A NOVEL APPROACH TO THE DESIGN OF DSP SYSTEMS USING MINIMUM COMPLEXITY FINITE STATE MACHINES

L.S. Dooley[1], A.C. Knoll[2], M. A. Wahab[1], A. Fauth[2] and M. Freericks[2]

[1]The Polytechnic of Wales
Dept. of Electronics and Information Technology
Pontypridd.Mid Glamorgan.CF37 1DL
Great Britain

[2]Technische Universität Berlin
Franklinstraße 28, Sekretariat FR 2-2
W-1000 Berlin 10
Germany

Abstract – The paper presents a new and different approach to the design and realisation of Digital Signal Processing (DSP) systems by utilising Finite State Machines (FSM). The DSP system is modelled by mapping all its potential states into an FSM, whose complexity is usually very high. The FSM mirrors the complete functionality of the system and thus describes its behaviour in full detail. Examples for FSMs of first and second order digital recursive filters are provided and the current version of the software simulating the FSM corresponding to any linear time-invariant DSP system is described. The potential of this approach including state reduction techniques as well as the inclusion of non-linear DSP systems is also outlined, and further future research intentions are briefly explored.

I. INTRODUCTION

Many of the current designs of DSP systems such as FIR/IIR digital filters and Fast-Fourier-transformers (FFT) are based on the translation of a signal processing algorithm into a hardware structure relying heavily on multipliers and adders. The advantage of this traditional approach is a relatively straightforward transformation from algorithm to hardware. However, its drawbacks are:

a) The structure of the hardware is usually highly irregular.
b) Many computations are performed repeatedly while the circuit is in operation, effectively reducing throughput. If all possible computations were performed prior to the transformation into hardware, it would obviously function much faster.
c) Due to the complexity of the hardware it is often very difficult to fully simulate and assess its run-time behaviour; hence the danger of possible unwanted limit cycle oscillations. Such effects often make it difficult or even impossible for the designer to fully predict the output of the circuit for any given situation.

The solution we propose to many of the problems encountered in the traditional design cycle is the use of finite state machines. Here, all possible states of the DSP system are first mapped into an FSM. This FSM is then examined for the effects of finite word length arithmetic, quantization errors and limit cycle oscillations. These effects are represented either by single states or whole groups of states. In a second step, depending on the requirements of the specific application, duplicate, redundant, unwanted and possibly erroneous states are removed algorithmically by employing reduction techniques. The third step is the transformation of the resulting minimum complexity finite state machine into a hardware circuit.

As can be seen from this cursory description, the FSM for non-trivial DSP-systems has a large but nevertheless finite number of states. In principle, each state as well as each state-to-state transition and its corresponding effect upon the output may be examined. The inputs and the outputs of the FSM are elements of finite sets (of numbers). The state transition mapping and the output mapping are defined over these finite sets. The mappings are operations of infinite accuracy and, consequently, free of all errors. By modifying the state machine it is possible to replace states that would cause the system to fail with "adjacent" states that do not cause erroneous system behaviour. The penalty paid is a slight general decrease in the systems's overall signal to noise ratio. Limit cycle oscillations, for example, can be detected using well-known graph search techniques, but, of course, may occasionally be too inefficient to be used in practice. Feeding the simulation output into an *acceptor* FSM could also give hints about the occurence of cycles. Therefore, under certain conditions, fully predictable behaviour can be guaranteed. Even though the final hardware may use more chip real estate when designed using an FSM model, its structure is very regular and erroneous outputs are much less likely. This is analogous to many computer programs, where storage efficiency achieved via algorithmic complexity can be sacrificed for speed and program safety.

In spite of the fact that VLSI technology may one day provide densities that allow for the one-to-one implementation of the most complex state machine, current technology is certainly not advanced enough to realise a state machine for a second-order 16-bit IIR filter, which would require the implementation of 2^{32} states. Obviously for this sheer number of states, the introduction of additional logic such as adders into the FSM design is inevitable in order to reduce the complexity (see below). Although in this paper we are dealing only with recursive filters, the approach may easily be extendend to any linear or non-linear DSP system with single-input single-output as well as multiple-input multiple-output [1]. Particularly interesting is the realisation of an FFT, which may be annotated as an IIR filter with complex coefficients by using Goertzel's algorithm [2].

II. PRELIMINARIES: FINITE STATE MACHINES

A finite state machine according to the Mealy model [1] is a mathematical model of a sequential system. It comprises a finite set of input values U (in our case this is the set of values the input signal can assume), a finite set of output values G (the output signal), a finite state set S and two mappings: the next state mapping f and the output mapping g. The latter are defined

0-7803-0593-0/92 $3.00 1992 IEEE

as: $s(n+1) = f[s(n), u(n)]$ and $g(n) = g[s(n), u(n)]$, where $u(n) \in U$, $g(n) \in G$ and $s(n) \in S$ are the input, output and present state of the system at the n^{th} clock cycle. The mapping f maps the present state and the input into the next state, while the mapping g produces the present output from the present state and present input. Several standard machines may be designed which are simple and easily implemented, while more complex FSM's may be realised by composition of finite state submachines (independent or interdependent).

III. A SIMPLE EXAMPLE: FIRST ORDER FILTER

An FSM may simulate a finite DSP system in which the output at any clock instant is a function of the past and present values of the inputs and machine states. When the continuous algebraic equation description of the system is given, the approach is to evaluate the system output for all possible finite inputs and system states. The calculated output will not be in the same finite set as the input and the current state; therefore, some form of approximation is necessary. However, the error of the system's output values will always be within ±0.5 LSB. Consider a first-order all-pole digital filter governed by the following difference equation:

$$y(n) = K_1 y(n-1) + x(n)$$

where the present output y(n) is a function of the past output y(n - 1) and the present input x(n). If we compare this simple difference equation with the definition of the FSM above then we get the corresponding FSM after the following steps:

1. Choose the word length of y(n) and x(n).
2. Define the quantisation method (roundoff, truncation, saturation).
3. Assign to each output quantisation level $\bar{y}(n)$ a member of the state set S and to each input quantisation level $\bar{x}(n)$ a member of the input set U of the machine.

The first-order filter may then be represented as a finite state machine and the next state may be obtained by quantising the infinite accuracy computation result of the difference equation

$$\bar{y}(n) = Q[K_1 \bar{y}(n-1) + \bar{x}(n)]$$

where Q denotes the quantisation operation. If the coefficient $K_1 = 0.55$, the input $x(n) \equiv 0$ (zero input sequence) and the output is represented in a 3-bit word, then the state table may be compiled as shown in Table 1:

$\bar{y}(n-1)$	y(n)	$\bar{y}(n)$
0	0	0
1	0.55	1
2	1.1	1
3	1.65	2
-4	-2.2	-2
-3	-1.65	-2
-2	-1.1	-1
-1	-0.55	-1

Table 1: FSM for first-order 3-bit system

Note that the last four entries for $\bar{y}(n-1)$ define all possible negative values. The output error (the difference between the output y(n) of the infinite accuracy filter and $\bar{y}(n)$) is at most ±0.5 LSB. In this simple example, the outputs are no different from those that would have been obtained from the arithmetic realisation. However, in a system containing more than one multiplier, the results obtained using the above method would be closer to the output of an infinite precision filter. It can be shown that an n^{th} order IIR filter may be represented as an n-dimensional FSM [1]. Before we proceed to the reduction process and the implementation of a real world filter example, we briefly explore the representation of a second order system. Let us assume the following second-order all-pole IIR filter:

$$y(n) = 0.7y(n-1) - 0.5y(n-2) + x(n).$$

Then, for zero input, the state table for a 2-bit system would be as follows:

$\bar{y}(n-1)$	$\bar{y}(n-2)$	y(n)	$\bar{y}(n)$
0	0	0	0
1	0	0.7	1
-2	0	-1.4	-1
-1	0	-0.7	-1
0	1	-0.5	0
1	1	0.2	0
-2	1	-1.9	-2
-1	1	-1.3	-1
0	-2	1.0	1
1	-2	1.7	2
-2	-2	-0.4	0
-1	-2	0.3	0
0	-1	0.5	1
1	-1	1.2	1
-2	-1	-0.9	-1
-1	-1	-0.2	0

Table 2: FSM for second order 2-bit FSM

It is clear that any input sequence other than zero would have to be added to the output y(n), because the law of superposition applies.

IV. STATE AND COMPLEXITY REDUCTION

We consider two different steps for the reduction of the original FSM: *State Reduction* (SR) which aims to minimise the total number of states thereby changing the behaviour of the original FSM (with all states present) and *Complexity Reduction* (CR) which reduces the amount of logic needed to realize the state-reduced FSM. In the first step of SR, the original FSM is searched for erroneous states such as those caused by over or underflow. Further erroneous conditions are limit cycle oscillations although, at this early stage, they are usually difficult to find and not relevant because the hardware circuit will be based upon the machine generated by the CR step following SR. Erroneous states found during SR may be replaced with suitably chosen adjacent states or state groups. The state reduction techniques currently employed for the FSM model are the removal of all duplicate states and the replacement of all underflow/overflow states (using saturation arithmetic).

Appropriate further techniques are currently under development and are based upon statistical methods, using three separate parameters to obtain a suitable measure of the relative importance of each particular state in the FSM. Using either a first or second order Markovian process, a *probability factor* P_i, a *complexity factor* C_i and an *information factor* I_i are combined to form a vector which is used to process the replete

state machine and to produce a *relevance quantity* (RQ), which is an objective measure of the importance of a state to the function of the resultant FSM. A threshold is then set so that states exhibiting values of RQ below the prescribed value are omitted by reassigning the transitions leading to them. Issues such as the frequency of the occurence of a state, the probability of undertaking a particular state transition and the path in the state transition graph which lead to a state transition are all incorporated in the computation of the RQ factor. It is envisaged that eventually, together with the detection of limit cycle oscillation states, the above reduction techniques will be fully automated within the FSiM simulator program (see below), thus providing a fully automatic environment to the user to omit states and to synthesize the corresponding performance of the FSM model.

The second step in the reduction process is complexity reduction: The machine is decomposed into suitable submachines that may or may not be interdependent. Here, known decompostion techniques from logic synthesis and logic testing [3; 4] may be used. An example realisation for an 8-bit IIR filter may look as shown in fig. 1: The submachines M1...M255 each realize an impulse response for input values of 1...255. All outputs of the 255 machines are added to yield the output value of the filter. In principle, M1...M255 are structurally identical, i.e. are multiples of each other. This does not, however, apply when cutoff effects in the vicinity of the end of the interval (0 or 255) become critical.

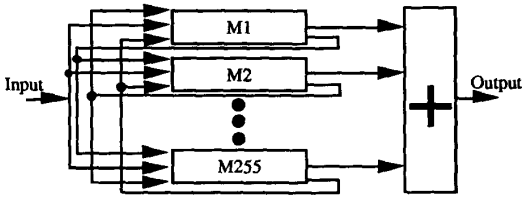

Fig. 1: A sample realisation of an LTI-IIR filter

Depending upon the actual filter coefficients, the machines handling input values near the limit of the range may be totally different from those in or around the centre. It is obvious that machines that are multiples of each other may easily be integrated into a low number of scalable machines, i.e. a large area of the dynamic range can be mapped into a few FSM's. Also, the scaling is not accomplished by a multiplier but by a number of *template machines*; which leads to a very high precision, particularly if the wordlength of the adder is sufficiently long.

V. THE SIMULATOR PACKAGE FSiM

As mentioned above, the state models of FSMs modelling non-trivial DSP systems may become very complex; hence they are rather intractable to compile and to evaluate by hand. In order to automate the design cycle (from FSM description to a hardware description language) a comprehensive software tool is being developed. Currently, a rudimentary package called FSiM has been realised, which provides key functions for FSM simulations. This package takes the mathematical specification of the FSM as the input and produces a complete description of the behaviour of the FSM. Apart from the specification of the DSP-system the only additional information that need be provided is the wordlength of the target system and its rounding mode. The simulation is normally based upon a unit impulse sequence but an arbitrary input may also form the basis for the simulation. A typical FSiM input file is shown in fig. 2, for a non-canonic second-order Butterworth low-pass filter using 16-bit input data and coefficient wordlengths together with positive and negative saturation arithmetic modes. State variables on the right hand side of an equation refer to their value during the previous cycle.

```
OUTPUT Y;
INPUT X;
STATE = (Xn_1, Xn_2, Yn_1, Yn_2);

Y = X + 2 * Xn_1 + Xn_2 +
   1.142878384 * Yn_1 - 0.4124832098 * Yn_2;
Yn_2 = Yn_1;
Yn_1 = Y;
Xn_2 = Xn_1;
Xn_1 = X;

WIDTH = 16;
ROUND = saturate;
```

Fig. 2: Example of an FSiM input file

In this input file multiple input/output systems may be defined and the order of the system is not limited. In the current version, the output of the simulator is a table of all states traversed within the FSM for a given input sequence. Moreover, all state transitions and erroneous conditions due to overflow and underflow are output (an excerpt from a complete listing is shown in the next section).

The simulator is written in C and it runs under UNIX. At present, only linear time invariant systems governed by difference equations such as the ones shown above may be simulated. However, non-linear functions such as sine and exponentials may be used to specify non-linear systems of a limited scope. In the future, the simulator will provide the ability to evaluate polynomial expressions and functions defined over a limited interval. It is also planned to include FSMs that change their structure over time by introducing conditional expressions into the FSM description. The FSM may then depend on time and, for example, upon the availability of a certain data value.

As state reduction techniques are developed they will also be included in the simulator. The simulator version currently being worked on will accept complex coefficients in the input description so as to make the realisation of FFTs via the FSM mechanism possible. It is also envisaged to incorporate facilities to graphically display the distributions of states and state transitions. This would give the user an intuitive feel for possible interaction with the state removal techniques. In the case of n-dimensional state machines, the system would provide the appropriate projections in two or three dimensions.

The detection of limit cycle oscillations essentially consists of searching cycles in a graph, a problem which is exponential in the number of states of the machine. Normally, however, self-sustained limit cycle oscillations do not exceed a certain cycle path length and the search may be stopped when a certain path length is reached. It is envisaged that provided a limit cycle oscillation path has been detected, the path will be broken and one of the states in the cycle replaced with another state, damping the oscillation to zero.

VI. FSiM AND AN ACTUAL IIR-FILTER FSM DESIGN

We now consider an IIR filter example of the Butterworth type, given by the following difference equation:

$$y_n = x_n + 2x_{n-1} + x_{n-2} + 1.142878384\, y_{n-1} - 0.4124832098\, y_{n-2}$$

(subscripts are used here to improve readability). FSiM takes this equation as a complete description of an FSM (see fig. 2) and produces the full state table, together with a listing of all state transitions. Fig. 3 is an excerpt from the whole listing and shows the states encountered after a unit-impulse is applied to the IIR filter described by the above equation. Note that the impulse response is finite due to rounding effects.

The research aim with respect to FSiM is to decrease the amount of computation resources necessary to tabulate the whole automaton. A block-diagram orientated graphical user interface within the framework of the CADiSP system [5] is also being developed. It has been designed to provide the user not only with a display of the output signal but also with a means of reducing states stepwise and interactively to simulate the corresponding performance. The user may eventually go back and forth between the different reduction levels to obtain an immediate feedback on the corresponding system performance.

n	x	y	$(y_{n-2}, y_{n-1}, x_{n-2}, x_{n-1})$
0	32767	32767	0,32767,0,32767
1	0	32767	32767,32767,32767,0
2	0	32767	32767,32767,0,0
3	0	23933	32767,23933,0,0
4	0	13837	23933,13837,0,0
5	0	5942	13837,5942,0,0
6	0	1083	5942,1083,0,0
7	0	-1213	1083,-1213,0,0
8	0	-1833	-1213,-1833,0,0
9	0	-1595	-1833,-1595,0,0
10	0	1067	-1595,-1067,0,0
11	0	-562	-1067,-562,0,0
12	0	-202	-562,-202,0,0
13	0	1	-202,1,0,0
14	0	84	1,84,0,0
15	0	96	84,96,0,0
16	0	75	96,75,0,0
17	0	46	75,46,0,0
18	0	22	46,22,0,0
19	0	6	22,6,0,0
20	0	-2	6,-2,0,0
21	0	-5	-2,-5,0,0
22	0	-5	-5,-5,0,0
23	0	-4	-5,-4,0,0
24	0	-3	-4,-3,0,0
25	0	2	-3,-2,0,0
26	0	-1	-2,-1,0,0
27	0	0	-1,0,0,0
28	0	0	0,0,0,0

Fig. 3: States of the Butterworth filter after unit-input

VII. CONCLUSIONS

Summarising the benefits of the FSM method as follows:

- Full control of the DSP–system's behaviour through extensive simulation: Every state that the system may ever encounter may be examined for desirability.
- The system's behaviour is fully predictible for any input signal.
- Elimination of undesired effects by removing states and replacing them with substitute states taken from the set of "sensible" states.
- It is well known that FSMs may be realized by combinatorial circuits of various technologies such as Read-Only-Memories, Programmable-Logic-Arrays, EPLDs or application specific circuits (ASICs). Therefore, the method provides the potential for flexible implementation and for optimizing hardware realizations for very high speed operation.
- Furthermore, the method lends itself to an implementation based on block-diagram orientated graphical user interfaces.
- Even if the system cannot be described analytically, an FSM may be constructed from the known input-output relationship (including the past history). State reduction techniques may then be applied as well and the effect of expunging certain states may be examined interactively. Depending upon the reduction level, the performance of the DSP system will be reduced but it will be realizable at lower cost. Note, however, that the user always keeps control of the properties of the system.

VIII. REFERENCES

[1] Y. Zhang and L.S. Dooley
A New Realisation for Digital Signal Processing Systems using Finite State Machines
London: McGraw-Hill, to appear early 1993.

[2] Oppenheim, A., Schafer, R.
Digital Signal Processing
Englewood Cliffs: Prentice-Hall, 1975, pp. 287...289

[3] S. Devadas and A. Newton
Decomposition and Factorization of Sequential Finite State Machines
IEEE Int. Conf. on Computer Aided Design, 1988

[4] P. Ashar, S. Devadas and A. Newton
Optimum and Heuristic Algorithms for Finite State Machine Decomposition and Partitioning
IEEE Int. Conf. on Computer Aided Design, 1989

[5] A. Knoll and R. Nieberle
CADiSP - A Graphical Compiler for the Programming of DSP in a completely symbolic way
IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Albuquerque, 1990

ACKNOWLEDEMENTS

The authors would like to acknowledge the grant support received under the ARC collaboration scheme from the British Council and the Deutsche Akademische Austauschdienst (Grant No. ARC-173)