# The Data Flow Graph in the ASCIS project

## Deliverable CD/m30/A1/1

Jens P. Brage,
Jos T.J. van Eijndhoven,
Kevin O'Brien,
Peter Poechmüller

Febr 25, 1992

# Chapter 1

# Introduction

The ASCIS consortium has decided to use a common format for the exchange of designs, on the level of data flow graphs. The obvious advantages of a common format are possible sharing of each others software, and exchange of actual design data. Furthermore it allows comparisons to be made between different synthesis approaches in the project, an important aspect in basic research.

The exchange format is positioned at the semantical level of data flow graphs for the following reasons:

- Data flow graphs are a suitable starting point for architectural synthesis, since they still allow maximal freedom in exploiting area/time tradeoffs, and do not impose real restrictions towards different different design styles.

- Data flow graphs are in general not considered desirable directly as user (designer) interface. Hence other dedicated systems are used for the designer interface (VHDL, Hardware C, Silage, ...). To start architectural synthesis, an initial data flow analysis is always required. By standardising on the result of this analysis, the different input methods become available for the different synthesis projects.

- Data flow graphs are semantically clean and simple, unlike many designer oriented specifications, thus forming an unambiguous behavioural definition suitable to interface to synthesis packages as well as formal verification.

As appendix an overview on the data flow graph's intended use, semantics, and syntax is included as preprint of a paper appearing at the European Design Automation Conference, 16-19 March 1992, Brussels.

The following sections are contributions from the ACIS partners in Eindhoven, Darmstadt, Lyngby, and Grenoble, on their efforts towards the DFG.

# Chapter 2

# The ASCIS DFG

## 2.1 The creation of the standard

The data flow graph standard as used in the ASCIS project was designed to combine a set of features, which make it worldwide unique:

- The data flow graphs are allowed to contain conditional execution as well as loop. constructs. The token flow semantics is responsible for a consise behavioural model, without the need for additional external control information.
- Having conditionals and loops uniformly contained in the data flow graph allows synthesis programs to perform several global optimisations, not hindered by block boundaries as are imposed by most other representations.
- A flexible and open approach to data typing is used, allowing numeric as well as bitwise operations on the same values, and supporting easy addition of new datatypes.
- Dedicated nodes can be used for input and output operations, allowing the specification of sequences of reads and writes on one physical port, conditional I/O, or the sharing of one physical port with several hierarchically structured subgraphs.
- The exchange format is a textual file with a braces oriented syntax as Lisp and EDIF. This makes the format easily extendible for local or future needs, while maintaining backwards compatibility with older programs who do not understand these. For a standardisation effort in a long-term research project, this was considered an important property.

The data flow graph itself was subject of research in subtasks A.1 and A.2, in order to make sure that its semantics were general enough to allow mapping of other behavioural description languages into them, and at the same time consise and accurate enough to allow formal verification.

The first software handling the textual format was made available to the other ASCIS partners at a project workshop in Grenoble, November 1990. Improved and extended releases were distributed in February 1991, July 1991, and February 1992.

## 2.2 Recent developments

During the last half year work was done to extend the DFG format to support intermediate or full synthesis results. Such an extension would greatly enhance possible cooperation between the partners, by allowing the comparison or use of each others algorithms for individual synthesis steps (scheduling, allocation, binding, network generation).

The extension is basically made by adding two new types of graphs, a control graph (CTG) and a network graph (NWG), next to the existing dataflow graph.

Whereas the dataflow graph defines the algorithmic behaviour, the control graph fixes the timing behaviour and hints on the controller design, and the network graph defines the hardware on which the algorithm executes: the final synthesis result.

The control graph basically corresponds to the finite state diagram as normally drawn for controllers. However we extended the semantics to allow concurrent multiple active states, and hierarchical structuring of such graphs. The result of scheduling can now be expressed as links between DFG nodes and CTG nodes.

The network graph describes the final network that results from the architectural synthesis. The nodes in the graph correspond to physical modules to be implemented on the IC. Initially the graph might contain a set of nodes only (i.e. no edges), indicating the set of hardware modules on which the algorithm must be executed, the result of module allocation. Later, links between DFG nodes and NWG nodes indicate which operations are mapped onto which modules, the result of binding. Equal notions hold for register(file)s and busses. Finally the fully interconnected network follows as result.

Besides links between nodes, links between graphs are supported. These express relations as 'this DFG is controlled by this CTG', or 'this NWG is a possible implementation of this DFG'.

These graph links allow -together with the hierarchy concepts- the description of the synthesis library (operations, modules, and their relations)

in the same terms, by adding an 'empty' DFG for each operation and an 'empty' NWG for each module.

Besides on these basic concepts, work was going on to augment the textual format, but more even to design a programming interface to manipulate these sets of graphs, suitable to be used in all tools. Initial documentation of this programming interface was distributed to the ASCIS partners.

# Chapter 3

# Work at TH-Darmstadt

The work at TH-Darmstadt is strongly related to the DFG-format developed by the Eindhoven University of Technnology. The internal format used at TH-Darmstadt had been developed in parallel to the Eindhoven format but is very similar in structure and semantics. Two converters are available which transfer the Darmstadt DCG-format into the Eindhoven format and vice versa. So far no principal transformation problems had been encountered which could restrict the conversion process and all functional descriptions representable in the Eindhoven or Darmstadt format are convertable. Therefore, all high level synthesis tools developed at TH-Darmstadt are accessable through the Eindhoven graph format.

Synthesis at TH-Darmstadt starts with a description in TH-HardwareC which is different from Stanford HardwareC. Stanford HardwareC was unacceptable due to some major restrictions which make it impossible to fully exploit many features absolutely required for the intended application domains. Major advantages of TH-HardwareC compared with Stanford HardwareC are:

- the availability of more complex variable types esp. **fix** and **float**
- array declarations are permitted for all available variable types
- there are no restrictions in the use of variables (esp. to control blocks like loops etc.) which are not known at compile time

A TH-HardwareC compiler is available performing a thorough data/control flow analysis. The result of the compiletion is a textual representation of the data/control flow format used at Darmstadt.

This format has to be fed into a consecutive optimizer which peforms some basic optimizations on the data/control flow graph. The first optimization step is dead code elimination which is not simply a check if nodes are somehow connected through edges to the remaining data/control flow graph. This check exploits semantical knowledge of the format to decide if certain

connections make sense or are rather redundant without influence on data stream computations.

The optimizer is also responsible for removal of redundancies generated by the compiler. These redundancies arise due to the fact that during compilation of a certain TH-HardwareC block there is no global information available on future use of variables generated in this block. Therefore, these variables are generally made available outside the loop, which requires additional graph elements despite they are never used in the consecutive TH-HardwareC code. This optimization can be very important in some special applications as e.g. the fifth order digital elliptic wave filter where a graph reduction of 50% had been achieved. Some additional redundancy removals are also performed through the optimizer.

After optimization the internal graph format can be transformed into the Eindhoven format. Through this interface TH-HardwareC specification is available to all ASCIS partners. For tool comparisons the following benchmarks are provided in TH-HardwareC:

- fifth order digital elliptic wave filter [SB90](small example for tool checking and first comparisons)
- parallel fifth order digital elliptic wave filter (for scheduler quality comparisons)
- state variable filter (real process control application)
- ignition control for combustion engines (very hard real process control application requiring background memory management)
- correlator1 (simple example taken from Leiserson [Le83] which requires global pipelining for achieving optimal solutions)
- correlator2 (functionally equivalent to correlator1 but specified with different TH-HardwareC constructs. This is a benchmark to check for description sensitivity of the same functional behaviour)

Another tool that can be applied before conversion to the Eindhoven flow graph format is a first version of a background memory manager. This tool transforms the data/control flow graph into a functionally equivalent representation with less arrays, reduced and reordered memory access operations. Since this tool only performs transformations on the internal graph format every ASCIS partner can use it either through the TH-HardwareC interface or through the Eindhoven graph format.

All structural synthesis tools which are already finished or currently under development at TH-Darmstadt (this includes various schedulers and a path to layout through the CADENCE framework in SOLO 2030 configuration)

can be used by all partners through the Eindhoven interface. However, it is certainly not possible to exchange structural synthesis results through a DFG format. At this moment in connection with structural synthesis the interchange format is mainly used at TH-Darmstadt to compare the performance of locally developed tools with those of other ASCIS partners.

# Chapter 4

# Proposed Extensions to the ASCIS DFG for External Communication

This section proposes a number of extensions to the basic ASCIS DFG representation. The extensions are based on the work of the TUL ASCIS group and are designed to support to following requirements:

- The ability to incorporate DFG models into systems consisting of non-DFG models.
- Pragmatics suited for description of 'one-shot' calculation applications.
- Expression of DFG descriptions in VHDL.

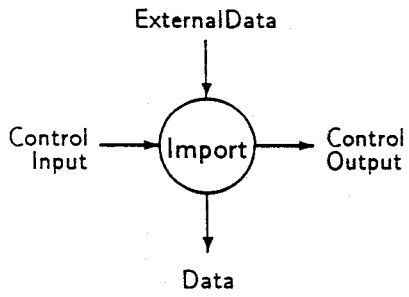
These requirements are achieved through the following facilities:

- The introduction of three nodes with special semantics for interfacing with the non-DFG world.
- A restriction on the semantics of the outer loop.
- A VHDL subset capable of representing DFGs.

This appendix details these extensions, and suggests a set of rules for converting between the pure ASCIS DFG and the extended DFG.

## 4.1 External Communication

In the TUL high-level synthesis work, a design representation consists of functional units interconnected by level-sensitive asynchronous protocols.




| Port | Direction | Type |
|---|---|---|
| ControlInput | Input | Control token |
| ExternalData | Input | Non-DFG signal |
| ControlOutput | Output | Control token |
| Data | Output | Data token |

Figure 4.1: The **Import** Node.

One possible representation of a functional unit is a DFG. In order to facilitate a direct representation of the communication with other functional units, TUL suggests the addition of three nodes with special semantics: **Import**, **Export** and **Wait**.

The following subsections details the semantics of each of these nodes. Notice that the **Import** and **Export** nodes do not provide any synchronization with the external world (unlike the ASCIS DFG **get** and **put** nodes). Further information may be found in [Bra91b].

### 4.1.1 The Import Node

The **Import** node (figure 4.1) has one DFG input and two DFG outputs. In addition, it has one non-DFG input. The behavior of the node is:

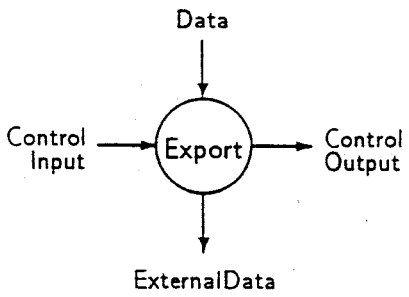
> When an **Import** node receives a control token on its control input, it samples the external signal and generates a corresponding data token on the data output. Also, a control token is output on the control output.
>
> Several **Import** nodes may sample the same input signal.

Consequently, the **Import** node guaranties:

- That the external signal is sampled after the ingoing control token arrives.
- That the external signal is sampled before the control token is transmitted on the control output.

### 4.1.2 The Export Node

The **Export** node (figure 4.2) is the inverse of the **Import** node. It has two DFG inputs and a single DFG output. Also, it has one non-DFG output. The behavior is:




| Port | Direction | Type |
|---|---|---|
| ControlInput | Input | Control token |
| Data | Input | Data token |
| ControlOutput | Output | Control token |
| ExternalData | Output | Non-DFG signal |

Figure 4.2: The **Export** Node.

When a control token is present on the control input and a data token on the data input, the external signal is updated to reflect the value of the data token. The control token is then copied to the output.

A given external signal may have several driving **Export** nodes (though a well-behaved DFG should never permit more than one such node to be able to fire at any given time). Also, the signal is assumed to remember the latest assigned value.

Consequently, the **Export** node guaranties:

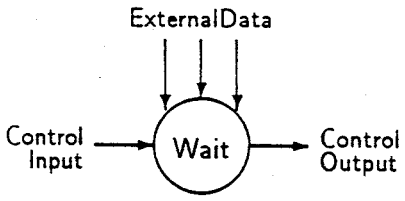
- That the external signal is untouched until the ingoing control token is present.
- That the external signal has stabilized before the control token is transmitted on the control output.

### 4.1.3 The Wait Node

This is possibly the most interesting of the new nodes, as it allows synchronization of DFG processing with the external world without employing busy-wait schemes. This have a number of advantages:

- The real intent of the construction (i.e., synchronization) isn't obscured by loop structures, etc.
- By the abstraction of the actual synchronization mechanism, an implementation is free to choose whatever method suits the actual low-level architecture best.
- Simulation of the system is considerably easier.

The **Wait** node (figure 4.3) has a DFG input and a DFG output. In addition, it has one or more non-DFG inputs. Its behavior is given by:




| Port | Direction | Type |
|---|---|---|
| ControlInput | Input | Control token |
| ExternalData, | Input | Non-DFG boolean signals |
| ControlOutput | Output | Control token |

Figure 4.3: The **Wait** Node.

When a control token is present on the input and any of the external signals are **TRUE**, the control token is copied to the output.

Thus, the temporal behavior is:

- The state of the external signals are ignored until the input control token has arrived.
- The output control token will not be generated until at least one of the external signals are **TRUE**.
- The control token may pass straight through the **Wait** node if any of the external signals are **TRUE** when the input control token arrives.

## 4.2 The Outer Loop

[vEdJS91], section 2.10, states the overall behavior of the ASCIS DFG: As the DFG represents a piece of hardware, it is possible to execute the algorithm several times over. In order to formalize this notion, an implicit, infinite outer loop is therefore assumed to surround the graph.

[vEdJS91] furthermore defines that the state of the graph after one such iteration is preserved and used as the initial condition for the following iteration. This is different from the TUL model of DFGs which calls for the re-initialization of the graph for each iteration of the outer loop.

The reason for this difference in the DFG pragmatics is due to differences in application areas: The TUL work focuses on applications characterized by 'one-shot' calculations (i.e., each iteration of the outer loop corresponds to the execution of a complex algorithm and the hardware is effectively reset between calculations), e.g., support processors for general purpose CPUs. Consequently, it is advantageous to be certain that no state is preserved between iterations.

## 4.3 VHDL Representation of DFGs

This section suggests the use of an alternative syntax for DFGs allowing the DFG to be represented in VHDL. The advantage of this is the possibility of representing (parts of) a design at several different levels within the same base language. In particular, the TUL ASCIS group has VHDL representations for procedural models [Bra91a], data flow graphs and register transfer level descriptions.

The VHDL DFG representation is based on the VHDL structural representation and obtains the correct semantics through a specialized use of the VHDL bus resolution functions [MB90b, MB90a].

## 4.4 Conversion Between the DFG Formats

The extensions listed above present problems in exchanging designs between the ASCIS DFG and the extended DFG. This section details the problems and suggests a set of conventions which permits exchange of designs. The treatment is divided into three parts according to the three sections above.

### 4.4.1 The Interface Nodes

The addition of the nodes for external communication represents the worst problem, as the two representations have different models of the external world. This difference can only be resolved by establishing a set of conventions for representing the communication protocols of one representation in the protocols of the other.

Converting from the ASCIS DFG representation to the extended representation is relatively straight-forward:

The communication of a DFG token can be implemented by adding two boolean control signals and employing a simple 4-phase handshake protocol.

Converting from the extended representation to the base representation is more difficult. Conversion will be possible in cases where the communication can be viewed as a handshake protocol as stated above. It may be possible to establish further such conventions.

Conversion cannot be achieved in the general case: The behavior of the **Wait** node cannot be represented in the ASCIS DFG (the **Wait** node potentially allows information to be lost — this is not possible using the ASCIS DFG nodes).

### 4.4.2 The Outer Loop

Converting from the TUL representation to the basic ASCIS DFG representation should not present any problems.

Conversion the other way could be achieved simply by always adding an explicit outer loop. However, for algorithms which does not utilize any global state, this might lead to inefficient synthesis.

Consequently, the addition of an explicit outer loop should be selective, either by manual control or by automatic analysis of the algorithm. The latter option will not always be possible but a safe method could be devised (by adding the loop, in case of doubt).

### 4.4.3 The VHDL Representation

The alternative representation as a VHDL subset does not really present any problems: A simple syntactic translator will be able to transform back and forth between the representations.

## 4.5 Summary

This chapter has presented a number of extension to the ASCIS DFG representation for the purpose of making the representation better suited for the work of the TUL ASCIS group.

The extension falls into three categories:

1. The addition of three nodes with special semantics for communication with the external world.
2. A restriction on the behavior of the outer loop.
3. A different syntax for representing the DFGs in a subset of VHDL.

## 4.6 Modelling Control-Flow Dominated Circuits With DFGs

In the ASCIS project, INPG has concentrated on the synthesis of control-flow dominated circuits. Inherent properties of such circuits include hierarchical control, global transitions and parallel state execution. Such properties rendered it difficult to represent such circuits using early versions of Eindhoven's DFG and thus it was decided to stay with VHDL as the standard means of representation. However, subsequent work at TUE overcame the main obstacles and it was proposed to start using this format in ASCIS2.

The research in ASCIS has culminated with the development of the AMICAL system, detailed in deliverables TIM3/m30/c3/1-2. Essentially, this system allows the interactive, manual or automatic allocation of functional units and connections to operations in a data-path. More recently, a path-based scheduling tool has been added to AMICAL that allows the direct generation of control-steps from a behavioural-level VHDL description.

A standard DFG is not an adequate means of representing control-flow dominated circuits within INPG's design framework for two reasons:

Inability to represent complex control structures

Incompatibility with AMICAL's input description

Complex control structures such as those mentioned previously are fundamental properties of control-flow dominated machines. These structures however are very difficult to represent using classical DFG approaches.

The AMICAL system accepts as input a set of macro-cycles containing instructions that can be executed in parallel. It is assumed that all register allocation has been done beforehand. This is in direct contrast to DFG principles where registers are not considered and operations are carried out on Values.

During an ASCIS workshop in Lyngby in September 1991, a methodology was proposed by TUE whereby their DFG would be able to accommodate control-flow dominated circuits. Rather than a single DFG, it was proposed to use three inter-acting representations that could specify the data-flow, the control-flow and the allocated resources. This representation would take the form of three inter-connected graphs: a data-flow graph similar to the early versions proposed by TUE, a control graph that would indicate the scheduled states and the flow of control between these states, and a network graph showing the resources used. In addition, links between the graphs would show which operations in the data-flow graph were to be scheduled in which state of the control graph and were to use which registers (if any) in the network graph, thereby satisfying all of the constraints imposed by both control-flow dominated circuits and INPG's design framework.

This work was proposed in ASCIS1 and was to be developed in ASCIS2. In the meantime, all designs and examples at INPG were represented in a subset of VHDL that is compatible with Proc-VHDL developed at TUL. This has the additional benefit of being able to use the expected link between Proc-VHDL and the DFG format.

# Bibliography


[Bra91a] Jens P. Brage. ProcVHDL: A VHDL subset for high-level synthesis. Technical report, Technical University of Denmark, 1991.

[Bra91b] Jens P. Brage. The semantics of a set of DFG nodes for high-level synthesis. Technical report, Technical University of Denmark, 1991.

[Le83] C. E. Leiserson and et. al. Optimizing synchronous circuitry by retiming. In *Third Caltech Conference on Very Large Scale Integration*. Computer Science Press, 1983.

[MB90a] Jan Madsen and Jens P. Brage. Flow graph representation in VHDL. Technical report, Technical University of Denmark, 1990.

[MB90b] Jan Madsen and Jens P. Brage. Using VHDL bus resolution functions for data flow graph modelling. In *Proceedings of First European Conference on VHDL Methods*, 1990.

[SB90] L. Stok and R. van den Born. *Logic and Architecture Synthesis for Silicon Compilers*, chapter EASY: Multiprocessor Architecture Optimisation, pages 313–327. North-Holland, 1990.

[vEdJS91] Jos T.J. van Eijndhoven, G.G. de Jong, and L. Stok. The ascis data flow graph: Semantics and textual format. Technical Report 91-E-251, Eindhoven Univ. of Tech., Jun 1991.

# Appendix A

# A Data Flow Graph Exchange Standard

Jos T.J. van Eijndhoven, Leon Stok
Preprint of a paper appearing in the proceedings of "The European Design Automation Conference", Brussels, March 1992