
Article:
Ma, Yunfeng and Soares Indrusiak, Leandro (2017) Hardware-accelerated analysis of real-time Networks-on-Chip. Microprocessors and Microsystems. pp. 81-91. ISSN: 0141-9331

https://doi.org/10.1016/j.micpro.2017.06.011

Hardware-accelerated analysis of real-time Networks-on-Chip

Yunfeng Ma and Leandro Soares Indrusiak
Department of Computer Science, University of York
Email: {ym608, leandro.indrusiak}@york.ac.uk

***Abstract*—A real-time Network-on-Chip (NoC) must guarantee that it is able to execute a set of tasks and deliver the communication packets that they generate, all within the respective deadlines even under a worst-case scenario. End-to-End Response Time Analysis (E2ERTA) is a mathematical formulation that can be used to test whether a particular NoC configuration is able to guarantee the timely execution of tasks and delivery packets. The complexity of E2ERTA calculation increases with the increase of the number of tasks and packet flows, and with the core count of the NoC. This paper presents an approach to accelerate E2ERTA calculations through the use of custom hardware and efficient implementation of its mathematical operations. We explore the performance of the proposed approach, and analyse its effectiveness against the state-of-the-art in the field. The results show a significant improvement in testing NoC guarantees, thus potentially enabling the use of E2ERTA as a fast and guaranteed deterministic admission controller for open and dynamic real-time systems. As a case-study, we integrate the proposed approach to a NoC optimisation framework aiming to accelerate the search for NoC configurations that meet all the NoC's hard real-time requirements.**

I. Introduction

Networks-on-chip (NoCs) can be designed to guarantee performance of real-time applications. Most of them use architectural features such as time-division multiplexing of links (TDM) [12], traffic regulators or virtual channels (VCs) [3]. A number of analytical techniques have been proposed to evaluate whether a specific NoC configuration can guarantee the performance of a specific application. Kiasari et al review them in [8]. In this paper, we focus on wormhole-switching NoCs with priority-preemptive VCs, which can be analysed using schedulability analysis [13]. More specifically, we use end-to-end response time analysis (E2ERTA) because it performs schedulability analysis of tasks running on NoC cores as well as packets flowing through NoC links, and is able to predict whether all tasks and packet flows can meet their deadlines even under a worst-case scenario.

Increases in the number of cores and links in NoCs, as well as in the complexity of applications (i.e. increasing number of tasks and communication flows), make E2ERTA calculation significantly harder. This cost is not critical if one is interested in evaluate the schedulability of a system during design time, in what is referred as the in static task allocation problem. However, the execution time of applying E2ERTA can be vital in other areas, such as in dynamic admission controllers. Such controllers are used to decide, during runtime, whether a system can successfully admit new applications without jeopardising the timeliness of previously admitted ones. Longer analysis directly increases waiting time before an admission decision can be made. Therefore, whether the computation time of E2ERTA can be reduced and the magnitude of such reduction are important issues.

The goal of this paper is to explore the possibility of applying custom parallel hardware to reduce the computation time of E2ERTA. It details and extends the approach first introduced in [15], using a hardware implementation of E2ERTA to enhance its timing performance and considering two variations of E2ERTA which accelerate it even further while providing less tight results. The performance of the hardware accelerated implementations are compared against a software-only baseline implementation of E2ERTA presented in [14]. Furthermore, we present a novel case study showing the benefits of integrating the hardware-accelerated implementation of E2ERTA into a NoC task mapping optimisation framework.

The paper is organised as follows: Section II reviews the related work and is followed by the system model in Section III; in Section IV, the problem will be discussed. The proposed hardware architecture and implementation are presented in Section V; the experimentation platform and results analysis are listed in Section VI. Section VII presents a case study with the integration of the hardware-accelerated E2ERTA into an optimisation framework. The paper is then closed with conclusions and future work.

II. Related Work

A. Schedulability test for priority-preemptive NoCs

Classic response-time analysis for fixed-priority tasks running on a single processor was first introduced in Liu and Layland's seminal paper [9]. Numerous extensions to that analysis were published over the past decades, considering for instance release jitters, offsets and multiple processors.

In NoCs, network links are shared by various packet flows. In [13], the authors modelled the links and flows as shared processors and tasks, respectively, and extended classic response time analysis to obtain the worst-case communication delay of each packet flow. In [7], the classic analysis from [9] and the analysis from [13] were combined to cover a task's end-to-end latency, which includes not only its computation time but also the time it takes for its packets to reach their destination. We refer to that analysis as E2ERTA, and we use it to test whether a set of communicating sporadic tasks is

end-to-end schedulable on a NoC, i.e. all their computations and communications finish by the respective deadlines.

B. Speed-up methods for response time analysis

The calculation of response time analysis is based on an iterative calculation. Since this iterative calculation needs an arbitrary number of iterations to compute the final results, the efficiency of the response time analysis is low. Therefore, Bini and Baruah [2] presented a pre-check metric to avoid the exact result computation in order to reduce the running time of the response time analysis. Besides, in [4], Davis et al. presented a lower bound of worst-case response time to reduce the number of iterations needed when executing the worst-case response time analysis. In [14], both these speed-up methods were combined with E2ERTA to improve its efficiency in a software implementation.

III. System Model

The timing performance of a NoC system can be evaluated by the worst-case end-to-end response times of its tasks. This means the time between the release of a task on its processor and the reception of the last flit of its longest packet by the destination processor, under the worst-case situation [7]. For a task to be schedulable on a NoC, its end-to-end response time has to be less than or equal to its deadline even under the worst-case situation.

A. NoC platform model

The configuration of a NoC can be presented by several parameters such as topology, routing algorithm, flow control, arbitration, and switching techniques. These can affect the structure of NoCs and further influence the performance. In this paper, we focus on NoC platform which has:

- mesh topology;
- XY routing algorithm;
- virtual channels and credit-based flow control;
- fixed-priority arbitration;
- wormhole switching.

B. Sporadic communication task model

Since the E2ERTA can be divided into tasks' response time analysis and flows' response time analysis, we need to model these two parts separately.

1) Task Model: Following the system model of tasks' response time analysis in the paper [1], the tasks can be modelled as $Task_i = \{c_i, t_i, p_i, d_i, r_i\}$.

- c_i is the worst-case computation time of $Task_i$;
- t_i is the period of $Task_i$;
- p_i is the priority of $Task_i$;
- d_i is the deadline of $Task_i$;
- r_i is the response time of $Task_i$;
- B_i is the maximum blocking time of $Task_i$;
- $lep(k)$ is the set of tasks with the priority lower than or equal to $Task_i$;
- $hp(i)$ is the set of tasks with higher priority than $Task_i$;
- u_i is the utilization of $Task_i$, it equals to $\frac{c_i}{t_i}$.

2) Flow Model: According to the schedulability analysis in the paper [13], the traffic flows can be presented as: $Flow_i = \{C_i, T_i, P_i, D_i, J_i^R, J_i^I, R_i, S_{id}, S_{ii}\}$.

- C_i is the basic latency of $Flow_i$;
- T_i is the period of $Flow_i$;
- P_i is the priority of $Flow_i$;
- D_i is the deadline of $Flow_i$;
- J_i^R is the release jitter of $Flow_i$;
- J_i^I is the interference jitter of $Flow_i$;
- R_i is the response time of $Flow_i$;
- S_{id} is the direct interference set of $Flow_i$;
- S_{ii} is the indirect interference set of $Flow_i$;
- L_i is used to calculate C_i, if C_i is not given;
- U_i is the utilization of $Flow_i$, it equals to $\frac{C_i}{T_i}$.

The S_{id} and S_{ii} present the direct and indirect interference set of $Flow_i$. The flows in these two sets can affect the worst-case response time of $Flow_i$ by pausing $Flow_i$'s communication. The definitions of them are based on the relationship between $Flow_i$ and higher priority flows.

- The flows in the direct interference set:
  - having higher priority than $Flow_i$;
  - sharing at least one link with $Flow_i$.
- The flows in the indirect interference set:
  - having higher priority than $Flow_i$;
  - having no shared link with $Flow_i$;
  - interfering with the flows in the direct interference set of $Flow_i$.

Figure 1 illustrates an example with four traffic flows, with $P_1 > P_2 > P_3 > P_4$. In this example, the $Task_3$ *and* $Task_4$ are allocated to IP(8); $Task_2$ and $Task_1$ are allocated to IP(5) and IP(2) respectively. The direct interference set and indirect interference set for each $Flow_i$ are listed in Table I.

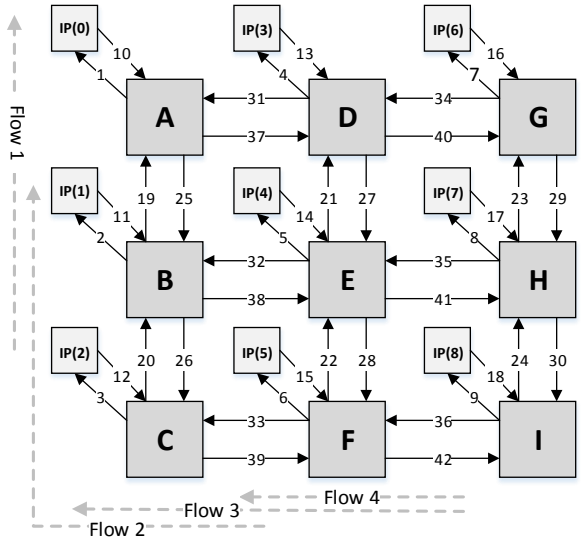

Fig. 1: Traffic Flow Relationship Example.

IV. Problem Statement

The complexity of calculation of E2ERTA is affected by two aspects, the characteristic of E2ERTA and the implementation method.

TABLE I: Traffic Flow Example

$Flow_i$	Direct interference set	Indirect interference set
$Flow_1$	$\{\phi\}$	$\{\phi\}$
$Flow_2$	$\{Flow_1\}$	$\{\phi\}$
$Flow_3$	$\{Flow_2\}$	$\{Flow_1\}$
$Flow_4$	$\{Flow_3\}$	$\{Flow_2\}$

A. Characteristic of E2ERTA

The E2ERTA can be divided into tasks and flows response time analysis. On each node, the tasks are released according to the priority order. Hence, the higher priority tasks can easily preempt lower priority tasks. This phenomenon can be seen in Figure 2 which follows the example of Figure 1 and considering the deadlines of all tasks are same and equal to period. Tasks 3 and 4 are released at the same time. However, as Task 3 has higher priority than Task 4, it can directly take the node and preempt the release of Task 4. Therefore the response time of tasks can be calculated by Equation 1.

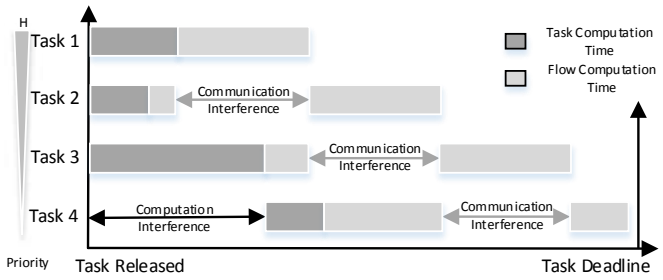

Fig. 2: E2ERTA Example.

$$r_i^{n+1} = c_i + \sum_{\forall j \in hp(i)} \lceil \frac{r_i^n}{t_j} \rceil c_j \tag{1}$$

Similarly, the higher priority flows can interrupt lower priority flows' transmission. In Figure 2, Tasks 2, 3 and 4 are all suffering the interferences from higher priority flows. The response time of flows can be calculated by Equation 2.

$$R_i^{n+1} = C_i + \sum_{\forall j \in S_{id}} \lceil \frac{R_i^n + J_j^R + J_j^I}{T_j} \rceil C_j \tag{2}$$

The authors in [7] assume that the release jitter of a traffic flow can be replaced by the worst-case response time of the initial task of the flow, that is $J_i^R = r_i$. Therefore, the E2ERTA can be written as Equation 3.

$$R_i^{n+1} = C_i + \sum_{\forall j \in S_{id}} \lceil \frac{R_i^n + r_j + J_j^I}{T_j} \rceil C_j \tag{3}$$

We could conclude the characteristic of E2ERTA from Equation 3 that the computation of E2ERTA is based on an iterative calculation. In the processing of this iterative calculation, a number of intermediate results are needed to be calculated before we can obtain the final results. The more intermediate results are required, the longer computation time will be cost. According to the requirement of E2ERTA, the termination condition of this iterative calculation is either $R_i^{n+1} = R_i^n$ *or* $R_i^{n+1} > D_i$. It means that for each calculation, the number of iterations is not fixed and consequently the number of intermediate results is not a constant. We make an assumption that we only consider the termination condition as $R_i^{n+1} = R_i^n$ and ignore $R_i^{n+1} > D_i$, since smaller D can terminate the iterative calculation early. Under this assumption, the lower priority a task has, the more number of intermediate results and more computation time it will suffer. Thus, the complexity of calculation of E2ERTA will be increased with the extended size of task set.

B. Implementation method

From the working process of the software version of E2ERTA (SW-E2ERTA) shown in Algorithm 1, we notice that before we can calculate the response time (line 6) by using Equation 3, we have to calculate the direct interference set (line 10) and indirect interference set (line 11). Additional computation processes such as routing (line 7), get task interference (line 3) and basic latency calculation of flows (line 9) should also be included if the E2ERTA starts from task mapping.

Algorithm 1 Software Version of E2ERTA Working Process

Input:
- Task Mapping,
- Task Information,
- Application Information.

Output:
- Response Time of Each Task,
- Response Time of Each Flow,
- Number of Unschedulable Tasks and Flows.

```
1:  function CALCULATE TASK RESPONSE TIME
2:      for each task, in priority order do
3:          Get Task Interference
4:          Get Task Response Time Analysis
5:          Store Results
6:  function CALCULATE FLOW RESPONSE TIME
7:      Perform Routing Algorithm
8:      for each flow, in priority order do
9:          Get Flow Basic Latency
10:         Get Direct Interference Set
11:         Get Indirect Interference Set
12:         Get Flow Response Time Analysis
13:         Store Results
```

Note: Task Mapping refers to Task Allocation. Task Information includes c,t,d. Application Information includes Initial Task, Destination Task and L.

Naturally, software implementation is not designed to support parallel computing. Hence, the next computing block can not start until the previous block has finished. However, some calculations of E2ERTA do not depend on each other. For example, the Task Response Time Analysis is not related to the partial blocks of Flow Response Time Analysis such

as Routing Algorithm, Get Flow Basic Latency, Get Direct Interference Set and Get Indirect Interference Set. In practice, the computation time of E2ERTA can be reduced if these blocks can be launched in parallel.

Besides, the efficiency of processing vector in software is low. For example, if we use the binary coding style to encode the results of Routing Algorithm, the results could be similar to what is shown in Figure 3a which follows the example in Figure 1 and others hidden links are not used and set as 0. To identify the relationship between $Flow_3$ and $Flow_4$, the SW-E2ERTA has to compare these two flows bit by bit. Even if we use integer coding style (an example presented in Figure 3b), the computation cannot be finished within one clock cycle. Similar phenomenon can also be found in Get Indirect Interference Set and Get Flow Basic Latency computation blocks. In addition, this phenomenon will become worse when the size of NoC increases, since larger size refers to more links and then results in more computation time. So, using the software method to implement E2ERTA suffers the limitation of computation time.

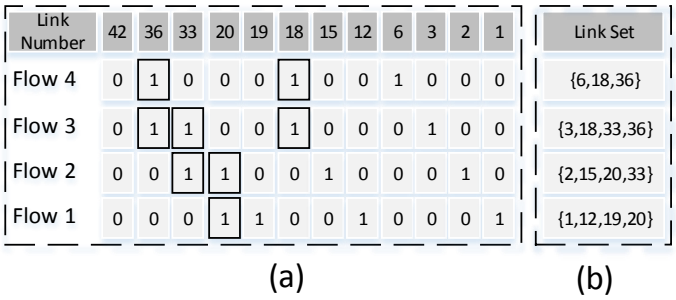

Fig. 3: (a) Binary Coding Example, (b) Integer Coding Example.

V. Hardware Implementation and Speed-Up components of E2ERTA

As analysed in the previous section, the characteristic of E2ERTA and the state-of-the-art implementation method cannot efficiently improve the computation time when the size of a task set and the size of a NoC are increased. In order to solve this problem, we discuss the possibility of using hardware method to implement E2ERTA and introduce a hardware architecture named as HW-E2ERTA. To alleviate the limitation of the characteristic of E2ERTA, we suggest two speed-up components which are Pre-Check (PRE) and New Lower Bound (NLB).

A. Hardware Implementation of E2ERTA

In the calculation process of E2ERTA, not all computation processes have to be launched sequentially. As discussed in Section IV, the Task Response Time Analysis and partial processes of Flow Response Time Analysis can be loaded simultaneously. Thus, we proposed a hardware implementation architecture of E2ERTA which has been shown in Figure 4. In this Figure, we could find that the Task and Flow Response Time Analysis can be released at the same time. In Flow

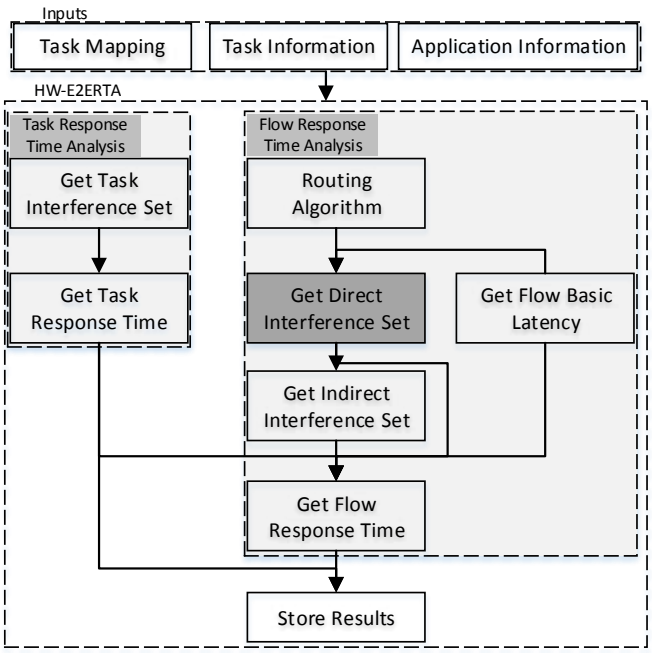

Fig. 4: HW-Architecture of E2ERTA.

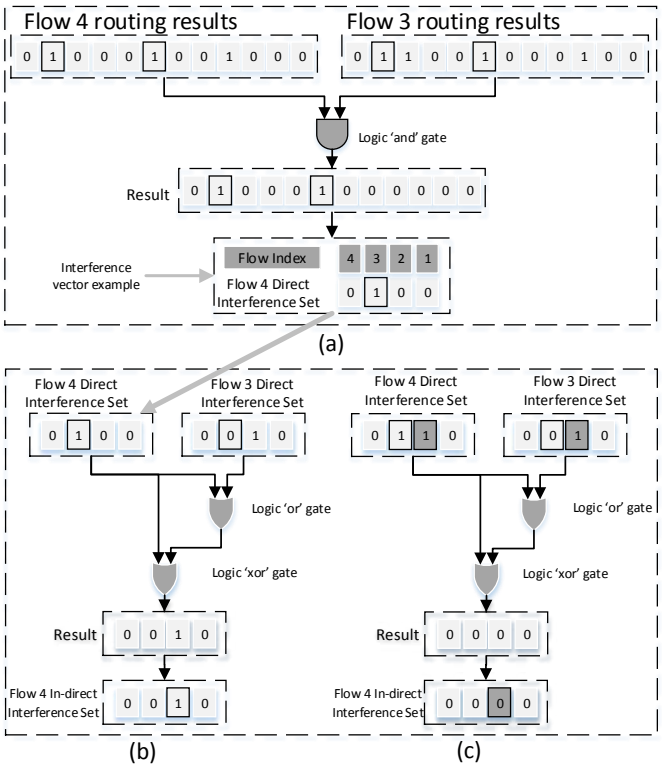

Fig. 5: (a) Example of Get Direct Interference Hardware Implementation Operation, (b) and (c) Examples of Get Indirect Interference Hardware Implementation Operation.

Response Time Analysis process, the two components Get Direct Interference Set and Get Flow Basic Latency can start in the meantime after they receive the results of Routing Algorithm. The Get Flow Response Time component will be launched when the Task Response Time Analysis, Get

Indirect Interference and Get Flow Basic Latency components are finished. Its results and the results from Task Response Time Analysis will be organised and stored.

As aforementioned, parallel computing is not the unique aspect that the software version fails to support. Vector operations are also the bottle-neck we need to make a breakthrough. We apply some logic gate operations to solve this problem. One example has been shown in Figure 5. In this Figure, the routing results are following the results in Figure 1. The width of interference vector is 4. An example has been shown in Figure 5a. The right end of the interference vector is $Flow_1$. The flow with value '1' ($Flow_3$) refers to this flow can interrupt the observed flow ($Flow_4$).

For Get Direct Interference Set component, we use the 'and' gate to identify the direct relationship between two flows, which has been shown in Figure 5a. We apply the logic 'and' operation between the routing results of $Flow_4$ and $Flow_3$. If these two flows have shared links which have been labelled in block rectangles, the result is not all zero. The relevant bit position is set as '1' in $Flow_4$'s Direct Interference Set. Otherwise, the result is all zero and the relevant bit is set as '0'.

When identifying the indirect relationship between two flows, we select the logic 'or' gate and the logic 'xor' gate, which has been described in Figure 5b and 5c to implement this operation. The example in Figure 5b inherits the sources from Figure 1, Figure 3 and Figure 5a. In this example, $Flow_4$'s Direct Interference Set is one of the inputs of 'or' gate and 'xor' gate. If $Flow_3$ can directly interrupt $Flow_4$ and its Direct Interference Set is not empty (can be preempted by other flows), $Flow_3$'s Direct Interference Set will be checked. Therefore, the other input is $Flow_3$'s Direct Interference Set. Similar to Get Direct Interference operation, if the result is not all zero, the relevant bit will be set as '1' to indicate the higher priority flow ($Flow_2$) which can indirectly interrupt $Flow_4$. However, if $Flow_2$ can preempt both $Flow_3$ and $Flow_4$, the result will remain as 0. This phenomenon has been illustrated in Figure 5c; $Flow_2$ has been labelled with dark gray.

B. Speed-Up Components of E2ERTA

As discussed in Section IV, the iterative characteristic of E2ERTA is a barrier for improving its efficiency. To alleviate the complexity of iterative calculation, the authors [2] presented a Pre-Check (PRE) method. They use the work load to find an upper bound of task's response time which has been shown in Equation 4. This can be used as a sufficient test for the schedulability test.

$$r_i^{ub} = \frac{c_i + \sum_{\forall j \in hp(j)} c_j(1-u_j)}{1 - \sum_{\forall j \in hp(j)} u_j} \tag{4}$$

Besides, the authors [4] explored this problem from a different view and pointed out a lower bound of response time of a task (referred as NLB). They use the lower bound to replace the original start value (usually is 0 or c). The results show fewer intermediate results and shorter computation time compared with original response time analysis. This lower bound can be found by using Equation 5, 6 and 7. The $I_j(R_{i-1})$ denotes the worst-case interference due to $Task_j \in hp(i)$ occurring during the response time of $Task_{i-1}$.

The authors [14] assembled these two ideas with E2ERTA in several schemes implemented on a software experimentation platform. Although the results show these schemes can improve the efficiency of E2ERTA, the abilities of these two ideas are not well explore due to the sequential natural of the software platform. Therefore, in hardware implementation, we can obtain better performance. However, fully implementing the NLB idea in hardware is complex. Thus, we select a compromised method.

$$I_j(R_{(i-1)}) = \left\lceil \frac{R_{(i-1)} + J_j}{T_j} \right\rceil C_j \tag{5}$$

$$\begin{aligned} R_i^{LB}(k) =& \frac{B_i + C_i + \sum_{\forall j \in lep(k) \cap hp(i)} I_j(R_{i-1})}{1 - \sum_{\forall j \in hp(k)} U_j} \\ &+ \frac{\sum_{\forall j \in hp(k)} J_j U_j}{1 - \sum_{\forall j \in hp(k)} U_j} \end{aligned} \tag{6}$$

$$R_i^{LB} = \max_{\forall k=1\ldots i} R_i^{LB}(k) \tag{7}$$

1) Proposed Upper Bound and Lower Bound: The calculation of E2ERTA includes a celling function. If we can release this celling function by using inequalities, the upper bound and lower bound of the response time of a task can be found. Here we modify Equation 3 and obtain the upper bound and lower bound listed as follows:

$$R_i \geq c_i + \sum_{\forall j \in S_{id}} \left\lceil \frac{R_i + r_j + J_j^I}{T_j} \right\rceil C_j \tag{8a}$$

$$R_i \leq c_i + \sum_{\forall j \in S_{id}} \left\lceil \frac{R_i + r_j + J_j^I}{T_j} + 1 \right\rceil C_j \tag{8b}$$

$$\Downarrow$$

$$R_i \geq \frac{c_i + \sum_{\forall j \in S_{id}} \left(r_j + J_j^I\right) U_j}{1 - \sum_{\forall j \in S_{id}} U_j} \tag{9a}$$

$$R_i \leq \frac{c_i + \sum_{\forall j \in S_{id}} \left[\left(r_j + J_j^I\right) U_j + 1\right] C_j}{1 - \sum_{\forall j \in S_{id}} U_j} \tag{9b}$$

2) Selected Upper Bound and Lower Bound: In tasks' response time analysis, the r_j *and* J_j^I are not existed and can be set as zero. Comparing with the Equation 4, the Equation 9b is pessimistic after setting r_j *and* J_j^I to zero. Therefore, we select Equation 4 as our upper bound of task's response time.

Move to lower bound, our proposed lower bound Equation. 9a is less tight than Equation 7. This is because Equation 7 select the maximum one from a series of lower bounds. Our result is one candidate in this series and may not the be the maximum one. However, the implementation of our lower bound is much easier than Equation. 5, 6 and 7. In addition, the inputs of Equation. 9a are the same as the inputs of 4. So combing these two Equations together can save additional resources and the upper bound and lower bound can be calculated simultaneously. Thus, we select Equation. 9a as our lower bound of task's response time. We can obtain

Equation 10a and b.

$$r_i^{lb} \geq \frac{c_i}{1 - \sum_{\forall j \in S_{id}} u_j} \tag{10a}$$

$$r_i^{ub} = \frac{c_i + \sum_{\forall j \in hp(j)} c_j(1 - u_j)}{1 - \sum_{\forall j \in hp(j)} u_j} \tag{10b}$$

In flows' response time analysis, the r_j *and* J_j^I are existed. We cannot directly select Equation 4 as our upper bound of flow's response time. In addition, considering the implementation complexity, the Equation 9a and b are similar with Equation 10. Partial components among them can be reused. This can further reduce the resources and implementation complexity. Therefore we select Equation 9a and b as the lower and upper bound of Flow's response time respectively.

C. Assembly Schemes

As a sufficient test, PRE cannot guarantee the calculation of response time analysis. It has to co-operate with other components like HW-E2ERTA or NLB. Here, we list some assembly schemes in Figure 6 which consists of four parts (a, b, c, and d). In (a) and (c), we put PRE, HW-E2ERTA or NLB in sequential order. If PRE has indicated the final response time of a task or a flow, the following HW-E2ERTA or NLB will be skipped. Otherwise, the HW-E2ERTA or NLB will be applied.

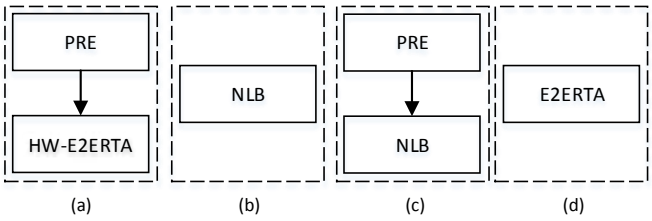

Fig. 6: Supported combinations (assemblies) of response time analyses.

VI. Experimentation Platform and Results Analysis

In this section, experimentation platform, experimentation configuration and results analysis will be discussed to show the performance of our proposed implementation.

A. Experimentation Platform

To evaluate the performance of HW-E2ERTA, PRE and NLB, we propose an experimentation platform which is an embedded system based on Xilinx VC707 develop board showing in Figure 7a. On this platform, we fully implement the SW-E2ERTA in the paper [14] on MicroBlaze. To gain an accuracy computation time, we introduce a hardware timer to measure the running time of SW-E2ERTA in the number of clock cycles.

The evaluations of HW-E2ERTA, PRE and NLB are also operated on this platform. We packet our hardware implementations as customer peripherals and mount them on an AXI bus

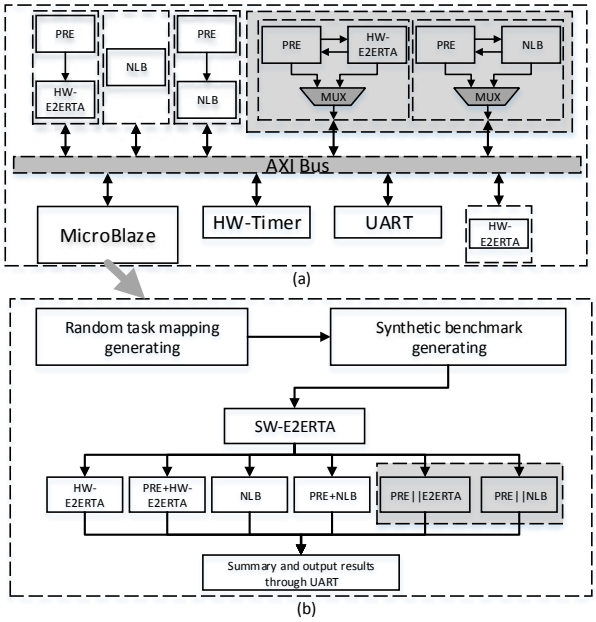

Fig. 7: (a) Experimentation Platform, (b) Testing Process.

Note: The blocks labeled in gray are parallelism implementation for future work among HW-E2ERTA and its accelerated components.

which is an on-chip interconnect link used in Xilinx system-on-chip design.

Each testing is started from the random testing data generating process and ended when all processes or components are tested. Figure 7b has shown the testing process. The MicroBlaze firstly generates testing data (a random task mapping and a Synthetic benchmark which include task information and application information). Then MicroBlaze launches SW-E2ERTA. When SW-E2ERTA has finished, MicroBlaze writes the testing data to each component and enables all of them simultaneously. After all tests have finished, the MicroBlaze collects data from each hardware component, and organises these results. The results are output through a UART port.

B. Experimentation Configuration

To measure the performance of our proposed implementation in various situations, we configure our experimentations as follows:

- the size of NoC is 3*3, 4*4, 5*5, 10*10,
- the size of task set is 16, 32, 64, 128,
- the utilization of task and flow is from 10% to 90%,
- the number of flows is considered as the size of the task set.

Because each experimentation will generate a random mapping and Synthetic benchmark, one time testing cannot illustrate the difference among all implementations and schemes. Therefore, we increase the number of testing times to 1000000, in order to have a better coverage.

C. Results Analysis

Figure 8 shows partial results of the experimentation, while more details are shown in Table II. All the Y-Axis in Figure 8

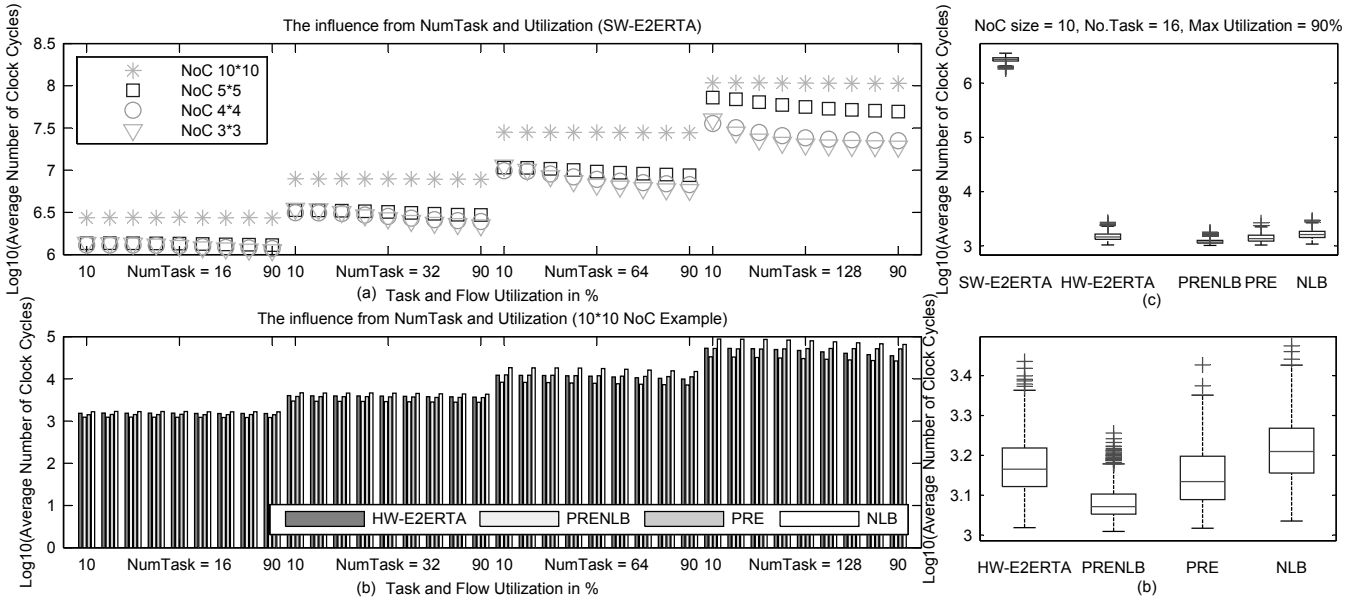

Fig. 8: (a) SW-E2ERTA on difference test benches, (b) hardware versions on 10*10 NoC. (c) SW-E2ERTA vs hardware versions, (d) PRE+HW-E2ERTA vs NLB vs PRE+NLB vs E2ERTA.

show the numbers of clock cycles that has been used to finish an E2ERTA computation. Because the numbers are too large, we arrange them in log_{10} style (i.e. an entry of 6.1 represents a time of $10^{6.1}$ clock cycles).

Figure 8a presents the performance of SW-E2ERTA affected by Size of NoC, Number of Task, and Utilization of Task and Flow. From these points, we can find that the larger size of NoC or the number of task the SW-E2ERTA has to calculate, the more evaluation time will be required. However, the influence from utilization of task and flow is in parabola style instead of a linear style. This can be seen from the examples in Table II, which are labelled in gray colour. The reason of this phenomenon is that the extremely lower or higher utilization can early terminate the iterative calculation of E2ERTA. We can make an assumption as follows:

- the current observed objective (Flow as an example) is i,
- the number of iterations required by lower, moderate and higher utilization are N_{lower}, $N_{moderate}$ and N_{higher} respectively,
- the number of clock cycles used to finish a single iterative calculation is nearly the same or equal to N_{SIC}.

When the utilization is extremely low, the E2ERTA may be terminated by $R_i^{n+1} = R_i^n$ within few iterations such as two or three iterations. The calculation will become harder with the utilization increased and results in more iterations required. However, when the utilization becomes extremely high, the complexity of E2ERTA will decrease. This is because E2ERTA can easily determine the observed objective and will miss deadline by finding $R_i \geq D_i$ within few iterations. Therefore we can get the inequality $N_{lower} \leq N_{moderate} \geq N_{higher}$ and further obtain the total execution time, which is $N_{lower} * N_{SIC} \leq N_{moderate} * N_{SIC} \geq N_{higher} * N_{SIC}$. Thus, the influence from utilization follows a parabola style.

Figure 8b shows the detailed performance of proposed implementation on 10*10 NoC with difference Number of Task, and Utilization of Task and Flow. It can be seen that the hardware implementation follows the characteristic of SW-E2ERTA, however, with much less computation time.

Figure 8c and d show the details of an example whose data has been labeled in gray colour in Table II. The X-Axis shows the different versions of E2ERTA or assembled schemes. From Figure 8c, we can see that all the hardware versions are much faster than SW-E2ERTA. For example, the HW-E2ERTA's average number of clock cycles is around $1 * 10^{3.179}$, while the SW-E2ERTA's is about $1 * 10^{6.435}$. This means HW-E2ERTA is approximately 1000 times faster than SW-E2ERTA. Such significant speed-up enables a much wider use of the E2ERTA in the optimisation process of NoC-based systems (i.e. guiding designers towards acceptable solutions), rather than only validating the final system. In the next section, we will address this specific scenario.

Figure 8b presents various assembly schemes, PRENLB, HW-E2ERTA, PRE and NLB, with the results ranking from best to worst. The reason why the NLB obtains the worst results is that NLB has to calculate the lower bound first and then start the exact calculation for each computation. We assume that:

- the number of clock cycles used to calculate the lower bound in NLB is N_{clb},
- the number of clock cycles used to compute the following exact calculation is N_{cec},
- the number of clock cycles used by HW-E2ERTA is N_{E2ERTA}.

We could get the total number of clock cycles used by NLB is $N_{clb} + N_{cec}$. We can guarantee $N_{cec} \leq N_{E2ERTA}$, but we cannot guarantee $N_{clb} + N_{cec} \leq N_{E2ERTA}$. Therefore, only using NLB may be slower than HW-E2ERTA.

Since PRE is a significant test, only using upper bound and lower bound cannot guarantee the final results. When PRE

TABLE II: Time to perform full response time analysis (log_{10} clock cycles)

		3*3					4*4					5*5					10*10				
# Tasks	U(%)	SW	HW	PRENLB	PRE	NLB	SW	HW	PRENLB	PRE	NLB	SW	HW	PRENLB	PRE	NLB	SW	HW	PRENLB	PRE	NLB
16	10	6.140	3.417	3.171	3.326	3.561	6.113	3.308	3.136	3.249	3.401	6.134	3.261	3.116	3.209	3.316	6.437	3.185	3.093	3.151	3.222
	20	6.136	3.409	3.160	3.319	3.554	6.113	3.305	3.133	3.245	3.399	6.135	3.261	3.114	3.208	3.319	6.438	3.190	3.093	3.158	3.228
	30	6.130	3.398	3.153	3.313	3.544	6.113	3.301	3.128	3.241	3.396	6.135	3.256	3.108	3.205	3.315	6.440	3.186	3.094	3.156	3.225
	40	6.120	3.373	3.136	3.301	3.524	6.110	3.292	3.121	3.237	3.390	6.132	3.250	3.104	3.197	3.308	6.438	3.189	3.092	3.154	3.226
	50	6.104	3.339	3.117	3.296	3.495	6.104	3.278	3.113	3.230	3.373	6.131	3.244	3.099	3.192	3.302	6.442	3.188	3.091	3.161	3.227
	60	6.087	3.294	3.101	3.287	3.458	6.094	3.255	3.099	3.215	3.350	6.126	3.232	3.094	3.185	3.288	6.438	3.181	3.088	3.153	3.220
	70	6.073	3.260	3.092	3.275	3.427	6.084	3.238	3.091	3.200	3.327	6.121	3.223	3.092	3.178	3.277	6.436	3.180	3.085	3.153	3.219
	80	6.060	3.230	3.079	3.263	3.398	6.079	3.227	3.089	3.201	3.311	6.117	3.209	3.083	3.166	3.261	6.438	3.181	3.086	3.155	3.222
	90	6.053	3.210	3.072	3.261	3.380	6.071	3.209	3.080	3.182	3.288	6.112	3.202	3.082	3.165	3.254	6.435	3.179	3.084	3.152	3.220
32	10	6.544	4.007	3.566	3.836	4.168	6.496	3.861	3.543	3.748	4.029	6.526	3.758	3.526	3.699	3.924	6.898	3.604	3.474	3.582	3.671
	20	6.537	3.998	3.607	3.875	4.160	6.494	3.859	3.541	3.746	4.026	6.523	3.750	3.516	3.683	3.917	6.896	3.598	3.470	3.579	3.665
	30	6.506	3.955	3.624	3.908	4.126	6.487	3.842	3.531	3.752	4.012	6.520	3.743	3.509	3.681	3.911	6.898	3.596	3.468	3.578	3.664
	40	6.466	3.890	3.576	3.906	4.078	6.469	3.807	3.513	3.754	3.986	6.513	3.724	3.494	3.675	3.896	6.897	3.597	3.466	3.579	3.666
	50	6.431	3.827	3.515	3.896	4.034	6.453	3.770	3.485	3.764	3.957	6.506	3.699	3.481	3.670	3.875	6.897	3.594	3.461	3.574	3.662
	60	6.399	3.766	3.458	3.879	3.993	6.432	3.719	3.463	3.758	3.921	6.494	3.664	3.466	3.661	3.848	6.896	3.587	3.458	3.571	3.655
	70	6.377	3.714	3.422	3.868	3.959	6.413	3.669	3.441	3.747	3.886	6.484	3.630	3.449	3.655	3.819	6.893	3.578	3.452	3.561	3.645
	80	6.362	3.678	3.398	3.857	3.937	6.403	3.633	3.425	3.740	3.862	6.473	3.598	3.438	3.645	3.790	6.894	3.575	3.449	3.561	3.642
	90	6.347	3.641	3.381	3.851	3.915	6.391	3.594	3.412	3.733	3.837	6.470	3.580	3.429	3.646	3.776	6.891	3.568	3.445	3.557	3.637
64	10	7.055	4.608	4.153	4.452	4.771	6.994	4.462	4.036	4.315	4.644	7.031	4.364	3.989	4.240	4.555	7.449	4.088	3.921	4.095	4.265
	20	7.005	4.546	4.268	4.541	4.722	6.986	4.451	4.120	4.373	4.634	7.027	4.358	4.019	4.263	4.549	7.450	4.083	3.916	4.087	4.262
	30	6.929	4.438	4.169	4.528	4.645	6.955	4.400	4.127	4.402	4.594	7.016	4.337	4.048	4.295	4.532	7.448	4.080	3.914	4.085	4.260
	40	6.870	4.341	4.049	4.498	4.581	6.921	4.333	4.068	4.396	4.547	7.002	4.295	4.032	4.306	4.500	7.448	4.074	3.903	4.077	4.255
	50	6.828	4.264	3.959	4.471	4.533	6.891	4.268	4.008	4.383	4.502	6.984	4.245	3.992	4.302	4.464	7.450	4.064	3.896	4.074	4.245
	60	6.806	4.212	3.898	4.455	4.504	6.870	4.211	3.949	4.368	4.465	6.970	4.198	3.951	4.296	4.432	7.447	4.045	3.885	4.065	4.228
	70	6.788	4.173	3.857	4.442	4.481	6.852	4.160	3.900	4.354	4.433	6.959	4.150	3.904	4.285	4.401	7.445	4.027	3.874	4.059	4.209
	80	6.774	4.140	3.823	4.431	4.463	6.839	4.117	3.863	4.341	4.408	6.948	4.105	3.867	4.274	4.371	7.444	4.013	3.864	4.055	4.191
	90	6.766	4.119	3.800	4.426	4.452	6.829	4.086	3.833	4.334	4.390	6.941	4.074	3.841	4.267	4.352	7.443	3.996	3.855	4.049	4.175
128	10	7.602	5.192	4.927	5.167	5.355	7.553	5.074	4.703	4.967	5.255	7.860	4.977	4.607	4.866	5.177	8.035	4.726	4.523	4.721	4.945
	20	7.445	4.974	4.733	5.126	5.201	7.503	4.993	4.775	5.038	5.193	7.840	4.956	4.701	4.936	5.157	8.036	4.722	4.517	4.712	4.941
	30	7.363	4.834	4.545	5.072	5.116	7.447	4.879	4.661	5.015	5.114	7.805	4.887	4.674	4.951	5.107	8.035	4.715	4.507	4.707	4.934
	40	7.325	4.758	4.447	5.044	5.074	7.409	4.783	4.549	4.983	5.054	7.773	4.809	4.600	4.936	5.053	8.032	4.697	4.496	4.712	4.920
	50	7.305	4.714	4.394	5.028	5.050	7.383	4.712	4.474	4.959	5.012	7.749	4.739	4.531	4.916	5.008	8.032	4.673	4.482	4.719	4.902
	60	7.294	4.687	4.363	5.020	5.036	7.367	4.658	4.419	4.941	4.981	7.728	4.680	4.474	4.898	4.972	8.030	4.641	4.465	4.721	4.879
	70	7.288	4.669	4.341	5.014	5.026	7.359	4.623	4.386	4.930	4.963	7.715	4.634	4.432	4.886	4.945	8.028	4.607	4.450	4.717	4.855
	80	7.284	4.657	4.329	5.010	5.020	7.353	4.598	4.362	4.922	4.949	7.704	4.597	4.398	4.875	4.924	8.027	4.574	4.434	4.714	4.832
	90	7.280	4.648	4.320	5.007	5.016	7.349	4.579	4.345	4.917	4.939	7.694	4.570	4.376	4.868	4.909	8.029	4.549	4.424	4.711	4.816

Note: SW refers to SW-E2ERTA, HW refers to HW-E2ERTA and U refers to the utilization of task or flow.

can identify the results, the total number of clock cycles will be reduced. However, once it cannot identify the results, a following exact calculation will be launched. That means the number of clock cycles will be increased resulting in that the calculation of PRE will cost time. Therefore, the performance of PRE can be worse than HW-E2ERTA.

Next is the PRENLB that has the abilities inherited from both PRE and NLB. It can avoid the exact test in some situations and guarantee the final results within a short running time when PRE is failed. We also make an assumption that:

- the number of clock cycles used to calculate the lower bound and upper bound is N_{ulb},
- the number of clock cycles used to compute the following exact calculation is N_{cec}.

For a single test, the total number of clock cycles used by PRENLB is either N_{ulb} or $N_{ulb} + N_{cec}$. Theoretically, the PRENLB cannot guarantee the performance in one time test, whether it is better than HW-E2ERTA's performance. However, after testing for 1000000 times, the average number of clock cycles cost by PRENLB is around $1*10^{3.084}$, while HW-E2ERTA's is about $1 * 10^{3.179}$. We can generally summarize that PRENLB is approximately 1.25 times faster than HW-E2ERTA.

VII. Case Study

The improvement of three orders of magnitude on the analysis time reported in the previous section might not seem so relevant in the case this technique is used at design time, as one would not mind waiting for a few minutes or even hours to test whether a particular NoC task mapping is schedulable and therefore safe to be deployed. However, as shown in [11] and [7], the analysis can actually be used to find a task mapping through a heuristic search approach. In such cases, the analysis is not applied once but millions of times before a fully schedulable mapping can be found. Within such an approach, the achieved improvement of three orders of magnitude means that a much wider search can be performed within an acceptable amount of time. This is of course useful for design-time optimisation, as it can cover a greater portion of the mapping solution space and can potentially find more efficient mappings. Furthermore, it opens the possibility of addressing open systems, where the application tasks and packet flows are not completely known at design time and may require the optimisation of the mapping after the NoC system has been deployed (e.g. due to the release of new tasks, or to the increased computation or communication demands of the existing ones).

As a case study, we implemented a heuristic search pipeline based on a genetic algorithm (GA), and used the hardware-accelerated analysis as its fitness function. It is effectively a hardware-accelerated implementation of the approach presented in [11] and [7], which we describe below and illustrate in Figure 9.

A GA works by manipulating chromosomes which represent an individual solution to the problem we are trying to optimise. In this case, a chromosome must represent a specific mapping of tasks to cores over a NoC. We choose a simple encoding also used in [11], where each gene of the chromosome

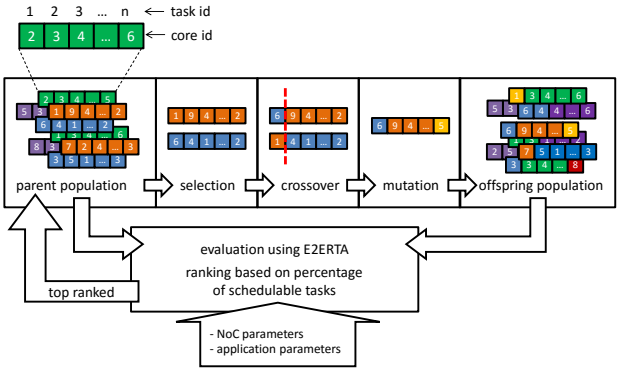

Fig. 9: Overview of the task mapping approach based on a genetic algorithm.

represents a task, and the contents of the gene store the number of the processing core onto which the task will be mapped. Therefore, the number of genes on a chromosome is the number of application tasks we are trying to map.

The GA optimisation pipeline starts with an initial parent population, represented by their chromosomes, which can be randomly created (i.e. randomly selecting the value of each gene of each chromosome). It then creates an offspring population by operating over the parent population using mutations (e.g. randomly changing the value of a gene) and crossovers (e.g. combining two halves of two chromosomes to create a new one). Finally, it applies E2ERTA as a fitness function, which will calculate how many of the tasks are end-to-end schedulable, and will use that to rank all chromosomes of the combined population and thus define which of them will be allowed into the next generation. The process is then repeated for a fixed number of times, or until a mapping without unschedulable tasks is found.

We implemented a GA pipeline in hardware, including all the control as well as the operations for crossover, mutation, ranking and selection, and we integrated the proposed E2ERTA hardware implementation as the fitness function. Again, this was done on a Xilinx VC709 development board. The GA control and operators, as well as the E2ERTA fitness function, are integrated as customer peripherals on the AXI bus, which are then initialized and controlled by the embedded MicroBlaze CPU.

The MicroBlaze CPU starts by loading the GA configuration (e.g. crossover rate, mutation rate, size of population) and stores the taskset information in custom memory structures that follow the same structure of the ones used in the synthetic benchmark generation described in Section VI. Then, GA controller will initialize the population and trigger the iterations that include offspring creation, application of the E2ERTA fitness function and selection. Once the iterations are concluded, the MicroBlaze CPU will collect data from hardware components, and output the results through a UART port.

To accelerate the application of the E2ERTA fitness function over the population, we exploited the inherent parallelism of GAs and allowed the instantiation of multiple E2ERTA hardware components, so that multiple chromosomes (i.e. mappings) could be evaluated concurrently. In this implementation, however, we design a simple control structure that enforces all E2ERTA instances to work in lockstep (i.e. all E2ERTA instance are given a chromosome to evaluate at the same time, and will only receive the next one when all of them are ready). In Table III we show the average number of clock cycles taken for a single generation.

TABLE III: Performance results of hardware-accelerated task mapping genetic algorithm

population size	number of E2ERTA instances	average execution time of GA operators per generation (clock cycles)	average execution time of E2ERTA (clock cycles)
6	2	792	243840
	3	792	229760
	4	792	229540
	5	792	229180
8	2	1060	500711
	4	1059	251672
16	2	2115	896560
	4	2112	812150

Table III also shows that, despite of the hardware acceleration, the fitness function still heavily dominates the execution time of the GA pipeline, which provides evidence of the usefulness of the approach presented in this paper. Without the hardware acceleration, the whole GA pipeline would effectively be three orders of magnitude slower. The results also show that the lockstep nature of this implementation prevents the full parallelising of the search. For example, the E2ERTA execution time using four instances is not always two times faster than when using two instances, because the lockstep behaviour forces three instances to be idle until the slowest instance finishes its execution. We leave as future work an improvement that allows each instance of E2ERTA to request another chromosome whenever they are ready, which will improve the practical usefulness of this case study.

VIII. Conclusions and Future Work

In this paper, we proposed a hardware implementation of End-to-End Response Time Analysis (E2ERTA). We also introduce two accelerated components PRE (Pre-check) and NLB (new lower bound). We assemble them in different schemes and compare their performance with a software implementation of E2ERTA. The results show that the hardware version is 1000 times faster than software version. After careful consideration, we found that only using one of the accelerated components cannot guarantee the desired performance. However, when combining these components together, the PRENLB scheme can obtain the best acceleration.

Our proposed HW-E2ERTA and its accelerated components can be used as a fast evaluation method to investigate whether

a task set is schedulable on a NoC according to a given mapping. To show its effectiveness, we reported a case study that reproduced state-of-the-art GA-based mapping optimisation for hard real-time NoCs, and showed that they can fully benefit from the acceleration obtained by the proposed approach. We also show that the use of parallel E2ERTA instances can be exploited in such case study, and the use of more sophisticated control mechanisms that avoid lockstep execution is left for future work.

REFERENCES

[1] N. C. Audsley, A. Burns, M. F. Richardson, K.Tindell and A. J. Wellings, *Applying new scheduling theory to static priority pre-emptive scheduling*, Software Engineering Journal, 8(5), 284-292, 1993.

[2] E. Bini, and S. K. Baruah, *Efficient computation of response time bounds under fixed-priority scheduling*, In Proc 15th Int Conf on Real-Time and Network Systems, 95-104, 2007.

[3] E. Bolotin, I. Cidon, R. Ginosar, A. Kolodny, *QNoC: QoS architecture and design process for network on chip*, Journal of Sys. Arch., 50(2-3), 105–128, 2004.

[4] R. I. Davis, A. Zabos, A. Burns, *Efficient Exact Schedulability Tests for Fixed Priority Real-Time Systems*, IEEE Trans. Computers, 57(9), 1261-1271, 2008.

[5] S. Furber and J. Bainbridge, *Future trends in SoC interconnect*, In Proc Int Symposium on System-on-Chip, 183-186, 2005.

[6] J. Henkel, W. Wolf, S. Chakradhar, *On-chip networks: A scalable communication-centric embedded system design paradigm*, In Proc VLSI Design, 2004.

[7] L. S. Indrusiak, *End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration*, Journal of Sys. Arch., 60(7), 553–561, 2014.

[8] A. E. Kiasari, A. Jantsch and Z. Lu, *Mathematical Formalisms for Performance Evaluation of Networks-on-chip*, ACM Comput. Surv., 45(3), 2013.

[9] C. Liu and J. Layland, *Scheduling algorithms for multiprogramming in a hard real-time environment*, J.ACM, 20, 46-61, 1973.

[10] P. K. Sahu, S. Chattopadhyay, *A survey on application mapping strategies for Network-on-Chip design*, Journal of Sys. Arch., 59(1), 60-76, 2013.

[11] M. N. S. M. Sayuti, L. S. Indrusiak, *Real-time low-power task mapping in Networks-on-Chip*, In IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 14-19, 2013.

[12] M. Schoeberl, *A Time-Triggered Network-on-Chip*, In FPL 2007.

[13] Z. Shi, A. Burns, L. S. Indrusiak, *Schedulability analysis for real time on-chip communication with wormhole switching*, IJERTCS, 1(2), 1-22, 2010.

[14] Y. Ma, M. N. S. M. Sayuti, L. S. Indrusiak, *Inexact End-to-End Response Time Analysis as fitness function in search-based task allocation heuristics for hard real-time network-on-chips*, In Proc ReCoSoC, 2014.

[15] Y. Ma, L. S. Indrusiak, *Hardware-accelerated Response Time Analysis for priority-preemptive Networks-on-Chip*, In Proc ReCoSoC, 2015.