Parallel Processing for a DSP Application using FPGA

Nonel Thirer, *Member, IEEE,* and Aviram Souhami

Abstract — **In this paper we discuss a parallel architecture for an FPGA system including several embedded simple micro-processors (μP), for a digital signal processing application (DSP). Each μP in the system has a different purpose and a separate code unit, but all the μPs share the same data unit. The architecture of the μP can vary in type – it may be designed in the traditional form of a μP, or as a FIR filter, a video pattern generator and so on. Such systems can constitute a good solution when the DSP's main process can be divided into several processes. Every μP can be reprogrammed to perform more than one function, and a superscalar operation mode can be introduced and controlled by the programmer. This type of platform was designed and experimented for an audio synthesizer system.**

Index Terms — **Audio synthesizer, FPGA, multi-processor, parallel processing.**

I. Introduction

Most DSP algorithms require complex tools and a massive amount of instructions. Often it is necessary to perform these algorithms in real time. The solution to that was to build an embedded system on a chip, which includes special digital hardware and a micro-controller or a micro-processor for better performance. In recent years, the FPGA/ASIC based systems drew a lot of attention, particularly with the introduction of the fast CMOS reprogrammable logic devices [1]-[4]. These allowed to manufacture FPGA based DSP devices with reprogrammable multi-processors.

II. System Architecture

An audio synthesizer system (fig.1) includes a user interface, an input block (including the data acquisition unit with one or more A/D converters), a DSP processing block and an output unit (including D/A converter and speaker).

During the synthesizing process, performed by the DSP block, the signal passes usually over three processing units (fig.2):

N. Thirer is with the Holon Institute of Technology, 58102 Holon, Israel (e-mail: Tirer_n@ hit.ac.il)

A.Souhami, is with Runcom Technologies, Rishon le Zion, Israel. (e-mail: aviram2k@gmail. com).

VCO (Voltage Controlled Oscillator), VCF (Voltage Controlled Filter) and VCA (Voltage Controlled Amplifier).

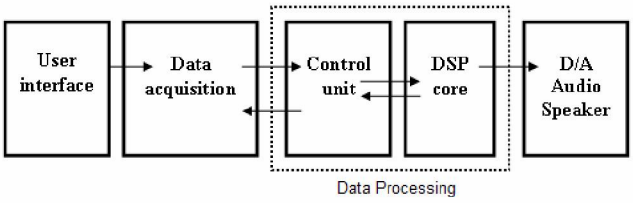

Fig.1 Audio Synthesizer System Block Diagram

Fig.2 Processing System Blocks

In order to simplify the implementation of this system on an FPGA platform and in order to permit a parallel execution of some phases, it is necessary to identify the common resources used in the process [5]. In the audio synthesizers, three basic functional units (fig.3) are used: an FM oscillator, a Filter and an Amplitude Modulator (AM-DSB-TC)

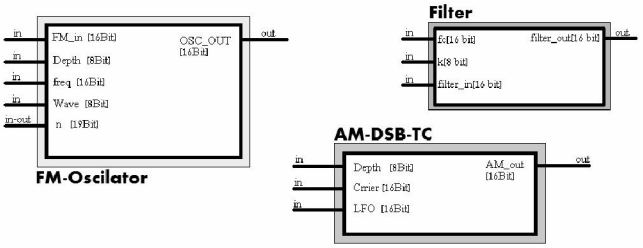

Fig.3 Basic Functional Units.

The idea is that every processing unit from fig.2 can be implemented by using the basic functional units from fig. 3. Thus the VCO stage is implemented using three FM oscillators (fig.4), the VCF stage is implemented using two FM oscillators, two AM Modulators and a Filter (fig.5), the VCA stage is implemented using a FM oscillator and a AM Modulator (fig.6).

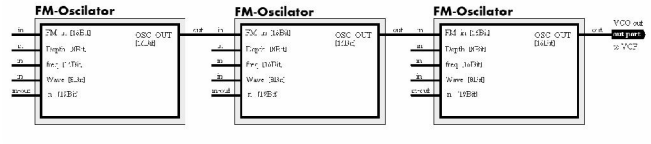

Fig.5 VCO Architecture

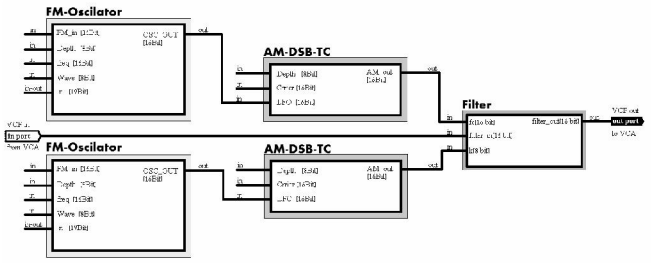

Fig.5 VCF Architecture

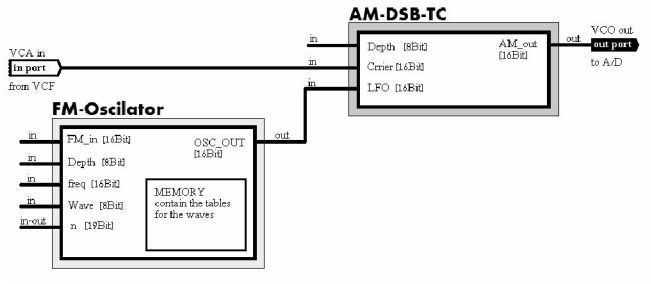

Fig.6 VCA Architecture

III. AN FPGA IMPLEMENTATION OF THE DSP BLOCK

The platform, as shown in figure 7, contains three software defined μPs with code segment on ROM device and a controller unit witch contains a small RAM memory. Every μP is programmed to perform one or every one of the DSP's base functions.

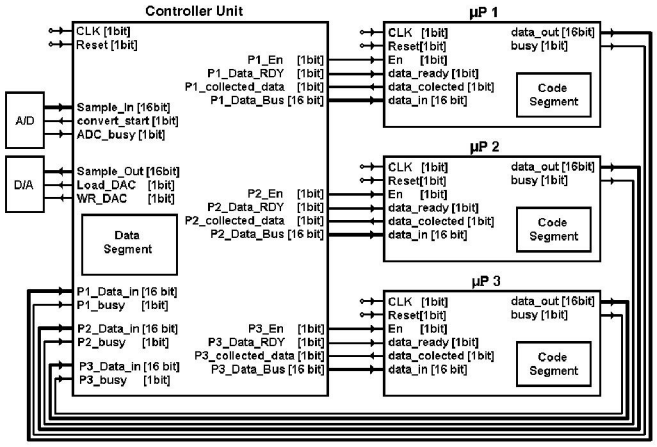

Fig.7 A FPGA Implementation of the DSP Processing Block

The main controller is responsible to data acquisition (reading samples from the A/D converter), data output to speaker (via D/A converter) and the "coordination" of the microprocessors' work. For this, the main controller unit loads one (or more) μPs with the data it needs for proper operation and enables the μPs to start their operation. The controller will wait until the μP finishes its process. Then it will collect the results and will configure again the same μP and/or others μPs.

Using only three types of μPs it is possible to perform all the required functions of the DSP's processing. Using identical μPs (by implementing all the functions in every μP) working in parallel makes it possible to improve performance in the VCF phase. Moreover, in our platform, a VCO phase of a new input data can be processing in parallel with the VCA phase of the last input data, increasing the throughput of the system, due to the pipelining operation of the system [5].

IV. DSP MICROPROCESSOR UNITS

In order to perform the specific DSP functions (oscillator, filter and amplitude modulator), the microprocessors were programmed as RISC (reduced instruction set computer) processors.

The algorithm of every μP is :

- Read data (including the function to be executed) from the controller unit and store it in registers or in a small memory.
- Wait for a synchronous event to start data processing.
- Start the process execution and generate a "Busy" signal.
- Complete the process execution, deactivate the "Busy" signal and write the result to the output register.

To communicate with the controller, the μP uses "data_in" and "data_out" ports and "data_ready" and "data_collected" control signals (fig.9), in a two wires handshake protocol. In this manner, the controller informs the microprocessor that new data is available (data_ready='1'). The μP reads the data from the bus and answers to the controller that the data was collected (data_collected='1'). Then the controller finishes the communication by sending data_ready='0' and the μP finishes the communication by sending data_collected='0'.

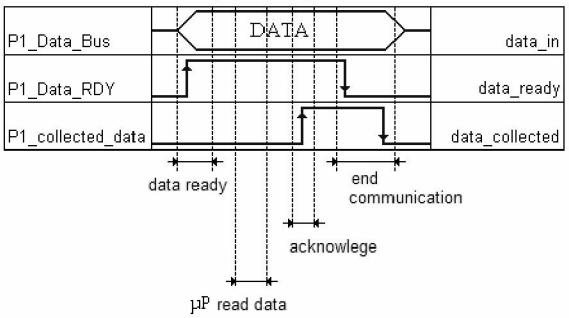

Fig.8 Communication Timing

V. THE CONTROLLER UNIT

The Controller Unit (CU) is the heart of the system. This module is used to transfer data between the memory, the DSP μP and the I/O modules, as well as to control each of the DSP's μPs module activity.

The CU module performs none of the DSP's operations, and this module behaves like a microcontroller without an arithmetical and logic unit (ALU). The DSP processors are the ALU of this module, but a generic ALU can be planted if needed.

The CU includes a "code ROM", which operates as a "code memory", and a state machine, that decodes the data from this memory (fig.9).

The module performs the following principal tasks:

- Moving data from memory to a specific DSP module or I/O and enabling it.
- Moving data from a specific DSP module to memory.
- Waiting for an event in "interrupt" ports which connects to the finish ports of the DSP modules.
- Enabling a specific DSP module or resetting it.
- Stopping and waiting for "new sample" events (start reading ROM from the beginning).

A reduced instruction set was implemented in order to simplify this module as much as possible. A better instruction set, including, for example, "call" and "jump" operations, can be added if the code that is used to control the system has specific subroutines or complex instructions that are repeated. In this manner we will need more logic elements, but a less ROM capacity and a smaller memory interface will be necessary.

The control unit includes two main processes (the "DECODE" process and the "READ" process) and a ROM which contains the instruction code. Every process is implemented as a state machine.

The "Decode" process includes three states:

- Idle state: the process waits for an event in the port "new sample", connected to the sample clock, which sends a pulse when a new sample is produced.
- Decoding state: after an event of new_sample has occurred, the machine starts decoding the data coming from the ROM data bus. The machine knows that the data in the port is ready when "data ready" = '1'.
- Waiting state: the machine waits to a specific hardware event to rise to '1'. This mode is useful when we want to collect one of the modules output data.

The "Read" process also includes three states:

- Reading state: in this mode the machine increases the address counter every clock.
- Restart state: the machine enters to this state when wait4enable='1' and resets the address counter. When wait4enable='0' the machine goes to reading state.
- Rewind state: when wait4event='1' then address counter is decreased by 2. The Decode process is now waiting for a hardware event and didn't fetch the last two addresses, so by decreasing the counter we can be sure that the instruction will be made in the right order.

These processes work in pipeline mode, increasing the system's performance.

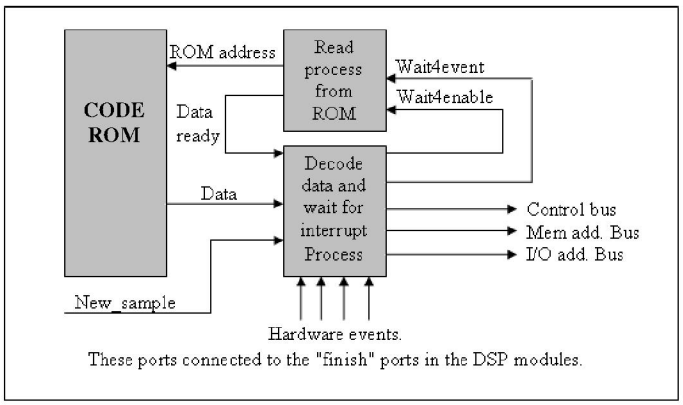

Fig.9 Controller Unit Architecture

VI. FPGA IMPLEMENTATION

The architecture of this system and the algorithms for the controller, for the microprocessors and for the communication between controller and microprocessors, were implemented and successfully tested using an ALTERA NIOS STRATIX development Kit and 10 bits parallel A/D and D/A converters (fig.10). To interface with the A/D and D/A converters, special controllers were implemented (A/D Ctrl, D/A Ctrl). Also, a dual port RAM was implemented to store the input data.

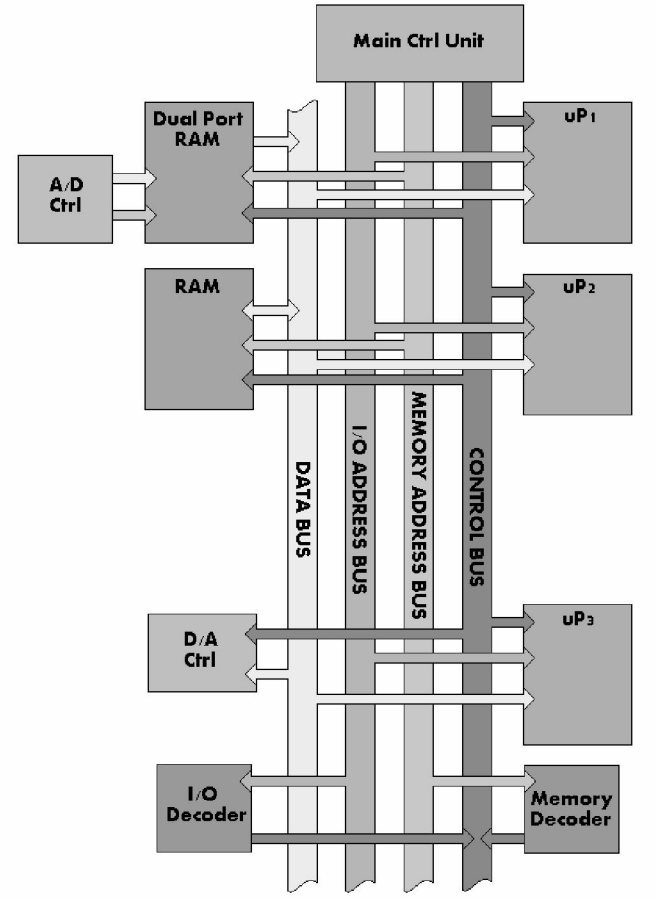

Fig. 10 System Architecture

VII. Conclusion

Our DSP algorithm for the synthesizer system is composed of three separate processes. In some cases, two or three processes will work in parallel to improve performance.

The Controller and the DSP microprocessors are programmed as RISC processors, to simplify the FPGA platform.

The parallel processing of data is provided by the architecture of our platform and the pipeline execution of controller unit processes and of the DSP modules.

References

[1] B. V. Herzen, "Signal Processing at 250MHz using High-Performance FPGAs" ACM International Symposium on FPGAs, 1997, pp. 62-68.

[2] T.S. Hall, D.V. Anderson, "A Framework for Teaching Real-Time Digital Signal Processing with Field-Programmable Gate Arrays", IEEE Trans. On Education, vol.48, 3, 2005, pp 551-558,

[3] Z.A.Zamindar, "Signal Processing Capability with the NuHorizons Spartan-3 Development Board" ,Xcell journal, 52,2005,pp.28-30

[4] M. Pradhan, "Simplified Micro-controller & FPGA Platform for DSP Applications", Proceedings of the 2005 IEEE Int'l Conf. on Microelectronic Systems Education (MSE'05), vol IV, pp 544-549.

[5] N. Thirer, I. David, I.Baal Zedaka, Uzi Efron, "Improvement of FPGA Pipelines Implementation" , SPIE Conf. "Optical Engineering and Instrumentation" San Diego, CA, USA,13-17 aug.2006 , paper 6294-37.