Priority-Based Hardware Scheduler for Self-Timed Data-Driven Processor

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

Abstract— *The embedded systems are often required to possess real-time characteristics in order to satisfy the time constraints of individual tasks. It is necessary for the system to schedule resource allocation for the tasks in consideration of the priority of each task. In most of embedded microprocessors, a context switch occurs to switch the task being executed preemptively and it leads performance overhead.*

This study focuses on a data-driven model to extinguish context switching in real-time processing. This paper presents priority-based schedulers for a self-timed data-driven processor (DDP). The proposed hardware schedulers are composed of a deadline fix unit, a load monitor, and a queueing unit and the real-time DDP is equipped with them. The evaluation results on benchmarking tasks indicates that both schedulers are lower than the ideal utilization. This indicates that both schedulers exhibit real-time processing capability enough to the room for more task requests.

Keywords: priority-based scheduling, hardware scheduler, self-timed pipeline, data-driven processor

1. Introduction

In recent years, embedded systems are spreading in various devices of home appliance, industrial equipment, and so on. The embedded systems are often required real-time characteristics. Such a system is called a real-time system. In a mission critical system, if a certain error recovering could not be functioned before the required time, there is a danger of leading to serious accident. In this way, in the real-time system, it is necessary to keep not only the correctness of its output data but also the time constraint (deadline) of its controlled object when outputting the output after receiving an input. Especially, in modern intelligent robotics and vehicles, more stringent time constraints are required due to more advanced functions with higher speed control.

The real-time system has aspects of priority control, and it is classified into hard real-time task, soft real-time task, and firm real-time task depending on the rigidity of the deadline to be controlled. Therefore, when a new execution request of a task is issued during execution of a current task, it is necessary to schedule resource allocation for the tasks in consideration of the priority of each task. There are typical scheduling algorithms such as rate monotonic (RM) [1], earliest deadline first (EDF) [1], and least slack time (LST) [2] scheduling. RM is a static priority scheduling algorithm, and the priority once assigned to the task does not change during system execution. In the RM, the highest priority is assigned to the task with the shortest period. The EDF and LST algorithms are categorized in dynamic priority scheduling because the priority of task changes during system execution. In the EDF, priority of a task is assigned according to its absolute deadline, i.e., the task with the earliest absolute deadline is assigned the highest priority. In the LST, the highest priority is assigned to the task with the shortest slack time. In these scheduling algorithms, when the task whose priority is higher than that of the currently executed task is requested, the higher priority task is preferentially executed. In most of embedded microprocessors, a context switch occurs to switch the task being executed preemptively because they are based on the Von Neumann sequential processor. In this case, the more preemptive executions are required, the more frequent task switching occurs. These task switching overhead leads to degradation of effective utilization for the task execution.

Therefore, this study focuses on a data-driven model of computation that is one of the promising models representing intrinsic parallelism by a directed graph so as to extinguish context switching in real-time processing. This paper discusses priority-based scheduling methods for a self-timed data-driven processor (DDP) and proposes their hardware implementations as priority-based hardware schedulers. Since the DDP operates according to data-driven model, it is possible to execute the low priority task and the high priority task simultaneously without any context switch. Therefore, real-time processing performance of the DDP is potentially superior to that of the sequential processor. This advantage is further enhanced by the synergetic effect of self-timed pipeline (STP) circuit implementation of the DDP because of familiarity between the STP and the data-driven model. However, the amount of hardware resource in any computing system is actually finite, so that even in the DDP, there are upper limit for executing multiple tasks. When exceeding the upper limit of hardware resources for executing current tasks in multiple, it is necessary to stop one or more appropriate task(s) temporarily and resume them after releasing the resource. Therefore, the proposed priority-based hardware schedulers support those stop and resume mechanism based on required task priorities.

ISBN: 1-60132-466-9, CSREA Press ©

2. Real-Time Architecture of DDP

2.1 Basic Architecture of DDP

The self-timed data-driven processor (DDP) discussed in this paper is basically designed to implement the dynamic data-driven processing architecture [3]. This architecture allows multiple contexts to be executed at the same time so that multiple data on an arc in a directed graph of the data-driven program must be distinguished along with each context. Therefore, the packet format representing data on the DDP is designed as shown in Fig. 1, where the *color* field holds a program identifier, the *gen* field holds a generation to identify the order of an element within a stream of multiple data, and the *dest* field denotes an destination node in the data-driven program. In terms of real-time processing, *color* identifies a task (a program) and *gen* indicates the activation order of the task (an instance of the task).

Fig. 2 shows a basic architecture of the DDP. Since the DDP performs multiprocessing, multiple data packets explained above flow on a circular pipeline structure composed of basic elemental units. Each unit of the DDP operates as follows:

- Merge Unit (M)
  The merge unit is a conflux part of external input packets and internally circulated packets. If packets from both directions arrive at the same time, this unit arbitrates appropriately, e.g., first-come-first-service, etc.
- Constant Memory (CST)
  The constant memory unit receives an operand packet with its operation code. If it is an operation with a constant value, the constant value fetched from the memory is appended to the operand packet. Otherwise, the operand packet passes through the CST without accessing the memory.
- Matching Memory (MM)
  The matching memory is an associative memory with multiprocessing context. The context of the real-time task is distinguished by color, generation and destination field of the incoming packet. The packet is temporarily stored in the associative memory of the MM. When its paired packet with the same context arrives, it generates an operand-pair packet from an incoming packet and an associated packet stored in the MM.
- Arithmetic Logic Unit (ALU) and Data Memory (DMEM)
  The arithmetic logic unit performs the arithmetic logic operation according to the operation code of the packet. In the case of load or store instruction, the ALU calculates the data memory address and appends it to the packet. By using this memory address, the data memory unit performs memory accesses for load or store instructions.
- Program Storage (PS)

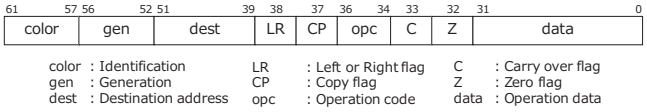

Fig. 1: An operand packet format in DDP.

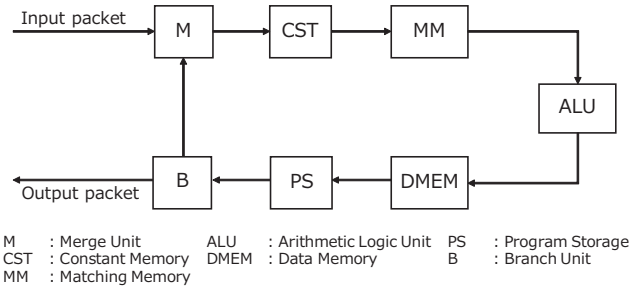

Fig. 2: Basic architecture of DDP.

  The program storage holds a data-driven program to be executed. The destination node in the result packet indicates the address for the next instruction. The PS concatenates the resultant data with the next operation code and destination node fetched from the program storage.
- Branch Unit (B)
  The branch unit outputs the packet to the outside or inside of the circular pipeline depending on its branch flag of the packet.

The circuit implementation of the DDP is basically the self-timed pipeline (STP) which realizes self-timed data transfer control among pipeline stages. Thus, the operation of the data-driven model is faithfully realized on a packet basis. As shown in Fig. 3, 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. Since the pipeline stage without effective data does not consume electric power, it can save the operating power autonomously. In addition, the STP itself has elasticity in terms of packet transfer so that it would flexibly mitigate the processing load fluctuation when the real-time application works on the DDP [3].

2.2 Real-Time Processing Features of DDP

Although the DDP directly realizes the data-driven multiprocessing and thus executes multiple prioritized tasks simultaneously, multiprocessing resource limitation cannot be exceeded in principle. In the case that the real-time tasks require processing resource beyond the multiprocessing capability of the DDP, task scheduling is necessary to eliminate the overload as well as to guarantee the time constraints of every task. In the DDP, the processing load can be easily

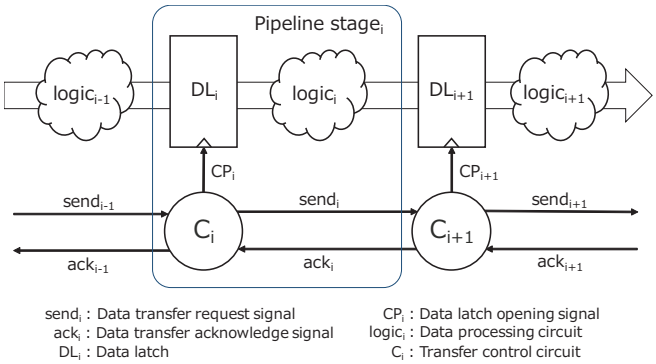

Fig. 3: Basic structure of STP.

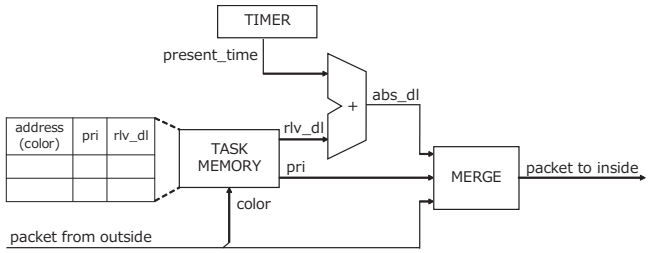

Fig. 4: A block diagram of deadline fix unit (DF).

observed as the number of packets flowing the circular pipeline. If the number of packets activated by all of the tasks can be estimated or predicted, we can decide whether a new task is possible to be invoked under the resource limitation or not. If it is impossible to invoke the new task, we might replace one or more of the current tasks with it to guarantee the deadline time of the new task. In this case, the scheduler must stop the replaced task(s) and then resume it (them) after the processing resource is available. In order to achieve the highest real-time processing performance, it is necessary to realize the scheduling processing without impairing the inherent performance of the DDP. For that purpose, there are smart features inherent in the DDP as follows and those features are utilized in our schedulers described in the next section.

- Processing load observation
  Strictly describing, the number of active packets representing the processing load of the DDP can be counted from the snap shot of valid packet in each pipeline stage. In order to prevent the DDP from overloading, it is enough to count the number of packets passing through a certain pipeline stage within the DDP. In this case, the processing load monitoring in a single pipeline stage is simply realized by the STP because of its locality and the overload condition can be adaptively settled by the threshold.
- Execution pending of tasks
  In the DDP, packets temporarily wait in the matching memory until a packet associative with the waiting packet arrives. If one or more packets of the low priority task are evacuated to a sort of priority queue, the task can be autonomously transited to the pending state. This means that the high priority task can be invoked in place of that.

3. Priority-Based Scheduler for DDP

The real-time processing features discussed in the previous section 2.2 are helpful to improve the scheduling performance of the DDP. Therefore, this section proposes a couple of priority-based hardware schedulers for the DDP. Those schedulers should calculate absolute deadline of the task from its relative deadline and present time, and monitor the processing load of current tasks before preempting and resuming tasks by manipulating priority queues of the waiting tasks.

In this section, those basic hardware components are presented and then alternative configurations of the schedulers are proposed by combining those components for the DDP.

3.1 Absolute Deadline of Task

Since the rigidity of the deadline varies depending on the type of real-time task, its priority level and relative deadline may be predefined for individual task. The relative deadline of the task is from the time when the task is requested to the time when the task execution must be completed, while the latter time is called the absolute deadline. As mentioned in the previous section, the input packet of the DDP holds *color* as a task identifier. Therefore, the predefined priority level (*pri*) and relative deadline (*rlv_dl*) stored in a lookup table can be fetched by the task identifier as shown in Fig. 4. After that, absolute deadline (*abs_dl*) is calculated from them and the packet attached with it is supplied to the DDP.

3.2 Real-Time Processing Load Monitor

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 following factors increase or decrease of the packet flow rate in the DDP.

- Factors that increase packet flow rate
  - To accept an external packet.
  - To duplicate a packet.
- Factors that decrease packet flow rate
  - To output a resultant packet.
  - To delete a packet.
  - To generate an operand-pair packet in matching memory.

Those factors can be detected from the execution control flags held in the DDP packet so that the load monitor can observe the number of active packets within the DDP.

3.3 Priority Queueing Unit

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. As shown in Fig. 5, the queueing unit mainly consists of queues and coincidence flip-flop with packet eliminator (CE-element). The CE-element can delete a packet stored in the pipeline stage. These queues operate on the local clock supplied from the CE-element (*CP*). Fig. 5 illustrates a configuration example of the queueing unit where three levels of priority can be dealt with. The number of queues in the queueing unit can be adjusted to the priority levels needed in the target application.

When the input packet (*packet_in_q*) arrives in the queueing unit, the packet is delivered to the priority queue corresponding to its priority level (*pri*). The priority queue is structured as shown in Fig. 6, and *packet_in_q* is stored to one of the empty ENTRYs in the priority queue. ENABLER manages availability of all ENTRYs to queue packets. The packets stored ENTRYs are sorted in descending order of priority by SORTER, and the packet with the highest priority (*highest_pri_packet*) is output.

In addition, the queueing unit has several controllers to maintain several priority queues for the task scheduling.

- CE_CTRL
  CE_CTRL controls whether queueing packets based on information from processing load monitor. When queueing the packet, CE_CTRL asserts the deletion control signal (*del_ctrl_sig*) to control the CE-element 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 *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 PPS module. PPS module manages whether the previous packet is queued or transferred to the next pipeline stage. If the packet is forwarded to the next stage, it is necessary to remove the packet from ENTRY. In that case, the packet deletion signal (*packet_del_sig*) to one of the priority queues is asserted to delete the packet from that queue.

In the queueing unit, it is possible to adapt to different scheduling algorithms by changing sort key of priority queue and controller logic.

3.4 Alternatives of Scheduler

For the scheduling function, the processing load monitor and the queueing unit must coordinate in the DDP. The

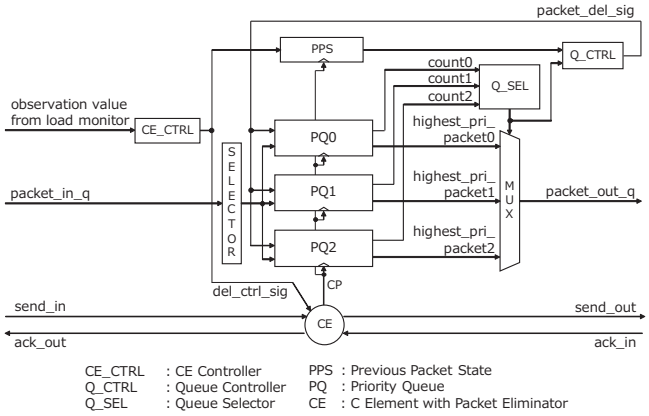

Fig. 5: A block diagram of queueing unit (Q).

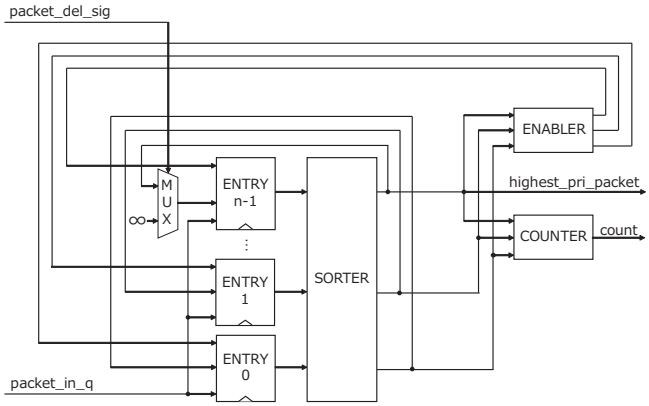

Fig. 6: A block diagram of priority queue (PQ).

possible location to attach the queueing unit to the DDP is at the external or internal portion. In this paper, the former is called input scheduling, and the latter is internal scheduling. Table 1 shows the comparison among typical scheduling methods and the proposed ones.

In order to simplify the hardware implementation of the proposed methods, we assume the following conditions; the first and second priority keys are the priority level and absolute deadline respectively, the multiplicity of executed tasks is proportional to the multiprocessing load of the DDP, and all tasks has single input.

(a) Input Scheduling

Fig. 7 shows a DDP architecture introducing the input scheduling method, in which a queueing unit is mounted at the input portion of the DDP. Invocation or postpone of task execution is controlled there. The queue control conditions are shown in Table 2. In the table, $\mathrm{N_{AP}}$, $\mathrm{N_L}$, and $\mathrm{N_{th}}$ denotes the number of currently executed active packets, counter value in the load monitor, and the upper threshold of $\mathrm{N_{AP}}$ respectively.

As mentioned in subsection 3.2, there are several factors

Table 1: Comparison of scheduling algorithm.

	RM	EDF	LST	Proposed method
Priority (static/dynamic)	Activation period (static)	Deadline (dynamic)	Slack time (dynamic)	Priority level (static) + Deadline (dynamic)
Highest priority	With shortest period	With earliest deadline	With shortest slack time	1st : highest level 2nd : earliest deadline
Parallelism grain	Thread level	Thread level	Thread level	Instruction level
Task type	Periodic task	Periodic/aperiodic task	Periodic/aperiodic task	Periodic/aperiodic task

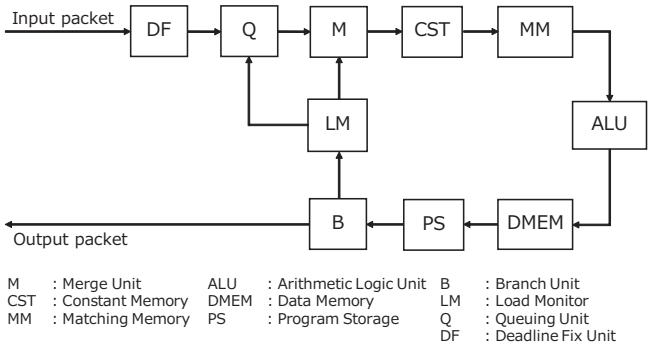

Fig. 7: Architecture of input scheduling.

Table 2: Control logic of input scheduling queue.

	N_L		
	up	down	stable
$N_{AP} < N_{th}$	pass*	dequeue**	–
$N_{AP} = N_{th}$	enqueue*	–	–
$N_{AP} > N_{th}$	enqueue*	–	–

* If (copy) then no operation.
** If (N_{QP} = 0, del or out) then no operation.
$N_{AP} = N_L$

affecting the number of active packets flowing in the DDP (N_{AP}). N_{AP} is proportional to the multiplicity of currently executed tasks. In order to deal with those factors, the queueing unit is controlled by decoding the execution control flags in the packet. If the factors increasing N_{AP} is detected, the priority queueing will be conducted based on N_{th}.

In this method, since the queueing unit is mounted outside the DDP, N_{AP} is equal to N_L. There is a possibility that packet may be stored in the queue only when the packet is input from outside. In this case, if $N_{AP} < N_{th}$, packet pass through the queue. If $N_{AP} \leq N_{th}$, packet is enqueued. If the factors decreasing N_{AP} is detected and the packets are stored in the queue, the packet is dequeued. In other cases, queue does not work. Table 2 summarizes the detail of those control logics for the queueing unit of input scheduling. In case of the input scheduling, once a task is invoked, it is not interrupted.

(b) Internal Scheduling

Fig. 8 shows a DDP architecture introducing the internal scheduling method, in which a queueing unit is mounted at the inside of the circular pipeline. If an intermediate data packet belonging to a task is captured and enqueued to the queueing unit, the operation nodes indicated by the destination node identifier of the packet will be suspended along with the data dependencies in the data-driven program. After the processing resource is available for the task, the captured packet is dequeued to be active and thus the execution of the task will be resumed.

In the DDP, there are some types of intermediate packets, i.e., operand packet, operand-pair packet, resultant data packet. In order to minimize circuit cost of the queueing unit, the shorter packet is the better to be queued. Furthermore, the queueing unit should be located to keep smooth packet flow within the circular pipeline. According to those design criteria, the queueing unit is placed between the branch unit (B) and the merge unit (M). That is, the resultant data packets are queued in this unit.

In this method, since the queueing unit is mounted inside the DDP, the packet flowing inside might be enqueued. Therefore, N_L counted at the load monitor indicates the sum of N_{AP} and the number of queued packets (N_{QP}). If the factors decreasing N_L 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 3 summarizes the detail of those control logics for the queueing unit of internal scheduling. It should be noted that in the case of internal scheduling, all packets must hold the priority level and absolute deadline for the task so that the packet size is larger than the case of input scheduling.

4. Evaluation

In order to evaluate both proposed hardware schedulers for the DDP, we designed two kinds of real-time DDPs using a 65nm CMOS standard cell library and simulated them at the post-synthesis design phase. These circuits are described in Verilog-HDL and synthesized by Design Compiler, Synopsys Inc. Their specifications are summarized in Table 4.

Table 5 shows the total cell area of DDPs with the input or internal schedulers. As shown in the table, the area of

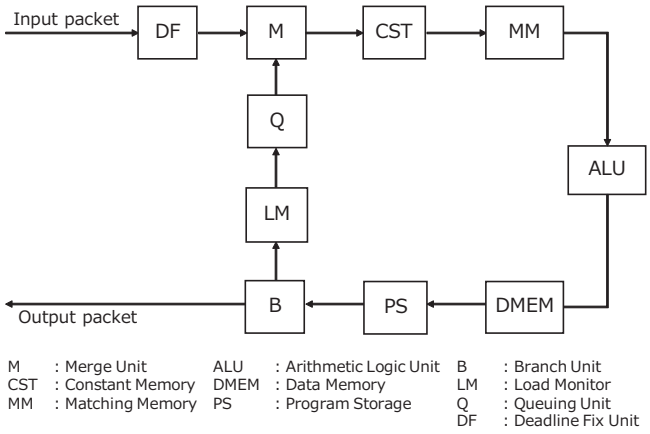

Fig. 8: Architecture of internal scheduling.

Table 3: Control logic of internal scheduling queue.

	N_L		
	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.
$N_{AP} = N_L - N_{QP}$

the DDP with the internal scheduler is about 1.3 times larger than that of the DDP with the input scheduler. This is mainly due to the difference of the packet size circulating in the DDP. Since the internal scheduling performs queueing process within the DDP, it is necessary to circulate packets in the DDP during maintaining priority level and absolute deadline of the packet. On the contrary, the packet for the input scheduling does not need any scheduling information. In addition, the internal scheduling may require more capacity of matching memory (MM) than the input scheduling because the intermediate operand packets must be held in the MM. This also causes an increase in circuit area. However, the internal scheduling enables flexible control of multiprocessing to suspend and resume the task even if the multiple tasks are running.

At the first step of the performance evaluation, the influence of the additional scheduling modules is analyzed.

Table 4: Specifications of two real-time DDPs.

Process	SOTB 65nm CMOS (V_{DD}: 0.75V)
# Stages of STP ring	10 stages
Queue	87 bit × 8 words × 3 queues
Constant memory	32 bit × 2048 words
Matching memory	65 bit × 64 words (input scheduling) 87 bit × 96 words (internal scheduling)
Data memory	32 bit × 8192 words
Program storage	10 bit × 8192 words

Table 5: Area cost of synthesized DDPs.

	Input scheduling	Internal scheduling
Cells	32120	38502
Area $[mm^2]$	0.173	0.222

This is because the effective throughput of the DDP circular pipeline depends on the pipeline stage that is composed of the longest critical path logic in the DDP. Table 6 shows the delay time of the matching memory and the queueing unit. As shown in the table, the delay time of the queueing unit is shorter than that of the matching memory. Therefore, the queueing unit can be adopted without degrading pipeline throughput of the original DDP.

Since the DDP can perform multiprocessing and its ALU can operate even while scheduling, it will have little influence on the DDP throughput performance. On the other hand, in terms of response time of the real-time DDPs, we measured the coordination time of the scheduling components. Table 6 shows the priority control time, i.e. the latency time from receiving a packet at the load monitor to deciding the priority scheduling result at the queueing unit. While the internal scheduling needs two pipeline stages for the priority control, the input scheduling needs five stages. Thus, the internal scheduling is faster than the input scheduling in terms of the priority control time.

As for the disturbance to internal packet flow within the circular pipeline, the internal scheduling is inferior to the input scheduling. Therefore, we measured the lap time of a packet within the DDP circular pipeline. As shown in the table, the lap time in the input scheduling was slightly shorter than that of the internal scheduling.

Table 6: Influence of scheduling modules.

		Input scheduling	Internal scheduling
Delay time of Q	$[ns]$	13.9	13.8
Delay time of MM	$[ns]$	15.5	23.9
Priority control time	$[ns]$	95.1	48.3
DDP lap time	$[ns]$	218.8	227.9

Q: Queueing unit, MM: Matching memory.

As discussed above, there are some pros and cons in both proposed scheduling methods for the DDP. The total performance effect of those may depend on the characteristics of target applications and their real-time constraints. Therefore, we tried to evaluate the proposed schedulers with a set of benchmark tasks by modifying evaluation tasks used in [4].

Table 7 summarizes a parameter set of real-time tasks used here. The execution time and period in the table are predefined for each task. Theoretically, the total utilization of this task set can be calculated by the normalized product-sum of all execution time and frequency. Thus, the ideal utilization of this task set is 52 %. Each task is started randomly between 0 and 5 ms. After that, they are executed

periodically. In this evaluation each task is assumed to be described in a straight line program. The measurement results shown in Table 8 indicate that the utilization of both scheduling methods are lower than the ideal utilization so as to accept more real-time tasks. This is because the DDP can execute multiple tasks simultaneously. The input scheduling has less influence on the internal packet flow in the DDP. Therefore, the input scheduling has a lower utilization than the internal scheduling. In the table, the first scenario is that there is no priority and only the deadline is dealt with in the schedulers and the second scenario is that T1, T3, and T5 in Table 7 are defined as high priority tasks and both the priority and deadline of the tasks are checked by the schedulers. In the case of the second scenario, both scheduling methods showed same utilization compared with the first scenario. This might be because there is no task request exceeding the processing load of the DDP. Therefore, we will evaluate the proposed scheduling methods in more severe scenarios as further works.

Table 7: Task characteristics.

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

Table 8: Utilization of DDPs.

Scenario		Input scheduling	Internal scheduling
Random	w/o priority level	44.3%	46.0%
	w/ two priority levels	44.3%	46.0%

5. Conclusion

In this paper, priority-based scheduling methods and their hardware implementation for the self-timed data-driven processor (DDP) are proposed. The DDP directly realizes the dynamic data-driven scheme and thus executes multiple tasks simultaneously. In the case that the tasks require processing resource beyond the multiprocessing capability of the DDP, the task scheduling is necessary to guarantee the time constraints of every task. The proposed hardware schedulers are composed of a deadline fix unit, a load monitor, and a queueing unit and the real-time DDP is equipped with them. The possible location to attach the queueing unit to the DDP is at the external or internal portion. Along with the location, we designed the proposed hardware schedulers: the input schedulers and the internal schedulers.

Both proposed hardware schedulers for the DDP are evaluated based on 65 nm CMOS standard cell library. The evaluation results on benchmarking tasks indicates that both schedulers have real-time processing capability enough to the room for more task requests. Since the input scheduling has less influence on the internal packet flow in the DDP, its utilization is lower than the internal scheduling. However, the evaluation conditions reported here are limited and the set of benchmark tasks used in the evaluation is one of typical examples. Therefore, the detail of behavior in the proposed schedulers and their effect should be measured and evaluated more. Furthermore, more practical set of benchmarks should be employed in evaluation.

As for future works, the embedded real-time processor is needed to be aware to its energy. Therefore, the energy-aware scheduling should be investigated with introducing sophisticated low-power techniques such as dynamic voltage scaling and power gating [5][6].

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, 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] J. Hildebrandt, F. Golatowski, 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.

[3] 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.

[4] Y. Tang, 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.

[5] K. Miyagi, S. Sannomiya, M. Iwata, and H. Nishikawa, "Low-Powered Self-Timed Pipeline with Runtime Fine-Grain Power Supply," Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications, pp.472-478, July 2012.

[6] M. Bambagini, M. Marinoni, H. Aydin, G. Buttazzo, "Energy-Aware Scheduling for Real-Time System: A Survey," ACM Transactions on Embedded Computing System, Vol.15, No.1, Article No.7, pp.1-34, Feb. 2016.