CHAPTER 4

Data Memory Minimization

This chapter deals with some algorithms to compute the minimal data memory requirements for rapid prototyping of various DSP applications, represented as synchronous data flow graphs are presented. These algorithms are applicable to consistent live graphs, which may be open or closed, with or without loops, containing zero or any number of initial tokens. A uniform token size over the whole graph as well as a uniform buffer cost is assumed. As a consequence, the buffer sizes are only influenced by their lengths. Therefore effort is made on minimizing the buffer lengths for all edges of the graph, such that the total buffer length of the application is minimized and still a deadlock free schedule is found.

4.1 Memory requirement in ASIC-DSP systems

It is clear from sections 3.4 and 3.5 that a programming model and design flow of an environment, that is target device independent, can take ASICs/FPGAs into its fold without much of a problem. FPGAs are programmable devices, complement the capabilities of DSPs and provide speed gains in some cases. However, they have limited memory resources. A heterogeneous development environment having FPGAs as well as DSPs provide a platform to meet the requirements of real time challenges.
In literature there exist no immediately related studies or work which deals with minimization of memory resources for ASIC-DSP based heterogeneous systems which are employed in rapid-prototyping environments. The reasons for this are twofold. Traditionally DSP applications are mapped onto DSP processors. The most stringent demand concerns the execution speed. Memory limitations are not at all present or presented very lightly. Most often, it is simply assumed to be sufficient. The closest related work on the topic is found in [104], [105], [106], [107], [108]. On a single processor, both program and data memory are minimized. First, the program memory is minimized by constructing single appearance schedules. Within the category of single appearance schedules, the data memory can also be minimized, of course at the cost of execution time. This kind of minimization only applies to single execution units. Another subject is designing with low power considerations. There, the number of accesses to the data memory is observed and minimized, which however does not directly relate to the size of the memory itself.

In a rapid prototyping environment, the target already is provided with a fixed amount of memory resources. The only requirement is that the implementation fits into this memory. This is in contrast to the field of ASIC design, where the need to minimize data and other memory is very strongly present as part of the area-performance trade-off.

Furthermore, in this approach, only a fixed but unknown ordering is assumed. This self-timed approach
however guarantees proper inter-device synchronization. In ASIC design, typically the final architectures are tailored optimally to the specific application and the chosen technology. They will hardly match a general task-edge model as used in SDF. However, when it does, it generally requires distinct tasks for which the timing of production and consumption of tokens is fixed and known. This knowledge is then used to synchronize the total system to one global clock, using a global controller that organizes the task executions. Several strategies can then be developed, where for instance the memory is reused as much as possible [102] [109].

The use of FPGA components in the emulation setup for rapid prototyping and the use of multi-processor setups inspired this research. The results are also useful for single processor systems, when the assumption of relatively large internal memory would be invalid. As already discussed in Chapter 2, the FPGA components have very scarce memory resources. Minimal buffer lengths are hence needed to obtain a deadlock free implementation, at minimal cost.

In Chapter 2, it was explained that by default every task assigned to an FPGA would be implemented as a separate execution unit, accompanied with an extended firing rule test circuit. Consequently, since all tasks execute as soon as all data are present, no schedule is enforced anymore. On the contrary, the execution order may very well not exhibit any repetitive pattern. Hence no static analysis can be performed anymore to compute the needed buffer lengths from the global schedule. Since it is not advisable to
use dynamic memory management on FPGAs (due to huge overheads), fixed buffer lengths have to be defined prior to scheduling and implementation. Therefore, it is absolutely necessary to know what the minimal buffer lengths are that still allow the application to execute without deadlock.

Since memory elements on the FPGAs are expensive, it is even preferable to actually implement the minimal sizes required. A scheduler can then create a schedule adapted to the given lengths, to obtain an estimate of the total execution time of one iteration and to guide the scheduling of other parts of the application that may not be assigned to FPGAs. But for applications with relatively large data elements or large differences in production and consumption numbers on the edges, the total data memory could become a problem. Then the computation of the minimal required buffer lengths indicate whether it will be possible to find any schedule at all.

In various rapid prototyping platforms it is assumed that the devices used are programmable and as ASICs are not programmable, FPGAs are incorporated and then the design is ported to the ASICs for large-scale production. Hence in this work the FPGA based approach is dealt with and the obtained minimised and optimised design is then ported to the ASICs. Thus giving DSP-ASIC based solutions that are tailor made to suit the requirements.

4.2 Minimal Buffer Lengths in Chains

A chain is a directly connected graph of n nodes and n-1 edges [110], [111], [112]. In a chain one path encompasses a particular node passing all edges and
other nodes till the last node. The path encounters every edge and node exactly once.

4.2.1 Elementary chain

The smallest chain one can construct has one edge and two nodes, as shown in Figure 4.1. This is called

![Elementary chain diagram](image)

Figure 4.1 Elementary chain.

elementary chain. Node $n_1$ produces $p$ tokens and node $n_2$ consumes $c$ tokens at each firing of the node. A FIFO buffer $b$ on the edge stores the tokens in the graph that have been produced but not yet consumed. Such tokens are called live tokens $t$, in the sequence [113].

The state equation of the elementary chain [114], [115] expresses the evolution in terms of the live tokens in the buffer as a result of an initial number of tokens $t^i$ and the number of node executions $r_{n1}$ and $r_{n2}$ of the producing and consuming node respectively. It is given as shown in equation (4.1).

$$ t = t^i + r_{n1} \cdot p - r_{n2} \cdot c $$ (4.1)

To obtain a schedule that can be repeated infinitely with limited buffer lengths, it should return to its initial state at the end of one iteration. The minimal number of task executions
needed to reach the initial state is called the repetition rate of the nodes. These values are obtained easily from the state equation:

$$r_{n1}.p - r_{n2}.c = 0$$  \hspace{1cm} (4.2)

The solution for equation (4.2) in the non-zero positive integer domain is:

$$r_{n1} = k \cdot \{c / \gcd(p, c)\} \text{ and } r_{n2} = k \cdot \{p / \gcd(p, c)\}$$,

where $\gcd(p, c)$ denotes the greatest common divider of $p$ and $c$ and $k$ is a constant.

$k=1$ renders the smallest non-zero integer solution:

$$r_{n1} = 1 \cdot \{c / \gcd(p, c)\} = c' \text{ and }$$

$$r_{n2} = 1 \cdot \{p / \gcd(p, c)\} = p'$$.  \hspace{1cm} (4.2a)

There exists no consistency condition. In other words, any combination of production and consumption numbers will lead to finite buffer lengths. Many schedules are possible that execute node $n_1$, $r_{n1}$ times and node $n_2$, $r_{n2}$ times. For each schedule, the maximum number of live tokens that occurs on the edge determines the required buffer length. Minimizing the buffer length thus corresponds to determining an appropriate scheduling methodology.

For the current graph, it is clear that a schedule that executes node $n_2$ whenever possible will achieve the goal, since it consumes as much tokens as possible before producing new tokens to the buffer and starts consuming tokens as soon as possible. Figure 4.2 shows an example where this principle is applied. For $p=4$ and $c=10,$
The schedule is constructed such that \( n_2 \) always has priority over \( n_1 \). Beneath the schedule, the number of live tokens is shown. The maximum is 12, which is much smaller than the worst case where \( n_1 \) is executed \( r_{n_1} \) times consecutively: \( r_{n_1} \cdot p = 20 \).

![Graph](image.png)

Schedule: \( n_1 \ n_1 \ n_1 \ n_2 \ n_1 \ n_1 \ n_2 \)

\[ t: \quad 0 \quad 4 \quad 8 \quad 12 \quad 2 \quad 6 \quad 10 \quad 0 \]

**Figure 4.2** Elementary graph with the subsequent states of the buffer corresponding to the schedule.

This priority principle is in fact generally applicable for all feed-forward graphs. A schedule that assigns increasing priority levels from the start to the end nodes of the graph will result in minimal buffer length requirements as it keeps the number of live tokens in the buffers as small as possible. In graphs with complexity, it is not evident how priority should be assigned such that the required buffer lengths are minimized. This calls for other more analytical methods to be employed.

In the next section, the expression in terms of \( p \) and \( c \) to compute the maximum number of live tokens, i.e. the minimal buffer length, is proved.
4.2.2 Computation of minimal buffer length

The buffer length of an elementary chain depends on the number of tokens produced and consumed by the nodes as well as their execution rates. The expression for minimal buffer length to avoid deadlock is derived in lemma 4.1.

Lemma 4.1:

Let $l$ denote the length of buffer $b$ and $t$ the number of tokens present at a time in $b$. Let the initial state of the buffer be empty ($t^1 = 0$). Then the minimum buffer length for an elementary chain can be expressed as:

$$l_{\text{min}} = p + c - \gcd(p, c)$$

Proof:

Case I

Let $t_{\text{max}} \leq p + c - \gcd(p, c)$

Initially, the number of tokens $t = 0$ and only $n_1$ is executable. It is executed, increasing $t$, until $t \geq c$. At that moment, $n_2$ is executable and thus it will be executed, decreasing $t$ until again $t < c$. From this behavior, it can be deduced that $n_2$ was not executable before execution of $n_1$. The number of tokens reaches $t_{\text{max}}$ due to this execution of $n_1$. Here, $t_{\text{max}}$ is the maximum number of tokens on the buffer. Mathematically it can be expressed as:

$$t + p = t_{\text{max}} \text{ while } 0 \leq t < c.$$  

This can be rewritten as:

$$t_{\text{max}} < p + c.$$
Since every value of \( t \) is a linear combination of \( p \) and \( c \), \( t \) can only be a multiple of their \( \gcd \) thus \( t_{\text{max}} \leq p+c-\gcd(p,c) \).

Case II

Let \( t_{\text{max}} \geq p+c-\gcd(p,c) \)

One period of the schedule consists of \( p'+c' \) steps. Hence \( t \) will loop through as many different values. Since \( t \) starts at zero, already one value is known. It is also known that \( t \) is a multiple of \( \gcd(p,c) \). If every multiple of \( \gcd(p,c) \) would occur as a state, the maximum value \( t \) can reach would be \( (p'+c'-1).\gcd(p,c) \). If however one or more of these values do not occur in the states, its place has to be taken by another and hence, a larger value will be there. Thus it can be stated that:

\[
t_{\text{max}} \geq (p'+c'-1).\gcd(p,c) = p+c-\gcd(p,c)
\]

Therefore, it can now be concluded that

\[
l_{\text{min}} = t_{\text{max}} = p+c-\gcd(p,c).
\]

Notice that in a graph without initial tokens, the minimal buffer length is such that only one node is executable at a time. To accommodate the presence of initial tokens, the equation has to be extended as given below in lemma 4.2.

4.2.3 Effect of initial tokens

The presence of initial tokens affects the length of the buffer required for a deadlock free schedule. The expression to calculate this effect is derived in next lemma.
Lemma 4.2:

Let \( t^i \) denote the initial tokens in buffer \( b \) and \( d \) denote \( \gcd(p,c) \). Then the required minimal buffer size is:

\[
\ell_{\text{min}} = \max\{p+c-d+(t^i \mod d), t^i\} \tag{4.4}
\]

Proof:

The state equation for a buffer with \( t^i \) initial tokens is:

\[
t = t^i + k_1.p - k_2.c
\]

\[
= (t^i \div (d) + k_1.p' - k_2.c').d + t^i \mod (d)
\]

where \( k_1 \) and \( k_2 \) are constants.

In a periodic schedule that returns the buffer to its initial state, the nodes \( n_1 \) and \( n_2 \) have to be executed \( k.p' \) and \( k.c' \) times, with \( k=1 \) producing the shortest schedule. If \( t^i \) corresponds to a possible state of the buffer in the schedule with zero initial tokens, the schedule for the initially present tokens \( t^i \) actually starts at the point where the state of buffer \( b \) reaches \( t^i \). When it reaches the end of that schedule, it loops back to the beginning to continue until again the beginning state \( (t = t^i) \) is reached. Therefore, the minimal buffer length from lemma 4.1 is still valid. If \( t^i \) however does not correspond to a possible state of \( b \) in the schedule with zero initial tokens, the largest part of \( t^i \), corresponding to maximum number of tokens, is determined in such a way that:

- if \( t^i > p+c-d \) then largest part corresponds to \( p+c-d \).
- and
- if \( t^i \leq p+c-d \) then this corresponds to \( t^i - t^i \mod (d) \).
For largest part, it represents a starting point in the original schedule and thus, does not increase the previously determined $l_{\text{min}}$. The remainder of $t^i$, which is $t^i - (p+c-d)$ in the first case and $t^i \mod(d)$ in the second case, however acts as permanently present number of tokens, which adds a constant to this length. Hence it can be concluded that for:

$$t^i > p+c-d \Rightarrow l_{\text{min}} = t^i$$
$$t^i \leq p+c-d \Rightarrow l_{\text{min}} = p+c-d + (t^i \mod(d))$$

This can be compacted into equation (4.4).

An example is shown in Figure 4.3 in which initial tokens are present. The minimum length of the buffer is seen to increase from the case when there are no initial tokens.

![Figure 4.3: Chain with 5 initial tokens.](image)

The $\gcd(p,c)$ i.e. of 4 and 10 is 2 and with zero initial tokens, the minimal length would be 12 as in case of chain in Figure 4.2. From lemma 4.2, $t^i \mod d$ (5 mod 2) i.e. one from the 5 initial tokens will permanently be present in the schedule. This increases the minimal length by one as given by:

$$l_{\text{min}} = \max(p+c-d+(t^i \mod d), t^i),$$

resulting in $l_{\text{min}} = 13$. 

110
An alternative method to obtain the same result is to formulate the conditions for deadlock and then negate this condition to obtain a guaranteed deadlock free result. Let \( \delta = t \mod(d) \):

- a deadlock requires that the consuming node can not execute due to insufficient tokens on the buffer, hence the buffer contains less than \( c \) tokens. Since the number of tokens in the buffer is always of the form: \( t = k \cdot d + \delta \), this can be rephrased as \( t < c - d + \delta \).

- at the same time the producing node should not be able to execute, due to a lack of empty locations in the buffer, hence the buffer should contain at least \( t \) tokens for which \( l - t < p \) holds good.

Combination of the two conditions results in:

\[
1 - p < t \leq c - d + \delta\]

which in turn gives

\[
l < p + c - d + \delta
\]

Thus a deadlock can only occur if \( l < p + c - d + \delta \). Vice versa, if \( l \geq p + c - d + \delta \), the elementary chain is guaranteed to be deadlock free. Of course, the final length has to be large enough to accommodate for the initial tokens, hence the minimal length for an elementary chain is given by:

\[
l = \max\{p + c - d + \delta, t^i\}\]

where \( \delta = t^i \mod(d) \).

This is, however, identical to the result expressed in equation (4.4).

### 4.3 n-node chains

In an n-node chain, it is still perfectly possible to find a schedule that does not deadlock
without adapting buffer lengths. The individual minimal lengths from previous section can hence be retained. This is proved in lemma 4.3.

Lemma 4.3:

The minimal buffer requirement for an n-node chain is equal to

\[ \sum_{k=1}^{n} l_{k}^{\text{min}} \]

Where \( l_{k}^{\text{min}} \) is the individual minimal length for buffer \( b_k \), possibly with initial tokens, as expressed in equation (4.4).

Proof:

From its definition, a chain will have exactly two nodes with only one edge in the chain. These nodes are called the border nodes of the chain. The other nodes, having two edges in the chain, are called intermediate nodes.

Suppose initially every buffer is assigned with its individual minimal length as given by equation (4.4). In the following section it will be proved that it is impossible to create a deadlock situation for a chain with the individual minimal buffer lengths of edges.

In a deadlock situation every node should be blocked on at least one of its incident edges. This is the case when, on at least one of its input buffers at most \( c-d \) tokens are present; or when on at least one of its output buffers at most \( p-d \) empty locations i.e. at least \( 1-p+d \) tokens, are available.
A procedure that attempts to create a deadlock situation for all possible combinations of nodes in a chain is described in the following description:

Choose one of the border nodes i.e. either sink or source and examine it:

Step1) put at least 1-p+d tokens on the output buffer of the chosen node and examine the connected node:

a) if it is a sink with one input buffer (border node)
then there is no deadlock.
Since 1 is always ≥ p+c-d, there are at least c tokens in its input buffer connecting to the previous node.

b) if it is a sink with two input buffers
then the node is not blocked on this buffer.
Since 1 is always ≥ p+c-d, there are at least c tokens in its input buffer connecting to the previous node.
Go to Step2 i.e. take the case of input buffer.

c) if it is a passing node connected with its input buffers
then the node is not blocked on this buffer.
Since 1 is always ≥ p+c-d, there are at least c tokens in its input buffer connecting to the previous node.
Go to Step1.

Step2) an input buffer connects the chosen node to the adjacent unexamined node:
put at most c-d tokens in its input buffer and examine the connected node.
a) if it is a source with one output buffer (border node) then there is no deadlock. Since 1 is always ≥ p+c-d, there are at least p empty places in its output buffer connecting to the previous node.

b) if it is a source with two output buffers Then the node is not blocked on this buffer. Since 1 is always ≥ p+c-d, there are at least p empty places in its output buffer connecting to the previous node. Go to Step1.

c) if it is a passing node connected with its output buffer The node is not blocked on this buffer. Since 1 is always ≥ p+c-d, there are at least c tokens in its input buffer connecting to the previous node. Go to Step2.

This procedure ends only when the second border node is encountered. Since the result for this node always indicates that it is not blocked, it can be concluded that it is impossible to have a deadlock in a chain with minimal buffer lengths. Hence the original individual minimal lengths can be maintained. The minimal buffer length of a chain is therefore simply the summation of the minimal buffer lengths for every elementary chain.

4.3.1 Feed-forward chains

The special chains where all edges have the same orientation as depicted in Figure 4.4, which will be
called feed-forward chains, have an extra property that only one buffer at a time can reach its maximum number of live tokens in a chain. It is deduced in the next lemma.

\[ \begin{array}{ccccccc}
  n_1 & p_1 & c_1 & n_2 & p_2 & c_2 & \ldots & n_{N-1} & p_{N-1} & c_{N-1} & n_N \\
  b_1 &   & c_1 & b_2 &   & c_2 &   & b_{N-1} &   & c_{N-1} & b_N 
\end{array} \]

*Figure 4.4: A feed-forward chain of n nodes.*

**Lemma 4.4:**

In any state of a feed-forward chain, with increasing execution priority from the first to the last node, no more than one buffer at a time will reach its maximum number of live tokens.

**Proof:**

Let \( b_k \) denote the \( k^{th} \) buffer in the graph (AG), \( p_k \) and \( c_k \) being the production and consumption of tokens onto that buffer respectively and \( t_k \) the current number of live tokens in \( b_k \). Suppose all \( t_k < c_k \) at a certain moment. Then the only possible action is to execute the first node, until the condition \( t_1 \geq c_1 \) is satisfied in the first buffer. At that instant, \( t_1 \) could reach \( l_1^{\text{min}} \). But the next action is to execute node \( n_2 \), decreasing \( t_1 \) and increasing \( t_2 \). If in \( b_2 \), \( c_2 \) can be reached, \( n_3 \) will be executed and this way, the tokens are passed on to the last node. If \( c_2 \) can not be reached, then certainly it is not maximum either. Thus, when the first node starts executing again, maybe leading to the maximum in \( b_1 \), \( b_2 \) is not at
maximum (anymore). The same theory holds for all other buffers in the graph.

### 4.4 Conclusion

In this chapter, the equation to compute the minimal buffer length of an elementary chain has been developed. This is the basic equation for all further computations on graphs, since every graph is composed of elementary chains. It is shown as to how in the special class of graphs i.e. the chains; the minimal buffer length is simply the sum of the minimal buffer lengths of its composing elementary chains. It is also clear that only one buffer can reach the maximum number of tokens at a time in a feed-forward chain.