# A HIGH-LEVEL DEVELOPMENT TOOL DEDICATED TO THE GENERATION OF CORES FOR THE IMPLEMENTATION OF IMAGE PROCESSING APPLICATIONS.

*Virginie Fresse[1] Stephen Marshall[1] Olivier Déforges[22]*

(1)
Dept. of Electronic and Electrical Engineering
University of Strathclyde, 204 George Street
Glasgow, G1 1XW United Kingdom.
Phone: +44/0 141 548 2250
E-mail: VirginieFresse@eee.strath.ac.uk
s.marshall@eee.strath.ac.uk

(2)
ARTIST/FRE URER laboratory
INSA RENNES
20 Avenue des Buttes des Coësmes, CS14315
35043 Rennes, France
Phone: +33/0 223 238 286
E-mail: odeforge@insa-rennes.fr

## ABSTRACT

This paper demonstrates the integration of a high-level development tool in a fast and easy-to-use prototyping process, called AVSynDEx. This process is dedicated to the implementation of real-time image processing applications on parallel and mixed platforms. The integration of this final environment, which is a high-level development tool for the generation of IP cores, completes the prototyping process. The designer of image-processing algorithms can finally develop and supervise the whole implementation process without any pre-requirement and within a short development time.

## 1- INTRODUCTION

AVSynDEx is an existing rapid prototyping process aimed at the implementation of digital signal processing applications on mixed architectures (multi-DSP+FPGA) [1][2]. It is based on the use of available and efficient CAD tools established along the design process so that most of the implementation tasks become automated. These tools and architectures are judiciously selected and integrated during the implementation process to assist a signal-processing specialist without relevant hardware experience. The image-processing designer can also develop and implement the algorithm within a short development time and without requiring highly-specialist hardware engineers. Nevertheless, this process allows the image-processing designer to implement the functions on FPGA on condition that the corresponding core (hardware processing) already exists. The generation of new cores is not possible and a hardware engineer is then needed.
This paper completes the development of the previous prototyping process, called AVSynDex [1], with the integration of a high-level development tool, DK1 Design Suite [5], for the generation of IP cores. The result leads to the final rapid prototyping process for the development and implementation of any image-processing applications on mixed architectures. The development and implementation of a stack filter is used to illustrate the benefits of such integration.

This work is granted by the Regional Council of Britanny

## 2- PROTOTYPING PROCESS

The methodology (figure 1) integrates two main CAD tools: Advanced Visual System (AVS) for the functional development of the application described as a static data flow graph [3], and SynDEx as generator of optimised distributed executive [4].

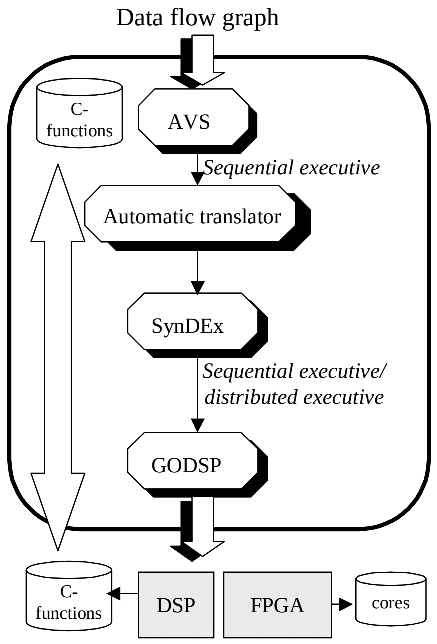

*Figure 1: Rapid prototyping process for implementation of image processing applications.*

The image-processing designer creates the data flow graph by means of the graphical development tool called AVS. The processing operations are C-functions, which are connected together by means of input and output ports (data transfers between two functions). An automatic translator and an academic tool called SynDEx, ensure the translation towards hardware and software descriptions, as explained in [1]. The target platform is a mixed architecture made up of multi-DSPC6x and a Virtex FPGA [2]. This architecture can be modified without changing the prototyping process and the available links.
Hardware and software processing is different for the same functionality:

- A software processing operation is a C-function, fully compatible to the initial processing operation. In this case, the functional behaviour of the implemented module is necessarily correct.
- A hardware processing is a core whose initial description is made with Hardware Description Languages (these languages being completely different). The generation of cores is non-automatic and cannot be achieved by the image-processing designer. So, a specialized hardware engineer must reproduce the overall functional behaviour of the C-function to generate the corresponding IP core. The functionality of the core may be incorrect if the HDL description does not reproduce the same functionality. This non-correspondence can only be detected during the hardware implementation.

In this prototyping process, there is a perfect correspondence between the C-function and the software module. Unfortunately, the hardware module does not possess any correspondence with initial module. This lack of correspondence leads to the possibility of a functionally incorrect hardware implementation. The only way to check its functionality is the implementation of the core. This constraint is time-consuming and the partitioning between the software and hardware part is difficult to achieve.

The solution is the integration of a high-level development tool called DK1 Design tool, which is a solution for an easy generation of IP core.

## 3- INTEGRATION OF THE DK1 TOOL

### *3.1. Presentation of DK1 Design tool*

Celoxica's DK1 design suite is a high-level environment, enabling the image-processing designer to generate IP cores [5]. The description is made by means of the Handel-C language. This language is grounded in ISO-C but contains several extensions required for hardware developments. Some expressions, statements, types, type operators and objects are used in C-only or Handel-C only. Generally, most of them are suitable for both languages.
Different extensions are proposed by Handel-C; some of them are listed below.

- Handel-C can process variables of arbitrary width. This means that the size must be judiciously specified. As the tool is dedicated to a hardware implementation, a variable is a register whose size is the width of the data. Operations on several variables are only possible for identical widths (otherwise the concatenation operator is required).
- An array of n variables corresponds to a set of n registers. It is also possible to create memory arrays and multi-dimensional memory arrays with *wom* (write only memory), *rom* or *ram* keywords. The use of *rom* and *ram* is restricted to one element access per clock cycle.
- In a way similar to C-language, a program is executed sequentially. The additional *par* statement allows parallel executions in a specific function. The parallelism can be at instruction-level or bloc-level.
- For the communication, channels provide the links between parallel branches. One parallel branch outputs data onto the channel and the other branch reads data from the channel. Channels also provide synchronisation between parallel branches. As with the variables, Handel-C provides flexible data path widths.

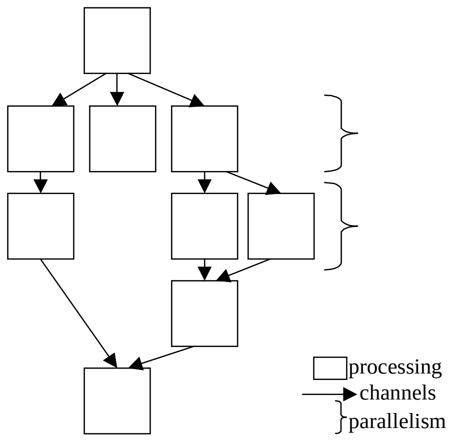

*Figure 2: Example of structure proposed by DK1 tool. Some parallel functions can be achieved; the second line indicates a parallelism between 3 functions (DK1 tool manages all synchronisations). Data is transferred by means of channels*

Parallel to this, the Handel-C language includes some restrictions such as: the function may not be called recursively; old-type function declarations and variable length parameter lists are not possible. It is also not possible to change a variable by casting or to use the floating point.
An example of a Handel-C program containing *par* statements and channels is given in figure 3.

```
Set clock=external "P1";

#define SIZE1 5                  /*Width of data*/
#define WIDTH 256                /*Size of memory*/
#define LOG2_WIDTH 8   /*With of pointers*/

chan SIZE1 queue_in;             /*Channels*/

void main(void)
{ ram unsigned SIZE1 Source[WIDTH*WIDTH];
   unsigned (LOG2_WIDTH*2) i,j;

par{
      {   for(j=0;j<WIDTH;j++)
              for(i=0;i<WIDTH;i++)
              queue_in?Source[j*WIDTH+i];}

              function1();
              function2();
      }
}
```

*Figure 3: Example of Handel-C program, 5bits-data are read from channel "queue_in" to load them in the memory called "Source". Parallel to this, "function1" and "function2" are executed (with the "par" statement).*

### 3.2. Generation of cores with DK1 design tool.

DK1 Design tool has several modes. Some of them are used for the generation of IP cores.
The generation of IP cores is shown in figure 4.
The initial AVS description is used to generate the Handel-C program. According to the previous rules, the designer turns the C-function into a Handel-C form. These languages being similar and the translation being at instruction level, the modifications are easy to achieve.

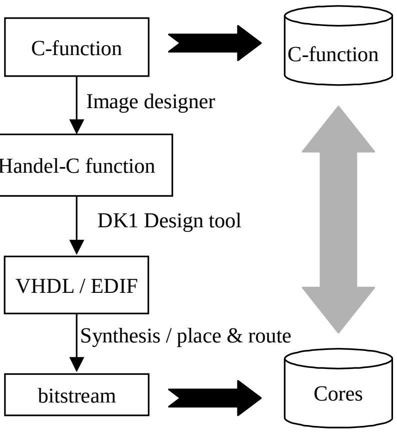

*Figure 4: Process of core generation with DK1 tool.*

DK1 Design Suite has a debug mode to check the functionality of the function (simulation for the program with a specific image). This mode is only used to check the resulting behaviour of the Handel-C function.

Then, the image-processing designer can specify two modes:

- The first mode (VHDL) generates the corresponding VHDL description. A synthesis tool is then used to generate the EDIF description.
- The second mode (EDIF) generates the EDIF description directly.

The structure of the Handel-C language always guaranties to be synthesized (this is not the case for a HDL description).
Finally, a place and route tool ensures the generation of the IP core. All these tools are easy to use and the image-processing designer does not need additional hardware expertise.

## 4- GENERATION OF A NON-LINEAR CORE

The development of a non-linear filter illustrates the benefits of such integration. The stack filter is a non-linear filter, which has proved to give excellent results in image restoration, noise reduction and optical character recognition. This filter is well suited for a hardware implementation but the complexity of such implementation leads to few examples of this occurring in practice.

### 4.1. Principle of the stack filter

All stack filters have two properties, a superposition property know as threshold decomposition, and an ordering property known as the stacking property [4].
The principle of the filter is given in figure 5.

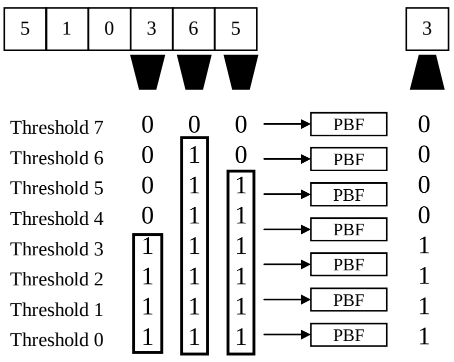

*Figure 5: Example of stack filter. The 3bits input sample is threshold in 8 levels (=$2^3$). Then, a series of PBFs computes the min value for each level. The result is a stack of '1' and '0' representing the output sample.*

- The threshold decomposition is first achieved for each pixel of the input window. The output of such decomposition is a set of binary threshold signals, each binary signal representing a level or threshold. So, for m-bits samples, there are k=$2^m$ binary signals.

The k binary signals for an n-valued sample, threshold$^k$(n), are defined according the following equation

$$threshold^{k}(n)=\begin{cases}0: Input(n)<k\\1: Input(n)\geq k\end{cases}$$

- Then, a series of **P**ositive **B**oolean **F**unctions are performed for every threshold. PBFs can only be used because they maintain the stacking property and hence validate the stack filter as explained in [6]. PBFs can remain unchanged for each threshold but any combination is allowed only if the output remains a stack of '1' with a stack of '0' on top.
- The resulting binary signals represent the output value, found by stacking the Boolean outputs excluding Threshold 0.

### *4.2. Generation of the non-linear core.*

The characteristics of the stack filter implemented are:

- Images are 256*256.
- Pixels are 8bit data (= 256 binary signals).
- The input window is 3*3 pixels.
- The PBFs are minimum operators

The only benefits are in the acceleration of the development process and the execution times (the result of processing on the images being identical to software implementation).

The first stage is the description of the functional algorithm by means of a C-function. This first function is executed on a PC to check the functional behaviour of the filter. The C-function is directly and easily implemented in a C6x DSP. The image-processing designer modifies the C-function to obtain a Handel-C form. The functionality is checked with the DK1 tool.

### *4.3. Results*

Table 1 shows the execution times according the target component. The result of a DSP implementation is not satisfactory: the stack filter contains several nested loops containing conditioning, which do not suit the VLIW architecture of the DSP.

| | PC P III-1GHz | DSP C6x 200 MHz | Virtex XCV400E |
|---|---|---|---|
| Time | 8s | 18,45 s | 2,282 ms |

*Table 1: Resulting execution times for a stack filter.*

Timing characteristics for the FPGA are:

- Minimum period: 38.927ns (Max. freq.: 25.689MHz)
- Maximum net delay: 8.445ns

The result of the FPGA implementation is satisfactory: the core can be used to implement real-time image processing applications. The execution time (2.282 ms,) allows the image-processing designer to implement the algorithm within real-time constraints.

| | |
|---|---|
| Flips-flops | 177 (1%) |
| LUT | 2 101 |
| RAM (16*1) | 160 |
| SLICES | 797 (16%) |

*Table 2: Resources for a stack filter.*

In addition to a fast execution time, the resources are relatively low as shown in the table 2. External memory is used to load and save samples (17 ns per memory access). The number of slices used is 797 (4800 are available in the Virtex FPGA). Each Virtex CLB contains four Logic cells organised in two similar slices. Consequently, 16% of the FPGA is used for the stack filter.

DK1 Design Suite ensures a fast generation of IP cores:

- The translation of a C-function into a Handel-C form was performed in 30 minutes.
- The generation of core form the Handel-C description was about 15 minutes.

## 5- CONCLUSION

Consequently, the DK1 Design Suite development tool is an ideal solution for the fast and easy generation of IP cores. An HDL description gives better results (in terms of Quality of Results QoR, which is an area-performance estimation) than a Handel-C description. Nevertheless, the Handel-C language is easier to use and the functionality is a direct translation from the C-function. This leads to a close functional behaviour between a C-function and the corresponding core. The hardware and software modules are functionally similar. Integrated in our prototyping process AVSynDEx, the image-processing designer can develop the algorithm by means of the AVS tool, and then generate the IP cores with DK1 Design Suite. The implementation of the application does not need highly specialist hardware engineers; the image-processing designer manages all the development stages.

The result is a rapid prototyping process for the implementation of real-time image-processing application. The development, the implementation and modifications and optimisations are performed in less than 1 day.

## REFERENCES.

[1] Nezan J.F., Fresse V., Deforges O.: "Fast prototyping of parallel architectures: an Mpeg-2 coding application. CISST proc., Las Vegas, USA, June 2001.

[2] Fresse V., Deforges O., Assouil M. "Rapid prototyping for mixed architectures". IEEE International Conf. on Acoustics, Speech, and Signal Process., 5-9 June 2000, Istanbul, Turkey

[3] Grandpierre T., Lavarenne C., Sorel Y.: "Optimized Rapid Prototyping for Real-time embedded heterogeneous multiprocessors". 7th Int. Work. on Hardware/software Co-design, May 1999 Rome, Italy.

[4] Advanced Visual Systems Inc. "Introduction to AVS/Express". http//www.avs.com.

[5] Celoxica Inc.: "DK1 Design Suite". User manual.

[6] WendtP.D., Coyle E.J., and al. "stack filters". IEEE Trans. on Acoustics, Speech and Signal Processing, Vol.ASSP-34, no4, pp 898-911, August 1986.