AN INVESTIGATION INTO USING EXPERT SYSTEMS FOR DIGITAL FILTER DESIGN

Ashok Krishnan and Kurt Kosbar
Department of Electrical and Computer Engineering
University of Missouri – Rolla

ABSTRACT

It can be challenging to select the best architecture for DSP filters for a given application. Design constraints often include both objective and subjective information. This paper discusses the initial results of an investigation into using expert system techniques to address this problem. The goal is a system that allows users to specify traditional constraints such as impulse response, frequency response, stability, SNR, etc., but they may also constrain the filter’s cost, complexity, or any parameter which can be clearly identified for the specific application.

KEY WORDS


INTRODUCTION

There are typically many ways to solve design problems. While some solutions will be clearly superior to others, there is usually no single, globally optimal, answer. A human expert in the field will learn the trade-offs of the particular problem, and understand the state of the art enough to avoid the grossly sub-optimal solutions. However different experts may select significantly different solutions, based on their experience, amount of effort put into the design, willingness to accept risk, and a host of other factors. As the complexity of problems grow, and as the amount of information relating to possible solutions also increases, it becomes difficult for any one person to remain abreast of all the possible methods for solving a design problem.

If the design constraints are well defined, and limited in scope, the complexity can be managed through the use of tools such as flow diagrams or other reasonably simple algorithms that will guide the designer toward the most useful solutions. These tools fail as the number of constraints increases, the number of options for the design increases, or when the constraints are poorly defined or not well understood by the designer. This is where a human expert is most useful. The expert can use his/her experience to help the design prioritize the requirements, and can fill in reasonable parameters where the designer is unsure of the exact requirements.
Expert systems (ES) is a branch of computer programming, and in particular, part of the area of artificial intelligence, that attempts to mimic the behavior of human experts. An expert will make decisions based not only on objective facts (which everyone agrees upon) but also on a set of heuristics, or “rules of thumb” which seem to work for him/her. Some of these rules and heuristics will be easy for the expert to identify, but others are so complex or subtle that the expert themselves may not be able to explain them. Human experts may express discomfort with particular solutions to problems, without being able to describe exactly what feature about the solutions makes them uncomfortable. While this discomfort may just be because the approach is unconventional, it also could be because of a flaw or failure mode that the designer can sense, but not articulate. Expert systems can use heuristics, so the user can express a preference for a certain approach or technique, without completely eliminating sub-optimal solutions.

This paper represents an initial investigation into applying expert system techniques to the design of digital signal processing (DSP) filters. There are experts in widely varying areas all working on DSP filter design. These designs vary dramatically depending on the nature of the signal being processed, and the amount of resources that can be used to filter it. Solutions found in one area, say speech processing, may not quickly propagate to engineers working on areas such as wireless communications. This is especially true for junior engineers in these areas, who are often overwhelmed learning about the problem they must solve, that they lack sufficient time to determine how an approach developed for a radically different problem may benefit them. A successful ES that performs DSP design may be able to point out how the engineer’s problem is related to one in a different field.

The area of expert systems is quite broad, with many different programming techniques. In the following sections we will discuss some of the basic approaches to expert systems, and how these approaches may be useful in DSP filter design.

**INDUCTIVE REASONING**

Inductive reasoning is essentially a learn-by-example approach. Human infants routinely learn how to speak a language by simply listening to others. After hearing enough examples of how words should be pronounced and associated, they can infer many of the rules of the language, and produce sentences which are at least understandable, if not always perfectly correct. An inductive expert system attempts to learn in the same manner. It will need to be supplied with a large number of good solutions/designs, and ideally a large number of poor designs also. It enters these designs into a database, then attempts to discover rules which will allow it to find one of the good designs independently. This area is also known as machine learning.

To see how inductive reasoning might work in the DSP example, consider the difference between real-time DSP (RTDSP) and non-real-time work. In RTDSP a processor is typically connected to both an analog-to-digital and a digital-to-analog converter. Sampling rates for these converters are often in the kHz to MHz range. The latency between the time a signal is sampled and the output must be produced may be a single sampling time, up to a few seconds. In a non-real-time-DSP application, the data will be sampled and placed on a storage medium such as a
solid state memory, magnetic disk, magnetic tape, optical disk, etc. At a far later time (compared to the sampling rate) the data is read from memory, processed, and the outputs displayed or recorded again.

Suppose we described a number of these systems to an inductive reasoning expert system. We would need to describe parameters such as the sampling rates, order of filters used, architecture of the filters, latency between input and output, and the type of processor, algorithm, or technology used to implement the filter. We may also wish to enter parameters such as the development time, power, weight, ease of documenting and maintaining the algorithm, and even an estimate of the number of engineers who have experience with this approach. Many of the systems which had delays measured in msec would probably have solutions which used programmable DSP processors, field programmable gate arrays or application specific integrated circuits. However, non-real-time-DSP applications, which have latencies of days or longer, may use general purpose platforms such as personal computers, and general purpose programming languages such as C++ or one of the many excellent numerical analysis software packages available in the commercial marketplace.

There are good reasons for selecting a programmable DSP chip in a RTDSP application, and a high level language for stored-data applications. Typically a general purpose computer does not have the proper bus structure, operating system, and numerical analysis capabilities to keep up with real-time data. However a C++ solution to a problem is generally easier to create, debug and document, and is also more portable and scalable than solutions which require the use of a particular DSP chip.

In inductive ES systems, we do not try to explain these reasons to the program; we simply feed it a number of examples, each time telling it the latency between input and output, and the platform and programming language we used to solve the problem. After it has been fed these examples, the ES hopefully discovers that long latency problems may be handled on a PC, which short latency (real-time) problems are not. Ideally, the ES would get to the point that it would create an interview question for the user, asking them to describe the latency between input and output. A novice engineer might not realize this makes a big difference in the final solution to a problem. We may even wish to have the ES attempt to bias the user toward stored-data solutions, especially when the user did not express a preference. The computer could be told that operations on general purpose operating systems and high level languages are easier to write and maintain, so if the user didn't specify a preference for latency, the computer would pick the C++/PC combination over writing a program in assembly for a particular processor. All this information it extracts from simply examining a number of DSP solutions.

One may wonder if it is reasonable to code an ES just to discover such a simple and obvious rule. The answer may be yes - because once coded the ES can learn as technology evolves. As DSP chips become faster, and coding tools improve, the data may show that engineers tend to implement more complex algorithms on the DSP chips as opposed to FPGAs or ASIC chips. It might also show a trend toward using C over assembly, C++ over C, DSP code generated by high-level tools over the others. It might also show a trend toward using DSP chips in non-real-time applications where the DSP board is an integral part of the PC. One might also see where the line is between ASIC chips, FPGA and programmable processors.
This is one of the goals of this project - to set up the mechanism which allows a program to discover a rule which we already know. Then if a large number of good designs can be fed into the inductive data base, we hope the program could discover new rules and relationships which we did not know, or track how the old rules change with time. Ideally, there would be correlations or rules which we never knew about that an inductive ES would discover.

A review of the literature has led us to be pessimistic about inductive systems discovering many significant new relationships that have escaped the notice of human experts. However, being able to the progress of technology appears to be well within the reach of these systems, if enough data is provided to the database.

SIMPLE RULE-BASED ALGORITHMS

One problem with inductive tools is that the programmer can only lead by example, the expert systems has to re-discover many of the rules that are already well known in an area. Inductive systems have great potential because they may discover correlations, or rules, which the original designers never knew about, but they typically need a very large number of examples (and some friendly nudging) to discover these rules. An alternative approach is to feed the expert system rules, rather than examples. In a simple rule-based algorithm, the ES is fed a series of if-then-else type rules.

In some ways, a rule-based expert system is similar to a procedural program, flow-chart or cookbook, which can be thought of as a series of rules or decisions. There is at least one significant difference - in an expert system one does not describe the order in which the rules will be executed. It is somewhat like a student listening to an experienced, but highly unorganized, instructor (similarities to the authors of this paper are purely coincidental). The student creates a database of such rules, although they are not in any particular order. When the student faces a design problem, he/she starts looking at the rules one at a time, to determine which of them apply to the problem at hand. If the data base of rules is sufficiently large, the student may be able to discover that only a few possible solutions satisfy all the rules that apply to the problem.

Of course a good student doesn't just copy down rules and file them away - a good student tries to organize the rules in a manner that makes solving problems quicker. There are at least two ways to organize these rules, known as forward-chaining and reverse-chaining. In forward chaining, the student starts by examining the specifications for the problem, and then tries to figure out which of the possible solutions will meet these specifications. A reverse-chaining expert system will start by examining possible solutions (essentially guessing at answers) and then checking to see if the guess will meet the specifications well.

To see how one might use a rule-based algorithm, consider the problem of selecting between using Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) digital filters. One might enter a series of rules regarding these filter types into the expert system, such as:
1. If FIR then filter is always stable
2. If IIR then filter may or may not be stable (difficult to use if coefficients are adaptive)
3. If FIR then it is easy to control amplitude and phase response
4. If IIR then it is more difficult to control amplitude and phase response
5. If FIR then filter tends to require longer execution times
6. If IIR then filter tends to be faster
7. If FIR then it is easy to make phase linear
8. If IIR then it is easy to make minimum phase

When someone uses a forward-chaining expert system, they may enter the fact that the filter needs to be linear phase. The forward-chaining expert system scans the rule based, and decides it should be biased toward using FIR filters for this application.

While forward-chaining seems to be the most straightforward technique, in many ways a reverse-chaining technique is more useful. An inexperienced DSP designer may not know that FIR and IIR filters exist, or more likely does not appreciate the design trade-offs involved in selecting one filter type over another. In reverse chaining the expert system may "guess" the final design will be an IIR filter. Based on this guess, it may ask the designer if they will wish to adaptively change the coefficients of the filter, or if the filter must be linear phase. Based on the response to these interview questions, the expert system may be able to eliminate the IIR filter as a reasonable design option, or confirm that it is a good choice.

**HYBRID / OBJECT ORIENTED EXPERT SYSTEMS**

The inductive and rules-based expert systems are in some sense at opposite extremes. The inductive system accepts only examples and from that tries to infer rules. The rules-based system accepts only rules, and from that tries to determine good examples of systems. The hybrid expert system combines these two approaches, typically by exploiting the power of object oriented programming languages. The expert system uses a series of object, frames or schema which contain a series of slots. Each of the slots can contain a value, example, rule or another object. This approach allows the human expert to encapsulate, or categorize, information. One can also allow objects to inherit information from more general classifications.

As an example, suppose one has a solution for a DSP filter design that uses a particular brand and model of DSP processor to implement an FIR filter. The decision to use a particular DSP chip may be because the user has experience using that particular device, support tools are readily available, or the processor has been qualified for use in a particular application, product or environment. A very different set of rules where probably used when deciding upon using an FIR filter. In the filter architecture design decision one may have looked at parameters such as stability, complexity, flexibility, familiarity, sensitivity to coefficient errors, signal-to-noise ratios, etc.

An object-oriented expert system allows a human expert to separate the concerns listed above into classes. For example, there may be a “processor” class, which contains objects for each of the major DSP chip manufacturers. In each of these objects one may express the available of the
processor, computing power, ease of use, software support, if the processor is qualified for use in a particular environment, etc. Some of these parameters may be altered by the user of the expert system. If a particular user is very familiar with the XYZ DSP chip, they may prefer solutions that use that particular chip, even if other parameters associated with that device make it less favorable to use.

The object-oriented expert system may have yet another class that is concerned with filter architecture (FIR, IIR, etc.), where items like complexity and stability are issues, but one does not care about the platform that will be used to implement the filter.

The final solution to a DSP design problem would be yet another object, which would inherit all of the properties of the higher level classes which define it. The quality of the final solution could then be judged by examining all the properties of the solution object. In a reverse-chaining expert system, many solution objects may be created, and all evaluated to see how well they meet the specifications of the problem. In a forward-chaining system one would attempt to use the specifications to select the one best solution from the many choices.

**CONCLUSION**

It appears that expert systems may be able to assist engineers in designing digital signal processing devices. They could not only benefit novice engineers, but could potentially assist in keeping practicing engineers current, and introducing them to results from designers working in widely different sub disciplines of DSP. To be highly effective, the expert system would need to have a large number of rules, examples, and in the case of hybrid system, objects. This would make a detailed design either a formidable task for a small group of engineers, or push the design toward an open-source model that encourages involvement from a wide range of DSP designers.

**REFERENCES**


