# Least Slack Time Hardware Scheduler Based on Self-Timed Data-Driven Processor

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

**Abstract**— *Embedded system in recent years is required to provide higher functionality and higher performance. Therefore, embedded microprocessor must be composed of many cores supporting with real time processing. As for dynamic scheduling algorithms for the real time processing, earliest deadline first (EDF) and least slack time (LST) scheduling are typical ones. Since EDF has no predictability of the deadline miss, it cannot be used for multi-/many-cores. LST algorithm predicts the deadline miss even if there is overload condition in a certain core of them. However, if two or more tasks have similar slack time, frequent task switch called "thrashing" occurs and the scheduling overhead increases.*

*This paper focuses on data-driven processor (DDP) that can execute multiple processes without overhead of switching tasks, and proposes an LST scheduling circuit for a single core DDP toward development of the multi-core DDP system. Using a 65nm CMOS cell library, we conducted the post-synthesis evaluation of the DDP equipped with the LST scheduling circuit, and then we measured actual delay time and area of it. As a result, we revealed that the DDP equipped with the LST scheduling circuit could operate without task switch overhead, and decreased task switches by virtue of the multiprocessing feature of the DDP.*

**Keywords:** Hardware scheduler, Least slack time scheduling, Data-driven processor, Self-timed pipeline

## 1. Introduction

Modern internet of things (IoT) devices are required to provide higher functionality and higher performance so as to be permeated as edge-heavy computing devices in various application field. Usually its performance improvement has been achieved by increasing the CPU clock frequency, but this has caused the increase of power consumption and heat generation. Therefore, microprocessor employed in such IoT devices must be composed of many cores to consume lower power. Furthermore, IoT devices applied to the mission critical systems such as automobile control and so on are required to support real time processing. From those requirements, microprocessor architecture for the future IoT devices should be designed as a multi-core with real time processing feature.

When multiple tasks with real time constraints such as deadline are requested to be executed, the execution of a requested task is scheduled in taking account to priorities of waiting tasks. In general, real time tasks in IoT devices are periodic or aperiodic. The aperiodic task cannot be treated with under the static scheduling scheme because the task priority is not updated in run time. As for dynamic scheduling algorithms for the real time processing, earliest deadline first (EDF) [1] and least slack time (LST) [2] scheduling are well-known scheduling policies. In the EDF scheduling, priorities are assigned in order of deadline; the highest priority task is the one whose deadline is nearest in time, and the lowest priority task is the one whose deadline is farthest away. 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.

In the case of real time scheduling in multi-/many-core, EDF has no predictability of the deadline miss since EDF does not care execution time when assigning tasks to cores. In the LST scheduling, predicts the deadline miss can be predicted along with the slack time, i.e., whether it is under zero or not. However, if two or more tasks might have similar slack time, frequent task switch called "thrashing" will occur and the runtime scheduling overhead will increase [3] [4].

This study focuses on a self-timed data-driven processor (DDP) that can execute multiple processes without overhead of switching tasks. In the DDP, multiple operations without data dependency can be executed in parallel. Therefore, it is possible to reduce "Thrashing" which is a disadvantage of the LST scheduling by simultaneously executing low priority task and high priority task without context switch. This paper describes a LST-based hardware scheduler based on the single core DDP toward development of the multi-core DDP system.

## 2. Real-Time Architecture of DDP

### 2.1 Data-Driven Processing Model

In the data-driven processing model, a program is defined by a set of nodes and a set of arcs as a dataflow graph as shown in Fig. 1. A node represents an operation to be executed, and an arc represents a data dependency between nodes. An operation within the data-driven program is ready to be executed when a set of all necessary operand data for executing the operation are available. After the operation is executed, its resultant data is fed to the next node as an

ISBN: 1-60132-487-1, CSREA Press ©

argument of its operation. Thus, nodes that do not have dependencies each other can be executed in parallel as long as the processing resource is enough to execute parallel nodes. That is, multiple real time tasks can be executed in parallel as well if they are independent in terms of data dependency. However, once real time tasks are requested beyond the parallel processing resource of the microprocessor, a real time scheduling is indispensable to guarantee their real time constraints.

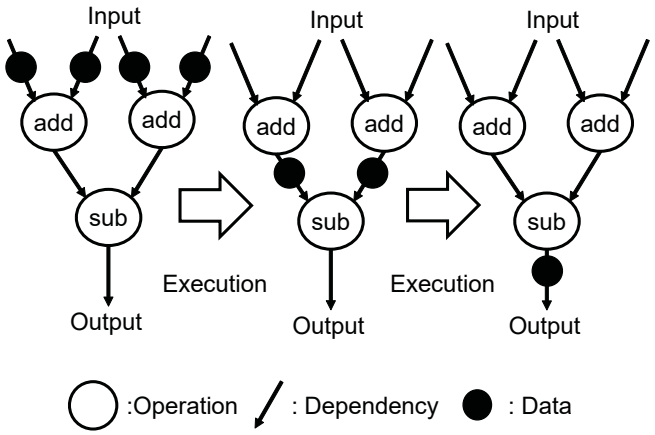



Fig. 1: Data-driven processing model.

## 2.2 Data-Driven Processor Architecture

Data-driven processor (DDP) has a high affinity to the self-timed pipeline (STP) because of their autonomous operation mode [5]. Fig. 2 shows a basic circuit of the self-timed pipeline. The data transfer within the STP is controlled by the data transfer circuit called C element. The C element ($C_i$) between the adjacent pipeline stages sends the data transfer request signal ($send_i$) to the succeeding pipeline stage and receives the data transfer acknowledge signal ($ack_i$) from that stage. When the data transfer request signal ($send_i$) is sent to the C element ($C_i$) in a transferable state, the C element ($C_i$) outputs a data latch opening signal ($CP_i$). The data latch ($DL_i$) that received $CP_i$ transfers data to data processing circuit ($Logic_i$). The pipeline stages without having valid data do not consume dynamic power at all, i.e., the STP circuit consumes power only during data transfer. Thus, the distinctive feature of the STP is its autonomous power saving.

Basic architecture of DDP shown in the Fig. 3 is composed of the following circular pipeline modules and it can be realized based on above STP circuit.

- Merge Unit (M)
  The merge unit plays a role of the confluence of external input packets and internal circulation packets. Those packets are output to CST in order of arrival.
- Constant Memory (CST)
  The constant memory unit stores constants each of which is fetched for an immediate value of an operation. In case of a constant operation, a constant addressed by a node identifier is packed into the incoming packet from the memory. Otherwise, the packet passes through the CST.
- Matching Memory (MM)
  The matching memory is an associative memory with multiprocessing context. In case of a binary operation, the operand packet is temporarily stored in the associative memory of the MM. When the another operand packet arrives, the stored packet is read out from the MM and those two operands are packed into a operand-pair packet.
- Arithmetic Logic Unit (ALU)
  The arithmetic logic unit performs arithmetic and bit-wise operations on integer binary numbers along with the instruction code held by the packet. In case of load or store operation, the ALU calculates the data memory address to add base address and offset. The resultant

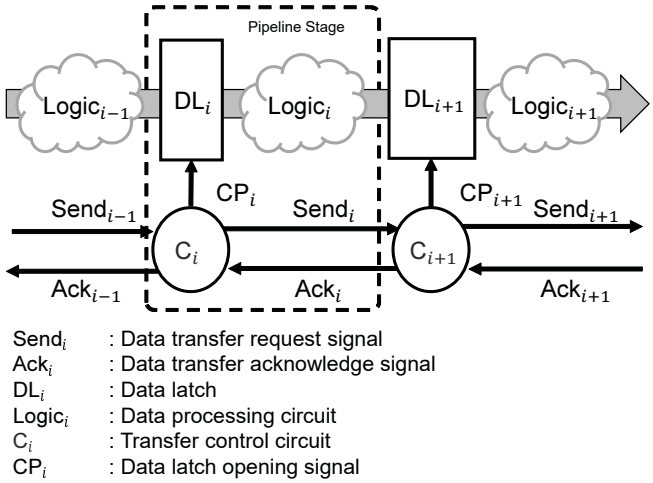



Fig. 2: Basic structure of STP.

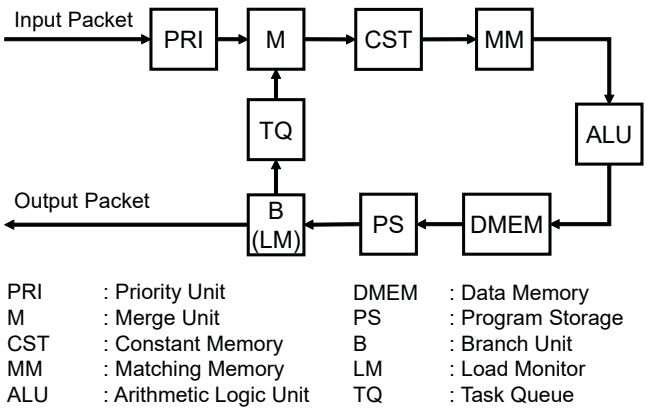



Fig. 3: Architecture of DDP.

packet is transferred to the next module, DMEM.

- Data Memory (DMEM)
  In case of a load or store operation, the data is read/ written from/ to the data memory.
- Program Storage (PS)
  The program storage stores the next destination node and the next operation code in the memory. The destination node and the operation code field of the incoming packet are replaced with the data read from the PS and the outgoing packet behaves as the next operand packet.
- Branch Unit (B)
  The branch unit judges whether to output the packet externally or again around the pipeline from the information held by the packet and transfers it to the proper direction.

As explained above, every packet behaves along with data dependencies defined in a program or programs. In other words, the number of activated packets in the circular pipeline indicates the degree of parallel processing load at that time. In case of real time processing, before the number of activated packets exceeds the capacity of the pipeline, an optimal scheduling is needed to prevent the DDP from overloading. From this viewpoint of real time scheduling, the priority unit, load monitor and the task queueing unit have been introduced into the self-timed DDP [6].

### 2.3 Least Slack Time Scheduling for DDP

In the dynamic scheduling policy, the priority unit (PRI) decides the initial priority of a requested real time task and produces a requested task packet with input data. The load monitor (LM) observes the processing load derived from the number of packets being processed in the DDP. When a packet is input from the outside or packet copy operation is executed, the number of packets increases. When a packet is output from the DDP, a packet is eliminated by a packet absorb operation, or two operand packets are packed into an operand-pair packet in MM, the number of packets decreases. The LM maintains the number of packets along with those increase and decrease events.

The task queue (TQ) manages activated packets to stop or resume real time tasks. The TQ maintains a queue for waiting (resumed) tasks along with the control logic shown in Table 1. In the table, $N_{AP}$, $N_{LM}$, and $N_{th}$ denotes the number of currently executed active packets, counter value in the load monitor, and the upper threshold of $N_{AP}$ respectively. As mentioned earlier, there are several factors affecting the number of active packets flowing in the DDP. $N_{AP}$ is proportional to the multiplicity of currently executed tasks. If the factors increasing $N_{AP}$ is detected, the priority queueing will be conducted based on $N_{th}$. $N_{LM}$ counted at the load monitor indicates the sum of $N_{AP}$ and the number of queued packets ($N_{QP}$). If the factors decreasing $N_{LM}$ is detected, the highest priority packet is dequeued in place of the gone packet. In this case, if the queue is empty, the incoming packet will be erased.Otherwise, the packet arriving at the queueing unit will normally pass through the unit as long as $N_{AP}$ does not exceeds $N_{th}$ . When passing through the unit, if the queue is not empty, the packet with the highest priority is dequeued in place of the incoming packet.

Table 1: Control of scheduling queue

| | $N_{LM}$ | | |
|---|---|---|---|
| | up | down | stable |
| $N_{AP} < N_{th}$ | pass | dequeue** | pass |
| $N_{AP} = N_{th}$ | pass | - | pass* |
| $N_{AP} > N_{th}$ | enqueue | - | enqueue |

* If ($N_{QP}$ != 0) then dequeue the highest priority packet.
** If ($N_{QP}$ = 0, del or out) then erase.
** If ($N_{QP}$ = 0, matched) then pass.
** If ($N_{QP}$ != 0, matched) then dequeue the highest priority packet.
$N_{AP} = N_{LM} - N_{QP}$

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 DDP core in multi core DDP system, it is possible to predict whether the target core will violate the deadline of the task or not. In this paper, a hardware scheduler for the LST scheduling policy is introduced in the DDP core. In this case, the slack time ($T_{slack}$) of a requested task must be calculated as its initial priority in the PRI. This slack time 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.

In the LM, the up-down counter is updated to inform the number of packets ($N_{LM}$) being processed in the DDP. In the TQ, waiting tasks are queued. Whenever the intermediate resultant packet arrives at the TQ, the slack time of every waiting task in the TQ should be updated and the queueing operation of the task is decided along with Table 1.

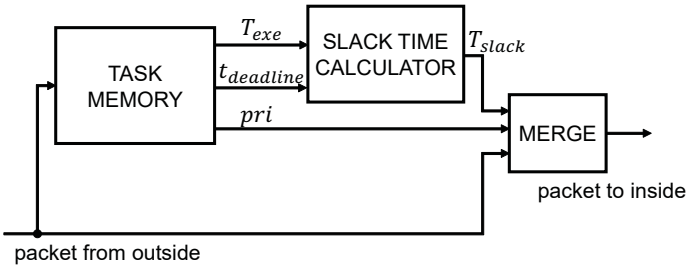
## 3. LST hardware scheduler Based on DDP

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. Once the task execution request in the DDP arrives at the DDP, the priority unit (PRI) calculates its initial slack time by referring to present time of the system clock and the requested task can be accepted to be executed in the DDP. During task execution, the load monitor (LM) observes the processing load within the DDP and the task queue (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.

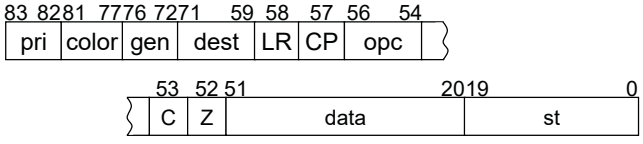
### 3.1 Slack Time Calculator

The initial slack time of a requested task is calculated with the priority unit of Fig. 3. This is called a slack time calculator (STC). The configuration of the STC circuit is shown in Fig. 4. In the STC circuit, the task memory stores relative deadline $t_{deadline}$, execution time $T_{exe}$, and priority class $pri$, which are indexed by task identifier of the task request packet. The slack time calculator calculates the initial slack time $T_{slack}$ by subtracting $t_{deadline}$ from $T_{exe}$. with the execution time and relative deadline. Finally, $pri$ and $T_{slack}$ are appended to the task request packet at the merge unit. The basic operand packet format flowing in the DDP is shown in Fig. 5. Here, $pri$ denotes a class of real time task, which is hard real time, soft real time, or normal task. Only in case of hard or soft real time class, the $T_{slack}$ will be effective in the DDP.




$T_{exe}$ : execution time $pri$ : priority class
$T_{deadline}$ : relative deadline $T_{slack}$ : slack time

Fig. 4: Slack Time Calculator.




pri : priority class
color : Identification
gen : Generation
dest : Destination address
LR : Left or Right flag
CP : Copy flag
opc : Operation code
C : Carry over flag
Z : Zero flag
data : Operation data
st : Slack time

Fig. 5: STC Packet Format.

### 3.2 Task Queue

The configuration of the task queue (TQ) circuit is shown in Fig. 6. TQ is mainly composed of priority queues (PQ's) and C element with packet eliminator (CE). When a packet (packet_in_q) arrives at the TQ, the packet is delivered to the priority queue corresponding to its priority class $pri$. Each PQ is associated with one of three priority classes in the figure. It is noted that the number of queues in the queueing unit can be adjusted to the required priority classes.

When there is enough space to execute a newly arrival task, its operand packet passes through the TQ without queueing. On the other hand, the real-time processing load is reaching closely to the multiprocessing capability of the DDP, the TQ controls to refrain from executing lower priority tasks. In this case, once the operand packet is enqueued in the priority queue, an operand packet whose priority is the highest is dequeued from the priority queue. The CE element can delete a packet transferred to the pipeline stage.

The TQ is composed of three controllers for managing the priority queues.

- CE_CTRL
  CE_CTRL controls whether a packet is queued based on information from the load monitor unit. When queueing the packet, CE_CTRL asserts the deletion control signal (del_ctrl_sig) to control the CE so that no packet is transferred to the next pipeline stage.
- Q_SEL
  Q_SEL controls the selector, MUX, which choses the highest priority packet from output packets of all priority queues. The count output from each priority queue indicates the number of the queued packets. If the count is zero, it implies the priority queue is empty.
- Q_CTRL
  Q_CTRL controls to delete a packet from any priority queue based on information of the previous packet state (PPS) module. The PPS module manages whether the previous packet is enqueued or transferred to the next pipeline stage. If the packet is forwarded to the next stage, it is necessary to remove the packet from the PQ. In that case, the packet deletion signal (packet_del_sig) to one of the PQ's is asserted to delete the packet from that queue.

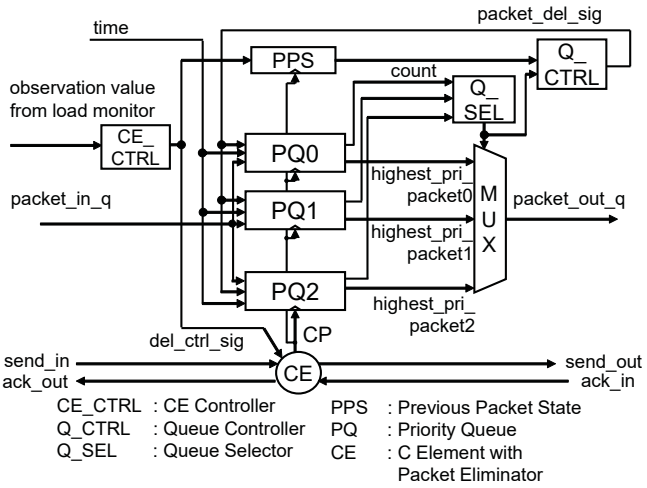



Fig. 6: Task Queue Circuit.

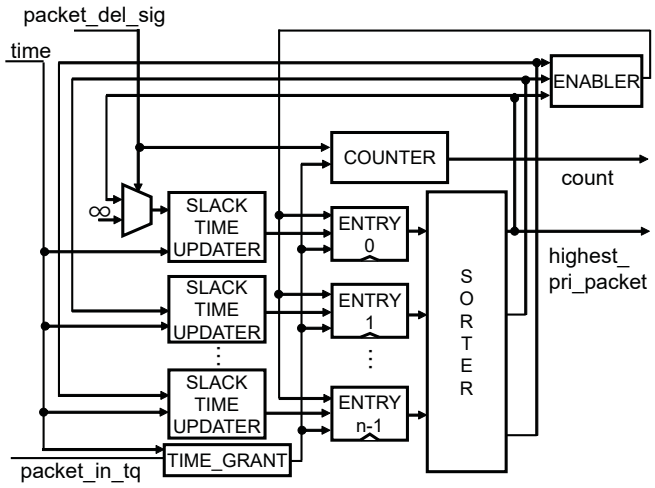



Fig. 7: Priority Queue Circuit.

### 3.3 Slack Time Updater

When a packet arrives at the TQ, it is queued in the queue corresponding to the priority class held by the packet. The configuration of the priority queue (PQ) is shown in Fig. 7. A packet arriving at TQ is queued in one of ENTRY's. ENTRY is controlled by write enable signal to grant its write permission to one of empty ENTRY so that the same packet is not queued to multiple ENTRYs, and the other ENTRY not to grant write permission. Packets queued in ENTRYs are sorted for the packet with the shortest slack time. The packet with the shortest slack time is output from the TQ. Otherwise, when the next packet arrives at the TQ, the slack time updater updates the slack time. The slack time of every queued tasks are updated when a packet arrives at the TQ. To update it is to subtract the time queued in priority queue from its previous slack time. By doing that, it is possible to suppress the amount of LST information to put in the queue.

## 4. Evaluation

In order to evaluate LST hardware scheduler based on DDP, it was designed using a 65 nm CMOS standard cell library and simulated after its synthesis. These circuits are described in Verilog-HDL and synthesized by Design Compiler, Synopsys Inc. Their specifications are summarized in Table 2.

Table 2: Design specifications of real-time DDPs.

| Process | SOTB 65nm CMOS ($V_{DD}$ : 0.75V) |
|---|---|
| Stages of STP ring | 10 stages |
| Queue | 85 bit × 8 words × 3 queues (EDF)<br>105 bit × 8 words × 3 queues (LST) |
| Constant memory | 35 bit × 8192 words |
| Matching memory | 85 bit × 96 words (EDF)<br>105 bit × 96 words (LST) |
| Data memory | 32 bit × 8192 words |
| Program storage | 20 bit × 8192 words |

Table 3 shows the total cell area of DDPs with EDF or LST scheduling. As shown in the table, the area of LST hardware scheduler based on DDP is about 1.13 times larger than that of EDF hardware scheduler based on DDP. This is because a circuit for update the slack time of all queued tasks in the TQ is additional and its data packet is extended for preset slack time field (20 bit) of the task so that the width of data path and queueing hardware of packets need more hardware cost.

Table 3: Area cost of synthesized DDPs.

| | EDF based on DDP | LST based on DDP |
|---|---|---|
| Cells | 5230930 | 5888726 |
| Area[$mm^2$] | 0.262 | 0.295 |

We evaluated how much thrashing can be alleviated by virtue of the multiple processing capability of the DDP. Five consecutive tasks consisting of 20 instructions with data dependence are executed on the DDP. The degree of multiprocessing capability is set to 1 or 4. This means that there is the case where multiprocessing does not be performed or the other case where multiprocessing is possible. We measured the number of task switching and the execution completion time. The results are shown in Table 4.

Table 4: Task switching and execution completion time

| | EDF | | LST | |
|---|---|---|---|---|
| Degree of multiprocessing | 1 | 4 | 1 | 4 |
| No. of task switches | 4 | 1 | 72 | 17 |
| Exec. completion time [ns] | 24972 | 10243 | 24972 | 7644 |

In case of the sequential processing, i.e., the degree of multiprocessing is 1, the number of task switching in the LST-based DDP was more than that in the EDF-based but the execution time in both schedulers is same. This result indicates that the proposed real-time DDP can cope with real-time tasking without task switch overhead even with different scheduling.

In case that the degree of multiprocessing is 4, the number of task switching in the LST-based DDP decreased compared with the above case. This result indicates that the multiprocessing capability of the DDP can reduce the number of task switching caused by the LST scheduling algorithm. Furthermore, the LST-based DDP can complete the execution of all tasks faster than the EDF-based DDP. This is because the priority of a task, i.e., its absolute deadline, is fixed once the task is initiated under the EDF scheduling policy. On the other hand, in case of the LST scheduling policy, the priority of a task, i.e., its slack time, might be raised relatively while other tasks are executed. As a result, a queued task might have a chance to preempt other executed tasks before one of the executed tasks is completed. This result implies that the LST-based DDP thus utilizes its

multiprocessing capability more efficiently than the EDF-based DDP.

We evaluated the designed LST hardware scheduler with a set of benchmark tasks described in [7]. Table 5 summarizes a set of parameters of each real-time task. The execution time and period in the table are predefined for each task. The period means the relative deadline of the task. In this task set, the hyper period is 20 ms. The utilization of each task is calculated by $(executiontime \times hyperperiod)/period$. In case of the sequential processor, the theoretical total utilization of this task set can be calculated by normalized sum-of-products of all execution times and frequencies and it is 52%. The degree of multiprocessing capability of the DDP $N_{th}$ is set to 2 or 4. The clock period for referring present time from the timer is set to 500 ns or 1 us. Each task is randomly requested during the beginning time 5 ms of the circuit simulation. The measurement results are shown in Table 6. The utilization of the DDP equipped with both schedulers are lower than that of the sequential processor so as to accept more real-time tasks. This is because the DDP can execute multiple tasks simultaneously. Also, the utilization and the number of task switches of the DDP equipped with LST scheduler are same as those of EDF one. This is because the task set is not so heavy for the DDP and thus LST algorithm behaves like the EDF.

Next, we suppose to request all tasks at the same time at the beginning time of the simulation. After that, they are executed periodically along with each period listed in Table 5. Since the external input port of the DDP is realized by the self-timed pipeline, each task request packet is fed in the interval time 260 ns. The results are shown in the Table 7. Utilization can be decreased in comparison with the former simulation. The utilizations of the EDF and LST schedulers are almost same. This is because the DDP can execute multiple tasks defined by $N_{th}$ and the remaining tasks are enqueued in the priority queue. The result indicates that both schedulers work well under the real time scheduling algorithm and multiprocessing capability of the DDP can be extracted by the hardware scheduler.

Table 5: 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% |

Table 6: Real time task execution results (Naive Condition).

(a) EDF

| Clock period | $500ns$ | | $1\mu s$ | |
|---|---|---|---|---|
| Degree of multiprocessing | 2 | 4 | 2 | 4 |
| No. of task switches | 10 | 0 | 10 | 0 |
| Utilization | 41.09% | 38.96% | 41.09% | 38.96% |

(b) LST

| Clock period | $500ns$ | | $1\mu s$ | |
|---|---|---|---|---|
| Degree of multiprocessing | 2 | 4 | 2 | 4 |
| No. of task switches | 10 | 0 | 10 | 0 |
| Utilization | 41.09% | 38.96% | 41.09% | 38.96% |

Table 7: Real time task execution results (Heavy Condition).

(a) EDF

| Clock period | $500ns$ | | $1\mu s$ | |
|---|---|---|---|---|
| Degree of multiprocessing | 2 | 4 | 2 | 4 |
| No. of task switches | 30 | 13 | 30 | 13 |
| Utilization | 31.82% | 21.23% | 31.82% | 21.23% |

(b) LST

| Clock period | $500ns$ | | $1\mu s$ | |
|---|---|---|---|---|
| Degree of multiprocessing | 2 | 4 | 2 | 4 |
| No. of task switches | 454 | 156 | 271 | 95 |
| Utilization | 32.05% | 21.64% | 31.80% | 21.60% |

# 5. Conclusion

This paper proposed a least slack time hardware scheduler for the self-timed data-driven processor (DDP). Since the DDP implements the dynamic data-driven processing scheme directly, it can execute multiple tasks at the same time as long as the amount of task requests do not exceed the multiprocessing capability of DDP. Because the DDP is equipped with the proposed hardware scheduler that consists of Priority unit, Load Monitor, and Task Queue, multiple tasks are scheduled along with their slack times as priorities.

We designed the proposed least slack time hardware scheduler by using 65 nm CMOS standard cell library and its post-synthesis evaluation has been conducted to measure the real time performance of the LST-based DDP. The evaluation result on the benchmark task, the proposed hardware scheduler demonstrates the multiprocessing capability with the priority scheduling when a set of tasks is requested. Thrashing that occurs when there is a task with the same slack time can be alleviated by multiple processing. In this case, it can be said that the scheduling overhead caused by thrashing has no influence to the real time performance. A set of benchmark tasks used in the evaluation is a typical example. Therefore, it is necessary to measure and evaluate details of the operation in the proposed scheduler by using more practical benchmark sets. Furthermore, it is necessary to apply the proposed core to real-time multi-core system and evaluate it. As for the feasibility study on real-time multi-core architecture for future IoT devices, we are now trying to propose and evaluate a decentralized hardware scheduler for the self-timed data-driven multiprocessor [8].

## 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.

## References

[1] 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.

[2] A. K. Mok, "Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment", Ph.D. Dissertation, MIT, 1983.

[3] V. Salmani, M. Naghibzadeh, and A.H.Taherinia, "Performance Evaluation of Deadline-based and Laxity-based Scheduling Algorithms in Real-time Multiprocessor Environments", in Proc. the 6th WSEAS International Conference on Systems Theory and Scientific Computation (ISTASC'06), August 2006.

[4] J. Hildebrandt, F. Golatowski, and D. Timmermann, "Scheduling Coprocessor for Enhanced Least-Laxity-First Scheduling in Hard Real-Time Systems", in Proc. 11th Euromicro Conference Real-Time System, pp. 208–215, 1999.

[5] 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.

[6] K. Fukuda, H. Shibuta, and M. Iwata, "Priority-Based Hardware Scheduler for Self-Timed Data-Driven Processor," in Proc. the 2017 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'17), pp. 245–251, July 2017.

[7] 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.

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