Event monitoring in highly complex hardware systems

by T. Buechner
R. Fritz
P. Guenther
M. Helms
K. D. Lamb
M. Loew
T. Schlipf
M. H. Walz

This paper presents a novel approach for monitoring disjunct, concurrent operations in heavily queued systems. A nonobtrusive activity monitor is used as an on-chip tracing unit. For each pending operation the monitor uses the hardware implementation of an event-triggered operation graph to trace the path of the operation through the system. In contrast to conventional tracing units, which collect and record information from one or more functional units for later analysis, the presented solution directly records the path taken by the operation through the system, making possible an immediate analysis of operation inconsistencies. For each followed path a unique signature is generated which significantly reduces the amount of trace data to be stored. The trace information is stored together with a time stamp for debugging and measuring of queueing effects and timing behavior in the system. The method presented has been successfully applied to the memory-bus adapter chips in the S/390® G5 and G6 systems.

Introduction

With the high level of integration in recent computer systems, it has become necessary to integrate hardware debugging functions within the system. While development debugging is typically performed by simulation and formal verification [1], error analysis on real hardware is possible only by using trace information that is generated by the hardware itself.

A system can be divided into different functional units. Common tracing units, such as the one described in [2] (further called *location-centric*; see **Figure 1**), use a fixed selection of signals from the different functional units to

- Trace off-chip interfaces.
- Trace interfaces between functional units.
- Record commands and data passing through a certain unit by copying them into on-chip trace arrays.

The characteristics of such a location-centric approach are as follows:

- Histories are short because of trace array limitations.
- The view of a recorded trace is limited to a specific functional unit (isolated view).

0018-8646/99/$5.00 © 1999 IBM

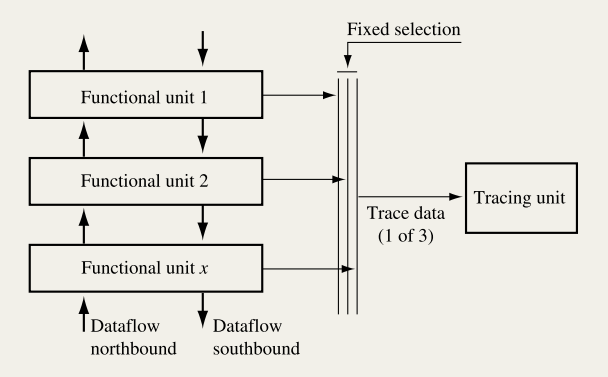

Figure 1

Location-centric tracing unit.

- Tracing and error checking are independent of each other.

Today's systems have evolved to include many queues and buffers in the dataflow in order to allow multiple simultaneous operations. The control logic of this dataflow is quite complex, which makes debugging the behavior of such a system very difficult. Location-centric monitoring is no longer sufficient for monitoring and analyzing the behavior of such a system. The amount of data to be recorded is too great, and the nonlinear growth of time dependencies between operations makes it very difficult to analyze queueing effects.

Approach

To overcome this problem, it is necessary to trace and directly record the path of an *operation* on its way through the system. This implies a transition from a location-centric approach to an *operation-centric* approach. In such an approach, the tracing unit has its focus on operations, not just on a single piece of logic or functional unit. The path of an operation inside a system is tracked and saved to an array. This method of tracing is independent of the boundaries of functional units, and may even reach across chip boundaries.

- *An easy real-life example*

Consider the traffic through a city with many traffic points, such as intersections and parking garages, which can be regarded as equivalent to functional units. Let there be different kinds of vehicles (operations) traveling through the city. In this example, the vehicle types might include motorcycles, passenger cars, vans, and trucks. The maximum number of vehicles in the city is limited. The goal is to trace the traffic through the city in such a way that the reason for a traffic jam or accident can be determined after it has occurred.

In a location-centric approach, there would be a camera installed at each traffic point, transmitting the complete real-time picture to a central traffic-monitoring station (tracing unit). The station would have a single tape recorder (VCR) to selectively record the pictures from one of the cameras. Even if the VCR input were switched frequently among the different cameras, it is obvious that later analysis of the whole traffic system from these widely scattered snapshots would be a difficult task. An alternative would be to install a VCR at each traffic point, but this solution is expensive, and the analysis and reconstruction of dependencies among the different recordings would be very complicated.

With operation-centric monitoring, at each traffic point a camera would take a snapshot of all license plates and transmit a picture from the vehicle and the license number to the traffic monitoring station. Here, a separate city map would be reserved for each vehicle that enters the city. Each vehicle would receive its own supervisor, who would know the legal paths for the particular type of vehicle and monitor its passing through the different traffic points. When a vehicle passed a traffic point, the supervisor would check whether the vehicle was allowed to be there at that time (for example, if it used an illegal "short cut," it would leave out intersections or reach them in the wrong sequence), and would mark the traffic point on the city map. When the vehicle reached the final traffic point on its way or if it committed a traffic violation, the marked path from the city map would be stored in the central archives, and the supervisor would be available to monitor the next vehicle to arrive.

Note that in the second approach the amount of information that must be stored is much smaller. If only a finite number of possible paths exist, and if information about the times at which vehicles pass traffic points is not needed, it is even possible to record the path information in a compressed format by assigning a unique code to each path.

- *Application to the system*

On entering the observed area of the system, each operation receives a unique operation identifier (OID), which it retains for its entire journey through all functional units. Each functional unit generates events for a specific operation (one event for each OID). A central tracing unit collects the events from the functional units and generates an overall trace of the operation. Only this central unit has to save information to an array.

The operation flow inside a system is monitored by a dedicated unit which has knowledge about the subtasks and their sequence for each operation that can pass through the system.

• *Operation graph-based-tracing*

An operation is typically composed of several subtasks, which are executed sequentially. Therefore, the sequence can be represented as a directed graph (operation graph). **Figure 2** shows an example of operation-graph-based tracing. Consider a system allowing up to n independent operations. Each operation has a unique operation identifier $OID = r$, $r = (0 \cdots n - 1)$. In every functional unit FU_p, $p = (1 \cdots m)$, there is event-generation logic that reports the completion of a subtask T_v, $v = (1 \cdots s)$, of an operation OP_r to an operation graph OG_r. Every operation has its own dedicated operation graph; for example, FU_1 reports an event "T_1 completed for $OID = 2$" to OG_2.

• *Hardware implementation*

The operation graph is implemented as a finite-state machine (OGFSM), as shown in **Figure 3**. Note that one single operation graph may describe the behavior of different types of operations. For each type of operation, a sequence of events can be defined that corresponds to the completion or initiation of a subtask.

The operation graph shown in Figure 3 permits the monitoring of three different types of operation:

1. The sequence
   IDLE → State 0 → State 3 → IDLE
   corresponds to operation type 1, with the subtasks
   event A from FU_1,
   event C from FU_2,
   event G from FU_3.
2. The sequence
   IDLE → State 0 → State 1 → State 3 → IDLE
   corresponds to operation type 2, with the subtasks
   event A from FU_1,
   event B from FU_2,
   event D from FU_4,
   event G from FU_3.
3. The sequence
   IDLE → State 0 → State 1 → State 2 → State 3 → IDLE
   corresponds to operation type 3, with the subtasks
   event A from FU_1,
   event B from FU_2,
   event E from FU_3,
   event F from FU_3,
   event G from FU_3.

• *Implicit error checking*

For the different operation types, only a subset of all possible events is allowed (e.g., for operation type 1 the events B, D, E, and F are not allowed). For a specific operation type, an event is allowed only in one specific state. The operation-graph finite-state machine is designed in such a way that every illegal event causes the state machine to branch to an error state. Possible error indications could be as follows:

- Operation type 1 is in state 0, but event G occurs before event C.
- For operation type 2 there is an event F, but operation is not expected to reach state 2.
- Event B occurs twice for operation type 3 (from the time at which IDLE is left until the time at which IDLE is reached again).

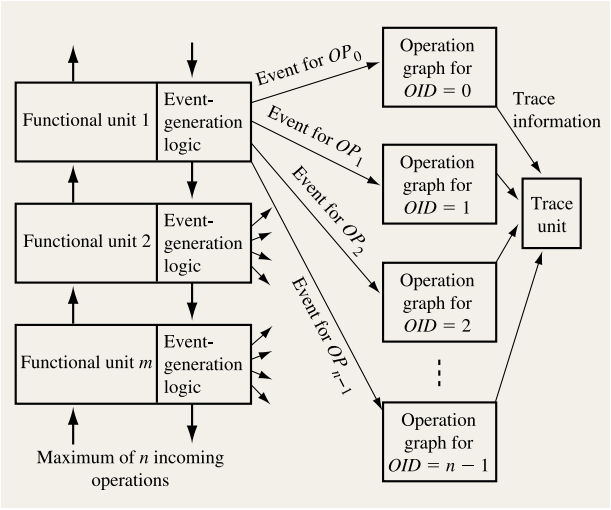

Figure 2

Operation-graph-based tracing.

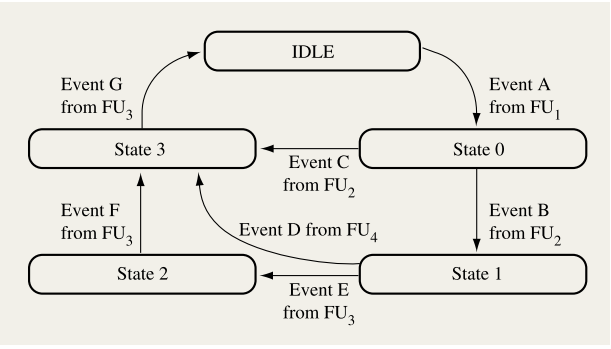

Figure 3

The operation-graph finite-state machine (OGFSM).

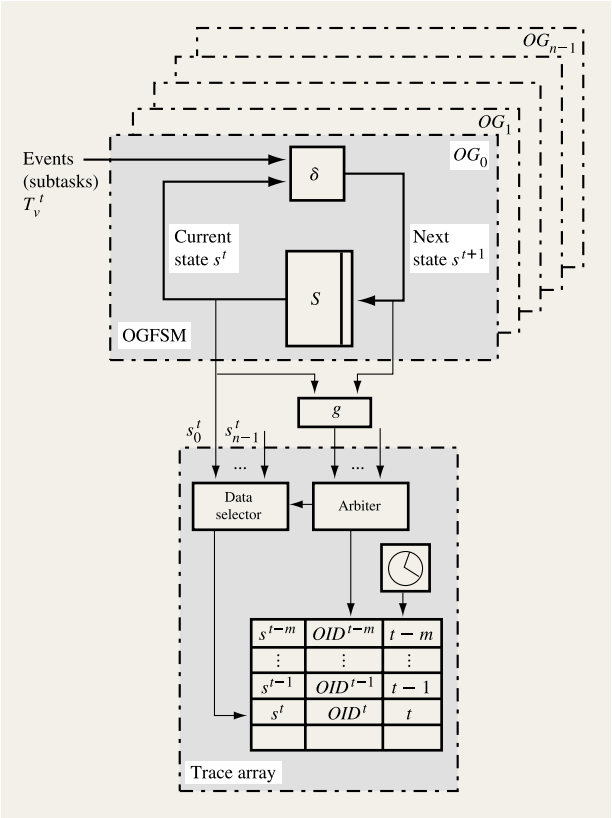

Figure 4

Interference tracing with operation-graph finite-state machines.

- *Interference tracing*

For the detection of interference problems, the tracing unit can be switched to a mode in which every state change in one of the operation-graph finite-state machines generates an entry in a trace array (**Figure 4**).

Each entry is marked with a time stamp which allows the measurement of temporal dependencies among the subtasks of several operations. The entries in the trace array contain the following information:

OGFSM for operation 3 in state 3 at time t;
OGFSM for operation 1 in state 0 at time $t + 5$ cycles;
OGFSM for operation 3 in state IDLE at time $t + 6$ cycles.

- *Trace data compression*

To minimize the cost of additional hardware for buffering and storing the trace data, the number of trace data entries must be kept as low as possible. One way to reduce the number of entries has been realized by recording only state changes in the OGFSMs. For a further reduction, the monitoring system can be used in a mode in which a data-compression mechanism (**Figure 5**) is used. In this mode, only one entry is generated for each operation. More detailed information about the operation can be derived later from the trace entry itself, which contains in a compressed format the complete path taken by the operation.

To each OGFSM a multiple-input signature register (MISR) is added which generates a signature from the state codes s^t by polynomial division. The signature is saved to the trace array under the following rules:

- The MISR is initialized when the OGFSM is in the IDLE state.
- The MISR is updated with every state change in the OGFSM.
- A branch back to IDLE causes a "save-to-trace-array" operation.
- A branch to the error state causes a "save-to-trace-array" operation.

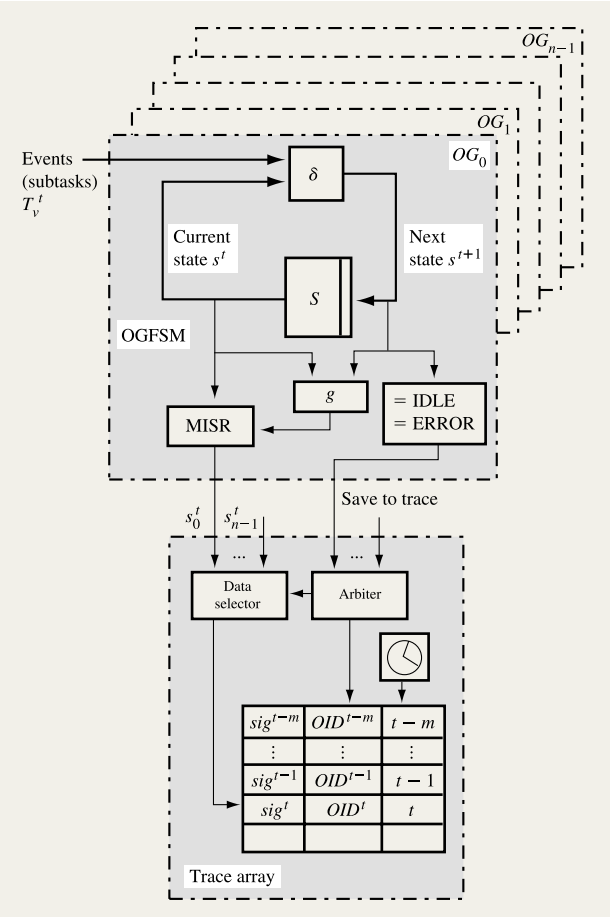

Figure 5

Data compression.

Signature-monitoring schemes for FSMs are used primarily to detect permanent and transient faults that lead to sequencing errors. To achieve this, the signatures obtained by polynomial division of the state codes with the selected polynomial must be identical for all states having the same successor [3]. In the approach presented here, the opposite method of generating the signature is used: The state assignment for the OGFSM and the polynomial are chosen in such a way that each existing sequence starting from the IDLE state has its own unambiguous signature. If loops are avoided within these sequences, a finite number of possible signatures exist that describe exactly the path taken by the OGFSM.

Example

Consider the operation-graph FSM shown in **Figure 6**. It consists of an IDLE state, the 13 event states SPLC, SNSC, UPDT, DCHK, CBSY, CCHK, CMDT, REQS, RQL2, L2GR, L2CX, and L2DX, and the error state ERR1.

Let there be two types of errors: unexpected events that force the OGFSM into the error state (type 1), and errors causing the OGFSM to stay in some given state because the expected event in that state does not happen (type 2). This leads to 197 possible sequences through the OGFSM, as shown in **Table 1**.

If we choose the four-bit OGFSM state encoding shown in Figure 6 and divide the state value by using a MISR with the polynomial $1 + x^3 + x^{10}$ (**Figure 7**), we obtain the 197 unambiguous signatures listed in Table 1.

Experimental results

The approach described above has been successfully applied to the memory bus adapter (MBA) chip in the IBM S/390* G5 and G6 systems. The MBA can be basically described as a bidirectional router for commands and data traveling between the processing units or memory and the I/O subsystem. Historically, the MBA was attached to the bus between the PU and the memory. In the present systems (see **Figure 8**) up to four MBAs are connected to the twelve PUs and four memory cards by two system controllers (SCs). The I/O subsystem is connected by up to 24 self-timed interfaces (STIs), six for each MBA. Three basic types of operation are executed by the MBA:

- *Storage operations*
  Storage operations are initiated by the channel, Integrated Cluster Bus (ICB) [4], etc., and transfer data from the channel to the memory (store operations), or from the memory to the channel (fetch operations).
- *PU sense/control operations*
  These operations are initiated by the PU and modify or read register contents in the MBA and from all chips in the I/O subsystem. For I/O, the MBA passes the operations on to the I/O subsystem and receives their responses.

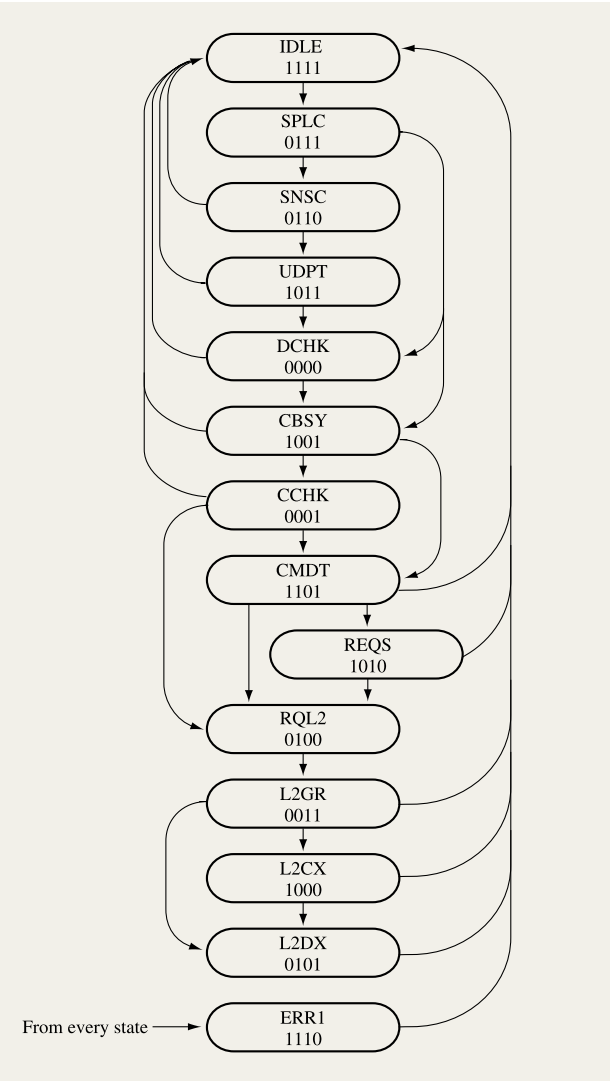

Figure 6

Operation-graph FSM.

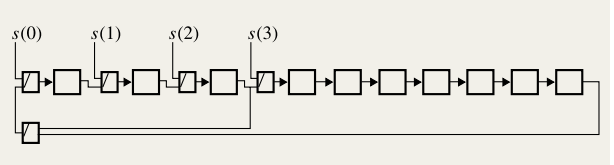

Figure 7

Ten-bit MISR.

Table 1 Possible sequences and their MISR values.

Sequence number	*MISR value*	*State sequence*	*Error type*
0	275	SPLC IDLE	
1	235	SPLC ERR1	1
2	1DA	SPLC SNSC IDLE	
3	035	SPLC SNSC	2
4	19A	SPLC SNSC ERR1	1
5	1AD	SPLC SNSC UPDT IDLE	
6	0DA	SPLC SNSC UPDT	2
7	1ED	SPLC SNSC UPDT ERR1	1
8	31A	SPLC DCHK IDLE	
9	1B5	SPLC DCHK	2
10	35A	SPLC DCHK ERR1	1
11	08D	SPLC DCHK CBSY IDLE	
12	29A	SPLC DCHK CBSY	2
13	0CD	SPLC DCHK CBSY ERR1	1
14	046	SPLC DCHK CBSY CCHK IDLE	
15	30D	SPLC DCHK CBSY CCHK	2
16	006	SPLC DCHK CBSY CCHK ERR1	1
17	1A3	SPLC DCHK CBSY CCHK CMDT IDLE	
18	0C6	SPLC DCHK CBSY CCHK CMDT	2
19	1E3	SPLC DCHK CBSY CCHK CMDT ERR1	1
20	3B1	SPLC DCHK CBSY CCHK CMDT REQS IDLE	
21	0E3	SPLC DCHK CBSY CCHK CMDT REQS	2
22	3F1	SPLC DCHK CBSY CCHK CMDT REQS ERR1	1
23	178	SPLC DCHK CBSY CCHK CMDT REQS RQL2 IDLE	
24	171	SPLC DCHK CBSY CCHK CMDT REQS RQL2	2
25	2FC	SPLC DCHK CBSY CCHK CMDT REQS RQL2 L2GR IDLE	
.	.	.	
.	.	.	
.	.	.	
.	.	.	
.	.	.	
.	.	.	
178	0ED	SPLC CBSY CMDT REQS RQL2 L2GR L2DX IDLE	
179	25B	SPLC CBSY CMDT REQS RQL2 L2GR L2DX	2
180	0AD	SPLC CBSY CMDT REQS RQL2 L2GR L2DX ERR1	1
181	177	SPLC CBSY CMDT REQS RQL2 ERR1	1
182	0EE	SPLC CBSY CMDT RQL2 IDLE	
183	25D	SPLC CBSY CMDT RQL2	2
184	037	SPLC CBSY CMDT RQL2 L2GR IDLE	
185	3EE	SPLC CBSY CMDT RQL2 L2GR	2
186	077	SPLC CBSY CMDT RQL2 L2GR ERR1	1
187	33B	SPLC CBSY CMDT RQL2 L2GR L2CX IDLE	
188	1F7	SPLC CBSY CMDT RQL2 L2GR L2CX	2
189	31D	SPLC CBSY CMDT RQL2 L2GR L2CX L2DX IDLE	
190	1BB	SPLC CBSY CMDT RQL2 L2GR L2CX L2DX	2
191	35D	SPLC CBSY CMDT RQL2 L2GR L2CX L2DX ERR1	1
192	37B	SPLC CBSY CMDT RQL2 L2GR L2CX ERR1	1
193	29B	SPLC CBSY CMDT RQL2 L2GR L2DX IDLE	
194	2B7	SPLC CBSY CMDT RQL2 L2GR L2DX	2
195	2DB	SPLC CBSY CMDT RQL2 L2GR L2DX ERR1	1
196	0AE	SPLC CBSY CMDT RQL2 ERR1	1
197	12A	ERR1	1

- *Internal sense/control operations*
  These operations are initiated by the I/O subsystem to sense, set, or reset certain bits in the MBA.

- *On-chip hardware support*
  Figure 9 shows a block diagram of the MBA chip. Operations from the I/O subsystem or the Integrated

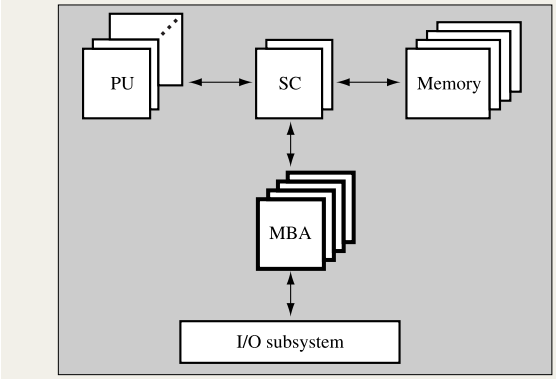

Figure 8

MBA in an S/390 system.

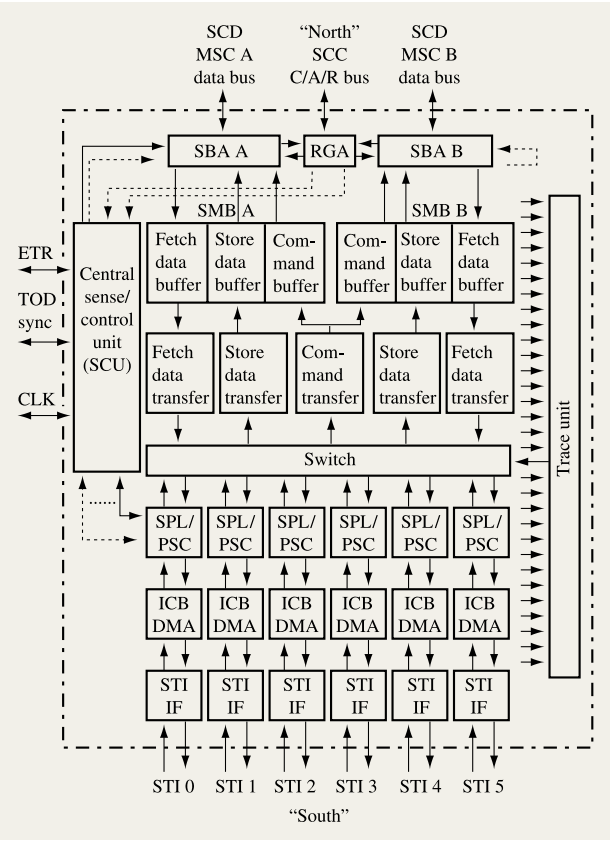

Figure 9

Block diagram of MBA chip.

Cluster Bus (ICB) are received at the STI interfaces and pass the ICB/DMA logic and the STI path logic (SPL). For each STI there may be four operations outstanding at the same time, which means that there are $6 \times 4 = 24$ possible concurrent operations on the MBA. A switch routes the operations from the STIs to a speed-matching buffer, where up to 16 operations can be stored, grouped separately as commands or data. Two storage bus adapters (SBAs) and a request/grant allocation unit (RGA) form the interface between the speed-matching buffer (SMB) and the SC chips. A central sense/control unit (SCU) handles all sense/control operations.

The OGFSMs reside in the tracing unit and track all operations coming from the STI interfaces. These are all storage operations, including the related "northbound" and "southbound" data transfers and the responses to PU sense/control operations returning from the I/O subsystem. Each OGFSM is responsible for one distinct operation outstanding on the MBA. The OGFSM itself is divided into two parts, one for the northbound traffic (master), and one for the southbound traffic (slave). The master part has already been shown in Figure 6.

For northbound operations (store, PU sense/control responses), the monitoring begins when a new operation arrives at the STI interface, and ends when the operation leaves the MBA at the MBA/SC interface (store, fetch, PU sense/control responses), or when the central sense/control logic has received the operation (internal sense/control).

The slave part of the OGFSM looks similar. It begins when the master part returns to the idle state. Responses to store and fetch operations are monitored from their arrival at the MBA/SC interface until all data and status information has been sent to the I/O subsystem by the STI interface. For internal sense/control operations, the path of the returned data and status is traced.

As Figure 9 shows, a large number of functional units must be monitored. **Table 2** shows the events reported by the event-generation logic of the different units. There can be up to four operations outstanding at each of the six STI interfaces. On the basis of these 24 outstanding operations, the total number of events is calculated as follows:

SBA:	24 * 6 * 2 =	288
SCU:	24 * 7 =	168
SPL:	24 * 13 =	312
SMB:	24 * 4 * 2 =	192
Switch:	24 * 3 =	72
Total:		1032

The 1032 events are monitored by 24 independent OGFSMs. For each OGFSM, there is one compression

Table 2 Trace events from the functional units.

Source	*Number*	*Event*
SBA	6	SBA raises req to SC SBA receives grant from SC SBA has sent the command SBA has transferred the store data SBA receives response from SC SBA sends status to SPL
SCU	7	SCU decodes I_CTRL/R_CTRL SCU decodes I_SNS/R_GSNS SCU Acknowledgment for vector update SCU sense data to SPL SCU DSNS response cross-check successful SCU DSNS response cross-check unsuccessful SCU received a DMA path reset
SPL	13	SPL receives a command SPL CMD_req to switch SPL DSNS-rsp-valid to SCU SPL command for SCU SPL header sent from response array to STI SPL command abort by FIB SPL F_ACK to switch SPL has data request (store, DSNS) SPL has activated HS_ERR SPL has sent out the data of an IP (fetch) SPL has sent a status to ISC_DMA SPL deletes the response array SPL did HS_PURGE
SMB (one side)	4	set_slot_busy Request to SBA set_slot_empty SMB fetch data available
Switch	3	Bad early status Good early status Switch has request for fetch data

unit that generates a unique signature for the path the operation has followed.

The tracing unit has its own 1KB on-chip trace array. It can be configured such that the buffer is cyclically overwritten, stored to memory after the tracing stops (e.g., due to an error), stored to memory after the trace data has reached the size of a memory line, or stored to memory by microcode activation.

To transfer the data from the on-chip-trace array into memory, the trace unit behaves like an additional SPL unit, as shown in Figure 9. This allows the use of existing hardware for tracing and limits the hardware overhead.

- *Software support*

A software program has been developed that eases the analysis of the on-chip trace buffer content, or the trace data stored to memory. The program translates the saved signatures back to a readable description of the operation paths. An example of the program output is shown in **Figure 10**.

Conclusions

In this paper, a novel approach has been presented for the monitoring and tracing of concurrent operations in heavily queued systems. The major advantages of this approach are as follows:

- The whole (sub)system is traced, not just some units on chip boundaries.
- The unique operation ID allows the entire course of an operation to be traced, even across chip boundaries.
- For concurrent operations, the timing behavior is traced (e.g., for analysis of overtakes, hangs, etc.).
- The operation graph is specified as a finite-state machine, with the possibility of real-time error checking.

```
* ---->  start for port  0 <----  operation graph for STI0 operations
   trace was stopped at: 6AEC59A 0D5D8B34 (second value compares to
absolute TOD)
* ---->  end for port  0 <----
* ---->  start for port  1 <----  operation graph for STI1 operations
   trace was stopped at: 6AEC59A 0D5D8B34 (second value compares to
absolute TOD)
* ---->  end for port  1 <----
* ---->  start for port  2 <----  operation graph for STI2 operations
   trace was stopped at: 6AEC59A 0D5D8B34 (second value compares to
absolute TOD)
* ---->  end for port  2 <----
* ---->  start for port  3 <----  operation graph for STI4 operations
  -normal-3-0B8A45- MISR     buff_ID=0/2    M=0B8 D_SNS rsp (fast SC
grant)              SPLC DCHK CBSY CMDT REQS RQL2 L2GR L2DX IDLE
                             buff_ID=0/2    S=A45 D_SNS
response                         SPRE SPPU EPTY IDLE
  -normal-3-0B8000- MISR     buff_ID=0/2    M=0B8 D_SNS rsp (fast SC
grant)              SPLC DCHK CBSY CMDT REQS RQL2 L2GR L2DX IDLE
                             buff_ID=0/2    S=000  MISR not started
  -normal-3-0B8A45- MISR     buff_ID=0/2    M=0B8 D_SNS rsp (fast SC
grant)              SPLC DCHK CBSY CMDT REQS RQL2 L2GR L2DX IDLE
                             buff_ID=0/2    S=A45 D_SNS
response                         SPRE SPPU EPTY IDLE
  -normal-3-0B8000- MISR     buff_ID=0/2    M=0B8 D_SNS rsp (fast SC
grant)              SPLC DCHK CBSY CMDT REQS RQL2 L2GR L2DX IDLE
                             buff_ID=0/2    S=000  MISR not started
  -normal-3-0B8A45- MISR     buff_ID=0/2    M=0B8 D_SNS rsp (fast SC
grant)              SPLC DCHK CBSY CMDT REQS RQL2 L2GR L2DX IDLE
                             buff_ID=0/2    S=A45 D_SNS
response                         SPRE SPPU EPTY IDLE
  -normal-3-0B8000- MISR     buff_ID=0/2    M=0B8 D_SNS rsp (fast SC
grant)              SPLC DCHK CBSY CMDT REQS RQL2 L2GR L2DX IDLE
                             buff_ID=0/2    S=000  MISR not started
  -normal-3-0B8A45- MISR     buff_ID=0/2    M=0B8 D_SNS rsp (fast SC
grant)              SPLC DCHK CBSY CMDT REQS RQL2 L2GR L2DX IDLE
                             buff_ID=0/2    S=A45 D_SNS
response                         SPRE SPPU EPTY IDLE
  -normal-3-0B8000- MISR     buff_ID=0/2    M=0B8 D_SNS rsp (fast SC
grant)              SPLC DCHK CBSY CMDT REQS RQL2 L2GR L2DX IDLE
                             buff_ID=0/2    S=000  MISR not started
  -normal-3-0B8A45- MISR     buff_ID=0/2    M=0B8 D_SNS rsp (fast SC
grant)              SPLC DCHK CBSY CMDT REQS RQL2 L2GR L2DX IDLE
                             buff_ID=0/2    S=A45 D_SNS
response                         SPRE SPPU EPTY IDLE
   -normal-3-0B8000- MISR     buff_ID=0/2    M=0B8 D_SNS rsp (fast SC
grant)              SPLC DCHK CBSY CMDT REQS RQL2 L2GR L2DX IDLE
                             buff_ID=0/2    S=000  MISR not started
  -normal-3-0B8A45- MISR     buff_ID=0/2    M=0B8 D_SNS rsp (fast SC
grant)              SPLC DCHK CBSY CMDT REQS RQL2 L2GR L2DX IDLE
                             buff_ID=0/2    S=A45 D_SNS
response                         SPRE SPPU EPTY IDLE
  -normal-3-0B8000- MISR     buff_ID=0/2    M=0B8 D_SNS rsp (fast SC
grant)              SPLC DCHK CBSY CMDT REQS RQL2 L2GR L2DX IDLE
                             buff_ID=0/2    S=000  MISR not started
  -normal-3-0B8A45- MISR     buff_ID=0/2    M=0B8 D_SNS rsp (fast SC
grant)              SPLC DCHK CBSY CMDT REQS RQL2 L2GR L2DX IDLE
                             buff_ID=0/2    S=A45 D_SNS
response                         SPRE SPPU EPTY IDLE
  -normal-3-0B8000- MISR     buff_ID=0/2    M=0B8 D_SNS rsp (fast SC
grant)              SPLC DCHK CBSY CMDT REQS RQL2 L2GR L2DX IDLE
                             buff_ID=0/2    S=000  MISR not started
  -normal-3-0B8A45- MISR     buff_ID=0/2    M=0B8 D_SNS rsp (fast SC
grant)              SPLC DCHK CBSY CMDT REQS RQL2 L2GR L2DX IDLE
                             buff_ID=0/2    S=A45 D_SNS
response                         SPRE SPPU EPTY IDLE

  -normal-3-0B8000- MISR     buff_ID=0/2    M=0B8 D_SNS rsp (fast SC
grant)              SPLC DCHK CBSY CMDT REQS RQL2 L2GR L2DX IDLE
                             buff_ID=0/2    S=000  MISR not started
  -normal-3-0B8A45- MISR     buff_ID=0/2    M=0B8 D_SNS rsp (fast SC
grant)              SPLC DCHK CBSY CMDT REQS RQL2 L2GR L2DX IDLE
                             buff_ID=0/2    S=A45 D_SNS
response                         SPRE SPPU EPTY IDLE
    -normal-3-0B8000- MISR     buff_ID=0/2    M=0B8 D_SNS rsp (fast SC
grant)              SPLC DCHK CBSY CMDT REQS RQL2 L2GR L2DX IDLE
                             buff_ID=0/2    S=000  MISR not started
```

Figure 10

Output of the trace analysis tool.

- Only one trace array is needed; multiple arrays with redundant data are not required. The amount of traced data is kept to a minimum.
- The amount of data to be stored in a trace array is further minimized by the generation of unambiguous signatures for each possible course of an operation.

The application of this method to the MBA chip has shown that it significantly reduces the time required for hardware debugging. In particular, the analysis of complex queueing effects is much easier than with the conventional location-centric approach used in previous systems.

The time required to analyze and solve problems, whether they occur during system bringup or after the customer begins to run applications, is a critical factor in time-to-market and customer satisfaction. The approach presented here is another important step to improve performance in those areas. The method described here can be used in a wide variety of applications that use multiple queues for the transport of concurrent operations. Examples of such systems are computer I/O subsystems, field buses, ATM switches, and OSI-based network and communication devices (routers, bridges, gateways, etc.) from layer 2 to 4, etc.

Acknowledgment

The authors gratefully acknowledge all members of the MBA team and other individuals at IBM for helpful discussions.

References

1. T. Schlipf, T. Buechner, R. Fritz, M. Helms, and J. Koehl, "Formal Verification Made Easy," *IBM J. Res. Develop.* **41,** No. 4/5, 567–576 (1997).
2. G. Goldrian and H. Ulland, "Tracing of Large Amounts of Data by Using Main Memory as a Trace Buffer," *IBM Tech. Disclosure Bull.* **40,** No. 6, 47–50 (1997).
3. R. Roches, R. Leveugle, and G. Saucier, "Efficiency Comparison of Signature Monitoring Schemes for FSMs," *Proceedings of the IFIP International Conference on VLSI'95*, Chiba, Japan, 1995, pp. 705–710.
4. T. A. Gregg, K. M. Pandey, and R. K. Errickson, "The Integrated Cluster Bus for the IBM S/390 Parallel Sysplex," *IBM J. Res. Develop.* **43,** No. 5/6, 795–806 (1999, this issue).

Received October 30, 1998; accepted for publication August 20, 1999

Thomas Buechner *IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (TBuechner@de.ibm.com).* Dr. Buechner received his M.S.E.E. degree from the University of Karlsruhe, Germany, in 1988, and the Ph.D. degree in computer science from the University of Stuttgart, Germany, in 1996. In 1993 he joined the IBM development laboratories in Boeblingen, Germany, where he is currently responsible for logic design and verification of complex and high-performance ICs for the IBM S/390 CMOS Parallel Enterprise Servers. His research interests include VLSI testing, formal verification, and computer-aided design methods. From 1989 to 1993 he worked as an R&D engineer at the Custom Processors and Test Department of the Institute for Microelectronics, Stuttgart, Germany, a federally funded ASIC design/fabrication/test center. During this time he was responsible for the design of several custom processors, and he did research on cost-effective test methods for semicustom ASICs. Dr. Buechner has been a member of the IEEE since 1989; he served as a technical committee member and session chair of the IEEE International ASIC Conference in 1996 and 1997, and as a program chair in 1998 and 1999.

Rolf Fritz *IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (rfritz@de.ibm.com).* Mr. Fritz studied electrical engineering at the Fachhochschule Offenburg. In 1981, after four years with Wandel u. Goldermann developing measurement sets for telecommunications, he joined the IBM S/390 development laboratories in Boeblingen. Starting in a firmware department, he developed code for recovery and error reporting for various S/390 systems. In 1993 he joined the MBA hardware team, where he has been responsible for the design and in part for the verification of the sense and control logic.

Peter Guenther *IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (pguenthe@de.ibm.com).* Mr. Guenther joined IBM in 1965 as a Field Engineer in the branch office at Hamburg. In 1969, he joined the IBM development laboratories in Boeblingen, where he has worked in many different areas, such as Bring-Up and Test, Tester Design, HW Design, System Test, and Simulation. Since 1990, he has been a member of the MBA team, where he leads the Input/Output subsystem verification team.

Markus Helms *IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (helms@de.ibm.com).* Mr. Helms studied electrical engineering at the Berufsakademie Stuttgart, graduating in 1993. His studies were accompanied by practical training at IBM in Sindelfingen. In 1993 he joined the IBM laboratories in Boeblingen as an R&D engineer, responsible for design verification and simulation of I/O adapter chips for S/390 systems. Since 1996 he has worked on logic design; one of his responsibilities is the design of the event-monitoring unit described in this paper.

Kirk D. Lamb *IBM System/390 Division, 522 South Road, Poughkeepsie, New York 12601 (kirklamb@us.ibm.com).* Mr. Lamb received a B.A. degree in English and German, *summa cum laude*, from Creighton University in 1980, a B.S.E.E. degree from Iowa State University in 1988, *cum laude*, and an M.S. degree in computer engineering from Syracuse University in 1993. He served in the Infantry of the U.S. Army from 1980 to 1984. In 1988 he joined IBM in Kingston, New York, where he worked as a digital logic designer for vector processors, memory subsystems, and cache subsystems. In 1994 he joined IBM Germany and designed part of the I/O subsystem of the S/390 system in Boeblingen. From 1996 to 1998 he managed the hardware design development tools department in Boeblingen. In 1998 he moved to IBM in Poughkeepsie, where he worked on SP adapter card development. Since February 1999 he has managed the Scalable Parallel Interconnection Design Department. Mr. Lamb is a professional engineer, registered in the state of Virginia.

Manfred Loew *IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (loews@de.ibm.com).* Mr. Loew studied physics at the University of Frankfurt, joining the IBM S/390 development laboratories in Boeblingen in 1984. After two years in S/390 microcode tools development, he joined the S/390 I/O hardware team, where he is responsible for hardware verification.

Thomas Schlipf *IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (schlipf@de.ibm.com).* Mr. Schlipf studied electrical engineering at the University of Karlsruhe. In 1985, after working for a time at the Robert Bosch Company, he joined the IBM S/390 development laboratories in Boeblingen. Since then he has been working on the hardware design of I/O chips and now leads the MBA team. Mr. Schlipf's interests are in the areas of computer architecture and formal verification. He is a member of the IEEE.

Manfred H. Walz *IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (MHWalz@de.ibm.com).* Mr. Walz received the Dipl. Ing. degree in electrical engineering from the Berufsakademie in Stuttgart in 1979. In 1979 he joined the IBM development laboratories in Boeblingen, working on memory for the 43XX systems. From 1985 to 1995, Mr. Walz led several memory development projects. In 1996 he joined the I/O subsystem development team. He has led the MBA development for the CMOS G6 S/390 system. In addition, Mr. Walz has served as lecturer at the Berusfakademie in Stuttgart; he is a member of the IEEE.