Proposal for unified system design meta flow in task-level and instruction-level design technology research for multi-media applications

Francky Catthoor, Diederik Verkest, Erik Brockmeyer

Abstract

This paper describes an attempt to bring together the many different system design flows existing in architecture and system design technology research, into a more abstract but unifying meta flow. Many existing system and architecture design flows have a strong resemblance and unnecessary overlap. Mainly due to a lack of a common and consistent terminology coupled to a common reference basis, it is now nearly impossible to compare and reuse (sub)steps. In addition, there is a too strong separation between research in different communities. To alleviate this problem, we introduce a more abstract but unifying meta flow which attempts to bridge the gap between the existing flows. From this meta flow, a particular design flow can be instantiated for a given application (domain) by leaving out the non-required stages/steps, by selecting a (sub)step sequence which is compatible with the partial meta-flow order, and by selecting the appropriate technique for all remaining (sub)steps (e.g. the type of scheduler). This paper focuses on the principles at the task- and instruction-level abstractions. It also provides an illustration of the power of the meta-flow principles for a realistic multi-media compression demonstrator from the MPEG4 context.

1 Motivation and objectives

Many different design flows (or partial flows) exist in the system and architecture design literature (see e.g. [2, 3, 8]), in many cases intended also for different application domains. Many of these flows have resemblances but it is usually difficult to pin-point where the corresponding parts are and where the different (sub)steps are situated. This is for a large part due to incompatible terminology and unnecessary details or lack of unambiguous information which obscure the global picture. In order to allow more “reuse” of knowledge and information, it is vital to overcome this problem. For this purpose, a common terminology basis is a first step. The basic high-level synthesis terminology proposed in [11, 12] was useful but not sufficient to cover the rapidly advancing research. Moreover, in order to really use a common terminology there is also a need to have a shared view of an underlying system design meta flow. Recently, the main trust in the design technology community has moved into the “system” and “software” areas and the underlying system design flow basis is largely lacking still.

At IMEC, an attempt has been made to remedy this situation. The resulting proposal is (to our knowledge) the first attempt to obtain unification over such a broad scope. Part of the results are summarized in this paper, focused on the task-level and instruction-level abstractions and illustrated by applications from the multi-media domain. We believe that the scope of validity of the full document contains all digital electronic systems. The targeted application domains are very broad but the main focus is on end-user telecom, multi-media processing, telecom protocols and telecom network components. Heterogeneous system architectures (“systems-on-a-chip”) are necessary to meet the system requirements for these application domains, encompassing instruction-set (programmable) and customized processors but also storage, control and interface components (see figure 1).

Figure 1. Heterogeneous embedded system realisation

For a particular application (domain), not all (sub)steps in the proposed flow are important. In fact, it represents a meta flow which is to be instantiated by omitting those (sub)steps which are irrelevant from the viewpoint of the specific type of application to be de-
signed (see subsection 2.1). The global unified meta flow presents a single overall framework which is surprisingly simple but certainly not trivial. The strict separation between different abstraction levels allows the designer to deal with the issues on each level more clearly and in a more decoupled fashion. Moreover, it enables providing very early exploration feedback, avoiding large scope iterations through the entire design process. Also the historical distinction between “hardware” and “software” is discarded, thereby removing the artificial and inefficient boundaries between much existing work. This meta flow attempts to meet the following objectives:

1. Achieve a common terminology, as stated above.
2. Simplify identification of related work and comparison.

In principle, a single common database of references related to each of the “hooks” in the unified flow can be envisioned. When someone is interested in a (sub)step, e.g. to reuse the approach, all related info is then easily accessible.

3. Establish framework to identify “holes” in a given (new) flow. In most application domains there will be many (sub)steps in the unified flow which are not relevant or which can be merged with other (sub)steps. However, when this issue is incorporated, all the remaining (sub)steps should be somehow present in the final solution and hence it is better to take this into account from the start of the system design support project.

4. Remove redundancy and unnecessary overlap. This stimulates knowledge reuse and in the end also software reuse.

2 Short overview and coherence issues

2.1 Characteristics

The characteristics of the proposed meta flow are:

1. It represents a meta-flow from which many instantiated design flows can be derived.
2. It is not a single detailed design flow focused on a specific target application domain. So even given this unified flow, the amount of research effort to arrive at a customized and efficient flow for a given target domain is still large. The coupling between efficient synthesis support and target applications still remains valid. Moreover, depending on the type of application domain which is considered, a specific step in the meta flow will be instantiated in a partly different way.

3. In principle, it is independent of the type of system design, as long as one remains within an electronic context.
4. The dependencies between the (sub)steps should be obeyed in any “logical” instantiated flow, but it is always allowed to group (sub)steps for simultaneous execution, and to reorder any pair of (sub)steps which does not exhibit a dependence (just like many schedules/allocations can be derived from a single data-flow graph).
5. It is definitely not a tool flow because then also the CAD algorithms should be incorporated and these are even more target domain specific.
6. It also does not incorporate design data models (e.g. data-flow vs control-flow) because that is not necessary and would hamper the unification. This abstraction of the underlying data model is a crucial enabling issue for deriving such a meta-flow.

![Figure 2. Basic structure of the unified meta flow](image)

2.2 Global view and principles

A global view on the proposal is provided in figure 2. In this unified flow, we can identify a number of abstraction levels, which each group their respective (sub)steps. The major steps in the system design trajectory are related to the algorithm design, the task-level concurrency, the data parallelisation into regular arrays, the instruction-level concurrency and the circuit-level concurrency. At each concurrency level, both the data transfer and storage issues (in the data transfer and
storage exploitation (DTSE) steps) and the concurrency management issues are addressed\(^2\). The DTSE steps are all related to the handling of complex data types such as indexed signals (arrays) or dynamically created lists (tables) (see figure 3). It involves global data analysis (for dependencies, accesses and memory size estimates), global data- and control-flow transformations (to remove redundancy and data-flow bottleneck respectively to increase access regularity and locality), data reuse decisions (related to the exploitation of memory hierarchy), storage cycle budget distribution (to meet timing constraints), memory allocation and assignment (for background memories) and data layout organisation issues (e.g. in-place mapping to reuse data space for arrays with a limited, partially overlapping life-time; or selecting other data organisation parameters in the cache and main memory).

In contrast, the concurrency management steps are related to concurrency extraction, concurrent execution (within a time budget), the synchronisation (ordering), the resource allocation (within a time budget) and the interface refinement between the concurrent entities (figure 4). It has to be stressed that the DTSE stages are always ordered before their concurrency stage counterpart at the 3 main abstraction levels. This is necessary because they provide vital information on the communication ordering constraints and costs issues related to the background memory storage/transfer of the most important internal signals in each task. Reversing the order would also mean that the concurrency stage provides too strong restrictions on the data communication ordering, which would be inconsistent with the reason for having a separate DTSE stage [4, 5]. If the considered application (module) is not data-dominated enough, the DTSE stages should be left out (see also below).

This issue is nicely illustrated in our exploration of the entire video decoder algorithm in a real-life H.263 video conferencing decoder [10]. We have globally transformed the data transfer scheme in the initial public domain system specification and have optimized the distributed memory organization. For the worst-case mode using Predicted and Bi-directional (PB) frames, the maximum memory power consumption is significantly reduced by a factor of 9, by combining instances of nearly all DTSE steps in figure 3.

The contributions of the data-path, control and address generation in the complete (custom) architecture realisation have been shown to be much less critical for this data-dominated design. Typically, one order of magnitude (or even more) difference is present between the non-optimized background memory and global bus communication cost and the non-optimized logic related contributions in such data-dominated (parts of the) applications. As a result, it is well motivated to apply the DTSE stage prior to the concurrency management stage at the instruction/operation level. Experiments at the task abstraction level (see section 3) have confirmed that the same principle holds at that level too.

Several propagation modes and paths exist through the flow of figure 2. By ordering all the (sub)steps in the meta flow in a meaningful order, the information generated in them refines the system model at each level, while maintaining the right amount of abstraction, i.e. enough information to deal with the issues at that level without unnecessary details. This propagation of constraints through the meta flow links all the (sub)steps together in a coherent unified framework\(^3\). Because of the nature of this unified meta flow, no redundancy is really present between the objective of each (sub)step; so the combination of all of these constraints constitutes the final system architecture.

Note that besides the top-down constraint propagation mode outlined above, it is also likely that a designer has to deal with bottom-up architecture constraint propagation, e.g. when the decision to use a certain processor is made in advance. This will most likely lead to subop-

\(^2\)No DTSE takes place at the circuit level, since there is no concept of “data”

\(^3\)These constraints are a key issue in the meta flow but the detailed list cannot be provided here.
timal results from a global cost viewpoint though.

The amount of iteration will be typically (heavily) reduced by applying the flow of the (sub)steps in the proposed order (as demonstrated by the many existing successful flows from which this unified one has been abstracted).

2.3 Overview of main steps

A rundown of the five “stages” (each addressing a major abstraction level) in the meta flow is given now with emphasis on the task- and instruction-level abstractions. The other levels will be addressed in future papers. The idea is to briefly introduce the issues that are dealt with in the different steps when we open some of the boxes in figure 2. It has to be stressed that several extra layers of refinement are present in the complete flow but due to lack of space we cannot provide a more detailed and formal explanation here.

1. **algorithm level**: Early stages in a design flow concentrate on generating a correct functional specification of the design. In order to have a complete and formal description of the system, it is necessary to describe the functionality in a specification language, which can be (but need not be) executable. For real-time systems, however, it is not only important to generate the correct results, but also to generate them on the right time instances. Therefore, the specification model must be able to deal with time and *temporal correctness* as well.

Next, data type refinement has to take place which has a very crucial impact on the system cost. This stage will however not be expanded in more detail here.

2. **task-level**: After the first stage, the designer ends up with an optimized system specification, typically consisting of a number of *concurrent communicating tasks*, annotated with a number of timing requirements which define the boundaries for temporal correctness. Each of these tasks can contain one or more threads of control. The purpose of *task concurrency management* is determining an optimal constraint-driven scheduling, allocation and assignment of the various threads and tasks to one or more abstract processors, which on this level are called *array processors*.

The main issues on this level are:
- **inter-task DTSE**: The goal is here to minimize the communication and storage requirement among the tasks (see figure 3). IMEC’s ACROPOLIS system-level data transfer and storage exploration research project [1] is e.g., focusing on this stage.
- **task concurrency extraction**: The first thing that has to be done is the analysis of the opportunities for task-level parallelism in the system. The limit imposed by inter-task data dependencies are examined in this part of the meta flow. Also transformations can be applied to arrive at a better “task” partitioning.
- **task scheduling**: The various tasks are (partially) ordered and a relative time is decided for their interaction.
- **thread scheduling**: In case a task with multiple threads of control is assigned to one array processor, an additional scheduling step takes place to decide the start time of each thread in the task (statically or dynamically).
- **array-processor allocation**: The designer makes a decision on the number of array-processors that will be used. A balanced choice driven by real-time constraints and maximum available parallelism should lead to a minimal cost allocation.
- **task to array-processor assignment**: Here, tasks are assigned to one or more array-processors. The threads contained in a task are shared on the array-processor it is assigned to.
- **inter-task interface refinement**: The interfacing requirements of communicating tasks are gradually refined in terms of data type conversion, scalar buffering including storage units and interconnect, and synchronisation refinement.

3. **array level**: A task-level analysis only considers a limited amount of the potential parallelism of an application as the number of tasks and threads is usually rather small. In a practical design flow, many applications require that another level of concurrency is exploited, namely *data parallelism*. This level will not be discussed further in this paper.

Every array processor can be further refined into a structure containing a number of *heterogeneous processors*, each consisting of one or more *homogeneous processors* (shown on the left of the instruction-level stage on figure 2). A homogeneous processor is characterized by only one architectural style, namely a type of custom or instruction-set processor (bit-serial, highly multiplexed, ...). In figure 2, the grid represents an array processor containing a number of these heterogeneous processors (the nodes in the array matrix).

4. **instruction-level**: Here, the instruction-level parallelism issues are decided. Note that the custom processor synthesis and instruction-set processor mapping steps also synthesize the detailed interface descriptions (up to...
the RT-level) from the higher-level behaviour which has
been specified during inter-task and inter-(homogeneous)
processor interface refinement. Note also that this ab-
straction level is the minimal one which needs to be ad-
tressed for any realistic application. If no concurrent
tasks are present and the application does not demand
for an array-processor based realisation style, everything
can be reduced to this stage in the instantiated meta-
flow. The latter situation is still true for most present-
day applications in industry. Which DTSE stages (if
any) are applied also depends heavily on the given ap-
application (domain). The main issues on this level are:

- **processor-level DTSE:** Storage and transfer exploration
  inside the scope of the heterogeneous processor (see
  figure 3). This stage has been the focus of e.g. IMEC’s
  “custom processor”-oriented ATOMIUM DTSE methodology [4, 5],
  Philips’ Phideo scheduling and memory management [9],
  and Irvine memory management [14, 13].

- **processor architecture integration:** Here, the internal
  structure of the heterogeneous processor is decided in
  terms of homogeneous processors. An allocation of
  custom and instruction-set processors is made and the
  thread(s) to be executed on this heterogeneous processor
  is statically scheduled on it. Also the internal interfacing
  “glue” between the processors is generated during this
  step.

- **high-level address optimisation:** The mapping of multi-
dimensional signals onto a one-dimensional memory
structure is optimized. It precedes both instruction-set
and custom processor mapping, since indexed signals

5 Many applications can indeed be directly written as a single
task, interacting with the environment, and array-processors are
only needed for very demanding applications (for the time being).
For a large part, this is however a decision of the system designer.
Future complex applications are being (and will more and more be)
specified as concurrent systems. The task-level stage is then vital
to deal with the added design complexity and to fully exploit the
design freedom which becomes available by adding this abstraction
level. Note also that in principle, the concurrency extraction step
at the task-level could be applied on an initial specification with a
single task, leading to a concurrent (transformed) description
for which the task concurrency management stage does become
relevant. Likewise, all concurrent tasks which are present in a
given specification could also be expanded into a single task if the
designer decides to skip the task-level stages from the meta flow.

6 If the data types in a given application (domain) are not com-
plex enough and hence are not dominating the impact of the con-
currency issues, it is best to skip the DTSE stages at the higher
abstraction levels in the meta flow. The meta flow instantiation
is then restricted to the processor-level DTSE stage for that
application (domain). If the cost associated with the most complex
data types is not dominant over the (scalar) foreground signals,
then even that processor-level DTSE stage should be skipped and
all storage related issues have to be dealt with during the fore-
ground memory management stage in the custom processor syn-
thesis and/or instruction-set processor mapping stages. The latter
are always needed, as any realistic application deals with at least
some data type instances.

can be handled in both of them. While it is true that
most compilers generate these mappings automatically,
they never explore the actually available search space.
High-level address optimisation lets the designer tune
and optimize this as a preprocessing step.

- **custom processor synthesis:** For the relevant parts of the
  heterogeneous processors.

- **instruction-set processor mapping:** Compilation and link-
ing is performed for the relevant parts of the heteroge-
eneous processors. Optionally, the instruction-processor
  itself and its companion compiler are synthesized during
  this step.

5. **circuit level:** The circuit-level concurrency manage-
ment stage maps the design on its target technology plat-
form, which can e.g. be a reconfigurable (e.g. FPGA)
target or a custom (e.g. ASIC) one.

A distinction is made between target technology plat-
form and target architectural style (for example selecting
a programmable or custom processor). These choices
are made at two very different places in the meta flow.
Abstraction is made of the terms “software” and “hard-
ware” which have a connotation which does not corre-
spond to an orthogonal choice at this stage. Instead
the terms custom processor synthesis and instruction-set
processor mapping are used. This separation between
the architecture level choice to either a custom or an
instruction-set processor style and the technology plat-
form selection allows all 4 combinations to be used in the
instantiated design flow. The conventionally used solu-
tions namely exclude the combination of an instruction-
set processor style mapped on a reconfigurable tech-
ology platform. Only recently, this combination is begin-
ing to be discussed and the meta flow clearly shows the
feasible (sub)choices and the restrictions imposed by the
dependencies between the substeps.

3 Illustration on MPEG4 test-vehicle

The recent MPEG-4 standard [15] is a key to multi-
media applications. The exploration below has been
performed on the MoMuSys Video VM Version 7.0. It
involves complex data-dominant algorithms. A custom
hardware or even instruction-set processor realization of
such a (de)coder has to be power efficient in order to
reduce the size of the chip packages (where it is embed-
del) or the battery (if used in a mobile application).

Our previous research shows clearly the dominant power
contribution of data transfer and storage of M-D (multi-
dimensional) array signals and other complex data types
in data-dominated designs [4, 10]. In this section, we will

7 Usually, one refers to an instruction set processor mapped
onto an application-specific IC technology as a “software” pro-
grammable processor.
show that by a careful use of the meta-flow principles (especially the split in a task- and instruction-level abstraction), it is feasible to exploit this feature to achieve large savings in the system power of a crucial part of MPEG-4, without sacrificing on the performance or on the system latency and without spending a heavily increased design time.

During our exploration we have focused only on the most dominant part of the MPEG-4 video encoder. Profiling experiments have shown that 67% of the original accesses take place for the luminance pixels in motion estimation (full-pel and half-pel), which we will focus on in this paper. The primary design goal has been to reduce memory transfers between large frame memories and datapath units. The cost of a data transfer is a function of the memory size, memory type, and especially the access frequency $F_{real}$. An accurate power model, derived from a VTI data sheet, is used for the exploration.

We have focused only on the most promising code transformation steps in our methodology [5]. By analyzing the loop structures and large signals (see also figure 5), the data locality of the algorithm implementation can be improved compared to the original sequence [15]. First the entire P-VOP is coded followed by the B-VOPs sequentially. Because of this implementation, the search area window will go M times through the previous VOP and (M-1) times through the next VOP. By combining the different (concurrently specifiable) motion estimation tasks which take place on the same VOP and position (using loop transformations at the task-level, according to our methodology [6]), a significant gain in the order of M reads from the VOP memory can potentially be achieved [5]. The actual choice depends also on the shape and size of the VOP and the group size (M), so it is parametrized in the code.

These task-level transformed algorithm codes are now passed to the data reuse decision step [7] (see also figure 3). By adding intermediate memories, the data can be kept closer to the datapath. This is related to traditional caching policies, but our approach is much more aggressive and global. In our systematic multi-step methodology several issues can be decoupled, allowing the potential reuse to be (more extensively) explored without exploding the design complexity. At the task-level abstraction, only the main signals shared or communicated between the interacting tasks are considered. Moreover, only the first levels of data reuse are explored because the lower levels involve too small signal copies which will have to be decided on during the more detailed processor-level DTSE stage.

Similar steps, applying our processor-level DTSE methodology have been performed for each of the distinct tasks in the motion estimation code. We will not provide the details here. The end result is shown in figure 6. The left hand side steps belong to the task-level stage where only the main VOP signal access was considered. A reduction by a factor 65 in background luminance pixel reads has been obtained this way. Experiments have however confirmed that after this substantial saving on the VOP access, significant power is also related to the “second order” signals which are only visible when the abstraction level is expanded to the processor-level. The right-hand side steps in figure 6 show that our processor-level DTSE code transformations allow to reduce also these contributions to a fraction of the original contribution. A substantial gain in global memory related average access power of a factor 13 is reached. This has been verified by simulation for the first 96 frames of the “hair” sequence in CIF resolution.

**Figure 5. Illustration of MPEG-4 video object plane (VOP) sequence for content based object coding. The arrows represent all motion estimation steps for one group of VOPs.**

**Figure 6. Total relative power for VOP memory and search area memories during application of DTSE stages**

The overall advantages of this approach are that: 1. The task-level DTSE stage can have a truly global scope because only the most important inter-task signals have to be considered. The processor-level DTSE stage
can focus in full detail on each task separately without having to consider the task interaction any longer. These two facts result in significant savings on design time as experienced in our exploration of the MPEG4 motion estimation routines.

2. In many applications, the concurrent tasks exhibit asynchronous process control interactions which prohibit the expansion of the full code into a single statically analyzable piece of code. In these cases, the detailed processor-level analysis and optimizations do not make sense beyond the boundaries of such tasks. In contrast, our experiments have shown that the coarser view of the task-level DTSE stage still allows to perform very relevant optimizations across the task boundaries.

The exploration space to optimize data storage and transfers is very large, and our experiments clearly show the importance of a formalized methodology to traverse it. It also demonstrates again that the DTSE optimizations have to be performed aggressively before the multimedia algorithms are realized on custom or instruction-set processor targets in the instruction-level concurrency management stage.

4 Conclusions

It appears to be feasible to unify all known/published system design flows in a digital electronic system design context in a single overall “framework” which is surprisingly simple but which is still not trivial. The general applicability of the proposed meta flow has been analyzed and verified in many different domains during the last year. In this paper, it is impossible to “prove” this generality however due to lack of space.

Apart from meeting the objectives stated in the abstract, we believe it can also be an enabler to bring together researchers from widely different communities. For instance, the historical distinction between “hardware” and “software” is leading to artificial and highly inefficient boundaries between much existing work. Our proposal largely removes this border. We also believe that the separation between different abstraction levels: algorithm, task concurrency, data parallelism, processor concurrency, custom vs instruction-set processor and technology platform (circuit level) – as motivated in our proposal – allows a clearer, more design-efficient, and more decoupled view on system design issues. This has been demonstrated successfully in the MPEG4 testvehicle design discussed above. All constructive feedback on this proposal is welcome to the authors.

Acknowledgements: The proposal in this document is based on the background of working 15 years in the VSDM architecture and system synthesis context and hence the ideas are indirectly influenced by many people who have worked here now and in the past in the design technology domains. Also interesting discussions with colleagues in other research groups, both academic and industrial, have influenced the final result.

References

[1] Multi-media compilation project (Acropolis) at IMEC http://www imec be/vsdm/domen/s/ design techm/Welcome html