Heterogeneous multiprocessor for the management of real-time video and graphics streams

DOI:
10.1109/4.881220

Document status and date:
Published: 01/01/2000

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Heterogeneous Multiprocessor for the Management of Real-Time Video and Graphics Streams

Marino T. J. Strik, *Member, IEEE*, Adwin H. Timmer, Jef L. van Meerbergen, *Senior Member, IEEE*, and Gert-Jan van Rootselaar

***Abstract*—This paper presents an application domain driven approach to the design of embedded systems on silicon, and it shows how this approach is used to design a chip for a multi-window TV application. We discuss all major design steps in a logical order starting with an application domain analysis. This lead to the choice of Kahn data flow graphs as the programming paradigm for high-throughput signal applications. Based on this analysis we designed a multiprocessor architecture which uses run-time reconfiguration. Finally, attention is spent toward the physical implementation and the deep-submicron problems we had to solve. The result is a chip that can manage up to 25 internal real-time video streams. The chip combines the flexibility of a programmable solution with the cost effectiveness of a consumer product.**

I. Introduction

THE DESIGN of embedded VLSI systems poses many challenges in areas such as design complexity, low power versus high speed, and hardware/software codesign. Given the advances in process technology, it will be increasingly difficult to utilize the intrinsic compute power of silicon by means of a monolithic single processor architecture. The reason is that the number and speed of functional units in a VLSI system are not dominant for the performance anymore. The main issue is to keep as many functional units busy as possible, which can only be achieved when *true task level parallelism* is exploited in a multiprocessor architecture, next to the instruction level parallelism inside one processor. Such a multiprocessor will also help to overcome one of the major future bottlenecks with respect to the performance of ICs, that is, the power consumption. By processing as much as possible in parallel, the clock frequencies and the V_{DD}, and therefore the power consumption, can be kept at acceptable levels. All this leads to a focus shift from computational aspects of an architecture (e.g., the pipelining of functional units) to the communication aspects of an architecture (e.g., the interconnection network and synchronization between processors).

Next to the on-chip communication issues, the bandwidth to off-chip memory is more and more a limiting factor for the performance of embedded systems as well. New memory types, like DDR SDRAM or Rambus memories, are not sufficient to handle the pace in which the bandwidth needs increase. It becomes especially critical when CPUs, peripherals, and other (co-)processors must use the same background memory in a unified memory architecture (UMA). In media systems for instance, signal processing applications, like MPEG video decoding, display processing, etc., must obtain a—more or less—guaranteed bandwidth, while the CPU and possibly some peripherals require low latency for the best performance. The interface and arbitration to background memory is therefore of growing importance.

Despite the popularity of state-of-the-art general-purpose CPUs, low cost and low power remain the dominant issues for the architectural trade-offs of embedded systems. For media applications, classical programmable solutions fail because of performance and a too low intrinsic computational efficiency (ICE) [1]. If we take a video stream with a 16-MHz pixel rate and a minimum of 50 operations per pixel, then we need at least 800 MOPS/stream. This would require about one CPU to perform just one function on one stream. This is not a cost-effective solution for consumer equipment for two reasons. First, the ICE can be improved by two orders of magnitude by more application specific implementations for the majority of the functions, and especially for the 'number crunching' parts of video applications. Those parts often do not require that much programmability and can therefore be implemented in a more dedicated fashion, in contrast to the higher control layers of video applications. Second, normal software stacks use coarse grain synchronization, in the video domain typically fields or frames, which leads to an explosion of the off-chip memory bandwidth costs.

Nevertheless, the software contents and flexibility of embedded systems are increasing, but the area costs and power consumption must be kept limited. This calls for a platform approach, in which a multitude of cores (CPUs, DSPs, and coprocessors) can be easily integrated into one system. With such a *heterogeneous* approach, the cost and power efficiency of DSPs and coprocessors can be combined with the flexibility of general-purpose CPUs, in order to obtain the best solution and trade-offs with respect to software programmability, flexibility, silicon area, and power consumption.

In this paper, we introduce a new solution for the *concurrent* processing of many *real-time* multimedia (video) streams, that addresses the most important issues of future embedded media systems. The presented heterogeneous multiprocessor architecture is built around new on-chip communication and synchronization concepts that enable:

1) true task level parallelism;
2) optimal use of the bandwidth to background memory;
3) sufficient programmability at lowest costs.

Manuscript received April 3, 2000; revised June 20, 2000.

The authors are with Philips Research Laboratories, Eindhoven 5656 AA, The Netherlands (e-mail: marino.strik@philips.com).

Publisher Item Identifier S 0018-9200(00)09425-7.

0018–9200/00$10.00 © 2000 IEEE

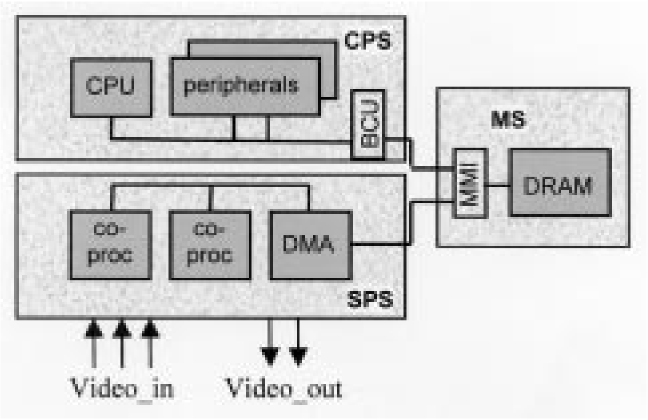

Fig. 1. Overall system partitioning.

Our solution can be seen as a subsystem within an overall architecture (see Fig. 1). In this overall architecture, a distinction is made between a typical control processing subsystem (CPS) (e.g., host processor applications and event-driven functionality), a signal processing subsystem (SPS), and a memory subsystem (MS). In this paper, we concentrate on the SPS subsystem.

In our first VLSI implementation, the IC has three independent and uncorrelated video input channels and two independent and uncorrelated video output channels. They are stored in and retrieved from an external SDRAM to be displayed on a TV set using PC-like multiple windows. Graphics data generated by an external CPU are read from SDRAM to be blended with the composition of the video. This way new functions such as internet access, electronic program guides, and e-commerce are added to the classical TV functions, while maintaining a high-end display quality and a simple user interface. Internally, the chip can exchange up to 25 video streams of 16-MHz pixel rate or more between the different processors in parallel. A dedicated streambased direct memory access (DMA) unit supports 20 streams going from and to an external SDRAM.

The main challenges for this design can be summarized as follows:

1) The chip supports multiple video windows with variable sizes. Sizing can easily be done incrementally at run-time without visual artifacts. Common TV architectures are not flexible enough to handle such functionality, while in common PC architectures there is no way that the absence of visual artifacts can be enforced in a setting with live video in multiple dynamically changing windows.
2) The chip delivers unprecedented video enhancement quality with functions such as high-quality scaling, dynamic noise reduction, peaking, CTI, and so on.
3) It allows for on-chip communication between multiple processors, alleviating from the bandwidth bottleneck to background memory.
4) Management of up to 25 internal video streams with hard real-time constraints.
5) Optimal utilization of the bandwidth to background memory by means of special DMA and arbitration schemes.
6) Unique combination of flexibility and efficiency of the implementation. Basic video processing kernels are identified and implemented in an efficient way. These kernels can subsequently be combined at the top (application) level in many different ways to implement different applications.

Because of the separation in the overall architecture between control and signal processing, we can use a tailored programming paradigm for the signal processing subsystem (see Section II). The new system concepts explained in Section III exploit the characteristics of that programming paradigm extensively. Of course, the design of the processor cores and the complete IC must fit the system concepts, which is the topic of Section IV. In Section V, we describe the chip metrics of a first IC realization.

II. Programming Paradigm

The classical (embedded) software paradigm is based on a sequential description of one or more tasks (threads). Those tasks synchronize for instance by means of interrupts or semaphores. The different tasks can be scheduled at run time by an operating system or real-time kernel. In most cases, that scheduling is based on fixed priority scheduling, where the priorities are determined by a rate monotonic analysis [3].

For a number of reasons, the classical software paradigm depicted above is not valid for high-throughput high-performance signal processing applications that have to be mapped onto a multiprocessor architecture with on-chip data communication between processors.

1) Signal processing applications are characterized by more or less periodic input and output streams of samples. Host applications and control-dominated applications on the other hand are characterized by their control constructs and possible event driven nature. For instance, branch prediction is an issue in such applications, while it is not much of an issue in signal processing applications.
2) Because signal processing applications work on large sets of samples, the notion of reconfigurable computing is apparent. For instance, if video samples are communicated between two tasks, such a 'channel' will exist for at least one field period, and in practice much longer. Such characteristics of the application domain can be used, for instance in the bus arbitration.
3) Fixed priority scheduling used in normal real-time kernels assume that the tasks to be scheduled are independent, meaning that the schedule and completion time of one task does not depend on the schedule and completion time of another task. However, if video tasks have to communicate large amounts of data on-chip to alleviate the bandwidth bottleneck to off-chip memory, then the tasks can not be treated as being independent anymore. Therefore, other scheduling methods have to be used, and we will show that a *self-scheduling approach* can be applied for such signal processing tasks.

For the reasons mentioned above, we choose to use a different programming paradigm for the signal processing part of an em-

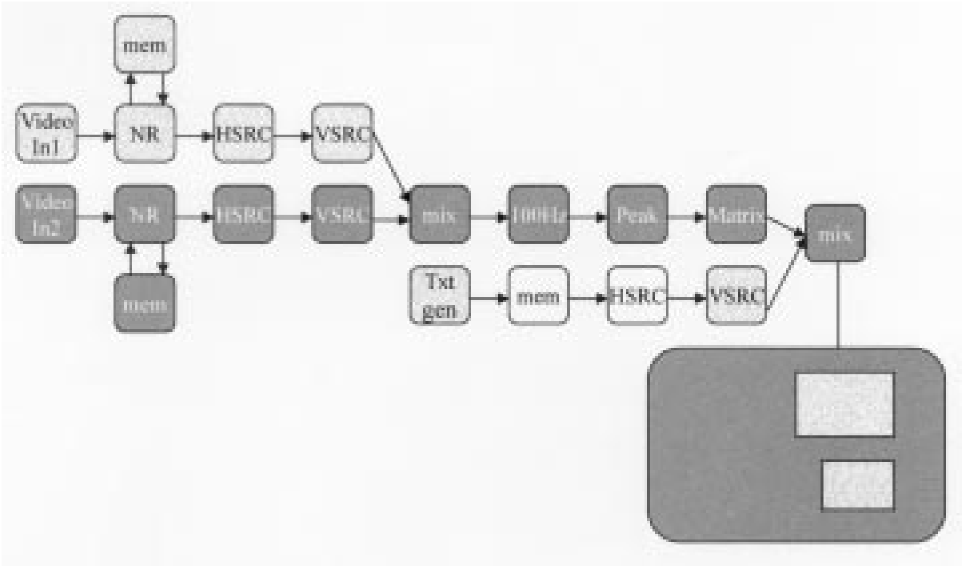

Fig. 2. Kahn process network.

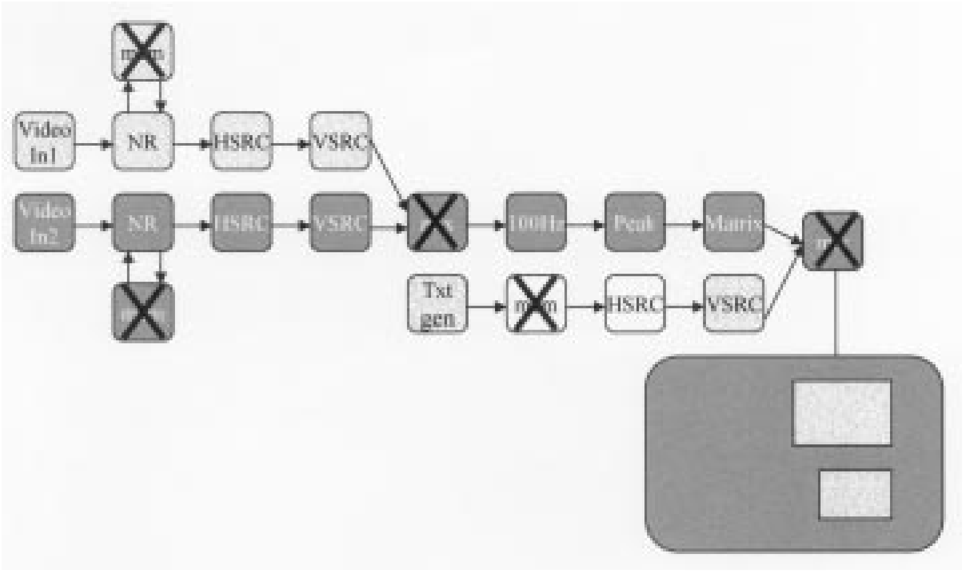

Fig. 3. Connections to background memory.

bedded system, in contrast to the normal approaches for general-purpose architectures. In our case, we model signal processing applications as Kahn dynamic data flow graphs [2], consisting of tasks interconnected by logical first-in-first-out (FIFO) channels (see Fig. 2). For the control processing part of the system we use standard approaches.

Fig. 2 shows an example of an application that is modeled using Kahn data flow graphs. The nodes in the graph represent basic video functions. The set of functions is limited and characteristic for the application domain. It includes noise reduction, vertical and horizontal sampling rate conversion, sharpness enhancement, video juggling, graphics blending, and so on. Different applications are represented by different graphs. The fact that we are dealing with well-defined kernels, which have to be connected in a flexible way, calls for reconfigurable computing. The switching between applications must be done on a field by field basis without artifacts visible on a TV set. Therefore, the reconfiguration must be done dynamically at run time.

Since the bandwidth to external memory is the major design constraint, it makes sense to distinguish two different situations when implementing FIFO channels of the Kahn graph. Some channels can be implemented using on-chip communication while in other situations the involvement of external memory can not be avoided. This is, for example, the case when two video streams, which are not synchronized, are mixed. The places where we have to 'cut' the graph and make connections to external memory are indicated in Fig. 3 for the example that we are using.

Summarizing, we make a distinction between the following levels of hierarchy:

1) an application graph: a Kahn graph that represents a complete mode setting, for example for a TV;

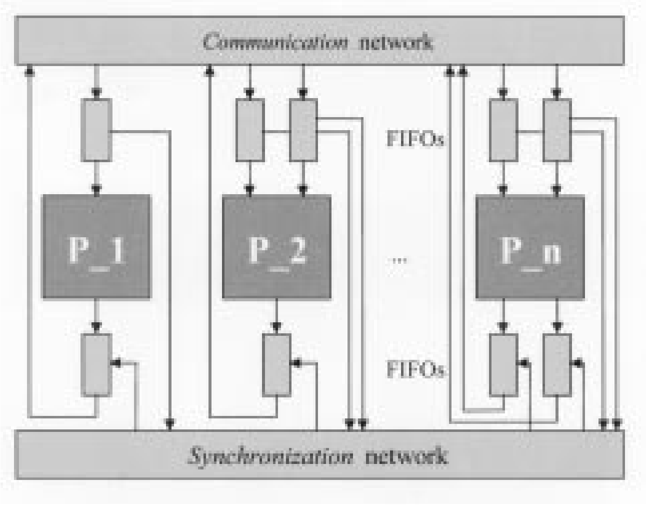

Fig. 4. Communication infrastructure.

2) a subgraph: a set of closely coupled tasks, which internally can communicate via on-chip means. Communication between two different subgraphs takes place via external memory;
3) a task represented by one node in the Kahn graph.

In what follows, we will develop the architecture step-by-step. We start by discussing the on-chip communication.

III. System Concepts and Architecture

A. Component and Communication-Based Design

Since the set of video processing functions is limited and known in advance we decided to develop for each function a separate processor which is optimized for the particular task. To allow the mapping of different application graphs, a reconfigurable network [9], [10] is added to the architecture (see Fig. 4). The network is programmable so that different application graphs can be executed.

An important architecting principle we applied is the *separation of concerns*, in our case, the separation between computation and communication. By inserting local buffers at each input and output of the processor, it is possible to decouple the calculation of new data from the transport of it. Second, the choice for FIFO buffers is motivated. They have been adopted because the edges in the Kahn graph represent (video) *signals*, i.e., measurable physical quantities sampled at discrete points in time and binary encoded. The only identification of the different samples in the stream is given by the order of the samples. Samples are produced only once and can not be lost on the communication channels. For streams with the aforementioned features, separation of communication and processing can be done with FIFOs. Third, the architecture is based on a *blocking* protocol, i.e., processors are stopped when at least one of the input FIFOs is empty or at least one of the output FIFOs is full. The reasons are as follows:

1) It allows for an efficient implementation since the buffer sizes can be kept small. In the current design, the size is equal to 32 pixels.
2) The field blanking, that is, the nonactive part of a video signal, can be used for soft real-time tasks. The detection whether a stream is in the blanking or not is a run-time decision, since input streams are not synchronous with respect to each other. This reuse for soft real-time tasks is easily implemented with a dynamic stream based model.
3) A stream-based computing model is often simpler to implement in comparison with a static synchronous system, because it can perform runtime (self-) scheduling based on local availability of data.
4) The concept is better scalable with respect to the addition or removal of processors. It is expected that processors will become increasingly dynamic. A good example is a variable-length decoder (VLD), which produces and consumes a data-dependent number of tokens.

Next to the *communication* aspects described above, we have to solve the *synchronization* issues. Again, we had to choose a solution which is totally different from current systems where the CPU synchronizes the different processors via an interrupt mechanism. Because of the high-throughput rates and the small grain size for on-chip communication, we have to go for a hardware-oriented approach. The problem is that the FIFOs at the output of the sending processor must be blocked when an input FIFO of a receiving processor is full. This is done by implementing a *synchronization network* (see Fig. 4). Inputs to this network are the full flags of the input FIFOs of the receiving processors. Outputs of the synchronization network are send to the output FIFOs of the sending processors. Via the correct programming of the network, the correct FIFO status can be passed on. This connection is always the inverse connection of the communication network: inputs and outputs are interchanged. This way we can synchronize at a very fine grain size, that is, at the level of individual pixels.

B. Processor Model

The same function (e.g., horizontal sampling-rate conversion, HS) can appear more than once in a Kahn graph. For cost reasons, the different instances of the same function will all be executed as different tasks on the same processor. This leads to the processor model [10] shown in Fig. 5. This figure shows an example with two input ports and two output ports. This could, for example, be a temporal noise reduction coprocessor that needs an input video signal and the filtered result from the temporal loop as inputs and has a video output and a backward channel to memory as outputs.

In our first IC realization, a processor can execute maximally four different task instances. Each input and output port is connected to a maximum of four (logical) FIFOs, labeled 1–4. In a similar way the state memory is duplicated four times. In this way we can relate independent FIFOs and state memories to each task that is executed on the processor. By not sharing the state memories we can have task switches on a clock cycle basis. It is thus possible in our approach to perform very fast task switching without the need for context saves. In the field of high-throughput signal processing, context saves are very expensive, as a huge amount of task state can be involved.

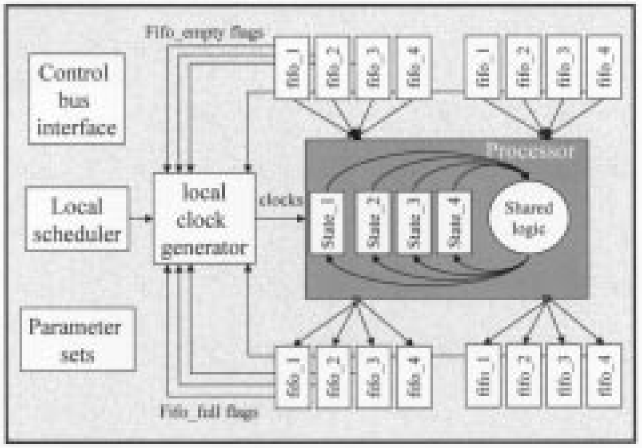

Fig. 5. Processor model and surrounding shell.

The processor is surrounded by a shell, as shown in Fig. 5. The shell is a generic interface between the processor and the communication network. It performs several functions. An important task is to stop the processor when an input FIFO is empty or an output FIFO is full. This is implemented by manipulating the clock in the "local clock generator". The details of the implementation are discussed later. Another task of the shell is to provide the interface toward a control bus. Via this interface parameters or instructions can be loaded which control the mode settings for the different tasks that are executed on the processor. The storage of these parameter sets is done by the shell.

C. Communication Network

The task of the communication network is to provide sufficient bandwidth for the on-chip data streams between the output and the input FIFOs of the processors. Different application graphs can be executed because the connections in the network are programmable. For every connection in the application graph, a path is created in the connection network using circuit switching.

The network is a so-called TST network with space and time switches (see Fig. 6). The reason to build such a network is to guarantee nonblocking connections between output FIFOs of processors and input FIFOs of succeeding processors, with a predetermined amount of bandwidth for each connection. At the input, a multiplexer is added, and at the output a demultiplexer is added. Using four time slots, the total bandwidth equals the sum of bandwidths of the individual channels. In the example of Fig. 6, two paths through the network are indicated.

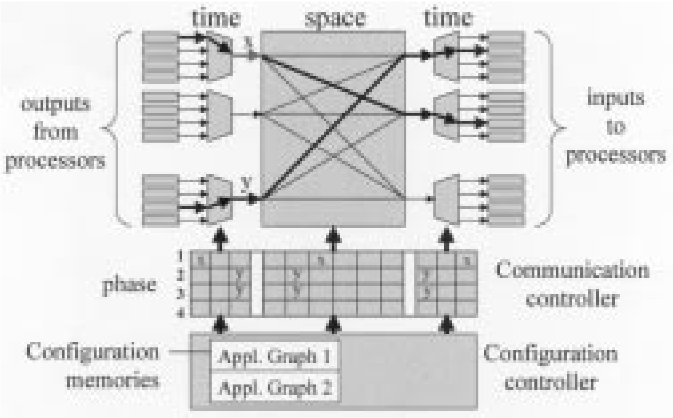

Fig. 6. Communication network.

Each path is controlled by the *communication controller*, which is basically the equivalent of a bus control unit (BCU) in a single bus architecture. This controller is responsible for the lowest level of control, i.e., the control of the steady-state situation within one and the same video field. It basically consists of a control memory with four different phases, which are activated cyclically. The number of four is related to maximal number of time slots in the communication network. The TST network is programmed by putting the correct code for the three different parts of the network in one or more phases, see Fig. 6. For example, the x connection is programmed in phase 1 via the correct code at the positions labeled with an x. This way, bandwidth is allocated corresponding to one phase. The connection labeled y has twice the bandwidth of one channel, because it is programmed during two phases.

At the next level of control we have designed the *configuration controller* (see Fig. 6). An essential element of this controller is the *configuration memory*, a memory that contains all information related to all communication channels of a complete application. This information is needed by the communication controller to set up a communication channel. This means that the correct information is written into the phase table of the communication controller. This can be interpreted as *activating* a communication. The phase table of the communication controller contains all channels that are activated at that moment. Note that the information in the configuration memory can be calculated at compile time, as the maximum bandwidth needs are known for all channels in an application graph [6].

We have included not one but two configuration memories. The reason is the following. Since we have to process two or more video streams that are not synchronized, it is impossible to define some point in time at which we can switch the whole application at once, since the field blankings may never coincide. This means that part of the application is still processing according to the previous settings while some other part is already executing a new application. A gradual and run-time controlled transition of the activity between the two application graphs is necessary. This is only possible using two configuration memories. In this way, we can have dynamically changing applications without visual artifacts in a display.

D. Background Memory Arbitration and Stream Caching

In the previous sections, we only discussed on-chip communication and synchronization concepts. Of course, the accesses to off-chip memory are very important for the overall performance of a system. As is shown in Fig. 7, a distinction can be made between random requests (in bursts) to background memory from CPUs and peripherals, and more periodic requests originating from signal processing applications. While a CPU or peripheral needs low latency for the best performance, signal processing/media applications need guaranteed bandwidth. The reason that signal processing applications do not need low latency is that they access background in a very regular predetermined manner, such that prefetching can be done optimally.

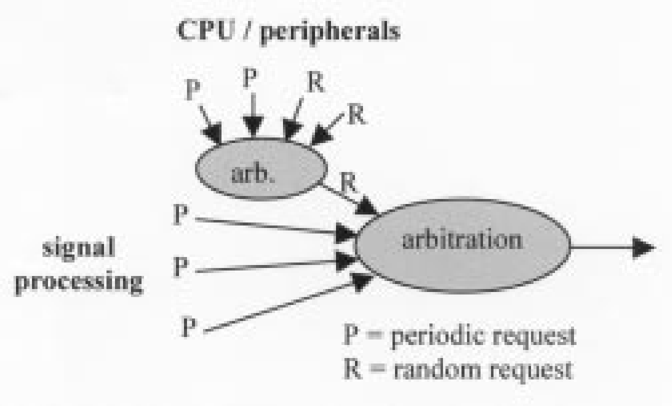

Fig. 7. Background memory arbitration.

To accommodate both types of requests, we implemented the arbitration scheme from [7]. In that scheme, a service cycle of N clock cycles is defined, in which M clock cycles are reserved for periodic requests. As long as enough cycles are available for the periodic requests, the random requests have highest priority. If there are just sufficient cycles left for the periodic requests, they are granted highest priority, instead of the random requests. In a well-balanced nonsaturated system, this scheme will give the highest possible performance, by keeping the average latency for the random requests as small as possible. In a saturated system (e.g., in a system in which the CPU is requesting too much bandwidth), any arbitration scheme will give the same average latency. In that case this arbitration scheme is suboptimal, as the variance of the latency can be quite high.

With the background memory arbitration discussed above, the amount of on-chip buffering required for each stream from and to background memory can be quite high. The reason is that random requests are allowed to monopolize the background memory for many clock cycles, if the number of clock cycles N in one service cycle is large enough. In [8] it is shown, that from an area and flexibility point of view, it is far more advantageous to have one central buffer pool between the processors and background memory, instead of local buffers at each processor.

In our IC realization, this central buffer pool is implemented as a kind of stream cache, in which 20 streams from and to background memory can be accommodated. Because one has to program the amount of buffering for each stream, and one can program the prefetching strategy, this stream cache can also be regarded as a kind of DMA engine.

IV. Design and Physical Implementation

A. Local Clock Generator

In Fig. 5, the important role of the local clock generator was already discussed. In this section we will discuss the details of the implementation (see Fig. 8). The basic idea is that tasks can be stopped by gating the clock. A running clock means that the corresponding tasks is active and that it is not blocked. The selection of the active task is done by the scheduler and the blocking information depends on the status of the FIFOs. Therefore, the scheduler and the FIFO flags are inputs for the local clock generator. The outputs are the different clocks and the select signal. The different clocks control the state update of each tasks separately. The select signal controls the multiplexer at the input of the logic.

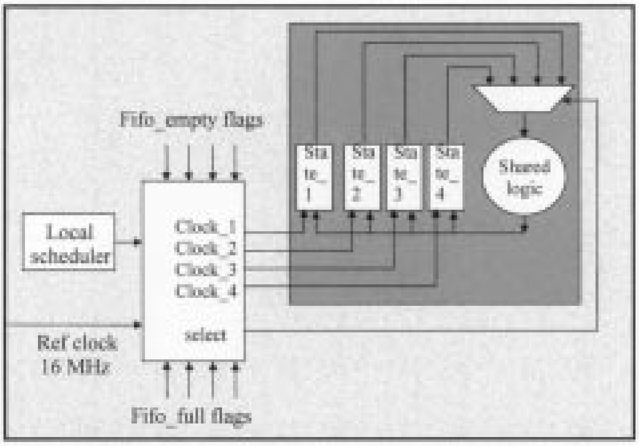

Fig. 8. Local clock generation.

As synchronization with the rest of the system is performed inside the shell, the processor is a pure stream processing implementation. Therefore, high-level synthesis tools like Phideo [5] can be used to design the processor. The shell adapts the periodic model of Phideo to the rest of the system which is much more dynamic.

The clock gating helps to reduce the power dissipation. Power dissipation is important for consumer products because heat sinks or fans must be avoided, and limited power dissipation allows use of a cheap package. Clock gating is also used to provide hardware breakpoints and single-step debug capabilities.

B. Clock Distribution

In order to deal with deep-submicron effects and routing delays the chip layout is organized in a hierarchical way using nine layout blocks and 30 different clock domains. At the top level, there is a relatively slow clock of 16 MHz which is used for synchronization between the different clock domains. Within each domain, local clocks of higher frequencies up to 96 MHz are used.

The clock circuitry in each layout block is built around a phase-locked loop (PLL) (see Fig. 9). The PLL compensates for the insertion delay due to the clock trees and the divider and gating logic. The PLL matches the phase of the output of the clock tree with the 16-MHz clock reference. The clock tree output is provided with a continuous running dummy clock tree also running at 16 MHz, as a PLL requires a closed control loop. The different clock trees originating from the same PLL must be matched according to timing constraints of communications paths between the domains. In our case, we chose to match the different clock trees in order to avoid additional components at clock boundaries.

C. Debug

With the increased complexity of systems on silicon debug becomes more and more important. Today it is key to the success of the product. The goal of debug is to find out why the chip does not work in its application environment, in our case a set-top box or a television set. This analysis must be done as fast as possible,

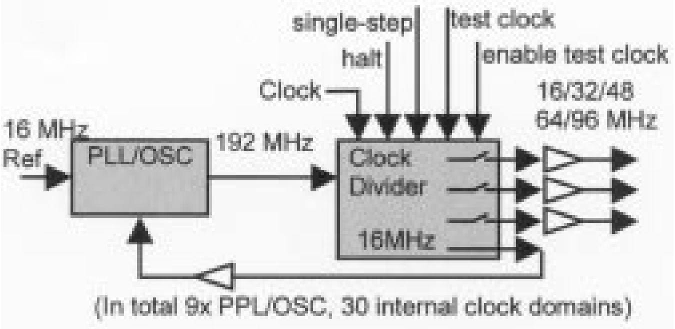

Fig. 9. Clock distribution per layout block.

so that the system, software, or chip can be redesigned quickly. This way the time from the manufacturing of the first samples (‘first silicon’) to a fully functional system incorporating the IC can be improved.

The chip can malfunction in its application for a variety of reasons: the board may contain errors, or the chip itself may be buggy. The chip may fail for various reasons related to the software as well as to the hardware. The software can contain bugs or the hardware can have design errors such as logic errors or timing errors that may have slipped through pre-silicon verification. Another possibility is that samples may have manufacturing errors that were not found using the stuck-at and IDDQ tests.

In the debug approach, two components are used: on-chip design-for-debug hardware (DFD), and debugger tool software that executes on a workstation. The DFD is added at design time. The debugger software communicates with the DFD via a serial interface, as shown in Fig. 10. Simulator-like features are offered with the real silicon; *all* flip-flops and embedded RAMs can be accessed as in a Verilog simulation. Wavetraces of internal signals can be displayed as shown in the figure. Breakpoints can be set to stop the chip at appropriate points in time.

The following types of DFD were added to the chip:

1) a serial interface (JTAG/IEEE 1149.1 Test Access Port);
2) multiplexers that connect all scan chains to the serial interface;
3) a clock controller with halt, single step, and ‘enable test clock’ features;
4) breakpoint controllers that can be programmed to halt the clock controller upon detecting certain event combinations.

The JTAG port provides a 5-pin serial interface to the chip. All debug features have been made accessible via the JTAG port.

In order to provide access to all the flip-flops on the chip, the scan chains are reused. For manufacturing test, the scan chains are multiplexed over the functional pins to allow parallel access to multiple scan chains on the chip. For debugging, however, the ability to access all the flip-flops while the IC operates in the application outweighs the speed requirement. For this reason, all the flip-flops of the chip have been made accessible (on a per-clock domain basis), through the JTAG port.

Scan chain access only works if the functional clock is halted. For this reason, the local clock generator described in Section IV-B has halt and enable-test clock pins.

After scanning out the required clock domains, and scanning in the original state again, functional clock cycles can be issued using the single-step pin on the clock controller. The single-step pin is activated using commands that are issued via the JTAG port. The halt pin is activated by the breakpoint controller.

In order to stop the chip at a reproducible point in time, on-chip clock cycle counters or breakpoint controllers are required. For simple applications, stopping the chip n cycles after the reset is sufficient to allow debugging.

For complex applications with data dependent processing times, it is often impractical or impossible to calculate up front at what clock cycle a given event will occur. Similarly, it impossible to predict the behavior of a system if the timing of its input signals is not deterministic. To facilitate debugging, *breakpoint controllers* are added that monitor signals that give meaningful information about the progress of computations. Thus, not only clock cycles are counted, but also, e.g., the occurrence of addresses on the control bus and switch matrix packet positions can be monitored.

The debug hardware has been successfully used to read internal RAMs, create bus traces, and monitor flip-flops in processing units.

D. Design Flow

A major factor to be able to design a system on a chip with a limited number of people in a limited time is a working design flow. Design entry is performed at RTL level and at behavioral level. An in-house behavioral synthesis tool [5] specialized in stream processing functions is used for the video scaling and enhancement functions. The behavioral synthesis tool produces RTL code, by deriving a data path and schedule. It also performs memory assignment for intermediate data values. All RTL code can then be synthesized with logic synthesis tools.

These designs are created starting from the function and can be considered bottom up. Simultaneously, a top-level floor plan is created. Interconnectivity and IC infrastructure (clock, power, reset, production test, and debug) are designed to build a system from the different independent functions.

Functional verification is done first at processor level. All modes are tested with self-checking test benches. Simulations are bundled in a regression system as soon as they run without errors. At system level several complete system simulations must be created. They will test the processor interconnectivity and infrastructure. Most of the problems are related to incomplete or incorrect specification and to timing which was not covered at block level using a test bench. A simulation for production test patterns is set up, to test this part of the design and to prove that the production test hardware performs correctly. Finally, the memories are verified either via a built-in self-test circuit or via scan chain access.

At *layout level* the blocks can be created in parallel. A number of steps are done based on placement information. These include, scan chain reordering to reduce the wire length used by scan chains, in-place optimization to match buffering with actual wire load, and buffer tree synthesis for nets with a high fanout. The clock net, which is one of the nets with a high fanout, also requires matching of the different delay paths from root to

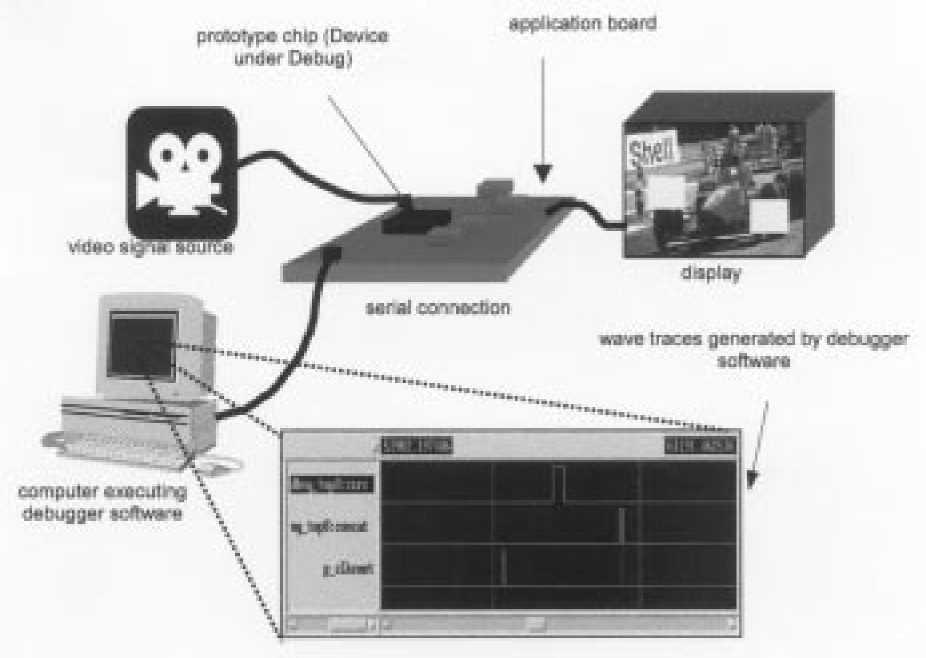
prototype chip (Device under Debug)
application board
video signal source
display
serial connection
wave traces generated by debugger software
computer executing debugger software

Fig. 10. Debug setup.

leaves. As the utilization is usually smaller than 90%, the empty locations can be filled with decoupling capacitor cells. They will reduce ground bounce and voltage loss in resistive power supply lines (IR-drop) in the design. Finally, routing will finish the creation of a block. A layout versus schematic comparison is capable of proving that the layout is equivalent with the desired circuit. Extraction tools are capable of estimating the parasitics from the design layout. With these, accurate timing analysis can be done for the extreme process cases. Under best-case conditions all paths are checked for hold times. Under worst-case conditions all path are checked for setup times. The next paragraph will discuss the very important aspect of timing closure in more detail.

The top-level assembles and interconnects all layout blocks. One important aspect to verify is the power distribution. Estimated power dissipation information can be used to assign power consumption to particular parts of the design. An extracted resistor model of the power grid is used to perform a static qualitative analysis on the power grid. IR-drop results produced with this type of verification are shown in Fig. 11.

It can be seen that the spot with the largest voltage drop is not perfectly in the middle of the chip. With the power routing in rings around blocks, this would be the ideal situation. Therefore, the power grid was adjusted such that the absolute value of the voltage drop was lowered and the largest voltage drop is shifted to the chip center.

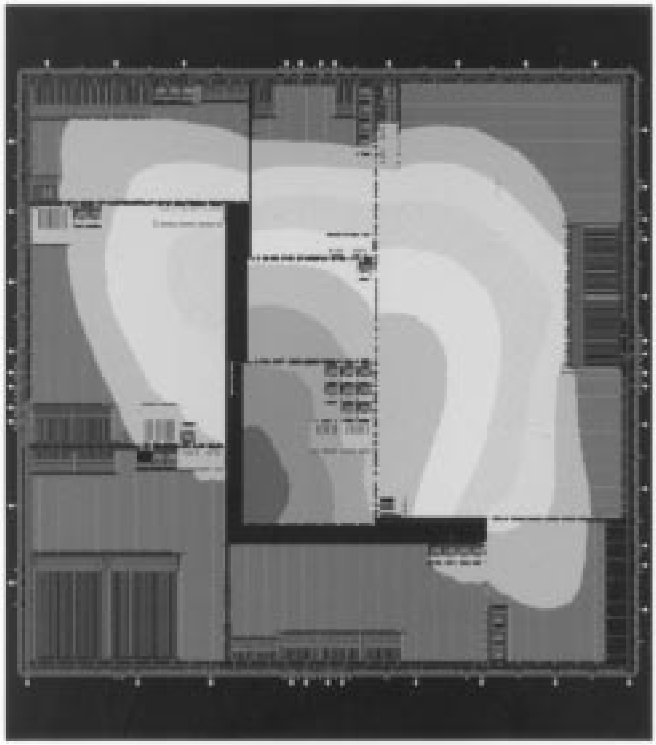

Fig. 11. IR-drop.

E. Timing Closure

Static timing analysis has an important role in the current synchronous ASIC design flow. There are a number of issues that have a large impact on the ability to prove that a design will work at speed. With designs getting bigger, a hierarchical approach for static timing analysis is required. This means that one can verify the internals of a block prior to having the complete design information. However, for an accurate result, the environment of the block must be accounted for (load of outputs, slope, and arrival time at the inputs). It must be possible as well to create a timing model for every block in order to perform a check at the higher level. This has an impact on hierarchy decisions. One requirement is for example the presence of all clocks

TABLE I
CHIP METRICS

Technology	0.35 μm
Clock frequency	Noise reduction = 16 MHz Sharpness enhancement = 32 MHz Horizontal zoom = 64 MHz Vertical zoom = 64 MHz Memory interface CPU interface = 48 MHz 30 internal clock domains
Power supply	3.3 Volt, 5 Watt
Transistor count	7.6 Million
Package	352 SBGA
Processing power	> 10 GOPS
Testability	Full scan, 15x macro test
Debug	Full internal state access via JTAG Hardware breakpoints
Die size	13 mm x 13 mm

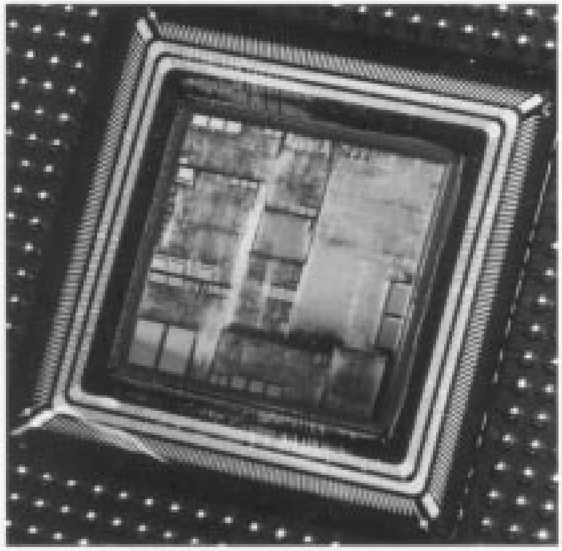

Fig. 12. Photo of the chip.

which relate to I/O as pins on the boundary of a block, in order to be able to create a timing model. As a result the clock distribution, gating and division logic should be at the highest hierarchy level.

V. CHIP METRICS

Table I provides measured metrics of working silicon for the multisynchronous system on a chip described in this paper. In total, 15 autonomous dedicated processors work in parallel. The 15 functional entities are regrouped into nine layout entities for which the backend trajectory is completed and which are combined at the chip top level. The chip is shown in Fig. 12.

VI. CONCLUSION

We have discussed the cost-effective design of silicon for challenging multiwindow TV applications. In contrast to PC windows, sizing can be done incrementally at run-time without artefacts. The chip is capable of managing up to 25 internal video streams with hard real-time constraints.

We have covered all relevant aspects ranging from the application domain analysis, the programming paradigm, the architecture and the physical implementation.

The application domain analysis resulted in the definition of a limited set of high-level functions which have to be combined in different flowgraphs that represent different applications. As a consequence, we have adopted Kahn data flow graphs as our programming paradigm.

To design the architecture, we decided to use a platform-based approach in which computation is separated from communication. The computation takes place in autonomously operating processors, each optimized for particular types of functions. The interfaces to the processor are standardised by defining processor shells.

The communication is implemented as a reconfigurable connection network. Use is made of circuit switching and a TST approach. Two configurations are stored to allow run-time dynamic reconfiguration. Worst-case processing performance is guaranteed. Self-scheduling data driven processing eliminates need for a cycle-accurate compile-time schedule. All this is necessary to avoid display artefacts.

Finally, we discussed the physical implementation in silicon. It was shown that the application-driven design style using high-level and RT-level synthesis leads to a set of challenges. More specifically, clock gating, clock distribution, debug, verification, and timing closure are discussed in detail. The design shows that different clock domains up to 96 MHz in a 0.35-μm process are possible using synthesis on a chip of 13 $\times$ 13 mm^2.

ACKNOWLEDGMENT

Many colleagues have contributed to the results presented in this paper, for which the authors are more than grateful. They would like to especially thank the people who have contributed to the realization of the silicon: E. Waterlander, F. Harmsze, A. Vaassen, L. Sevat, M. Oosterhuis, H. van Herten, E. Jaspers, J. Janssen, G. Essink, J. Leijten, and P. Wielage.

REFERENCES

[1] T. A. C. M. Claasen, "High speed: Not the only way to exploit the intrinsic computational power of silicon," in *ISSCC Dig. Tech. Papers*, 1999, pp. 22–25.

[2] E. A. Lee and T. M. Parks, "Dataflow process networks," *Proc. IEEE*, vol. 83, pp. 773–801, 1995.

[3] C. I. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. Assoc. Comput. Mach.*, vol. 20, no. 1, pp. 46–61, 1973.

[4] G. J. van Rootselaar and B. Vermeulen, "Silicon debug: Scan chains alone are not enough," in *Proc. IEEE Int. Test Conf.*, 1999, pp. 892–902.

[5] J. L. van Meerbergen, P. Lippens, W. Verhaegh, and A. van der Werf, "PHIDEO: High-level synthesis for high-throughput applications," *J. VLSI Signal Processing*, vol. 9, no. 1–2, pp. 89–104, Jan. 1995.

[6] A. H. Timmer, F. J. Harmsze, J. A. J. Leijten, M. T. J. Strik, and J. L. van Meerbergen, "Guaranteeing on- and off-chip communication in embedded systems," in *Proc. IEEE Computer Society Workshop VLSI '99*, Orlando, FL, pp. 93–98.

[7] S. Hosseini-Khayat and A. Bovopoulos, "A simple and efficient bus management scheme that supports continuous streams," *ACM Trans. Computer Systems*, vol. 13, no. 2, pp. 112–140, 1995.

[8] F. J. Harmsze, A. H. Timmer, and J. L. van Meerbergen, "Memory arbitration and cache management in stream-based systems," in *Proc. DATE 2000*, Paris, France, Mar. 2000, pp. 257–262.

[9] J. A. J. Leijten, J. L. van Meerbergen, A. H. Timmer, and J. A. G. Jess, “Stream communication between real-time tasks in a high-performance multiprocessor,” in *Proc. DATE 1998*, Paris, France, Mar. 1998, pp. 125–131.

[10] J. A. J. Leijten. (1998, Nov.) Real-time constrained reconfigurable communication between embedded processors, Ph. D. dissertation. Eindhoven Univ. Technol., Eindhoven, The Netherlands. [Online]. Available: http://www.ics.ele.tue.nl/es/papers/sld.shtml

Marino T. J. Strik (S’93–M’95) received the degree in electrical engineering from the Eindhoven University of Technology, Eindhoven, The Netherlands, in 1992. In 1994, he completed the post-graduation designers course of the Eindhoven University of Technology. These two years he worked on high-level synthesis and code generation for application domain specific processors.

He joined Philips Research, Eindhoven, in 1995 in the digital VLSI group. There he was involved in IC design methodology with a focus on digital system on a chip realization. Recently, he joined Philips Semiconductors to support the development of processor peripherals and interconnectivity IP where the focus is on design reuse methodology.

Adwin H. Timmer was born in Apeldoorn, The Netherlands, in 1966. He received the electrical engineering and the Ph.D. degrees from the Eindhoven University of Technology, Eindhoven, The Netherlands, in 1990 and 1996, respectively.

In 1995, he joined the Philips Research Laboratories, Eindhoven. In 1998, he was a Visiting IC Architect with the Philips Semiconductors WSG business line, Mountain View, CA. His current interests are in IC architectures for high-performance signal processing applications, system-level design methods, hardware/software co-design, and compilation techniques for embedded DSPs.

Jef L. van Meerbergen (M’87–SM’92) received the electrical engineering and the Ph.D. degrees from the Katholieke Universiteit Leuven, Belgium, in 1975 and 1980, respectively.

In 1979, he joined the Philips Research Laboratories, Eindhoven, The Netherlands. He was engaged in the design of MOS digital circuits, domain-specific processors, and general-purpose digital signal processors. In 1985, he started working on application-driven high-level synthesis. Initially, this work was targeted towards audio and telecom DSP applications. Later, the application domain shifted towards high-throughput applications. His current interests are in system-level design methods, heterogeneous multiprocessor systems, and reconfigurable architectures. He is the Associate Editor of *Design Automation for Embedded Systems*. He is a part-time Professor at the Eindhoven University of Technology, Eindhoven.

Dr. van Meerbergen is a Philips Research Fellow. His Phideo paper received the Best Paper Award at the 1997 ED&TC conference.

Gert-Jan van Rootselaar received the M.S. degree in electrical engineering from the Eindhoven University of Technology, Eindhoven, The Netherlands, in 1995.

Currently he is a Research Scientist at the Department of VLSI Design Automation and Test, Philips Research Laboratories, Eindhoven. He is responsible for the development of methods and tools for debugging first silicon of system-on-chip ICs.