EnSuRe: Energy & Accuracy Aware Fault-tolerant Scheduling on Real-time Heterogeneous Systems

Saha, S., Adetomi, A., Zhai, X., Kasap, S., Ehsan, S., Arslan, T. & McDonald-Maier, K.

Original citation & hyperlink:
Saha, S, Adetomi, A, Zhai, X, Kasap, S, Ehsan, S, Arslan, T & McDonald-Maier, K 2021, EnSuRe: Energy & Accuracy Aware Fault-tolerant Scheduling on Real-time Heterogeneous Systems. in 2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS). IEEE, 2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design , Torino, Italy, 28/06/21. https://dx.doi.org/10.1109/iolts52814.2021.9486707

DOI 10.1109/iolts52814.2021.9486707
ISBN 9781665433709

Publisher: IEEE

EnSuRe: Energy & Accuracy Aware Fault-tolerant Scheduling on Real-time Heterogeneous Systems

Sangeet Saha[1], Adewale Adetomi[2], Xiaojun Zhai[1], Server Kasap[3], Shoaib Ehsan[1], Tughrul Arslan[2],[1]Klaus McDonald-Maier
[1]*Embedded and Intelligent Systems Laboratory, University of Essex, UK*
[2]*Ewireless Research Group, School of Engineering, University of Edinburgh, UK*
[3]*School of Computing, Electronics and Maths, Coventry University, UK*
{[1]sangeet.saha, [1]xzhai, [1]sehsan, [1]kdm }@essex.ac.uk, {[2]Adewale.Adetomi, [2]T.Arslan}@ed.ac.uk , [3]server.kasap@coventry.ac.uk

***Abstract*—Energy efficient scheduling of real-time applications without violating real-time constraint has recently become an active research domain. Execution-time of contemporary real-time tasks can individually be divided into: i. execution of the mandatory part within the deadline to obtain a result of acceptable quality, followed by ii. a partial/complete execution of the optional part to improve accuracy of the initially obtained result. Since the mandatory part has stringent timing constraint, provision must be made against any possible run-time fault during execution. In this paper, we propose an energy efficient real-time scheduling strategy called *EnSuRe*, which (i) employs a "time-partitoning" based strategy for executing real-time tasks on primary processors, having low power consumption. The allocation seeks to enhance the accuracy of a task maintaining the deadline and (ii) provides reliability against a fixed number of transient faults by selectively executing backup tasks on backup processor, with high power consumption. Dynamic Power Management was employed to improve the energy efficiency of the overall systems. Simulation results reveal that *EnSuRe* consumes nearly 25% less energy, compared to existing techniques, while satisfying the fault tolerance requirements. *EnSuRe* is also able to achieve 75% system accuracy with 50% system utilisation. Further, the obtained simulation outcomes are validated on benchmark tasks via a fault injection framework on Xilinx ZYNQ APSoC heterogeneous dual core platform.**

***Index Terms*—Heterogeneous processors, Real-time systems, Fault-tolerant scheduling, Energy efficiency**

I. Introduction

In real-time computing, correctness does not only depend on the precision of the results, but also on time at which these are produced. For such critical systems, approximated results obtained within the deadline are preferable over the accurate results generated after this deadline. Utilising approximate computation approaches, a real-time task can be decomposed into a mandatory part, followed by an optional part [1]. The mandatory part must be executed entirely in order to produce an acceptable result within a deadline, while the optional part will be executed for further refinement of the generated result and to provide a higher accuracy of the applications executed.

However, as the mandatory parts have timing constraint, provisions must be made against faults. While executing a task, a processor can often be plagued by either permanent or transient [2] faults. Transient faults are result from factors such as electromagnetic interference or nuclear radiation. Transient fault causes an error in the output of a single task. In order to handle these faults, typically tasks are re-executed on a backup processor to deliver the correct result [2].

However, such re-execution of tasks introduces an energy overhead. Power/energy constraints for real-time systems are particularly important, as these devices often depend upon restricted power source such as batteries [2]. To incorporate energy-aware execution of tasks, two main techniques are widely adopted i.e. i.) *Dynamic Voltage Scaling (DVFS)* technique which trade offs between processor speed and power dissipation [3] and ii.) *Dynamic Power Management (DPM)*, which keeps idle system components in low-power sleep states to preserve power [4]. Recently, we are increasingly witnessing use of heterogeneous (asymmetric) multicore systems, where processing units with different power/performance reside on the same chip, to improve the energy efficiency of the system. ARM's big little systems, Xilinx ZYNQ platform are the examples of such heterogeneous systems [5], [6].

In real-time scheduling, recently the authors in [7], [8], [9], have studied the combined problem of minimizing energy consumption while providing fault tolerance guarantees. However, these studies are limited to either uniprocessor systems or homogeneous multiprocessors. For heterogeneous systems, the authors in [2], [4], [10], have employed standby sparing and primary/backup techniques to provide energy aware fault tolerant solutions. However these works consider hard real-time tasks, not emerging approximation based real-time tasks. Moreover, all of these studies employ standard scheduling scheme like Earliest-Deadline- First (EDF) and Earliest-Deadline-Late (EDL) scheduling policies. The authors also made a strict assumption that all tasks share a fixed and common deadline. In modern safety critical systems, such assumption is no longer generally valid, because based upon their respective criticality, individual tasks must have unique deadlines. Thus, the proposed techniques may perform poorly on multiprocessor system, where multiple tasks require to complete execution requirements within multiple deadlines.

We propose *EnSuRe*, an energy and accuracy aware reliable scheduling strategy for real-time tasks executing on heterogeneous multiprocessor system. *To the best of our knowledge, EnSuRe is the first scheduling mechanism which considers "energy and accuracy" simultaneously to incorporate fault tolerance on a heterogeneous system.* The major contributions

of *EnSuRe* are summarized as follows:

- *EnSuRe* employs a "time-partioning" based task allocation strategy which can effectively allocate tasks on multiprocessor platform based on distinct deadlines. This strategy maintains proportional fairness, while executing task's mandatory parts and utilises available slack periods by executing task's optional parts to enhance accuracy.
- *EnSuRe* tolerates a fixed number of faults [11]. Upon detection of a fault, *EnSuRe* attempts to re-execute the backup tasks within dynamically adjustable slots, such that the deadline of the task remains satisfied and utilisation of the higher power consuming backup processor can be minimised.
- Simulation based experiments with benchmark tasks reveal that *EnSuRe* consumes 25% less energy as compared to the existing techniques.
- *EnSuRe* has also been implemented on heterogeneous ZYNQ APSoC platforms with a fault injection framework. Obtained simulation trends are validated using benchmark task set.

II. System Model and Assumptions

A. Platform and Task Model

In [12], the authors showed multiple cores can be partitioned as primary and backup cores. The adopted architecture model in *EnSuRe* consists of a high-performance (HP) backup core with high power consumption, and two relatively low performance (LP) primary core with low power consumption. We consider a real-time application ($\mathcal{A}$), which consists of a set of n real-time tasks $T = \{T_1, T_2, ..., T_n\}$. Each task T_i ($1 \leq i \leq n$) is logically decomposed into a mandatory part, with execution requirement of M_i to be finished within deadline, d_i and an optional part with an execution requirement of O_i.

In a heterogeneous system, as different cores are operating at different frequencies, the same task may require different execution times on each of these cores. Assuming both the cores are operating at their highest frequencies (denoted by $f^{LP}_{max}, f^{HP}_{max}$, respectively), we define the temporal resource demand of a task T^{HP}_i on HP core as by the tuple $< M^{HP}_i, O^{HP}_i, d_i >$ and similarly, for LP core, this will be denoted as follows: T^{LP}_i: $< M^{LP}_i, O^{LP}_i, d_i >$.

B. Power Model

Power consumption of a processor can be divided in two parts, i. static power consumption (idle power) and ii. dynamic power consumption. Let us assume, Pow^{LP}_{idle} and Pow^{HP}_{idle} denote the static power consumption of LP and HP cores, respectively. If a processor executes task T_i, then the dynamic power consumption can be measured as $P_i(f) = a_i f^3 + \alpha_i$, where a_i indicates the switching capacitance, f denotes the processing frequency , and α_i is the frequency-independent power consumption [10]. *EnSuRe* employs the Dynamic Power Management (DPM) technique on both cores to minimize the energy consumption. Hence, as soon as *EnSuRe* finds any idle core, it attempts to bring the core into a low power state through DPM. However, during this transition period, a certain amount of energy and time are consumed. For simulation purposes, we assume these factors are negligible. However, for implementation on ZYNQ platform this issue has been considered. The total energy consumption within a scheduling length is calculated by summing up the energy consumption of each indvidual core.

C. Fault and recovery Model

EnSuRe utilizes both cores for fault recovery. The LP cores will be used as primary core where tasks will be executed by default and the HP core will be treated as backup core, which will only be activated to re-execute any faulty tasks of primary processor. Hence, each task T_i will have two versions i.e. primary copy (to be executed on LP cores) and backup copy (to be executed on HP core). Like existing fault tolerant mechanisms, we also assume that the fault detection overhead has been incorporated into the WCETs of tasks [2], faults are detected at the end of a task's mandatory part and optional part execution through the sanity (or consistency) checks (e.g. parity or signature checks) [3].

It has been assumed that mandatory portion of primary version of each task suffers from one transient fault in the scheduling window (defined in later Section).

D. Problem description

Given a set of real-time tasks to be executed on a heterogeneous multiprocessor system, devise a scheduling strategy such that 1) Total k number of faults are tolerated within the scheduling window 2) All tasks meet their respective deadlines 3) System accuracy is enhanced and 4) Strategy remains energy efficient.

III. Proposed Approach: *EnSuRe*

A. Schedule generation phase

EnSuRe employs *a time-partitioning based* scheduling approach for a set of n real-time tasks $\mathcal{A} = \{T_1, T_2, \ldots, T_n\}$ on the multiprocessor system. The technique maintains time denoted by the deadlines of the tasks. The difference between any two consecutive deadlines (say, the η^{th} and $(\eta-1)^{th}$ task deadline) is referred to as *"time-window"* TW_η and TWL_η denote the length of the η^{th} time-window TW_η and can be calculated using equation 1:

$$TWL_\eta = d_\eta - d_{\eta-1} \tag{1}$$

Each task T_i in $\mathcal{A}$ has a stipulated execution rate demand defined by its $weight$, $wt_i = \frac{M_i}{d_i}$, where M_i denotes the mandatory execution requirement and d_i denotes its deadline. For any time-window (TW_η) of duration TWL_η, each task T_j is allocated a workload-quota (Qu^η_j time-slots) proportional to its weight that can be calculated as:

$$Qu^\eta_j = (\lceil wt_j \times TWL_\eta \rceil) \quad \forall T_j \in \mathcal{A} \tag{2}$$

It is noted that within a time-window (say, TW_η), as all the available primary core(s) will operate in parallel, the total system-wide capacity for that time-window is: $TWL_\eta \times m_{pri}$,

where m_{pri} is the number of available primary core. In order to obtain a feasible schedule, this system-wide capacity must compensate the sum of workload-quota of all tasks, i.e. $(\sum_{j=1}^{n} Qu_j^{\eta})$. Thus, a necessary condition for scheduling to be feasible within TW_{η} is:

$$\sum_{j=1}^{n} Qu_j^{\eta} \leq TWL_{\eta} \times m_{pri} \tag{3}$$

EnSuRe selects tasks and attempts to allocate them starting from the first primary core, as per their workload-quota (Qu_j^{η}). However, the combined sum of task workload-quota in the core should be less than the time slice interval TWL_{η}. The available slack AS_i^{η} of the i^{th} primary core for the η^{th} time-window after finishing the allotted workload-quota can be calculated as:

$$AS_i^{\eta} = TWL_{\eta} - \sum_{j=1}^{n} Qu_j^{\eta} \tag{4}$$

According to our strategy, this available slack will be utilized for the execution of optional portion of tasks so that the system accuracy can be enhanced. In order to allocate the optional portion of tasks within a time-window, we have defined a factor called "Urgency Factor (UF)", the urgency factor (UF_i) of task T_i can thus be defined as:

$$UF_i = d_i - t^{slack} \tag{5}$$

where t^{slack} denotes the time instant where the slack time starts within a time-window. After calculating the UF_i value for each task within the time-window, we will store tasks based on their UF value in ascending order. Hence, it can be noted that tasks with a closer deadline will be selected first. This will increase the probability that within a deadline a task will complete the entire mandatory portion and will attempt to maximise the execution of optional parts to enhance accuracy.

B. Implication of the time-partitioning strategy of EnSuRe

In [12], the authors employed EDF scheduling scheme two schedule primary version of tasks on two primary processors. However, in such scenario, a time-partitioned approach provides better resource utilisation than existing EDF scheduling. We will now exhibit the efficacy of time-partitioning strategy via an example.

Let us consider 3 periodic real-time tasks $\{T_1, T_2, T_3\}$ with weights $\frac{9}{10}, \frac{9}{10}, \frac{4}{20}$. Now, we will try to schedule these tasks using EDF and *EnSuRe*, respectively on two main processors (denoted as V_1 and V_2). EDF will consider tasks with the earliest deadlines and it can be observed in Figure 1, EDF allocates T_1 and T_2, as they both share an earliest deadline of 10. So T_3 can be activated the earliest at the 9^{th} time-unit. However, this will leave one processor empty which can thus be utilised for optional part execution. It can also be observed that the remaining 3 units of T_3 can not be completed by the 20^{th} time-unit because T_1 and T_2 will again appear at 10 and consume (9+9)= 18 units. Thus, T_3 will miss its deadline.

On the other hand, *EnSuRe* maintains proportional fairness inside each time-window. We can develop the entire schedule

Algorithm 1: *EnSuRe*

Input: Temporal parameters of tasks $\in \mathcal{A}$ and time-windows;
Output: Generate fault-tolerant schedule for the application
for *each time-window* TW_{η} **do**
  /***** For primary core(s), Schedule generation *********/
  Calculate Qu_j^{η} for each task using Equation 2;
  if equation 3 **NOT** satisfied **then** RETURN;
  while $\mathcal{A} \neq NULL$ **do**
    Execute task T_j in the primary core(s) for Qu_j^{η} time; Remove T_j from $\mathcal{A}$ if $Qu_j^{\eta} == 0$;
  Determine Available Slack (AS_j^{η}) using Equation 4;
  Calculate UF_j for each task T_j using Equation 5;
  Store the UF values in ascending order in set $\mathcal{U}$;
  while $AS_i^{\eta} \neq NULL$ *OR* $\mathcal{U} \neq NULL$ **do**
    Execute optional portion of $T_j \in \mathcal{U}$;
  /***** For backup core, fault handling *********/
  If Tasks are schedulable **then**
  Create *backup* list in non-increasing order of M_i^{HP};
  for *first* k *tasks in backup do* **do**
    BES = *BES* + M_i^{HP};
  BST= TWL_{η} - BES;
  Reserve *BES* unit of slots on HP from BST instant;

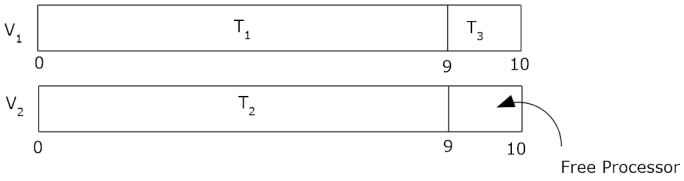


Fig. 1: EDF based schedule

into two time-windows. In each time-window *EnSuRe* will execute tasks as per their allotted work-load quota and properly utilising resources. The feasible schedule with *EnSuRe* has been shown in Figure 2. It can be observed that all tasks can be successfully scheduled by *EnSuRe*.

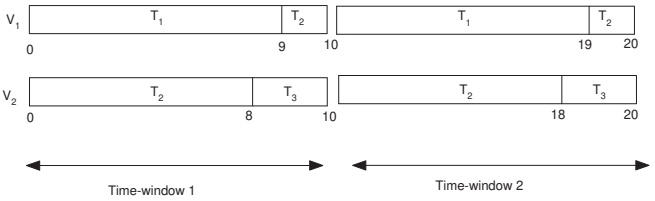


Fig. 2: Time-partition based schedule (*EnSuRe*)

C. Fault handling phase

After scheduling, *EnSuRe* creates a list called *"backup"* in non-increasing order of M_i^{HP}. As *EnSuRe* needs to handle only k number of faults, it reserves an execution slot on HP for possible backup task execution. We termed this slot as *"BES (Backup Execution Slot)"*. BES contains the execution slot for the k tasks (from the beginning) in *backup* list as per their M_i^{HP}. Then *EnSuRe* decides when to activate this *"BES"* slot

inside a time-window. Thus, the *"BST (Backup Start Time)"* is calculated. The concept behind this *BST* calculation is to activate the *BES* slot on the HP as late as possible, in order to save energy.

Dynamic Adjustment of *BES*: when a mandatory portion of a primary task finishes its execution, the fault detection mechanism is executed. If it is found that the task is executed with zero error, then the result is committed. This in turn, removes the task from the *backup* list. Hence, as soon as a primary task completes successfully, the size of the*"BES"* slots on the HP core reduces dynamically. The backup tasks will only be executed, if a fault is detected on LP primary core. Algorithm 1 shows the pseudocode of *EnSuRe* .

IV. Illustration with Example

Let us assume a system consisting of a set of four real-time tasks T_1, T_2, T_3 and T_4 to be executed on a LP primary core and a HP backup core. As shown in [3], this system is characterized by assuming, f_{max}^{LP} = 0:8; f_{max}^{HP} = 1.0; P_{idle}^{LP} = 0.02; P_{idle}^{HP} = 0.05, $a_i^{HP} = 1.0$, $\alpha_i^{HP} = 0.1$, $a_i^{LP} = 0.3$, $\alpha_i^{LP} = 0.03$. The task's parameters on the LP primary cores are as follows: T_1^{LP} = $< 12, 6, 60 >$, T_2^{LP} = $< 14, 6, 60 >$, T_3^{LP} = $< 15, 10, 90 >$, T_4^{LP} = $< 18, 10, 90 >$. The length of the first time-window is $TWL_1 = 60$ (earliest task deadline = 60). The length of the second time-window becomes $TWL_2 = 90 - 60 = 30$. In this example, we have illustrated the task allocation performed by *EnSuRe* for the first time-window only. In the first time-window, the workload-quota for each task can be determined by equation 2 and T_1 through T_4 will have workload-quota as: $Qu_1^1 = Qu_4^1 = 12$, $Qu_2^1 = 14$, $Qu_3^1 = 10$, respectively. It can be observed that Equation 3 is satisfied. Figure 3 shows the schedule generated by *EnSuRe* in time-window TW_1. After the allotment, we can observe that the LP core has an available slack (AS) of 12 time unit.

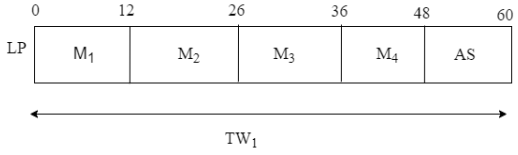


Fig. 3: Allocation of tasks on LP primary processor

Now, *EnSuRe* allocates optional parts of tasks T_1 and T_2, respectively as show in Figure 4.

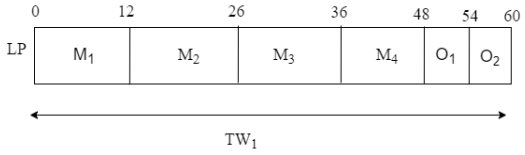


Fig. 4: Allocation of optional parts utilising slack

The task's parameters on the HP secondary cores are as follows: T_1^{HP} = $< 8, 4, 60 >$, T_2^{HP} = $< 10, 4, 60 >$, T_3^{HP} = $< 12, 6, 90 >$, and T_4^{HP} = $< 14, 6, 90 >$. Let us assume, $K = 2$ i.e two faults to be tolerated, In the *backup* list, tasks will be stored in non-increasing order based on their M_i value and *backup* can be denoted as: $\{M_4^{HP}, M_3^{HP}, M_2^{HP}, M_1^{HP}\}$. As $k = 2$, *EnSuRe* will reserve backup slot (*BES*) of units of 26 units (execution requirements in worst case), as shown in Figure 5. This configuration consumes energy of $80.8\ mJ$.

It can be observed that if *EnSuRe* uses the HP core as primary and the LP core as spare, then for this task set *EnSuRe* would consume $84.37\ mJ$. As *EnSuRe* always attempts to fully utilise the primary processor to increase the accuracy and thus, HP will remain fully occupied.

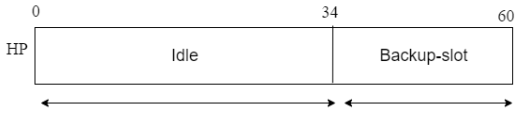


Fig. 5: Backup slot adjustment on HP spare core

V. Experiments an Analysis

A. Experimental Setup

Performance evaluation of the proposed *EnSuRe* has been carried out through a comprehensive set of simulation based experiments considering, real-time tasks and fault injection framework. *Normalized Energy Consumption (NEC)* and *Normalized Achieved Accuracy (NAA)* have been used for evaluation. NAA can be defined as the ratio between *total executed optional portion* and *total available optional portions for all tasks*. The simulated architecture is using a high-performance core with normalized frequency f_{max}^{HP} = 1.0 and a low-power core with normalized frequency f_{max}^{LP} varying in the range [0.6; 0.9], as shown in [3].

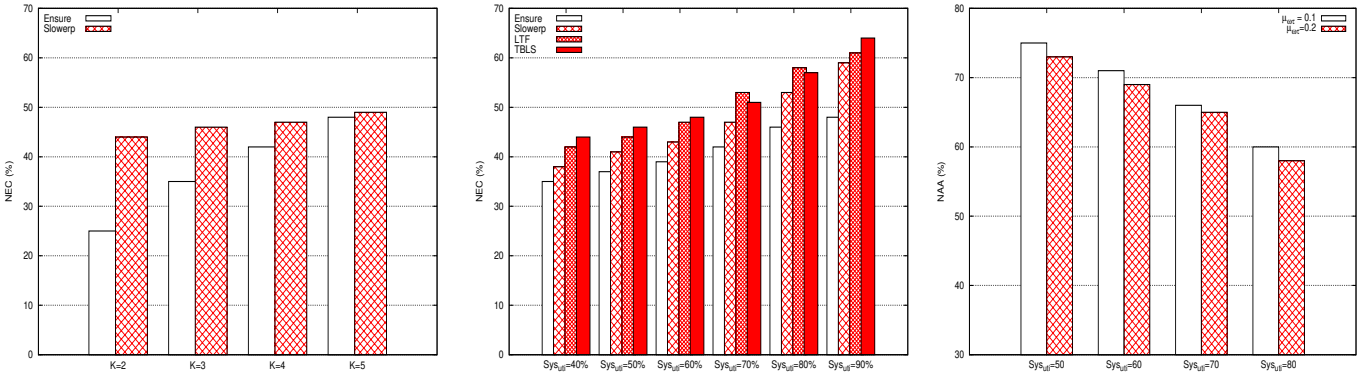
Task's Characteristic: The ranges of the mandatory portion M_i and the optional portion O_i are obtained from [1]. Tasks can consume between 4×10^7 and 6×10^8 clock cycles. The weights ($wt_i = \frac{M_i}{d_i}$) of the tasks have been taken from normal distribution with standard deviation $\sigma_{wt} = 0.1$ and two different values of mean, $\mu_{wt} = 0.1$, $\mu_{wt} = 0.2$. Task deadlines have also been generated from a normal distribution. Given the tasks weights, we can obtain the total workload of the system (Sys_{WL}) by summing up the weights of all the tasks. Given the system workload, the total system utilisation (Sys_{uti}) can be derived by:

$$Sys_{uti} = \frac{Sys_{WL}}{m_{pri}} \times 100\% \tag{6}$$

For a given the system utilisation (Sys_{uti}), the average number of tasks (ρ) can be achieved as:$\rho = \frac{Sys_{uti} \times m_{pri}}{100 \times \mu_{wt}}$ For simulation, we have generated various types of data sets by setting different values for the following parameters:

1) *Average individual task weight:* It has been obtained by the mean of the distribution from which task weights have been generated. Two values of μ_{wt}, 0.1 and 0.2 have been considered.
2) System Utilisation Sys_{uti}: We have varied the system utilisation Sys_{uti} value from 40% to 90%.
3) Number of faults k: k has been varied in the range [1,5].

In heterogeneous systems, a particular task may consme different execution times and power based on the processor



(a) Impact of number of faults (b) Impact of system-utilisation (c) NAA (%) varying μ_{wt}

Fig. 6: Performance of EnSuRe

characteristics. Hence, as shown in [3], we define a time-scaling factor $tscale_i = \frac{C_i^{LP}}{C_i^{HP}}$, and a power-scaling factor $pscale_i = \frac{P_i^{LP}}{P_i^{HP}}$ for each task T_i. The values of $tscale_i$ and $pscale_i$ are randomly generated within the ranges $1.4 \leq tscale_i \leq 2.3$ and $1.4 \leq 1/(tscale_i \times pscale_i) \leq 2.1$

B. Results and Analysis

1) Evaluating the impact of k: Figure 6(a) exhibits how energy consumption varies with increasing number of faults. Here, $f_{max}^{HP} = 1.0$ and $f_{max}^{LP} = 0.8$ and Sys_{uti} remains fixed at 70% on the power-efficient LP core and average individual weight remains $\mu_{wt} = 0.1$. As per the trends in Figure 6(a), it can be concluded that the higher the number of faults, the higher is the energy consumption for *EnSuRe*. However, *SlowerP* [10] consumes a fixed energy consumption. This behavior of *SlowerP* can be argued by the fact that irrespective number of faults, this strategy keeps a backup space for all tasks. In contrast, for *EnSuRe* as k increases the *BES* also increases which in turn increases overall power consumption.

2) Evaluating the impact of utilisation: Figure 6(b) shows how the energy consumption varies with respect to varying system utilisation. The number of faults set as $k = 4$. It may be observed from Figure 6(b) that with the increasing system utilisation, the energy consumption also increases for both *EnSuRe* and *SlowerP* provided the individual task weight remains the same. This is because for a given μ_{wt}, higher values of Sys_{uti} result in a higher number of tasks (ρ), resulting in the LHS ($\sum_{j=1}^{\rho} Qu_j^{\eta}$) of equation 3 to become larger. Due to this, the probability of failure of the condition (equation 3) increases for a given number of faults. Higher task number also reduce the idle times of both cores and hence, results in higher energy consumption. However, in all system utilization, *EnSuRe* outperforms *SlowerP*. This is because, *EnSuRe* reserves a fixed amount of backup slots on HP core based on k, while on other hand *SlowerP* employs a rigid strategy by reserving backup slots for each task.

We have further compared *EnSuRe* with two existing strategies "LTF" and "TBLS" as proposed in [3]. "LTF" means largest task first, as it can be observed tasks with higher execution length is given higher priority thus, in order to maintain deadline, the HP core is also used for primary execution which leads to high energy consumption. "TBLS" is threshold based list scheduling, in this technique tasks will be allocated to LP core upto a certain utilisation and then it will be allocated to HP core. Similarly, in this technique, the HP core is completely utilised for primary as well as backup execution and thus, it consumes higher energy. It can be observed that in case of highest system utilisation (Sys_{uti}=90%), *EnSuRe* consume 25% less energy than "TBLS".

From Figure 6(c), *EnSuRe* is able to achieve 75% accuracy when Sys_{uti} is 50%. However, as the utilisation increases the slack in primary core(s) decreases and thus, NAA decreases with the increase in Sys_{uti}. It has to be noted that for a Sys_{uti}, if the average individual task weight (μ_{wt}) varies from 0.1 to 0.2, the NAA remains comparable. This phenomena exhibits the robustness of *EnSuRe* irrespective of task's weight. The "time-partioning" is the key reason behind such robustness because within each time-window "EnSuRe" maintains fairness by executing tasks based on work-load quota.

VI. Hardware Implementation

A. Architectural Setup

We have implemented *EnSuRe* on a heterogeneous system on a Xilinx Zynq-7000 All-Programmable SoC [13], with Arm Cortex-A9 CPU in the Processing System (PS) side, which serves as the HP core; and FPGA fabric in the Programmable Logic (PL) side, which is used to implement the LP core and other system components. Figure 7 shows the diagrammatic representation of the proposed architecture. The LP core utilised a TMR MicroBlaze. The Memory Arbiter is a combination of AXI memory interconnects interfacing an AXI CDMA module with a DDR memory, the LP core, and the HP core. The Mailbox and Mutex are for coordinating communication and signalling between the HP and LP subsystems. Specifically, the Mailbox is for transactional communication between the HP and LP while the Mutex is used to prevent conflict in access to shared resources. The

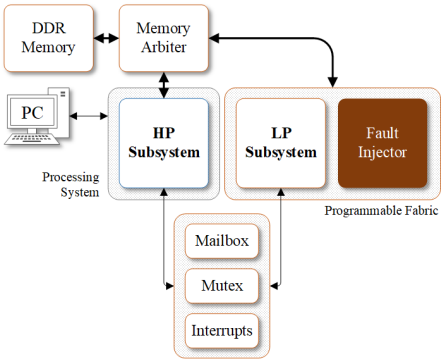


Fig. 7: The ZYNQ test-bed

signalling of switch-over from LP to HP core is via interrupts. For power management, we implement a Dynamic Power Manager (DPM) that is able to control the power consumption of the system dynamically. The backup subsystem is always held in a low-power state by dynamically scaling down the CPU frequency and clock-gating system modules. This is a software-driven solution that requires setting register values in the PS. A processor reset (watchdog-triggered reset) is then used to force the processor to exit from the standby condition. The host PC executes the *EnSuRe* algorithm.

B. Fault Injection and Detection Framework

The fault injection framework needed to confirm the integrity of the TMR MicroBlaze Subsystem relies on the *TMR Inject* IP core. Fault injection is actually carried out by injecting a different instruction at a certain instruction address of one of the three processors. This causes a mismatch among the processors and such mismatch is detected by a TMR comparator. To inject a fault in one of the three processors, the software writes the instruction, address and CPU ID to the TMR Inject core. We then check that the expected comparator mismatch has occurred by reading the TMR Manager First Failing Register at address offset 0x04. We prevent the TMR Manager from mitigating the injected fault by writing to the TMR Manager Comparison Mask Register. The framework is shown in Figure 8.

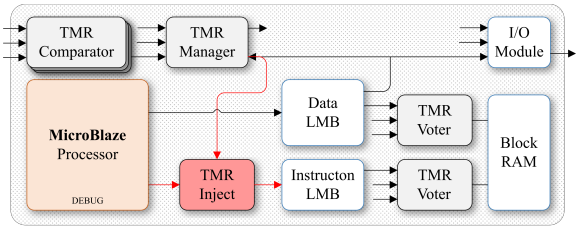


Fig. 8: Fault injection and detection

C. Resource consumption

The architecture is implemented on the ZedBoard, which is a Znyq-7000 board with the XC7Z020-CLG484-1 chip. The entire architecture utilizes 38.94% of the available FPGA slices. Table I gives the resource utilisation in the architecture.

TABLE I: Resource Utilisation of key components

Module	Utilisation (%)		Utilisation (%)	
	Flip Flops	LUTs	Flip Flops	LUTs
TMR MicroBlaze	9496	15049	8.92	28.37
Mutex	92	74	0.091	0.14
Mailbox	263	4.14	0.19	0.49
Total	9787	15431	9.20	29.01

D. Energy consumption

We have created synthetic tasks from MiBench benchmark [1]. The execution times for HP core and LP core are measured for ARM core (freq: 650 MHz) and MicroBlaze core (freq: 100 MHz). We have evaluated the *EnSuRe* by injecting ($k = 3$) faults.The average scheduling length is taken as 30000 ms and we executed the simulations 5 times by injecting the faults at arbitrary positions in the scheduling length. The final value is calculated from the average of these obtained values. Based on the power report of Vivado tool, ARM works as the secondary core and with the aid of DPM, ARM cores are powered down by reducing their frequency of operation to 50 MHz and consumes 0.420 watt. However, the primary MicroBlaze operates at 100 MHz and consumes 0.123 watt. Table II shows the energy consumption of *EnSuRe* and SlowerP for the entire scheduling length. It can be observed that the results obtained through software simulation are aligned with the hardware implementation outcomes.

TABLE II: Enrgy Consumption in Joule

Avg. number of tasks	EnSuRe	SlowerP
8	7.83	11.26
12	9.68	14.57
16	13.58	17.84

VII. Conclusion

In this paper, we have presented a fault-tolerant scheduling strategy, *EnSuRe* for real time tasks executing on a heterogeneous cores. We presented "time-partitioned" based scheduling scheme for allocation and execution of tasks to the available primary processor such that tasks could meet their deadlines and accuracy can also be enhanced. Next, our proposed intelligent technique to dynamically adjust the backup execution slot on spare processor, provides less energy consumption and tolerance against fixed number of transient faults. As per the obtained simulation behavior, it can be argued that *EnSuRe* can be employed for energy efficient operation and the simulation outcomes were further validated on ZYNQ APSoC heterogeneous systems with benchmark tasks.

References

[1] L. Mo, A. Kritikakou, and O. Sentieys, "Approximation-aware task deployment on asymmetric multicore processors," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1513–1518.

[2] Y. Guo, D. Zhu, H. Aydin, J.-J. Han, and L. T. Yang, "Exploiting primary/backup mechanism for energy efficiency in dependable real-time systems," *Journal of Systems Architecture*, vol. 78, pp. 68–80, 2017.

[3] A. Roy, H. Aydin, and D. Zhu, "Energy-efficient fault tolerance for real-time tasks with precedence constraints on heterogeneous multicore systems," in *2019 Tenth International Green and Sustainable Computing Conference (IGSC)*. IEEE, 2019, pp. 1–8.

[4] P. P. Nair, R. Devaraj, and A. Sarkar, "Fest: Fault-tolerant energy-aware scheduling on two-core heterogeneous platform," in *2018 8th International Symposium on Embedded Computing and System Design (ISED)*. IEEE, 2018, pp. 63–68.

[5] A. Majumder, S. Saha, and A. Chakrabarti, "Task allocation strategies for fpga based heterogeneous system on chip," in *IFIP International Conference on Computer Information Systems and Industrial Management*. Springer, 2017, pp. 341–353.

[6] J. Zhou, K. Cao, P. Cong, T. Wei, M. Chen, G. Zhang, J. Yan, and Y. Ma, "Reliability and temperature constrained task scheduling for makespan minimization on heterogeneous multi-core platforms," *Journal of Systems and Software*, vol. 133, pp. 1–16, 2017.

[7] M. A. Haque, H. Aydin, and D. Zhu, "On reliability management of energy-aware real-time systems through task replication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 3, pp. 813–825, 2016.

[8] M. Fan, Q. Han, and X. Yang, "Energy minimization for on-line real-time scheduling with reliability awareness," *Journal of Systems and Software*, vol. 127, pp. 168–176, 2017.

[9] B. Zhao, H. Aydin, and D. Zhu, "Energy management under general task-level reliability constraints," in *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*. IEEE, 2012, pp. 285–294.

[10] A. Roy, H. Aydin, and D. Zhu, "Energy-aware standby-sparing on heterogeneous multicore systems," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2017, pp. 1–6.

[11] R. M. Pathan, "Real-time scheduling algorithm for safety-critical systems on faulty multicore environments," *Real-Time Systems*, vol. 53, no. 1, pp. 45–81, 2017.

[12] Y. Guo, D. Zhu, and H. Aydin, "Generalized standby-sparing techniques for energy-efficient fault tolerance in multiprocessor real-time systems," in *2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, 2013, pp. 62–71.

[13] L. Crockett, D. Northcote, C. Ramsay, F. Robinson, and R. Stewart, *Exploring Zynq MPSoC: With PYNQ and Machine Learning Applications*, 2019.