Decentralized Hardware Scheduler for Self-Timed Data-Driven Multiprocessor

Kazuma FUKUDA, Yushin WADA, and Makoto IWATA
Graduate School of Engineering, Kochi University of Technology,
Kami, Kochi, 782-8502 Japan

Abstract— *For the future internet of things (IoT) devices with edge-heavy computing, it is necessary to perform real-time processing on multicore system. Since most of IoT devices often execute aperiodic real-time tasks, the dynamic scheduling policy is indispensable for them. In order to reduce the task scheduling overheads, the hardware scheduler must be implemented on the multicore system. Therefore, this paper proposes a decentralized hardware scheduler for the self-timed (clockless) data-driven multiprocessor, in which least slack time (LST) scheduling is adopted to prevent the deadline miss of hard real-time tasks. Each switching module composing the multistage interconnection network autonomously decides the local destination for the task allocation. This is also realized by the self-timed pipeline circuit. As a result, the decentralized task scheduling is realized on the network of the multicore system. The effectiveness of the proposed scheduler is evaluated through our architecture simulator.*

Keywords: Multiprocessor, Decentralized hardware scheduler, Least slack time scheduling, Self-timed pipeline, Data-driven processor

1. Introduction

Recent Internet of things (IoT) devices have permeated as edge heavy computing devices, and they are required to operate with higher functionality and higher performance. Several years ago, the performance of most embedded systems has been improved by increasing the operating frequency of a single-core processor. However, from the viewpoint of power consumption and heat dissipation, there is a limit to improving the performance in a single-core. Therefore, most of embedded microprocessor architectures have proceeded to the multicore architecture [1]. In addition, in a mission critical system such as modern intelligent robotics and vehicles, if a certain error recovering could not be functioned before the required time, there is a danger of leading to serious accident. Therefore, in the future IoT devices, it is necessary to achieve real time characteristics on the multicore system.

In the multicore system, several cores are mutually connected to realize inter-processor communication among the cores. That is called an interconnection network. In order to realize real-time processing in a multicore system, it is necessary to assign requested tasks to a core or more cores so as to satisfy the time constraints of each task. In addition, when dynamic load fluctuation of the task is likely to exceed the upper limit of the processing load in the core, it is necessary to distribute the processing load among the cores by requesting processing to other cores. Therefore, in the multicore system, each core desirably realizes real time processing with small overhead for scheduling and distributing the required tasks.

In order to satisfy the time constraints of each task in each core, it is necessary to perform task scheduling considering the priority of each task. In general, real time tasks of IoT devices are periodic or aperiodic. Aperiodic tasks are difficult to process with the static scheduling scheme such as rate monotonic scheduling because task priorities are not updated at runtime. Earliest deadline first (EDF) [2] and least slack time (LST) [3] are well known for the dynamic scheduling policy. In the EDF policy, the highest priority is assigned to the task that is closest to the deadline in all requested tasks. In the LST one, the highest priority is assigned to the task that has shortest slack time in them. Since the EDF algorithm does not care about the execution time, a deadline miss cannot be predicted in the multicore system. In contrast, the LST can predict deadline misses by updating slack time in runtime.

In this study, we focus on data-driven processor (DDP) [4] which is possible to execute multiple operations without data dependence in parallel. Therefore, low priority tasks and high priority tasks can be executed in a multiplexed manner without a context switch. Therefore, those tasks can be executed in a multiplexed manner without a context switch. The LST hardware scheduler for DDP is proposed in [5]. In order to perform task scheduling in a multicore system, it is necessary to manage the real time information of each task. From the viewpoint of scalability, it is thought that it is preferable to manage it in a distributed manner rather than a centralized one. DDP needs no interrupt handling and can execute operations triggered by the arrival of an event. Therefore, the overhead of task input / output to / from the DDP is smaller than that of the conventional Von Neumann sequential processors, and autonomous distributed control of tasks is considered to be high affinity to the data-driven execution principle.

In this paper, a decentralized hardware scheduler for

ISBN: 1-60132-487-1, CSREA Press ©

the self-timed data-driven multiprocessor is proposed, in which the multi-stage interconnection network composed of decentralized small switches plays both roles of autonomous task scheduling and allocation. In the following section, the requirements for distributed task scheduling on the interconnection network and necessary features of the cores are discussed. Section 3 describes the proposed hardware scheduler and core architecture. Section 4 illustrates the performance evaluation results based on our newly developed architecture simulator. Finally, the conclusion and further works are described in the last section.

2. Real Time Processing for Multicore

In the multicore system, when a task is requested to be executed, the system must decide which core should execute the task. This kind of real time task assignment should be realized to satisfy the time constraints of all requested tasks. In order to satisfy the deadline constraint of the hard real-time task, it is well known to set a deadline for each task in the application and to guarantee the deadline limit for the entire application when each task keeps its deadline [6]. Therefore, in this paper, it is assumed that the application is divided beforehand into multiple subtasks with individual deadlines in advance. For the optimal task scheduling on the multicore system, it is necessary to utilize various execution conditions of all tasks and all cores. However, it is difficult to gather and analyze all of the execution information at the single centralized scheduler without sacrificing the scalability of the multicore system. Considering the number of cores would increase even in the future IoT devices, the decentralized scheduler would be preferable for the real time multicore system. In this section, we will discuss the requirements on the interconnection network and core architecture to implement the decentralized real-time task scheduler.

2.1 Requirements on Network

In this study, it is supposed that the inter-core network of the multicore system is implemented as a multi-stage interconnection network, which composed of small packet switching elements, e.g., 2-input and 2-output switching module. Thus, the decentralized scheduling can be realized by autonomously operating each packet switch in the network and determining the allocated core at the switch. Furthermore, a requested task possesses the real time task information, such as deadline, execution time, slack time, and maximum parallelism, and so on. The operation of each switch based on these information is as follows:

- Round robin
  The input tasks are transferred alternately. It is unnecessary to hold the information of the task at the switching module, while the assignment situation of the core cannot be considered at the switch.
- Round robin with threshold
  Basically, the input tasks are transferred alternately, but not to cores that exceed the threshold. It is necessary to store the passed task as a history. In the switch modules, only information on tasks that passed through can be stored and the transfer status within the switch can be managed.
- Priority base
  The transfer destination of the input task is determined by the priority of the task. EDF and LST are typical as dynamic scheduling algorithms for the real time processing. Since the EDF algorithm cannot predict deadline miss, it cannot be used for the multicore system. In the LST scheduling, priorities are assigned in order of slack time, i.e., the maximum time that a task can be delayed on its activation to complete within its deadline. Thus, before assigning a requested task to a core in the multicore system, it is possible to predict whether the target core will violate the deadline of the task or not. The slack time (T_{slack}) of the task is calculated by equation 1 as follows.

$$T_{slack} = (t_{deadline} - t_{req}) - T_{exe} \quad (1)$$

  where $t_{deadline}$, t_{req}, and T_{exe} denotes the absolute deadline, task request time, and (worst case) execution time of the task, respectively.

For each input task, strict scheduling and semi-strict scheduling could be considered for each switch module.

- Strict scheduling
  All tasks keep their deadline when knowing the worst case execution time of the task and assigning the input task to the core. Since the network operates only when assigning input tasks, it is possible to suppress the number of switches required for operation. However, it cannot cope with load fluctuation due to branching and repetition involving the assigned task. In addition, it is necessary to estimate the execution time excessively for preventing any deadline miss.
- Semi-strict scheduling
  At the time of assigning input tasks, scheduling is performed based on the average execution time of each task. Therefore, if the dynamic load of the task becomes larger than the pre-estimated load due to branching or repetition, it is necessary to re-distribute the load among the cores. Since the load balancing among the cores is performed, the number of switch modules required for operation might be more than that of the strict scheduling.

From the viewpoint of cooperation between the core and the switch module, it is necessary to consider whether to feedback information on the core to the switch.

- With feedback information
  When there is feedback from the core to the switch,

the switch can grasp the dynamic load change of the core, so the information held by the switch becomes more accurate. However, as feedback information flows through the network, network traffic will increase. In addition, it is necessary to consider that there is a time lag before the state of the core reaches the switch.

- Without feedback information
  When there is no feedback from the core to the switch, the switch cannot grasp the state of the core. However, traffic on the network can be suppressed.

In the multistage interconnection network such as omega network, the number of switching stages necessary for N cores is $\log_2 N$. The packet flowing at the first stage can be transferred to N cores and thus the packet flowing at the i-th stage can be transferred to limited number of cores $N/2^i$. If the switching modules does not receive the execution condition of cores as feedback information, the switching modules at the i-th stage can keep only the history of the requested tasks, i.e., tasks assigned to one of $N/2^i$ cores at the switching module. In the following section, it is discussed that those limited information enables us to schedule and allocate tasks to the multicore at each switching module in the decentralized manner.

2.2 Requirements on Cores

In this paper, we intend to realize the dynamic task scheduling on the core processor. Thus, the core allows a higher priority task to preempt processing resource used for the lower priority tasks. In order to improve real-time performance on the core, the overhead of task switching or preemption should be minimized on the core. In this point, the self-timed data-driven processor (DDP) could be preferable to do that because the DDP executes all independent operations in fine-grain parallel execution scheme. This means that task switching always occurs at the primitive operation level without any performance degradation as long as the amount of processing load caused by requested tasks does not exceed the capacity of parallel processing in the DDP. If it exceeds that, the task scheduler adopted in the DDP have to operate. In this case, the least slack time scheduling policy could prefer to adopt predictable LST of deadline miss like network. However, when the LST scheduler is applied within the core, frequent task switch called thrashing occurs when slack times of two or more tasks are similar, and the overhead of scheduling at the time of execution increases. Thus, it is necessary to be able to process multiple tasks without the overhead of the task switch. Therefore, we focus on the self-timed data-driven processor (DDP) [4] that can execute multiple tasks without performance overhead of task switching or preemption.

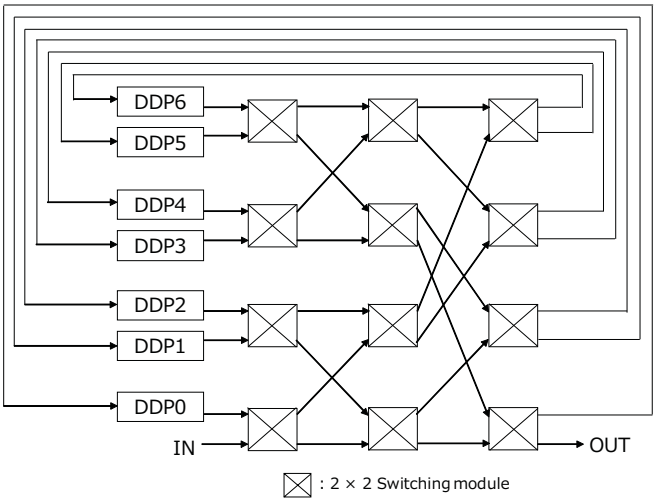

Fig. 1: An Example multicore configuration of 7-core DDP.

3. Decentralized Hardware Scheduler Architecture

Based on the discussion in the previous section, we will discuss a decentralized hardware scheduler for self-timed data-driven multiprocessor. Fig. 1 shows the overall multicore system including cores and interconnection network. This figure shows an example configuration of 7 cores, one input port, and one output port. The necessary number of input ports depend on the trade-off balance between the total amount of processing load and input traffic. Each switching module in the network is implemented by the self-timed pipeline (STP) as shown in Fig. 2. The basic transfer control circuit of the STP is a Confidence flip-flops (C-element). In the STP, when a valid packet arrives at the pipeline stage, the C-element exchanges data transfer request signal (send) and data transfer acknowledge signal (ack) with its succeeded C-element. CB is a C-element with a branch function, the destination pipeline stage is determined by the control signal output from logic, and the send and the ack are exchanged between the neighbor stages. CM is a C-element with a merge function, and it arbitrates packets arriving from two stages. In addition, the STP itself has elasticity in terms of packet transfer. Thus, if packets from both directions arrive at the same time, this unit arbitrates along with the predefined arbitration policy such as first-come-first-service, etc. In addition, DDP is adopted as the core. Transfer control of packets in DDP is also realized by STP. It would flexibly mitigate the processing load fluctuation when the real-time application works on the DDP. Since the pipeline stage without effective data does not consume electric power, it can save the operating power autonomously. The LST hardware scheduler for the DDP core is reported in [5] and the discussion in this paper is thus emphasized at viewpoint of the multicore system.

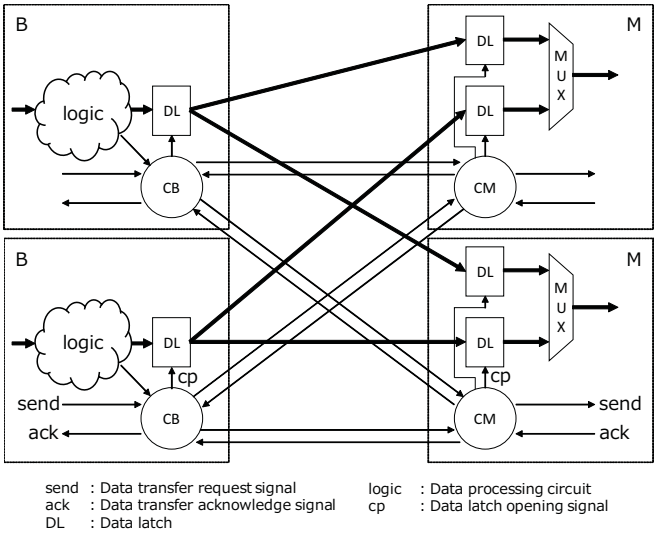

Fig. 2: Structure of STP-based switching module.

3.1 Network Architecture

In case that the number of cores is N, the number of stages of the network is $log_m N$. In the proposed method, switching modules belonging to the i-th stage decides which output port is allocated for the requested task. If i is less than a threshold $\mathrm{S_{th}}$, the switching modules roughly schedule the requested tasks based on round robin policy. Otherwise, the switching modules allocate the requested tasks based on their priorities, i.e., slack time in this case. In addition, it is assumed that the tasks are strictly scheduled at the arrival time of the requested task at each switching module and there is no feedback information from the core to the switching module. Therefore, in the switching modules at the former stages, the arriving tasks are allocated to the connected switches in turn. In the latter stages, the switching modules decide which group of cores connected to the output port should be assigned the task to. When the i-th task arrives at the multicore system, at the input module, the present time is stamped as its requested time $t_{req}(i)$ and the estimated execution time $T_{exe}(i)$ and the relative deadline of the i-th task are fetched from the memory storing all of the task information. After that, its slack time $T_{slack}(i)$ is calculated from equation 1, and they are attached to the task request packet. In order to decide a group of cores based on slack time base, the switching module should choose a group of cores with the most margin.

To simplify the following descriptions, it is assumed that i = $\log_2$N. The above margin is represented by a variable T'_{slack}, which is the minimum slack time of the tasks assigned to a core. Thus, T'_{slack} of each core is calculated in case that a task is assigned to each of all candidate cores, and then the requested task should be assigned to the core that has the largest T'_{slack} in the candidate cores. A pseud code for calculating this effective slack time $T'_{slack}(i)$ of a core due to assigning the i-th task is described in Algorithm 1. $T'_{skack}(i)$ can be classified into the following three conditions if the task is assigned to the core. Here, $t_{req}(i)$ is the execution request time of the i-th task, and $t_{req}(l)$ is the request time of the l-th task which is the first task assigned after the core was idle. This means that the core has been busy from the time $t_{req}(l)$ to the time $t_{req}(i)$ continuously. The sum of $T_{exe}(k)$ calculates the total time necessary from the l-th task to the i-th task, i.e., $(\mathrm{i}-\mathrm{l}+1)$ tasks.

1) Case 1: No task is executed in the core.
   The case shown in the Fig. 3(a) is that no task is executed in the core when requesting a task. At this time, the requested task can be immediately executed. T'_{slack} is fixed to the slack time of the requested task and i is assigned to l.
2) Case 2: The slack time of the requested task is shorter than that of one or more of the existing tasks.
   The case shown in the Fig. 3(b) is that the requested task will finish to be executed prior to one or more existing tasks. Therefore, the effective slack time of the core can be calculated by difference between the previous effective slack time $T'_{slack}(i-1)$ and the slack time of the requested task $T_{exe}(i)$.
3) Case 3: The slack time of the requested task is the longest in those of the existing tasks.
   The case shown in the Fig. 3(c) is that the requested task is enqueued and the existing tasks could be executed prior to the i-th task. Therefore, the effective slack time of the core can be calculated based on the slack time and it is updated by subtracting the remaining total execution time from the l-th task to the (i-1)-th task.

Each switching module carries out this algorithm for each connected core and assigns the request task to the core where $T'_{slack}(i)$ is the largest. Only for the core to which the requested task is assigned, its $T'_{slack}(i)$ is updated at the switching module.

3.2 Core Architecture

The basic architecture of the LST-based DDP adopted in the core is shown in the Fig. 4. There is a load monitor (LM) and a task queue (TQ) inside the DDP. During task execution, the LM observes the processing load within the DDP and the TQ manages a set of accepted tasks based on their slack time. The slack time of the waiting tasks are decreased by the waiting (i.e., queueing) time within the TQ. In order to monitor the processing load of the DDP, increase or decrease of the packets activated within the DDP is observed by counting the packets passing through the particular stage. The cause of the processing load fluctuation can be detected from the execution control flags held in the DDP packet so that the load monitor can observe the number

Algorithm 1 Effective slack time $T'_{slack}(i)$ of a core due to the i-th task assignment.

if $((t_{req}(i) - t_{req}(l)) - \sum_{k=l}^{i-1} T_{exe}(k) \geq 0)$ **then**
  {/* Case 1 */}
  $T'_{slack}(i) = T_{slack}(i)$
  $l = i$
else if $T'_{slack}(i-1) > T_{slack}(i)$ **then**
  {/* Case 2 */}
  $T'_{slack}(i) = T'_{slack}(i-1) - T_{exe}(i)$
else
  {/* Case 3 */}
  $T'_{slack}(i) = T_{slack}(i) - (\sum_{k=l}^{i-1} T_{exe}(k) - (t_{req}(i) - t_{req}(l)))$
end if

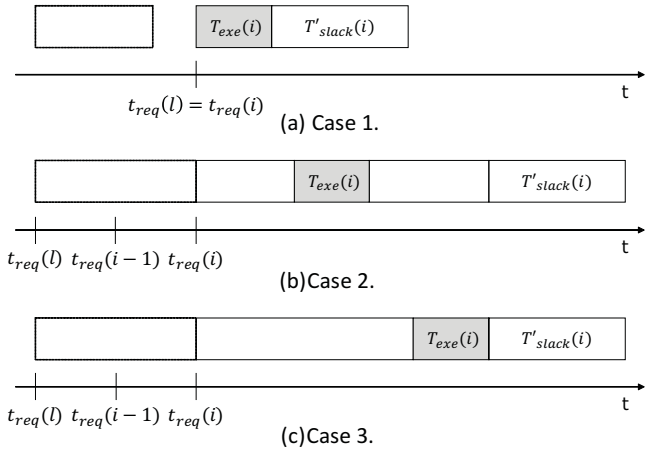

Fig. 3: Preemption pattern of the i-th task.

of activated packets within the DDP. When the real time processing load are close to the multiprocessing capability of the DDP, execution control to stop or resume tasks can be conducted by enqueueing or dequeueing packets at the TQ. There is a sorter in TQ, and the queued tasks are sorted in ascending order of the slack time.

Since the DDP executes tasks on a packet-by-packet basis, fine-grained task scheduling can be realized in accordance with the tasks allocated by the switching module of the network. Therefore, in our feasibility study, we are trying to realize the decentralized LST-based hardware scheduler without utilizing the feedback information from the cores and investigate its potentiality through the experimental evaluations described in the next section.

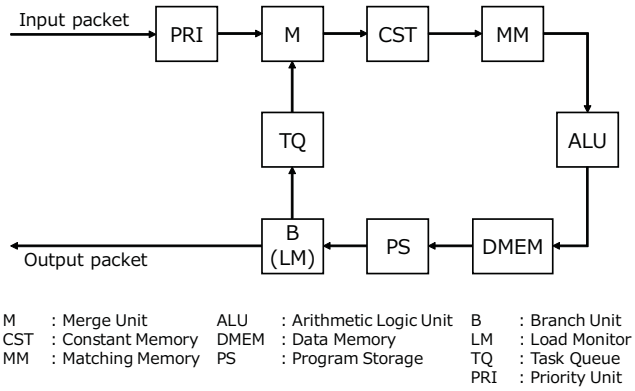

Fig. 4: Architecture of LST-based DDP.

4. Evaluation

In order to evaluate the proposed hardware scheduler, we developed a multicore architecture simulator to estimate the utilization of every core. This simulator was written in Python 3.6. In this simulator, the LST-based DDP cores and the multistage interconnection network composed of the switching modules with implementing the algorithm 1 are modeled as shown in Fig. 5. Algorithm 1 is executed for each core connected to the switching module. Then, by comparing $T'_{slack}(i)$, the task is allocated to cores with longer $T'_{slack}(i)$. In order to characterize the core performance, the degree of multiprocessing capability N_{th} and the circulation time of operand packets within the DDP can be set for each core. The tasks can be executed simultaneously as long as the activated packets do not exceed N_{th}.

To verify the estimated utilization of the simulator, we compared it with that of the post-synthesis circuit simulation that is described in [5]. The comparison results are shown in Table 1. Those results indicate that the developed simulator can estimate the utilization of the LST-based DDP core with errors of about 2 %. The simulation time was about several ten times faster than the circuit simulation time. It can be said that the developed multicore simulator can be utilized to estimate the utilization of the core with reasonable precision.

Therefore, we evaluated the proposed decentralized hardware scheduler by using this architecture simulator. The average utilization of all cores in this evaluation are summarized in Table 2. The real-time task set used here is listed in Table 3 [8]. This task set is multiplicated depending on the number of cores in the multicore system. Furthermore, we assume that only the switching modules belonging to the last stage adopt the LST scheduling policy described in Algorithm 1 while the remaining switching modules allocate the requested tasks to the cores based on the round robin. As seen in Table 2, the utilization of each core is almost similar in cases of 4, 8, and 16 cores. The task scheduling by the proposed scheduler works well. However, in case of over 32 cores, deadline misses sometimes occurred at some

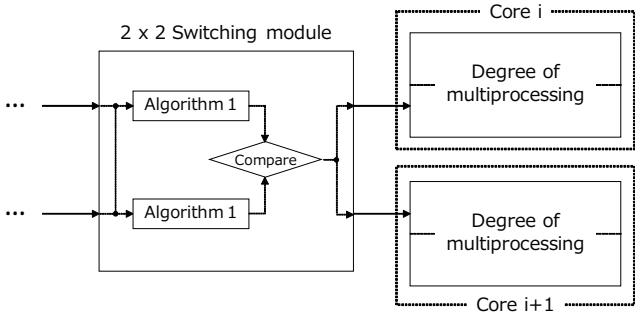

Fig. 5: Simulation model of switch and core.

Table 1: Comparison of utilization.

Clock period		500 ns		1 μs	
Degree of multiprocessing (N_{th})		2	4	2	4
Util.	Circuit sim.	32.1 %	21.6 %	31.8 %	21.6 %
	Architecture sim.	31.8 %	21.7 %	31.8 %	21.7 %

cores. This is because the round robin policy employed in the switching modules at the former stages might not be optimal in terms of load balancing. We need the further investigation of the decentralized real-time scheduling scheme.

5. Conclusion

This paper discussed a decentralized hardware scheduler for self-timed data-driven multiprocessor. The scheduler proposed in the papers can be implemented in the switching modules composing the multistage interconnection network. Since the scheduler autonomously operates only with the real time information carried by the requested task, it allows real-time tasks to keep their real-time constraints without any performance degradation of the multicore system. Furthermore, we developed an architecture simulator of our multicore system to evaluate the multicore system introducing proposed scheduler. Preliminary results indicated that the scheduler itself works well but its scalability is slightly poor. Thus we need further investigation to improve the scalability of the proposed decentralized scheduler.

Acknowledgement

Although it is impossible to give credit individually to all those who organized and supported our project, the authors would like to express their sincere appreciation to all the colleagues in the project.

The circuit design work was supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc. and Cadence Design Systems, Inc.

Table 2: Average utilization of core.

Taskset	4 sets	8 sets	16 sets	32 sets
# of cores	2	4	8	16
Degree of multiprocessing (N_{th})	2	2	2	2
Average utilization	54.5 %	54.1 %	52.9 %	– *

* : Deadline miss

Table 3: Task characteristics.

Name	Exec. Time $[ms]$	Period $[ms]$	Slack Time $[ms]$	Util.
T1 simple	0.06	2	1.94	3%
T2 monitor	0.10	10	9.90	1%
T3 compute	1.00	10	9.00	10%
T4 network	1.00	10	9.00	10%
T5 service	1.20	20	18.80	6%
T6 input	0.50	5	4.50	10%
T7 output	1.00	10	9.00	10%
T8 PWM	0.04	2	1.96	2%
			Total:	52%

References

[1] L. D. Xu, W. He, and S. Li, "Internet of Things in Industries: A Survey," IEEE Transactions on Industrial Informatics, Vol. 10, No. 4, pp. 2233–2243, Feb. 2014.

[2] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," Journal of the ACM, Vol. 20, No. 1, pp. 46–61, Jan. 1973.

[3] J. Hildebrandt, F. Golatowski, and D. Timmermann, "Scheduling Coprocessor for Enhanced Least-Laxity-First Scheduling in Hard Real-Time Systems," Proceedings of the 11th Euromicro Conference on Real-Time Systems, pp. 208–215, June 1999.

[4] H. Terada, S. Miyata, and M. Iwata, "DDMP's: Self-Timed Super-Pipelined Data-Driven Multimedia Processors," Proceedings of the IEEE, Vol. 87, No. 2, pp. 282–296, Feb. 1999.

[5] Y. Wada, K. Fukuda, and M. Iwata, "Least Slack Time Hardware Scheduler Based on Self-Timed Data-Driven Processor," Proceedings of the 2018 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'18), July 2018 (to be presented).

[6] A. K. Singh, P. Dziurzanski, H. R. Mendis, and L. S. Indrusiak, "A Survey and Comparative Study of Hard and Soft Real-Time Dynamic Resource Allocation Strategies for Multi-/Many-Core Systems," ACM Computing Surveys, Vol. 50, No. 2, Article 24, pp. 1–40, Apr. 2017.

[7] J. Hildebrandt, F. Golatowski, and D. Timmermann, "Scheduling Coprocessor for Enhanced Least-Laxity-First Scheduling in Hard Real-Time Systems," Proceedings of the 11th Euromicro Conference on Real-Time Systems, pp. 208–215, June 1999.

[8] Y. Tang and N. W. Bergmann, "A Hardware Scheduler Based on Task Queues for FPGA-Based Embedded Real-Time Systems," IEEE Transactions on Computers, Vol. 64, No. 5, pp. 1254–1267, May 2015.