An Architecture and Layout for Register-Transfer Level IC Design

F. Mavaddat

Department of Computer Science

&

VLSI Group

University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

Research Report CS-84-18
September 1984

An Architecture and Layout for Register-Transfer Level IC Design

Farhad Mavaddat

Dept. of Computer Science
&
VLSI Group
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1
(519) 885-1211 x3400

ABSTRACT

A new processor architecture and its silicon layout are explained. This architecture is used to describe an IC design methodology based on Register-Transfer level specification concepts.

Salient features of the proposed architecture are the mapping of multi-length logical registers into a physically homogeneous data path and the efficient handling of arrayed data.

This architecture permits in-circuit testing of a design under friendly test and debug conditions, prior to its freeze in silicon.

An Architecture and Layout for Register-Transfer Level IC Design

Farhad Mavaddat

Dept. of Computer Science
&
VLSI Group
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1
(519) 885-1211 x3400

1. Motivation

The VLSI implementation of Register-Transfer based designs has enjoyed considerable attention in the past (Hart82, Tsen82, Mead79, Joha79, Sisk82, Ance83). Most of the layouts proposed employ a register file and ALU combination to realize the data path and a perpendicular set of signals to effect control. This is an exceptionally efficient layout in which placement and routing are highly optimized. This optimality is one of the main reasons for the successful design of powerful microprocessors (Patt82, Burk83). Layout efficiency can be seriously hampered when certain designs need to deviate from this basic structure. The following are some of the motives for deviation:

1) Array Handling: To handle arrays, certain applications may need to index through a group of consecutive registers. ROM-based microcode prevents address modification as a way of stepping through consecutive addresses. Placing index registers in the data path, lead to address calculation inefficiencies and communication problems, and spoils the routing efficiency inherent in the structure.

2) Variable Length Registers: The use of multiple length registers leads to non-homogeneous register layout or to registers having unused parts. Both possibilities waste silicon area.

3) Immediate Operands: Orthogonality of control and data lines complicates the task of transferring literals from the program area to the data path.

4) Input/Output: Some form of interaction with the environment is needed.

While the third and the fourth requirements have received some attention in the past (Joha79, Ance83,Mead79), the first two seem to be open to design suggestions. The remainder of this report discusses a modified organization

of the conventional layout which satisfies these four requirements within the framework of a coherent design.

2. Organization and layout

To accommodate these requirements within the format of a regular and orthogonal design, a processor architecture and layout is proposed which preserves the layout efficiency in terms of regularity and inter-module communication, while fulfilling all of the stated requirements.

The proposed architecture consists of three modules, as shown in Figure 1. They are the microcode ROM, the control unit, and the data path. In silicon, the adjacent modules are separated by a well-defined channel. This facilitates the efficient interconnection of modules using channel routing algorithms.

2.1. Microinstruction Format and ROM Unit

Microinstructions have several fields representing control functions in either coded or decoded form. The mask pattern and the branch address share the same field. This sharing is possible because the number of microinstructions in the ROM is small and branch instructions never need masking. All other microorders have their unique fields within the microinstruction.

Control functions are coded if the number of instructions in the control program is relatively large. The savings in coding of small programs may be offset by the increased silicon area and propagation delay associated with a decoder.

2.2. The Control Unit

The control unit is responsible for fetching the microcode from ROM and generating control signals for the data path. It has several parts, each of which has a regular, horizontally-repetitive cell structure. This simplifies the custom design of applications ICs.

2.2.1 The Address part

The address part receives the address field of the microinstruction and generates a signal which selects the register sourcing the operand or receiving the result in the data path. While the input to this unit may or may not be encoded, the output is always fully decoded (see Figure 2).

In order to realize the array-indexing capability, a horizontal shift register is placed in the address unit, in which one bit corresponds to every data register in the data path. We shall refer to this register as the address register.

The output of the address part is derived either from its current input or from the contents of the address register. To start array processing, the starting address of the array is strobed into the address register from the address field of the microinstruction. References to array elements are made by selecting the address register instead of the direct address input. Indexing through successive array elements is performed by shifting the contents of the address register to the left or right.

2.2.2 Condition Tester

The condition test unit compares a field in the microinstruction to selected bits in the data path. A successful comparison results in the transmission of a "success" signal to the operation decoding unit. This signal can result in the execution of a program branch.

Test points inside the data path are reached through connectors placed between neighbouring registers. To have an efficient data path layout, the test points are placed as the most significant bits of the physical register, whenever possible. This can be easily achieved for the sign bits.

2.2.3 Operation Decoding Part

This unit receives the Op-Code from the microinstruction and sends control signals to the ALU and to other Control Unit parts. It also receives the result of the test operation from the Condition Tester.

Control signals from the decoder which are internal to the control unit allow the loading of the microinstruction counter with the branch address, and the control of the address register inside the address part.

Except for the part which transmits the ALU and shifter codes, this unit has a fixed structure and is copied from the cell library into the design layout. Designs without array indexing require simpler Operation Decoding Parts.

The microinstruction counter has a conventional design.

2.3. The Data Path

The data path consists of a number of registers, a mask/slice unit, a barrel shifter, the ALU, and an accumulator register as shown in Figure 3.

2.3.1. The Register Part

The physical registers of the register part are logically allocated among the register requirements of the application. This allocation is influenced by many factors and some inefficiency may be tolerated in order to improve speed or program flexibility. The number of physical registers required, and their width, depend on the allocation process and will be decided by the designer or the design package after an analysis of the application requirements.

2.3.2. The Mask/Slice Unit

The mask/slice unit is multiplexed between the mask and slice operations. The operation of the mask/slice unit is bi-directional. In the mask mode it performs an "and" function between the value placed on the path and the mask pattern received from the control unit. Under the control of another signal an all "1" value can be forced on the data path which, when masked properly, creates any desired pattern on the data path. This is the mechanism for transferring literals from the ROM area into the data path.

In the slice mode, only selected bits of the data path are enabled. The enabled bit slices correspond to the set bits in the mask pattern. This mechanism is used to pack multiple-length logical registers into more efficient and homogeneous physical registers.

Some user-defined applications may not require any mask or slice operations. Also, it may prove to be more efficient to hard-wire some literals onto the data path rather than transferring them from the ROM area.

2.3.3 The Barrel Shifter

The barrel shifter is used to perform the regular user-defined shifts as well as those needed to operate on data allocated to different slices of the data path.

Only shift values explicitly used in the applications or those found necessary because of non-aligned slice allocations will be implemented. Some applications may not need a shift unit at all, or a simple shifter may prove sufficient.

2.3.4. The ALU Slice

Since the ALU slice is tailored to the application, only the required functions are implemented. All ALU bit slices are homogeneous and no attempt is made to streamline the ALU function on a per bit or per field basis.

The data-paths described here, while custom designed, are highly layout efficient.

3. Architecture of Input and Output Ports.

Input/Output ports are connections to the data path from outside the system. These connections can be made either directly to the data path under the control of the microprogram, or through an intermediate register port.

These register ports are read and written from the data path, and from the outside, allowing the implementation of more sophisticated input/output protocols.

To establish some handshaking with the outside, a flag (sense) register is associated with every data port. These flag (sense) registers are similar to the data ports in

their function and features. It is the duty of the producer program to set the flag after writing into the data port. The receiver program must clear the flag after reading the values. To speed up communication, the write signal could set the flag while the read signal could clear it.

In most cases data ports will be unidirectional. On one side of the port it will be possible to read or write, but not both. The complementary function will be possible on the opposite side of the port. When bidirectional data ports are necessary, the application program must observe a higher level protocol.

4. Design Verification and Test

One of the advantages of working with a parametric architecture is the possibility of building a maximum system using off-the-shelf ICs for test and verification purposes. These verification engines allow the in-circuit testing of a design prior to freezing it in silicon.

We have implemented one such engine and are currently using it to gain some design experience. The engine, implemented in F-TTL, has an eight-bit physical data path with twelve registers, a barrel shifter, a powerful ALU, and 256 48-bit words of writable control store.

To perform in-circuit emulation of application ICs, we are using a probe with a large number of grips on the engine

side. Each grip is connected to a pin of a plug on the application side. These are connected to the I/O points on the registers in a different configuration for each new application.

To gain better insight into the design of real ICs, we are concentrating on the design of peripheral chips for 8085 based microprocessor systems. The ability to verify a design under real circuit conditions, coupled with the use of 8085 in-circuit emulators, has created new opportunities for testing ideas and designs. Design errors can be found easily and quickly, relative to the effort needed using software simulation methods.

5. References

Ance83, Anceau, F., CAPRI: A design methodology and a Silicon Compiler for VLSI Specified by Algorithms, Third Caltech Conference on Very Large Scale Integration, Edit. R. Bryant, California Institute of Technology, Computer Science Press, 1983, pp 15-31.

Burk83, Burkhart, K. P., Forsyth, M. A., Hammer, M. E., Tanksalvala, D. F., An 18-MHz, 32-Bit VLSI Microprocessor, Hewlett-Packard Journal, August 1983, pp 7-10.

Hart82, Hartenstein, R. W., Basics of Structured Design Methodologies: Data Path and Finite State Machines, in Design Methodologies for VLSI Circuits, Sijthoff & Noordhoff International Publishers, The Netherlands, 1982.

Joha79, Johannsen,D., Bristle Blocks: A Silicon Compiler Proceedings, 16th. Design Automation Conference, July, 1979.

Mead79, Mead, C., Conway, L., Introduction to VLSI Systems, Addison-Wesley, 1979.

Patt82, Patterson, D. A., Sequin, C. H., A VLSI RISC, Computer, September 1982, pp 8-20.

Sisk82, Siskind, J. M., Southard, J. R., Crouch, K. W., Generating Custom High Performance VLSI Designs from Succinct Algorithmic Descriptions, Proceedings, 1982 Conference on Advanced Research in VLSI, M.I.T., 1982, pp 28-39.

Tsen82 Tseng, C. J., Siewiorek, D. P., The Modeling and Synthesis of Bus Systems, DRC-18-24-82, Design Research Center, Carnegie-Mellon University, 1982 Pittsburgh, PA., 15213.

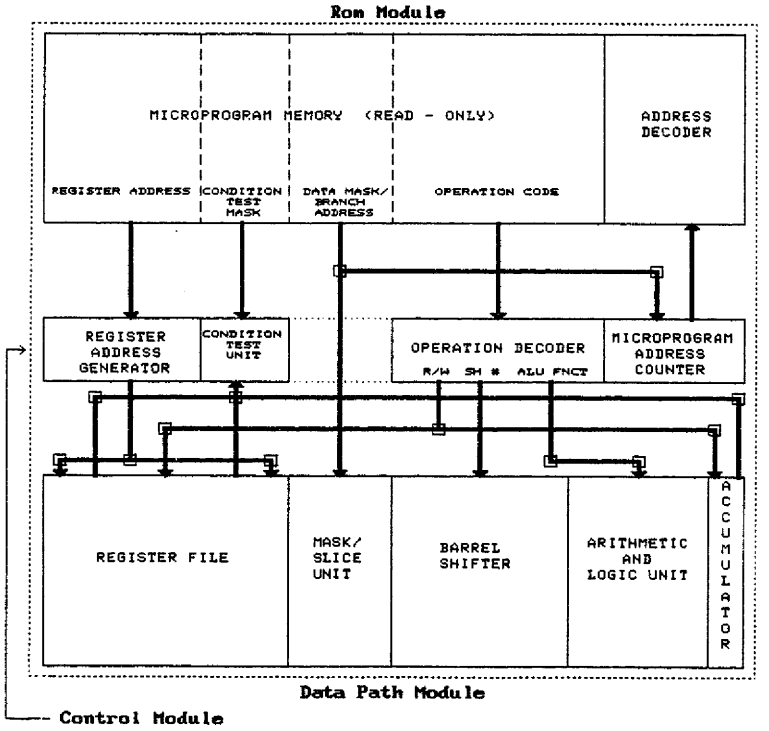

FIGURE 1 GENERAL SILICON LAYOUT

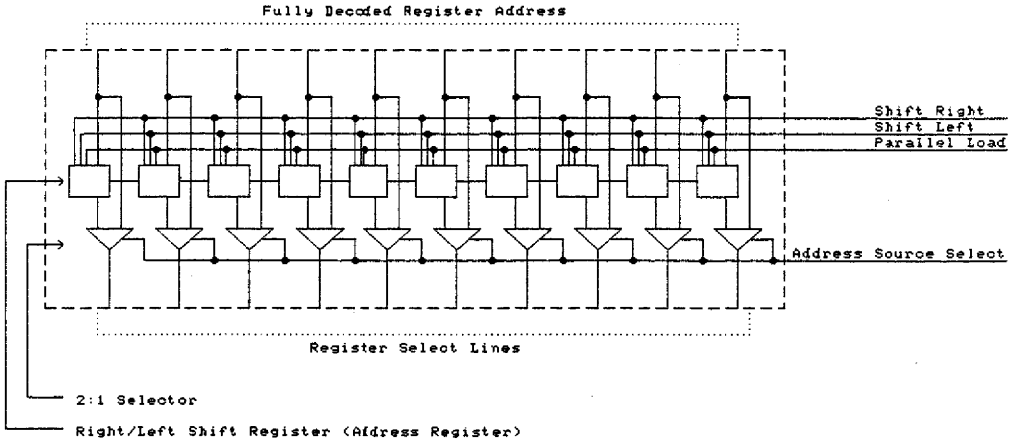

FIGURE 2 REGISTER ADDRESS GENERATOR

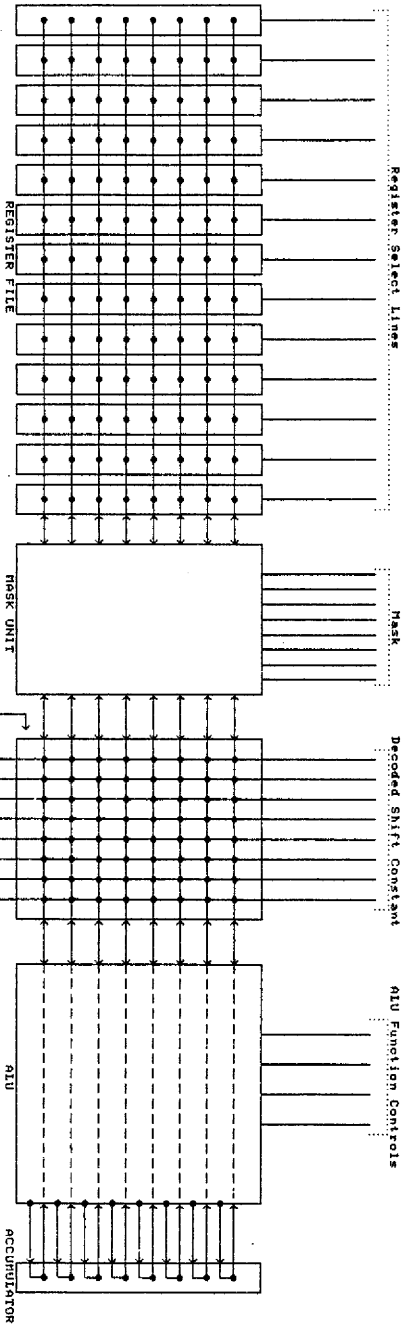

Figure 3 **THE DATA PATH**