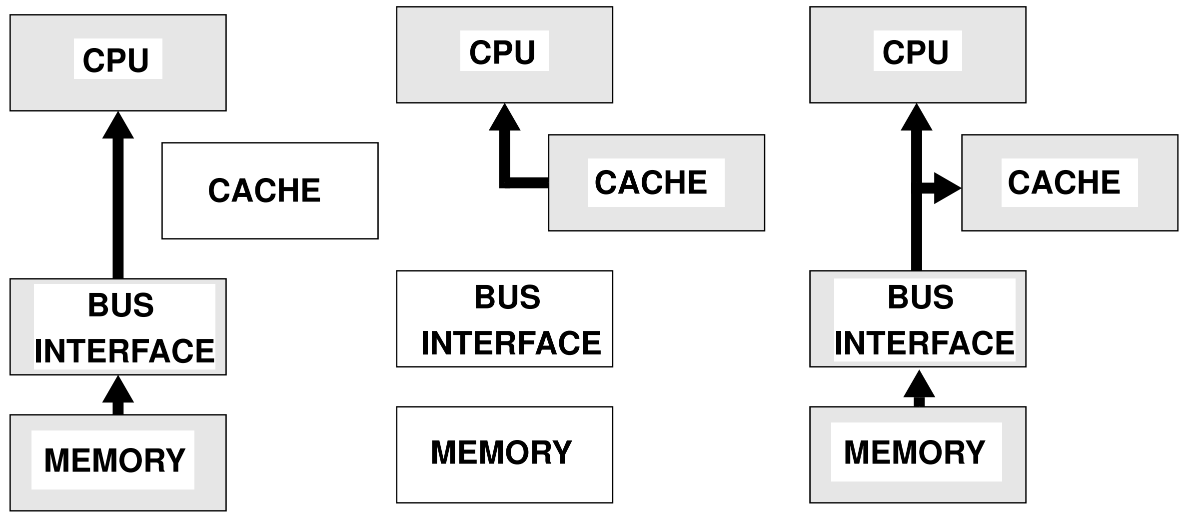
This list represents an increasing order of performance, as less and less main memory transfers are required and an access to the main meory has an associated time requirement of 3-10 times higher, than a cache acces. No unique policy is required for the entire address range of the system, as the user can define different memory update policies for the user and supervisor modes and pages or segments, depending on the logical - physical address translation mechanism selected.

- The effectiveness of the cache is a function of the cache hit rate, which depends on the locality of the memory accesses. A cache inhibit may result in a very high time overhead, if the memory accesses are overwhelmingly local.
- Writethrough memory update policy increases the time, as simultaneously with a cache write a memory access is generated as well. Usually the majority of the memory accesses is of the read type, accordingly the expected time overhead is moderate.
- However, if the memory access rate is increased above a limit, than the total computing time may superseede the simple sum of the partial overhead times due to bus saturation..

3.1.1. Observability vs. speed

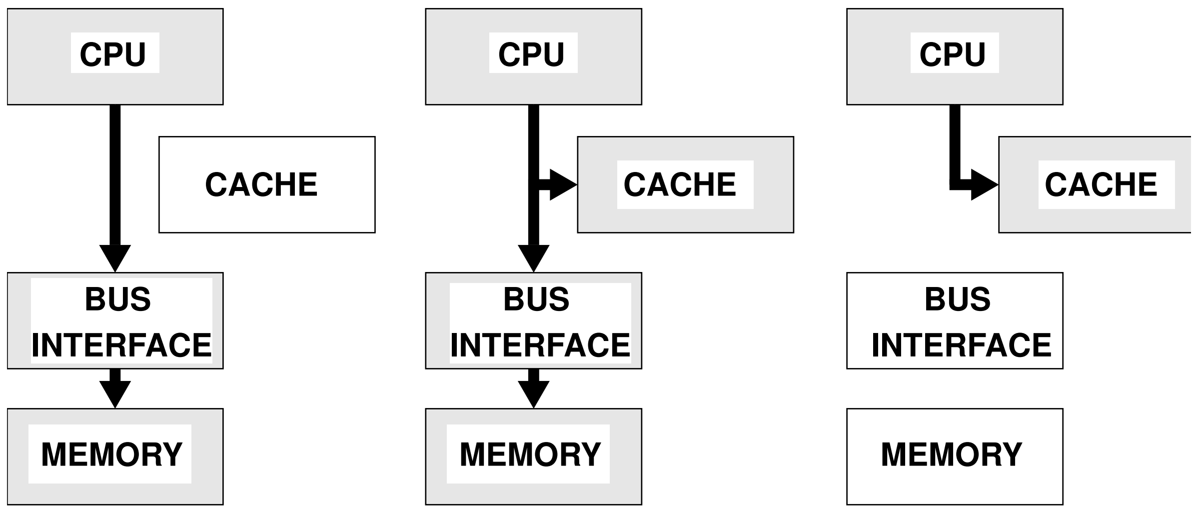
Cache utilization is rather contradicting from the point of view of master-checjker based fault tolerance:

- CPU-cache communication requires at most only a few components, assuring a higher reliability, than that on the system bus.
- No other direct fault detecting mechanism exists for the processor-MMU circuitry, as the master-checker mode, requiring regular sytem bus cycles for a proper observability.
- Accordingly, the crucial point for an effective utilization of master-checker mode is the selection of a proper cache-memory update policy, as a proper compromise between com-



NO CACHE CACHE UPDATE OF THE CACHE

Read operation



NO CACHE WRITETHOROUGH COPYBACK

Fig. 8. Write operation

puting speed and fault tolerance. The main design goal is a simultaneous good observability (sufficiently frequent system bus cycles) and a reasonable low overhead in computing time by "hidden" observations, overlapping with *the normal operation*.

Two potential approaches exist to this problem:

- A *global policy,* providing an application independent solution by the operation system
- An *application-dependent* policy encountering the features of the user program in the selection of the cache handling policy as well.

The characteristics of instruction and data caches differ essentially:

• The *instruction cache* is only read by the CPU and bus transfer are generated only in the case of a cache miss, initiating the loading of a new program segment from the memory into the cache. The cache miss frequency is very low, moreover unpredicatable due to the dependcy on the dynamic behaviour of the program. Correspondingly, p*ractically neither di-*

rect observability of the instruction cache, nor a proper the control flow monitoring by master checker mode *is assured*. A potential way for monitoring control flow is the check of some program-flow related signatures, in the data domain. The simplest way for this is the check of the subroutine call addresses on the stack.

- Data cache is of read-write type, accordingly data are checked on the system bus. The selection of the *memory update policy* is rather tedious due to the additional dependence on the read/write ratio of memory data:
  - C*opyback* can result in a too *long error latency, so* this mode is preferred only for *local, frequently used variables* , with a frequent *reinitialization* (e.g. cycle variables).
  - In *writethrough mode, time overhead* results from the processor-cache-memory writes. However, this overhead is moderate for data, which are more often subject to read as to write. The increased observability of the cache data promiaes a good compromise between performance and fault tolerance.

5.3.2 Experimental results

The cumulative effects of this factors will be illutrated by the followin bechmark results, measured on a MC88k system running under UNIX:

- A solver for a Poisson partial differential equation systems using the multigrid method, a highly computing intensive program of a greet level of data locality, as the calculation of a new grid value uses only the values of the adjacent grid points. As many variables influence the new value on a grid point, the rate of data writes is low.
- A dhrystone benchmark,, widely used in the comparison of the computing power of different computers is a very computing intensive program, but with a moderate memory usage.
- A solver for the two-dimensional stationary Navier-Stokes equation, of a very low level of data access locality , as all of the system state space variables are required in the calculations.

The multigrid differential equation solution program is a typical representative of In order to examine the effect of the memory size on the computing time an experiment series was performed by using different grid granularities.

3.2.1. Global memory update policy results

- Multigrid program (Fig. 9.) using a relative scale based on the highest speed run results for differenmt grid granularities and thus memory requirements). The high locality of data accesses results in a very effective cache use with only a very few of swaps. (Typically the the actual grid point and of its neighbours were cache resident.) In the case of the lowest dimension (a grid of dimension 65×65) this is even much more dominant as the total memory required and the cache size are of the same order of magnitude. In this application many variables influence the new value on a grid point. Accordingly the rate of data writes is low and there is no significant difference between the copyback and writethrough memory update policies.
- Dhrystone (Fig. 10. .)is a small size, but very computing intensive program, accordingly a data cache inhibit results in a computing time increment factor as high as one to ten by the heavy use of a very small data space (in the case of data caching, no cache swaps are required) and a nearly 1:1 memory read-write rate.
- Navier-Stokes (Fig. 11.) uses a large data space with distributed memory accesses. A significant increase due to writethrough originates in a high write rate. As by all of the computation intensive programs instruction cache plays a crucial role in the computing time.

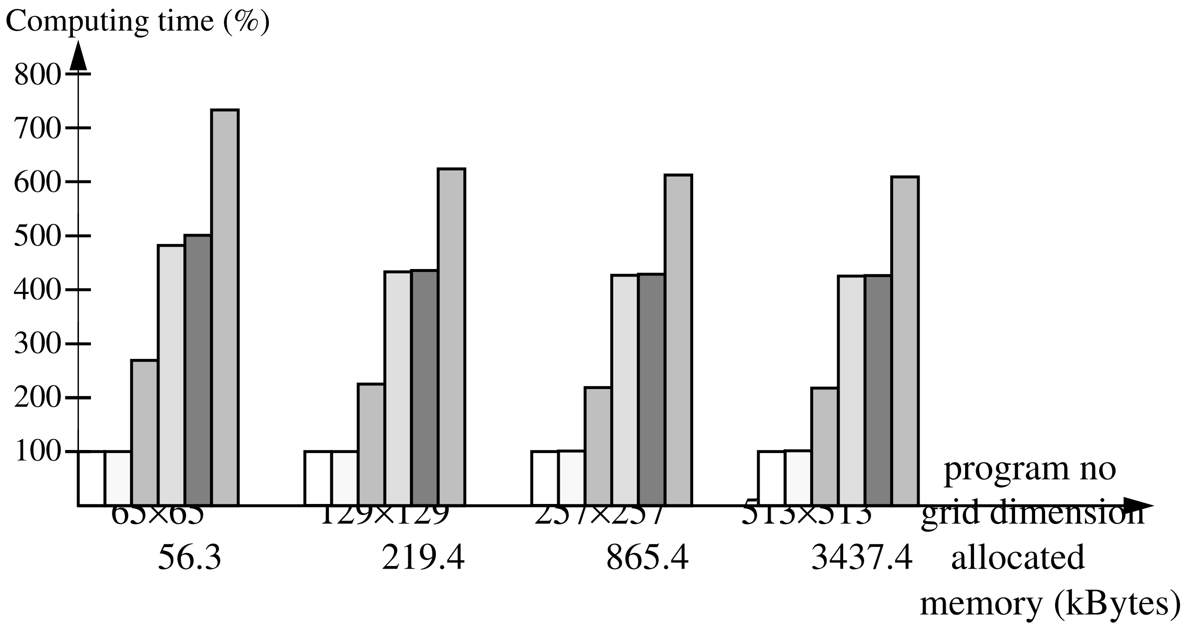


Fig. 9. Multigrid results

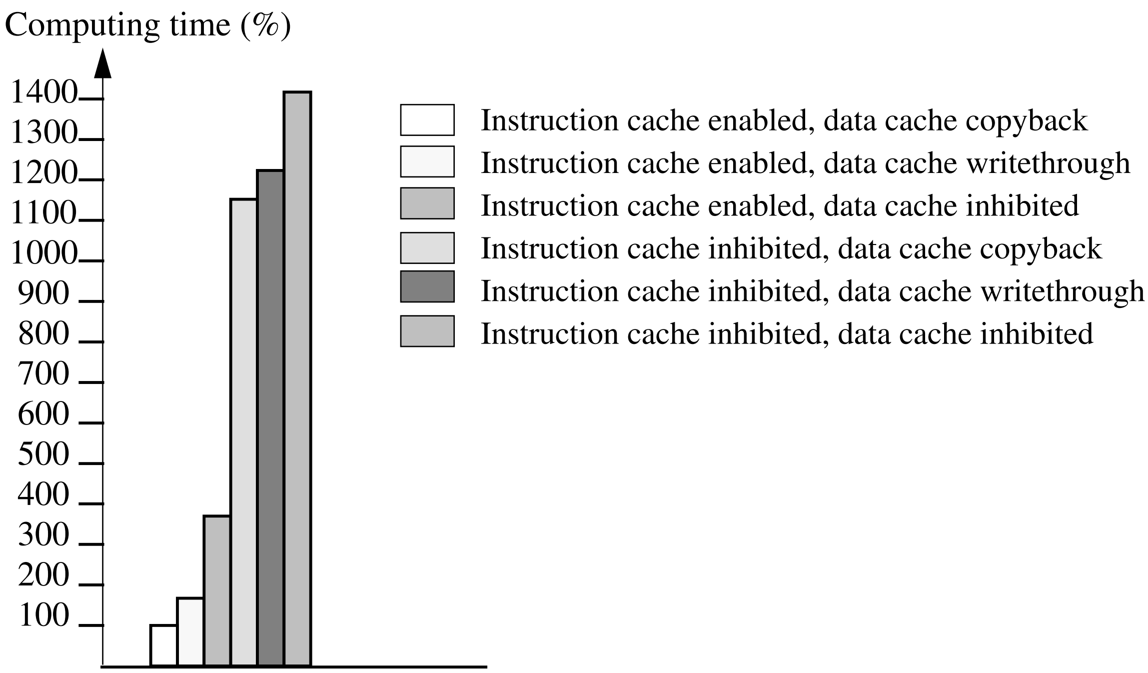


Fig. 10. Dhrystone results

As consequence, there exists no unique solution for all applications, providing a proper compromise between fault tolerance and computing power. An increased observability of the crucial data can be assured by a writethrough mode, . however , this can not be done in a global manner, (c.f. the effect of data writethrough mode in the multigrid and dhrystone program) without a significant computing speed loss for some applications.

3.2.2. Application dependent memory update policy

As a global memory update policy results in an unacceptably high loss in the computing

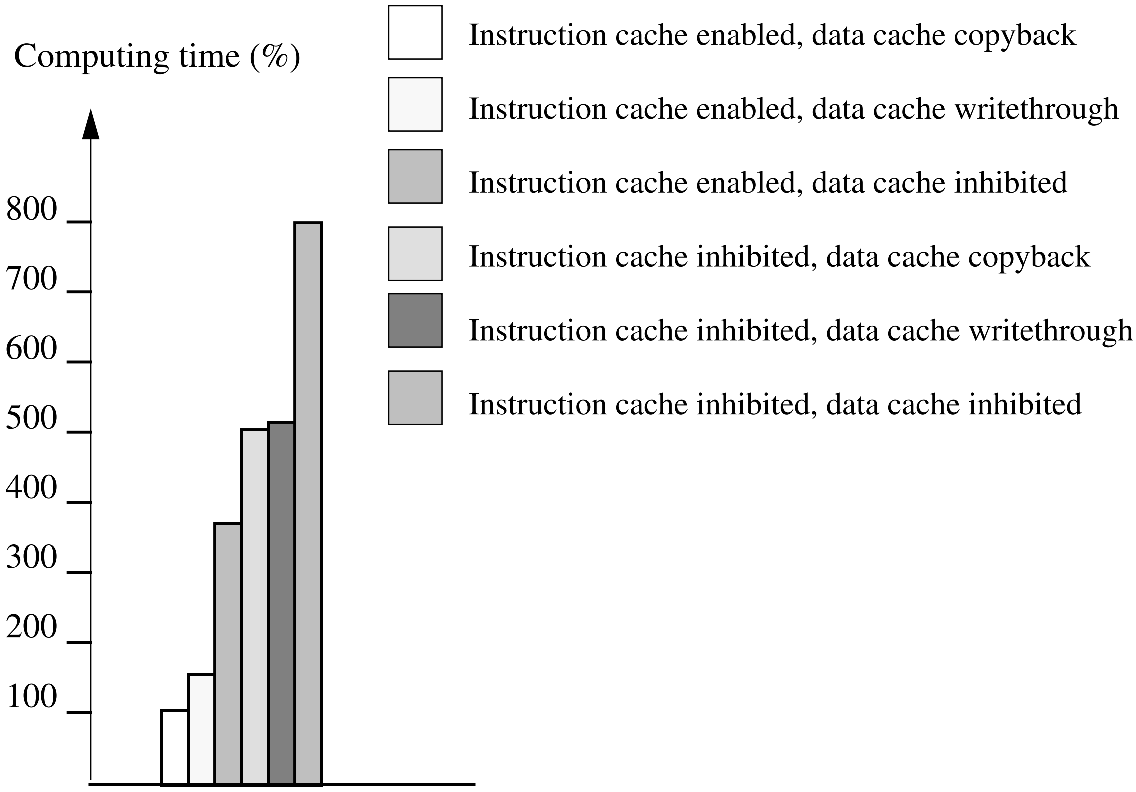


Fig. 11. Navier-Stokes results

speed or in a low observability, in an "intuitively optimal solution" the most crucial system variables are kept in the cache, using *writethrough* mode:

- The *check of data on the stack(s)* at their push time onto the stack, an implicit partial program flow check is implemented by monitoring all subroutine return addresses. Writethrough mode for the stack area increases the access time significantly, due to the nearly equal count of stack reads and writes. However, as typically only return addresses and pointers to data structures are passed between program segments, stack writes are only a small portion of all write transfers.
- Caching *crucial user data* (as checkpoint data) using *writethrough* mode can significantly improve the fault detection capability of the system, e.g. in iterative algorithms at the end of each iteration cycle the majority of errors can be detected by checking on a 20-25% of all data.
- *Local user and supervisor variables* are to be cached using the *copyback* policy (if not allocated on the stack) in order to keep the time overhead low.

5.4 Comparison of the different self-test techniques

In this section we give a short summary on the different concurrent test facilities of the system. As comparison base we will use the ESIC method [16]. , in order to compare the high level fault detection features of this methods.

5.4.1 Master-checker mode vs. watch-dog processors

Master-checker mode detects faults in the CPU and MMU of a fault coverage of by nearly 100% (Fig. 12.) and no common mode errors. area consists approximately the mezzanine module.

Fault site		Fault coverage in% Master-checker	Fault coverage in% Watch-dog	Fault coverage in% Master-checker+ watch-dog
Processor (internal)	Control flow errors	100	80	100
	Data flow errors		0	
Code cache			50-80	
Data cache	Control flow variable			
	Data flow variable		0	
External (e.g. in the main memory)	Control flow variable	0	50-80	50-80
	Data flow variable		0	0

Fig. 12. Master-checker mode and watch-dog

A watchdog detects only fatal or long transient control flow errors. As a global, user-near protection mechanism it covers a larger part of the hardware, as the master-checker mode, but the coverage of kernel errors is significantly smaller, than in master-checker mode. The main insufficiency of a watch-dog is the lack of data checks. On the other hand, as watch-dog processors use an independent hardware, the probability of correlated errors in the kernel and in the watch-dog is very small. As the watch-dog uses even independent reference for the checks, this method is much more insensitive against external faults.

4.1.1. Simultaneous use of the master-checker mode and of the watch-dog processor

The benefits from the simultaneous use of the master-checker mode and the watch-dog processor exceed the simple superposition of the advantages in the different methods. Naturally, if for some class of errors only a single method assures protection, the resulting fault detection will be determined by its properties. However in the case of data dependent control flow errors in a single processor we can observe a benign interference for errors resulting in a syntactically correct, but semantically faulty signature stream. This cathegory of errors (e.g. false selection in a conditional branch instruction) is undetected by the watch-dog processor, but master-checker will detect it.

4.1.2. Simulated watch-dog

It is possible to implement a “pseudo watch-dog” by the master-checker mode. In the simplest form the same labelling of the program graph can be used, as in an assigned signature method. During program run the signatures will be stored in a dummy external memory location, emulating the input interface register of a WDP. As signatures are stored by means of an bus write operation, their match in the master and in the checker processor will be immediately checked, accordingly any non-catastrophic control flow error affecting a single processor only will be detected with a latency of at most of a single signature interval.

The detection of common mode errors can be improved in a multiprocessor configuration by starting a watch-dog process on some other processor.

References

[1]. MC88100 RISC Microprocessor User's Manual (Second edition)
Prentice-Hall, Englewood Cliffs, 1990.

[2]. MC88200 Cache/Memory Management Unit User's Manual (Second edition)
Prentice-Hall, Englewood Cliffs, 1990.

[3]. HM88K HYPER Module 32-Bit RISC Processor Mezzanine Module User's Manual (HM88KUM/D1), Motorola, Jan. 1991.

[4]. MVME188 VMEmodule RISC Microcomputer Support Information (SIMVME188/D1), Motorola, Apr. 1990.

[5]. Product letter MC88XXX/E3
Motorola, May 10, 1991.

[6]. 188bug Debugging Package User's Manual (MVME188BUG/D1)
Motorola, March 1990.

[7]. Rabenstein:, A.
Fehlererkennung mit dem Master/Checker-Verfahren- Implementierung und Bewertung beim Motorola 88000 System
Diploma Thesis A the Institute for Mathematical Machines and Data Processing (III), Friedrich-Alexander University Erlangen-Nuremberg, 1992.

[8]. Rabenstein:, A.
Anhang zur Diplomarbeit:
Fehlererkennung mit dem Master/Checker-Verfahren- Implementierung und Bewertung beim Motorola 88000 System
Diploma Thesis at the Institute for Mathematical Machines and Data Processing (III), Friedrich-Alexander University Erlangen-Nuremberg, 1992.

[9]. Namjoo, M.; McCluskey, E. J.:
Watchdog Processors and Capability Checking, Proc. 12th FTCS, pp. 245-248, 1982

[10]. SYSTEM V/88. Release 3.2 User's Reference Manual (68NW9209H41A)
Motorola, Feb. 1990.

[11]. MVME374 Multi-Protocol Ethernet Interface Module User's Manual (MVME374/D1)
Motorola, Jul. 1988.

[12]. 374bug Debugging Package User's Manual (MVME374BUG/D1)
Motorola, March 1990.

[13]. MVME327A Firmware User's Manual (MVME327AFW/D2)
Motorola, Jan. 1991.

[14]. MVME374 Firmware User's Manual (MVME374FW/D2)
Motorola, Jan. 1991.

[15]. E. Michel- W. Hohl:
Concurrent Error Detection Using Watchdog Processors in the Multiprocessor System MEMSY", Proc. 5th Intl. Conf. Fault-Tolerant Computing Systems, Nuremberg, Informatik Fachberichte 283, pp. 54-64, Springer, September 1991

[16]. Michel, E.:
Fehlererkennung mit Überwachungsrechner in Multiprozessorsystemen;
Dissertation, Friedrich-Alexander university Erlangen-Nuremberg, 1992