Macro-programmable DSP architecture for parallel/pipelined data path units, targeted for FFT based algorithms

Andreas Drollinger, Alexandre Heubi, Peter Balsiger, Fausto Pellandini
Institute of Microtechnology, University of Neuchâtel
Rue A.-L. Breguet 2, 2000 Neuchâtel, Switzerland
URL: www-imt.unine.ch/esplab
e-mail: andreas.drollinger@imt.unine.ch

Abstract

A macro-programmable DSP architecture is presented, which is very well situated for the implementation of algorithms with regular data flow graphs, like FFTs. A smart grouping of the algorithm together with the macrocode concept reduce drastically the control and address generation overhead of the DSP and shorten the computation time. This is finally manifested in very low-power consumption, small DSP size, high throughput combined with a high flexibility of the DSP architecture.

1. Introduction

Several DSP algorithms have regular data flow structures and use just a small set of basic operations, which are repetitively executed. Taking as example an FFT filterbank: Its operation set is limited to the radix-2 butterfly. Butterfly operations of a higher radix number make the FFT computation more efficient. With some supplementary simple operations even sophisticated, FFT based filterbank algorithms like the WOLA[1] algorithm can be performed and gain coefficients can be applied on the processed data in the frequency domain [1], [2].

It is quite inefficient to use general purpose DSPs for such algorithms because they do not allow to profit from the algorithm's regularity in order to save program code or to shorten the execution time. General purposes DSPs execute each operation individually and needs for that a detailed description that includes the internal (macro-) operation sequence and the parameter addresses. For each operation, general purposes DSPs have to read all these informations together with control instructions. This lowers the execution time and increases the power consumption.

On the other hand, a hardwired processor works much faster, because it doesn't lose cycles for the control instructions. But the flexibility, due to the hardwired control and datapath units is low.

The presented DSP architecture offers a solution that is flexible because it is macro-programmable and that has a similar efficiency to hardwired processors. It is especially efficient for well-organized algorithms like FFTs.

The remainder of this paper describes the development of a macro-programmable DSP architecture. Section 2 presents the structuring concept for regular algorithms, while section 3 describes the DSP architecture for this kind of algorithms. Section 4 shows an application example and some results. Finally, conclusions are presented in Section 5.

2. The algorithm structuring

Before considering the algorithm's structuring, some terms should be introduced:

- *A function* is a part of the algorithm, which has to be executed without interruption.
- *A pass* is a sub-part of a function and groups identically operations together in a way that all data are read one time from the memory, processed and written back to the memory.
- *An operation* is a sub-part of a pass and corresponds to a basic datapath operation.
- *A cycle* is the number of clock cycles, which are needed by the datapath unit to process an operation.

Figure 1 shows schematically the data flow of a 16-complex-point FFT spectrum analyzer with a windowing and time folding feature like it is used in the WOLA transformation. It has three

[1] WOLA: weighted overlap add

main functions: analysis, gain application and synthesis.
The analysis function has 3 passes. The first one contains MAC operations and is used for windowing and time folding. The next two passes contain radix-4 butterfly operations. The gain application function has only one pass with multiplication operations. The last function corresponds to the first one, but the pass order and the operations are reversed.

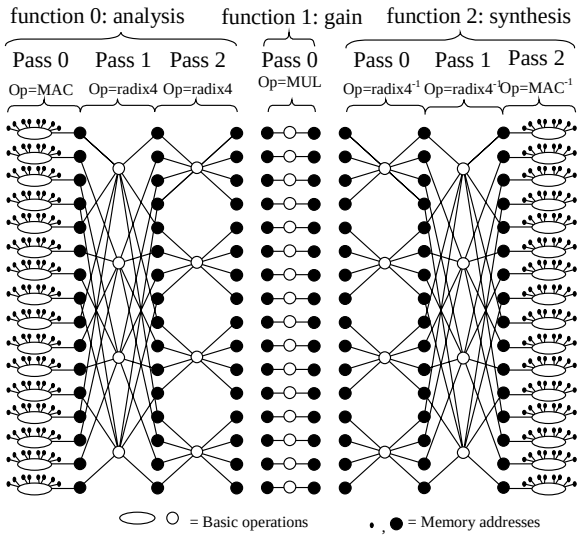

Figure 1: Data flow of an FFT spectrum analyzer has 3 functions: Analyzis, gain application and synthesis

The structuring of data flow graph as shown in Figure 1 has several nice characteristics, which simplify an implementation:

- Each pass processes ones all data words
- Each pass contains 2^n operations (n=1,2,3..)
- The address generation becomes very easy if the operations of a pass are processed in the correct order.

These characteristics are conserved in each configuration of an algorithm family. But of course, the detailed algorithm parameters like the number of passes or the address generation function depend on the configuration.

Advantages of this kind of algorithm structuring

An implementation of an algorithm that is structured as proposed above profits on the one hand from the fact that a pass contains only one operation type and on the other hand from the simplified address generation using bit reversing functions.

Because a pass contains just one operation type it isn't necessary to code each operation of the pass individually. It is sufficient to define the operation type ones on the pass begin. In that way the program code becomes smaller and the number of program memory accesses decreases.
The same simplifications and advantages are possible if a parameter address follows a simple rule. Just the rule has to be included in the program code instead one parameter word for each operation. For FFT based algorithms the address rules can easily be coded because the rules are mostly limited to the family of the bit reversing functions.

3. The DSP architecture

The presented DSP architecture has especially been developed for tiny, ultra low-power applications that use FFT based algorithms in a flexible way. Hearing aid devices are a typical application example [4]. The following design constraints, which have to be satisfied by the DSP architecture, are mainly defined by the destination applications:

- Minimal hardware size (DSP size and memory requests)
- Minimal power consumption
- High flexibility
- Low supply voltage (down to 0.9 Volt)
- The DSP must be able to treat the algorithms at a clock frequency of only 1 MHz due to the low-voltage related speed reduction of CMOS gates.

The choice of the DSP architecture is mainly influenced by the knowledge that all operations are the same inside a pass and that the addresses for data and coefficient follow well defined rules. Once, all parameters for a pass, including the operation type and the address rules, are defined, these parameters are sufficient to process an entire pass.
Figure 2 shows how this conclusion has been transformed in a basic DSP structure. The head of the DSP is the control unit. It defines at each pass begin the pass parameters, which are transferred together with the processing or cycle state via the configuration bus to the so-called functional units. The last ones contain address generators, data interfaces and the datapath and are responsible for the data processing during

e pass. When a pass is completely processed, the control unit defines the configuration for the next pass and restarts the functional units again. This continues until the last pass of the function has been processed.
The signal *start* activates the DSP for the treatment of one function that is defined by *function number*. The end of the processing time is indicated by *function end*.

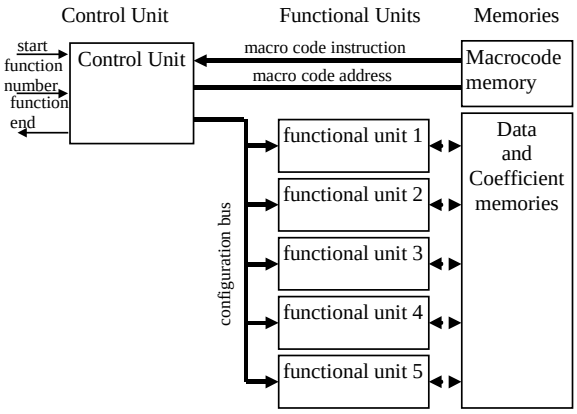

Figure 2: The basic structure of the DSP

The control unit gets the pass parameter from a program memory. In order to avoid storing redundant information in the program memory, the program code has been organized as shown in Table 1. Some parameters are valid during all the functions. They are stored in the so-called *global configuration* word. The parameters that are valid during all passes of one function as well as the number of the function passes itself are stored in the *function configuration* word. All other parameters, which are valid only during one pass, are stored in the *pass configuration* words. This includes the number of pass operations, the operation type and the configuration for the functional units. Depending on the information quantity, the configuration words may occupy one or multiple physical addresses of the program memory.
Because the program code does not contain individual instructions for each operation but pass instructions, which allow the execution of the entire pass like a macro, one speaks about macrocode instead about program code.

global configuration
function 1 configuration
pass 1 configuration
pass 2 configuration
...
function 2 configuration
pass 1 configuration
pass 2 configuration
...
function 3
pass 1 configuration
pass 2 configuration
...
function 4
...

Table 1: The Macrocode organization

In order to get a better architecture, the control unit is separated into 3 units, which are organized hierarchically in the same way, as the algorithm has been structured (function, pass, operation).
The *function controller* has access to the macrocode memory. When it is started for the execution of a function, it reads first from the macrocode memory the global configuration word(s), then the function configuration word(s). Then, it reads for each pass the pass configuration word, starts the pass controller and waits until the last one indicates the processing end of the pass. All configuration words are temporary stored in registers inside the function controller.
Each pass, the *pass controller* starts the operation controller as many times as it is defined by the function configuration. The pass state is made accessible for the other units via *pass configuration*.
The *operation controller* counts the number of execution cycle of an operation and provides the counter state via the operation configuration bus to the functional units. In order to get a wait-state-free system with a pipelined datapath unit the operation controller needs often multiple counters, one for each functional unit.
Finally, the functional units have just to execute each cycle the tasks, which are defined by the different configuration busses.
Because the memories for macrocode and for coefficients are never used at the same time, they can be merged into one single memory. Using the first part of the memory for the macrocode allows that the start address of the coefficients can be specified by the macrocode. This makes the memory organization flexible and allows the use of different coefficient tables. The final architecture is shown in figure 3.

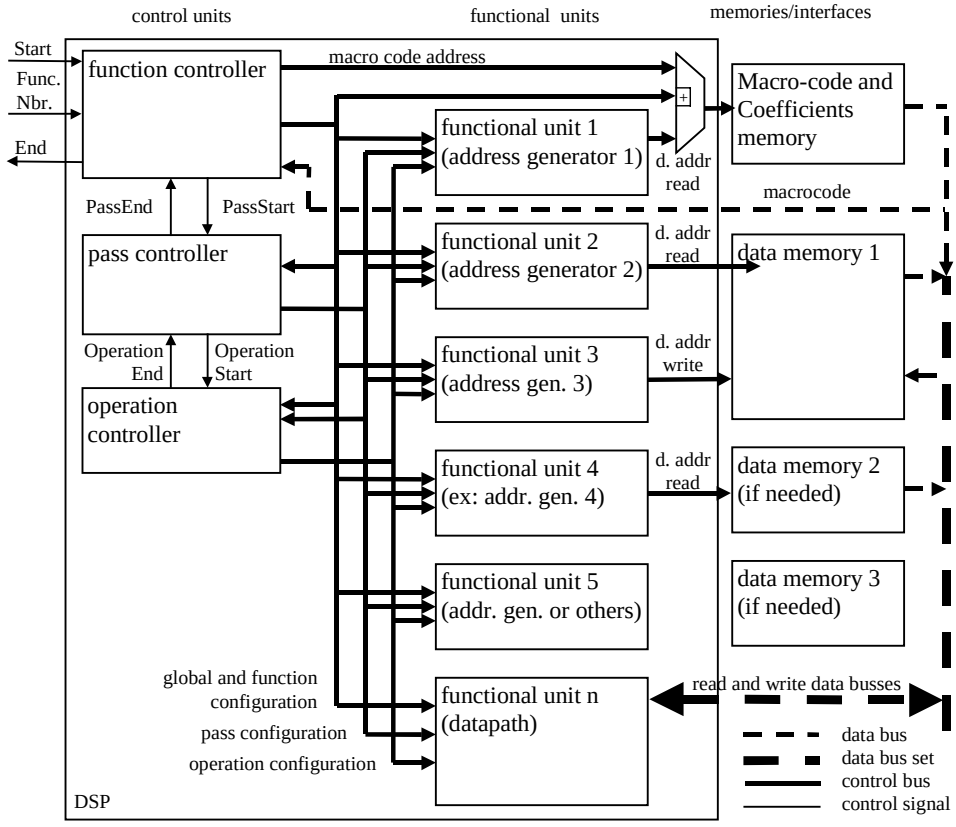

Figure 3: The macro-programmable architecture

The functional units

Datapath and address generators are the most important functional units. But also interface units, control units for circular memories (FIFO), units for block floating point arithmetic, etc. may be realized as functional unit.

A typical block layout for address generators is shown in figure 4: A first block creates an initial address value by selecting the relevant counter state bits. Then, this value is transformed by a bit reversing function before the unused MSB bits are cleared by masking. The combination of the received value with an offset value may be interesting in one of the following tree cases: Realization of circular memories (FIFO emulation), reduction of a 360˚-sinus/cosinus-table on a eighth of the size, mirroring of symmetric window functions.

The datapath unit contains a control part and the real datapath. Depending on the requirements, the last one may contain just one or more than one arithmetic units. The right side of figure 4 shows a structure that has several attractive properties for the processing of FFTs: The organization of the arithmetic units in a first array of multipliers and two arrays of adders is suitable for the treatment of radix-4 butterflies. Depending on the requirements and the possibility of a parallel access of the data and coefficients, each array may contain one ore more than one arithmetic unit. Input registers prevent multiple memory access for the same data word and intermediate registers and output registers create a data pipeline. The control part is usually very small and generates the control signals for the MUX, registers and arithmetic units.

4. Application example and results

The presented macro-programmable architecture has been used for the implementation of a low voltage (0.9 Volt) FFT processor for ultra-low-power applications [5]. The macro-programmability offers a solution for more than 400 different FFT based WOLA filterbank configurations.

In order to improve processing quality and interface flexibility, some supplementary functional units have been introduced; a block floating point unit improves the quantification noise and an input- and output-fifo-controller simplify interfacing to AD/DA converters.

Four address generators are used in this application. The datapath unit contains two MAC units and 4 adders. It is able to process complex operations like radix-2 and radix-4 butterflies.

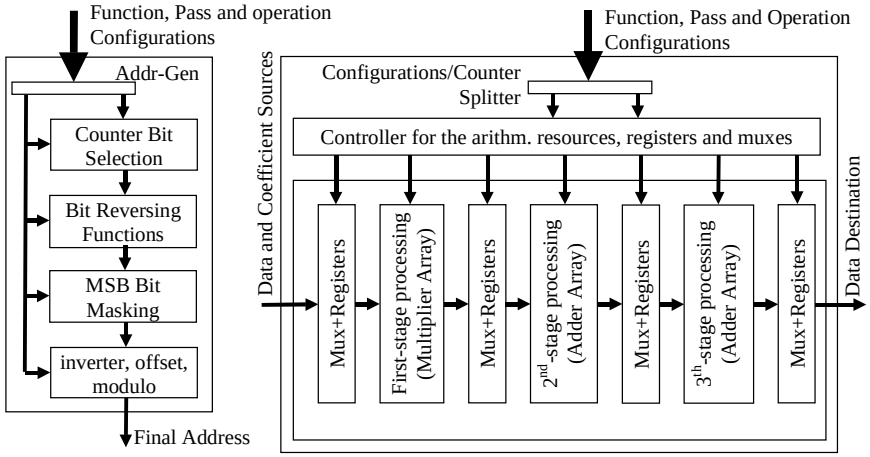

Figure 4: A typical structure of an address generator together with a datapath unit

The processor requires approximately 19 kGE[2]. It has been integrated on a 0.35 μm and a 0.18 μm technology. For the 18 μm integration, depending on the WOLA configuration, the power consumption is between 60 μA and 220 μA @ 1.2 Volt. A typical 16-complex-point WOLA configuration as it is used often in hearing aid devices consumes about 110 μA only @ 1.2 Volt.

Block name	***Size (GE)***	***Comments***
Controller	2045	All 3 controllers toghether (700 GE used for the configuration registers)
Address generators	1301	All 4 address generators toghether
Coeff. interface	1269	Coefficient interface unit with integrated sinus and cosinus table
FIFO controller	646	Controller for the input and output fifo
datapath	12606	Datapath with controller, registers and block floating point unit
Total	**18753**	Entire processor + glue logic

Table 2: Size of the FFT processor (reference technology: Alcatel-Mietec 0.35 μm CMOS)

Table 2 shows the size of the processor and some of its sub-blocks. Due to the smart, macro-programmable approach, the size of the controller part and the address generators could be limited to 10.9% and 6.9% respectively in comparison with the entire processor size without loosing throughput and system flexibility.

[2] GE: gate equivalents, kGE: 10^3 GE

5. Conclusion

The presented DSP architecture is an ideal solution for algorithms with well-organized data flow graphs. It is flexible because it is macro-programmable and has a similar efficiency to hardwired processors.

Because the control units, address generators and datapath unit of the presented DSP architecture work parallel, no cycle is lost for control and address operations: the datapath unit can work with its maximal throughput. The concept allows the use of highly parallel and pipelined datapath units.

The control flow, addressing of data and coefficient and the operation of the datapath unit is entirely controlled by the macrocode. This guaranties a high flexibility of the DSP architecture during evaluation, development, test and working phase.

The macrocode concept reduces drastically the control and address generation overhead of the DSP system. This results finally in very low consumption and small size.

References

[1] R. E. Crochiere & L. R. Rabiner, “Multirate Digital Signal Processing”, Prentice-Hall, 1983.

[2] R. Brennan & T. Schneider “A Flexible Filterbank Structure for Extensive Signal Manipulations in Digital Hearing Aids,” Proc. of ISCAS-98, Monterey, CA.

[3] T. Schneider & R. Brennan “A Multichannel Compression Scheme for a Digital Hearing Aid,” Proc. of ICASSP-97, Munich, Germany.

[4] Schneider, R. Brennan, P. Balsiger, A. Heubi, F. Pellandini, "An Ultra Low-power Programmable DSP System for Hearing Aids And Other Audio Applications", Proc. of ICSPAT-99, Orlando, FL, November 1-4, 1999.

[5] A. Drollinger, C. Wälchli, D. Sun, L. Grisoni, C. Calame, A. Heubi, P. Balsiger, F. Pellandini, R. Brennan, T. Schneider, "An Ultra Low Power WOLA Filterbank Implementation in Deep Submicron Technology", Proc. of COST 254, Neuchâtel, CH, May, 5-7, 1999.