Dedicated Controllers

. . . there is money to be made

Michael J. Myers
Ann Arbor MI 48105

Mike's article demonstrates that there are money-making opportunities out there in dedicated controller applications. And . . . the hobbyist should be taking advantage of the situation and doing something about it. His article should stimulate some thought along these lines, and if nothing else, get you busy with some experimenting so you'll be ready for that money-maker when it comes along. — John.

Are you tired of playing games? Try some machine language programming on your 8080, 6800, or 6502 and make some money with it! Let me start my story at the beginning.

About two years ago I was approached to design a process controller on a consulting basis for a small company producing special film processors. They don't have enough electronics work to warrant having an electronics shop, or even a full-time technician.

The problem was to sequence a number of valves and pumps to control a custom film processor. The process is rather slow, taking a period of up to an hour. The customer needed a programmable sequencer that could cause the valves and pumps to operate in a preprogrammed time sequence. The programming was to be straightforward, as the pro-

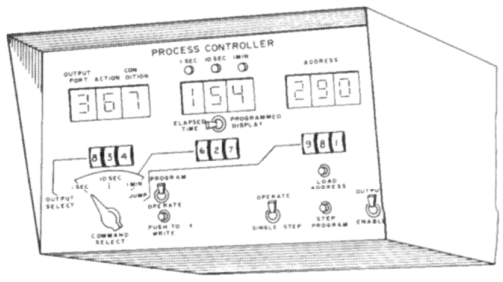

Fig. 1. Process controller front panel.

cess to be run might change frequently. Capability to control four or five processors simultaneously was also required.

The First, Non-micro Version

At the start of the project we looked at microprocessors (the 8008 was just getting into general use), and on the basis of my experience with digital logic and inexperience with microcomputers, I decided to go the hard-wired logic route. The description of this first unit, which follows, is presented for the purpose of illustrating the differences in complexity and ease between the hard-wired and microcomputer approach. The hard-wired version is still in daily use by the customer and operating faithfully.

I chose CMOS logic and memory so battery backup could be provided. The memory capacity is 256 16-bit words. Since most of the users are not familiar with hexadecimal notation, and thumbwheel switches are available in Binary-Coded Decimal (BCD), the words are divided into four BCD digits. There are three types of instructions, with variations on one of them: Fig. 1 is an illustration of the controller front panel.

A Select instruction is identified by the first digit, as are the others. The second digit identifies one of ten output ports. Each output is a 4-bit latched type. The third digit specifies the output to be present at the port (a 9 specifies the 4 bits to be 1001, etc.). The last digit of a select instruction specifies one of three conditions. Zero means "end this step at the programmed time." The digit 1 indicates a wait for a condition to be present, even if the programmed time is reached (used, for example, in a wash step in which wash is continued until the impurity of discharge water reaches a low level, measured by its conductivity). The number 2 means "proceed on an external signal", even if the programmed time has not yet been reached.

The second type of command is a time instruction. The first digit indicates the time units to be counted; seconds, ten seconds, or minutes. The last three digits indicate how many intervals are to be counted. For example, the code for ten seconds, followed by 036, would indicate a time of 360 seconds, or six minutes, during which to hold the output specified in the select instruction.

The third type of instruction is a jump, consisting of the jump code and a three-digit address to which to jump.

Because programming takes some time, and some of the sequences are used frequently, provision is made for ten subroutines stored in PROM. Each subroutine is thirty-two words long and is stored in a pair of 8223 fusible link programmable memory chips. These are accessed by jumping to addresses 300 to 309. LED displays are provided for current memory location, programmed time for this step, elapsed time for this step, and the output selected by the select code previous to the current time instruction.

As the project developed, interfaces were designed to drive nine solenoid valves and three pumps. Each interface uses two output ports; one for the valve information and one for the pumps. Valves operate one at a time, and are specified by one decimal digit (zero is an all off code). The three pumps each have a bit assigned to indicate their status, either on or off, and therefore may operate independently.

The finished programmer has ten different printed circuit card designs, and eighteen cards total. There are about 200 CMOS and TTL integrated circuits. It took two months of evenings and two solid weekends to wire it, and about as long to get it all debugged. It was delivered in January, 1976, and has operated since with no failures.

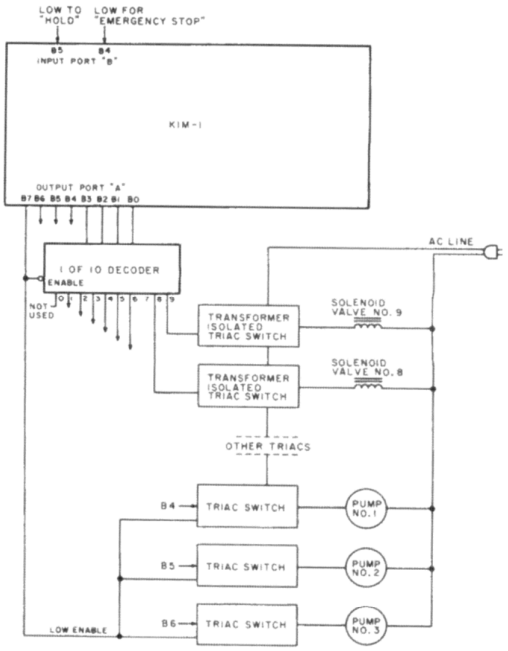

Fig. 2. KIM-based process controller block diagram.

Recently I received a call from the customer asking about more programmers. We discussed a simpler version, able to drive only one processor, and the possibility of a duplicate of the original.

Calling on KIM for the 2nd Version

I have been eyeing the growing use of microcomputers for some time, and was beginning to feel that if I didn't get some hands-on experience with one, I would soon be obsolete. I went to the local computer store and came home with a KIM-1 board.

My previous experience in programming was essentially having written a few programs for an HP-65 programmable calculator. I eagerly started reading the MOS Technology programming manual and KIM-1 user manual, which gives a sample program and instructions for using the tape cassette interface and the self-contained hex keyboard. After a few puzzles things started to make sense, and I was able to do some simple operations, like add two two-digit numbers and store the results in memory.

The KIM interval timer looked like a natural for making a clock program, so I started on one.

I could see that this machine could easily do the sequencer job and immediately wrote my customer asking for a purchase order for some hours of my time to study the application of KIM for his use and to come up with some operating programs. Figuring that this was as good a way as any to learn the KIM, I started programming.

After a week of evenings I had a program that went through a table of data containing desired output conditions and times in a fixed repeated four-word format of: 1. output condition; 2. hours; 3. minutes; 4. seconds.

I connected some LEDs with buffers to an output port and verified operation of the program. A long look at the KIM monitor program (well documented in the user manual) gave me a clue as to how to display the time on the KIM. After a few false starts that was accomplished, and I wrote the customer again.

A few days later I received a call telling me that a purchase order was on the way for my time and one system to be assembled with connectors compatible with the interfaces, and a cassette recorder checked for compatibility with KIM. He said, "Fine on the program, but how hard would it be to incorporate the hold feature of the original unit? Also it would be nice to have a choice of going through the sequence once, or for a programmed number of times."

Back to the drawing board. Another couple of evenings and the following were incorporated. After the last desired step, program 00 to turn all outputs off. Then:

1. EE for unconditionally ending the program after one time through the sequence.
2. BB for branch to beginning and continue to repeat the sequence until manually stopped.
3. BC for branch to the beginning counted. The count for the desired number of repeats to be placed in the next location. This count is decremented once each time through the sequence. When it reaches zero the operation terminates.

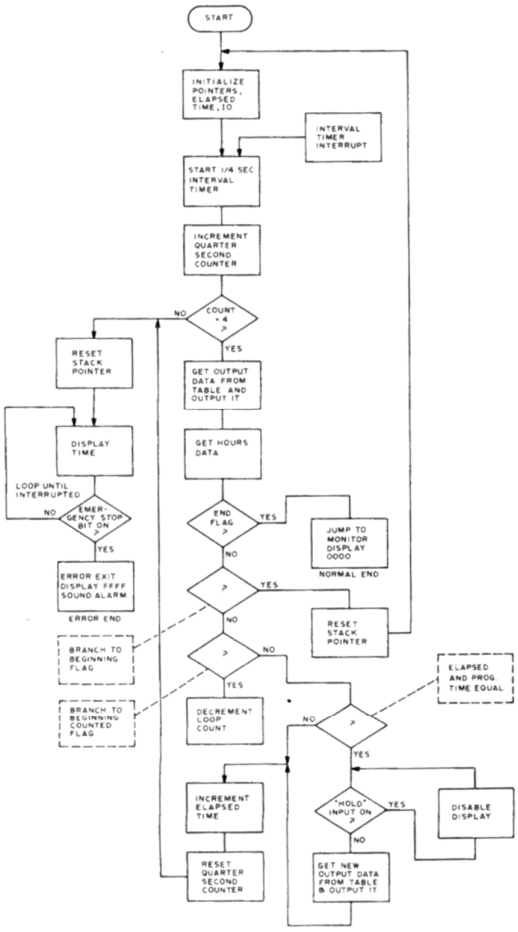

Fig. 3. Flowchart of controller program.

It occurred to me that the KIM tape load routine has two possible endings. When the tape loads correctly, the display indicates 0000XX; if there is an error, it displays FFFFXX. A look through the monitor listing shows that a jump to the appropriate addresses will cause either of these two actions. Any normal ending of the operation will leave the display showing 0000XX.

KIM's second input/output port uses pin B7 for the interval timer interrupt output, and pin B6 is not brought out (refer to Fig. 2). B5 therefore is now programmed to cause the program to hold at the end of a step, even though the time has elapsed, as long as B5 is held low. During this condition the display is blanked.

B4 is programmed as an emergency stop input when it is pulled low, for example, by a "machine jammed" signal from a limit switch. The sequence terminates and KIM displays FFFXX and sounds an alarm.

The KIM system has one bad habit: When in monitor mode or when power is first applied, all I/O lines revert to being inputs until otherwise instructed by the program. That sounds like a good idea except for one problem. There are pull up resistors, so all the inputs go high, looking like outputs that are on. Since only three bits are required to control three pumps, the last bit of output port A (B7) is used as a low enable. That is, it must be made low by the program to enable the interface outputs.

At this point I am reserving the B port bits 0-3 for the more complex multiple output interface unit. Bits 2-0 will select interface 0 to 7 and bit 3 will be a strobe to latch the data from the A outputs into the interface selected.

The Controller Software

Fig. 3 is a flowchart of the controller routines. The listings are shown in Program A. The data table started out to be all of page 2 of the KIM memory. I later found out how to increment my base address for the indirect, Y addressing, and added all of page 3 as well. A maximum of 127 steps and a terminating code may therefore be programmed.

Note that the times are total elapsed. The timer does not reset to zero at the end of the previous time interval. Also, the Initialize routine only reenters values that are changed as the program runs. Several constants must be set up for proper operation. These will be on the program cassette and so will not be of concern to the user. These locations and values are: HRL,00; MINL,40; SECL,40; INDLO,00. (The program listing has been verified by entering it by hand from the listing, and testing its operation.)

Program B is a sample of the programming required by the user to get data into the system during initialization. After this is done the first time it isn't necessary to go through these steps again because the program and this data are recorded on the cassette.

Some Closing Thoughts

This program might be of interest to anyone wanting to "run the house" on their computer. If, for example, the last step ends at 23:59:59, a 24 hour clock is set up. The BB instruction is used to keep it running.

Eight outputs are then available to: 1. Switch the thermostat between day and

Addr	Data	Comment
0200	00	SET ALL OUTPUT BITS TO ZERO
0201	00	HOURS FOR THIS STEP =0
0202	01	MINUTES = 1
0203	00	SECONDS = 0
0204	0F	SET OUTPUT B7 TO B4 OFF, B3 TO B0 ON
0205	00	HOURS STILL ZERO
0206	02	MINUTES =2
0207	00	SECONDS =0
0208	F0	SET OUTPUT B7 TO B4 ON AND B3 TO B0 OFF
0209	00	HOURS = 0
020A	03	MINUTES = 3
020B	15	SECONDS = 15
020C	00	OUTPUTS ALL OFF
020D	BC	START OVER AGAIN (REPLACES HOURS DATA)
020E	09	COUNT 9 TIMES THROUGH, THEN STOP

Program B. Sample user program to sequence outputs.

night modes; 2. Turn on the coffee pot in the morning; 3. Turn on the radio, alarm, tape player, etc.; 4. Control the yard or front porch light; 5. Control a night light; 6. Control an attic fan, shutting it off at an appropriate time late at night; 7. Control lights around the house in a random-looking manner while you are away. You can use a 96 hour cycle and have the lighting pattern repeat every four days; 8. Program the TV to shut off at a predetermined time. I'm certain you can think up others.

I would like to emphasize a few points. I now have no I/O device other than the KIM display and hex keyboard. My system consists of a KIM board and a power supply. Yes, it did get to be a pain to reenter a couple of hundred hex digits with the keyboard just to add a CLD instruction near the top of the program, but it's really not all that bad.

The original hard-wired programmer cost my customer $5000, plus a trip to the plant for installation and training. This time he will get a packaged KIM-1 system, a cassette recorder/player, several cassettes (each a prerecorded copy of the operating portion of the program), a complete program listing, and operating instructions for recording his end program and data on a cassette. When the same process is to be run, he only has to load from a cassette. This convenience eliminates the need for a subroutine in PROM. Total cost this time will be a one-time charge of $600 for the program and $450 for the packaged and tested KIM system. Additional units will cost $450 each. Additional programmed features as they come to the customer's mind will be supplied on the basis of an hourly charge for programming.

What would a duplicate of the original cost? I already have the PC board negatives and extensive wiring lists and corrected schematics, so of course there would be a reduction in cost. The labor, however, is a very large item, and I would have to charge $3000 or so.

At this point, you are saying, "Great! Here's a guy that has been designing circuits for a long time telling us how easy it is to make money with a computer. I'm no engineer. How can I do it?"

Frankly I think it is easier than any of us imagine. There are lots of companies that are not in the electronics business (believe it or not). Some of these are not in a position to be able to support full-time electronics people. The microcomputer field is moving so fast that most companies don't yet realize what can be done. Spread the word. Go find an application and go to it! "Where do I begin to look?" you say. Contact a nearby small university or college. Likely customers are people doing research in chemistry, psychology, physiology, music schools, language departments, small medical facilities.

Go talk to some of these people. Arrange a brainstorming session. Listen to what they need. Tell them what can be done. Maybe you'll have to work free the first time around, but you don't get paid for playing games either. And in my opinion it is just as much fun! Who knows, maybe you (or I) will find an application that will lead to a whole new business ... opening up a whole new area for the application of microprocessor technology. The time is right now. The applications far exceed the number of people with the knowledge to make them work. ■

Program A. Listing for process controller program (continued on following page).

Addr	B1	B2	B3	Label	Oper.	Operand	Comment
000A	A9	00		INIT	LDA	#$00	
000C	85	01			STA	HR	THESE INSTRUCTIONS RESET CLOCK TO ZERO
000E	85	02			STA	MIN	
0010	85	03			STA	SEC	
0012	85	09			STA	YBUF	INITIALIZES Y TO START OF DATA TABLE
0014	8D	00	17		STA	PAD	SET OUTPUT PORT DATA TO ALL ZERO
0017	8D	FF	17		STA	IRQHI	SET UP HI BYTE OF INTERRUPT VECTOR
001A	A9	02			LDA	#$02	
001C	85	08			STA	INDHI	HI BYTE OF INDIRECT BASE ADDRESS
001E	A9	10			LDA	#$10	
0020	85	00			STA	CNT	INITIALIZE QUARTER SECOND COUNTER
0022	A9	2C			LDA	#$2C	
0024	8D	FE	17		STA	IRQLO	LO BYTE OF INTERRUPT VECTOR
0027	A9	FF			LDA	#$FF	
0029	8D	01	17		STA	PADD	DEFINES OUTPUT PORT A AS OUTPUT
002C	A9	17		INTR	LDA	#$17	INTERRUPT TO HERE
002E	18			CORR	CLC		THIS IS A TIMING LOOP TO CORRECT THE TIME
002F	E9	00			SBC	#$00	PER INTERRUPT TO BE ¼ SECOND
0031	D0	FB			BNE	CORR	
0033	A9	F4			LDA	#$F4	COUNT FOR INTERVAL TIMER (COUNTS 244 X 1024
0035	8D	0F	17		STA	TIMER	MICROSECONDS, 144 SHORT, LOOP CORRECTS)
0038	06	00			ASL	CNT	SHIFT CNT LEFT
003A	90	75			BCC	INT6	BRANCH IF NOT FOUR COUNTS
003C	A4	09			LDY	YBUF	
003E	B1	07			LDA	IND,Y	GET FIRST OUTPUT DATA FROM TABLE

```
0040  8D  00  17        STA  PAD       APPLY DATA TO OUTPUT PORT
0043  C8              INY
0044  B1  07          LDA  IND,Y     GET HOURS DATA, TIME TO END THIS STEP
0046  C9  EE          CMP  #$EE      TEST FOR END PROGRAM FLAG
0048  F0  12          BEQ  END       IF EQUAL GO TO END ROUTINE
004A  C9  BB          CMP  #$BB      TEST FOR "BRANCH TO BEGINNING" FLAG
004C  F0  11          BEQ  RESTART
004E  C9  BC          CMP  #$BC      TEST FOR "BR. TO BEGIN., COUNTED" FLAG
0050  D0  13          BNE  CHR       IF NOT COMPARE HOUR
0052  C8              INY
0053  B1  07          LDA  IND,Y     GET LOOP COUNT FOLLOWING BC FLAG
0055  18              CLC            SET BORROW FLAG
0056  E9  00          SBC  #$00      SUBTRACT 00 WITH BORROW (SUBTRACT 1)
0058  91  07          STA  IND,Y     RETURN DECREMENTED LOOP COUNT TO BC FLAG+1
005A  D0  03          BNE  RESTART   BRANCH IF LOOP COUNT NOT ZERO
005C  4C  25  19  END     JMP  MONITOR   TAPE LOAD ROUTINE NORMAL EXIT, DISP. 0000XX
005F  A2  FF      RESTART LDX  #$FF      RESET STACK POINTER, ADVANCED BY INTERRUPT
0061  9A              TXS            SUBSTITUTES FOR RTS INSTRUCTION
0062  4C  0A  00        JMP  INIT      GO TO INITIALIZE (NORMAL PROGRAM START POINT)
0065  C5  01      CHR     CMP  HR        COMPARE PROGRAMMED TO ELAPSED HOUR
0067  F0  03          BEQ  CMIN      IF HOURS EQUAL CHECK MINUTES
0069  4C  9A  00        JMP  INCR      IF NOT, INCREMENT CLOCK
006C  C8          CMIN    INY
006D  B1  07          LDA  IND,Y     GET MINUTES DATA
006F  C5  02          CMP  MIN       COMPARE MINUTES PROGRAMMED TO ELAPSED
0071  F0  03          BEQ  SEC       IF EQUAL CHECK SECONDS
0073  4C  9A  00        JMP  INCR      IF NOT, INCREMENT CLOCK
0076  C8          CSEC    INY
0077  B1  07          LDA  IND,Y     GET SECONDS DATA
0079  C5  03          CMP  SEC       COMPARE PROGRAMMED TO ELAPSED
007B  F0  03          BEQ  ADVANCE   IF EQUAL GET NEW OUTPUT DATA
007D  4C  9A  00        JMP  INCR      IF NOT, INCREMENT CLOCK
0080  C8          ADVANCE INY
0081  C0  00          CPY  #$00      Y HAS CARRIED IF ZERO
0083  D0  02          BNE  HOLD      IF NO CARRY GO TO HOLD
0085  E6  08          INC  INDHI     IF CARRY INCREMENT INDHI (ADVANCE PAGE)
0087  A9  00      HOLD    LDA  #$00
0089  8D  42  17        STA  PBD(KIM)  DISABLES DISPLAY IN HOLD MODE
008C  A9  20      LOP     LDA  #$20      MASK TO TEST PBD 5
008E  2C  02  17        BIT  PBD       TEST PBD 5
0091  F0  F9          BEQ  LOP       IF LOW, HOLD IN LOOP
0093  B1  07          LDA  IND,Y     IF NOT, GET NEXT OUTPUT DATA
0095  8D  00  17        STA  PAD       UPDATE OUTPUT
0098  84  09          STY  YBUF      SAVE Y
009A  A2  03      INCR    LDX  #$03      THESE STEPS ADVANCE CLOCK 1 SECOND
009C  A9  00          LDA  #$00
009E  F8              SED
009F  38          INCR1   SEC
00A0  75  00          ADC  HR-1, X   ADD 1 TO SEC (FIRST LOOP) MIN (2ND) HR (3RD)
00A2  95  00          STA  HR-1, X   PUT BACK IN PROPER CLOCK LOCATION
00A4  75  03          ADC  HRL-1, X  ADD SECL (1ST LP) MINL (2ND) HRL (3RD)
00A6  90  05          BCC  INT4      IF NO CARRY, DONE INCR CLOCK, LEAVE LOOP
00A8  95  00          STA  HR-1, X   IF CARRY, SEC=0 (1ST) MIN=0 (2ND)
00AA  CA              DEX
00AB  D0  F2          BNE  INCR1     CLOCK INCREMENT LOOP AGAIN
00AD  A9  10      INT4    LDA  #$10      RELOAD 4 COUNTER FOR INTERRUPTS
00AF  85  00          STA  CNT
00B1  A2  FF      INT6    LDX  #$FF      RESET STACK POINTER, OTHERWISE WOULD REQUIRE
00B3  9A              TXS            RTI INSTRUCTION
00B4  D8              CLD            CLEAR DECIMAL MODE
00B5  58              CLI            CLR. INTERRUPT MASK
00B6  A5  01      DPLY    LDA  HR        START DISPLAY ROUTINE
00B8  85  FB          STA  FB        MONITOR BUFFER FOR LEFT TWO DIGITS
00BA  A5  02          LDA  MIN
00BC  85  FA          STA  FA        MIDDLE TWO DIGITS
00BE  A5  03          LDA  SEC
00C0  85  F9          STA  F9        RIGHT TWO DIGITS
00C2  20  1F  1F        JSR  SCANDS    MONITOR SUBROUTINE, SCAN DISPLAY
00C5  A9  10          LDA  #$10      MASK TO TEST PB4
00C7  2C  02  17        BIT  PBD       BIT TEST PB4
00CA  F0  03          BEQ  STOP      IF PB4 LO, GO TO ERROR STOP
00CC  4C  B6  00        JMP  DPLY      RETURN TO DISPLAY LOOP UNTIL INTERRUPTED
00CF  A9  00      STOP    LDA  #$00
00D1  8D  00  17        STA  PAD       DISABLE OUTPUT
00D4  4C  29  19        JMP  MONITOR   TAPE LOAD ROUTINE, ERROR EXIT, DISP. FFFFXX
```

Sequencer Program Locations Used

Address	Label	Comment
0000	CNT	FOUR COUNTER, COUNTS QUARTER SECONDS
0001	HR	CLOCK CURRENT TIME IN HOURS
0002	MIN	CLOCK CURRENT TIME IN MINUTES
0003	SEC	CLOCK CURRENT TIME IN SECONDS
0004	HRL	HOUR LIMIT; 100+LIMIT CAUSES CARRY
0005	MINL	MINUTE LIMIT INITIALIZED TO 40
0006	SECL	SECOND LIMIT, INITIALIZED TO 40
0007	INDLO	LO BYTE BASE ADDRESS FOR IND,Y ADDRESSING
0008	INDHI	HI BYTE BASE ADDRESS FOR IND,Y ADDRESSING
0009	YBUF	SAVE Y LOCATION