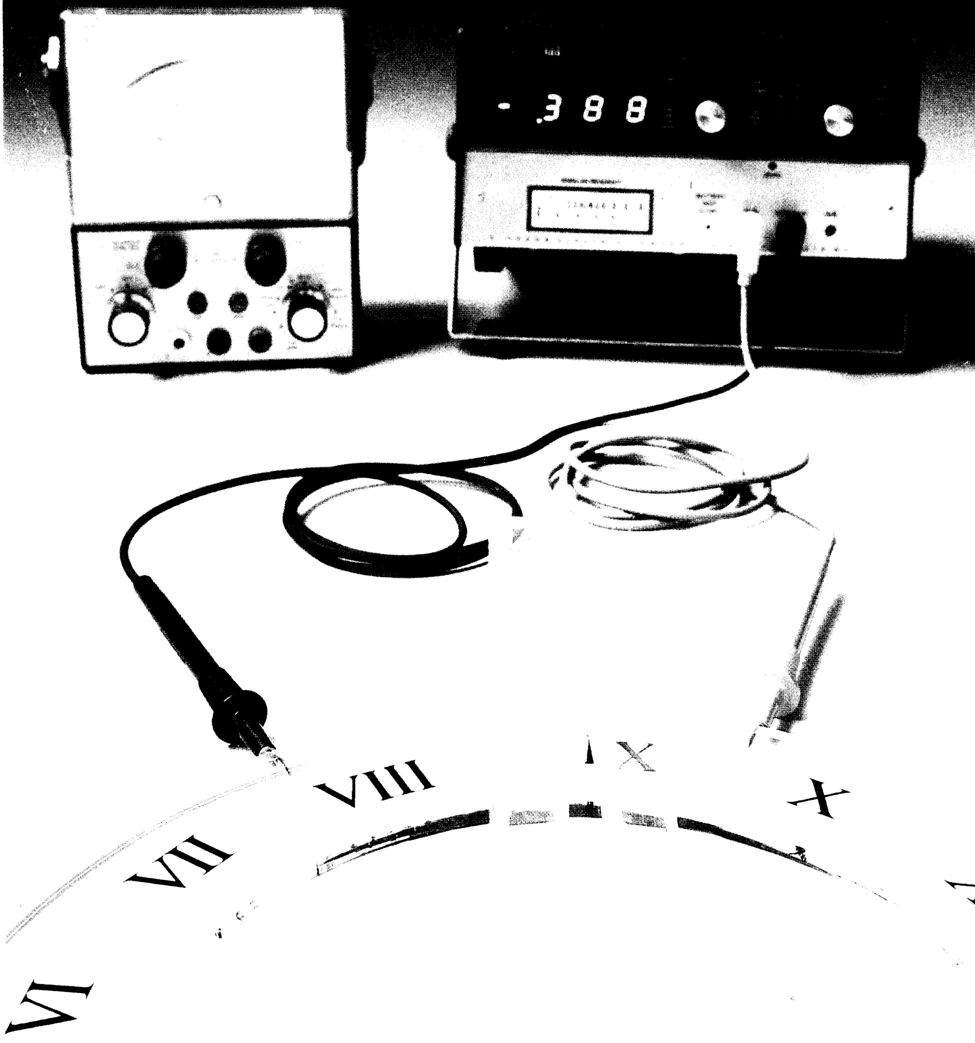
21831

Real-Time Control With the TRS-80®

Russell Genet



BLACKSBURG CONTINUING EDUCATION SERIES™
edited by Larsen, Titus & Titus

Real Time Control With the TRS-80®

by Russell Genet

Howard W. Sams & Co., Inc.
4300 WEST 62ND ST. INDIANAPOLIS, INDIANA 46268 USA

FIRST EDITION
FIRST PRINTING—1982

International Standard Book Number: 0-672-21831-3
Library of Congress Catalog Card Number: 82-50020

Edited by: *C. Herbert Feltner*
Illustrated by: *Jill E. Martin*

Printed in the United States of America.

Preface

This book is a direct result of a real problem that I needed to solve. My hobby is Astronomy, and in my back yard I built an observatory that I use to make highly precise measurements of the changing light from distant stars. The results of these observations, after suitable analysis, have appeared in papers in several scientific journals devoted to Astronomy. These form my small contribution to increasing man's understanding of the universe. The observations were initially recorded on a strip chart, and the analysis was done on a hand calculator. The problem was that reading the strip charts and analyzing the data with a calculator was not only quite time consuming, but very boring and tedious. Even worse, one did not know what the quality of the observations were until the data was reduced, and this resulted in making observations on nights when atmospheric conditions were of insufficient quality.

The obvious solution to this problem was to have a computer record and analyze the data and promptly feed back the results. Knowing that this was not a new idea, I set out to find how others had fared in connecting up computers to instruments on telescopes. What I found was startling. Past attempts to do this had been very expensive, time consuming, and sometimes complete failures. An analysis of the failures suggested that in each case one or more serious errors had been made in planning out the overall approach. An analysis of the successes, including the expensive ones, suggested certain common features in the approaches that led to the desirable outcomes. By capitalizing on the experiences of those who preceded me, I was able to develop the hardware and software for a complete system in just a few months in my spare time at a low cost. Furthermore, the system worked the first time (well, almost the first time) as intended. While I had, out of necessity, documented what I did as I went along, the idea of putting it into a book belongs to Dr. Jon Titus. Besides being a chemist, editor, and microcomputer expert, Jon is a fellow amateur astronomer.

The intent of this book is to suggest to you, the reader, how to go about planning and developing a real-time data-logging or control system in a manner that will avoid the major pitfalls that have trapped so many others. This will save you considerable time and much money. To do this, it has been necessary to take something of a "systems" approach to things, and to illustrate many of the points in a case example. While the specific hardware and software details of the case example (my astronomical observatory, of course) may not always be directly applicable to your problem, you should find that the reasoning behind the details will be applicable.

This book is divided into two major sections. The first section introduces real-time control with microcomputers, and develops a general purpose control system. While the TRS-80 Model I is used in the examples, other microcomputers with BASIC languages could be utilized with only minor modifications. A number of programming examples (all in BASIC) and actual hardware experiments are included in Section I to help develop your skills.

The second section is a detailed case example of the development of the real-time control system at the Fairborn Observatory. The main reason this case example was included was to not only demonstrate the usefulness of an evolutionary systems approach, but to demonstrate the power and utility of microcomputers programmed in BASIC to perform a serious and complex control task. Besides providing many details that should be useful in other situations, the case example clearly illustrates that by using programmable peripheral chips the hardware for a control system need not be complex, and, in fact, can be simple and low cost.

Properly approached, connecting your computer to the outside world to do the real-time tasks can be straightforward, fun, and even exciting, with ultimate success never in doubt. Improperly approached, it can quickly become a frustrating nightmare. It is my hope that the time I have spent in translating my notes and thoughts into a book will help you to get started properly in the fascinating world of real-time control with small computers.

My special thanks to Dr. Jon Titus whose many good suggestions have greatly improved this book, to Drs. Doug Hall and Ken Kissell who introduced me to serious astronomy, to Dorothy K. Mote whose role went well beyond that of a normal typist, and to Mike Genet for preparing drafts of many of the drawings and figures.

This book is dedicated to my wife, Ann, who after twenty years and five children still cheerfully puts up with my wild projects.

RUSSELL M. GENET

Contents

SECTION 1

A General Purpose Control System

CHAPTER 1

Introduction

The original real-time control system was the human being. A person is admirably equipped with various sensors and actuators, and has a central computer that is capable of sophisticated decision making and learning. Prior to the industrial revolution, people not only served as the control system but as the prime mover. Now people provide the control and let the machines do the hard work.

It was only natural that machines would evolve towards more self-control, and an early example was a weighing machine that turned off a filler nozzle when the right weight was reached. There are many reasons why it is often desirable to take at least the lower level, routine control tasks away from people and give them to machines. Among these reasons are economy, accuracy, and reliability. In some cases a speed not within human capabilities is called for, while in others, the environment precludes the use of people.

While one tends to think of industrial production when talking of real-time control, there are many other applications. Some of the most interesting are in the area of scientific and engineering experiments, where various factors must be manipulated and large amounts of data gathered and analyzed.

For many years, control systems consisted primarily of simple mechanical and electrical components, usually dedicated to specific functions. While some general purpose controllers using relays have been around for a number of years, the advent of large-scale integrated circuits and the minicomputers and microcomputers is what really started the current revolution in control systems.

Without a doubt, the most sophisticated real-time control systems are the Viking orbiters and landers, and the Voyager systems sent to Jupiter, Saturn, and beyond. These semi-intelligent robots represent the current state of the art in real-time control systems, and were able to operate and survive in a hostile environment far from the earth. While this book won't tell you everything you need to know to build your own Viking lander (and, in any event, your TRS-80® warranty is only good for Earth), you can learn enough to make your own real-time control system of considerable sophistication.

COMPUTERIZED CONTROL SYSTEMS

Many real-time control systems are not computerized at all. The typical furnace system in a house is a simple control system that does not use a computer. Rather complex control systems have been made using many relays, and various other electromechanical devices. All these systems are highly specialized however, and if one wants to make a change it must be a hardware change. The advantage of computerized systems is their flexibility and the ease with which changes or additions can be made. Other advantages of computerized control systems are their high reliability, speed, and sophistication.

As long as computers were large and expensive, their use in real-time control systems was very limited, but as they became cheap and small, they started to see increasing use in control systems. In fact, very few newly designed control systems are other than computerized systems, and noncomputer based systems will not be considered in this book.

Once one has decided to use a computer, there are still many options open. The lowest cost systems use one- or two-chip microprocessors with fixed programs stored in read only memory (ROM). Typical of such systems are microwave oven controls, automotive controls, and the "brains" of many specialized machines and instruments. In large quantities, this approach is the most economical. For a single system, the requirement to develop a machine language program and store it in a ROM is rather costly in terms of time spent, requires a good knowledge of assembly language, and can require special equipment such as the ROM programmer. If you like machine language programming and have the necessary equipment, this approach is one to consider.

On the other end of the spectrum, many minicomputers such as the PDP-11, etc., have been used in real-time control systems. These computers are fast and flexible, and are a good choice if you have enough money.

Finally, there are the relatively low cost microcomputer systems of which the TRS-80 is a prime example. For a rather low price one can purchase a complete computer system. All the development has been taken care of, and all one has to do is plug it in, turn it on, and start programming. I chose this approach because I wanted to concentrate all my efforts on the control system, not the computer. The TRS-80 was chosen because of its low price, readily available maintenance, and the availability of peripherals and software from many sources. There have been important extra benefits in selecting the world's most popular computer. Among these are many nearby experts in the local TRS-80 club, a well-written magazine (80-Microcomputing), and, of course, a large enough audience to make writing a book worthwhile!

LANGUAGES AND INTERRUPTS

As mentioned earlier, many computerized control systems run using machine language. The great advantage of machine language is its speed. The speed is usually so great that a priority interrupt scheme can be used to handle the various control functions so quickly that they appear to be occurring essentially simultaneously. These advantages are many, and are the reason that many of the professionally designed control systems run in machine language. The programming task is considerable, however, and unless one really enjoys this sort of thing, it can take most of the fun out of designing and implementing real-time control systems.

A language that has most of the speed of machine language but is somewhat easier to program is FORTH. This increasingly popular language was devised by astronomers to control radio telescopes and process the data collected from them. If you plan to do extensive real-time control with the TRS-80, you might want to consider acquiring and using this language.

The most easily learned and popular language is BASIC. It comes in two fundamental varieties. One takes a program of BASIC instructions and translates or "compiles" them into machine-language statements. The resulting machine-language program executes fairly rapidly. The other variety is generally called "interpretive" BASIC. In interpretive BASIC, each statement is broken down into a series of executable instructions, and when these are finished, the next statement is interpreted, and so on through the program. While interpretive BASIC is the slowest of all, it is the most convenient because it doesn't require compiling, it is easily edited, and, of course, it is the language resident in the ROM in the TRS-80.

In spite of its relative slowness (still a lot faster than a person!), interpretive BASIC was chosen as the language for the control system and experiments described in this book. The main reason for choosing it was that it is the easiest to use, thus maximizing the fun and minimizing the drudgery. It also opens up the exciting world of real-time control to many who would be "put off" by having to do everything in machine language or who would have to learn another language. Finally, for many control systems, it is perfectly adequate to the tasks at hand. In those cases where high speed is absolutely essential, a machine-language subroutine can be readily added to the interpretive BASIC program. Doing everything in interpretive BASIC makes handling priority interrupts rather difficult however, and as explained in *TRS-80® Interfacing, Book 2* (Titus, and Larsen, published by Howard W. Sams & Co., Inc.), the TRS-80 is not equipped to handle interrupts efficiently without some modification. An important goal of this book is to have you use the TRS-80 without modification, so no interrupts will be used. This is not too much of a handicap and does much to keep things relatively simple and straightforward, as in essence only one thing is done at a time. It is still possible, of course, to use "software interrupts" where after each little task is completed the system can check the status of various "flags" and decide what is the most important task to do next.

COMPUTER OPERATOR COMMUNICATIONS

Although computerized control systems are very smart and fast, they usually still require some overall direction from a human supervisor making high-level decisions or at least monitoring the control system. This requires that communication be established between the human and the computer. If the operator is familiar with computers, the system being controlled can be placed near the computer, and if the environment the system operates in is a benign one that will not adversely affect computer operation or reliability, then the computer keyboard and video display can be used for communications. These conditions cannot be generally met, so in many cases it is necessary to keep the computer in a "safe" environment and communicate with it remotely. Typically, one does not want to put a computer in a machine shop, cold observatory, etc. Greasy or frozen TRS-80s have been known to go whacky.

Fortunately, it is quite easy to establish remote operator/ computer communications. In very simple control systems, a few switches and light-emitting diodes (LEDs) often will suffice. For more complex systems the number of switches, LEDs, etc., can easily get out of hand, and their flexibility is limited. A generally preferable approach is to use a small remote video monitor to display the computer's questions, status, data, etc., and a small hand-held keypad for the system operator's responses or instructions. The computer's questions can be given in plain English on the monitor and the possible responses also laid out in plain English. An example would be: WOULD YOU LIKE TO REPEAT THIS OPERATION? YES PRESS 1, NO PRESS 2. This approach to computer/operator communications is very flexible (only requiring software changes in the BASIC program), minimizes operator training and errors, keeps the computer in a safe, benign environment (usually unattended), and is almost universally applicable. We will use this approach exclusively.

PROGRAM MODULARITY

Even when using interpretive BASIC, the development of a program for a computerized real-time control system can be a considerable undertaking. It is best to approach this task by breaking the program into natural functions or tasks and programming and debugging these one at a time. While this might not be necessary for the simplest control system, we want to be able to design and successfully implement systems of considerable complexity and sophistication so we will take a modularized programming approach from the beginning.

To coordinate or "call" the different program modules, we will have a master module or main program. At its heart will be the "Main Menu." Rather like a menu at a restaurant, our main menu will display on the remote video monitor all the major things that the control system is able to do, and the operator will respond by entering his selection on the remote keypad. The *main menu* at an automated oil refinery might look like that shown in Example 1-1.

After a choice is made, then a secondary menu might be displayed for further, more detailed choices, or execution might begin immediately. After a task is completed by the computer, it can be programmed to always come back to the main menu so that the operator can give it further instructions.

Each of the modules (menu items) can be developed and checked out independently. As the system evolves, one can easily add new items to the menu or delete those no longer required.

Example 1-1. Main Menu for an Oil Refinery

```
                    Main Menu (Choose One)

 1. Refinery Start Up Sequence
 2. Check Distillator Status
 3. Check Crude Oil Supplies
 4. Activate Emergency Fire Procedures
 5. Refinery Shutdown Sequence
 6. Control System Self-Test Procedures
 7. Instructions on How To Operate the System
```

MAKING IT EASY FOR THE OPERATOR

While the person who designed the control system is usually quite capable of operating it, the hallmark of a really good design is a system that can be operated by a person who knows absolutely nothing about computers (may not even like computers, let alone trust them). We are, of course, going to do a very good design, so good in fact that when our Aunt Nelly visits, she will be able to operate the system after a brief demonstration (we won't even tell her about the computer hidden away in a warm room).

This can be accomplished by liberal use of instructions and by keeping everything in carefully worded, plain English. If a system doesn't pass the "Aunt Nelly" test, then it should be reprogrammed until it does.

HARDWARE

The bare TRS-80 was not designed to be directly connected to stepper motors, relays, light sensors, and the various other goodies in our control system, so we must design and build an interface for this. There are a number of considerations to be made in selecting our approach to the interface.

First, it would be helpful if the interface were flexible so that when we wanted to make changes in the control system (or even go to another control system entirely) we did not have to make lots of hardware changes. Second, we would like to keep the cost and complexity of the interface to a minimum. Finally, we would like the interface to be easy to build yet highly reliable.

The approach I feel that best meets these conditions is to split the interface into two parts. One part will be placed next to the computer and will be very nonspecialized and suitable for almost any control system requirement. The other part will be placed by the system to be controlled and will be somewhat customized to the particular control system in question. The unit near the computer will plug into the computer bus, of course, and will handle the high-speed communication with the computer. It will be connected to the other unit over "slow" speed ribbon cable. The remote unit will contain the electronics to condition the signals for the various sensors and actuators, as well as the remote keypad. An overall block diagram is shown in Fig. 1-1.

To minimize the cost and complexity of the computer interface, we will avoid using simple, small-scale chips, but will instead use rather fancy large-scale integration (LSI) chips that were designed expressly for microcomputer interfacing. In this way we can benefit from all the hard work that the designer put into the chip—some chips are almost computers in their own right. By using chips with three-state outputs, we can connect them directly to the TRS-80

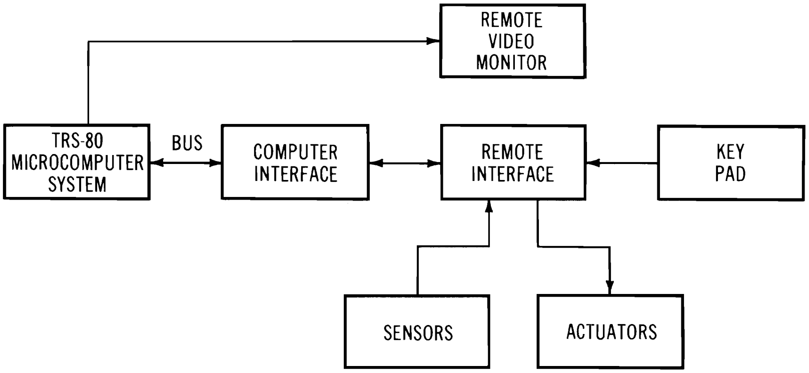


Fig. 1-1. Block diagram of a control system.

bus without any buffering. Best of all, these LSI *interface chips* are programmable and can flexibly perform many different functions when you make the appropriate software changes in the control program. At no time will we be the least concerned about what is going on inside these LSI chips, nor for that matter, what is going on inside the TRS-80 either! On the other hand, we will worry *a lot* about what goes in and comes out of the chips.

There are a number of ways to mount these chips, connect them, and connect the computer interface to the computer. The fancy commercial interfaces put the chips on printed circuit boards, and plug the boards into a card cage. Since a good card cage would cost more than the entire system, we'll take a cheaper approach. A very easy approach from a wiring viewpoint is to plug the chips into breadboard sockets and make the wiring connections. This is a workable approach, but as the number of connections grows it gets rather messy; one tends to worry about the reliability of the connections and breadboards aren't exactly cheap or compact. As a sort of middle-ground, a wire wrap approach on a perforated grid board, or perfboard, was selected. This provides us with a permanent unit with reliable connections that is low in cost and relatively easy to build. For those not familiar with wire wrapping, instructions will be given at the proper point.

Every attempt has been made to use low-cost, easily obtained parts throughout. Most of the parts can be picked up at your local Radio Shack store. High-priced sensors and actuators have not been used (although they are sometimes described). This is in line with the author's conviction that real-time control is neither difficult, nor does it need to be expensive.

APPROACH

The approach taken in this book is to learn by doing. By understanding how one particular control system was designed, built, and programmed, you will be able to adapt the techniques to your own particular situation. In most instances, you will be able to use most of the computer interface as given, as it is very generalized and flexible. The details of the remote interface will depend on the particular sensors and actuators in your control system, but a sufficient variety of some of the commonly used sensors and actuators is given in the examples so that you can readily modify things to fit your own situation.

Wherever possible, the reasons behind the various approaches taken are given, and the alternatives discussed. By getting a feel for the possibilities open to you and the sort of trade offs involved in choosing an alternative, you will be in the best possible position to successfully design and implement your own real-time control system.

Finally, while the automated photometer for observing eclipsing binary stars used as an example in this book may be the *last* control system you would want to build, *a nontrivial example of a useful and proven control system is very necessary to illustrate some of the subtle yet important aspects of control system design.* So while your application will certainly be entirely different, it will still be necessary to dwell on some of the fine points of this particular application so that you will be able to do the same for yours. So, for now, pretend that astronomy is your true love, and that you would rather stay up all night observing eclipsing binaries than anything else (well, almost anything else).

CHAPTER 2

Building a Simple Control System

The simplest control system is shown in Fig. 2-1. Push the button and the light will come on; release the button and the light goes off. However, as we are interested in computerized control, we must take a somewhat different approach to this simplest of all control problems. When we push the button, it will have to tell the computer that we want the light turned on, and the computer will then have to turn on the light itself. Furthermore, we need to do this in a way that will allow us to expand what the computer does in situations of great sophistication and complexity. Properly done, the simplest control system will get us well along the road to our goal of a sophisticated, general-purpose real-time control system.

To get in the proper spirit of things, we can rephrase the simplest control system in more general language. The switch or "button" is a remote sensor or input, while the light is a remote actuator or output. We will assume that there can be many sensors and actuators in our system, and although we will connect only one of each to begin with, we will do it in a way such that many more (even hundreds) can be easily added later. The relationship between the sensor and actuator (input and output) can be thought of

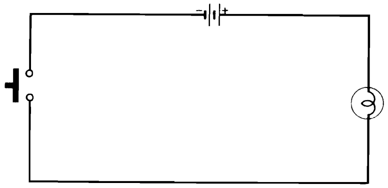


Fig. 2-1. The simplest control system.

as the *control algorithm* or *control program*. This is shown in Fig. 2-2.

One important feature of the system shown in Fig. 2-2 is the capability for almost unlimited addition of sensors and actuators. One could, of course, just push a button on the TRS-80 keyboard and use the cassette control output of the computer to actuate the light, but this would not have any growth capability.

Another important feature of the system shown in Fig. 2-2 is that the link between the inputs and outputs, the algorithm, can be changed by merely changing the control program. We could, for instance, have the light stay on for 2 seconds every time we push the button, or we could have the light blink out the square of the number of times we pushed the button in rapid sequence, or . . . .

It will be our goal in this chapter to build an interface for the TRS-80 and write a control program that will cause an LED to light when we push a switch. This may not sound like much, but actually once this has been done successfully and understood, you will be well along towards a sophisticated real-time control system. This is so because the approach we will take to do this simple control task is very general and powerful.

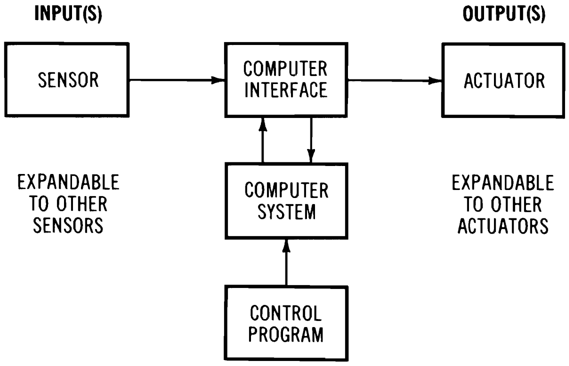


Fig. 2-2. Block diagram of a computerized control system.

HOW THE TRS-80 SPEAKS TO THE OUTSIDE WORLD

Microcomputer designers have taken two general approaches in allowing for communications between their systems and the outside world. In the first approach, they make no special provisions to speak of and the microcomputer user (that's us) has to make the outside world look like memory locations to the computer. This is done by completely decoding the address lines and using the READ and WRITE commands going to and from the memory in the computer. To pass numbers to and from the outside world in BASIC requires the use of PEEK and POKE statements with appropriate memory locations. This approach is called "memory mapped" input/output (I/O).

In the second approach, specific provisions have been made to facilitate communications with the outside world. The TRS-80 has been designed to have 255 "ports" through which communications can be channeled to and from the outside world. The BASIC language of the TRS-80 has INP and OUT commands to facilitate communications with the ports, and the TRS-80 bus connector has $\overline{\text{IN}}$ and $\overline{\text{OUT}}$ lines to signal when it wants to receive numbers or put them out to these ports. This is all quite convenient, and is one of the reasons I chose the TRS-80 for my system.

Even with the convenient TRS-80 layout there is a complication that must be taken into account, and this is that all the ports have to share the same data bus. If more than one port were connected to the data bus at the same time, then there would be total confusion, so provisions must be made to only connect one at a time. Also, as the data bus is used to both bring data into the TRS-80 and put it out (i.e., it's bidirectional), there needs to be some control as to which direction the data is moving.

Fig. 2-3 shows how all this is taken care of. Let us follow through a typical sequence of events. Assume that the control pro-

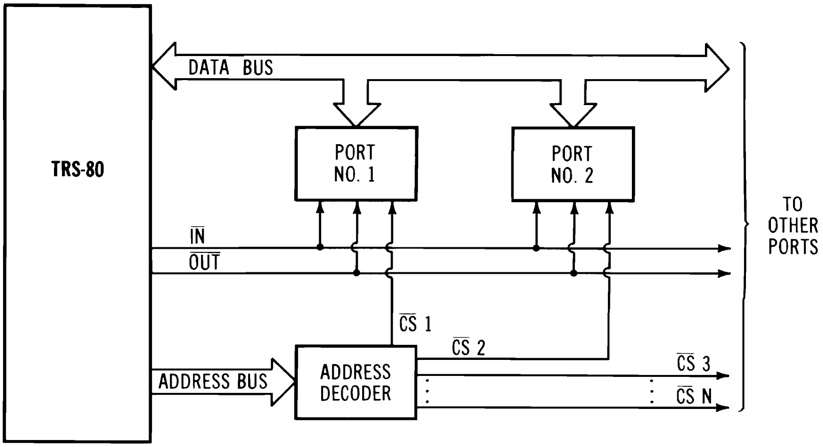


Fig. 2-3. TRS-80 port connections.

gram asks for an input from Port No. 1. First, the TRS-80 will put out a 1 (binary 00000001) on the lower 8 address lines. The address decoder responds by bringing the $\overline{\text{CS}}$ 1 (chip select) line low and this, in turn, causes Port No. 1 to connect itself to the data bus (all others remaining disconnected). The TRS-80 then brings the $\overline{\text{IN}}$ line low, and this signals the selected port to input its data. After waiting for a few microseconds, the TRS-80 stops sending out the input request and the address for Port 1.

Now let us assume that the control program wants to put some data out to Port No. 2. First, the TRS-80 will put out a 2 (in binary 00000010) on the lower 8 address lines. The address decoder will bring $\overline{\text{CS}}$ 2 low, and Port No. 2 will connect itself to the data bus. The TRS-80 will then place the data on the bus and bring the $\overline{\text{OUT}}$ line low. Port No. 2 has to latch onto the data quickly because in microseconds the TRS-80 is off doing other things.

Doing It in BASIC

There are two commands in Level II BASIC that allow you to communicate directly with the ports. Consider the following line:

```
10 OUT 128, 15
```

This line, when encountered, will do the following: First, it will put out the number 128_{10} on the lower 8 address lines as an 8-bit binary number. Electrically this will be:

A7	A6	A5	A4	A3	A2	A1	A0
1	0	0	0	0	0	0	0

The number 15_{10} will then be placed on the data lines:

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	1	1	1

Finally, the $\overline{\text{OUT}}$ line will be brought low ($\overline{\text{OUT}} = 0$). Remember that a logic zero is approximately zero volt and a logic one is between +2.8 and 5.0 volts.

This all happens in a small fraction of a second, so our interface must be able to very quickly activate the correct port and latch the output as the computer immediately moves on to other things. All of our communication from the computer out to the "real" world will use this one simple command. Its general form is OUT A, B, where A is the port address and B is the decimal equivalent of the 8-bit number being sent out.

The command to get things into the computer is even simpler. An example is:

```
20 C = INP(16)
```

This causes 16_{10} to be placed on the address lines (lower 8, A7–A0):

```
A7  A6  A5  A4  A3  A2  A1  A0
0   0   0   1   0   0   0   0
```

The $\overline{\text{IN}}$ line is then brought low ($\overline{\text{IN}}$ = 0), and whatever is on the data lines at that exact instant will be pulled into the computer and assigned as the value of the variable, "C," an integer which will always be between 0 and 255 inclusive. All our communications coming in from the real world will be via this one simple command.

Peripheral Interface Chips

There are a number of LSI chips available that were designed to work with this port scheme that can also be connected directly to the TRS-80 bus. Such a direct connection requires that when not in use these chips automatically disconnect themselves from the bus, and also requires that the chips "know" when to put data onto the bus or take it off. Such chips are often called three-state (data in, data out, or disconnected) LSI peripheral chips. Many of these chips can perform different functions under computer control, and some are actually small computers in themselves—these are programmable peripheral interface chips (LSI three-state, of course). Since one of these devices can replace a whole board full of small-scale TTL chips at a fraction of the cost, and are much more flexible and easy to use, we will use them exclusively.

These programmable interface chips often combine several ports into a single chip. The 8255 PPI (programmable peripheral interface) chip, for instance, has three ports that can be used for either input or output. A fourth port address is used to control the 8255 PPI chip itself. Some microcomputer bus-compatible A/D converters can handle up to 16 different ports/signals all in one chip. It is customary to use the lowest of the address lines to select ports within a single chip. We will reserve address lines A3, A2, A1, and

A0 for selection of ports within chips. This leaves address lines A7, A6, A5, and A4 for the selection of the chips themselves. Our interface is thus organized into 16 major port groups, with each port group in turn having up to 16 ports of 8 bits each.

Some Circuit Details

Referring to Fig. 2-4 you can see that address lines A7, A6, A5, and A4 are connected directly from the TRS-80 bus to the 74154 chip. The 74154 decodes the various patterns of logic 1s and 0s present on these address lines, generating a 1 out of 16 output that corresponds unfailingly to the pattern present. The one active output will be a logic 0, while all others are logic 1s. Each one of these 16 lines can be connected to a peripheral interface chip to activate it at the proper moment with a chip select ($\overline{\text{CS}}$) signal. As the first experiment uses only one interface chip, only one of the 16 $\overline{\text{CS}}$ lines will be used, leaving 15 for future expansion.

Note that in Fig. 2-4 that address lines A1 and A0 are tied directly from the TRS-80 bus to the 8255 PPI chip. This allows direct computer selection of one of the three 8-bit ports on the 8255 or the control "port" that programs the 8255 itself.

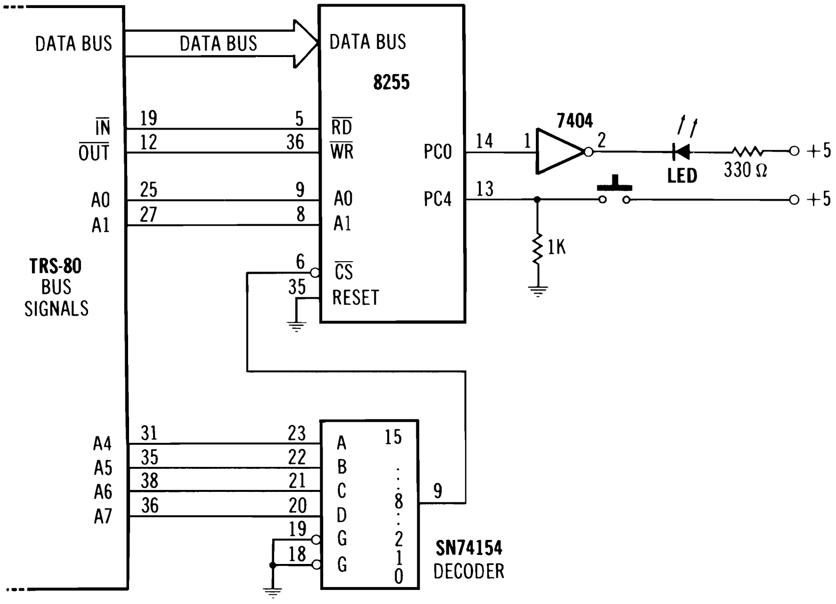


Fig. 2-4. Schematic diagram for a simple control system.

Note that the data and $\overline{RD}$ and $\overline{WR}$ signals for the 8255 are connected directly to the $\overline{IN}$ and $\overline{OUT}$ lines on TRS-80 bus. It is necessary to connect the RESET pin on the 8255 to ground. At the beginning of each of your control programs you will have to tell the 8255 how you want it to be "configured" (i.e., which ports will be input and which will be output), so you will not need or want to use the reset input.

Note also that the 74154 is constantly generating $\overline{CS}$ signals whether or not the computer is asking for an input or output from one of the ports. While one could OR the $\overline{IN}$ and $\overline{OUT}$ signals and use this to turn the 74154 on only when an input or output was requested, this isn't really necessary as the peripheral interface chips are only activated when they receive a $\overline{CS}$ AND either an $\overline{IN}$ or $\overline{OUT}$ request.

As the output current capability of the 8255 is not sufficient to drive an LED, a 7404 hex inverter was added, but you will use only one section, leaving five for other things. An LED and its current-limiting resistor and a push-button switch complete the circuit.

CONTROL PROGRAM

With the interface circuit understood (more or less), we need to write a control program for the simple control system. The control algorithm is simple, when we push the button we want the LED to light.

Addressing the 8255

The first task will be to figure out the port addresses of the 8255 and discuss what they mean. To generate the $\overline{CS}$ signal for 8255 No. 1 (there will be more 8255s later), we use the truth table for the 74154 shown in Table 2-1. Since we want output No. 8 on the 74154 to be low to generate the $\overline{CS}$ signal for the 8255 chip (and the rest of the outputs high), this means that the D input on the 74154 must be high, while inputs A, B, and C are low. As these are connected directly to the TRS-80 bus, this means that address bus signal A7 must be high, while A6, A5, and A4 must be low. Since address bits A2 and A3 are not connected to anything yet, we do not care what they are and they are noted as "X" or don't care states. Address bits A1 and A0 have special meaning to the 8255 as shown in Table 2-2.

For example, if we wanted to address Port A on the 8255, we

Table 2-1. 74154 Truth Table

Inputs						Outputs															
G1	G2	D	C	B	A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L	L	L	L	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	L	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	H	L	H	H	H	H	H	L	L	H	H	H	H	H	H	H	H	H
L	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H
L	L	H	L	L	L	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H
L	L	H	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H
L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H
L	L	H	H	L	L	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H
L	L	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L
L	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	L	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H

Table 2-2. 8255 Port Addresses

A1	A0	8255 Status
0	0	Port A
0	1	Port B
1	0	Port C
1	1	Internal Instruction Register

would need A7=1, A6=0, A5=0, and A4=0 and both A1 and A0 (from Table 2-2) would have to be logic 0. Since address bits A3 and A2 have not been used in the addressing scheme for the 8255 chip, they can be either logic 1s or 0s. To avoid confusion, we have preset them to logic 0s. The 8-bit address (A7–A0) would thus be 10000000_2, or 128_{10}. Thus, Port A on the 8255 is Port 128 from the viewpoint of the TRS-80. Port B on the 8255 is 129, Port C is 130, and the internal instruction register on the 8255 is 131.

The first line in the control program will be used to configure the 8255; we know that it will look like:

```
10 OUT 131, X
```

where X is the decimal number that represents the binary bits that are used to configure the 8255 and 131 is the port number for the internal instruction register.

Control Word for the 8255

The control word (8 bit number) used to configure an 8255 can be determined from the information in Fig. 2-5.

For our simple control system we will be using only Mode 0 of

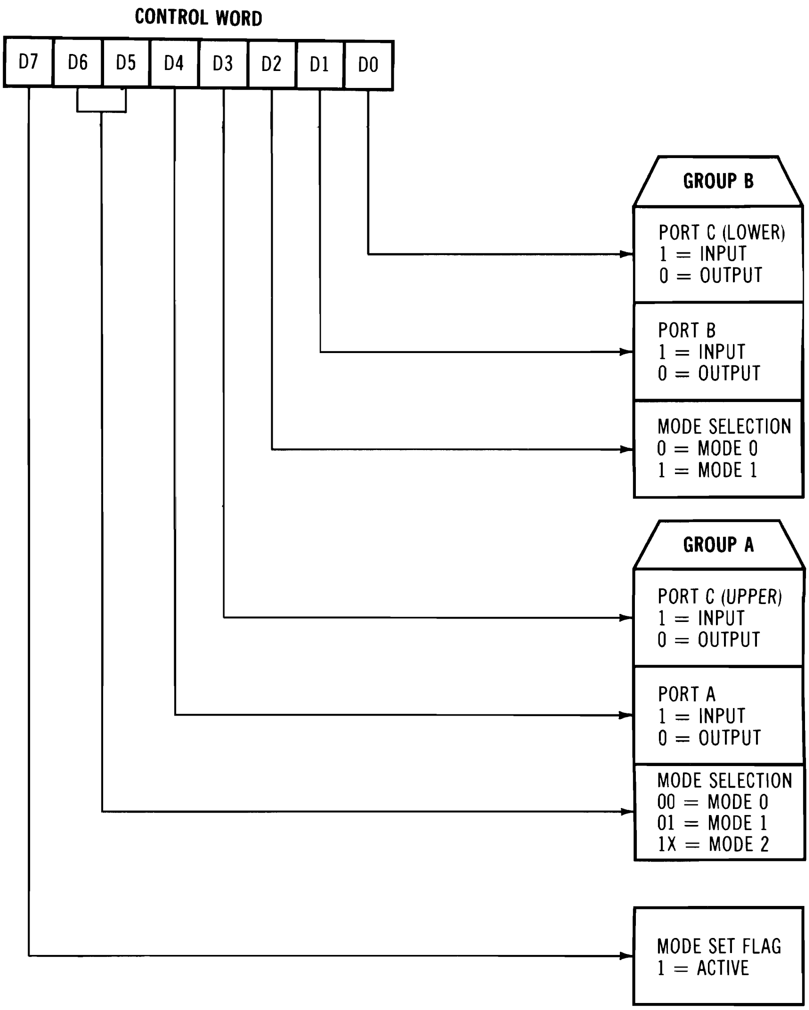


Fig. 2-5. The 8255 control word. (Courtesy Intel Corp.)

the 8255, and we desire all the ports to be in the output mode except Port C (Upper, C7–C4) which we want in the input mode to sense our switch. Note that Port C can be split into four lower (C3–C0) and four upper (C7–C4) bits, a handy feature when one does not need a full 8-bit port. In our case, we will use just one bit (PC0) from the lower half of Port C for the LED, and one bit (PC4) from the upper half of Port C for the switch. While it would have been easier to use one bit from Port A and one from Port B, it will be much more instructive to use the bits in Port C, and our approach will be useful later on. The control word we want, then, is 10001000_2, or 136_{10}. This places all the ports in the output mode except for Port C (upper) which is in the input mode. The first line of our control program is thus:

```
10 OUT 131,136
```

Inputting From the 8255

Next, we need to check and see if the switch has been depressed by the system operator. To do this we need to input from Port C on the 8255. Using Tables 2-1 and 2-2 we can determine that the address for Port C on the 8255 is 10000010_2 or 130_{10}. If we let the variable, A, be the decimal equivalent of what the computer inputs from Port C, then we can write our next program line as:

```
20 A = INP(130)
```

When we push the switch, closing the contacts, it will make Port C, bit PC4 a logic one and A will equal 00010000_2 or 16_{10}. When the computer sets bit PC0 in the lower portion of Port C to a logic 1 to turn on the light, the value at Port C would now be read in as 00010001_2 or 17_{10}. To keep the lower part of Port C from interfering with our reading of the upper part of Port C, or to keep other bits in the upper part of Port C from interfacing with the single PC4 bit we are interested in, we can mask out all of the bits except the PC4 bit. This can be done with the logical AND statement in Level II BASIC. We write:

```
20 A = INP(130) AND 16
```

To see how this works, recall that for the result of an AND operation to be "true" or logic 1, both of the inputs must be 1. If either or both are 0, then the result is 0. Consider the following example:

PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	
0	0	0	1	0	0	0	1	Actual port value
0	0	0	1	0	0	0	0	Mask 16_{10}
0	0	0	1	0	0	0	0	Result of AND

When the actual value is "filtered" through the mask (ANDed with it), only the PC4 bit can come through. Thus, there are only two possible results, namely 00000000_2 or 00010000_2. In decimal this would be 0 or 16.

Let us now assign the variable "B" to = 0 if the switch is open and = 1 if the switch is closed. We can then write

```
30 IF A=16 THEN B=1 ELSE B=0
```

So we can see what is happening in our control system we will write:

```
40 PRINT INP(130), A, B.
```

Outputting to the 8255

Now we need to turn the LED on if B=1 and turn it off if B=0. To do this, and also to demonstrate another important feature of the 8255 in control systems, we will use the bit set/reset feature of Port C. This feature allows us to set or reset individual bits in Port C without disturbing the status of the other bits. This handy feature reduces the computer software overhead needed to keep track of things. To use it we address the 8255 just as we did when we configured it (address 131_{10}), and then use the information in Fig. 2-6 to determine the proper bit set/reset control word. To turn on bit PC0 we have a control word of 00000001, while to turn it off we have 00000000. These are 1 and 0 decimal, of course, and conveniently correspond to our values of "B." Thus, we can write:

```
50 OUT 131, B
```

Finally, having turned the light on or off or having left it as it was, we need to go back and check to see if the switch has changed. For this we can write:

```
60 GOTO 20
```

The entire control program is listed in Example 2-1.

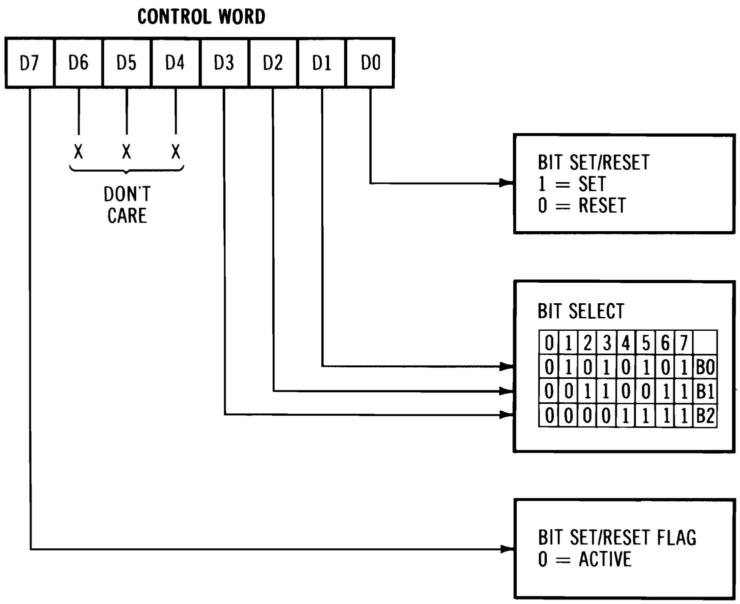


Fig. 2-6. The 8255 bit set/reset control (Courtesy Intel Corp.)

Example 2-1. Simple Control Program

```
10 OUT 131, 136
20 A = INP(130) AND 16
30 IF A=16 THEN B=1 ELSE B=0
40 PRINT INP(130), A, B
50 OUT 131, B
60 GOTO 20
```

EXPERIMENT NO. 1
THE KILOBUCK LIGHT SWITCH

Purpose

The purpose of this experiment is to give you some practical experience in wiring up an interface, connecting it to your computer, and operating a real-time control system.

Discussion

This will be your most difficult experiment. If you can get the light to come on when you push the button (assuming you haven't cheated and connected them in series with a battery), then you are well on your way towards mastering real-time control systems. All of the rest of the experiments will be mere minor variations on this major theme.

STEP 1: Mounting the Components

The first task is to mount the components. They can be mounted on an integrated circuit perfboard (Radio Shack part No. 276-1396 or equivalent) which is just the right size to later place in one of the attractive, but low-cost Radio Shack cases. Six threaded one inch long spacers can be used to hold the perfboard up off the work surface so the pins on the DIP sockets are not bent or shorted out. Two 40-pin, one 24-pin, and two 14-pin DIP sockets for wire wrapping are mounted approximately as shown in Fig. 2-7. Also, two binding posts (one red and one black) are mounted as shown.

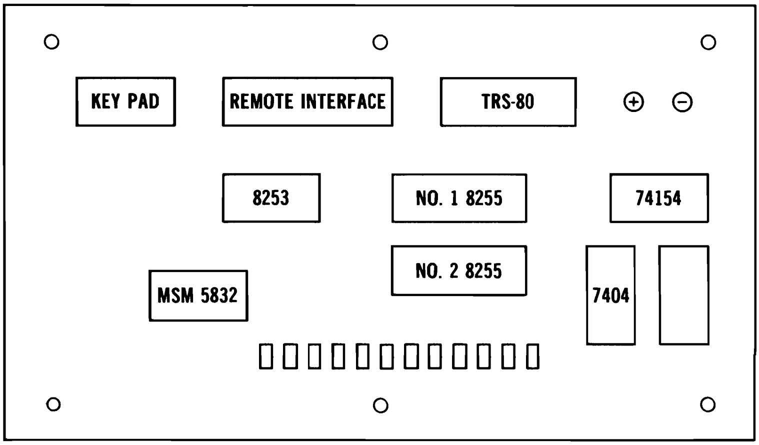


Fig. 2-7. Component mounting (top view).

STEP 2: Wire wrapping

Wire wrapping is used almost exclusively because it is very fast and easy to do, and gives a reliable product that does not take up much space. A Radio Shack battery-powered wire wrapping gun was used to make the connections. The wire was first placed between the points to be connected, allowing 1 inch on each end for stripping and 1 inch for a little slack, and then cut. After stripping each end, the wire is inserted in the tool, including about ¼ inch of the insulated portion, placed over the pin, and the gun is held firmly and trigger pressed. Zip, and you have a nice connection. You will be amazed at how easy it is, and wonder why you hadn't done it before!

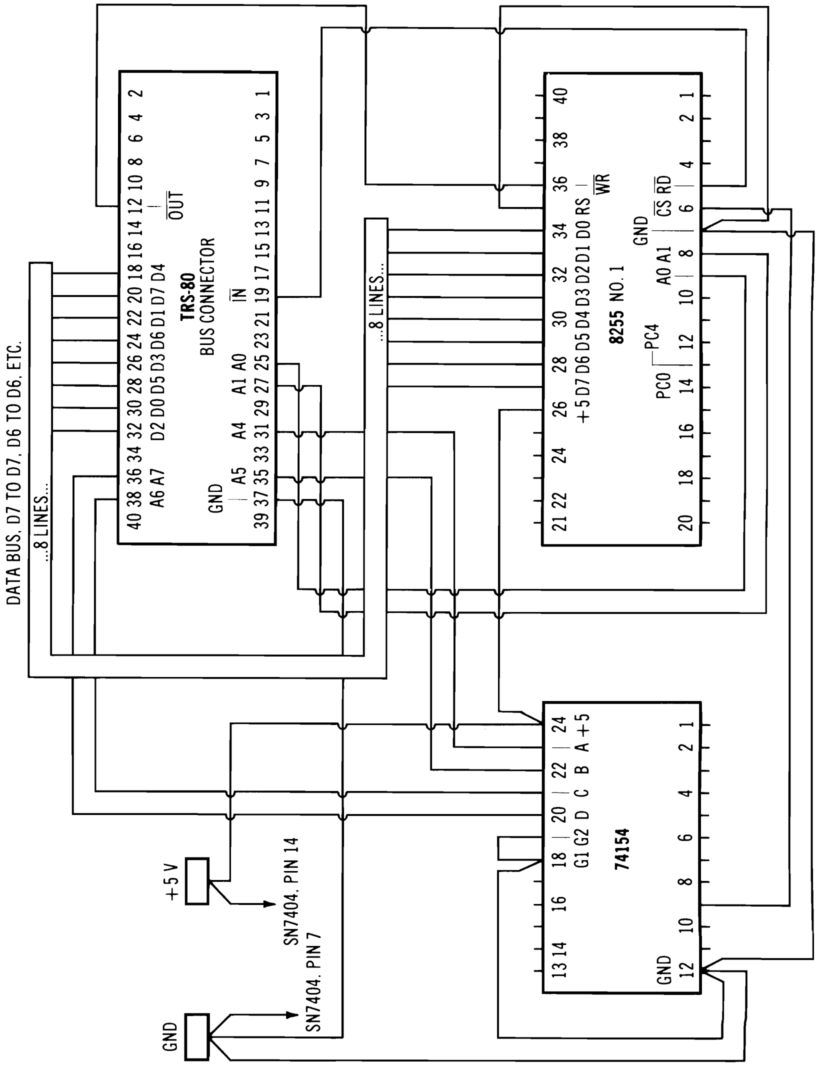
After each wire is connected as shown in Fig. 2-8, it helps to place a checkmark on each end so you won't forget what has and has not been connected. The wires going to the binding posts will have to be soldered to them, of course.

A lot has been written about the layout and routing of wires. Some writers suggest going the shortest path with no slack in the wires, while others like everything in neatly laced bundles. I prefer something about halfway between the two, a method sometimes referred to as "rat's nest." While perhaps lacking esthetic appeal, I tend to feel that a good looking project is one that works and is mounted in a nice case with a good panel layout.

STEP 3: Connecting Up the Computer

For the TRS-80 programmer/owner, connecting the computer to something other than authorized Radio Shack peripherals (or proven peripherals from other companies) may seem like a traumatic or even dangerous thing to do. Rest assured that there is nothing that you can do that will do anything worse than crash a program. You can't damage the computer no matter now many mistakes you make in wiring. I regularly wire up my equipment while the computer is on and connected up and, in fact, use the computer to check out my wiring as I do it.

To connect the interface circuits to the computer you will need a 40-pin edge connector (Radio Shack part No. 276-1558 or equivalent), a piece of 40-conductor ribbon cable (Radio Shack part No. 278-771 or equivalent), and a 40-pin DIP plug connector. It doesn't hurt to write "this side up" on a piece of tape so you will always know which way the edge connector goes. If you can have the connectors put on the cable for you by a friend or local electronics firm, it may save you trouble later; if these facilities are not available you can do it without special tools. Carefully line up the cable by putting it in a vise to punch the little pins through the cable. To keep from bending the DIP connector pins in the vise, let them sink into a piece of ½-inch balsa wood. The cable has to be pushed all the way down on to the pins or a poor connection will be made causing strange things to happen. If the equipment should stop working later on, consider this as a likely reason.



(A) Overall wiring diagram.

Fig. 2-8. Wiring diagram for experiment No. 1.

STEP 4: Power Supply

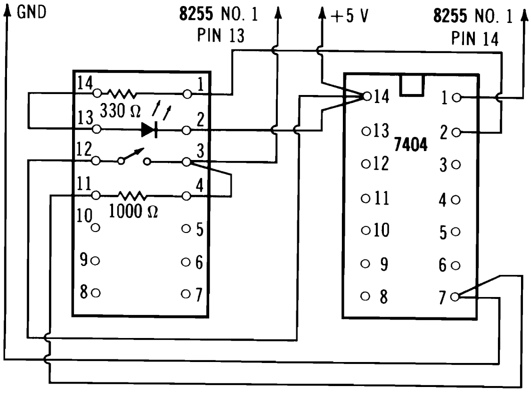
The TRS-80 has just enough power to take care of itself, so a separate 5-volt power supply for the interface is required. Any well-regulated supply that provides 5 V dc at several amps will do. Do not try to get by with a small, underpowered supply that just puts out a few hundred milliamps.

STEP 5: Run It

Connect the computer, power supply, sensor (switch), and actuator (LED) to your computer interface. Turn everything on, key in the program, type in RUN, and hit ENTER. Now, ignoring the computer (as later on it will be in the next room or even in a different building), press the button (switch) and the light should come on. If it does, then you definitely do first class work. If it doesn't, then join all the rest of us and start troubleshooting.

STEP 6: Troubleshooting

Troubleshooting is something of an individual art, but for what it's worth, I'll mention how I go about it. To make troubleshooting interesting, I usually only use a multimeter set on the 10 V dc scale. In the circuit for Experiment No. 1, I would first check to see that there was power at each of the chips. Then I would check to see if the LED was reversed. One can then write simple programs to check various functions. For instance, to see if the 74154 is doing its job, the following program will work:



(B) Partial wiring diagram for 8255 No. 1.

(Power connections not shown for clarity.)

```
10 OUT 128, 1
20 GOTO 10
```

Key in the program, type in RUN, but do not press ENTER yet. Connect a voltmeter between ground and pin 9 of the 74154. The voltmeter will read somewhere around 4 volts, indicating that address 1000XXXX is being used every so often as the computer is just sitting there. Now press ENTER while watching the voltmeter closely. If all is well, it should drop a wee bit as the program is forcing address 1000XXXX to be used more than it normally would be.

To check the operation of the 8255, one could write:

```
10 OUT 131,144           (control word configuring all ports
                         as output except Port A as input)
20 PRINT INP(128)        (Reads Port A)
30 GOTO 10
```

One can then connect an alligator clip to +5 V dc, and with the program running touch the pins for PA0, PA1 . . . PA7. On the video monitor one should correspondingly see 1, 2, . . . 128 displayed. In my case all the inputs to the 8255 worked, but none of the outputs worked. After fussing around for an hour or two, I gave up and went to bed. The next morning the first thing I thought of while sipping my tea was the reset pin on the 8255. Grounding it cured all my problems. In troubleshooting, each case is unique, of course, but so far I haven't run across a case I couldn't solve with just a voltmeter and simple little 3 or 4 line program. It never hurts to bring in a knowledgeable friend, of course!

What You Should Get

Assuming that you have everything going now, with the program running, press the button in and quickly release it and then very quickly hit the BREAK button on your keyboard. You should get something like this:

```
 0     0     0
 0     0     0
16    16     1
17    16     1
17    16     1
 1     0     0
 0     0     0
 0     0     0
```

What is happening is that the TRS-80 senses that the switch is closed, then it turns on the light; it senses the switch is open, then it turns off the light—third line (16 16 1) is the moment when the switch was closed but the light not yet turned on. No control system responds instantaneously, of course, so perhaps they should be called "almost-real-time control systems." In the case of Level II BASIC, not noted for its high speed, and especially when there is some display or printing in the process (or loop), then the delay might even become noticeable. For many, even most control situations, a delay of $^1/_{10}$ second is not very important.

Having successfully built your first computerized control system, the next tasks will not be to build fancier control systems, but instead we will turn our attention to improving our ability to communicate with the computer from our presumably remote control position.

CHAPTER 3

Remote Keypad and Video Monitor

To establish communications between the computer and the control system operator, we decided in Chapter 1 that the operator (out in the dirty workplace or cold observatory, etc.) would be remote from the computer itself. In addition, we decided that we wanted to minimize the number of switches and displays. Finally, we wanted to communicate with the control system operator in plain English, so the operator would not have to be trained in the use of the computer, and, in fact, would not have to know that the control system was even run by a computer at all.

The easiest way to meet these requirements is to use for all operator inputs a single hand-held keypad, and to transfer information to the operator via a single video monitor. This approach not only eliminates a maze of special switches and display indicators, but changes in the communication between the operator and computer do not involve any hardware changes, only software changes. This makes evolution of the system towards increasing operator convenience and clarity easier to implement.

COMPUTER/OPERATOR COMMUNICATIONS

Computers are incredibly fast, but they are not very bright and are rather rigid in their approach to things. The human operator is comparatively slow, but much more intelligent and flexible, although perhaps somewhat error prone. A good real-time control system tries to take advantage of the best of both worlds. This requires coordinated communications between the computer and the operator. An effective approach to this task is shown in Fig. 3-1. To understand how this system works, we will go through the process one step at a time.

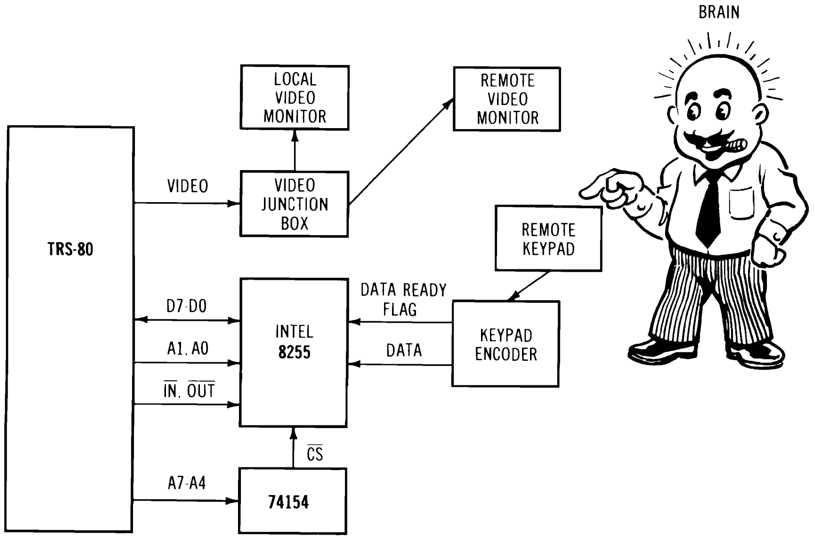


Fig. 3-1. Computer/operator communications.

The computer always initiates the dialog by a video display. The display shows up on the local video monitor (which normally no one will be watching) and also on the remote video monitor. The display would normally ask (in plain English) for a choice to be made by the operator, and would number the choices. In essence the computer says, "Master, I have finished the last thing you told me to do, here are the things I can do next. Which one do you want me to do?" Sometimes the computer will just ask for information. In any event, as soon as the request is made, the computer starts checking for the operator's response. This is done by checking through the 8255 the status of the DATA READY flag. If it were important that the operator respond quickly, the computer could ring a bell, and if lack of a prompt decision would be life threatening, the computer, after a short wait, could have a default option it could go to. (Isaac Asimov's Laws of Robotics could apply here.)

In the case of my observatory, the computer never rings any bells or takes any action on its own, but just patiently keeps checking for my response.

The next step in the communications process is that the operator sees the computer's request. The operator's computer (brain) decides on the appropriate course of action and commands the operator's finger to push the appropriate button on the remote keypad. Once the key is pushed, the keypad encoder activates the DATA READY flag, and the next time the TRS-80 checks this flag through the 8255 interface it knows that it can now read the data and does so.

Why couldn't the computer just keep trying to read the data and when it finally appeared just read it, and thus dispense with the flag and the 8255? Without the DATA READY flag, the computer would sometimes read the data before it was completely encoded, thus getting a garbled number. Also, it would not be able to differentiate between a zero and no key being pushed at all, for when a zero is pushed, all that happens is that the DATA READY flag is activated.

Once the computer reads the operator's choice from the 8255, it can display the choice at the bottom of the video monitor, giving the operator some immediate feedback. Also, the computer can check to see if the operator's input was valid or not, and ask for it again if it was invalid. For instance, if there were three choices to which the response should have been a 1, 2, or 3, the computer can ask again if any other key than these is depressed on the remote keypad. "Please try again, my master."

If the response requires more than one digit, then a delay needs to be introduced before the next digit is entered. This can be accomplished one of two ways. First, the keypad encoder has a delay built into it that is adjustable by putting in different size capacitors. Second, the TRS-80 can execute a delay by running through a FOR-NEXT loop.

Once the computer has the complete response, it can begin to execute it immediately, or if it is a particularly polite computer it can say, "Your choice XYZ is now being executed."

SOME PRACTICAL CONSIDERATIONS

While in normal operation, the keypad would be plugged into the remote interface, it is convenient for system debugging to be able to also plug it into the computer interface. The author neglected to do this in his first system, and the result was that system debugging became something of a track event with frequent dashes out to the observatory (the remote position) to enter numbers on the remote keypad. In the present interface, this problem is solved by also providing a place to plug in the remote keypad in the computer interface.

In the original system, the encoder chip for the keypad was located in the computer interface. This had two drawbacks. First, eight lines dedicated to the keypad had to be used. Second, and much more serious, it was found that cables going to the remote interface that became wet in bad weather could cause false keying. This was due to a feature of the encoder chip that allows up to 50K of resistance in the switch contacts. If the resistance between the wires dropped to 50K or less, then the encoder thought a switch was closed whether it really was or not. These problems were solved by putting the encoder chip in the hand-held keypad. This only used five lines, and as the output of the encoder chip is three-stated, these lines could in theory be used to control other three-state devices. False keying was eliminated even during the heaviest rainstorms.

SIMPLE KEYPAD INPUT PROGRAM

To demonstrate the use of the keypad we will write a simple program that will display on the video monitor any key that we press on the keypad.

As with our previous control program, our first action is to program the 8255 in the configuration we desire with the appropriate control word. To do this, we remember that to select the 8255 A7 must be high and A6–A4 low (i.e., $1000XXXX_2$). To complete the address we refer to Table 2-2 and see that for the configuration process (control) A1, A0 = 11. Thus, our address is $1000XX11_2$, or for simplicity 10000011_2. In decimal this is 131. The control word, itself, can be determined by examining Fig. 2-5. Let us use only Mode 0, and have everything as output except Port A (keypad unit input) and the upper portion of Port C (which is connected to our switch), with both of these as input. Our control word would then be 10011000_2 or 152_{10}. The first line of the program is thus:

```
10 OUT 131, 152
```

As we do not want to read the encoder during a transition, we must wait until the data available (DA) flag goes high. To do this, we can continuously check the status of a bit PA4 in the 8255. As soon as it goes high we can then proceed to read the 4-bit code that will indicate which key was pushed. To do this we note for Port A that A1,A0 = 00, so our address will be 10000000_2 or 128_{10}. To isolate the bit in the A4 position we use the "AND mask" again with the mask being 00010000_2 or 16_{10}. Putting these into a little loop we have:

```
20 IF INP(128) AND 16 = 16 THEN 30 ELSE 20
```

To read the keyboard (KB) we want only the lower 4 bits so we need a mask of 00001111 or decimal 15. After reading this we can clear the old number off of the screen and put the new one on.

```
30  KB = INP(128) AND 15
40  CLS
50  PRINT KB
```

At this point we can insert an adjustable delay so that there is enough time to release one's finger without getting the number repeated. While this could be done by changing the value of the debounce capacitor, it is easier to do in software.

```
60 FOR IX = 1 TO 300: NEXT
```

Finally, one has to go back and check to see if another key has been pressed, and this completes the program.

```
70 GOTO 20
```

My four-year-old boy particularly likes this program and he says the numbers as he pushes the keys in both English and Spanish (his favorite program is Sesame Street). The entire program is shown in Example 3-1.

EXPERIMENT NO. 2
REMOTE COMMUNICATIONS

Purpose

The purpose of this experiment is to demonstrate simple computer/operator communications. It will also get you one step

Example 3-1. Keypad Input Program

```
10 OUT 131, 152
20 IF INP(128) AND 16=16 THEN 40 ELSE 20
30 KB = INP(128) AND 15
40 CLS
50 PRINT KB
60 FOR IX=1 TO 300: NEXT
70 GOTO 20
```

further in the building of your general-purpose interface with the outside world.

Discussion

To get the full benefit of this experiment, it is helpful to actually place the keypad and remote video monitor in another room. Keep in mind that the "operator" (some person expected to routinely operate the control system) and the computer programmer/system developer (that's you) are generally not the same person. To sharpen your skills and get in the proper spirit of things, it is good to have someone who knows nothing about computers play at being the operator.

STEP 1: Connections to the Computer Interface

The connections to the computer interface are shown in Fig. 3-2. Two wire wrap DIP sockets have been added to the board. One is a 40-pin socket that will be used eventually to correct the interface to the remote interface. The other socket is a 16-pin socket used to connect the remote keypad during system checkout or modification. In normal system operation the keypad unit would not be plugged in here, but would instead be plugged in at the remote interface.

The wiring consists of the four bits of data coming from the keypad unit. This is suitable for a hexadecimal or BCD code with "A" being the least significant of the four bits, and "D" being the most significant bit. These are connected to the aux keypad socket, the remote interface socket, and to Port A on the 8255. A data available (DA) line is also connected to all three sockets. Finally, a ground and +5 V dc (VCC) are connected to all three.

Connections previously made to the 8255 have not been shown to simplify matters. Wire wrapping is used as before, and it is a good idea to check off each wire as it is connected.

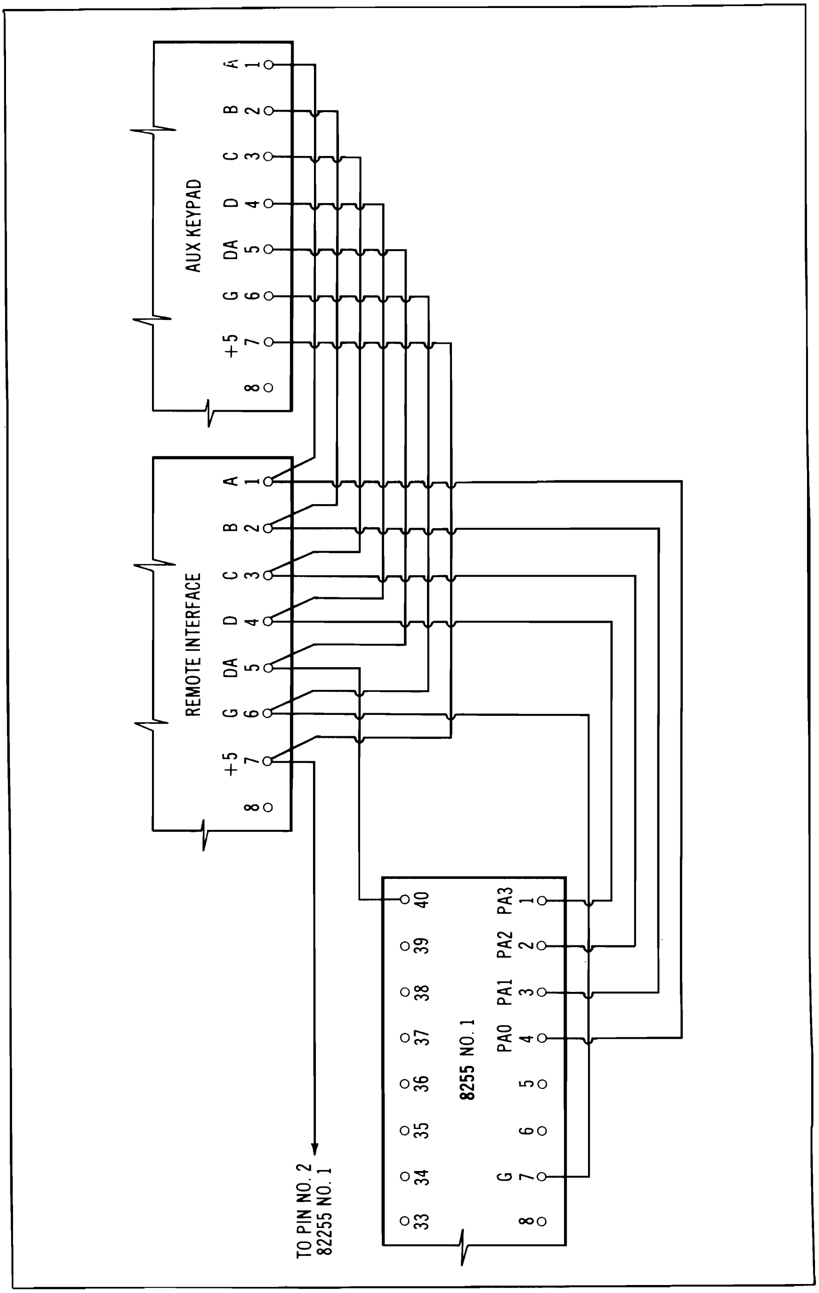


Fig. 3-2. Wiring diagram for the keypad input.

STEP 2: Keypad Unit

Any convenient box can be used to make the keypad unit. It should not be so big that it can't be hand-held. A 2-inch by 4-inch by 6-inch box should do nicely. A ready-made 16-key hexadecimal keypad is convenient to use and can be mounted directly on the box. However, one can just mount 10 push-button switches and wire them as shown in Fig. 3-3.

One could, of course, connect all 16 switches on a hexadecimal keyboard, but this is really not necessary as the computer can ask

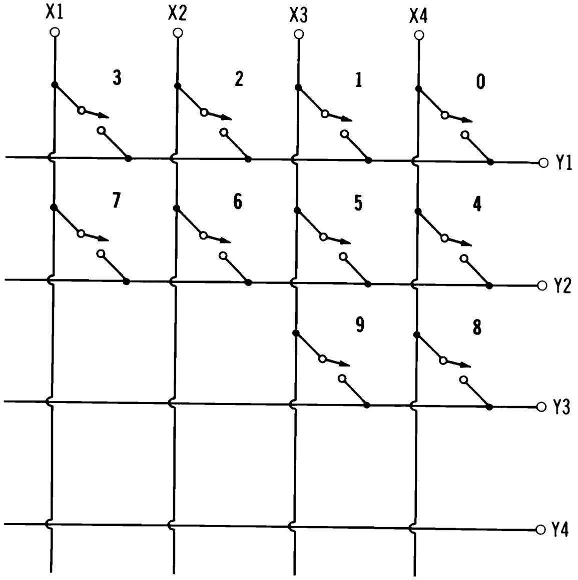


Fig. 3-3. Making up your own keypad.

for multidigit entries, so any more than 10 switches are not really required. The switches do not have to be laid out on the box panel in the order shown, only wired that way.

The wiring diagram for the keypad unit is shown in Fig. 3-4. The keypad, itself, is connected to the encoder chip with columns X1 through X4 and rows Y1 through Y4. The output of the encoder is the 4-bit signal A through D which is connected to a 16-pin wire DIP socket (this allows the keypad unit to be connected to either the computer interface or the remote interface with a 16-pin DIP jumper cable).

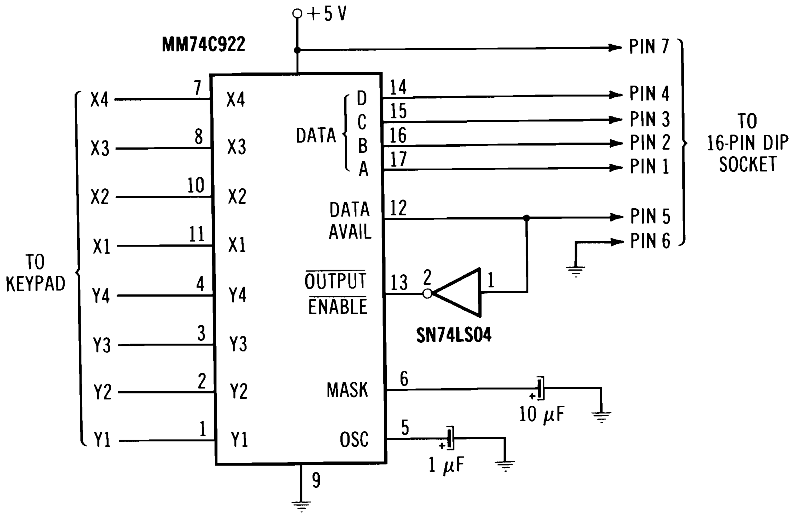


Fig. 3-4. Remote keypad wiring diagram.

Two external capacitors are required. One sets the scan frequency while the other sets the debounce time constant. The debounce period can be left fairly short and a "debounce delay" included in the software (this way it is adjustable). A single inverter is required also. It should be either a CMOS- or LS-type. Connection of the ground and +5 V dc rounds out the circuit.

STEP 3: Remote Video Monitor

A small black and white tv and tv modulator or a regular video monitor can be used for the remote indicator. If a small junction box is wired as shown in Fig. 3-5, both the local and remote monitors can be operated simultaneously, or alternatively one can switch between the two. It is often convenient to have both going simultaneously so that the local operator and a person at the computer location can both observe the information or data. In such a case, an intercom can provide for verbal communication. Note: A coaxial cable, such as RG-58, must be used to connect up the remote monitor.

STEP 4: Run It

Now connect all the circuits together, fire up the power supply and computer, and key in (on the computer keyboard!) the simple

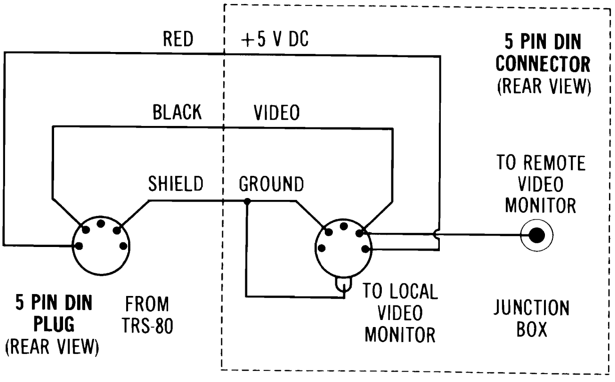


Fig. 3-5. Remote video monitor junction box wiring schematic.

program given earlier. Now (big moment) press one of the keys on the remote keypad. Either congratulate yourself or get busy troubleshooting depending on the outcome.

EXPERIMENT NO. 3 (Optional) WHAT SHALL I DO NEXT, OH GREAT MASTER?

Purpose

Put everything you've learned so far together on your own and develop your creative talents.

Discussion

So far we have a remote sensor (switch), a remote actuator (light), a remote video monitor that allows the computer to speak to the control system operator, and a remote keypad that allows the operator to speak to the computer. The challenge of this experiment is to put all these elements together (without making any hardware changes) in your own unique system.

STEP 1: Plan Out Your Control System

Here is where you can brainstorm. If you are fresh out of ideas, here is one you can try. Clear the screen and display "I AM AT YOUR SERVICE, PRESS ANY KEY TO GET STARTED." Now have the computer keep checking for the DATA READY flag. Once it is detected, clear the screen and display:

```
            Your Choices Are:

          1. Turn Light On
          2. Turn Light Off
          3. Blink Light 20 Times

      Enter Your Choice On The Keypad Please
```

Have the computer check for the ready flag, read the data, verify that it is acceptable, and relay choice back to the operator. Then have the computer start checking the switch to see when it is pushed. Once pushed have it execute the appropriate action and when completed clear the screen and redisplay the choices (the menu) again.

STEP 2: Write and Debug the Control Program

You already know how to do this. After all, it's only Level II BASIC with a bunch of INP and OUT statements.

STEP 3: Checkout

Now get your Aunt Nellie (or equivalent) to operate the system. *Don't* tell her that there is a computer hidden away in the other room. Now if Aunt Nellie can operate it, anyone can. If she has trouble with it, then perhaps your English on the monitor isn't plain or detailed enough, and you need to modify things a bit and try it out again on Uncle Bert.

Having successfully established computer/operator communications, can we now move on to adding more sensors and actuators for something really fancy? No, the *time* is not ripe for this. First we need to add the *time* in real-time control.

CHAPTER 4

What Time Is It?

By incorporating a clock into a computerized control system, it is possible to take specific actions at specified times. When used as a data logger in scientific experiments, or as a monitor in industrial processes, it is convenient for a real-time control system to know the time and also the date.

There are a number of ways of implementing a clock in a computerized control system. One of the easiest is to use the clock function built into many microcomputers. The TRS-80 Model I, for instance, incorporates this function when the expansion interface is added to the system. While easy to implement, this approach does have a few drawbacks. First, the operation of the internal clock must be suspended during data access or when loading from tape or disk, causing some inaccuracies. As the clock requires periodic program interrupts to function, it may have to be inhibited during critical interface operations. Also, the clock obviously has to be set on powering up the computer each time. Finally, the Model I without the expansion interface does not have this capability. In spite of these drawbacks, there are many situations where the computer's own clock would suffice, and using what is already available has much to commend it.

A second approach is to construct a clock using a programmable counter chip. In an earlier version of the present interface, an Intel 8253 programmable counter was used as a clock. This highly versatile LSI chip has three computer programmable counters/dividers of 16 bits each. To form a clock a 1.0 MHz clock signal from a crystal oscillator was fed to the 8253. Each of two counters were used as divide by 1000, and the third was used to count the resulting 1 Hz (one per second) pulses. While considerable software was required to read and set this clock, no interrupt was required, and the software came into play only during the actual setting or reading of the clock. A distinct advantage of this sort of clock is that 1.0 MHz crystal oscillators of very high accuracy and stability are available, although high accuracy tends to cost "high dollars."

A third approach is indebted to digital wristwatches. With minor modification in their internal circuitry to ease interfacing with microcomputers, and an increase in size of packaging to make them easier to handle, the chips used in digital wristwatches can now be connected to your computer. As with the digital watches, these devices also give the date and even the day of the week if one wants it. These LSI chips use a small 32.768 kHz crystal in conjunction with a built-in oscillator. While these are a great step forward, it is expected that clock chips designed specifically for use with microcomputers should be available shortly.* Most of the clock chips are very low power devices, and they can be operated with small batteries when the computer and interface are turned off, thus being already set when the system is powered up again.

THE OKI MSM5832

For the interface described in this book, the OKI MSM5832 microprocessor real-time clock/calendar chip was selected because it has many desirable features and is readily available at an affordable price. Its only real disadvantage is that it must be interfaced to the computer via an 8255, but as we are already familiar with the 8255, and it is also low in cost, this is not a severe disadvantage and gives another good example of peripheral device interfacing.

The basic approach taken in this interface is shown in Fig. 4-1. As can be seen, a second 8255 serves as an interface between the high-speed TRS-80 and the much slower MSM5832. Bits PA3–PA0 in Port A of 8255 No. 2 serve to pass data to and from the MSM5832. The direction of the data flow is controlled by bits PC1 and PC2 of the 8255 No. 2 which control the READ and WRITE modes of the data lines.

*After writing the book, but just prior to its publication, several such chips have become available. The National Semiconductor MM58174 is typical.

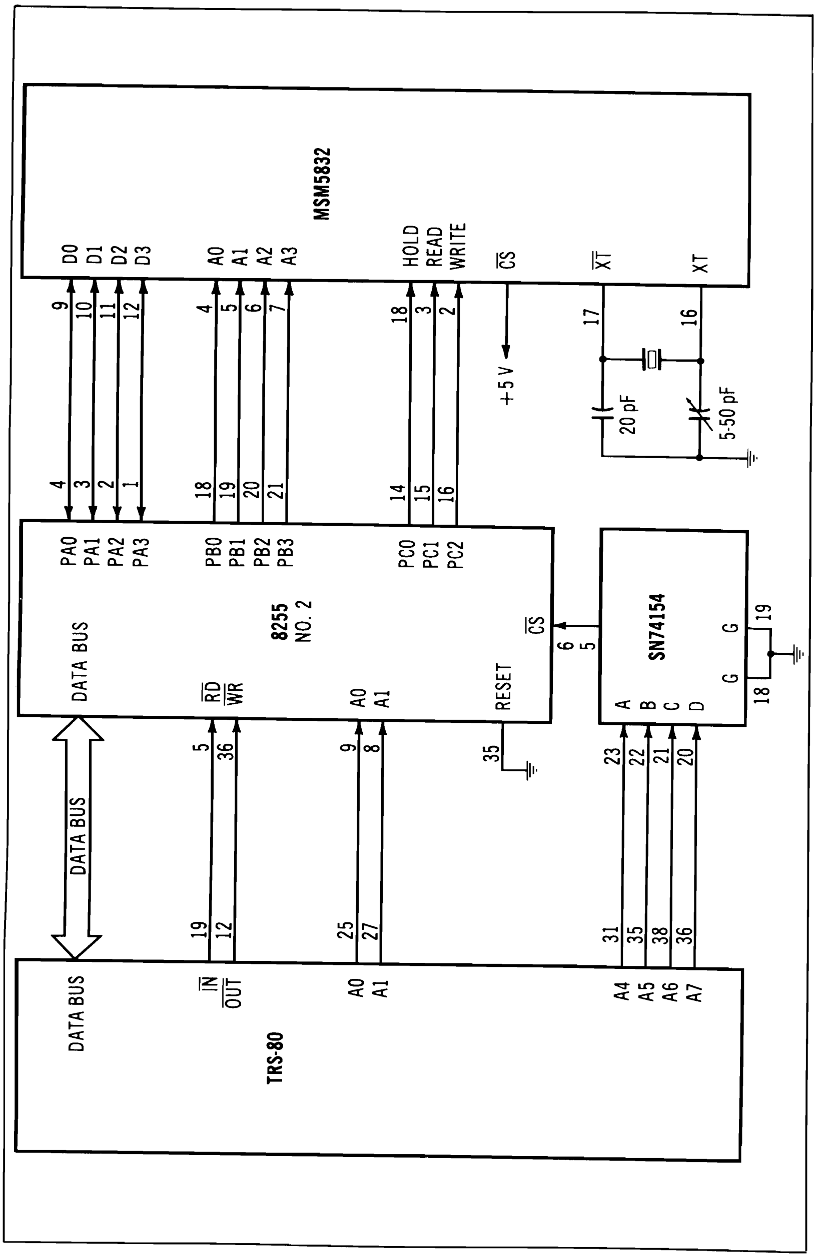


Fig. 4-1. Clock schematic wiring diagram. (Power connections and pull-up resistors not shown for clarity.)

What MSM5832 function one is reading and writing to is controlled by bits PB3–PB0 of 8255 No. 2 which form the address lines for the MSM5832. The meanings of these addresses are given in Table 4-1. As can be seen from this table, the main functions are reading or writing various digits with respect to seconds (S), minutes (M), hours (H), week of the month (W), day of the month (D), month of the year (M), and year of the century (Y). In cases where two digits are required, the least significant is indicated by a "1" and the most significant by a "0." In addition, there are provisions for selecting a 12- or 25-hour format (we will use the latter), flagging whether am or pm in the 12-hour format (need not concern us), and a provision for leap years.

The HOLD capability allows one to perform various operations with the clock without disturbing the clock. For instance to read the clock, it can be put into HOLD so that the output digits are frozen as of the instant it was put in HOLD, and one can read the digits one at a time without worrying about them changing while being read. While this is being done, the internal clock in the MSM5832 does not lose any time, and as soon as the HOLD is released things are back to normal.

A SIMPLE PROGRAM TO READ THE SECONDS

Before getting involved in a program that exercises many of the capabilities of the MSM5832, let's first consider a simple program. Let's see how we could read the least significant "seconds" digit.

First, we must program the second 8255 for the configuration we want. To read the clock data we want Port A as input and Ports B and C as output (refer to Fig. 4-1). The address for programming the second 8255 is 01000011 or 67 decimal, and the control word for the configuration we want is 10010000_2 or 144_{10}. This gives:

```
10 OUT 67, 144
```

Now to read the clock we must first put it into a HOLD so the digits won't be changing while we are reading it (it will keep accurate time if we don't "hold" it for over 1 second, a very long time even for interpretive BASIC). To put it in HOLD we need to set bit 0 of Port C (PC0=1) as follows:

```
20 OUT 67, 1
```

Next, we need to put out the 4-bit address for the least significant seconds digit. This address can be found from Table 4-1 for the MSM5832. The address is 0000, and we need to output this on Port B:

```
30 OUT 65, 0
```

To read the chip we need to set PC1 = 1:

```
40 OUT 67, 3
```

And then read the data from Port A:

```
50 S1=INP(64)
```

We can then display the data and release the HOLD (PC0=0):

```
60 CLS
70 PRINT S1
80 OUT 67, 0
```

And finally, throw in a bit of a delay so that the video display doesn't flicker too badly, and then go back and do it all over again:

```
 90 FOR I=1 TO 50: NEXT
100 GOTO 20
```

The entire program is shown in Example 4-1.

Example 4-1. Seconds Read Program

```
 10 OUT 67, 144
 20 OUT 67, 1
 30 OUT 65, 0
 40 OUT 67, 3
 50 S1= INP(64)
 60 CLS
 70 PRINT S1
 80 OUT 67, 0
 90 FOR I=1 TO 50: NEXT
100 GOTO 20
```

MENU PLEASE, WAITER

So far we have been content with writing little demonstration programs that are ends in themselves. Now, however, we are going to set the date and time and read these. As this is no longer a minor programming task, we need to do this in a way so that we will not have to go back and reprogram them later for the final control program. Thus, we need to start constructing the core of our control program.

As discussed in Chapter 1, the programming task is made much easier (and flexible for later changes) if it is done in distinct modules, and the various modules selected by a Main Menu. This particular one-digit main menu has a maximum of 9 selections (not as good as a Chinese restaurant), and if more are desired any main menu selection can call a sub-menu, etc. To start with, only selections "2" and "7" are shown to set and read the clock, respectively. These particular places on the menu were used because that's where they ended in the last version of the control program on an earlier version of the interface. As the goal here is a gradual evolution of the design, there is no need to change things unnecessarily.

Before getting to the Main Menu, we need to identify the overall program, program 8255 No. 1, so we can use the remote keypad, and set up an array for the clock. This is shown in Example 4-2. Now we can put in the Main Menu with its two selections as shown in Example 4-3.

Example 4-2. Control Program Initiation

```
1 REM FAIRBORN OBSERVATORY CONTROL PROGRAM 11
3 REM PROGRAM #1 8255
4 OUT 131, 152
8 DIM T(12)
```

Example 4-3. Main Menu

```
1000 REM MAIN MENU
1010 CLS PRINT "FAIRBORN OBSERVATORY CONTROL PROGRAM"
1020 PRINT "MAIN MENU—MAKE YOUR SELECTION"
1060 PRINT "2 SET CLOCK/CALENDAR"
1110 PRINT "7 DISPLAY DATE/TIME"
1200 GOSUB 2000
1220 ON KB GOTO 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000,
     11000
```

Line 1200 goes to the remote keypad subroutine to input the code from the keyboard that corresponds to the menu selection chosen by the operator, and the command on line 1220 directs the program to the program module corresponding to the menu selection.

The keypad input routine shown in Example 4-4 is just like our keypad demonstration program with the line numbers changed.

Example 4-4. Remote Keypad Subroutine

```
2000 REM REMOTE KEYPAD SUBROUTINE (1 DIGIT)
2005 1F INP(128) AND 16=16 THEN 2015 ELSE 2010
2010 GOTO 2005
2015 KB=INP(128) AND 15
2020 PRINT KB
2025 FOR IX=1 TO 300: NEXT
2030 RETURN
```

SETTING THE CLOCK

Now, we are ready to write the program to set the clock. From line 1220 in Example 4-3 we know that Main Menu Selection No. 2 must start on line 4000. The entire Set Clock Routine is shown in Example 4-5.

Example 4-5. Program To Set the Clock

```
4000 REM SET CLOCK ROUTINE
4005 CLS
4007 OUT 67,128
4010 PRINT "ENTER DATE YYMMDD (6 DIGITS)"
4012 FOR I=12 to 7 STEP -1
4015 GOSUB 2000
4017 T(I)=KB: NEXT
4020 PRINT "ENTER TIME HHMM (4 DIGITS)"
4022 I=5 TO 2 STEP - 1
4025 GOSUB 2000
4027 T(I) =KB: NEXT
4030 T(5)=T(5)+8
4035 PRINT "PRESS ANY KEY TO START CLOCK"
4040 GOSUB 2000
4045 OUT 67,1: OUT 67,2
4050 FOR I=0 TO 12
4055 OUT 65,1: OUT 64, T(1)
4060 OUT 67,5: OUT 67,4
4065 NEXT 1: OUT 67,0
4070 GOTO 9000
```

After identifying the routine (4000) and clearing the screen, the first task is to program the second 8255. We want Mode 0 and all ports in the output mode. For this we need 10000000 or 128_{10}. Next

(4010) we ask for the year, month, and day of month. From Table 4-1 we can see that the addresses for these go from 12 down to 7, and if we store these numbers in an array with the same index number as is required by the address inputs on the clock, we can simplify things when we load the clock (4012) since the array index can be output, the address control inputs on the clock chip, automatically selecting the proper "location" for the data. At 4015 we go to the keypad for a number and put in the array (4017). We then ask for the hours and minutes (4020), and place them in the array from 5 down to 2 (4022) to (4027).

Why didn't we ask for the seconds, as certainly we wish to set them also? The reason is that the MSM5832, when it is set, automatically puts both seconds digits to 0. This means that you must set the clock right on the minute. Accurate clock-setters use a radio to set their clocks. In the U.S. radio station WWV broadcasts time signals from Fort Collins, Colorado on 5, 10, and 15 MHz. These can be conveniently picked up on the Radio Shack time cube or a shortwave receiver, and the time is very accurately controlled by

Table 4-1. MSM 5832 Function Table

ADDRESS INPUTS				INTERNAL COUNTER	DATA I/O				DATA LIMITS	NOTES
A_0	A_1	A_2	A_3		D_0	D_1	D_2	D_3		
0	0	0	0	S 1	*	*	*	*	0 ~ 9	S_1 or S_{10} are reset to zero irrespective of input data D_0 ~ D_3 when write instruction is executed with address selection
1	0	0	0	S 10	*	*	*		0 ~ 5	
0	1	0	0	MI 1	*	*	*	*	0 ~ 9	
1	1	0	0	MI 10	*	*	*		0 ~ 5	
0	0	1	0	H 1	*	*	*	*	0 ~ 9	
1	0	1	0	H 10	*	*	†	†	0 ~ 1 / 0 ~ 2	D_2 = "1" for PM D_3 = "1" for 24 hour format D_2 = "0" for AM D_3 = "0" for 12 hour format
0	1	1	0	W	*	*	*		0 ~ 6	
1	1	1	0	D 1	*	*	*	*	0 ~ 9	
0	0	0	1	D 10	*	*	†		0 ~ 3	D_2 = "1" for 29 days in month 2 D_2 = "0" for 28 days in month 2 (2)
1	0	0	1	MO 1	*	*	*	*	0 ~ 9	
0	1	0	1	MO 10	*				0 ~ 1	
1	1	0	1	Y 1	*	*	*	*	0 ~ 9	
0	0	1	1	Y 10	*	*	*	*	0 ~ 9	

(1) * data valid as "0" or "1"
blank does not exist (unrecognized during a write and held at "0" during a read)
† data bits used for AM/PM, 12/24 HOUR and leap year
(2) If D_2 previously set to "1", upon completion of month 2 day 29, D_2 will be internally reset to "0"

Courtesy OKI Semiconductor, Inc.

Cesium-beam clocks at the National Bureau of Standards. Even the big astronomical observatories use the $35.00 time cube to set their $1000 clocks that run the $1,000,000 telescopes. If you are happy with one-tenth second accuracy, you can set your clock by ear. "On the tone the time will be 13 hours and 42 minutes universal time . . . beep." If you want to get it to the nearest millisecond, you need electronic triggering and an oscilloscope and have to be concerned with how long it takes the radio signal to get to you from Fort Collins (about 7 milliseconds for the Fairborn Observatory). Even microsecond accuracy can be had, but it is expensive. We'll settle for one-tenth of a second or so.

The sharp reader will note that we didn't pick up T(6), the day of the week. It is redundant information, although the program could be modified to include it.

Line 4030 adds 8 to T(5). In binary, 8 is 1000_2, and from the clock function table we can see that bit D3 needs to be set to 1 for a 24-hour format. This sets that bit.

We now go to the keypad (4035 and 4040) and wait for *any key to be pressed when the tone is heard* on the minute from the time cube, and then immediately set the clock. To do this we put the clock in HOLD (4045) by setting bit PC0=1 and also make sure we are not in the READ mode by setting bit PC1=0 (last part of 4045). For the 13 digits we need to load into the clock (4050), we put the address out from Port B (4055 first part), and then put the data out from Port A (4055 last part). We then bring the read high (PC3=1, 4055 first part), and then bring it back low (PC3=0, 4055 last part). We repeat this for the rest of the digits, changing the address and data each time, and then release the HOLD (4065). To see that it is properly set, the program automatically jumps to the read routine (4070).

READING THE CLOCK

In reading the clock, one needs to consider the format in which the data is to be displayed, and also how it might be used in data logging. It is very convenient in data logging to have a single 6 digit number for the time, and another 6 digit number for the date. It is also convenient for the actual reading of the clock to be contained in a subroutine distinct from the display of the time on the video monitor. The program module shown in Example 4-6 can display the date and time on the video monitor (by going to 9000) or can

Example 4-6. Program To Read the Clock

```
9000 REM READ CLOCK
9001 GOSUB 9003
9002 GOTO 9045
9003 REM SUBROUTINE TO READ CLOCK
9004 OUT 67,144
9005 OUT 67,1: OUT 67,3: OUT 67,4
9010 FOR 1=0 to 12
9015 OUT 65,1
9017 T(I)=INP(64)
9020 NEXT 1: OUT 67,0
9022 T(5)=T(5) AND 3
9030 DATE=T(7)+10*T(8)+100*T(9)+1000*T(10)+10000
     *T(11)+100000*T(12)
9040 TIME=T(0)+10*T(1)+100*T(2)+1000*T(3)+10000*
     T(4)+100000*T(5)
9042 RETURN
9045 CLS: PRINT DATE, TIME
9050 PRINT "PRESS ANY KEY TO CONTINUE"
9055 IF INP(128) AND 16 = 16 THEN 9060 ELSE 9057
9057 FOR I=1 to 250: NEXT: GOTO 9000
9060 GOTO 1000
```

just set the variables DATE and TIME each to six digit values by using GOSUB 9003 from anywhere in the control program.

The read subroutine first programs 8255 No. 2 with Port A as input and Ports B and C as output (9004). The HOLD and READ are set high (PC0 and PC1=1) and WRITE set low (PC2=0) by line 9005. Going through the 13 digits (9010) the address of each digit is put out on Port B (9015) and the digit read via Port A (9017). After the last digit has been read, the HOLD is released (9020).

As bits D2 and D3 of the tens of hours value (T(5)) represent AM/PM and 12/24 hour format, respectively, we need to mask them out with 0011_2 or 3_{10} (9022). To combine the individual digits into two 6 digit numbers, we can multiply by successive powers of 10 (9030 and 9040). This gives us DATE and TIME and if this was called by the time display menu selection, we return to 9002 and jump to 9045 where we clear the screen and print the date and time.

We need both to update the display, and, when the system operator gets tired of watching it or is satisfied that the clock is properly set, some means for getting back to the main menu so he can get on with other things. Line 9055 checks to see if a key is pressed, and if not does a delay and reads the clock again. As soon as a key is pressed, it jumps out of the loop and goes back to the main menu (9060).

EXPERIMENT NO. 4
SETTING AND READING THE CLOCK

Purpose

The purpose of this experiment is to add a second 8255 and the MSM5832 to your general-purpose interface and use them to assess accurate time.

Discussion

In this experiment we will be mainly concerned about connecting up the 8255 and the MSM5832 and demonstrating their proper functioning by using the programs developed earlier in this chapter.

STEP 1: Connecting a Second 8255 PPI Chip

Rather than connecting everything at once, it is easier to first connect 8255 No. 2, and then connect the MSM5832.

Shown in Fig. 4-2 is the wiring diagram for 8255 No. 2. As can be seen, it is simply connected in parallel with the first 8255 chip. The $\overline{CS}$ was connected to pin 5 on the 74154 4-to-6 address decoder. By referring to the 74154 truth table (Table 2-1), we can conclude that the 8255 No. 2 will be selected by address 0100XXXX. The "base" address is 01000000_2 or 64_{10}. From our experience with the first 8255 with a base address of 128, we can guess that with the second 8255, Port A will be at address 64_{10}, Port B at 65_{10}, Port C at 66_{10}, and the Internal Instruction Register at 67_{10}.

The operation of the second 8255 can be checked with the following program:

```
10 OUT 67,128   (Mode 0, all ports output)
20 OUT 64,15    (Port A should be 00001111 which you
                can check with a voltmeter.)
```

STEP 2: Connecting the MSM5832 Clock Chip

The MSM5832 needs pull-up resistors (unfortunately) as well as two external capacitors and, of course, the crystal for the oscillator. The pull-up resistors can be between 4K and 20K, with the 10K value used by the manufacturer. Wiring can be made easier by buying resistors already available in a DIP package and plugging

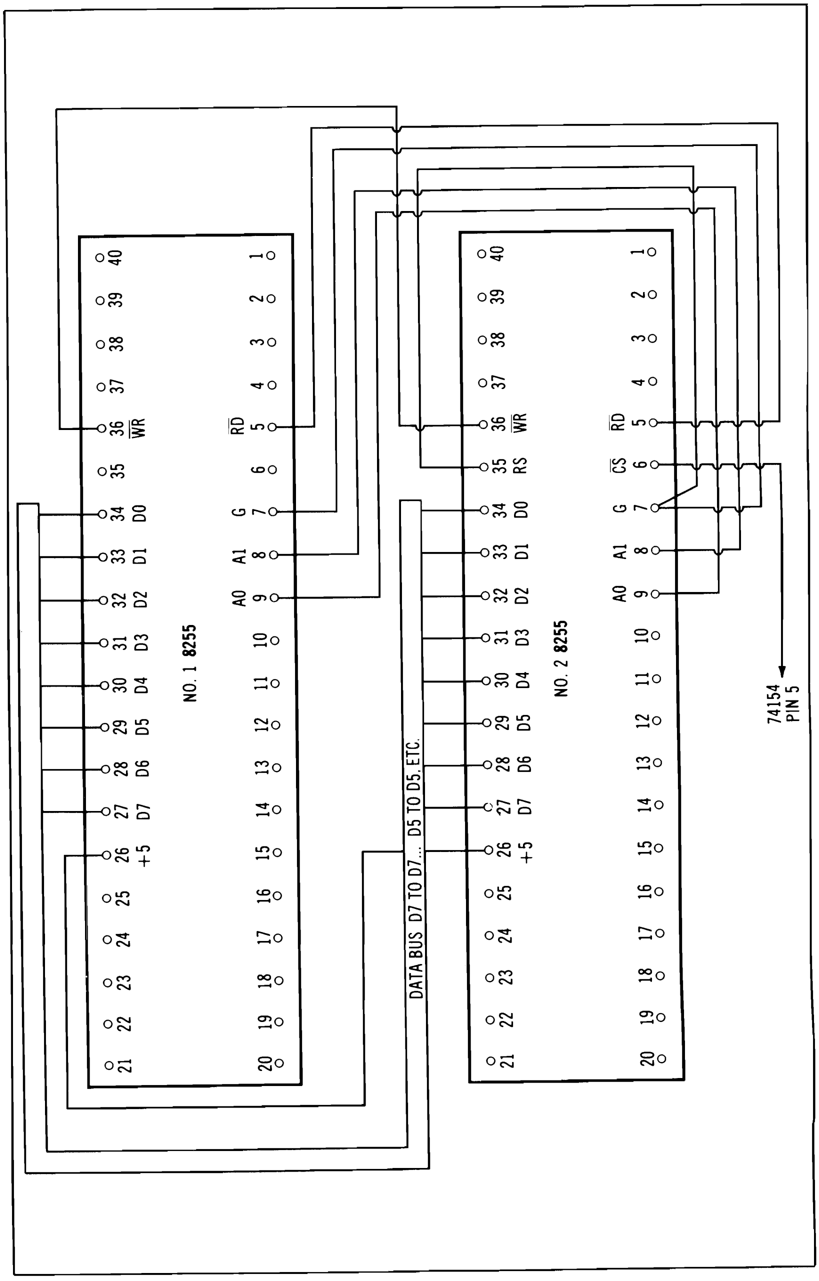


Fig. 4-2. Wiring schematic diagram for the No. 2 8255.

them into a couple of wire wrap sockets. If these packages are not readily available, bend the wires over on each end, and insert the wires in appropriately spaced holes. Push down on the resistors to seat them on the board. You can then solder a wire in common to one side of each resistor and use the other side for wire wrapping. As wire wrapping does not make a good connection unless it is wrapped on something with corners, I recommend soldering the wires after wrapping them on the free ends of the resistors.

One of the capacitors needs to be a small variable type so that the clock oscillator frequency can be fine-tuned a bit. Before final adjustments were made, one clock lost a minute each day, but with a bit of patience, one can adjust this variable capacitor so that the clock will be within a second per day.

The wiring schematic for connecting the MSM5832 is shown in Fig. 4-3.

STEP 3: Program and Run

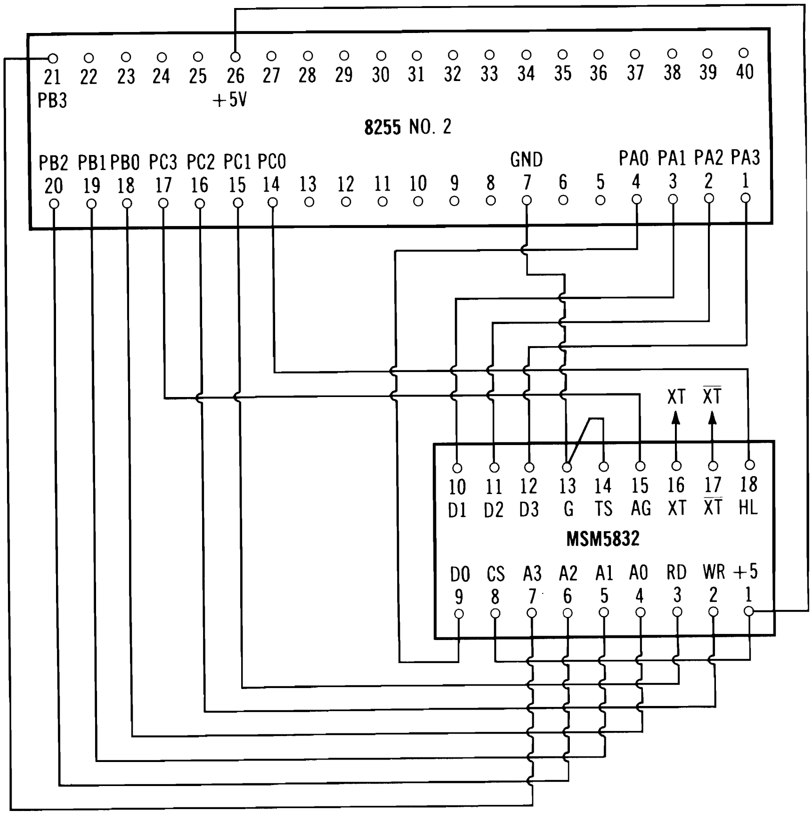
Key in the programs developed earlier in this chapter and see if your additions to the interface work properly.

A NOTE ON RFI AND TROUBLESHOOTING

The Radio Shack Time Cube puts out noise when placed by the TRS-80. The interference from the TRS-80 even tends to run out the control system cable to my remote position in the observatory. I found, however, that if I set the Time Cube on our clothesline post and turn the volume way up, I can hear it just fine. Every clear night just as it is getting dark the neighbors hear "tick, tick, tick, at the tone the time. . . ." Besides checking my clock, this lets the neighbors know that the star-nut is at it again and they kindly turn off their porch lights.

My program didn't run right the first time (as so often happens). I would set the time, but when it jumped to the read part, all I got was zeros. Messing around a bit, I found that if I went directly to reading the time without setting it first, I got some garbage at least. This was traced down to line (9004) which had 134 instead of 144. It seems that 128+16 does not equal 134 (forgot to carry). The garbage was straightened out by reading the digits in the right order (I was going backwards) and masking out D2 and D3 on T(5).

In troubleshooting a program, I often find it handy to insert a STOP statement at a questionable point, and then examine each of the variables (such as T(5) by keying in ?T(5)). One can alternatively put lots of PRINT . . . commands in test programs so you can see what is happening. These can then be edited out when the program runs properly. Most of troubleshooting is just verifying that things are doing what you think they ought to be, until you run across something strange. While it is something of an art acquired by practice, one good way to get the practice is to take a working circuit and program, insert a STOP at different points, examine the variables with a "?" and at the same time measure for logic highs or lows with a multimeter or simple logic probe. You will find that

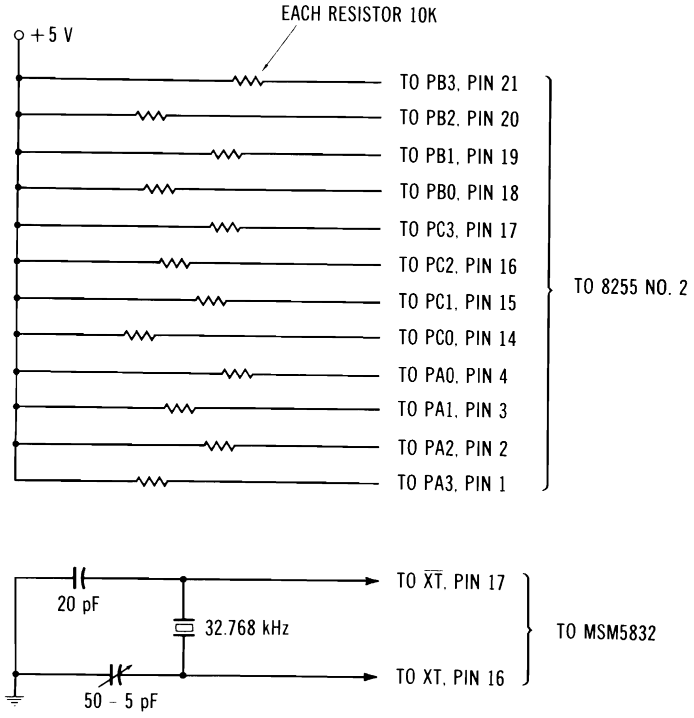


(A) Connecting the MSM5832 to 8255 No. 2.

Fig. 4-3. Clock

your brain and inventiveness on the keyboard are better than expensive test equipment.

Now that we have computer/operator communications established, can we at long last add some goodies? How about a counter such as that used to count the filled beer bottles at my favorite brewery?



(B) Pull-up resistors, external capacitors, and crystal.

connections.

CHAPTER 5

Counting

Counters have many uses in control systems. An obvious example is counting the number of parts passing a particular point in an assembly process. In conjunction with optical incremental angle shaft encoders, counters can be used to determine very accurate positions of machine tools or measuring devices used in experiments. In conjunction with voltage-to-frequency (V/F) converters, counters can be used as very precise analog-to-digital converters. Counters can also be used as accurate timers or delay devices. Counters can also be used as dividers.

Most of the functions mentioned in the preceding paragraph can be done by the computer, itself, particularly if programmed in machine language. However, the use of external counters greatly simplifies the programming task. Also, and sometimes of critical importance, the use of external counters can free the computer to do other things while the counting is being done. By using external counters, lots of different counts can be going on all at once, with the computer running around between them checking progress, analyzing the results, and issuing new orders.

To use external counters to their full advantage, it is necessary that their functions and counts be programmable. It is very handy if a count can be read while in progress (on the fly) without disturbing the count. Finally, it is convenient if the counters can be directly tied to the TRS-80 bus to eliminate the requirement for buffer chips.

The Intel 8253 meets all of these requirements at an affordable price. It contains three independent counters(either 16- or 8-bit)in either binary or binary coded decimal (bcd). They can be operated independently in one of 6 different modes. They have three-state outputs and can be tied directly to the TRS-80 bus.*

A TYPICAL COUNTER APPLICATION

Rather than speak in generalities about the Intel 8253 programmable counter, we will consider an application that not only uses all three of the counters in the 8253, but also uses these in a coordinated fashion with all the other chips on our general-purpose interface. In this example the requirement is to make two simultaneous independent counts of pulses from an external source that occur in some precise time interval. To do this, we will use one of the three counters in the 8253 to define the time interval, and use the other two counters to make the actual counts.

Shown in Fig. 5-1 is a schematic of the approach to meet these requirements. Some detail internal to the 8253, itself, has been shown to clarify what is happening. Let's consider a typical sequence of operations:

- The TRS-80 selects the 8253 via the 74154.
- The counters are programmed for the appropriate function (as down-counters in this case).
- Via one of the 8255 chips the MSM5832 clock is programmed to output a precise 1024-Hz signal.
- Counter No. 0 is loaded with the number 1024×10 (10240) so that when it is told to start counting down its output will be changed in exactly 10 seconds (our count time in this example).
- Counters Nos. 1 and 2 are loaded to their maximum value (64,536) so that they will have the greatest counting range. Of course, we really want to up-count, but by down-counting and subtracting the end count from the starting count we achieve the same thing.
- Everything is set to go, and we trigger the process with a pulse from the other 8255 that opens the gate and enables Counter No. 0. This, in turn, enables Counters Nos. 1 and 2.
- When Counter No. 0 reaches terminal count at the end of exactly 10 seconds, it shuts off the gates on Counters Nos. 1 and 2, freezing their results.

*Just prior to publication, Intel announced the 8254. It is almost identical to the 8253 except that it can count at a 10 MHz instead of 2 MHz rate.

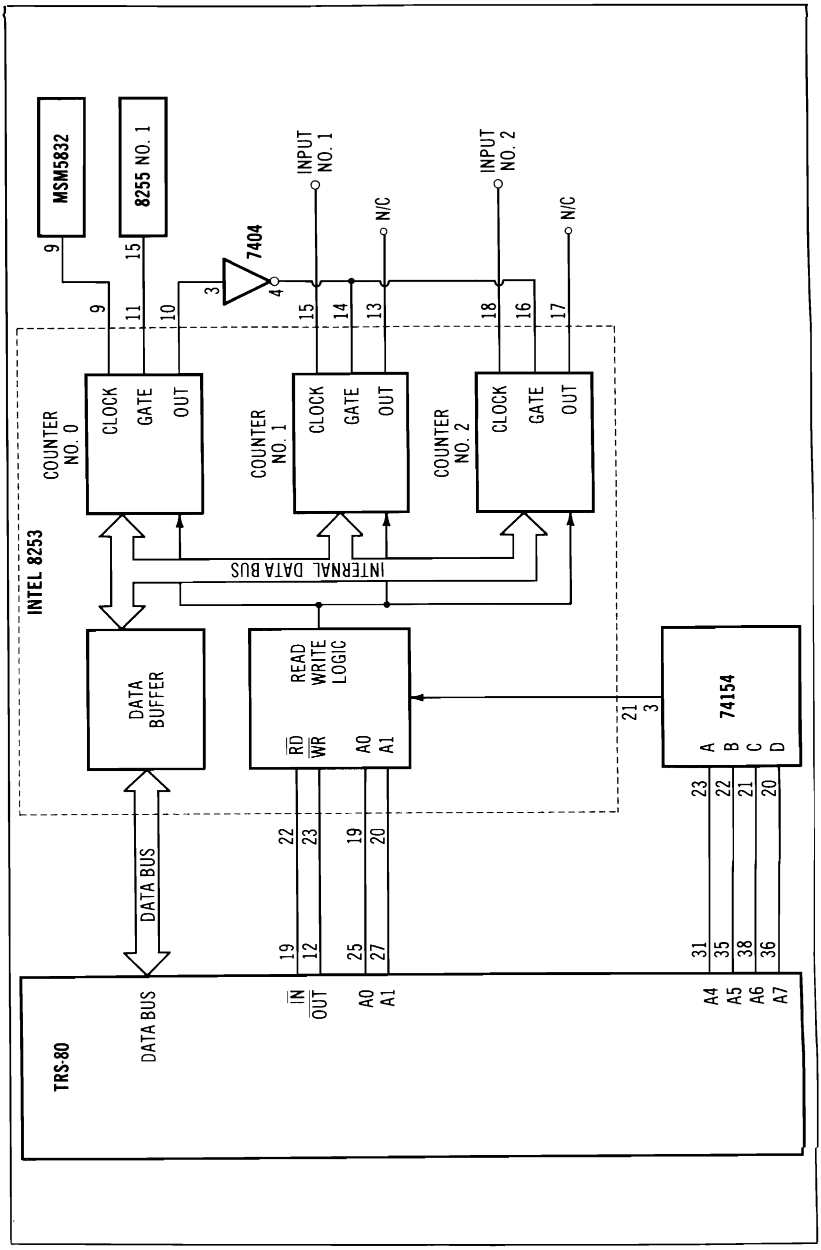


Fig. 5-1. Intel 8253 used as a timer/dual counter. (Power connections not shown for clarity.)

- The TRS-80 can now go in and read the final counts, and if desired, automatically start the entire sequence all over again.

The 8253 can be read "on the fly." This means that while counts are in progress the counts can be read without effecting the outcome. This is a handy feature if one wants to display a curve versus time of the inputs. Another handy feature is that one can take the counts from one cycle and use this to adjust the counting time for the next cycle. This can make the system adaptive or autoranging.

The example we have just discussed is just what one would need if one had data coming from two sources, such as two stars. This particular application will be treated in greater detail later in this chapter.

THE 8253 AS A PRECISION CLOCK

To illustrate the flexibility of the 8253, consider the application shown in Fig. 5-2. Instead of down-counting, the 8253 can be used as a programmable divider. I used this feature to form a precision clock that could be adjusted against the radio signal from the National Bureau of Standards radio station WWV to within a millisecond.

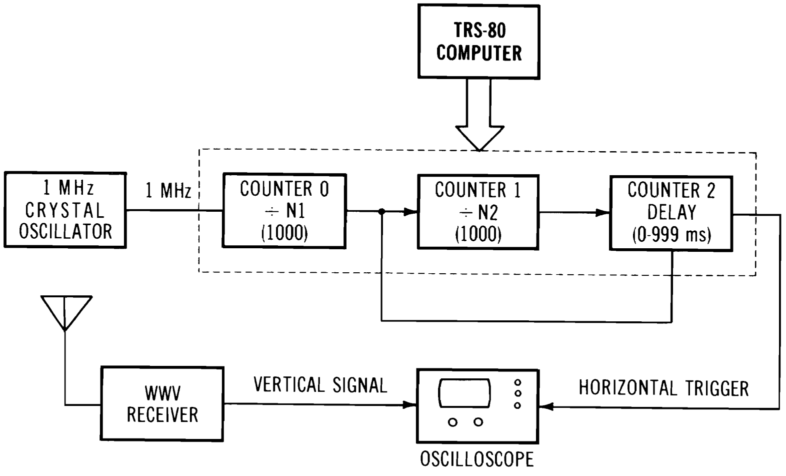


Fig. 5-2. Intel 8253 used as a precision clock.

A 1 MHz crystal oscillator feeds Counter No. 0 which divides by 1000 to produce a 1 kHz signal. This feeds Counter No. 1 that further divides by 1000 to produce a 1 Hz signal. It also feeds Counter No. 2 which is programmed to be a delay from 0 to 999 milliseconds. The output from Counter No. 2 is used as the horizontal sweep trigger on an oscilloscope. The vertical input to the oscilloscope is taken off of a WWV receiver (such as the Radio Shack Time Cube). The delay is adjusted from the TRS-80 to achieve synchronization.

As you might suspect, there are astronomical measurements that require precise timing. An interesting case is the disappearance of stars behind the dark limb of the moon. Then there is the emergence of the moons of Jupiter from Jupiter's shadow, not to mention. . . .

PROGRAMMING EXAMPLE

As with the clock program, we will find the counter program to be useful later, so we will build it in as a part of our control program. We can put it on the Main Menu with:

```
1050 PRINT "1 COUNT TEST"
```

This will transfer control to line 3000, where we can have a skeleton program shown as Example 5-1. This calls the actual count routine at 2500. Lines 3012 to 3015 allow a return to the Main Menu when any key is pressed.

The counter subroutine starting at line 2500 is given in the example, and is explained in some detail as it is a good illustration of the cooperative use of multiple programmable interface chips.

At line 2505, the No. 2 8255 is programmed to read the MSM5832 clock chip. This is the same as the operation at line No. 9004 in Example 4-6.

The MSM5832 can, besides being a clock, be a precise signal generator. The available reference signals and the conditions necessary for their use are shown in Table 5-1. We will be using the 1024 Hz output. To meet the conditions, we need on No. 2 8255 to have PC0=0, PC1=1, and PC2=0. In addition, we need to put out 1111_2 or decimal 15 on the address lines from Port B. Line 2510 in Example 5-1 does all these in the order stated.

Next, we need to program the 8253 counters to their proper

Example 5-1. Counter Program

```
2500 REM COUNTER SUBROUTINE
2505 OUT 67, 144: REM PGM #2 8255
2509 REM SET MSM5832 TO NOT HOLD, READ, NOT WRITE
2510 OUT 67, 0: OUT 67,3: OUT 67,4: OUT 65,15
2514 REM SET CNT 0 MODE 1, CNT 1 & 2 MODE 2
2515 OUT 35,50: OUT 35,116: OUT 35,180
2519 REM SET CNT 0 TO 10240, CNT 1 TO 0
2520 OUT 32,0: OUT 32,40: OUT 33,0: OUT 33,0
2525 PRINT"PRESS ANY KEY TO COUNT"
2530 GOSUB 2000
2539 REM TRIGGER PC1 #1 8255
2540 OUT 131,3: OUT 131,2
2545 IF INP(130)=0 THEN 2554 ELSE 2550
2550 GOTO 2545
2554 GOSUB 9003
2555 COUNT=INP(33)+256*INP(33)
2560 COUNT=65536-COUNT
2565 RETURN
3000 REM COUNT TEST
3005 GOSUB 2500
3010 PRINT "COUNT=";COUNT
3012 IF INP(130)=0 THEN 3005 ELSE 3015
3015 GOTO 1000
```

Table 5-1. Reference Signal Outputs

CONDITIONS	OUTPUT	REFERENCE FREQUENCY	PULSE WIDTH
HOLD = L	D_0 (1)	1024 Hz	duty 50%
READ = H	D_1	1 Hz	122.1 μs
C.S. = H	D_2	1/60 Hz	122.1 μs
$A_0 \sim A_3$ = H	D_3	1/3600 Hz	122.1 μs

Courtesy OKI Semiconductor, Inc.

(1) 1024 Hz signal at D_0 not dependent on HOLD input level

modes of operation. We will be using Mode 1 which is a programmable one-shot, and Mode 2 which is a divide-by-N rate generator. The control word is defined as shown in Table 5-2. For the No. 0 counter, we want to use both bytes, set it for Mode 1, and want binary as opposed to bcd operation. For this, our control word is 00110010_2 or 50_{10}. For the No. 1 counter, we desire the same operation, except in Mode 2 which gives us 01110100_2 or 116_{10}, while for the No. 2 counter we get 10110100_2 or 180_{10}. These, along with the control address of 35, are shown in line 2515 of Example 5-1.

Table 5-2. 8253 Mode Control Words

Control Word Format

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

Definition of Control

SC—Select Counter:

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Illegal

RL—Read/Load:

RL1	RL0	
0	0	Counter Latching operation (see READ/WRITE Procedure Section).
1	0	Read/Load most significant byte only.
0	1	Read/Load least significant byte only.
1	1	Read/Load least significant byte first, then most significant byte.

M—MODE:

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD:

0	Binary Counter 16 bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

Courtesy Intel Corp.

To load the counters with their initial values they must be loaded least-signficant byte (lsby) first, and then the most-significant byte (msby). For numbers less than 256, the least-significant-bit (lsb) is equal to the number and the most-significant-bit (msb) is zero. For numbers 256 or greater (up to 64,535), divide the number by 256 and the integer portion of the result is the msb. The lsb is the number minus 256 × msb. In other words, the number is:

$$\text{number} = \text{msby} \times 256 + \text{lsby}$$

Our clock reference is 1024 pulses per second (pps), and let us assume that we want to count for 10 seconds. The number we wish to load in counter No. 0 is then 1024 pps times 10 seconds or 10240. The msb is the integer portion of 10240/256 or 40. The lsb is 10240 − (40 × 256) or 0 in this case. They are loaded in the lsby, msby order, or 0 and then 40.

To simulate an up-counter one can load in a very large initial value and count down. At the end of the counting period the final count can be read and subtracted from the initial value for the result which would be equivalent to an up-count. The largest value that can be loaded directly is 11111111_2 lsby and 11111111_2 msby or 65,535 decimal. This is loaded as 255_{10}, 255_{10}. The counters automatically carry around, however, so if 00000000_2, 00000000_2 is loaded a single count will bring up 11111111_2, 11111111_2. Thus, if one loads 0, 0 then one has effectively started with 65,535 + 1 or 65,536, and this is what we will do.

As the addresses of Counters 0 and 1 are 32 and 33, respectively, and we wish to load 10240 into Counter 0 and 65,536 into Counter 1, we have the command 2520.

Next, we need to start the counters when the operator is ready. To do this we wait for any key to be pressed (2520 and 2530) and then use a software trigger from 8255 No. 1 Port C (PC1) to momentarily bring the gate high on Counter No. 0. This is done by setting and then immediately resetting the PC1 bit (2540).

We then need to be able to check whether the count is finished or not. For this we have used the gate level of Counters 1 and 2 via Port PC5 of the No. 1 8255 (2545 to 2550). This is a "flag" for the counter status. If it is high it is counting, if it is low it is not counting. We could have used a timing loop in the program to make sure that we waited until the counting was over, but this would have tied up the computer for the 10 seconds, while with a flag we can go do other things, checking the flag on occasion.

As soon as the count is finished, we need to read the clock (2554) and Counters 1 and 2 (2555 and 2556). To get the equivalent (real) up-count we simply subtract from 65,536 (2560 and 2561). The program then returns to (3010) to display the results, and on pressing any key returns to the main menu, completing the sequence.

EXPERIMENT NO. 5
DUAL COUNTER/TIMER

Purpose

The purpose of this experiment is to demonstrate two of the many capabilities of the versatile 8253. In this case we will demonstrate its use (simultaneously) as a timer and as a counter.

Discussion

This experiment uses all the chips in the entire interface in a coordinated fashion. It nicely demonstrates the flexibility and power of a microcomputer and programmable peripheral chips.

STEP 1: Connecting the 8253

A 24-pin DIP socket is placed next to the first 8255, and the connections are made to the TRS-80 bus via the pins on the 8255, as shown in Figs. 5-3 and 5-4. As the V_{CC} and gnd pins on the 8255 were already wrapped three high, these power connections were picked up from the nearby remote interface socket.

The $\overline{CS}$ is connected to Pin 3 of the 75154 address decoder. This corresponds to a base address of 32_{10}.

STEP 2: Run the Experiment

To check the operation of our circuit and program, we need to put in a known reference signal into the clock inputs of Counters 1 and 2. We can use the 1024 pps output of the MSM5832 for this by running temporary jumpers from Pin 9 (DO) of the MSM5832 to Pins 8 and 12 of the remote interface socket.

If we do this and everything is running properly, then we get a 10-second count that gives 10239 count for Counters 1 and 2. This is one short of what we would have expected and illustrates an important peculiarity of the 8253. It does not use an external clock to drive its logic, but instead depends on the input signals to do this, and this results in the count always being exactly one short. While this is easily taken care of when using counters individually (just add one to the result), if two counters are put in series to get a 32-bit counter, then special care must be taken.

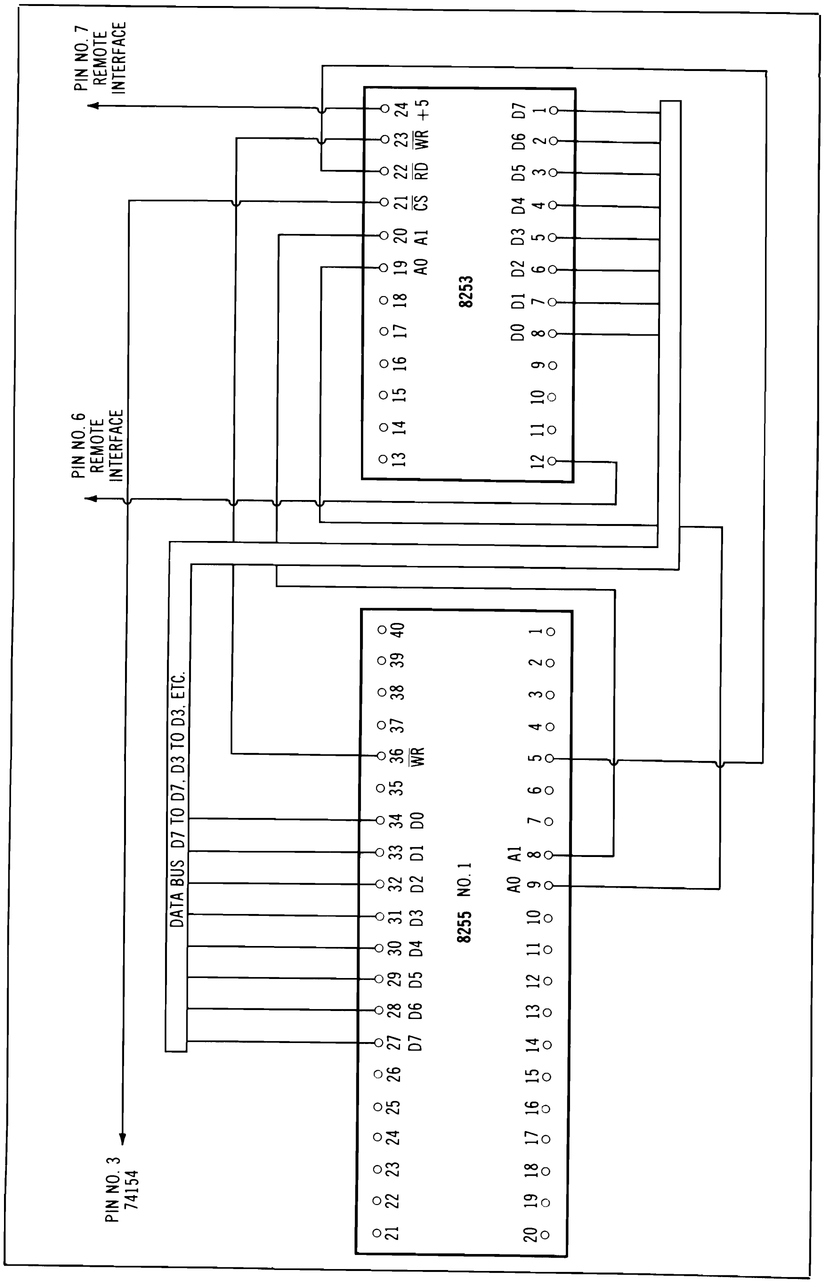


Fig. 5-3. Initial connections for the 8253.

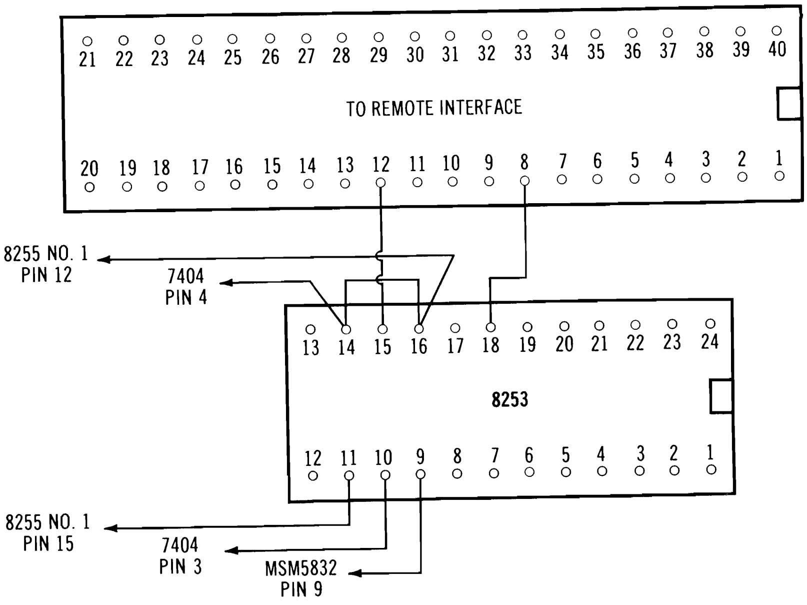


Fig. 5-4. Remaining 8253 connections.

STEP 3: Troubleshooting

The author's program did not run the first time. With everything running, the voltmeter (10-volt scale) was put on Pin 9 of the MSM5832. With a 1024 pps signal of 50 percent duty cycle, the meter should have read about 2.5 volts. Instead, it read 0 then 5 volts then 0, etc., in 1 second swings. Careful checking revealed that I had forgotten to put out the address 1111_2 to the MSM5832. This was added (2510) OUT 65, 15. The program still didn't run and in checking things, it seemed like there was no detectable trigger pulse (2540). This was traced to using PC0 instead of PC1 and re-writing the line cured the problem and also showed a very slight but detectable movement of the voltmeter needle when the program flashed by this line.

We are now all set to count beer bottles from two production lines simultaneously. One might recall the "down-counting song" that goes "99 bottles of beer on the wall, if one of those bottles should happen to fall, 98 bottles of beer on the wall. . . ." Perhaps you are more interested in measuring voltages from external sensors? A counter can do this if we first convert the voltages into variable frequency pulses.

CHAPTER 6

Voltage/Frequency Converters

In typical control systems there are various sorts of sensors such as temperature sensors, pressure sensors, etc. The outputs from these sensors are often varying voltages, or varying resistances, which can easily be converted into varying voltages. A digital control system cannot handle these analog voltages directly, so provisions must be made for converting them into a digital form.

The first thing that comes to mind, of course, is an analog-to-digital converter (a/d). These devices work very nicely, and many are available that may be directly interfaced to a microcomputer. If high resolution is required, a/d converters can be a bit expensive, and if both high resolution and high speed are required, they can be very expensive. Some of the lower-cost a/d converters have been nicely covered in TRS-80® Interfacing, Books 1 and 2 (Howard W. Sams & Co., Inc., Indianapolis, IN 46268), and the reader who is interested in using lower-cost a/d converters is directed to these books.

Another approach to digitizing analog signals that is sometimes overlooked is the use of a voltage-to-frequency converter (v/f). These devices (even the best ones) are not expensive, and they put out a train of pulses, the frequency of which is proportional to the analog voltage that is applied to them. In many applications, one does not want the *instantaneous* value of the analog voltage, but would like the *average value* over some short period of time, such as 10 seconds. Such an average value is much less affected by noise and random fluctuations that often are present along with the "real" signal of interest.

While one could use an a/d converter to sample a signal 100 times a second for 10 seconds, and then calculate the average value of these 1000 samples, it is faster and more accurate (for the same amount of money) to use a v/f converter and simply count the pulses for exactly 10 seconds. A change in the average value of the unknown voltage will be directly reflected by a proportional change in the total count for 10 seconds. The main reason that a v/f converter is not used more often for analog-to-digital conversion is that one has to have both a counter and a precisely timed counting interval to use the v/f converter. In our case we have both, and by adding a v/f converter we will have a capability of measuring signals with a resolution of 16 bits (and an accuracy that depends on the linearity of the v/f converter). This capability includes automatic averaging. As long as we don't need to find average signals faster than a few per second, as is our case, this is a good approach to turning analog signals from sensors into something that can be directly measured by our computer.

When sensors are remotely located from the computer, as is our case, then v/f converters are really quite preferable to a/d converters. If one uses an a/d converter, then it normally would have to be physically located on the interface next to the computer, as extending the high-speed data and address lines from the computer bus out for hundreds of feet is not something one would even want to contemplate. With the a/d converter near the computer, this would mean that the analog signals would have to be brought from the remote sensors to the a/d converter. The possibilities for noise pick-up, small offset voltages, line inductance or capacitance, etc., are almost endless. On the other hand, if one places a v/f converter right at each sensor, the signal is immediately converted to a series of digital pulses, and these can be transmitted over even very long distances without any degradation.

THE ANALOG DEVICES AD537 V/F CONVERTER

The analog device AD537 v/f converter is a low-cost device that has a linearity of 0.05 percent, and can operate over an 80 dB range of signal levels. Not only can it be used as a v/f converter, but can be used as a f/v (frequency-to-voltage) converter, a resistance-to-voltage converter, and a temperature sensor.

In your control system you might want to have a number of these devices. They can, for instance, be used to sense angular position with quite good accuracy if connected to a high-resolution single-turn potentiometer. A good type of potentiometer for this is the Beckman series of "plastic pots." These have as good a linearity as wire-wound potentiometers and also have essentially infinite resolution. They cost about $20.00 (1981).

Other AD537s could be used for temperature sensors to control chemical reactions, check for overheating of critical parts, etc. As the number of external sensors grew, one could either keep adding more 8253 programmable counters, or if one only needed to check one sensor at a time, the output of an 8255 could be used to control AND gates to switch in (i.e., multiplex) between the various sensor v/f outputs. A 1-of-8 or 1-of-16 encoder would also do this nicely.

Of course the AD537 can be used to measure voltages, and this is what I have used it for in my system. In my case, the voltage came from a dc amplifier, and was proportional to the intensity of light received from a star being observed. As voltages can represent many different quantities from different types of sensors, this is the most general case, and will form the basis for our experiment.

EXPERIMENT NO. 6-1
THE TRS-80 AS A DIGITAL VOLTMETER

Purpose

The purpose of this experiment is to demonstrate the use of a v/f converter in conjunction with a counter and precise interval timer to measure voltages from remote sensors.

Discussion

The hardest part of this experiment has already been performed previously as the Dual Counter/Timer experiment. In this earlier experiment we used one section of the Intel 8253 as a precise 10-second timer (or any other amount of time set in software). The other two sections were used as counters. The program developed in the example read the number of counts received over the counting interval. This is exactly what we need now, so we won't, of course, write another program, since the one we already have will do the job. All that is necessary is that we properly connect up an AD537 and connect its pulse output to the inputs of one of the counters.

STEP 1: Connect up the AD537

Connect up an AD537 v/f converter as shown in Fig. 6-1. The AD537 should be placed as close to the sensor as is possible to minimize the chance of noise pick-up. In many cases, it is possible to build it in as part of the sensor itself.

The 0.001 μF capacitor should be of high quality and have a very low temperature coefficient. The two potentiometers shown may be of the trim pot variety. Layout is not critical, and a small perfboard would be suitable for mounting the parts. The output should be transmitted to the computer interface via a coaxial cable if possible. If very long distances are involved, the use of line drivers and receivers should be considered.

STEP 2: Adjustments

If a high impedance earphone is connected between the output and ground (or a small speaker through a 300 ohm resistor), you will be able to hear the actual pulses coming from the v/f converter. As an input for testing, you can connect a 10K potentiometer between 5 volts and ground and take a variable voltage off the wiper arm as the input to the v/f converter. Varying the position of this potentiometer

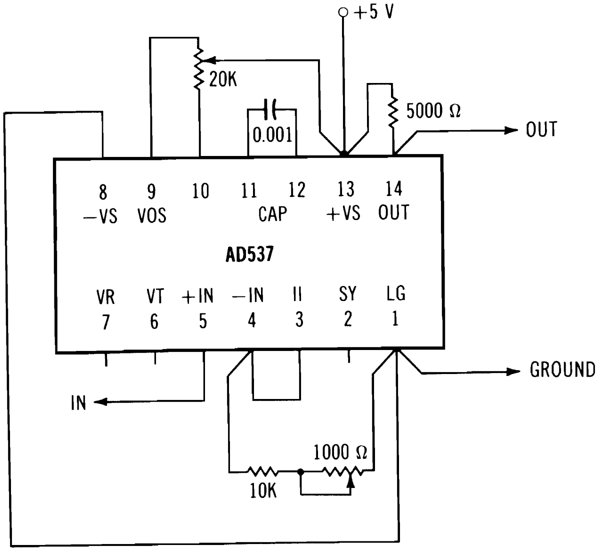


Fig. 6-1. AD537 voltage-to-frequency converter wiring schematic.

should cause the output frequency (tone) to change from very low to quite high. The 1K and 20K potentiometers on the v/f converter can then be adjusted. The 1K potentiometer sets the zero point, while the 20K potentiometer adjusts the maximum frequency.

The maximum frequency best to use is a trade off between dynamic range and resolution on one hand, and linearity on the other. Greatest linearity is achieved if the maximum (full scale) frequency is kept below 10 kHz, while the greatest dynamic range is achieved by using a full-scale frequency of 100 kHz. For most situations it is best to stay at or below 10 kHz. I find it handy to be able to actually hear the output, and my hearing above 10 kHz has been badly dulled by too many hours spent flying small airplanes.

STEP 3: Connect and Run

With the v/f running and adjusted, connect the v/f output to the input of the Intel 8253 counter. Load in the program from the counting experiment, and presto, your computer is now a digital voltmeter.

EXPERIMENT NO. 6-2
WORLD'S MOST EXPENSIVE THERMOMETER

Purpose

The purpose of this experiment is for you to develop some confidence on your own in using the AD537 as an external sensor.

Discussion

The AD537 has been cleverly designed so that without any additional sensors, it may be used as a remote temperature sensor. The output frequency of the AD537 can be made to be proportional to the absolute temperature. Absolute temperature, for those of you who have (like me) forgotten your physics, is measured from absolute zero. It can be converted into other, more familiar scales with a little computer algorithm.

The details of this experiment will be left to you, but here are a few hints. First, check a v/f source for a good discussion. Second, dust off that old physics book to see how to convert between temperature scales. Third, the already developed counter program can form the core of your software. Finally, I don't recommend that you stick the AD537 in your mouth even if you do feel a bit feverish! Now you are on your own.

CHAPTER 7

Stepper Motors

So far we have been concerned with digital interfacing, remote communications, keeping time, and counting. While all this is fine, we haven't seen the computer actually move something, and it is computer-controlled movement that is most impressive to watch.

The easiest way to get computer-controlled movement is to simply connect a TTL-compatible solid-state relay to an output bit on one of the 8255 ports and turn a motor, or other power-handling device, on and off. Another bit and a similar relay can be used to control the direction of the motor. To position the shaft of the motor with any precision, it is necessary to sense its position with some device such as a small switch actuated by a cam, or a potentiometer and an analog-to-digital converter. For many years, this was the primary approach to movement in control systems, especially when these were primarily analog systems.

With the widespread use of digital computers in control systems, a new "digital motor" has seen increasing use and is now the prime generator of movements in modern control systems. The digital motor is, of course, known as a *stepper motor.* Not wishing to waste any time on antiquated approaches, we will concentrate our attention entirely on stepper motors as our generators of motion.

TYPES OF STEPPER MOTORS

Commensurate with their important status, stepper motors, or "steppers" as they are usually called, come in many varieties. A major division is rotary and linear steppers. Rotary steppers turn an output shaft a precise amount for each step, while linear steppers extend a shaft a precise amount for each step. Linear steppers are actually just a specialized form of rotary stepper with a rotating element that advances or retards a threaded shaft.

Steppers come in many sizes from tiny units that weigh just a few ounces, to monstrous units that weigh over 100 pounds. Some steppers have a straight output shaft directly connected to the rotor, while others have built-in gear reducers. For the rotary steppers, different steppers have different step angles, depending on the number of poles in the motor. Some of the more common step angles (ungeared) are 1.8, 7.5, and 15.0 degrees. A wide selection of gear ratios is available to reduce these step angles by factors of up to 3000. Some of the more popular types of stepper motors are made by Hurst, Airpax (North American Phillips), and Slo Syn.

SELECTION OF A STEPPER MOTOR

As with component selection for most applications, selecting a stepper is a trade off between many parameters, some of which are conflicting. Important parameters include step angle, step precision, stepping speed, torque, and cost. For applications where large step angles, low torque, moderate speed, and low cost are important, then there are a number of Hurst and Airpax steppers available for around $40.00. For smaller step angles, higher speeds and greater torque, the more expensive Slo Syn motors may be more appropriate (typically $130 and more). However, Slo Syn motors can often be purchased from various ''surplus'' outlets for $30.00 to $50.00 and are a very good general purpose choice.

STEPPER CONTROLLERS

Unlike other motors, steppers require a digital controller. Most steppers have four different windings, and in order to get the motor to step, these windings have to be energized in a specific sequence. The normal full-step sequence is shown in Table 7-1. Note that at any time two windings are always energized, but the two change from one step to the next. To reverse the direction of the motor, one simply runs through the sequence in the opposite direction.

One approach to energizing the windings is to use a small computer and to have a routine in the software that generates the

Table 7-1. Full Step Sequence

Step	SW1	SW2	SW3	SW4
1	ON	OFF	ON	OFF
2	ON	OFF	OFF	ON
3	OFF	ON	OFF	ON
4	OFF	ON	ON	OFF
1	ON	OFF	ON	OFF

proper sequence of control signals. Several output bits on an 8255 PPI along with appropriate current amplification could be used to drive the motor. While this is a simple approach, it can tie up a computer somewhat. This is particularly true in our case where we are operating in BASIC without interrupts. If we were to use this approach, it would only step motors very slowly indeed.

Another approach is to have the computer provide a pulse train for steps in one direction, and another pulse train for steps in the opposite direction, at a rate of one pulse per step. This requires external logic circuits to translate the forward or reverse pulses to the proper sequence of energized stepper windings. In BASIC, moderate speeds of about 25 steps/second can be achieved, and this is entirely adequate for most control functions.

If higher speeds are desired, and one wants to still program in BASIC, then the Intel 8253 programmable counter can be used in conjunction with an on-board oscillator, such as the reference signal from the OKI clock chip. The basic strategy here is to load the number of steps you want into the 8253, and let it take care of counting out the steps at any speed desired, automatically closing a gate and stopping the steps exactly at the terminal count. This way, not only can you step the motor as fast as it will go, but the computer is not tied up waiting for the motor to reach its final position, before it can go do other things.

If the device being turned by the motor has a high moment of inertia (is on the massive side), one must either step slowly (perhaps with a geared stepper to reduce the load), build up to speed gradually (called ramping) and slow down gradually, or use a larger stepper with higher torque than would be required if ramping were not used. Most steppers tend to lose torque as they are stepped at a faster speed, and at some point they will start missing steps if one tries to step them faster than their capabilities allow. How fast a given stepper will step before it starts missing steps is quite dependent on the driving electronics.

HURST EPC-011 STEPPER MOTOR CONTROL

The Hurst stepper controller is very convenient to use, and is also not expensive. It can be obtained for less than $40.00 (1981) from Hurst Mfg. Corp., Box 326, Princeton, Indiana 47670. The controller board plugged into the socket provided with the board is shown in Fig. 7-1.

The schematic for the board is shown in Fig. 7-2, and a typical noncomputerized hookup is shown in Fig. 7-3. The heart of the controller is the SAA1027 chip, made by North American Phillips. This chip provides all the logic functions for driving a stepper motor, as well as some output amplification. Except for the smallest motors, however, additional power amplification is required, and the Hurst controller provides this with the four MJE180 transistors.

In the typical noncomputerized hookup shown in Fig. 7-3, it can be seen that there are three controls. These controls are the direction of rotation (CW or CCW), run or hold, and external rate adjust (i.e., the motor speed). The run/hold allows greater torque when holding the motor from turning when not running, but using it can cause the motor to skip a step and we will not be using it at all. To

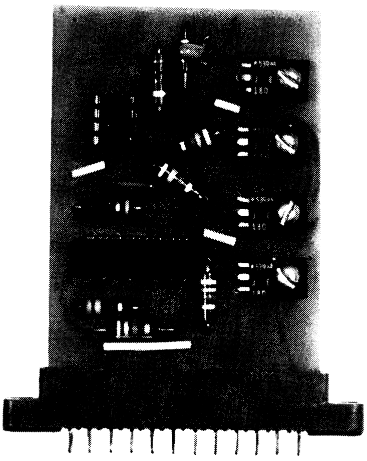


Fig. 7-1. Hurst stepper controller board. (Courtesy Hurst Mfg. Corp.)

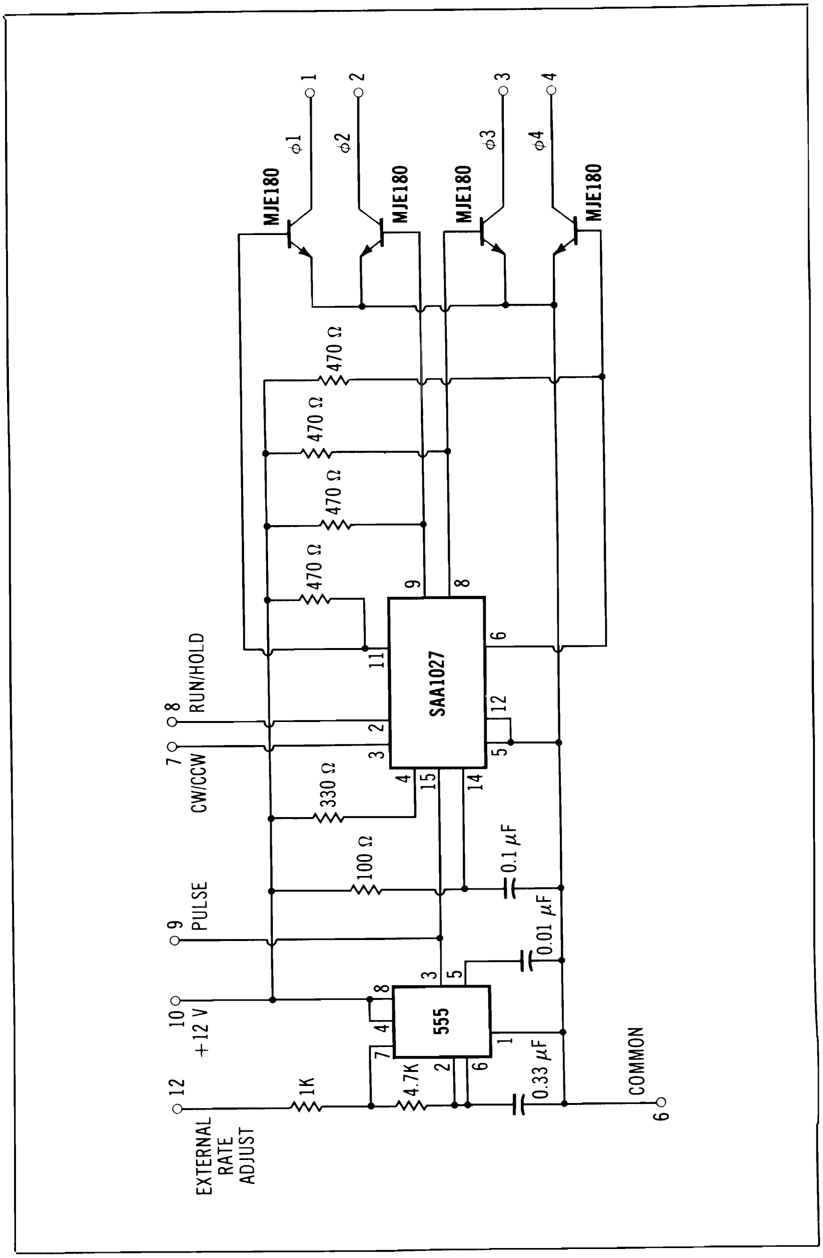


Fig. 7-2. **EPC-011 stepper controller schematic diagram.** (Courtesy Hurst Mfg. Corp.)

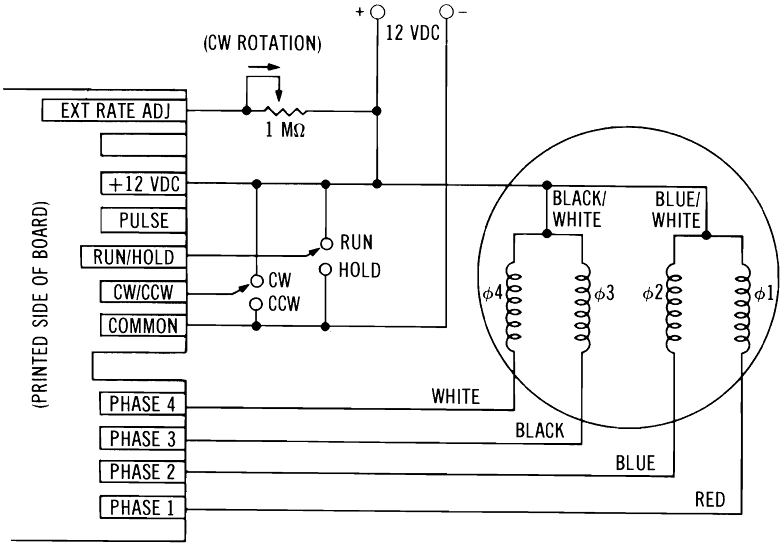


Fig. 7-3. Typical noncomputerized controller application. (Courtesy Hurst Mfg. Corp.)

defeat the run/hold we will simply connect the run/hold to +12 V dc at all times. The external rate adjustment is simply a variable resistance that sets the frequency on a 555 used as a pulse source. As we will be using pulses from the computer, we will not use the external rate adjust, but will instead input pulses to the "pulse" connection, pin No. 9.

The Hurst controller operates off 12 V dc, and the logic levels of 0 and 12 volts are not directly compatible with the 0 and 5 volt logic levels in our interface. This can be taken care of by using an open-collector inverter, such as the 7407. This TTL device has 0 and 5 volt logic on the input, but the output is determined by an externally connected voltage which, in this case, will be 12 volts.

The Hurst controller is easy to use, low in cost, and can save lots of time and effort. It is particularly easy to use with the Hurst 12-volt steppers, but can be used with other steppers of lower voltage if two resistors of appropriate value are placed in the power return lines from the motor. Note: Just prior to publication, Hurst released a hybrid motor controller chip, Hurst part No. 220001. This 20 pin chip accepts TTL logic levels from the computer directly and outputs go directly to Hurst steppers. Everything you need is on a single chip.

DIRECT USE OF THE SAA1027 WITH AIRPAX STEPPERS

With the smaller Airpax steppers where high-stepping rates are not needed, the SAA1027 chip can be used to drive the stepper directly. The diagram for doing this is shown in Fig. 7-4. As with the Hurst controller, this is 12-volt logic, so a 7407 must be used in connecting this to the computer interface.

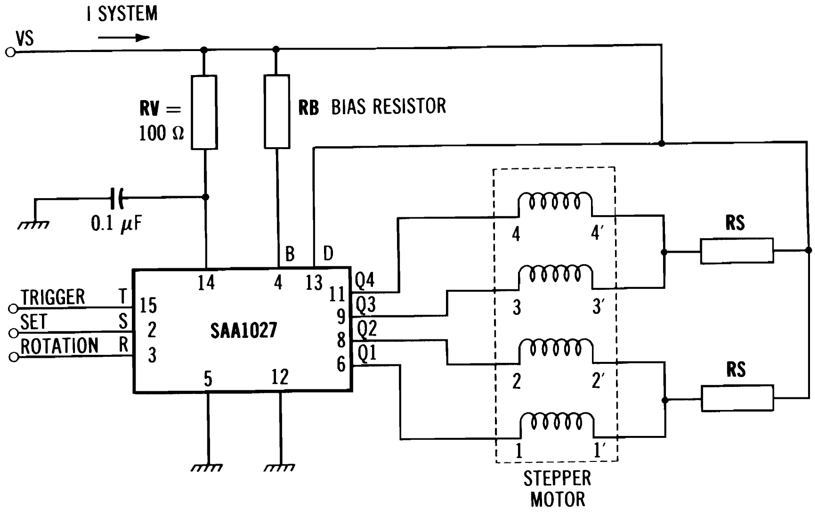


Fig. 7-4. Typical SAA1027 driver application. (Courtesy Airpax North American Philips Controls Corp.)

HIGH PERFORMANCE STEPPER CONTROLLER

The stepper controller shown in this section was originally designed by Dr. Frank Melshiemer of DFM Engineering in Boulder, Colorado. Frank designs and builds large telescopes and special instrumentation. This particular design will operate a fairly hefty Slo Syn stepper at 1000 steps per second from a dead start without missing a step.

The stepper controller was laid out on a perfboard as shown in Fig. 7-5 (top view). The controller is wired as shown in Fig. 7-6 (bottom view).

The 74123 monostable circuit provides two one-shots for the forward and reverse pulse trains. This provides some immunity against noise and provides a clean signal for the 74193 up/down

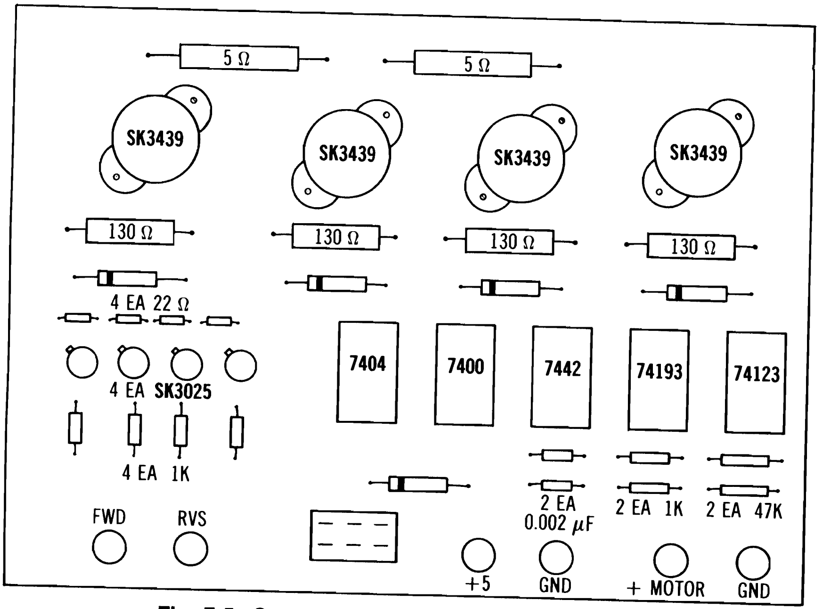


Fig. 7-5. Stepper Controller layout (top view).

counter. While the outputs of the 74193 are four lines with varying logic levels, these are not provided in the sequence needed by the stepper. However, an added 7442 decoder and 7400 NAND gates cleverly provide just the sequence desired. The 7404 is merely a buffer used to drive the low-power transistors which, in turn, switch the high-power transistors on and off, and step the motor.

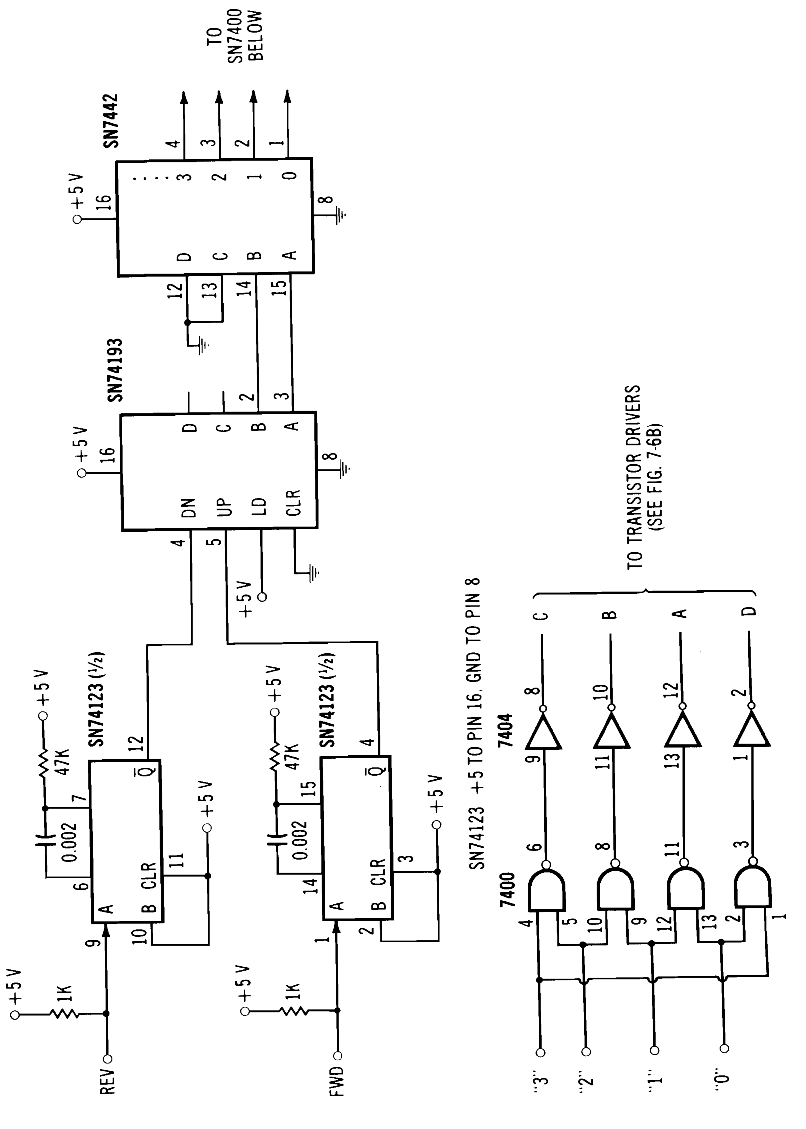
EXPERIMENT NO. 7-1
COMPUTERIZED WINDSHIELD WIPER

Purpose

The purpose of this simple experiment is to demonstrate computer control of a stepper motor through the computer interface.

Discussion

With a stepper connected to one of the controllers discussed earlier in this chapter, all we need is to tell the stepper which direction we wish it to go, and then provide a pulse for each step. This, of course, is no more complicated than turning a light on and off, and



(A) Controller schematic—Part 1.

Fig. 7-6. Controller

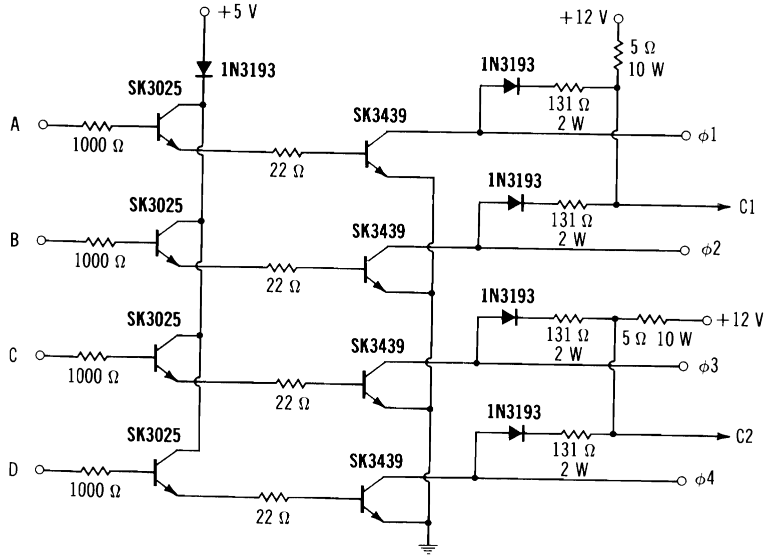
as we know how to do this already, it is merely a case of applying what we already know. The high performance stepper controller, which was the basis for this experiment, simply has just two inputs. One is for pulses in one direction, and the other is for pulses in the other direction. While the software example in this experiment was written for this particular case, by understanding how this operates, one could write a program with relative ease for one of the other stepper controllers discussed previously.

STEP 1: Get the Controller Working

The first thing you need to do is to connect up one of the controllers discussed earlier in this chapter, and make sure that when pulsed with 5 volts it steps. You can check this by just repeatedly touching a wire connected to 5 volts to the "pulse" input.

STEP 2: Connection to the Interface

We need to have the computer generate forward and reverse pulses. You probably recall that the computer generated pulses in the first experiment when a light was turned on and off, so if you



(B) Controller schematic—Part 2.

schematics.

remove the light you can obtain the pulses at pin 1 on the auxiliary 14-pin DIP socket. You also need some "reverse" pulses, preferably buffered by a 7404 and you can get these if you wire the addition to the computer interface as shown in Fig. 7-7.

The stepper motor also needs to be wired to a plug that matches the one on the control board. This can be done as shown in Fig. 7-8. Wires can then be run from Pins 1 and 7 of the auxiliary socket (just poke them in the top) to the forward and reverse inputs on the stepper controller.

STEP 3: Program

To demonstrate stepper operation we will step the motor a selectable number of steps in one direction, pause, step back to the original position, pause, etc. To do this we first need to configure the No. 1 8255 exactly as we did in our first experiment:

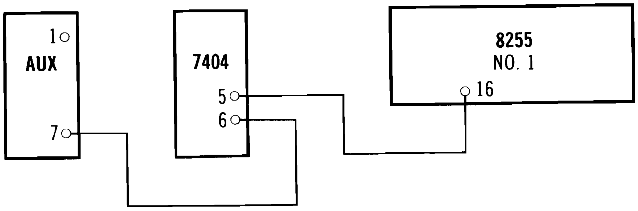


Fig. 7-7. Reverse pulse schematic diagram.

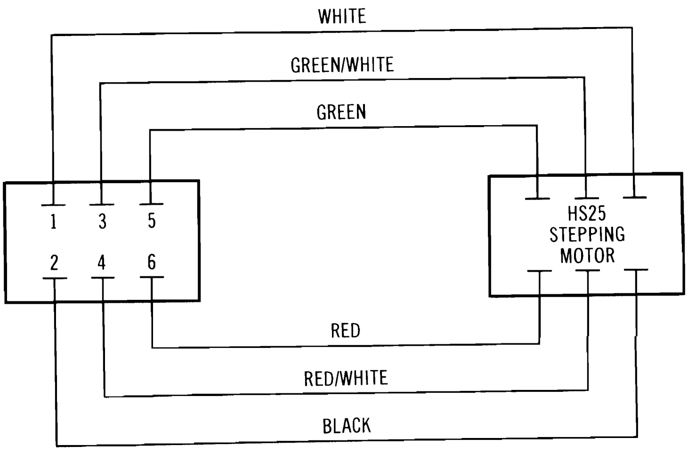


Fig. 7-8. Motor schematic diagram.

```
10 OUT 131,136
```

Then we will find out how many steps the operator wants

```
20 INPUT"STEPS"; S
```

and then count out the steps

```
30 FOR I=1 TO S
```

and for each step we will set and reset bit PC0

```
40 OUT 131,0: OUT 131,1: NEXT
```

then pause

```
50 FOR I=1 TO 500: NEXT
```

then do the same thing only for Port PC2

```
60 FOR I=1 TO S
70 OUT 131,4: OUT 131,5: NEXT
80 FOR I=1 TO 500: NEXT
```

then start all over again

```
90 GOTO 30
```

A complete listing is provided in Example 7-1.

I tried this out for the first time and the stepper didn't even move. A fast check revealed that I had forgotten to plug in the stepper. I called the wife in to impress her with my accomplish-

Example 7-1. Stepper Program

```
10 OUT 131, 136
20 INPUT "STEPS"; S
30 FOR I = 1 TO S
40 OUT 131, 0: OUT 131, 1: NEXT
50 FOR I = 1 TO 500: NEXT
60 FOR I = 1 TO S
70 OUT 131, 4: OUT 131, 5: NEXT
80 FOR I = 1 TO 500: NEXT
90 GOTO 30
```

ment and she suggested that the windshield wiper on the car could do that and certainly at lower cost.

EXPERIMENT NO. 7-2
GOING UP?

Purpose

The purpose of this experiment is to demonstrate some of your creative talents in real-time control.

Discussion

The "windshield wiper" experiment certainly did not exercise the capabilities of your control system interface and computer. In this experiment you need to plan out an application and then develop the necessary software.

In case you don't already have an idea of your own, here is one for you to consider. Place a small diameter drum on the shaft of the stepper (dowel rod would do). Wind some string around the drum and fasten the loose end to a miniature "elevator." On a piece of cardboard behind the elevator mark off some "floors" evenly spaced (perhaps cut in a door at each floor).

For the control program, have the monitor say "you are at floor X, what floor would you like to go to?" The desired floor would be entered on the remote keypad, and the stepper motor would then move the elevator to the desired floor. Obviously, you will need an algorithm to figure out, given that you are at floor X and want to go to floor Y, how many steps in which direction that you need to go.

Once you get this program working, then you could add some nice touches. One would be to display what floor you are at as you pass by it. Another would be to have a directory of what is on each floor in the "building." Then, of course, you could add a button by each door to signal the computer that someone wants on at this floor. With multiple people wanting to use the elevator then an algorithm could be developed to determine the most efficient way of picking them up without any one person waiting too long.

CHAPTER 8

You Can Do It!

The age of the low-cost microcomputer is upon us. In much less than a decade they have grown from a kit offered by a small firm in Albuquerque and purchased by a few experimenters, to hundreds of thousands of systems bought by all sorts of people. It has been projected that by 1985 there will be 40 million microcomputer systems in operation.

The impact of this vast proliferation of computer systems will surely be immense, even something of a minor revolution. In other areas, the small one-chip microprocessors are creating their own revolution, but this is a revolution of a different kind entirely. Microcomputers, as opposed to microprocessors, are full-fledged, complete computer systems, capable of being programmed and operated by many different persons with little if any specialized training. Much of the programming ease of microcomputer systems is due to the flexibility and ease of using interpretive BASIC.

Control systems started out as assemblages of relays and other components. The coming of large mainframe computers did not have much of an impact on control systems, as they were generally too expensive and unreliable to apply to this task, and in any event control systems tended to require dedicated computers. With the coming of minicomputers, computerized control systems began to make serious inroads on their relay-controlled predecessors. The cost of minicomputers was in the tens of thousands of dollars, and programming was usually done by experts in editor/assembler languages. With the advent of one-chip microprocessors, specialized control systems at very low per unit cost became possible, although the initial investment required to develop the software for these controllers remained considerable.

With low-cost microcomputer systems becoming increasingly numerous, it was only natural that they should see increasing use in control systems. Initially, microcomputers in control applications have tended to be treated like microprocessors, with programming done in editor/assembler by highly skilled programmers, essentially ignoring the highly developed interpretive BASIC stored in ROM and using only the smallest part of the large memory capability of the microcomputers. While it can be expected that this approach will be continued by some control system experts in the near future, an approach that more fully utilizes the inherent capabilities of microcomputer systems seems likely to become more prevalent as time passes. The central theme of this more complete usage of microcomputer system capability is the exclusive use of interpretive BASIC in conjunction with highly capable programmable peripheral chips.

Rather than give details on a large number of sensors, actuators, and input/output techniques, a somewhat detailed description of an actual control system used in serious work will be given in the second section of this book. This was done because the major pitfalls in implementing a control system are not the details on particular sensors and actuators, but are major flaws in the overall approach. This point is sufficiently important that these typical flaws will be discussed in this final chapter of Section I and the case example in Section II will demonstrate how these pitfalls were actually circumvented.

TRAP NO. 1—UNFAMILIARITY

The first trap for the unwary control system developer is an insufficient knowledge of the situation to be controlled and the problems and viewpoints of the eventual control operator. In many cases, the control system is being developed to automate a previously purely manual process. In these cases the control system developer should become very familiar with the manual process. This is best done by becoming proficient at the manual process itself. If it is a manufacturing process, then the control system developer ought to spend some time working in each of the jobs the control system is going to automate (carefully disguised as a worker to avoid being lynched, of course). The case example in this section is an astronomical observatory that makes observations of the color and light intensity of variable stars, and it was purposely set up as a very conventional, completely manually operated system even though the plan was to eventually automate much of the process.

TRAP NO. 2—GETTING CARRIED AWAY

The second trap for the unwary is to try to automate an entire process in one fell swoop. While this approach occasionally succeeds, it is very risky and often ends in failure as well as disillusionment with the value of computers and automation. As a general rule, it is preferable to find which portions of a process are more amenable to automation and which would also provide the greatest benefits in terms of decreased work or increased accuracy. One can then concentrate on these portions, and only after they have been fully debugged and refined, and accepted as being beneficial, tackle the more difficult aspects. This evolutionary approach to things is perhaps the most important key to success. In the case example, as mentioned, the first system was purely a manual one. The system was operated as a manual system for five months and the time and effort taken to do the various tasks studied in some detail. Carefully selected tasks were then automated and the system was operated for another eight months in this configuration. Based on this experience, several additional functions were automated in the third and present version of the system. After sufficient experience is gained with this version, a fourth version will certainly be forthcoming.

There is something very appealing about a fully automated system. This is especially so for those types of persons naturally attracted to computers (the author included). Who is there who hasn't thrilled to Isaac Asimov's appealing stories about robots? However, when one is responsible for developing a serious control system, such prejudices must be set aside and one must strive towards developing the most efficient man/machine system. Complete automation should not be the goal *per se*. If, in the long run, it happens to be the most efficient approach, then fine, but this should not be assumed from the beginning; generally it should only be approached in a gradual manner, and the developer should always remain conscious of his own prejudices.

TRAP NO. 3—WEIRD COMPUTERS

The third trap, and also one that computer enthusiasts tend to be prone to, is wanting to have the latest and fanciest of computers to run the control system, or even worse, developing one's own computer system. Be assured that there will be more than enough problems to challenge the control system developer without in any way attempting to push the frontiers of computers themselves. The TRS-80 Model I was chosen for the computer in the Fairborn Observatory Control System because it was a known entity without any "surprises," and there were available maintenance, software, documentation, and peripherals. No modifications were made to the computer, and only standard peripherals built by the same manufacturer were used (except for the control system interface, of course). There were no problems at all with the computer system itself at any time in the developmental or operational portions of the program.

TRAP NO. 4—IT'S ONLY SOFTWARE

The fourth trap is underestimating the magnitude and difficulty of the software development. If the first three traps are avoided, then the software development should not be too much more difficult and time consuming than all the rest of the development, including all the hardware, put together. If the first three traps are not avoided, then the software development task can quickly balloon to such proportions that either the entire project cannot be done, or if it is done, the results are generally unsatisfactory. Realizing the importance of the software task, the author bought the control system computer even before the first shovelful of earth was turned in the construction of the observatory. This allowed many months of familiarization with the computer before serious planning was done on automating any of the tasks. This approach may be worth considering for your development.

SUMMARY

In summary, if the control system developer is thoroughly familiar with the process to be controlled, approaches computerized control in careful, conservative, evolutionary steps, uses a common well-known computer without modification, and fully realizes the time and effort that must be devoted to software development, then chances are excellent that a very workable control system of gradually increasing efficiency will result. It is the intent of the chapters in Section II of the book to illustrate this process in some detail.

To do this, it will be necessary to give some background in astronomical photoelectric photometry, describe the initial completely manual system and the analysis of its operation (Mark I system), the development, operation, and analysis of the system incorporating some computerized control functions (Mark II system), and the development, operation, and analysis of the Mark III system, the current operational system.

SECTION 2

Control System Development Case Example

CHAPTER 9

Astronomical Photoelectric Photometry

A serious real-time control system application is of necessity rather complex, and to explain how such a system has evolved over time requires some understanding of the particulars of the application. Even for a particular application, the best approach to take for a microcomputer-based real-time control system is highly dependent on many of the specifics in each situation. In my case, for instance, I was not interested in getting entangled in the highly detailed programming that the use of a machine-level editor/assembler language would entail, nor did I ever consider a system that would entail wiring up tens or hundreds of chips. Fortunately, it was possible to do the job in BASIC with just a half dozen chips, and I feel that many other fairly complex tasks would benefit by this approach, although the particulars would vary. By understanding my problem and the evolutionary approach I took to solving it, you will be in a better position to tackle your own particular problem.

Of necessity, any case example is a personal story that can only be told by the control system developer. It is my hope that you will find the story of development of microcomputer control at the Fairborn Observatory both interesting in its own right and also helpful to you in gaining a systems evolutionary perspective. We begin now at the beginning of the story.

REASONS FOR INTEREST

Some of my earliest childhood memories involve astronomy. When I was 7 years old, my grandparents, who lived in Pasadena, California, took me up to see the 100-inch telescope on nearby Mt. Wilson. I can still remember being very impressed by the telescopes on Mt. Wilson, and an equally vivid memory of feeding the very tame deer in the picnic area with watermelon rinds. That summer a number of books were read on astronomy (not all understood), a telescope was built using my grandmother's magnifying glass (she was a stamp collector) and an eyepiece from my little microscope. The images were terrible, but one could see the craters on the moon and the moons of Jupiter. The telescope, along with a telegraph system and crystal radio, were definite neighborhood attractions that summer, and my grandparents never once complained about the horde of kids in the back yard.

On returning to my home in Yucaipa, California, a more proper telescope was assembled after mowing many lawns to pay for the various pieces. It was a 2-inch refractor using an achromatic lens, a drain pipe, and many other miscellaneous parts scrounged from diverse sources. The telescope worked very nicely. Over the following years, while I retained a keen interest in astronomy, it became overshadowed by a succession of radio receivers, illegal transmitters, rockets of increasing size, hydrogen balloons, and various other paraphernalia. All too soon one is an adult, and as an electrical engineer I became involved in the design of guidance systems for those most fascinating of adult toys, big rockets. I never lost my interest in astronomy however, always reading the astronomy articles first in *Scientific American,* and keeping an eye out for new books at the library.

A few years ago, we bought a modest house out in the country in western Ohio. I couldn't help notice how dark it was at night. The Milky Way almost jumped out at one on the nicer moonless nights. Being very careful not to mention it to my wife as the house was in complete shambles and needed repair and additional rooms, I started considering putting together a back-yard observatory. I knew, however, that I would not be content with just casually gazing around, but would want to do some "real" astronomy, just like the overgrown boys I had seen playing with their big toys on Mt. Wilson. Professional astronomers have many ways of defining what constitutes "real" astronomy; but perhaps the most accessible indicator is research that is considered significant enough to be published in leading astronomical journals. With this in mind, I surveyed several leading astronomical journals to see what, if anything, was published based on observations from small ground-based optical telescopes.

What I found was both surprising and encouraging. For this book, I have updated a portion of the original survey, limiting it to the prestigious *Astronomical Journal* for 1975 to 1979. I found that there were 30 papers that were based on observations from at least one telescope or aperture that was 16 inches or less. The observational technique in 28 of the 30 papers was photoelectric photometry. Of the 28 papers based on photoelectric observations, 9 were of eclipsing binaries, 5 were of stellar color (such as cluster magnitude-color arrays), 5 were of asteroids (or comets), and 3 each were of intrinsically variable stars, planetary or asteroid occultations, and miscellaneous.

Based on this admittedly limited survey, it appeared that serious astronomical research was indeed being done with optical ground-based telescopes of 8 to 16 inches aperture. These smaller telescopes were being utilized by advanced amateurs, small colleges and large institutions almost entirely for photoelectric photometry. While eclipsing binaries were most frequently observed, considerable research in the areas of stellar color and asteroids was also reported with slightly less reported in the areas of intrinsic variable stars and planetary occultations.

Based on my survey, it took no great leap of intuition to decide to concentrate on photoelectric photometry. As I had a special interest in stellar evolution, and eclipsing binaries present interesting problems in this area, I decided to concentrate on this particular area of astronomy. The decision was thus made to build, equip, and operate an observatory devoted to making photoelectric observations of eclipsing binary stars.

Before describing my approach to developing the observatory, it would be helpful to discuss eclipsing binary stars, and some fundamental considerations in observing them photoelectrically. This discussion is necessarily brief, and the reader is referred to the following books for more detail: *Photoelectric Astronomy for the Amateur,* F. B. Wood, 1963; *The Photoelectric Photometry of Variable Stars,* D. S. Hall and R. M. Genet, 1981; and *Interacting Binaries,* J. V. Sahade and F. B. Wood, 1978.

PHOTOELECTRIC PHOTOMETRY OF ECLIPSING BINARY STARS

As we happen to live on a planet circling a single star, one tends to get the impression that most stars are single stars. This is not the case at all, and, in fact, about half of the stars are members of multiple star systems, with binary (two star) systems being the most numerous of the multiple star systems. If the planet Jupiter had been somewhat larger, its central pressure and temperature would have been sufficient for thermonuclear fusion, and while dark skies would have been much rarer, binary stars would have seemed normal. For a description of life in a binary system, there is none better than Isaac Asimov's immortal classic of science fiction, *Nightfall.*

Eclipsing binaries are not different from other binaries, except that the plane of revolution of the two stars just happens to lie along the line of sight from our solar system. For us, however, this is important because as the stars take turns eclipsing each other, their combined light is affected in ways that allow us to learn much about them.

Eclipsing binary stars come in all sorts of sizes and shapes. Some of them are so widely separated that it takes hundreds of years for them to revolve around each other, and the eclipses may last months or even years. With such a wide separation, each star of the binary pair pretty much minds its own business, evolving in the same manner as if it were a single star. When the stars are much closer, not only do they have more frequent and shorter eclipses, but matter can flow from one star to the other. Such a mass flow between stars not only changes the time it takes for them to revolve around each other, but it can alter the evolutionary course of the stars. It is thought that in some cases the transfer of mass leads to the spectacular blowup known as a novae. Binary stars close enough for mass transfer to occur are known as interacting binaries.

When eclipsing binaries are side by side (as viewed from the earth) we see the full light from both stars (see (A) in Fig. 9-1). As the brighter star moves in front of the fainter one (see (B) in Fig. 9-1) the light from the fainter star is blocked by the brighter one and the total light we see is reduced—this is the secondary eclipse. The stars then move side by side again (see (C) in Fig. 9-1). When

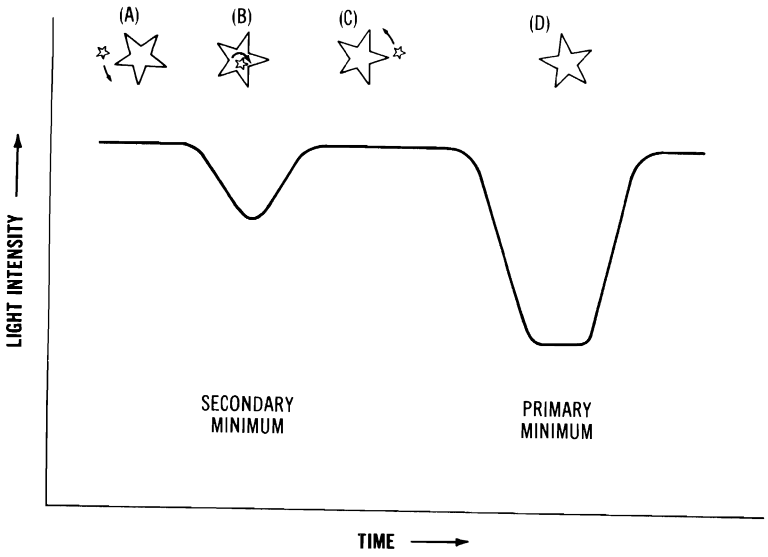


Fig. 9-1. Light curve of an eclipsing binary star.

the fainter star moves in front of the brighter one, the light is reduced to its lowest value (see (D) in Fig. 9-1), and this is the primary eclipse. As one star starts moving across the other, the light is gradually reduced until one star is directly in front of the other. The light is minimum then and as the star moves on the light starts increasing again. The times of minimum light for the two types of eclipses are known as the primary and secondary minima.

The exact time of the primary minimum (which is the easiest of the two to observe) is of great interest to astronomers because as mass transfers between the two stars it causes the period of revolution of the two stars to change, and this is observable as a change in the time of primary minimum. With even a small telescope (6 to 12 inches diameter), a photoelectric photometer, a shortwave radio (such as the Radio Shack Time Cube), and a programmable calculator or microcomputer (to do all the calculations), one can determine the times of minimum of more than a hundred eclipsing binaries with the same amazing precision as the largest observatories. It is a somewhat time consuming process, tying up a telescope and astronomer for many hours. As there are a lot more stars than there are professional astronomers, these stars have not gotten the coverage that many professional astronomers feel that they need. It is because of this (and the several other important uses of photometers on small telescopes) that professional astronomers are so supportive of amateurs equipped for photoelectric photometry.

Photoelectric photometry would be a much more straightforward procedure if we lived on the moon. Most of us, however, must be content to look at the stars through the earth's atmosphere. As might be expected, the atmosphere dims the light coming from a star. This, in itself, wouldn't be so bad, but the amount of the dimming depends on the direction one is looking, being least looking straight up (less air to look through), and most looking towards the horizon (where there is more air to look through). Besides being dimmer towards the horizon, stars also appear redder (just like the sun does). This is due to the scattering of the shorter wavelength blue light by the atmosphere and is the reason the sky is blue during the day. The atmosphere, as anyone can see, is much clearer on some nights than it is on others. The clearness of the atmosphere (transparency) also changes during the course of a single night, and if you aren't on a mountain top in the middle of the desert, it also changes its transparency slightly in just minutes. All this affects the measured intensity of starlight. As if this weren't enough, the brightness of the sky background, against which the stars must be measured, also changes. Our moon (no friend of the photometrist) is the biggest offender, but lights from nearby cities reflected slightly off of high altitude moisture or dust also add to the changing background.

With all these atmospheric effects changing the light from and around the stars, it would at first seem impossible to be able to detect minute changes in the starlight caused by eclipses or starspots. However, by employing a procedure called "differential photometry," the task is reduced from the impossible to merely the difficult. Differential photometry is laborious in terms of the number of observations that must be made, and without a computer, it is very tedious in terms of the extensive calculations and corrections that must be made. The results, however, are exceedingly accurate, even from a small observatory at a poor observing site in or near a city.

The basic idea behind differential photometry is simple enough. A nonvariable (steady) star is selected near the variable star to be studied. This nonvariable star is called the comparison or "comp" star, and it is selected to be similar to the variable star in both color (spectral type) and brightness (magnitude). Observations are then made in the following order: comp star, comp star background, variable star, variable star background, comp star and comp star background. The difference in the brightness between the variable star and the comparison star is then calculated as follows. As a star is essentially just a point of light, it cannot be measured just by itself. As already mentioned, the background changes from night to night and even within a single night. To overcome this, one measures the star and the background together (which can't be avoided), and then immediately measures just the background by itself near the star (making sure there are not any visible stars in the background position!). Subtracting the background measurement from the star plus background measurement leaves one with just the star.

Now, if one could easily measure both the comp and variable stars at exactly the same time (and they were very close together), then the difference in the two star brightnesses (after the backgrounds were subtracted out) would be the differential magnitude (brightness) we are looking for. With just a single photometer, however, we can only make one measurement at a time. However, by measuring the comp star before and after the variable star (equally spaced in time) and taking the average value, one can get close to what the value of the comp star would have been had it been measured at the same time as the variable star.

Even all these precautions are not enough, however, as the comp and variable stars are rarely exactly the same color, and they are not exactly the same place in the sky (thus one has to look through slightly more atmosphere for the lower one). These second order effects can mainly be eliminated by analytic corrections based on a model atmosphere. Finally, dust on the mirrors, slight differences in the responses between different detectors, and other subtle effects must all be taken into account if the observations at any one observatory are to be compatible with those taken at other observatories, or even with observations at the same observatory taken later with a different detector or a cleaned mirror. To take these effects into account, the photometric system must be occasionally calibrated against standard stars, and via analysis an equation for transforming observations to the standard photometric system must be determined and applied to all observations (fudge factors). The most frequently used standard system is the UBV (Ultraviolet, Blue, Visual) system established by H. L. Johnson and W. W. Morgan in 1953.

When all the precautions discussed are properly observed and applied, the result is observations of uncanny accuracy and uniformity from night to night and observatory to observatory.

An example of such observations is shown in Fig. 9-2. Shown on the ordinate is the difference in magnitude between the eclipsing binary 44 Boo, and the comparison star 47 Boo. Shown on the abscissa is the "phase." It is often inconvenient or impossible to get a complete light curve of an eclipsing binary on a single night. As these stars cannot noticeably change their period of revolution over the course of just a few nights, one can combine observations from different nights into a single light curve by moving the various binary 44 Boo, and the comparison star 47 Boo. Shown on the abscissa is the "phase." It is often inconvenient or impossible to get a complete light curve of an eclipsing binary on a single night. As these stars cannot noticeably change their period of revolution over the course of just a few nights, one can combine observations from different nights into a single light curve by moving the various times to an equal starting point. This point is usually the primary minimum, and the orbital revolution from this point is known as the "phase." In Fig. 9-2, results were combined from eight different

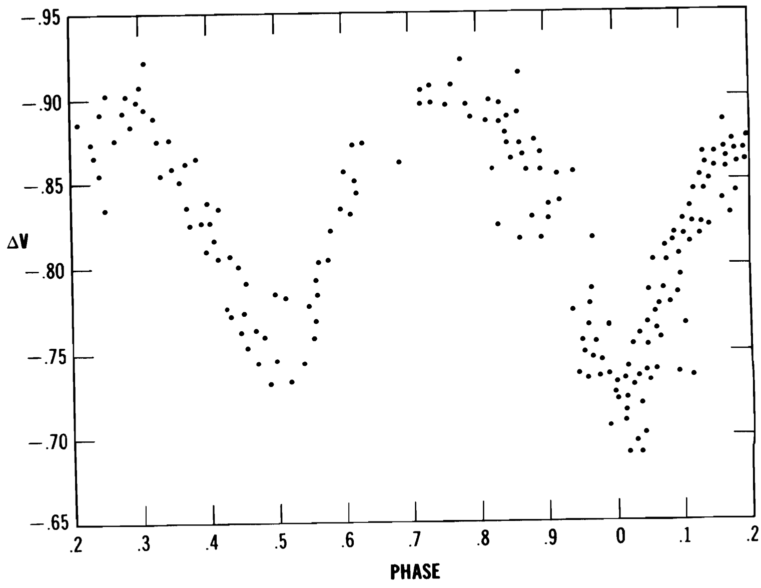


Fig. 9-2. Light curve of 44 Boo.

nights of observations at the Fairborn Observatory. The scatter in the points is due to small, rapid changes in atmospheric attenuation, and also to actual changes in the light of 44 Boo itself (it is notorious for its rapidly changing light curve).

Having discussed why the serious amateur astronomer would do photoelectric photometry, touched briefly on the nature of eclipsing binaries, and mentioned the method of differential photometry for observing variable stars, we are prepared to take a look at an actual photometric system. We are also in a position to note that a microcomputer would be a handy device not only to reduce the work of the laborious calculations involved, but to actually record the data itself, or maybe even help in making the tedious observations themselves.

CHAPTER 10

The Mark I Photometric System

The Mark I Astronomical Photoelectric System was designed and built in seven months of part-time work, and cost about $2,000 (1979) including a new TRS-80 Model I, Level II computer with 16K RAM. In the Mark I system, the TRS-80 was used strictly for data reduction and analysis with all entries of data via the keyboard. The purpose of the Mark I system was to gain experience in operating a conventional, totally manual photometric system.

EQUIPMENT

The Mark I system was housed in a specially constructed observatory 3 miles south of the small town of Fairborn, in western Ohio. Rather than having the more conventional dome on the observatory, there is a roof that rolls off the observatory proper onto a small shop. The roll-off roof is cheaper and less obtrusive than a dome, and one does not have to be constantly adjusting the position of the dome as the stars move or one observes a different star. The telescope is mounted on a reinforced concrete pier that extends below the ground line some 7 feet. The top of the pier can be seen in Fig. 10-1.

To counter the rotation of the earth and make the stars appear to stand still, the main axis of the telescope is aligned on the celestial north pole and a worm gear driven by a synchronous motor turns the telescope in step with the stars. The precision 12-inch worm gear and clutch can be seen to the left of the telescope in Fig. 10-1. Motion about this "polar" axis is measured in hours of Right Ascension (RA).

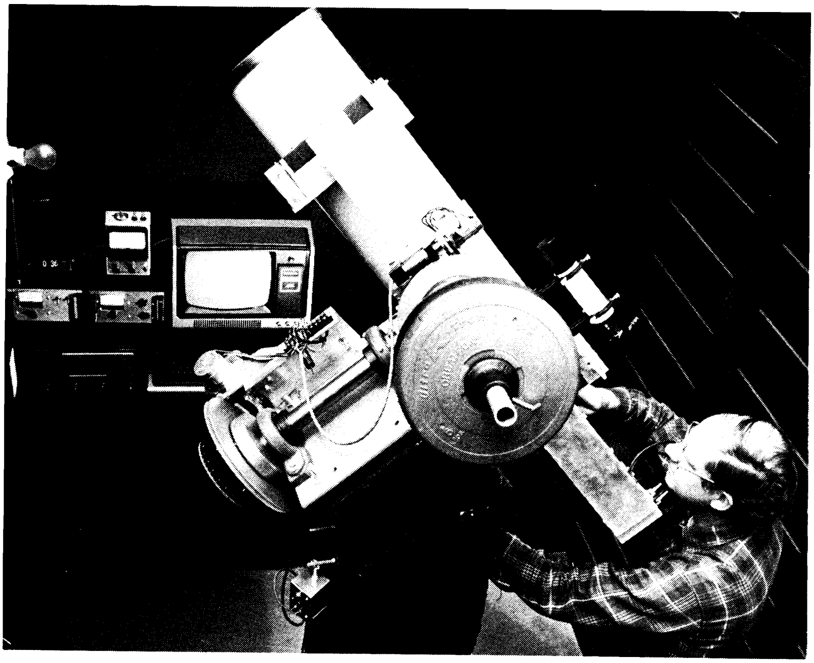


Fig. 10-1. Mark I system—the telescope is mounted on pier.

The second axis, known as the Declination (Dec) axis is perpendicular to the polar axis. On this particular type of mount, the telescope is on one end of the Declination axis, with counterweights on the other end as can be seen in Fig. 10-2. A smaller worm gear and clutch can be seen in Fig. 10-2 just to the left of the pier center along the Declination axis.

The telescope, itself, is an 8-inch f/15 Cassegrain. As diagrammed in Fig. 10-3, the light enters the top of the tube (A), reflects off of the primary mirror at the bottom of the tube (B), travels back up the tube and reflects again off of a smaller secondary mirror (C), travels back down through the tube and passes through a hole in the middle of the primary mirror (D) into the photometer head (E).

The photometer head can be seen more clearly in Fig. 10-4, and is shown diagrammatically in Fig. 10-5. As shown in Fig. 10-5, light

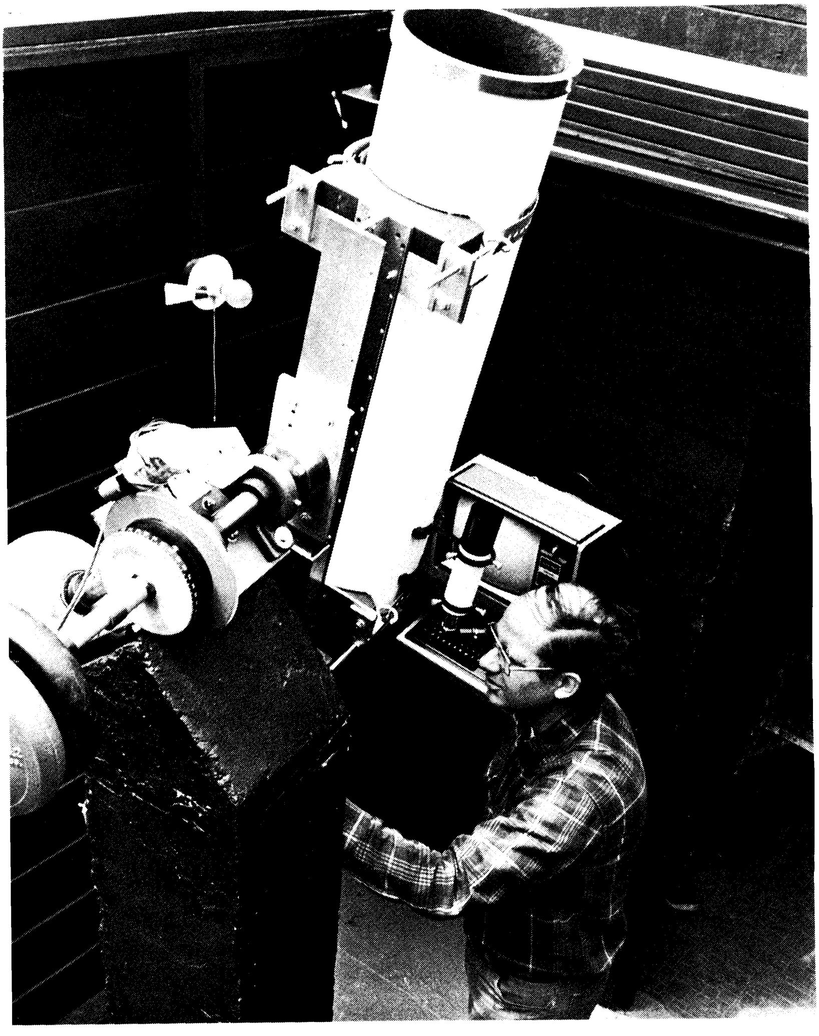


Fig. 10-2. Detail on the declination axis.

from the telescope enters the previewer section first (A). With the first flip mirror moved to the dotted position (B), the light is reflected to the eyepiece on the right (C). This wide-angle, low power eyepiece is used to roughly center the star on illuminated cross-hairs within the eyepiece. Once the star is roughly centered, the first flip mirror is moved out of the light path by rotating it to the

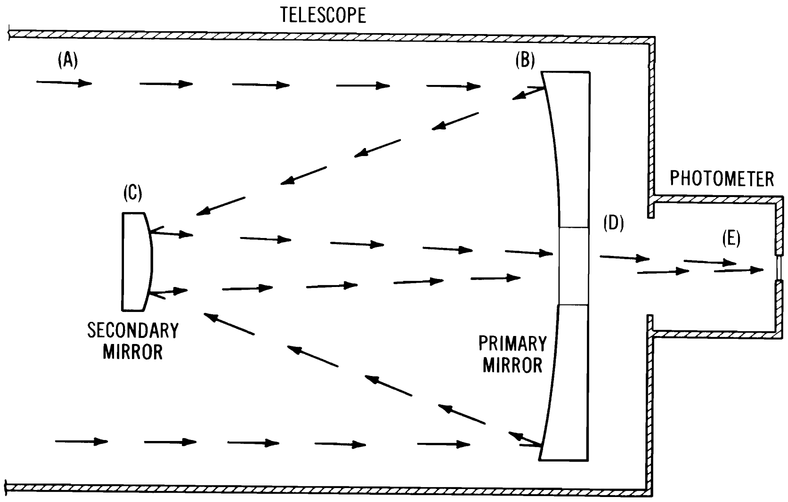


Fig. 10-3. Light path in a Cassegrain telescope.

straight-up position (D). The light then comes to a focus in a small hole in a thin sheet of metal called the diaphragm (E). The diaphragm is mounted on a wheel so that different size diaphragms can be rotated into position to suit seeing conditions and the object being observed. The diaphragm hole is rather small, being just an oversized pinhole, so that only the light from the star being observed will enter and the light from all other stars will be blocked.

Next, the image of the star must be precisely centered in the middle of the small diaphragm. To do this, the second flip mirror is moved to the dotted position (F), and the star and diaphragm are viewed with the high power microscope eyepiece (G). Once the star is centered (very carefully—and here is where the sturdy pier, hefty mount, and accurate drive pay off) the second flip mirror is moved out of the light path (H).

The light then passes through a Fabry lens (I) that not only focuses the light on the photocathode of the photomultiplier tube, but makes the system somewhat insensitive to small motions of the star image within the diaphragm.

The light then passes through special glass filters (J) mounted on a wheel so that different filters can be moved into position. This particular photometric system has filters that isolate the ultraviolet, blue, and yellow (visual) portions of the spectrum. The filters are usually referred to as the U, B, and V filters individually or UBV

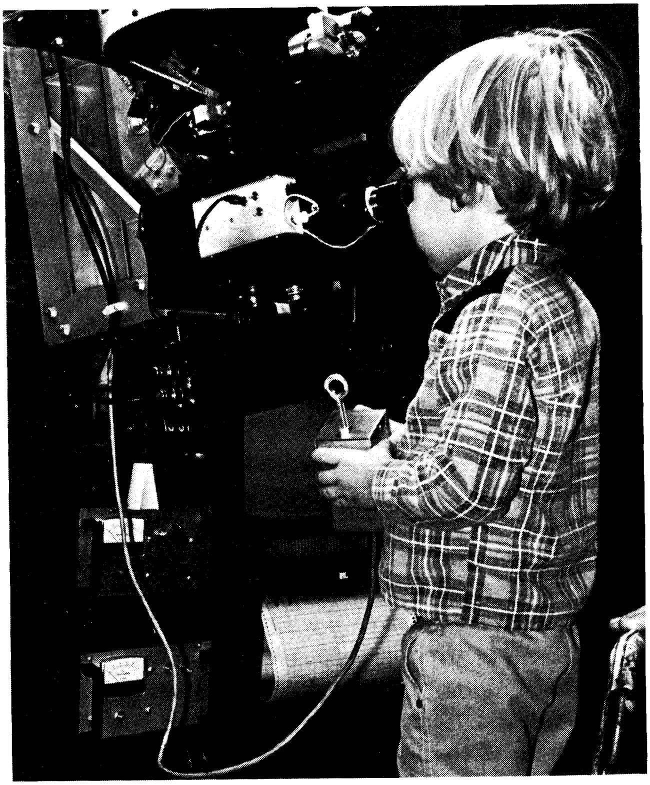


Fig. 10-4. The photometer head and electronics.

filters collectively. By isolating different portions of the spectrum, the photometric system can be used to determine effective stellar temperatures.

The photomultiplier (K) not only converts the light into electricity, but it amplifies it about 1,000,000 times. The photomultiplier used in the Mark I system is an RCA 1P21. Besides the photocathode, it has nine dynodes and an anode. Approximately 1000 volts dc is applied across the tube such that there is about a 100 volt drop between each of the elements. When a photon of light strikes the photocathode, a single electron will be released. The electron is accelerated to the first dynode and on hitting it knocks out about four electrons which are accelerated to the second dynode, where about 16 electrons are released. This continues until there are about 1,000,000 electrons collected by the anode.

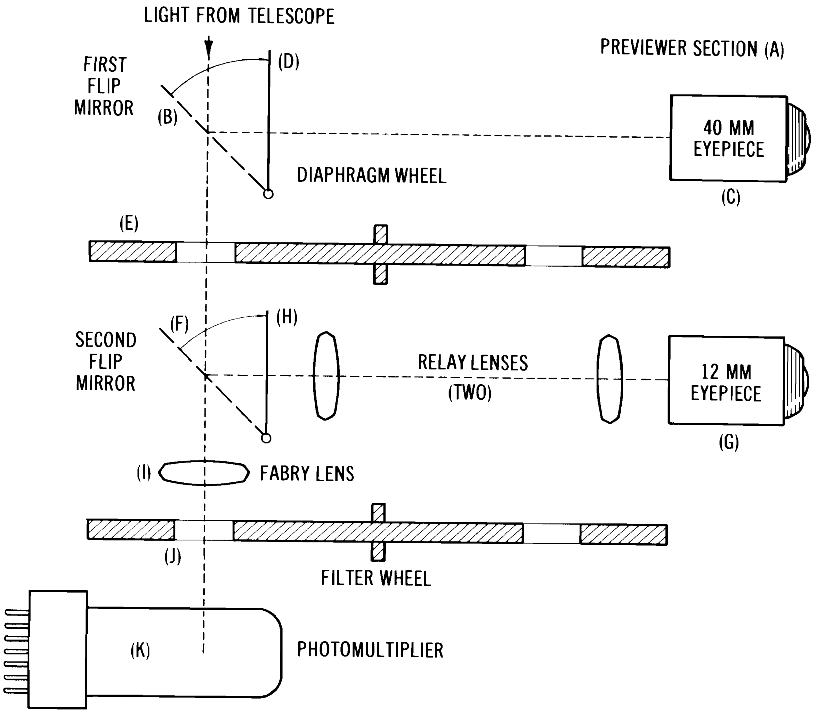


Fig. 10-5. Photometer head diagram.

Starlight is very feeble, however, and even with the million-fold amplification of the photomultiplier tube, the current output is very small, being between a nanoamp and a microamp for typical stars as viewed on smaller telescopes. To be useful, this weak current must be built up with a dc amplifier so that it will be able to operate a meter, strip chart recorder, or other electronic equipment.

With the Mark I system, the intensity of the light coming from the comparison and variable stars was originally read off of the meter on the dc amplifier, but a strip chart recorder was soon added to the system for the recording task. After each reading, which lasted about 30 seconds and is called a "deflection," the strip chart would be annotated with information about which star was observed, the color of the filter used, and the time at which the observation was made. A night's observations results in several feet of strip chart paper covered with squiggley lines and annotations. This mass of data then had to be reduced and analyzed.

DATA REDUCTION

Reading the strip chart and reducing and analyzing the data in photoelectric photometry is a tedious process. The first reductions on the Mark I system were made using a TI-30 hand calculator to see what effort was involved. As the effort was sizable, a program to do the reductions was quickly written for the TRS-80.

In many cases where a microcomputer-based control system would be appropriate, there is a data reduction or data analysis task involved. This is often the ideal place to first introduce a microcomputer into the process. One gets familiar with both the use of the microcomputer and the data and its reduction and analysis, paving the way for other uses.

This program is for observations taken in the sequence comp star, comp background, variable star, variable background, comp star . . . ending always with comp star and comp background. The data is keyed into data statements. This was found to be much more efficient than using INPUT statements, as an input error is often not discovered until the run is made (when it becomes obvious), and one does not want to have to go back and key in all the data again. This is an important consideration in all applications.

The inputs to the program include the date of the observations, stars observed, filter color, diaphragm size, and the individual observations (deflections and time). The individual observations are arrayed in sequences, with each sequence except the last one containing deflections and times on four observations (comp star, comp background, variable star, variable background). The last sequence contains just the comp star and its background.

The program then calculates the raw (uncorrected) differential magnitudes, and also converts Universal Time to geocentric Julian Date (the fraction of a day). A listing of this program is given as Example 10-1 and a sample output is given as Example 10-2.

DATA ANALYSIS

The output of the foregoing data reduction program gives results that have not been corrected for the secondary effects of difference in atmospheric extinction between the comparison and variable stars, are still in the local (instrumental) color system instead of the standard UBV system, and the times (fractional dates) are earth-

Example 10-1. Photometry Reduction Program

```
25000 REM PEPANL1/BAS 19 APRIL 1980 FAIRBORN OBSERVATORY
25001 CLS:PRINT"PEP DATA ANALYSIS PROGRAM"
25002 PRINT"25030 DATA JUL DATE,MONTH(#), DAY1,DAY2,FILTER"
25003 PRINT"25045 DATA STAR NAMES(COMP/VAR),# SEQUENCES"
25004 PRINT"25201 DATA CS/UT, CS/CNT,CB/UT,CB/CNT,VS/UT,
      VS/CNT,VB/UT,VB/CNT
25005 PRINT"ENTER DATA THEN RUN 25010"
25009 STOP
25010 REM RUN STARTS HERE AFTER DATA ENTERED
25012 REM DATA JULIAN DATE, MONTH, DAY1, DAY2, FILTER
25020 READ DJ,MN,D1,D2,F$
25030 DATA 4348,4,18,19,"V"
25035 REM DATA STAR NAMES (COMP/VAR), NUMBER OF SEQUENCES
25040 READ SN$,NS
25045 DATA "27/28",5
25050 LPRINT"FAIRBORN OBSERVATORY PEP RESULTS"
25060 LPRINT"YEAR 1980"MONTH ";MN;" DAY1 ";D1;" DAY2 ";D2"
25065 LPRINT"JULIAN DATE ";DJ;" (PLUS FRACTION BELOW)"
25066 LPRINT"FILTER ";F$;" DIAPHRAGM 271 ARCSEC"
25068 REM ANALYSIS ROUTINE
25069 LPRINT SN$
25070 LPRINT" UT JD F MAG D N"
25100 DIM PD(N5+1,2,2,2)
25115 DIM M(N5)
25120 FOR I=1 TO N5:FOR J=1 TO 2:FOR K=1 TO 2:FOR L=1 TO 2
25130 READ PD(I,J,K,L)
25140 NEXT L:NEXT K:NEXT J:NEXT I
25150 I=N5+1:J=1
25160 FOR K=1 TO 2;FOR L=1 TO 2
25170 READ PD(I,J,K,L)
25180 NEXT L:NEXT K
25200 REM PEP DATA CS/UT, CS/CNT, CB/UT, CB/CNT, VS/
      UT,VS/CNT,VB/UT, VB/CNT
25201 DATA 145,15417,147,11217,151,19711,153,12648
25202 DATA 156,18309,159,13553,203,21905,206,13627
25203 DATA 209,18619,212,13476,226,20613,230,12216
25204 DATA 234,17483,237,12674,241,20252,243,12657
25205 DATA 247,17704,252,12673,257,20669,300,12750
25206 DATA 304,17800,307,12922
25295 FOR I=1 TO NS
25300 NM=PD(I,2,1,2)-PD(I,2,2,2)
25305 DN=PD(I,1,1,2)-PD(I,1,2,2)+PD (I+1,1,2)-PD(I+1,1,2,2)/2
25310 A=NM/DN
25320 B$=" "
25330 DM=-2.5*LOG(A)/2.3025851
25335 DM=FIX(DM*1000)/1000
25340 UT=PD(I,2,1,1):HR=FIX (UT/100):MN=UT-100*HR
```

```
25350 HD=(HR+MN/60)/24
25355 HD=FIX(HD*1000)/1000
25360 IF HD(0.5 THEN 25370 ELSE 25400
25370 JD=0 5+HD
25380 GOTO 25410
25400 JD=-0.5+HD
25410 LPRINT UT;JD;B$4;B$;F$;B$;DM;
25415 M(I)=DM
25420 NEXT I
25430 FOR I=1 TO NS
25440 SUM=SUM + M(I):NEXT I
25450 AVE=SUM/NS:LPRINT"AVERAGE";AVG
25460 FOR I=1 TO NS
25470 SS=SS+(M(I)-AVG)↑2:NEXT I
25475 SS=SS/(NS=1)
25480 SIG=SQR(SS):SIG=FIX (SIG*1000)/1000:LPRINT"STD DEV";SIG
```

Example 10-2. Reduction Program Results

```
                    FAIRBORN OBSERVATORY PEP RESULTS
                   YEAR 1980 MONTH 4 DAY1 18 DAY2 19
                  JULIAN DATE 4348 (PLUS FRACTION BELOW)
                     FILTER V DIAPHRAGM 271 ARCSEC
                               27/28 LMI

UT   JD   F    MAG  D N
151  577  V -  494  1 0
203  585  V -  545  1 0
226  601  V -  568  1 0
241  611  V -  471  1 0
257  622  V -  509  1 0
                                            AVERAGE - .5174
                                              STD DEV .039
```

centered times as opposed to sun-centered times. As light takes some 16 minutes to cross the earth's orbit, we would not want times to be affected by where the earth happened to be in its orbit at the time of the observation. Correcting for differential extinction, transformation of results from the instrumental to standard system, and changing from geocentric to heliocentric time (Julian date) are all tedious calculations best performed by a computer.

An analysis program was developed by Dr. D. S. Hall for use on the computer at Dyer Observatory. The program was written in FORTRAN and a number of new features and refinements were built into it over the last several years. While this program has proven to be quite adequate for the reduction of data received from over a dozen observers in Dr. Hall's cooperative program, a number of small college and amateur conservatories have microcomputers and would like to be able to reduce their own data on their individual observational programs. To meet this need, the original FORTRAN program in use at Dyer Observatory was simplified and translated into BASIC. For your application you may find, as I did, that tried and proven software for the analysis tasks may already be in existence. Adapting existing software to your own situation may be easier than developing it from scratch.

Once these simplifications of the original FORTRAN program were made, the original variables were renamed. This was required because most versions of BASIC recognize only the first two characters of a variable name. The variables and their meanings are given in Table 10-1. The analysis program, itself, is given as Example 10-3.

After loading the program, key-in RUN and the computer will immediately print out the information (different for each observatory, of course) shown in Example 10-4.

It will then ask (on the video monitor) for detailed astronomical data on the stars being observed and will echo these inputs back as shown in Example 10-5.

The program then asks JULIAN DATE, XXXX?, whereupon the last four digits of the JD of the first night of observations on the stars in question is entered. The program then computes the solar correction (time difference if observed from the sun) for this star pair on this night and prints it out along with a header for the reduced data. The fractional JD (geocentric) and raw instrumental delta magnitude (local V) of the first observation are then asked for. After these are keyed in, there is about a 10-second pause as the computer works things out in double precision and prints the results. It then asks, ANOTHER OBSERVATION THIS JD THIS START 1=YES, 0=NO. This process is repeated until all the observations on all the nights on all the stars desired are reduced. A sample of two observations are shown in Example 10-6.

The column heading and outputs represent the following. RAW JD(GEO) is the fractional Julian Date corresponding to geocentric Universal Time, while DELTA MAG is the differential magnitude in the instrumental photometric system (V). These first two columns are merely echoes of the input and are printed out as an aid to later verification of proper keying in of the data (an important point to consider in the development of any system). The HELIOC

Table 10-1. Program Variables

Variable	Meaning
V1	Variable RA (hours)
V2	Variable RA (minutes)
V3	Variable Dec (degrees)
V4	Variable Dec (minutes)
VR	Variable RA (radians)
VD	Variable Dec (radians)
C1	Comparison RA (hours)
C2	Comparison RA (minutes)
C3	Comparison Dec (degrees)
C4	Comparison Dec (minutes)
CR	Comparison RA (radians)
CD	Comparison Dec (radians)
TH	Longitude of Sun (radians)
JD	Integer Portion of JD (e.g., 4274)
R	Intermediate Computation
S1	Intermediate Computation
S2	Intermediate Computation
SC	Solar Correction (to Geo. JD)
JH	Heliocentric JD (e.g., 4274.5943)
JF	Fraction of JD (Geo.) (e.g., 0.5943)
CY	Cycle (of eclipsing binary)
EP	Epoch (of eclipsing binary, e.g., 2443654.38572)
PD	Period (of eclipsing binary, in days)
PH	Phase (of eclipsing binary)
ST	Local Sidereal Time
SA	Local Sidereal Angle
VH	Variable Hour Angle (radians)
CH	Comparison Hour Angle (radians)
SL	Sine of Latitude
CL	Cosine of Latitude
VX	Variable Airmass
CX	Comparison Airmass
DX	Differential Airmass
MX	Mean Airmass
DS	Delta Magnitude (standard V)
DR	Delta Magnitude (instrumental V)
DC	Delta Color Index (Delta B-V)
V	Variable Visual Magnitude (Std. V)
CV	Comparison Visual Magnitude (Std. V)
ON$	Observatory Name
LN	Observatory Longitude (degrees)
LA	Observatory Latitude (degrees)
SN$	Star Names
EL	Epsilon (determine by calibration)
KV	k'v Principal Extinction Coefficient in V

Example 10-3. Photometry Analysis Program

PROGRAM LISTING

```
10  DEFDBL A-Z
100 REM PEPAN2/BAS PROGRAM
110 PRINT"PEPAN2 PROGRAM, F.O. 26 MAY 80"
120 REM I/0 of OBSERVATORY DATA
130 NO$="FAIRBORN OBSERVATORY"
140 LN=83.97261
150 LA=39.80075
155 LPRINT NO$
160 LPRINT"LONGITUDE=";
161 LPRINT USING"###.#####";LN
162 LPRINT"LATITUDE=";
163 LPRINT USING"###.#####";LA
164 EL=027: KV=.25
166 LPRINT"EPSILON=";: LPRINT USING "#.###"; EL
167 LPRINT"EXTINCTION", :LPRINT USING"#.###";KV
170 I=1
200 REM INPUT DATA ON STARS
210 PRINT"INPUT DATA ON STAR PAIR NO.";I
215 INPUT"STAR NAMES";SN$
220 INPUT"VARB RA, HR, MN";V1, V2
230 INPUT"VARB DEC, DEG, MN"; V3, V4
240 INPUT"COMP RA, HR, MN"; C1, C2
250 INPUT"COMP DEC, DEG, MN"; C3. C4
260 INPUT"EPOCH";EP: INPUT"PERIOD";PD
270 INPUT"DELTA (B-V)";DC: INPUT"COMP V MAG"; CV
275 LPRINT SN$
280 LPRINT"VARB RA=";V1," HR"; V2;" MN","VARB DEC=";V3; "DEG";
    V4," MN", "COMP RA="; C1; "HR "; C2; "MN", "COMP DEC="; C3,"
    DEG ";C4," MN"
290 LPRINT"EPOCH="; EP, "PERIOD="; PD, "DELTA(B-V)="; DC,
    "COMP V="; CV
295 J=1
300 REM I/0 JD AND COMPUTER SOLAR CORRECTION
310 PRINT"INPUT DATA ON"; SN$; "FOR NIGHT NUMBER";J
320 INPUT"JULIAN DATE, XXXX";JD
322 VR=(V1+V2/60)*15/57.296
324 VD=(V3+V4/60)/57.296
326 CR=(C1+C2/60)*15/57.296
328 CD=(C3+C4/60)/57.296
330 TH=(4.8686153+.0172497*(JD-3874))
340 R=(1-.016719*.016719)/(1+.016719*COS(TH-4.956105))
350 S1=R*COS(TH)*COS(VR)*COS(VD)
360 S2=R*SIN(TH)*SIN(.409273)*SIN(VD)
365 S3=COS(.409273)*COS(VD)*SIN(VR)
370 SC=-.005775*(S1+ S2+S3)
```

```
380 LPRINT"JULIAN DATE="; JD, "SOLAR CORRECTION=";
381 LPRINT USING"#.#####";SC
390 K=1
395 LPRINT
397 LPRINT"RAW JD(GEO) DELTA MAG HELIOC JD STD V MAG ORB
    PHASE MEAN X STD MAG"
400 REM INPUT DATA ON OBSERVATION, ANALYZE AND OUTPUT
405 PRINT"INPUT DATA ON"; SN$; " JD=";JD;"OBSERVATION NO.";K
410 INPUT"FRACTIONAL JD, DELTA MAG"; JR, DR
420 JH=JD+JF+SC
430 CY=(JH+2440000-EP)/PD
440 PH=CY-FIX(CY)
450 ST=24*((JH-3407.7489)/ 365.2422-LN/360+1.002738* (JF-.5))
460 SA=ST*15/57.296
470 VH=SA-VR : CH=SA-CR
480 SL=SIN(LA/57.296) : CL=COS(LA/57.296)
490 VX=1/(SL*SIN(VD)+CL*COS(VD)*COS(VH))
500 CX=1/(SL*SIN(CD)+CL*COS (CD)*COS(CH))
510 DX=VX-CX : MX=(VX+CX)/2
520 DS=DR+EL*DC-KV*DX : V=DS+CV
530 LPRINT USING"######.####"; JF, DR, JH, V, PH, MX, DS
600 REM CONTROL SECTION
610 PRINT"ANOTHER OBSERVATION THIS JD THIS STAR?"
620 INPUT"1=YES, 0=NO"; X
630 IF X=1 THEN 640 ELSE 650
640 K=K+1 : GOTO 400
650 PRINT"ANOTHER JD THIS STAR?"
660 INPUT" 1=YES, 0=NO"; X
670 IF X=1 THEN 680 ELSE 690
680 J=J+1 : GOTO 300
690 PRINT"ANOTHER START?"
700 INPUT"1=YES, 0=NO"; X
710 IF X=1 THEN 720 ELSE 730
720 I=I+1 : GOTO 200
730 STOP
```

Example 10-4. Analysis Program Results

```
FAIRBORN OBSERVATORY
LONGITUDE  = 83.97261
LATITUDE   = 39.80075
EPSILON    =  0.027
EXTINCTION =  0.250
```

JD is the Heliocentric Julian Date. STD V MAG is the Standard V Magnitude of the variable star corrected for extrication and transformation. ORB PHASE is the orbital phase based on the provided

Example 10-5. More Analysis Results

```
44 I B00       47 K B00
VARB RA= 15 HR 2 MN     VARB DEC= 47 DEG 50 MN
COMP RA= 15 HR 4 MN      COMP DEC= 48 DEG 20 MN
EPOCH= 2439370 . 4222    PERIOD= .267816      DELAT-(B-V)=.7
COMP V= 5.5
```

Example 10-6. At Last the Answers

```
JULIAN DATE=4374
   RAW      DELTA     HELIOC     STD     ORB       MEAN X
 JD(GEO)     MAG        JD      V MAG   PHASE     ST D MAG
  0.5950  -0.8750  4374.5978 4.6449  0.1256     1.1696 -0.8551
  0.5972  -0.8850  4374.6000 4.6349  0.1339     1.1631 -0.8651
```

ephemeris (an equation describing the times of minimum). MEAN X is the mean airmass with 1.0 being the smallest possible value when the stars are observed at the zenith. ST D MAG is the Standard V Differential Magnitude.

Once all the corrections and transformations have been made, the data is often ready for publication in astronomical journals in that form. Additional analysis is sometimes performed on large amounts of data from individual observatories, or a group of observatories working together in a cooperative program. Such analysis includes least squares determination of times of minima, fitting of photometric observations to astrophysical models, etc. The description of such specialized analysis is beyond the scope and intent of this book. Suffice it to say that it often takes hours of CPU time on very large machines, and is definitely a batch operation not related to real-time requirements.

SUMMARY

There are two important lessons to be gleaned from this chapter. First and foremost, a thorough familiarity with the process to which one intends to apply real-time microcomputer techniques to is an absolute prerequisite to success. If, as in my case, one has to build and run a purely manual system to achieve this objective, then by all means do so.

Second, if the opportunity exists to apply a microcomputer to routine data reduction and analysis tasks, this is the best time to do so. Not only will familiarity be gained with the microcomputer, but one can become familiar with the actual data and its reduction and analysis. This can also allow other persons to become familiar with the microcomputer and accustomize themselves to the notion that it is not just a toy, but can save time and effort. If existing software can be adapted to your problem, take advantage of it.

CHAPTER 11

The Mark II Photometric System

The Mark I Photometric System just described was designed in February of 1979, and actual construction of the Fairborn Observatory began in April of 1979. The Mark I system became operational in August 1979 and remained in operation until December of that year, when it was shut down due to the very cold weather (after 10 years in Ohio, I still have a Californian's view of the winters here).

The five months of experience gained in operating a conventional, manual photometric system raised my respect for the persistence of the pioneers of photoelectric photometry who were able to keep this up year after year. It also provided the incentive necessary to develop something better than manual annotations on a strip chart during the observing runs and the long hours of drudgery after the runs that it took to read the strip charts and enter mountains of data into the computer. Here, if there ever was one, was a clear-cut base for computerized logging of the data.

RESEARCH AND PLANNING: THE CRUCIAL STEPS

At the onset, I realized that making a real-time connection between a microcomputer and instrumentation on a telescope was not a task`to be taken lightly. My first step, and one I cannot recommend too highly, was to research similar applications of computers in the past. I did this not only by reviewing the literature, but by writing many letters and making a number of first-hand visits. What

I found was startling. Some of the applications that appeared so rosy in the literature turned out, on closer examination, to be outright failures, and expensive ones at that. The successes that had been achieved had generally been quite expensive and very time consuming. I spent some time in trying to determine the various causes for success or failure and tried to emulate the former and avoid the latter. Based on this research, I planned the Mark II Photometric System.

The first crucial planning question was how much of the operation did I want to automate? On the ambitious side, I could automate the operation of the entire telescope and photometer in one fell swoop. On the conservative side, I could automate only the collection of the data and its initial reduction. As the previous projects that had failed had generally been ones that had tried to automate everything at once, and since the real drudgery I wished to avoid was annotating the strip charts, reading the strip charts, and entering the raw data into the computer (and not the operation of the telescope and photometer which I rather enjoyed), I opted for the conservative approach. This saved me from certain disaster. In planning out your approach, consider all the various options, then get control of any tendency towards optimistic ambition and opt for a conservative approach that nets the most immediate gain for the smallest investment. Goof here and you've blown it!

The next crucial decision was the location of the computer. I had initially assumed that the computer would, of course, be located right beside me in the observatory so that I could easily enter things on its keyboard, watch the video, and check any printouts. However, my research of previous experience indicated that computers don't work well at all in the inhospitable environment of observatories (what with the roof open to the sky). Furthermore, having just one computer and wanting to use it for analysis, word processing, and other tasks in my study, I began to get this horrifying picture of having to disconnect my precious computer and haul it out to the observatory before each observing run. How long would the connectors stand up to such abuse, and what if I slipped on the sidewalk or stumbled on the steps in the dark? No, the computer would have to stay in the warm study while I went out in the cold. While your workplace environment may not be as inhospitable as mine, it may not be an ideal environment for a microcomputer either. Greasy-fingered workmen can play havoc with keyboards, metal shavings in the expansion interface can lead to shorts, chemicals spilled on the video monitor ruin its looks, not to mention what a good dose of dust will do to the disk drive!

Having decided to leave the computer protected in the study, I needed to decide how I was going to communicate with it. Initially, it had been my dream to have all sorts of flashing LEDs, meters, switches, and knobs. How impressed visitors to the observatory would be! Research of previous experience suggested that there were very few situations that couldn't be taken care of neatly by just a video monitor and keypad. Not only were more complicated approaches unnecessary, but they tended to be expensive, inflexible, and less reliable. The old "KISS" (Keep It Simple, Stupid) dictum certainly applies here. Keep this in mind on your system.

There were three options in getting data to and from the study and observatory. One was to extend the high speed computer bus out to the observatory. The second was to use serial communication over a single data line. This was given serious consideration as it greatly simplifies connections between the computer and the remote location(s). If there were many remote locations, and/or they were at a considerable distance from the computer, then this would be a logical approach, and may be the approach that best fits your situation. In my case with less than 100 feet to go and a single location, I felt that parallel communication over a number of lines would be the simplest and cheapest approach.

For the computer interface itself, I had to decide between a large number of TTL chips or a few flexible, programmable interface chips specifically tailored to the task.

The question of programming language and the use of real-time interrupts were given very serious thought. The interpretive BASIC stored in ROM in the TRS-80 is very slow. Most real-time control systems use real-time interrupts, implying at least some machine language requirement. A number of astronomical observatories had found the FORTH language to be very attractive for their real-time control tasks. However, I already knew BASIC and liked it, wasn't very good at Editor/Assembler, and wasn't at all forthcoming on FORTH. Furthermore, interrupts greatly complicate programming logic, making software development and debugging more complicated. With these thoughts in mind, I decided to see if I could do it all in BASIC without any interrupts or machine language subroutines. I figured, if worse came to worse, that I could always add a machine language subroutine where high speed was of the essence. This never proved to be necessary.

In your application you might, as I did, get away with using BASIC exclusively, and avoiding interrupts. On the other hand, your system may need almost instantaneous response in critical situations (such as the crane about to run off of the end of the track and fall in the molten steel). Don't despair, just add in a single general-purpose alarm interrupt (to let the computer know it should stop what it is doing and check all the alarm flags) and a very small machine language subroutine to take care of this. Don't, whatever you do, leap to the wild conclusion that since part of it has to be in machine language, the whole thing might as well be. How horribly primitive this would be, and what a waste of years spent in developing high level languages and cheap memories! Besides, this is a book on real-time control with micro*computers,* not a book on real-time control with micro*processors.* We take the high (level) road and they take the low road.

Having discussed the philosophical, but vitally important, aspects of planning the Mark II Photometric System, it is time to consider the goodies.

EQUIPMENT

A block diagram of the system is shown in Fig. 11-1. The equipment contained in the observatory is shown on the right side of the diagram, while that located in the office is shown on the left. The

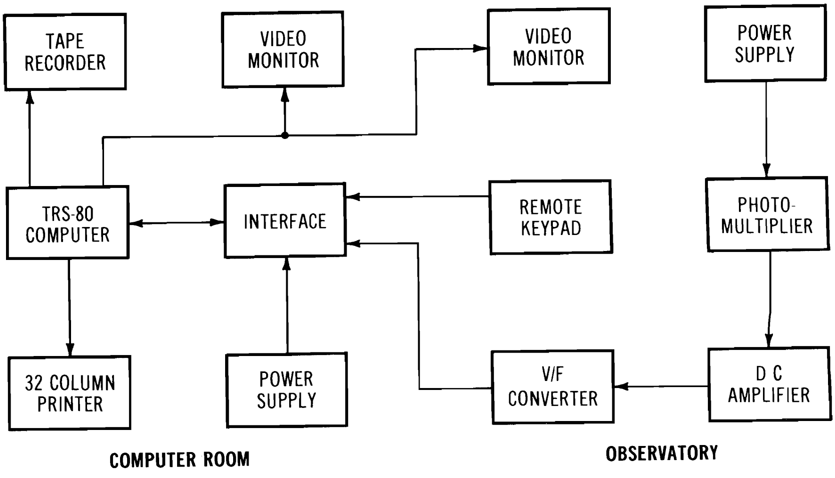


Fig. 11-1. Mark II block diagram.

equipment in the observatory is shown in Fig. 11-2. The photometer head is permanently mounted on the 20-cm f/15 Cassegrain telescope. It contains a previewer with wide angle eyepiece, diaphragms of various sizes mounted on a wheel, a microscope for centering the star in the diaphragm, filters on a wheel for three color photometry, a Fabrey lens, and the 1P21 photomultiplier tube. If this sounds and looks familiar, it is because it is the Mark I system virtually intact. Think gradual evolution.

The equipment in the office is shown in Fig. 11-3. The expansion interface (not required to operate the Mark II system) is under the video monitor. The disk drive and modem (also not required) are shown at the top left, while the power supply (5 V dc) and the general-purpose, real-time control interface are in the middle and the printer on the bottom. The system was developed and operated without the expansion interface, disk drive, and modem, and these were added just as an operating convenience and with an eye towards meeting anticipated future requirements.

The interface, which is an earlier and more primitive version of the one described in Section I of this book, contains only six chips. Three chips are 7404 hex inverters used for the clock oscillator and for generation of chip select signals. Two chips are Intel 8253 that contain three independent 16-bit counters that are directly programmable and readable by the computer. The final chip is an 8255 Programmable Peripheral Interface (PPI).

A 1 MHz oscillator is the basic local time reference. Two of the counters in one of the 8253 PIT chips are used to divide it down to 1 KHz and 1 Hz references, while the third one is used as a delay adjustable via software in one millisecond steps to achieve synchronization with the National Bureau of Standards radio station WWV. Another 8253 PIT chip counts the seconds, counts the pulses arriving from the v/f converter, and precisely times the integration (software-setable). The 8255 PPI chip reads the keypad encoder, issues discretes, or checks status flags.

OPERATION

The operation of the system will be described by going through an actual sequence. The computer is powered up simply by turning it on. The operating program is loaded off the disk and the Julian Date (JD), month, and double-date are entered. At this point, control is automaticaly transferred to the observatory, the video monitor

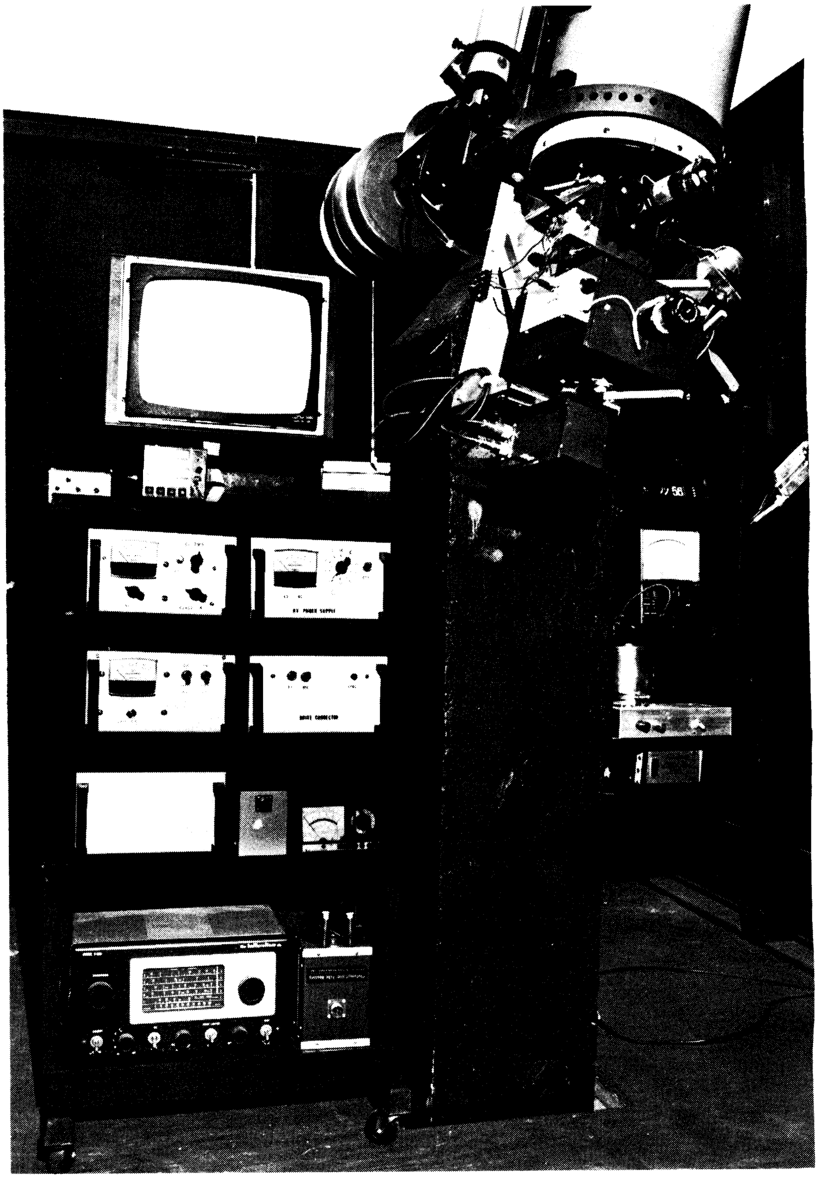


Fig. 11-2. Mark II observatory equipment.

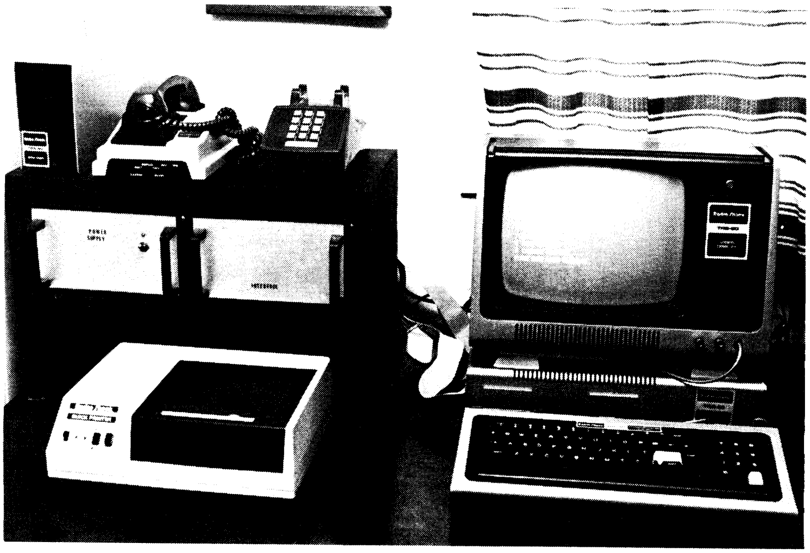


Fig. 11-3. Mark II computer equipment.

and lights are turned off in the office, and the door is latched to preclude unauthorized access by junior members of the staff.

On arriving in the observatory the "Main Menu" is displayed on the remote monitor. The time cube is activated and set on a post outside the observatory to avoid the considerable interference from the computer that travels along the control lines. Option 1, "Synchronization of the Clock," is selected from the menu. For "slow" variables close synchronization of the local clock with WWV is not required and any three-digit delay may be keyed in. For somewhat faster variables the WWV "tick" and local clock "tick" may be synchronized by ear to within one-tenth of a second. More precise alignment, such as required for occultations, requires an oscilloscope to observe the WWV tick crossover.

Once the WWV and local clock seconds ticks are aligned satisfactorily for the type of observation, the clock is set by asking for menu Option 2. The computer asks for the hours and minutes of the next "at the tone." These are entered on the keypad and, when the tone is heard, full synchronization will be maintained if any key is pressed within a second. The time is then displayed in the upper right hand corner of the screen. It may be checked at any later time by asking for menu Option 7 which displays the time.

With the clock now set, menu Option 3 is selected, the list of the current program stars is displayed, and a pair (variable and comparison) is chosen. The list of program stars is updated about once a month, adding new ones in the East and dropping old ones in the West.

A sequence of observations on the selected star pair (differential photometry) is then initiated by selecting menu Option 4. The computer asks NUMBER OF COLORS? While the computer can handle any number of colors, the remote video monitor can display only up to three conveniently. Assuming that one color (V) is desired, a "1" is entered on the hand-held keypad. The computer then responds with an observer prompt which includes: identification of variable and comparison stars, whether the variable or comparison is to be observed, whether the star or the background is to be observed, and the color of the filter.

The observer (without computer assistance) must then find and center the indicated star (or sky offset) and select the indicated filter. At the beginning of any sequence the observer must also adjust the gains on the dc amplifier so that the brightest star through the "brightest" filter is near full scale. Once everything is set, pushing any key starts the precisely timed integration. As the pulses from the v/f converter are being counted, the counter is read "on the fly" and this reading is used to generate on the video monitor a display of light intensity versus time, similar in many respects to the display on a strip chart recorder. Such abnormalities during the integration as the star drifting out of the diaphragm, a star drifting into the diaphragm (during background), an unnoticed cloud, an accidentally kicked cable, etc., can be seen easily on the display. At the end of the integration the final count is displayed along with the time. As with all other displays, pushing any key advances the observer to the next step or returns the observer to the main menu as appropriate.

In this case, the next step is a display of all the light intensity counts and observation times in an array. The Comparison Star (CS) counts are all in a single column as are the Comparison Background (CB), Variable Star (VS), and Variable Background (VB) counts. This allows one to tell at a glance how the latest count compares with all previous counts in the sequence. It is immediately apparent if one is observing the wrong star, forgot to flip down the mirror during the background, etc. One notes mentally whether the reading is "out of line" or not.

The computer then asks for a choice. One can select: (1) TERMINATE SEQUENCE, (2) REPEAT OBSERVATION, or (3) DATA OK CONTINUE. One can terminate at any time. If the just completed observation is questionable, one may repeat it. If everything is proper, pressing "3" brings up the prompt for the next observation. The combination of the video display, the count array, and the ability to repeat or terminate as well as continue has proven to be very useful. This observer, at least, would have a difficult time going through an entire observing sequence late at night without making an error of some sort, so error detection and recovery procedures are an important prerequisite in this computerized system. In designing your own control system, it is *imperative* that you make similar provisions. This is of critical importance.

By means of prompts, displays, and commands, the sequence on a pair of stars is completed. Terminating a sequence automatically returns one to the main menu. To retain the raw counts, times, and identifying information for the record, menu Option 8 is asked for and the data is automatically printed out as shown in Example 11-1. These are actual data on a "not so good" night. The columns are Comparison Star Universal Time (CS/UT), Comparison Star Count (CS/CT), etc.

By asking for Option 9, the raw differential magnitudes will be calculated and both printed and displayed. The printout is as shown in Example 11-2. Note that the mean and the standard deviation are given for the sequence. The standard deviation has proven useful as a gauge of the photometric quality of the night, on occasion resulting in observatory shutdown when the night was hopeless. Prior to this prompt feedback I used to bravely charge on for hours, only to find out days later when I finally reduced the data that I had been wasting my time. You may have a similar situation where fast computer number crunching is valuable.

At this point another pair of stars can be selected or the observ-

Example 11-1. Data Record

```
COMP/VARBL                                   47B00/44B00
CS/UT   CS/CT   CB/UT   CB/CT   VS/UT   VS/CT   VB/UT   UB/CT
31702   11609   31754   3234    31913   20474   32003   3256
32112   11205   32207   3296    32312   19348   32404   3292
32505   11153   32558   3321    32916   19781   33024   3269
33256   11297   33346   3282    33449   19741   35548   3284
33701   11451   33754   3253      0       0       0       0
```

Example 11-2. Observational Results

```
FAIRBORN OBSERVATORY PEP ANALYSIS—1980
JUL DATE = 4394 MONTH = 6 DAY1 = 3 DAY2 = 4
COMP/VARBL = 47B00/44B00 FILTER = VISUAL

           UT                  JD                  MAG
         31913              .63834                −.813
         32312              .64111                −.774
         32916              .64532                −.797
         33449              .64917                −.768

AVERAGE = 0.788
STD DEVIATION = ± 0.021
```

atory can be shut down. Prior to shutdown, the clock is usually checked again against WWV as a precaution. It is not unusual to do the final analysis of the data on the same night as the observations are made. Finally, the computer printouts are added to the observatory notebook and various remarks (observing conditions, equipment notes, etc.) are written in by hand.

It has been found that while the Mark II system does not significantly speed up the actual process of making the observations, by totally eliminating backlogs of unread strip charts and unreduced data, it does increase considerably the amount of time spent observing. It seems that a pile of unread strip charts is a real deterrent to observing, at least for this observer.

CHAPTER 12

The Mark III Photometric System

As you have seen, the Mark I system was a totally conventional, manual system except that a microcomputer was used as a "smart" calculator to reduce the data, make the various transformations and corrections to the data, and in some cases to provide final analysis such as a least-squares determination of the time of primary minimum, the exact time when the least light is received from an eclipsing binary star. While the use of the microcomputer saved considerable time in data reduction and analysis as compared to hand calculations, a microcomputer would not have been bought for this purpose alone as the author has access to a number of computers where batch jobs could be run, and the use of the microcomputer at this point was really just that of a "batch processor."

The experience gained in operating the Mark I system was an absolutely essential prerequisite for the design of the Mark II system, and the experience gained in operating the TRS-80 as a straight "calculator" proved to be valuable in applying it as a data logger/prompter on the Mark II system. Probably of equal importance, during the time the Mark I system was in operation, research on photometric systems in general and automated photoelectric systems in particular continued. A number of successful, and some not so successful, systems were carefully studied to determine the reasons for their success, or lack thereof. This respite also allowed the author to take a crash self-study course in digital electronics (vacuum tubes were still in style when I went to school). The Blacksburg series were the texts used in this effort.

The performance of the Mark II system was essentially flawless from the start and it was in operation almost every clear night for over a year. A couple of operational inconveniences were corrected via software changes in the first week of operation, and the keypad encoder chip was moved out to the observatory to avoid miskeying after torrential downpours soaked the connecting cables. No other changes were required or desired. The elimination of the reading of strip charts, writing down and keying in of data, the flexible repeat and terminate options, the instantaneous data arrays, and the almost instantaneous reduction of the data all contributed to an efficient, error-free operation. The result was a big increase in productivity as time was spent in making the observations, not in recording and reducing them.

PANDORA'S BOX AND THE LAW OF DIMINISHING RETURNS

The Mark II system was so efficient that I seriously questioned whether further automation would really warrant the effort involved. However, as I gathered data night after night on various eclipsing binaries, flipping the little mirrors, changing the filters, moving the telescope from star to background, and moving the telescope between the stars and centering each star, my not fully occupied mind was bound to consider various possibilities.

When one has a microcomputer connected up to a telescope, everyone immediately assumes that the computer is being used to point the telescope at the correct star. This possibility was considered at some length, but eventually discarded (at least for now) for a number of reasons. First, in photoelectric photometry of variable stars, one tends to look at the same stars night after night, occasionally dropping one star off the program as it goes into the trees in the west, and adding a new star to the program as it gets high enough in the east. The result of recording data on the same stars night after night is that one gets very adept at locating the star visually. Once a star has been on my program for just a few nights, with the telescope pointing in some random direction, I can locate the star, move the telescope to it, and center it in the finder within 10 seconds when showing off (within 20 seconds normally). Thus, computerized pointing of the telescope simply would not save much time or effort. Second, it is an expensive and difficult task to point a telescope with a computer, particularly if one wants a pointing accuracy in the arcseconds range.

In reviewing previous efforts in computerized astronomical photometry, it was interesting to note that several of the efforts had been either less than successful or completely unsuccessful for essentially the same reasons. The objective in these cases was a fully automated photometric system (as opposed to an efficient system whether fully automated or not). As full automation necessarily involved automated pointing of the telescope and centering of the stars, these efforts got bogged down in the intricacies of this difficult task. Of course there are a few telescopes used primarily for photometry that do have successful automated pointing systems, but almost all of them are very expensive, and without exception, the software involved is complex and massive. As pointed out in Chapter 8, full automation for automation's sake is a trap for the unwary control system developer, and one that plays on the inherent nature of those enchanted with computers. Keep this in mind!

In improving on the Mark II system there appeared to be two areas where further automation could be made with little risk. One was the changing of the filters between the various color pass bands. As the filters were mounted on a wheel, it seemed likely that the wheel could simply be placed on a stepper motor. The other area for improvement involved looking at the sky background near each star. In the Mark II system, this was done simply by moving the telescope slightly and checking to see that there were no visible stars in the field. Instead of moving the telescope one could move the diaphragm slightly, and as the diaphragms were mounted on a wheel, a stepper could also be used for this task. Thus, the decision was made to add two stepper motors to the system to control the positions of the diaphragm and filter wheels.

It is perhaps instructive to examine in retrospect the extensive impact of this seemingly minor change to the system. To take full advantage of computerized control of the diaphragm and filter wheels, the telescope needs to be able to keep a star accurately centered for a long sequence of star and background measurements in the different colors. Without computerized control of these functions it is customary to recenter the star after each individual measurement. Thus, tracking accuracy requirements are greater when one uses a semi-automated sequence of observations. On small, lightweight telescopes, the movement of steppers and diaphragm and filter wheels tends to degrade, not improve, tracking accuracy. Finally, a photometer head with precision stepper motors is significantly larger and heavier than one without them.

After giving these considerations some thought, it was clear to me that my 8-inch Cassegrain telescope could not meet these requirements, and that a more massive, roomy, and accurately tracking telescope was called for. While in theory the 8-inch telescope could have been modified to meet these requirements, the modifications would have been extensive and it would have meant taking the telescope out of operation for an extended period of time, something I was loath to do. Thus, the decision was made to build a second telescope.

With the lid to Pandora's box now fully open, other decisions followed with great rapidity. With a massive mount in a roomy expansion to the observatory building, it seemed a shame to put just an 8-inch telescope on the mount. A 16-inch seemed much more appropriate. As a mirror of this size is prohibitively expensive, the obvious solution was to grind one myself.

As I wanted to leave the Mark II system totally intact, this meant that a new computer interface would have to be built. Rather than stuffing wires in a breadboard, the more permanent wire wrapping seemed appropriate. Also, as the new system would undoubtedly keep growing, more complete address decoding seemed wise. With a good clock/calendar chip now readily available, I couldn't go without one of course. Also, for this grand telescope, it seemed appropriate to have a really first-rate photometer with an end-on photomultiplier, pulse counting, etc. (i.e., the works). Finally, numerous requests for documentation on the poorly documented Mark II system suggested that more extensive documentation of the Mark III system would be appropriate. Dr. Jon Titus of the Blacksburg Group suggested that a book would be a good format. All these changes required extensive revamping of the software, although being a believer in gradual evolution, the Mark II software was used as a working baseline and gradually modified one piece at a time.

While I am immensely pleased with my 16-inch telescope and Mark III control system, and I can, indeed, gather photometric data with considerably increased efficiency and accuracy, the effort and cost involved was much greater than that involved in the Mark I and Mark II systems combined. This is a clear illustration of the law of "diminishing returns" in control system automation. It is also a good illustration of the many unforeseen ramifications that can be encountered as one tries to automate a previously manual control function. Keep this in mind when you plan out your system. Don't be automation greedy!

OBSERVATORY AND TELESCOPE

The observatory portion of the Mark III system is housed in a 12 by 12 foot structure with a roll-off roof. The roll-off roof was selected in preference to the traditional dome as it was cheaper, less obtrusive (my wife didn't want a big shiny dome in her yard to attract the curious), and one doesn't have to fiddle around with dome positioning all the time. I also like the unobstructed view of the sky even though I usually avoid trying to look up at night.

A two-pier mount was chosen not only for its superior stability, but because the wide spacing of the bearings on the polar axis (one on each pier) allows precise alignment to the north celestial pole to be made with greater ease. The two piers, which are heavily reinforced with steel and connected together underground, contain over 4 cubic yards of concrete.

Between the two piers is the polar axis of the telescope. It is made of 10-inch channel iron ½-inch thick and weighs some 800 pounds, including the shafts, collars, and bearings. The polar axis is turned with a 14-inch precision worm gear that tracks stars with great accuracy.

The telescope, itself, is a truss arrangement made of circles cut from ½-inch plate steel connected together with 1¼-inch steel tubing. A Fiberglas tube inside the truss is used to keep out stray light and dust, but is not part of the structural support *per se*.

The 16-inch diameter primary mirror is a piece of 3–inch–thick quartz, kindly donated by Dr. Ken Kissell, a professional astronomer. It was ground, polished, and figured to an f-ratio of 3.75 under the supervision of Ed Jones on the grinding machine at the Miami Valley Astronomical Society. The secondary mirror in the Classical Cassegrain optical system is a 4¼-inch diameter mirror and was ground, polished, and figured by Arthur Eresman. The entire optical system (including the baffels) was designed by Dr. Frank Melshiemer. The effective f-ratio of the system is f-18, and the effective focal length is 288 inches. The field-of-view of the system is 1 degree, and the linear field diameter is 5 inches. The back focus is 18 inches, giving plenty of room for instrumentation.

An overall view of the observatory is given in Fig. 12-1, while Fig. 12-2 shows the telescope in more detail.

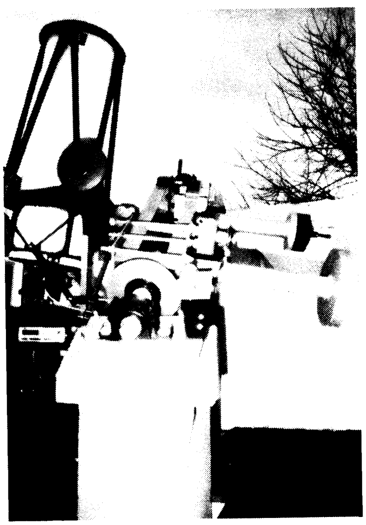


Fig. 12-1. Overview of the observatory (16-inch).

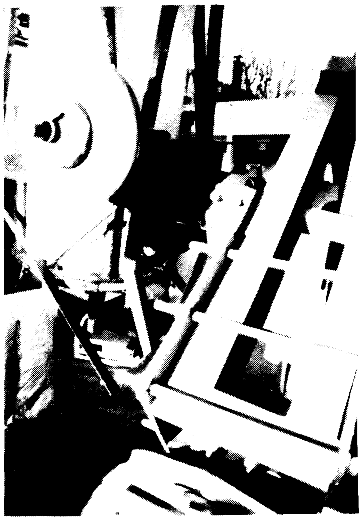


Fig. 12-2. The telescope in more detail.

PHOTOMETER HEAD

The photometer head is perhaps the most critical portion of the photometer system. For the Mark III system, the author helped Robert C. Wolpert and the staff at EMI Gencom, Inc., design a research grade pulse counting stellar photometer that has been named the Starlight-1. A close-up picture of this photometer mounted on the 16-inch Cassegrain telescope is shown in Fig. 12-3.

We wanted to design the Starlight-1 so that the detector would not be the limiting factor in measurement accuracy. This required a photomultiplier tube which had the following characteristics:

1. Greater than 30% quantum efficiency at peak wavelength
2. Extremely low background noise
3. A gain sufficient for single photoelectron detection
4. Small dimensions
5. UBV capability without requiring cooling
6. Interchangeability with mechanically identical PMT's but with different spectral sensitivities
7. Operate at less than 1000 V dc
8. Low in cost

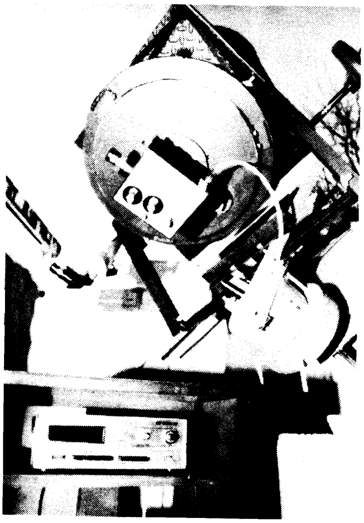


Fig. 12-3. Close-up of the Starlight-1 mounted on the 16-inch telescope.

The obvious choice was the end-window EMI 9924A. It has a RbCs photocathode with superb UBV sensitivity, a dark current of less than 50 picoamps, an excellent plateau curve at less than 200 Hz, it is only 1⅛ inches in diameter, it has many spectral response variants which are mechanically identical, and (last but not least) it is low in cost. This photomultiplier tube, when coupled to a fast, high gain, differential amplifier/discriminator, will provide single photoelectron detection capability. In addition to the EMI 9924 PMT, the detector assembly consists of a 500–1600 volt dc-dc power supply and a 2 by 4 inch circuit board which contains the PMT voltage divider network, a LeCroy MVL 100 amp/disk and a gain control pot.

The photomultiplier tube is housed in a foamed chamber with rfi and magnetic shielding. This foamed chamber has been designed to be easily replaced (at a later date) by a self-contained, thermoelectric, liquid-cooled chamber. The chamber is also designed to allow easy replacement of the photomultiplier tube.

All of the above items comprise about 40 percent of the available volume in the 6.5 by 4.5 by 7.2 inch photometer head. The remainder of the available space is completely filled by a complex optical assembly.

The optical assembly was the most difficult part of the Starlight-1 to design. Twice the design had to be scrapped as it neared completion in order to incorporate some newly suggested feature. The final design incorporates every desirable feature that was suggested by experienced photometrists.

Incoming light will first pass through one of the diaphragms in a six-position, aluminum diaphragm turret. Each position is detented for positive and accurate aperture location. Each aperture position also has an offset detent which will allow sky background measurements to be taken without requiring the entire telescope to be repositioned. There is a full open 0.750-inch aperture which contains an LED (light emitting diode) illuminated reticle. The LED is located in the flip prism mechanism and will light only when the prism is in the "viewing" position. The intensity of the LED is controlled by an external on/off brightness control dial. The same LED also backlights each aperture plate, allowing the observer to easily distinguish between the sky field and the edges of the aperture.

After passing through the aperture, the signal can either be reflected into the viewing microscope by a flip prism, or allowed to pass through one of the filters in a 6-position aluminum filter turret. Each position on the filter turret is also accurately detented for positive location. The UBV standard ultraviolet, blue, and visible filters are provided with the basic Starlight-1 system. These are especially corrected for the EMI 9924A, but do not differ greatly from the filters typically used with side-window PMTs. An opaque filter position serves as a shutter which protects the PMT from ambient light while the photometer is not in use. The two unused filter positions can be utilized for red, infrared, or other user selected filters.

After light passes through the desired filter, it will pass through a Fabrey lens and onto the PMT window. The entire optical system is designed so that there is no light loss from f/5.0 or higher focal ratio telescope.

All metal surfaces in the photometer head have either been anodized or painted black, and great care has been taken to prevent light leaks. The four control dials are conveniently located on a single outside panel. The diaphragm and filter turrets have illuminated position indicators on their dials which allow positive position identification without the need to turn on observatory lights. The third dial controls the LED illumination, and the fourth will flip the prism into the desired position. Top and side views of the photometer are shown in Figs. 12-4 and 12-5.

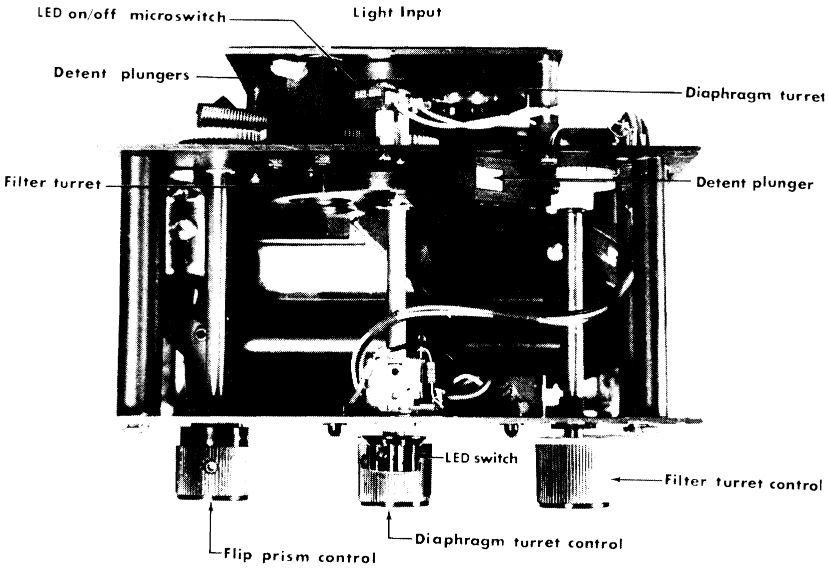


Fig. 12-4. Top view of the photometer head. (Courtesy EMI Gencom.)

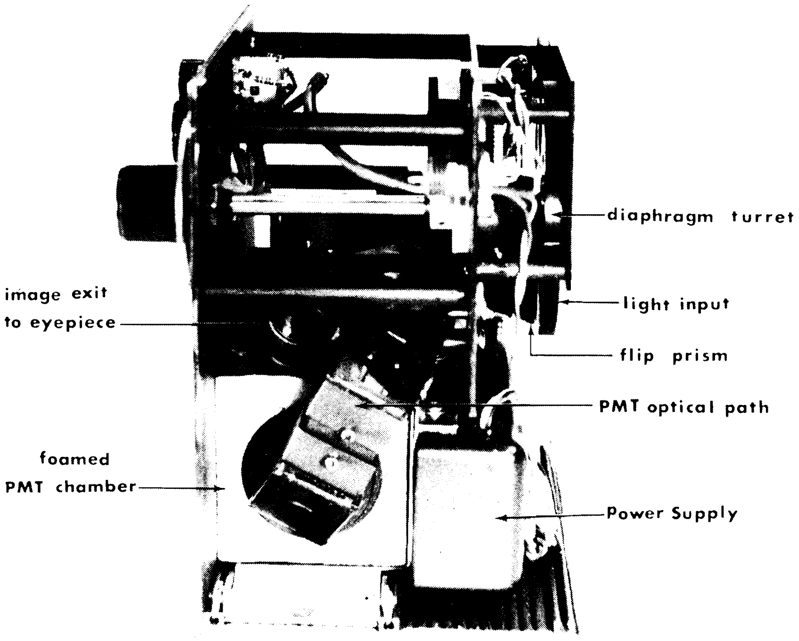


Fig. 12-5. Side view of the photometer head. (Courtesy EMI Gencom.)

PHOTON COUNTER

We decided very early in the design of the Starlight-1 that our stellar photometer would use photon counting techniques. When detecting very low light levels, photon counting offers optimum measurement accuracy and improved signal-to-noise ratios over dc measurements.

The presence of a photomultiplier output signal pulse of approximately 20 microamps will cause the amplifier/discriminator to output a 50-nanosecond differential ECL pulse into the counting electronics.

The photon counter has a 100-nanosecond pulse pair resolution which is commonly referred to as the dead time, d. As the mean pulse rate from the photomultiplier tube increases, a greater percentage of pulses will fail to be resolved by the preamp and counter electronics. The actual counting rate, N, can be approximated by $N=n(1+nd)$ where n is the number of counts per second displayed on the Starlight-1. If n is 1000 Hz, the percent error due to instrument dead time is only 0.01 percent, which is much smaller than the percent error due to standard Poisson counting statistics, but at 100,000 Hz the dead time error is 1.0 percent and the above correction should be applied to achieve maximum accuracy.

The Starlight-1 uses an 8-digit, LED display to show an output of up to 10^8 counts. If further counts are received an overflow indicator will light. (In practice, the overflow point would only be reached by making extremely long measurements of bright objects. Count rates of 1000 to 10,000 counts per second would be typical for most applications.) The display has been designed to hold the previous reading while the next measurement is in progress. This feature allows three 10-second measurements to be made and recorded within 30 seconds.

When using the digital readout, the user can choose one of two basic modes of operation—GATE and UNIT COUNT. In the GATE mode, the user can preselect counting time intervals of 10, 1, 0.1, or 0.01 seconds. At the end of the selected time interval, the accumulated counts are displayed. In the UNIT COUNT mode, counts are continuously updated to the LED display until the user signals the counter to stop. This mode is useful if the operator wants to use a single counting interval of longer than 10 seconds. The user can gate the counter manually through the START, STOP, and RESET push button control switches which are mounted on the front panel and/or by using the remote control option.

The digital output has many desirable features over other modes of output. First, you do not have to invest in costly chart recorders, microcomputers, current metering equipment, etc., in order to determine stellar magnitudes. The Starlight-1 is a complete unit in itself; no accessories are required. Second, the average of your readings is an exact number, not an "eyeball" estimate of a moving needle or chart recorder output. The average background count is also an exact number to be subtracted from the variable, comparison, or check star counts. The background count is never lost in the "mud" at the bottom of an analog scale.

Finally, you eliminate all potential sources of error which could be inherent in an additional piece of equipment. There is no need to change scales and set the zero point. The digital readout is probably the easiest output mode for an inexperienced person to work with and obtain accurate results.

We realized that there are many people who would prefer to use a chart recorder since it does have its own advantages of continuous data update and a permanent data recording. For these users, the Starlight-1 provides a post discriminator analog output signal which is completely independent of the digital counting system. In fact, both output modes can be used at the same time if it is desired! The analog circuitry provides a 1-volt full-scale output with a 1-second time response on all scale ranges. Four ranges—10^3, 10^4, 10^5, and 10^6 counts/volt—are provided with front panel switch selectable control. A 10-turn front panel potentiometer permits full-scale zero suppression.

For users who prefer direct signal access, a buffeted 50 nanosecond TTL output is available via a rear panel BNC connector. For maximum versatility, automation and sophistication in data collection and reduction, the Starlight-1 photon counter provides an external 25-pin I/O port for direct interface to a home or observatory microcomputer. In the UNIT COUNT mode, the computer will control the counting time interval; in the GATE mode, the Starlight-1 timing circuitry (accurate to within 50 ppm) will control the counting interval. In either mode, the computer can perform the necessary data reductions, corrections (dead time, extinction, transformation, etc.), and output format. The only limitation is the user's programming ability. The counter unit is shown in Fig. 12-6 with top cover removed.

Note: The computer used in the Mark III system interface has been described in great detail in Section I of this book.

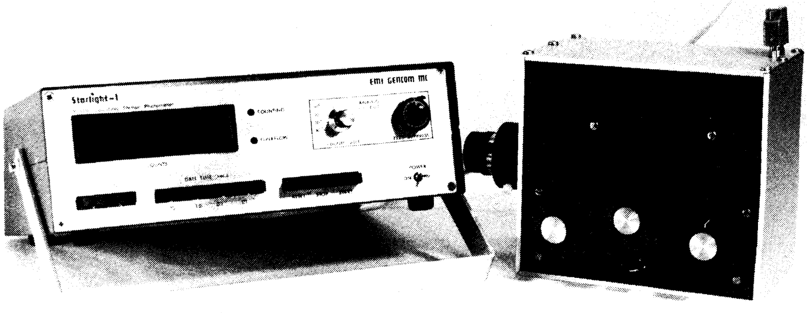


Fig. 12-6. Photometer electronics unit. (Courtesy EMI Gencom.)

A POSTSCRIPT

My editor has allowed me to include an update prior to publication. Two changes, noted earlier, have been incorporated in the current system. One is the use of the Hurst Mfg. hybrid stepper controller chip. The other is the use of the Intel 8254 chip where high-speed counting is involved. Clock/calendar chips are now available that can be directly tied to a microcomputer bus without an intermediary Intel 8255. A National Semiconductor MM58174 has been used in my latest system with good results.

The EMI Gencom Starlight-1 Stellar Photometer has been tested with very favorable results. If one desires to use stepper motors to control the diaphragm and filter wheels on this photometer, they must be added externally, and are not part of the photometer *per se*.

With the help of Douglass J. Sauer, I have added steppers for telescope slewing, and optical angle shaft eoncoders to determine telescope position. This will bring telescope pointing under direct computer control. To speed up the collection of data and reduce the effects of atmospheric variability on the observations, a second photometer is being added to the system. One photometer measures the variable star while the other measures the comparison star—simultaneously, of course. Two X-Y stages, each driven by two linear steppers, are used to position the photometers.

CHAPTER 13

The Mark III Software

In my review of computerized control systems used in astronomy in general and photometric systems in particular, it was clear that the Achille's heel of any system was the software. The great power and flexibility of a computer is an open invitation to disaster. One can easily envision all sorts of nifty little things one can do, and all in software! The sheer magnitude of the programming task can easily and quickly get out of hand, however, especially when one considers the requirements of thorough debugging and documentation. Unless one is extremely conservative in what is attempted, unmitigated disaster is the almost sure consequence. A few key early decisions and very strict adherence to them is the only thing that saved me.

The early decisions were: the use of a popular, well-documented microcomputer; exclusive use of interpretive BASIC; no hardware interrupts; a single input and output for operator communications; the use of independent program modules tied together by a main menu; and gradual evolution of the software. The temptations to deviate from these decisions were many. Editor/assembler is so much faster, but I always found a way around this by astute use of peripheral chips. Hardware interrupts are convenient at times, but one can make do with software checking of flags. Flashing lights, bells, and whistles would jazz up the system and sticking to the plain video monitor and small keypad has been extremely difficult and runs totally counter to my natural Rube Goldberg nature. It was sometimes difficult to get new program modules to play properly with the old ones, and after much evolution a program starts looking rather rag-tag and the temptation is strong to chuck the whole thing and write a clean looking one from scratch. All these temptations have been successfully resisted, however, and if the result is a bit unusual and patchy looking, just keep in mind not only that it works, but that I didn't have to spend an undue amount of time on programming.

In general, I have not strived for an efficient, compact program. Computer memory is very cheap these days, and unless you work for pennies an hour, it is better to buy more memory and write an inefficient, but hopefully easier understood, program, than to save space with a compact program. As I am not a professional programmer, such an approach doesn't bother me in the least, nor do I get hung up on avoiding an occasional GOTO so I can be with the in-group on the latest approach to structured programming. As with wire wrapping, I sort of like a modified rats-nest approach to things. With these rationalizations out of the way, let us proceed to the Mark III software.

As the program is naturally broken into modules, we will treat each one separately. Some of the modules have been discussed at some length in the first section of this book, so they will be included here merely for completeness with little discussion.

MAIN MENU

The Main Menu is the central clearinghouse for the subprograms and occupies lines below 2000 as shown in Example 1301. After identifying information and clearing the screen, 8255 No. 1 is programmed so that the remote keypad can be assessed. A one-dimensional array "T" is set aside for the clock subroutine. The number of stars in the display list "NL" is established, and an array set aside for the names of the stars and other identifying information (NS$). An array "M" is set aside for use later in calculating statistics. The data for the star names is then read, followed immediately by the star names in DATA statements.

After this preliminary information, the main menu proper starts on line 1000. The menu options are simply displayed on the screen using print statements. After the last menu item is displayed, the program jumps to the keypad subroutine for the operator's choice. On return, the choice is equal to "KB" and in line 1220 the branch is made to the operator's menu choice.

Example 13-1. Main Menu

```
1     REM FAIRBORN OBSERVATORY CONTROL PROGRAM II
2     CLS
3     REM PROGRAM #1 8255
4     OUT 131,152
8     DIM T(12)
10    NL=11: REM NUMBER OF STARS IN LIST
11    DIM NS$(NL)
12    DIM M(5)
100   REM DATA FOR STAR NAMES
120   FOR I=1 to NL: READ NS$(I): NEXT I
121   DATA "SPECIAL"
122   DATA "47 O DRA / HD17551"
123   DATA "BS 7275 / BS 7229"
124   DATA "26 AQL / HD182475"
125   DATA "HR 8156 / BD24 4370"
126   DATA "73 CYG / 71 CYG"
127   DATA "42 CAP / HD206893"
128   DATA "33 PSC / HD587"
129   DATA "ZET AND / ETA AND"
130   DATA "13 CET / HD3807"
131   DATA "39 CET / 34 CET"
200   DIM PD(5,2,2,2,2)
1000  REM MAIN MENU
1010  CLS
1020  PRINT"FAIRBORN OBSERVATORY PHOTOMETRIC PROGRAM"
1030  PRINT"YOU HAVE REQUESTED THE MAIN MENU"
1050  PRINT"1  SYNCRONIZE SECONDS WITH WWV"
1060  PRINT"2  SET CLOCK TO UNIVERSAL TIME"
1070  PRINT"3  SELECT STAR FROM MEMORY SET"
1080  PRINT"4  CONTROLLED PEP SEQUENCE"
1090  PRINT"5  DISPLAY FINDER MAP ON SELECTED STAR"
1100  PRINT"6  DO SIMPLE PHOTOMETRIC SEQUENCE"
1110  PRINT"7  DISPLAY CURRENT TIME"
1120  PRINT"8  PRINT OUT MATRIX"
1130  PRINT"9  CALCULATE DELTA MAGS"
1200  GOSUB 2000
1220  ON KB GOTO 3000,4000,5000,6000,7000,8000,9000,10000,11000
```

REMOTE KEYPAD SUBROUTINE

The remote keypad subroutine which starts on line 2000 is shown in Example 13-2. In line 2005, a check is made if the data ready flag is high coming from the remote keypad encoder. If it is low it keeps checking until it is high whereupon it sets KB equal to the lower 4

Example 13-2. Keypad Subroutine

```
2000 REM REMOTE KEYPAD SUBROUTINE
2005 IF INP(128) AND 16=16 THEN 2015 ELSE 2010
2010 GOTO 2005
2015 KB=INP(128) AND 15
2020 PRINT KB
2025 FOR IX=1 TO 300: NEXT
2030 RETURN
```

bits, displays the selected number (to help the operator know what key was pressed), does a short software delay to avoid double keying, and returns. See the earlier explanation in Section I of this book for more detail.

COUNTER SUBROUTINE

The counter subroutine starts on line 2500 and is shown in Example 13-3. It has been explained in detail earlier.

Example 13-3. Counter Subroutine

```
2500 REM COUNTER SUBROUTINE
2503 OUT131,152: REM PGM #1 8255
2505 OUT 67,144: REM PGM #2 8255
2509 REM SET MSM5832 TO NOT HOLD, READ, NOT WRITE
2510 OUT 67,0: OUT 67,3: OUT 67,4: OUT 65,15
2514 REM SET CNT 0 MODE 1, CNT 1 & 2 MODE 2
2515 OUT 35,50: OUT 35,116: OUT 35,180
2519 REM SET CNT 0 TO 10240, CNT 1 TO 0
2520 OUT 32,0: OUT 32,40: OUT 33,0: OUT 33,0
2525 PRINT"PRESS ANY KEY TO COUNT"
2530 GOSUB 2000
2539 REM TRIGGER PC1 #1 8255
2540 OUT 131,3: OUT 131,2
2545 IF INP(130)=0 THEN 2554 ELSE 2550
2550 GOTO 2545
2554 GOSUB 9003
2555 COUNT=INP(33)+256*INP(33)
2560 COUNT =65536-COUNT
2565 RETURN
```

SET CLOCK ROUTINE

The routine to set the clock is shown in Example 13-4. It also has been explained earlier in much detail.

Example 13-4. Clock Set Subroutine

```
4000 REM SET CLOCK ROUTINE
4005 CLS
4007 OUT 67,128
4010 PRINT"ENTER DATE YYMMDD (6 DIGITS)"
4012 FOR I=12 TO 7 STEP -1
4015 GOSUB 2000
4017 T(I)=KB: NEXT
4020 PRINT"ENTER TIME HHMM (4 DIGITS)"
4022 FOR I=5 TO 2 STEP -1
4025 GOSUB 2000
4027 T(I)=KB: NEXT
4030 T(5)=T(5)+8
4035 PRINT"PRESS ANY KEY TO START CLOCK"
4040 GOSUB 2000
4045 OUT 67,1: OUT 67,2
4050 FOR I=0 TO 12
4055 OUT 65,I: OUT 64, T(I)
4060 OUT 67,5: OUT 67,4
4065 NEXT I: OUT 67,0
4070 GOTO 9000
```

STARSELECT PROGRAM

The routine for selecting the stars to be observed is shown in Example 13-5. After clearing the screen and providing a header, the matrix of star names is displayed. The program then jumps to the remote keypad routine for the operator's choice. The selected star "SS" is then set equal to KB and a return made to the Main Menu.

Example 13-5. Star Select Subroutine

```
5000 REM STARSELECT PROGRAM
5010 CLS:PRINT"SELECT STAR FROM LIST"
5020 FOR I=1 TO NL
5030 PRINT I,NS$(I):NEXT I
5040 GOSUB 2000
5050 SS=KB:FOR I=1 TO 200: NEXT I
5060 GOTO 1000
```

CONTROLLED PEP SEQUENCE

This subprogram, listed in Example 13-6, is one of the longer and more involved of the modules. After clearing the screen, the photometric data "PD" array is initialized to zero. While this

Example 13-6. Controlled PEP Sequence

```
6000 CLS: REM CONTROLLED PEPSEQUENCE
6001 FOR I=1 TO 5: FOR J=1 TO 2: FOR K=1 TO 2: FOR H=1 TO 2:
     FOR L=1 TO 2
6002 PD(I,J,K,H,L)=0
6003 NEXT L: NEXT H: NEXT K: NEXT J: NEXT I
6010 NS=0
6011 HP=1
6012 PRINT"NUMBER OF COLORS?"
6015 GOSUB 2000
6017 NC=KB
6020 FOR I=1 TO 5
6030 NS=NS+1
6040 FOR J=1 TO 2
6050 FOR K=1 TO 2
6060 FOR H=1 TO NC
6100 CLS: REM INSTRUCTION DISPLAY
6110 PRINT"OPERATING INSTRUCTIONS"
6120 PRINT"COMP/VARBL",NS$(SS)
6130 PRINT"SEQUENCE NUMBER",I
6140 IF J=1 PRINT"COMP" ELSE PRINT"VARIABLE"
6150 IF K=1 PRINT"STAR" ELSE PRINT"BACKGROUND"
6160 IF H=1 PRINT"VISUAL" ELSE PRINT"BLUE,ETC"
6165 GOSUB 12000
6170 PRINT"PUSH ANY KEY TO START INTEGRATION"
6180 GOSUB 2000
6200 REM CALL PEP & TIME SUBROUTINES
6210 GOSUB 2500
6220 GOSUB 9003
6230 PRINT"TIME",TIME
6240 PD(I,J,K,H,1)=TIME
6250 PD(I,J,K,H,2)=COUNT
6260 GOSUB 2000
6280 CLS
6300 REM DISPLAY MATRIX
6305 PRINT"COMP/VARBL",NS$(SS)
6310 FOR HX=1 TO NC
6320 PRINT
6330 PRINT" CS/UT CS/UT CB/UT CB/UT VS/UT VS/CT VB/UT VB/CT"
6340 FOR IX=1 TO NS
6350 PRINT: N=0
6360 FOR JX=1 TO 2
6370 FOR KX=1 TO 2
6380 FOR LX=1 TO 2
6410 PRINT TAB(6*N) PD(IX,JX,KX,HX,LX);
6415 N=N+1
6420 NEXT LX: NEXT KX: NEXT JX: NEXT IX
6430 NEXT HX
```

```
6440 GOSUB 2000
6455 FOR IK=1 TO 200: NEXT IK
6500 CLS:REM OBSERVATION DISPOSITION
6510 PRINT"1 TERMINATE SEQUENCE"
6520 PRINT"2 REPEAT OBSERVATION"
6530 PRINT"3 DATA OK, CONTINUE SEQUENCE"
6540 GOSUB 2000
6545 FOR IK=1 TO 200: NEXT IK
6550 ON KB GOTO 1000,6100,6560
6560 NEXT H: NEXT K: NEXT J: NEXT I
```

would not be required for the first set of observations, it could be for later observations as observations can be terminated at any point and older data might not be over-written in the later part of the array. The observation number in the sequence, NS, is set to zero at this point.

The program then asks for the number of colors, NC, the observations are to be made in. This is an important flexibility in the program as it is often desirable to make observations in just the V band, while at other times one might want to make observations in V and B or in V, B, and U. As mentioned earlier, the colors are: V for visual (actually yellow); B for blue, and U for ultraviolet.

With these preliminaries out of the way, the control program starts with line 6020. The main FOR loops are controlled by the counters, I, J, K, H, and L. Counter I is the set of observations in the sequence, and while it normally runs from 1 to 5, it may be terminated at any point by the operator. Counter J is the star identifier with J=1 for the comparison star and J=2 for the variable star. Counter K is the star/background identifier, with K=1 for star, and K=2 for background. Counter H is the filter color identifier with H=1 for V, H=2 for B, and H=3 for U. Counter L is used for data packing (one of my few efficiency moves) with L=1 being the time, and L=2 being the count from the v/f converter and thus the actual photometric reading. These various counters are used not only to control the data taking sequence, but also to store the data (time or count) in the proper place in the data array for use by other subroutines later.

Starting on line 6100, the instructions are given to the operator. The operator is told what stars he is supposed to be looking at, whether it is the comparison or variable star he should now be looking at, and whether it is the star itself or the nearby background. He is then instructed which color filter should be in place, and told to push any key when ready to start the count.

At this point (line 6210), the count routine is called and immediately on completion of the count, the routine is called to read the clock. The time is then displayed, and the time and count stored as data in the proper array elements. At line 6020 an operator controlled pause is introduced by going to the keypad. This gives the operator time to look over the displayed light curve and think about its quality. By pushing any key the program then displays all the data gathered so far. This is of considerable importance, as it allows the operator to compare the latest observation with previous ones so any gross irregularities can be spotted (such as forgetting to flip the microscope mirror out of the light path). As it is convenient to see like readings under each other, the different colors are displayed separately on the screen. To avoid upsetting the array counters I, J, K, etc., new counters, IX, JX, KX, etc., are used for the display. At line 6440, the keypad is again called to introduce an operator controlled delay.

Pushing any key brings up the guide to the disposition of the most recent observation. TERMINATION returns one to the main menu, REPEAT brings up the instructions exactly as before, while DATA OK steps one to the next observation in the sequence. The normal sequence is comparison star, comparison background, variable star, and variable background. This is repeated up to five times in a row.

FILTER STEPPER SUBROUTINE

In the control program, the counter "H" gives the position index that the filter wheel ought to be at. If a new variable "HP" for the previous position of the filter wheel is introduced, then a simple algorithm exists for determining how many filter positions one needs to move and the direction of movement. If the V position of the filter wheel is index 1, B is 2, and U is 3, and clockwise is + and counterclockwise is −, then we have the situation as shown in Table 13-1.

The algorithm is shown in Fig. 13-1 and its computer realization is shown in Example 13-7.

DIAPHRAGM STEPPER SUBROUTINE

The case of the diaphragm stepper is even simpler, as it merely has to "wag" between the star and background.

Table 13-1. Filter Position Logic

HP Previous Position	H Position Desired	H-HP Movement Desired	Comment
1	1	0	V only mode
1	2	+1	Step from V to B
2	1	−1	Step back from B to V
2	3	+1	Step from B to U
3	1	−2	Step back from U to V

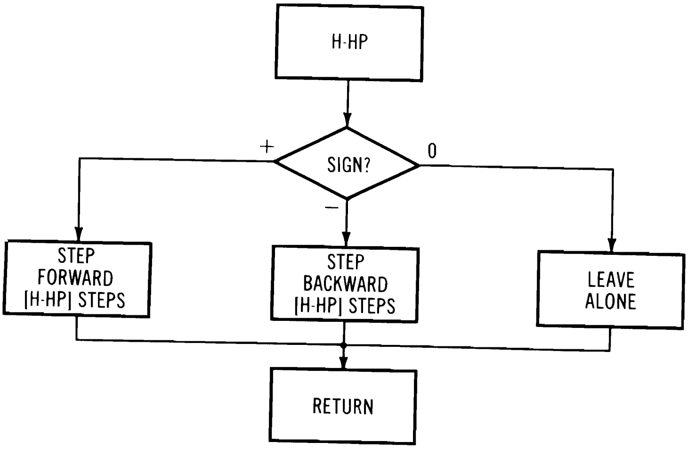


Fig. 13-1. Algorithm for the diaphragm stepper.

Example 13-7. Filter Step Subroutine

```
12000 REM FILTER STEP SUBROUTINE
12010 Q=H-HP
12020 SQ=SGN(Q)
12030 AQ=ABS(Q)
12040 ON SQ+2 GOTO 12100,12200,12300
12100 REM STEP BACKWARDS
12110 QX=AQ*17
12120 FOR QZ=1 TO QX
12130 OUT 131,4: OUT 131,5: NEXT
12140 GOTO 12400
12200 REM STAY SAME
12210 GOTO 12400
12300 REM STEP FORWARD
12310 QX=AQ*17
```

```
12320 FOR QZ=1 TO QX
12330 OUT 131,0: OUT 131,1: NEXT
12400 HP=H
12410 RETURN
```

READ CLOCK PROGRAM

The program to read the clock is shown in Example 13-8. As it has been previously described in detail, no comment will be made here.

Example 13-8. Clock Read Subroutine

```
9000 REM READ CLOCK
9001 GOSUB 9003
9002 GOTO 9045
9003 REM SUBROUTINE TO READ CLOCK
9004 OUT 67,144
9005 OUT 67,1: OUT 67,3: OUT 67,4
9008 IQ=I
9010 FOR I=0 TO 12
9015 OUT 65,I
9017 T(I)=INP(64)
9020 NEXT I: OUT 67,0
9022 T(5)=T(5) AND 3
9030 DATE=T(7)+10*T(8)+100*T(9)+1000*T(10)+10000*T(11)+100000*T(
9040 TIME=T(0)+10*T(1)+100*T(2)+1000*T(3)+10000*T(4)+100000*T(5)
9041 I=IQ
9042 RETURN
9045 CLS: PRINT DATE,TIME
9050 PRINT"PRESS ANY KEY TO CONTINUE"
9055 IF INP(128) AND 16=16 THEN 9060 ELSE 9057
9057 FOR I=1 TO 250: NEXT: GOTO 9000
9060 CLS: FOR I=1 TO 500: NEXT
9065 GOTO 1000
```

MATRIX PRINT SECTION

Once a photometric sequence is completed or terminated, it would be convenient to be able to retain a record of the raw data in the event questions should arise later. This, of course, is standard scientific practice. Shown in Example 13-9 is a routine that prints out the data in the data array. It might be noted that it is almost identical to the program statements used to place the data array on the video monitor except that PRINT has been replaced with LPRINT.

Example 13-9. Matrix Print Subroutine

```
10000 CLS: REM MATRIX PRINT SECTION
10300 REM PRINT MATRIX
10305 LPRINT"COMP/VARBL",NS$(SS)
10310 FOR HX=1 TO NC
10320 LPRINT
10330 LPRINT" CS/UT CS/CT CB/UT CB/CT VS/UT VS/CT VB/UT VB/CT"
10340 FOR IX=1 TO NS
10350 LPRINT: N=0
10360 FOR JX=1 TO 2
10370 FOR KX=1 TO 2
10380 FOR LX=1 TO 2
10410 LPRINT TAB(6*N) PD(IX,JX,KX,HX,LX);
10415 N=N+1
10420 NEXT LX: NEXT KX: NEXT JX: NEXT IX
10430 NEXT HX
10440 CLS:PRINT"PRINTOUT COMPLETED"
10450 FOR IK=1 TO 300:NEXT IK
10455 LPRINT
10460 GOTO 1000
```

ANALYSIS PROGRAM

One of the nicest features of the Mark II and Mark III systems is the almost instantaneous analysis at the end of each set of readings. No longer does one have to wait until the next day or week to know the results of the analysis, and what the standard error of observation is for each set. This is not a mere satisfying of idle curiosity, for if the standard error is too large, then further observations are usually a total waste of time, and it is nice to know this right away rather than making a whole night of useless observations (even astronomers like to sleep at night on occasion).

The program that does the analysis is shown in Example 13-10. It is interesting to note that this program was derived from an earlier program used to analyze the data read-off of strip charts in the Mark I system. This is a good example of control system evolution. If some process involves manual calculations, a microcomputer can be used to make these calculations. The system is then extended to log the data. Finally, active control of the system is initiated.

After printing and displaying header information, date, etc., the

Example 13-10. Analysis Subroutine

```
11000 CLS:LPRINT:REM DELTA MAN ANALYSIS
11010 A$="FAIRBORN OBSERVATORY PEP ANALYSIS—1980"
11020 PRINT A$: LPRINT A$
11030 PRINT"JUL DATE=";DJ;"MONTH=";MT;"DAY1=";D1;"DAY
      2=";D2
11040 LPRINT"JUL DATE=";DJ;"MONTH=";MT;"DAY1=";D1
      "DAY2=";D2
11045 FOR H=1 TO NC
11047 PRINT:LPRINT
11050 PRINT"COMP/VARBL=";NS$(SS);:LPRINT
      "COMP/VARBL=";NS$(SS);
11060 IF H=1 THEN A$=" VISUAL" ELSE A$=" BLUE, ETC."
11070 PRINT"FILTER=";A$: LPRINT"FILTER=";A$
11080 PRINT" UT JD MAG": LPRINT" UT JD MAG"
11100 FOR I=1 TO NS-1
11110 NM=PD(I,2,1,H,2)-PD(I,2,2,H,2)
11120 DN=(PD(I,1,1,H,2)-PD(I,1,2,H,2)+ PD(I+1,1,1,H,2)-
      PD(I+1,1,2,H,2))/2
11130 A=NM/DN: B$=" "
11140 DM=-2.5*LOG(A)/2.3025851
11150 DM=FIX(DM*1000)/1000
11160 UT=PD(I,2,1,H,1)
11162 HR=FIX(UT/10000)
11164 MN=FIX((UT-10000*HR)/100)
11166 SC= UT-10000*HR-100*MN
11170 HD=(HR+MN/60+ SC/3600)/24
11180 HD=FIX(HD*100000)/100000
11190 IF HD<0.5 THEN 11200 ELSE 11210
11200 JD=0.5+HD: GOTO 11220
11210 JD=-0.5+HD
11220 PRINT UT;JD;B$;B$;DM: LPRINT UT;JD;B$;B$;DM
11230 M(I)=DM: NEXT I: SUM=0
11235 IF NS <3 THEN 11300
11240 FOR I=1 TO NS-1: SUM=SUM+M(I): NEXT I
11250 AVG=SUM/(NS-1): PRINT"AVERAGE=";AVG: LPRINT
      "AVERAGE=";AVG
11260 FOR I=1 TO NS-1: SX=SX+(M(I)-AVG)[2: NEXT I
11270 SX=SX/(NS-2): SIG=SQR(SX)
11280 SIG=FIX(SIG*1000)/1000
11290 PRINT"STD DEVIATION=";SIG: LPRINT"STD DEVIATION=";SIG
11300 A=INP(130)
11310 IF A=0 THEN 11300
11320 FOR I=1 TO 200: NEXT I: GOTO 1000
```

program calculates the differential magnitudes according to the formula

$$DM = -2.5 \text{ Log } \frac{Vs - Vb}{1/2\,(Csx - Cbx + Csy - Cby)}$$

where,

DM is the differential magnitude,
Vs is the variable star (count),
Vb is the variable background,
Csx is the comparison star before the variable,
Cbx is the comparison background before the variable,
Csy is the comparison star after the variable,
Cby is the comparison background after the variable.

These various values are stored in the PD array. For instance, VS (1) is PD(I,2,1,H,2). In line 11140 the factor of 2.3025851 is a conversion from logarithms base 10 to base e, as the TRS-80 can only handle base e. Starting on line 11160, the time the variable star was observed is broken down into hours and minutes and then converted into a fractional Julian day. The calculated results are then displayed and printed. The last portion of the program calculates the mean and standard deviation of the observed values.

This completes the description of the Mark III software. There are less than 300 lines of code, with no attempt at compacting or efficiency. As control programs go, it is quite short, yet does a lot.

Appendix A

Sources for Additional Information

8255 and 8253 Chips. Intel Corp., 3065 Bowers Ave., Santa Clara, CA 95051.

MSM5832 Chip. OKI Semiconductor, Inc., 1333 Lawrence Expressway, Suite 401, Santa Clara, CA 95051.

AD537 Chip. Analog Devices, Route 1 Industrial Park, P. O. Box 280, Norwood, MA 02062.

Stepper Motors and Controllers. Hurst Mfg. Corp., Box 326, Princeton, IN 47670.

Astronomical Photoelectric Photometry. The book "Photoelectric Photometry of Variable Stars—A Practical Guide for the Smaller Observatory," by D. S. Hall and R. M. Genet is available for $17.95 postpaid from Fairborn Observatory, 1247 Folk Road, Fairborn, Ohio 45324.

Appendix B

Modifications for the TRS-80 Model III

While the programs and hardware described in this book were developed on a Radio Shack TRS-80 Model I, Level II, they are applicable to the Model III also. There are three differences.

First, the pin assignments on the Model III are different from those on the Model I. Just keep this in mind when you wire up the cable going from the computer to the interface.

Second, at the very beginning of any control program with the Model III, it is necessary to enable the output by setting bit D4 at I/O port ECH, or 236_{10}. To do this without disturbing the other bits use the statement

```
OUT 236, (INP(255) OR 16)
```

This will set the necessary bit every time you run your program.

Finally, to get data into the Model III from the interface, the $\overline{\text{EXTIOSEL}}$ (External I/O Select) line must be brought low. The $\overline{\text{EXTIOSEL}}$ line should be brought low every time one of the chips on the interface is selected (i.e., a $\overline{\text{CS}}$ generated) *and* the $\overline{\text{IN}}$ line is low. A simple way of doing this is illustrated in Fig. B-1 for an 8255 chip. Other approaches will also work as long as the condition previously stated is met.

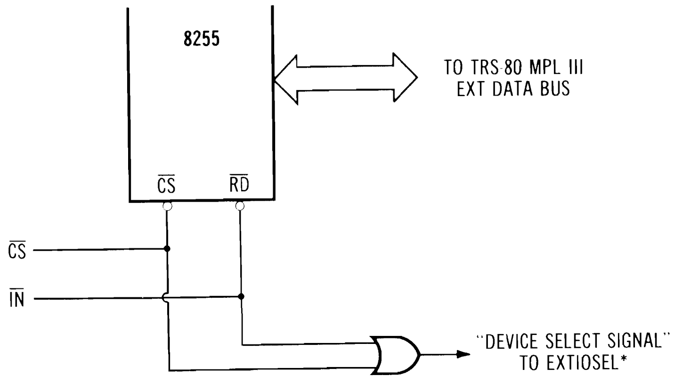

8255
TO TRS-80 MPL III EXT DATA BUS
$\overline{CS}$
$\overline{RD}$
$\overline{CS}$
$\overline{IN}$
"DEVICE SELECT SIGNAL" TO EXTIOSEL*

Fig. B-1. When both a $\overline{\text{CS}}$ and $\overline{\text{IN}}$ are low, the $\overline{\text{EXTIOSEL}}$ line is brought low, letting the Model III know that it is about to get some data from the outside world.

Index

The Blacksburg Group

According to Business Week magazine (Technology July 6, 1976) large scale integrated circuits or LSI "chips" are creating a second industrial revolution that will quickly involve us all. The speed of the developments in this area is breathtaking and it becomes more and more difficult to keep up with the rapid advances that are being made. It is also becoming difficult for newcomers to "get on board."

It has been our objective, as The Blacksburg Group, to develop timely and effective educational materials that will permit students, engineers, scientists, technicians and others to quickly learn how to use new technologies and electronic techniques. We continue to do this through several means, textbooks, short courses, seminars and through the development of special electronic devices and training aids.

Our group members make their home in Blacksburg, found in the Appalachian Mountains of southwestern Virginia. While we didn't actively start our group collaboration until the Spring of 1974, members of our group have been involved in digital electronics, minicomputers and microcomputers for some time.

Some of our past experiences and on-going efforts include the following:

–The design and development of what is considered to be the first popular hobbyist computer. The Mark-8 was featured in Radio-Electronics magazine in 1974. We have also designed several 8080-based computers, including the MMD-1 system. Our most recent computer is an 8085-based computer for educational use, and for use in small controllers.

–The Blacksburg Continuing Education Series™ covers subjects ranging from basic electronics through microcomputers, operational amplifiers, and active filters. Test experiments and examples have been provided in each book. We are strong believers in the use of detailed experiments and examples to reinforce basic concepts. This series originally started as our Bugbook series and many titles are now being translated into Chinese, Japanese, German and Italian.

–We have pioneered the use of small, self-contained computers in hands-on courses for microcomputer users. Many of our designs have evolved into commercial products that are marketed by E&L Instruments and PACCOM, and are available from Group Technology, Ltd., Check, VA 24072.

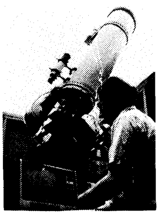
–Our short courses and seminar programs have been presented throughout the world. Programs are offered by The Blacksburg Group, and by the Virginia Polytechnic Institute Extension Division. Each series of courses provides hands-on experience with real computers and electronic devices. Courses and seminars are provided on a regular basis, and are also provided for groups, companies and schools at a site of their choosing. We are strong believers in practical laboratory exercises, so much time is spent working with electronic equipment, computers and circuits.

Additional information may be obtained from Dr. Chris Titus, the Blacksburg Group, Inc. (703) 951-9030 or from Dr. Linda Leffel, Virginia Tech Continuing Education Center (703) 961-5241.

Our group members are Mr. David G. Larsen, who is on the faculty of the Department of Chemistry at Virginia Tech, and Drs. Jon Titus and Chris Titus who work full-time with The Blacksburg Group, all of Blacksburg, VA.

Real-Time Control With the TRS-80®

- Helps you plan and develop a real-time data-logging or control system in a manner that will avoid major pitfalls and save you time and money.
- Emphasizes high reliability, speed, and sophistication.
- Shows you how to communicate with the computer by remote keypad and video monitor.
- Describes counters as timers, delay devices, and dividers.
- Averages signals automatically with a voltage-to-frequency converter.
- Discusses stepper motors.
- Explains how to eliminate the reading of strip charts, writing down data, and keying in data.
- Permits error-free operation.
- Includes a detailed case example (with programs) of the development of a real-time control system.



Russell M. Genet was born and raised in southern California. As a young boy, he enjoyed building radios, telescopes, and model rockets. At 17, he joined the Air Force and served as an electronics technician in North Africa and as an electronics instructor in Mississippi. After receiving his electrical engineering degree from the University of Oklahoma, he served as a missile guidance project officer. Currently, Russ is an Air Force civil servant at Wright-Patterson AFB, where he serves as director of a research section at the Human Resources Laboratory. While Russ does fly small planes, and has an amateur radio station (N8HH), his main hobby is astronomy. He is currently using a TRS-80® in his back yard observatory to study eclipsing binary stars.

[illegible] C.
43C [illegible] 268 USA

$14.95/21831

ISBN: 0-672-21831-3