1. Sam
2. Helen (File)

143.

NASA TECHNICAL MEMORANDUM

NASA TM X-2500

NASA TM X-2500

INEXPENSIVE PROGRAMMABLE CLOCK FOR A 12-BIT COMPUTER

by James E. Vrancik

Lewis Research Center
Cleveland, Ohio 44135

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION • WASHINGTON, D. C. • FEBRUARY 1972

ERRATA

NASA Technical Memorandum X-2500

INEXPENSIVE PROGRAMMABLE CLOCK

FOR A 12-BIT COMPUTER

by James E. Vrancik

February 1972

The attached page 13 was inadvertently omitted in printing and should be inserted in the report.

Issued August 1973

1. Report No. NASA TM X-2500	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle INEXPENSIVE PROGRAMMABLE CLOCK FOR A 12-BIT COMPUTER		5. Report Date February 1972 6. Performing Organization Code
7. Author(s) James E. Vrancik		8. Performing Organization Report No. E-6589
9. Performing Organization Name and Address Lewis Research Center National Aeronautics and Space Administration Cleveland, Ohio 44135		10. Work Unit No. 112-27 11. Contract or Grant No.
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D. C. 20546		13. Type of Report and Period Covered Technical Memorandum 14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

An inexpensive programmable clock was built for a Digital Equipment Corporation (DEC) PDP-12. The instruction list includes skip on flag; clear the flag, clear the clock, and stop the clock; and preset the counter with the contents of the accumulator and start the clock. The clock counts at a rate determined by an external oscillator and causes an interrupt and sets a flag when a 12-bit overflow occurs. An overflow can occur after 1 to 4096 counts. The clock can be built for a total parts cost of less than $100 including power supply and I/O connector. Slight modification can be made to permit its use on larger machines (16 bit, 24 bit, etc.) andlogic level shifting can be made to make it compatible with any computer.

17. Key Words (Suggested by Author(s)) Digital clock Computer clock Counter	18. Distribution Statement Unclassified - unlimited

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 14	22. Price* $3.00

* For sale by the National Technical Information Service, Springfield, Virginia 22151

INEXPENSIVE PROGRAMMABLE CLOCK FOR A 12-BIT COMPUTER

by James E. Vrancik

Lewis Research Center

SUMMARY

An inexpensive programmable clock was built for a Digital Equipment Corporation (DEC) PDP-12. The instruction list includes skip on flag; clear the flag, clear the clock, and stop the clock; and preset the counter with the contents of the accumulator and start the clock. The clock counts at a rate determined by an external oscillator and causes an interrupt and sets a flag when a 12-bit overflow occurs. An overflow can occur after 1 to 4096 counts.

The clock can be built for a total parts cost of less than $100 including power supply and I/O connector. Slight modification can be made to permit its use on larger machines (16 bit, 24 bit, etc.) and logic level shifting can be made to make it compatible with any computer.

INTRODUCTION

A computer alone is a useless tool. It has the ability to solve many problems but these problems must be described to it and it must be told how to solve them. Thus, the engineer must be able to communicate with the computer. Likewise, any computer that solves a problem and then keeps the answer to itself is useless. It too must have a way of communicating with the engineer. Communication is accomplished by devices such as teletypes, line printers, tape readers, etc. To the computer, each device must have a name and often two names to distinguish the direction of data flow. Along with a name, it must have a means of transmitting data or information. This is accomplished by a computer interface.

This report describes a device called a programmable clock and its computer interface to a DEC minicomputer, the PDP-12. The theory presented in this report is perfectly general and the clock can easily be interfaced to a different computer with only slight, it any, modification.

There are commercially available devices that can do just what this programmable clock does, and more, but they are relatively expensive. The purpose of this work is to overcome the cost barrier and develop a very inexpensive interval timer that can be programmed by the computer. The design of the clock itself is relatively straightforward, but the design of the interface between the clock and the computer is more complicated and is considered in detail in this report.

COMMUNICATION TECHNIQUES

There are two popular fundamentally different techniques used to communicate with a computer. The first mode can be called the wait mode. It involves giving a device a command and then waiting until the command has been executed and the results are available. For instance, if the computer were programmed to take a voltage measurement, it could issue a command to the voltmeter to begin taking a reading. The computer would then enter a mode that would continuously monitor a "done flag" (flip-flop), which would be set by the voltmeter when it had completed its analog to digital conversion. When the done flag is set, the data are ready and the computer would then read the voltmeter and clear the flag. This straightforward technique for servicing devices makes the software very simple. The problem is most input-output (I/O) devices are extremely slow compared to the computer speed, and thus a lot of computer time is wasted.

The second mode of communication is the interrupt mode. In this mode the computer again issues a command but does not wait for execution. Instead, it may continue to execute other instructions. When the device is done, it calls the computer by setting the done flag, which grounds the interrupt bus. When the computer senses that the interrupt bus is grounded, it realizes that a device is trying to get service, and it goes through a software routine to determine which device is calling and how to service it. In the example of the voltmeter, the computer would issue a command to take a reading and then continue doing other jobs. When the voltmeter finished its conversion, it would set its done flag and ground the interrupt bus. The computer would realize it was being called. It would remember what it was doing and begin monitoring all the device done flags until it came to the voltmeter flag. It would realize the voltmeter was the one calling and service it and then return to its interrupted job. The software for this type of device servicing is much more complicated but well worth it for saving computer time. The interrupt mode capability can be turned on and off by software. The actual mechanics of the interrupt are discussed in the section on the clock programming.

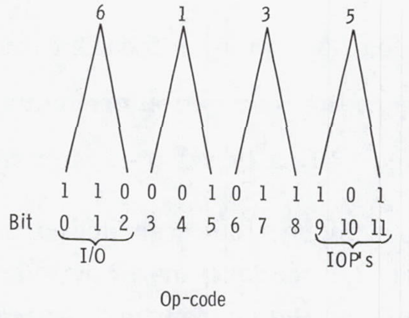
Op-Code

The word length for the PDP-12 is 12 bits. The easiest way of discussing the word is to divide it into four groups of 3-bits each, with each group an octal number. Thus, instead of 010111001011, we have $(2713)_8$. The first octal number is the op-code. Therefore, there are 8 op-codes. Op-codes 0 to 5 are instructions like add, jump, deposit, etc., op-code 6 is used exclusively for I/O instruction, and op-code 7 is for instructions that require no address like shift, skip, halt, complement the accumulator, etc. Here we will confine our discussion to op-code 6, the I/O op-code. The next two octal numbers are the name of the device being referenced. In this particular case, the programmable clock's name is 13_8 so an instruction 613X is an I/O instruction referencing the programmable clock. The last octal number (X) is used to generate pulses that permit the actual communication between the computer and the device.

These pulses are called IOP pulses or input-output pulses. If the pulses are actually used by a device, then they are renamed IOT pulses or input-output transfer pulses.

Device Selector

The logic required to use the IOP pulses to communicate is called a device selector. The two middle octal number (and their complements) of the memory buffer (12 signal lines) are brought out of the computer to each device. Six of the 12 lines are connected to the device selector giving the device a unique name (number) (see sketch).



If the signal lines $\overline{3}$, $\overline{4}$, 5, $\overline{6}$, 7, and 8 are AND'ed together and the output is used to enable gates, those gates will only be opened when device 13 is called.

Figure 1 shows a typical device selector. If the 6 input AND gate are assumed wired to select device 13, then the three succeeding AND gates will be enabled during this I/O instruction.

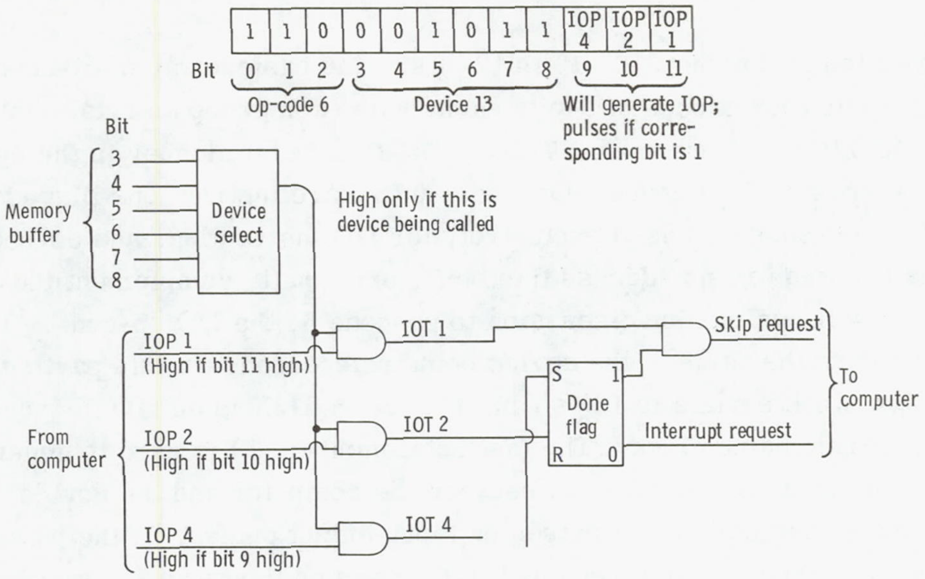

1 1 0 0 0 1 0 1 1 IOP 4 IOP 2 IOP 1
Bit 0 1 2 3 4 5 6 7 8 9 10 11
Op-code 6
Device 13
Will generate IOP; pulses if corresponding bit is 1
Bit
3 4 5 6 7 8
Memory buffer
Device select
High only if this is device being called
IOP 1 (High if bit 11 high)
IOP 2 (High if bit 10 high)
IOP 4 (High if bit 9 high)
From computer
IOT 1
IOT 2
IOT 4
S 1
Done flag
R 0
Skip request
Interrupt request
To computer

Figure 1. - Device selector.

Typically, but not necessarily the IOP 1 pulse is used to generate a skip pulse to set the skip flag if it was actually this device that was calling the computer. Also typically the IOP 2 pulse is used to clear the flag within the calling device. The IOP 4 pulse can be used for other purposes depending on the application. The device itself sets its own flag when it wants to be serviced.

The software that would be used to communicate would be something like this. If the software program was written in the wait mode and the computer was waiting for a response from the device, then the instruction would be

```
6131                  /skip on flag, or skip the next instruction if the flat is set

Jump up one           /jump to the instruction preceding this one

Service device 13     /device 13 finally set its done flag and can now be serviced
```

If the software is in the interrupt mode, the interrupt capability is turned on, and when device 13's flag is set, an interrupt request line is pulled to ground and the program is interrupted. The computer finishes the current instruction, automatically stores the address of the next instruction in location 0000, turns the interrupt off, and goes to location 0001 for its next instruction. The program at this point would resemble the following:

```
0000    0000          /will contain the address of the next instruction before
                        the interrupt
```

```
0001     Skip if flag 1        /skip next instruction if flag 1 set (6011) device 1 is
                                 some other device in the system
0002     Skip
0003     Jump device 1 service routine
0004     Skip if flag 13       /skip next instruction if flag 13 set (6131)
0005     Skip
0006     Jump device 13 service routine
0007     etc.
(Device 13 service routine)
0200     6131                  /clear the flag
         Service device 13
         Turn interrupt back on
         Jump back to program being executed before interrupt by using location in
           0000 as link
```

Programmable Clock

The general scheme of the programmable clock is shown in figure 2. An I/O preset pulse or an IOT 2 pulse clears the flag, resets the on-off switch, and clears the counter. The pulse is actually the output of a one-shot that is triggered by any convenient generator. In programming, the usual techniques would be to issue an IOT 2 (6132), load the accumulator with the two's complement of the number of pulses desired before an overflow occurs, and then issue an IOT 4. The IOT 4 opens the gate between the accumulator and the preset terminals of the counter for 500 nanoseconds. This presets the counter. The IOT 4 also turns on the on-off switch allowing the oscillator pulses to be counted by the counter. If in the wait mode, the computer continuously issues IOT 1's until an overflow occurs, and the flag is set allowing the IOT 1 to set the jump flag. If in the interrupt mode, when the flag becomes set the interrupt request line is grounded and an interrupt will occur. Interrogation of the flag by an IOT 1 will indicate the clock has caused the interrupt and may be reset if desired. In either case, an IOT 2 should be issued to clear the counter, reset the flag, and stop the counter. If the flag is left set, it will give a false indication of an interrupt if the interrupt is turned on or if an IOT 1 is issued.

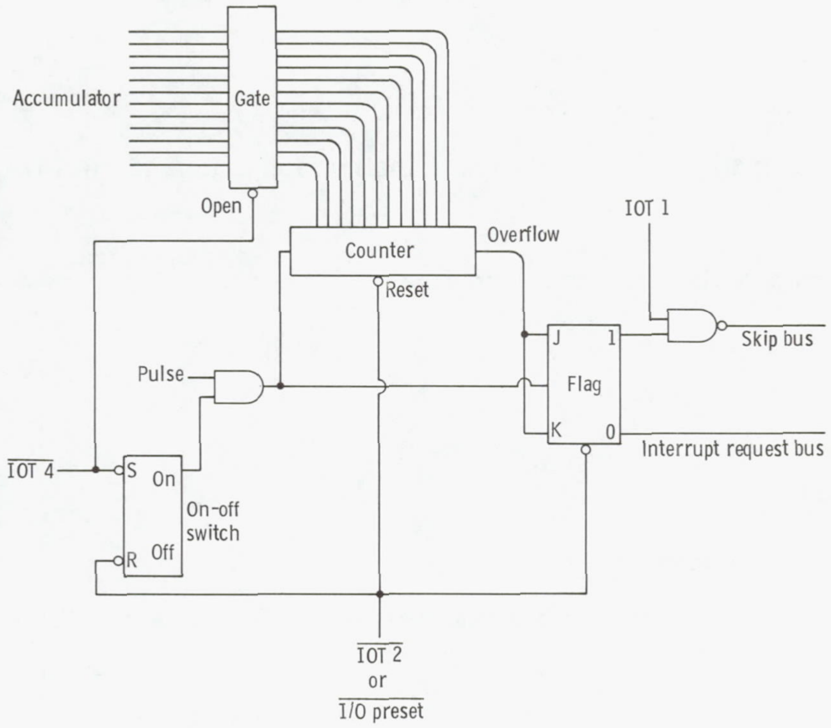


Figure 2. - Programmable clock block diagram.

DESIGN DETAILS

The engineering schematics used to describe the device are the following:

(1) Device selector and IOT generator

(2) Preset data gate

(3) Counter

(4) On-off switch, done flag, clock pulse generator

The numbers close to the terminals of the integrated circuit (IC) gate are the pin numbers and numbers such as 2 Ac and c indicate the second AND gate IC (actually NAND) and the third NAND within the IC, respectively. Actually, c is redundant since the pin numbers specify the exact gate.

Device Selector and IOT Generator

The complete device selects and IOT generator is shown in figure 3. The output of inverter 21_d is high only when the device 13_8 is called. This enables the three NAND gates 5Aa, 5Ab, and 5Ac. Thus, when an IOP 1 or IOP 4 pulse occurs, they become

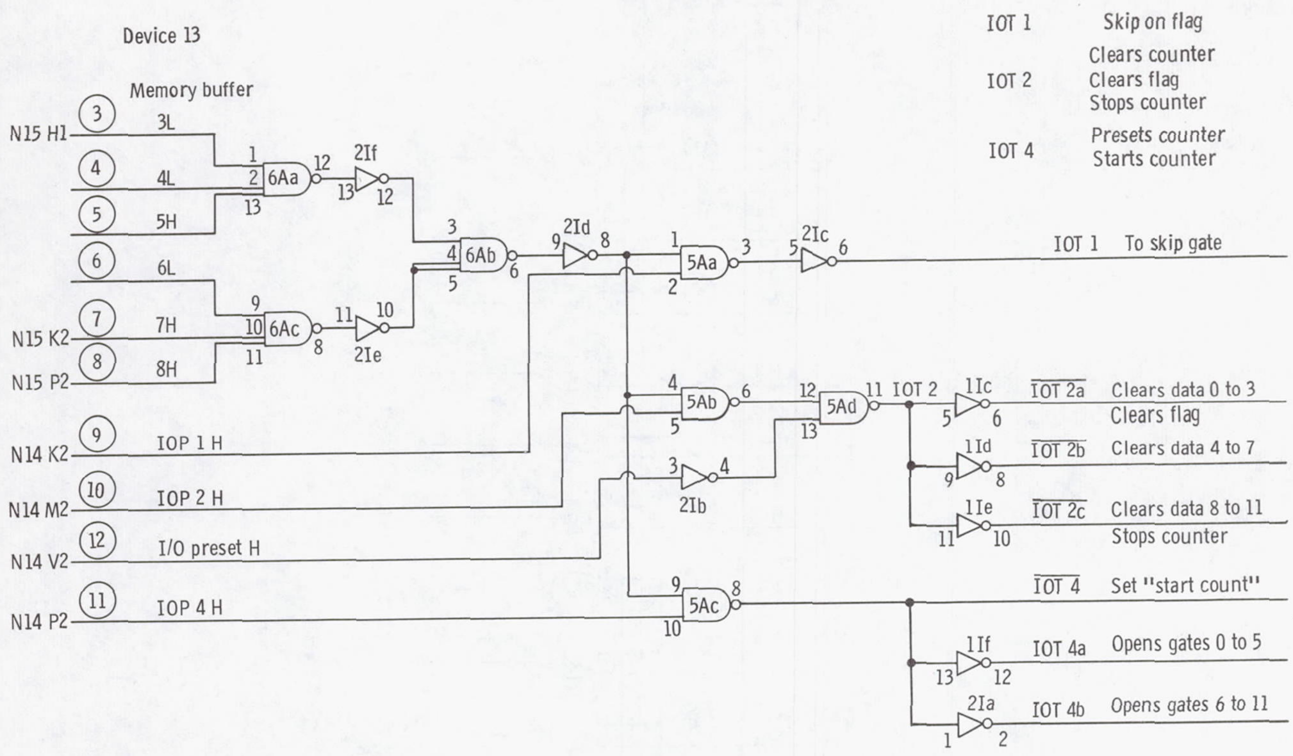


Figure 3. - Device selector and IOT generator.

IOT 1 and IOT 4 pulses which communicate back to the computer or to the device. An IOT 2 pulse occurs when either input (pin 12 or 13) to 5Ad goes low. Pin 13 goes low if an I/O preset pulse occurs. An I/O preset pulse is generated by a pushbutton on the computer console to initialize all I/O devices before starting. Pin 12 can go low if device 13 is being called and an IOP 2 occurs.

Additional inverters were used to accommodate fanout requirements for the IOT 2 and IOT 4 lines.

Preset Data Gate

The preset data gate is shown in figure 4. The IOT 4 pulse opens all the preset gates 1A, 2A, and 3A. If any input line is high, the output will be low and will preset the appropriate bit in the counter. The most significant bit in the computer is numbered bit 0. I chose to number the most significant bit in the counter number 11.

Numbers such as N14H2 are computer numbers and refer to the DEC-PDP-12. Row N, column 14 specifies the card location in the computer and pin B, side 1 specifies the pin location on the card.

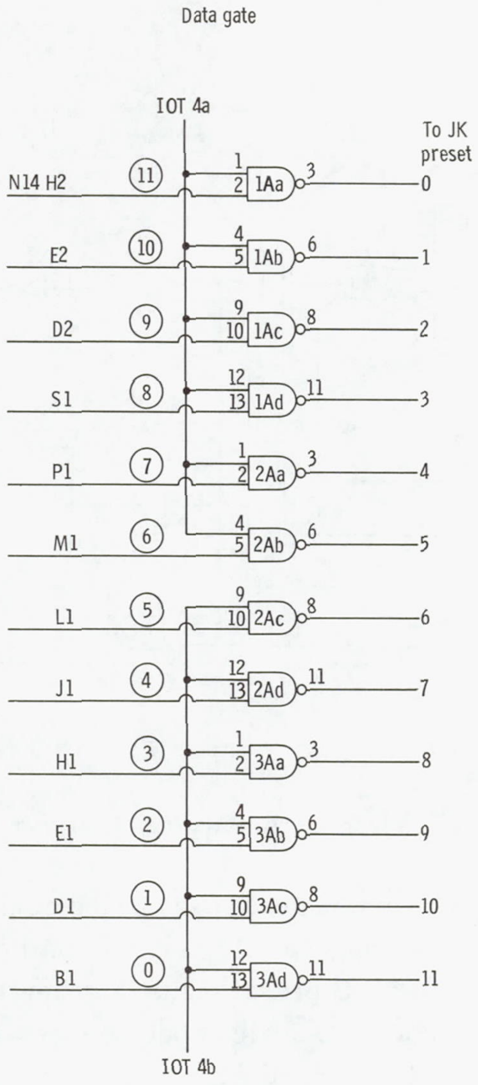


Figure 4. - Preset data gate block diagram.

Counter

The 12-bit counter is shown in figure 5. The counter is actually three completely separate four-bit counters. The first flip-flop (FF1) changes state each clock pulse. The second flip-flop changes state only when the first flip-flop is one. The third changes state only when the first and second flip-flops are one, and the fourth flip-flop changes state only when all three previous flip-flops are one.

Each of the three four-bit counters has a one's detector that changes state if all flip-flops within that counter are one. These one's detectors are used to enable the clock pulse to get through to the next four-bit counter. Thus, the second four-bit counter

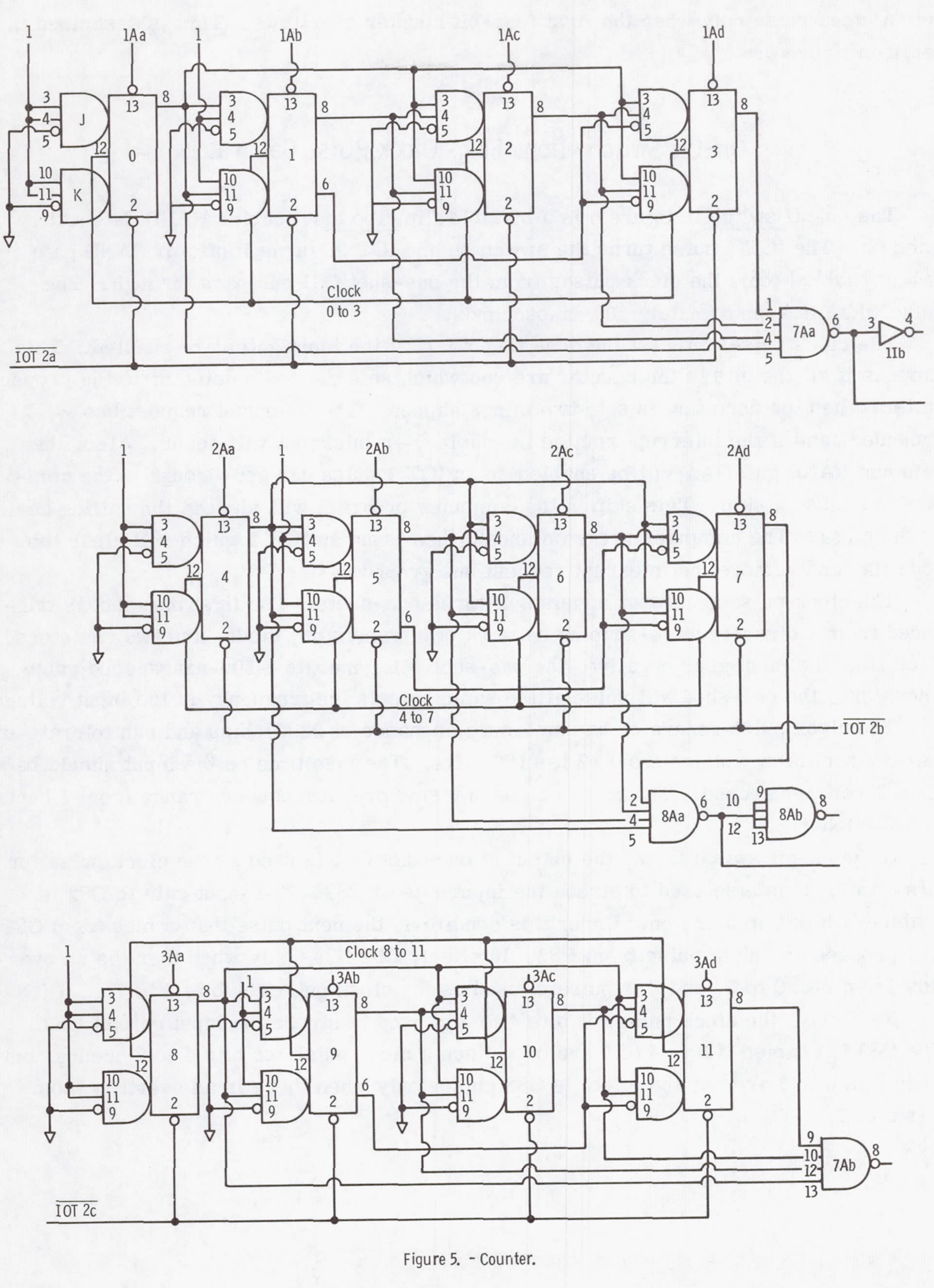


Figure 5. - Counter.

gets a clock pulse only when the first four-bit counter overflows. This is described in detail in figure 6.

On-Off Switch, Done Flag, Clock Pulse Generator

The on-off switch in figure 6 is a bistable flip-flop that enables NAND date 4Ab when on. The IOT 4 pulse turns the switch on and IOT 2c turns it off. If NAND gate 4Ab is enabled (on), the clock pulses from the one-shot OS1 can pass through. The pulse width is approximately 500 nanoseconds.

The clock pulses will set the done flag FF 12 if the input gates are enabled. This happens if all the bits in the counter are one which sets the one's detector to the proper state. When the done flag is set, two things happen. The interrupt request bus is grounded, and if the interrupt routine is enabled, an interrupt will occur. Also, the skip bus NAND gate 4Aa will be enabled so an IOT 1 pulse can get through to the computer to cause a skip. This skip in the computer program will identify the calling device as the clock. The computer program should then issue an IOP 2 which will clear the done flag and remove the interrupt request bus ground.

The clock pulse generator consists of three one-shots. The first one-shot is triggered from a discrete wave-shaping network of a transistor, diode, and two resistors. Each time the input goes negative, the one-shot will generate a 500-nanosecond pulse. Therefore, the one-shot will pulse at the same rate as the frequency of the input voltage.

The clock pulse generator has an input impedance of 22 kilohms and can tolerate and use any oscillator voltage from ± 2 to ± 100 volts. The rise time on the input should be at least 1 volt per second. The clock has been tested over a frequency range from 1 hertz to 500 kilohertz.

If the on-off switch is on, the output of one-shot OS1 is used as the clock pulse for bits 0 to 3. It is also used to strobe the input gate of OS2. The input gate to OS2 is enabled if bits 0 to 3 are on. Under this condition, the next pulse that comes from OS1 will produce an output pulse from OS2. In effect, OS2 fires only when there is an overflow from bits 0 to 3, and this pulse is used as the clock pulse for bits 4 to 7.

Similarly, the clock pulse for bits 4 to 7 is used to strobe the input gate of OS3. The OS3 is enabled if bits 4 to 7 are on. Since a clock pulse for bits 4 to 7 occurs, only when bits 0 to 3 are on, the effect is OS3 pulses only when there is an overflow from bits 0 to 7.

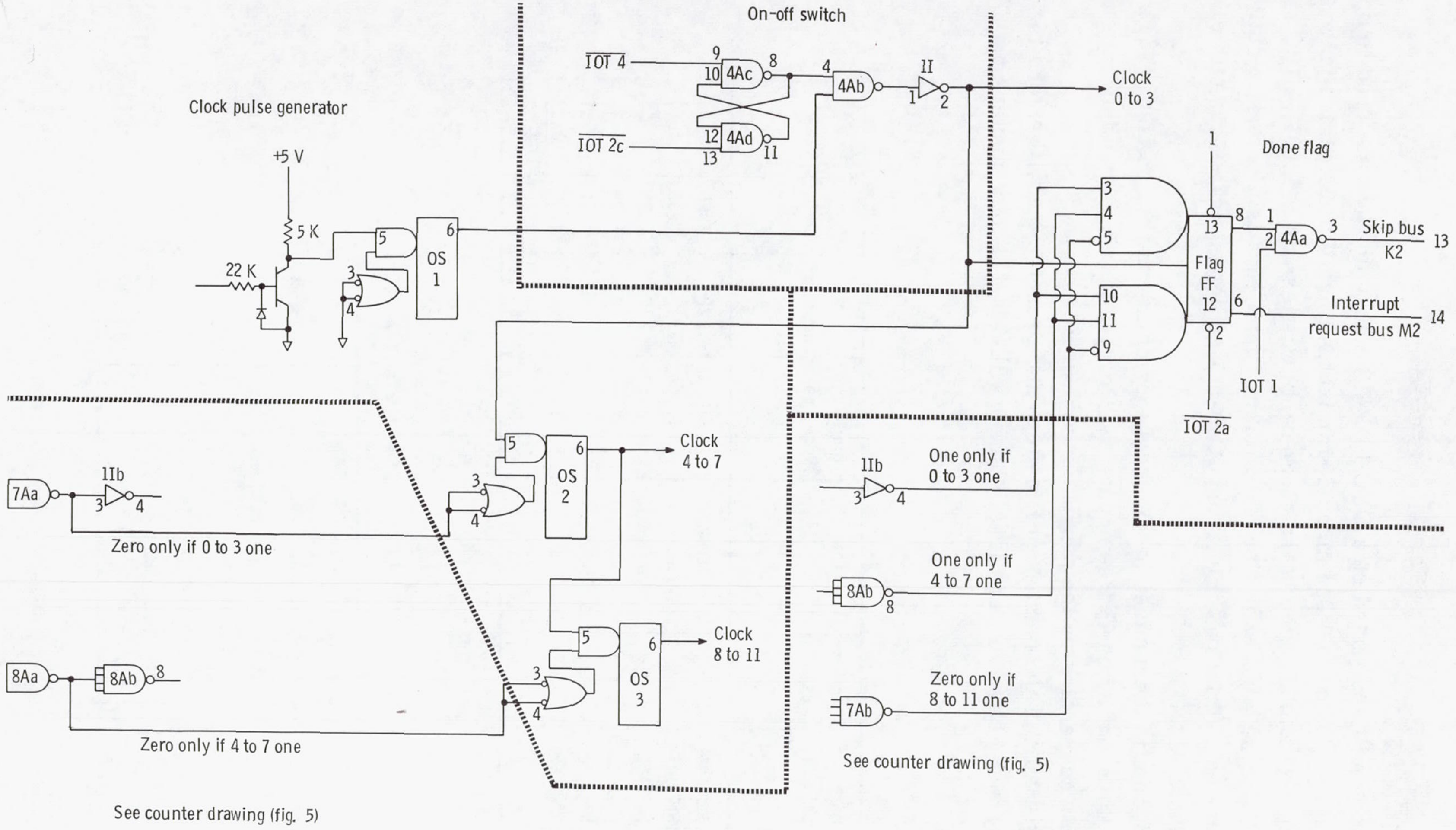


Figure 6. - On-off switch, done flag, clock pulse generator.

RESULTS

This clock was built and tested on a DEC PDP-12 computer and was found to operate correctly as designed. A parts list is included in tables I and II along with a price list of all components. The total price of the clock parts was less than $100. All IC logic elements are TTL. The clock was designed with minimum cost as a prime objective. The fact that every gate (total of 55) in every IC was used demonstrates the gate efficiency of the design.

The counting rate is determined by the frequency of an external oscillator. Each cycle increases the count by one. The counter will overflow and give an interrupt after one to 4096 counts, depending on the programmed preset.

A simple extension of the number of flip-flops and preset gates will extend the number of bits to any size computer, and level shifting will adapt it to computers not TTL compatible.

TABLE I. - INTEGRATED CIRCUIT COSTS

Integrated circuit	Number used	Name	Number of gates per integrated circuit	Total number of gates	Cost each	Total cost
SN7400	5	2 input NAND	4	20	$1.86	$ 9.30
SN7404	2	Inverter	6	12	1.55	3.10
SN7410	1	2 input NAND	3	3	1.24	1.24
SN7420	2	4 input NAND	2	4	1.54	3.16
SN7470	13	JK flip-flop	1	13	1.97	25.61
SN74121	3	One-shot	1	3	3.46	10.38
				55		$52.79

TABLE II. - TOTAL COSTS

Integrated circuits	$52.79
Power supply	29.00
Digital Equipment Corporation plug in cards	12.00
Transistor, diode, resistors	1.00
Mounting board, wire, etc.	4.00
	$98.79

NASA-Langley, 1972 —— 8 E-6589

CONCLUDING REMARKS

An inexpensive programmable clock was designed, built, and tested on a DEC-PDP-12 minicomputer. The clock is capable of being programmed by the computer to give an interrupt after 1 to 4096 time increments. The time increment is dependent on the frequency of an external oscillator. The interface to the computer allows the computer to stop the clock, preset the clock time interval, start the clock, clear the done flag, and skip or flag. The total cost of the parts for the clock was less than $100.

Lewis Research Center,
National Aeronautics and Space Administration,
Cleveland, Ohio, October 29, 1971,
112-27.