Bit-Slice Design: Controllers and ALUs

Donnamaie E. White

Bit-Slice Design: Controllers and ALUs

Donnamaie E. White
Advanced Micro Devices, Inc.

Garland STPM Press
New York & London

Figures reproduced with the permission of *Advanced Micro Devices*.

15 14 13 12 11 10 9 8 7 6 5 4 3 2

Library of Congress Cataloging in Publication Data

White, Donnamaie E. 1942–
Bit-slice design.

Includes index.
1. Bit slice microprocessors—Design and construction. I. Title.
TK7895.M5W49 621.3819'58'2 79-7465
ISBN 0-8240-7103-4

Published by Garland STPM Press
136 Madison Avenue, New York, New York 10016

Printed in the United States of America

Preface

This text has been compiled from the current and highly popular Customer Education Seminar, ED2900A, "Introduction to the Am2900 Family," offered by Advanced Micro Devices. No attempt was made to duplicate all of the material presented in the customer seminar. The intent was instead to present a true "introduction" for the undergraduate hardware or software student that could be covered in one quarter or semester. The ED2900A seminar assumes that the attendee either has a background in assembly level programming or has a background in SSI/MSI design. This text also makes this assumption.

The flow is an orderly evolution of a CCU design, adding one functional block at a time. The material is presented in a dual approach, referring to both the hardware and the firmware, or the software impact, as each feature is discussed.

The controllers are presented first, followed by the RALUs and their support chips. Interrupts are presented in two sections broken down by the hardware evolution. The final chapter provides a "typical" configuration of an Am2900 state-machine architecture CPU.

Chapter 1 is an introduction to the reasons why microprogramming should be selected as the means of implementing a control unit. This chapter also presents a discussion of language interrelationships cover-

ing topics from the typing of the conventional programmer languages to the functioning of the hardware through the microprogram. The basic concept of what a control unit does is described using a primitive CCU (computer control unit). The 2900 Family is also introduced and this bipolar bit-slice family will be used throughout the text. The concepts, however, apply to *any* microprogrammable system.

Chapter 2 begins the design evolution of a controller and introduces timing considerations. The hardware-firmware duality of the design decisions are stressed. In relation to the CCU used as an example, the concept of a mapping PROM is introduced. Only PROMs are discussed, although DEMUX networks, gate arrays, and PLA (programmable logic array) and PLA-type logic units are often used to perform the decode operation. Microprogram memory (control memory) is also presented. PROMs are referred to throughout the text although ROMs, PROMs, EPROMs, WCS (writeable control storage), and even parts of main memory may serve as the control memory. Only single-level control memory is refered to in the text although some designs exist which use two-level control stores (nanoprogramming).

Chapter 3 continues the evolution of the controller adding subroutines, nested subroutines, loops, and case statements to the tools available to the microprogrammer. The concept of overlapping field definitions in a microinstruction is introduced in relation to the branch-address and counter-value fields. This is an elementary form of variable formatting, the use of which should be minimized for clarity. The controller evolution leads to the microprogrammable sequencers—the Am2909 and Am2911—and the next address control block, the Am29811. (The letters A or B following a chip identification refers to the latest version available and may vary over time.) The various versions are pin-compatible and differ usually in die size and speed.

The case statement introduces the Am29803A, a device which assists in implementing up to a 16-way branch.

Microprogram memory implementation is briefly discussed, introducing the use of the Am27S27 registered PROM, dc and ac loading, and the effects on sequencer timing of excessive capacitive load.

Chapter 4 continues the evolution of the CCU, introducing interrupt handling (the interrupt controller is discussed later). The interrupts are introduced here to demonstrate the $\overline{OE}_{VECT}$ requirement of the next address control block. The evolution finishes with a detailed discussion of the Am2910 instructions. The instructions are discussed in their conventional usage. A number of instruction set variations are possible by tying control lines to different instruction lines ($\overline{CCEN}$ to I_3, for example) and by ignoring the $\overline{PL}$, $\overline{VECT}$, and $\overline{MAP}$ outputs of the Am2910 and driving the output enables of these devices from the pipe-

line register (microinstruction) itself. The Am2914 interrupt controller is covered briefly.

Chapter 5 covers the RALUs—the Am2901 and Am2903—in a series of evolving steps as were the microsequencers. Every conceivable consideration cannot be discussed here, but enough is presented to cover the architecture of the Am2901.

Chapter 6 covers some basic operations and presents their microcodes to demonstrate microcode selection for these devices. Two's complement multiply is covered in some detail to highlight the differences between the Am2901 and the Am2903.

Chapter 7 describes the "typical" CPU as suggested by Advanced Micro Devices for the "typical" user. It covers the Am29705 two-port RAM and the Am2904 "LSI Glue" multiplexer-register support chip.

An instructor's manual of exercises and solutions has been prepared and is available from Advanced Micro Devices.

Although the text is original, many of the drawings have appeared in application notes and data sheets previously published by Advanced Micro Devices and are reproduced with the permission of Advanced Micro Devices. Those application notes written by the Bipolar Applications Department have served as the principal reference material. Principal authors of these notes, to whom I am indebted for their assistance and advice are:

John Mick, Engineering Manager, Systems and Applications, Digital Bipolar Products.

the late Michael Economidis, Section Manager, Systems and Applications, Bipolar Memory and Programmable Logic, Mr. Economidis was an expert on the Am2914.

Jim Brick, Applications Engineer, Bipolar Microprocessors

Vernon Coleman, Senior Applications Engineer, Systems and Applications, Bipolar Microprocessor Circuit Definition.

William Harmon, Manager, Systems and Applications, Bipolar Microprocessing.

Contents

1 Introduction

Over the years, there has been an evolution of the universal building blocks used by logic circuit designers. In the mid-1960s, there were SSI gates; NAND, NOR, EXOR, and NOT or INVERT. In the early 1970s, MSI blocks, registers, decoders, multiplexers, and others made their appearances. In the late 1970s, ALUs (arithmetic logic units) with on-board scratchpad registers, interrupt controllers, microprogram sequencers, ROMs/PROMs, and other LSI devices up to and including a complete one-chip microprocessor (control, ALU, and registers) became readily available.

SSI (small scale integration) is defined here to include chips containing approximately 2–10 gates. MSI (medium scale integration) is used for chips containing 20–100 gates. LSI (large scale integration) chips contain 200–1000 gates, with the upper limit continually extending as VLSI (very large scale integration) becomes a reality. The AmZ8000 CPU contains 17.5K gates; the M68,000 claims to contain 68,000 transistors.

Selection of the Implementation

Today, a designer is faced with three basic choices in implementation: (1) SSI/MSI hardwired logic; (2) 9080A/8080A (8-bit) or AmZ8000-In8086-M68000 (16-bit) MOS fixed instruction set (FIS) microproces-

sor; or (3) microprogrammable bit-slice architecture with the 2900 Family or other similar family. There are a number of factors which influence the decision as to which implementation is best for the application.

Architecture

In terms of the design architecture, any FIS MOS microprocessor by definition has its own predefined internal architecture, and this constrains the design options available. This fact is acceptable if the architecture provided by the selected MOS device satisfies the one desired for the application. An SSI/MSI implementation allows the designer to specify in complete, exact detail the architecture desired. With bit-slice devices, some constraints are placed on the designer, but most of the system architecture is left to user definition via the selected interconnections and the microprogram.

Size

The real estate or board space (rack space, etc.) is often of concern in a design because of space limitations. An FIS MOS microprocessor may use 3–6 chips for a typical average control system, versus 100–500 chips for the same system implemented in SSI/MSI and 30–60 chips for a compromise bit-slice design.

Word Length

The word length necessary for the system, whether a computer, controller, signal processor, or whatever, is usually known in advance. FIS MOS microprocessors can be used where their word length is compatible with the design objective. MOS devices exist for 4-, 8-, and 16-bit data word systems. Using SSI/MSI, any word length may be accommodated. Using bit-slice (the 2900 Family is expandable in multiples of 4 bits), a wide variety of useful word size systems are possible. When bit-slice does not conveniently match, SSI/MSI can be used to "patch" the basic bit-slice design.

Instruction Set

The instruction set that the system under design is to support has a major impact on the choice of implementation. The high dollar investment in software, which currently exceeds the hardware investments with a ratio as high as 10 to 1, often results in the prime directive of software compatibility: the new design *must* support the existing instruction set. FIS MOS microprocessors have a fixed instruction set. If there is an MOS microprocessor whose instruction set supports the design instruction set, then a microprocessor-based design can be

used. The current FIS microprocessors support assembly level languages and have software to support BASIC, PL/1, FORTRAN, PASCAL, and even COBOL. If the design has an unusual instruction set requirement, it would require that a program written in the desired instruction set be passed through an additional software process prior to actual MOS device execution.

The two most widely known 16-bit devices are the In8086, with its 8080-based architecture and instruction set, and the AmZ8000, with a general-register architecture and an instruction set based on the IBM SYS/370 and the DEC PDP 11/45.

An SSI/MSI design can be customed tailored to support any desired instruction set. A bit-slice design can be microprogrammed to support any desired instruction set. The principal difference between these two approaches is that one is done exclusively in hardware and the other (bit-slice) is done in hardware and firmware.

Speed

Another design criterion or specification is the required speed of the design. SSI/MSI using Schottky TTL and bit-slice (2900 Family) can support systems with 125 ns cycle times. MOS microprocessors are slower, with approximate cycle times of 1–2 μs. The newer MOS devices support 4–5 MHz clock speeds. The newer bit-slice devices are targeted for 100 ns microcycle systems. When instruction times are given for an MOS microprocessor, the instruction is a machine level instruction. To properly compare this with bit-slice or SSI/MSI, *macro*instruction execution times must be used where a macroinstruction is a machine instruction which the microprogram supports. Bit-slice designs exist with effective macroinstruction times of 320 ns (HEX-29) and 200 ns (SUPER-16) for register–register operations (Chapters 8 and 9 of AMD's Bit-Slice Microprocessor Design Series).

Tradeoffs

Design tradeoffs are summarized in Table 1-1. Basically, where high speed, long word lengths, or critical instruction sets occur, MOS FIS cannot be used. If design time–parts count–board space restrictions also exist, or if production volume does not support the effort required to do an SSI/MSI design (considered the most difficult to do correctly), the bit-slice devices are the best choice. It should also be noted that a microprogrammed bit-slice design is upgraded or changed, usually through a change of PROM or a reload or patch of writable control store, more readily than is a hardwired SSI/MSI design.

Bit slice devices are applied to three basic areas: machines with long words, machines with special instruction sets, and high-speed

Table 1–1 Design Tradeoffs

	SSI/MSI	Bit-Slice Devices	FIS MOS Microprocessor
Architecture	Any Desired	Pseudoflexible	Predesigned
Physical size (typical)	500 chips	50	3–6
Word length	Any Desired	Multiples of 2, 4	4, 8, 16
Instruction set	Any desired; May be wired	Any desired may be microprogrammed	Constrained if speed a problem
Speed	100–200 ns	100–200 ns	1–2 μs
Design time	Long, slow, if done correctly	Fast	Fast
Debug	Difficult	Development systems aid process	Development systems aid process
Documentation	Tedious, often outdated	Forced via microprogram	Software is major portion
Upgrades	Up to a full redesign required	Easily done, can be preplanned	Easily done
Cost	Highest	Medium range	Lowest

machines. The best examples are signal processors, with a low volume per particular specification and which require high speed and a long data word, and emulators such as the one for the SIGMA 9 (32-bit word) and the one for the GE 400 (24-bit word), where software compatibility to the existing system at increased throughput is mandatory. Variable instruction set minicomputers have also been developed using bit-slice which allow custom-tailored instruction sets to be microprogrammed around one fixed hardware implementation.

Microprogramming

Microprogramming is to hardware design what structured programming is to software design. If a bipolar (Schottky TTL) machine is to be built, in bit-slice or in SSI/MSI, its control should be microprogrammed. First suggested by Wilkes as a methodical way of handling the control unit of a system, it is now recognized as the best approach. Why?

First, random sequential logic circuits are replaced by memory (writable control store or ROM [read-only memory] or PROM [programmable ROM] or related devices). This results in or forces a more structured organization on the design.

Second, when a unit is to be upgraded, a field engineer can replace the appropriate PROM considerably easier than hardwiring and patching new components onto a crowded printed circuit board (PCB) with all of the associated pitfalls of such activity.

Third, an initial design can be done such that several variations exist simply by substituting one or more PROMs (changing the microprogram), and enhanced versions can be preplanned such that version B is constructed by simply adding a PROM or two to version A, simplifying production. The basic units would contain sparsely populated PCBs with upgrades provided for in the etch and connections. In these cases, simply adding PROMs (and changing others as required) expands the system. This technique is also commonly used for RAM memory (read–write memory) expansion.

The microprogram, documented in the definition file and in the assembly source file, serves as the principle documentation of the firmware. This, coupled with the modularity of the design as enforced by the use of microprogram control, provides a better opportunity for clearer documentation than multipaged schematics can provide.

Last, diagnostic routines can be included in the PROMs supplied with the final system and can be called in by a field engineer through a test panel and executed to aid debug. Some diagnostic routines could be microprogrammed into the system such that they are routinely executed in the normal running environment. For more severe testing, the

normal PROM memory could be swapped with a special test memory simply by substituting PROMs.

Advantages of LSI

If bipolar has been chosen over MOS because of speed, LSI is preferable to SSI/MSI for several reasons.

First, costs are reduced. LSI requires fewer parts and therefore fewer boards and less rack space. There is less etch and fewer pin connections with LSI as more and more of the connections are moved inside the package.

Second, using LSI improves reliability. Approximately 80% of the failures of working systems are caused by broken etch or by bent pins and other broken external connections. Using SSI/MSI, a typical controller might use 300 16-pin DIPs, for a total of 4800 pins. The same controller done with LSI might use 30 40-pin DIPs, for a 1200 pin total; the other connections having been moved inside of the package.

The 2900 Family is going to be introduced in this text. It is a design rule that every design should use industry-standard parts. The Am2900 family is considered to be the industry standard for bipolar bit-slice devices. It is a microprogrammable family of LSI-level complexity. Table 1-2 summarizes its advantages.

The Am2900 Family

The 2900 Family components include or will soon include (1) CPU-ALU and scratchpad register units: Am2901, Am2903, and the new Am29203; (2) microprogram sequencers and controllers: Am2909/2911 and Am2910; (3) bipolar memory: various devices, including error detection and correction controllers and support devices (Am2960 Series); (4) interrupt controller and support devices: Am2914, Am2913, and Am2902; (5) bus I/O: Am2950 and support devices; (6) DMA sup-

Table 1-2 Microprogramming with LSI—Advantages

More structured organization
Field changes—may be as simple as replacing a PROM
Adaptions—may be as simple as replacing a PROM
Expansions—preplanned, may be as simple as adding a PROM
Better documentation
Hardware and firmware can be designed in parallel
LSI uses fewer parts
LSI has better reliability
Diagnostic PROM can aid debug, maintenance

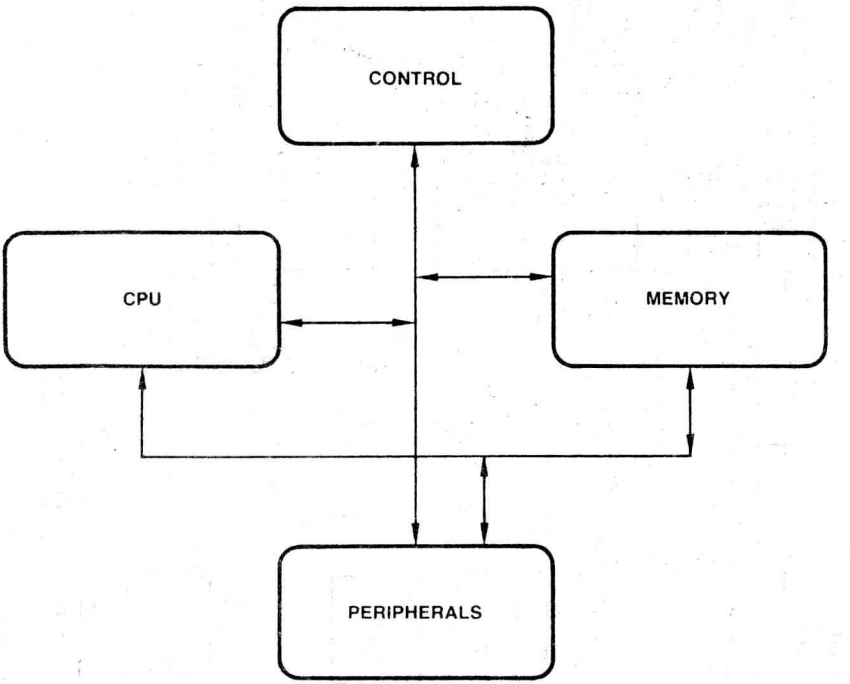

Figure 1–1. Simplex system block diagram.

port: Am2940 and Am2942; (7) timing support via microprogrammable microcycles: Am2925; (8) main memory program control: Am2930 and Am2932; and (9) the new 16-bit Am29116.

Consider a simplex block diagram of a basic computer, shown in Figure 1-1. The essential blocks of this diagram are (1) the CPU (central processing unit), containing the ALU and scratchpad registers, the PC (program counter), and MAR (memory address register); (2) the main memory, where active programs and data are stored; (3) peripherals, including backup memory, input, and output; and (4) the CCU (computer control unit), which supervises everything else and contains the control logic instruction decode and the PROMs. The CPU is where data is processed; the CCU is where instructions are processed.

From this simple overview, progress to Figure 1-2 and the generalized computer architecture blocked out to show the various members of the 2900 Family and their applications.

Language Interrelationships

Programming classes relate source code—written by the user in some programming language—to object code—the machine level, machine-

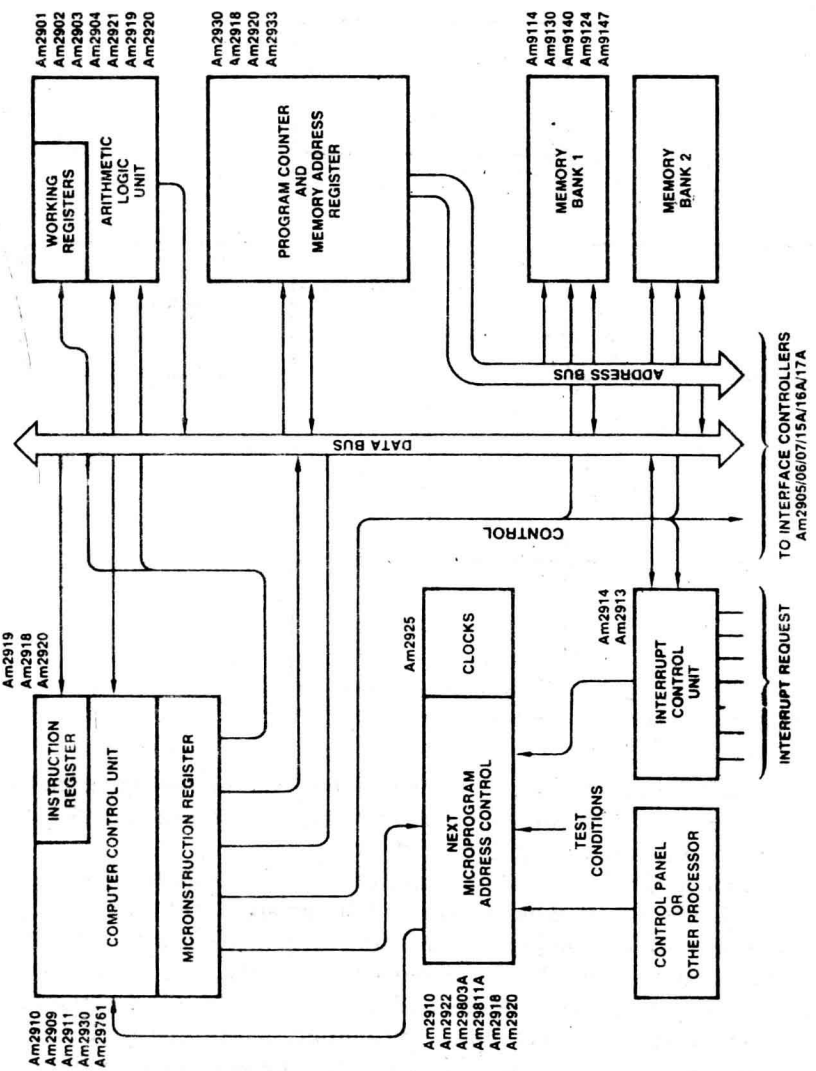

Figure 1–2. Generalized computer architecture.