Formalized Three-Layer System-Level Reuse Model and Methodology for Embedded Data-Dominated Applications

F. Vermeulen[1] F. Catthoor[2] D. Verkest H. De Man[2]
IMEC, Kapeldreef 75, Leuven, Belgium
[1] also Ph.D. student at the Katholieke Univ. Leuven
[2] also Professor at the Katholieke Univ. Leuven

Abstract

In embedded data-dominated applications a global system-level data transfer and storage exploration phase is crucial in obtaining an efficient solution. We have developed a novel formalism to describe reusable blocks such that the essential part of the design exploration freedom is retained. This formalism is the basis for a system-level reuse methodology which allows to reuse large parts of the design as structural VHDL and describes the costly data access related constructs at higher levels in the code hierarchy. Compared to a reuse approach based on fixed blocks, considerable power and area savings can be obtained, as demonstrated on real-life video and modem applications.

1 Introduction

Multi-media and communication systems are more and more being realized as heterogeneous systems-on-a-chip. An increasing part of these designs consists of reused IP. An important research area is how to obtain a power efficient solution in designs based on reused components.

Here we target embedded data-dominated applications. This includes real-time data-dominated applications which deal with large amounts of complex data types. This occurs especially in real-time multi-dimensional signal processing (RMSP) applications, in multi-media processing and in advanced communication front-ends, which handle indexed array signals (usually in the context of loops). This domain contains many important applications like video coding, medical image archival, multi-media terminals, artificial vision, speech and audio coding, xDSL modems, and wireless LAN modems.

In these application domains power management and reduction is becoming a major issue [1, 3, 5, 8]. As demonstrated by recent work at Princeton [9], at IMEC [2] and in the IRAM project at Berkeley, the most important power contribution in such data-dominated applications is due to the data storage and transfers, both in custom hardware and programmable processors. Also area and (in the programmable case) performance are heavily impacted by data accesses in the memory hierarchy. Hence, system architecture design for data-dominated applications should for a large part be steered by the organization of the global data transfer and storage. This observation remains valid in a reuse context, motivating the topic of this paper.

2 Related work

In the design reuse community, a lot of effort is put in facilitating reuse of blocks as is [13, 14]. Several research activities are focusing on the specification of reuse based designs [11, 12], targeting correct embedding and functional flexibility. They do not take into account the cost issues of embedding an IP block in another context than it was initially intended for. Hierarchical synthesis research [26, 24, 25] focuses on the reuse of datapath modules and control blocks. Optimization of the memory architecture is not included in this exploration. The flexibility of such reusable components is also limited to the interfaces, while a full exploration needs to include the storage organization internal to the block (for example merging an external buffer with an internal one).

At IMEC, we have demonstrated that through a combination of global and aggressive system-level data-flow and loop transformations with a heavily partitioned custom memory organization [16], up to a factor of 9 in peak power consumption can be saved compared to conventional solutions for the H.263 video conferencing decoder standard [7] and for other realistic multi-media kernels [10]. This clearly substantiates that the power bottleneck for custom processors can be alleviated to a significant extent. This can happen without an area or speed penalty. Actually, our experiments have shown that also area [7] and speed [6] can benefit significantly from our approach when these costs are explicitly incorporated in the exploration goals. This surprising result is mainly due to the fact that the initial algorithm is heavily reorganized while mapping it onto the dedicated memory organization. This reorganization has a positive impact on all cost parameters. The only real penalty is the increased design complexity which can be dealt with by appropriate tool support.

In this paper, we will show how the expertise gained by our previous work in this target domain can be used to obtain a systematic IP reuse methodology for data-dominated low power designs. It will allow to remove background storage and communication overhead also when an IP block is embedded in another context than it was originally designed for, while still avoiding the greater part of the redesign effort. In current design practice, subsystems are optimized separately. This strategy leads to a good solution in terms of design time reduction and performance (throughput) but unfortunately, it will typically give rise to a significant buffer overhead to handle the mismatch between the data produced and consumed in the different subsystems, as shown on Figure 1. For data dominated applications, this has a severe negative impact on the power, area, and system latency. Reusing blocks of a much finer granularity is an unsatisfactory solution, since this allows only for a very partial reuse of the design effort. To avoid both a major performance cost and the redesign cost, we propose another system-level IP reuse methodology intended for soft and firm IP blocks. It will remedy the identified problems with only a small penalty in terms of design time, as explained below.

The intuitive principles of our methodology have been published in [27]. For illustrative examples, we refer to that paper. The formalism and its properties, necessary for rigorously applying our methodology in practice were not presented yet. This paper also presents the design exploration at reuse time in a modem application. The next section develops the formalized three-layer description that forms the heart of our methodology to specify reusable data-dominated subsystems. Section 4 states the detailed reuse objectives and maps them on the formal description. We also show how the data transfer and storage exploration (DTSE) trade-offs are represented in this formalism, which results in an added IP value. A procedure that leads to the desired description concludes that section. Section 5 summarizes the global reuse design flow based on our description and identifies potential design transfer points in the design flow. Also this material is novel. Section 6 and 7 demonstrate the applicability of the formalism and the methodology on real-life video and modem applications.

3 Formalized three-layer description

In this section we will define the three layers which we identify in a design in order to enable our reuse methodology. At the lowest layer, we find the scalar operations; above that we describe the loops and indexing structure and at the top layer we will describe the global process control.

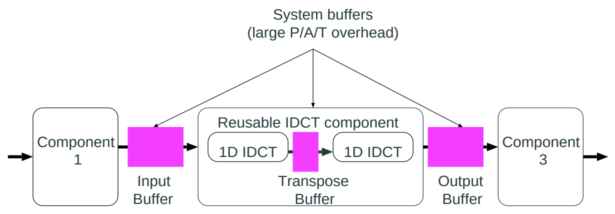

Figure 1. System buffer overhead introduced by monolithic reuse of 2-D IDCT.

3.1 Scalar layer

Entities at the scalar layer are defined as scalar operations having n scalar signals as input and m scalar signals as output. Referring to [19], a signal s is defined as a set of events e, which is a tag-value couple: $s = \{e | e \in T \times V\}$. In the context of scalar signals in digital systems, $T \subset \mathbb{R}$ and $V \subset \mathbb{Q}$. A scalar operation $o \subset S^n \times S^m$ with S the set of all signals, provides a mapping of n input signals to m output signals. Operations are functional and causal. Furthermore, we require that all output signals depend on all input signals. If this would not be the case, the function needs to be split into two or more subfunctions with a subset of the outputs. A network of operations can be represented by a hierarchical CDFG [23].

3.2 Loop and Indexing layer

Each entity at the indexing layer is represented by a set of indexing functions, defining multi-dimensional operations in function of scalar operations. An l-dimensional signal u is defined as $u = \{e | e \in T \times V^{k_1.k_2.\ldots.k_l}\}$. Selection of a scalar signal out of an l-dimensional signal is possible with an associated function $u' \subset T \times V^{k_1.k_2.\ldots.k_l} \times \mathbb{Z}^l \times T \times V$ mapping $(t, (v_1, v_2, \ldots, v_{k_1.k_2.\ldots.k_l}), (\iota_1, \ldots, \iota_l))$ on $(t, v_{\iota_1+k_1(\iota_2+k_2(\iota_3+\ldots))})$ where $\iota = (\iota_1, \ldots, \iota_l)$ is the index. A multi-dimensional operation is defined as $(n+m)$-tuples $(u_{\mathrm{in}1}, \ldots, u_{\mathrm{in}n}, u_{\mathrm{out}1}, \ldots, u_{\mathrm{out}m})$ satisfying

$$\Big(u'_{\mathrm{in}1}\left(u_{\mathrm{in}1}, f_{\mathrm{in}1}(i)\right), \ldots, u'_{\mathrm{in}n}\left(u_{\mathrm{in}n}, f_{\mathrm{in}n}(i)\right), \\ u'_{\mathrm{out}1}\left(u_{\mathrm{out}1}, f_{\mathrm{out}1}(i)\right), \ldots, u'_{\mathrm{out}m}\left(u_{\mathrm{out}m}, f_{\mathrm{out}m}(i)\right)\Big) \\ \in o,$$

where

- o is a scalar operation,

- iterator $i \in L \subset \mathbb{Z}^d$ where
  - L is the loop scope and
  - d is the dimension of the operation,
- $f_j \subset \mathbb{Z}^d \times \mathbb{Z}^{l_j}$ with l_j the dimension of u_j. f_j is an indexing function, mapping an iterator i to an index for a multi-dimensional signal. The set of indexing functions is I.

Data transfer and storage issues related to multi-dimensional signals and operations can be effectively modeled with a class of Polyhedral Dependency Graphs (PDGs) [20], extended in [21] for some important non-affine functions.

Indexing through an indexing function maps a multi-dimensional signal to a scalar signal, called a data stream. An indexing layer entity is formally defined as being $\subset I^n \times O$, with I the set of indexing functions and O the set of operations defined at the scalar layer.

From this definition it follows that the transformation and refinement of the scalar layer operations is independent of the indexing layer entities. This property will be crucial for our methodology and has not been published in earlier work.

3.3 Process control layer

In current complex data-dominated systems-on-a-chip, multiple data manipulation operations in separate threads of control are executing in parallel. The data manipulations inside these threads are described as indexing layer entities. The specification of the concurrent thread behavior requires a separate layer. However, threads that can be statically ordered, can be described as a single thread of control.

The conditions guarding the execution of the indexing layer entities are described at the process control layer. These conditions are defined either in function of parameters (e.g. mode settings) or events external to the system, or in function of constraints internal to the system. In this category we also find system-level resource constraints and timing constraints (synchronization, latency, execution rate).

Referring to [18], the system level can be described by a Multi-Thread Graph (MTG), which captures all the process control level aspects in the MTG layer. It can also be proven that the manipulations inside one indexing layer entity, encapsulated in an MTG operation node, are independent from the constructs internal to the MTG layer.

4 Proposed reusable description

Our reuse methodology aims at finding an optimal trade-off between two objectives:

1. reusing previous design effort as much as possible, and
2. losing as little freedom in the data transfer and storage exploration as possible when reusing parts of designs without change.

In this section, we will show how the proposed formalism can be used to represent reusable designs in a form that allows on one hand to reuse designs at levels close to implementation but still synthesizable from a structural VHDL (sVHDL) level (soft and firm IP in VSI terminology [14]), and on the other hand, exhibits a well-targeted flexibility to enable a sufficiently global exploration of the main data storage and transfer search space.

The key point in our approach is the formalized three-layer description in the block description, as illustrated in figure 2.

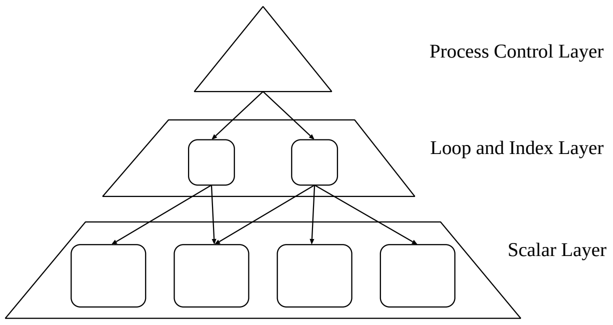

Figure 2. Layered description of reusable designs.

4.1 Scalar layer: structural reuse

The scalar layer entities describe the arithmetic, local control, local storage and communication implementing a scalar operation o. Also data-dependent control constructs (if, while, ...) are described at the scalar layer. Loops on arrays of data are described at the indexing layer.

In a reuse context, the goal is to reuse the scalar operations without any change. This means that the design will be transferred at an optimized sVHDL level. Typically these design blocks will contain an FSMD with a datapath and local control and storage. Thanks to the definition of the scalar layer, reuse "as is" will not have any negative impact on the second objective.

It is important to note that the scalar layer entities defined here will typically be larger than the arithmetic building blocks like adders and multipliers that are used in current designs. Below we show a radix-4 operation in VHDL. The `fpadd`, `fpsub`, and `fpmpy` modules implement floating point addition, subtraction and multiplication. The radix-4 component as it will be reused, will typically be very optimized (not shown here) up to the standard cell level. Optimizations will not only be internal to the individual floating point operations, but might also involve the global radix-4 component.

```
entity fftr4 is
 port (in1r, in1i, in2r, in2i, in3r, in3i, in4r, in4i,
       w1r, w1i, w2r, w2i, w3r, w3i: in  std_logic_vector(..);
       out1r, out1i, out2r, out2i, out3r, out3i, out4r, out4i
                                   : out std_logic_vector(..));
end fftr4;

architecture mod of fftr4 is
[..]
begin
 mpy1:  fpmpy port map (w1r, in2r, m1);
 mpy2:  fpmpy port map (w1i, in2i, m2);
 mpy3:  fpmpy port map (w1r, in2i, m3);
 mpy4:  fpmpy port map (w1i, in2r, m4);
 mpy5:  fpmpy port map (w2r, in3r, m5);
 mpy6:  fpmpy port map (w2i, in3i, m6);
 mpy7:  fpmpy port map (w2r, in3i, m7);
 mpy8:  fpmpy port map (w2i, in3r, m8);
 mpy9:  fpmpy port map (w3r, in4r, m9);
 mpy10: fpmpy port map (w3i, in4i, m10);
 mpy11: fpmpy port map (w3r, in4i, m11);
 mpy12: fpmpy port map (w3i, in4r, m12);
 sum1:  fpsub port map (m1, m2, br);
[..]
 sum19: fpsub port map (apcr, bpdr, out3r);
 sum20: fpsub port map (apci, bpdi, out3i);
end mod;
```

4.2 Indexing layer: behavioral reuse

At the indexing layer, the nested loop operations on multi-dimensional data streams in the multi-media application are described. These large arrays are typically stored in background memory. The optimal organization of the multi-dimensional data flows will depend on the context in which the component is reused. This means that, in contrast to a scalar layer function, no optimized and fixed description at the sVHDL level is desired here. Instead, the description should express the multi-dimensional data-flow present in the algorithm, favoring an algorithmic (functional) description in e.g. C or (high-level) behavioral VHDL (bVHDL).

As a simple example of an indexing layer entity, we show an implementation of the third step in a 512 point radix-2 FFT (taken from the ADSL modem context), written in C:

```
 for (k = 0; k < 256; k++){
   l = ((k<<1)&0x1F8) | (k&0x03);
   l = ((l&0x01)<<8) + ((l&0x02)<<6) + ((l&0x04)<<4) +
       ((l&0x08)<<2) + (l&0x10) +      ((l&0x20)>>2) +
       ((l&0x40)>>4) + ((l&0x80)>>6) + ((l&0x100)>>8);
  FFT2(out[l], out[l+64], in[l], in[l+64], W[(k&0x03)<<6]);
 }
```

`FFT2` is here a scalar layer entity. The one-dimensional iterator is `k` and `l, l+64,` and `(k&0x03)<<6` are the indexing functions.

The most important aspect in a reuse context is that code described at the indexing layer is abstracting enough of the lower-level implementation to heavily simplify the exploration and verification (even without tool support). Moreover, the design effort to synthesize the modified bVHDL to sVHDL and to verify the result, can be largely automated [10].

4.3 Process control layer: behavioral reuse

Control constructs which cannot be statically ordered are outside the scope of the loop nest optimizations in [10]. For this reason, the process control flow is described in a separate layer.

Suppose that in an ADSL application the FFT in the receive path and the IFFT in the transmit path are sharing the radix-*n* hardware. The execution of the receive and transmit path operations are independent and are function of dynamic traffic and channel parameters. This means that both operations are described in separate threads. Apart from instantiating both threads, the process control layer also expresses the resource constraints on the shared hardware units. For most multimedia and telecom kernels, this layer is currently relatively simple still. Therefore, no further details of how to handle complex process control behavior will be given in this paper.

4.4 Reusable component description

A reusable description consists of the combination of the entities at the three layers. Writing designs as a combination of constructs at different abstraction levels implies a relatively limited increase in effort to design the reusable component. This is motivated however by the fact that we obtain a maximal reuse of the design effort, while still guaranteeing a nearly complete exploration of global data and control flow optimizations in the DTSE methodology. The cost advantage obtainable in this way, will be illustrated in section 6 and 7.

Note that the proposed description still provides a bit-true executable model of the design. This is important to validate the transformations and to verify the integration in a reuse context.

4.5 Obtaining the reusable component description

The starting point can be either a functional or a structural description, the first when designing for reuse, the latter when redesigning for reuse. The first step is the identification of scalar layer entity boundaries. This step is based on the definition in 3.1, requiring scalar input and output streams. Each scalar layer entity is then mapped into an internally optimized structural description with scalar data stream interface.

It is important to remark here that in order to increase even further the amount of reuse in the design, an experienced designer could decide to structurally reuse parts of the design that do not belong to the scalar layer. For example, small loops and array structures can be integrated in a scalar layer entity, if the potential loop and indexing layer optimization gain would be limited. Such diversions of the formalized layering, trading off exploration freedom for reuse of design effort, will be based on thorough knowledge of the design and constitutes part of the added value of the IP block.

In the second step, the boundaries of the indexing layer entities are defined. Control constructs that are not loops belong to the process control layer. The reuse vs. exploration freedom trade-off may legitimate however to move some process control constructs to the scalar layer, e.g. an external parameter steering a global condition which does not remove any opportunities in the data transfer and storage exploration when moved to the scalar layer. For example the selection between two different filtering operations, initially implemented as two separate indexing layer entities called from the process control layer, could be written as a single indexing layer entity where the calls to the scalar layer entities now potentially have an extra parameter to indicate the operating mode.

Once the boundaries between indexing and process control layer are identified, each indexing layer entity is mapped into a behavioral description, instantiating scalar layer entities and the process control layer is mapped to a behavioral description, instantiating indexing layer entities.

5 Overall reuse methodology

Figure 3 illustrates the design steps starting from the proposed description and leading towards a netlist.

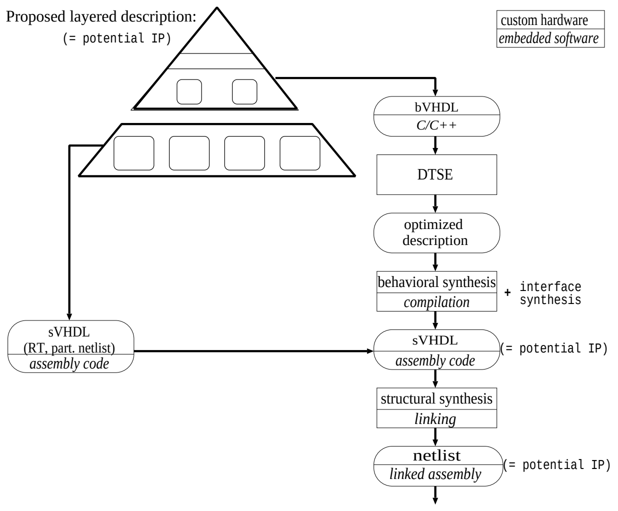

Figure 3. Reuse design flow, identifying potential points of design transfer.

At reuse time, the loop and indexing layer descriptions are used as input for a system-level DTSE approach [2, 16], delivering an optimized data transfer and storage solution in the actual implementation context. The transformed behavioral description of the indexing and process control layer is mapped to an RT description using existing behavioral synthesis tools, instantiating the scalar layer entities. Interface synthesis is used to generate the interface between hardware and software blocks. After structural synthesis, this results in a global netlist description of the IP block or in linked assembly code in the software case.

Depending on the applied reuse business model, one can either provide the layered description as IP block, allowing the IP integrator to optimize the IP block in their design flow and application context. Alternatively, the IP provider can perform the optimizations, providing a very cost-effective customer-specific solution at a reduced engineering cost for many customers. In this case either sVHDL or a full netlist can be produced.

6 Demonstration on ADSL modem

In this section we discuss the data transfer and storage exploration in an Asymmetric Digital Subscriber Loop (ADSL) modem built with reusable components using the proposed methodology. It is not our goal to present an optimal topology but to demonstrate that our methodology enables a complete exploration of the relevant design freedom in data-dominated applications. It will be shown that an optimal solution depends strongly on the interface to the neighboring components and on the implementation technology, since this will change the cost ratios between large and small memories (caches), logic, and interconnect. This justifies the presented reuse methodology, exposing the flexibility in the data transfer and storage organization and reusing unrelated logic optimizations. The formalized model will be illustrated on several components of the ADSL application.

Figure 4 shows part of an ADSL modem [15]. After the FFT a frequency domain equalizing (FEQ) and a rotor operation are applied to cope with the physical medium impairments and the sender–receiver frequency mismatch. The filter coefficients are updated dynamically during operation and are calculated in software.

In figure 4b, we show the three layer description of the FFT-FEQ-rotor part, which consists of scalar layer entities for the radix-2 FFT and the complex product operation. At the indexing layer we have the description of the indexing for the different FFT-stages implementing an in-place butterfly and for the FEQ- and ROTOR-stage. The system layer aspects are implemented in software.

With a 9-steps radix-2 implementation in a .35 μm reference ASIC technology of the FFT, the central RAM accesses consume around 200 mW. This is a large power contribution compared to the FFT arithmetic operations, which amount to around 57 mW.

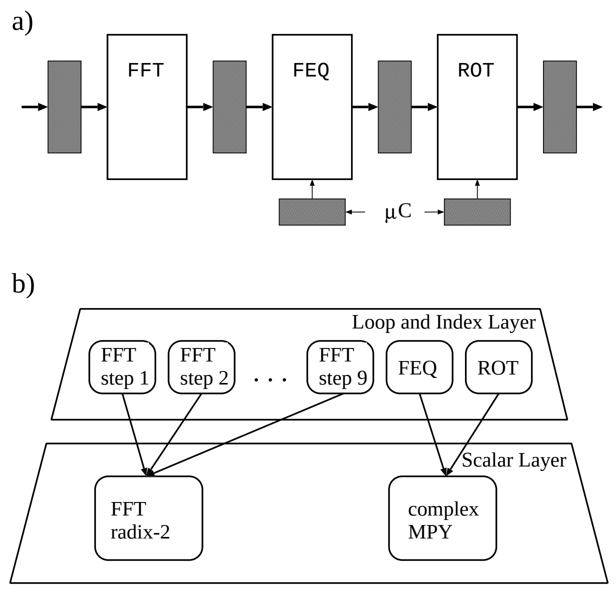

Figure 4. ADSL FFT-FEQ-ROT a) block diagram, b) 3-layer description.

A first alternative replaces the radix-2 stages with radix-4 stages. This is an algorithmic transformation at the indexing layer, where a new indexing layer entity for the radix-4 FFT replaces two radix-2 stages (optimized radix-*n* modules are assumed to be available in a reuse library). Applying this substitution four times reduces the power for the accesses to the central RAM to 115 mW. The arithmetic power decreases with 16% (see Fig. 5). In a complete exploration of the FFT, one has to look also at higher radix configurations, giving an additional central RAM and arithmetic power gain. This gain comes at a cost of extra design effort and a much larger area (also increasing the interconnect power), so a clear trade-off is involved here. Another strategy to explore is the reorganization of the butterfly computation to increase data locality [22], making the usage of a data cache more power efficient.

	FFT logic	*storage*
9*radix-2 – FEQ – ROT	57 mW	199 mW
4*radix-4 – radix-2 – FEQ – ROT	48 mW	115 mW
4*radix-4 – (radix-2,FEQ,ROT)	48 mW	94 mW

Figure 5. Power savings in an ADSL FFT-FEQ-ROT resulting from DTSE exploration.

A second alternative involves a better interfacing of the FFT component with the neighboring components. We observe that loop merging can be applied between the last stage of the FFT and the FEQ and rotor block, bypassing intermediate RAM accesses. Performing this optimization at reuse time transforms the indexing layer entities, while the scalar layer entities remain unchanged. If this optimization would have been identified in the conventional approach, a full redesign of the hardware IP block would have been necessary. With our methodology, it is possible to reuse most of the design (at sVHDL level) while only modifying the appropriate loops and indexed signal accesses. Another 18% power reduction results from this optimization which can be obtained with limited redesign effort.

During the exploration also the applicability on reusable software components was substantiated. The filter coefficients for the adaptive FEQ are calculated in software. In an optimized design, the coefficient update is synchronized with the filter operation, avoiding a double buffering. If the software is described according to the presented methodology, i.e. with a heavily optimized (partly assembly level) code at the scalar layer and only the loops accessing the indexed signals in high-level C code at the indexing layer. Then, generating the coefficients in the desired order is only a question of changing the last loop. This way, the large intermediate buffer will be avoided resulting in a significant area saving (in the order of 1 mm^2 for a .35 μm reference technology).

With this example it is shown for a realistic demonstrator how the methodology allows to make trade-offs at reuse time over the boundaries of predefined IP components in both hardware and software. Counted in number of final gates, the amount of reuse was well above 90%.

7 Other demonstrators

In [27] a two-dimensional IDCT as found in MPEG decoding has been presented as a component to be reused. The indexing layer in this example consists of two loops over 1-D IDCT operations. Applying the proposed reuse methodology allowed to avoid buffering at the interface in many realistic configurations where conventional approaches would insert buffers. The total power and are savings were of the same order as the cost of the datapath.

In other experiments, larger and even more data-dominated systems have been studied on which a global exploration was performed. The results indicate that in these cases an even larger cost gain can be expected. Aggressive optimization of a complete H.263 video conferencing decoder has shown a reduction of the peak memory power consumption by a factor 9 compared to a conventional implementation [7]. In a parallel processor implementation of a QSDPCM (Quadtree Structured Difference Pulse Code Modulation) video codec, a global area and power optimization step early in the system design cycle reduced area and power costs by orders of magnitude [17]. In the context of software development for embedded processors, a system level data transfer and storage exploration on an MPEG-2 decoder has resulted in an average factor of 3.7 in power reduction [6].

Using the proposed formalism and methodology, the same exploration is still possible in a reuse based system design for these application domains. The proposed three-layer formalism has actually been successfully applied to several of these designs already.

8 Conclusion

We have presented the formalism behind a systematic power optimizing design-for-reuse methodology for soft and firm IP blocks in data-dominated applications. This methodology makes it possible to minimize data transfer and storage related power and area when reusable IP blocks are put into another context than what they were designed for, by applying a global exploration of data storage and transfer. The extra design-for-reuse effort incurred by the designer is minimal, whereas experiments on realistic drivers in several application domains have shown significant savings in power and area.

Further research is necessary to identify and formalize the constraints from the process control layer which drive the task level parallelism exploration (as in [17]).

Acknowledgments

We thank S.Janssens and L.Nachtergaele for their advice.

References

[1] R.W.Brodersen, "The network computer and its future," *Proc. IEEE Int. Solid-State Circuits Conf.*, San Francisco, CA, pp.32-36, Feb. 1997.

[2] F.Catthoor, F.Franssen, S.Wuytack, L.Nachtergaele, H.De Man, "Global communication and memory optimizing transformations for low power signal processing systems," *VLSI Signal Processing VII*, J.Rabaey, P.Chau, J.Eldon (eds.), IEEE Press, New York, pp.178-187, 1994.

[3] A.Chandrakasan, M.Potkonjak, R.Mehra, J.Rabaey, R.W.Brodersen, "Optimizing power using transformations," *IEEE Trans. on Comp.-Aided Design*, Vol.CAD-14, No.1, pp.12-30, Jan. 1995.

[4] "Low power CMOS design," (eds. A.Chandrakasan, R.Brodersen), *IEEE Press*, 1998.

[5] T.Meng, B.Gordon, E.Tsern, A.Hung, "Portable video-on-demand in wireless communication," special issue on "Low power electronics" of the *Proc. of the IEEE*, Vol.83, No.4, pp.659-680, Apr. 1995.

[6] D.Moolenaar, L.Nachtergaele, F.Catthoor, H.De Man, "System-level power exploration for MPEG-2 decoder on embedded cores : a systematic approach," *Proc. IEEE Wsh. on Signal Processing Systems*, Leicester, UK, pp.395-404, Nov. 1997.

[7] L.Nachtergaele, F.Catthoor, B.Kapoor, D.Moolenaar, S.Janssens, "Low power storage exploration for H.263 video decoder," *VLSI Signal Processing IX*, W.Burleson, K.Konstantinides, T.Meng, (eds.).

[8] D.Singh, J.Rabaey, M.Pedram, F.Catthoor, S.Rajgopal, N.Sehgal, T.Mozdzen, "Power conscious CAD tools and methodologies: a perspective," special issue on "Low power electronics" of the *Proceedings of the IEEE*, Vol.83, No.4, pp.570-594, April 1995.

[9] V.Tiwari, S.Malik, A.Wolfe, M.Lee, "Instruction-level power analysis and optimization of software," *Journal of VLSI Signal Processing*, No.13, Kluwer, Boston, pp.223-238, 1996.

[10] F.Catthoor, S.Wuytack, E.De Greef, F.Balasa, L.Nachtergaele, A.Vandecappelle, "Custom Memory Management Methodology – Exploration of Memory Organisation for Embedded Multimedia System Design", ISBN 0-7923-8288-9, Kluwer Acad. Publ., Boston, 1998.

[11] J.Öberg, A.Kumar, A.Jantsch, "An Object-Oriented Concept for Intelligent Library Functions," *Proc. VLSI Design '98 Conf.*, Chenai, India, pp. 355-358, Jan. 1998.

[12] M.Koegst, P.Conradi, D.Garte, M.Wahl, "A Systematic Analysis of Reuse Strategies for Design of Electronic Circuits," *Proc. DATE '98 Conf.*, Paris, France, pp. 292-296, Feb. 1998.

[13] R.Gupta, Y.Zorian, "Introducing Core-Based System Design," *IEEE Design & Test of Computers*,Vol. 14, No. 4, pp. 15-25, Oct. 1997.

[14] VSI Alliance, "VSI Alliance Architecture Document," *VSI Alliance*, 1.0 edition, 1997.

[15] ANSI, "Asymmetric Digital Subscriber Loop Standard," *ANSI T1E1.4 Committee*, Contribution 95-007, June 1995.

[16] F.Catthoor, S.Wuytack, E.De Greef, F.Franssen, L.Nachtergaele, H.De Man, "System-level transformations for low power data transfer and storage," in paper collection on "Low power CMOS design" (eds. A.Chandrakasan, R.Brodersen), *IEEE Press*, pp.609-618, 1998.

[17] K.Danckaert, K.Masselos, F.Catthoor, H.De Man, C.Goutis, "Strategy for Power Efficient Design of Parallel Systems," *IEEE Trans. on VLSI Systems*, Vol.7, No.2, June 1999.

[18] F.Thoen, J.Van Der Steen, G. De Jong, G.Goossens, H.De Man, "Multi-Thread Graph - A System Model for Real-Time Embedded Software Synthesis" *Proc. European Design & Test Conf*, pp. 476-481, Paris, France, March 1997.

[19] E.Lee, A.Sangiovanni-Vincentelli, "A framework for comparing models of computation," *IEEE Trans. on CAD of ICs & Systems*, Vol.17, No.12, pp.1217-29, Dec. 1998.

[20] F.Franssen, F.Balasa, M.van Swaaij,F.Catthoor,H.De Man, "Modeling Multidimensional Data and Control Flow" *IEEE Trans. on VLSI Systems*, Vol.1, No.3, Sept. 1993.

[21] E.De Greef, "Storage Size Reduction for Multimedia Applications" *PhD. Thesis — K.U.Leuven*, Jan. 1998.

[22] B.Baas, "An Energy-Efficient Single-Chip FFT Processor" *IEEE Symp. on VLSI*, Honolulu, HI, June 1996.

[23] D.Lanneer, "Design Models and Data-path Mapping for Signal Processing Architectures" *PhD. Thesis — K.U.Leuven*, Mar. 1993.

[24] O.Bringmann, W.Rosenstiel, "Resource Sharing in Hierarchical Synthesis" *Int. Conf on CAD*, pp. 318-325, San Jose, CA, Nov. 1997.

[25] D.Rao, F.Kurdahi, "Hierarchical Design Space Exploration for a Class of Digital Systems" *IEEE Trans. on VLSI Systems*, pp. 282-295, Vol.1, No.3, Sep. 1993.

[26] A.Jerraya, H.Ding, P.Kission, M.Rahmouni, "Behavioral Synthesis and Component Reuse with VHDL" *Kluwer Academic Publishers*, 1997.

[27] F.Vermeulen, F.Catthoor, D.Verkest, H.De Man, "A System-Level Reuse Methodology for Embedded Data-Dominated Applications" *IEEE Wsh. on Signal Processing Systems*, pp.551-560, Boston, MA, Oct. 1998.