Journal of VLSI Signal Processing 16, 41–55 (1997)
© 1997 Kluwer Academic Publishers. Manufactured in The Netherlands.

Architectural Synthesis of Digital Signal Processing Algorithms Using "IRIS"

D.W. TRAINOR, R.F. WOODS AND J.V. McCANNY
Department of Electrical & Electronic Engineering, The Queen's University of Belfast, Ashby Building, Stranmillis Road, Belfast BT9 5AH, Northern Ireland

Received March 4, 1996; Revised August 14, 1996

Abstract. In this paper, we present the IRIS architectural synthesis system for high-performance digital signal processing. This tool allows non-specialists to automatically derive VLSI circuit architectures from high-level, algorithmic representations, and provides a quick route to silicon implementation. By incorporating a novel synthesis methodology, called the Modular Design Procedure, within the IRIS system, parameterised models of complex and innovative DSP hardware can be derived and automatically assembled to create new DSP systems. The nature of this synthesis methodology is such that designers can explore a large range of architectural alternatives, whilst considering all the architectural implications of using specific hardware to realise the circuit. The applicability of IRIS is demonstrated using the design examples of a second order Infinite Impulse Response filter and a one-dimensional Discrete Cosine Transform circuit.

1. Introduction

In recent years, considerable research has been carried out into the design of novel VLSI architectures for digital signal processing (DSP) applications [1]. The highly structured nature of many DSP functions makes it possible to derive regular circuit architectures, such as systolic arrays, that are ideal for VLSI implementation. Exploitation of algorithm parallelism, and using techniques such as pipelining, can tailor architectures to particular sampling rate, area and power requirements.

In the field of electronic systems design generally, and DSP design in particular, systems are becoming increasingly complex, performance specifications are becoming more stringent, and time-to market pressures are shortening design cycles. In response, attention is now being directed at employing design automation to carry out more abstract, system-level design tasks, by using CAD tools to automatically derive circuits from algorithmic descriptions and required performance specifications. Since this process involves translating a required behaviour into an equivalent circuit structure, it is referred to as *architectural synthesis*.

The purpose of this paper is to describe the IRIS architectural synthesis tool for high performance DSP. Unlike other approaches [1–7], the emphasis in IRIS is to allow architectural exploration of algorithms. The process involves using processing unit models to generate optimal architectures based on those units. The main advantage of IRIS is that it offers the designer full freedom to investigate a wide range of architectures using user-preferred blocks or novel processing techniques. It is also tightly coupled to conventional VHDL synthesis tools, and has been used to produce practical and realistic designs.

The remaining text of this paper describes the synthesis methodology and the operation of the IRIS architectural synthesis tool. In Section 2, an overview of commonly applied architectural synthesis techniques is given, together with some important limitations exhibited by these techniques when applied to the synthesis of novel, high performance DSP circuits. This is illustrated using an IIR filter example in Section 3. A solution to these difficulties is also presented in Section 3, which introduces a radical new synthesis methodology, the Modular Design Procedure, and discusses its automation within IRIS. The capabilities of the IRIS system are discussed during Section 4, and demonstrated throughout Sections 3 and 5, by carrying out novel implementations of second order recursive filtering and Discrete Cosine Transform algorithms. Finally, Section 6 offers some conclusions that can be drawn from the results of this research.

2. Architectural Synthesis Methodologies

Considerable effort has been focused on deriving methods for mapping DSP algorithms onto VLSI architectures, including techniques based on sets of Recurrence Equations [2], dependence graphs [1] and, algebraic methods [3]. Whilst many of these specialist tools produce architectural representations from high levels of abstraction, they are unable to produce functionally-correct, implementable circuits, as issues such as internal word growth, truncation and data organisation, have not been considered. These issues affect the latency and timing of data and thereby invalidate any derived architecture. It is left to the IC designer to modify and refine the initial architectures taking into considerations these design details. The second order Infinite Impulse Response filter example in Section 3 demonstrates the radical differences that implementation-level performance criteria can create between abstract algorithmic representations and a corresponding physical circuit.

Synthesis systems, such as CATHEDRAL, PHIDEO, HYPER and MARS [4–7] can take algorithmic descriptions and apply scheduling, assignment and hardware mapping techniques to synthesize an architecture. With these systems, hardware mapping, the process that maps a flow graph onto the available hardware blocks, is carried out after the various scheduling and assignment procedures. The assumption is therefore made that hardware mapping does not alter the structure or functionality of the design. For the hardware units generally used in reported design examples synthesized by these tools, this may be a valid assumption. However, it has been demonstrated [8] that if the hardware units are complex pipelined processors, hardware mapping can invalidate the architecture. This needs to be resolved if the circuit is to operate correctly once implemented using the chosen hardware, and has been shown to be a complex task [8].

A vital issue relating to the various synthesis methodologies is the manner in which the design space may be explored. Most of the current architectural synthesis tools apply a fixed set of synthesis procedures, regardless of the different properties of the circuit in question. Hence, with these tools, a number of algorithmic transformation methods are derived and applied uniformly to all problems. This approach may not be appropriate for many designers, who explore design trade-offs in different ways, depending on the specifications and design properties for the particular problem currently under consideration. For these designers, a tool based on fixed synthesis principles is undesirable, since what is required is a system that allows the designer to explore and trade-off different design issues in a completely open manner.

We believe that there is considerable scope for a design methodology that allows designers to easily implement and optimise architectures based on processing elements, and hence specific hardware, of their own choosing. Designers could use favourite processing units (which allows the re-use of previously optimised circuit layouts) or clever novel technologies, such as redundant arithmetic processors [9], which can offer clear advantages for some applications. Therefore, the IRIS synthesis system has been developed, which enables the extraction of parameterised expressions from complex VLSI processing elements, and uses these expressions to achieve functionally-correct solutions for circuits built from these processors. Designers can quickly create and evaluate architectures that utilise existing hardware blocks and can be easily realised using commercial silicon design tools. Also, designers can take advantage of novel circuit designs, gaining architectural knowledge as well as synthesis capability.

3. Architectural Synthesis Using the IRIS System

The design input of the IRIS system is a Signal Flow Graph (SFG) representation of the algorithm, consisting of zero-delay processing nodes connected by edges which may be weighted with an appropriate number of delays. For example, consider the algorithm carried out by a second order Infinite Impulse Response filter, which is defined by Eq. (1).

$$y_n = a_0x_n + a_1x_{n-1} + a_2x_{n-2} + b_1y_{n-1} + b_2y_{n-2} \quad (1)$$

Figure 1 shows a SFG that is functionally equivalent to the algorithm of Eq. (1). Each processing node represents a Multiply/Accumulate (MAC) operation, and the black circles on particular graph edges represents a single delay, which is necessary to compute the algorithm.

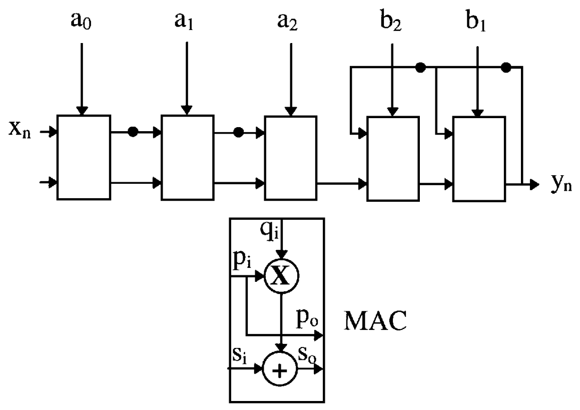

Figure 1. SFG for second order IIR filter.

The synthesis technique involves replacing the generic processing nodes of the SFG with models of heavily-parameterised pipelined processing units from a library and applying a methodology called the Modular Design Procedure (MDP) [8]. Unlike other systems [4–7] within IRIS the processor parameters exactly model the performance of the individual processor.

The process of synthesis thus involves replacing the zero-delay nodes in the original signal flow graph by the model of the practical processor. The original Signal Flow Graph with zero-delay operators defines the algorithmic functionality but this is changed when practical processors are inserted that have different latencies. If the timing effects of using practical processors is not addressed then the resulting architecture will implement a different algorithm.

IRIS addresses this problem by retiming the circuit to preserve the original algorithm but in such a way to minimise the number of delays that are added. This is achieved by calculating the maximum number of delays within all the loops in the Signal Flow Graph and then using retiming [10] to ensure the global timing in the synthesized architecture is equivalent to that in the original SFG. The circuit will then have a maximum possible sampling rate. If this maximum possible sampling rate meets the specified sampling rate, then the design has been successful and the user has achieved an optimised solution for the hardware used. If the maximum possible sampling rate is in excess of the sampling rate required, then the user can investigate alternative designs using different, namely slower and more hardware efficient, blocks or using hardware sharing, to achieve a more efficient solution. Subsections 3.1 and 3.2 discuss these issues, using the second order IIR filter as a design example.

3.1. *Derivation of IRIS Processor Models*

In order to utilise complex processors as blocks that can be used to synthesize DSP systems in IRIS, models of the various processors need to be derived and placed in a library within the tool. It is these models that replace the mathematical operations of the SFG, and it is the information contained within the models that is used by IRIS to determine the data timing changes caused by replacing the zero-delay SFG operation with complex, pipelined hardware.

The processor models abstract much of the structural detail of the processor architecture, but retain enough performance-related information so that the effects of placing the models into the SFG can be determined. When a processor model is derived, the MDP demands that two processor performance measurements must be associated with the model. The first of these is the data format, or “time shape” of data entering or leaving each input or output of the processor. The time shape may be defined as the position in time of the bits or digits of the data value relative to each other [8]. Figure 2 shows some examples of typical data time shapes.

The detailed structure of the particular processor, and particularly the placement of internal pipelining latches, will determine the data time shape at each processor input and output. It is necessary to maintain information on the processor time shapes to determine what extra circuitry, if any, needs to be placed between connected processors to convert between the format of the data at the output of the first processor, and the format expected at the input of the second.

The second processor performance issue that must be addressed by the equivalent IRIS processor model is the latency through each datapath in the processor. These latency values must be incorporated in the processor model since the number of clock cycles required for a particular processor to produce its results has profound effects on the timing of data throughout the entire circuit.

An important design feature of the IRIS system is that the information on data time shapes and datapath latencies within the processor model can be parameterised in terms of other important processor design criteria, such as the data wordlengths used and the number of internal pipelining stages. Therefore, design changes, such as changes in data wordlengths,

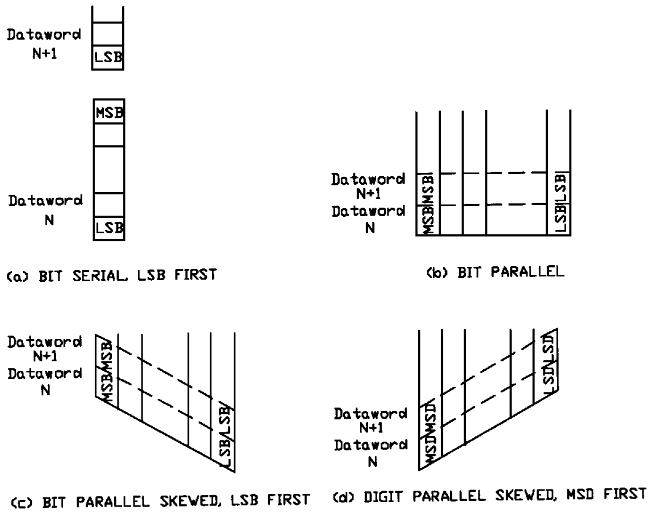
Dataword N+1
LSB
MSB
Dataword N
LSB
(a) BIT SERIAL, LSB FIRST
Dataword N+1
Dataword N
MSB
LSB
(b) BIT PARALLEL
(c) BIT PARALLEL SKEWED, LSB FIRST
MSD
LSD
(d) DIGIT PARALLEL SKEWED, MSD FIRST

Figure 2. Typical data time shapes.

can be easily taken into account. This parameterisation is also complementary to the recent trend of the production, and purchase, of libraries of parameterised "mega-functions" [11], which are usually written in a Hardware Description Language. These libraries can increase design re-use and reduce time to market, by allowing systems to be constructed by connected complex re-usable blocks of hardware.

Whilst a library of processor elements is available to the user, a processor interface capability is available to allow the incorporation of new processing blocks within IRIS. These blocks could be application-specific components from commercial vendors, allowing a tight coupling between IRIS and conventional compilers. Alternatively, these processors could use novel processing techniques, such as redundant arithmetic [9], which would be difficult to capture using conventional synthesis tools.

3.1.1. *Derivation of IRIS Processor Models for the Second Order IIR Filter.*

In order to demonstrate the construction of IRIS processor models, consider the implementation of the second order IIR filter SFG shown in Fig. 1. This design requires the insertion of blocks of hardware, capable of the MAC operation, into the processing nodes of the SFG. Examples of high-performance modules, structurally defined at the bit level, that could be used to realise the filter circuit are the Carry-Save and Signed Binary Number Representation (SBNR) MAC processors [9] shown in Figs. 3 and 4.

In Fig. 3, the two signal p and q are multiplied, and the result added to the signal s. The labels p_i, q_i and s_i, refer to data entering the processor, whilst p_o, q_o and s_o refer to output data. The superscript of each label refers to the significance of the data bit to which that

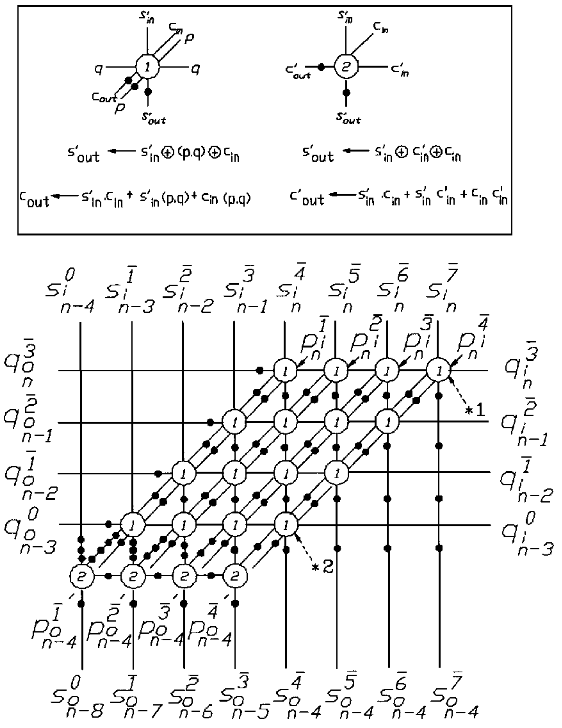
$s'_{out} \leftarrow s'_{in} \oplus (p.q) \oplus c_{in}$
$c_{out} \leftarrow s'_{in}.c_{in} + s'_{in}(p.q) + c_{in}(p.q)$
$s'_{out} \leftarrow s'_{in} \oplus c'_{in} \oplus c_{in}$
$c'_{out} \leftarrow s'_{in}.c_{in} + s'_{in}c'_{in} + c_{in}c'_{in}$
*1
*2

Figure 3. Structure of carry-save MAC processor.

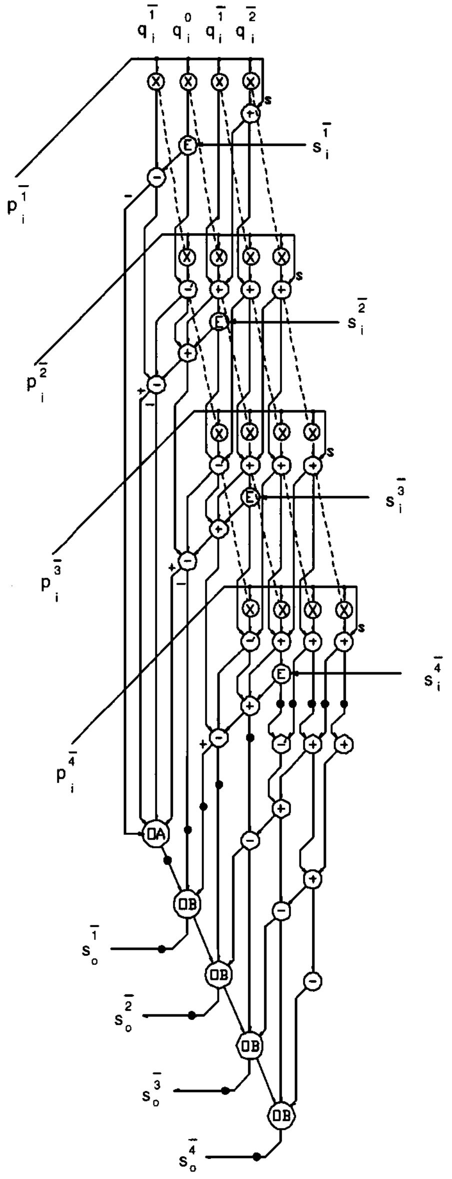

Figure 4. Structure of SBNR MAC processor.

superscript is attached e.g., if a bit is identified with a superscript, "1", that bit is of significance 2^{-1}, i.e., 0.5. The subscript of each label indicates the relative clock cycle at which the bit signal enters or leaves the processor e.g., a bit labelled "$n-1$" enters or leaves the processor one clock cycle after a bit labelled "n". This notation defines the timing of the various bits of data as they pass through the processor.

The scheduling of the various operations in the Carry-Save MAC processor results in a number of pipeline stages, implemented by the addition of latches, depicted by black dots in Fig. 3. These latches define the timing of the various operations within the MAC processor. An important point to notice is the extra latches that are added to the datapath of the *s* signal at the right hand side of the structure. These latches ensure that each bit of the input s_i travels through the same number of pipeline stages before emerging at s_o, therefore the relative timing of the various bits of data entering s_i is maintained at s_o. This allows several MAC processors to be cascaded without having to convert the data timing from the output of one processor to the input of the next. This arrangement is useful for several important DSP circuits, particularly digital filters, which can be easily implemented using cascaded MAC processors.

Figure 1 shows that the IIR filter SFG exhibits recursion, where previously generated results form the inputs for future iterations of the algorithm. Previous research has shown that for such structures, processors that exhibit low, wordlength-independent latency and use redundant number systems to produce results most significant digit first can produce more efficient implementations of the feedback paths [9]. The SBNR MAC processor shown in Fig. 4 represents such a structure, multiplying the p_i signal and the q_i signal (which is represented by the dotted lines in the figure) and adding the s_i signal. The use of SBNR allows the removal of carry propagation chains within the processor and hence the production of results in a most significant digit first format.

Examples of IRIS processor models are shown in Figs. 5 and 6. These models have been derived from the MAC structures of Figs. 3 and 4, and will be used

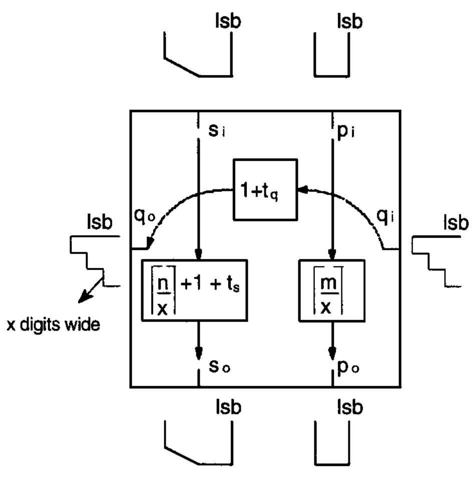

Figure 5. Carry-save MAC model.

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase® Smarter legal research.