RC23014 (W0311-156) November 21, 2003
Electrical Engineering

IBM Research Report

A Semi-Custom VLSI Design Flow and Its Application to the Branch Address Calculator in IBM Power4 Microprocessor

Pong-Fei Lu, Gregory A. Northrop, Kevin A. Chiarot
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

A Semi-custom Design Flow and Its Application to the Branch Address Calculator in the IBM Power4 Microprocessor

Pong-Fei Lu*, Gregory A. Northrop*, Kevin Chiarot**

ABSTRACT

In this paper we present the design and implementation of the branch address calculator in the Instruction Fetch Unit (IFU) of the IBM Power4 Microprocessor which operates at 1.7 GHz in a 0.18 μm SOI technology. A semi-custom methodology combining flexible custom circuit design with automated tuning and physical design tools is shown to provide new opportunities for optimization of designs throughout the development cycle. The resulting branch calculator supports a 3-cycle branch redirect loop to the L1 cache, which is key to the IFU performance. To achieve high fetch bandwidth, eight branch calculators are used to calculate the branch addresses in parallel for the eight instructions from the L1 cache. The replication of hardware makes the power-performance tradeoff an important issue in the circuit implementation. It is shown that with careful optimization, high performance can be achieved with a robust, tuned static design, thereby maintaining a power efficient design point.

Index Terms- Standard cell, circuit tuning, semi-custom design, methodology, adder, branch address.

1. INTRODUCTION

The development of high performance microprocessors requires concurrent design at many levels (logical, circuit, physical) with large teams and tightly interlocked schedules. Often the best design flow is one that most effectively addresses the natural conflicts within this flow (e.g., logic stability vs. timing closure), in contrast to one that simply applies the most modern or aggressive approach in each domain. For example, full custom design is very effective at optimizing performance and achieving high area efficiencies, particularly where elements are identical across the bit range of a dataflow stack, where design hierarchy can be exploited with careful tiling to produce highly optimal designs. This clearly applies to register files, working registers, multiplexors, and the like. However, many of the most labor intensive and critical functions, particularly those that implement more complex numerical functions (adders, incrementers, decrementers, and comparators) do not make such a compelling case for full custom design. They are far less regular across the stack, more complex, and often do not tile very easily. They are often timing critical, and although the logical function usually has a stable definition early in the design process, the most appropriate circuit architecture may change throughout the design cycle depending on the maturity of the input timing assertions and the requirements at the outputs.

Also, a new processor design project usually spans several generations of CMOS technology from the concept phase to the final product, and may often include multiple technology remaps over the life of the product cycle. The circuit simulation model for the technology under development needs to be extrapolated from the current one, and may contain large uncertainties. This 'pre-manufacturing' model used for performance assessment in the early design phase may change significantly in device electrical characteristics as well as the physical design rules throughout the development cycle. With conventional full-custom design styles, the long lead-time of manual layout works for those complex logic blocks mandates early commitment in the physical design before the technology is fully stable. The design may therefore run a risk of requiring major rework to accommodate late timing fixes or design changes. A more versatile design style combining flexible layout and performance tuning is desired to achieve the needed fast turn-around time in a fluid design environment. This paper describes such a 'semi-custom' design methodology, and its application to the design of the branch calculator of IBM Power4 processor. The coordinated use of a common parameterized gate representation, standard cell generation capabilities, place and route merged with custom physical design, static transistor level timing coupled with formal circuit tuning, have all led to significant improvements in both quality-of-result and time-to-market in the conventional static CMOS design domain. Inside the Instruction Fetch Unit (IFU) branch address calculator, a 24-bit binary adder, a 38-bit incrementer and a 38-bit decrementer are designed using this new methodology. Coupled with careful pipeline optimization, this implementation meets the 3-cycle loop requirement for the taken-branch address redirect, which is pivotal to the IFU performance. Quick turnaround for a late change in the adder design also demonstrates the flexibility and performance tuning advantages of this semi-custom methodology in a changing design environment.

The paper is organized as follows. Section 2 gives an overview of the semi-custom methodology flow. Section 3 describes the micro-architecture requirements and the circuit implementation of the branch address calculation in Power4 microprocessor. Section 4 presents the physical design of the functional blocks including the adder, incrementer, and decrementer using the semi-custom design technique. Section 5 concludes the paper.

Table 1 List of parameterized logic types

Logic type	Comments
INVERTER	
NAND2…NAND4	Multiple T
NOR2…NOR3	Multiple T
AOI21 & OAI21	Multiple T
AOI22 & OAI22	
XOR2 & XNOR2	Pass-gate style
MUX2	2-Way transmission gate mux
PGMERGE	G+P*C (restricted AOI21)
DRXOR2	Dual rail xor (true & comp inputs)

2. SEMI-CUSTOM DESIGN FLOW

2.1 Primitive parameterized bookset

The basic building block used in this methodology is a set of parameterized gates, called the primitive bookset, an example of which is shown in Fig 1. These gates are generally a single level of conventional (inverting) static CMOS, with complimentary pull-up and pull-down nfet and pfet trees, and the

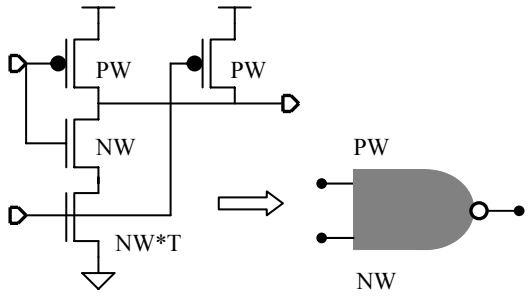

Figure 1. Parameterized gate and sample instance, showing parameters NW and PW. The expressions on the transistors are widths, and T represents an optional taper factor.

parameterization simply scales the pfets with a parameter PW and the nfets with a parameter NW. In general, each fet can have an additional fixed multiplier, T, which is used to define multiple flavors of each logic type making tradeoffs between the delays from each input pin. With the exception of the XOR and XNOR functions, all these primitive gates are a single level of inverting logic. A complete set of represented topologies can be found in Table 1.

Together, this basic set of parameterized gates is capable of covering most of the design space for combinational static circuitry found in a conventional ASIC library, as more complex functions, such as wide ANDs and ORs are typically composed of multiple levels of these gates. Note that this bookset is only a schematic representation; there is no directly associated layout representation.

A cell generation tool, described in more detail in section 2.5, is designed specifically to produce layout in a row based standard cell image for arbitrary values of NW and PW for each primitive cell. The design flow used to implement custom designs that use the primitive bookset is summarized in Fig 2. This flow is largely automated, with most of the work contained in the initial design and the place and route floorplan. After an initial pass, iteration of the design is a relatively rapid process, making possible cost-effective trials of multiple design implementations, and late adjustments for changing performance requirements, both of which are impractical when applying full custom physical design techniques.

2.2 Design flow step details

This section catalogs some of the details of each step in the flow. Note that not all steps are always needed, and a wide variety of combinations have been used, depending upon the details and needs of each design. The primary goal of this flow is to retain as much detailed control of the design at the schematic level as possible while relying on layout automation, particularly place and route techniques, for a rapid physical implementation.

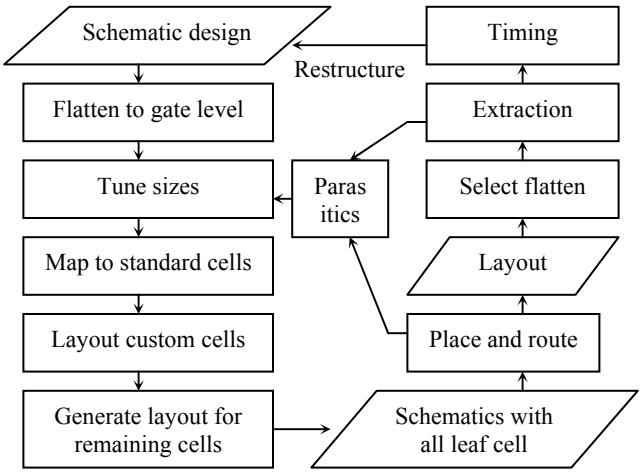

Figure 2. Complete design flow using the primitive bookset. The custom designed schematics can be a hierarchical combination of parameterized and custom designed cells.

2.2.1 Schematic design (contents)

The initial schematic design can be all or part of a macro, but generally should encompass function that can be floor-planned in a simple block, and whose components can be routed automatically. It can contain hierarchy, with the assumption that it will be flattened to a set of routable leaf cells going into the final physical design. It can contain a mixture of parameterized primitive gates, cells from a standard cell library, and custom cells.

2.2.2 Gate level flattening

Even though the design will be flattened to the leaf cell level going into the physical design process, there can be significant advantage to flattening a hierarchical design while still in schematic form, particularly when tuning a design with low symmetry in its structure or timing constraints. The flattened schematic gives the circuit tuner maximum flexibility in adjusting transistor widths ('PW' and 'NW') of each gate although the starting schematic is usually hierarchical for ease of entry and readability. Both the hierarchical and the flat schematics are kept in the database.

2.2.3 Static circuit tuning

Automated tuning of fet widths to optimize the timing slack of a macro is one of the key elements of this methodology. This tuning allows the exploitation of the advantage of the sizing flexibility of the primitive bookset to optimize the timing and area of the design. An expanded discussion of the tuning method can be found in section 2.4.

2.2.4 Mapping to standard cell library

After sizing, many of the parameterized gates can often be mapped to cells from a fixed standard cell library, with only a modest change in fet sizes and a minimal impact on timing. This helps to control data volume since all remaining cells must be uniquely generated for use specifically by the macro in question.

2.2.5 Inclusion of custom leaf cell physical design

Custom leaf cells for the design must be abstracted for place and route. These can be either standard cell image, or bit-slice blocks that are pre-placed.

2.2.6 Automated cell generation

Any parameterized cells that remain after mapping to the standard cell library are generated. Details of the generation process can be found in section 2.5.

2.2.7 Floorplanning, place and route

An interactive floorplanning and place and route environment is used to complete the layout of the designs. This environment features:

1. Manually constructed floorplan with detailed wire contracts and pin locations. If the semi-custom block is imbedded in a macro, the wire contracts need to accommodate through-wires at the next level of hierarchy, so that re-routing will not interfere with macro assembly. As long as the abstract is obeyed, the semi-custom block can be re-designed in arbitrary ways even after the top level macro is done.
2. Customizable row configurations for placeable cells, capable of multiple site types. Typical designs use a row height image which is 16 wiring tracks high, but for designs involving many small devices, 12-track or 14-track images might be more suitable.
3. Pre-placement of non-placeable and placeable cells.
4. Placement constraints (regions, weights) to better guide the placement and routing tools.
5. Code-based or manual pre-routing.
6. Grid based router (Cadence Warp Router).
7. Incremental (EC) placement to retain stable timing when iterating a design.

In addition, the finished design has the following enhancement options:

1. Flattening and post-processing the final layout to trim the poly-gate and unused metal contact shapes in the (multi-fingered) leaf cells to reduce parasitic capacitances.
2. Backfill of the 'gaps' in the layout with de-coupling capacitors for better power-supply noise reduction.

2.2.8 Cell count and data volume reduction

After completion of place and route, designs that have made heavy use of tuning and cell generation will have a large number of unique cells, many used only once or a small number of times. Selective flattening of these cells in the layout, with a corresponding change back to the parameterized schematic representation, helps strike a balance between a high cell count and the large data volume of a completely flat layout.

2.2.9 Parasitic feedback into tuning

The effects of wiring parasitics have become quite important, accounting for as much as 30% of the delay in even moderate sized macros. Lumped capacitance, from either wire length measurements (automatically extracted from the place and route data) or from a subsequent full extraction of the layout, can be merged into the schematic netlist and the circuit re-tuned to compensate for its effect.

2.2.10 Design restructuring & alternate circuits

The combination of automated sizing and timing-based real physical design allows the designer to try multiple restructurings and circuit architectures, and make a confident comparison of the relative quality of each approach. This part of the flow, and the associated change in the approach to design, are the most important part of this methodology, and the place where the most benefit will be found when it is fully applied.

2.3 Circuit Tuner

A pivotal driver of this methodology is the circuit tuner used to automate the sizing of transistors. The tool, called EinsTuner [2-4], formulates the optimization problem through static timing, optimizing slack in the presence of timing assertions. The large, non-bitslice circuits for which semi-custom design is best suited are ideal candidates for static tuning, since the tuner can keep track of a large number of critical paths as tuning proceeds.

The EinsTuner static tuner is built on top of a static transistor-level timing tool (EinsTLT), which combines a fast event-driven simulator (SPECS) with a timing tool (Einstimer). The SPECS simulator provides timing information (delay and slew) along with first derivatives with respect to circuit parameters, specifically transistor width. EinsTuner uses this information to formulate the optimization problem for solution by a large-scale general-purpose nonlinear optimization package LANCELOT [5], generally optimizing a linear combination of slack and area, nominally treating all fet widths as free parameters. Additional features of this tool that make it effective in a practical design environment include:

- Parasitics (lumped capacitance or RC) from physical design.
- Area modeled as the sum of fet widths.
- A fet-width ratioing mechanism, used to constrain fet widths to match hierarchy or gate parameterization.
- Input capacitance, node slew, effective pull-up and pull-down (beta ratio), and min/max fet-width constraints.
- Complete interactive environment (GUI), including size constraint generation and back annotation.

In its current state of development, EinsTuner is capable of tuning in excess of 3000 gates with run times normally < 24 hours (with lumped capacitance). Thus even large circuits, such as a 64-bit adder (~2000 gates), can be tuned with reasonable turn around time. Experience has shown that tuning results are largely independent of the starting point, meaning that a designer can have a high degree of confidence that the results from a run are optimal for the conditions and design.

2.4 Cell generation

A key component of the semi-custom design methodology is the use of a cell generator to create layout corresponding to the parameterized gates. This in-house tool, called C-cell, is not a general-purpose cell compiler, but rather a script-based system designed to produce optimal layout, but only for the defined parameterized bookset. The definition of the parameterized gate set is tightly integrated into this tool, delivering a framework that supports semi-custom design in a number of ways:

- Contains functions to generate a set of layouts and associated views for use as a standard cell library, based upon a list of cell specifications: (gate, NW, PW).
- Provides features to parse a custom schematic, forming a list of cells to generate to replace parameterized gates, and minimizing the number of required cells assuming a maximum allowed deviation in size. After this, cells are

generated and modified custom schematic is created to reference the generated cells.

- Includes a facility to convert between parameterized and standard cell (fixed-size cell) representation.
- Has an integrated floorplanning aid with an interface to the place and route tool.
- Does layout post-processing, including selective layout flattening.

In the cell generation part of the tool, topology and technology specific code takes as input the gate type, size parameters NW and PW, and a global cell image, and generates layout after selecting the optimal configuration from a range of finger partitioning and topology options. In practice the measure of optimality is cell area, but factors such as wireability, manufacturability, etc., could also be weighted in the selection. While this system is not capable of implementing an arbitrary gate topology, it has been very successful in the domain of conventional static CMOS, where there are a small number of effective topologies, and optimization of the simplest (nand/nor) types is vital. To date, the effort required to migrate and modify this approach from technology to technology has been easily justified in its use in both the high-performance synthesis environment and in semi-custom design.

3. Branch Target Address Calculation in IBM POWER4 Microprocessor

In the previous section, we described the new semi-custom design methodology. In this section, we describe the application of this methodology to the branch calculator of the IBM Power4 processor design [6,7]. We will first review the micro-architecture requirements of the design and will then discuss the circuit implementation.

3.1 Micro-architecture

The branch address calculator contains the logic required to calculate the branch-taken next instruction address for four different types of branch instructions: (1) Branch I-form (‘*b*’); (2) Branch Conditional B-form (‘*bc*’); (3) Branch Conditional to Link Register XL-form (‘*bclr*’); and (4) Branch Conditional to Count Register XL-form (‘*bcctr*’) [8]. The B- and I-form instructions contain different width immediate fields that specify either an absolute address or a relative offset to the current instruction address (CIA). Thus, the result of a calculation is either a sign-extended absolute address or the sum of the CIA and the sign-extended relative offset. The calculated target addresses will then be sent to the branch instruction queue for handling at a later stage, or will be sent to the Instruction Fetch Address Register (IFAR) in case of a branch re-direction.

Figure 4 shows the timing and the block diagram of the branch redirect dataflow in IBM Power4 design. In cycle 1, eight instructions of 32 bits each are delivered by the I-cache (64KB, direct-mapped [6]). In cycle 2, the cache data en route to the branch calculator is muxed with other bypass data, and then fed to the calculator where eight branch addresses are calculated in parallel. In cycle 3, the final branch address is selected through an 8:1 priority mux, merged with other IFAR dataflow and then sent back to I-cache to re-fetch. Physically, this loop path traverses through four macros with long wires in between: (1) I-cache SRAM; (2) the bypass mux macro; (3) the branch-calculator macro; and (4) the IFAR macro containing the instruction fetch logic. With this 3-cycle loop, predicted-taken branches can be executed with only a penalty of a 2-cycle bubble, which represents the best achievable branch performance with perfect prediction accuracy. Any cycle slip in this path will incur a significant CPI (cycle per instruction) penalty. Since this is the most critical path in the IFU, separate master and slave latch designs are used to eliminate setup time penalty as well as to facilitate flexible latch placement (see [9] for details). The latch points in Figure 4 denote the positions of scan-testable cycle boundary (c1, or master) latches; the mid-cycle (c2, or slave) latches are not shown.

The detailed block diagram of the branch address calculator is shown in Figure 5. Note that PowerPC is a Big-Endian architecture; that is, bits in a word are ordered from left to right. The instruction is represented by ‘I(0:31),’ where bit 0 is the MSB. First, bit 4 (‘I(4)’) of each instruction is examined to differentiate between the “Branch” (op-code(0:5)=b’010000’) and “Branch Conditional” (op-code(0:5)=b’010010’) instructions to create a correct sign-extended immediate operand from either I(6:29) or I(16:29). This immediate operand is then added to the CIA, which is the instruction cache sector (1 sector=eight instructions) address, IFAR(38:58), concatenated with the instruction’s position (i.e. word-offset) in the sector and b’00’ since the instructions are always word-aligned (Table 2). The result of the addition is then 4-way muxed with: 1) the original sign-extended immediate operand based on the Absolute Address bit (“AA”=I(30)) of the instruction to support an absolute address [8], 2) the data from Link Stack (a register file for storing recently used Link Register data) for ‘*bclr*’ instruction, and 3) the data of Count Cache (a register file for storing recently used Count Register data) for ‘*bcctr*’ instruction [7,10].

Table 2

Instruction slot	IFAR(38:58) || word-offset field (0:2)
0	IFAR(38:58) || b’000’
1	IFAR(38:58) || b’001’
2	IFAR(38:58) || b’001’
3	IFAR(38:58) || b’011’
4	IFAR(38:58) || b’100’
5	IFAR(38:58) || b’101’
6	IFAR(38:58) || b’110’
7	IFAR(38:58) || b’111’

Although conceptually the eight branch calculators are eight individual copies, there are opportunities for sharing. Since the immediate fields are always 24 bits, the most significant 38 bits are common to all eight instructions. Like the low-order bits, there are four possible branch outcomes of address bit (0:37):

1) The sign-extended I(6) for immediate branch mode (‘*b*’ or ‘*bc*’ instructions with ‘AA’=1);
2) The link stack data (‘*bclr*’ instruction) for branch to link;
3) The count cache data (‘*bcctr*’ instruction) for branch to count;

4) The incremented or decremented IFAR(0:37) for relative branches.

In the latter case, if the sign-extended I(6) is a '0', indicating a forward branch, the IFAR will be incremented by one if the carryout of the 24-bit adder is a '1;' otherwise it will be unchanged. Likewise, if the sign-extended I(6) is a '1,' indicating a backward branch, the IFAR will be decremented by one if the carryout of the 24-bit adder is a '1' (i.e. underflow); otherwise it will be unchanged. Note that while the decrementer and incrementer can be shared since all 8 instruction use the same IFAR(0:37), the 24-bit adder and the associated target address muxes are unique to each instruction, thus are replicated 8 times (Figure 5). The selects for the muxes, 'sel(0:3)', which are unique to each instruction, are generated by a random logic macro outside the dataflow stack.

3.2 Floorplan and critical path

Physically, the branch calculation logic is partitioned into 3 macros: (1) 'IFTA,' which includes the adder and muxes for the low-order 24 bits; (2) 'IFTI,' which includes the incrementer, the decrementer, and the muxes for the high-order 38 bits; and (3) 'IFTM,' which includes various muxes to select the final 62-bit branch addresses sent to the IFAR and branch instruction queues (Figure 5). Note that a 'macro' is the basic block in timing and chip integration in the IBM Power4 design methodology [9]. The logic is partitioned into these three sections primarily to limit the total transistor count. With this choice of partitioning, the extraction, timing, and checking tools have reasonable runtime. Because of the distinct dataflow of high-order and low-order bits, the separation of IFTA and IFTI is relatively straightforward and there are only two interface signals from IFTA to IFTI for each of the eight instructions: the adder 'carryout' and the sign-extended I(6). Figure 6 shows the micrograph of the IFU and the three 'IFT' macros relative to the I-cache and IFAR macro. Figure 7 shows the close-up view of the IFT* macros. The total IFU data stack is constrained in height due to the processor core chip floorplan allocation; it is therefore crucial to minimize the vertical dimension of the adder layout that is replicated eight times in a 2 column x 4 row matrix.

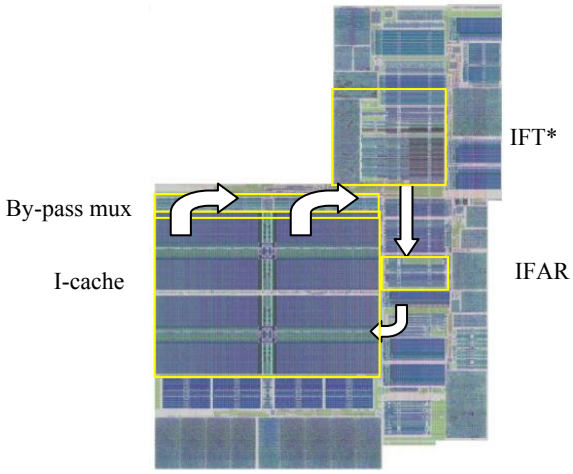

Figure 6: Micrograph of IFU. The arrows show the dataflow of the 3-cycle path. There are eight instructions (32 bytes) from I-cache outputs which turn 90 degrees to the IFT* macros. The branch redirect address is calculated in the IFT* macros, then sent to IFAR fetch logic macro for final selection. The target bits (48:58) are sent back to I-cache in the case of branch redirect.

The dataflow in Fig 5 represents the logic contained in cycle 2 and the first half of cycle 3 in the timing diagram (Fig. 4). The I-cache access consumes all of cycle 1, and the data are driven horizontally across the I-cache width (Figure 6). Leaving the cache, the total delay time is nearly half the cycle. This time includes the bypass mux, the buffers and the wire delay, and the 2:1 mux at the IFTA input. As a result, only half of a cycle for the adder delay remains to make the setup time at the cycle boundary c1 latch in cycle 2. Cycle 3 is fully consumed by the delay of the 8:1 branch mux in IFTM, buffer delay to drive to IFAR macro, the delay of the 3:1 IFAR late mux, the wire delay, and the fanout to various arrays. Therefore, there is no slack to steal from in cycle 3 and the adder delay must be less than a half-cycle in order to make the critical path.

The incrementer and decrementer are less timing-critical compared to the adder in performance. In Figure 5, the IFTI muxing structure for bit (0:37) is optimized by taking advantage of the timing skew in different paths. The signals from the Link Stack, the Count Cache (both register files outside the datastack), and the IFAR are all clock-launched out of latches. The signals arrive early and can be pre-muxed with the sign-extended I(6), which are also early compared to the incrementer and decrementer results. This arrangement effectively reduces the 5-way mux to a 2-way mux. The 2:1 mux, relevant only to relative branches, is gated by the ANDing of 'sel(3)' (selecting adder result) and the sign-extended 'I(6)' to determine whether it is a forward or a backward relative branch. If sel(3)=0, identical pre-muxed results flow through both 2-way muxes, and the carryout bit has no effect (i.e. non-relative branch). If sel(3)=1 and I(6)=0 (1), the incrementer (decrementer) result flows through one of the two 2:1 muxes, while the IFAR bits flows through the other 2:1 mux. The adder carryout gates the final selection. This pre-muxing scheme relaxes the performance requirements for the incrementer and the decrementer as there is nearly a cycle for the decrement/increment delay and 2:1 mux function. Note that the carryout is latched up in IFTA before it is sent to IFTI to drive 38 muxes. The latter forces the duplicate latches for the two interim 2:1 mux results in IFTI in order to align the cycle (cf. Figure 5). The end results are further muxed in the IFTM according to

priority encoding logic before they are sent to the branch queues, and in the case of predicted taken branches, sent back to IFAR to re-fetch from the instruction cache.

The sum and carryout from the 24b adder are both critical paths of the branch target address calculator. The incrementer and the decrementer are not as performance critical, but need to be layout efficient to avoid increasing the data stack height since the whole IFU floorplan is constrained in the vertical dimension. Therefore these used a 14-track row height instead of the 16-track image for the standard cells used elsewhere throughout this project.

4. Circuit Implementation

Due to the large transistor count of the chip (170 million), and the power constraints faced by the design team, the Power4 design is predominantly a static design [9]. Dynamic circuits are of used sparingly in SRAMs and other critical regions. Since the 24b adder is replicated eight times, it is important to stay within a strict power budget while striving for performance. Although well-tuned dynamic designs can achieve high performance levels, they are noise sensitive as well as power-hungry. Long design times and the sensitivity to model changes (p-to-n strength ratio, capacitive coupling, etc.) are also ill suited for an evolving design environment as in the Power4 project. In comparison, the static circuits enjoy an unequivocal robustness. Our approach to tackle the performance issue is through circuit tuning. The tuner optimizes the use of device widths in the most economical way by sizing up the gates in the critical path while sizing down the gates in non-critical path. With a tool-assisted physical design (PD) environment as described in Section 2, the design can be iterated rapidly. The designer chooses the best circuit architecture, and manages the timing assertions. The tuner then adjusts the device sizes to minimize the delay. The PD can be completely de-coupled from the macro assembly as long as the abstract is obeyed during design iterations. The advantage of this semi-custom design methodology is its tunability and PD flexibility. Using this approach, we have seen 10-15% performance improvement in typical logic macros for a given area/power constraint .

The surrounding latches and muxes use full custom design for better layout density. The semi-custom blocks are imbedded in the IFTA (adder) and IFTI (incrementer and decrementer) macro as shown in Figure 7. The IFTM macro is a full custom design. This mixed design style balances the layout density and design flexibility. The top-level macro wiring is done manually; the global wires are blocked out in the abstract of the imbedded semi-custom blocks to avoid conflicts during iterations.

4.1 Adder Circuit Architecture

The 24-bit adder is a static, carry-look-ahead Ling adder [11]. It is well known that the carry formation of the Ling adder is a stage faster than that of the usual adder. The 24 bits are grouped into six 4-bit blocks for carry propagation. Each block is sub-divided into two 2-bit groups due to the fan-in limit of 2 in the AOI and OAI in the bookset in Table 1. Within each group, the local sum lgeneration uses conditional-sum logic. The local sum is implemented in parallel with the 'pseudo-carry' [12], and the result of the pseudo-carry bit is used to select the final value.

One key circuit decision in the early design phase is to imbed the 4:1 mux in the adder and to hide the delay behind the adder (Figure 8). The assumption is that the I-cache data through the adder is the critical path, and the mux selects and register file data arrive early. Since the carry bit is the longest path, the partial sums can be pre-muxed with the other 3 inputs and the final result selected by the carry. The latter leads to a skewed 'late-mux' design as shown in Fig. 9(a). As typical in early phases of a new processor design project, this 'dataflow-centric' view drives the 3-cycle path delay assessment. It is assumed that controls are non-critical; however, this assumption turns out not to be true, as will be discussed later.

4.2 Circuit tuning and layout

Once the schematic design is complete and verified, the physical

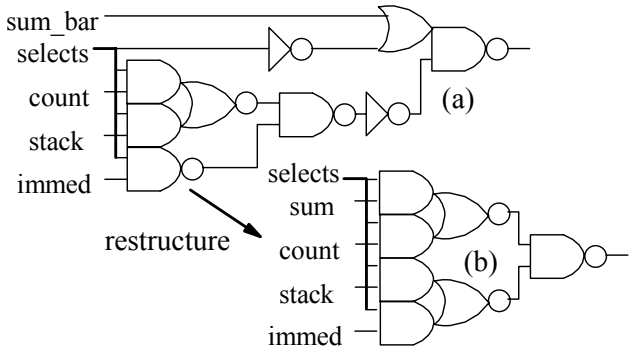

Figure 9: Branch mux implementations at the output of the adder. Initial implementation (a) assumed selects (from controls) were not critical, and final (b) with optimized control and sum paths.

design is implemented according to the flow shown in Fig. 2. The schematic hierarchy is first flattened to the gate level, then 'EinsTuner' is used to adjust the device widths. Using the netlist and the input/output (i/o) assertions (timing, capacitance load) as the input, device sizes (NW, PW) are optimized for each gate to minimize the delay. Other constraints such as total device width, max/min slew rates and beta ratio can also be set by the user. The convergence depends on the constraints; but in most cases it is well behaved. The tuner strives for minimum cycle time by examining all possible paths [2]. Therefore, the quality of the end result (i.e. whether the correct critical paths are optimized, and how much the actual performance gain is) depends on the preciseness of the i/o timing assertions. The latter presents a particular challenge for designs with transparent latches. Since the latches are all in flush mode, the timing assertions at the adder depend on the path delay from the I-cache to IFTA. To better control the timing and wireability, the major I-cache buses to IFT macros are engineered pre-routes [13,14]. To assist the critical paths through the low-order (38:61) bits, the IFU dataflow stack is arranged with the LSB (bit 61) facing the I-cache to shorten the wire length (Figure 7). The adder timing assertions are estimated from the IFTA macro timing 'back assertions' generated by the chip timing run by Einstimer. Note that not all branch addresses (0:61) have the same required arrival time as only bits (48:58) are used for I-cache address for the 3-cycle path depicted in Figure 4. These 11 bits are used to select one out of the 2K sectors of the

64KB L1 cache. Therefore, non-uniform timing assertions are applied to stress the timing-critical low-order bits (48:58).

The optimized parameterized gates are fed to the C-cell generator to generate the layouts and abstracts. The cell is 16 tracks in height, consistent with the standard cell library. However, no mapping back to standard cell library (as described in 2.2.4) is done since the adder is performance critical. The layout is done by place and route (P/R) in Cadence Silicon Ensemble. The abstract that defines P/R constraints is customized through manually adjusting the pin placement and metal blockage. A limited number of M2 and M3 tracks available for use in the adder as some tracks are reserved for the macro level routing through the stack. The abstract serves as a contract between the imbedded adder module and the macro. It is kept unchanged when iterating the design, so that the macro level wiring can be preserved. The final layout is flattened into shapes and a 'trimming' routine is used to cut excess metal and PC shapes of the unused pins to reduce parasitics. The trimming step improves the delay by about 2-3%. After the layout is extracted and timed, the parasitic capacitances are inputted to the tuner to iterate the process. The turnaround time is less than a day for macros containing ~500 gates. Usually two to three iterations are sufficient to bring the design to convergence.

4.3 Adder design iterations

The key issue of the semi-custom design methodology is the accuracy of the timing assertions fed to the tuner. As the control signal timing is unknown at the beginning of the processor design, the timing assertions are largely estimates that may not be substantiated as the chip timing stabilizes. In the 24-bit adder example, the assumption of the mux selects timing was found to be incorrect at the first tape-out when the control macros were built and timed. The arrival time of the select path was as late as the local sum from the adder, and the circuit construct in Fig. 9(a) was improper. To address these issues, the mux was re-designed with a balanced AOI-NAND scheme (Fig. 9(b)). The new schematic was then re-tuned and iterated based on the correct timing assertions. The new design was completed in a week, and the negative slack was reduced by more than 80 ps. Because of the contract-based place and route, the adder re-design did not cause any global wiring change. The adder delay including the built-in 4:1 mux is slightly greater than nine FO4, where one FO4 delay unit is the average delay of an inverter with fanout of four. The layout uses seven 16-track rows.

The other issue is that a new microprocessor design project like Power4 often straddles across several different technology generations. Each technology migration induces device model changes (for example, different p-to-n strength ratio) which are difficult to adjust to for full-custom designs. One benefit of the new semi-custom design approach is that the timing shifts can be readily accommodated by the tuner, and with the flexible physical design techniques, adjustments can be made in a timely fashion. We estimate that the semi-custom design approach reduces the total design time by at least 50%, even for designers unfamiliar with the place and route tools, who need to make an initial investment in learning the tools.

4.4 Incrementer and Decrementer

The physical design of the 38-bit incrementer and 38-bit decrementer follows the same flow of the adder design, but the area instead of the speed is the more constrained parameter in design tuning. Logically, the incrementer increases the input by one, which means that for every bit position 'i', if all of the previous lower-order bits are one, there is a carry, and the output needs to change state. Thus the logic expression is:

incr(i) = a(i) XOR (AND(all_previous_bits))

Conversely, the decrementer reduces the input by one. The latter means that for every bit position 'i,' if none of the previous lower-order bit is a one, the output needs to change state. The logic expression is:

decr(i) = a(i) XNOR (OR(all_previous_bits))

Thus, the critical path for the incrementer is a wide-AND tree, while for the decrementer it is a wide-OR tree. The wire-AND propagation for the incrementer is shown in Figure 10; the wide-OR has exactly the same topology except that the inputs are inverted. The chain is built by alternating NAND3 and NOR2 gates. It turns out that there are many small gates in the incrementer and the decrementer design due to the much simpler logic cone compared to the adder. For a row-based design, each gate has the same height. Small gates are not area efficient as they leave a lot of unused silicon area. Since the IFTI layout adds directly to the IFT* data stack height (see Figure 7), it is advantageous to reduce the row height for better efficiency. The C-cell generator described in 2.4 supports different cell images (12/14/16 tracks). Both designs are tuned by limiting the area to fit three 14-track rows. The standard 16-track row image would have cost six more tracks (48 vs. 42) per layout. The incrementer and decrementer delays are 10.7 and 11.4 FO4, respectively.

5. Conclusion

The semi-custom VLSI design methods described in this paper were shown to improve the ultimate quality and performance of high-speed circuits in a number of ways. The true impact extends well beyond the ability to either improve an existing design, for example, through circuit tuning, or to implement one quickly using place and route. Being able to adapt quickly to changes associated with global timing convergence is a major advantage of the semi-custom design approach. The biggest gains come through changing the approach to custom design. This methodology provides a true 'on-demand' solution in an evolving design environment, be it for the performance enhancement or for the area saving. We demonstrated the successful application of the semi-custom design methodology to the branch address calculator of the IBM Power4 microprocessor. The AC LBIST testing showed that the 3-cycle path described in this paper operates at more than 1.7GHz at 1.6 V.

6. ACKNOWLEDGMENTS

We thank Mary Wisniewski for providing the chip micrograph, Howard Levy and Eric Shorn the early IFT schematic design work, Brian Konigsburg for leading the IFU logic team, Joshua Frederich for AC LBIST data, and Jim Warnock for many circuit discussions, and IFU Circuit Team at Watson Research Center for collaboration. We also thank Chandu Visweswariah, Phil Strenski, and Ee Cho (circuit tuning), Joe Nocerra and Ching Zhou (cell generation), Brian Curran, Tom McPherson (timing convergence), and many designers for their experiences and suggestions.

7. REFERENCES

[1] G. A. Northrop, P.-F. Lu, "A Semi-Custom Design Flow in High-Performance Microprocessor Design," Design Automation Conference, 2001. Proceedings , pp. 426 –431.

[2] C. Visweswariah and A. R. Conn, Formulation of static circuit optimization with reduced size, degeneracy and redundancy by timing graph manipulation, IEEE International Conference on Computer-Aided Design, pages 244-251, November 1999.

[3] A. R. Conn, I. M. Elfadel, W. W. Molzen, Jr., P. R. O'Brien, P. N. Strenski, C. Visweswariah, and C. B. Whan, Gradient-based optimization of custom circuits using a static-timing formulation, Proc. Design Automation Conference, pages 452-459, June 1999.

[4] A. R. Conn and C. Visweswariah, Overview of continuous optimization advances and applications to circuit tuning, Proc. International Symposium on Physical Design, April 2001, Sonoma County, CA.

[5] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A). Springer Verlag, 1992.

[6] Anderson, Carl J., et. al., Physical Design of a Fourth-Generation POWER GHz Microprocessor, ISSCC Digest of Technical Papers, 232-233, Feb 2001.

[7] J. M. Tendler et. al., "POWER4 system microarchitecture," IBM J. Research and Development, 46, 2002, 5-25.

[8] PowerPC AS User Instruction Set Architecture, Book I.

[9] J. D. Warnock et. al., "The Circuit and Physical Design of the POWER4 Microprocessor," IBM J. Research and Development, 46, 2002, 27-51.

[10] http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/power4.html

[11] Huey Ling, "High-Speed Binary Adder," IBM J. Res. Develop., Vol. 25, No. 3, May 1981, pp. 156-166.

[12] M. J. Flynn and S. F. Oberman, "Advanced Computer Arithmetic Design," Chap. 1, 2001, John Wiley & Sons.

[13] "The physical design of on-chip interconnections," M. Y. L. Wisniewski, E. Yashchin, R. L. Franch, D. P. Conrady, D. N. Maynard, G. Fiorenza, I. C. Noyan, in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, March 2003, vol. 22, no. 3.

[14] M. Y. L. Wisniewski et al., "The Physical Design of On-Chip Interconnections: Part I: Quantification of Interconnect Properties," Research Report RC-22345, IBM Thomas J. Watson Center, Jan. 20, 2002.

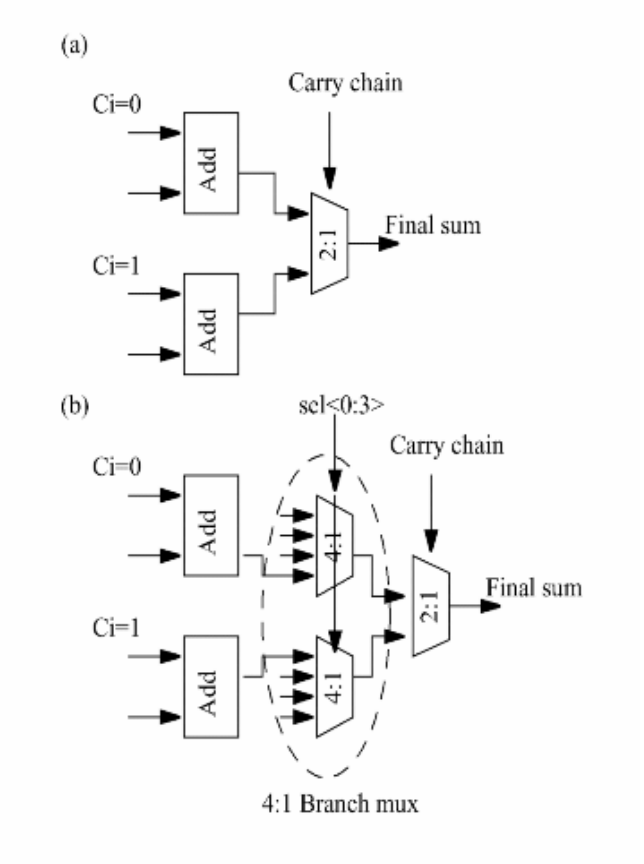

Figure 8: Schematic of (a) conventional carry-select adder, and (b) carry-select adder with imbedded 4:1 mux. The three other inputs to the 4:1 mux are link stack, count cache, and the immediate operand.

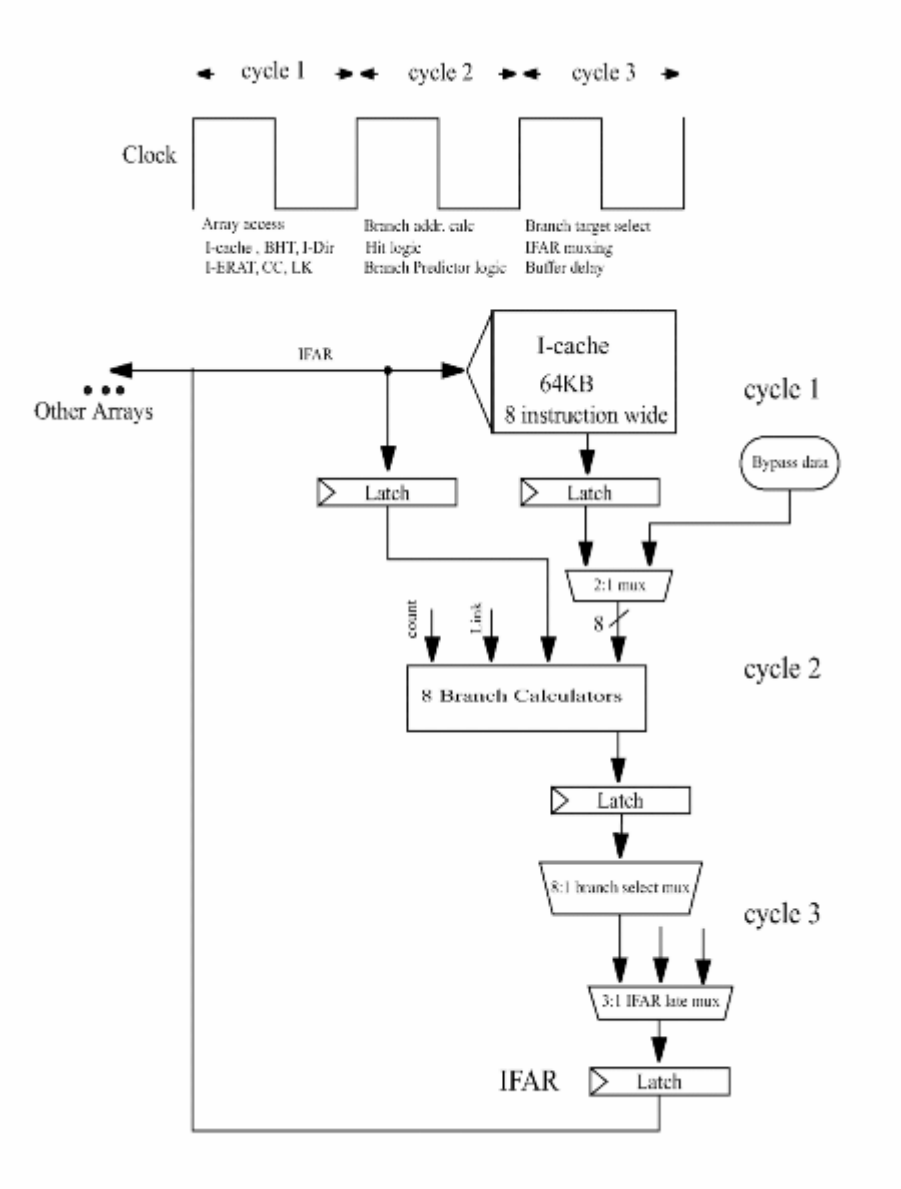

Figure 4: Timing and block diagram of the 3-cycle branch redirect path. Cycle 1 is the array access (BHT: Branch History Table; I-ERAT: Instruction Effective-to-Real Address Translator; I-Dir: Instruction Directory; CC: Count Cache; LK: Link Stack). Cycle 2 is the branch address calculation. Cycle 3 is the branch target address muxing and IFAR logic.

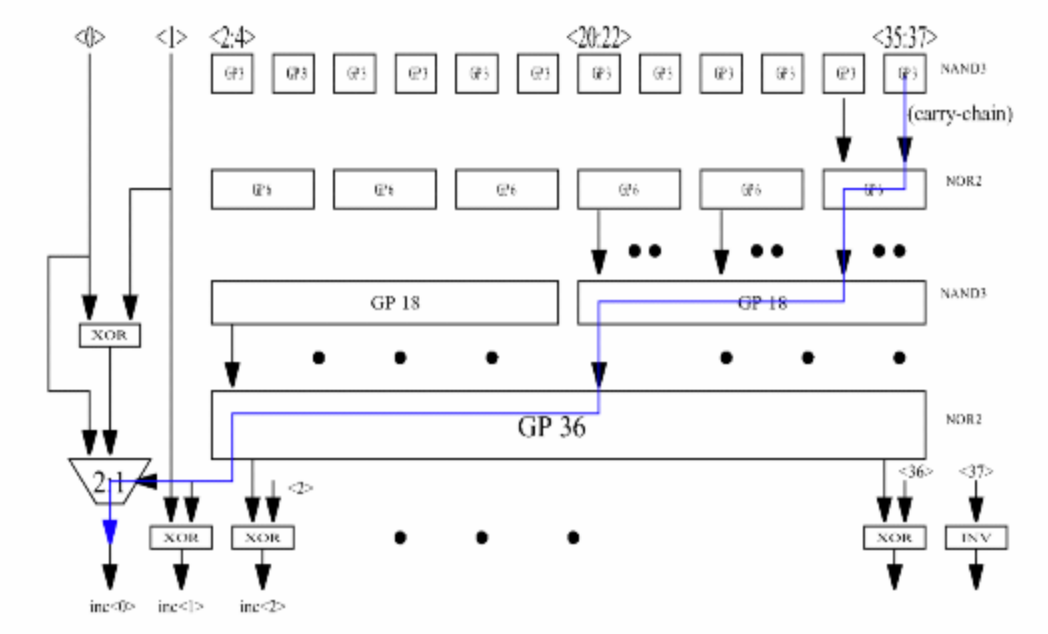

Figure 10: Carry-chain of an incrementer. The grouping is shown as 3-2-3-2, with alternating NAND and NOR. The (LSB) incr<37> is the inversion of the input. The (MSB) incr<0> is XOR of bits <0:1> and the carry-in.

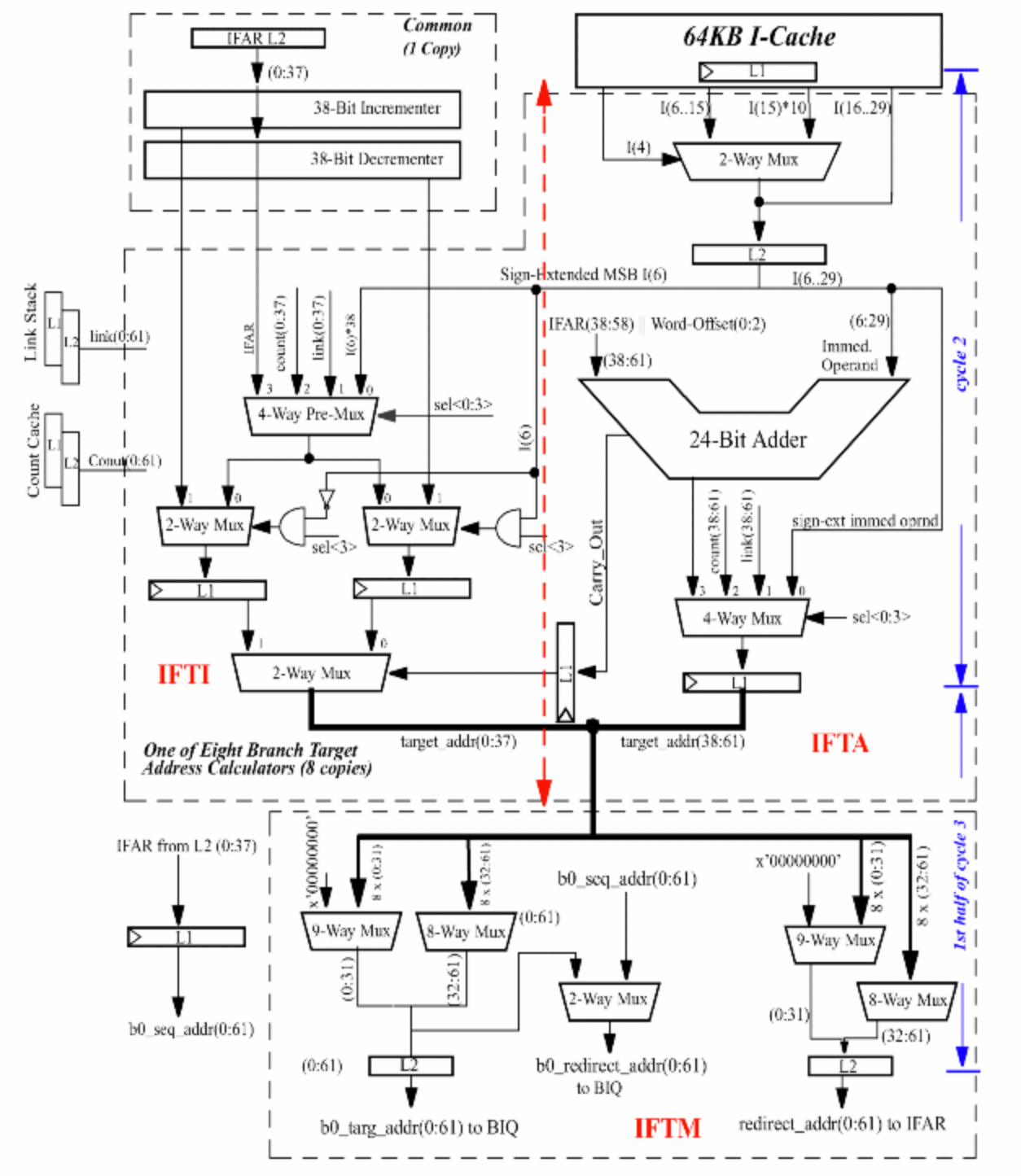

Figure 5: Block diagram of the branch calculator. The red arrowed line is the boundary of the dataflow of the high-order bits (0:37) (IFTI) and the dataflow of the low-order bits (38:61) (IFTA). There are eight unique copies of the adder and the associated mux in IFTA and IFTI for each instruction. The decrementer and incrementer are shared among the eight instructions. The Link Stack and Count Cache are register files outside the datastack. The IFTM macro contains various muxes for generating target addresses of the branch instruction. The whole branch calculator occupies cycle 2 and the first half of cycle 3 in Figure 4.

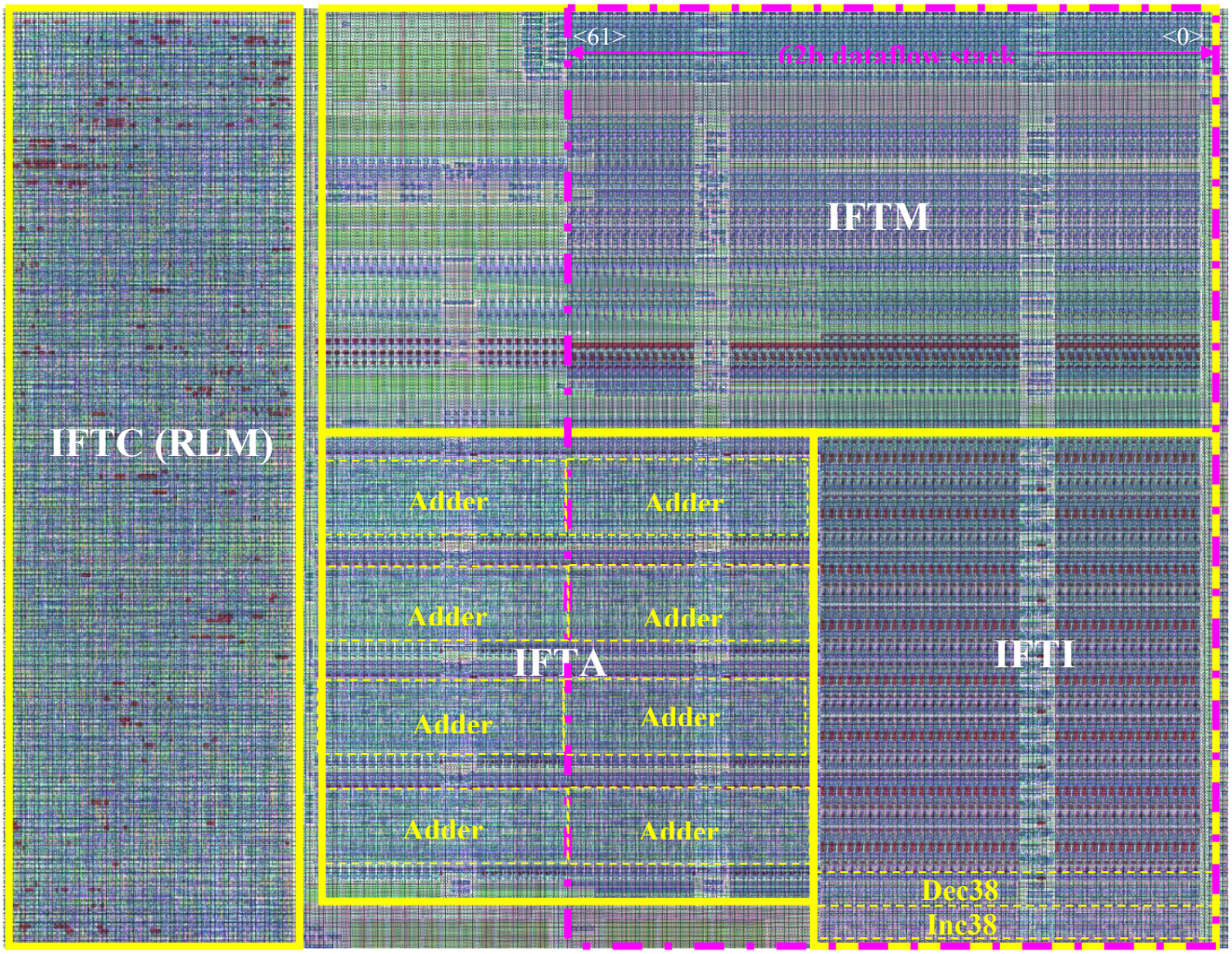

Figure 7: Chip micrograph of the IFT* macros. The adder, incrementer, and decrementer are semi-custom designs. The IFTC is the synthesized RLM that generates the select signals to drive the muxes in the IFTA, IFTI, and IFTM macros. The dash-dotted line oulines the 62 bit datastack. Bit 61 faces the cache SRAMs to reduce the wire length of the critical path.