One of the most interesting applications of Digital Signal Processing is Low-Level Speech Recognition. This paper will describe the design of a whole structure made up of two substructures: an LPC filter to compact the information present in the voice trace and a Bit Serial Neural Network which is used for the phonetic encoding of this trace. These substructures are designed with a Silicon Compiler adapted for Digital Signal Algorithms and Bit-Serial Architectures.

Introduction

Silicon compilation is a technique for producing automatic synthesis of integrated circuits layout from their description at the level of System Behaviour. A first approach to general silicon compilation is FIRST (Fast Implementation of Real-Time Signal Transforms) which was designed mainly for Digital Signal Processing (DSP) applications [3]. With this Silicon Compiler many DSP applications can be described strictly in terms of Data-Flow Graphs, by interconnecting their functional units within a given network. FIRST uses bit-serial operators to implement its functions, and this often yields a better fabrication cost/function ratio. Moreover Bit-Serial Architectures are distinguished by their communication strategy, allowing good interconnections within and between VLSI chips and tightly pipelined arithmetic structures. Our approach follows the method of top-down design described by the following steps:

1. Algorithm (Rec. Equations) → Computing Architecture (Virtual Architecture)
2. Computing Architecture → Signal Flow-Graph (FIRST Description)
3. Signal Flow-Graph → Primitive List (VIRTUAL Architecture)
4. Primitive List (FIRST Description) → Functional Description

Finally the process of instantiating the structure is a bottom-up implementation, which is automatically carried out by the tool. The main problem found in the implementation of a design in VLSI is to derive a computation scheme and a corresponding architecture which can eventually be expressed as a hardware flow-graph [1]. This approach consists in reformulating the linear difference equation of the DSP algorithms or some equivalent representation of the problem as a recurrence. That is, we identify any set of repeated operators to implement its functions, and this often yields a better fabrication cost/function ratio. Moreover Bit-Serial Architectures are distinguished by their communication strategy, allowing good interconnections within and between VLSI chips and tightly pipelined arithmetic structures. Our approach follows the method of top-down design described by the following steps:

1. Algorithm (Rec. Equations) → Computing Architecture (Virtual Architecture)
2. Computing Architecture → Signal Flow-Graph (FIRST Description)
3. Signal Flow-Graph → Primitive List (VIRTUAL Description)
4. Primitive List (FIRST Description) → Functional Description

These processors may be designed independently from the networks that support them. Ultimately the process must be realized as a flow-graph of simple functional elements taken from the base library.

- Implementing state memory: multiplexing and net synthesis: Multiplexers are connected at the inputs of the relevant processors. Associated with such a processor a fixed amount of "virtual" state memory is required. This memory can be implemented by the inherent processor latency and by the addition of word and bit delays that are FIRST primitives.

- Implementing the control network: Each primitive permits control requirements that must be fulfilled by the system design as a secondary control network that has no nodes in common with the signal net and may be viewed as an independent design [7].

In our case, we have used this style of design supported by FIRST to develop two different structures to be used lately in Speech Processing, these being:

A) An Adaptive Lattice Filter for LPC Extraction.

B) A VLSI Bit-serial Neural Network to support the Phonetic Encoding of LPC vectors.

Design of an adaptive lattice filter

This lattice structure is a form of a prediction error (PE) filter. Its applications are found in the equalization of voice channels, spectral analysis and spectral shaping. In our case, it is being used as an LPC filter, in which the speech spectrum is obtained to be used in phonetic encoding instead of the signal itself. The lattice coefficients, $k(i)$, $i = 0, 1, \ldots, n$, are a compact representation of the original signal, the so-called PARCOR coefficients [6]. The algorithm being implemented is taken from Satorius and Alexander [6], its general structure being the following:

1) All signals and coefficients are set to 0.
2) The input is assigned to both forward and backward traces:
   \[ f(0,t) = b(0,t) = u(t) \]
3) The forward and backward traces are updated:
   \[ b(i,t) = b(i-1,t-1) - k(i,t)f(i-1,t) \]
   \[ f(i,t) = f(i-1,t) - k(i,t)b(i-1,t-1) \]
4) The new PARCOR coefficient is evaluated:
   \[ k(i,t) = k(i,t-1) + \mu(i,t) \cdot [f(i,t)b(i-1,t-1) + f(i-1,t)b(i,t)] \]
5) The step size, $\mu$, is updated, according to:
   \[ \mu(i,t) = \mu(i,t-1) [1 + \alpha - \beta u(i,t-1) \cdot f^2(i,t) + b^2(i,t)] \]
6) Repeat from 3) for $i = 1, \ldots, n$. 

The new PARCOR coefficient is evaluated:
The virtual architecture corresponding to expressions (2) and (3) in the above recursion may be seen in Fig. 1.

The technical specifications of the design are the following:

- Sampling frequency: 20KHz
- Number of stages: 16
- Input/Output precision: 12 bits

The system wordlength is 26 bits in fixed point and two's complement format.

Fig. 1. Signal Flow-Graph

Fig. 2a and 2b show the Flow-Graphs corresponding to the implementation of expressions (4) and (5).

Fig. 2a. Flow-Graph of the Lattice Algorithm.

Fig. 2b. Flow-Chart of Step-Size Recursion.

The Computing Architecture chosen is intended to support the multiplexing of sixteen virtual processors on the same physical processor, at a bit rate of 8320 MHz, which is required to operate 26 bits data formats at the input sampling frequency, if the requirements of the 5u cell library are to be met, concerning maximum clock speeds. In order that the algorithm could be partitioned into its arithmetic components, five chips will be necessary. Memory and multiplexing elements must be added up to these operators. The five Arithmetic Function Chips will be the following:

- Lattice chip (CL)
- PARCOR chip (CK)
- Step-Size chips (CMU1 and CMU2)
- Control chip (COS)

In the present paper we will give a brief description of the Lattice Chip and its related Lattice Operator, the rest of the designs following the same philosophy, although they will not be given here. The Flow-Graphs of the Lattice Chip and the Lattice Operator primitive may be seen in Fig. 3 and Fig. 4.
What follows is a brief description of both flowgraphs in FIRST-style:

OPERATOR LATTICE (ctrlin -> ctrlout) fin, k, bin -> fout, bout
SIGNAL lat1 THROUGH 4
MULTIPLY [1, 12, 0, 0] (ctrlin -> NC) k, fin -> lat2, NC
MULTIPLY [1, 12, 0, 0] (ctrlin -> clat1) k, bin -> lat3, NC
SUBTRACT [1, 0, 0, 0] (clat1) lat1, lat3, GND -> fout, NC
SUBTRACT [1, 0, 0, 0] (clat1) lat4, lat2, GND -> bout, NC
BITDELAY [20] fin -> lat1
BITDELAY [20] bin -> lat4
CBITDELAY [1] (clat1 -> ctrlout)
FND
CHIP CL (pevent, pctrl1, pctrl2 -> pctrl3) pk, psigin
-> pfou1, pfout, pbout, pbim
SIGNAL k, sigin, lfin, lk, lbin, il, l2, fin, fot, bin, bout
CONTROL ctrl1, ctrl2, ctrl3, event, cl1, cl2, cl3
PADDIN (pctrl1, pctrl2, pevent -> ctrl1, ctrl2, event)
-> pk, psigin -> k, sigin
PADDOUT (ctrl3 -> pctrl3) fin, fot, bout, bin -> pfou, pfout, pbout, pbim
PADOORDER VDD, pfou1, pfout, pbout, pbim, pctrl3, GND,
CLOCK, pk, event, pctrl1, pctrl2, psigin
MULTIPLY [5, 0, 0] (ctrl2) sigin, fot -> lfin
MULTIPLY [5, 0, 0] (ctrl2) sigin, fot -> lfin
MULTIPLY [26, 0, 0] (cl1) l1, GND -> l2
FLIMIT [26, 10, 0] (cl2) lfin -> fot
LATTICE (cl2 -> clat1) lfin, lk, lbin -> fot, bout
FLIMIT [26, 10, 0] (cl2) lbin -> bin
RITDELAY 151 k -> lk
WORDDELAY [15, 12, 0] (cl2) l2 -> lbin
CBITDELAY [5] (event -> cl1)
CBITDELAY [5] (ctrl1 -> clat)
END

Preliminary design of a VLSI bit-serial neural network

Through the present section, the design of a Neural Network is being conducted using the same philosophy exposed in precedent sections. The resulting Neural Network chips, together with the LPC encoder, will constitute a complete Phonetic Coder for Spanish in real time. Some words would have to be said concerning the implementation of Neural Networks using the serial approach. According to Buttler: "...an advantage of bit-serial arithmetic in a neural network is that it minimizes the interconnect requirements by eliminating multi-wire buses. Pipelining makes optimal use of the high bit-rates possible in serial systems allowing good communications within and between VLSI chips." [2].

The functions that a neural network simulate are the ability to analyze many solutions simultaneously, the work with corrupted data and the fault tolerance behaviour. We consider a neuron as a state machine that is either "on" or "off". The synapses weight the signals from a transmitting neuron and they become more or less excitatory or inhibitory to the receiving neuron. The set of synaptic weights determines the stable states and represents the information stored by the system [8]. The neural state is:

$$V_j = h(x_j) = f(T_{ij}V_j + I_j)$$ (6)

where $I_j$ is a direct input that may be arbitrarily strong to force some value on $V_j$. As a first goal, we have implemented a neural network with fixed functionality and modest number of synthetic neurons, the building of a dynamically programmable network being an objective for future research. The whole neural architecture may be seen in Fig. 5.

The architecture we have taken as a reference is the proposed in [2], in which, a neuron is represented by a circle with its column of "n" synapses (squares) communicating among each other. Every synaptic operator adds the weighted contributions from other neurons down the column. We have introduced several modifications to that design in the Synaptic Operator with a 5-state activating function to be adapted to the Silicon Compiler being used. The Synaptic Operator will have the structure shown in Fig. 6.

Fig. 5. Architecture of the Neural Network for n=4.

Fig. 6. Synaptic Operator.

The structure above described may be easily implemented using the Silicon Compiler. We can see this structure as a graph with some special characteristics. Each synaptic operator must implement two functions:

1) Multiply the state of the neuron $V_j$ by the synaptic weight
2) Add the product to the node global activity.

The 5-state activating function allows us to use reduced arithmetic, thus the multiplication of a synaptic weight by $V_j$ simply requires a right-shift of
1 bit (X0.5) or 2 bits (X0.25). The arithmetic resources in the synaptic operator are the following:

An 8-bits shift register: The purpose of this register is to hold the synaptic weights. We may implement this element by bit-delay primitives.

A Bit-serial Add/Subtract: The use of the FIRST Primitives ADD and SUBTRACT is enough to design this unit. These arithmetic elements generate the second operation of the synaptic operator.

Control signals: To implement the multiplication by \( V \) we must create with the control-generator signals that produce one or two shifts depending on the 5-states signals which come from the control of the activating function. In the adder and subtractor outputs we use a third signal from the control of the activating function \( V_t \) as an input signal to the second multiplexer which will choose between the output of the adder or the subtractor to carry the result to the following operator.

What follows is a FIRST-style coding of the synaptic system:

```
OPERATOR SINAP (ctrl1, ctrl2 ->) in1, in2 -> out
SIGNAL in1, outmux11, outmux12, outmux122, inadd, inmux21, inmux22, inmux3
CONTROL ctrl1, ctrl2
BITDELAY [16] in1 -> in1
CBITDELAY [16] (ctrl1 -> c1)
DSHIFT [2,0] (c1) in1 -> outmux11
DSHIFT [1,0] (c1) in1 -> outmux12
BITDELAY [1] outmux112 -> outmux122
MULTIPLEX [1,0,0] (ctrl12) outmux11, outmux122 -> inadd
CRITDELAY [1] (c1 -> c12)
ADD [1,0,0,0] (c12) inadd, in2, NC -> inmux21, NC
SUBTRACT [1,0,0,0] (c12) in2, inadd, NC -> inmux22, NC
MULTIPLEX [1,0,0] (ctrl12) inmux21, inmux22 -> inmux3
MULTIPLEX [1,0,0] (ctrl12) NC, inmux3 -> out
END
CHIP CHISINAP (pctr11, pctr12 ->) pin1, pin2 -> pout
SIGNAL in1, in2, out
CONTROL ctrl1, ctrl2
PADIO (pctr11, pct212 -> ctrl1, c12) pctl1, pin2 -> in1, in2
PASOUND VDD, pin1, pin2, pctl1, pctl2, GND, CLOCK, pout
SINAP (ctrl1, ctrl2 ->) in1, in2 -> out
CONTROLGENERATOR (- NC, ctrl1, ctrl2)
CYCLE [16]
CYCLE [1]
ENDCONTROLGENERATOR
END
SYSTEM SSINAP sin1, sin2, sout
CONTROL sctr11, setcl2
CHISINAP (sctr11, setcl12 ->) sin1, sin2 -> sout
END
```

We must take into account that a full synaptic computation requires two complete shift register cycles. During the first cycle the synaptic weight is multiplied by the neural state and during the second, the most significant bit of the resultant \( \hat{V}_{ji} \) is sign-extended on the rest of the shift register. This requires 8 bits more to take into account the sign extension in the ongoing addition.

**Final structure and conclusions**

Through this paper we have shown the design of two Bit-Serial Architectures, for the purposes of voice encoding. These designs would be incorporated in a global system, which may be seen in Fig. 7.