Stream Communication between Real-Time Tasks in a High-Performance Multiprocessor

Jeroen A.J. Leijten[1,2], Jef L. van Meerbergen[1], Adwin H. Timmer[1], and Jochen A.G. Jess[2]

[1] Philips Research Laboratories, WAY4, Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands
[2] Eindhoven University of Technology, PO Box 513, 5600 MB Eindhoven, The Netherlands

Abstract

The demands in terms of processing performance, communication bandwidth and real-time throughput of many multimedia applications are much higher than today's processing architectures can deliver. The PROPHID *heterogeneous multiprocessor architecture template aims to bridge this gap. The template contains a general purpose processor connected to a central bus, as well as several high-performance application domain specific processors. A high-throughput communication network is used to meet the high bandwidth requirements between these processors. In this network multiple time-division-multiplexed data streams are transferred over several parallel physical channels. This paper presents a method for guaranteeing the throughput for hard-real-time streams in such a network. At compile time sufficient bandwidth is assigned to these streams. The assignment can be determined in polynomial time. Remaining bandwidth is assigned to soft-real-time streams at run time. We thus achieve efficient stream communication with guaranteed performance.*

1 Introduction

Multimedia applications require the execution of a large number of tasks on a variety of multimedia data. The amount of processing power demanded by such applications is very large, typically dozens of GOPS [7]. The performance of single processor solutions exploiting fine-grain instruction-level parallelism is insufficient to meet such demands. Much higher performance is obtained by making use of coarse-grain task-level parallelism. This enables parallel execution of large independent tasks and can be exploited in powerful multiprocessor solutions.

The granularity of tasks depends heavily on the application domain. Normally, this granularity is chosen such that the complexity of each task is manageable for the application designer and that efficient reuse of tasks and modular design are possible. An example of a task of typical granularity is variable length decoding (VLD) in MPEG. With such a granularity, the required intertask communication bandwidth may be as high as dozens of Gbits per second. Common single-bus and shared-memory solutions fall short in providing such large bandwidths at reasonable costs.

For different tasks different timing constraints with respect to throughput and latency may exist. Often these are real-time constraints, which must be met under all circumstances. However, some tasks may have more lenient timing constraints using fall-back mechanisms, and can be regarded as soft-real-time tasks, with less strict deadlines. In a typical multimedia application, these different types of tasks coexist.

This paper is concerned with guaranteeing the throughput of streams with hard-real-time constraints in the PROPHID multiprocessor architecture [5][6], while mixing these streams with soft-real-time streams. To increase hardware efficiency, resource sharing is supported via multiplexing of processing and communication resources. We will show that real-time throughput under such circumstances can be guaranteed by using some ideas from the field of *digital switching* [8], normally used in, for example, telephony networks. We will extend these ideas to the PROPHID multiprocessor architecture.

2 PROPHID architecture

Figure 1 shows the PROPHID heterogeneous multiprocessor architecture template [6]. This template can be instantiated for a given target application domain. This means that the number of processors and the type of each processor may be different for each IC implementation. Basically, the architecture is aimed at high-throughput applications, requiring about 20 processors each delivering 1 GOPS, with a total interprocessor communication bandwidth of about 40 Gbits/sec.

The template consists of two main parts. First, a general purpose microprocessor (CPU) is responsible for control-oriented tasks, such as the interaction with the environment, and possibly low to medium performance signal processing tasks. Secondly, a number of *application domain specific* (ADS) processors implement the time or power critical tasks. These processors can span a wide range of programmability going from fixed-datapath-like units at one end to fully programmable processors at the other end. A central bus connects the CPU to the ADS processors. A second communication network is used to meet the required intertask communication bandwidth of dozens of Gbits per second. The high-throughput signal data of the ADS processors is sent through this communication network.

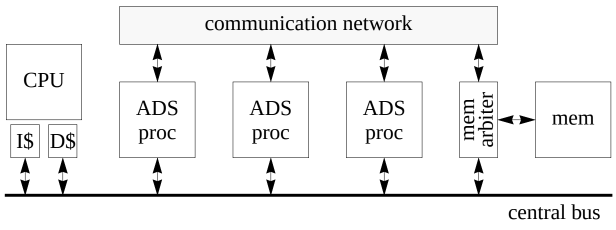

Figure 1 PROPHID heterogeneous high-performance multiprocessor architecture.

Both parts of the architecture have access to a shared background memory via a dedicated memory arbiter that efficiently arbitrates between the more or less randomly distributed central bus requests and the primarily periodic requests from the high-throughput communication network.

The remainder of this paper will focus on the high-performance part of the architecture, that is, the ADS processors and the high-throughput communication network. In particular, we will describe how the throughput of hard-real-time streams is guaranteed in the communication network. In Section 7 we will explain how, after these bandwidth requirements have been satisfied, the remaining bandwidth can be used to transfer soft-real-time streams. First, however, we will introduce our model of an ADS processor.

3 High-performance processing

The high-performance signal processing part of an application can be represented by task graphs, in which the nodes represent large autonomous tasks and the edges represent stream communication between these tasks [5][6][10]. To obtain high performance, concurrent execution of tasks is exploited by using multiple autonomous ADS processors that can perform tasks independently, and thus in parallel with other processors (see Figure 1).

Because high-throughput tasks require large amounts of data, buffering of data on communication channels is an important issue. Due to bandwidth limitations such buffering cannot be implemented in shared background memory. Because the stream concept implies that the order of the communicated data is preserved, a natural implementation of a communication channel between ADS processors is a first-in-first-out (FIFO) buffer.

To increase hardware efficiency, multiple tasks can be executed on the same processor. These tasks must be executed *time-interleaved*, in order to keep buffer sizes to a minimum. On each ADS processor time-interleaved execution is possible for all tasks that originate in the application domain for which the specific ADS processor has been optimized. Since the rate of high-throughput multimedia data is of the same order of magnitude as the clock rate of any feasible implementation, the number of tasks that can be executed interleaved on the same processor is limited, typically to a number between one and five.

Processors that perform time-interleaved execution of different tasks have to be able to save the state of a suspended task and restore the state upon resumption of that task. In the case of high-performance signal processing where the data rate is proportional to the clock rate, little time and bandwidth are available to save or restore state. Therefore, the state of suspended tasks cannot leave the processor to be stored elsewhere, for instance in background memory. Fast context switches can be achieved only by using local memory inside the processor to temporarily save state. This leads to processors with multiple state spaces for storing data and shared logic for performing computations on the data. Our model of an ADS processor is given in Figure 2. Here, for a processor p, $\tilde{I}(p)$ and $\tilde{O}(p)$ are defined as the set of input terminals and the set of output terminals, respectively, and $m(p)$ is defined as the number of state spaces. With $m(p)$ state spaces a maximum number of $m(p)$ tasks can be executed interleaved on the processor, without external context saves. Depending on the costs, the exact implementation of the model may be different for each ADS processor, without violating the model.

Performing a context switch without penalty is possible if the context to be saved is zero. Therefore, the same state space can be reassigned to another task if the state of the task which is currently using that state space is zero. For example, many video algorithms have zero state during the vertical blanking. Hence, it is possible to reassign processor state spaces to new tasks in the blanking. In this way a switch to a completely different set of tasks can be performed.

When several tasks want to use the same processor hardware, conflicts may occur. To avoid this, a controller in each processor decides which task is to be activated. This means that each processor has *local* control over processing. To guarantee that all the tasks are completed on time, such a controller must ensure fair scheduling of processing tasks. To enable fair scheduling and prevent starvation or deadlock, the order in which data belonging to different streams must be processed may be different from the order in which streams arrive at a processor input terminal. Therefore, each processor state space requires its own input and output FIFOs. Accordingly, $m(p)$ FIFOs are connected to each terminal of processor p in Figure 2. Later on we will show that several FIFOs may share the same on-chip memory.

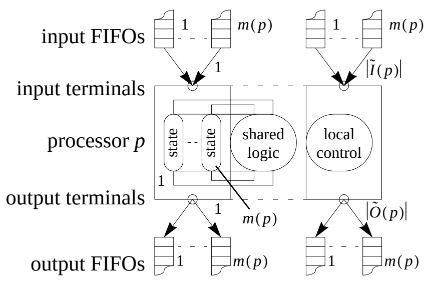

Figure 2 Processor p with $|\tilde{I}(p)|$ input terminals, $|\tilde{O}(p)|$ output terminals and $m(p)$ state spaces.

4 Communication network

The function of the communication network is to provide high-throughput connections between all the ADS processors. We can define this function more formally as follows. We define A as the set of FIFOs connected to processor output terminals, and B as the set of FIFOs connected to processor input terminals. A can also be regarded as the set of input terminals of the communication network, and B as the set of output terminals of the communication network. The function of the communication network is to provide connections from FIFOs $a \in A$ to FIFOs $b \in B$. For a set P of ADS-processors the total numbers of input terminals $|A|$ and output terminals $|B|$ of the communication network are given by

$$|A| = \sum_{p \in P} |\tilde{O}(p)| \cdot m(p), \tag{1}$$

$$|B| = \sum_{p \in P} |\tilde{I}(p)| \cdot m(p). \tag{2}$$

The problem of creating connections between input and output terminals for periodic, or continuous, data streams has been thoroughly studied in the field of circuit switching [8]. A network is said to be *nonblocking* if a free input channel can always be connected to a free output channel, irrespective of other connections. To limit the number of switches required in a network, without losing the important nonblocking property, multiple switching stages are used. In these stages combinations of so-called *time division* (T) switches and *space division* (S) switches usually are used.

The analogy between T-switches and S-switches is shown in Figure 3. In an S-switch physical switches are used to connect input wires to output wires, thus creating physical links between input and output channels. In a T-switch, a single physical line is used to transport the different streams to be switched. To this end these streams are grouped into service cycles, with samples of different streams stored in separate time slots within such a service cycle. A T-switch changes the order of time slots within a service cycle and thus creates links between input and output channels. In the example of Figure 3 stream s_1 is transported from input channel a_2 to output channel b_3 and stream s_2 is transported from input channel a_4 to output channel b_2.

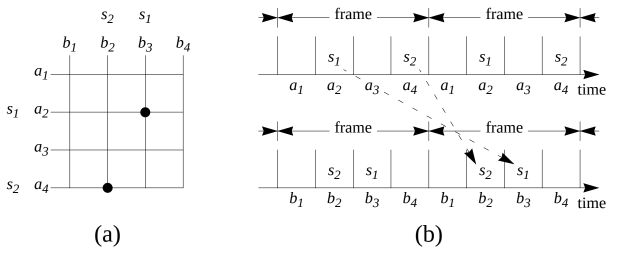

Figure 3 Analogy between (a) switching in the space domain and (b) switching in the time domain.

A well-known three-stage switching network is the T-S-T network. The first and third stages of this network consist of T-switches, whereas the second stage consists of a time-shared S-switch, in which new physical connections between input and output channels are created for each time slot. The time-sharing of physical communication lines is obtained by dividing time into service cycles, where each service cycle consists of a set K of time slots (see Figure 4).

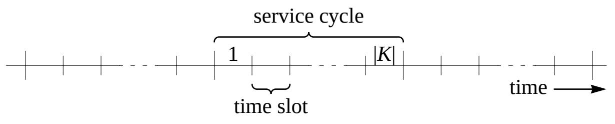

Figure 4 Definitions of time.

With K being the set of time slots available on the time-shared S-switch in the second stage, n is the maximum number of time slots that can be used by streams sharing the input of a T-switch in the first stage, or the output of a T-switch in the third stage. Naturally, it is required that

$$n \le |K|. \tag{3}$$

Clos [2] derived the condition for a three-stage network to be nonblocking, which is given by

$$|K| \ge 2n - 1. \tag{4}$$

This requirement stems from the assumption that existing connections cannot be reconfigured when a connection is added, because this would temporarily stall communication transfers. In regular T-S-T networks, such as telephony networks, this is a valid assumption. However, this assumption is not necessary for the PROPHID communication network. In fact, we will show in Section 5 that $|K| = n$ is sufficient if all the connections in the communication network may be reconfigured each time a new connection is needed. First, however, we will explain how, by analogy with the concepts of a T-S-T network, a three-stage time-shared network limits the cost of the PROPHID high-throughput communication network.

Since an ADS processor and the communication network are autonomous units, the order in which a collection of FIFOs is accessed by these units may differ. Such behaviour at the boundaries of a collection of FIFO buffers corresponds to the functionality of a T-switch. In fact, we can create T-switches by grouping FIFOs and connecting them to a single terminal of a time-shared connection network, that is, an S-switch. This is done as follows.

We define X and Y as the set of input terminals and the set of output terminals, respectively, of the connection network. Let $R = I \cup O$ be the set of all the processor terminals, where

$$I = \bigcup_{p \in P} \tilde{I}(p) \quad \text{and} \quad O = \bigcup_{p \in P} \tilde{O}(p)$$

are the set of input terminals and the set of output terminals, respectively, of all the processors. Let for FIFO $a \in A$, $o(a) \in O$ be the processor output terminal and $x(a) \in X$ be the connection network input terminal to which this FIFO is connected. Similarly, let for FIFO $b \in B$, $i(b) \in I$ be the processor input terminal and $y(b) \in Y$ be the connection network output terminal to which this FIFO is connected (see Figure 5).

Note that although multiple FIFOs can be connected to a single processor terminal, the entire bandwidth available to that processor terminal can, in an extreme situation, be conveyed via a single FIFO to the same connection network terminal. Therefore, to count the minimum amount of bandwidth needed by a connection network terminal, we must sum the bandwidths required by all processor network terminals that have access to that connection network terminal via one or more FIFOs. For each processor terminal $r \in R$ we define $\tilde{c}(r)$ as the maximum amount of communication bandwidth in terms of time slots per service cycle needed by r. Let $c(x)$ and $c(y)$ be the total number of time slots per service cycle required at network terminal $x \in X$ and $y \in Y$, respectively, that is

$$c(x) = \sum_{r \in O | \exists_{a \in A} : r = o(a) \wedge x(a) = x} \tilde{c}(r) \quad \text{for all } x \in X, \quad (5)$$

$$c(y) = \sum_{r \in I | \exists_{b \in B} : r = i(b) \wedge y(b) = y} \tilde{c}(r) \quad \text{for all } y \in Y. \quad (6)$$

Equations (5) and (6) state that the number of time slots required at a network terminal is equal to the sum of the number of time slots required by all the processor terminals connected to that network terminal via one or more FIFO buffers. The T-S-T network parameter n is determined by the maximum number of time slots required by any network terminal, hence

$$n = max(\{c(x) | x \in X\} \cup \{c(y) | y \in Y\}) \quad (7)$$

Given a specific application domain, designers can choose a network configuration in the range of networks with $1 \leq |X| \leq |A|$ and $1 \leq |Y| \leq |B|$. Here, the extreme $|X| = |Y| = 1$ implies a fully shared single bus solution, whereas the extreme with $|X| = |A|$ and $|Y| = |B|$ implies a fully connected S-switch without sharing. Given the chosen set of input terminals X and output terminals Y of the S-switch, groups can be made of arbitrary FIFOs, to construct T-switches. By cleverly choosing groups, an optimum configuration with comparable bandwidth requirements on each physical channel can be determined. Note, however, that the bandwidth of the S-switch may not be exceeded, that is, condition (3) should hold. If this condition is not satisfied, one of the following three measures must be taken:

- choose a different grouping of FIFOs resulting in a lower n, such that $n \leq |K|$,
- increase the clock frequency of the communication network, thereby increasing the amount of bandwidth available per time slot and thus decreasing $\tilde{c}(r)$, and consequently n, such that $n \leq |K|$,
- increase the clock frequency of the communication network as well as the number of time slots $|K|$ per service cycle, thereby increasing the amount of time slots while keeping the amount of bandwidth per time slot constant. Then n remains constant, while $|K|$ increases such that $n \leq |K|$.

In the next section we will show that, given a network configuration that satisfies condition (3), $|K| = n$ time slots on the S-switch are sufficient to be able to assign adequate communication bandwidth to all hard-real-time streams.

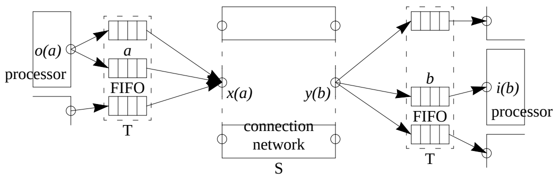

Figure 5 T-S-T network configuration.

5 Time slot assignment

5.1 The problem

The problem of finding a time slot assignment can be mathematically formulated as follows. Given is a set S_h of hard real-time streams. Let for stream $s \in S_h$, $a(s) \in A$ be the FIFO connected to the connection network input terminal $x(a(s)) \in X$, and $b(s) \in B$ to be the FIFO connected to the connection network output terminal $y(b(s)) \in Y$ between which stream s must be transferred. The binding of streams to FIFOs is performed at compile time, so that each stream is mapped onto a unique FIFO attached to a connection network input terminal and a unique FIFO attached to a connection network output terminal.

In our applications some streams may require more bandwidth than others. Therefore, we assume that several time slots can be assigned to the same stream. For each stream $s \in S_h$ the bandwidth required for communication between $x(a(s))$ and $y(b(s))$ is given in terms of a number of time slots $\tilde{d}(s) \in \{1, 2, \ldots, |K|\}$.

We define $d(x)$ and $d(y)$ to be the sums of time slots required by streams $s \in S_h$ that are mapped onto connection network terminals $x \in X$ and $y \in Y$, respectively, that is

$$\forall_{x \in X} : d(x) = \sum_{s \in S_h | x(a(s)) = x} \tilde{d}(s), \tag{11}$$

$$\forall_{y \in Y} : d(y) = \sum_{s \in S_h | y(b(s)) = y} \tilde{d}(s). \tag{12}$$

For any application the number of time slots required at a single connection network terminal may not exceed the maximum bandwidth that can be offered by such a terminal. If more bandwidth is required by an application, the application does not fit and is not feasible for the given instance. Therefore, for any feasible application,

$$\forall_{x \in X} : d(x) \le c(x), \tag{13}$$

$$\forall_{y \in Y} : d(y) \le c(y). \tag{14}$$

with $c(x)$ and $c(y)$ given by equations (5) and (6), respectively. Therefore, after applying equation (7) the following conditions can be derived:

$$\forall_{x \in X} : d(x) \le n, \tag{15}$$

$$\forall_{y \in Y} : d(y) \le n. \tag{16}$$

The assignment problem for the available bandwidth can be described as follows. Determine for all $s \in S_h$, a set of time slots $\tilde{K}(s) \subseteq K$, such that

$$|\tilde{K}(s)| = \tilde{d}(s), \tag{17}$$

$$\forall_{s_1, s_2 \in S_h | x(a(s_1)) = x(a(s_2)) \wedge s_1 \ne s_2} : \tilde{K}(s_1) \cap \tilde{K}(s_2) = \varnothing, \tag{18}$$

$$\forall_{s_1, s_2 \in S_h | y(b(s_1)) = y(b(s_2)) \wedge s_1 \ne s_2} : \tilde{K}(s_1) \cap \tilde{K}(s_2) = \varnothing. \tag{19}$$

Equation (17) states that the bandwidth obtained in terms of time slots must correspond to the required bandwidth. Equations (18) and (19) state that an input or output terminal of the connection network may not be assigned twice in one and the same time slot.

5.2 Solution

The assignment problem described above can be transformed into a bipartite multigraph matching problem. Consider a bipartite multigraph $Q = (X, Y, E)$, with the set of vertices constituted by the set of input terminals X and the set of output terminals Y, and E being the set of edges between X and Y. The set of edges E is constructed as follows. For every time slot required between a pair of input and output terminals an edge is added, that is, for each $s \in S_h$, $\tilde{d}(s)$ edges between $x(a(s))$ and $y(b(s))$ are added to E. Note that the number of edges between two vertices may be more than one. According to (15) and (16) the maximum degree, that is, the maximum number of edges attached to the same vertex, in the bipartite multigraph is n. Therefore, the maximum number of edges between two vertices is also equal to n.

The original assignment problem can be restated as follows: find a q-colouring[1] of the edges in the bipartite multigraph with $q \le n$. Then, every colour represents a single time slot to which different streams are assigned. To prove that a q-colouring with $q \le n$ always exists, we use a corollary derived from the König-Hall theorem [3].

Corollary. *In a bipartite multigraph $G = (X, Y, E)$, there exists a matching that saturates[2] all the vertices with maximum degree.* ❑

From this corollary the following theorem can be derived.

Theorem. *The chromatic index[3] of a bipartite multigraph G with maximum degree h is $q(G) = h$.*

Proof. From the corollary above, graph $G = (X, Y, E)$ contains a matching E_0 that saturates every vertex of degree h. Colour the edges in E_0 with the first colour. Next, consider the bipartite multigraph $G_0 = (X, Y, E - E_0)$, constructed by removing all the edges in E_0 from G. The maximum degree of G_0 is $h - 1$. Graph G_0 contains a matching E_1 that saturates every vertex of the degree $h - 1$. Colour E_1 with the second colour, and proceed with $G_1 = (X, Y, E - E_0 - E_1)$. On repetition of this procedure, all the edges of G will be coloured, using h colours. ❑

Since the maximum degree of graph Q is n we conclude from the above theorem that a colouring of graph Q with a maximum of n colours exists. Hence, the theorem proves that $|K| = n$ time slots are sufficient to transport all streams. Thus, the connection network can have a minimum size and the available time slots can be fully utilized.

1. A q-colouring of the edges of a graph $G = (X, Y, E)$ is defined as a partition of the edge set E into q subsets that are matchings. A matching of G is defined to be a set $E_0 \subset E$ of edges such that no two edges of E_0 are adjacent, that is, no two edges are attached to the same vertex [1].
2. A vertex $v \in X \cup Y$ is said to be saturated by a matching E_0 if an edge of E_0 is attached to v [1].
3. The chromatic index $q(G)$ of a graph G is defined as the smallest number of colours needed to colour the edges of G such that no two adjacent edges have the same colour [1].

The complexity of finding a maximum matching in a bipartite multigraph $G = (V, E)$ with $V = X \cup Y$ is $O((|E| + |V|)\sqrt{|V|})$ [4]. For the time slot assignment the number of edges $|E|$ in the bipartite multigraph Q is at most $(1/2)(|X| + |Y|)|K|$ and the number of matchings to be found is at most $|K|$. Therefore, the complexity of finding a time slot assignment is $O((|X| + |Y|)^{3/2}|K|^2)$.

6 High-performance architecture

An implementation of the PROPHID high-performance architecture part is given in Figure 6.

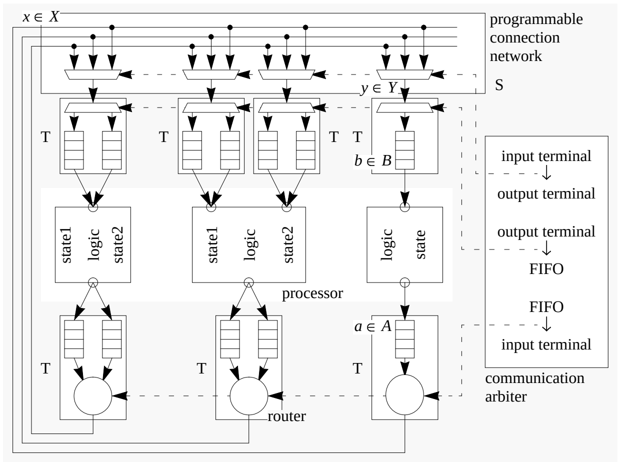

Figure 6 High-performance part of the architecture.

In the architecture the first stage of T-switches is formed by the boxes containing the FIFOs of set A. Special router units form the interface between these FIFOs and the inputs of the communication network, depicted in grey. A programmable connection network, which in this case is a switch matrix, contained in the communication network, forms the time-shared S-switch. Finally, the third stage of T-switches is formed by the boxes containing the FIFOs of set B. In the example shown in Figure 6 three ADS processors with several terminals and state spaces are depicted. Here, the T-S-T configuration has been chosen such that a connection network terminal has been created for every processor terminal, so $X = O$ and $Y = I$. FIFOs are grouped so that all FIFOs of a single processor terminal are connected to the same T-switch. With this configuration the T-S-T parameter n is uniquely determined by the maximum communication bandwidth required by a single terminal of any processor. In cases where the maximum communication bandwidth requirement of all the processor terminals is of the same order of magnitude this is a viable option. This is the case in many video applications.

The communication network is controlled by a special communication arbiter that provides instructions to the network. These instructions determine in each time slot which FIFOs in A are connected to connection network input terminals in X, which connection network input terminals in X are connected to which connection network output terminals in Y, and which FIFOs in B are connected to connection network output terminals in Y. This means that the communication arbiter has *global* control over the creation of connections in the communication network.

7 Mixing hard and soft real-time streams

7.1 Hard real-time streams

The time slot assignment obtained using the q-colouring method of Section 5 can be represented in the form of a table (see Figure 7). The rows of this table represent $|K|$ different time slots. Furthermore, three groups of columns represent instructions for the communication network in each time slot. The first group defines for each input terminal of the connection network from which FIFO it may read data. The second group defines for each output terminal of the connection network to which input terminal it is connected. Finally, the third group defines for each output terminal of the connection network to which FIFO data may be written. Each time a new time slot starts the corresponding row in the time slot table is selected by the communication arbiter as the active row. Switching to a new row costs zero clock cycles. Therefore, no bandwidth is lost between time slots. Such a bandwidth efficiency would not be possible with regular bus protocols, which are usually optimized for random traffic instead of continuous streams. The overhead involved in the setup of bus connections using regular bus protocols seriously complicates guaranteeing real-time throughput.

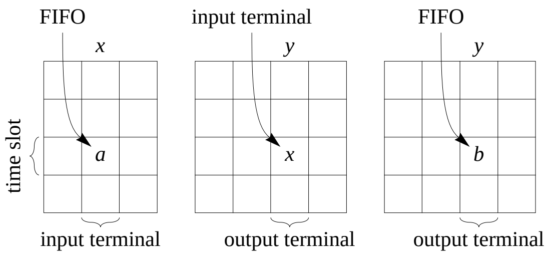

Figure 7 Time slot assignment table.

At compile time the terminals and FIFOs to be used by each stream in a set of task graphs are fixed and a time slot assignment is calculated for hard real-time streams. In the field blanking of a video signal this binding and the corresponding time slot assignment is loaded into the time slot table. By reloading the table a new set of task graphs can be activated in each blanking.

7.2 Soft real-time streams

Time slots that remain unused by real-time streams can be used for streams that have no hard real-time deadlines.

To this end the communication arbiter uses an additional instruction row that is filled at run-time (see Figure 8). By merging the active row of the time slot table and this soft real-time row, the instruction code for the network is obtained. During the merging, entries in the active row of the time slot table have the highest priority to ensure that the bandwidth for hard real-time streams is guaranteed. Common bus arbitration schemes are suitable for filling the soft real-time row. The details of filling this row are however beyond the scope of this paper.

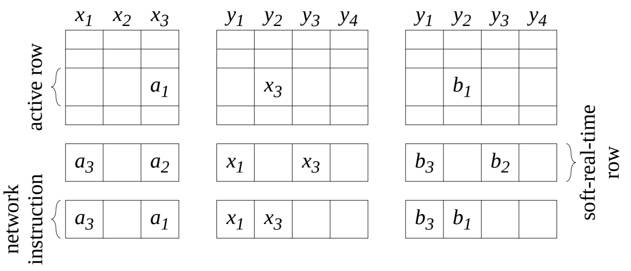

Figure 8 Merging hard and soft real-time streams.

A typical application for PROPHID is multiwindow television. In this application video streams must be treated as hard real-time streams, while teletext pictures may be treated as soft real-time streams. The field blanking of the video streams leaves sufficient unused bandwidth to schedule soft real-time streams, such that, for example, the scaling of a teletext picture takes only a few video frames to complete.

8 Implementation costs

We analysed the costs involved in implementing the high-throughput communication network. We designed a 64 MHz, 1.9 mm^2 communication arbiter in a 5-metal layer 0.35 micron technology using VHDL synthesis and standard-cell layout. Using the same method and technology we implemented a 64 MHz, 0.1 mm^2 router serving four FIFOs. Experiences with the Philips Video Signal Processor (VSP) [9] have shown that a full switch matrix provides a favourable ratio of bandwidth over area cost. With the increased transistor density of today's semiconductor technology a standard cell implementation of a switch matrix in a limited area is a good option, especially since our multiplexing scheme limits the size of the switch matrix. The cost of buffering will depend on the number of time slots per service cycle and on the size (in number of clock cycles) of a single time slot. By keeping the size of a time slot small, it will be possible to keep buffer sizes small, too. For example, a typical video application uses a maximum multiplexing factor of 4 tasks per processor, with a service cycle size of 4 time slots, and a time slot size of 16 clock cycles. The corresponding typical FIFO size is 32 samples. Furthermore, several FIFOs sharing the same network and processor terminals are never accessed at the same time for reading or writing, and can therefore be mapped onto the same memory, which limits the implementation costs of these FIFOs.

9 Conclusions

We have presented a method for ensuring real-time throughput for data streams in a high-performance multiprocessor architecture, which is part of the PROPHID heterogeneous multiprocessor architecture template for multimedia applications. Our method involves a compile-time assignment of bandwidth for hard real-time streams, using a bipartite graph colouring algorithm. This is done by dividing time into service cycles consisting of a number of time slots. In each time slot the correct links are created in a shared programmable connection network. A special communication arbiter schedules data transfers at run time, using the compile-time assignments as a starting point. Time slots that are not assigned to hard real-time streams at compile time can be assigned to soft real-time streams at run time by this arbiter. With these concepts, the PROPHID architecture is capable of transferring multiple hard real-time as well as soft real-time data streams efficiently over a number of parallel physical links, making optimal use of the available bandwidth.

References

[1] C. Berge, "Graphs", *North-Holland Mathematical Library*, vol. 6, part 1, 3rd rev., North-Holland, Amsterdam.

[2] C. Clos, "A Study of Nonblocking Switching Networks", *Bell System Technical Journal*, vol. 32, no. 2, pp. 406-424, 1953.

[3] P. Hall, "On Representations of Subsets", *Journal of the London Mathematical Society*, vol. 10, pp. 26-30, 1934.

[4] J.E. Hopcroft and R.M. Karp, "An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs", *SIAM Journal of Computing*, vol. 2, no. 4, pp. 225-231, December 1973.

[5] J.A.J. Leijten et al., "PROPHID: A Data-Driven Multi-Processor Architecture for High-Performance DSP", *Proceedings of the 1997 European Design & Test Conference*, March 1997.

[6] J.A.J. Leijten et al., "PROPHID: A Heterogeneous Multi-Processor Architecture for Multimedia", *Proceedings of the 1997 International Conference on Computer Design*, October 1997.

[7] B. de Loore et al., "A Video Signal Processor for Motion-Compensated Field-Rate Upconversion in Consumer Television", *Proceedings of the 1996 International Solid-State Circuits Conference*, pp. 248-249, February 1996.

[8] M. Schwartz, "Telecommunication Networks: Protocols, Modeling and Analysis", *Addison-Wesley*, 1987.

[9] H. Veendrick et al., "A 1.5 GIPS Video Signal Processor (VSP)", *Proceedings of the 1994 Custom Integrated Circuits Conference*, pp. 95-98, May 1994.

[10] J.A. Watlington and V.M. Bove, "Stream-Based Computing and Future Television", *Proceedings of the 137th SMPTE Technical Conference*, pp. 69-79, September 1995.