INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

University Microfilms International
300 N. Zeeb Road
Ann Arbor, MI 48106

Nguyen, Thuyen Dinh

A CASE STUDY OF FLEXIBLE DISTRIBUTED PROCESSING SYSTEM IN COPIER DEVELOPMENT

The University of Arizona M.S. 1986

A CASE STUDY OF FLEXIBLE DISTRIBUTED PROCESSING SYSTEM IN COPIER DEVELOPMENT

by

Thuyen Dinh Nguyen

A Thesis Submitted to the Faculty of the

ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE
WITH A MAJOR IN ELECTRICAL ENGINEERING

In the Graduate College

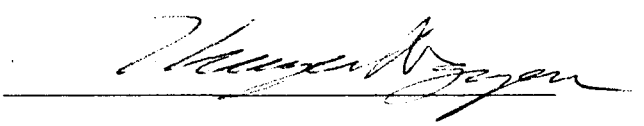
THE UNIVERSITY OF ARIZONA

1 9 8 6

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the Departmental Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his judgment the proposed use of the material is in the interest of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: 

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

Dr. Ralph Martinez
Professor of Electrical Engineering

5 May 86
Date

DEDICATIONS

This thesis is dedicated to my parents

Thi & Minh Nguyen and my sponsors Robert & Barbara Kokich

ACKNOWLEDGEMENTS

The author wishes to thank his advisor, Dr. Ralph Martinez, for constant encouragement and advice which made the completion of this thesis possible.

Thanks are also due to the IBM Corporation, Information Products Division, Boulder CO, for providing logistical support and facilities which made the documentation of this thesis possible.

TABLE OF CONTENTS

Table of Contents -- Continued

LIST OF ILLUSTRATIONS

ABSTRACT

This thesis describes how a distributed processing system (DPS) accommodates rapid and flexible prototyping of the system's subcomponents. The case study involves prototyping of the system's Master element after its development was delayed. The thesis describes an example of this flexibility in commercial copier engineering development.

CHAPTER 1

INTRODUCTION

From automated bank teller applications to automated manufacturing, distributed processing systems are being used in great number to provide economical and efficient solutions to today's challenges. This thesis describes a case study of how DPS also helps in engineering development when needs arise for quick prototyping of the system's elements. DPS is shown to provide flexibility in engineering development of commercial copier.

Statement of problem

A distributed processing system involving multiple microprocessors is designed to control a commercial copier. However as engineering development progresses, the Master element falls behind in schedule with respect to the other system's components, mainly Slave microprocessors, and begins to affect the overall development schedule of the copier product.

Design Approach

A scheme is formulated to use one of the system's Slave microprocessor to emulate some of the Master's func-

tions. The emulated functions allow the Slaves to be integration-tested as a whole system verifying its supposed functions. This prototyping of the Master also permits it to be more thoroughly designed.

The author's contribution is in the design, implementation and verification of the Master-emulating prototype, referred to as the Slaves Driver in this thesis. The author was also involved in the implementation of one of the Slaves.

Benefits of design approach

Using one of the Slaves to emulate the Master in controlling the other Slaves is probably the easiest and quickest method to implement a Master prototype. The hardware involved is minimal, not much more than swapping transmit with receive signal, and the software can be fairly similar to the Slave software in certain 'overhead' functions like the program supervisor, kernel, etc..

CHAPTER 2

HARDWARE ENVIRONMENT DESCRIPTION

The scenario which brought about the need for the Master emulator will first be discussed in this section. Following that will be a detailed description of the DPS hardware design.

System Overview

This section presents an overview of the distributed processing system (DPS) in consideration, the scenario of a mis-delivery of the master component and the recovery of master component emulation by one of the slaves.

High-level descriptions of the DPS in concern

The application in consideration is the machine control of a copier (named CX for the remainder of this document.) which includes a number of mechanical components. Among these are: paper input, image capture (scanner), image process & development, fusing and output collating. Classically, the control of these components have been handled by electronics means, i.e. programmable logic arrays (PLA) and small- to medium-scale-integration integrated circuits (MSI)

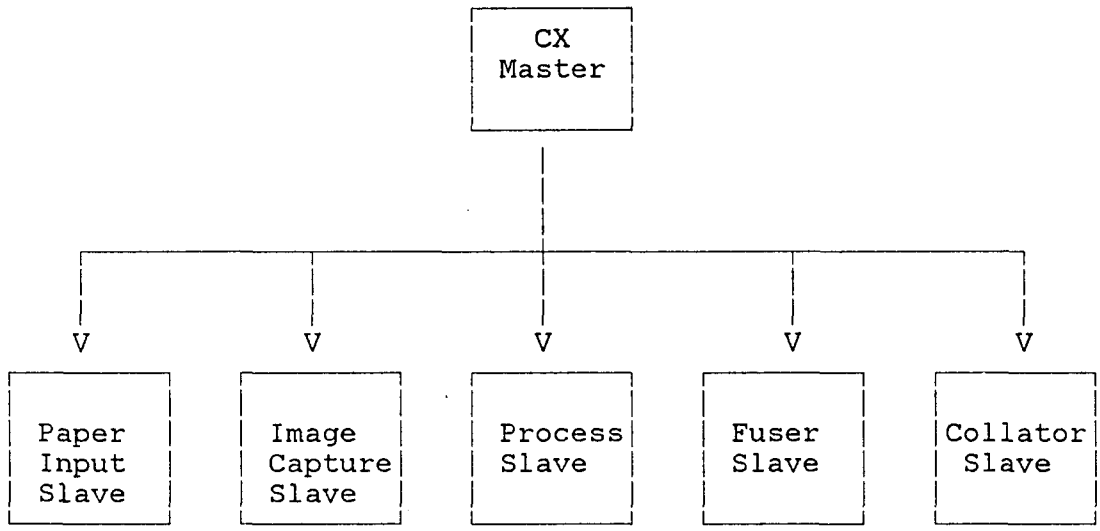


Figure 1. 'CX Distributed Processing Architecture'

The inevitable and spectacular price/performance improvements of microprocessors have changed all that. As microprocessors have moved into the home in the form of consumer products such as TVs, microwave ovens and PCs, they have also replaced PLAs and MSI ICs in industrial device control applications as well.

To take advantage of this revolution, CX is architected to have a number of microprocessors with one acting as a master and the rest as slaves. While the master is responsible for handling 'global' tasks such as IPL (initial program load), control the operator's panel interface and

coordinating the slaves, the slaves would be responsible for controlling the individual machine areas.

This shift towards microprocessors doesn't come without trade-offs. And although less PLA and MSI logic are required, it requires a great deal of programming to substitute for control previously performed by ICs. However, as any programmer would grimly accept, updates/changes in software are preferable to those in PLAs/MSIs. CX's DPS architecture is depicted in Figure 1 on page 4.

Description of delivery slippage of the master element of the DSP

CX's product development schedule plans for the following to be done chronologically: (as depicted in Figure 2 on page 6)

1. Parallel design of the individual slaves/master hardware and software.

2. Parallel implementation of the individual slaves/master hardware and software.

3. Parallel hardware <-> software integration debug of slaves/master

4. Slaves <-> master integration debug

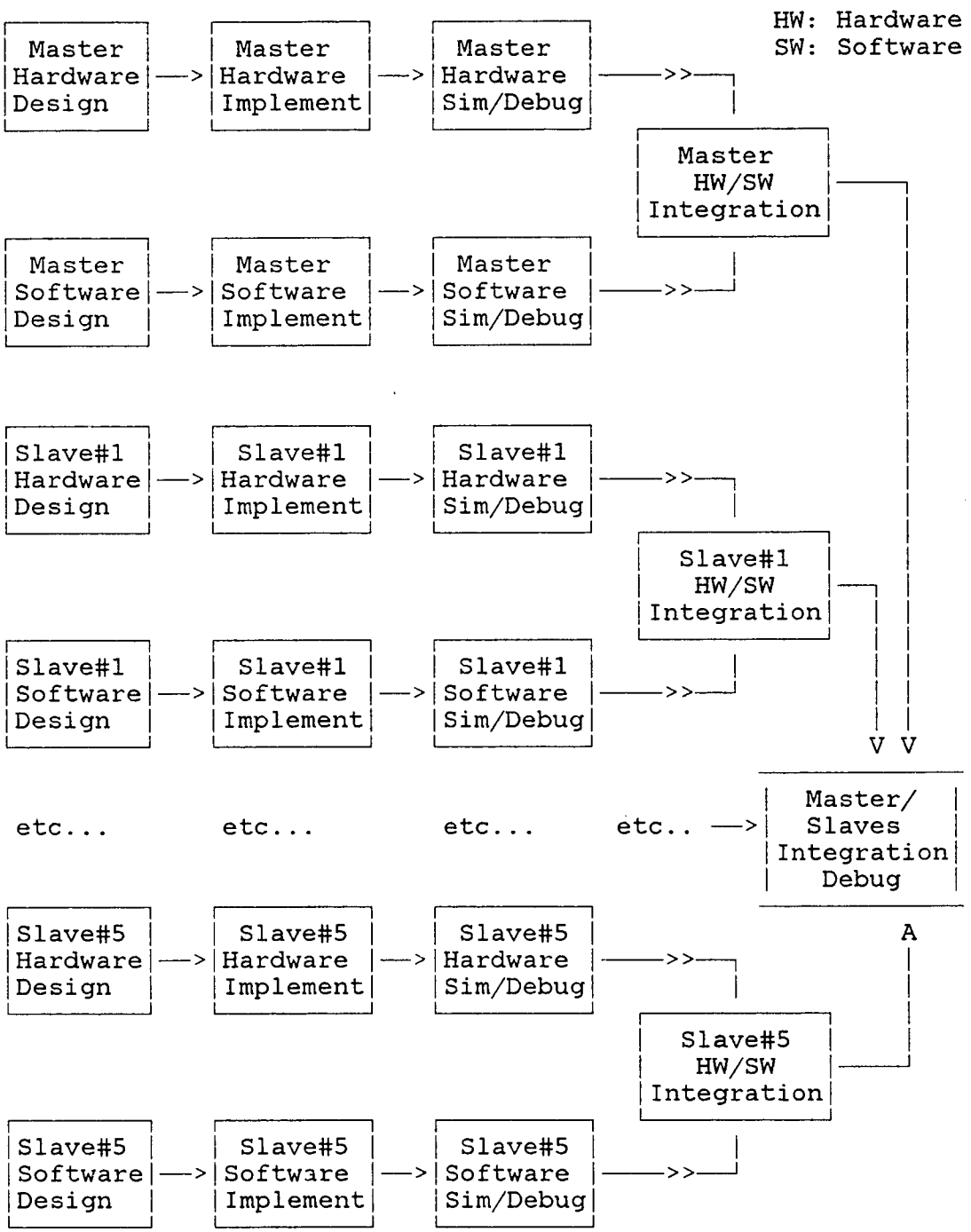


Figure 2. 'CX Product Development Procedure'

As the plan was implemented, it became apparent that the Master development path was not on schedule due to a number of reasons. First of all, the slaves were basically 8-bit microcontrollers, each with fairly simple instruction set and fast I/O operations. The master was really a full-fledged 16-bit high-integration microprocessor, intended to be a multi-tasking data processor.

Therefore the master software design had to be fairly complex, requiring a team of engineers working together on the development of the software. By contrast, the functional requirements of each slave were sufficiently independent that only one software engineer was assigned to the design and implementation of each slave's software.

Without a master, the slaves would still be able to function by themselves but lacking the master coordination they could not work in synchronization and thus failed to perform the functions of a copier.

Description of the recovery scheme by masterizing one of the slaves

Though it was possible to reinforce the master software development team with extra help or even with the slave software engineers, this would not have necessarily improved the situation as the point of diminishing returns for manpower addition was probably near. Additionally, because the Master microprocessor is completely different from the slaves', it would have taken some period of education (con-

cerning processor architecture, I/O, interrupts, instruction set etc..) before the added manpower could make any contribution.

It was then decided that a potential schedule-recovery scheme could work if one of the slaves was given some master-emulation capabilities and could then serve as the coordinator of all slaves including itself. Though this seems like duplicating the master's functions, it really only implements a few of the master's many functions.

Primarily the 'pseudo-master', hereafter referred to as the Slaves Driver, has to be able to drive all the slaves from the idle state, when all devices are off, into the active state, when devices are activated. Before going into the active state all slaves have to be synchronized by the Slaves Driver to maintain global machine timing integrity. Once in the active state, the Slaves driver must issue software commands to all slaves which together perform the tasks required to copy a single image.

All other master's functions are excluded from the Slaves driver's requirements. Examples of excluded functions are:

- Multi-tasking operating system
- Machine service diagnostics
- Control of the machine's operator panel
- Machine error retries

- Soft error recovery
- Etc..

And although the slave's microcontroller instruction set capabilities were not really suitable for the data processing task of the master, it was possible with some efforts to implement the Slaves driver with its intended functions. Eventually, the Slaves driver was verified to perform its functions in coordinating all slaves to perform the copier's tasks. And while the Slaves driver is never intended to emulate all the master's functions, it allows commencement of machine testing while the master software development continues.

Evaluation of the recovery scheme for this DPS vs uniprocessing

If the copier control had been uni-processor based, not only would the design/implementation have been much less parallel but there would not have been a structured approach in the short-term scheme of substituting the master to minimize adverse effects on development schedules. On the other hand, the distributed nature of the design nicely allows the detachment of the master and attachment of an interim master substitute in the form of the Slaves Driver.

Connection configuration

CX's control architecture is pretty strict classical master-slaves style (see Figure 1 on page 4 again.) The Master initiates all communications, Slaves will respond to the Master. Though the Master can communicate with any Slave , no Slave can communicate with another Slave (i.e. each Slave only talks to the Master.) Each Slave uses the same microprocessor, though the I/O configuration might differ from Slave to Slave . The Master microprocessor is more powerful than each Slave 's in spite of the fact that they are made by the same manufacturer, Intel ®

Master element

CX's master is an Intel 80286 microprocessor, a full 16-bit micro with integrated peripherals such as DMA Controller, Interrupt Controller and address decode logic. The master communicates with the Slaves serially using built-in UART facilities. Its functions are primarily global: coordinate all Slaves , error recovery, machine diagnostics, program load control, operator interface interaction, operator messages generation, etc.

Paper Input Slave

Blank sheets are fed into the machine by this Slave from a paper drawer. Each sheet is pulled from drawer and guided into the Process area where it will receive an image.

This Slave also controls the elevation of the drawer. As the drawer is emptied, the Slave program allows an operator to lower the drawer so that it can be filled with blanks. After paper loading, the program allows the raising of the drawer to the top, readying for continued operations.

Image Capture (Scanner) Slave

As the user presses the START_COPY button, the Scanner Slave sweeps a narrow strip of light across the original document, whose negative (reversed) image is automatically recorded onto the copier 's photoconductor. For each page scan completion, a duplicated image will be created and delivered at the copier 's output. For multiple copies, the Scanner repeats its actions creating multiple images on the photoconductor which are then matched to the blank sheets to produce duplicate copies.

CX allows an original to be scaled, i.e. enlarged or reduced in size when copied. After the user selects the scaling ratio (or use default of 1:1) the readings are sent to this Slave from the Master. The Scanner Slave then adjusts the scanning optical lenses to the correct distances which will produce the desired scaling. Normally, this will take a few seconds, during which the user will see the message 'Please Wait'

Process Slave

This Slave controls the electro-photographic operations of a copier . It receives a blank sheet of paper from the Paper Input Slave and transfers the document's negative image (which had been produced by the Scanner Slave) onto the paper to form a positive duplicate as the original document.

It guides the blank sheet to maintain precise transfer, i.e. each copy mustn't be skewed or located differently relative to the paper's edge, either as compared to the original or to each other within a multiple copy batch. This Slave also controls the darkness and contrast settings of the copies (selectable by users) by appropriately adjusting the toner concentration (the higher toner concentration, the darker produced copies) and scanner lumination (the brighter the lumination, the more contrast produced, i.e. more differences between gray and dark areas)

Fuser Slave

As each sheet exits from the Process Slave , it contains an image formed by the copier 's tiny toner particles. Because the toner is only attached to the paper electrostatically (a very weak force), the image must be 'fused' onto the paper. The Fuser Slave performs this by roll-pressing each copied sheet thereby 'glue-ing' the toner to the copy allowing more durable handling. To prevent jams,

the Fuser Slave checks once per page, according to commands from the Master, for arriving and departing sheets. When an error occurs, it shuts down the rolls' rotation and reports to the Master which then informs the operator through the message panel of a paper jam and request manual intervention

Collator Slave

This Slave receives a fused copy from the Fuser Slave and delivers it to the copier 's output. Normally, this means the upper output tray which is convenient for small jobs. However if a multi-page document is to be duplicated into multiple copies, then the 'Collate' option would be used. In this case, the Collator Slave directs each copied page into a different output slot. At completion, each slot contains the duplicated document with all its pages in the same order as the original's.

Before pressing the START_COPY button, the user can set the page count to indicate duplication quantity. As each copy arrives successfully to the copier 's output, the Collator Slave reports the event to the Master, which then can decrement the page count as shown on the operator message panel. When the page count reaches zero, all copies will have been produced and can be retrieved either from the upper output tray or the Collator's slots.

Slave Microprocessor description

Each Slave uses a 40-pin Intel 8051 microcontroller as its 'brain.' The 8051 provides limited I/O-mapped I/O. However, virtually limitless additional I/O can be achieved by memory-mapped I/O. Each microcontroller contains on-chip ROM, RAM, UART whose intentions are to reduce needs for additional external peripheral ICs.

Intel 8051 Controller

The 8051 is an 8-bit microcomputer with 1-us (microsecond) machine cycle. Most instructions take one cycle to execute. Two instructions, Integer Multiply & Integer Divide, take 4 cycles (i.e. 4 us.) The rest take two cycles to complete execution. Its 16-bit Program Counter permits 64K address space to accommodate memory-consuming applications. Though this may not seem like much compared to the current wave of 32-bit microprocessors, it must be noted that microcontrollers, like those used in consumer products such as microwave ovens, refrigerators are more typically 4-bit micros allowing perhaps 2K address spaces!

Its 8-bit Stack Pointer provides enough stack space for most applications. The actual stack resides on-chip and is part of the internal RAM. Subroutine invocation (i.e. CALL operation) automatically stores the return address on the stack and a RETURN operation automatically restores the Program Counter to its next address. PUSH & POP operations

allow the stack to be used for local temporary variables and for storing processor context (Accumulator, etc.) when servicing interrupts.

Internal ROM

Each 8051 contains 4K bytes of on-chip Read-Only Program Memory which is mask-programmed according to data provided to the manufacturer from the user. This memory extends from address 0000 to 4095; upon processor reset, control is transferred to address 0000 and the internal ROM program starts to execute. Although additional program memory (ROM or RAM) can be added beyond address 4095, the 8051 will fetch instructions whose addresses are below 4K from the on-chip program memory.

Internal Data Memory

There are 128 bytes of on-chip RAM which contains program variables. Because they are on-chip, access time is fast and most instructions referring to internal RAM take 1 cycle. There are specific parts of the internal RAM which can be used for special purposes as depicted in Figure 3 on page 16. The first 32 bytes can be used as 4 register banks, each containing 8 general purpose scratchpad registers. Only one set of 8 registers is active at any one time; therefore the register access instructions require only a 3-bit field and generally only take up 1 byte! In contrast, instructions accessing other internal RAM bytes require an 8-bit address.

Note:
- Any internal RAM byte can be used for general purpose as variable storage or as part of the program stack. Some have special purposes as outlined below.
- All addresses are in hexadecimal notation

Address range	Special Purpose
0 - 7	Register Bank #0
8 - F	Register Bank #1
10 - 17	Register Bank #2
18 - 1F	Register Bank #3
20 - 2F	Bit-addressable Space
30 - 7F	
.....	

Figure 3. 'Internal Data RAM Organization'

The next 16 bytes of the internal Data RAM can be accessed bitwise as 128 individual bit entities. They are typically used to store flags or logical conditions, each of which can be modified in 1 machine cycle using 1 instruction! Each of these bits can also be used as a 'jump' condition (i.e. jump to address xxxx if bit yy is cleared)

Interrupt Architecture

There are 5 separate sources of interrupt, each of which can be prioritized as either 'high' or 'low' priority.

Two are for external signals which can be either edge-triggered or level-triggered. The software selects edge- or level-triggered method in interpreting the signal as interrupt. Another source is the Serial Interrupt, generated when the UART receives a valid frame of data.

The last 2 sources come from the processor timers, each generate an interrupt when it expires (i.e. reaches 0000 after counting up.) Each timer can be configured to count machine cycles in which case it is a delay timer (e.g. delay 10000 machine cycles <==> delay 10000 us.) Alternately, each timer can be configured to count external signal transitions in which case it is an events counter (e.g. delay 120 ticks from a digitized 60-Hz signal in order to realize a 2 second delay.) Each interrupt source can be masked by software when necessary. The reset signal, which interrupts processor execution and restarts it from address 0000, is never maskable.

Direct I/O (I/O-mapped I/O)

There are 4 direct I/O ports, each 8 bits wide, occupying 32 of the 8051's 40 pins. Each port can be accessed either as a whole byte or as 8 individual bit-wide I/O lines. As shown in Figure 4 on page 18, depending on usage of other 8051 facilities, not all 32 lines are usable for unrestricted I/O. Certainly, all Slaves use its PORT 0 without restrictions. Because any direct I/O line is accessible as a bit variable, operations on them is similar to those on the

Address range	Comment
PORT 0: bits 0 - 7	Usable as I/O only if all software program resides in on-chip 4K byte ROM Not usable as I/O if additional software requires external RAM, which is accessed by using PORT0 as LOW-ORDER address lines
PORT 1: bits 0 - 7	Usable as I/O lines without restrictions
PORT 2: bits 0 - 7	Usable as I/O only if all software program resides in on-chip 4K byte ROM Not usable as I/O if additional software requires external RAM, which is accessed by using PORT 2 as HI-ORDER address lines
PORT 3:	
RD (bit 7)	Not usable if external RAM is required, instead it's used as memory READ signal
WR (bit 6)	Not usable if external RAM is required, instead it's used as memory WRITE signal
T1 (bit 5)	Not usable if counter 1 is used, instead it's used as an event signal
T0 (bit 4)	Not usable if counter 0 is used, instead it's used as an event signal
INT1 (bit 3)	Not usable if external interrupt 1 is used, instead it's used as interrupt signal
INT0 (bit 2)	Not usable if external interrupt 0 is used, instead it's used as interrupt signal
TXD (bit 1)	Not usable if UART is used, instead it's used as TRANSMIT signal
RXD (bit 0)	Not usable if UART is used, instead it's used as RECEIVE signal

Figure 4. 'Direct I/O Ports'

bit-addressable area. Each signal can be set, cleared or read in 1 machine cycle.

I/O Expander (memory-mapped I/O)

To provide additional I/O lines, each Slave has one I/O expander using the Intel 8255 chip. Each I/O expander has 3 byte-wide ports, two of which can be programmed as all-input or all-output; the 3rd port is regarded as 2 nibble-wide half-ports, each of which can be programmed as input or output. With this, each Slave adds 24 I/O lines to its overall I/O capabilities.

Each I/O expander port is selected as memory-mapped I/O and accessed whenever its address is present on the memory bus. Obviously the I/O expander ports addresses differ from the external RAM addresses to avoid access conflict. Since the expander ports are only accessible bytewise, expander I/O is a little bit more involved. For example to change a bit on an expander port, the program first has to read from the expander port to retrieve what's out there already, mask the appropriate bit (logical AND / OR), then write the masked result back out. Generally, it takes about 5 times longer to operate on expander I/O than direct I/O.

Instruction set overview

The 8051 provides 5 principal addressing modes:

1. Register: refers to any one of the 8 general purpose scratchpad registers located in the currently active register bank.

2. Immediate: refers to the instruction's constant value operand
3. Direct: refers to any one of the 128 internal RAM bytes. Also refers to 8-bit processor facilities such as SP (Stack Pointer), SBUF (UART serial communications buffer), etc.
4. Register Indirect: refers to an internal RAM byte whose address is contained in a general purpose register.
5. Base+Offset Indirect: refers to the program memory byte whose address is formed by adding a 16-bit address register to an 8-bit offset register.

All typical Arithmetic-Logical operations are supported: MOVE, ADD, SUBTRACT, INCREMENT, DECREMENT, LOGICAL_OR, LOGICAL_AND, JUMP_ON_ZERO, JUMP_IF_EQUAL, JUMP_IF_DIFFERENT, etc.. Using the 1-byte opcode and up to 2 more bytes of operands, 8051 instructions operate on logical entities from bit to nibble to byte to word (16 bits) A full 8051 instruction repertoire is described in appendix A

Master - Slaves Protocol overview description

Communications between the Master and Slaves take place using the Universal Asynchronous Receiver/Transmitter (UART) standard. This allows communications to be carried out serially over 2 signal lines, one each for transmit and receive.

TXD: Serial communications transmit from Master to a Slave
RXD: Serial communications receive by Master to a Slave

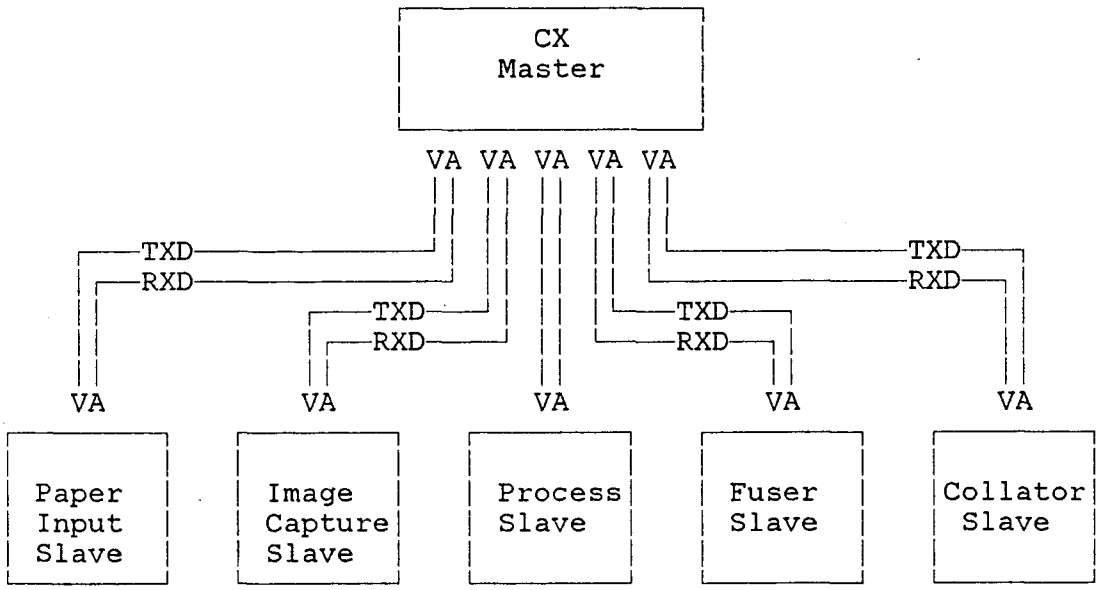


Figure 5. 'Communications Direct Connect Configuration'

Multidrop configuration

In interconnecting the Master with Slaves , two approaches are possible:

1. Direct connect: where each Slave is connected directly to the Master through dedicated transmit/receive lines. Thus the Master has multiple sets of transmit/receive lines, each routed directly to a specific Slave as il-

TXD: Serial communications transmit from Master to any Slave
RXD: Serial communications receive by Master from any Slave

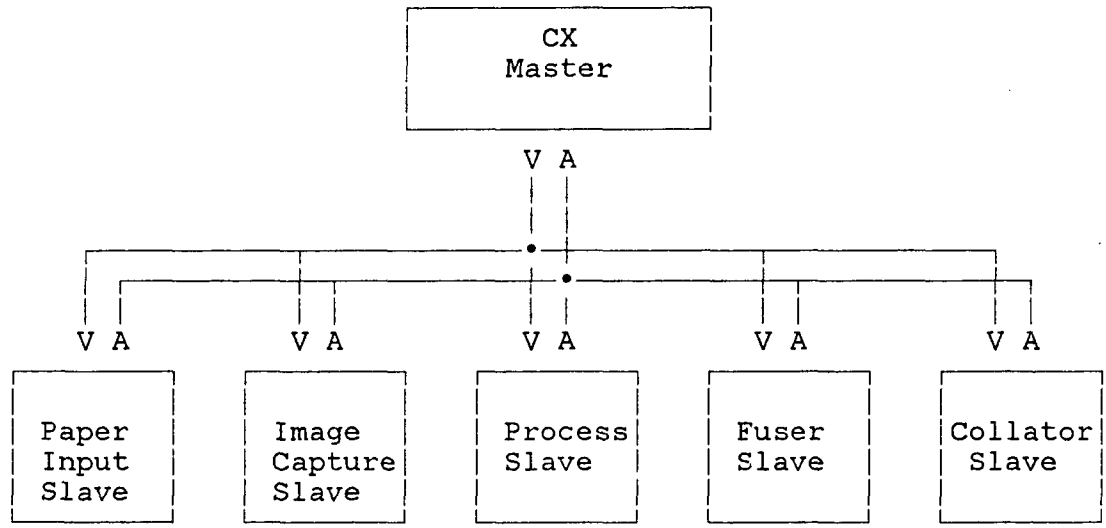


Figure 6. 'Communications Multidrop Connect Configuration'

lustrated in Figure 5. This approach simplifies the communications multiplexing among Slaves by the Master since it only has to select the appropriate set of communications lines to talk to any Slave . In simplifying the software, it requires additional hardware in that there must be one additional set of transmit/receive lines for each Slave added to the DPS.

2. Multidrop connect: where the Master and each Slave is connected to a common communications bus. This simplifies hardware in that any Slave can be 'dropped' or at-

tached onto the existing bus without affecting the hardware of any existing DPS components. Besides, this approach maximizes communications bandwidth of the available medium in that the bus is used whenever any communications take place; whereas in the direct connect configuration only 1 out of n (in this case n=5) sets is utilized at any one time.

CX uses the multidrop connect configuration mainly to simplify communications hardware; this is also a more typical choice for its distributed processing nature. CX's configuration is depicted in Figure 6 on page 22. In return, now each Slave has to know when the bus contents is intended for it and when to ignore when it's for someone else. Each Slave does this by using the 'wakeup' feature of the 8051 built-in UART as described in Figure 7 on page 24.

Built-in UART

The 8051 provides a built-in UART on-chip to facilitate serial communications. The 8051's UART hardware implements standard UART specifications in automatically detecting START bit, shifting serial data into a parallel register and recognizing STOP bit. The communications speed is programmable by software from 122 to 31250 bits per second. This allows compatibility to the typical rates from 110 through 19200 baud. Alternately, the speed can be selected to be

The following events take place chronologically to allow the Master to address any Slave :

<A> Slaves program the UART to interrupt 8051 only if wakeup=1

<B> Master transmits a data frame containing address of the destined Slave and an affirmed wakeup bit (wakeup=1)

<C> Slaves get interrupted when the wakeup data frame arrives. The interrupt service routine compares received address with its own. If they compare, the addressed Slave reprograms its UART to receive subsequent data frames

<D> Master transmits data frames intended for the Slave addressed in step <C> with a negated wakeup bit (wakeup=0) Only the addressed Slave can receive these frames, the other Slaves still only respond to frames with wakeup=1!

<E> After the last data frame, the Master transmits a control frame still with a negated wakeup bit (wakeup=0) This tells the addressed Slave of 'End-of-Data' so it can disconnect

<F> Upon receiving the control frame, the addressed Slave re-programs its UART to interrupt only if wakeup=1, reverting itself to step <A>

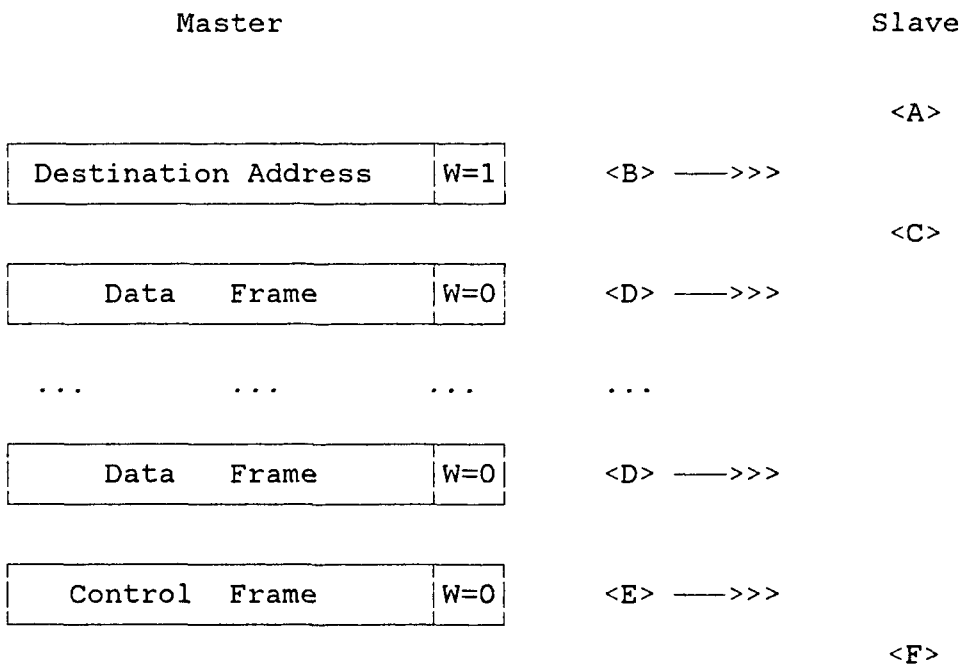


Figure 7. 'Use of 8051 wakeup facility to address Slave on multidrop connection'

fixed at 1/64 of the processor oscillator or 12MHz/64 = 187500 b/s.

The software interfaces with the 8051 UART through a byte interface. As each serial data bit is received, the UART shifts it into a byte-wide register, eliminating needs for software to read in each bit. When a whole data frame is received, as indicated by the STOP bit and time delay, the UART generates a software interrupt. The interrupt service routine then can read the received frame from a byte-wide buffer called SBUF. Similarly, the software writes into this SBUF register in order to transmit a data frame out onto the Transmit line. The UART hardware automatically performs the reversed parallel-to-serial conversion while transmitting data to conform to the UART standard.

Transmission Mode (Poll/wakeup)

The 8051 UART can be programmed to receive only serial datum which has a positive 'wakeup' bit. If this wakeup bit is affirmed in a data frame which contains the address of the receiving Slave then each Slave can be addressed uniquely and waken up to receive its data frames. To allow for transparent byte-size data transmission, the wakeup bit is actually a ninth data bit transmitted by the UART hardware right before the STOP bit. A scenario of wakeup bit usage is given in Figure 7 on page 24.

CHAPTER 3

SOFTWARE ENVIRONMENT DESCRIPTION

Though not a great deal of hardware was involved to implant the Slaves Driver , a fair amount of software was needed to emulate the master's tasks. First, the master software was to be written in a high level language whereas the Slaves Driver must be written in the 8051 assembly language which is also used by the Slaves software engineers. Secondly, the 8051 microcontroller was really more adept at 'bit-flipping' than the data-processing tasks the Slaves Driver now have to emulate.

Software Overview

The Slaves Driver principal tasks are to produce a copied image through the process of controlling all Slaves However, before this can happen, the Slaves programs must be IPL'ed into their memory and the copier hardware, albeit scattered throughout the Slaves , must be coordinated to work together through commands from the Slaves Driver.

Slave program load

Since the Slave program is located in RAM they must be downloaded by the master. Obviously, the Slaves Driver has to emulate this feature to make the Slaves functional.

Sequencing of Slaves

After all Slave programs are loaded, the Slaves are considered to be in the WAIT state and expect to receive WAIT -state commands from the master to carry out Slave -specific tasks. To emulate the master, the Slaves Driver has to issue a specific sequence of WAIT -state commands to the Slaves in order to bring them gradually from the WAIT to the COPY state

Synchronization of Slaves

Once the Slaves are in the COPY state, they are prepared to accept commands to work together in performing a copier 's functions. Again, to emulate the master, the Slaves Driver issues specific COPY -state commands to all Slaves so that in synchrony they can achieve what each Slave can't individually.

Slave Error detection

When an Slave detects an error (e.g. paper jam), it tries to handle the condition (e.g. deactivate motors it controls) but more importantly, reports the condition to the master so that the other Slaves can be halted as well. The Slaves Driver emulates this master's function in halting all

Slaves in case of error so as to minimize potential machine damage.

Software Modules

IPL Task

This IPL task emulates the master in sending IPL data to the Slaves so that they can be stored into each Slave 's RAM before execution begins. To maintain communications compatibility with the Slaves the existing communications hardware is used, therefore the Slaves don't know whether they are IPL'ed from the 'real' master or an emulated master. As a result of this, the loader program handling the IPL of each Slave responds and performs the task correctly as though the IPL data were coming from the 'real' master.

Commands Sequencer

First, the Slaves Driver is responsible for sequencing all Slaves from the WAIT to the COPY state. Secondly, once in the COPY state it is also responsible for issuing the appropriate combination of commands to coordinate all Slaves . Each Slave expects one command as each sheet of paper passes through the copier Each Slave accepts that command and interprets it with regards to the sheet currently under its area of hardware control.

Going from the WAIT to the COPY state, the Slaves Driver must command the Slaves individually to perform the following tasks chronologically:

1. Issue command to the fuser Slave to assure that the fuser temperature is adequately warm to perform fusing of toner to paper. If not, fuser lamps need to be activated.
2. Issue command to the process Slave to activate the drum to start it turning, the drum 's motion automatically generates the TACH signal which will be used for machine timings.
3. Issue command to the Image Capture Slave to activate the appropriate optical lens which are used in focusing the image to be copied as well as in scaling (enlarging, miniaturizing) of images.
4. Issue command to the Paper Input Slave to assure that the paper drawer is in place and raised to the top in preparation for paper feeding.
5. Issue command to the Collator Slave to assure that the Collator bins are in place and sufficient room remains in preparation for accepting processed copies.
6. At this point, the Slaves Driver issues a global command to synchronize all Slaves which then reset their internal clocks and operate in sync with each other and the Slaves Driver

Once in the COPY state, the Slaves Driver then issues COPY -state commands to each Slave , once per sheet, to coordinate them in performing a copier 's tasks. Each command is Slave -specific. For example, commands for the Paper In-

put Slave might be: Pick a sheet, Position it in a platen, Transfer it to the Process area for image development; whereas commands for the Image Capture Slave might be: Scan the input image, Set contrast (for varying copy/background ratio), Set intensity (for varying darkness of copies)

The contents of all commands above reside in the commands table. The Command Sequencer program is table-driven by this table and merely loads the current command from the table, then passes it to the Communications Driver which performs the actual command transmission to the Slaves

Communications Driver

This module directly carries out communications between the Slaves Driver and each Slave by emulating the original master - Slaves protocol. Basically the communications are 2-way: commands are transmitted by this module to be received, interpreted and carried out (by means of controlling devices under the Slave control) by the Slave ; when a Slave detects errors it reports them to the Communications Driver which then returns them verbatim to its caller.

Again, to maintain compatibility with the original protocol, this module performs the necessary protocol overhead such as: polling the Slaves , generate and perform data checks (parity, checksum), handshaking and terminating the transmission per specifications.

Commands Table

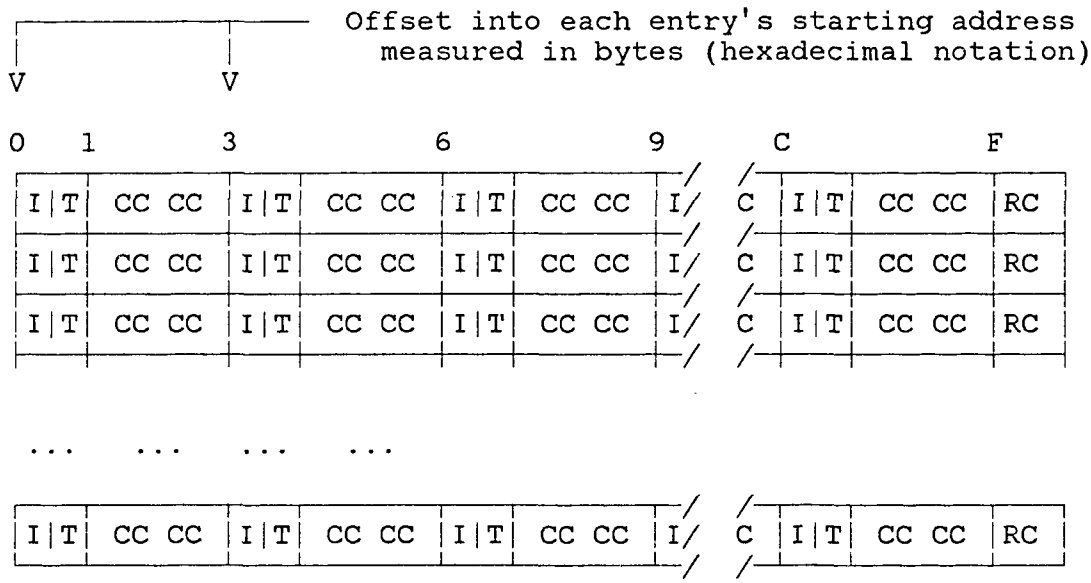
Since the copier is still in a developmental stage, the commands issued by the Slaves Driver to specify tasks to be carried out by the Slaves , though defined in the design phase, are still subjected to fine-tuning as machine tests reveal problem areas. Therefore, it's important that the commands are easily modified to facilitate expedient fine-tuning in the lab without having to go through the process of program alteration, reassembly and program download.

Obviously, this calls for the commands to be stored in one single table where they are easily accessible by the debugging engineers. Having them all in one table also expresses the inter- Slave relationship of the commands, e.g. how a 'Scan Input Image' command (sent to the Image Capture Slave) works with a 'Pick a sheet' command (sent to the Paper Input Slave), etc. all the way to a 'Collate a sheet' command to deliver the reproduced copy at the copier 's output.

Table data structure. As depicted in Figure 8 on page 32, the Commands Table consists of a number of entries, each contains commands to be transmitted to all Slaves per page time . To allow repetitive entries, a 'repeat count' feature is available. In turn, each commands entry is comprised of 5 sub-entries, one per Slave , and a repeat count. Each sub-entry contains:

1. The Slave id (range: 1 thru 5) indicating the id of the receiving Slave

Notes:
- The commands table contains a number of command entries
- Each entry is depicted as 1 horizontal set of boxes below
- There are 16 bytes in each command entry
- Each entry contains five 3-byte Slave command & an entry repeat count



LEGEND

I : Slave Identification, specifies the Slave receiving command
T : Command Type, specifies WAIT or COPY command types
CC: Command Contents, specifies actions or tasks to be done by the receiving Slave
RC: Repeat Count, specifies the number of times the entry is executed (sent)

Figure 8. 'Commands Table Structure'

---

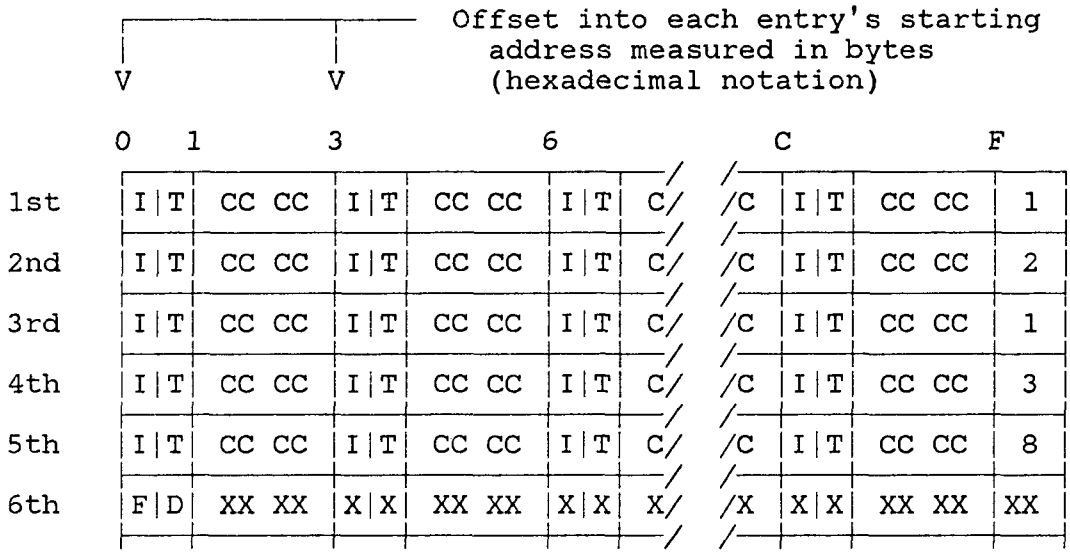
2. Command type (range: 0 or 1) indicating the type as WAIT or COPY and command length (range: 0 thru 3) indicating

the number of bytes of command contents immediately following current byte. As agreed by all Slaves , this length is fixed to 2.

3. Command contents which describe Slave - specific actions

Commands Entry Looping feature. The commands sent to the Slaves tend to be repetitive after an initial 'ramp-up' period. Therefore, a 'table-looping' feature allows such portion of the table to be re-used rather than continuously extending the table. To use the 'table-looping' feature, one must codify the entry's 1st 'I' field (Slave ID field) as an 'F' hexadecimal; note that this doesn't conflict with the typical values (1 thru 5) when this field is used to indicate Slave ID.

When an entry's 1st 'I' field is coded as 'F', then the Command Sequencer module will 'back-up' from the last entry an amount as specified by the 'T' field. In the example described by Figure 9 on page 34, if the current entry is the 6th, the current entry's 1st 'I' field contains 'F' and its 'T' field contains 'D' hexadecimal, then the current entry will be re-set to: 6 + 'FD' = 6 + (-3) = 3; note that 'FD' is regarded as the signed 2's complement representation of -3. The final result is that the 3rd entry will follow the 6th one, thereby creating a 'loop' starting at the 3rd ending at the 6th entry.



The Commands Table above will be traversed as follows:

- 1st entry sent once, as indicated by its repeat count=1
- 2nd entry sent twice, as indicated by its repeat count=2
- 3rd entry sent once, as indicated by its repeat count=1
- 4th entry sent thrice, as indicated by its repeat count=3
- 5th entry sent 8 times, as indicated by its repeat count=8
- Now, the 6th contains a table loop command, forcing current entry re-set to 6 + (FD) = 6 + (-3) = 3, i.e. 3rd entry
- 3rd entry sent once, as indicated by its repeat count=1; commands will from now on be repeated from 3rd entry to 6th entry

LEGEND

X : Don't Cares, values don't affect execution because the entry's I-field is coded to indicate table-looping

Figure 9. 'Commands Table Traversing Example'

Commands Entry Repeat Count feature. To take further advantage of entry repeatability, an entry repeat count is available for duplicating the effects of multiple contiguous

identical entries. See Figure 9 on page 34 for example. The repeat count of '00' is reserved as 'infinite' in which case the current entry will be sent ad infinitum. As a practical application, this feature can be used to indicate a run of n pages where n is specified as the repeat count. Note that together with the entry 'looping' feature, a table can be created economically to indicate sequences like: Image 100 copies, Pause 10 pages, then repeat.

Configuration Specifications

To facilitate machine bring-up, there is a configuration byte which contains the information of whether or not a Slave is currently 'on-line' By appropriately initializing this value, one can include or exclude Slave or Slaves to fit the current machine configuration. See Figure 10 on page 36 for format. For example, as the machine is brought up, if the Image Capture Slave is the first Slave available for debug (i.e. driven by the Slaves Driver) then all bits in the configuration byte are OFF except for that of the Image Capture Slave .

As the next Slave comes on-line, for example the Paper Input Slave , then its on-line bit will be set, enabling the Slaves Driver to exert control over it. On the other hand, if the Collator Slave happens to be having trouble and needs servicing, then the it can be taken off-line easily by disabling the appropriate configuration bit allowing continued debug/testing of all other Slaves . It is primarily a

Notes:
- If the on-line bit is on, then the specified Slave is on-line and ready to be driven.
- For example, if the configuration byte has the value 'C8' hexadecimal, then the Paper Input, Image Capture and Collator Slave are on-line, the others are off-line

Bit 7 (MSBit)	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 (LSBit)
Paper Input online	Image Capture on-line	Process on-line	Fuser on-line	Collate on-line	NOT USED	NOT USED	NOT USED

Figure 10. 'Machine Configuration Byte Structure'

useful feature in the debug phase which provides maximum flexibility/tolerance of machine's componental availabilities.

Status Retrieval Module

When an Slave detects and reports an error, the Slaves Driver must be able to handle this as the master would have and report the error to the machine operator. To obtain maximum flexibility, the Slaves Driver allows the user to specify 'fatal' errors (which require machine shutdown) in a 'Stop-Error Table' the structure of which is described in Figure 11 on page 37.

Byte offset:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Stop-Errs#1	00	80	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Stop-Errs#2	00	00	C0	01	00	00	00	00	00	00	00	00	00	00	00	00
Stop-Errs#3	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Stop-Errs#4	00	0E	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Stop-Errs#5	00	00	00	F0	00	00	00	00	00	00	00	00	00	00	00	00

As an example, the above Stop-Error mask entries mean:

- When monitoring status of Slave #1, if bit 7 (MSBit) of byte 1 is '1', then shutdown machine
- When monitoring status of Slave #2, if bit 7 or 6 of byte 2 or bit 0 of byte 3 is '1', then shutdown machine
- Effectively, no monitoring status of Slave #3 takes place since the Stop-Error entry is all 00's, i.e. 0 AND X=0
- When monitoring status of Slave #4, if bit 2,1 or 0 of byte 1 is '1', then shutdown machine
- When monitoring status of Slave #5, if bit 7,6,4 or 3 of byte 3 is '1', then shutdown machine

Figure 11. 'Fatal- (Stopping-) Error Table Structure'

Each time a Slave reports an error, the Slaves Driver masks it (Logical AND Operation) against the Stop-Error entry for that Slave . If the result is non-zero (i.e. an 'ON' bit of the entry is in the same position as a Slave error) then the machine is shutdown. This capability allows the machine to run in existence of errors which are either known to be 'false alarms' or triggered by known machine malfunctions in

progress of being repaired. Of course, this feature can be misused as it can 'hide' machine problems unless they are truly being attended to.

CHAPTER 4

DEVELOPMENT SYSTEM ENVIRONMENT

The Slaves Driver is developed using available microprocessor-based development system accommodating design, documentation, software development, software & hardware emulation and verification using the same family of development system.

Development/Debug tool usage

Intel iPDS development system

The Intel Personal Development System (iPDS ®) is a multi-purpose 8085-based development system. It uses the Intel ISIS ® operating system which is widely used by other Intel development systems such as Intellec ® Series/II, Series/III and Series/IV. Under ISIS the user can perform 8051 program creation by using an editor program, 8051 program assembly, linkage and program library archival. Its peripherals include secondary floppy drive(s), matrix printer and RS-232 communications port.

Intel EMV51 emulator

The iPDS also supports an 8051-emulation program to verify program functions. The EMV51 ® (Emulation Vehicle for

8051) program interacts with a special 8051 chip to allow full-speed program execution, i.e. not just a simulator. The special 8051 chip has special bond-out signals which are used to control execution as well as to obtain real-time execution traces. Available features are: program breakpoints, instruction tracing, program single-stepping, command macro language, etc..

Intel 8751 EPROM

The 8051 contains 4K of factory-masked-programmed ROM which contains the Slave software. To allow ROM user-programmable capability, Intel provides the 8751 chip which is identical in function to the 8051 except that the 4K ROM is also Erasable-Programmable-ROM (EPROM). These 8751s are used extensively in CX's developmental prototype since they allow quick turn-around time in programming the EPROMs and re-usability after erasures by ultra-violet lamps. While the 8051 ROM is for non-erasable write-once read-many-times program memory, the 8751 EPROM is spec'ed to be reusable for thousands of erase/rewrite cycles.

Subsystem Testing Procedure

Because each Slave is fairly independent of each other, each can be tested on the laboratory bench by itself for Slave - specific functions and 'house-keeping' functions such as Slave supervisor program or program segments which involve data manipulation, i.e. algorithm-oriented. This

kind of independent testing can be useful for about 50% of the program for each Slave . The advantage, of course, is that these activities take place in parallel with all Slaves being tested pretty much simultaneously.

However there are Slave functions which require co-ordination of one Slave to work with another Slave in order to complete all Slave functions. For example, in order to verify the actual paper-handling function of the Collator Slave it would be necessary to synchronize the Paper Input Slave with the Collator Slave by use of the Slaves Driver .

System Integration Procedure

After the Slaves have been unit-tested independently, the Slaves Driver is then is to bring each Slave into the system. The Paper Input Slave is the first to be controlled by the Slaves Driver since it is at the 'front-end' of the copier and has no pre-requisite. Then the Scanner Slave is added to the system to verify synchronization between feeding blank sheets and scanning of the original copy. Soon after the Process Slave is integrated into the system to check out copy image exposure.

The Fuser Slave is added next in order to produce handle-able copies. Lastly, addition of the Collator Slave completes the copier functions. This bring-up is chosen because it would minimize impact to the current configuration each time a Slave is integrated into the system.

CHAPTER 5

SUMMARY AND CONCLUSIONS

The Slaves Driver was design to provide quick prototyping of the copier Master element. The distributed processing nature of the copier design made the implementation of the Slaves Driver easier than it would be otherwise.

Current status and constraints

The Slaves Driver was used successfully to integrate the Slaves coordinating them in performing functions of a copier. It permits development of the Master element to include more elaborate functions. Most of all it allows earlier testing of the copier, i.e. earlier discovery and investigation of problems and installation of solutions and/or enhancements to the copier design.

Its constraints lie in its prototypal nature in that it is most useful in the developmental stage of copier development. As such its interface to the test operator is not always the same as that between the real copier and the customer. Therefore some of the subtle problems of human-factor engineering, i.e. man-machine interface, can't be emulated.

Technology outlook for copier

(Views expressed in this section are strictly speculations of the author and are unrelated to those of any organization)

Like the blurring line between communications and computers, the line between copier and other printing devices is more blurry than ever before. Among a few trends are:

1. More commonality among printing devices: Kodak® is getting into printers, with the current trend of corporate mergers it would be possible for companies like Xerox ® to acquire photography companies like Polaroid ®.
2. The rainbow copier: following the leads of multi-colored graphics terminals and printers the upcoming question will probably be '... but can it reproduce 4096 hues of colors?'
3. The 'smart' copier: with improvements in microprocessor power, copiers will eventually have image-enhancement capabilities similar to ones used by space agencies (e.g. NASA) 5 or 10 years ago. This will be an optional mode the user can select and will probably result in slower copying speed when used.

APPENDIX A

8051 MACHINE INSTRUCTIONS REPERTOIRE

This section lists the machine instruction repertoire for the 8051 microprocessor. The listing is organized alphabetically by 8051 mnemonics.

The following abbreviations are used in the 'Operand Format' field:

- bit - bit address expression
- code - code address expression
- data - data expression
- R - register r0 thru r7
- @R - @R0 or @R1
- # - immediate expression
- A,DPTR,C, or PC - indicated register

'Code Bytes' field indicates the number of bytes the instruction takes. 'Execute Cycles' field indicates the number of machine cycles it takes to execute that instruction.

8051 Mnemonic	Description	Operand Format	Code Bytes	Execute Cycles
ACALL	2K Call	code	2	2
ADD	Add	A,@R	1	1
		A,R	1	1
		A,data	2	1
		A,#	2	1
ADDC	Add with Carry	A,@R	1	1
		A,R	1	1
		A,data	2	1
		A,#	2	1
AJMP	2K Jump	code	2	2
ANL	And Logical	A,@R	1	1
		A,R	1	1
		A,data	2	1
		A,#	2	1
		data,#	3	2
		data,A	2	1
		C,bit	2	2
		C,¬bit	2	2

Figure 12. '8051 Machine Instructions Set' (Part 1 of 7)

8051 Mnemonic	Description	Operand Format	Code Bytes	Execute Cycles
CJNE	Compare/Jump ¬Equal	A,#,code	3	2
		A,data,code	3	2
		@R,#,code	3	2
		R,#,code	3	2
CLR	Clear	C	1	1
		bit	2	1
		A	1	1
CPL	Complement	C	1	1
		bit	2	1
		A	1	1
DA	Decimal Adjust	A	1	1
DEC	Decrement	A	1	1
		@R	1	1
		R	1	1
		data	2	1
DIV	Divide	AB	1	4

Figure 12. '8051 Machine Instructions Set' (Part 2 of 7)

8051 Mnemonic	Description	Operand Format	Code Bytes	Execute Cycles
DJNZ	Decrement/Jump <> 0	R,code	2	2
		data,code	3	2
INC	Increment	A	1	1
		@R	1	1
		R	1	1
		data	2	1
		DPTR	1	2
JB	Jump if Bit is set	bit,code	3	2
JBC	Jump/Clear Set Bit	bit,code	3	2
JC	Jump if Carry Set	code	2	2
JMP	Jump	@A+DPTR	1	2
JNB	Jump if Bit Not Set	bit,code	3	2
JNC	Jump if Carry Set	code	2	2
JNZ	Jump if ACC ¬= 0	code	2	2

Figure 12. '8051 Machine Instructions Set' (Part 3 of 7)

8051 Mnemonic	Description	Operand Format	Code Bytes	Execute Cycles
JZ	Jump if ACC = 0	code	2	2
LCALL	Long Call	code	3	2
LJMP	Long Jump	code	3	2
MOV	Move	A,@R	1	1
		A,R	1	1
		A,data	2	1
		A,#	2	1
		C,bit	2	1
		DPTR,#	3	2
		R,#	2	1
		R,A	1	1
		R,data	2	2
		bit,C	2	2
		data,#	3	2
		data,@R	2	2
		data,A	2	1
		data,data	3	2
		@R,#	2	1
		@R,A	1	1
		@R,data	2	2

Figure 12. '8051 Machine Instructions Set' (Part 4 of 7)

8051 Mnemonic	Description	Operand Format	Code Bytes	Execute Cycles
MOVC	Move Code Memory	A,@A+PC	1	2
		A,@A+DPTR	1	2
MOVX	Move Ext Memory	A,@DPTR	1	2
		A,@R	1	2
		@DPTR,A	1	2
		@R,A	1	2
MUL	Multiply	AB	1	4
NOP	No-operation	-	1	1
ORL	Or Logical	A,@R	1	1
		A,R	1	1
		A,data	2	1
		A,#	2	1
		data,#	3	2
		data,A	2	1
		C,bit	2	2
		C,¬bit	2	2
POP	Pop	data	2	2

Figure 12. '8051 Machine Instructions Set' (Part 5 of 7)

8051 Mnemonic	Description	Operand Format	Code Bytes	Execute Cycles
PUSH	Push	data	2	2
RET	Return	-	1	2
RETI	Return from Interrupt	-	1	2
RL	Rotate Left	A	1	1
RLC	Rotate Left Carry	A	1	1
RR	Rotate Right	A	1	1
RRC	Rotate Right Carry	A	1	1
SETBIT	Set bit	C	1	1
		bit	2	1
SJMP	Short Jump	code	2	2
SUBB	Subtract w/ Borrow	A,@R	1	1
		A,R	1	1
		A,data	2	1
		A,#	2	1

Figure 12. '8051 Machine Instructions Set' (Part 6 of 7)

8051 Mnemonic	Description	Operand Format	Code Bytes	Execute Cycles
SWAP	Swap nibbles	A	1	1
XCH	Exchange	A,@R	1	1
		A,R	1	1
		A,data	2	1
XCHD	Exchange low nibble	A,@R	1	1
XRL	Exclusive Or	A,@R	1	1
		A,R	1	1
		A,data	2	1
		A,#	2	1
		data,#	3	2
		data,A	2	1

Figure 12. '8051 Machine Instructions Set' (Part 7 of 7)

LIST OF REFERENCES

1. Component Data Catalog. Santa Clara: Intel® Corporation, 1982.

2. EMV-51 Emulation Vehicle User's Guide. Santa Clara: Intel® Corporation, 1982.

3. MCS-51® Family of single-chip Microcomputers User's Manual . Santa Clara: Intel® Corporation, 1981.

4. Schaffert, R.M. Electrophotography. London: Focal Press, 1980.

5. Weitzman, Cay. Distributed Micro/Minicomputer Systems. Englewood Cliffs: Prentice-Hall Inc., 1980.