4

Simulation and Synthesis Techniques for Soft Error-Resilient Microprocessors

Makoto Sugihara
Kyushu University
Japan

1. Introduction

A single event upset (SEU) is a change of state which is caused by a high-energy particle striking to a sensitive node in semiconductor devices. An SEU in an integrated circuit (IC) component often causes a false behavior of a computer system, or a soft error. A soft error rate (SER) is the rate at which a device or system encounters or is predicted to encounter soft errors during a certain time. An SER is often utilized as a metric for vulnerability of an IC component.

May first discovered that particles emitted from radioactive substances caused SEUs in DRAM modules (May & Wood, 1979). Occurrence of SEUs in SRAM memories is increasing and becoming more critical as technology continues to shrink (Karnik et al., 2001; Seifert et al., 2001a, 2001b). The feature size of integrated circuits has reached nanoscale and the nano-scale transistors have become more soft-error sensitive (Baumann, 2005). Soft error estimation and highly-reliable design have become of utmost concern in mission-critical systems as well as consumer products. Shivakumar et al. predicted that the SER of combinational logic would increase to be comparable to the SER of memory components in the future (Shivakumar et al., 2002). Embedding vulnerable IC components into a computer system deteriorates its reliability and should be carefully taken into account under several constraints such as performance, chip area, and power consumption. From the viewpoint of system design, accurate reliability estimation and design for reliability (DFR) are becoming critical in order that one applies reasonable DFR to vulnerable part of the computer system at an early design stage. Evaluating reliability of an entire computer system is essential rather than separately evaluating that of each component because of the following reasons.

1. A computer system consists of miscellaneous IC components such as a CPU, an SRAM module, a DRAM module, an ASIC, and so on. Each IC component has its own SER which may be entirely different from one another.
2. Depending on DFR techniques such as parity coding, the SER, access latency and chip area may be completely different among SRAM modules. A DFR technique should be chosen to satisfy the design requirement of the computer system so that one can avoid a superfluous cost rise, performance degradation, and power rise.
3. The behavior of a computer system is determined by hardware, software, and input to the system. Largely depending on a program, the behavior of the computer system varies from program to program. Some programs use large memory space and the

others do not. Furthermore, some programs efficiently use as many CPU cores of a multiprocessor system as possible and the others do not. The behavior of a computer system determines temporal and spatial usage of vulnerable components.

This chapter reviews a simulation technique for soft error vulnerability of a microprocessor system (Sugihara et al., 2006, 2007b) and a synthesis technique for a reliable microprocessor system (Sugihara et al., 2009b, 2010b).

2. Simulation technique for soft error vulnerability of microprocessors

2.1 Introduction

Recently, several techniques for estimating reliability were proposed. Fault injection techniques were discussed for microprocessors (Degalahal et al., 2004; Rebaudengo et al., 2003; Wang et al., 2004). Soft error simulation in logic circuits was also studied and developed (Tosaka, 1997, 1999, 2004a, 2004b). In contrast, the structure of memory modules is so regular and monotonous that it is comparatively easy to estimate their vulnerability because that can be calculated with the SERs obtained by field or accelerated tests. Mukherjee et al. proposed a vulnerability estimation method for microprocessors (Mukherjee et al., 2003). Their methodology estimates only vulnerability of a microprocessor whereas a computer system consists of various components such as CPUs, SRAM modules and DRAM modules. Their approach would be effective in case the vulnerability of a CPU is most dominant in a computer system. Asadi et al. proposed a vulnerability estimation method for computer systems that had L1 caches (Asadi et al., 2005). They pointed out that SRAM-based L1 caches were most vulnerable in most of current designs and gave a reliability model for computing critical SEUs in L1 caches. Their assumption is true in most of current designs and false in some designs. Vulnerability of DRAM modules would be dominant in entire vulnerability of a computer system if plain DRAM modules and ECC SRAM ones are utilized. As technology proceeds, a latch becomes more vulnerable than an SRAM memory cell (Baumann, 2005). It is important to obtain a vulnerability estimate of an entire system by considering which part of a computer system is vulnerable.

An SER for a memory module is a vulnerability measurement characterizing it rather than one reflecting its actual behavior. SERs of memory modules become pessimistic when they are embedded into computer systems. More specifically, every SEU occurring in memory modules is regarded as a critical error when memory modules are under field or accelerated tests. This implicitly assumes that every SEU on memory cells of a memory module makes a computer system faulty. Since memory modules are used spatially and temporally in computer systems, some of SEUs on the memory modules make the computer system faulty and the others not. Therefore, the soft errors in an entire computer system should be estimated in a different way from the way used for memory modules.

Accurate soft error estimation of an entire computer system is one of the themes of urgent concern. The SER is the rate at which a device or system encounters or is predicted to encounter soft errors. The SER is quite effective measurement for evaluating memory modules but not for computer systems. Accumulating SERs of all memories in a computer system causes pessimistic soft error estimation because memory cells are used spatially and temporally during program execution and some of SEUs make the computer system faulty. This chapter models soft errors at the architectural level for a computer system, which has

several memory hierarchies with it, in order that one can accurately estimate the reliability of the computer system within reasonable computation time. We define a *critical SEU* as one which is a possible cause of faulty behavior of a computer system. We also define an *SEU vulnerability factor* for a job to run on a computer system as the expected number of critical SEUs which occur during executing the job on the computer system, unlike a classical vulnerability factor such as the SER one. The architectural-level soft-error model identifies which part of memory modules is utilized temporally and spatially and which SEUs are critical to the program execution of the computer system at the cycle-accurate ISS (instruction set simulation) level. Our architectural-level soft-error model is capable of estimating the reliability of a computer system that has several memory hierarchies with it and finding which memory module is vulnerable in the computer system. Reliability estimation helps one apply reliable design techniques to vulnerable part of their design.

2.2 SEUs on a word item

Unlike memory components, the SER of a computer system varies every moment because the computer system uses memory modules spatially and temporally. Since only active part of the memory modules affects reliability of the computer system, it is essential to identify the active part of memory modules for accurately estimating the number of soft errors occurring in the computer system. A universal soft error metric other than an SER is necessary to estimate reliability of computer systems because an SER is a reliability metric suitable for components of regular and monotonous structure like memory modules but not for computer systems. In this chapter, the number of soft errors which occur during execution of a program is adopted as a soft error metric for computer systems. In computer systems, a word item is a basic element for computation in CPUs. A word item is an instruction item in an instruction memory while that is a data item in a data memory. A collective of word items is required to be processed in order to run a program. We consider the reliability to process all word items as the reliability of a computer system. The total number of SEUs which are expected to occur on all the word items is regarded as the number of SEUs of the computer system. This section discusses an estimation model for the number of soft errors on a word item. A CPU-centric computer system typically has the hierarchical structure of memory modules which includes a register file, cache memory modules, and main memory modules. The computer system at which we target has N_{mem} levels of memory modules, $M_1, M_2, \cdots, M_{N_{\mathrm{mem}}}$ in order of accessibility from/to the CPU. In the hierarchical memory system, instruction items are generally processed as follows.

1. Instruction items are generated by a compiler and loaded into a main memory. The birth time of an instruction item is the time when the instruction item is loaded into the main memory, from the viewpoint of program execution.
2. When the CPU requires an instruction item, it fetches the instruction item from the memory module closest to it. The instruction item is duplicated into all levels of memory modules which reside between the CPU and the source memory module.

Note that instruction items are basically read-only. Duplication of instruction items are unidirectionally made from a low level to a high level of a memory module. Data items in data memory are processed as follows.

1. Some data items are given as initial values of a program when the program is generated with a compiler. The birth time of such a data item is the time when the program is loaded into a main memory. The other data items are generated during execution of the program by the CPU. The birth time of the data item which is made on-line is the time when the data item is made and saved to the register file.
2. When a data item is required by a CPU, the CPU fetches it from the memory module closest to the CPU. If the write allocate policy is adopted, the data item is duplicated at all levels of memory modules which reside between the CPU and the master memory module, and otherwise it is not duplicated at the interjacent memory modules.

Note that data items are writable as well as readable. This means that data items can be copied from a high level to a low level of a memory module, and vice versa. In CPU centric computer systems, data items are utilized as constituent elements. The data items vary in lifetime and the numbers of soft errors on the data items vary from data item to data item.

Let an SER of a word item in Memory Module M_i be SER_{M_i}. When a word item w is retained during Time $time(w)$ in Memory Module M_i, the number of soft errors, $error_{M_i}(w)$, which is expected to occur on the word item, is described as follows:

$$error_{M_i}(w) = SER_{M_i} \cdot time(w). \tag{1}$$

Word item w is required to be retained during Time $retain\_time_{M_i}(w)$ in Memory Module M_i to transfer to the CPU. The number of soft errors, $error_{\text{all\_mems}}(w)$, which occur from the birth time to the time when the CPU fetches is given as

$$error_{\text{all\_mems}}(w) = \sum_i SER_{M_i} \cdot retain\_time_{M_i}(w) \tag{2}$$

where $retain\_time_{M_i}(w)$ is necessary and minimal time to transfer the word item from the master memory module to the CPU, and depends on the memory architecture. This kind of retention time is exactly obtained with cycle-accurate simulation of the computer system.

2.3 SEUs in instruction memory

Each instruction item has its own lifetime while a program runs. The lifetime of each instruction item is different from that of one another and is not necessarily equal to the execution time of a program. Generally speaking, the birth time of instruction items is the time when they are loaded into main memory, from the viewpoint of program execution. It is necessary to identify which part of retention time of an instruction item in a memory module affects reliability of the computer system. Now let us break down into the number of soft errors in an instruction item before we discuss the total number of soft errors in instruction memory. The time when a CPU fetches an instruction item of Address a for the i-th time is shown by $if(a, i)$. $if(a, 0)$ denotes the time when the instruction is loaded into the main memory. An example of several instruction fetches is shown in Fig. 1. In this figure, the boxes show that the copies of the instruction item reside in the corresponding memory modules. The labels on the boxes show when the copies of the instruction items are born. In this example, the instruction item is fetched three times by the CPU.

On the first instruction fetch for the instruction item, a copy of the instruction item exists in neither the L1 nor L2 cache memories. The instruction item resides only in the main

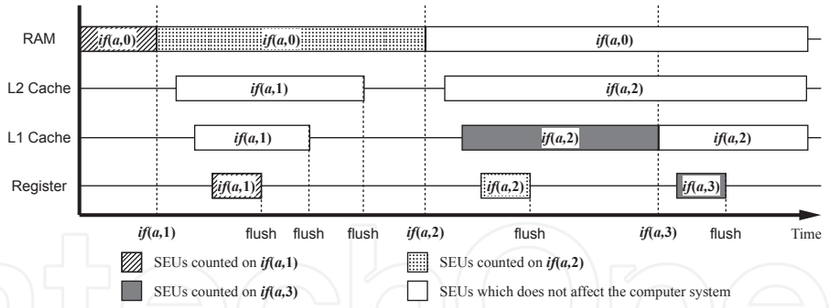


Fig. 1. SEUs which are read by the CPU.

memory. The instruction item is required to be transferred from the main memory to the CPU. On transferring the instruction item to the CPU, its copies are made in the L1 and L2 cache memory modules. In this example, we assume that some latency is necessary to transfer the instruction item between memory modules. When the instruction item in a source memory module is fetched by the CPU, any SEUs which occur after completing transferring the instruction item have no influence on the instruction fetch. In the figure, the boxes with slanting lines are the retention times whose SEUs make the instruction fetch at $if(a,1)$ faulty. The SEUs during any other retention times are unknown to make the computer system faulty.

On the second instruction fetch for the instruction item, the instruction item resides only in the main memory, same as on the first instruction fetch. The instruction item is fetched from the main memory to the CPU, same as on the first instruction fetch. The dotted boxes are found to be the retention times whose SEUs make the instruction fetch at $if(a,2)$ faulty. Note that the SEUs on the box with slanting lines in the main memory are already treated on the instruction fetch at $if(a,1)$ and are not treated on the one at $if(a,2)$ in order to avoid counting SEUs duplicately.

On the third instruction fetch for the instruction item, the highest level of memory module that retains the instruction item is the L1 cache memory. SEUs on the gray boxes are treated as the ones which make Instruction Fetch $if(a,3)$ faulty. The SEUs on any other boxes are not counted for the instruction fetch at $if(a,3)$. Now assume that a program is executed in a computer system. Given an input data to a program, let an instruction fetch sequence be $i_1, i_2, \cdots, i_{N_{\text{inst}}}$ to run the program. And let the necessary and minimal retention time for Instruction Fetch i_i to be on Memory Module M_j be $retain\_time_{M_j}(i_i)$. The number of soft errors on Instruction Fetch i_i, $error(i_i)$, is given as follows.

$$error_{\text{single\_inst}}(i_i) = \sum_j SER_{M_j} \cdot retain\_time_{M_j}(i_i). \tag{3}$$

The total number of soft errors in the computer system is shown as follows:

$$\begin{aligned} error_{\text{all\_insts}}(\boldsymbol{i}) &= \sum_i error_{\text{single\_inst}}(i_i) \\ &= \sum_{i,j} SER_{M_j} \cdot retain\_time_{M_j}(i_i) \end{aligned} \tag{4}$$

where i={ i_1,i_2,…,i_N_inst}. Given the program of the computer system, $retain\_time_{M_j}(i_i)$ can be exactly obtained by performing cycle-accurate simulation for the computer system.

2.4 SEUs in data memory

Data memory is writable as well as readable. It is more complex than instruction memory because word items are bidirectionally transferred between a high level of memory and a low level of memory. Some data items are given as an input to a program and the others are born during the program execution. Some data items are used and the others are unused even if they reside in memory modules. The SEUs which occur during some retention time of a data item are influential in a computer system. The SEUs which occur during the other retention time are not influential even if the data item is used by the CPU. A data item has valid or invalid part of time with regard to soft errors of the computer system. It is quite important to identify valid or invalid part of retention time of a data item in order to accurately estimate the number of soft errors of a computer system. In this chapter, valid retention time is sought out by using the following rules.

- A data item which is generated on compilation is born when it is loaded into main memory.
- A data item as input to a computer system is born when it is inputted to the computer system.
- A data item is born when the CPU issues a store instruction for the data item.
- A data item is valid at least until the time when the CPU loads the data item and uses it in its operation.
- A data item which a user explicitly specifies as a valid one is valid even if the CPU does not issue a load instruction for the data item.

The bidirectional copies between high-level and low-level memory modules must be taken into account in data memory because data memory is writable as well as readable. There are two basic options on cache hit when writing to the cache as follows (Hennessy & Patterson, 2002).

- *Write through*: the information is written to both the block in the cache and to the block in the lower-level memory.
- *Write back*: the information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.

The write policies affect the estimation for the number of soft errors and should be taken into account.

2.4.1 Soft error model in a write-back system

A soft-error estimation model in write-back systems is discussed in this section. Let the time when the i-th store operation of a CPU at Address a is issued be $s(a,i)$ and the time when the j-th load operation at Address a is issued be $l(a,j)$. Fig. 2 shows an example of the behavior of a write-back system. Each box in the figure shows the existence of the data item in the corresponding memory module. The labels on the boxes show when the data items are born. In the example, two store operations and two load operations are executed. First, a store operation is executed and only the L1 cache is updated with the data item. The L2 cache or main memory is not updated with the store operation. A load operation on the data item which resides at Address a follows. The data item resides in the L1 cache memory and is transferred from the L1 cache to the CPU. The SEUs on the boxes with slanting lines are

influential in reliability of the computer system by the issue of a load at $l(a,1)$. The other boxes with Label $s(a,1)$ are unknown to be influential in the reliability. Next, the data item in the L1 cache goes out to the L2 cache by the other data item. The L2 cache memory becomes the highest level of memory which retains the data item. Next, a load operation at $l(a,2)$ is issued and the data item is transferred from the L2 cache memory to the CPU. With the load operation at $l(a,2)$, the SEUs on the dotted boxes are found to be influential in reliability of the computer system. SEUs on the white boxes labeled as $s(a,2)$ are not counted on the load at $l(a,2)$.

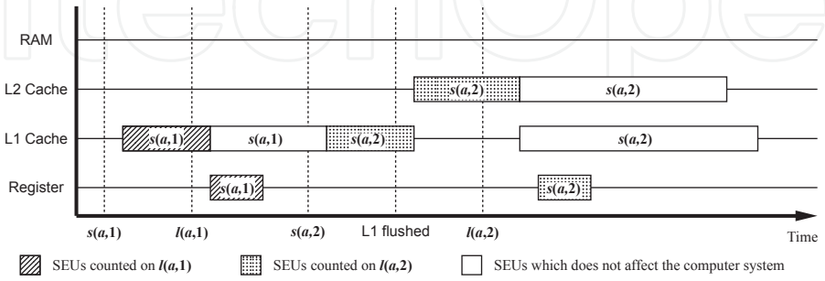


Fig. 2. Critical time in the write-back system.

2.4.2 Soft error model in a write-through system

A soft-error estimation model in write-through systems is discussed in this section. An example of the behavior of a write-through system is shown in Fig. 3. First, a store operation at Address a is issued. The write-through policy makes multiple copies of the data item in the cache memories and the main memory. Next, a load operation follows. The CPU fetches the data item from the L1 cache and SEUs on the boxes with slanting lines are found to be influential in reliability of the computer system. Next, a store operation at $s(a,2)$ comes. The previous data item at Address a is overridden and the white boxes labeled as $s(a,1)$ are no longer influential in reliability of the computer system. Next, the data item in the L1 cache is replaced with the other data item. The L2 cache becomes the highest level of memory which has the data item of Address a. Next, a load operation at $l(a,2)$ follows and the data item is transferred from the L2 cache to the CPU. With the load operation at $l(a,2)$, SEUs on the dotted boxes are found to be influential in reliability of the computer system.

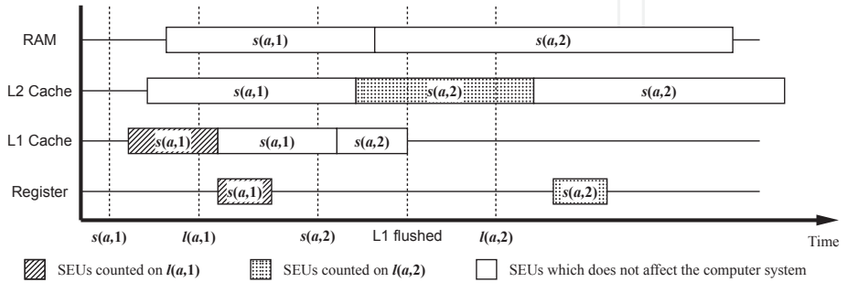


Fig. 3. Critical time in the write-through system.

2.5 Simulation-based soft error estimation

As discussed in the previous sections, the retention time of every word item in memory modules needs to be obtained so that the number of soft errors in a computer system can be estimated. We adopted a cycle-accurate ISS which can obtain the retention time of every word item. A simplified algorithm to estimate the number of soft errors for a computer system to finish a program is shown in Fig. 4. The input to the algorithm is an instruction sequence, and the output from the algorithm is the accurate number of soft errors, $error_{\text{system}}$, which occur during program execution.

First, several variables are initialized. Variable $error_{\text{system}}$ is initialized with 0. The birth times of all data items are initialized with the time when the program starts. A for-loop sentence follows. A cycle-accurate ISS is executed in the for-loop. An iteration loop corresponds to an execution of an instruction. The number of soft errors is counted for every instruction item and is accumulated to variable $error_{\text{system}}$. When variable $error_{\text{system}}$ is updated, the birth time of the corresponding word item is also updated with the present time. Some computation is additionally done when the present instruction is a store or a load operation. If the instruction is a load operation, the number of SEUs on the data item which is found to be critical in the reliability of the computer system is added to variable $error_{\text{system}}$. A load operation updates the birth time of the data item with the present time. If the instruction is a store operation, the birth time of all changed word items is updated with the present time. After the above procedure is applied to all instructions, $error_{\text{system}}$ is outputted as the number of soft errors which occur during the program execution.

Procedure EstimateSoftError

Input: Instruction sequence given by a trace.

Output: the number of soft errors for the system, $error_{\text{system}}$

begin

$error_{\text{system}}$ is initialized with 0.

Birth time of every word iterm is initialized with the beginning time.

for all instructions **do**

// Computation for soft errors in instruction memory

Add the number of critical soft errors of the instruction item to $error_{\text{system}}$.

Update the birth time on the instruction item with the present time.

// Computation for soft errors in data memory

if the current instruction is a load **then**

Fig. 4. A soft error estimation algorithm.

2.6 Experiments

Using several programs, we examined the number of soft errors during executing each of them.

2.6.1 Experimental setup

We targeted a microprocessor-based system consisting of an ARM processor (ARMv4T, 200MHz), an instruction cache module, and a data cache module, and a main memory module as shown in Fig. 5. The cache line size and the number of cache-sets are 32-byte and 32, respectively. We adopted the least recently used (LRU) policy as the cache replacement policy. We evaluated reliability of computer systems with the two write policies, write-through and write-back ones. The cell-upset rates of both SRAM and DRAM modules are shown in Table 1. We used the cell-upset rates shown in (Slayman, 2005) as the cell-upset rates of plain SRAMs and DRAMs. According to Baumann, error detection and correction (EDAC) or error correction codes (ECC) protection will provide a significant reduction in failure rates (typically 10k or more times reduction in effective error rates) (Baumann, 2005). We assumed that introducing an ECC circuit makes reliability of memory modules 10k times higher.

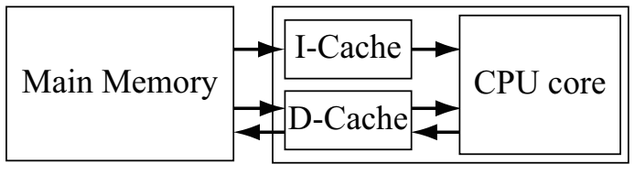


Fig. 5. The target system.

	Cell Upset Rate			
	[FIT/bit]		[errors/word/cycle]	
	w/o ECC	w. ECC	w/o ECC	w. ECC
SRAM	1.0×10^{-4}	1.0×10^{-8}	4.4×10^{-24}	4.4×10^{-28}
DRAM	1.0×10^{-8}	1.0×10^{-12}	4.4×10^{-24}	4.4×10^{-32}

Table 1. Cell upset rates for experiments.

We used three benchmark programs: Compress version 4.0 (Compress), JPEG encoder version 6b (JPEG), and MPEG2 encoder version 1.2 (MPEG2). We used the GNU C compiler and debugger to generate address traces. We chose to execute 100 million instructions in each benchmark program. This allowed the simulations to finish in a reasonable amount of time. All programs were compiled with "-O3" option. Table 2 shows the code size, activated code size, and activated data size in words for each benchmark program. The activated code and data sizes represent the number of instruction and data addresses which were accessed during the execution of 100 million instructions, respectively.

	Code size S_{code} [words]	Activated code size AS_{code} [words]	Activated data size AS_{data} [words]
Compress	10,716	1,874	140,198
JPEG	30,867	6,129	33,105
MPEG2	33,850	7,853	258,072

Table 2. Specification for benchmark programs.

2.6.2 Experimental results

Figures 6, 7, and 8 show the results of our soft error estimation method. Four different memory configurations were considered as follows:

1. non-ECC L1 cache memory and non-ECC main memory,
2. non-ECC L1 cache memory and ECC main memory,
3. ECC L1 cache memory and non-ECC main memory,
4. and ECC L1 cache memory and ECC main memory.

Note that Asadi's vulnerability estimation methodology (Asadi et al., 2005) does not cover vulnerability estimation for the second configuration above because their approach is dedicated to estimating vulnerability of L1 caches. The vertical axis presents the number of soft errors occurring during the execution of 100 million instructions. The horizontal axis presents the number of cache ways in a data cache. The other cache parameters, i.e., the line size and the number of lines in a cache way, are unchanged. The size of the data cache is, therefore, linear to the number of cache ways in this experiment. The cache sizes corresponding to the values shown on the horizontal axis are 1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB, and 64 KB, respectively.

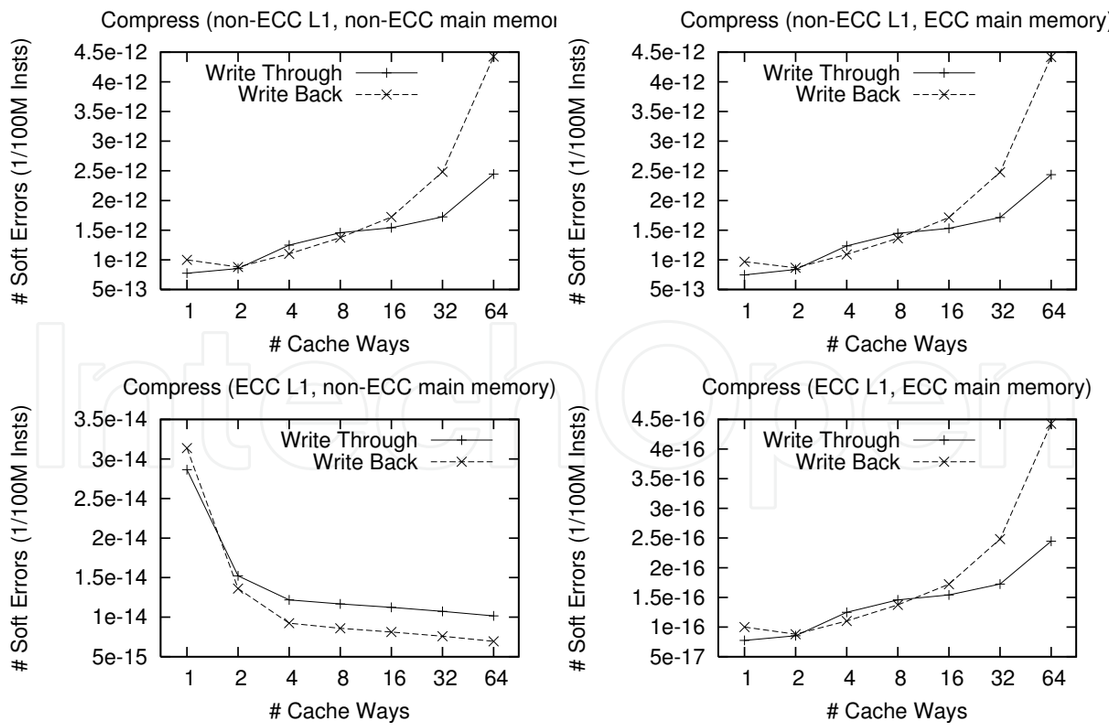


Fig. 6. Experimental results for Compress.

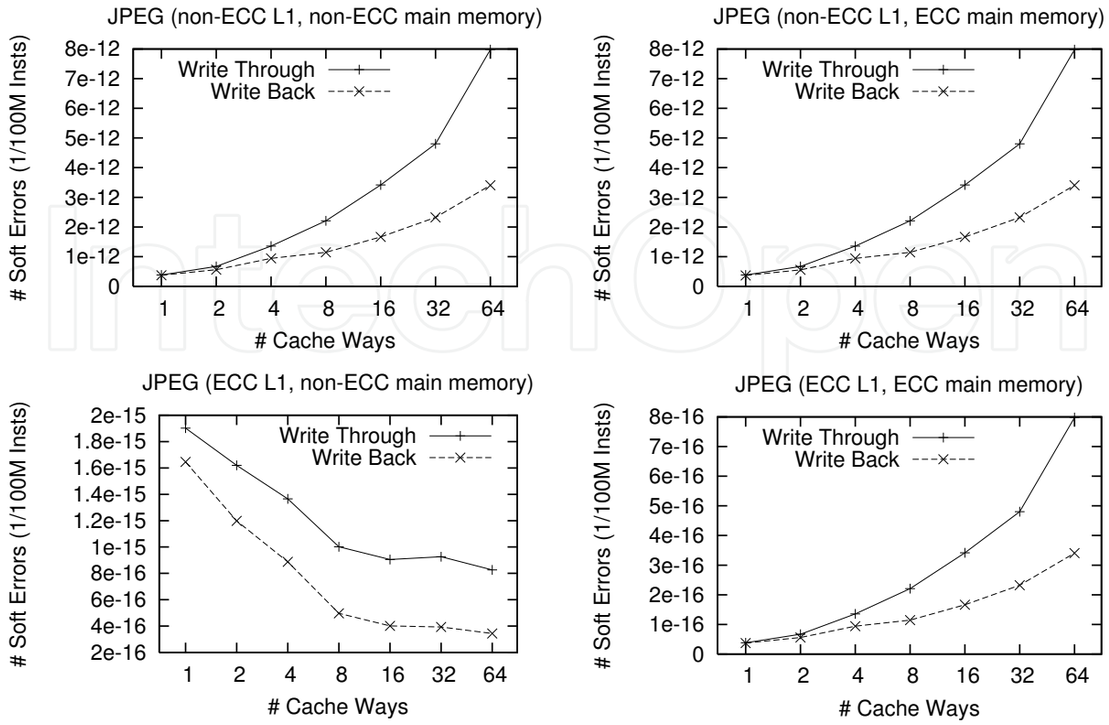


Fig. 7. Experimental results for JPEG.

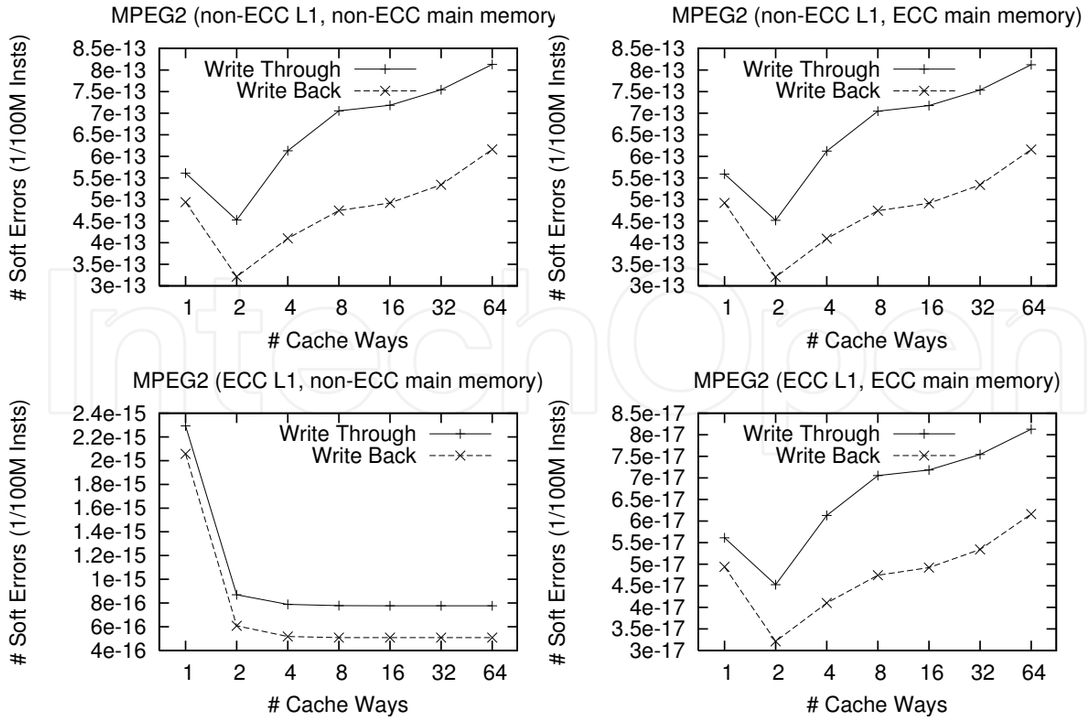


Fig. 8. Experimental results for MPEG2.

According to the experimental results shown in Figures 6, 7, and 8, the number of soft errors which occurred during a program execution depends on the reliability design of the memory hierarchy. When the cell-upset rate of SRAMs was higher than that of DRAMs, the soft errors on cache memories became dominant in the whole soft errors of the computer systems. The number of soft errors in a computer system, therefore, increased as the size of cache memories increased. When the cell-upset rate of SRAM modules was equal to that of DRAM ones, the soft errors on main memories became dominant in the system soft errors in contrast. The number of soft errors in a computer system, therefore, decreased as the size of cache memories increased because the larger size of cache memories reduced runtime of a program as well as usage of the main memory. Table 3 shows the number of CPU cycles to finish executing the 100 million instructions of each program.

		The number of cache ways in a cache memory (1 way = 1 KB)						
		1	2	4	8	16	32	64
Compress	WT	968	523	422	405	390	371	348
	WB	1,058	471	325	303	286	267	243
JPEG	WT	548	455	364	260	247	245	244
	WB	474	336	237	129	110	104	101
MPEG2	WT	497	179	168	168	167	167	167
	WB	446	124	110	110	110	110	110

Table 3. The number of CPU cycles for 100 million instructions.

Table 4 shows the results of more naive approaches and our approach. The two naive approaches, M1 and M2, calculated the number of soft errors using the following equations.

$$SE_1 = \{S_{\text{cache}} \cdot SER_{\text{S}} + (S_{\text{code}} + AS_{\text{data}}) \cdot SER_{\text{D}}\} \cdot N_{\text{cycle}} \quad (5)$$

$$SE_2 = \{S_{\text{cache}} \cdot SER_{\text{S}} + (AS_{\text{code}} + AS_{\text{data}}) \cdot SER_{\text{D}}\} \cdot N_{\text{cycle}} \quad (6)$$

where S_{cache}, S_{code}, AS_{code}, AS_{data}, N_{cycle}, SER_{S}, SER_{D} denote the cache size, the code size, the activated code size, the activated data size, the number of CPU cycles, the SER per word per cycle for SRAM, and the SER per word per cycle for DRAM, respectively. M1 and M2 appearing in Table 4 correspond to the calculations using Equations (5) and (6), respectively. Our method corresponds to M3. It is obvious that the simple summation of SERs resulted in large overestimation of soft errors. This indicates that accumulating SERs of all memory modules in a system resulted in pessimistic estimation. The universal soft error metric other than the SER is necessary to estimate reliability of computer systems which behave dynamically. The number of soft errors which occur during execution of a program would be the universal soft error metric of computer systems.

			The number of cache ways						
			1	2	4	8	16	32	64
Compress	WT	M1	2267	2417	3869	7394	14216	27068	50755
		M2	2263	2415	3867	7393	14214	27067	50754
		M3	776	852	1248	1458	1541	1724	2446
	WB	M1	2478	2175	2976	5530	10423	19461	35410
		M2	2474	2173	2975	5529	10439	19460	35410
		M3	999	881	1101	1372	1722	2484	4426
JPEG	WT	M1	1262	2083	3324	4735	9013	17867	35556
		M2	1255	2078	3320	4732	9010	17864	35553
		M3	384	670	1355	2209	3417	4801	7977
	WB	M1	1092	1540	2160	2355	4024	7593	14759
		M2	1087	1536	2157	2354	4023	7592	14758
		M3	369	558	941	1147	1664	2323	3407
MPEG2	WT	M1	1197	838	1550	3167	6310	12217	24411
		M2	1191	836	1548	3069	6118	12215	24410
		M3	561	453	613	705	718	754	813
	WB	M1	1073	578	1019	2016	4016	8017	16016
		M2	1067	577	1018	2015	4015	8016	16015
		M3	494	321	410	474	492	534	616

Table 4. The number of soft errors which occur during execution [10^{-17}errors/instruction].

2.7 Conclusion

This section discussed the simulation-based soft error estimation technique which sought the accurate number of soft errors for a computer system to finish running a program. Depending on application programs which are executed on a computer system, its reliability changes. The important point to emphasize is that seeking for the number of soft errors to run a program is essential for accurate soft-error estimation of computer systems. We estimated the accurate number of soft errors of the computer systems which were based on ARM V4T architecture. The experimental results clearly showed the following facts.

- It was found that there was a great difference between the number of soft errors derived with our technique and that derived from the simple summations of the static SERs of memory modules. The dynamic behavior of computer systems must be taken into account for accurate reliability estimation.
- The SER of a computer system virtually increases with a larger cache memory adopted because the SER is calculated by summing up the SERs of memory modules utilized in the system. It was, however, found that the number of soft errors to finish a program was reduced with larger cache memories in the computer system that had an ECC L1 cache and a non-ECC main memory. This is because the soft errors in cache memories were negligible and the retention time of data items in the main memory was reduced by the performance improvement.

3. Reliable microprocessor synthesis for embedded systems

DFR is one of the themes of urgent concern. Coding and parity techniques are popular design techniques for detecting or correcting SEUs in memory modules. Exploiting triple modular redundancy (TMR) is also a popular design technique which decides a correct value by voting on a correct value among three identical modules. These techniques have been well studied and developed. Elakkumanan et al. proposed a DFR technique for logic circuits, which exploits time redundancy by using scan flip-flops (Elakkumanan, 2006). Their approach updates a pair of flip-flops at different moments for an output signal to duplicate for higher reliability. Their approach is effective in ICs which have scan paths. We reported that there exists a trade-off between performance and reliability in a computer system and proposed a DFR technique by adjusting the size of vulnerable cache memory online (Sugihara et al., 2007a, 2008b). The work presented a reliable cache architecture which offered performance and reliability modes. More cache memory is used in the performance mode while less cache memory is used in the reliability mode to avoid SEUs. All tasks are statically scheduled under real-time and reliability constraints. The demerit of the approach is that switching operation modes causes performance and area overheads and might be unacceptable to high-performance or general-purpose microprocessors. We also proposed a task scheduling scheme which minimized SEU vulnerability of a heterogeneous multiprocessor under real-time constraints (Sugihara, 2008a, 2009a). Architectural heterogeneity among CPU cores offers a variety of reliability for a task. We presented a task scheduling problem which minimized SEU vulnerability of an entire system under a real-time constraint. The demerit of the approach is that the fixed heterogeneous architecture loses general-purpose programmability. We also presented a dynamic continuous signature monitoring technique which detects a soft error on a control signal (Sugihara, 2010a, 2011).

This section reviews a system synthesis approach for a heterogeneous multiprocessor system under performance and reliability constraints (Sugihara, 2009b, 2010b). To our best knowledge, this is the first study to synthesize a heterogeneous multiprocessor system with a soft error issue taken into account. In this section we use the SEU vulnerability factor as a vulnerability factor. The other vulnerability factors, however, are applicable to our system synthesis methodology as far as they are capable to estimating task-wise vulnerability on a processor. If a single event transient (SET) is a dominant factor to fail a system, a vulnerability factor which can treat SETs should be used in our heterogeneous multiprocessor synthesis methodology. Our methodology assumes that a set of tasks are given and that several variants of processors are given as building blocks. It also assumes that real-time and vulnerability constraints are given by system designers. Simulation with every combination of a processor model and a task characterizes performance and reliability. Our system synthesis methodology uses the values of the chip area of every building block, the characterized runtime and vulnerability, and the given real-time and vulnerability constraints in order to synthesize a heterogeneous multiprocessor system whose chip area is minimal under the constraints.

3.1 Performance and reliability in various processor configurations

A processor configuration, which specifies instruction set architecture, the number of pipeline stages, the size of cache memory, cache architecture, coding redundancy, structural redundancy, temporal redundancy, and so on, is a major factor to determine chip area,

performance and reliability of a computer system. One must carefully select a processor configuration for each processor core of their products so that they can make the price of their products competitive. From the viewpoint of reliability, processor configurations are mainly characterized by the following design parameters.

- Coding techniques, i.e. parity and Hamming codes.
- Modular redundancy techniques i.e. double modular redundancy (DMR) and triple modular redundancy (TMR).
- Temporal redundancy techniques, i.e. multiple executions of a task and multi-timing sampling of outputs of a combinational circuit.
- The size of cache memory. We reported that SRAM is a vulnerable component and the size of cache memory would be one of the factors which characterize processor reliability (Sugihara et al., 2006, 2007b).

Design parameters are required to offer various alternatives which cover a wide range of chip area, performance, and reliability for building a reliable and small multiprocessor. This chapter mainly focuses on the size of cache memory as an example of variable design parameters in explanation of our design methodology. The other design parameters as mentioned above, however, are applicable to our heterogeneous multiprocessor synthesis paradigm.

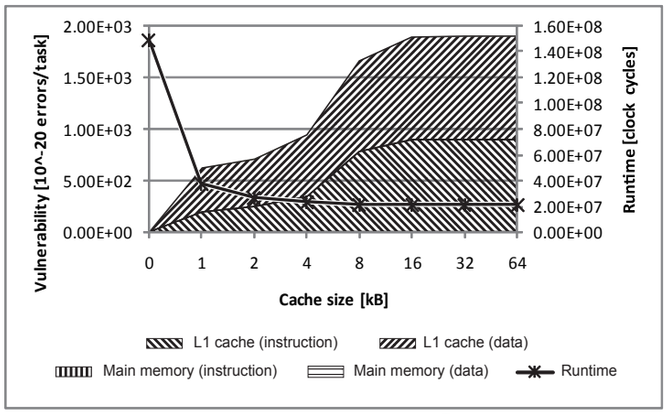


Fig. 9. Cache size vs SEU vulnerability and performance for susan (input_small, smooth).

Fig. 9 is an example that the cache size, which is one of design parameters, changes runtime and reliability of a computer system. We assumed that the cache line size is 32 bytes and that the number of cache-sets is 32. Changing the number of cache ways from 0 to 64 ranges from 0 to 64 KB of cache memory. For plotting the graph, we utilized an ARM CPU core (ARMv4T instruction set, 200 MHz) and a benchmark program susan, which is a program from the MiBench benchmark suite (Guthaus et al., 2001), with an input file input small and an option "-s". We utilized the vulnerability estimation approach we had formerly proposed (Sugihara, 2006, 2007b). For the processor configuration, we assumed that SRAM and DRAM modules have their own SEC-DED (single error correction and double error detection) circuits. We regarded SETs in logic circuitry as negligible ones because of its infrequency. Note that vulnerability of SRAM in the L1 cache is dominant in the entire vulnerability of the system and that of DRAM in main memory is too small to see in the figure. The figure shows that, as the cache size increases, runtime decreases and SEU

vulnerability increases. The figure shows that the SEU vulnerability converged at 16 KB of a cache memory. This is because using more cache ways than 16 ones did not contribute to reducing conflict misses and did not increase temporal and spatial usage of the cache memory, which determined the SEU vulnerability factor. The cache size at which SEU vulnerability converges depends on a program, input to the program, and cache parameters such as the size of a cache line, the number of cache sets, the number of cache ways, and its replacement policy. The figure shows that most of SEU vulnerability of a system is caused by SRAM circuitry. It clearly shows that there is a trade-off between performance and reliability. A design paradigm in which chip area, performance and reliability can be taken into account is of critical importance in the multi-CPU core era.

3.2 Heterogeneous multiprocessor synthesis

It is quite important to consider the trade-off among chip area, performance, and reliability of a system which one develops. As we discussed in the previous section, chip area, performance and reliability vary among processor configurations. This section discusses a heterogeneous multiprocessor synthesis methodology in which an optimal set of processor configurations are sought under real-time and reliability constraints so that the chip area of a multiprocessor system is minimized.

3.2.1 Overview of heterogeneous multiprocessor synthesis

We show an overview of a heterogeneous multiprocessor synthesis methodology, that is a design paradigm in which a heterogeneous multiprocessor is synthesized and its chip area is minimized under real-time and SEU vulnerability constraints. Figure 10 shows the design flow based on our design paradigm. In the design flow, designers begin with specifying their system. Once they fix their specification, they begin to develop their hardware and software. They may use IP (intellectual property) of processor cores which they designed or purchased before. They may also develop a new processor core if they do not have one appropriate to their system. Various processor configurations are to be prepared by changing design parameters such as their cache size, structural redundancy, temporal redundancy, coding redundancy, and anything else which strongly affects vulnerability, performance, and chip area. Increasing design parameters expands the number of processor configurations, enlarges design space to explore, and causes a long synthesis time. Design parameters should be chosen to offer design alternatives among chip area, performance, and reliability. Even if any design parameter can be treated in a general optimization procedure, design parameters should be carefully chosen in order to avoid large design space exploration. A design parameter which offers slight difference regarding chip area, performance, and reliability would result in a long synthesis time and should be possibly excluded from our multiprocessor synthesis. Software is mainly developed at a granularity level of tasks. ISS is performed with the object codes for obtaining accurate runtime and SEU vulnerability on every processor configuration. SEU vulnerability can be easily obtained with the vulnerability estimation techniques previously mentioned. We used the reliability estimation technique (Sugihara et al., 2006, 2007b) throughout this chapter but any other technique can be used as far as it is capable of estimating task-wise reliability on a processor configuration. When SETs become dominant in reliability of a computer system, one should use a reliability estimation technique which treats SETs. Our heterogeneous multiprocessor

synthesis paradigm is basically independent of a reliability estimation technique as far as it characterizes task-wise runtime and vulnerability. One should specify reliability and performance constraints from which one obtains the upper bound of the SEU vulnerability factor for every task, the upper bound of the SEU vulnerability for total tasks, and arrival and deadline times of all tasks. From the specification and the hardware and software components which one has given, a mixed integer linear programming (MILP) model to synthesize a heterogeneous multiprocessor system is automatically generated. By solving the MILP model with the generic solving procedure, an optimal configuration of the heterogeneous multiprocessor is sought. This chapter mainly focuses on defining the heterogeneous multiprocessor synthesis problem and building an MILP model to synthesize a heterogeneous multiprocessor system. Subsection 3.2.2 formally defines the heterogeneous multiprocessor synthesis problem and Subsection 3.2.3 gives an MILP model for the problem.

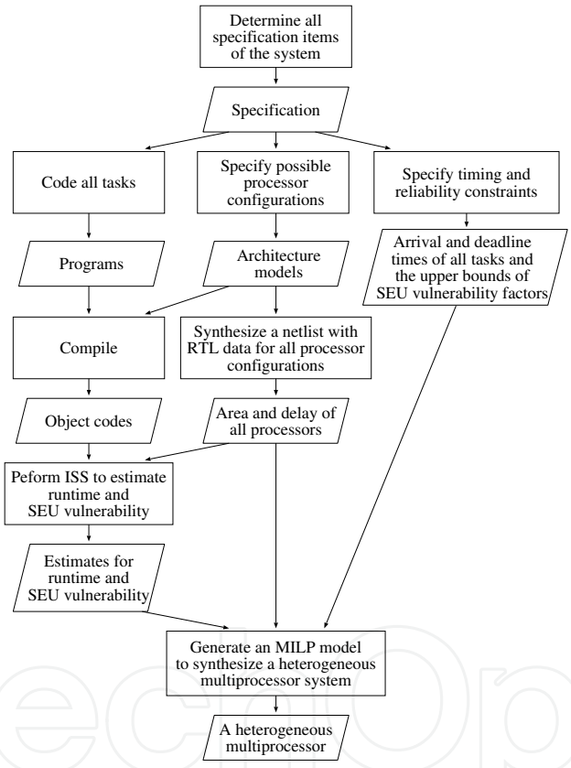


Fig. 10. Our design paradigm.

3.2.2 Problem definition

We now address a mathematical problem in which we synthesize a heterogeneous multiprocessor system and minimize its chip area under real-time and SEU vulnerability constraints. We synthesize a heterogeneous multiprocessor on which N_{task} tasks are executed. N_{CPU} processor configurations are given as building blocks for the heterogeneous multiprocessor system. The chip area of Processor Configuration k, $1 \leq k \leq N_{\mathrm{CPU}}$, is given with A_k. We assume that all the tasks are non-preemptive on the heterogeneous multiprocessor system. Preemption causes large deviations between the worst-case

execution times (WCET) of tasks that can be statically guaranteed and average-case behavior. Non-preemptivity gives a better predictability on runtime since the worst-case is closer to the average case behavior. Task i, $1 \leq i \leq N_{\text{task}}$, becomes available to start at its arrival time T_{arrival_i} and must finish by its deadline time T_{deadline_i}. Task i runs for Duration $D_{\text{runtime}_{i,k}}$ on Processor Configuration k. The SEU vulnerability factor for Task i to run on Processor Configuration k, $V_{i,k}$, is the number of critical SEUs which occur during the task execution. We assume that one specifies the upper bound of the SEU vulnerability factor of Task i, V_{const_i}, and the upper bound of the SEU vulnerability factor of the total tasks, $V_{\text{const}_{\text{all}}}$.

The heterogeneous multiprocessor synthesis problem that we address in this subsection is to minimize the chip area of a heterogeneous multiprocessor system by optimally determining a set of processor cores constituting a heterogeneous multiprocessor system, the start times $s_1, s_2, \cdots, s_{N_{\text{task}}}$ for all tasks, and assignments of a task to a processor core. The heterogeneous multiprocessor synthesis problem P_{HMS} is formally stated as follows.

- P_{HMS}: For given N_{task} tasks, N_{CPU} processor configurations, the chip area A_k of Processor Configuration k, arrival and deadline times of Task i, T_{arrival_i} and T_{deadline_i}, duration $D_{\text{runtime}_{i,k}}$ for which Task i runs on Processor Configuration k, the SEU vulnerability factor $V_{i,k}$ for Task i to run on Processor Configuration k, the upper bound of the SEU vulnerability factor for Task i, V_{const_i} , and the upper bound of the SEU vulnerability factor for total tasks, $V_{\text{const}_{\text{all}}}$, determine an optimal set of processor cores, assign every task to an optimal processor core, and determine the optimal start time of every task such that (1) every task is executed on a single processor core, (2) every task starts at or after its arrival time and completes by its deadline, (3) the SEU vulnerability of every task is less than or equal to that given by system designers, (4) the total SEU vulnerability of the system is less than or equal to that given by system designers and (5) the chip area is minimized.

3.2.3 Problem definition

We now build an MILP model for Problem P_{HMS}. From the assumption of non-preemptivity, the upper bound of the number of processors of the multiprocessor system is given by the number of tasks, N_{task}. Let $x_{i,j}$, $1 \leq i \leq N_{\text{task}}$, $1 \leq j \leq N_{\text{task}}$ be a binary variable defined as follows:

$$x_{i,j} = \begin{cases} 1 & \text{if Task } i \text{ is assigned to Processor } j, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

Let $y_{j,k}$, $1 \leq j \leq N_{\text{task}}$, $1 \leq k \leq N_{\text{CPU}}$ be a binary variable defined as follows:

$$y_{j,k} = \begin{cases} 1 & \text{if one takes Processor Configuration } k \text{ as the one of Processor } j, \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

The chip area of the heterogeneous multiprocessor is the sum of the total chip areas of all processor cores used in the system. The total chip area A_{chip}, which is the objective function, is, therefore, stated as follows:

$$A_{\text{chip}} = \sum_{j,k} A_k y_{j,k}. \tag{9}$$

The assumption of non-preemptivity causes a task to run on only a single processor. The following constraint is, therefore, introduced.

$$\sum_j x_{i,j} = 1, 1 \le \forall i \le N_{\text{task}}. \tag{10}$$

If a task is assigned to a single processor, the processor must have its entity. The following constraint, therefore, is introduced.

$$x_{i,j} = 1 \rightarrow \sum_k y_{j,k} = 1, 1 \le \forall i \le N_{\text{task}}, 1 \le \forall j \le N_{\text{task}}. \tag{11}$$

The reliability requirement varies among tasks, depending on the disprofit of a failure event of a task. We assume that one specifies the upper bound of the SEU vulnerability factor for each task. The SEU vulnerability factor of Task i must be less than or equal to V_{const_i}. The SEU vulnerability factor of a task is determined by assignment of the task to a processor. The following constraint, therefore, is introduced.

$$\sum_{j,k} V_{i,k}\, x_{i,j} y_{j,k} \le V_{\text{const}_i}, 1 \le \forall i \le N_{\text{task}}. \tag{12}$$

The SEU vulnerability factor of the heterogeneous multiprocessor system is the sum of the SEU vulnerability factors of all tasks. The SEU vulnerability of the computer system V_{chip}, therefore, is stated as follows.

$$V_{\text{chip}} = \sum_{i,j,k} V_{i,k}\, x_{i,j} y_{j,k}. \tag{13}$$

We assume that one specifies an SEU vulnerability constraint, which is the upper bound of the SEU vulnerability of the system, and so the following constraint is introduced.

$$V_{\text{chip}} \le V_{\text{const}_{\text{all}}}. \tag{14}$$

Task i starts between its arrival time T_{arrival_i} and its deadline time T_{deadline_i}. A variable for start time s_i is, therefore, bounded as follows.

$$T_{\text{arrival}_i} \le s_i \le T_{\text{deadline}_i}, 1 \le \forall i \le N_{\text{task}} \tag{15}$$

Task i must finish by its deadline time T_{deadline_i}. A constraint on the deadline time of the task is introduced as follows.

$$s_i + \sum_{j,k} D_{\text{runtime}_{i,k}}\, x_{i,j} y_{j,k} \le T_{\text{deadline}_i}, 1 \le \forall i \le N_{\text{task}} \tag{16}$$

Now assume that two tasks $i1$ and $i2$ are assigned to Processor j and that its processor configuration is Processor Configuration k. Formal expressions for these assumptions are shown as follows:

$$x_{i1,j} = x_{i2,j} = y_{j,k} = 1. \tag{17}$$

Two tasks are simultaneously inexecutable on the single processor. The two tasks must be sequentially executed on the single processor. Two tasks i1 and i2 are inexecutable on the single processor if $s_{i1} < s_{i2} + D_{\text{runtime}_{i2,k}}$ and $s_{i1} + D_{\text{runtime}_{i1,k}} > s_{i2}$. The two tasks, inversely, are executable on the processor under the following constraints.

$$x_{i1,j} = x_{i2,j} = y_{j,k} = 1 \rightarrow \left\{\left(s_{i1} + D_{\text{runtime}_{i1,k}} \le s_{i2}\right) \vee \left(s_{i2} + D_{\text{runtime}_{i2,k}} \le s_{i1}\right)\right\},$$

$$1 \le \forall i1 < \forall i2 \le N_{\text{task}},\ 1 \le \forall j \le N_{\text{task}}, \text{ and } 1 \le \forall k \le N_{\text{CPU}}. \quad (18)$$

The heterogeneous multiprocessor synthesis problem is now stated as follows.

Minimize the cost function $A_{\text{chip}} = \sum_{j,k} A_k\, y_{j,k}$

subject to

1. $\sum_j x_{i,j} = 1, 1 \le \forall i \le N_{\text{task}}.$
2. $x_{i,j} = 1 \rightarrow \sum_k y_{j,k} = 1, 1 \le \forall i \le N_{\text{task}}, 1 \le \forall j \le N_{\text{task}}.$
3. $\sum_{j,k} V_{i,k}\, x_{i,j} y_{j,k} \le V_{\text{const}_i}, 1 \le \forall i \le N_{\text{task}}.$
4. $\sum_{j,k} V_{i,k}\, x_{i,j} y_{j,k} \le V_{\text{const}_{\text{all}}}.$
5. $s_i + \sum_{j,k} D_{\text{runtime}_{i,k}}\, x_{i,j} y_{j,k} \le T_{\text{deadline}_i}, 1 \le \forall i \le N_{\text{task}}.$
6. $x_{i1,j} = x_{i2,j} = y_{j,k} = 1 \rightarrow \{(s_{i1} + D_{\text{runtime}_{i1,k}} \le s_{i2}) \vee (s_{i2} + D_{\text{runtime}_{i2,k}} \le s_{i1})\}, \quad 1 \le \forall i1 < \forall i2 \le N_{\text{task}}, 1 \le \forall j \le N_{\text{task}}, \text{and } 1 \le \forall k \le N_{\text{CPU}}.$

Variables

- $x_{i,j}$ is a binary variable, $1 \le \forall i \le N_{\text{task}}, 1 \le \forall j \le N_{\text{task}}.$
- $y_{j,k}$ is a binary variable, $1 \le \forall j \le N_{\text{task}}, 1 \le \forall k \le N_{\text{CPU}}.$
- s_i is a real variable, $1 \le \forall i \le N_{\text{task}}.$

Bounds

- $T_{\text{arrival}_i} \le s_i \le T_{\text{deadline}_i},\ 1 \le \forall i \le N_{\text{task}}.$

The above nonlinear mathematical model can be transformed into a linear one using standard techniques (Williams, 1999) and can be solved with an LP solver. Seeking optimal values for the above variables determines hardware and software for the heterogeneous system. Variables $x_{i,j}$ and s_i determine the optimal software and Variable $y_{j,k}$ determines the optimal hardware. The other variables are the intermediate ones in the problem. As we showed in Subsection 3.2.2, the values N_{task}, N_{CPU}, A_k, T_{arrival_i}, $D_{\text{runtime}_{i,k}}$, $V_{i,k}$, V_{const_i} , and $V_{\text{const}_{\text{all}}}$ are given. Once these values are given, the above MILP model can be generated automatically. Solving the generated MILP model optimally determines a set of processors, assignment of every task to a processor core, and start time of every task. The set of processors constitutes a heterogeneous multiprocessor system which satisfies the minimal chip area under real-time and SEU vulnerability constraints.

3.3 Experiments and results

3.3.1 Experimental setup

We experimentally synthesized heterogeneous multiprocessor systems under real-time and SEU vulnerability constraints. We prepared several processor configurations in which the system consists of multiple ARM CPU cores (ARMv4T, 200 MHz). Table 5 shows all the

processor configurations we hypothetically made. They are different from one another regarding their cache sizes. For the processor configurations, we adopted write-through policy (Hennessy & Patterson, 2002) as write policy on hit for the cache memory. We also adopted the LRU policy (Hennessy & Patterson, 2002) for cache line replacement. For experiment, we assumed that each of ARM cores has its own memory space and does not interfere the execution of the others. The cache line size and the number of cache-sets are 32 bytes and 32, respectively. We did not adopt error check and correct (ECC) circuitry for all memory modules. Note that the processor configurations given in Table 5 are just examples and the other design parameters such as coding redundancy, structural redundancy, temporal redundancy, and anything else which one wants, are available. The units for runtime and vulnerability in the table are M cycles/execution and 10^{-18} errors/execution respectively.

	L1 cache size [KB]	Hypothetical chip area [a.u.]
Conf. 1	0	64
Conf. 2	1	80
Conf. 3	2	96
Conf. 4	4	128
Conf. 5	8	192
Conf. 6	16	320

Table 5. Hypothetical processor configurations for experiment.

We used 11 benchmark programs from MiBench, the embedded benchmark suite (Guthaus et al., 2001). We assumed that there were 25 tasks with the 11 benchmark programs. Table 6 shows the runtime, the SEU vulnerability, and the SER of a task on every processor configuration.

As the size of input to a program affects its execution time, we regarded execution instances of a program, which are executed for distinct input sizes, as distinct jobs. We also assumed that there was no inter-task dependency. The table shows runtime and SEU vulnerability for every task to run on all processor configurations. These kinds of vulnerabilities can be obtained by using the estimation techniques formerly mentioned. In our experiments, we assumed that the SER of SRAM modules is 1.0×10^{-4} [FIT/bit], for which we referred to Slayman's paper (Slayman, 2005), and utilized the SEU vulnerability estimation technique which mainly estimated the SEU vulnerability of the memory hierarchy of systems (Sugihara et al., 2006, 2007b). Note that our synthesis methodology does not restrict designers to a certain estimation technique. Our synthesis technique is effective as far as the trade-off between performance and reliability exists among several processor configurations.

We utilized an ILOG CPLEX 11.2 optimization engine (ILOG, 2008) for solving MILP problem instances shown in Section 3.2 so that optimal heterogeneous multiprocessor systems whose chip area was minimal were synthesized. We solved all heterogeneous multiprocessor synthesis problem instances on a PC which has two Intel Xeon X5365 processors with 2 GB memory. We gave 18000 seconds to each problem instance for computation. We took a temporal schedule for unfinished optimization processes.

	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7
Program name	bscmth	bitcnts	bf	bf	bf	crc	dijkstra
Input	bscmth_sml	bitcnts_sml	bf_sml1	bf_sml2	bf_sml3	crc_sml	dijkstra_sml
Runtime on Conf. 1	1980.42	239.91	328.69	1.37	2.46	188.22	442.41
Runtime on Conf. 2	1011.63	53.32	185.52	1.05	1.66	43.72	187.67
Runtime on Conf. 3	834.11	53.25	93.68	0.32	0.63	42.97	134.31
Runtime on Conf. 4	684.62	53.15	75.03	0.26	0.51	42.97	93.31
Runtime on Conf. 5	448.90	53.15	74.86	0.26	0.51	42.97	86.51
Runtime on Conf. 6	205.25	53.15	74.86	0.26	0.51	42.97	83.05
Vulnerability on Conf. 1	4171.4	315.1	376.1	1.7	3.1	171.2	2370.3
Vulnerability on Conf. 2	965179.8	41038.1	334963.9	1708.0	2705.0	132178.3	277271.4
Vulnerability on Conf. 3	1459772.8	94799.9	546614.4	1540.6	3154.7	152849.7	385777.1
Vulnerability on Conf. 4	2388614.3	222481.6	709463.0	1301.9	3210.0	186194.8	591639.0
Vulnerability on Conf. 5	5602028.0	424776.5	740064.1	1354.9	3367.6	191300.9	846289.5
Vulnerability on Conf. 6	6530436.1	426503.9	740064.1	1354.9	3367.6	193001.8	1724177.3

Task 8	Task 9	Task 10	Task 11	Task 12	Task 13	Task 14	Task 15	Task 16
dijkstra	fft	fft	jpeg	jpeg	jpeg	jpeg	qsort	sha
dijkstra_lrg	fft_sml1	fft_sml2	jpeg_sml1	jpeg_sml2	jpeg_lrg1	jpeg_lrg2	qsort_sml	sha_sml
2057.38	850.96	1923.92	238.82	66.30	896.22	229.97	153.59	95.28
832.04	412.71	935.99	86.04	32.56	319.03	111.72	75.57	20.04
626.39	286.91	641.06	58.85	18.51	270.63	59.29	46.12	17.23
434.72	224.98	479.29	52.79	14.62	198.36	51.36	45.00	17.06
400.41	183.04	417.04	51.17	14.12	192.59	50.00	44.05	16.74
382.88	182.60	417.02	50.89	14.12	191.62	49.23	43.04	16.74
11417.5	3562.3	12765.0	4160.3	169.2	56258.2	755.9	10589.2	140.6
1252086.8	463504.7	1091299.2	140259.8	53306.2	11540509.4	161705.0	118478.2	30428.2
1811976.1	667661.5	1598447.8	184417l.5	70113.3	11850739.6	206141.0	130503.2	46806.2
2880579.7	1133958.1	2651166.5	316602.2	118874.8	1151005.5	415712.0	174905.9	88481.7
4148898.8	1476214.0	3038682.2	501870.4	197558.2	1855734.6	620950.8	223119.3	153368.5
8638330.6	4042453.5	3223703.4	655647.4	283364.1	2480431.9	1181311.0	323458.3	153589.2

Task 17	Task 18	Task 19	Task 20	Task 21	Task 22	Task 23	Task 24	Task 25
sha	strsrch	strsrch	ssn	ssn	ssn	ssn	ssn	ssn
sha_lrg	strgsrch_sml	strsrch_lrg	ssn_sml1	ssn_sml2	ssn_sml3	ssn_lrg1	ssn_lrg2	ssn_lrg3
991.69	1.75	43.02	143.30	28.42	12.13	2043.75	849.21	226.69
208.21	1.04	23.63	30.08	11.71	5.10	390.87	379.17	105.44
177.25	0.62	14.33	20.96	7.45	2.82	282.18	245.82	58.83
173.88	0.45	10.49	20.25	5.09	2.42	279.57	148.28	43.05
173.88	0.45	10.48	20.24	5.07	2.42	279.48	147.57	43.02
173.88	0.45	10.48	20.24	5.05	2.42	279.45	147.57	43.01
1465.8	1.2	68.7	222.9	121.9	44.3	16179.7	38144.7	11476.0
317100.1	1106.5	27954.0	52800.4	12776.3	7369.5	515954.7	467280.9	267585.5
487613.4	1611.7	51986.9	55307.3	21487.3	8247.0	665690.1	930325.9	309314.3
929878.2	1732.8	80046.3	79470.4	24835.8	10183.9	2215638.8	1152520.6	315312.6
1618482.9	1773.3	87641.1	168981.9	31464.6	13495.2	2748450.9	1373224.1	377518.1
1620777.6	1773.3	89015.0	196048.8	46562.1	16895.8	2896506.3	1662613.3	439999.9

Table 6. Benchmark programs.

3.3.2 Experimental results

We synthesized heterogeneous multiprocessor systems under various real-time and SEU vulnerability constraints so that we could examine their chip areas. We assumed that the arrival time of every task was zero and that the deadline time of every task was same as the others. We also assumed that there was no SEU vulnerability constraint on each task, that is $V_{\mathrm{constraint}_i} = \infty$. Generally speaking, the existence of loosely-bounded variables causes long computation time. It is quite easy to guess that the assumptions make exploration space huge and result in long computation time. The assumption, however, is helpful to obtaining the lower bound on chip area for given SEU vulnerability constraints. The deadline time of all tasks ranged from 3500 to 9500 million cycles and SEU vulnerability constraints of an entire system ranged from 500 to 50000 [10^{-15} errors/system]. Fig. 11 shows the results of heterogeneous multiprocessor synthesis. Chip area ranged from 80 to 320 in arbitrary unit. When we tightened the SEU vulnerability constraints under fixed real-time constraints, more processor cores which have no cache memory were utilized. Similarly, when we tightened the real-time constraints under fixed SEU vulnerability constraints, more processor cores which had a sufficient and minimal size of cache memory were utilized. Tighter SEU vulnerability constraints worked for selecting a smaller size of a cache memory while tighter real-time constraints worked for selecting a larger size of a cache memory. The figure clearly shows that relaxing constraints reduced the chip area of a multiprocessor system.

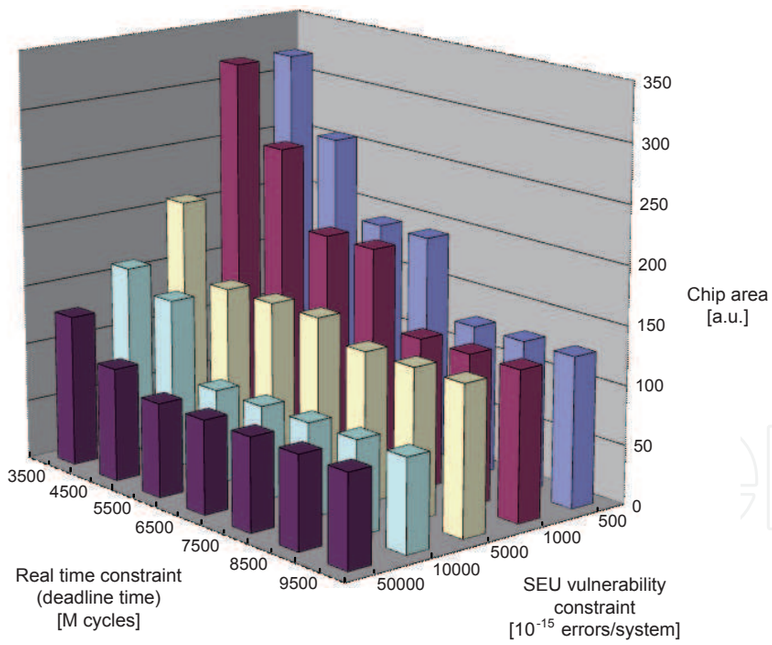


Fig. 11. Heterogeneous multiprocessor synthesis result.

We show four synthesis examples in Tables 7, 8, 9, and 10. We name them HS_1, HS_2, HS_3, and HS_4 respectively. For Synthesis HS_1, we gave the constraints that $T_{\mathrm{deadline_i}} = 3500$ [M cycles] and $V_{\mathrm{const_{all}}} = 5000$ [10^{-15} errors/system]. In this synthesis, a heterogeneous multiprocessor was synthesized which had two Conf. 1 processor cores and a Conf. 2 processor core as shown in Table 7.

For Synthesis HS_2, we gave the constraints that $T_{\text{deadline}_i} = 3500$ [M cycles] and $V_{\text{const}_{\text{all}}} = 500$ [10^{-15} errs/syst]. Only the constraint on $V_{\text{const}_{\text{all}}}$ became tighter in Synthesis HS_2 than in Synthesis HS_1. Table 8 shows that more reliable processor cores were utilized for achieving the tighter vulnerability constraint.

For Synthesis HS_3, we gave the constraints that $T_{\text{deadline}_i} = 3500$ [M cycles] and $V_{\text{const}_{\text{all}}} = 50000$ [10^{-15} errs/syst]. Only the constraint on $V_{\text{const}_{\text{all}}}$ became looser than in Synthesis HS_1. In this synthesis, a single Conf. 4 processor core was utilized as shown in Table 9. The looser constraint caused that a more vulnerable and greater processor core was utilized. The chip area was reduced in total.

For Synthesis HS_4, we gave the constraints that $\text{T}_{\text{deadline}_i} = 4500$ and $V_{\text{const}_{\text{all}}} = 5000$ [10^{-15} errs/syst]. Only the constraint on T_{deadline_i} became looser than in Synthesis HS_1. In this synthesis, a Conf. 1 processor core and a Conf. 2 processor core were utilized as shown in Table 10. The looser constraint on deadline time caused that a subset of the processor cores in Synthesis HS_1 were utilized to reduce chip area.

	Tasks
CPU 1 (Conf. 1)	{10, 13, 20, 25}
CPU 2 (Conf. 1)	{17, 23}
CPU 3 (Conf. 2)	{1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 14, 15, 16, 18, 19, 21, 22, 24}

Table 7. Result for HS_1 ($T_{\text{deadline}_i} = 3.5 \times 10^9$ cycles, $V_{\text{const}_{\text{all}}} = 5 \times 10^{-12}$ errs/syst).

	Tasks
CPU 1 (Conf. 1)	{1, 2, 3, 4, 5, 6, 7, 11, 18, 22}
CPU 2 (Conf. 1)	{8, 9, 14, 15, 16, 21}
CPU 3 (Conf. 1)	{10, 12, 13, 19, 25}
CPU 4 (Conf. 1)	{17, 20, 23}
CPU 5 (Conf. 1)	{24}

Table 8. Result for HS_2 ($T_{\text{deadline}_i} = 3.5 \times 10^9$ cycles, $V_{\text{const}_{\text{all}}} = 5 \times 10^{-13}$ errs/syst).

	Tasks
CPU 1 (Conf. 4)	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25}

Table 9. Result for HS_3 ($T_{\text{deadline}_i} = 3.5 \times 10^9$ cycles, $V_{\text{const}_{\text{all}}} = 5 \times 10^{-11}$ errs/syst).

	Tasks
CPU 1 (Conf. 1)	{1, 6, 10, 14, 16, 19, 21, 25}
CPU 2 (Conf. 2)	{2, 3, 4, 5, 7, 8, 9, 11, 12, 13, 14, 15, 17, 18, 20, 22, 23, 24}

Table 10. Result for HS_4 ($T_{\text{deadline}_i} = 4.5 \times 10^9$ cycles, $V_{\text{const}_{\text{all}}} = 5 \times 10^{-12}$ errs/syst).

3.3.3 Conclusion

We reviewed a heterogeneous multiprocessor synthesis paradigm in which we took real-time and SEU vulnerability constraints into account. We formally defined a heterogeneous multiprocessor synthesis problem in the form of an MILP model. By solving the problem

instances, we synthesized heterogeneous multiprocessor systems. Our experiment showed that relaxing constraints reduced chip area of heterogeneous multiprocessor systems. There exists a trade-off between chip area and another constraint (performance or reliability) in synthesizing heterogeneous multiprocessor systems.

In the problem formulation we mainly focused on heterogeneous "multi-core" processor synthesis and ignored inter-task communication overhead time under two assumptions: (i) computation is the most dominant factor in execution time, (ii) sharing main memory and communication circuitry among several processor cores does not affect execution time. From a practical point of view, runtime of a task changes, depending on the other tasks which run simultaneously because memory accesses from multiple processor cores may collide on a shared hardware resource such as a communication bus. If task collisions on a shared communication mechanism cause large deviation on runtime, system designers may generate a customized on-chip network design with both a template processor configuration and the Drinic's technique (Drinic et al., 2006) before heterogeneous system synthesis so that such collisions are reduced.

From the viewpoint of commodification of ICs, we think that a heterogeneous multiprocessor consisting of a reliable but slow processor core and a vulnerable but fast one would be sufficient for many situations in which reliability and performance requirements differ among tasks. General-purpose processor architecture should be studied further for achieving both reliability and performance in commodity processors.

4. Concluding remarks

This chapter presented simulation and synthesis technique for a computer system. We presented an accurate vulnerability estimation technique which estimates the vulnerability of a computer system at the ISS level. Our vulnerability estimation technique is based on cycle-accurate ISS level simulation which is much faster than logic, transistor, and device simulations. Our technique, however, is slow for simulating large-scale programs. From the viewpoint of practicality fast vulnerability estimation techniques should be studied.

We also presented a multiprocessor synthesis technique for an embedded system. The multiprocessor synthesis technique is powerful to develop a reliable embedded system. Our synthesis technique offers system designers a way to a trade-off between chip area, reliability, and real-time execution. Our synthesis technique is mainly specific to "multi-core" processor synthesis because we simplified overhead time for bus arbitration. Our synthesis technique should be extended to "many-core" considering overhead time for arbitration of communication mechanisms.

5. References

Asadi, G. H.; Sridharan, V.; Tahoori, M. B. & Kaeli, D. (2005). Balancing performance and reliability in the memory hierarchy, *Proc. IEEE Int'l Symp. on Performance Analysis of Systems and Software*, pp. 269-279, ISBN 0-7803-8965-4, Austin, Texas, USA, March 2005

Asadi, H.; Sridharan, V.; Tahoori, M. B. & Kaeli, D. (2006). Vulnerability analysis of L2 cache elements to single event upsets, *Proc. Design, Automation and Test in Europe Conf.*, pp. 1276–1281, ISBN 3-9810801-0-6, Leuven, Belgium, March 2006

Baumann, R. B. Radiation-induced soft errors in advanced semiconductor technologies, *IEEE Trans. on device and materials reliability*, Vol. 5, No. 3, (September 2005), pp. 305-316, ISSN 1530-4388

Biswas, A.; Racunas, P.; Cheveresan, R.; Emer, J.; Mukherjee, S. S. & Rangan, R. (2005). Computing architectural vulnerability factors for address-based structures, *Proc. IEEE Int'l Symp. on Computer Architecture*, pp. 532–543, ISBN 0-7695-2270-X, Madison, WI, USA, June 2005

Degalahal, V.; Vijaykrishnan, N.; Irwin, M. J.; Cetiner, S.; Alim, F. & Unlu, K. (2004). SESEE: soft error simulation and estimation engine, *Proc. MAPLD Int'l Conf.*, Submission 192, Washington, D.C., USA, September 2004

Drinic, M.; Krovski, D.; Megerian, S. & Potkonjak, M. (2006). Latency guided on-chip bus-network design, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 25, No. 12, (December 2006), pp. 2663-2673, ISSN 0278-0070

Elakkumanan, P.; Prasad, K. & Sridhar, R. (2006). Time redundancy based scan flip-flop reuse to reduce SER of combinational logic, *Proc. IEEE Int'l Symp. on Quality Electronic Design*, pp. 617-622, ISBN 978-1-4244-6455-5, San Jose, CA, USA, March 2006

Guthaus, M. R.; Ringenberg, J. S.; Ernst, D.; Austin, T. M.; Mudge, T. & Brown, R. B. (2001). MiBench: A Free, commercially representative embedded benchmark suite, *Proc. IEEE Workshop on Workload Characterization*, ISBN 0-7803-7315-4, Austin, TX, USA, December 2001

Hennessy, J. L. & Patterson, D. A. (2002). *Computer architecture: a quantitative approach*, Morgan Kaufmann Publishers Inc., ISBN 978-1558605961, San Francisco, CA, USA

Karnik, T.; Bloechel, B.; Soumyanath, K.; De, V. & Borkar, S. (2001). Scaling trends of cosmic ray induced soft errors in static latches beyond 0.18 μm, *Proc. Symp. on VLSI Circuits*, pp. 61–62, ISBN 4-89114-014-3, Tokyo, Japan, June 2001

Li, X.; Adve, S. V.; Bose, P. & Rivers, J. A. (2005). SoftArch: An architecture level tool for modeling and analyzing soft errors, *Proc. IEEE Int'l Conf. on Dependable Systems and Networks*, pp. 496–505, ISBN 0-7695-2282-3, Yokohama, Japan, June 2005

May, T. C. & Woods, M. H. (1979). Alpha-particle-induced soft errors in dynamic memories, *IEEE Trans. on Electron Devices*, vol. 26, Issue 1, (January 1979), pp. 2–7, ISSN 0018-9383

Mukherjee, S. S.; Weaver, C.; Emer, J.; Reinhardt, S. K. & Austin, T. (2003). A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor, *Proc. IEEE/ACM Int'l Symp. on Microarchitecture*, pp. 29-40, ISBN 0-7695-2043-X, San Diego, CA, USA, December 2003.

Mukherjee, S. S.; Emer, J. & Reinhardt, S. K. (2005). The soft error problem: an architectural perspective, *Proc. IEEE Int'l Symp. on HPCA*, pp.243-247, ISBN 0-7695-2275-0, San Francisco, CA, USA, February 2005

Rebaudengo, M.; Reorda, M. S. & Violante, M. (2003). An accurate analysis of the effects of soft errors in the instruction and data caches of a pipelined microprocessor, *Proc. Design, Automation and Test in Europe*, pp.10602-10607, ISBN 0-7695-1870-2, Munich, Germany, 2003

Seifert, N.; Moyer, D.; Leland, N. & Hokinson, R. (2001a). Historical trend in alpha-particle induced soft error rates of the Alpha(tm) microprocessor," *Proc. IEEE Int'l Reliability Physics Symp.*, pp. 259–265, ISBN 0-7803-6587-9, Orlando, FL, USA, April 2001.

Seifert, N.; Zhu, X.; Moyer, D.; Mueller, R.; Hokinson, R.; Leland, N.; Shade, M. & Massengill, L. (2001b). Frequency dependence of soft error rates for sub-micron CMOS technologies, *Technical Digest of Int'l Electron Devices Meeting*, pp. 14.4.1–14.4.4, ISBN 0-7803-7050-3, Washington, DC, USA, December 2001

Shivakumar, P.; Kistler, M.; Keckler, S. W.; Burger, D. & Alvisi, L. (2002). Modeling the effect of technology trends of the soft error rate of combinational logic, *Proc. Int'l Conf. on Dependable Systems and Networks*, pp. 389-398, ISBN 0-7695-1597-5, Bethesda, MD, June 2002

Slayman, C. W. (2005) Cache and memory error detection, correction and reduction techniques for terrestrial servers and workstations, *IEEE Trans. on Device and Materials Reliability*, vol. 5, no. 3, (September 2005), pp. 397-404, ISSN 1530-4388

Sugihara, M.; Ishihara, T.; Hashimoto, K. & Muroyama, M. (2006). A simulation-based soft error estimation methodology for computer systems, *Proc. IEEE Int'l Symp. on Quality Electronic Design*, pp. 196-203, ISBN 0-7695-2523-7, San Jose, CA, USA, March 2006

Sugihara, M.; Ishihara, T. & Murakami, K. (2007a). Task scheduling for reliable cache architectures of multiprocessor systems, *Proc. Design, Automation and Test in Europe Conf.*, pp. 1490-1495, ISBN 978-3-98108010-2-4, Nice, France, April 2007

Sugihara, M.; Ishihara, T. & Murakami, K. (2007b). Architectural-level soft-error modeling for estimating reliability of computer systems, *IEICE Trans. Electron.*, Vol. E90-C, No. 10, (October 2007), pp. 1983-1991, ISSN 0916-8524

Sugihara, M. (2008a). SEU vulnerability of multiprocessor systems and task scheduling for heterogeneous multiprocessor systems, *Proc. Int'l Symp. on Quality Electronic Design*, ISBN 978-0-7695-3117-5, pp. 757-762, San Jose, CA, USA, March 2008

Sugihara, M.; Ishihara, T. & Murakami, K. (2008b). Reliable cache architectures and task scheduling for multiprocessor systems, *IEICE Trans. Electron.*, Vol. E91-C, No. 4, (April 2008), pp. 410-417, ISSN 0916-8516

Sugihara, M. (2009a). Reliability inherent in heterogeneous multiprocessor systems and task scheduling for ameliorating their reliability, *IEICE Trans. Fundamentals*, Vol. E92-A, No. 4, (April 2009), pp. 1121-1128, ISSN 0916-8508

Sugihara, M. (2009b). Heterogeneous multiprocessor synthesis under performance and reliability constraints, *Proc. EUROMICRO Conf. on Digital System Design*, pp. 333-340, ISBN 978-0-7695-3782-5, Patras, Greece, August 2009.

Sugihara, M. (2010a). Dynamic control flow checking technique for reliable microprocessors, *Proc. EUCROMICRO Conf. on Digital System Design*, pp. 232-239, ISBN 978-1-4244-7839-2, Lille, France, September 2010

Sugihara, M. (2010b). On synthesizing a reliable multiprocessor for embedded systems, *IEICE Trans. Fundamentals*, Vol. E93-A, No. 12, (December 2010), pp. 2560-2569, ISSN 0916-8508

Sugihara, M. (2011). A dynamic continuous signature monitoring technique for reliable microprocessors, *IEICE Trans. Electron.*, Vol. E94-C, No. 4, (April 2011), pp. 477-486, ISSN 0916-8524

Tosaka, Y.; Satoh, S. & Itakura, T. (1997). Neutron-induced soft error simulator and its accurate predictions, *Proc. IEEE Int'l Conf. on SISPAD*, pp. 253–256, ISBN 0-7803-3775-1, Cambridge, MA , USA, September 1997

Tosaka, Y.; Kanata, H.; Itakura, T. & Satoh, S. (1999). Simulation technologies for cosmic ray neutron-induced soft errors: models and simulation systems, *IEEE Trans. on Nuclear Science*, vol. 46, (June, 1999), pp. 774-780, ISSN 0018-9499

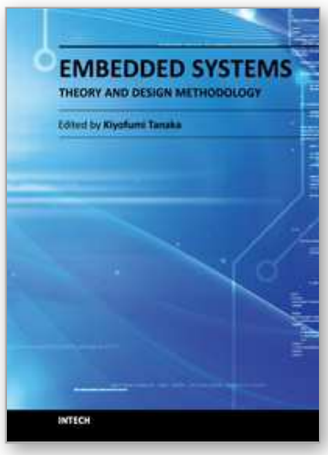
Tosaka, Y.; Ehara, H.; Igeta, M.; Uemura, T & Oka, H. (2004a). Comprehensive study of soft errors in advanced CMOS circuits with 90/130 nm technology, *Technical Digest of IEEE Int'l Electron Devices, pp. 941–948,* ISBN 0-7803-8684-1, San Francisco, CA, USA, December 2004

Tosaka, Y.; Satoh, S. & Oka, H. (2004b). Comprehensive soft error simulator NISES II, *Proc. IEEE Int'l Conf. on SISPAD*, pp. 219–226, ISBN 978-3211224687, Munich, Germany, September 2004

Wang, N. J.; Quek, J.; Rafacz, T. M. & Patel, S. J. (2004). Characterizing the effects of transient faults on a high-performance processor pipeline, *Proc. IEEE Int'l Conf. on Dependable Systems and Networks*, pp.61-70, ISBN 0-7695-2052-9, Florence, Italy, June 2004

Williams, H. P. (1999). Model Building in Mathematical Programming, John Wiley & Sons, 1999

ILOG Inc., CPLEX 11.2 User's Manual, 2008



Embedded Systems - Theory and Design Methodology
Edited by Dr. Kiyofumi Tanaka

ISBN 978-953-51-0167-3
Hard cover, 430 pages
Publisher InTech
Published online 02, March, 2012
Published in print edition March, 2012

Nowadays, embedded systems - the computer systems that are embedded in various kinds of devices and play an important role of specific control functions, have permitted various aspects of industry. Therefore, we can hardly discuss our life and society from now onwards without referring to embedded systems. For wide-ranging embedded systems to continue their growth, a number of high-quality fundamental and applied researches are indispensable. This book contains 19 excellent chapters and addresses a wide spectrum of research topics on embedded systems, including basic researches, theoretical studies, and practical work. Embedded systems can be made only after fusing miscellaneous technologies together. Various technologies condensed in this book will be helpful to researchers and engineers around the world.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen