MASTER

Comparison of communication architectures and mapping techniques for advanced MP-SoC platforms

Baert, R.

Award date:
2006

TU/e technische universiteit eindhoven

Faculty of Electrical Engineering
Section Design Technology For Electronic Systems (ICS/ES)
ICS-ES 879

Master's Thesis

Comparison of Communication Architectures and Mapping Techniquesfor Advanced MP-SoC Platforms

R. Baert

Supervisor: prof.dr.ir. J.L. van Meerbergen
Date: September 2006

Contents

Abstract

This report compares two multi-processor system-on-chip solutions which are currently under development: MPSoC by the DESICS division at IMEC, and Hijdra by Philips Research. It covers both the hardware architectures and the design flow of both systems, which are described and compared. A case study using a video encoding application shows the differences in a practical and realistic situation. Also possibilities for hybrid solutions are proposed, combining the strengths of both systems.

Acknowledgements

I would like to thank the members of the MP-SoC activity at IMEC for giving me the opportunity to work on my thesis within their group and for their support. I am especially grateful to my supervisors, Erik Brockmeyer and Théodore Marescaux.

This thesis would not have been possible without the cooperation of the people from the Hijdra project and ESAS group of Philips Research. Special thanks go to Marco Bekooij for providing the tools and information about Hijdra, and Martijn Coenen for his support for the Æthereal simulator.
Finally I would like to thank my TUE supervisor Prof. Jef van Meerbergen.

Chapter 1

Introduction

As consumer electronics tend to get more and more functionality combined in one device, the underlying platform should offer high performance, while still being flexible and cost efficient. The design of these embedded systems is based increasingly on utilizing multiple processors. However programming these systems raises new challenges for designers, such as how to partition an application into parts running on different processors, how these tasks should communicate and share central resources [Mar06].

The MPSoC activity within DESICS tries to address these challenges with the development of design tools and platform templates which help the designer in mapping an application, meeting platform constraints and performance requirements. The tools are based on code transformations and the DTSE methodology [CdGW98]. The Hijdra project at Philips Research also develops a platform template and design flow for the same target application domain, but base their design flow on data-flow theory [BMPP+04].

The goal of this thesis is to compare both approaches, to see what their strengths and weaknesses are. A unified architecture was designed and built and the application mapping case study was performed using both design flows. The case study uses QSDPCM as driver application.

Chapter 2 introduces the different platform templates variants are. Chapter 3 explains the application mapping design flows. Chapter 4 presents the results of the case study. Finally an unified, hybrid mapping is proposed, utilizing the best aspects of both IMEC/MPSoC and Hijdra.

1.1 Related work

The problem of mapping applications to multiprocessor systems has been the subject of many research projects.

Task scheduling and synchronization using dataflow techniques is described extensively in [SB00].

In [GNL01] the authors extend the traditional FIFO communication mechanisms to make it more flexible and scalable by instead of reading and writing tokens in

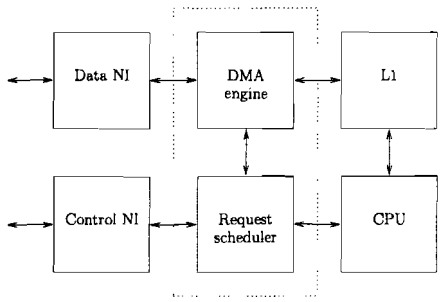


Figure 1.1: Distributed memory server (DMS)

a FIFO allowing a process to claim and release data or space in the FIFO. This means it allows a different granularity of synchronization and transportation, and avoids unnecessary copying of data in shared memory systems.

Besides the two communication assists which will be discussed in the remainder of this report, one other such device is found in literature [HBB+04]. It is called a distributed memory server (DMS) and is capable of performing simple block-transfers with different priority levels. A separated network is used for data and control, currently for both a bus is used. Internally it works with a single DMA engine and a controller.

A case study using MPEG-4 demonstrates the use of this DMS, however no tool support for helping the designer in the application mapping process is reported. Estimates for chip area are given for this case study, but no further performance results.

Chapter 2

Platform template

In this chapter an overview of the different platform templates and comparison is given. A platform template defines a set of constraints an architecture has to satisfy to allow the reuse of hardware and software components. A platform consists of a hardware architecture and a programming model with the accompanying runtime library. The resulting platform is an instantiation of the platform template, and can be both an input to the mapping process or result from it.

The architecture templates of Hijdra and IMEC/MPSoC are quite similar, therefore a single high level view can be given for both, see figure 2.1. The template defines tiles which are connected by an interconnect; the number of tiles can vary per instantiation. The tiles include a special hardware component to handle communication and synchronization, and special attention is given to these so called communication assists (CA). The other components in a tile are a CPU, a network interface (NI), a local memory (L1) and some kind of arbitration for the access to the L1. The IMEC/MPSoC platform template forsees tile with only memories, whereas the Hijdra template assumes a local processor is needed to produce and comsume data.

Both platform templates foresee a Network-on-Chip (NoC) as interconnect between the tiles of a multiprocessor system-on-chip. Advantages of a NoC over a traditional bus include better scaling of performance and design effort [BM06]. The first is achieved because a NoC consists of point-to-point wires (links) in between routers and network interfaces. Therefore the bandwidth is not shared globally but only per shared link and connections can be pipelined. Unlike a shared bus, the network as a whole does not suffer from added parasitic capacitances when more IP's are connected, as more network resources are added. The fact that the arbitration is distributed over the routers means that the delay for arbitration per hop is not affected by the size of the network. However, the latency of arbitration is dependent on the number of hops, and incurred at every hop. Techniques such as increasing the flit size are used to hide the arbitration latency
Also identical routers can be used, which enables IP reuse. However there can be also disadvantages, such as increased latency due to internal network congestion and, for larger networks, the multiple hops a data packet has to take before

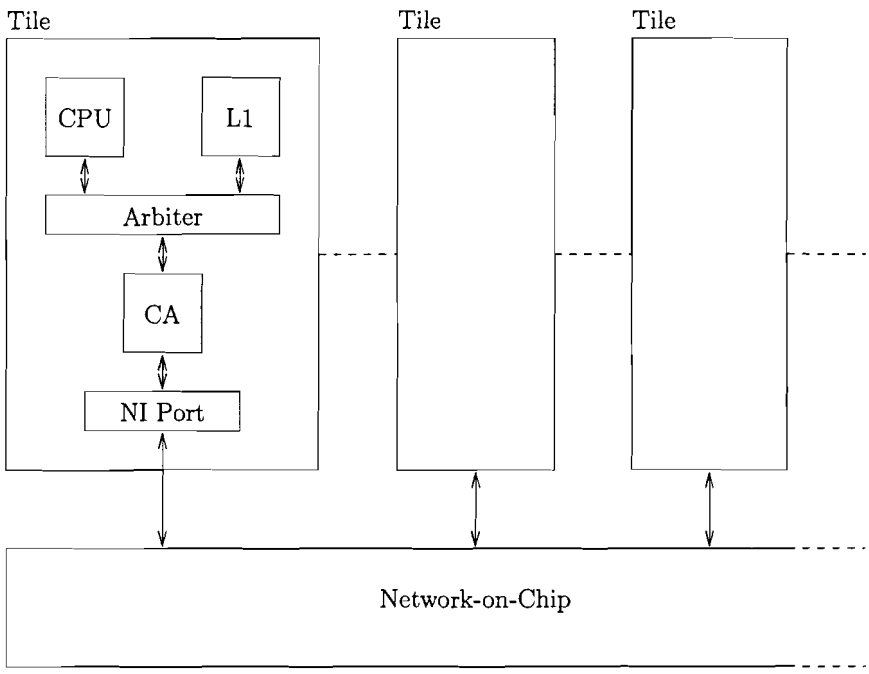


Figure 2.1: High level architecture template

arriving at its destination. Also there is a need to adapt interfaces of existing IP blocks to translate the bus-based protocols to the native NoC protocol. The interface of the AEthereal network [GDR05], which is used in both templates, is through FIFO's in the NI's. The network will transport data from the source NI to the destination NI according to connections specified at design time.

In the next section the IMEC/MPSoC platform template is introduced, followed by the Hijdra platform template in section 2.2. The last section of this chapter summarizes the key differences between them.

To limit the scope of the thesis, the runtime component of both platforms is not considered. These runtime functions include task scheduling, interrupt handling and resource management, and are especially useful when multiple applications are running simultaneously and/or the availability of resources is changing. It would also be difficult to port these functions to a common processor, since it contains a considerable amount of processor specific code.

For simplification L1 memory access conflicts are not considered, and thus not modeled by the simulator. It is possible to solve this issue, either by using a hardware arbiter, or to use a banked memory and avoid monopolization of a bank by choosing a suitable memory mapping.

2.1 IMEC/MPSoC platform template

The platform template as proposed by the IMEC/MPSoC project consists of two kinds of tiles connected via an interconnect, namely processor tiles and memory tiles. A processor tile consists of a local memory (L1), a communication assist (CA) and a processor, for example a TI C62 family DSP or an ARM, connected through a local bus. In its simplest form, a memory tile is composed of just a memory, a CA and a NI. Typically the amount memory in a memory tile is much larger than that in a processor tile and it is used as L2 memory. In practice this could also mean that it is produced using a different memory technology than the L1 memory, for example embedded DRAM instead of SRAM.

2.1.1 Communication assist

The communications assist (CA) is a hardware component that manages the transfers of data between different memories in a system. The CA performs transfers independently of the processor, such that communication and computation happen in parallel. The goal of this is to hide the latencies of the communication caused by the interconnect. The CA in the IMEC/MPSoC platform can be seen as an advanced distributed DMA controller. Distributed means in this context that the CA's at both ends of the connection are working together to execute a block-transfer, using a communication protocol on top of the network protocol. Block-transfers (BT) are transfers of a data-structure, or a part of data-structure, e.g. a macro-block of a larger image. Source and destination data-structure of a block-transfer may have different size and layout and are also independent of the block-transfer size.

The following phases can be distinguished in the execution of a block-transfer, see also figure 2.3:

```
BT_ISSUE(1,...); /* size=6, low-priority */
...
for(i=0; i<4; i++)
{
  ...
  BT_ISSUE(2,...); /* size=2, high priority */
  ...
  BT_SYNC(2);
  ...
}
...
BT_SYNC(1);
```

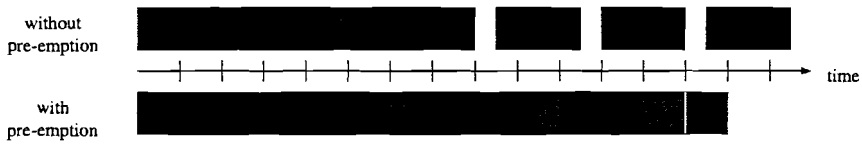


Figure 2.2: BT pre-emption example. The high-priority blue BT will interrupt the red BT when the red BT finished a chunk. When the blue BT is finished, the transfer of the red BT can continue. Transfer of the red BT fills the gaps between the blue BT, making better use of the channel capacity.

1. the transfer is issued by a processor by writing the transfer parameters to a control register; this operation is non-blocking
2. the local CA then interprets these parameters and translates them into a transfer descriptor, which is sent to the remote CA
3. the remote CA can get all information about where to read or write the data from the transfer descriptor
4. now the actual data transfer can take place, addresses are generated locally by the initiating and target CA
5. the remote CA sends back an acknowledgment in case of a write transfer
6. the processor that issued the block-transfer can check whether it has finished by calling a sync-function

Additional features of the CA include splitting up a large transfer into multiple smaller transfers (chunking) by pre-empting a transfer. This way two transfers on the same connection can be multiplexed, in which the priorities determine which transfer has access to the connection, as shown in figure 2.2. Each transfer is stored in its own context and the CA selects the context first based on priority, and within the same priority level in a round-robin fashion.

Two-dimensional transfers are also supported, which enables the transfer of parts of a larger two-dimensional data-structure, e.g. a still image or video frame, without losing the spatial relation between the different parts. For example two neighboring macro-blocks of an image can be transferred in two separate block-transfers, while in the destination memory they can form a single data-structure as if the two were transferred at once.

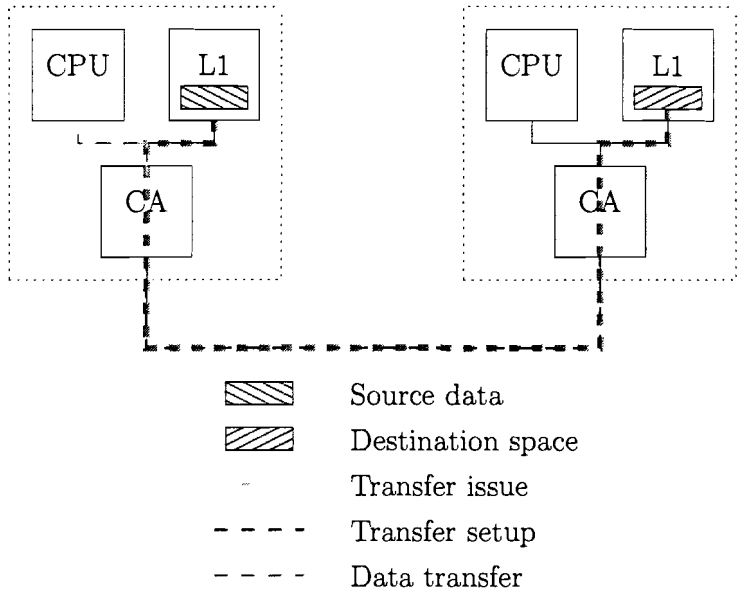


Figure 2.3: Data transfer between tiles

The CA can handle several transfer simultaneous by using a separate DMA engine for incoming and outgoing transfers, see figure 2.5. The processor which issued a transfer can later check if the transfer has finished, allowing it to do other processing in the meantime. An atomic write operation is provided for implementing synchronization mechanisms in shared memory.

Optionally modulo addressing can be enabled for block-transfers, which is best explained by the example in figure 2.4. Copy 2 crosses the bounds of the source data, and therefore "wraps around" to the rectangles indicated by the fine white dotted line, resulting in the copy shown.

The two-dimensional and modulo addressing functionality of the CA facilitate its use for managing large data-structures in a memory hierarchy. See section 3.1 for a description of how these features are used.

2.2 Hijdra platform template

The architecture template of the multiprocessor system proposed by the Hijdra project is shown in figure 2.1. It is designed to be predictable, such that it can guarantee a minimum throughput and maximum latency, which make it suitable for streaming, real-time applications. It features a communication assist (CA) to decouple computation from communication, which makes it possible to analyze the software running on the processors independently of the internal behavior of the interconnect.

Different tiles in the architecture can communicate with each other by sending tokens. A token is a container in which a fixed amount of data is stored, that is

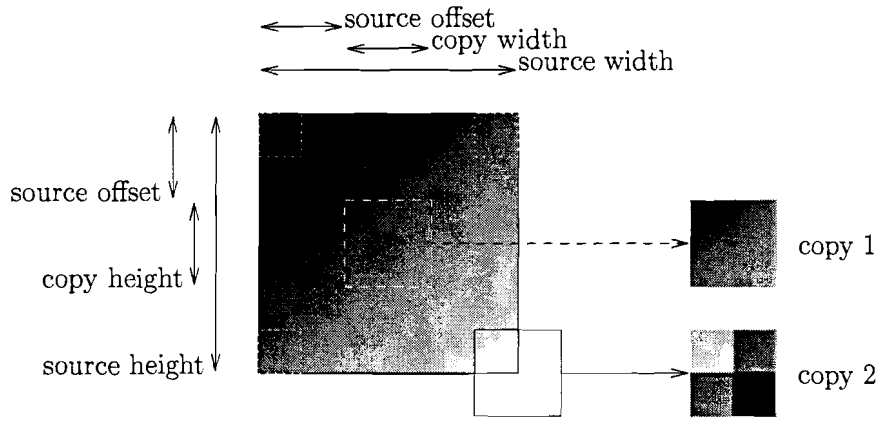


Figure 2.4: Modulo addressing illustrated on two-dimensional source data. The offsets, widths and heights of copy 1 are indicated. Copy 2 shown the effects of the modulo mode.

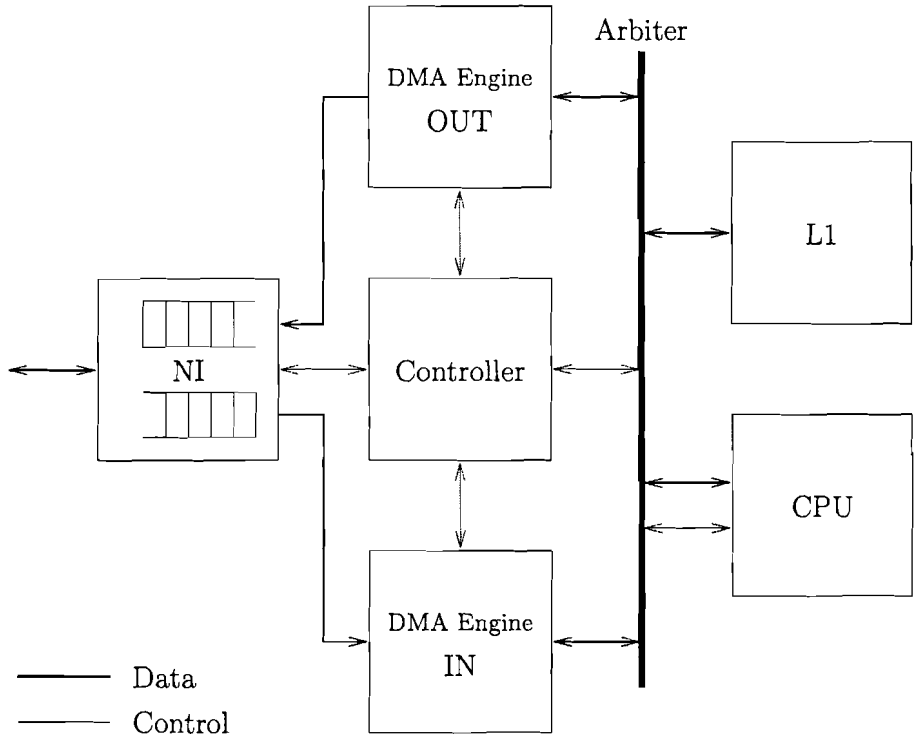


Figure 2.5: IMEC/MPSoC Communication Assist

the unit of communication on a particular channel and is produced or consumed in a single execution of an actor. The tokens are buffered in logical FIFO's, which are distributed over the L1 memories of the sending and the receiving tile. The basic function available to the application are the claiming and releasing of token data and space. This way input tokens can be read, output tokens written and the memory can be released when no longer needed by the processor.

The claiming of data and space is a blocking operation and it is the primary method of synchronizing actors. A consequence of this is that synchronization of actors and communication of tokens is coupled.

This programming model has several limitations. First of all, an actor only has access to claimed tokens and his state. However the order in which the tokens are received may not be the order in which they are processed. This can be solved by storing the tokens to which access is required in the state of the actor, but this may lead to high memory requirements for the L1. Related to this issue is the limitation that tokens are accessible during 1 firing of the actor only. If the token is to be re-used later, it should be explicitly copied to the state. Secondly, only a single producer and consumer per FIFO are allowed. If certain data is needed by multiple actors, it should be explicitly copied into multiple FIFO, again raising the requirements for L1 memory.

2.2.1 Communication assist

The Hijdra CA is a extended DMA controller, which handles transfers of multiple data streams between token FIFO's in local memory and corresponding network data FIFO's in the network interface, see figure 2.6. Since the CA only communicates with components inside a tile, there is no need for a protocol layer on top of the NoC, avoiding the overhead of such protocol[1].

The steps in the transfer of data from the producing processor tile to the consuming tile are as follows:

1. the producing tile writes a token of data into the logical FIFO in local data memory, and marks it as free to transfer by updating the FIFO registry (releasing data)

2. the CA notices that new data is available and, as soon as space is available, copies the data word by word to the corresponding network interface FIFO

3. the interconnect network transfers the data from the producing NI FIFO to the NI FIFO on the consuming tile

4. the CA in the consuming tile notices that new data is available and, as soon as space is available, copies the data word by word to the corresponding logical FIFO in local data memory

5. when the whole token is transfered, it is made available for reading to the consuming processor

[1]Assuming a connection oriented network with a one-to-one mapping of logical FIFO and NI FIFO, and direct access to the NI FIFO without the overhead of a NI shell.

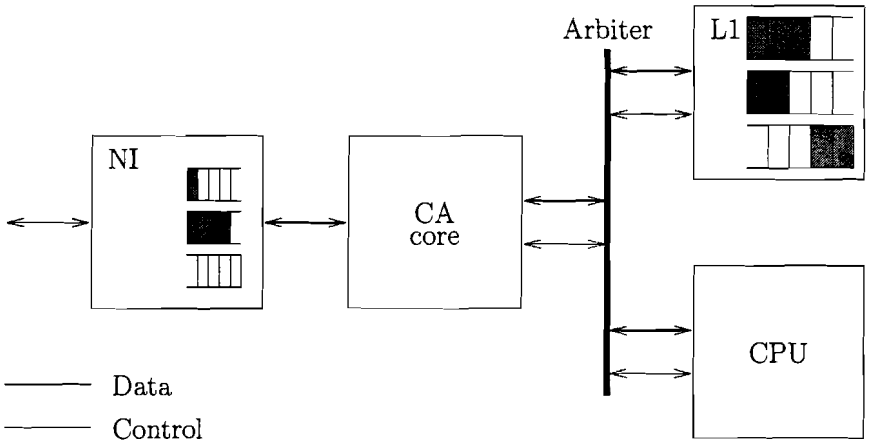


Figure 2.6: Hijdra Communication Assist

The CA acts as an autonomous component, not as a slave device of the CPU. It is only programmed once with parameters like FIFO start addresses, FIFO sizes and token sizes. After that it will independently initiate the outgoing transfers, receive incoming transfers and manage the pointers in the register array.

There exist several implementations of the CA, but in this report the implementation of [Wic05] is considered. It is implemented in VHDL, while others exist in SystemC or as high level SDF model [Moo04].

Figure 2.7 gives a more detailed view of the Hijdra CA. The arbiter is integrated into the CA, and arbitrates accesses to the local data memory. The arbitration policy was chosen to be Time Division Multiple Access (TDMA), which guarantees maxima for latency and minima for bandwidth and therefore keeps the system predictable.

The other major component is called the CA Core. It contains a Control Unit for channel administration and transfer scheduling and a Register Array which stores the pointers to the logical FIFO's. It is assisted by a dedicated Address Decoder and Pointer Handler.

2.3 Platform comparison

There are three main points in which the Hijdra and IMEC/MPSoC platform can be compared, namely the communication, the synchronization and the memory organization. These three points are discussed below. A fourth point could have been computation, but since both platforms decouple communication and computation, there are no notable differences on this point.

The Hijdra platform restricts itself to communication of tokens via FIFO's. This restriction ensures that the interface for the application is very clear and simple, while the complexity of the CA is also low. The IMEC/MPSoC platform on

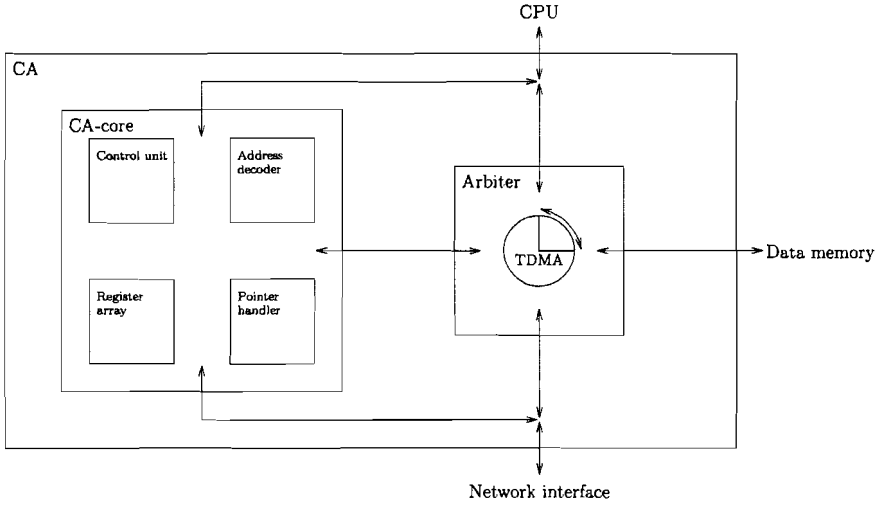


Figure 2.7: Detailed view of the Hijdra CA. For clarity, not all internal connections are shown

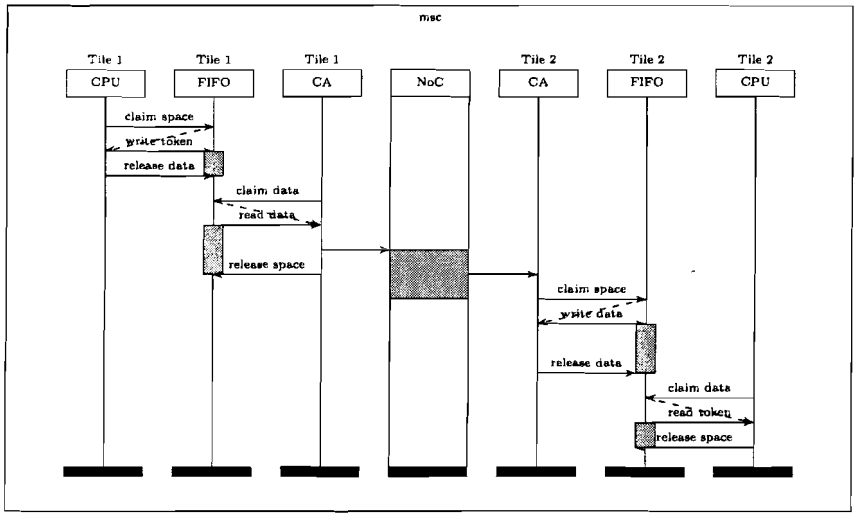


Figure 2.8: Hijdra communication example

the other hand employs so called block-transfers. These are highly configurable and must be configured for every transfer of data. This flexibility results in the potential efficient reuse of data and use of scratch-pad memory. There is however also a cost attached, namely the complexity of the CA and the application source code, which makes the use of design tools inevitable.

The flexibility of the IMEC/MPSoC platform is largely due to the fact that it provides an extra protocol layer on top of the AEthereal network. This protocol is used by the CA's to communicate with each other and provides the communication primitives to the application. The advantages of this extra layer is not only its flexibility and extensibility, but it makes large parts of the platform independent of the interconnect used. Disadvantages include its additional complexity and the additional latency of communication. The Hijdra CA operates very closely with the AEthereal network interface, which makes it more difficult to switch to other types of interconnect, e.g. connection-less networks. Also the configuration of the CA and NI are more tightly coupled.

The flexibility of the BT interface makes it possible to implement a large part of the Hijdra application interface with only minor modifications of the CA and communication protocol. One could even consider token communication as a subset of BT communication, with the following restrictions:

- tokens must have a fixed size, set at design time, while the size of a block-transfer can be set at every transfer
- tokens do not maintain spatial relationships, and are essentially one-dimensional, since their placement in memory is unpredictable. BT's can be placed in memory such that the blocks together form again a valid data-structure, e.g. a part of an image
- the communication of tokens is coupled with the synchronization, whereas BT's do not imply synchronization
- data can not be reused or shared, since only one connection is attached to a FIFO. There can be only one producer and consumer of a token, unless it is explicitly copied by an actor. The examples in figure 2.9 and 2.10 illustrate this.
- when using strictly SDF, communication can not be conditional and may result in the useless transfer of data

In appendix 6.1 this claim is demonstrated by giving an implementation of this extension of the IMEC/MPSoC CA, named Hybrid CA.

As mentioned before the granularity of synchronization of tasks running on different processors is coupled to the granularity of communication. Furthermore, in the given Hijdra platform implementation the transportation is also coupled to the communication, although this is not strictly necessary as [GNL01] shows. The granularity of synchronization is hereby also defined. On the IMEC/MPSoC platform the synchronization of threads is independent of that of transfers. Both can be chosen to be at a different granularity, which potentially can lead to lower synchronization overhead. This freedom makes the task of the designer or design tool more complicated however.

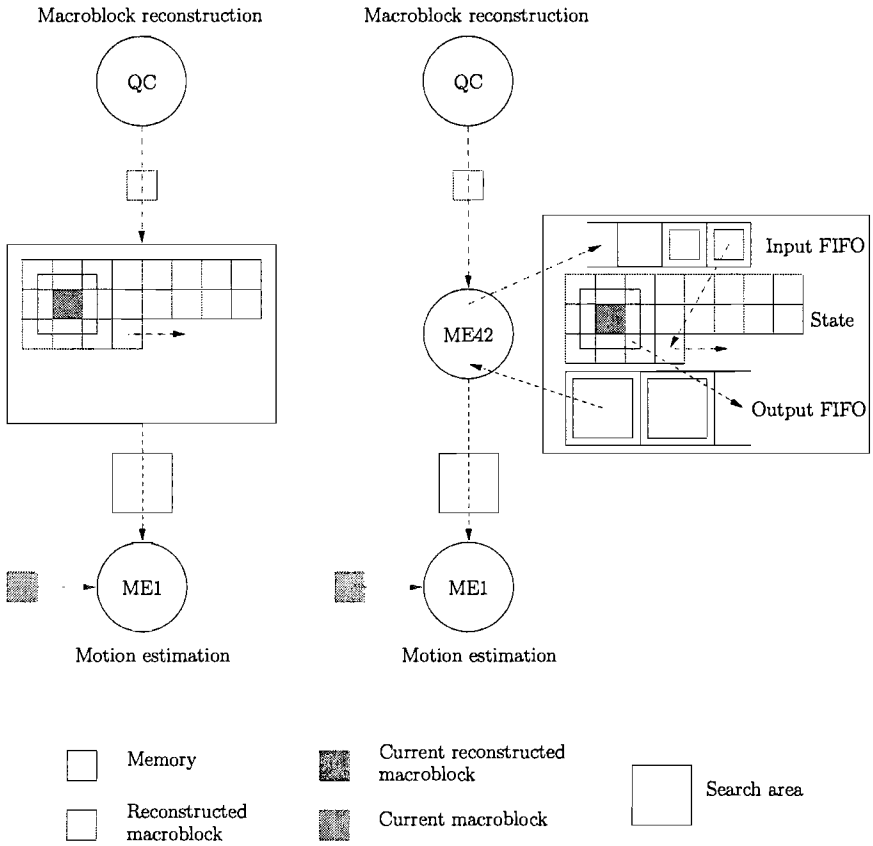


Figure 2.9: Example of how BT's enable efficient reuse of data. The left side shows how multiple threads can randomly access data-structure in shared memory. The right shows how this must be done when only FIFO communication is available. In this case the one actor has to manage the data-structure, and it can only be accessed via FIFO's, which results in three times more memory accesses. These additional accesses are reading from and writing to the input logical FIFO (red arrows), and output logical NI FIFO (red arrows).

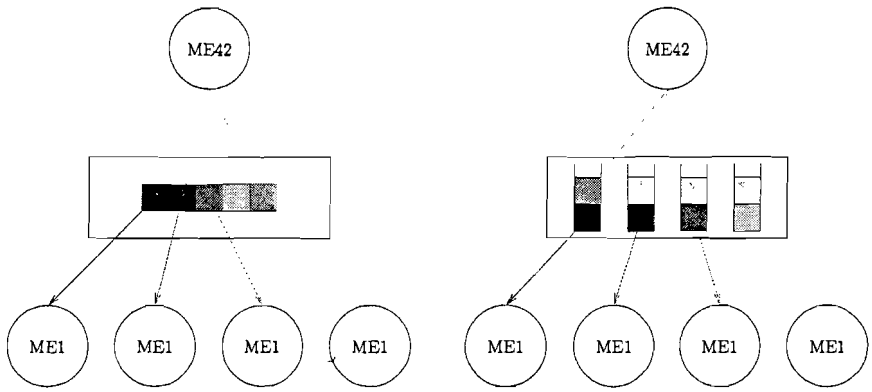


Figure 2.10: Example of how multiple consumers of a single data-structure in shared memory enable efficient reuse of memory. The left side shows how the same data structure can be accessed by multiple consumers via block transfers. The right side shows how the same function must be implemented when using only FIFO communication. Because the FIFO's must be alive all the time and only one consumer can read from a FIFO, more memory is required.

The memory organization is also a point of interest when comparing both platforms. The Hijdra platform as available for this exercise uses strictly private local memory, which can by design only be accessed by the processor and communication assist of that tile. It stores the FIFO's and actor state. Message passing is the matching type of communication for these kind of systems. The IMEC/MPSoC uses its local memory primarily as a scratch-pad memory which caches data of the shared L2 memory, taking full advantage of the different memory characteristics. All communication and synchronization is handled via this shared memory. The new implementation of the IMEC/MPSoC tile, as described in section A.1, for the AEthereal simulator now also allows processor-to-processor communication, which would make it a distributed shared memory system, but since support this type of communication is only recently added to the design tools, it is not used in the application case study (it is used in the Hybrid CA however). Of course, it is in principle possible to implement one kind of communication and memory usage on top of the other.

Chapter 3

Application mapping design flow

In this chapter the application mapping design flows for two MPSoC methodologies will be explained and compared.

The following terms are used in the context of application mapping [MPS06], see also figure 3.1:

application An application is a service that is as a whole to the interest of an end-user. Multiple applications can be run on an platform concurrently. Applications should not suffer from interference of each other. An application can consist of one or multiple modules and some functionality to bind them together. An example of an application is a video player, including video and audio decoding and the user interface.

module A module has a well defined function that can be used in various applications. Typically it is a demanding part of the whole application. A module consist out of one or multiple tightly interacting threads. The tight interaction enables efficient use of shared data with explicit synchronization. All threads that are part of a module start and stop simultaneously. An example of a module is a MPEG-2 decoder.

thread A thread is a schedule of kernels running on a single processor. The kernels are bound together by a "skeleton", which contains the intra-tile communication and synchronization. An example of a thread is the motion estimation of a part of an image.

kernel A kernel is a monolithic piece of code that executes without ever blocking internally. It does not have to wait for external events (for instance waiting for data). Therefore, the kernel is allowed to read and write data from within the processor tile only (e.g. local L1 memory), and does not have to synchronize with other parts of the application. An example of a kernel is the motion estimation of a macro block.

To limit the scope of the thesis, it is assumed that only a single module will run at any time on the platform, i.e. the application consists of one module. Also only a single thread per processor is allowed to avoid the need for a run time thread scheduler on the processor. This can easily achieved by statically

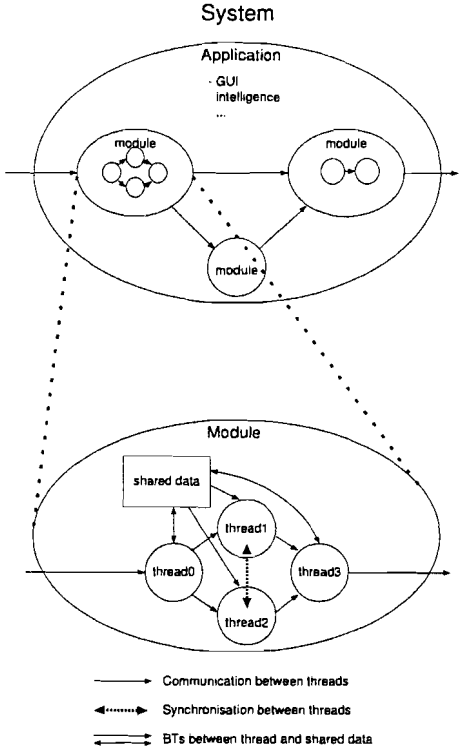


Figure 3.1: Terminology

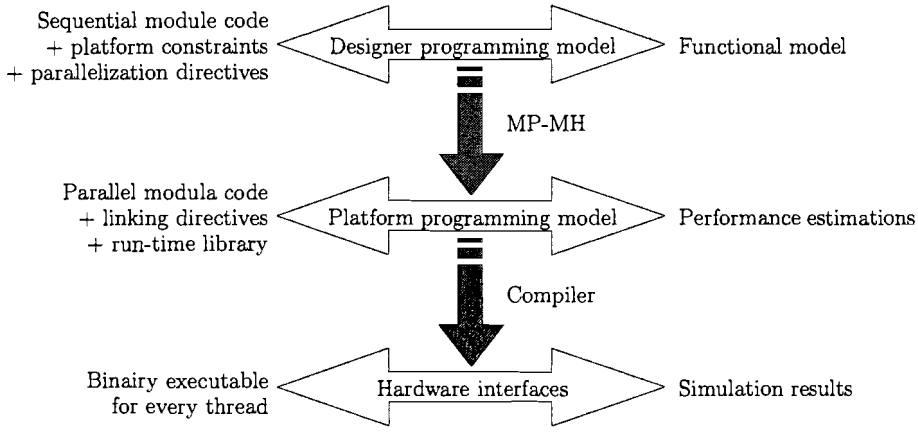


Figure 3.2: The IMEC/MPSoC flow makes a distinction between the designer and the platform programming model. The designer should stay at the highest level (sequential code), and the translation to the platform model and hardware interfaces is done by tools. The left side lists the input to each level, the right side the output.

scheduling the threads assigned to a single processor and then merging them into one.

The kernels, which are the most computational intensive parts of an application, are for both design flows equally optimized. This step are indicated as "performance optimizations" in figure 3.3 and 3.4. These optimizations include exploiting the available instruction level parallelism though software pipelining, which are very useful for a VLIW processor such as the used TI C62.

An other common step is the "platform independent optimization". A major goal of these optimizations is to increase the locality of memory accesses and possibly reduce them by techniques such as loop-transformations.

The next section describes the application mapping process as proposed by the IMEC/MP-SoC project. Section 3.2 explains the different steps of application mapping with the design flow of Hijdra. Finally in section 3.3 the strengths and weaknesses of both design flows are summarized and compared.

3.1 IMEC/MPSoC design flow

The IMEC/MPSoC application mapping design flow originates from the Data Transfer and Storage Exploration (DTSE) methodology [CdGW98]. It provides tools to help the designer explore and evaluate possible solutions, especially concerning memory usage. The IMEC/MPSoC design flow makes a distinction between the designer programming model and the platform programming model, see figure 3.2.

A key characteristic of the design flow is that it takes C source code as input and generates C code with linking directives as output. The linking directives

contain information for the compiler specifying the placement fo data-structures in the memory hierarchy. Furthermore the transformed code contains data transfer (block-transfer) functions. This combination enable the efficient use of a memory hierarchy and allow hiding the latency of memory accesses by prefetching data.

The resulting source code can in principle be compiled for any target architecture, however calls to specific API functions, such as the block-transfer functions, are emulated in software when not available in the target architecture. The concurrent execution on multiple processors can be emulated in software with simulation kernels like Tipsy [Coc] or SystemC, in which case timing information can be annotated manually. This method gives functional verification and timing estimates, and can be further refined by adding instruction set simulators and interconnect simulators, such as the combination of the TI C62 ISS with AEthereal.

The steps of the design flow are as follows (see figure 3.3):

1. platform independent code optimization
2. single processor memory assignment & transfer scheduling
3. parallelization, multi processor memory assignment & transfer scheduling
4. merge performance optimized code, compile & link

The single processor memory re-use analysis, memory assignment and block-transfer scheduling is automatically performed by the Memory Hierarchy (MH) tool [BMCC03] of the Atomium tool suite. MH takes the application source code and architecture constraints, and first performs reuse and lifetime analysis for all arrays in the application. This results in a collection of copy candidates (CC's), opportunities for making data reuse copies. Based on estimations for energy consumption, execution and communication time, and required memory, MH will select the optimal combination of CC's for a given platform model. These copies and the original arrays can now all be assigned to a memory layer, and the transfers of data between the memory layers can be scheduled. MH tries to schedule these BT's in parallel with computation, such that the latency of the transfer is hidden.

The output for this tool can be used for a single processor platform, but it provides also information to the designer to help decide the parallelization directives for the Multi Processor MHLA (MP-MHLA) tool [IBDD06]. MP-MHLA uses these user-supplied directives for automatically parallelizing the application. This way the user can work on the sequential code only and can do the parallelization exploration easily. The MH and MP-MHLA tools work on code which is not yet optimized for the specific platform because it is then easier to analyze, the platform specific optimizations are merged at a later stage.

The whole flow is schematically shown in figure 3.3. The red arrows mean that information about the application is passed on from one phase to an other. Blue arrows mean that the transformed source code is passed on. The result of the whole flow is transformed and optimized source code and linking directives, which can be compiled for an ISS, a single- or multi-processor platform.

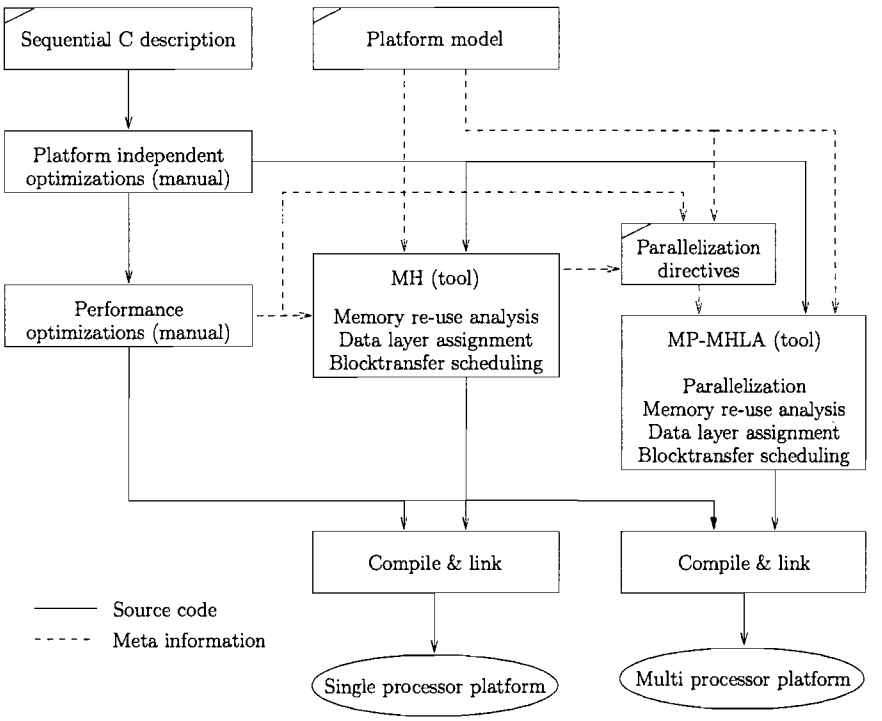


Figure 3.3: IMEC/MPSoC application mapping design flow

3.2 Hijdra design flow

The Hijdra project defines actors, tasks, jobs and applications, which in terms of granularity correspond respectively to the previously defined kernel, thread, module and application. A job consists of interdependent tasks and all jobs together form the application. All tasks that belong to a single job start and stop simultaneously. As mentioned in the introduction of this chapter, only a single task is assumed, so the application consists of one job only. Therefore no scheduling of jobs is needed.

The input to the mapping process, as depicted in figure 3.4, is a sequential C description of a job [Par05]. This description is manually rewritten in another description in which the task-level parallelism is made explicit. These tasks need to be written using the semantics of an actor. The ensemble of rewritten actors form a data-flow graph (DFG). This means they communicate through FIFO's containing fixed-sized tokens, start only after there are sufficient tokens on all inputs and sufficient space on the outputs, and have a bounded execution time. A single execution, called firing, can not be interrupted or stalled and therefore the availability of the required input data and output space should be checked beforehand.

Now the resulting process network can be simulated using the Hapi framework [Moo04]. Hapi provides an interface to construct the process network, define FIFO's with certain size and methods to read and write from them. Using Hapi the process network can be functionally verified. Also actor execution time information acquired from profiling can be added, so the simulation will give estimates of the timing behavior of the application. Furthermore a model of a communication channel can be added, in this case a guaranteed throughput channel of an AEthereal network (figure 3.5). Now the effects of different FIFO sizes and certain network parameters can be explored. The Hapi simulation does not take into account the mapping of tasks to processors and assumes worst case behavior of the network channel.

With the information gathered from the Hapi simulations a decision must be made about how to map the tasks to processors and whether there are tasks that should be duplicated to improve throughput, i.e. using data-level parallelism.

Now that the layout of the process network is fixed, final verification can take place using a cycle accurate simulator, consisting of the AEthereal network-on-chip simulator, instruction set simulators that run the task code and hardware components for the FIFO management.

The need to rewrite the application to a data flow graph has several consequences. The data-structures of the original application must be partitioned into tokens. If certain data-structures, or parts of them, are needed at more than one actor, thay need to be explicitly copied. All this means that the application can not be incrementally adapted to the design flow, but must be rewritten at once, making it a challenging and error-prone process.

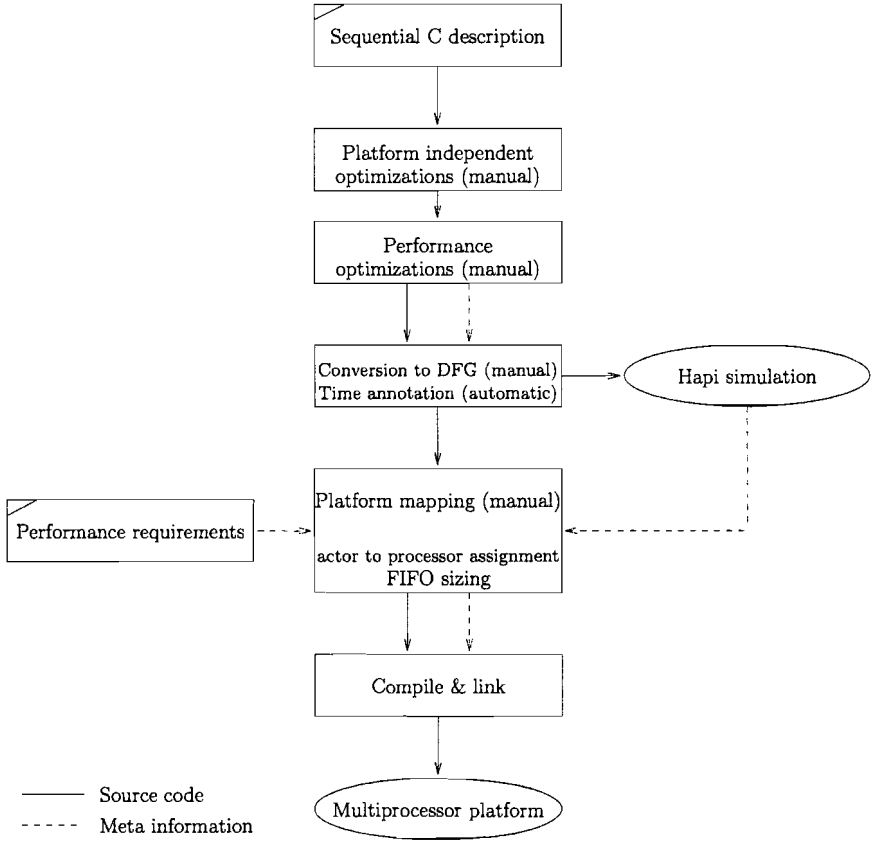


Figure 3.4: Hijdra application design flow. Tools exist for the "platform mapping" step, but they were not available.

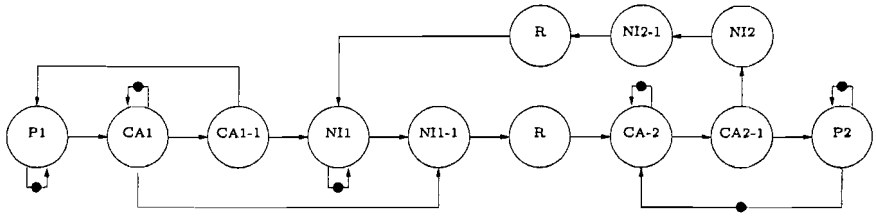


Figure 3.5: Hapi SDF model of a GT channel of the AEthereal NoC [Moo04]

3.3 Design flow comparison

Hijdra starts from the multiprocessor viewpoint, i.e. an explicitly parallelized application, but offers no structural solution yet for efficient memory management by exploiting a hierarchical memory system. However the architecture can in principle support a form of memory hierarchy, a tool to add the data transfers between the memory layers does not exist. A tool like the MH tool from IMEC could be used for this in a intermediate step in the design flow, before the platform mapping. The Hijdra flow takes a performance constraint as input and uses as much memory, i.e. FIFO space, as required to meet this constraint.

IMEC/MPSoC starts from single threaded application with memory hierarchy management, but supports multiprocessing only via distributed shared memory. It takes the L1 memory size as an input to the flow, and attempts to find a mapping solution that is as performant as possible within the constraints of MH. Processor-to-processor communication can in principle be done, but it uses the complex BT interface to do so. This also means that the addressing in the local memory of the receiving tile is done by the sending tile, which complicates the memory mapping. Only static scheduling within applications at design-time is currently used, which could lead to over-synchronizing because synchronization granularity is coupled to copy-candidates in the tools. Because copy candidates are coupled to the loop structure in the application, loop-tiling is the preferred way to achieve data-level parallelism.

Hijdra always synchronizes on every token; data-level splits are achieved by distributing the token stream evenly over a number of identical threads. This will in general lead to a very balanced split, but it could require that shared common data, such as input parameters, have to be duplicated and distributed as well.

Similarities between flows are the cycle profiling of the tasks to guide parallelization decisions. Therefore both flows will often lead to similar parallelizations, although the code might be modified during the mapping process which could influence the profiling results. This means also that typically the same level of throughput performance is achievable, although the cost in terma of memory, communication and processing may vary.

Both design flows benefit from the increased data locality and a regular control flow achieved by loop transformations. In the IMEC/MPSoC flow this leads to more reuse of arrays, better copy-candidates and eventually lower memory requirements. For the Hijdra flow it results in smaller tokens and lower need for duplication of data.

The Hijdra design flow ensures predictability by using the theory of SDF to statically analyze the application. The IMEC/MPSoC flow achieves it by using static-order scheduling only.

Chapter 4

Application mapping case study

The goal of the case-study is to verify the expectations from the platform and design flow analysis presented in the previous chapters using a real life application. The QSDPCM video encoder application is used for this purpose, which is available in sequential C source code.

The target performance is 30 frames per second at VGA resolution. Based on the profiling of the sequential, kernel optimized code shows that 6 processors are required. The goal is to achieve the target performance with a minimum required L1 memory and a minimum processor load, i.e. only the usefull load, however these requirements should be well balanced over all processor tiles.

The results of this case study are thus measured in terms of memory requirements, both L1 and L2, and of processor load. Also the scalability of the mapping solution, the design effort and energy consumption estimate are desired results.

During the case study it is assumed that communication is not a performance bottleneck, so that the focus can be on computation. This assumption is reasonable since the bandwidth interconnect used can adjusted to make the assumption hold. For simplification, instruction memory requirements are not considered.

The mapping using the IMEC/MPSoC design flow was not done starting from scratch, because a mapped version already existed [Bro05]. This version has been modified where necessary to match the new platform simulator. A similar mapping exercise has also been carried out using a 3D reconstruction algorithm [Bae05], however this application was not considered suitable for this thesis.

4.1 QSDPCM video encoder

The quad-tree structured differential pulse code modulation (QSDPCM) algorithm is an inter-frame compression technique for video images [Str88][Ost04]. It consists of a hierarchical motion estimation, quad-tree quantization, variable length encoding and image reconstruction. This algorithm was chosen because of its availability, its regular algorithm structure and moderate complexity.

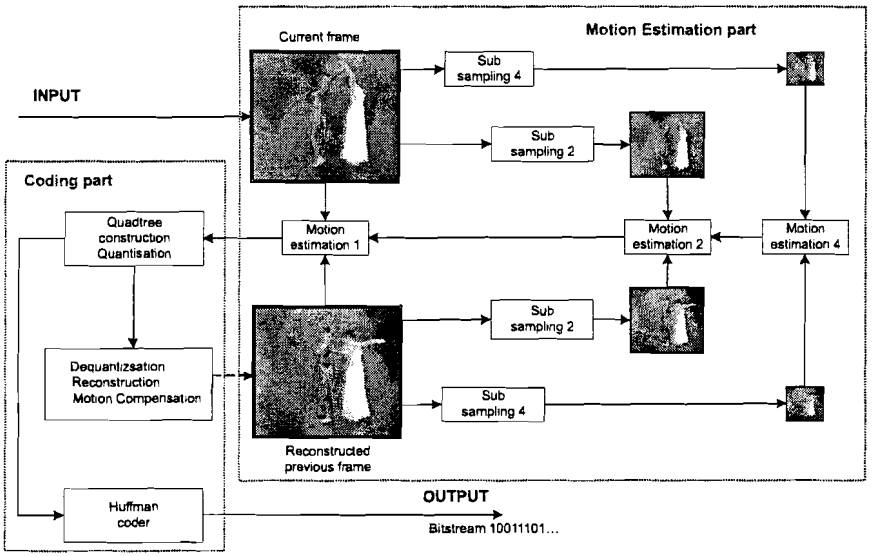


Figure 4.1: Overview of the QSDPCM algorithm

The hierarchical motion estimation consists of 3 steps: first at quarter resolution (ME4), then half resolution (ME2) and finally at full resolution (ME1). It determines the motion vector with which the difference between the current and reconstructed previous image can be encoded using the shortest code word. For this purpose the input image is sub-sampled by factor four (SS4) and two (SS2).

Quad-tree quantization works by recursively splitting the motion compensated predicted error frame from the ME1 step into four quadrants until, within a certain threshold, such a quadrant can be approximated by its quantized mean value, see figure 4.2.

The image reconstruction decodes the frame by dequantization and motion compensation.

4.2 Application analysis

To get an idea of the issues that can be expected during the mapping process, the application is first analyzed. The computation time of the different functions can be measured by compiling and running the performance optimized application on an instruction set simulator (ISS). The results are displayed in figure 4.3. From these measurements it is clear that the ME1 task is by far the most demanding. It is about 4 times slower than the other motion estimation tasks or the complete quantization and coding together.

Analysis of the code shows several possibilities for functional parallelism in the algorithm. The profiling of the application shows where data parallelism could be beneficial. All kernels can be executed in parallel by constructing a

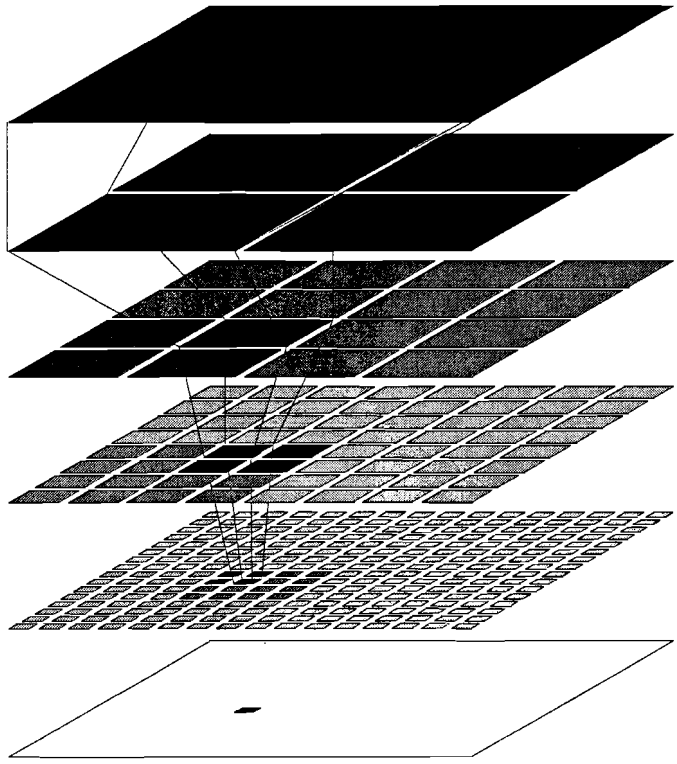


Figure 4.2: Quad-tree construction and quantization (source:[Ost04])

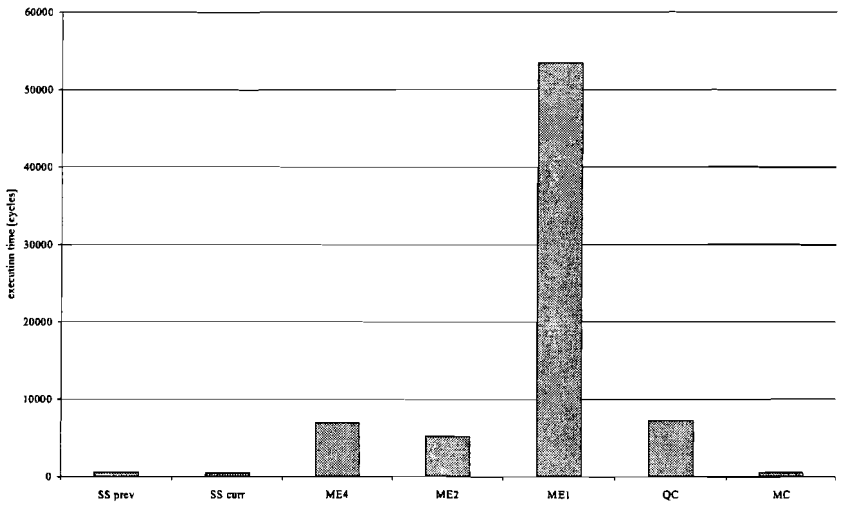


Figure 4.3: QSDPCM kernel execution times for QCIF resolution

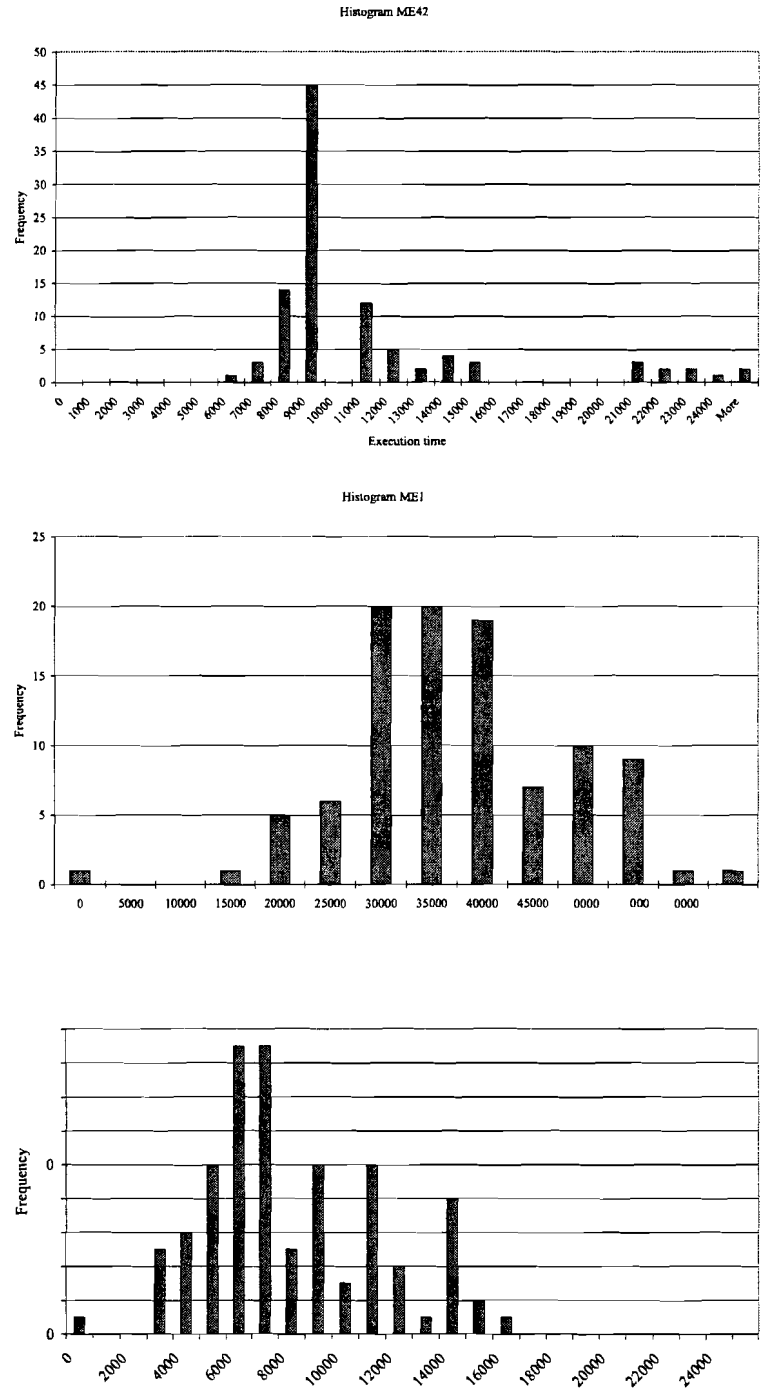


Figure 4.4: QSDPCM kernel execution time histograms for QCIF resolution

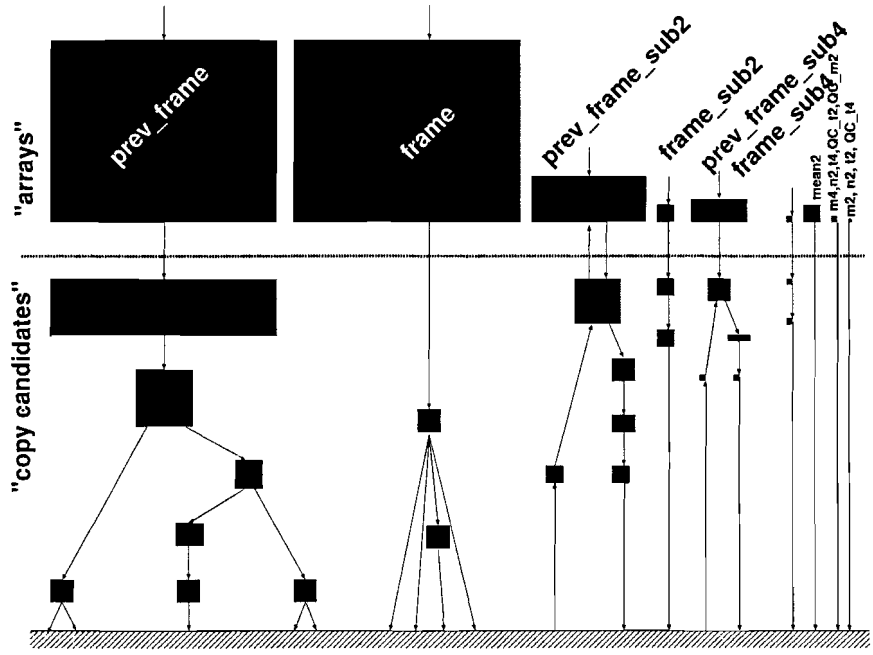


Figure 4.5: Simplified data reuse-tree of the QSDPCM application

functional pipeline with each kernel in a different pipeline stage. However this would not lead to balanced execution times for all pipeline stages. Therefore multiple kernels can be grouped into one pipeline stage. But even then the pipeline stages will not be well balanced, because the ME1 kernel will always be dominant in execution time. Since the ME1 kernel can not be efficiently decomposed into smaller kernels, another solution should be found, for example distributing the work load of the kernel over multiple processors by using data level parallelism.

Further analysis of the timing behavior of the three pipeline stages shows that they have significant variation. The execution time varies with up to a factor of three, as the histograms in figure 4.4 show.

4.3 IMEC/MPSoC mapping

In the IMEC/MPSoC application mapping design flow efficient memory use is an important concern. Therefore not only the execution time information is gathered and analyzed, but also the memory accesses. For this purpose ATOMIUM/Analysis is used, which provides a detailed report about which data-structures are accessed from which place in the application and the frequency of the accesses. Other tools from the ATOMIUM tool suite use this information to extract the reuse and lifetime of arrays.

After the memory access analysis, the MH tool decides on the assignment of

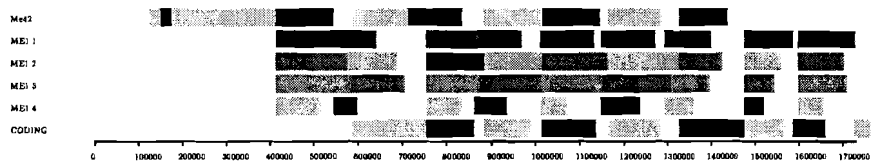


Figure 4.6: IMEC/MPSoC kernel execution schedule for QCIF resolution. On the vertical axis are the 6 processors, running respectively the ME42, four times ME1 and QC kernels.

data to the different memory layers and the scheduling of the data-transfers. This transfer of data is done using block-transfers, which actually makes a copy of the data, either the full data-structure or a part of it, from one memory layer to another. All the different opportunities are found by the Data Reuse (DR) analysis, and can be structured as a tree (figure 4.5). A trade-off must be made to choose the right size of the copy The overhead of the block transfer header is lower when choosing large copies, but they require also more space in L1.

Although not necessary, in this first attempt the threads within the application are statically scheduled by using barrier synchronization at the Y-loop level, which iterates over the macro-block stripes of the image. The resulting schedule is shown in figure 4.6. The ME42 thread on processor 1 has to subsample at least 2 lines of macro-blocks before it can start with the motion estimation. The ME1 threads can only start after a whole line of macro-blocks is processed by ME42, but then all ME1 threads can start at the same time. The QC thread on processor 6 can only start when the ME1 threads have finished. The result of the chosen scheduling granularity is thus a long pre-amble.

From the schedule it is also clear the for QCIF resolution, the balancing of the ME threads is not very good. The last one, on processor 5, has only two macro-blocks per stripe to process, while the others process 3. This results in a bad utilization of processor 5.

4.3.1 Platform instantiation

The platform instantiation that results from the mapping process is shown in figure 4.7. Two processor tiles share a network interface, but the memory tile has its own network interface because it will need most bandwidth. A star topology with a single router is chosen because it can handle the required bandwidth easily. In this case using more routers increases the communication latency, energy consumption and area requirements.

The connections that are used are all Guaranteed Throughput (GT), and are all assigned a single time-slot out of the eight available in the configured time-slot wheel.

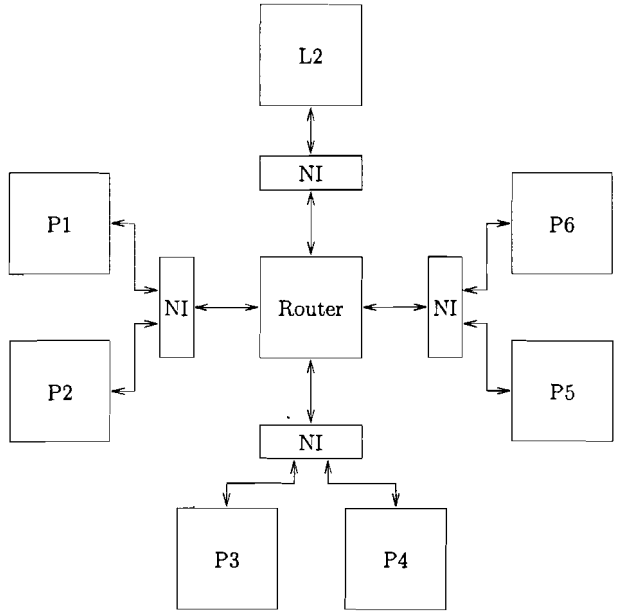


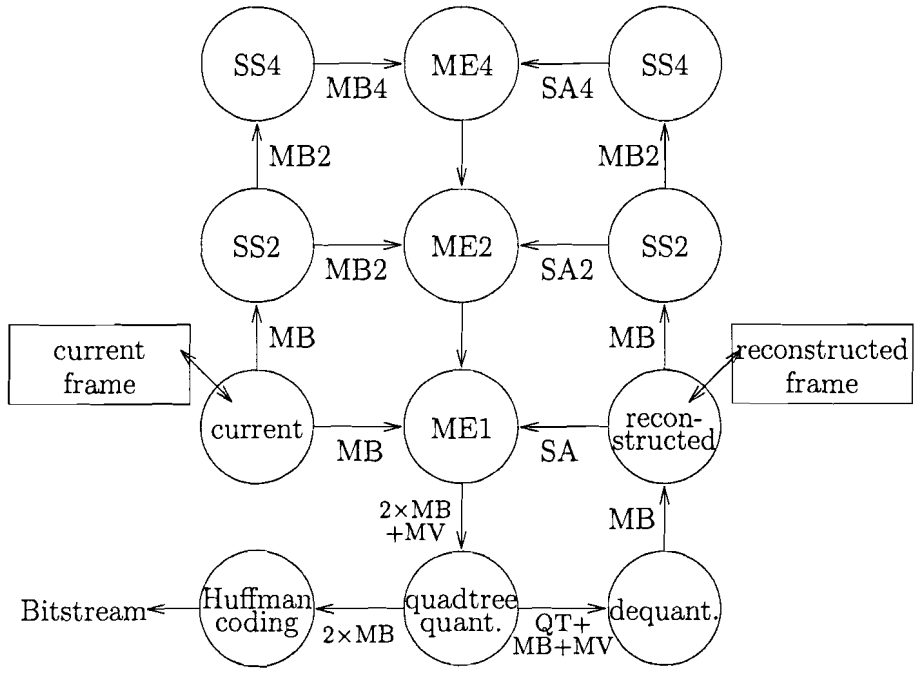
Figure 4.7: Platform instantiation for the IMEC/MPSoC mapping

4.4 Hijdra mapping

Figure 4.8 shows the QSDPCM algorithm after its conversion to DFG. Most of the functional parallelism is made explicit by splitting the application into 12 separate actors. The current and reconstructed frame are stored in local state of the corresponding actor, which have the task to distribute the appropriate parts of these images to the actors who need them. The output is a encoded video bit-stream, but I/O accesses are not modeled.

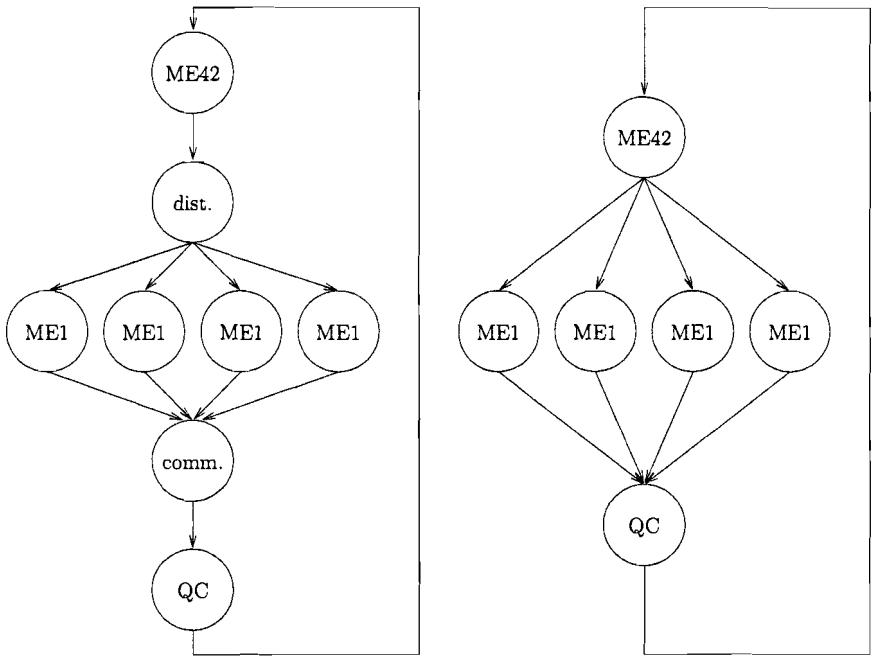
Synchronization takes place on the macro-block level, or X-loop level. This is a relatively fine-grain level, but the FIFO's make sure that variation in actor execution time is absorbed so that processor stalls for external communication are avoided.

The arcs that represent the FIFO's in between the actors are annotated with the data structures which are in the tokens on that arc. The table shows their full names and sizes, where minimum size means the amount of data that is actually used by the receiving actor. However, depending on the exact implementation, it could be necessary to send more data, because not all dependencies are known when or where the token is sent. For example in the motion estimation, a macro-block is matched within a search area of 58×58 pixels, but the motion vector is determined in three hierarchical steps. Therefore the possible search area for the last step (ME1) is already narrowed down to 18×18 pixels. The "reconstructed" actor should however know in advance which 18×18 pixels to send, which depends on the motion vector as estimated by the ME2 actor. The motion vector should be communicated via an additional connection from



Symbol	Data structure	Size (bytes)	Minimum size (bytes)
MB	Macro Block	16×16	16×16
MB2	1× sub-sampled MB	8×8	8×8
MB2	2× sub-sampled MB	4×4	4×4
SA	Search Area	58×58	18×18
SA2	1× sub-sampled SA	24×24	12×12
SA4	2× sub-sampled SA	12×12	12×12
MV	Motion Vector	2	2
QT	Quad Tree	16×16	16×16

Figure 4.8: QSDPCM as SDF graph



Symbol	Task
ME42	Motion Estimation at $\frac{1}{4}$ and $\frac{1}{2}$ resolution
dist.	Distributor
ME1	Motion Estimation at full resolution
comm.	Commutator
QC	Quad-tree quantization and Coding

Figure 4.9: QSDPCM in (a) Hapi and (b) AEsim/Hybrid

the ME2 to the "reconstructed" actor. This solution reduces the data to be communicated, however the additional dependency increases the latency.

Based on the profiling information and parallelization options as described in section 4.2, actors that are part of the same functional pipeline stage are merged into one task, see figure 4.9a. This saves the overhead of intra-tile FIFO communication and run-time scheduling. It also means that the scheduling order is now fixed at design time. An additional advantage is that the merged actors can share their state.

There are different types of data flow graphs which give different levels of guarantees. Homogeneous Synchronous Dataflow (HSDF) can be analyzed at design time and gives most guarantees, but has as limitation that every firing (execution) of an actor, exactly one token is consumed per input FIFO and one is

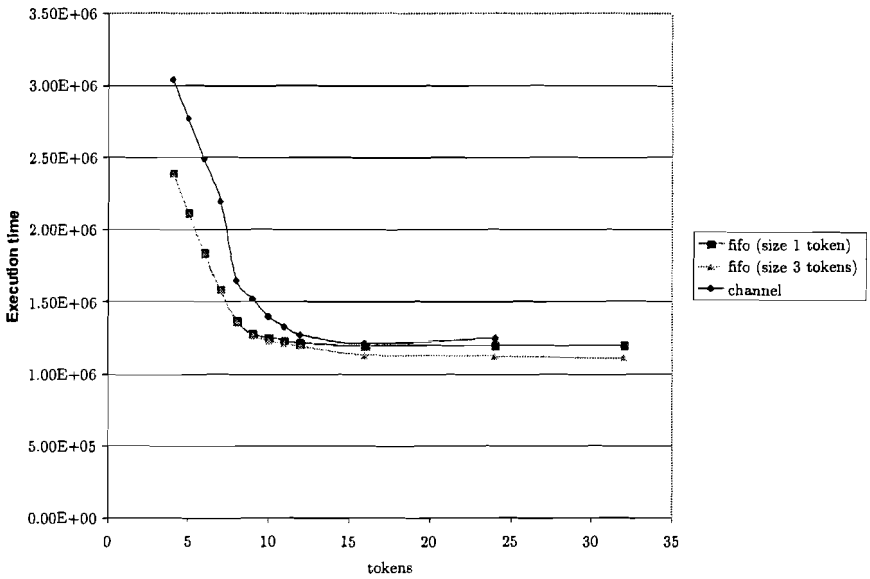


Figure 4.10: Token amount and FIFO size exploration

produced per output FIFO. It is possible to make a data split in HSDF, but then distributor and commutator actors have to be added to change the granularity of the tokens between ME42 and distributor and between commutator and QC. The distributor does not change the contents of the tokens, but only copies them. The commutator reverses this process by reading in one token from each input FIFO and writing them to the output FIFO such that the order of the tokens is the same as before the distributor.

Unfortunately the design time analysis tool for HSDF graphs (Heracles) was not available, therefore the exploration was done by simulation.

In the mapping to the AEsim simulator, the distributor and commutator have been merged into the ME42 and QC actor respectively. This was necessary because of the absence of a run-time scheduler and thus the limitation of one actor per processor, but it also improves latency and memory requirements. This results in the dataflow graph of figure 4.9b.

Since there is almost a whole frame delay in the loop in the dataflow graph, from the QC actor to the ME42 actor, there is a lot of freedom to apply re-timing to achieve optimal functional pipelining. This has the advantage of increased throughput, but on the downside latency is also increased.

The effects of re-timing can explored by adding more initial tokens. From figure 4.10 it becomes clear that first the throughput increases a lot, but eventually it stabilizes. The total execution time can even increase because of the time it takes to empty the functional pipeline increases when more tokens are present.

It turns out that, although the actors have significant variation in execution time, the size of the FIFO's has no large effects on the overall execution time. Also the effect of adding the model of the communication channel is limited

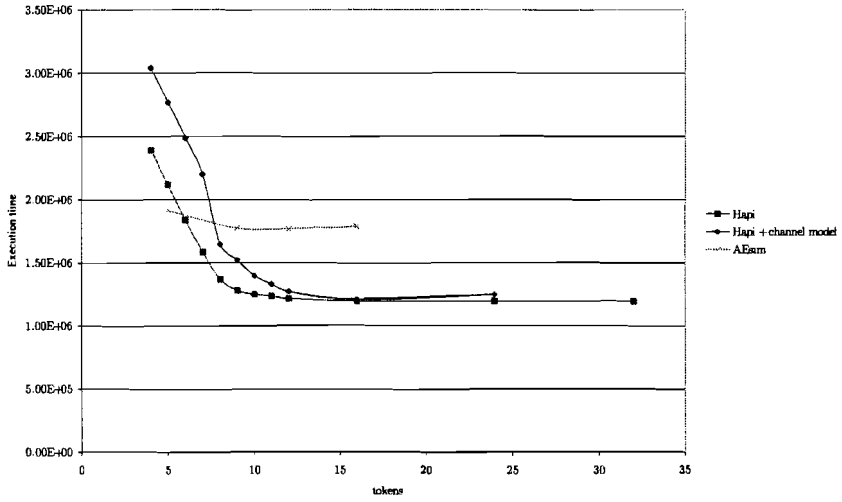


Figure 4.11: The results measured with AEsim compared to the estimates of Hapi

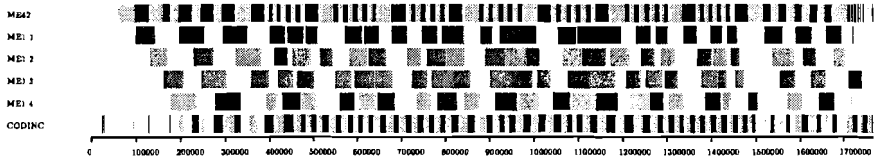


Figure 4.12: Hijdra actor execution schedule for QCIF resolution

when enough tokens circulate in the graph. This means that the latency of the interconnect can be effectively hidden.

When the results obtained from the Hapi and AEsim are compared in figure 4.11, it shows that AEsim results are much worse than what Hapi estimated. This can be explained by the fact that the Hapi extimation is based on the profiling of the kernels, but when rewriting the application to a DFG, the kernels can slightly change. Moreover the "current" and "reconstructed" actor were added to manage memory, but no time was accounted for that. The processing required for these copy functions was estimated to be almost neglectable, but due to the unpredictable control flow that they exhibit when processing macro-blocks on the edge of image, they execute particularly slow on a VLIW architecture such as the used TI C62. As a result the balancing of the processor loads was disrupted, resulting in a very high load for the ME42 processor, which amplified the effect. In the future this can be avoided by profiling the actors again after the rewrite to DFG.

The scheduling graph in figure 4.12 also shows the high load of the processor running the ME42 actor. Also the processing of macro-blocks on the edge of

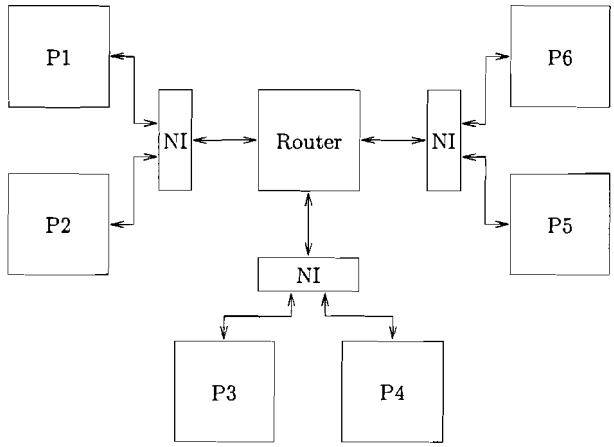


Figure 4.13: Platform instantiation for the Hijdra mapping

the image can be recognized by the much longer executions of ME42.

4.4.1 Platform instantiation

The platform instantiation that results from the mapping process is shown in figure 4.13. The network topology and mapping of tiles and network interfaces is identical to the IMEC/MPSoC case, i.e. two processor tiles share a network interface, but there is no memory tile. For the same reasons a single router topology is chosen.

The connections that are used are all Best Effort (BE), because there are nine connections and only eight GT connections can be defined in the version of AEsim available. Tests were performed to verify that this has no significant effect on the timing behavior since there is enough bandwidth in both cases. How ever further experiments are required to make a fair comparison in terms of communication

To be able to make a fair comparison between the IMEC/MPSoC solution and the Hijdra solution, the simulation platform should be kept as similar as possible. The existing implementation of a Hijdra processor tile for AEthereal consisted of an ARM and a custom designed CA. An ARM can however not be compared to the TI C62 DSP in the IMEC tile, which is very well suited for video processing.

There are three potential solutions to this: the ARM can be replaced with the TI in the Hijdra tile, the IMEC/MPSoC CA can be replaced with the Hijdra CA, or the IMEC/MPSoC CA can be extended to support the functions of the Hijdra tile. In chapter 2.3 it is explained that the IMEC/MPSoC CA provides an extensible protocol on top of the NoC, and this can be used to provide the extra required services. This, as well as the fact that the Hijdra tile and CA could not be made to function correctly in the available timeframe, led us to prefer the last solution over the other ones. Details about the implementation of token FIFO support by the IMEC/MPSoC CA can be found in section 6.1.

Token	size (byte)	#FIFOs	FIFO depth	total (byte)
ME42 ⇒ ME1	596	8	3	14.304
ME1 ⇒ CODING	596	8	3	14.304
CODING ⇒ ME42	268	2	3	1.608

Table 4.1: Token memory requirements

4.5 Results

4.5.1 Execution time

The execution time is the time it takes for the application to encode one video frame, producing an encoded bit-stream and a new reconstructed frame. This time is measured in processor cycles, since the processor clock is set at a fixed number of cycles per second.

A related measure is the processor load, which is defines as the number of cycles a processor is active, i.e. not waiting for synchronization, divided by the total execution time. When the loads of all processors are considered, we get an indication of how well the parallelization of the application is balanced. The goal is to spend as little time as possible for synchronizing the processors, as this will generally increase the total execution time. This measure is also dependent on the application, where applications with a more dynamic control flow will generally be less efficient.

In figures 4.14 and 4.15 the execution time break-up for the IMEC/MPSoC mapping is shown. The same phenomena as in the thread scheduling graphs can be seen, such as the coarse-grain synchronization which results in a large pre-amble and bad thread balancing which results in low utilization of the fourth ME1 thread.

In figures 4.16 and 4.17 the execution-time break-up for the Hijdra mapping is shown. Again, the same effects as in the thread scheduling, such as the very high load of the ME42 processor at QCIF resolution. Probably for QCIF, three ME1 actors would yield the same performance, because they are only used for approximately 60% and are well balanced.

4.5.2 Memory requirements

Memory requirements for the Hijdra mapping are easily calculated. Multiplying the size of one token with the number of tokens that fit in a FIFO gives the size of a single FIFO; summing all the sizes of the input and output FIFOs gives the required L1 size of one task.

In case of the IMEC/MPSoC mapping using MH, the L1 size is an input to the tool and in this case 4Kbyte is chosen.

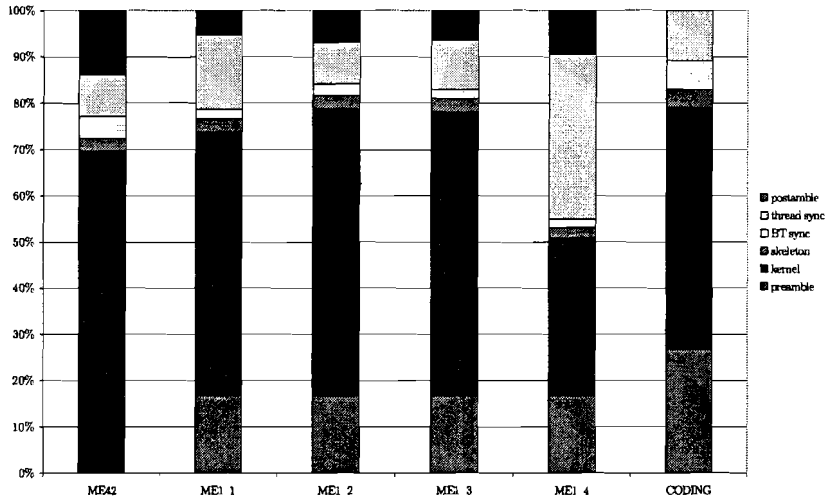


Figure 4.14: Execution time break-up for IMEC/MPSoC mapping at QCIF resolution

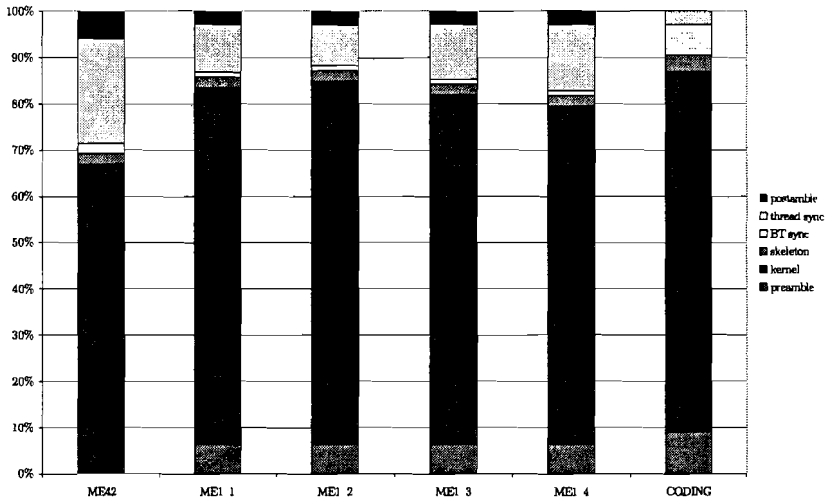


Figure 4.15: Execution time break-up for IMEC/MPSoC mapping at VGA resolution

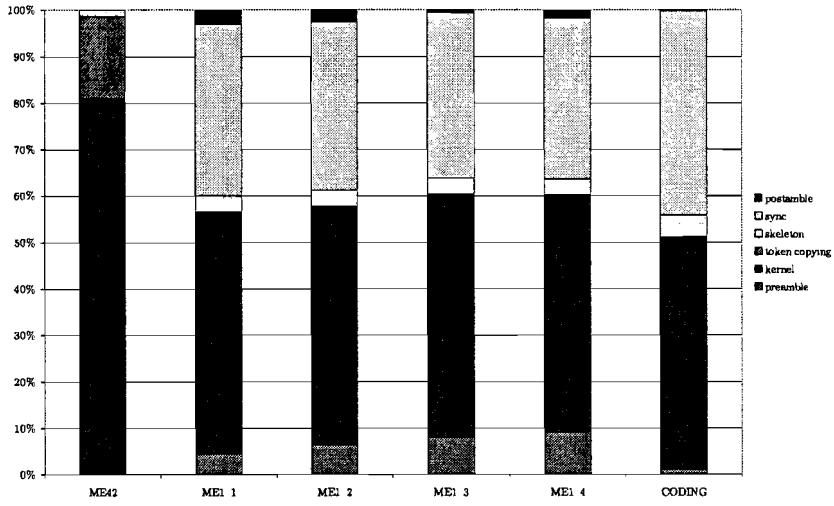


Figure 4.16: Execution time break-up for Hijdra mapping at QCIF resolution

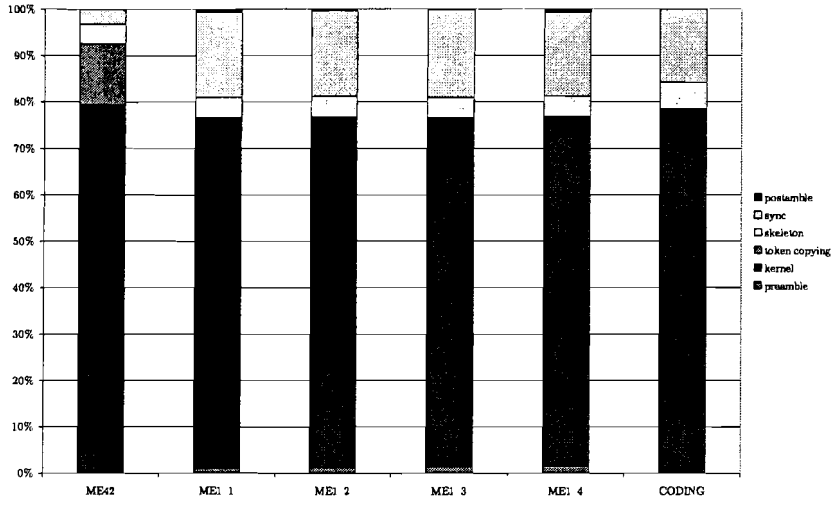


Figure 4.17: Execution time break-up for Hijdra mapping at VGA resolution

Actor	FIFO	state QCIF	state VGA	total QCIF	total VGA
ME42	7.956	57.024	691.200	64.980	699.156
ME1	3.576	0	0	3.576	3.576
CODING	7.956	0	0	7.956	7.956

Table 4.2: Actor memory requirements (in bytes)

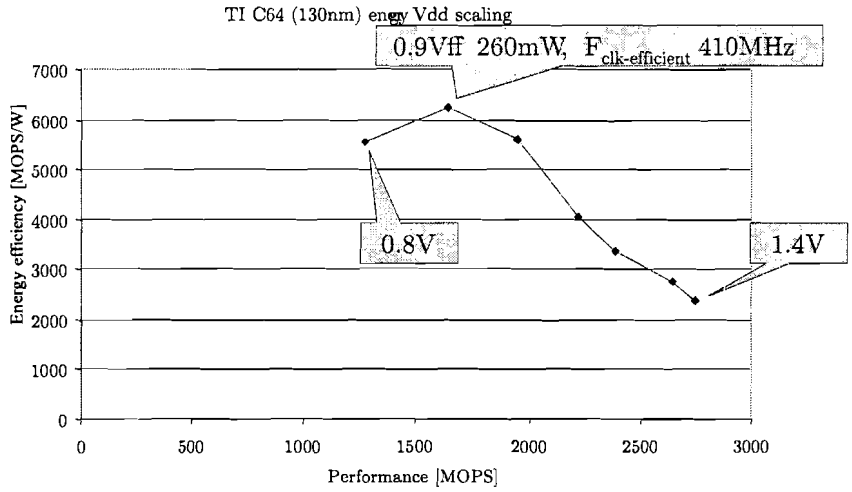


Figure 4.18: The relation between performance and energy efficiency for a TI C64 VLIW DSP show that the highest efficiency is achieved at a clock frequency of 410 MHz.

4.5.3 Energy

The energy consumption of a systems can be attributed to three sources: the processors, the memory and the interconnect. The first source is mainly dependent on the clock frequency at which it operates. In [AAH+02] this dependency is given for a processor similar to the one used in the simulator, see figure 4.18.

The energy consumption of the L2 memory is dependent on the type of memory, i.e. SRAM, SDRAM etc., the size of the memory and the number of accesses. In general the energy per access will increase as the size of the memory increases. For some types of memory, most notably SDRAM, also the access pattern is important because the memory controller has only direct access to one line at a time. If an access is not in the currently loaded line, a new line has to be fetched which costs a lot of energy. Therefore the energy cost per access is more difficult to predict and requires accurate memory models. However when fetching large blocks, it is reasonable to assume most of the line is used and the energy can be estimated by distributing line access energy over all bytes in the line.
For other types of memory the average consumption per access can be used.

The energy consumed by the NoC is dependent on several factors, such as

format	width	height	# macroblocks	aspect ratio
QCIF	176	144	99	11:9
VGA	640	480	1200	4:3
HDTV 720p	1280	720	3600	16:9

Table 4.3: Image resolutions

flit-clock, quality of service (BE or GT), and the actual number of flits sent. However, since the network parameters are not optimzed, and the two mappings have to use a different quality of service, no realistic comparison can be made.

4.5.4 Scalability

The scalability of the mapping solution can be examined by using a different image resolutions as input. The solution was tested with QCIF and VGA resolutions and predicted for HDTV 720p. Table 4.5.4 shows the size of these resolutions. This was however not verified for HDTV because the simulation would take an extremely long time.

The Hijdra solution is in terms of execution time largely linearly dependent of the image resolution due to the streaming nature of the implementation. However the fact that a larger image has relatively less macroblocks on the edge, combined with the knowledge that macroblocks on the edge take longer to process for the ME42 actor, lead to the expectation that it will scale at least linearly with the image size. The ME42 actor is only the bottleneck during pre- and postamble, therefore this only a small advantage.

The IMEC/MPSoC mapping solution is expected to benefit from the larger number of macroblocks on a row because of two reasons. First of all the number of macroblocks assigned to a processor when applying loop tiling will vary less, because it is more likely that that number of MBs is a multiple of the number of threads. For example dividing the 11 macroblocks of the QCIF resolution over 4 ME1 threads yields a variation of 1 out of 3, while the 40 MBs of the VGA resolution can be exactly divided by 4. Secondly the variation in processing time will decrease when considering more MBs. Both these benefits will result in a lower amount of thread synchronization compared to the overall execution time.

A larger image also means a larger preamble in the IMEC/MPSoC case. The absolute duration of the preamble is the time it takes to subsample three rows of macroblocks and thus dependents on the width of the image. The relative duration is only dependent on the height of the image. If the aspect ratio remains constant, the relative duration of the preamble decreases linearly with the image size. If however the image becomes wider, as with both VGA and HDTV cases when comparing to QCIF, the relative preamble will decrease less than linearly. Overall the IMEC/MPSoC mapping is expected to scale better than linearly.

Figure 4.19 shows the measured results with a linear extrapolation for HDTV resolution. As explained before, this linear extrapolation can be considered

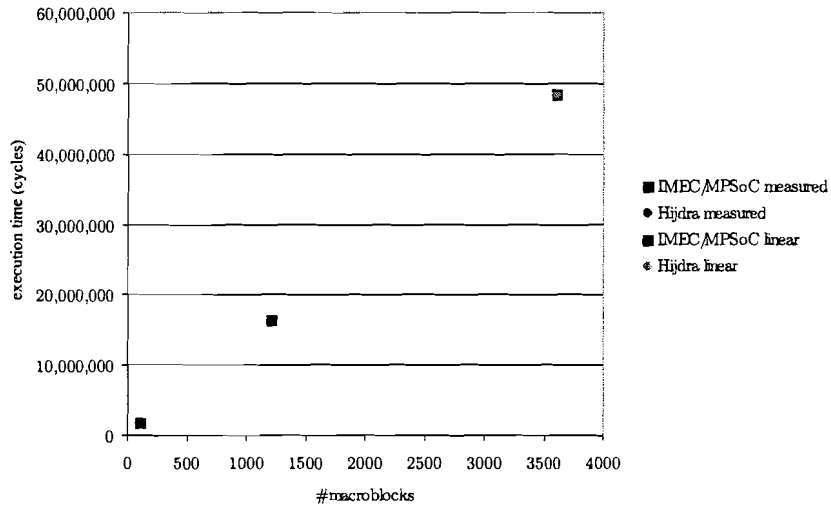


Figure 4.19: Scalability: QCIF and VGA resolutions are measured on the AEsim simulator, HDTV is extrapolated

worst case; in reality the execution time will be shorter. From the graph it becomes clear that the results of both flows are very close with respect to execution time, but Hijdra is slightly faster.

4.5.5 Design effort

Design effort is a measure for the effort a designer has to spend to achieve a certain goal or a set of goals. The goal is difficult to quantify, since it is usually something like "as fast as possible" or "as energy efficient as possible". More effort will in general result in a better solution, since the design space is too large to cover completely.

It is also extremely difficult to measure the design effort, since it depends heavily on the designer and the experience he has with a certain application mapping design flow. However it is possible to investigate the support for automatic design space exploration.

In this respect both the Hijdra and the IMEC/MPSoC design flow offer a high level simulation environment, respectively named Hapi and Tipsy. At this level the timing behavior and parallelization options can be effectively investigated. Both require manual adaption to use these environments, but Hapi has much stricter requirements on the data-structures to be used and the communication between tasks, namely that they respect the SDF model. Because of this it is required to rewrite all the tasks of the application together. The IMEC/MPSoC flow on the other hand allows the designer to apply the necessary changes incrementally.

The effects of memory size on the performance can also be investigated in the high level simulation. In case of the Hijdra flow, just a single parameter per FIFO can be varied to increase or decrease the size of that FIFO. If the number of actors increases, the number of combinations of different FIFO sizes can grow very large, which would complicate the exploration. It should however be possible to automate this task.

In case of the IMEC/MPSoC design flow, the memory size and hierarchy is an input to the MH tool. Varying this size will requires MH to make a new trade-off in the memory assignment and generate new application code. Fortunately this is a fully automated process and MH will even give estimates for the performance.

No exhaustive exploration has been performed on the possible combination of topology, mapping and connection parameters for the AEthereal network-on-chip, because it is beyond the scope of the thesis and the number of possibilities is very large. Even though this exploration could be automated when suitable heuristics are developed, the exploration would take long because of the simulation time. A simple topology is chosen for both mappings in this case study. The assumption is that communication should not be the bottleneck and this can easily be achieved since the network can deliver 2Gbps of bandwidth.

In theory, it should be trivial to move the mapped application from a high level simulator like Hapi to a lower level simulator like AEthereal. In practice however this can lead to a lot of practical issues, because the input descriptions often differ. Still the low-level, detailed mapping is worth the effort because now all details must be clarified and some do turn out to be more relevant than previously expected.

An example of that is the zero padding of the search-area in the Hijdra mapping. In the original application this could be distributed over all the ME1 actors and was thus incorporated in the execution time information of that actor. During the mapping this functionality had to be moved to the ME42 actor, which became much more loaded while the ME1 became lighter. This had a negative effect on the balancing and could have lead to other mapping decisions if known beforehand. The low level simulator is therefore an important verification of the mapping flow.

Chapter 5

Conclusions

In the analysis of both the Hijdra and the IMEC/MPSoC platform templates and application mapping design flows, no indications were found that the performance in terms of execution time that can be achieved would be very different. Also the case study shows no significant differences, although the Hijdra solutions is a little faster. They both achieved a processor utilization of approximately 80% for target resolution, which is reasonably good.

In terms of memory requirements, only IMEC/MPSoC offers a structural solution for handling small L1 memories in combination with large data structures. Even though the available L1 memory on future platforms is not clear, the trend toward higher resolutions for video will require such a structural solution. This problem is addressed in the next chapter, in which a hybrid solution is proposed.

Hijdra mapping of the QSDPCM application benefits from fine-grain synchronization and low overhead FIFO communication. The current MPSoC implementation of the CA protocol is not the most efficient, but this can be solved in the future. A more flexible synchronization mechanism could be beneficial for the MPSoC solution.

Chapter 6

Hybrid solutions and future work

Section 6.1 explains the extensions made to the CA of the MPSoC IP block to support also FIFO style communication and enable hybrid mapping solutions. Section 6.2 describes the possibilities of a hybrid mapping. The last section of this chapter summarizes the future work on this subject.

6.1 Hybrid CA

This section explains the modifications that had to be made to implement token FIFO communication using the IMEC/MPSoC CA. The modifications can be grouped into four classes: extension of the CA protocol, extension of the CA itself, extension of the memory controller and the implementation of the application interface.

The CA protocol is extended with a "FF_WR" command. The main differences with the existing "BT_WR" command are that "FF_WR" commands are executed in the same order as they are issued, and that there is no acknowledgment returned to the sender when it is finished. The first is required because the destination memory addresses are only generated when the transfer is received by the destination CA based on the current state of the FIFO registry, that is dependent on the target processor activity. The acknowledgment is not required because synchronization already takes place by means of claiming and releasing space in the FIFO.

The extensions of the CA functionality are only triggered when a "FF_WR" command is executed. The existing functionality used for block-transfers is not altered and can thus still be used, even in combination with the new command.

The sending CA, when starting to execute a "FF_WR" command, first claims the data in the FIFO, then adds the source address of the token to the transfer descriptor, and subsequently lets the CA core handle the transfer as usually. When the CA core is finished with this, the read counter in the FIFO registry is updated. The receiving CA first claims space in the FIFO which corresponds to the NI FIFO at which it receives the transfer. It also adds the destination address of the token based on the current state of the FIFO registry and lets

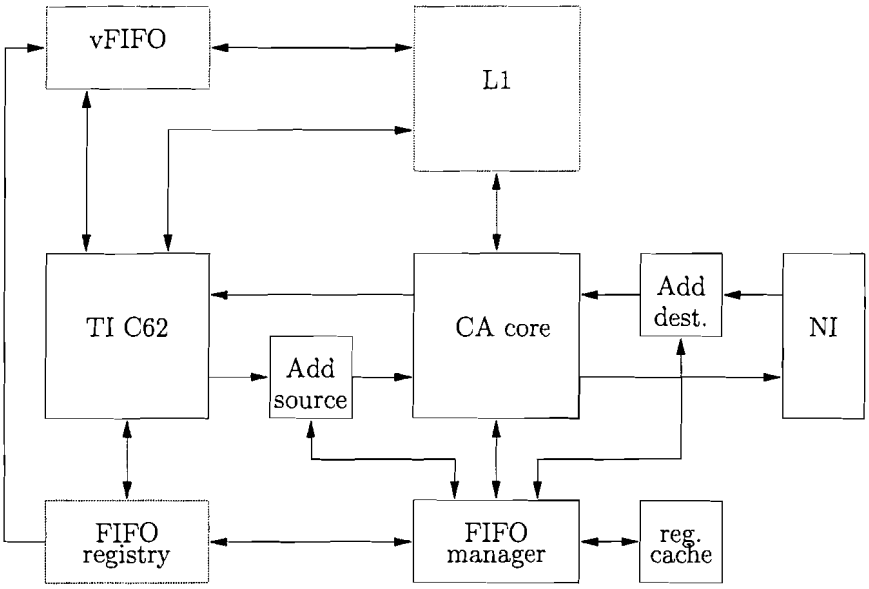


Figure 6.1: Schematic diagram of the extended MPSoC CA. Blue boxes are part of the CA, green boxes are part of the memory. The L1, TI C62, CA core and NI components are the existing parts of the MPSoC CA.

BT parameter		Sending CA	Receiving CA
Source	address	constant	ignore
	width	constant	keep
	height	— fixed —	
	X offset	insert	keep
	Y offset	— fixed —	
Destination	address	ignore	constant
	width	ignore	constant
	height	— fixed —	
	X offset	ignore	insert
	Y offset	— fixed —	
Copy	width	constant	constant
	height	— fixed —	
Chunk size		— fixed —	
Modulo mode		— fixed —	
BT command		— fixed —	

Table 6.1: The use of the BT descriptor for FIFO communication

the CA core further handle the actual transfer. When the transfer is done, the CA updates the write counter in the FIFO registry.

Table 6.1 shows the use of the existing BT descriptor for FIFO communication. Here "fixed" means the value the same for all FIFO's and is set at design time, e.g. the height is set to 1 because only 1-dimensional transfers are used. "Constant" means the value can be different for each FIFO, but remains the same during the whole existence of that FIFO, e.g. the start-address and size of the FIFO. "Insert" means that this value should be set by the CA at run-time for each transfer based on the current FIFO registry values, e.g. the offset within the FIFO where the token is located. "Ignore" and "keep" mean that these values are not used, or should be left untouched respectively.

To be able to reuse as much of the functionality of the CA core as possible, the decision was made to implement the FIFO's as word-FIFO, not token-FIFO. Also the way of differentiating between a full and an empty FIFO is different than in the original Hijdra implementation. Hijdra uses a "wrap flag", which doubles the address range of the FIFO such that the write counter always has a larger value than the read reserve counter. In the hybrid implementation a one-word buffer is used in between the read reserve counter and the write counter in case of a full buffer, i.e. when the two are equal, the FIFO is empty. A consequence is that tokens can start at any word-aligned address within the FIFO, and that tokens can in principle have different sizes. Also could a token be physically split over the end and beginning of the FIFO address range. This makes it however more complex to address a token from a application running on the processor. Therefore a address translation component has been added to the memory interface of the processor, which translates the currently claimed data or space to a consecutive address range with a fixed start address, called a virtual FIFO (vFIFO). (See figure 6.2) A disadvantage is that in a real implementation this can cause additional latency when accessing the FIFO because

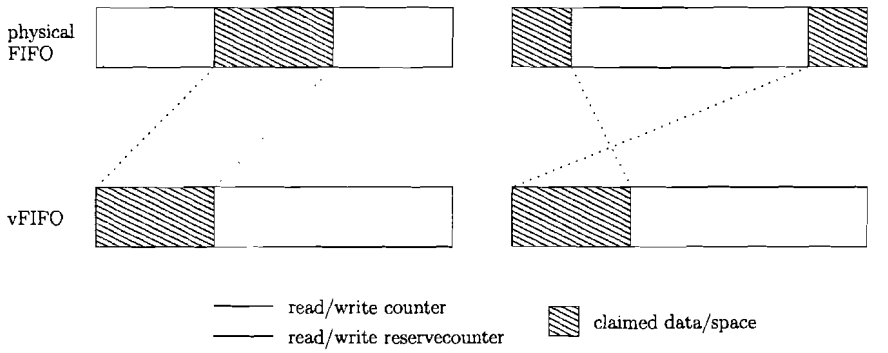


Figure 6.2: vFIFO

of the address calculation that has to be performed on every access, but in the current simulator this is not modeled.

The FIFO registries are stored in the L1 memory, such that they can be configured at run-time. In appendix A.2 the interface is listed, which also shows the structs which store the registries. The fifo_info struct points to list of fifo_registry structs for incoming and outgoing FIFO's. A fifo_registry struct contains the origin (start address) and length of the FIFO, the origin of the vFIFO, the read and write counters and reserve counters, and the corresponding network channel. Most of these values are not changed frequently and are therefore also cached in the FIFO manager, which reduces bus transactions to the L1 memory.

Also listed in appendix A.2 is the claim/release interface which uses counters and reserve counters as described in [GNL01]. It allows for variable size and decoupled claiming and releasing of data and space, and direct access to the claimed memory space. The peek_read/peek_write-functions are non-blocking functions to check the available data/space in a FIFO, the peek_wait function block until the requested data/space is available. The interface listed here is for the application side, the CA has a similar interface.

The hybrid CA does not take into account the space available in the NI FIFO for determining the amount of data to be sent, while the Hijdra CA can pre-empt the execution of a transfer when the NI buffer is full or empty. It can then switch to a transfer on a different connection The hybrid CA can only pre-empt after a chunk has been finished. The advantage of the Hijdra CA is that it potentially needs smaller FIFO's to achieve the same performance.
Also, by using the BT protocol, the FIFO communication as implemented does not have the low overhead advantage that the Hijdra CA has. This protocol can however be further optimized, which would make it negligible when communicating large tokens.

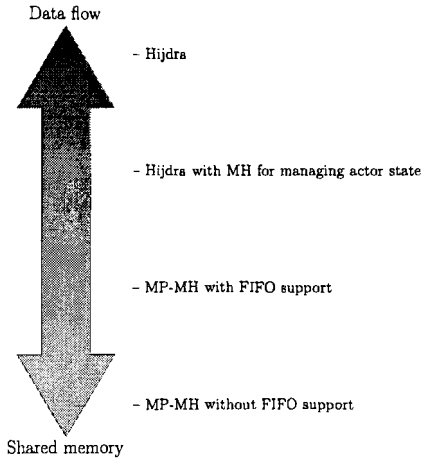


Figure 6.3: Mapping solution space

6.2 Hybrid mapping

In the case study, both mapping solutions have their advantages and disadvantages. They can be considered two extremes in a larger mapping solution space as depicted in figure 6.3. However also intermediate, or hybrid, solutions are possible, potentially combining the advantages of both Hijdra and MPSoC.

The case study made clear that the pure dataflow implementation of Hijdra can result in high requirements for L1 memory. This is mainly due to the fact that an actor needs to store data it wants to re-use in its state, even though this reuse is only much later. A possible solution is to use the single processor MH tool to analyze the re-use and lifetimes of the data-structures in the state of each actor. This could result in a solution where the state is stored in L2 memory, and the data that is actually required during a firing is pre-fetched to L1, e.g. the search area for ME1 in QSDPCM. This way it is possible to exploit the benefits of a memory hierarchy.

The MP-MH tool of MPSoC has recently already moved toward such hybrid mapping by supporting processor-to-processor communication instead of communication just through the shared memory. However, because the existing platform simulators did not support it, it is not yet used. There should also be a standardized interface between the mapping tool and the platform. This interface can then be used to efficiently communicate small data-structures, which do not have to be shared with other processors, e.g. motion vectors in QSDPCM.

In essence, the possibility to choose the type communication creates a new dimension in the solution space, for which the designer a tool must make a choice. This trade-off should consider the cost of the memory, the cost of communication and synchronization for the possible solutions within given constraints.

A possible hybrid mapping solution for QSDPCM is shown in figure 6.4. It uses FIFO communication between the ME42, ME1 and QC threads, but instead of storing the current, previous and reconstructed frame in the state of ME42, it

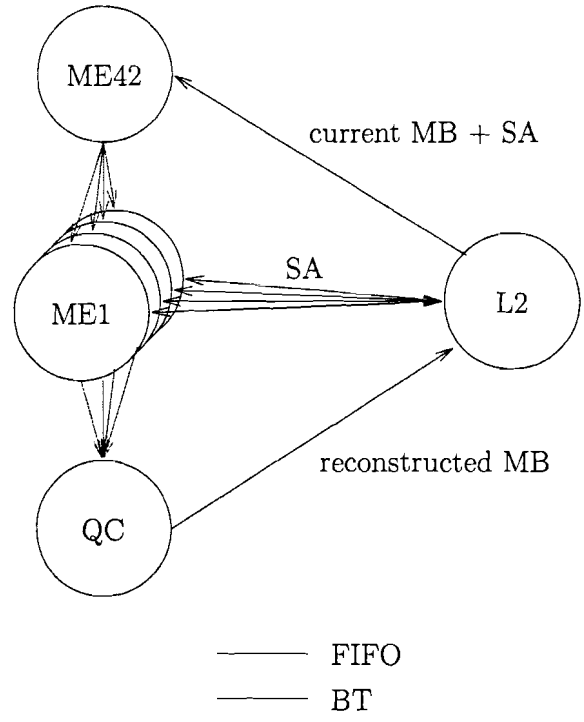


Figure 6.4: Possible hybrid mapping for QSDPCM

is stored in a L2 memory. All actors can access the part of these frames using BT's, and QC can write the reconstructed macro blocks to the right place also using BT's. For some data-structures a trade-off must be made, e.g. the current macro block used by ME42 can be forwarded directly to ME1 by mans of FIFO communication, or ME1 can fetch it from L2 memory itself.
Communication inside a thread is thus statically scheduled, while inter-thread communication is more dynamic, depending on the FIFO depth.

There are a few issues to be solved before exploring how possible hybrid solutions can be done efficiently. First of all, the hybrid CA is only available in the AEthereal based simulator. This is not very well suited for design space exploration because of its relative low speed. It is also not suited for debugging the application, and since there is not yet compatible tool support, mapping the application has to be done manually.

6.3 Future work

In the current Hijdra application mapping, not all features of the Hijdra platform are used. This is mainly because not all features were available in the used platform simulator. It would however be interesting to see the effects of the run-time scheduler, especially when multiple applications share the platform resources. Also within MPSoC a run-time manager is being developed. Future work could be to include these in a case study and then again compare the results. This would give a measure for the composability of both design flows.

In the case study, it turned out that the kernel profiling data differ significantly from the actual actor execution times. In the future, these actors should be profiled again after the application is rewritten to a DFG. It is possible that a different, better mapping will be the result.

The results of the MPSoC mapping could be further improved when the latest features of the mapping tools and the platform are incorporated. Especially L1-to-L1 block-transfers and a more flexible synchronization scheme could result in a better solution.

The parameters for the NoC ware not explored, and therefore it is not possible to accurately estimate th energy consumption of the interconnect. This could include the comparison of communication requirements and measurements of the actual usage. To estimate the energy consumption of the platform as a whole, also accurate models for the memories should be added and possibly also for the processor.

In general it would be interesting to work out one or more hybrid mapping solutions. The platform in principle already supports this, but should be tested more thoroughly. Also the interface between the design flow and the platform should be made compatible.

Appendix A

Simulator

The results presented in section 4.5 are gathered by simulation of the platform. The used simulator is based on the AEthereal network-on-chip simulation framework, in which custom IP blocks, such as an IMEC/MPSoC tile with a TI C62 ISS and an IMEC/MPSoC CA, have been integrated.

In the next section the implementation of the IMEC/MPSoC IP block is described.

A.1 IMEC IP block

The IMEC/MPSoC IP Block is an instantiation of a tile of the platform template described in section 2.1. The processor used is a Texas Instruments TMS320C62x, a VLIW DSP processor, because it is well suited for the video encoding task as used in the case study. Also an instruction set simulator (ISS) for this processor is publicly available [Cup99], which makes it easier to add custom memory mapped I/O devices.

The internal components of the tile are connected through a bus. For this the OCCN framework [CCG+03] is used, which defines a generic interface for on-chip interconnects. More specific, the StdBus is used, which is a TLM model of a simple bus. The L1 SRAM is a banked memory and does take into account access conflicts from the CPU ports, but not the conflicts caused by the CA.

The use of the block-transfer issue and sync interface is given in code fragment A.1. It shows a simplified fragment of the full resolution motion estimation ME1. The TI compiler is instructed to put the sdr_frame variable in SDRAM section by the use of a pragma. The local copy variable is defined with a size double the copy size to allow pre-fetching one macro-block while processing the other, i.e. the transfer is issued one loop iteration before it is used. This also means the copy has to be initialized, and that during the last loop iteration no transfer has to be issued anymore.

The existing CA was adapted to the interface of the AEthereal 3.0 simulation framework and extended with several new features, including the ability to handle multiple network connections per CA and the support for processor to processor communication. The first feature is used in the memory tile, which

```
/ Store frame (of size N by M pixels)
   in SDRAM section of memory map        /
#pragma DATA SECTION(sdr frame, " sdram");
unsigned char sdr frame[N M];

/ Define local copy for macroblocks
   (of size 16 by 16 pixels)             /
unsigned char frame copy[2][16 16];

/ Initialize copy /
BLOCKXFER ISSUE(sdr frame, M, N,
                frame copy[(lower bound)&1], 16, 16,
                (lower bound) 16, y 16,
                16, 16, 7, dma id);

for (x lower bound; x upper bound; x++)

   / Block until BT has finished /
   BLOCKXFER SYNC(1);

   / Issue prefetching of next macroblock, unless
      it's the last loop iteration                    /
   if (x+1 upper bound)

      BLOCKXFER ISSUE(sdr frame, M, N,
                      frame copy[(x+1)&1], 16, 16,
                      (x+1) 16, y 16,
                      16, 16, 7, dma id);


   / Process macroblock /
   ...
```

Code fragment A.1: Example of BT issue and sync

now is much more similar to a processor tile and only contains a single CA. The second feature is used in the implementation of the Hybrid CA, described in the next section. Also some features of other experimental CA implementation were merged into the new one.

A.2 Hybrid programming interface

```
/  fifo.h  : Application  interface  to  L1 FIFO management
   April  2006
   Rogier  Baert
   rogier.baert imec.be
  /
#ifndef  FIFO H
#define  FIFO H

#include  "simc62.h"

typedef  struct  fifo registry
  void  origin;
  void  v origin;
  unsigned  length;
  volatile  void  volatile  writec;
  volatile  void  writersvc;
  volatile  void  volatile  readc;
  volatile  void  readrsvc;
  unsigned  channel;
 fifo registry;

typedef  struct  fifo info
  unsigned  nrInFIFO;
  volatile  fifo registry  InFIFO;
  unsigned  nrOutFIFO;
  volatile  fifo registry  OutFIFO;
 fifo info;

void  claim space(volatile  fifo registry ,  unsigned);
void  peek wait space(volatile  fifo registry ,  unsigned);
unsigned  char  peek space(volatile  fifo registry ,  unsigned);
void  release space(volatile  fifo registry ,  unsigned);

void  claim data(volatile  fifo registry ,  unsigned);
void  peek wait data(volatile  fifo registry ,  unsigned);
unsigned  char  peek data(volatile  fifo registry ,  unsigned);
void  release data(volatile  fifo registry ,  unsigned);

void  init registry(fifo info  f info,
                    unsigned  char  dir,  /  in  0,  out  1  /
                    volatile  fifo registry  f,
                    void  FIFO,
                    void  vFIFO,
                    unsigned  length);

#endif  /  FIFO H  /
```

Code fragment A.2: fifo.h

Bibliography

[AAH+02] S. Agarwala, T. Anderson, A. Hill, M.D. Ales, R. Damodaran, P. Wiley, S. Mullinnix, J. Leach, A. Lell, M. Gill, A. Rajagopal, A. Chachad, M. Agarwala, J. Apostol, M. Krishnan, Duc Bui, Quang An, N.S. Nagaraj, T. Wolf, and T.T. Elappuparackal. A 600-MHz VLIW DSP. IEEE Journal of Solid-State Circuits, 37(11):1532–1544, November 2002.

[Bae05] Rogier Baert. MPSoC mapping of the 3D reconstruction application. Technical report, Technische Universiteit Eindhoven, 2005.

[BM06] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. ACM Computing Surveys, 38(1):1–51, 2006.

[BMCC03] E. Brockmeyer, M. Miranda, H. Corporaal, and F. Catthoor. Layer assignment techniques for low energy in multi-layered memory organisations. In DATE'03: Proceedings of the conference on Design, Automation and Test in Europe, page 11070, Washington, DC, USA, 2003. IEEE Computer Society.

[BMPP+04] Marco Bekooij, Orlando Moreira, Bart Mesman Peter Poplavko, Milan Pastrnak, and Jef van Meerbergen. Predictable embedded multiprocessor system design. In SCOPES'04: Proceedings of the 8th International Workshop on Software and Compilers for Embeded Systems, pages 77–91. Springer-Verlag, January 2004.

[Bro05] Erik Brockmeyer. Updated design flow proposal for 3MF platform definition and mapping. Technical report, IMEC, Leuven, 2005.

[CCG+03] Marcello Coppola, Stephane Curaba, Miltos Grammatikakis, Guiseppe Maruccia, and Francesco Papariello. The OCCN user manual ST Microelectronics, 1.0.1 edition, 2003.

[CdGW98] Francky Catthoor, Eddy de Greef, and Sven Wuytack. Custom Memory Management Methodology Exploration of Memory Organisation for Embedded Multimedia System Design. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[Coc] Johan Cockx. Tipsy website www.imec.be/design/background/tipsy.

[Cup99] Vinodh Cuppu. Cycle accurate simulator for TMS320C62x, 8 way VLIW DSP processor, 1999.

[GDR05] Kees Goossens, John Dielissen, and Andrei Radulescu. Æthereal network on chip: Concepts, architectures, and implementations. IEEE Des Test, 22(5):414–421, 2005.

[GNL01] Om Prakash Gangwal, André Nieuwland, and Paul Lippens. A scalable and flexible data synchronization scheme for embedded HW-SW shared-memory systems. In ISSS '01: Proceedings of the 14th international symposium on System synthesis, pages 1–6, New York, NY, USA, 2001. ACM Press.

[HBB+04] Sang-Il Han, Amer Baghdadi, Marius Bonaciu, Soo-Ik Chae, and Ahmed A. Jerraya. An efficient scalable and flexible data transfer architecture for multiprocessor SoC with massive distributed memory. In DAC '04: Proceedings of the 41st annual conference on Design automation, pages 250–255, New York, NY, USA, 2004. ACM Press.

[IBDD06] Ilya Issenin, Erik Brockmeyer, Bart Durinck, and Nikil Dutt. Multiprocessor system-on-chip data reuse analysis for exploring customized memory hierarchies. In DAC '06: Proceedings of the 43st annual conference on Design automation, pages 49–52, July 2006.

[Mar06] Grant Martin. Overview of the MPSoC design challenge. In DAC '06: Proceedings of the 43st annual conference on Design automation, pages 274–279, July 2006.

[Moo04] Arno Moonen. Modelling and simulation of guaranteed throughput channels in a hard real-time multiprocessor system. Technical report, Nat.Lab., Januari 2004.

[MPS06] MPSoC team. Apollo definition: "Platform Mapping". Technical report, IMEC, Leuven, July 2006.

[Ost04] Christian Osterkorn. DTSE illustrated on a quadtree structured difference pulse code modulation application. Technical report, IMEC, Leuven, Januari 2004.

[Par05] Sonali Parmar. Predictable multiprocessor system-on-chip design for soft real-time streaming applications. Technical report, Stan Ackermans Institute, Technische Universiteit Eindhoven, 2005.

[SB00] Sundararajan Sriram and Shuvra S. Bhattacharyya. Embeded Multiprocessors Scheduling and Synchronization Marcel Dekker Inc., New York, NY, USA, 2000.

[Str88] P. Strobach. QSDPCM - a new technique in scene adaptive coding. In Proceedings of the 4th European Signal Processing Conference EUSIPCO-88, pages 1141–1144. Elsevier, 1988.

[Wic05] Johan Wickström. Design and implementation of a communication assist in a real-time multiprocessor system. Master's thesis, Chalmers University of Technology, Göteborg, Sweden, 2005.