RC23027 (W0312-044) December 10, 2003
Electrical Engineering

IBM Research Report

Design Methodology for Low Power High Performance Semi Custom Processor Cores

V. Zyuban, S. Asaad, T. Fox, A. Haen, D. Littrell, J. Moreno
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Design Methodology for Low Power High Performance Semi Custom Processor Cores

V. Zyuban, S. Asaad, T. Fox, A. Haen, D. Littrell, J. Moreno

IBM T.J. Watson Research Center

Abstract

This paper describes a semi-custom design methodology that was prototyped/demonstrated through the development of low-power high-performance DSP core. The developed methodology achieves significant speed improvement and reduction in power and area, compared with standard ASIC flow, without compromising its generality and high productivity. Because of the fast turn-around time from RTL description to post PD timing results, and stable convergence on timing the developed flow enables optimizations spanning multiple levels of the design hierarchy. Such optimizations proved much more effective than those that focus on any single phase of the design, which makes the described flow a compelling choice for the development of embedded processor cores.

Keywords

Design, Methodology, ASIC, Synthesis, Microprocessor, Core

I. Introduction

The design and implementation of high-performance embedded processor cores is characterized by conflicting requirements. These include (1) the functionality to be provided by the instruction set architecture (ISA), which depends on the target domain of the embedded processor; (2) the ever increasing demand for higher performance; (3) usually, stringent power consumption and area limitations; and (4) short development schedule driven by time-to-market considerations. An attempt to fulfill all these requirements at once throughout the design process is -in general- not feasible, thus some compromises need to be made early in the design cycle. One of the common choices that characterizes many current embedded systems is the use of "soft-cores," with the associated reliance on tools to achieve short

turn-around time and the be able to handle changes late in the design cycle. Moreover, the use of soft-cores simplifies porting designs across multiple foundries, a desirable feature nowadays. In fact, recent years have seen a dramatic growth in the development of "soft intellectual property" (soft-IP), representing a major trend in the embedded systems industry. Only companies whose products are manufactured in very large volume can tolerate the cost associated with the development of "hard-cores," that is, cores that have been improved through the use of custom circuits and low-level (eg., VLSI layout) tools and techniques [1].

Although the use of soft-cores offers significant advantages, the resulting systems tend to exhibit lower performance, larger area, and higher power consumption than what can be achieved in a customized implementation. Such differences have led to the search for "semi-custom" design flows that could achieve many of the benefits of full customized implementations, yet offer the flexibility and turn-around time that is typical of the soft-core counterparts.

The aspects mentioned above are even more relevant when designing and implementing a new instruction set architecture, because in such cases it is possible to perform tradeoffs encompassing multiple design levels, from ISA through micro-architecture, logic and circuit design. Exploiting such opportunities requires the ability to quickly iterate throughout the design space, yet obtain appropriate results in terms of area, frequency and power consumption.

In this paper, we describe the methodology used in the implementation of a digital signal processor (DSP) architecture, whose performance and power targets were such that a conventional soft-core approach would not be adequate, yet the design constraints (resources available and time to completion) did not allow for a custom design approach. The implementation was carried out in conjunction with the development of the instruction set architecture, enabling the evaluation of tradeoffs at all levels in the design. The salient features of the DSP architecture have been described in [11]; details on the methodology and metric developed to perform power/performance tradeoffs among features of the architecture have also been provided there. The implementation in a 0.13um foundry technology

was targeted to reach 500 MHz under worst-case conditions, and limited to consume 350 mW when operating at 1.5V, 105 degrees Celsius.

Significant progress has been reported recently in physical synthesis tools [2], [17], [14], as well as placement and routing tools for datapaths [8], [6], [16], [5]. However, such contributions focus mostly on a single phase or design tool. In contrast, we focus on a methodology which spans multiple layers of the design hierarchy, from microarchitectural definition down to layout, and which enables optimizations across different design phases. Consequently, the contributions of this paper are the detailed description of the techniques and tools deployed for a semi-custom design flow, which improve in design productivity over custom design flows, such as those used in the design of high-end processors [3], [15], [7], [13], and make it applicable to the design of embedded systems. Although we have used a DSP architecture as a vehicle for developing this methodology, the resulting design flow is applicable to embedded processor cores in general.

We first provide a summary of the main limitations of the typical soft-core ASIC design flow, and then describe the various steps in the methodology deployed. In Section III, we discuss techniques used for assembling and timing the core, and the steps used to enable hierarchical synthesis and pre-placement of components. In Sections IV, we illustrate these aspects further by describing the procedure for balancing synthesis assertions, grouping latches for clock splitters, the pre-placement of latches and splitters, as well as the instantiation and pre-placement of decoupling buffers. In Section V we describe the clock gating methodology and techniques used to increase the portion of clock-gated latches in a processor core. In Section VI we demonstrate the effectiveness of the developed methodology through the design of a large multi-ported register file, which had proven not feasible through conventional synthesis procedures. In Section VII, we complete the description of the methodology by illustrating the hierarchical design of a DSP core.

II. Limitations of the standard ASIC flow

The ASIC design flow is typically chosen whenever there are limited resources which do not allow supporting custom design, in-house tools and libraries, or whenever the schedule is very tight. However, the standard ASIC flow and the associated tools have a number of known limitations [1], [8], [6], [16], [5]. First, the power level of gates is chosen based on fan-out and total size of the design in flat synthesis. As a result, gates driving short wires tend to be oversized, whereas gates driving long wires tend to be underpowered, resulting in power overhead and loss of performance. Second, the switching activity of wires is ignored during placement and routing, resulting in power waste. Also, the criticality of wires for timing is ignored during placement and routing, resulting in non-optimal or "scenic" routes on critical paths, and further loss of speed. Clock trees generated by ASIC tools are typically not well balanced, resulting in large clock skews. Finally, the integration of custom-designed components is not supported, and the application of advanced power reduction techniques used in custom designs, such as back bias control, power gating, or data retention [10], is not allowed in the current generation of ASIC libraries.

III. Overview of the Design Methodology

The main objective of our methodology was to exceed the performance and power characteristics of designs built using standard ASIC flow, without compromising its productivity and generality. The second objective was to enable integration with custom components, and enable the application of power reduction techniques not provided by ASIC flow, such as power gating, reverse bias, and data retention [18], [10]. Finally, in order to remove some of the critical paths or power hot spots in the design of a newly developed architecture, it may be necessary to use optimizations spanning multiple levels of the design hierarchy. In order to enable such optimizations, the methodology must allow a fast turn-around time from RTL level description to post PD timing results.

Figure 1 depicts our design flow. Although not a required step, the architectural and microarchi-

tectural definition of the microprocessor core is entered to an architectural database [11]; this is the "golden" definition of the design. Architectural features such as pipeline latencies, functional units, ports to the register files, bypasses, etc. are stored in the architectural database. An architectural simulator is automatically updated from the design database and used for verification of the architecture and VHDL description.

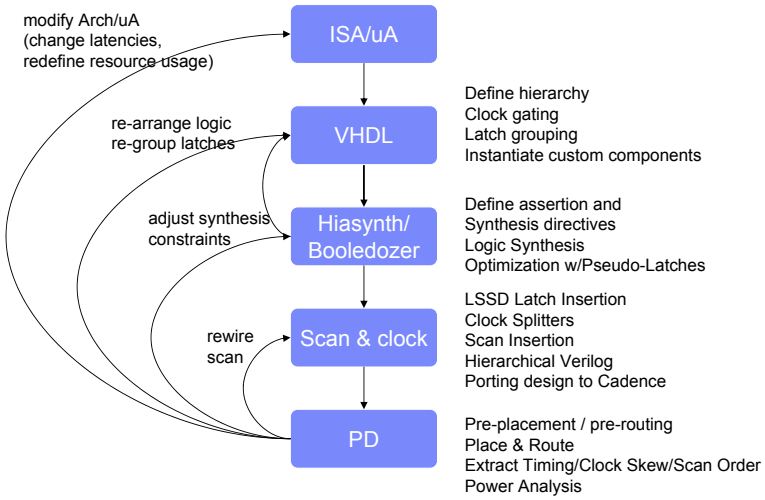

Fig. 1. Overview of the methodology

In our methodology, like others [3], [15], the design of the processor core is organized into units. The number of units is chosen such that the size of each individual unit can be handled by synthesis and physical design tools. The partitioning between units is chosen such that the interfaces between the units are intuitive, and different units can be designed by different designers in parallel.

The physical design (PD) environment supports many IBM internal design and analysis tools written for server design groups [13]. It also allows integration of custom components in the design. Custom components can be used at several levels of the design hierarchy. At the lowest level, individual custom cells can be added to the design library, such as latch or XOR gate. Then, custom designed components can be used, such as custom functional units (adder, shifter, multiplier), or multi-bit latches and registers. Finally, the custom designs can be used for the whole units, such as register files, memories, vector pipeline. This environment also allows application of custom design techniques for reducing leakage power, mentioned earlier.

The overall design flow consists of two steps: individual units are built first, then the core is assembled and global timing is performed. We now describe these two steps in detail.

A. Synthesis and Physical Design of Units

The VHDL view contains the hierarchical RTL description of the design. All latches are explicitly instantiated in VHDL as behavioral flip-flops, connected to an ideal clock; that is, the VHDL description does not have clock trees, clock splitters or support for scan. Clock gating is explicitly defined in VHDL code, and so is the grouping of latches for clock splitters.

The VHDL view is passed through the hierarchical synthesis step. Synthesis directives, timing and capacitive assertions are created for every level of the design hierarchy. All optimizations during the synthesis step are performed with behavioral flip-flops. Static timing analysis is performed on the library-mapped netlist. Wire capacitances are estimated by timing tools, based on the size of the design and fan-out of each wire. Depending on the results, synthesis assertions and directives are adjusted, and the synthesis is repeated until satisfactory timing results are achieved.

In the post-synthesis step, the behavioral flip-flops are replaced with LSSD latches [12], clock splitters are instantiated, and latches are connected into scan chains. Then, the design is saved as a hierarchical mapped netlist and ported into the physical design environment [13].

At the physical design phase (PD), a physical image is created for every unit, library cells are pre-placed, and clock and power/ground grids are pre-routed. Placement and routing are done using Cadence design tools. A post-PD netlist is extracted, and timing and power analysis is performed with IBM internal tools normally used in the design of high-end microprocessors [3], [15], [7].

Depending on the results, one or more of the steps may need to be repeated. Changes may involve re-routing the scan chains, adjusting the synthesis capacitive and timing assertions, re-arranging logic at the VHDL level, or re-grouping latches for clock splitters. The more severe the timing or power problem, the more steps are repeated in the iteration, as shown in Figure 1.

Because of the techniques introduced, as described below, a consistent convergence on timing results is achieved. According to our experience, the typical number of design iterations ranges from 3 to 5; since all steps in the design process are fully automated, the typical time of going through one design iteration ranges from 2 to 5 hours, depending on the number of steps involved. If a particular problem can not be fixed through the steps described above, this indicates an inherent problem in the microarchitectural definition of the processor. In such cases the micro-architectural definition need to be appropriately adjusted, and all phases of the design need to be repeated. One example of microarchitectural modifications resulting from this process that we experienced was an increase in the latencies of an inter-unit data transfer bus.

B. Core assembly and timing methodology

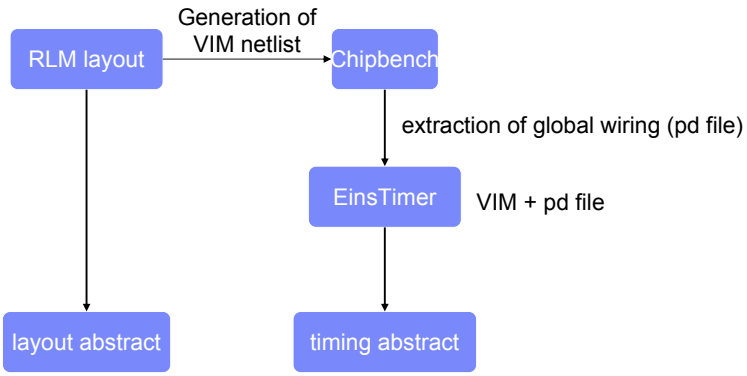

Fig. 2. Generation of timing and layout abstracts

Core assembly and global timing is done in parallel with the design of the units. For core assembly and global timing each unit is represented by the corresponding layout and timing abstracts.

For each unit, a layout abstract and a timing abstract are created, as shown in Figure 2. For the timing abstract, the global wiring between cells is extracted using IBM's Chipbench [13]; this information, along with the unit netlist and the standard cell library timing rules, is passed on to EinsTimer [9], IBM's sign-off timer.

Core assembly and timing are done hierarchically using abstracts representations of the blocks one level below, as shown in Figure 3. The top floorplan is created by instantiating those layout abstracts,

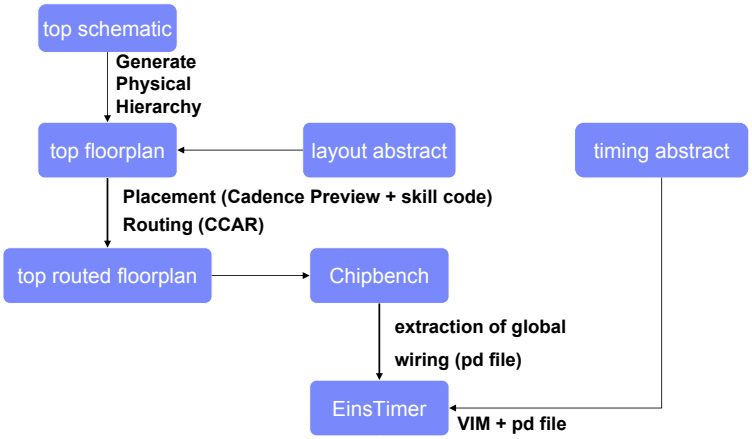

Fig. 3. Core assembly and global timing

placement is done in Cadence Preview using Skill code, and routing is performed with Cadence's Chip Assembly Router (CCAR). The global wires of the top routed floorplan are extracted in the same manner as for the units, and the result is fed back to EinsTimer along with the top floorplan netlist and the timing abstract of each unit. Depending on the results, some modifications to achieve timing closure might be needed anywhere from microarchitecture down to PD, and this process is re-iterated until timing closure is achieved.

IV. Design Techniques used in the Flow

We now give a detailed description of the techniques used in the design flow.

A. Hierarchical Synthesis of Components

The VHDL description of every unit is broken up into components, a few thousand gates each, as shown in Figure 4. Timing assertions, capacitive assertions and synthesis directives are generated for every component, and components are synthesized independently of each other. Then, the synthesized modules of all components of a unit are loaded and the top view of the unit is synthesized, with only limited optimizations across the hierarchy boundaries [4].

Some portion of the logic in each unit, referred to as "dust" in Figure 4, does not get assigned to any of the components. This logic typically consists of some control logic and datapath multiplexors. The logic is synthesized during the synthesis of the top view of the unit. Since the wire model at this

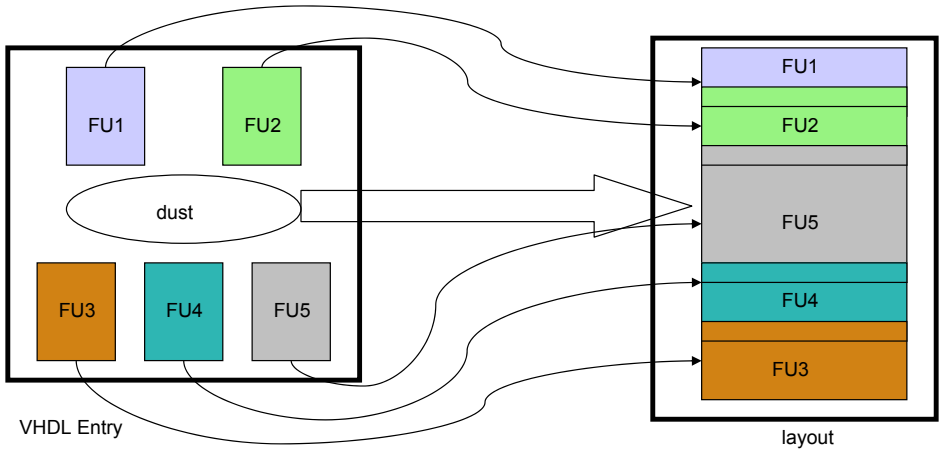

Fig. 4. Hierarchical Synthesis and Pre-placement of Components

point is based on the size of the whole unit, gates comprising the dust use library cells with power levels much higher than those comprising the components. Designers were challenged to minimize the size of the logic that was part of the dust, typically achieving less 10% to 20% of the size of the unit.

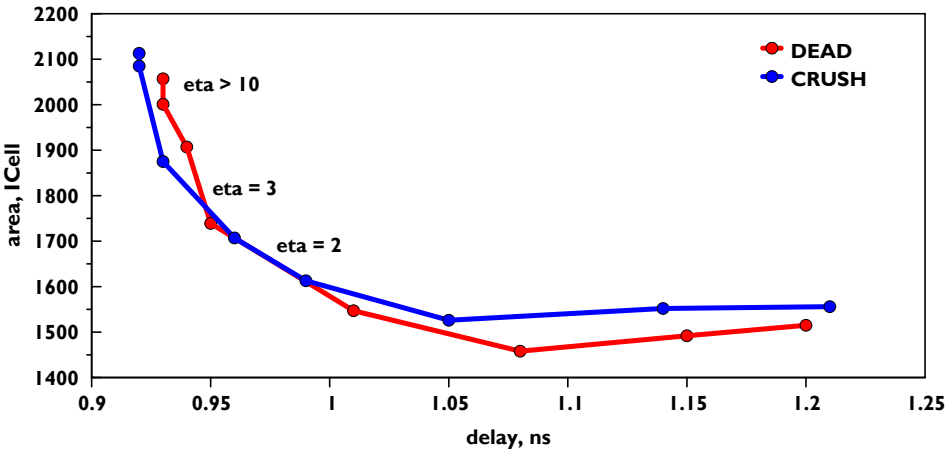

Fig. 5. Area-delay tradeoffs for an adder

To get energy-efficient designs out of the synthesis of individual components, it is essential to optimally set the synthesis timing constraints. Synthesis design constraints determine how the synthesis algorithms transform and size the logic, thus they directly impact the tradeoff between delay, area and power. Typically, the individually synthesized components do not have latched boundaries at each, or even some, of its input and output ports, because the inputs may be passing through some re-buffering or logic before getting to the input ports of the components; similarly, the outputs from the component may have to pass through a multiplexing network and/or bypasses before they are

latched. Therefore, there exists some flexibility in budgeting the cycle time between the components and the rest of the logic which may be part of the "dust". The apportionment of delay across several hierarchical boundaries has a significant effect on the area and power of the unit.

A methodology for balancing hardware intensity introduced in [20], [19] is used for generating the synthesis timing assertions. Components are synthesized with a range of constraints, so that the hardware intensity can be quickly estimated for different assertions as $\eta \approx \frac{\%\text{AREA}}{\%\text{RAT}}$, where %AREA is a relative increment in area and %RAT is a relative decrement in the Required Arrival Time (or a relative increment in the arrival time). For higher accuracy energy, rather than area, should be used to measure the hardware intensity, obtained by simulating the component. The *aggregate* hardware intensity is an energy-weighted average of hardware intensities in all components. The optimal value of the *aggregate* hardware intensity for the chosen technology (0.13um), design library (IBM CU11) and power supply voltage (1.5V) was determined to be $\eta = 2$ [20]. Then, the key rule is to balance timing assertions in such a way that most of the components sit on the part of the curve, where every percent reduction in delay costs approximately two percent in energy (or area), Figure 5. It is especially important to avoid designs that sit on the steep part of the curve for those components that consume a lot of energy and/or used (that is un-gated) in most of the cycles.

Figure 5 shows a graph of Area vs. Delay for an adder. Two logic restructuring algorithms are used against several targeted timing assertions, yielding a range of data points from which the design with a desired value of hardware intensity can be selected [20].

B. Pre-placement of components

A physical image is created for every unit, then a set of overlapping regions is created within the physical image for the pre-placement of the components, as shown in Figure 4. The sizes of the regions and the overlaps between them are chosen to provide sufficient flexibility for the automated placement and routing. Gates comprising the components are assigned to the corresponding regions for pre-

placement. Gates comprising the dust are not pre-placed, letting the automatic placement tools find the best locations for those gates. As a result, gates comprising the dust end up driving significantly larger wire capacitance than those inside components; however, these gates use higher power ASIC library cells, designed to drive long wires.

One benefit of synthesizing components independently is that different components get best power, performance and area characteristics when synthesized with different directives. Second, during the synthesis of components, the wire capacitance model used by the tool is based on the size of the component, rather than that of the whole unit. The wiring model inside components is adequate because gates inside components are assigned to the corresponding placement regions, and the wires that they drive are limited to the same regions (with the exception of component output wires which are driven by higher-power gates). As a result, gates inside components are implemented with low power library cells. Since the fraction of logic that is part of the dust is usually limited to below 20%, the majority of the gates in the design are implemented with low power library cells, thus reducing both area and power dissipation.

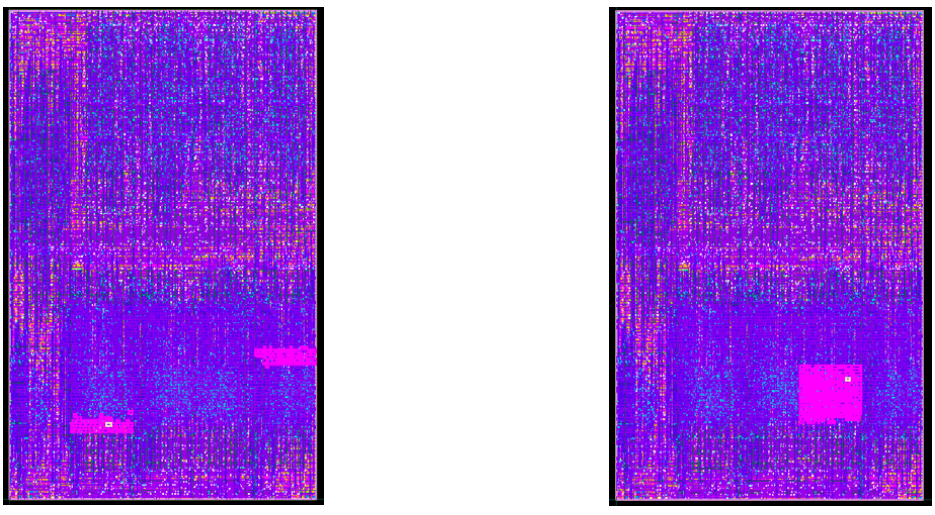

Fig. 6. Hierarchical Synthesis and Pre-placement of Components, Vector Pointer Unite layout; a: bit reversal component and pointer update component; b: pointer rotate component

As an example, Figure 6 shows the layout of the Vector Pointer Unit in the eLite DSP [11] imple-

mented in a 0.13 um bulk technology. The size of this unit is approximately 0.6mm by 0.8mm. The layout is organized into five columns. The first column corresponds to control logic and vector pointer update control logic. Columns 1 to 5 correspond to four slices of the unit, each one responsible for generating vector register file indices for the corresponding slice of an associated SIMD vector engine. Highlighted in Figure 6(a) are gates inside the bit reversal unit in slice 0, pointer update component (essentially an adder) in slice 3, and the pointer rotate component in slice 2 (Figure 6(b)). Note that cells comprising components are placed inside very limited regions.

C. Grouping of latches for clock splitters

To allow the opportunity of licensing RTL as a soft-core, and to simplify code maintenance, edge-triggered behavioral flip flops are used in the VHDL. Thus, the RTL description of the processor has no notion of LSSD [12], scan chains, clock splitters or clock distribution.

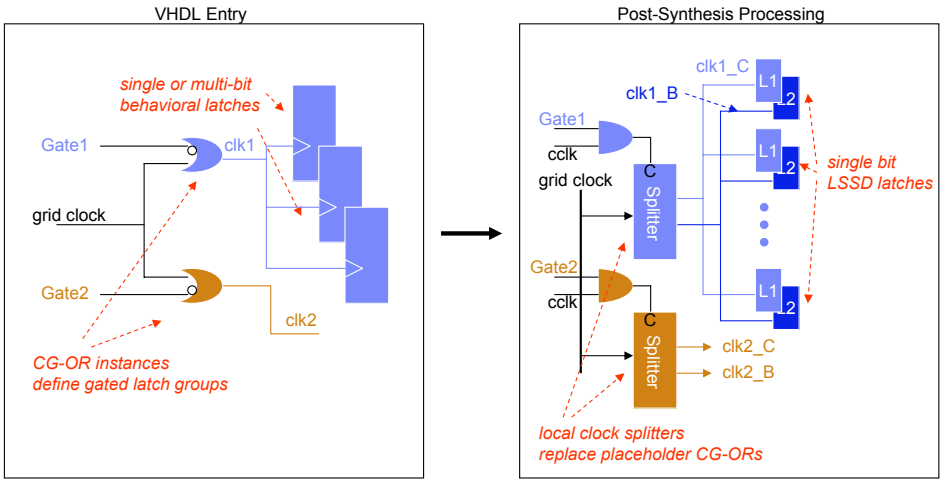

Fig. 7. Grouping of latches for clock splitters: latches eligible for clock gating

Multi-bit latches eligible for clock gating (the eligibility criterion is discussed is section V) are grouped in the RTL description, as shown in Figure 7. The clock inputs of latches in the same group are connected to the output of an OR gate, instantiated in the RTL for every group of latches. One input of this OR gate is connected to the grid (ideal) clock, and the other input is connected to the clock gating signal. The clock gating OR gate has special timing rules that check the setup (guard)

time for the arrival of the gate signal with respect to the grid clock. Based on simulations of extracted netlists, we set the guard time to 0.3ns. This timing rule is used for the synthesis and timing of the logic generating the clock gating signal.

At the post-synthesis step, the edge-triggered flip flops are replaced with master-slave LSSD latches, scan chains are connected, and the clock gating OR is replaced with a clock splitter and an AND gate, according to the "early" clock gating methodology defined in IBM CU11 ASIC library. One input of the AND gate is connected to the clock gate signal, and the other input is connected to the test clock CCLK. The output of the AND gate is connected to the input of the clock splitter that stops the master and slave clocks to the latches at the low and high levels, respectively. This "early" clock gating methodology is more efficient than a "late" clock gating methodology because it stops any switching in both the master and slave latches, and does not require level sensitive re-timing latch on the path of the clock gate signal.

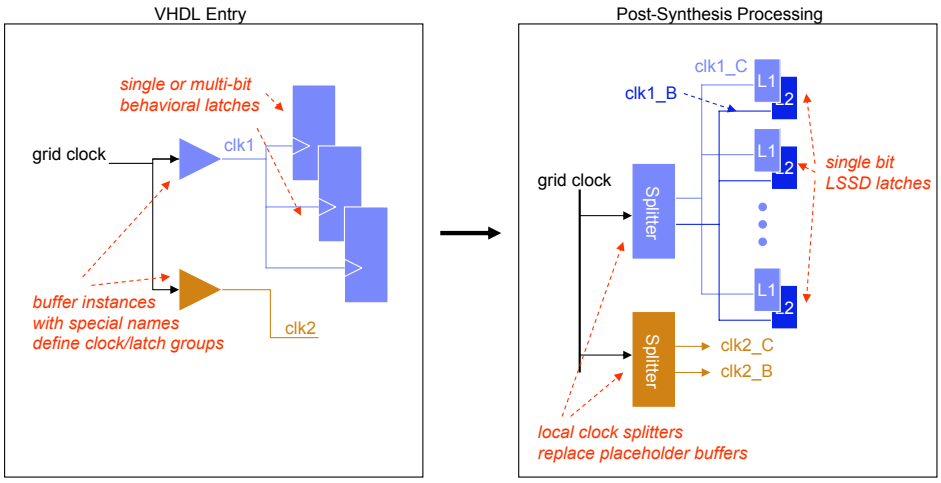

Fig. 8. Grouping of latches for clock splitters: latches not eligible for clock gating

Latches that are not eligible for clock gating are grouped in the RTL code according to Figure 8. A clock buffer is instantiated for every group of latches. The input of the clock buffer is connected to the grid (ideal) clock, and the output is connected to the clock inputs of all behavioral flip-flops in the group. At the post-synthesis step, the edge-triggered flip flops are replaced with master-slave LSSD latches, scan chains are connected and the clock buffer is replaced with a clock splitter. The grouping

of latches defined in the RTL is preserved.

D. Pre-placement of latches and clock splitters

At the physical design stage, a set of overlapping regions is created in the unit physical image for pre-placing clock splitters, clock gating cells and latches. Multi-bit dataflow latches are pre-placed with bit precision, seeding the bit ordering in the datapath. The size of pre-placement regions for latches and the overlap between them is determined experimentally to allow sufficient flexibility for the placement and routing tools, but still providing adequate control over the placement of latches and clock wires. The typical size of every latch pre-placement region is 48 wiring tracks (3 placement rows) in the vertical direction by 48 to 60 wiring tracks in the horizontal direction; the typical overlap between the regions is 36 wiring tracks in the horizontal direction and 36 wiring tracks in the horizontal (placement row) direction.

The clock grid is pre-routed within each unit as M4 clock spines with the x-coordinates specified by the contract rules established at the global floorplan. For timing analysis, the clock signal at the clock grid is assumed to be ideal (zero skew); the clock gating cells, clock splitters, and local clock distribution are part of the timing path.

Our technique for grouping and pre-placing latches provides the following advantages. First, the local clock wires, running from clock splitters to individual latches, are short resulting in significant reduction in power dissipation. Moreover, the length of the local clock wires is under control, with a controllable variation between different groups of latches, and it does not change between iterations through the design. This results in small clock skew, higher frequency and improved convergence on timing.

The bit-precise placement of dataflow latches provides seeding for bit ordering in the datapath, resulting in improved routability and savings in power and area. Since the coordinates of latches are allowed to change only within the limits of the corresponding pre-placement regions, the ordering of the

scan chain produces consistent and predictable results, further improving speed and reducing power.

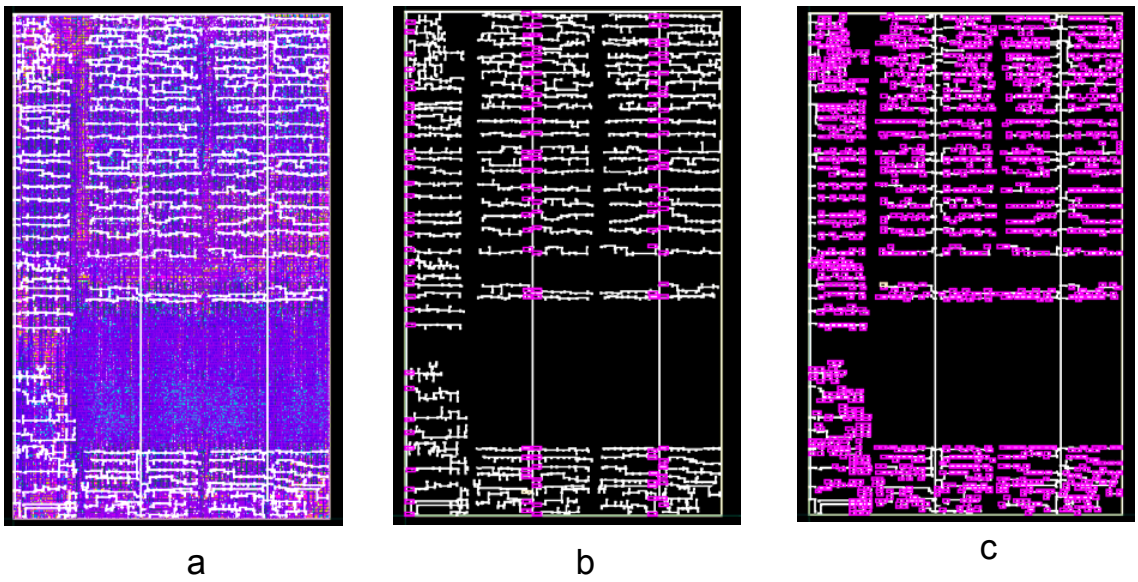

Fig. 9. Grouping of latches for clock splitters (layout examples); a: local clock wires; b: clock splitters; c: individual latches

Figure 9 illustrates the pre-placement of latches and clock splitters. There are approximately 2000 latches, including register files, and approximately 150 clock splitters. Figure 9(a) shows the layout view with highlighted clock spines and local clock wires. Figure 9(b) highlights the clock wires and clock splitters, illustrating that clock splitters are always placed next to the closest clock spines for better balancing of the clock grid and lower power dissipation (the clock grid switches every cycle, even when all latches in the unit are clock gated). Figure 9(c) shows the individual latches. Notice that the latches are not strictly aligned, which indicates that the automatic placement tool had some flexibility in placing the latches. This made routing easier, and allowed us to change the location of N-well and substrate contact cells without adjusting the placement of individual latches.

As an illustration, Figure 10 shows the layout of the same unit built without pre-placing latches and clock splitters. The grouping of latches for clock splitters in VHDL was preserved, however. When the grouping of latches is not done in VHDL, the local clock wiring gets even more convoluted. Notice that the area of the design built without pre-placing latches is about 20% larger than the area of the design in Figure 9, demonstrating that area reduction resulting from pre-placement of latches is quite

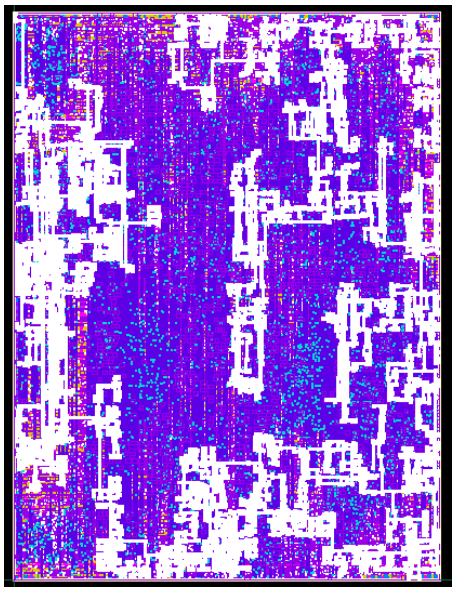

Fig. 10. Local clock wires, no latch pre-placement

measurable.

E. Instantiation and pre-placement of decoupling buffers

Instantiation and pre-placement of decoupling buffers is used when a long wire or non-critical block of logic needs to be decoupled from the critical path, in order to improve timing. The typical case for using this technique is when an output of the latch drives a functional unit that is on the critical path and a signal is going out of the unit, shown as case 1 in Figure 11. The second case, shown as case 2 in Figure 11, is when an output of a latch drives several functional units, some of which are not timing critical. If the decoupling buffer is not instantiated explicitly in VHDL, it will be inserted by the synthesis tool (although sometimes the tool may attempt to increase the size of the gates on the critical path instead); in such case we do not have a handle on the cells comprising the decoupling buffer, and have to rely on the automated placement tools to place them in appropriate locations. Quite often those cells are placed rather far from the latch, leaving a long piece of wire overloading the gates on the critical path, and resulting in poor convergence on timing.

To solve the problem, decoupling buffers are instantiated in VHDL and preserved in the synthesis and post-synthesis steps. Overlapping regions for pre-placing the buffer cells are created in the physical

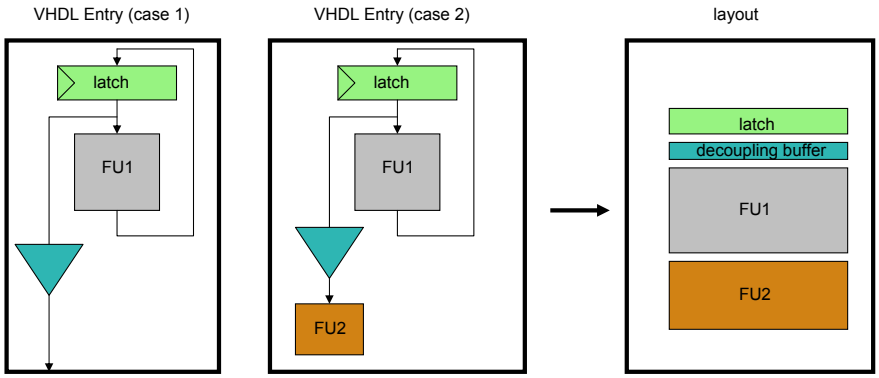

Fig. 11. Instantiation and pre-placement of decoupling buffers: methodology

image of the unit, very close to the pre-placement regions for the corresponding latches, as illustrated in Figure 11.

This technique provides the following advantages: the power level of the decoupling buffers is precisely controlled, without impacting the gates constituting the components . This allows the tools to keep small the power level of most books inside the unit, using high-power books only where they need to drive long wires or high fan out. The decoupling of high capacitance nodes from critical paths improves speed and improves the convergence on timing.

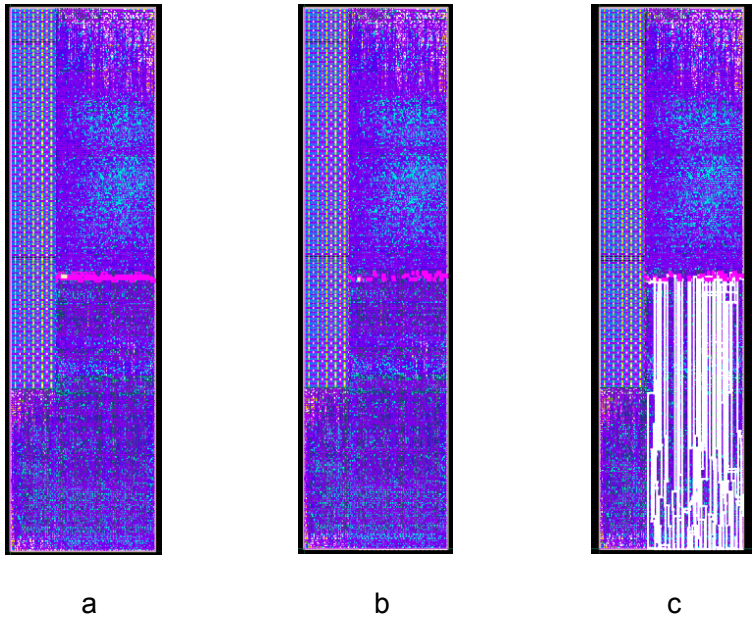

Fig. 12. Instantiation and pre-placement of decoupling buffers; a: 40-bit datapath latch; b: decoupling buffer; c: output wires

Figure 12 shows the layout of one slice of the 40-bit accumulator datapath in the eLite DSP. High-

lighted in Figure 12(a) is a 40-bit datapath latch which supplies one of the operands to the functional units located above the latch, and also drives wires going out of the datapath. The output pins are located at the bottom of the layout. Figure 12(b) shows the instantiated decoupling buffer that decouples the capacitance of the output wires from the critical path. Comparing Figure 12(a) and Figure 12(b), we see that the cells of the decoupling buffer are placed next to the corresponding latches, so that the wire load on the critical path is minimized. Figure 12(c) shows the output wires going from the decoupling buffer to the output pins.

V. Clock gating

In an effort to save power, our methodology attempts to clock gate any multi-bit latch that is 8 or more bits wide that is not being used in the current cycle. (The overhead for clock gating latches with smaller number of bits was measured to be too high due to the need to instantiate a separate clock splitter for each individually clock-gated latch, as well as logic and possibly extra latches to generate and re-time the clock gate signal.) In many cases though, the control logic for generating the clock gate signal for latches eligible for clock gating may be too complex to make the tight setup time on the clock gate path. Still, it is desirable to be able to clock gate multi-bit latches at least in the *majority* of cycles they are not used. We call such a scheme a "loose" clock gating. To define the "loose" clock gating condition, we work backwards from the clock gating timing restrictions, attempting to cover by clock gating as many conditions a latch is not used as possible.

For example, the 32-bit latch in the Integer Unit at the output of several functional units should not be clocked if the current instruction has failing predication. However, such a precise resolution on clock gating conditions requires reading the appropriate bit from the Condition Register, and comparing it to the immediate bit in the instruction field. This logic does not meet the timing constraints of the clock gating setup time. In this example, the Integer Units's data latch is clocked (un-gated) under the "loose" conditions, whenever the Integer Unit has a valid data instruction, regardless of predication. Of

TABLE I

USAGE OF CLOCK GATING IN THE SCALAR UNITS.

Scalar Units	Decoder	Branch	Integer	Address
total number of latches (including registers)	792	889	1,038	1,040
latches eligible for clock gating	473	734	975	911
latches with fine grain gating implemented	0	414	783	719
latches with loose clock gating implemented	0	96	192	192
latches with no clock gating implemented	473	224	0	0

course in this manner, the data latch switches in some cycles when it does not need to, but the "loose" clock gating methodology proves to be a good compromise between the "all or nothing" strategy.

Still, there are cases in the design in which latches can not be clock gated even with the "loose" gating strategy. In our design this occurred many times in the Branch Unit and in the Decoder, where control logic could not be pipelined. Table I shows four categories for each of the scalar units in the eLite DSP. We don't show data for the vector units, because almost all of the eligible latches are clock gated in those units. The first row shows the total number of latches in each of the units, including architected registers and register files. The second row shows the number of latches that were eligible for clock gating. The third row shows the number of latches where the "strict" fine grain clock gating was implemented. These are the latches that are clocked only when the data at the output are used, and clock gated in all other cycles. The fourth row shows the number of latches where "loose" gating was used. The last row shows the number of latches that were unable to be gated even with "loose" gating technique because the logic for generating the clock gating conditions did not meet the timing constraints.

The grouping and pre-placement of latches, clock splitters and clock gating cells, shown in Figure 7, significantly reduced the setup time requirement on the clock gating paths. For a 500 MHz design with OR-style (or early mode) clock gating, the time allowed for calculating clock gating conditions

was increased from 0.1ns to 0.6ns, allowing from 3 to 4 levels of logic on the clock gating path. With a stricter setup time on the clock gating signal, the number of clock gated latches would be dramatically reduced.

VI. Application to the design of Large Register Files

The design of large multi-ported register files can present a significant challenge to the ASIC flow. In this section we further demonstrate the effectiveness of our design methodology through the design of a 512-entry x 16-bit wide register file with 8 read ports and 4 write ports. This register file is one of the key components in the architecture of the DSP as described in [11].

The VHDL for the register file is constructed hierarchically as follows. We begin by designing a register file with a small number of entries, such that it can be synthesized flat. We call this a leaf cell. In this case we use 8 entries as our starting point. In the leaf cell, all latches in a given register entry are grouped for the purpose of clock splitter insertion, as described in Section IV-C.

Next, we aggregate a number of leaf cells to form a bigger register file in a hierarchical manner. In this example, we use 4 leaf cells to create a 32-entry register file. We repeat this hierarchical composition until we arrive at a register file with the desired number of entries. Thus, the 512-entry register file is implemented in 4 hierarchical levels: 8-entry leaf cells, 32-entry, 128-entry and 512-entry.

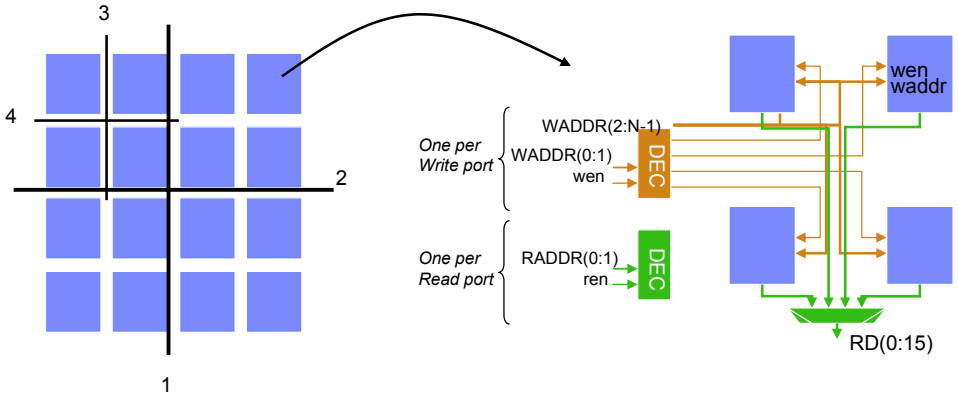

Fig. 13. Hierarchical Synthesis of Large Register Files

At any level of the hierarchy, the structure is the same as shown in Figure 13. For each write port, the most significant write address bits are decoded to enable writing to only one of the underlying

register file partitions, while the least significant address bits and data busses are broadcast to all underlying register file partitions. Similarly, for each read port, the most significant read address bits are used to select the data output from one of the underlying register file partitions, while the least significant address bits are routed to the read ports of all underlying register file partitions. The number of read/write address bits used for decoding is log2(number of partitions), in this case log2(4) = 2 bits.

Taking advantage of the regularity of this scheme, a Perl program was developed to automatically generate the hierarchical VHDL for the entire register file. The program facilitates rapid exploration of various design points by parameterizing all relevant design choices, e.g. the size of the leaf cells, the number of partitions per hierarchical level, the total number of entries required, the number of read ports, the number of write ports and the width of each entry.

The synthesis process for the register file was carried out hierarchically, also with the aid of automated scripts. First, the leaf cell was synthesized, then each level of the hierarchy from the bottom up. The timing and capacitive loading constraints were multiplied by a factor (> 1) for every level of the hierarchy so as to match the increased area and average wire length of that level compared to the one below. Again, these factors are parameters accessible to the designer. Adequate values were found empirically by making several trial passes.

The output of the synthesis step is a mapped netlist that was post-processed to replace behavioral flip-flops with LSSD latches, add clock splitters and scan chains, as described in section IV-C.

During the physical design, the physical image of the register file was hierarchically organized into regions that match the synthesis hierarchy. Leaf cells were assigned to the placement regions at the bottom of the hierarchy. All remaining logic at the second level of the synthesis hierarchy, including decoders, output multiplexors, write control circuits and other "dust", Figure 13, was assigned to the pre-placement regions at the second level of the physical hierarchy, so that all the logic that glue the leaf cells into a larger register file at the next level of the hierarchy was placed next to the corresponding

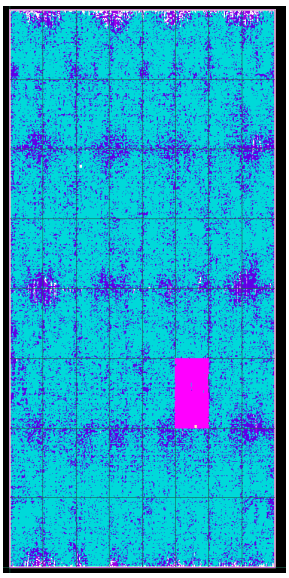

Fig. 14. Layout of 512 entry, 16-bit 8R / 4W ports vector register file

leaf cells. The process was repeated until the top of the hierarchy was reached. The process was fully automated with Skill scripts that drive the physical design tools.

The layout of the completed 512-entry register file design is shown in Figure 14. Highlighted are the gates constituting on of the leaf cells in the register file. The area of the implemented 512-entry register file is $3.19mm^2$, while the internal cell area of the mapped netlist is 1.2M ICells. The placement and routing overhead is only 30%, including the overhead of N-well and substrate contacts. The placement and routing time was 3 hours. The register file times at 510MHz under nominal conditions.

VII. CONCLUSIONS

A semi-custom design methodology has been prototyped/demonstrated through the development of low-power high-performance digital signal processor core. Significant speed improvement was achieved, compared with standard ASIC flow, with some critical paths reduced from 3ns to 2ns. A significant area reduction was achieved due to the dominant usage of low-power cells and improved routability. Power savings in the range of 50% were achieved. The careful pre-placement of clock splitters and clock gating circuitry allowed more time for calculating the clock gating conditions. For 500 MHz design with OR-style clock gating, the time allowed for calculating clock gating conditions was increased from

0.1ns to 0.6ns, allowing us to clock gate 90% of eligible latches, using the highly efficient OR-style (early) clock gating.

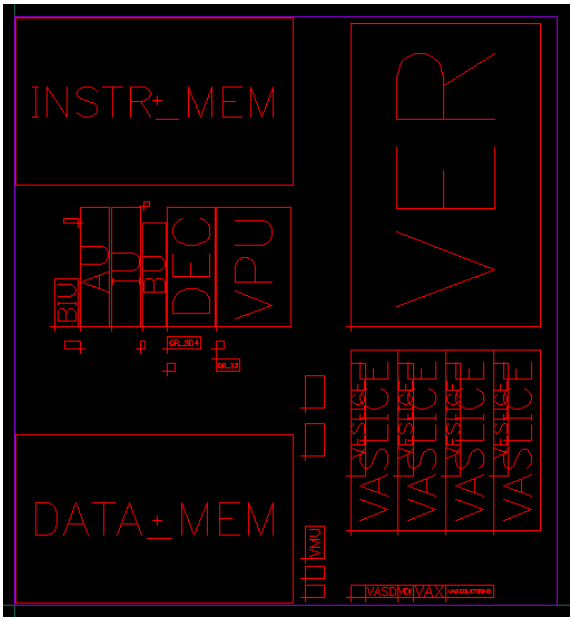

Fig. 15. eLite DSP core: placed components

This methodology does not compromise the generality and high productivity of the standard ASIC flow. It preserves generic RTL (VHDL) code which is easy to maintain and simulate, and which can be used to implement the design in different technologies. The code does not use any technology specific gates, such as LSSD latches, clock buffers or clock splitters, and it does not have any scan chains.

Short time from VHDL to layout and post-PD timing results was achieved. The fast turn-around time to close on timing allowed us to go through up to 3 VHDL-to-layout iterations per day performed by 2 to 3 designers, with consistent convergence on timing. The large size of individual macros (on the average, 100K ICells per macro) resulted in a small number of macros, which simplified the core integration.

As an example releasing a test chip implementing a MAC unit with testability support took only 4 days: the synthesis and post-synthesis step took 1 designer x 1 day; Layout, post-PD timing took 1 designer x 1 day. Three iterations to close on timing at 500MHz under worst-case conditions took 2 designer x 1 day. The power grid and PD closure (LVS, DRC, chip pads) took 1 designer x 1 day.

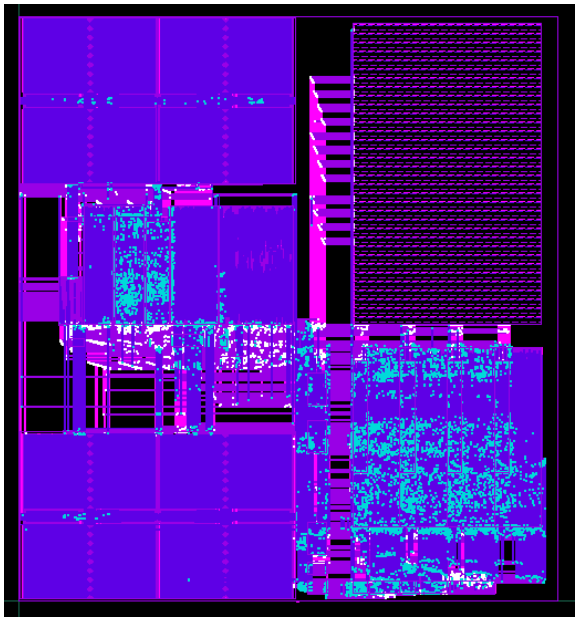

Fig. 16. eLite DSP core: placed and routed core

Finally, Figures 16 show the layout of the eLite DSP core. The core consists of 4-wide SIMD vector unit, located in the bottom-right corner, a set of scalar units, decoder and vector pointer unit, located in the middle, 12-port custom register file, located in the upper-right corner, and low-power custom instruction and data memories.

References

[1] D. H. Allen et al. Custom circuit design as a driver of microprocessor performance. *IBM Journal of Research and Development*, 44(6):799–822, 2000.

[2] S. Askar and M. Ciesielski. Analytical approach to custom datapath design. In *International Conference on Computer-Aided Design*, pages 98 –101, November 1999.

[3] R. M. Averill et al. Chip integration methodology for the IBM S/390 G5 and G6 custom microprocessors. *IBM Journal of Research and Development*, 43(5):681–707, 1999.

[4] D. Brand et al. In the driver's seet of booledozer. In *International Conference on Computer-Aided Design*, pages 518–521, 1994.

[5] Y. Chan et al. Physical synthesis methodology for high performance microprocessors. In *Proceedings of Design Automation Conference*, pages 696–700, 2003.

[6] D. Chinnery and K. Keutzer. Closing the gap between ASIC and custom. In *Proceedings of Design Automation Conference*, pages 637–642, 2000.

[7] A. Conn et al. Gradient-based optimization of custom circuits using a static-timing formulation. In *Proceedings of Design Automation Conference*, pages 452–459, June 1999.

[8] W. Dally and A. Chang. The role of custom design in ASIC chips. In *Proceedings of Design Automation Conference*, pages 643–647, 2000.

[9] IBM Microelectronics Division. Einstimer user's guide and language reference, 1995.

[10] S. Kosonocky et al. Low-power circuits and technology for wireless digital systems. *IBM Journal of Research and Development*, 47(2/3):283–298, 2003.

[11] J. Moreno et al. An innovative low-power high performance programmable signal processor for digital communications. *IBM Journal of Research and Development*, 47(2/3):299–327, 2003.

[12] Moser. LSSD test architecture. IBM Tech. Disc. Bull., August 1981.

[13] G. P. Rodgers, I. G. Bendrihem, T. J. Bucelot, B. D. Burchett, and J. C. Collins. Infrastructure requirements for a large-scale, multi-site vlsi development project. *IBM Journal of Research and Development*, 46(1):87–95, 2002.

[14] L. Stok et al. Booledozer: logic synthesis for ASICs. *IBM Journal of Research and Development*, 40(4):407–430, 2001.

[15] Warnock et al. The circuit and physical design for the Power4 microprocessor. *IBM Journal of Research and Development*, 46(1):27–53, 2002.

[16] T. Ye, S. Chaudhuri, F. Huang, H. Savoj, and G. De Micheli. Physical synthesis for ASIC datapath circuits. In *IEEE International Symposium on Circuits and Systems*, pages III–365 – III–368, May 2002.

[17] T. Ye and G. De Micheli. Data path placement with regularity. In *International Conference on Computer-Aided Design*, pages 264 – 270, 2000.

[18] V. Zyuban and S.V. Kosonocky. Low power integrated scan-retention mechanism. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 98–102, August 2002.

[19] V. Zyuban and P. Strenski. Unified methodology for resolving power-performance tradeoffs at the microarchitectural and circuit levels. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 166–171, August 2002.

[20] V. Zyuban and P. Strenski. Balancing hardware intensity in microprocessor pipelines. *IBM Journal of Research and Development*, 47(5/6):595–598, 2003.