Soft Error-Aware Design Optimization of Low Power and Time-Constrained Embedded Systems

Rishad A. Shafik[†], Bashir M. Al-Hashimi[†], Krishnendu Chakrabarty[‡]
[†]School of ECS, University of Southampton, Southampton, SO17 1BJ, UK, *e-mail: {ras06r, bmah}@ecs.soton.ac.uk*
[‡]Department of ECE, Duke University, Durham, NC 27708, USA, *e-mail: krish@ee.duke.edu*

***Abstract*— In this paper, we examine the impact of application task mapping on the reliability of MPSoC in the presence of single-event upsets (SEUs). We propose a novel soft error-aware design optimization using joint power minimization with voltage scaling and reliability improvement through application task mapping. The aim is to minimize the number of SEUs experienced by the MPSoC for a suitably identified voltage scaling of the system processing cores such that the power is reduced and the specified real-time constraint is met. We evaluate the effectiveness of the proposed optimization technique using an MPEG-2 decoder and random task graphs. We show that for an MPEG-2 decoder with four processing cores, our optimization technique produces a design that experiences 38% less SEUs than soft error-unaware design optimization for a soft error rate of 10^{-9}, while consuming 9% less power and meeting a given real-time constraint. Furthermore, we investigate the impact of architecture allocation (varying the number of MPSoC cores) on the power consumption and SEUs experienced. We show that for an MPSoC with six processing cores and a given real-time constraint, the proposed technique experiences upto 7% less SEUs compared to soft error-unaware optimization, while consuming only 3% more power.**

I. Introduction

Dynamic Voltage Scaling (DVS) is an effective power minimization technique often employed in hand-held devices to extend battery life [1]. However, it has been reported that the reduction of supply voltage causes an exponential increase in the rate of soft errors, particularly that of SEUs, leading to degradation of reliability [2]. This is further exacerbated by device miniaturization and continuing technology scaling [3]. As a result, reliability is emerging as a key challenge for low power system design [4]. Several researchers have proposed number of power-aware fault tolerance techniques, such as hardware redundancy [4], time and information redundancy [5], task re-execution or replication [6], checkpointing or pre-emptive online scheduling [7]. Recently, fault tolerance-based optimization of cost-constrained distributed real-time systems has been proposed in [8]. The fault tolerance in [8] is achieved through mapping and assignment of fault tolerance policies to different processes. In [9], an approach to fault-tolerant design using process re-execution and scheduling of low power MPSoC applications has been proposed. The works reported in [8, 9] do not consider the impact of application task mapping on the reliability of MPSoC architecture. In this work, we present the first study (to the best of our knowledge) of the impact of task mapping on reliability of MPSoC applications in the presence of SEUs. Based on this study, we propose a novel soft error-aware design optimization to minimize the power consumption and improve reliability in terms of minimized number of SEUs experienced through application task mapping, while meeting a given real-time constraint.

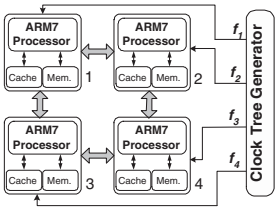

Fig. 1: MPSoC architecture with four processing cores

Scaling, s	f, MHz	V_{dd}, V
1	200	1
2	100	0.58
3	66.7	0.44

TABLE I: Different operating f and V_{dd} for different voltage scalings of ARM7TDMI

II. Preliminaries

A. Architecture Model

We consider a homogeneous MPSoC architecture, A, composed of $\mathcal{C}$ identical processing cores with dedicated inter-core communication due to its high performance [1]. Fig. 1 shows one such architecture with four processing cores. Each processing core consists of an ARM7 processor, data and instruction cache (8kbits and 16kbits), and private memory (512kbits). The cache and memory sizes have been chosen to provide high availability of data and parallelism among the processing cores. To minimize power, clock tree generator is used to feed different voltages and frequencies to the MPSoC processing cores (Fig. 1). The processor dynamic power is

$$P_{dyn} = \alpha C_L f {V_{dd}}^2 \quad . \tag{1}$$

The DVS technique reduces power consumption in (1) by reducing V_{dd} and f. For ARM7TDMI, V_{dd} (in volts) is expressed in terms of f (in MHz) as [10]

$$V_{dd}(f,s) = \left[0.1667 + \frac{4.1667 \times f}{10^3 \times s}\right], \tag{2}$$

where s is the voltage scaling coefficient. Table I shows the different voltage scaling options used in this work. The impact of choice of voltage scaling levels on design optimization is discussed in Section V.

B. Application and Fault Models

We model an application as a directed, acyclic task graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ with N nodes. Each node $t_i \in \mathcal{V}$ represents one computational task within the application and each edge $d_{ij} \in \mathcal{E}$ represents inter-task communication and dependency. An application is realized on MPSoC by distributing the tasks among the processing cores through task mapping process. Fig. 2 shows an example task graph of MPEG-2 video decoder using eleven tasks. The tasks' computational and communication costs of tasks are shown with numbers on the nodes and edges, Fig. 2. The computational cost represents execution time of each task and the communication cost

represents the time required to transfer data between tasks (all costs are multiples of 5.5×10^6 clock cycles). All costs are obtained from SystemC cycle-accurate simulation assuming 32-bit inter-core transfer.

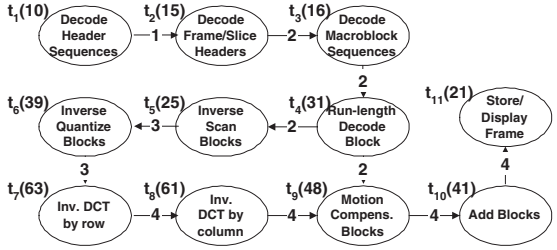

Fig. 2: MPEG-2 video decoder task graph

In this work, fault injection is carried out using SEU-based fault model employing the technique reported in [11]. The fault injection is initiated through replacement of original data/signal types to fault injection enabler types, which facilitates a centralized list of register space for fault injections. For a given soft error rate (SER, expressed as number of SEUs per bit per cycle), the number of SEUs to be injected is identified and their locations are determined using Poisson distribution. Using SystemC cycle-accurate simulation, register usage including cache and memory registers and the number of SEUs experienced are found.

III. Impact of Task Mapping on Reliability

Reliability of an MPSoC application in the presence of SEUs is related to the total number of SEUs experienced [12]. For a soft error rate (SER) of λ_i (SEUs per bit per clock cycle), the total number of SEUs experienced by an MPSoC with $\mathcal{C}$ processing cores is given by [12] as

$$\Gamma = \sum_{i=1}^{\mathcal{C}} R_i T_i \lambda_i = T_M \sum_{i=1}^{\mathcal{C}} R_i \alpha_i \lambda_i \quad , \tag{3}$$

where T_i is the execution time (in clock cycles), R_i is the register usage (in bits per clock cycle) of i-th processing core and T_M is the multiprocessor execution time (in clock cycles, $\forall_i$:T_i=$\alpha_i T_M$). The register usage, R_i, is a measure of average number of registers used over the execution time by a processing core and defined as [11]

$$R_i = \frac{1}{T_i} \sum_{t=1}^{T_i} R_{i,t} \quad , \tag{4}$$

where $R_{i,t}$ is the register usage (in bits) at t-th clock cycle of the i-th processing cores. The R_i in (4) of a processing core depends on the nature of processing being carried out by the tasks mapped, data dependency and resource sharing among them. The T_M in (3) affects multiprocessor performance and depends on the number of mapped tasks on a processing core and the data dependency among them. As a result, when more related tasks are mapped on a processing core, T_M increases for a given operating frequency but the register resources related to tasks are localized reducing the overall register usage (R=$\sum_i R_i$). On the other hand, when tasks are distributed among processing cores to achieve higher parallelism, T_M decreases at the expense of increased R due to higher duplication of shared register resources among tasks. An example of this trade-off follows. In the MPEG-2 decoder (Fig. 2), the tasks t_5 and t_6 share nearly 6.4kb registers, while the tasks t_6, t_7 and t_8 share about 8kb registers among them. To reduce register usage, for example it is possible to map tasks t_5, t_6, t_7 and t_8 on a processing core. However, due to computationally intensive nature of these tasks, T_M will be high. To reduce T_M, an alternative option is to map tasks t_5 and t_6 on a processing core, while the tasks t_7 and t_8 can be mapped on another core. However, this gives a duplication of about 14.4kb registers (increased R) between the processing cores. Because of this register usage (R) and multiprocessor execution time (T_M) trade-off, the MPSoC experiences varying number of SEUs (Γ) for different task mappings given by (3). The Γ also depends on the voltage scaling of the MPSoC processing cores as it affects λ_i in (3). To demonstrate the impact of application task mapping and voltage scaling on the number of SEUs experienced (Γ), a total of 120 task mappings were carried out using the MPEG-2 decoder (Fig. 2) on the MPSoC architecture (Fig. 1). Fig. 3 shows the T_M, R and Γ obtained through SystemC simulation and fault injection (Section II-B) using an SER of 10^{-9} SEU per bit per cycle (i.e. 1 SEU per 10ms for 1kb register bank) as an example. Three key observations are made:

Observation 1: Fig. 3(a) shows the trade-off between multiprocessor execution time (T_M, ms) and overall register usage (R). As can be seen, when tasks are mapped to reduce R by localization of tasks, T_M increases. On the other hand, as tasks are mapped to reduce T_M, register resources shared among tasks are duplicated, leading to increased register usage, R.

Observation 2: Fig. 3(b) shows the total number of SEUs experienced (Γ) and multiprocessor execution time, T_M (in ms), when all the decoder processing cores are scaled by 1 (f=200MHz and V_{dd}=1V). It can be seen that when tasks are distributed among processing cores to reduce T_M, the decoder experiences more higher Γ given by (3) due to higher R (Fig. 3(a)). When tasks are localized to reduce R, the decoder also experiences higher number of SEUs due to increased T_M (Fig. 3(a)). This results in a concave curve for Γ given by (3), with the minimum Γ located around the middle of T_M range.

Observation 3: Fig. 3(c) shows the total number of SEUs experienced (Γ) and multiprocessor execution time (T_M, in ms) when all the decoder processing cores are scaled by 2 (f=100MHz and V_{dd}=0.58V). As can be seen, Γ increases by approximately 2.5 times due to lowered V_{dd} from 1V to 0.58V (due to V_{dd} and λ relationship in [2]) and T_M is increased by approximately 2 due to reduced f from 200MHz to 100MHz.

The above observations demonstrate the impact of application task mapping and voltage scaling on the MPSoC decoder reliability in the presence of SEUs. Hence, an interesting design optimization problem is to identify suitable voltage scaling of the MPSoC processing cores to minimize power consumption and to improve reliability through application task mapping, while meeting a real-time constraint.

IV. Proposed Design Optimization

In this section, we propose a novel design optimization using joint power minimization and reliability improvement of an application implemented on an MPSoC architecture. Fig. 4 shows flowchart of the proposed design optimization with three major steps: power minimization, soft error-aware application task mapping and iterative assessment. For a given SER and real-time constraint, the design optimization

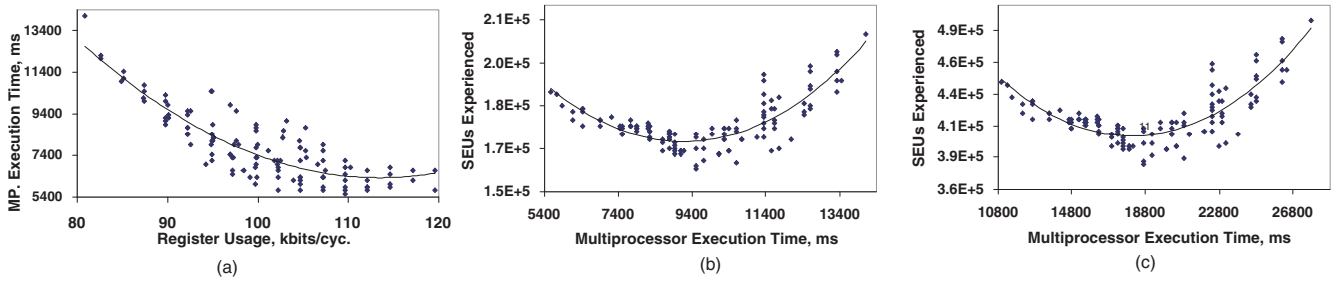

Fig. 3: (a) Trade-off between execution time and register usage, (b) SEUs experienced and MPSoC execution time, and (c) SEUs experienced and execution time (all cores scaled by 2), for task mappings with MPEG decoder with four processing cores

is initiated by power minimization (step 1) through voltage scaling of the MPSoC cores. This is followed by soft error-aware application task mapping (step 2) to minimize the number of SEUs experienced for the chosen voltage scalings in step 1. These two steps are repeated and assessed in step 3 until a design with minimized power consumption and minimized SEUs experienced is found, meeting the real-time constraint. The design optimization steps are discussed next.

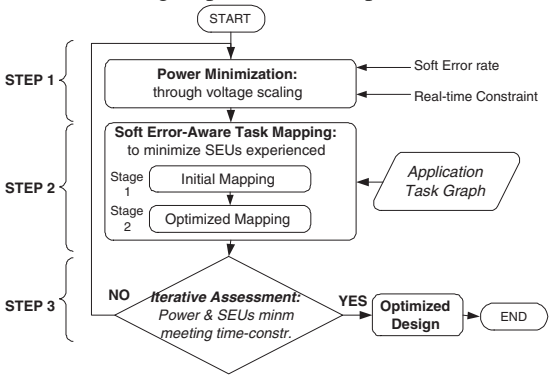

Fig. 4: Flowchart of the proposed design optimization

A. Power Minimization and Iterative Assessment

Power minimization of the proposed design optimization is performed using the voltage scaling algorithm, Fig. 5(a). The voltage scaling algorithm, *nextScaling*, starts with the lowest voltage scaling on all identical cores and generates the next set of higher voltage scalings, *nextS*, based on the previous set of coefficients, *prevS* (Fig. 5(a)). In each iteration, *nextS* is updated as the *prevS* reduced by 1 on a processing core until the voltage scaling reaches the nominal voltage scaling level (s=1, lines 3-6). When the nominal level (s=1) is reached by a core, *nextS* is updated by increasing voltage scaling of the core by 1 (line 9) and reducing the voltage scaling of the next processing core by 1 in steps (lines 7-11). The aim is to generate non-repetitive combinations and reduce the number of voltage scalings that need to be investigated. For example, for an architecture with four processing cores (Fig. 1) and three scaling options (Table I), the voltage scaling scaling algorithm, Fig. 5(a), generates 15 unique combinations starting with scaling coefficient of 3 for all cores, Fig. 4(b), compared to a total of 3^4=81 possible combinations.

With each chosen voltage scaling resulting from the scaling algorithm (step 2, Fig. 4) is carried out to minimize the number of SEUs experienced through application task mapping. The resulting power consumption and SEUs experienced are assessed in step 3. Using (2), the dynamic power consumption, P, of the MPSoC with $\mathcal{C}$ processing cores can be expressed

```
//C = no of cores, prevS = previous scaling
[nextS] = nextScaling(prevS): begin
 1: copy prevS into nextS
 2: for i := 1 to C
 3:   if prevS[i] > 1: begin //1 is lowest scale
 4:     nextS[i] := prevS[i]-1;
 5:     break;
 6:   end if
 7:   else
 8:     for k := i to C
 9:       nextS[k] = prevS[k]+1;
10:     end for
11:   end else
12: end for
13: return nextS;
end
```

(a)

Iter. #	Scaling Coefficients: Core 1, s_1	Core 2, s_2	Core 3, s_3	Core 4, s_4
1	3	3	3	3
2	3	3	3	2
3	3	3	3	1
4	3	3	2	2
5	3	3	2	1
6	3	3	1	1
7	3	2	2	2
8	3	2	2	1
9	3	2	1	1
10	3	1	1	1
11	2	2	2	2
12	2	2	2	1
13	2	2	1	1
14	2	1	1	1
15	1	1	1	1

(b)

Fig. 5: (a) Voltage scaling algorithm, (b) Voltage scaling coefficients for four processing cores

as a function of voltages scaling identified in step 1, s_i, as

$$P = C_L \sum_{i=1}^{\mathcal{C}} \alpha_i f_i(s_i) V_{dd_i}^2(s_i), \tag{5}$$

where $f_i(s_i)$ and $V_{dd_i}(s_i)$ take values (Table I) depending on the chosen scaling coefficient s_i. The iterative assessment (step 3) is carried out until a better design is found with minimized power consumption (by (5)) and minimized SEUs experienced (by (3)) within a chosen search-time.

```
//  C: no of cores,  G: application task graph with     N tasks, t is the current task
//M: mapping of all cores,     A: MPSoC arch.,  Q: task queue,  L: temporary list
[M] = InitialSEAMapping  (G, C, A):  begin
 1:  push G[0] into Q    // push the first task into Q
 2:  for i:= 1 to C-1  and  Q is not empty
 3:    t := Q.front(); M[i].map(t); delete all mapped tasks from Q
 4:    while  Ti < T Mref  and  no. of unmapped tasks in G > (C-i)
 5:      L : = dependents of    t //(sorted by minimum SEUs)
 6:      if L is empty  and  Q is not empty
 7:        swap last two elements in Q
 8:      else if  Q is not empty
 9:        t = first element in L    //task with minimum SEUs and Time
10:       M[i].map(t); delete t from L; move tasks in L into Q and empty L
11:     else  break  while; end if
12:    t = Q.front();
13:   end while
14: end for
15: return  M;
end
```

Fig. 6: Initial soft error-aware mapping algorithm

B. Soft Error-Aware Application Task Mapping

The problem of application task mapping on MPSoC cores to minimize SEUs experienced (Γ) is an NP-complete problem [8]. We propose a soft error-aware application task mapping in two stages (step 2, Fig. 4): the stage 1 is the initial soft error-aware application task mapping, followed by stage 2 of search based optimized application task mapping. Fig. 6 shows the initial soft error-aware application task mapping algorithm (stage 1), *InitialSEAMapping*, which aims to simplify the optimization process by reducing the number of

task movements. The *InitialSEAMapping* starts with mapping the task with no predecessor in task graph (G) (line 1). The dependents of the currently mapped task in G are then sorted by SEUs experienced if they were to be mapped with the current task and stored in a dependency list, L. The task with the minimum SEUs in L is then mapped next (lines 5-10). This is continued until the execution time of the current core does not exceed the real-time constraint ($T_{M_{ref}}$) and the number of unmapped tasks left in task graph G is higher than the number of remaining cores to ensure tasks are mapped in all cores (lines 4-13). The unmapped tasks are stored in a queue, Q (line 10), which are then mapped gradually to the other cores using the same criteria. After all tasks are mapped, the initial mapping (M) is returned (line 15).

After the initial soft error-aware task mapping (*InitialSEAMapping*, Fig. 6), the design optimization is continued further through optimized mapping (stage 2, step 2, Fig. 4). We use a search based mapping optimization, *OptimizedMapping*, Fig. 7, employing list scheduling for scheduling tasks [8]. The *OptimizedMapping* starts with scheduling

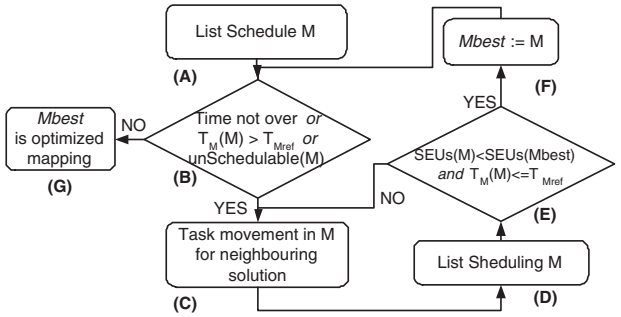

Fig. 7: Flowchart of optimized mapping, *OptimizedMapping*

the initial task mapping, M (step A, Fig. 7). The mapping M is then checked to see if it violates the schedulability requirements or real-time constraints (step B). If any such violation is found, the optimization proceeds with generating neighboring task movements to find out a possible next mapping solution (step C). This mapping (M) is then list scheduled, if schedulable (step D). This is followed by comparison with the the previous best solution, *Mbest*. If M is better than *Mbest* in terms of lower number of SEUs experienced and meets the given real-time constraint, it is then updated as the new *Mbest* (steps E-F). The optimization steps C-F are repeated until the specified search time is not over (step B). Once the search time is over, *Mbest* is returned as the optimized design for the chosen voltage scalings.

In *OptimizedMapping*, the multiprocessor execution time (T_M, in seconds) for an application task mapping is found by the dividing the total number of execution cycles of all mapped tasks by the effective number of cycles executed by processing cores per second for chosen voltage scaling (step B, E, Fig. 7), i.e.

$$T_M = \left[\sum_{i=1}^{\mathcal{C}}\sum_{j=1}^{N}\left(t_j^i + \sum_{k=1}^{N} d_{j,k}^i\right)\right] \Big/ \left[\sum_{i=1}^{\mathcal{C}} \alpha_i f_i(s_i)\right], \quad (6)$$

where t_j^i is the execution time (in clock cycles) of the j-th task mapped on i-th processing core, $d_{j,k}^i$ is the dependency time (in clock cycles) between j-th and k-th task ($j,k = 1:N$) due to selection of j-th task on i-th processing core. The total number of SEUs experienced (Γ) for an application task mapping and chosen voltage scaling on MPSoC processing cores is found in *InitialSEAMapping* (line 5, Fig. 6) and in *OptimizedMapping* (step E, Fig. 7) through (3). The per core execution time (T_i) and register usage (R_i) in (3) are given in terms of mapped tasks as

$$\forall_i : T_i = \sum_{j=1}^{N}\left(t_j^i + \sum_{k=1}^{N} d_{j,k}^i\right), \text{ and} \quad (7)$$

$$\forall_i : R_i = \left|\left(\bigcup_{j=1}^{N}\bigcup_{k=1}^{N} R_{j,k}^i\right)\right|, \quad (8)$$

where $R_{j,k}^i$ is the set of registers shared between j-th and k-th tasks for being mapped on i-th processing core (j=k defines the local register usage of j-th task). As can be seen in (8), R_i is given by cardinality of the register set arising out of union of all $R_{j,k}^i$ in i-th processing core. The proposed optimization is carried out by iterative search through N tasks, with each iteration generating maximum two task movements out of maximum search space of (N-1) dependent tasks. This is followed by second stage search through maximum (N-1) tasks for minimum number of SEUs experienced. As a result, *OptimizedMapping* has worst-case complexity of $O(2N(N-1)(N-1))\approx O(N^3)$.

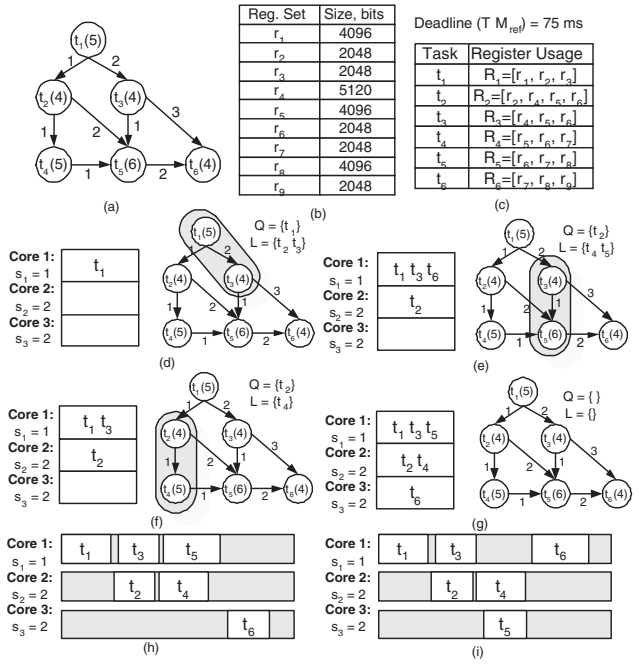

Fig. 8: Example illustration of the soft error-aware application task mapping

An example illustrating the proposed soft error-aware application task mapping algorithm is shown in Fig. 8. In Fig. 8(a), an application task graph with six tasks is shown (all costs are multiples of 60×10^4 cycles) and in Fig. 8(b)-(c) the application registers and their distribution for different tasks are shown. Fig. 8(d)-(g) show the incremental task mapping using *InitialSEAMapping* algorithm, Fig. 6, and finally, Fig. 8(h)-(i) show scheduling and task movements using *OptimizedMapping*, Fig. 7. The chosen voltage scaling for the processing cores are: s_1=1, s_2=2 and s_3=2 and deadline is assumed to be $T_{M_{ref}}$=75ms. As can be seen, after the first task, t_1, in the application task graph, Fig. 8(a), is mapped to processing core 1, the *InitialSEAMapping* mapping algorithm selects t_3, followed by t_6 from dependency list, L. This is

because task t_3 gives the least number of SEUs experienced compared to t_2 and t_5 shown in gray, Fig. 8(d), with the $R_{j,k}$ values from Fig. 8(c). Note that after allocating t_1, t_3 and t_5 on core 1, the deadline constraint cannot be satisfied with further allocation of tasks and the mapping algorithm carries on with the mapping of core 2 selecting tasks t_2 and t_4, which give minimum SEUs experienced, Fig. 8(f). Finally, the unmapped task t_6 in queue (Q) is mapped to core 3, Fig. 8(g). After *InitialSEAMapping* (Fig. 6) is completed, *OptimizedMapping* list schedules the tasks, Fig. 8(h), found through step A, Fig. 7. However, with the chosen voltage scalings for the architecture processing cores, this mapping cannot satisfy the real-time constraint of 75ms. The *OptimizedMapping* swaps t_5 with t_6 in the fourth iteration (step C, Fig. 7) and gives the minimum number of SEUs experienced for the chosen voltage scaling, while meeting $T_{M_{ref}}$=75ms.

V. Experimental Results

We evaluate the effectiveness of the proposed soft error-aware design optimization using four design optimization experiments, Table II. The experiments are carried out using MPEG decoder implemented with the architecture, Fig. 1. The first three experiments, Exp:1, Exp:2 and Exp:3, are soft error-unaware optimization with different design objectives using application task mapping obtained through simulated annealing [13]. Exp:4 is the proposed design optimization. In all experiments, power minimization is obtained through iterative voltage scaling (step 1, Fig. 4) to meet the real-time constraint of decoding a *tennis* video bitstream (ftp://ftp.tek.com/tv/test/streams/Element/) of 437 frames at 29.97 frames per second (fps). The mapped tasks, the voltage scaling and the resulting power consumption (P, mW) are shown in col. 2-4, while the register usage (R, kbits/cyc), the multiprocessor execution time (T_M, clock cycles) and the number of SEUs experienced (Γ) are given in cols. 5-7 (Table II). We impose a time-limit of 40 minutes to search the design space for each voltage scaling of the MPSoC processing cores. All experiments are carried out on an Intel(R) Core(TM)2 2GHz CPU running RHEL5. The number of SEUs experienced (col. 7, Table II) is found by fault injection technique (Section II-B) assuming a SER of 10^{-9} SEUs/bit/cycle (i.e. 1 SEU per 10ms for 1kb register bank).

Exp:1 demonstrates the impact of design optimization with minimized register usage, R. As expected, the design produced gives the least register usage (R) when compared to the other three experiments. The reduced R in Exp:1 is obtained at the expense of highest multiprocessor execution time, T_M as described in Section III, which makes it harder to scale down the voltages of the decoder cores. As a result, Exp:1 gives a design that has higher power consumption than the optimized design produced in Exp:4. However, the design produced in Exp:1 experiences lower SEUs than that in Exp:4. This is because, the proposed design optimization in Exp:4 gives lower voltages of the decoder cores, and hence lower power consumption compared to the design produced in Exp:1. The design produced in Exp:2 is optimized for high parallelism. This gives reduced multiprocessor execution time (T_M), which allows the voltages of the decoder processing cores to be scaled down. As a result, Exp:2 gives lower

Exp.	**Mapped Tasks**		**scal.** s_i	**P**, *mW*	**R**, *kb/c.*	T_M ($\times 10^9$)	Γ ($\times 10^5$)
Exp:1 (Reg. Usage [13])	Core 1	t_1, t_2, t_3	2	9.53	80	1.89	3.46
	Core 2	t_4, t_5	2				
	Core 3	t_6, t_7, t_8, t_9, t_{10}	1				
	Core 4	t_{11}	2				
Exp:2 (Paral-lelism [13])	Core 1	t_1, t_2, t_3, t_4, t_9	3	4.04	118	1.18	5.22
	Core 2	t_5, t_6, t_7	2				
	Core 3	t_8	3				
	Core 4	t_{10}, t_{11}	2				
Exp:3 (Reg. Usage &Paral.[13])	Core 1	t_1, t_2, t_3, t_4, t_5	2	4.15	92	1.26	4.18
	Core 2	t_6, t_7	2				
	Core 3	t_8, t_9	3				
	Core 4	t_{10}, t_{11}	2				
Exp:4 (Proposed)	Core 1	t_1, t_2, t_3, t_4, t_5, t_6	2	4.25	89	1.32	3.93
	Core 2	t_7, t_8	2				
	Core 3	t_9	3				
	Core 4	t_{10}, t_{11}	2				

TABLE II: Comparison of soft error-unaware and the proposed soft error-aware optimizations using MPEG decoder MPSoC with four cores

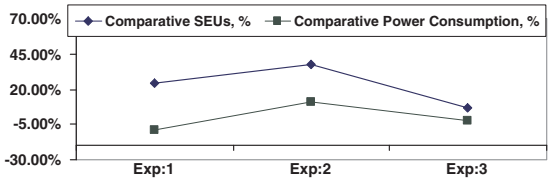

Fig. 9: Comparison of SEUs experienced (Γ) and power consumption (P) between Exp:1, Exp:2, Exp:3 and Exp:4

power consumption than Exp:4. Note that this reduction in multiprocessor execution time (T_M) in Exp:2 is achieved at the expense of the highest register usage (R). Due to lower voltage scaling of the decoder cores and higher register usage, the design optimized for high parallelism, Exp:2, experiences the highest number of SEUs when compared to the other three experiments. In Exp:3, the design has been optimized for both register usage and high parallelism. Such optimization gives a good trade-off between multiprocessor execution time and register usage, and minimizes the product: $T_M \times R$. However, this does not necessarily minimize of the number of SEUs experienced given by (3). The design produced in Exp:4 employs soft error-aware task mapping (Fig. 6) and therefore gives less number of SEUs experienced than the design produced in Exp:3. Note that, although the voltage scaling of the decoder cores are similar, proposed design optimization (Exp:4) gives slightly higher power consumption compared to the design produced in Exp:3 due to more tasks being mapped in core 1 and core 2 of the decoder. Fig. 9 shows comparison of power consumption (P) and SEUs experienced (Γ) by the decoder design in Exp:1, Exp:2 and Exp:3, compared to that of Exp:4. All experiments are carried out with same voltage scaling coefficients (s_1=2, s_2=2, s_3=3 and s_4=2) for an SER of 10^{-9}. As can be seen, the design produced in Exp:4 reduces the number of SEUs experienced by upto 38% compared to the optimized design in Exp:2, while consuming 9% lower power. When compared with the design produced in Exp:1, the optimized design in Exp:4 reduces SEUs experienced by 28%, while consuming only 7% higher power.

To demonstrate the impact of architecture allocation on the proposed design optimization, Table III shows the power consumption (P) and the number of SEUs experienced (Γ) using the design produced in Exp:4. A number of applications, including MPEG decoder and random task graphs of 20 to 100 tasks were used. The cost and the number of dependents in the random task graphs were generated using uniform

App.	2 Cores		3 Cores		4 Cores		5 Cores		6 Cores	
	P, mW	Γ $\times 10^5$	P, mW	Γ $\times 10^5$	P, mW	Γ $\times 10^5$	P, mW	Γ $\times 10^5$	P, mW	Γ $\times 10^5$
MPEG-2	9.1	2.13	5.9	3.17	4.25	3.93	6.34	4.95	7.24	5.36
20 tasks	10.1	0.47	4.15	1.13	4.34	2.27	5.16	2.73	6.36	3.49
40 tasks	6.2	1.07	5.1	1.78	5.2	2.87	6.16	3.46	7.11	4.35
60 tasks	7.8	1.87	4.13	3.25	5.1	4.82	4.9	5.74	5.3	7.15
80 tasks	11.2	1.95	6.1	3.76	4.4	6.13	6.14	7.24	6.69	9.13
100 tasks	10.4	2.40	5.48	4.58	4.8	6.25	5.94	8.83	6.34	11.13

TABLE III: Power consumption and SEUs experienced for different applications and different number of architecture processing cores

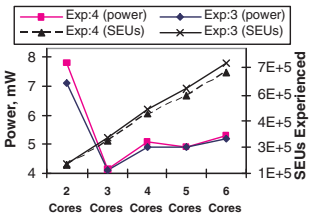

Fig. 10: P (mW) and Γ comparison between Exp:3 and Exp:4

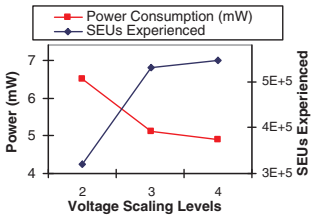

Fig. 11: P (mW) and Γ for different scaling levels

probability distribution with computation cost between 1 and 30, communication cost between 1 to 10 (all costs as multiples of 3.5×10^6 clock cycles), task register usage between 1kbits to 5kbits and the number of dependents was found by exponential distribution between 0 to $N/2$, where N is the number of tasks. The deadline for random task graphs were set to $1000\times N/2$ms. For these task graphs, the design optimization is carried out with imposed time limits of 50, 65, 80, 100, 130 minutes for 20, 40, 60, 80 and 100 tasks, respectively. The P (mW) and Γ for architecture allocations from two to six cores are shown in cols. 2-6 (Table III). Two observations are made. Firstly, the architecture allocation with minimum power consumption (P) depends on the application and real-time constraint. In the case of the MPEG decoder, the least power consumption is found with four cores for the given real-time constraint of decoding *tennis* video bitstream at 29fps. Secondly, with increased number of architecture cores, the number of SEUs experienced increases. It is because with higher number of cores, multiprocessor execution time (T_M) reduces due to higher parallelism enabling higher power reduction through lower voltage scaling. Also, due to reduced T_M, register usage (R) is increased (Section III). As a result, the decoder with six processing cores experiences the highest number of SEUs, compared to the lowest for the decoder with 2 processing cores (row 2, Table III). Similar observations for power consumption and the number of SEUs experienced are also observed with random task graphs. Fig. 10 shows the power consumption (in mW) and the SEUs experienced by the optimized designs produced in Exp:4 and Exp:3 using the random task graph of 60 tasks. As can be seen, the proposed optimization, Exp:4, consistently outperforms the design produced using joint optimization of reduced R and high parallelism, Exp:3, with upto 7% reduction of SEUs experienced for an SER of 10^{-9}. This reliability improvement is achieved with only 3% higher power consumption using an MPSoC with six cores.

To show the impact of choice of voltage scaling levels, Fig. 11 shows the power consumption (mW) and the number of SEUs experienced by the optimized designs produced in Exp:4 with different voltage scaling levels. The design optimizations are carried out using MPSoC with six processing cores with random task graph of 60 tasks and employing the following voltage scaling levels: 2 levels (with 1V−200MHz, and 0.58V−100MHz), 3 levels (Table I) and 4 levels (introducing 1.2V−236MHz in Table I). As can be seen, with 4 scaling levels the proposed design optimization (Exp:4) is able to minimize power further by 4% with only 3% increase in the number of SEUs experienced compared to 3 scaling levels (Fig. 11). This is because with more scaling options, the power minimization (step 1, Fig. 4) has higher flexibility with more combinations of voltage scaling generated by the voltage scaling algorithm (Fig. 5(a)). With 2 scaling levels, it is possible to reduce the number of SEUs experienced by 42% at the cost of 28% higher power consumption compared to 3 scaling levels due to limited voltage scaling options (Fig. 11).

VI. Conclusions

We investigated the impact of application task mapping on the reliability of multiprocessor system-on-chip (MPSoC) (Section III). We proposed a novel soft error-aware design optimization for low power time-constrained MPSoCs (Section IV). Using an MPEG decoder and random task graphs, we showed that proposed optimization can significantly reduce the number of SEUs experienced compared to soft error-unaware optimization, while power consumption is minimized and the real-time constraint is met (Section V). Furthermore, we investigated the impact of architecture allocation on the power consumption and the number of SEUs experienced using the proposed optimization technique (Section V).

Acknowledgement

The authors would like to thank the EPSRC-UK for funding this work in part under grant number EP/E035965/1.

References

[1] B. M. Al-Hashimi, Ed., *System-on-Chip: Next Generation Electronics*. IEE Press, 2006, May, Ch:17.

[2] V. Chandra and R. Aitken, "Impact of technology and voltage scaling on the soft error susceptibility in nanoscale CMOS," in *Proc. of DFT-VLSI*, 2008, pp. 114–122.

[3] F. Dabiri *et al*, "Reliability-aware optimization for DVS-enabled real-time embedded systems," in *Proc. of ISQED'08*, 2008, pp. 780–783.

[4] A. Maheshwari *et al*, "Trading off transient fault tolerance and power consumption in deep submicron (DSM) VLSI circuits," *IEEE TVLSI*, 12 (3), pp. 299–311, March, 2004.

[5] A. Ejlali *et al*, "Combined time and information redundancy for SEU-tolerance in energy efficient real-time systems," *IEEE TVLSI*, 14 (4), pp. 323–335, April, 2006.

[6] G. Chen *et al*, "Energy-aware computation duplication for improving reliability in embedded chip microprocessors," in *Proc. of ASPDAC*, Japan, 2006, pp. 134–139.

[7] Y. Zhang and K. Chakrabarty, "Energy-aware adaptive checkpointing in embedded real-time systems," in *Proc. of DATE'03*, 2003, pp. 10918.

[8] V. Izosimov *et al*, "Design optimization of time- and cost-constrained fault-tolerant distributed embedded systems," in *Proc. of DATE'05*., 2005, pp. 864–869.

[9] P. Pop *et al*, "Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems," in *Proc. of the CODES+ISSS'07*.USA: 2007, pp. 233–238.

[10] J. Pouwelse *et al*, "Dynamic voltage scaling on a low-power microprocessor," in *Proc. of 7th MobiCom*, July, 2001, pp. 251–259.

[11] R. A. Shafik *et al*, "SystemC-based minimum intrusive fault injection technique with improved fault representation," in *Proc. of IOLTS*, Greece, July, 2008, pp. 99–104.

[12] R. A. Shafik *et al*, "Soft error-aware voltage scaling technique for power minimization in application-specific MPSoC," *ASP JOLPE*, vol. 5, no. 2, pp.145–156, August, 2009.

[13] H. Orsilla *et al*, "Automated memory-aware application distribution for multi-processor system-on-chips," *Journal of Systems Architecture: the EUROMICRO Journal*, 53 (11), pp. 795–815, 2007.