PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 39, NO. 2, PP. 85-102 (1995)

A COMPABILITY BASED ALLOCATION METHOD IN HIGH LEVEL SYNTHESIS[1]

Péter ARATÓ and István BÉRES

Department of Process Control
Technical University of Budapest
H-1521 Budapest, Hungary
XI., Műegyetem rkp. 9
email: arato@seeger.fsz.bme.hu, beres@seeger.fsz.bme.hu
tel: (36-1)463-2196, fax: (36-1)463-2204

Received: June 30, 1995

Abstract

This paper presents a model and a method for the allocation during the high level datapath synthesis of pipelined ASIC architectures starting with a behavioral description of the system consisting of theoretical operational units with arbitrary operation duration. As a part of the Scheduling and Allocation Method (SAM), a compatibility relation is used for determining the operations to be allocated in one processor element.

The aim of the procedure is to reduce the number of processors that are necessary for the realization of the theoretical operational units. The method presented in this paper can provide a better solution to the resource allocation problem in many cases by handling the conditional branches. The constraints for the types of processors to be applied can be different depending upon the hardware resources.

Keywords: high-level synthesis, behavioral description, scheduling, allocation, pipelining.

1. Introduction

The high level synthesis is a design method starting with a behavioral description derived from the problem to be solved by a digital system and yielding a register transfer- and processor level structure. The behavioral description of the datapath – usually represented by a graph or a high level language – is based on elementary operations. Two important steps of the high level synthesis are the *scheduling* of the datapath by a proper start control of the elementary operations and the *allocation* of the elementary operations into processors aimed at the optimal cost-speed trade-off. In this way, the number of necessary control steps and necessary number of different processors can be determined, leading to an increasing throughput of a behavioral datapath. In the case of pipelining, the throughput is

[1]The research work outlined in this paper was supported by the grant OTKA-776 of the Hungarian Academy of Sciences.

characterized by the *restarting period* (R) defined as the shortest time the datapath requires before accepting a new input data.

The resource allocation is generally handled as a separate subtask of high level synthesis and usually it is only mentioned in papers which introduce effective scheduling algorithms. In most cases, the scheduling and allocating of pipelined systems require extra considerations and a desired restarting period cannot be given in advance [5,6,8,9,10], the duration of the elementary operations are assumed to be uniform, as either a single clock cycle or a control step.

The method presented in this paper is based on a compatibility relation between concurrent operations, it can handle operations with different arbitrary duration times and ensures a desired pipeline restarting period.

A synchronized datapath is assumed and the duration of each operation is considered to be the necessary number of the clock cycles for each. Control of the datapath is outside of the scope of this paper, but after having solved the scheduling and allocation, a simple centralized counter-like or distributed handshake control can easily be designed.

2. The Graph Representation of the Datapath

The behavioral description of a solution to a problem with an input vector $X = (x_1 \dots x_n)$ and an output vector $Y = (y_1 \dots y_m)$ can be represented by a directed graph. The nodes of the graph are the *elementary operations* $e(i)$ and the directed edges show the data connections between the operations. This graph is called Elementary Operation Graph (EOG).

If the output of $e(i)$ is connected to the one of the inputs of $e(j)$ then $e(i)$ and $e(j)$ are data-connected and it is represented with $e(i) \rightarrow e(j)$. In this case, there is a directed edge from the node $e(i)$ to the node $e(j)$ in the EOG. Each elementary operation may receive data from more than one other operation, but it is assumed to have only a single output vector. If the output supplies several inputs then several edges can represent these connections from the output of $e(i)$ to the inputs of the other operations.

With assumptions on recursive and non-recursive loops the datapath can be considered as an assembly of independent sequences of operations called *transfer sequences* starting with an operation which receives input data, going through the directed edges and ending with an operation which produces the output data of the datapath. Thus, $S(i,k)$ denotes the k-th transfer sequence beginning with $e(i)$.

The *duration time* of an operation $e(i)$ is $t(i)$ if it requires $t(i)$ clock cycles from that time when all the inputs are available to the time when the result of the operation is stable on the output of $e(i)$. In literature [4,7],

$t(i)$ generally is interpreted as the number of the control steps required for the execution of the operation $e(i)$.

It is assumed that the datapath has a dataflow character, thus any operation:

1. – requires its input during the whole duration time,
2. – holds its actual output stable until the next start,
3. – may change its output during the whole duration time.

A point of time $b(i)(h)$ can be associated with each of the h-th inputs of each $e(i)$ referring to the serial number of the clock cycle in which the first data arrives on this input. The first start of $e(i)$ is initiated at

$$b(i) = \max(b(i)(h)) \; , \tag{2.1}$$

where $b(i)$ is the beginning point of time of the duration of $e(i)$.

After the duration time $t(i)$, the output data of $e(i)$ is available for $e(j)$ where $e(i) \rightarrow e(j)$ holds. This output must be stable for the duration of $e(j)$. Therefore, $e(i)$ may receive a new start with new data only after finishing the duration time of $e(j)$. Thus, the restarting period of $e(i)$ must be longer than $t(i) + t(j)$ [1,3]. This limit is the length of the restarting period, called the *transfer score* $q(i)$ of $e(i)$:

$$q(i) = t(i) + \max(t(j)) \; , \tag{2.2}$$

where the max function is necessary if $e(i)$ is connected to more than one $e(j)$.

The shortest restarting period which can be achieved by the EOG must be greater than the longest transfer score:

$$R >= \max(q(i)) + 1 \; . \tag{2.3}$$

During the design process the designer usually wants to achieve a given restarting period. If the restarting period is shorter than the R in *Eq.* (2.3) a procedure must be found for reducing the minimum restarting period of the EOG.

3. Scheduling

At first, two simple reducing algorithms are presented below as systematic interactions into the EOG to obtain the shortest latency specified in advance. This is the first step of the scheduling.

3.1 Reducing the Shortest Latency of the EOG

3.1.1 Inserting Buffer Registers

Let $e(i) \rightarrow e(j)$ be held and let a *buffer register* $e(p)$ be inserted between $e(i)$ and $e(j)$ with a duration of $t(p) = 1$ [1,2,3]. The new sequence is $e(i) \rightarrow e(p) \rightarrow e(j)$. The new transfer scores can be calculated as

$$q(i) = t(i) + 1 \; , \tag{3.1}$$
$$q(p) = 1 + t(j) \; . \tag{3.2}$$

The new shortest restarting period is:

$$R = \max(q(i), q(p)) + 1 \; . \tag{3.3}$$

This R is a smaller number than it was before if $t(i)$ and $t(j)$ are greater than 1. Thus if this $e(i)$ was the bottleneck of the EOG then the smallest restarting period R of the EOG is reduced.

In other words, for achieving the restarting period R, a buffer register must be placed after every operation which has a bigger transfer score than $R - 1$.

3.1.2 Reducing the Restarting Period Applying Multiple Functional Units

From *Eqs.* (3.1), (3.2), (3.3), the minimum of the restarting period that can be achieved is

$$R = \max(t(i)) + 2 \; . \tag{3.4}$$

If a smaller value is desired then more copies of $e(i)$ have to be connected in parallel. Let $e(i)$ be applied in $c(i)$ copies connected in parallel and let $e(i) \rightarrow e(j)$ be assumed [1,2,3]. The first copy of $e(i)$ ($e(i,0)$) starts functioning at $b(i,0) = 0$. Its duration is $t(i)$ and it must hold the output during $t(j)$ for the next $e(j)$. The second copy $e(i,1)$ is initialized at $b(i,1) = R$ and the $(n+1)$-th copy $e(i,n)$ begins to compute at $b(i,n) = n \times R$. The first copy starts again at $b(i,0) = c(i) \times R$ with the $c(i) + 1$-th data where $c(i)$ is the number of the same operations which were connected parallel. The $c(i) + 1$-th data cannot arrive earlier than $t(i) + t(j) + 1$. Formally:

$$c(i) \times R \geq q(i) + 1 \; . \tag{3.5}$$

From (3.5):

$$c(i) \geq (q(i)+1)/R\,, \tag{3.6}$$

where $R > 0$ always holds. Thus, for achieving a value of R we must apply $c(i)$ copies of $e(i)$, where $c(i)$ is interpreted as the nearest integer which is greater than the result of the previous division. A buffer register must be inserted before each copy of $e(i,n)$ because the input data must hold during $t(i)$ for each copy.

The combined algorithm is introduced as follows:

```
for all $e(i)$ in EOG do
    if $q(i) + 1 > R$ then
      insert a buffer after $e(i)$
    if $t(i) + 2 > R$ then
    $c(i) = (t(i) + 1 + R)$ div $R$
      insert $c(i)$ buffers and $c(i)$ copies of $e(i)$ in the EOG
```

With this algorithm, the shortest latency of a transfer sequence without recursive loops can be reduced to 3 [2].

4. Allocation

The allocation is the part of the high level synthesis when the processing units are formed from elementary operations. A processor is the real unit which must be realized. The constraints for forming the processor units and covering the elementary operations with these processor units and the savings of resources depend strongly on the groups of elementary operations realizable in one unit, the structure of the connections, the limits on the number of processing units, etc. [4,7].

A very simple approach to the allocation is to consider the processing units as real resources, with the notion that each of them can realise one or more elementary operations which are never busy simultaneously. The *busy state* of the operation is not only the duration time but it lasts until the end of its data hold period as well, this time can be determined by the transfer score. Let the operations be called *concurrent operations* if their busy states are overlapped in time. The maximal sets of non concurrent elementary operations can represent and specify the real processing units involving the structural description of the system [2,4].

If two elementary operations $e(i)$ and $e(j)$ are non-concurrent then these operations can be *allocated in a processor* $\{e(i), e(j)\}$ and an $e(i)$ operation can always be allocated in a processor with itself: $\{e(i), e(i)\}$. This relation ($\{\}$) is a compatibility relation as shown:

1. Reflective: $\{e(i), e(i)\}$ is always true.
2. Symmetric: If $\{e(i), e(j)\}$ is true then $\{e(j), e(i)\}$ is true.
3. Non transitive: if $\{e(i), e(j)\}$ and $\{e(j), e(k)\}$ are true then $\{e(i), e(k)\}$ is not necessarily true.

4.1 Concurrent Operations

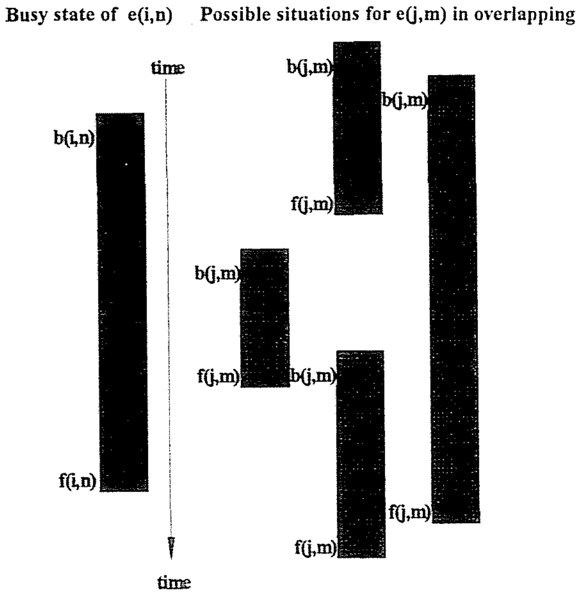

Fig. 4.1.

Fig. 4.1 shows all possible variations of the busy state overlapping for two operations $(e(i,n), e(j,m))$, where $b(i,n)$ and $b(j,m)$ stand for the beginning and $f(i,n)$ and $f(j,m)$ for the finishing points of time of the busy state for $e(i,n)$ and $e(j,m)$, respectively.

From the definitions of the transfer score and start time, $q(i)$, $q(j)$ and $b(i,n)$, $b(j,m)$, respectively, the finishing points of time $f(i,n)$ and $f(j,m)$ of the busy states of $e(i,n)$ and $e(j,m)$ can be found as follows:

$$b(i,n) + q(i) = f(i,n) \ , \tag{4.1}$$

$$b(j,m) + q(j) = f(j,m) \ . \tag{4.2}$$

According to *Fig. 4.1* $e(i,n)$ and $e(j,m)$ are concurrent operations if any of the following inequalities is satisfied:

$$b(i,n) \le b(j,m) < f(j,m) \le f(i,n) \ , \tag{4.3}$$
$$b(i,n) \le b(j,m) \le f(i,n) \le f(j,m) \ , \tag{4.4}$$
$$b(j,m) \le b(i,n) < f(i,n) \le f(j,m) \ , \tag{4.5}$$
$$b(j,m) \le b(i,n) \le f(j,m) \le f(i,n) \ . \tag{4.6}$$

The inequalities (4.3) and (4.4) cover the same situations when the $e(j,m)$ busy state starts at the same time or during the busy state of $e(i,n)$. Also the inequalities (4.5) and (4.6) cover similar situations when the $e(i,n)$ busy state starts at the same time or during the busy state of $e(j,m)$. As *Eqs.* (4.1) and (4.2) show, the followings are always true:

$$b(i,n) < f(i,n) \ , \tag{4.7}$$
$$b(j,m) < f(j,m) \ . \tag{4.8}$$

From the previous inequalities (4.1–4.8) it can be concluded that the $e(i,n)$ and $e(j,m)$ are concurrent if and only if either of the following two inequalities is true:

$$b(i,n) \le b(j,m) \le b(i,n) + q(i) \ , \tag{4.9}$$
$$b(j,m) \le b(i,n) \le b(j,m) + q(j) \ . \tag{4.10}$$

In a pipeline mode, if the beginning point of time of $e(i,0)$'s and $e(j,0)$'s busy state are $b(i) = b(i,0)$ and $b(j) = b(j,0)$, respectively, then

$$b(i,n) = b(i) + (n + k(i,n) \times c(i)) \times R \ , \tag{4.11}$$
$$b(j,m) = b(j) + (m + k(j,m) \times c(j)) \times R \ , \tag{4.12}$$

where $k(i,n)$ and $k(j,m)$ are arbitrary non negative integers and $(n + k(i,n) \times c(i))$ and $(m + k(j,m) \times c(j))$ are the serial number of the input vector (X) received by the EOG and processed by $e(i,n)$ and $e(j,m)$.

Substituting *Eqs.* (4.11) and (4.12) into (4.9) and (4.10):

$$b(i) + (n + k(i,n) \times c(i)) \times R \le b(j) + (m + k(j,m) \times c(j)) \times R \le \\ b(i) + (n + k(i,n) \times c(i)) \times R + q(i) \ , \tag{4.13}$$
$$b(j) + (m + k(j,m) \times c(j)) \times R \le b(i) + (n + k(i,n) \times c(i)) \times R \le \\ b(j) + (m + k(j,m) \times c(j)) \times R + q(j) \ . \tag{4.14}$$

From the previous two formulas it can be written:

$$b(i) - b(j) \leq [(m + k(j,m) \times c(j)) - (n + k(i,n) \times c(i))] \times R \leq \\ b(i) - b(j) + q(i) \tag{4.15}$$

$$b(i) - b(j) \geq [(m + k(j,m) \times c(j)) - (n + k(i,n) \times c(i))] \times R \geq \\ b(i) - b(j) - q(j)\ . \tag{4.16}$$

The left sides of the inequalities are identical, therefore:

$$b(i) - b(j) - q(j) \leq K \times R \leq b(i) - b(j) + q(i)\ , \tag{4.17}$$

$$K = [m + k(j,m) \times c(j)] - [n + k(i,n) \times c(i)]\ . \tag{4.18}$$

Thus, $e(i,n)$ and $e(j,m)$ are concurrent *if and only if* at least one integer K and non negative integers: $k(i,n)$ and $k(j,m)$ can be found which satisfy the inequality (4.17) and (4.18) Diophantos equation.

4.1.1 Solutions in the Case of Number of Copies

Substituting *Eqs.* (4.18) back into (4.17):

$$b(i) - b(j) - q(j) \leq \{[m + k(j,m) \times c(j)] - \\ - [n + k(i,n) \times c(i)]\} \times R \tag{4.19}$$

and

$$b(i) - b(j) + q(i) \geq \{[m + k(j,m) \times c(j)] - \\ [n + k(i,n) \times c(i)]\} \times R\ . \tag{4.20}$$

From the previous two formulas it can be written:

$$b(i) - b(j) - m \times R + \\ (n + k(i,n) \times c(i)) \times R - q(j) \leq k(j,m) \times c(j) \times R \tag{4.21}$$

and

$$b(i) - b(j) - m \times R + \\ (n + k(i,n) \times c(i)) \times R + q(i) \geq k(j,m) \times c(j) \times R\ . \tag{4.22}$$

Introducing the notation:

$$A = b(i) - b(j) - m \times R + (n + k(i,n) \times c(i)) \times R \tag{4.23}$$

and rewriting the inequalities (4.21) and (4.22) in one, because the right sides are identical:

$$A - q(j) \leq k(j,m) \times c(j) \times R \leq A + q(i) \ , \tag{4.24}$$

where $c(i)$ is the number of the copies of $e(i)$, thus $c(i) > 0$ is always true, also by definition, the restarting period R is always greater than 0 and from the definition in *Eq.* (4.12) $k(j,m)$ must be a non negative integer, thus:

$$A - q(j) \geq 0 \ , \tag{4.25}$$

which involves from *Eq.* (4.23):

$$k(i,n) \geq [b(j) - b(i) + R \times (m-n) + q(j)]/[c(i) \times R] \ . \tag{4.26}$$

The inequality (4.24) involves an interval for $k(j,m) \times [c(j) \times R]$:

$$I = q(i) + q(j) \ . \tag{4.27}$$

If this I interval is greater than or equal to $[c(j) \times R]$ then a non negative integer $k(j,m)$ can always be found for any $k(i,n)$ which satisfies *Eq.* (4.26). These two non negative integers denote that $e(i,n)$ and $e(j,m)$ are concurrent.

From the definition of $c(j)$:

$$c(j) \geq (q(j)+1)/R \ , \tag{4.28}$$

a lower and an upper bound can be given for $c(j)$ as:

$$q(j) + 1 \leq c(j) \times R \leq q(j) + R \ . \tag{4.29}$$

It can be proven, as is shown in *Eq.* (4.30), that even the upper bound of $c(j) \times R$ is smaller than the I interval in *Eq.* (4.27), because if $c(i)$ is greater than 1, then $q(j) \geq R$ holds:

$$q(j) + R \leq q(i) + q(j) = I \ . \tag{4.30}$$

In this case $e(i,n)$ and $e(j,m)$ are concurrent because the I interval from *Eq.* (4.24) is always greater than the upper bound of $c(j) \times R$. The steps from *Eq.* (4.21) to *Eq.* (4.30) are symmetric in $e(i,n)$ and $e(j,m)$ so if

$$R \leq q(i) \tag{4.31}$$

or

$$R \leq q(j) \ , \tag{4.32}$$

then $e(i,n)$ and $e(j,m)$ are concurrent, because the solution for *Eq.* (4.17) and *Eq.* (4.18) can always be found. In other words *if an $e(i)$ is multiplied* (because $q(i) \geq R$) *then any $e(i,n)$ copy of this $e(i)$ is concurrent with any other $e(j)$ in the EOG.*

4.1.2 Concurrence of Non Multiplied Operations

If $c(i) = c(j) = 1$ (non multiplied operations), then $n = m = 0$ which makes *Eqs.* (4.17) and (4.18) much simpler:

$$b(i) - b(j) - q(j) \leq K \times R \leq b(i) - b(j) + q(i) , \tag{4.33}$$

$$K = k(j) - k(i) . \tag{4.34}$$

If K exists *Eq.* (4.34) always has a solution (any integer can be written as a difference of two non negative positive integers). Thus, if $e(i)$ and $e(j)$ are not multiplied operations, then they are concurrent *if and only if* at least one integer K can be found which satisfies *Eq.* (4.33).

The paragraph following the *Eq.* (4.27) proves:

$$\text{If} \quad q(i) + q(j) \geq R , \tag{4.35}$$

then $e(i)$ and $e(j)$ are concurrent. A simple algorithm can be set up:

```
for all $e(i)$ in EOG do
    if $c(i) = 1$ then
        for all $e(j)$ where $j > i$ do
            if $c(j) = 1$ and $q(i) + q(j) < R$ then
                $A = [b(i) - b(j) - q(j)]/R$    /× realization of 4.33 ×/
                $B = [b(i) - b(j) + q(i)]/R$
                if [int $(A)$ = int $(B)$ and sign $(A)$ = sign$(B)$ and
                   int $(A)! = A$ and int$(B)! = B$] then
                     $e(i)$ and $e(j)$ are not concurrent
```

The complexity of this algorithm is $O(n \times n/2)$ if the EOG has n operations before the multiplication.

Another algorithm can be found if the busy state of the elementary operations is kept folding into one restarting period with a modulo division by R. In this case *Eq.* (4.35) has a descriptive meaning as both elementary operations must fit into one restarting period. Let the starting and the finishing points of time of the busy state be modified:

$$b'(i) = b(i) \bmod R , \tag{4.36}$$
$$f'(i) = f(i) \bmod R , \tag{4.37}$$
$$b'(j) = b(j) \bmod R , \tag{4.38}$$
$$f'(j) = f(j) \bmod R , \tag{4.39}$$

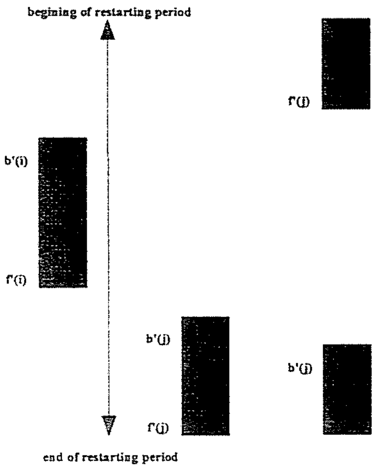

Fig. 4.2.

where $b'(i)$ and $b'(j)$ show the beginning points and $f'(i)$ and $f'(j)$ show the ending points of time for the busy states of $e(i)$ and $e(j)$ in one restarting period. *Fig. 4.2* shows the possible situations when $e(i)$ and $e(j)$ are not overlapped. In this case:

$$f'(i) < b'(j) \ , \tag{4.40}$$

$$f'(j) > b'(j) \text{ or } f'(j) < b'(j) \ . \tag{4.41}$$

If the original beginning and ending points of time are written back from *Eqs.* (4.36) - (4.39) into *Eqs.* (4.40) - (4.41) then:
if

$$[b(i) + q(i)] \bmod R < b(j) \bmod R \ , \tag{4.42}$$

then

$$[b(j) + q(j)] \bmod R > b(j) \bmod R \tag{4.43}$$

or

$$[b(j) + q(j)] \bmod R < b(i) \bmod R \ . \tag{4.44}$$

4.2 Handling the Conditional Branches

The conditional checking operation can be interpreted in the EOG by completing it with special elementary operations, called *case operations*, which *select only one* single transfer sequence from the possible transfer sequences (following the operation) in each period of the pipeline mode. In the next period, according to the pipeline mode, again only one single transfer sequence is selected, which may be the same as or different from the previous one.

In formula (4.18) the integer K represents the difference of the serial numbers of the input vector (X) received by the EOG and processing by $e(i,n)$ and $e(j,m)$. The behaviour of the *case operation* defined in the previous paragraph denotes in *Eq.* (4.18) that:

$$K \neq 0 , \tag{4.45}$$

holds for $e(i,n)$ and $e(j,m)$ being in different conditional branches belonging to the same case operation in the EOG. It means that a formal solution $K = 0$ of the inequality (4.17) does not denote the concurrence of $e(i,n)$ and $e(j,m)$ being in different conditional branches of the same case operation.

Let all solutions K be written as:

$$K = \dots k^{\wedge\wedge}, k^{\wedge}, 0, k', k'' \dots \tag{4.46}$$

This series has at least two elements if either of $c(i)$ and $c(j)$ is greater than 2 or both of them are greater than 1, then the interval $I = q(i) + q(j)$ in *Eq.* (4.27) is always greater or equal to $2R$. Let $c(i) \geq 2$ be assumed then $q(i) \geq 2R$ and $q(j) > 0$, so $I = q(i) + q(j) \geq 2R$. If $c(i) \geq 1$ and $c(j) \geq 1$ then $q(i) \geq R$ and $q(j) \geq R$, so $I = q(i) + q(j) \geq 2R$. These constraints for $c(i)$ and $c(j)$ can be written as $c(i) + c(j) > 3$, because both of them are always greater than 0 (by definition). Thus if $K = 0$ exists and $c(i)+c(j) > 3$ then $I = q(i)+q(j) \geq 2R$ which denotes that at least $K = k^{\wedge}$, 0 or $K = 0, k'$ always exists, too. In this case if $K = 0$ is excluded then:

$$0 < K \times R \leq b(i) - b(j) + q(i) \tag{4.47}$$

or

$$b(i) - b(j) - q(j) \leq K \times R < 0 , \tag{4.48}$$

is always true. Let *Eq.* (4.47) be assumed (the same process can be done with *Eq.* (4.48), too). From *Eq.* (4.47)

$$b(j) < b(i) + q(i) , \tag{4.49}$$

which denotes that $e(j)$ starts its busy state earlier than $e(i)$. Thus if $b(j)+q(j) \geq b(i)$ then $e(i,n)$ and $e(j,m)$ are concurrent. The opposite case:

$$b(j) + q(j) < b(i) \tag{4.50}$$

must be assumed for non-concurrency. From *Eq.* (4.47) as it was made in chapter 4.1.1

$$m \times c(j) \times R + (n + k(i,n) \times c(i)) \times R \leq k(j,m) \times c(j) \times R \qquad (4.51)$$

and

$$b(i) - b(j) - m \times c(j) \times R + (n + k(i,n) \times c(i)) \times R + q(i) \geq k(j,m) \times c(j) \times R \,. \qquad (4.52)$$

Introducing the notation:

$$B = m \times c(j) \times R + (n + k(i,n) \times c(i)) \times R \qquad (4.53)$$

and rewriting the inequalities (4.51) and (4.52) into one because the rights sides are identical:

$$B \leq k(j,m) \times c(j) \times R \leq B + b(i) - b(j) + q(i) \,. \qquad (4.54)$$

The inequality (4.54) involves an interval J for $k(j,m) \times [c(j) \times R]$:

$$J = b(i) - b(j) + q(i) \,. \qquad (4.55)$$

If this interval J is greater or equal to $[c(j) \times R]$ then $k(j,m)$ can always be found for any $k(i,n)$ in *Eq.* (4.53) which denotes that $e(i,n)$ and $e(j,m)$ are concurrent. From a rewritten form of *Eq.* (4.50):

$$q(j) < b(i) - b(j) \qquad (4.56)$$

a smaller number for $b(i) - b(j)$ can be substituted into (4.55), because if this J' (smaller than J) is greater than $[c(j) \times R]$, then $k(j,m)$ can always be found, too.

$$J' = q(j) + q(i) \,. \qquad (4.57)$$

Since *Eq.* (4.57) is similar to (4.27), the same steps to *Eqs.* (4.27)–(4.32) can be executed. In other words, if $c(i) + c(j) > 3$, then $e(i,n)$ and $e(j,m)$ are always concurrent even if they are in different conditional branches of a case operation. If $c(i) + c(j) \leq 3$ the *Eqs.* (4.17) and (4.18) must be calculated to decide the concurrence of $e(i,n)$ and $e(j,m)$.

4.2.1 Embedded Case Operations

Fig. 4.3 shows a situation in which $e(i,n)$ and $e(j,m)$ belong to the same case operation, but there is another case operation between the first case operation and $e(j,m)$. The worst case (the most frequent use of $e(j,m)$) involves that the second case operation activates only the conditional branch containing $e(j,m)$. In this case, the other branches of the second case operation can be ignored considering the concurrence between $e(i,n)$ and $e(j,m)$. Thus, the former constraints are unchanged. It is obvious that the conditional branches of the second case operation must be examined separately from the first one as shown in the previous section. This procedure can be applied for arbitrary number of case operations nested hierarchically according to *Fig. 4.3*.

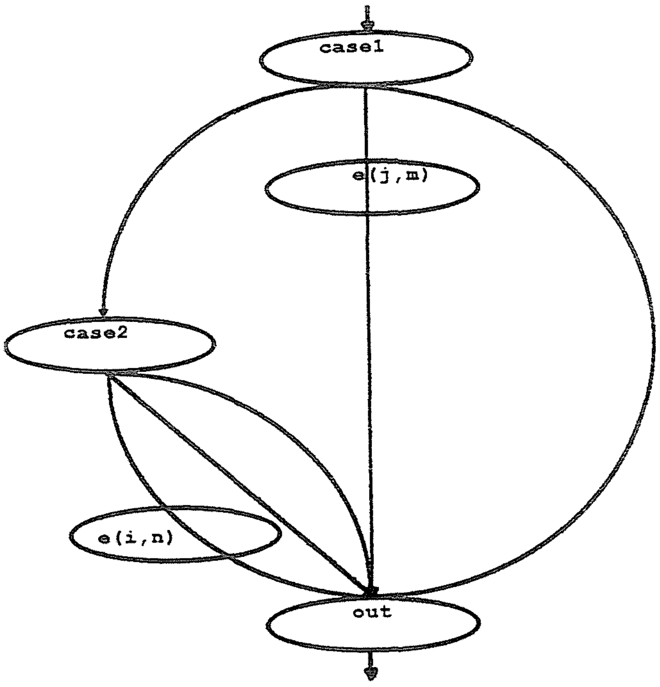

Fig. 4.3.

The algorithm, given in 4.1.2 section can handle the conditional branches with a simple modification :

```
for all $e(i)$ in EOG do
        for all $e(j)$ where $j > i$ do
            if $c(j) + c(i) \leq 3$ then
              $A = [b(i) - b(j) - q(j)]/R$
              $B = [b(i) - b(j) + q(i)]/R$
              if [$e(i)$ and $e(j)$ are in a different transfer
                sequence of a case operation and
                int($A$) = int($B$) = 0]
              or
                [int($A$) = int($B$) and sign($A$) = sign($B$) and
                int($A$)! = $A$ and int($B$)! = $B$]
              then
                $e(i)$ and $e(j)$ are not concurrent
```

The complexity of this algorithm is $O(n \times n/2)$ if the EOG has n operations before the multiplication.

5. Results

The program WinSam implements the method described in this paper. The input graphs of the FFT and the FIR filter [11] are shown in *Fig. 5.1* and *Fig. 5.2* as benchmarks. The results are summarized in *Table 5.1* and *Table 5.2*. The duration times are assumed 6 for a multiplier ($\times$), and 3 for an adder (+) and 1 for a buffer.

The third example in *Fig. 5.3* is designed to explain the advantage of handling the case operations as described in this paper. *Table 5.3* contains the results obtained by handling the operation named 'case' as a real conditional checking and not as an ordinary elementary operation. In this case, two processors can be saved for each R. (The duration times of the operations are shown in *Fig. 5.3*.)

Table 5.1
FFT

R	9	11	13	15
processors	36	34	29	28
buffers	46	46	46	25

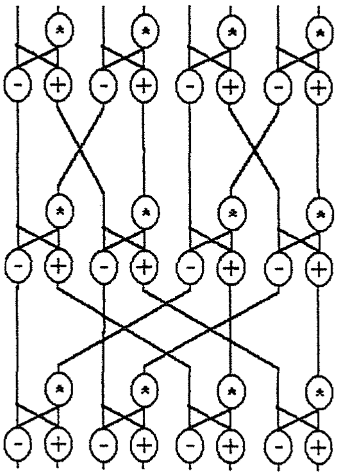

Fig. 5.1. FFT

Table 5.2
FIR filter

R	9	11	13	15
processors	23	22	22	22
buffers	16	8	0	0

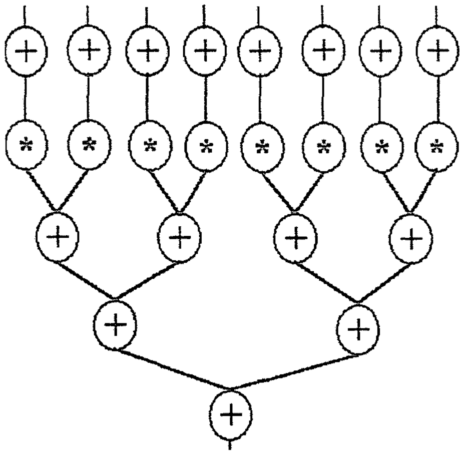

Fig. 5.2. FIR filter

Table 5.3
Example with case operation

R	9	11	13	15	17
processors (without case feature)	11	10	9	8	7
processors (with case feature)	9	8	7	6	5

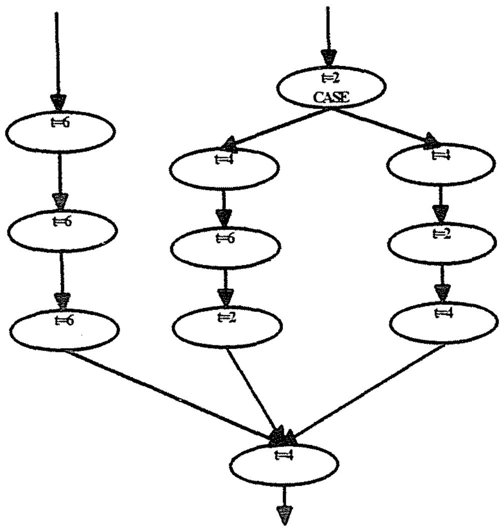

Fig. 5.3. Example with case operation

References

1. ARATÓ, P.: Logic Synthesis of VLSI Structures Based on a Pipelined Dataflow Model, Department of Process Control, Technical University of Budapest, Hungary.
2. ARATÓ, P. – BÉRES, I. – RUCINSKI, A. – DAVIS, R. – TORBERT, R.: A High-Level Datapath Synthesis Method for Pipelined Structures, *Microelectronics Journal*, Vol. 25, No. 3, 1995.
3. BÉRES, I.: Design Method for ASIC Signal Processing Units, Diploma Thesis at the Department of Process Control, Technical University of Budapest, Hungary, 1992 (in Hungarian).
4. CAMPOSANO, R.: From Behaviour to Structure: High-Level Synthesis, *IEEE Design and Test of Computers*, Vol. 10, pp. 8–19, 1990.
5. DEVADAS, S. – NEWTON, A. R.: Data Path Synthesis from Behavioural Descriptions: An Algorithmic Approach, *Int'l Symposium on Circuits and Systems*, Vol. 2, pp. 768–781, 1989.
6. DEVADAS, S. – NEWTON, A. R.: Algorithms for Hardware Allocation in Data Path Synthesis, *IEEE Transactions on Computer Aided Design*, Vol. 7, pp. 171–180, 1989.
7. DUTT, N. D. – GAJSKI DANIEL, D.: Design Synthesis and Silicon Compilation, in *IEEE Design and Test of Computers*, pp. 8–23, December 1990.

8. HWANG, C.-T. – LEE, J.-H. – HSU, Y.-C.: A Formal Approach to the Scheduling Problem in High Level Synthesis, in *IEEE Transactions on Computer-Aided Design*, Vol. 10, No. 4, pp. 464-475, April 1991.
9. PARK, N. – PARKER, A.: SEHWA: A Program for Synthesis of Pipelines, *Proc. 23rd. Design Automation Conference*, 1986, pp. 454–460.
10. PAULIN, P. G. – KNIGHT, J. P.: Force-Directed Scheduling for the Behavioural Synthesis of ASIC's, *IEEE Transactions on Computer Aided Design*, Vol. 6, pp. 661-679, 1989.
11. High-Level VLSI Synthesis, Edited by Raul Camposano, Wayne Wolf, Kulwer Academic Publisher, 1991.