An Investigation into the Design Costs of a Single Chip Multigauge Machine

Lawrence Snyder, Chyan Yang

Department of Computer Science FR-35
University of Washington
Seattle, WA 98195
Technical Report 86-06-01

June 2, 1986

This work was supported in part by ONR Contract N00014-85-K-0326 and DARPA MDA907-85-K0072

An Investigation into the Design Costs of a Single Chip Multigauge Machine*

Lawrence Snyder, Chyan Yang
Department of Computer Science
University of Washington

Abstract

Multigauge computers can operate either with their full datapath width or with the datapath split into separate narrower width machines. Such a concept has been previously shown to provide parallelism when the data values are small. This paper reports on the costs of gauge-shiftable computers. Specifically a single chip microprocessor, the 32-bit Quarter Horse machine, is redesigned to be gauge shiftable under software control to two 16-bit microprocessors. A complete accounting of the effects of gauge shifting on computer architecture is thus realized. Care is taken to identify the effects on the design of SIMD and MIMD executions. The general costs of gauge shifting for other implementation strategies are also identified.

1 Introduction

Multigauge parallel machines are parallel architectures whose processor elements can be split up to operate on smaller size data values; for example, a 32-bit machine might be divided into four 8-bit machines to work on character data. The chief benefit is that the normally serial processor elements are converted into parallel processors with the attendant speedup. Multigauge computation was argued to be fundamentally different from serial computation, and especially beneficial to an important class of algorithms called two-tier algorithms, algorithms with a massive amount of "low level" data processing on "small" data values coupled with "high level" processing [1]. Having previously identified substantial benefits for multigauge computation, we now consider its costs.

There are a variety of ways of assessing costs. We have selected a mixed strategy that assures both that we will be complete and precise and that the results will be generally applicable: We analyze how to convert a particular single chip sequential computer, the Quarter Horse microprocessor [2],

* Funded in part by ONR Contract N00014-85-K-0326 and DARPA MDA907-85-K0072.

to be a multigauge computer. Then for each modification we assess both its consequences for the Quarter Horse, and whether this modification is a general phenomenon or simply unique to this particular machine. In this way no architectural effect of multigauging will be missed, yet the general consequences of the concept will also be clear.

The Quarter Horse is a 32-bit microprocessor implemented at the University of Washington as a single custom chip in 3μ CMOS [2]. The name derives from the fact that it is a standard reduced instruction set architecture, a simple "workhorse" computer, whose original development was accomplished in only 90 days. The computer used here is a compact version of that original design, the Quarter Horse II (QII2). This machine has approximately 40% of the die free and was designed to be easily enhanced. Several extensions are planned. The importance of using this Quarter Horse II as the basis of a multigauge machine is this: *The absolute performance improvement of multigauging can be established by comparison with the base machine, and the relative improvement can be established by comparison with other extensions that utilize the available silicon differently.* The value of multigauging to processor element design can thus be determined by fair experimentation. Notice that this comparative performance analysis is the subject of a future paper to be written when all designs are completed and fabricated; this paper addresses the design requirements of multigauge computation and their impact on sequential processor architectures.

1.1 Multigauge Architecture Review

Recall [1] that a multigauge architecture utilizing its full B-bit datapath width is said to be executing in *wide gauge* or *wide track* mode. When the datapath is partitioned into k separate machines each with a b-bit wide datapath ($kb = B$) the machine is said to be in *narrow gauge* or *narrow track* mode. For this analysis, $B = 32, b = 16, k = 2$. The narrow gauge machines can be of two varieties: SIMD, in which they execute the same instruction stream, and MIMD in which they execute separate instruction streams. We will consider the impact of both cases here.

1.2 Quarter Horse Architecture Review

Recall [2] that the Quarter Horse is a 32-bit dual bus architecture with a reduced instruction set. Figure 1 shows a schematic of the datapath, Figure 2 shows a floor plan of the chip, and Figure 3 shows the instruction format.

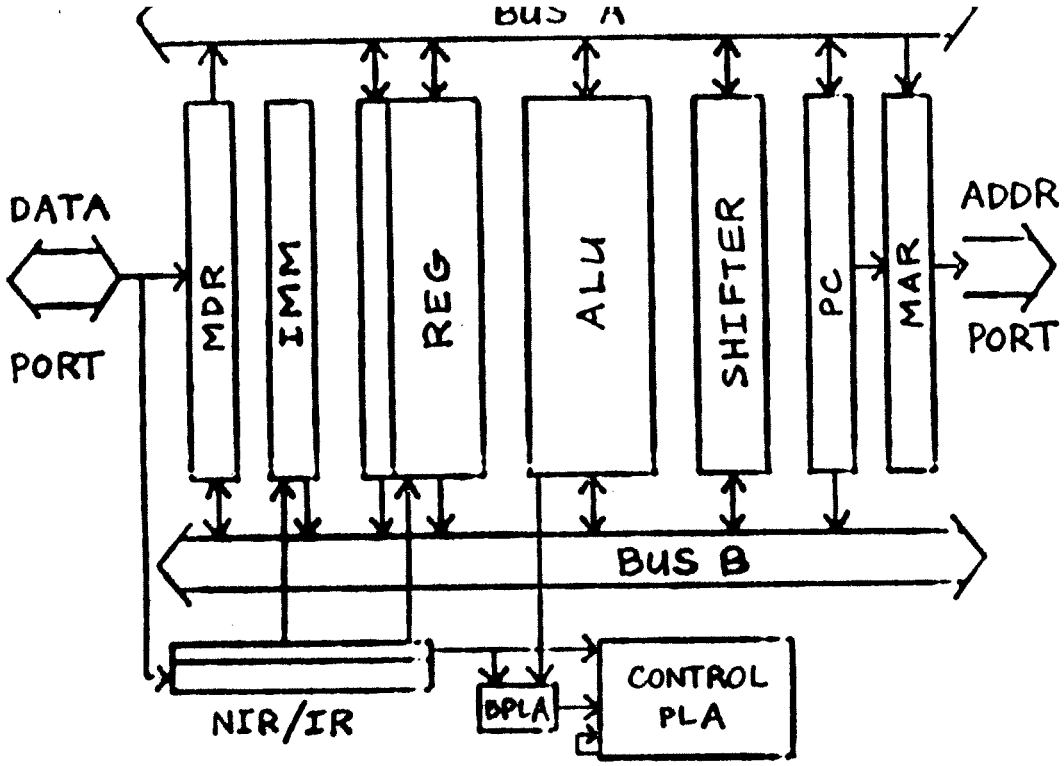


Figure 1: Datapath Schematic of Quarter Horse

Table 1 shows the instruction set. Several aspects of the architecture deserve special mention.

There is an address port through which all addresses to main memory pass and a data port through which all instruction and data values pass. Although the Quarter Horse II only uses 67 pins for external communication (including 3 reserved exclusively for testing), it will not be assumed here that there are sufficient pins for each narrow gauge machine to have 32 pins allocated to each port; the partitioning of the ports is a problem we will solve.

The Quarter Horse is microprogrammed using a finite-state PLA for control. Microinstructions execute at 75ns each; a typical macroinstruction requires six microinstructions, although some instructions are shorter and others, such as the *load character* and *store character*, are much longer. There are no external clocks in the datapath; all external clocking is used to drive the PLA which in turn controls the datapath. It is not possible for more than one sequence of control signals to be derived simultaneously from the PLA, so the various gauges of machines will require separate control.

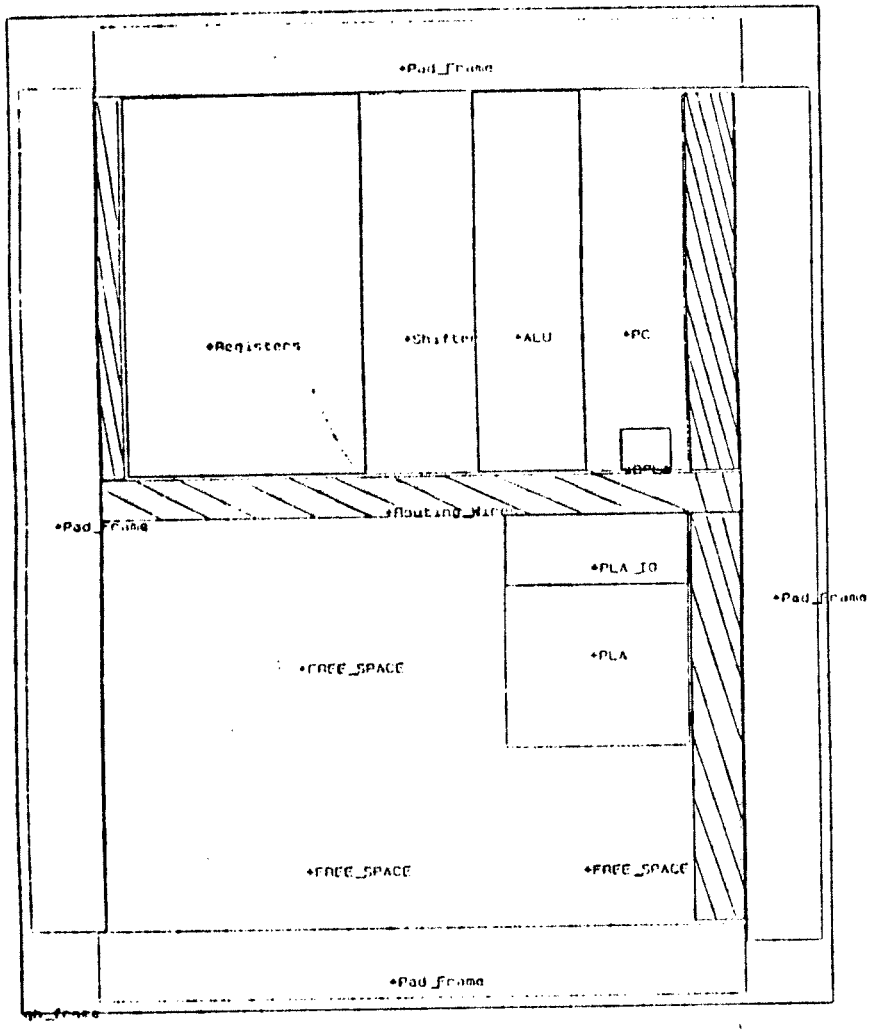


Figure 2: Floor Plan of Quarter Horse

OPcode	SCC	IMM	RD	RA	11 Immediate RB	
31	23	22	21	16	11	6 0

SCC: Set Condition Code
IMM: use immediate

RD: Destination Register
RA: Source Register A
RB: Source Register B

Figure 3: Instruction Format of Quarter Horse

OPCODE	INSTRUCTION	<Mnemonic>	EFFECT
00	ADD	ADD	R[D] <— R[A] + R[B]
01	Add with carry	ADDC	R[D] <— R[A] + R[B] + C
02	Subtract	SUB	R[D] <— R[A] - R[B]
03	Subtract with carry	SUBC	R[D] <— R[A] - R[B] - C
04	Subtract inverse	SUBI	R[D] <— R[B] - R[A]
05	SUBI with carry	SUBIC	R[D] <— R[B] - R[A] - C
10	Shift left logical	SLL	
11	Shift right logical	SRL	
12	Shift left arithmetic	SLA	
13	Shift right aritmetic	SRA	
14	Rotate left	RL	
15	Rotate right	RR	
20	Store	STO	M[ea] <— R[D]
21	Store character	STOC *	
22	Load	LD	R[D] <— M[ea]
23	Load character	LDC *	
24	Conditional jump	JMP	PC <— ea if cond
25	Call	CALL	R[D] <— PC * PC <— ea
28	Store relative	STOR	
29	Store character relative	STOCR *	
2A	Load relative	LDR	
2B	Load character relative	LDCR *	
2C	Conditional relative jump	JMPR	
2D	Call relative	CALLR	
30	Restore PSW	RPSW	
31	Save PSW	STPSW	R[D] <— PSW
80	bitwise CLEAR	CLR	R[D] <— 0
81	bitwise NOR	NOR	R[D] <— R[A] NOR R[B]
82	bitwise A AND B	NAAB	R[D] <— R[A] AND R[B]
83	bitwise NOT A	NOTA	R[D] <— R[A]
84	bitwise A AND B	AANB	R[D] <— R[A] AND R[B]
85	bitwise NOT B	NOTB	R[D] <— R[B]
86	bitwise XOR	XOR	R[D] <— R[A] XOR R[B]
87	bitwise NAND	NAND	R[D] <— R[A] NAND R[B]
88	bitwise AND	AND	R[D] <— R[A] AND R[B]
89	bitwise XNOR	XNOR	R[D] <— R[A] XNOR R[B]
8A	bitwise B	B	R[D] <— R[B]
8B	bitwise A OR B	NAOB	R[D] <— R[A] OR R[B]
8C	bitwise A	A	R[D] <— R[A]
8D	bitwise A OR B	AONB	R[D] <— R[A] OR R[B]
8E	bitwise OR	OR	R[D] <— R[A] OR R[B]
8F	bitwise SET	SET	R[D] <— 1

* The microcode for these instructions is given as an example of how extensions may be made to the basic instruction set.

Table 1. QII Instruction Set

2 Overview

In this section we analyze the transformations to the QII2 required to create a single chip multigauge machine design. For this discussion of the 32-bit processor, the WG machine refers to the whole 32-bit datapath, NG_1 and NG_2 refer to two datapath partitions [bit0, bit15] and [bit16, bit31], respectively.

2.1 Instruction Set

Although there is no necessity for the wide gauge and narrow gauge variants of a multigauge machine to execute the same instruction repertoire, there are benefits in doing so. First of all, code generation by the compiler may be unified with a common set of instructions. Second, the functional components implementing the instructions may be shared. For example, the carry chain can be used in segments (with added logic) by the narrow gauge machines. If the logic is not sharable then at least the design can generally be reused. Thus, the availability of the circuitry or the design motivates multiple use. Third, the control circuitry for the wide gauge machine *may be* usable for controlling the narrow gauge machine(s). This is likely in the SIMD case; also it is somewhat more likely with microcode than hardwired control and may justify using microcode over multiple copies of hardwired control. Again, even if the control logic itself is not reusable, the fact that the design can be reused favors common instruction sets.

Compelling as the argument may be for a common instruction set, there are arguments for different instruction sets. The primary justification is to overcome difficulties introduced by gauge shifting. For example, as described in section 2.3 and 2.4 special segment registers may be used to overcome the problems of narrow address width caused by partitioning the address pins among the narrow gauge machines. Manipulating the segment registers will require special instructions. As another example, it may be sufficient to provide the interrupt support instructions for only the wide gauge machine, since narrow gauge machines are not likely to do interrupt handling. Another problem involves the use of wide, B-bit, values in narrow gauge MIMD mode: The need for wide values is for such things as saving addresses on subroutine calls. If full registers are used, then instructions to manipulate the separate b-bit units are necessary; if only portions of registers are used then several must be employed causing instructions like *call* to operate differently, that is, they require different control logic in the different modes (see section 2.5.1).

The problem does not arise in SIMD mode.

Perhaps the most difficult problem with the instruction set is caused by the fact that the port through which the instructions are fetched from memory must be shared among narrow gauge MIMD machines. If it is time-multiplexed among the narrow gauge machines, then the same size and format instruction can be used in both wide and narrow gauge modes, but there is a potential bottleneck. Alternatively, if the port is partitioned, the bottleneck is removed but the instruction size of the narrow gauge machine is now a fraction of that of the wide gauge machine. This means that the instruction must either be formed with parts from multiple fetches or must be of a different size and format. This severely complicates the problem of making the instruction sets the same. The problem does not arise in SIMD mode.

For the multigauge Quarter Horse, we have chosen to use a 16-bit instruction for the narrow gauge machines. This imposes the following modifictions to the base instruction set.

1. The register set is reduced from 32 to 16 to save an address bit in each operand.

2. Condition codes are automatically set for arithmetic instructions rather than being optionally set, removing a tag bit.

3. The instructions are two operands, saving an operand addressing field.

4. The immediate data is 16 bits and is stored in the next half word.

5. The interrupt instructions have been removed; interrupts automatically return to wide gauge mode.

Implementing the remaining instructions, plus the mode shift and segment registers manipulation instructions can be done exactly in the seven bits of the limited opcode; the instruction format is shown in Figure 4. Clearly, the control circuitry cannot be shared!

2.2 Control

There are three possible control strategies: hardwired logic, true microprogramming, and PLA. (Notice that we use PLA to refer to a programmable logic array with feedback state bits, i.e., a PLA is a finite state machine.)

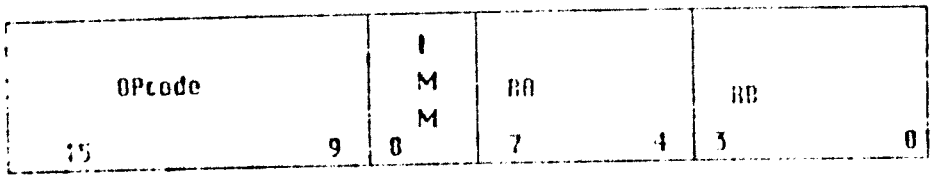


Figure 4: Narrow Gauge Quarter Horse Instruction Format

There is little to distinguish the three alternatives for use in gauge shifting that does not also apply to a single sequential machine. The main considerations seem to be area and power, since multiple copies of the control will be required. For MIMD multigauge computation, $k+1$ control units will be required; for SIMD multigauge machine 2 control units, one for wide track and one to be shared by the narrow track machines will be required. A careful design based on a common instruction set could possibly reduce the SIMD case to a single control unit; a multiported microcode for narrow gauge computation could *conceivably* reduce the MIMD case to 2 control units, one for each gauge.

In the multigauge Quarter Horse design three PLA control units are used for the MIMD mode design, and two for the SIMD mode design. PLAs were chosen simply because that was what was used for the original Quarter Horse. A similar number of branch PLAs are also required since their outputs are control inputs.

Because control logic is such a significant part of the multigauge design, the area utilization is of some importance. The size of the wide gauge PLA is

$$1782\mu \times 2162\mu = 3.85mm^2$$

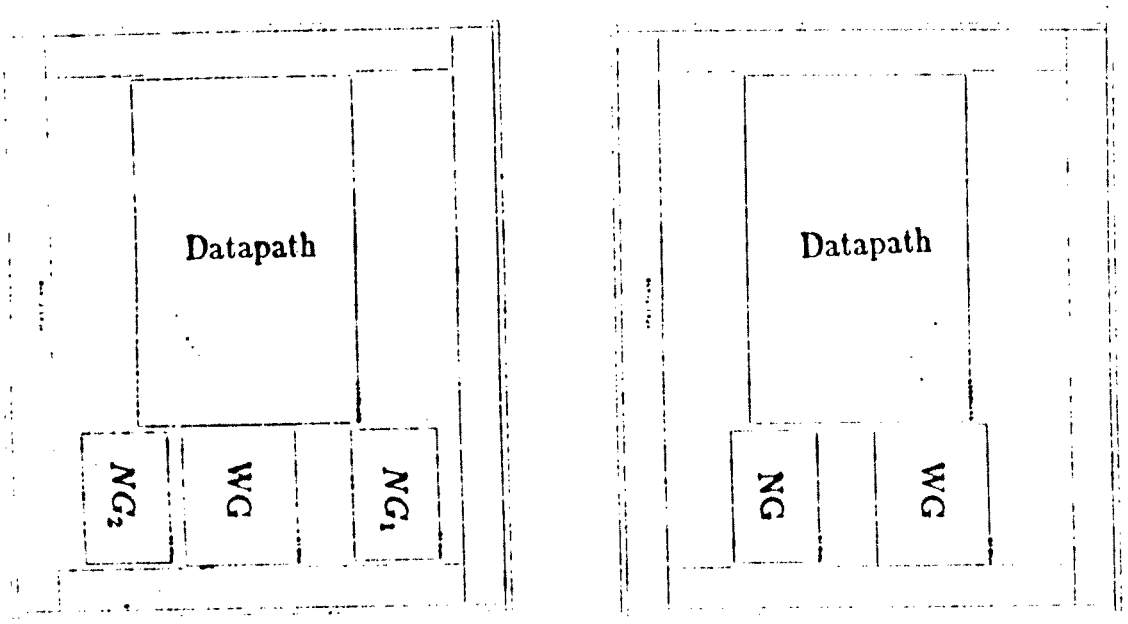
while the size of narrow gauge PLA is estimated to be

$$1351\mu \times 2024\mu = 2.73mm^2.$$

These components are sufficiently large and bulky that they complicate the floor plan considerably. Figures 5-1 and 5-2 show the organization for the MIMD and SIMD cases.

2.3 Memory Addressing

When considering the memory addressing of a multigauge machine, we encounter a dilemma between inadequate addressability and insufficient silicon



5-1 5-2

Figure 5: MIMD and SIMD floorplans

area. If the silicon space is considered to be the scarce resource, we can split a 32-bit wide gauge PC (program counter and memory address register) into two 16-bit narrow gauge PCs for supporting MIMD. Then, each narrow gauge processor will have a smaller address space and the addressed word is 16 bits shorter. In other words, the wide gauge PC has an address space of $2^{32} \times 32$ bits while the two narrow gauge processors together have $2 \times 2^{16} \times 16$ bits address space. Thus partitioning the PC prevents the narrow gauge machines from referencing the wide gauge machine's whole address space. In MIMD mode, since we would like the narrow gauge machines to operate independently for operations such as increment, interrupt and address generation, each narrow gauge processor should have its own counters and latches. Therefore, we need an extra PC for the narrow gauge processors and this extra PC should be of the same size as the 32-bit one.

The two PCs can be time-multiplexed through the same address port, but this will cause a potential performance bottleneck. Having no extra pins for the memory address port, seems to imply in the MIMD case that we must split the 32-bit PC into two 16-bit fields. In order to have a similar addressing capability between wide gauge mode and narrow gauge mode, we have to have segment registers in the memory module and special instructions for handling them. A similar argument is applicable to SIMD mode.

2.4 Memory Interface

As a consequence of the discussion on memory addressing (section 2.3), a single chip multigauge machine requires that its memory have compatible support. The memory interface includes at least one pin to receive a signal from the processor indicating which mode, wide gauge or narrow gauge, is currently operating. Another pin is required to distinguish a regular memory access from a request of load/store segment register. When operating in narrow gauge mode, special instructions for handling a segment register could be issued, e.g. loading(storing) a value from(to) a segment register. Whether to separate the program into two partitions in memory, instruction and data, is another research topic in multigauge machine performance. To simplify our discussion, we do not separate them here. Consequently, we can have a segment register for each narrow gauge machine which is used for accessing both instructions and data. These two segment registers in the memory module are disabled when the machine is operated in wide gauge mode. Separate address registers for each narrow gauge machine are also required.

The memory space of the wide gauge machine is divided into two parts and each narrow gauge machine addresses one of them. During narrow gauge mode, each narrow gauge machine sends a 16-bit PC value to the memory, and the memory can be viewed as blocks of 64K (2^{16}) words. For each narrow gauge machine, the address register is formed by the concatenation of the segment register value and the value received from the PC. When executing the *load segment register* instruction, the PC value received should be routed to the specified segment register. Program locality will guarantee infrequent updates of segment registers.

2.5 Datapath

The datapath is the real "workhorse" of the processor since it is the only place information is manipulated. In this section we describe what we have to do to make a datapath gauge shiftable. In the following discussion, two buses are overlayed horizontally on the datapath while all datapath control lines and internal power lines go through it vertically. This arrangement of the datapath has been frequently adopted by most recent processor designs [3, 4, 5]. Design issues will be identified and then design strategies will be proposed. The gauge switch, G, is a slice of transmission gates or pass transistors inserted between bit15 and bit16 and controlled by the gauge shifting instruction. In general, the area expansion due to insertion of transmission

gates is significant for most processors' datapath.

2.5.1 Registers

The bit slice in a register is independent of its position, i.e., any i^{th} bit slice has nothing to do with its neighbors. Therefore, a register requires less effort to split into two than other components. We discuss the modifications required for SIMD mode first, and then for MIMD mode.

The register array requires no modification for SIMD mode. As the narrow gauge instruction can address only 16 registers (section 2.1), half of the registers are invisible to narrow gauge machines. This can be accomplished by setting the most significant address bit, an input to the register decoder, to zero. No other modifications are required for the registers to support SIMD multigauge computation. With a moderate cost of having an extra decoder, both narrow gauge machines in SIMD mode can use all the 32 registers. Then, when a branch instruction is executed, the memory address could be formed by concatenating two narrow gauge registers. In fact, registers appear logically a 32-bit capacity to the PC and the memory address register. Both narrow gauge machines may set their condition codes asynchronously; a delayed no-op mechanism may resolve them gracefully.

Recall [6] that for a regular register design, a word line runs through each bit slice. Therefore, a register needs some muxes and pass transistors to partition it for independent operation in MIMD mode. To be specific, in MIMD mode, the wide gauge register array needs pass transistors to connect vertical word lines between bit15 and bit16. In addition, a column of transmission gates should be inserted between word15 and word16 for isolating the two narrow gauge machines. Moreover, the register array needs pass transistors connecting bus and data lines. The result of these modifications is two narrow gauge machines split the wide gauge register array into 4 parts and each machine uses only a quarter of it: NG_1 uses [bit0, bit15] and [word16, word31], NG_2 uses [bit16, bit31] and [word0, word31]. For MIMD mode, besides setting the most significant bit to zero like the SIMD mode does, the register decoder should have a column of transmission gates inserted between word15 and word16 for isolating the two narrow gauge machines.

Alternatively, with moderate costs, we can have an extra decoder to support the MIMD mode without introducing the transmission gates inserted between word15 and word16. All 32 registers are available to each narrow gauge machine. Each narrow gauge machine can use 16 registers for regular operations and the other 16 registers for internal storage such as values

of memory segment registers. With as little as one micro-operation cycle we can efficiently achieve memory interface support. Two decoders may or may not stack on the same side of register depending on the specific design. SIMD mode doesn't require additional I/O circuitry but MIMD may require separate I/O circuitry. If a four-ported register file is available, all the modificaions discussed above could be greatly simplified.

Having discussed the modificaions required for a general register design, we now turn to the case study chip, the QH2. The insertion of G into QH2 should be implemented by a row of transmission gates since we are using CMOS and pass transistors are not adequate for conducting logic-1. The current QH register array has vertical polysilicon word lines which are too dense to insert transmission gates into for gauge shifting control. The insertion of the transmission gates also makes the register array irregular, unless the size of the entire array is doubled. If we use two decoders to support MIMD mode, these two decoders should be on the opposite sides of the register array since we cannot stack up two decoders on the same side of the array. The power net (Vdd and GND) for the register file does not require much surgery. If we do not want to insert the costly transmission gates between bit15 and bit16, i.e., the gauge switch, we might physically cut off all vertical control lines. Then, in the wide gauge operating mode, we have to connect each PLA datapath control line to both sides of datapath, assuming the open space between bit15 and bit16 due to the cut-off is not too large.

2.5.2 Shifter

A barrel shifter can shift a datapath word left/right an arbitrary number of bits and is found in many datapath implementation [2, 3, 4, 5]. Figure 6 shows a schematic of a 4 × 4 barrel shifter which could easily be found in a dual bus microprocessor [2, 3, 4]. Connected to shifter I/O lines bit_R and bit_L through a special designed I/O circuitry, data buses are horizontally overlayed on the top of the shifter and are not shown in the figure.

The shifter is basically a two dimensional logic, i.e., it is diagonal in nature. This can be demonstrated by a simple example of its operation: a right shift (logical) by two positions. In Figure 6, when shifter is ready to shift and the *shift2* line is active (at any time only one *shift constant* line could be activated), the data to be shifted are presented on bit_R lines. The values input as bit_{R2} and bit_{R3} are output as bit_{L0} and bit_{L1}, respectively. When the shifter is operating a left shift the data traverse in the opposite

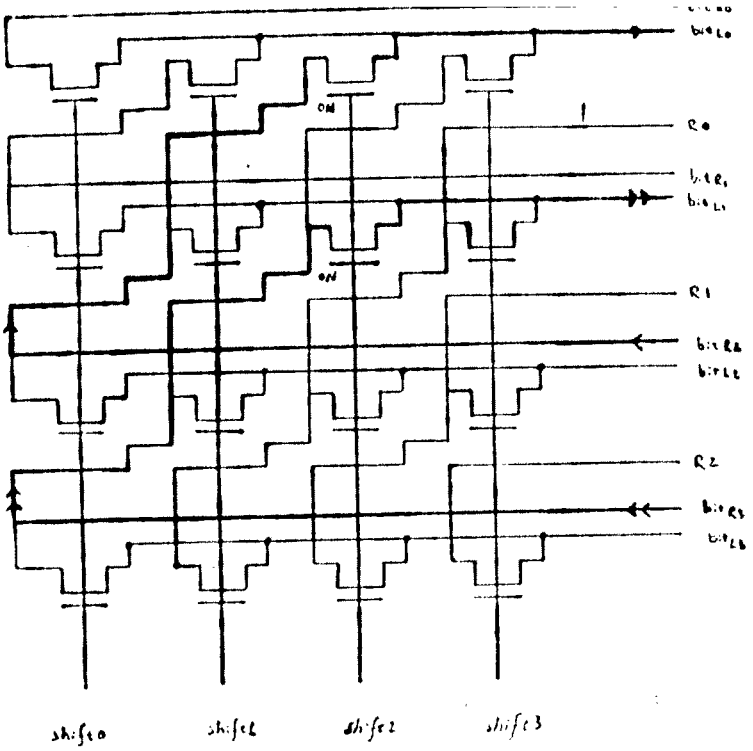


Figure 6: Barrel Shifter Schematic

direction as a right shift. Notice that with moderate overhead a barrel shifter can support rotate and sign extension for arithmetic shift [6].

No additional decoder is needed for SIMD mode but a separate one for MIMD mode is required. Insertion of a slice of transmission gates will at least double the shifter size, which is too costly. Merely a cut-off between bit15 and bit16, imitating the strategy used in register file, will not make it a wide gauge shifter nor two narrow gauge shifters. One dimensional slicing is not sufficient for splitting a "two dimensional" device and preserving its functionality. Because of the tremendous costs involved in splitting a barrel shifter, it is not worth trying to provide a shifter for the narrow gauge machines. This decision will reduce the PLA_{NG} size although it is not significant.

The shifter used in QH is a barrel shifter based on the RISC barrel shifter [3]. In short, to transform the QH into multigauge suggests not directly supporting shifting instructions.

2.5.3 ALU

The two major parts in the ALU are the ALU-flags and the ALU-array. The ALU-array, either implemented by a carry chain or carry look-ahead logic, contains 32-bit slices. In general, a carry look-ahead ALU is not regular

and takes a lot silicon area in VLSI design. Therefore most processors are designed with carry chain ALUs [4, 5]. ALU-flags collectively refers to the circuitry for computing flags and the registers holding the values.

We now examine the case of the Quarter Horse design. Without supporting interrupts for narrow gauge mode (see section 2.5.4), we can simplify the modifications of the ALU. If we cut off all control lines as that in register case, the gauge switch only needs a "carry" connection and a moderate expansion for the overflow detection of the narrow gauge mode. The condition codes are automatically set in narrow gauge mode depending on the result of the arithmetic operation. We may duplicate the flag logic for supporting the conditional branch instructions. However, the insertion of the flag logic will increase the vertical dimension of the datapath and lose the pitch matching we have now.

2.5.4 Program Counter

If it were not for pad limitations, we can simply add an extra PC for MIMD mode. A single PC for instruction addressing is adequate for SIMD mode. For both SIMD and MIMD modes, there may require a duplicate memory address register. Therefore, if we do not have enough silicon to have the luxury for having an extra PC, we have to split a 32-bit PC into 2 16-bit PCs. A counter is a unidirectional device, its carry may propagate from bit0 to bit31. In PC, fortunately, only the "carry" line requires a transmission gate to achieve the gauge switching property; this cost is moderate. The implications of shorter PCs, however, are important and have been discussed in sections 2.3 and 2.4.

The analysis of the QH2 implementation revealed that we do not have enough spare pads. Therefore, we can not have duplicated PCs of 32 bits due to pin limitations, although for easy addressability and compatible computation power we need one for each narrow gauge (see section 2.3). In other words, the MIMD mode needs separate program counters for NG_1 and NG_2; each of them can have only 16 bits. We can only partition the current counter into two for narrow gauge machines by gauge switch, which is too costly. The current PC can count up to 28-bit since 4-bit slices are used for flags traffic to the PSW. In other words, for the MIMD mode we need to count up to 32-bit and remove the PSW since we need not support the interrupt mechanism for both wide gauge and narrow gauge cases. This reduces PC size in the horizontal direction. In SIMD mode, we need only one counter to address instructions, i.e., all 32 bits can be used. Like the

control path in the register file, the vertical control lines can be cut off at the boarder of bit15 and bit16. By doing so, separate control paths should be provided for both narrow gauge datapaths.

2.5.5 Other Components

Generally, each bit slice of either an instruction register (IR) or a memory data register (MDR) is basically independent of its position, i.e., it is "bit-sliceable". Sign extension logic is tied to the MDR; therefore, it should go with the MDR.

QH2 has one level of instruction prefetch and the prefetched instruction is latched in the *next instruction register* (NIR). QH2 squeezed its IR, NIR and MDR into customized tri-state pads. The strategy was to take advantage of unused silicon real estate. However, in order to have these three components reside in a tri-state pad, we are forced to use both wider sides of the pad frame for the instruction/data port. Therefore, IR, NIR and MDR are distributed along the opposite borders, and two separate control buses are required. Also, the sign extension for an immediate addressing was implemented within the pad corresponding to the most significant bit of immediate value. If it were not for the sign extension support for both the wide gauge and narrow gauge modes, splitting up these components into narrow gauge mode could be easily achieved. Since NIR, IR and MDR are tightly coupled into the pad no free space can be found to support the immediate addressing in narrow gauge mode. We now prefer to carve these three components out of the tri-state pad and let them occupy the inner silicon area leaving the space of pad frame for testability design, if any.

3 Tweaking the QH Floor Plan

To simplify our estimation of the floor plan, let us assume that the datapath with its power net still remains the same size as QH2, i.e. $5448\mu \times 3477\mu$. The insertion costs of the gauge switch are assumed negligible, i.e., we are tweaking the QH floor plan to make it gauge shiftable only based on the sizes of PLAs and the size of current datapath. We do so to illustrate that even in the best case we find it difficut to make it gauge shiftable.

The sizes of PLA_{WG} and PLA_{NG} are $1782\mu \times 2162\mu$ and $1351\mu \times 2024\mu$ respectively (see section 3.2). The branch PLAs are omitted in the floor plan since they can be located underneath the metal-2 ground strip at bottom of

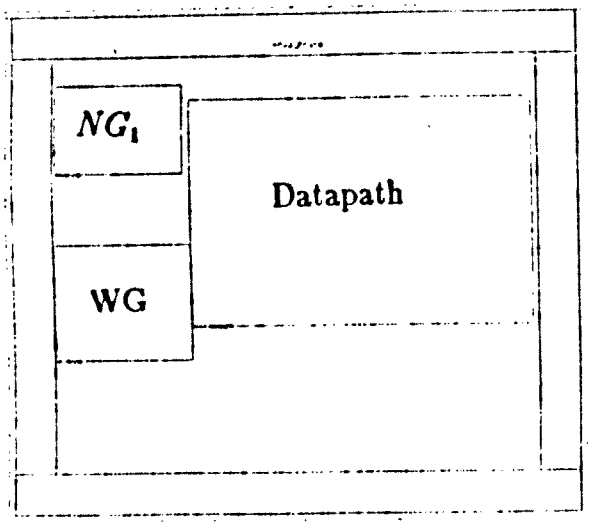
datapath. QH2 uses an 84-pin pad frame with die size of $7900\mu \times 9200\mu$. If the pads are excluded, the size of inner silicon area is $6420\mu \times 7720\mu$.

The SIMD and MIMD cases are essentially the same; we treat the SIMD case. If we place the datapath along the longer dimension, the 7720μ side, and if the PLAs are placed at its left with the width of $PLA_{WG} = 2162\mu$, we get a total of 7610μ horizontally (see Figure 7-1). This means that we have only 110μ left for routing and other logic, such as IR and MDR, which is not sufficient. If we rotate both PLAs 90 degrees, the situation is improved by increasing a 380μ channel which is not enough for sharing the routing of PLA_{WG} and PLA_{NG} output lines to both sides, top and bottom, of datapath (see Fig. 7-2). Another approach is to place the PLAs at the bottom side of the datapath, which will give us about 900μ for routing. A wide gauge datapath alone, will take up all this space if we want its output lines to go onto both sides of the datapath (see Fig. 7-3). Alternatively, we may place the datapath as QH2 did, i.e., the datapath is located along the shorter side of the frame (see Fig. 7-4). By doing so, the open space available, the space for routing, is not enough for routing two PLAs outputs to both sides of datapath, and this does not count the necessary data/address routing. Note that our analysis is a first order approximation since it does not take into account of power net, components other than datapath and PLAs, and routing at chip assembly level. Putting all the components together will make the situation worse. In these trial floorplans, we have seen difficulties finding enough room to put the PLA_{WG} and PLA_{NG} onto the same chip area as QH2 used, even though other detailed routing wires are not estimated.

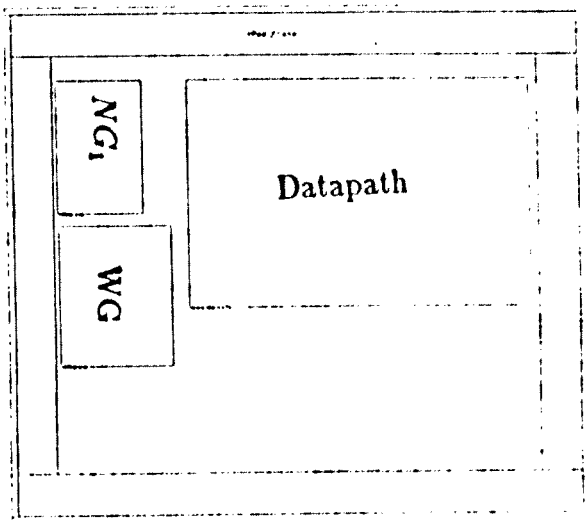
4 Conclusion

We have reviewed the concept of multigauge architecture and the Quarter Horse architecture. Careful considerations of single chip multigauge design has been discussed in section 2, for both the general case and for the specific Quarter Horse case. The difficulties of splitting a case study chip, the Quarter Horse II, have been illustrated. With careful design based on a common instruction set, it could be possible to reduce the SIMD case to a single control unit; hence it is easy to make a single chip multigauge machine for SIMD. MIMD mode, however, is much harder.

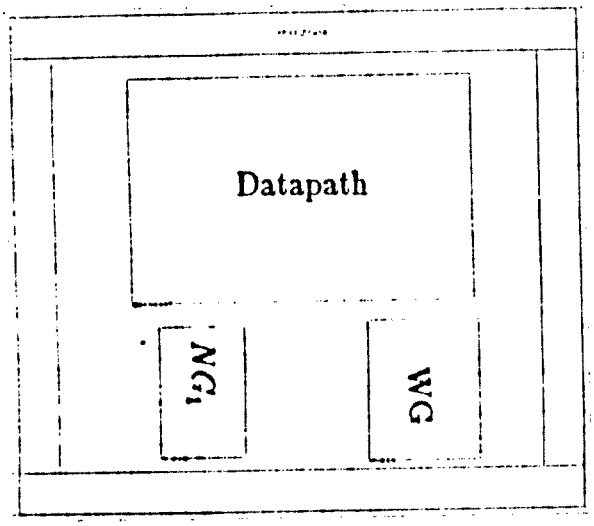
From the analysis, we also confidently believe that it is very difficult to do gauge shifting narrower than 16-bit due to multiple copies of control PLAs and wiring complexity. In other words, the SIMD *threshold* is really



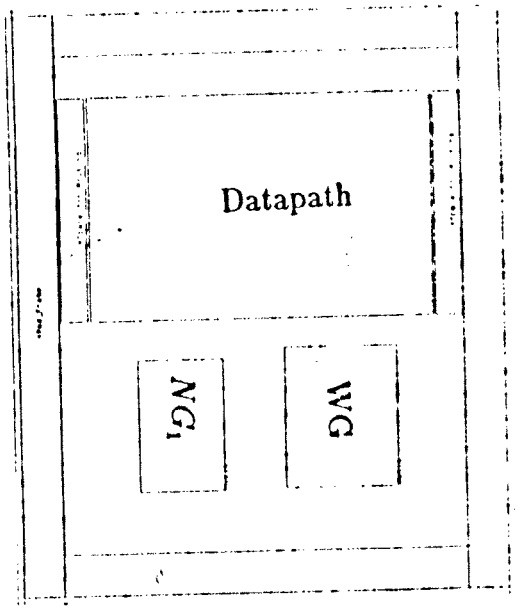
7-1



7-2



7-3



7-4

Figure 7: SIMD Floor Plan for QH

16, not 8 as was suggested in the previous paper [1]. Memory addressing is a tough problem but can be solved by the mechanisms we have proposed, e.g., memory segment registers and special instructions for handling them.

Although the datapath is the "heart" of a processor, it is not the whole problem one has to solve. Other significant costs are also of concern such as the insertion of the gauge switch and addressing complexity. An open question remains for further research: Is splitting a single chip processor for SIMD mode really a win?

References

1. L. Snyder,
"An Inquiry into the benefits of Multigauge Parallel Computation," *IEEE ICPP 1985*, pp. 488-492.

2. S Ho, B. Jinks, T. Knight, J. Schaad, L. Snyder, A. Tyagi, and C. Yang,
"The Quarter Horse: A Case Study in Rapid Prototyping of a 32-bit Microprocessor Chip," *IEEE ICCD: VLSI* 1985, pp. 161-166.

3. R. W. Sherburne, Jr., M. G.H. Katevenis, D. A. Patterson and C. H. Séquin,
"Datapath Design For RISC," *Proceedings of the 1982 Conference on Advanced Research in VLSI,* MIT, pp. 53-62.

4. C. Mead and L. Conway,
Introduction to VLSI Systems, Addison-Wesley, Reading, Massachussets, 1980.

5. L. A. Glasser and D. W. Dobberpuhl,
The Design and Analysis of VLSI Circuits, Addison-Wesley, Reading, Massachussets, 1985.

6. S Ho, B. Jinks, T. Knight, J. Schaad, L. Snyder, A. Tyagi, and C. Yang,
"The Architecture of the Quarter Horse," *Technical Report,* University of Washington, 1985.