Journal of Computers Vol. 27, No. 4, 2016, pp. 95-110
doi:10.3966/199115592016122704008

A Transaction-based Design Model and Its MPEG-2 Encoder Design

Yun-Lung Lee,[1,*] Jer Min Jou[1]

Department of Electronic Engineering, National Cheng Kung University
Tainan 701, Taiwan
jou@j92a21.ee.ncku.edu.tw, jackyli2k@gmail.com

Received 13 May 2015; Revised 13 August 2015; Accepted 22 November 2015

Abstract. The speculative circuit is a kind of special parallelizing circuits which could execute the sequential operations in parallel by breaking the dependencies between them through the prediction approach. That is, the consumer operations no longer needs to wait the computed results of the corresponding product ones by predicting them, so that the consumer operations can be executed with the corresponding product ones at the same time. However, the predicted value is speculatively generated, so there still needs an additional recovery mechanism which will recover the speculative execution when the predicted value is detected not to match the computed one. Therefore, the dedicated conflict management mechanism and data synchronization mechanism are needed when designing a speculative circuit. In this paper, we present a new design model called "transactional speculative circuit model" (TSCM), which is based on the concept of transactional memory (TM), to design the speculative circuits. Moreover, this model also can be used to design non-speculative parallelizing circuits. We use TSCM to design a new speculative MPEG-2 encoder, and the experiment results show the performance can be efficiently improved.

Keywords: MPEG-2, speculation, transactional memory

1 Introduction

In recent years, the researches of efficient parallel execution have become one of the important fields in SoC (System on Chip) design. How to aggressively explore the concurrency and efficiently utilize the function units of a SoC system are the significant design challenges. The maximum utilization of the function units of a SoC system is that all of them keep running without idling. But, actually, the situation of the full utilization of the function units rarely occurs due to existing the control and data dependencies between operations allocated at the function units. Transactional memory (TM) [1-4] has the data management mechanisms and conflict management mechanisms which can be used for resolving the dependence conflict and ensuring the data synchronization among the function units in a SoC system.

In order to increase the utilization rate of the function units, the speculative execution has been proposed which predicts the values of control and/or data dependencies among the program components and will allow more functional units executing in parallel. If the predicted ones is hit, then the performance of SoC could be efficiently improved. However, if any miss occurs, then the executing function units must be discard and recovered correctly. Therefore, efficient design of a SoC with a speculative style will be confronted with the problem for speculative data synchronization. Hence, a suitable mechanism for their synchronization is required.

Besides, we also take the variable executing latency for each speculative function unit into consideration. The variable executing latency will lead to unexpected conflicts during speculative execution. Therefore, a suitable mechanism for conflict management of a speculative execution of SoC is also required.

* Corresponding Author

The rest of this paper is organized as follow. First, we will introduce the basic properties of transactional memory, and then propose a new transaction-based model. After that, the speculative MPEG-2 encoder by using the new model is designed. Finally, the experiment results are listed for explaining the performance improvement of MPEG-2 encoder.

2 Relative Work

2.1 Transactional memory

In the transactional memory [6, 9, 11] system, there are special mechanisms for data version management and conflict management to keep execution synchronization. "Conflict" is the case that more than two processes in the circuit access the same data at the same time and at least one process writes this data. Unlike the lock-based system, "transaction" is a basic unit for program processing and is unnecessary to use a lock circuit for exclusively accessing the data. Instead, the TM system will keep the atomicity and the isolation of transactions to keep data synchronization.

"Atomicity" means that a transaction legally commits its results when all operations in this transaction are executed completely and there is no conflict among the results. Otherwise, this transaction commits unsuccessfully and should be restarted. All results of it executing will be discarded or recovered. "Isolation" means that the modified data is not visible for other transactions until its transaction commits, and other transactions should only read the original value of the data.

To keep the two properties, data version management mechanisms and conflict management mechanisms are required during designing a TM-based circuit.

Version management. Data version management mechanism decides the position where to put the old value (original value) and new value (modified value) of the data to be synchronized 0. The eager strategy is that its old value is recorded in another (new) space and the new value will replace the old one and be used directly. The lazy strategy is that the new value is put in the new space and the old value will be replaced after the transaction commits.

Conflict management. Conflict management mechanism decides when to detect the data value conflicts and resolves these conflicts. The eager strategy is that the conflicts are detected during transaction executing and are resolved as soon as possible. The lazy strategy is that conflicts are detected during transaction executing or before transaction commit and are resolved until transaction committing.

Transactional memory detects the conflicts according to the read/write set of each transaction, and the read/write set is used to record what data are accessed by the corresponding transactions.

Nested transaction. The nested transaction is a special transaction which includes one or more transaction. The nested transaction is classified into three types: closed nested transaction, open nested transaction, and flattening nested transaction according to the isolation of the inner transaction.

The isolation of closed nested transaction includes all the inner transactions, and the committed data of inner transaction can only accessed by the other transactions which are included in the outmost transaction; the committed data is outside visible after the outmost transaction committing. The inner transaction should restart when the outmost transaction conflicts and aborts. And, the inner transaction is partial aborted while the inner transaction conflicts.

The isolation of open nested transaction includes only the operation in the outmost transaction, and the committed data of the inner transaction is outside visible after the inner transaction committing. The inner transaction does not be affect while the outmost transaction aborts.

Flattening nested transaction is the special closed nested transaction which is flattened as a general transaction, and all the inner transactions would become the general operations. So, the outmost transaction should abort while any inner transaction conflicts.

Although the performance is high for the operations executing in parallel, the hazards among the operations still decreases the parallelization. Hence, the speculation of the dependencies among the operations is necessary.

2.2 Speculation

In order to decrease the idle time of the function units [8, 10], the better method is to apply the speculative execution. That is, the computed results of the execution are speculative and may need to be recov-

ered when a miss occurs [5]. Therefore, the operation with later order may execute earlier, and the operation with earlier order might execute concurrently with the later one. There is the important principle that the operations have to commit in sequential order, and the later operation should write back its results after the earlier operation. So, the data version management mechanism is required for deciding where to put the original value and the modified value.

There are two methods for managing the modified values. The first method is "check-point" that any access actions will directly modify the data, and the original data will be stored in other places. The second method is "re-order buffer" that the modified value is put in the additional buffer and will be discarded when the speculation is hit or be updated to the current work space when being miss.

2.3 Motivation

In order to increase the performance of the circuit, the operations in the circuit should be executed as soon as possible; the parallel design is required for executing operations in the same time. However, the data synchronization and the data dependency among these operations in the parallel design is the bottleneck for improving the performance. The operations with the dependency would execute in order, but these operations with no dependency have the chance to execute in parallel. There is the risk that more than one operation access the same data at the same time, and the correction of this data may be wrong. Therefore, the mechanism for the data synchronization is required, transactional memory is one of the lock-free system which is good for executing operation in parallel.

Another bottleneck is the data dependency which limits the parallelization of the operation execution, and the solution of this problem is that the data input of the later operation gets the speculative value from a predicting circuit. The speculation of data input may be not correct, so it is necessary for protecting the unmodified data from wrong accessing. Therefore, the operations have to execute out of order and commit in order, and the unmodified data is keep in original place or other where.

Although the concept of speculation is already clear, the implementation of speculative circuit is still not easy for design. How to maintain the committing order of these operations and when to detect the false speculation and recover the false result of the operation is not presented specifically. Hence, it is necessary to combine the transactional architecture with the concept of speculation.

There is already the good mechanisms for managing the values in TM system, so the transactional design for speculative is suitable; however, it is difficult for keeping the correct serial order in TM system. In TM system, the later transaction commits legally even though the earlier one does not finish its execution. Therefore, the earlier transaction has to get the higher priority. Besides, the false speculation should be looked as the conflict, and the modified values may be discarded when the conflict is detected; otherwise, the operation with false speculation would still commit legally.

As a result, it is a good choice to design the speculative circuit with the transactional architecture, so we shall develop a TM design model and use it to design a speculative circuit as follows.

3 Transaction-based Model

3.1 Transactional speculative circult model

We propose a transactional speculative circult model here based on the concept of TM to design the speculative circuits. There is the problem of data synchronization during speculative circuit executing, and the data version management in TM will be used to avoid losing data and occurring wrong. Besides, uncommitted values will also be avoided from incorrectly accessing.

The variable executing time for complex functional units should be considered in the design of the control path in a speculative circuit. In this condition, the controller usually cannot determine whether a unit has finished, so the read/write set in the TM system are used for indirectly determining when a unit finishes its work. In the design of the speculative circuit, a detector is required to monitor the read/write sets, and a special controller is designed to determine when the functional units finish. Besides, conflict management and data synchronization in the speculative circuit are also designed and kept in the detector. The distributer controller design is required for controlling the operation with variable latency, and there must be a global controller for integrating the behavior of distributed controllers.

Adding the predictor circuits is required, and the predictor speculates the value of control dependency

and/or data dependency which must be solved for the speculative functional units. The predictor is easily designed by the SoC designer, and the rate of successful speculation in the speculative circuit will dominate to the whole circuit's performance. Consequently, the basic executing unit in TSCM should include the executing circuit, the detector, and the predictor, and this basic executing unit is called "transactional module". In other words, the operation is treated as a transaction in the design model.

3.2 The analysis of control path in TSCM

"Speculative scheduling" is the main method for analyzing the behavior of the speculative circuit, and the result of the speculative scheduling is used for designing the control path of the speculative circuit in TSCM.

There are three steps for analyzing in the speculative scheduling. First, we should divide a sequential circuit into more than one operation, and there are possibly some dependencies existing among these operations. One operation should be scheduled in one control step.

Second, we would find the predictable dependencies. If the value of the dependency is easy enough for predicting, the dependency is called "speculative dependency." When the data input of the operation is the speculative dependency, the operation is called "speculative operation." By contract, the other operation with no speculative dependency is called "normal operation."

Third, the speculative operations are speculatively scheduled in earlier control step, and the normal operations may execute in parallel with the later operations; however, the value of the speculative dependency is still not sequentially produced. The predictor is required for speculatively producing the value of the speculative dependency, and the speculative operation executes speculatively by using the value which is produced by the predictor. It is impossible that the speculation of the predictor is always correct, so the detector is required for comparing the speculative value and the sequential value. When the speculative value does not match the sequential value, it is necessary for keeping the correct order of operation executing. The wrong result of speculative operation should be discarded, and the sequential value which is produced by dependent operation is used for recovering the wrong execution of the speculative operation. In order to improve the performance of speculative circuit, iteration overlap is required in the speculative scheduling. The operations which are scheduled in different iterations should also execute in parallel for deceasing the idle time of the executing circuits.

In the above descriptions, there are several control signals needed in the design model. It is difficult for the controller to detect when the operation with variable latency finishes, and the "finishing signal" which is sent from the detector is required in this design model; of course, the "beginning signal" is also required to enable each controller of the operations. These controllers send the "normal executing signal" for controlling the serial behavior of operations, and the "speculative executing signal" is sent from the controllers for controlling the speculative behavior of the operations. In order to know whether the speculation of the speculative operation is correct or not, the "committing signal" is required to send from the detector to the controller of speculative operation. There is a special case that the speculative operation starts after the correct input is produced by the dependent operation, the "pipeline beginning signal" should be sent from the controller of the dependent operation; this signal is used for start the serial execution of the speculative operation. At last, the "end signal" should be sent by the controller for outside delivering the information of the finishing the operation. This information is used to directly enable particular controller, or the global controller decide whether to continue the execution of the circuit basing on the information.

Speculative scheduling. In order to decrease the idle time of the operating circuits, speculative scheduling is a good method for early executing those speculative operations which are scheduled behind the dependent operations. These speculative operations may execute before the correct data input is produced, and the critical path might shorten when the speculation of these speculative operations is correct; otherwise, there is overhead in recovering the false result of speculative operations.

As Fig. 1 shown, there are one normal operation o1 and another speculative operation o2, and the speculative operation may be recovered because of the false speculation. When the speculation of speculative operation o2 is correct, the normal operation o1 and the speculative operation o2 could execute in parallel in the same control step C1' as Fig. 1(a) shown. If there is a false speculation in the speculative operation o2, the additional control step C2' is required for recovery of the speculative operation as Fig. 1(b) shown. "Pre o2" is the predictor which speculatively produces the data input of operation o2, and "Det o2" is the detector which compares the correct input and the speculative input.

(a) Correct speculation for speculative operation (b) False speculation for speculative operation

Fig. 1. Speculation of dependency between operations

In the following we describe the detail of the speculative scheduling by using some examples.

Multiple speculations. In the previous section, we describe the behavior of the speculative circuit which includes a normal operation and only one speculative operation. If there is more than one speculative operation and one of the speculative operations is dependent on another speculative operation, these two operations should execute speculatively in the early control step which the normal operation execute in.

We assume that there is a sequential circuit ckt1 which is divided into four operations (o1, o2, o3, and o4), and the second operation o2 and the third operation o3 are speculative operations; of course, the other operations are the normal operations. The data dependencies among these operations are presented in Fig. 2.

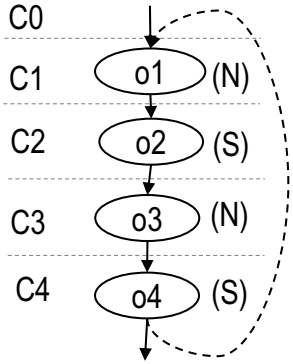


Fig. 2. Behavior of speculative circuit ckt1

In Fig. 2, the solid line means the data dependency between two operations, and there should be some registers for keeping the value of these data dependencies. The speculative operation should execute in the same control step which the operation that the speculative operation is dependent on, so operation o3 could execute in parallel with the speculative operation o2. Therefore, there are two speculative operations execute in the one control step. In case of correct speculation for all of the speculative operations, the performance of the speculative circuit would be three times higher than the original one. However, the serial order for committing is still kept in TSCM, and the later operation o3 could not write back its result before the earlier operation o2. If there is false speculation for the earlier operation o2, the later operation o3 should still wait committing until the operation o2 finishing committing. After the operation o2 committing, the detector of the operation o3 checks the correction of the speculation and the operation o3 commits successfully when the speculation is correct; otherwise, it is necessary for recovering the result of the operation o3.

In fact, that only one operation is scheduled on each executing circuit is not a general case. The hardware requirement is too high if there are too many operations are included in the speculative circuit. Hence, the sharing of the executing circuits is necessary for reducing the hardware cost.

Resource sharing. In spite of the sharing of the executing circuits, the sharing of the registers which put the result of executing circuits is also a general case. We shall describe all of the case for resource sharing in this section.

The speculative circuit ckt1 as shown in Fig. 2 is used as an example again, and we assume the specu-

lative operation o2 and the normal operation o3 share the same executing circuit T2 (see Fig. 3); besides, o1 and o4 are both the normal operations. In the first iteration of execution for ckt1, o1 and o2 are scheduled in the same control step for parallel execution, and o3 is dynamically scheduled after the speculative operation o2 finishing. The normal operation o3 could only execute alone because of the sharing of the executing circuit T2. The executing circuit can only execute one operation at the same time, so these operations which share the same executing circuit should execute in serial. Thus, the speculative operation o2 in the next iteration would not execute before o3 which is in the first iteration finishing.

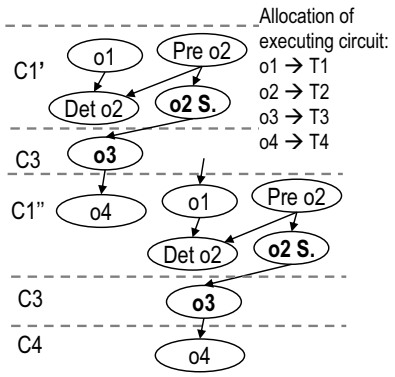


Fig. 3. Behavior for the sharing of the executing circuit

Note that the next iteration starts after the speculative operations o2 commits. If the first operation o1 which is in next iteration starts at the time when the speculative operation o2 in current iteration still executes, the speculative operation o2 in next iteration could not execute speculatively because of the executing circuit sharing. As soon as scheduling can affect the usage of the hardware that is designed for the speculation. Consequently, the timing for begin the next iteration is that all of the speculative operations which is dependent on the highest priority operations finish. The highest priority operations mean that all of the normal operations which may execute in the first control step of the iteration. There is the risk that all the highest priority operations begin too early, and then all the speculative operations which are dependent on these highest priority operations cannot execute speculatively forever. Therefore, it is necessary for restricting the time to begin the next iteration.

In addition to the sharing of the executing circuits, the sharing of the registers may also affect the timing to start the next iteration. We use the speculative circuit ckt1 as shown in Fig. 2 as an example for describe the behavior of the circuit in TSCM again.

We assume that the normal operation o1 and the normal operation o3 share the same writing register *R* as shown in Fig. 4, and there is only one speculative operation o2. The conflict should be avoided while o3 in the current iteration and o1 in the next iteration execute in the same control step. Otherwise, the result of o3 may be overwritten by o1. The normal operation o4 might execute incorrectly because of the wrong input which is produced by o1 in the next iteration. So, the normal operation o1 in the next iteration should be scheduled in the next control step C1', but there is still conflict between o1 in the next iteration and o4 in the current iteration; however, this conflict is allowed in TSCM. The write-after-read (WAR) dependency between o1 and o4 do not always lead to error execution, and o1 could execute with o4 in parallel while o1 writes back result later than o4 reads the data from the register R. Because the operation o4 is a transaction in TSCM, the information that the data reading of o4 finishes is outside knew after the execution of o4 finishing; therefore, the timing for o1 committing is when the operation o4 in the previous iteration finishes.

Partial abort of nested transaction. It is necessary for recovering the result of the operation while the speculation is false, and all the result of the operation is discarded; however, it is not a good choice if the execution of the operation is complicated and time-consuming. The partial abort property is suitable when an operation can be divided into many sub-operations, and the result of these sub-operations whose speculation is correct can be reserved during the recovery of the operation.

We take the behavior of the circuit in Fig. 4 as an example, and o2 is the only one speculative operation in this circuit. If the o2 could be individually divided into three sub-operations as shown in Fig. 5, the data input can be produced by the different predictor. Hence, we would clearly distinguish that the

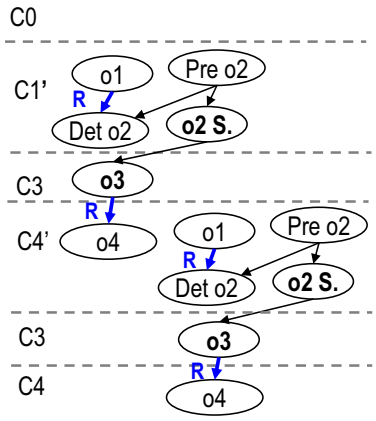


Fig. 4. Behavior for the sharing of the register

false speculation is for which sub-operation. We assume o2_sub1, o2_sub2, and o2_sub3 are the sub-operations which are divided from o2, and o2_sub3 is the only one sub-operation whose speculation is correct. In speculative scheduling, there should be two additional control steps for partial abort of o2_sub1 and o2_sub2; moreover, the result of o2_sub3 is reserved and the time cost is saved.

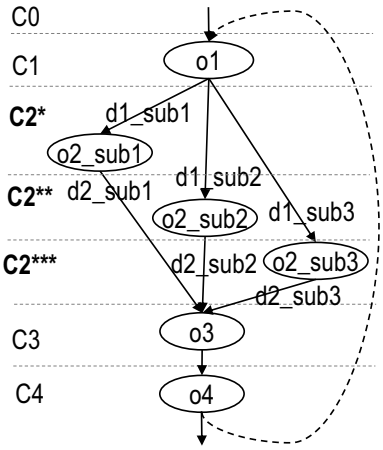


Fig. 5. Three sub-operations dividing from o2

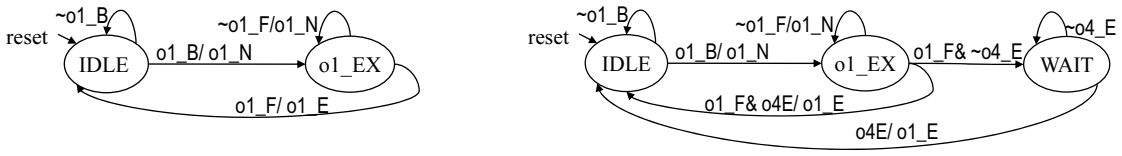
3.3 The design of control path in TSCM

In the previous section, the length of executing time for the operations which execute in the same control step seem to be equal, and the latency of execution for the operation seems to be fixed in any iteration. However, the operation with variable latency should be taken into consideration. The distributed design of the local controller is required in TSCM for controlling the operation with variable latency, and it is necessary to use a global controller for integrating the behavior of all the local controllers. There should be one controller for one executing circuit, and these operations which share the same executing circuit would be controlled by only one controller.

Because the property of the operation for controlling is different, we classify the finite state machine (FSM) of the controllers into four types. “Normal FSM” is using for controlling the normal operation, and the “speculative FSM” is using for controlling the speculative operation; besides, “special normal FSM” is using for controlling the normal operation with WAR dependency. When a circuit is shared by the normal operation and the speculative operation, “mixed FSM” is required for controlling the two types of operations.

The normal operation O_N should be enabled when all O_D finishes its execution. O_N can commit while it finishes the execution, or O_N has to wait all O_{DP} finish its execution. O_D means the operation which O_N is dependent on, and O_{DP} means the operation which has the write-after-read dependency with O_N and the

earlier committing order than O_N. The normal operation with no WAR dependency could commit as long as the execution finishing, so there are only two states required for the normal FSM as shown in Fig. 6(a).



(a) Normal FSM (b) Special normal FSM

Fig. 6. FSM for controlling the normal operation

The state of FSM transfers to "O_N_EX" state when the normal operation O_N is enabled; otherwise, the state stays in "IDLE" for waiting the beginning signal from global controller or other local controller. Above two states are also included in the special normal FSM, and there is one additional state for waiting the finishing of the execution for all O_{DP}; this state is named "O_N_WAIT". If all O_{DP} finish, O_N could commit and the result of O_N should be write back the registers. For example, o1 is the normal operation which is controlled by the FSM as shown in Fig. 6(a); if the WAR dependency exists between the normal operation o4 and o1, it is necessary to use the special normal FSM for controlling the normal operation o1 as shown in Fig. 6(b). "o1_B" is the beginning signal of o1 for enabling the execution of o1, and "o1_N" is the normal executing signal for controlling the serial execution of o1. "o1_F" is the finishing signal which is sent from o1 detector, and the signal is only delivered between o1 detector and o1 local controller; hence, another signal is required for sending the information about the finishing of o1 to global controller. As the description in section 3.2, the end signal is used to deliver this the information about the finishing of the operations, and "o1_E" is the end signal for the normal operation o1.

The speculative execution of the speculative operation O_S should be enabled when all O_D starts its execution and all O_D do not finish its execution; otherwise, the serial execution of O_S should be enabled. The state of speculative FSM transfers from "IDLE" to "O_S_EX_S" for beginning the speculative execution of O_S, or the state transfers from "IDLE" to "O_S_EX_N" for beginning the speculative execution of O_S. If the speculation of O_S is false, "O_S_EX_N" is also for recovery of the speculative execution of O_S. "WAIT" state is require while the speculative execution of O_S finishes and any of the O_{DP} does not finish; the correct data input is produced after all O_{DP} finishes.

As shown Fig. 7, there are two signals are used for start the execution of speculative operation o2. "o2_B" is the beginning signal for start the speculative execution of o2, and "o2_P" is the pipeline beginning signal for start the serial execution of o2 while the dependent operation is already end. After the starting of speculative execution of o2, the speculative executing signal "o2_S" is sent from local controller for controlling the speculative execution of o2. The committing signal "o2_C" is sent from detector for delivering the information about the correction of the speculation, and this signal is valid after the execution of dependent operation o1 ending.

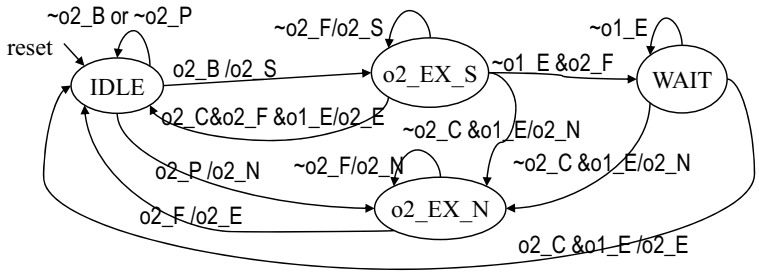


Fig. 7. Speculative FSM for controlling the Speculative operation

The mixed FSM is required for the executing circuit which is shared by the normal operation and the speculative operation. In fact, the mixed FSM is the combination of the speculative FSM and the normal FSM, and the "IDLE" state is shared by these FSM; of course, the mixed FSM for controlling the normal operation with WAR dependency is the combination of the speculative FSM and the special normal FSM.

The distributed local controllers should be integrated by the global controller. The highest priority operations would be enabled by the global controller, and then the other operations are enabled by the dependent local controller. The global controller must to check the condition is that the speculative operations which are dependent on the highest priority operations could start the speculative execution while the highest priority operations are enabled. Another condition which the global controller has to check is how much iteration the speculative circuit has already executes, and the global detector is required to record the times of iteration and to decide when to end the execution of the speculative circuit. The following algorithm is for the global controller of the circuit whose behavior is shown in Fig. 8.

```
Algorithm Global_controller;
input: En, L1, o1_E, 03_E, o4_E,
output: o1_B
begin
    if En==1 then
    begin
        output o1_B  to LC_T1
    end
    else if L1&o1_E&o3_E then
    begin
        output o1_B to LC_T1
    end
    else
        "IDLE";
end
end_ Algorithm
```

Fig. 8. Algorithm of global controller

The highest priority operation in the behavior which is shown in Fig. 8 is the normal operation o1, so the global controller enables o1 when *En* equals to one and the speculative circuit starts. After the beginning of the first iteration, the global controller has to check the two conditions which are described in the previous paragraph.

There is not only the difference of control path for different operation but also the difference of data path. The normal operation and speculative operation would be responded to different design of data path.

3.4 The design of data path in TSCM

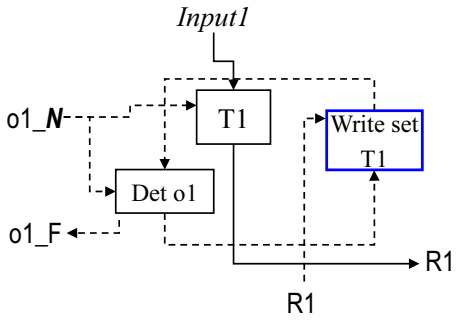
The entire data includes the transactional modules and the registers, and the buffer is required for the normal operation with WAR dependency. The data put in the buffer would be written back the register after the hazard resolving.

The normal transactional module for the normal operation includes one executing circuit, one detector, and one read/write set; read/write set is used to record the address of data accessing in TM system. The executing circuit is the unit for executing operation, and the detector is used to detect when the normal operation finishes. Because of the variable latency for the operation, the information in write set is required by the detector for indirectly detecting the finishing of the normal operation. If there are many operations sharing the executing circuit, more than one detector is required for different finishing condition in normal execution. The different data input may be required in different control steps, so the tri-state buffer may be included in the transactional module for deciding which input the executing circuit should accept.

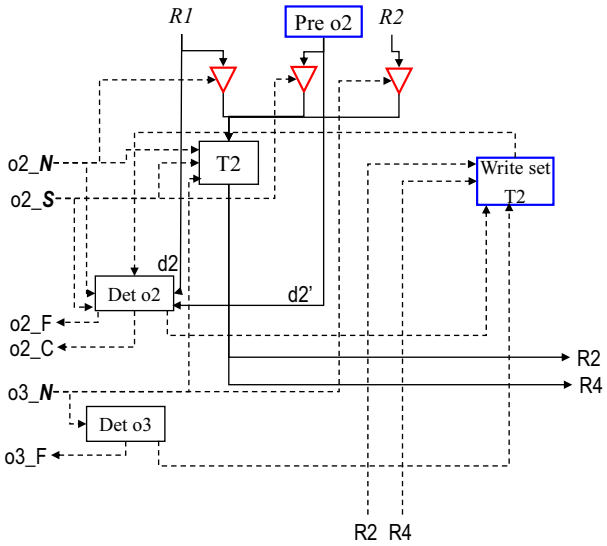
In spite of the units included in normal transactional modules, it is necessary to add the predictor for producing the speculative input of speculative operations into the speculative transactional modules. The tri-state buffer must be included in the speculative transactional module because of the different input for speculative execution and serial execution; the serial execution is used to recover the false speculation of

the speculative operation. Besides, the detector should detect not only the finishing of the execution but also the correction of the execution, so the correct value of data input sent from other modules should be deliver to detector; therefore, a detector should include a counter for recording the progress of the execution and a comparator for check the correction of the execution.

For instance, there is a normal operation o1 which is executing on executing circuit T1, and the result of execution is written in the register R1; then, the architecture of transactional module for o1 is shown as Fig. 9(a). The signal which delivers between o1 detector ("Det o1" in Fig. 9(a)) and write set T1 is used to detect the finishing of the execution. Another transaction module which is shown as Fig. 9(b) is shared by the speculative operation o2 and the normal operation o3, and the executing circuit T2 is shared by o2 and o3. According to the control signal of local controller, the data input may be sent from R1, R2, or o2 predictor ("Det o2" in Fig. 9(b)). The detector should detect the correction of the execution while the data input of T2 is sent from o2 detector; in other case, the detector should only detect the finishing of the execution of o2 or o3.



(a) Normal transactional module



(b) Speculative transactional module

Fig. 9. The architecture of transactional module

In the following we shall take the MPEG-2 design as an example to explain in details how to design a speculative circuit by using the proposed TSCM.

4 Transaction-based MPEG-2 Design

4.1 Introduction of MPEG-2

MPEG-2 is one of encoding formats, and the MPEG-2 file is combined by a series of pictures. There are three encoding mode in MPEG-2 format.

I-frame is the first picture in a GOP (group of pictures), and it is the reference frame for the other pictures; so, I-frame is encoded individually. I-frame is encoded in intra-mode for little distortion; however, a picture with little distortion leads to lower compression ratio.

The other pictures in GOP are P-frames and B-frames which are located between the I-frame and P-frames; the P-frame and B-frame have two reference frames and the I-frame is the reference frame of these P-frames. Hence, they are encoded in the inter-mode. The encoder finds the similar frame from the reference frames, and then "motion vector" and "numerical difference between the similar frame and P-frame" are the data for encoding. The values of vectors and the values of the numerical difference are little in general cases, so the compression ratio is high.

4.2 Analysis of MPEG-2

The basic behavior of a MPEG-2 encoder is presented in Fig. 10; there are many control steps in Fig. 10(a) and the marks C0 to C8 correspond to different control steps. One I-frame is encoded from C0 to C2, and two P-frames are encoded from C3 to C8. In the beginning, we analyze the MPEG-2 encoder and divide the circuit behavior into four function units, which are encoder (ENC), decoder (DEC), motion estimation (ME), and motion compensation (MC), respectively, shown in Fig. 10.

We first analyze the dependencies among these function units in the MPEG-2, and find that the dependencies between ME and MC are suitable for speculating. The function of ME is to find the most similar frame from the reference frames, and then computes the motion vector (MV) of each macro-block in the current frame.

The values of dependencies which are required by MC can be provided in advance by the predictor (Pre MC) in Fig. 10(b), and the detector will check whether the speculative value is correct or not. If speculative value of motion vector is not correct, the executing time of the circuit will be extended as shown in Fig. 10(c).

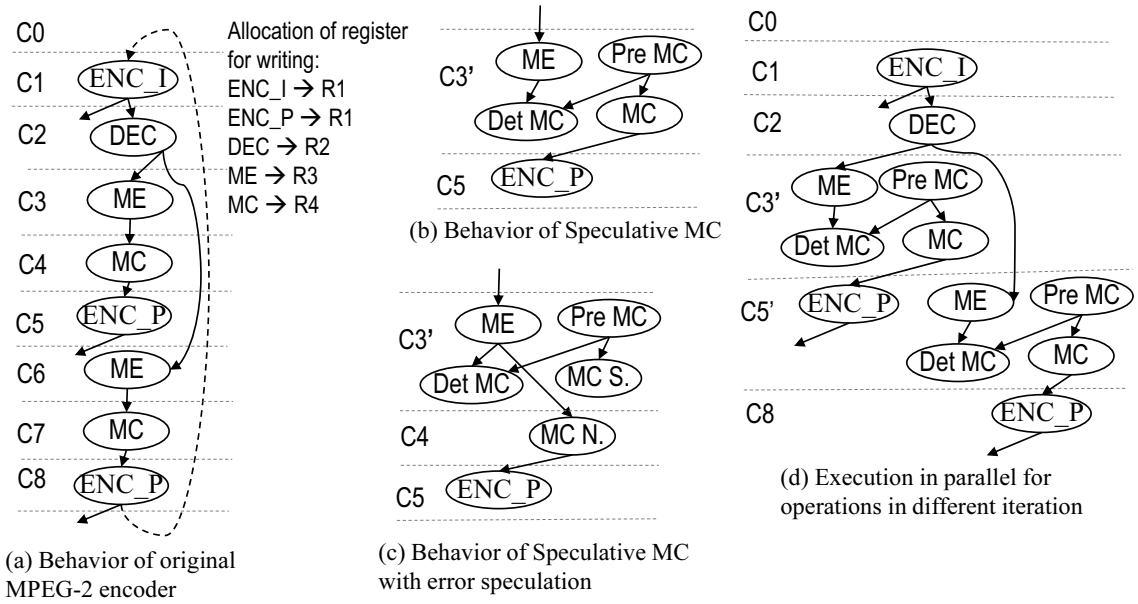


Fig. 10. Behavior of MEPG-2 encoder

Each function unit's operations which are located in the different iterations therefore may execute in parallel for decreasing the execution time of the whole circuit. In Fig. 10(d), the parts which are used for encoding different frames will locate in different iterations. In order to correct the false speculative results of the predicted motion vectors, the MC circuit is seemed like a variable latency functional unit, and a distributed controller is required and had been designed to manage these functional units with the variable executing times as follows.

4.3 Circuit architecture

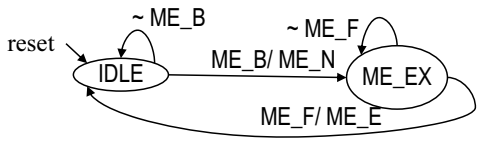
We explain the design of distributed controller with the FSM shown in Fig. 11, and the control steps' names in Fig. 10(c) are then labeled with the states' names. There are two styles in TSCM for designing the FSM: the serial FSM and the speculative FSM.

We first take the serial behavior of ME circuit into consideration, and the normal FSM is sufficient for controlling ME circuit as shown in Fig. 11(a). The normal FSM is enabled by "beginning signal" (ME_B), and then FSM delivers "normal executing signal" (ME_N) for behavior controlling (i.e. IDLE →ME_EX). After getting "finish signal" (ME_F) from the detector, the state of the serial FSM transfers back "idle state" (i.e. ME_EX →IDLE); then, the "end signal" outputs from the serial FSM. Behavior controlling of DEC are similar to that of ME, so we also use the same method to design the FSMs for the local controller of DEC (shown in Fig. 12(a)).

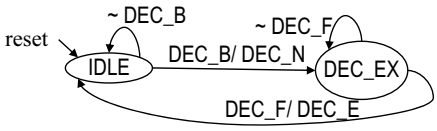
Next, we investigate the speculative behavior in the MC circuit and then design it. The speculative FSM is analyzed and designed to control the MC circuit as shown in Fig. 11(b). The speculative FSM is enabled by "beginning signal" (MC_B), and then delivers "speculative executing signal" (MC_N) for behavior controlling (i.e. idle→MC S.). When the speculative FSM gets the "finish signal" from the detector, it means that the speculative execution finishes; however, the speculative execution is not undoubtedly successful.

If ME circuit finishes early, the detector of the MC circuit will early know whether the speculative value is correct or not; therefore, "commit signal" and "finish signal" will be concurrently delivered from the detector of MC. The state in speculative FSM transfers back the idle state if the speculative value of the motion vector is correct (i.e. WAIT→idle), or the MC circuit is restarted by using a correct value when the speculative one is false (i.e. WAIT→ MC N.). The "WAIT" state in Fig. 11(b) is required for the functional units with variable executing times. On the contrary, the MC circuit should wait until the ME circuit finishes (i.e. MC S.→WAIT), and the result of speculative MC execution may be modified when the ME circuit finishes.

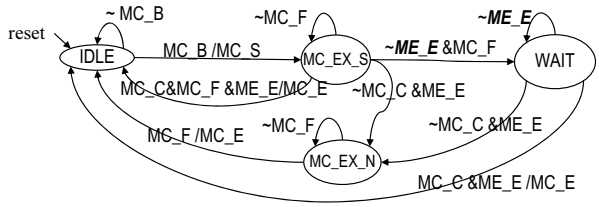
At last, the FSM of executing circuit ENC is share by ENC_I (variable length coding for I_frame) and ENC_P (variable length coding for P_frame), the normal FSM for ENC local controller should be designed as shown in Fig. 12(b).



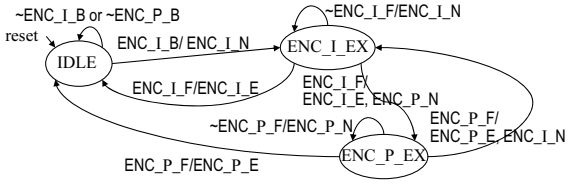
(a) Normal FSM for ME controlling



(a) Normal FSM for controlling DEC



(b) Speculative FSM for MC controlling



(b) Normal FSM for controlling ENC

Fig. 11. FSM of local controller in speculative MPEG-2

Fig. 12. Normal FSM for ENC controlling

A global controller is required for controlling all distributed local controllers in the whole speculative MPEG-2 circuit, and the global controller will manage when to execute a functional unit and keep data synchronization among those distributed controllers. The global controller for the speculative MPEG-2 encoder is designed and shown in Fig. 13. "L1" in it is the variable for deciding whether to continue the loop for the P-frame encoding in the Speculative encoder, and "L2" is the one for I-frame encoding. With the proposed TSCM, the global controller is easily designed.

```
Algorithm Global_controller;
input: En, L1, L2, DEC_E, ENC_P_E
output: ENC_I_B, ME_B
begin
    if En==1 then
    begin
        output ENC_I_B  to LC_ENC
    end
    else if DEC_E then
    begin
        output ME_B to LC_ME
    end
    else if ~L1&L2&ENC_P_E then
    begin
        output ENC_I_B to LC_ENC
    end
    else
        "IDLE";
end
end_ Algorithm
```

Fig. 13. Algorithm of global controller in speculative MPEG-2 encoder

By the way, in our design more than one operation in the speculative circuit is allowed to share the same local controller and the same functional circuit, and these operations which share the same sources should only execute in serial.

The data path of speculative MPEG-2 is shown in Fig. 14. The executing times of ENC and MC are variable in this speculative encoder design, so the information of read/write sets is analyzed and designed when designing the detectors of ENC and MC. The respective computation circuits and detectors for the ENC, DEC, and ME circuits are designed for construction of the whole speculative module, which is called "transactional module"; of course, a transactional module is the basic unit for executing the circuit in TSCM.

Because of the difference between serial execution and speculative execution in the speculative data path for MC circuit, one predictor and two tri-state buffer are required in designing the transactional module. The function of these tri-state buffers is used to select the correct inputs and to avoid misusing the uncommitted data.

5 Result of Synthesis

We had implemented and synthesized the speculative MPEG-2 encoder in Verilog, and its synthesis results are listed in Table 1. The synthesis tool is Quartus II 8.0, and the device name used for synthesizing is EP1S80F1508C5.

The largest unit in the diagram is encoder circuit, but the critical path may not be located in the circuit when these circuits execute in parallel. The value of data (i.e. MV and numerical difference) for P-frame encoding is little, so the executing time of ENC is shorter; therefore, the critical path will be located in motion estimation and motion compensation (MEMC). It is necessary for MEMC optimizing in speculative encoder. The encoder is tested by encoding 20 frames, and each group of picture (GOP) includes one I-frame and three P-frames. The detail executing times of the encoder is listed in Table 2.

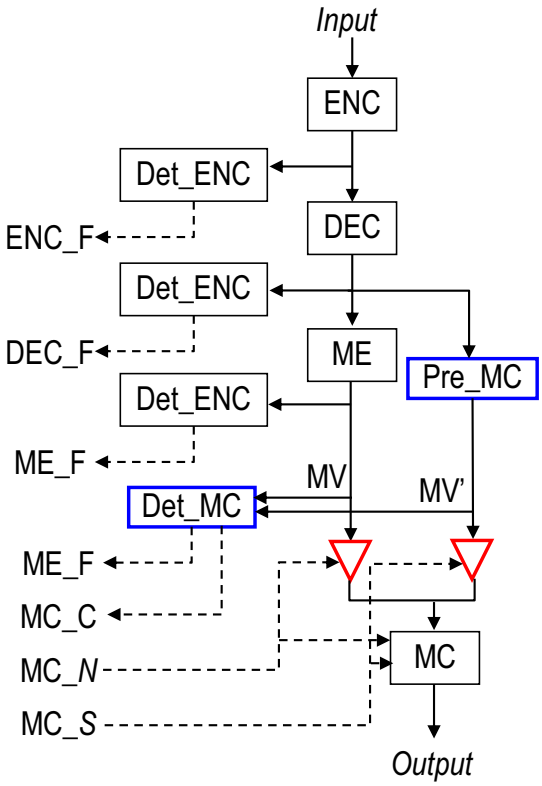


Fig. 14. Architecture of data path in speculative MPEG-2

Table 1. Result for synthesis of speculative MPEG-2

Logic elements in synthesis of Encoder	Logic elements in synthesis of Decoder	Logic elements in synthesis of Speculative MEMC	Logic elements in synthesis of Speculative controller	Total pins in synthesis	Maximum operating frequency
12470	4677	1824	201	1189	28.34 MHz

Table 2. Executing time for speculative MPEG-2 encoder

Exe. Time Benchmark	Average cycle for original MEMC	Average cycle for speculative MEMC	Total cycle for speculative MPEG-2	Total cycle for original MPEG-2
akiyo_cif.yuv	337872	324748	23099391	35561500
coastguardo_cif.yuv		331926	27113921	43770660
mobileo_cif.yuv		336129	32366145	54344240
foremano_cif.yuv		335132	26038211	41551520
containero_cif.yuv		331859	26155374	41583900
flowero_cif.yuv		336295	29054463	47660720
waterfallo_cif.yuv		336312	29630149	49016860
stefano_cif.yuv		336221	28150032	45968400
tempeleo_cif.yuv		336065	29621736	48916180

The executing time reduction of speculative MEMC is less than 1 millisecond as listed in Table 2; and the speedup of speculative MEMC is 1.039 times higher than that of original one when the hit rate in speculation is 47.92% (see Table 3). The benefit of speculating in MEMC circuit is not significant, and it is because that the executing time of MC is ten times shorter than that of ME; so the best case of speedup for speculative MEMC is 1.1 times higher. Therefore, the most important factor for performance improvement is not the speculation in MEMC.

Table 3. Speed up for speculative MPEG-2 encoder

Speed up Benchmark	Speculative region in MEMC	MEMC	Original MPEG-2	Hit rate (%)
akiyo_cif.yuv	1.049	1.039	1.351	47.92
coastguardo_cif.yuv	1.028	1.018	1.381	26.98
mobileo_cif.yuv	1.015	1.005	1.404	14.72
foremano_cif.yuv	1.018	1.008	1.373	17.62
containero_cif.yuv	1.028	1.018	1.371	27.17
flowero_cif.yuv	1.014	1.005	1.390	14.23
waterfallo_cif.yuv	1.015	1.005	1.396	14.18
stefano_cif.yuv	1.015	1.005	1.388	14.44
tempeleo_cif.yuv	1.015	1.005	1.394	14.90

In fact, the scheduling in Fig. 10(d) significantly shortens the critical path of speculative MPEG-2. The executing time reduction of speculative MPEG-2 is 20 milliseconds as listed in Table 2; and the most important factor for performance improvement is that each part's operations, which is located in different iterations, execute in parallel.

The results of synthesized circuit are listed in Table 4, and the speculative encoder produces a 0.75% area overhead. The additional hardware is located in the speculative MEMC (as Fig. 11) and the global controller (as Fig. 13). The trade-off between the 0.75% area overhead and the 1.35 times speedup is a worth deal.

Table 4. Area comparison between original MPEG-2 and Speculative MPEG-2

Total logic elements in synthesis of Speculative MPEG-2	Total logic elements in synthesis of original MPEG-2	Additional elements	Percentage of additional area
18366	18229	137	+0.75%

6 Conclusion and Future Work

We have proposed a TSCM model and designed a transaction-based MPEG-2 encoder using the mode. The transaction-based design could improve the performance of circuit. Although additional hardware was required, the circuit executing time would significantly decrease.

Transaction memory is a good design for synchronization, so the parallelization of the programs and the operations is proved; however, the improvement of parallelization will not certainly increase the performance of the system. If the ratio of successful committing for the transactions is too low, most of result for transactions will be discarded after parallelizing execution; the performance of the system may not be improved obviously. How to schedule the operations for executing in parallel is an important topic in the future. Besides, the new value of the data which is modified during the transaction executing is outside read legally after the transaction commits, and it is steady method for protecting the new value from wrongly accessing. But in the case that a new value is produced early and waits committing for a long time will decrease performance of the system. It is because that some transactions which need the new value must wait the certain transaction committing and do not execute early. So, it is important to select the suitable operations for combining into a transaction.

The transaction-based encoder in this paper is a small-size circuit, so the memory requirement for this circuit is little. However, the amount of the required memory is proportional to the size of transaction-based circuit. The amount of the memory required by the circuit will impact the speed of data delivery; and it will be our next target.

Another target is the automatic design of the transaction-based circuit in TSCM. The current design of speculative scheduling is still the static scheduling, and the speculative circuit is still analyzed by SoC designer; this method is only suitable for simple design of speculative circuit. If there are more elements and more complicated behavior in the circuit, we will spend a lot of time and effort for analyzing and designing. Therefore, how to automatically produce the acceptable design of the speculative circuit in TSCM is an important target in the future.

References

[1] M. Lupon, G. Magklis, A. Gonzalez, A dynamically adaptable hardware transactional memory, a dynamically adaptable hardware transactional memory, in: Proc. of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2010.

[2] Q.L. Meunier, F. Petrot, Lightweight transactional memory systems for NoCs based architectures: design, implementation and comparison of two policies, Journal of Parallel and Distributed Computing 70(10)(2010) 1024-1041.

[3] H.E. Ramadan, C.J. Rossbach, D.E. Porter, O.S. Hofmann, A. Bhandari, E. Witchel, MetaTM/TxLinux: transactional memory for an operatingsystem, ACM SIGARCH Computer Architecture News 35(2)(2007) 92-103.

[4] A. Shriraman, S. Dwarkadas, M.L. Scott, Implementation tradeoffs in the design of flexible transactional memory support, J. Parallel Distrib. Comput 70(10)(2010) 1068-1084.

[5] C. Tian, M. Feng, V. Nagarajan, R. Gupta, Copy or discard execution model for speculative parallelization on multicores, in: Proc. of 41st IEEE/ACM International Symposium on Microarchitecture, 2008.

[6] J. Bobba, K.E. Moore, H. Volos, L. Yen, M.D. Hill, M.M. Swift, D.A. Wood, Performance pathologies in hardware transactional memory, in: Procs. of the 34th Intl Symp on Computer Architecture, 2007.

[7] S. Kulkarni, P. Kulkarni, J. Deshpande, P. Kakade, P. Ranadive, M. Tasgaonakar, S. Deshpande, Taxonomy for transactional memory systems, International Journal of Computer Science and Information Technologies 2(2)(2011) 766-775.

[8] C. Tian, M. Feng, V. Nagarajan, R. Gupta, Speculative parallelization of sequential loops on multicores, International Journal of Parallel Programming 37(5)(2009) 508-535.

[9] C. Zilles, L. Baugh. Extending Hardware Transactional memory to support non-busy waiting and non-transactional actions. 〈http://janvitek.org/events/TRANSACT/slides/zilles.pdf〉, 2006.

[10] A.A. Del Barrio, S.O. Memik, M.C. Molina, J.M. Mendías, R. Hermida, A distributed controller for managing speculative functional units in high level synthesis, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 30(3)(2011) 350-363.

[11] C.J. Rossbach, O.S. Hofmann, D.E. Porter, H.E. Ramadan, A. Bhandari, E. Witchel, TxLinux: using and managing hardware transactional memory in an operating system. 〈https://s3-ap-southeast-1.amazonaws.com/erbuc/files/3ab1443a-c7b0-4ce1-9428-9147923f2bd4.pdf〉, 2007.