THE CPLD AS A GENERAL PHYSICAL MODELING SYNTHESIS ENGINE

Sergio D. Baron

Universidad Nacional de La Plata
Centro de Técnicas Analógico Digitales (CeTAD)
Calle 48 y 116, La Plata 1900, Argentina
s.d.baron@ieee.org

Daniel A. Gil

Universidad Nacional de La Plata
Centro de Técnicas Analógico Digitales (CeTAD)
Calle 48 y 116, La Plata 1900, Argentina
dgil@gioia.ing.unlp.edu.ar

ABSTRACT

In this paper we propose a system based on a Complex Programmable Logic Device (CPLD) as a physical modeling synthesis engine and a hardware description language (VHDL) to implement the physical modeling synthesis algorithms. An evaluation of VHDL and CPLD technologies for this application was performed. As an example we have programmed the Karplus-Strong plucked string algorithm using VHDL on an Altera CPLD.

1. INTRODUCTION

When implementing algorithms, one has the choice of hardware or software implementation. Before the existence of high-level hardware description languages, hardware algorithm implementation was costly and a very long process. Now with the powerful VHDL description languages and the availability of high density CPLDs, it can be a suitable choice.

We have found that physical modeling synthesis (PM) algorithms are very suitable for hardware implementation. In PM algorithms one will find mainly delay lines, look-up tables (LUT), simple addition operations, simple multiplication and division operations, etc., all of which are very easily implemented in hardware.

In this paper we don't compare between the combination of VHDL and CPLD implementation of PM algorithms and other - for example DSP - implementations. We just demonstrate the use of a proven technology in the field of music synthesis.

2. THE PM ALGORITHM IMPLEMENTATION IN VHDL

In the process of implementing PM algorithms in VHDL [1] we are faced with two tasks. One is to map the basic signal processing elements to their hardware equivalent, and the other is to program the algorithm itself using those building blocks. Since there are many elements that are common to most of the PM synthesis algorithms, we use the ability of VHDL to build proprietary libraries, which in turn are called on the main VHDL model.

When a large library of basic PM synthesis building blocks is completed one can program almost any synthesis engine using a relatively small number of code lines. We can even envision a visual programming environment, similar to CCRMA's Synth Builder [2], whose output is VHDL code for the programmable platform.

3. THE PROGRAMMABLE LOGIC (CPLD)

Since the introduction of the CPLD to the market, there have been improvements on the density and speed of this type of device. We have reached a point where the programmable logic devices are widely used in diverse designs, from reprogrammable CPUs and reconfigurable radio (software radio) [3], to music synthesis.

For hardware implementation we use Altera devices. We choose this specific brand because we have used these devices on different projects, we know the architecture of the different families of Altera CPLDs and mainly because it has a very effective programming environment.

It is clear we need a microgranular programmable logic architecture with a flexible signal path allocation capability, so we use the FLEX family of Altera devices [4]. These programmable chips use an external configuration memory, this memory can be an EEPROM or a RAM. So it is possible to use a double port RAM memory for configuration, making the system reprogrammable on the fly and enabling its use on a PC-based synthesis board or a stand alone synthesizer that could change from timbre to timbre at the push of a button.

4. IMPLEMETATION OF THE KARPLUS-STRONG PLUCKED STRING ALGORITHM

To probe the feasibility of our project we had to try to implement a physical modeling synthesis algorithm. The Karplus-Strong (K-S) plucked string algorithm [5] was the logic choice: it started the modern waveguide filter [6] physical modeling synthesis theory and it is very simple and straightforward to implement.

4.1. The basic Karplus-Strong algorithm

The K-S algorithm started as a modification of the simple wavetable synthesis. Kevin Karplus and Alex Strong added a modifier consisting of a FIR filter in the form of a mean value operator and a feedback loop. This is represented by the following function:

$$Y_t = {}^{1}/_{2} \; (Y_{t-p} + Y_{t-p-1}) \qquad (1)$$

For an easier hardware mapping figure 1 represents the block diagram of the algorithm.

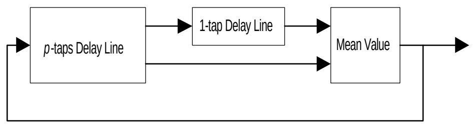

Figure 1. *Block Diagram of the basic K-S algorithm.*

4.2. Mapping the algorithm to the hardware using VHDL.

For this example we implemented the algorithm shown in figure 2, excluding the noise generator:

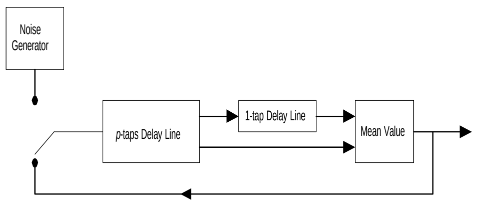

Figure 2. *Block Diagram of the implemented algorithm.*

4.2.1. The Library

We built a basic library where some basic components were implemented:

```
-------------------------------------------------------
    LIBRARY ieee;
    USE ieee.std_logic_1164.all;
-------------------------------------------------------
    PACKAGE pm IS
      COMPONENT Adder8a
        PORT(dataa,datab:IN STD_LOGIC_VECTOR(7 DOWNTO 0);
             aclr:       IN STD_LOGIC := '0';
             clock:      IN STD_LOGIC := '0';
             cin:        IN STD_LOGIC := '0';
             add_sub:    IN STD_LOGIC := '1';
             result:     OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
             cout, overflow: OUT STD_LOGIC);
      END COMPONENT;

      COMPONENT Mux8
          PORT(Noisy,Plunky:IN STD_LOGIC_VECTOR(7 DOWNTO 0);
               selm1:       IN STD_LOGIC;
               Outy:        OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
      END COMPONENT;

      COMPONENT Divis
          PORT(D:           IN STD_LOGIC_VECTOR(7 DOWNTO 0);
               cin:         IN STD_LOGIC;
               pre:         IN STD_LOGIC;
               clr:         IN STD_LOGIC;
               load:        IN STD_LOGIC;
               res:         OUT STD_LOGIC;
               Q:           INOUT STD_LOGIC_VECTOR(7 DOWNTO 0));
      END COMPONENT;

      COMPONENT Shifter
          PORT(D:   IN        STD_LOGIC_VECTOR(7 DOWNTO 0);
               pre: IN        STD_LOGIC;
               clr: IN        STD_LOGIC;
               load:IN        STD_LOGIC;
               Q:   OUT       STD_LOGIC_VECTOR(7 DOWNTO 0));
      END COMPONENT;

      COMPONENT barrelns
          PORT (D:    IN      STD_LOGIC_VECTOR(7 downto 0);
                pre:  IN      STD_LOGIC;
                clr:  IN      STD_LOGIC;
                load: IN      STD_LOGIC;
                Q:    OUT     STD_LOGIC_VECTOR(7 downto 0));
      END COMPONENT;

   END pm;
```

Adder8a is an eight-bit adder with carry-out from the Altera library. This is used in the mean value operation.

Mux8 is an eight-bit wide 2-in 1-out multiplexer that is used to select the input to the p-taps delay line as seen on figure 2.

Divis is nine-bit register that divides its input by 2. This is used in the mean value operation.

Shifter is an eight-bit-wide shift register used to map the p-taps delay line.

Barrelns is an eight-bit-wide shift register used to map the 1-tap delay line.

4.2.2. The K-S algorithm architecture implementation.

This is the VHDL structural description code for our K-S algorithm architecture implementation:

```
----------------------------------------------------------
-- K-S algorithm.
----------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
library work;
USE work.pm.all;
----------------------------------------------------------
ENTITY Pmunit2 IS
      PORT( selmux              : IN         STD_LOGIC;
            preshi,clrshi,ldshi : IN         STD_LOGIC;
            prebns,clrbns,ldbns : IN         STD_LOGIC;
            zero                : IN         STD_LOGIC;
            predv,clrdv,lddv    : IN         STD_LOGIC;
            aclr,clock,add_sub  : IN          STD_LOGIC;
            Noisy         : IN STD_LOGIC_VECTOR(7 downto 0);
            overflow,res        : OUT         STD_LOGIC;
            Q             : OUT STD_LOGIC_VECTOR(7 downto 0));
      END Pmunit2;

ARCHITECTURE structure OF Pmunit2 IS
      SIGNAL ismp1 : STD_LOGIC_VECTOR(7 downto 0);
      SIGNAL ismp2 : STD_LOGIC_VECTOR(7 downto 0);
      SIGNAL ismp3 : STD_LOGIC_VECTOR(7 downto 0);
      SIGNAL ismp4 : STD_LOGIC_VECTOR(7 downto 0);
      SIGNAL ismp5 : STD_LOGIC_VECTOR(7 downto 0);
      SIGNAL ismp10 : STD_LOGIC;
      SIGNAL ismp11 : STD_LOGIC;

BEGIN

m8:  mux8 PORT MAP(noisy,ismp5,selmux,ismp1);
sh:  shifter PORT MAP(ismp1,preshi,clrshi,ldshi,ismp2);
bns: barrelns PORT MAP(ismp2,prebns,clrbns,ldbns,ismp3);
a8:  adder8a PORT MAP
   (ismp2,ismp3,aclr,clock,zero,add_sub,smp4,ismp10,overflow);
d: divis PORT MAP(ismp4,ismp10,predv,clrdv,lddv,ismp11,ismp5);

Q<= ismp5;
res<=ismp11;

END structure;
```

As seen in this code, the main architecture makes use of the library components and maps the input/output of each to perform the K-S algorithm.

Note: the code for the library components can be found at *http://www.ing.unlp.edu.ar/sergio/cpldpm.html*

To simplify its interpretation this architecture is depicted in figure 3:

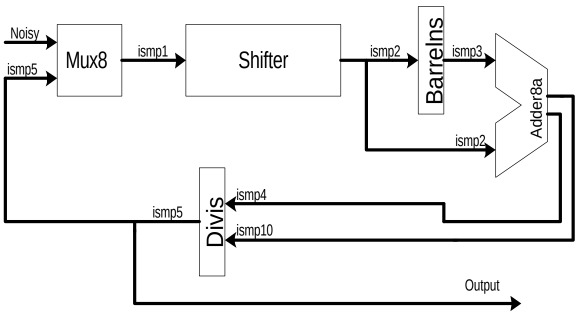

Figure 3. *Block Diagram of the K-S algorithm architecture*

4.3. Testing

While working on this project we used the ability of the development environment to test designs directly mapping to a CPLD, so the output of the simulation is guaranteed to be exactly the output of the actual hardware. We have pre-fed the delay-line with a random wave and run the circuit many times obtaining mostly very pleasant plucked string sounds. Figure 4 depicts a time domain waveform of an example:

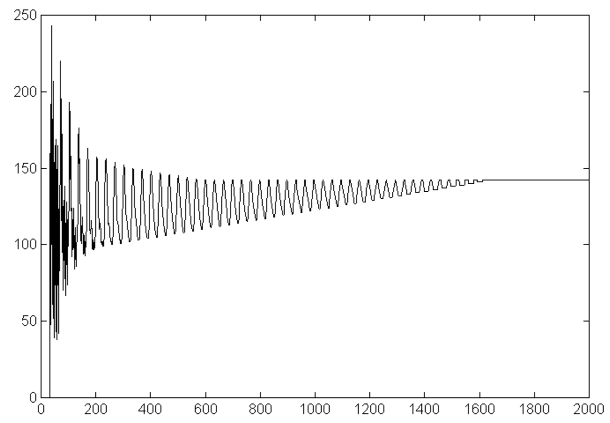

Figure 4. *Plucked string sound generated using a CPLD*

At the time of this writing we are testing the algorithms on the Altera educational boards and there's work done on a hardware prototype of our system. These examples can be heard at:

http://www.ing.unlp.edu.ar/sergio/cpldpm.html

5. CONCLUSIONS

In this paper, we propose the use of VHDL and programmable logic for physical modeling synthesis. A basic PM component library was compiled and an architecture for the Karplus-Strong plucked string algorithm was designed.

While working on the K-S implementation we were faced with some drawbacks, such as synchronization problems, glitches, arithmetic overflows, etc. None of which are found in software implementations. In contrast we have a system capable of generating one sample word per clock cycle for the K-S synthesis algorithm.

Since the above mentioned problems were solved, we are now working on this technology implementing many other PM algorithms.

6. ACKNOWLEDGMENTS

The basis for this work was developed by the first author when he was a visiting scholar at Stanford University CCRMA's "DSP for Synthesis: Physical and Spectral Modeling" workshop in the summer of 1995. The authors would like to thank professors Perry R. Cook for his support and Julius O. Smith III for advice.

7. REFERENCES

[1] Compass Design Automation, *VHDL Scout: A Practical Introduction to IEEE Std 1067-1987, the VHSIC Hardware Description Language.*

[2] http://www-ccrma.stanford.edu/
CCRMA/Software/SynthBuilder/SynthBuilder.html

[3] Mark Cummings, Shinichiro Haruyama, "FPGA in the Software Radio", *IEEE Communications Magazine, Vol. 37 No. 2, February 1999.*

[4] http://www.altera.com

[5] Kevin Karplus, Alex Strong, "Digital Synthesis of Plucked-String and Drum Timbres", *Computer Music Journal, Vol. 7, No. 2, summer 1983.*

[6] Julius O. Smith III, "Music Applications of Digital Waveguides", *Technical Report Stan-M-39, CCRMA, Department of Music, Stanford University. Stanford, California.*