The Television Pause Function

by

Michael R. Truog

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements of the Degree of

Bachelor of Science in Electrical Engineering and Engineering

at the Massachusetts Institute of Technology

May 1989

Signature of the Author

Michael R. Truog
Department of Electrical Engineering and Computer Science
May 16, 1989

Certified by

Walter Bender
Principle Research Scientist, Media Laboratory
Thesis Supervisor

Accepted by

Leonard A. Gould
Chairman, Department Committee on Undergraduate Theses

The Television Pause Function

by

Michael R. Truog

Submitted to the
Department of Electrical Engineering and Computer Science

May 16, 1989

In Partial Fulfillment of the requirements of the degree of
Bachelor of Science in Electrical Engineering and Engineering

ABSTRACT

A new pause function is designed and implemented which enables a television viewer to pause a live television program for a variable length of time, and then return at a later time and continue watching without missing any portion of the program. The system uses three VCR's controlled by digital circuitry to provide this feature.

Thesis Supervisor: Walter Bender
Title: Principle Research Scientist, Media Laboratory

This work was supported in part by I.B.M.

Table of Contents

Chapter 1. Introduction

This thesis describes a circuit that adds a new pause option to television. Often people are in a situation where they have just started watching a television program when some minor interruption, such as a phone call, takes them away from the viewing area for a short time. To continue their viewing, there are two options. Their first option is to push record when they leave, rewind the tape after the show is over, and watch from where they had left off. The second option is simply to skip the portion they missed. The new 'television pause' function I have designed is a third option which will enable the viewers to continue watching the program from where they left off to completion as soon as they return.

Although the television program continues to be broadcast, the 'television pause' feature leads the viewer to believe that the broadcast has been put on hold. Since the broadcast does not stop, some method was needed to store the video signal, so that it could be played back at any time. Before describing my circuit, some background information on video signals and their storage must be given.

Video signals are transmitted as electro-magnetic radiation from either an antenna or a satellite. They are received by viewers though a receiving antenna and are displayed as pictures on a television set (TV). The general design of the video signal is such that it is composed of discrete packets of information called frames. Each frame is 1/30th of a second in length and corresponds to a TV picture 1/30th of a second in duration. Within each frame, different frequencies and the amplitudes of these frequencies determine what the picture looks like. The frequencies present in video signals range from 30 Hz to 4.5M Hz.

There are two methods for storing these signals for later playback. One is digitizing the signals and storing the information on a disk in the form of bytes of video information. This method provides very easy access to video data. Using conventional means it is extremely costly and also difficult to do due to the high frequency at which the signals are broadcast. A much more simple method for storing video signals is to record the video using a video cassette recorder (VCR). The VCR stores the signal in its analog form on a magnetic tape, which can be read at any time later. The VCR method is much less expensive but is very awkward to use. (VCR functions and their inherent problems are discussed in Chapters 2 and 4.) To make this function realizable as a consumer product the cost of the function is very important, which makes the choice of methods simple.

Chapter 2. Definitions and Explanation of Terms

VCR stands for video cassette recorder. The brand and model used for this project was the JVC model HR-S5000U Super VHS recorder. The following VCR functions are used to implement the 'television pause' function:

Record -- This function causes the VCR to store the video signal on the tape. It is analogous to a computer writing to memory. The signal is stored in discrete packets, frames, similar to the way it is broadcast. Before each frame is recorded, a signal is written on the tape to mark the start of a new frame. On playback this signal informs the VCR that a new frame is ready to be read. This signal is referred to as a control pulse. One other note to make about the record function is that there is a delay between when record is pressed and the VCR starts recording. This delay was important to account for in the design of the 'television pause' feature. The delay results from the method used by the VCR to either record or play. In its rest state, the tape is removed from the recording/playback head. To write or read the video signal, the tape has to be wrapped around the recording head so that the recording head is in physical contact with the tape. The recording delay is the amount of time it takes to wrap the tape around the recording head. The delay is approximately three seconds.

Play -- This function reads the video information stored by the record command and causes the recorded signal to be displayed on a TV. The control pulses written by record are read by play and used to insure that each frame is played back accurately. The tape must be touching the recording/playback head to read the tape, so the same delay that is present in recording is also found on playback.

Pause -- This function suspends the playback of a recorded video signal. It is used only when the VCR is in 'play' mode.

Search Rewind -- This function is used to rewind the tape, while always reading the video and control pulse information. It is invoked by pressing rewind while the VCR is playing, causing the tape to be rewound without removing it from the record/playback head. It is used instead of straight rewind because tape position needs to be known by my circuit and reading and counting control pulses is the only way to figure tape movement.

Search Forward -- This function is the fast forward version of search rewind.

The Intel 8751 is an eight bit microprocessor. It has programmable ROM on chip as well as 128 bytes of RAM.

Chapter 3. Design Requirements

There are five requirements that my circuit had to fulfill. First, there should be only one button for the viewer to push to activate the television pause function. One button makes the function easier to use for a public that feels inundated with buttons and functions on entertainment equipment. Second, the user should not have to wait to watch the program after he returns. Upon returning, the only delay the user should have to wait for is only the play function delay. Third, the user should be able to watch the entire program from where he left off. No portion of the program should be omitted. Fourth, the user should not be able to tell the difference between the paused program and the original broadcast of the program other than the lessened quality caused by playing from tape. It is extremely important that the picture quality is not reduced by the use of the new pause function. The final requirement is that the user should be able to catch up to actual time by fast forwarding through commercials.

Chapter 4. VCR Deficiencies

To accomplish these goals, the software algorithm had to account for three short comings that are inherent in home video recorders. The first is that VCR functions do not react immediately. For instance, once the 'play' button is pushed, a delay of approximately three seconds takes place before any video is displayed on the screen. The software needed to plan for this delay by allowing set up times after asserting any of the VCR's functions.

Second, keeping an accurate record of position on the video tape is very difficult. The counter on most VCR displays is only accurate on a macroscopic scale. Using this counter, real position will change by a few seconds after a rewind or a stop function is used making it impossible to find a certain frame. My circuit must be able to find the tape position within a single frame accuracy, so another method was needed. This method was to count the frames from where recording began and thus the relative distance between any two frames would always be known. Counting frames is accomplished by counting control pulses, since a control pulse is recorded before each frame. Each VCR's tape position is extremely important in deciding when to turn on or off VCR functions, so whenever the tape is moving my circuit should know how far it has moved from the number of control pulses read. This prohibits any tape movement that does not read control pulses. The functions that do not read control pulses are rewind and fast forward. Instead of these functions, search rewind and search forward are used, since they read control pulses. The software keeps records of these relative positions by counting the control pulses and storing them as variables.

Third, the VCR's have very slow seek times. Finding a certain packet of information, such as a frame, can be long process. This is because the tape is a one dimensional storage device. Thus, data can only be searched for in two directions, forwards and backwards. Search forward and search rewind are the VCR functions that correspond to these directions. Having a second dimension to move makes reading much faster, since large portions of information can be skipped over quickly. A good example of a two dimensional storage medium is a computer disk. Not having this ability, my circuit had to plan for search times that were on the same order of magnitude as the write and read times. Normal search speeds are seven times play speed in standard play and seventeen times play speed in extended play mode.

Chapter 5. Design Overview

The circuit must perform the 'television pause' function defined by the design requirements but at the same time it needs to work around the VCR deficiencies. The method I used to implement the television pause function is fairly simple. One VCR is recording at all times after the 'television pause' button is pushed. This makes sure the viewer is able to see the complete program. Once the viewer returns, one VCR is playing at all times, so there are no noticeable breaks on playback. Finally, when switching between VCR playback outputs, the VCR next in line to play must be ready to play the beginning of its recorded segment before the VCR that is playing is finished. Three VCR's are needed to allow these three states to occur simultaneously and my circuit has to control them. The system diagram is shown in figure 5.1. This method and its implementation are described in detail in the next two sections.

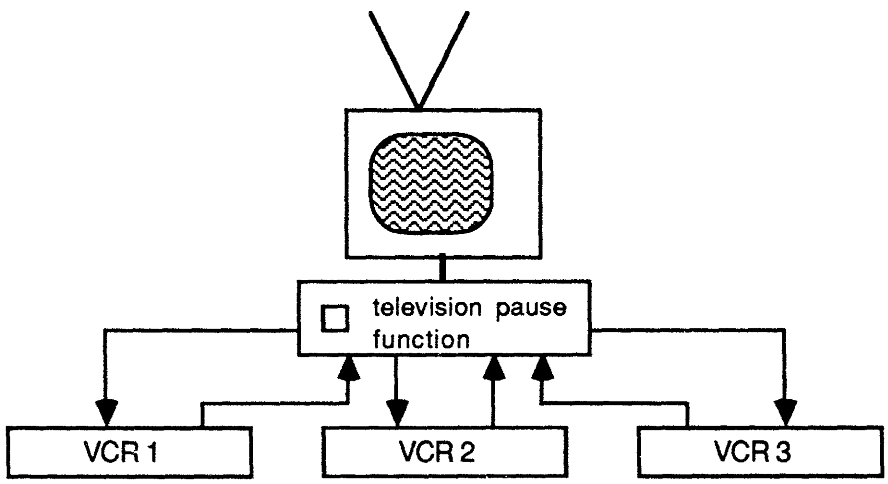


Figure 5.1 System Diagram

Chapter 6. Software Design

6.1 Goals of the Software

In designing the software, I had the following goals: the state of each VCR should be known at all times (this includes knowing the function each

VCR is performing and each tape position at any moment); signals sent by the 8751 software should be used by the hardware to cause the VCR's to perform certain functions, and to combine the first two goals into a state machine that produces the new pause function and would meet the five requirements listed in section III.

6.2 Software Organization

The software is written in Intel 8051 assembler code. The code is organized so that all the variables needed are placed in internal RAM. The listing of variables, what they are for, and their RAM locations is on the first page of the assembly code listing in Appendix A. The software has three sections : initialization, the pause, and the playback loop.

6.2.1 Initialization

The initialization section is entered upon power up or an external reset. The microprocessor reacts by resetting its program counter to location 0. At this location, there is a jump command to the start of the program. Upon power up or reset, the circuit should not be interacting with any of the VCR's. The software clears all control registers at this point by writing four initialization bytes.

6.2.2 Pause

After initializing the system, the software waits for the user to push the pause button. Once pushed, the software reacts by switching control of the

VCR functions from the VCR control panel to my circuit. It then sends the record signal for all the VCR's to the control registers. The VCR's begin recording the program and the software counts the length of the pause. However, the first VCR does not continue recording until the user returns. Instead only 44 seconds of the program are recorded. By recording only a short segment, the tape can be rewound and ready to play before the user returns. The requirement that the user should not wait is fulfilled here, but is subject to the constraint he is gone for at least one minute.

Three steps must be taken before rewinding the first VCR. First, to simplify the switching between outputs of the VCR's, the recorded material on the VCR's must contain an overlap. This overlap is taken care of by recording on the second VCR during the first 44 seconds. However, VCR 2's counter is set so that the code only thinks that it has recorded for six seconds. This way VCR 2 will not be rewound passed the switch point. The second step is to stop VCR 1. Once the 'stop' signal is sent, the software waits for two seconds to insure that the VCR will have time to react to the 'stop' signal before the software sends another signal. The third step is to send the 'play' signal so that when the tape needs to be rewound, it will be in search rewind mode instead of rewind mode. The difference may seem subtle but it is very important. The first goal of the software, knowing each VCR's state, is dependent upon receiving accurate control pulse inputs. The only way to obtain accurate information from the video tape is to have the video head touching the tape. In play and record modes, the head is always in contact with the tape. So whenever the tape is being moved, the VCR has to be in either play, record, search forward, or search rewind. Figure 6.1 is a diagram of these steps.

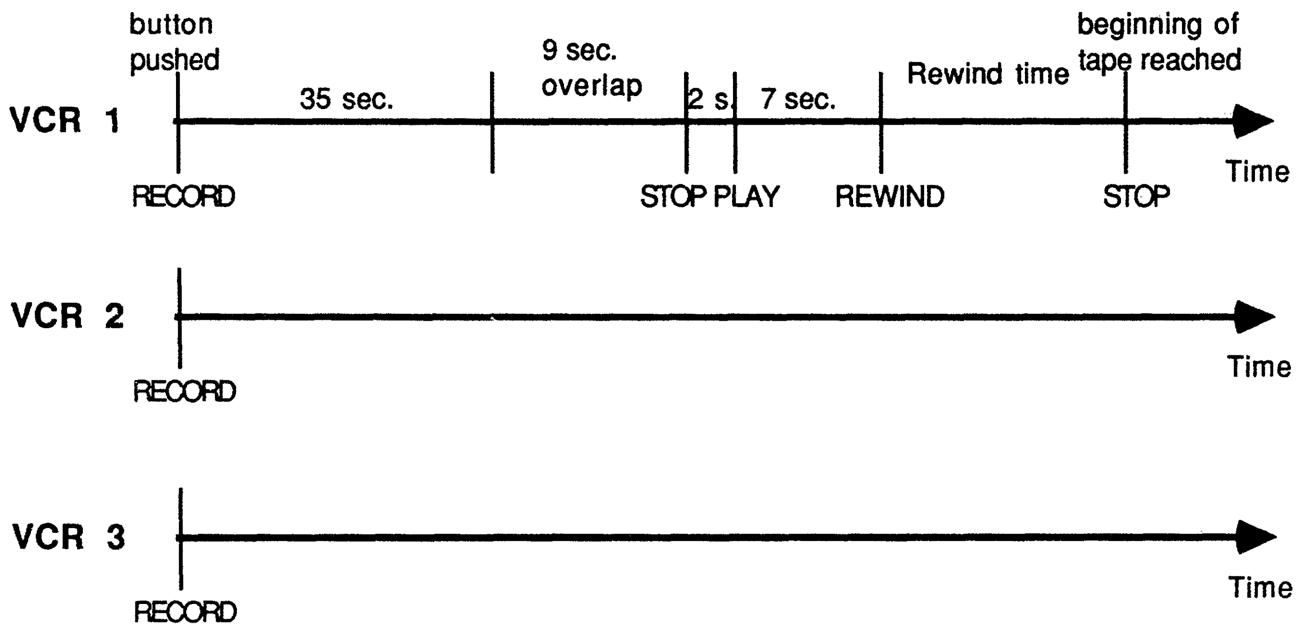
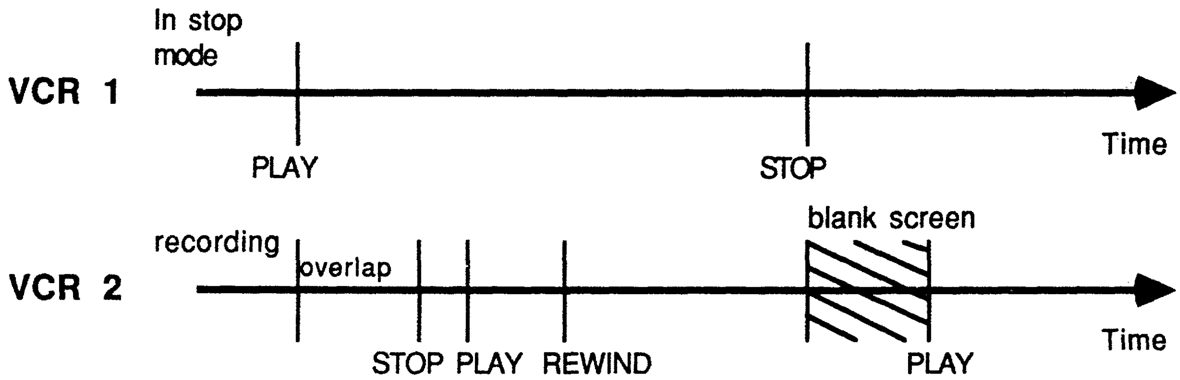
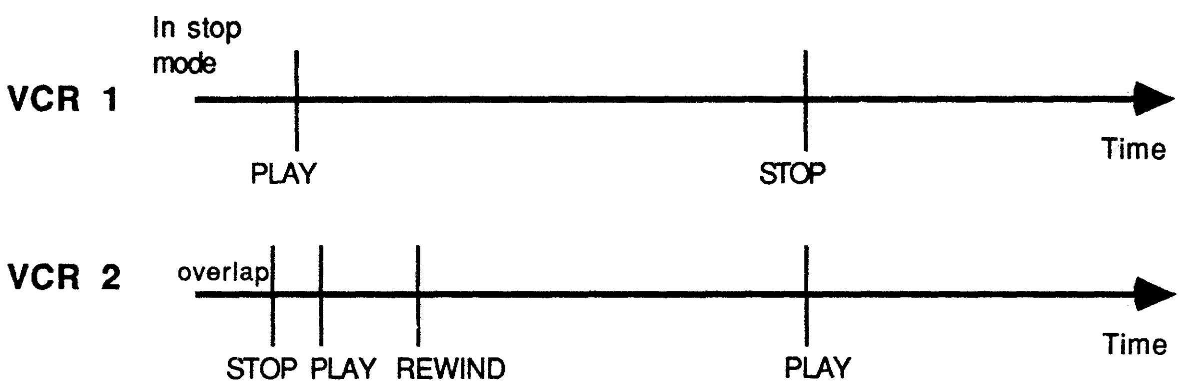


Figure 6.1 Initial Record and Rewind Sequence

After another delay of seven seconds, VCR 1 is rewound. Each frame that is encountered during the rewind is counted and compared to the number of frames recorded. When they are equal, the beginning of the recording has been found and the VCR is sent the signal to stop. The time it



(A) Too much material was recorded to rewind in time resulting in a break in the playback



(B) Recording limit was not exceeded resulting in a smooth playback

Figure 6.2 Switching Timing Problem (A) and Solution (B)

takes to rewind is stored for future use. Figure 6.1 also shows these actions. At this point, the user could return and immediately view the portion he missed.

By setting the minimum pause time at only one minute, not only have I limited the amount of material that can be recorded on the first VCR but also the second VCR. Since the first VCR only has 44 seconds of video to play when the user returns, the second VCR only has 44 seconds to rewind. This problem and its solution are illustrated in figures 6.2a and b. The amount of material that can be recorded on VCR 2 is dependent upon the rewind speed. The rewind speed is found by dividing the rewind length by the rewind time found when VCR 1 was rewound. The limit for recording on VCR 2 in seconds is:

$$(44 - 8) * \text{rewind time}$$

where the eight corresponds to the set-up time necessary to find the beginning of each recorded section.

For very short pause times, this limit will not be reached (See figure 6.3a). For these cases, the first VCR is told to play while the third VCR records the nine second overlap of VCR 2. After the overlap, the second VCR is stopped and rewound using the same search rewind procedure that was used for VCR 1.

When the pause times exceed the recording limit for VCR 2, a different path is followed (figure 6.3b). Once this limit is reached, the overlap time for the third VCR is set to be six seconds. VCR 2 is then stopped but not rewound. Just as there was a minimum pause time, there is also a maximum

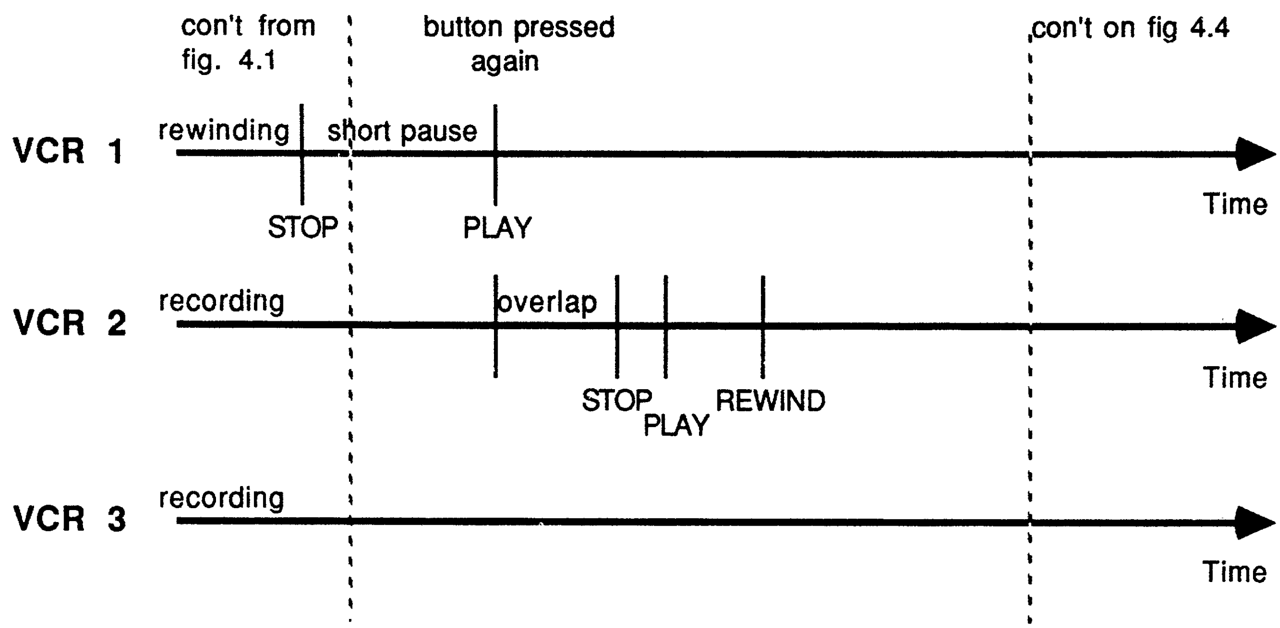


Figure 6.3a User returns before VCR 2's recording limit

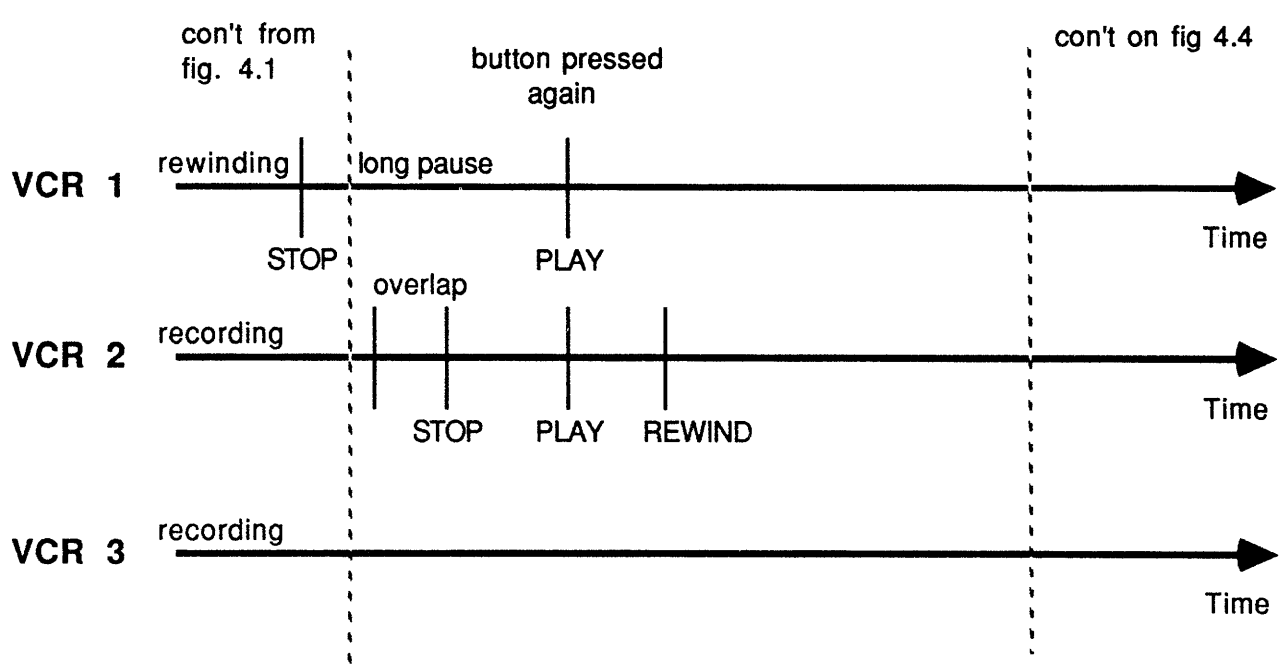


Figure 6.3b User returns after VCR 2's recording limit

pause time. This constraint results from using sixteen bit counters counting thirty frames a second. 2^{16} is reached in 36 minutes.

When the user finally returns, the first VCR is told to play and then the second goes into search rewind. At this point, both paths will meet again.

Once the beginning of VCR 2's recorded segment is found, the signal 'play' is sent. This is unlike the first VCR because the second VCR needs to find the exact frame where switching will take place. Finding this frame and switching VCR's is a five step process shown in figure 6.4. The VCR is

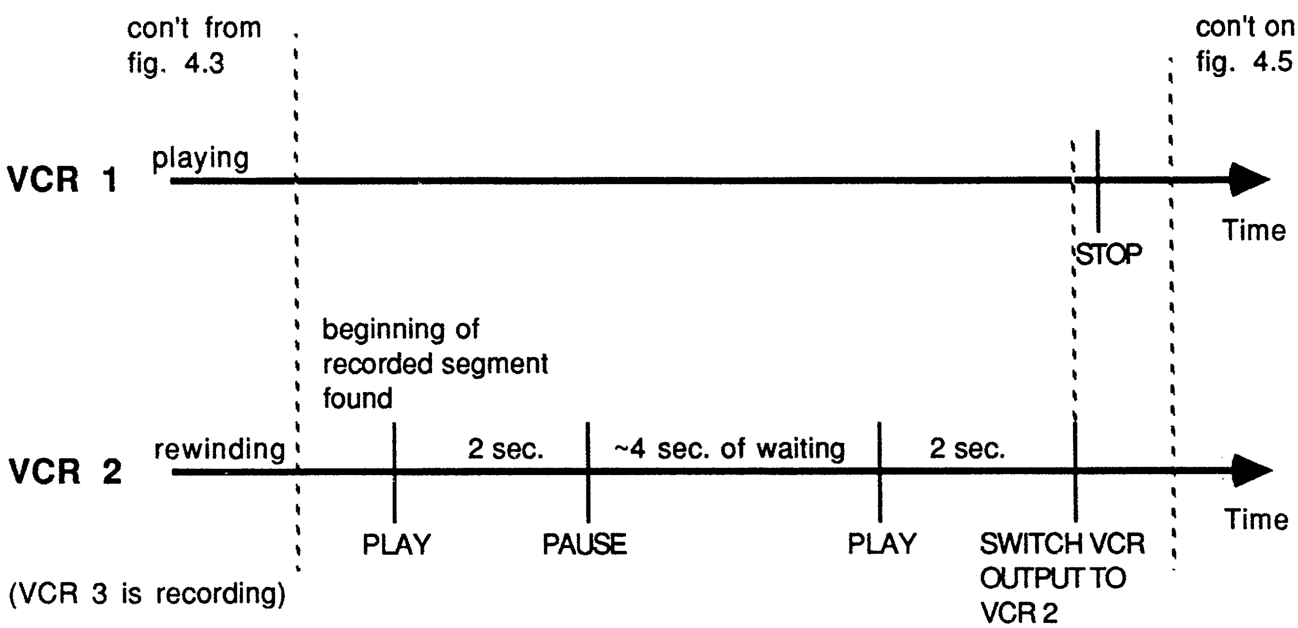


Figure 4.4 Sequence to switch outputs of VCR's

allowed to play to the frame two seconds before the switching point, where it is paused (the normal pause function). While VCR 2 is in pause mode, the software waits for the segment playing on the first VCR to get near its end. When the frame on VCR 1 is eight frames behind VCR 2's position, VCR 2 continues playing. After two seconds, VCR 1 will have caught up to VCR 2 and the output to the TV is switched from VCR 1 to VCR 2. These switches are fairly accurate so as to meet the fourth requirement of the circuit -- that the user does not realize this is not a live broadcast. After the switch has been made, VCR 1 is stopped.

6.2.3 The Playback Loop

The software then enters the third section, the loop. The loop is a series of eight steps that are repeated until the stop button on my circuit is pressed (See figure 6.5). The process consists of recording an overlap, search rewinding, finding the frame upon which switching will take place, then switching the VCR outputs, and finally stopping the VCR that finished playing. This process is very similar to that used for VCR 2. After each pass through the loop, the commands are switched so that each VCR cycles through this process. In figure 6.5, the commands given for the second pass is given in parentheses beneath the first pass commands.

During the loop, the user is allowed to fast forward through parts of the recorded video, thus fulfilling the fifth requirement. To fast forward, the fast forward button on my circuit is pushed to start the fast forward mode, and it is pushed again to end this mode. However, there are restrictions. If any VCR is rewinding or is within eighteen seconds of rewinding, the fast forward function is not possible, since breaks would develop when the VCR's are switched.

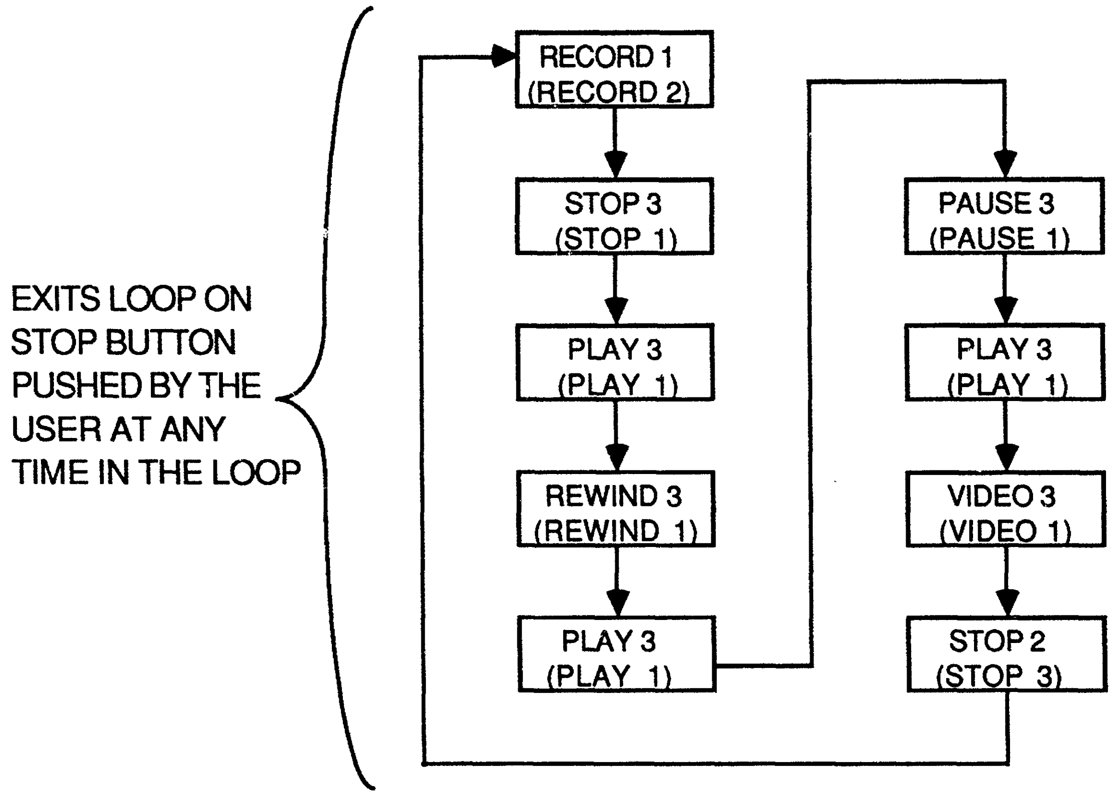


Figure 6.5 Flowchart of the loop

Chapter 7. Hardware Design

7.1 Hardware Requirements

The hardware has two duties : implementing the control signals generated by the software to work the VCR and receiving and configuring the inputs to a form the software can use. To implement the control signals, the circuit was connected to the internal circuits of the VCR. For the VCR to react to my circuit's control, the signals that are used by the VCR to produce certain functions had to be duplicated. The duplication had two requirements. The first is that my circuit needed to output the correct voltages. Thus to produce a 'stop' command, a signal of zero volts needed to be sent to the VCR. The voltage requirements are listed in table 7.1. The second requirement for

Table 7.1
Voltages That Produce VCR Functions

FUNCTION	VOLTAGE
Stop	0.00 V
Pause	0.50 V
Play	1.00 V
Rewind	1.50 V
Fast Forward	2.30 V
Record	3.50 V
No Function	5.10 V

duplication was that the voltages needed to be stable for a at least a quarter of a second. The reason for this is to insure that the VCR will have enough time to see the command. This means that a voltage cannot be generated and replaced within a quarter of a second. The second duty of the hardware also has two requirements. These are that the software should receive all inputs and that the software should only have to look at each input once. When these two requirements are met, the amount of code necessary is greatly reduced.

7.2 Hardware Organization

7.2.1 The Controller

The hardware is implemented in four sections that are shown in a block diagram in figure 7.1 (Also see appendix B for the schematics). The first

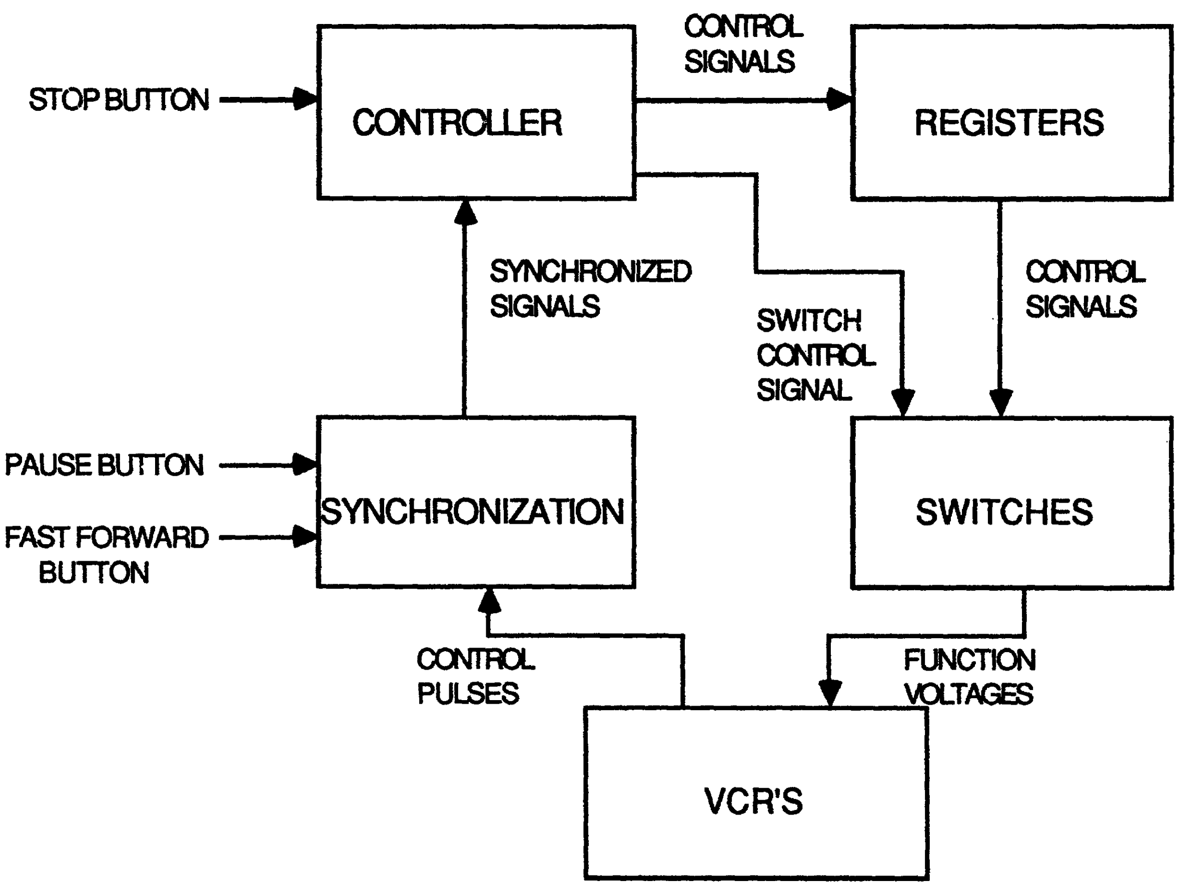


Figure 5.1 Hardware Block Diagram

section is the controller. This section is comprised of the Intel 8751 microprocessor and a 16R8 pal. Its function is to keep track of the state of each of the VCR's and output certain signals based on these states. The state of each VCR is determined by the software encoded in the 8751. The software also produces the output signals. It does this by writing data to its output ports. The pal produces the load signals used to store these signals in the second section. The pal also determines whether my circuit should have control of the VCR's functions or not though its output 'cntl on.'

7.2.2 The Registers

The second section is made up of four registers. The sole purpose of these registers is to hold the control signals sent from the microprocessor. As explained above, any inputs to the VCR's need to be stable for a least a quarter second. It is impossible for the microprocessor to meet this requirement by itself since the written data is only valid for a very short time (approximately one microsecond). The registers can hold the data for a much longer period and thus provide the stability that is necessary.

7.2.3 The Switches

The third section uses the output of the registers to produce a signal the VCR's can understand and will react to. The signals that need to be produced are the voltages listed in Table 7.1. To produce these voltages, a set of resistors in series with switches placed at each of the nodes are used. Three resistor-switch groups were used, each one corresponding to one VCR. The resistors are connected to pins within the VCR and each switch has its 'on' input connected to ground. The resistors form one half of a voltage divider, with the other half inside the VCR. By turning on any of the switches and thus connecting a node to ground, the voltage divider is changed as is the voltage sent to the VCR. Each switch produces a voltage corresponding to a VCR function such as 'play' or 'rewind', so turning on the 'record' switch is the same as pushing the 'record' button on the VCR control panel.

This section must also be able to switch control from the VCR control panel to my circuit, and vice versa. My circuit should be connected to the VCR circuitry when the pause function is in use, but the VCR should maintain control of its functions when the pause function is not being used. To be able to electronically choose between control sources, switches are connected to both the VCR and my circuit and are controlled by the 'cntl on' output of the pal. This arrangement is shown in figure 7.2.

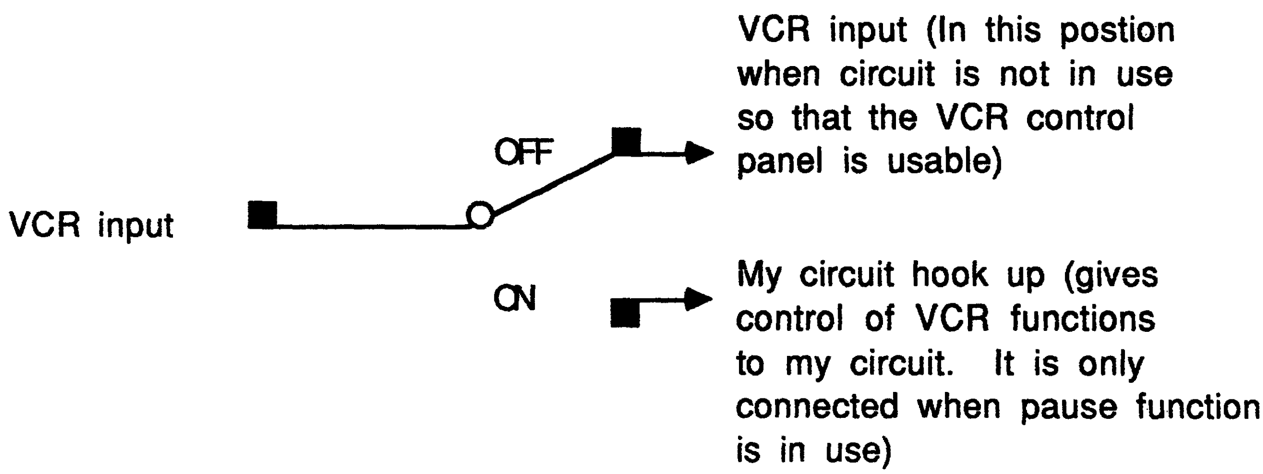


Figure 5.2 VCR Function Control Switch

7.2.4 The Synchronization Section

Aside from outputting signals, my circuit is constantly receiving signals both from the VCR and the user. The VCR sends control signals to my circuit so that position information can be determined. The user can send three signals to my circuit. These signals are 'television pause', 'fast forward', and 'stop.' The duration and occurrence of these signals can vary greatly. The software is written so that each signal is only seen once, so the final section of the hardware is used to synchronize and configure the inputs to a form that can be used by the software. The section uses four pals to accomplish the synchronization. The pals use a scheme where the input is held until the software wants to use it. The pals wait until the input signal is unasserted before they look at the signal again, thus eliminating the problem of multiple reads of the same signal.

8. Implementation

In implementing the 'television pause' function, I had to transfer control from each VCR to my circuit, assemble the assembler code, and debug my circuit. Debugging the circuit and assembling the code are fairly standard and do not need to be explained. In transferring control to my circuit, wires within each VCR needed to be cut and rewired within my circuit. The VCR schematics in figures 6.1 and 6.2 show the connections made to the insides of the VCR's. The switches shown on board 29 in figure 6.1 are buttons on VCR control panel. The lines were cut at the input of board 30 because the wires were the most accessible at this point. Emulating the functions of board 29 and 30 required little more than relays, resistors and a diode making this board a good choice. By cutting these wires I disabled many of the buttons on the control panel, but by hooking them up as shown in figure 7.2, the buttons were reactivated when the 'television pause' function is not in use. All the functions that my circuit needed to use, were controlled by placing signals on the five lines shown in figure 6.1. Figure 6.2 shows the point where I soldered to get the record and play control pulses.

9. Applications

This circuit has two main uses. One use is that it allows the user to miss a portion of a television program and be able to view the remaining program upon returning. The circuit can also be used as a commercial eliminator. By pushing the 'pause' button at the beginning of the show and returning ten minutes later, the user will be viewing the show from video tape. When a commercial is reached, the 'fast forward' button should be pushed. My circuit will then tell the VCR to search in fast forward mode. When the commercials are over, the 'fast forward' button is pushed again, so that my circuit knows to end the search mode. By allowing the user to fast forward through unnecessary portions, he will be able to approach real time and thus save viewing time.

10. Possible Additions

One feature that could be added to this circuit is a multiple pause feature. Using this feature the user could miss a section of the program, return to watch a portion of the program, then repeat this process as many times as needed. A good deal of software would have to be written but it would not require any additional hardware since the 'pause' button could be used.

Once digital encoding of video becomes more prevalent and less expensive, a digital recording technique should be used in place of the VCR's. If the search time of a disk became fast enough to switch within a time much less than the vertical sync of a TV signal, then only two or only one disk(s) would be needed for storage and much of the guess work as to timing would be eliminated. This would be a clear advantage over the slow and inaccurate VCR's.

Appendix A: Assembler Code Listing

VARIABLE	FUNCTION	LOCATION
countl	2 byte	R0
counth	counter	R1
vcr1l	frame count	R2
vcr1h	for vcr 1	R3
vcr2l	frame count	R4
vcr2h	for vcr 2	R5
vcr1rl	rev. frame count	R6
vcr1rh	for vcr1	R7
ptimel	amount of time the	30H
ptimeh	vcr plays for (in frames)	31H
rpointl	point at which vcr	32H
rpointh	must rewind	33H
startptl	the beginning of the	34H
startpth	vcr's recorded segment	35H
vcr3l	frame count	38H
vcr3h	for vcr 3	39H
vcr2rl	rev. frame count	3AH
vcr2rh	for vcr 2	3BH
cycle	# pass through the loop	3CH
rspeed	rewind speed	3DH

```
ORG     0000

        setb PSW.3          ;use only registers in
        setb PSW.4          ;register bank 3
```

```
;vcr1, vcr2, and vcr3 count the distance from the
;time record is initiated on each respective vcr.
;By storing these as variables, the software always
;knows where each vcr is in relation to its record point.
;First, the processor initializes the circuit. It does
;this by loading the registers and setting vcr1 to 0.

            mov R2, #0              ;vcr1=0
            mov R3, #0              ;
            mov DPTR, #0            ;
            mov A, #0               ;
            movx @DPTR, A           ;clear regs1-2
            mov A, #8               ;
            movx @DPTR, A           ;load video1 into regs3-4

;After the set-up, the processor waits for the user to
;push the television pause button.

get_button: movx A, @DPTR           ;read the inputs
            mov P1, A               ;put in bit addressable mem.
            mov C, P1.7             ;move pause button to carry
            jnc get_button          ;loop if button not pressed

;Once the user pushes the pause button, the processor
;tells all the vcr's to record. All three record because
;this saves alot of time in the early stages of the
;function. Also the time between switching is small
;in the early stages.

            mov DPTR, #14336        ;
            mov A, #64              ;
            movx @DPTR, A           ;load regs1-2 with rec.1, rec2,
                                    ;and rec3. Also switch control of
                                    ;functions to my circuit
            mov DPTR, #0            ;
            mov A, #8               ;
            movx @DPTR, A           ;load regs3-4 with video1

;The first vcr records for 44 sec., after which it rewinds
;and gets ready to play when the user gets back.
;The user is not allowed to push the pause button
;again within a minute because of this recorded segment.
;By adjusting this record time, the length of time the
;user must wait is also adjusted.

wait44:     movx A, @DPTR           ;read inputs
```

```
          mov P1, A               ;
          mov C, P1.1             ;mov cpr1 to carry
          jnc wait44              ;loop if not cpulse
          ;increment vcr1
          cjne R2, #255, wait44a  ;see if high byte needs inc
          inc R3                  ;inc high byte
wait44a:  inc R2                  ;inc low byte

          ;see if 44 sec have been recorded

          cjne R2, #40, wait44    ;see if low byte is equal
          cjne R3, #5, wait44     ;see if high byte is equal

          ;set vcr1 and vcr2

          mov A, R2               ;vcr1=vcr1+60+30
          add A, #90              ;low byte
          mov R2, A               ;
          mov A, R3               ;high byte
          addc A, #0              ;just add carry
          mov R3, A               ;

          mov R4, #180            ;vcr2=180 six seconds of recording
          mov R5, #0              ;

          ;stop vcr 1 so that it can be rewound

          mov DPTR, #0            ;
          mov A, #1               ;
          movx @DPTR, A           ;load r1-2 with stop1
          mov A, #8               ;
          movx @DPTR, A           ;load r3-4 with video1

          ;figure the amount of time vcr 1 has of recorded
          ;material to playback (actually it is already known:
          ;[44sec. + 1 sec. -.5 sec.]*30=1335. The 1 sec. refers
          ;to the inevitable over-rewind distance that must be
          ;played back. The .5 sec. refers to the fact that the
          ;outputs must be switched before the end of the recorded
          ;segment.
          ;distance from the

          mov 30H, 55             ;ptime=1335
          mov 31H, 5              ;

          ;Before the vcr comes to a full stop, there is
```

```
        ;a delay.  The processor waits for two seconds before
        ;giving any other commands.  The stop delay is larger
        ;than 2 sec. but the vcr's will accept another command
        ;at this time.

        mov R0, #0              ;set count=0

wait2:  movx A, @DPTR           ;read inputs
        mov P1, A               ;
        mov C, P1.2             ;move cpr2 to carry
        jnc wait2               ;loop if no cpulse

        inc R0                  ;count=count+1
        cjne R0, #60, wait2     ;loop if 2 sec. have not elapsed

        ;update vcr2 since count was used instead of vcr2

        mov A, R4               ;
        add A, #60              ;vcr2=vcr2+1
        mov R4, A               ;low byte
        mov A, R5               ;now add carry to high byte
        addc A, #0              ;
        mov R5, A               ;

        ;start vcr1 playing(to get ready for search rewind)

        mov DPTR, #256          ;
        mov A, #0               ;
        movx @DPTR, A           ;load r1-2 with play1
        mov DPTR, #0            ;
        mov A, #8               ;
        movx @DPTR, A           ;load r3-4 with video1

        ;wait for 6 sec. before rewinding (to make sure vcr
        ;is playing)

        mov R0, #0              ;set count=0

wait6:  movx A, @DPTR           ;read inputs
        mov P1, A               ;
        mov C, P1.2             ;move cpr2 to carry
        jnc wait6               ;if not cpulse, loop

        inc R0                  ;count=count+1
        cjne R0, #180, wait6    ;loop if 6 sec. have not passed
```

```
        ;Now vcr1 is ready to be rewound
        ;Rewind vcr1

        mov DPTR, #0          ;
        mov A, #0             ;
        movx @DPTR, A         ;don't send any signals to r1-2
        mov A, #9             ;
        movx @DPTR, A         ;load r3-4 with rewind1 and video1

        ;update vcr2

        mov A, R4             ;
        add A, #180           ;vcr2=vcr2+180
        mov R4, A             ;low byte
        mov A, R5             ;
        addc A, #0            ;add carry to high byte

        ;rewind vcr1 to the beginning of its recorded segment

        mov R0, #0            ;set count=0
        mov R1, #0            ;

        mov R6, #0            ;set vcr1r=0(this variable counts
                              ;back to vcr1=0
        mov R7, #0            ;

rewind: movx A, @DPTR         ;read inputs
        mov P1, A             ;
        mov C, P1.2           ;mov cpr2 to carry
        jnc rewindb           ;if not cpr2 check cpp1
                              ;count=count+1
        cjne R0, #255, rewinda ;if low byte of count will overflow
        inc R1                ;then increment high byte
rewinda: inc R0               ;increment low byte

rewindb: mov C, P1.4          ;mov cpp1 to carry
        jnc rewind            ;jmp if no cpp1
                              ;vcr1r=vcr1r+1
        cjne R6, #255, rewindc ;if low byte of count will overflow
        inc R7                ;then increment high byte
rewindc: inc R6               ;increment low byte

        ;check to see if beginning of recorded segment
        ;has been found

        mov A, R2             ;compare with vcr1
```

```
            mov P3, R6               ;
            cjne A, P3, rewind       ;first low byte
            mov A, R3                ;then second byte
            mov P3, R7               ;
            cjne A, P3, rewind       ;

            ;beginning of tape has been found
            ;stop vcr1

            mov DPTR, #0             ;
            mov A, #1                ;
            movx @DPTR, A            ;load r1-2 with stop1
            mov A, #8                ;
            movx @DPTR, A            ;load r3-4 with video1

            ;update vcr2

            mov A, R4                ;vcr2=vcr2+count
            add A, R0                ;low byte
            mov R4, A                ;
            mov A, R5                ;now high byte
            addc A, R1               ;

            ;find out what speed the vcr's are rewinding at

            ;slowest they will rewind at is 5x so this is the default

            mov A, R1                ;mov high byte of count to A
            jz find_rspeed6          ;if count>256 then figure speed
            mov 3DH, #5              ;rewind is slow so set default at 5
            jmp find_rpoint          ;skip find_rspeed

            ;figure out the speed (can be 5x, 6x, 7x, 14x, 15x, or 16x)

find_rspeed5:cjne R0, #235, find_rspeed5a    ;if count>235 then rspeed is 5
find_rspeed5a:jc find_rspeed6                ;if count<235 then jmp to next check
            mov 3DH, #5              ;rspeed=5
            jmp find_rpoint          ;skip other checks

find_rspeed6:cjne R0, #201, find_rspeed6a    ;if count>201 then rspeed is 6
find_rspeed6a:jc find_rspeed7        ;if count<201 then jmp to next check
            mov 3DH, #6              ;rspeed=6
            jmp find_rpoint          ;skip other checks

find_rspeed7:cjne R0, #100, find_rspeed7a    ;if count>100 then rspeed is 7
find_rspeed7a:jc find_rspeed14       ;if count<100 then jmp to next check
```

```
            mov 3DH, #7              ;rspeed=7
            jmp  find_rpoint         ;skip other checks

find_rspeed14:cjne R0, #94, find_rspeed14a  ;if count>94 then rspeed is 14
find_rspeed14a:jc find_rspeed15      ;if count<94 then jmp to next check
            mov 3DH, #14             ;rspeed=14
            jmp  find_rpoint         ;skip other checks

find_rspeed15:cjne R0, #88, find_rspeed15a  ;if count>88 then rspeed is 15
find_rspeed15a:jc find_rspeed16             ;if count<88 then jmp to next check
            mov 3DH, #15             ;rspeed=15
            jmp  find_rpoint         ;skip other checks

find_rspeed16:mov 3DH, #16           ;the highest rewind speed is 16

            ;figure out where to start rewinding vcr2

find_rpoint:mov  A, #57              ;rpoint=825*rspeed+60+135-30+60
            mov B, 3DH               ;    =825*rspeed+225
            mul  AB                  ;825*rspeed (low byte first)
            mov 32H, A               ;store low byte of mul in rpoint
            mov 33H, B               ;temp store overflow

            mov  A, #3               ;high byte next
            mov B, 3DH               ;
            mul  AB                  ;
            add A, 33H               ;add overflow to high byte
            mov 33H, A               ;store high byte in rpoint
                                     ;B will be 0 so it is not used

            mov  A, 32H              ;+225
            add A, #225              ;
            mov 32H, A               ;found and stored low byte of rspeed
            mov  A, 33H              ;figure high byte
            addc A, #0               ;
            mov 33H, A               ;found and stored high byte of rspeed

            ;reset variables vcr1r, vcr2r, and count

            mov  R6, #0              ;vcr1r=0
            mov  R7, #0              ;

            mov  3AH, #0             ;vcr2r=0
            mov  3BH, #0             ;

            mov  R0, #0              ;count=0
```

```
            ;vcr2 can only record for a finite amount of time
            ;based on how fast it can rewind.  Once rpoint (figured
            ;above) is reached vcr2 must be stopped.  The following
            ;section finds this point

fill_vcr2:  movx A, @DPTR            ;read inputs
            mov P1, A                ;
            mov C, P1.2              ;move cpr2 to carry
            jnc fill_vcr2b           ;if not cpr2 check if button pushed

            cjne R4, #255, fill_vcr2a ;inc vcr2
            inc R5                   ;inc high byte if necessary
fill_vcr2a: inc R4                   ;inc low byte

fill_vcr2b: mov C, P1.7              ;move button input to carry
            jc button2               ;jmp if viewer returned

            ;if viewer doesn't return

            mov A, 32H               ;check if vcr2 has reached its
            mov P3, R4               ;recording limit (vcr2=rpoint)
            cjne A, P3, fill_vcr2    ;loop if this point is not reached
            mov A, 33H               ;
            mov P3, R5               ;
            cjne A, P3, fill_vcr2    ;

            ;vcr2 has reached its limit
            ;it is stopped and vcr3 is set

            mov DPTR, #0             ;
            mov A, #2                ;
            movx @DPTR, A            ;load r1-2 with stop2
            mov A, #8                ;
            movx @DPTR, A            ;load r3-4 with video1

            mov 38H, #180            ;vcr3=180
            mov 39H, #0              ;this corresponds to 6 sec.

            ;vcr 3 is recording while 1 and 2 have recorded
            ;segments and are stopped
            ;At this point, the code waits for the user to
            ;push the pause button again.

return:     movx A, @DPTR            ;read inputs
            mov P1, A                ;
```

```
          mov C, P1.3              ;check if cpr3
          jnc returnb              ;if no cpulse see if button is pressed

          mov A, 38H               ;inc vcr3 (first check if
          cjne A, #255, returna    ;low byte is going to overflow)
          inc 39H                  ;inc high byte
returna:  inc 38H                  ;inc low byte

returnb:  mov C, P1.7              ;move button input to carry
          jnc return               ;loop if not pushed again

          ;Viewer has returned. Start playing back program
          ;from vcr 1.  Also tell vcr 2 to play so that it
          ;can be rewound.

          mov DPTR, #768           ;
          mov A, #0                ;
          movx @DPTR, A            ;load r1-2 with play1 and play2
          mov DPTR, #0             ;
          mov A, #8                ;
          movx @DPTR, A            ;load r3-4 with video1

          ;vcr 2 must wait for 4 sec. before rewinding (so that
          ;it has already started playing before rewinding)

wait4:    movx A, @DPTR            ;read inputs
          mov P1, A                ;
          mov C, P1.4              ;move cpp1 to carry
          jnc wait4b               ;if not cpp1 then see if cpr3

          cjne R6, #255, wait4a    ;inc vcr1r (high byte if
          inc R7                   ;necessary, then
wait4a:   inc R6                   ;low byte)

wait4b:   mov C, P1.3              ;move cpr3 to carry
          jnc wait4                ;if not cpr3 then loop

          inc R0                   ;inc counter
          cjne R0, #120, wait4     ;loop if 4 sec. have not elapsed

          ;vcr2 is now ready to be rewound.  Rewind vcr 2.

          mov DPTR, #0             ;
          mov A, #0                ;
          movx @DPTR, A            ;no signals for r1-2
          mov A, #10               ;
```

```
            movx @DPTR, A           ;load r3-4 with rewind2 and video1

            ;update vcr3

            mov A, #120             ;vcr3=vcr3+120
            add A, 38H              ;
            mov 38H, A              ;
            mov A, 39H              ;
            addc A, #0              ;
            mov 39H, A              ;

            ;jump over the other path

            jmp  paths_meet         ;

            ;This path is followed when vcr 2 is not filled before
            ;the viewer returns.  In the following code, vcr 1
            ;starts to play, while vcr 2 and vcr 3 are recording.
            ;The point at which vcr 2 must rewind so that it can
            ;record as much as possible but still be able to
            ;rewind in time to switch is searched for.

            ;First vcr 1 is told to play.

button2:    mov DPTR, #256          ;
            mov A, #0               ;
            movx @DPTR, A           ;load r1-2 with play1
            mov DPTR, #0            ;
            mov A, #8               ;
            movx @DPTR, A           ;load r3-4 with video1

            ;Second the rewind point for vcr 2 is looked for.

finish:     movx A, @DPTR           ;read inputs
            mov P1, A               ;
            mov C, P1.4             ;check if cpp1
            jnc finishb             ;if not cpp1 see if cpr2

            cjne R6, #255, finisha  ;inc vcr1r (high byte if
            inc R7                  ;necessary then
finisha:    inc R6                  ;low byte)

            ;refigure the point at which vcr 2 rewinds
            ;because vcr 1 has moved

            clr C                   ;clear the carry
```

```
            mov A, 32H              ;rpoint=rpoint-rspeed
            subb A, 3DH             ;
            mov 32H, A              ;
            mov A, 33H              ;
            subb A, #0              ;
            mov 33H, A              ;

finishb:    mov C, P1.2             ;check if cpr2
            jnc finishd             ;jmp to check if rpoint is reached

            cjne R4, #255, finishc  ;inc vcr2 (high byte if
            inc R5                  ;necessary, then
finishc:    inc R4                  ;low byte)

            ;check if rewind point has been reached

finishd:    clr C                   ;clear carry
            mov A, R4               ;
            subb A, 32H             ;is vcr2< rpoint
            mov A, R5               ;
            subb A, 33H             ;
            jnc finish              ;loop if this point has not been
                                    ;reached

            ;vcr 2 must be rewound now. Order to do this
            ;is stop2, play2, then rewind.

            mov DPTR, #0            ;
            mov A, #2               ;
            movx @DPTR, A           ;load r1-2 with stop2
            mov A, #8               ;
            movx @DPTR, A           ;load r3-4 with video1

            ;vcr3 must be given a reference count

            mov 38H, #180           ;vcr3=180
            mov 39H, #0             ;

            ;wait 2 sec. for vcr2 to stop

wait2_2:    movx A, @DPTR           ;read inputs
            mov P1, A               ;
            mov C, P1.4             ;move cpp1 to carry
            jnc wait2_2b            ;if not cpp1 check cpr3

            cjne R6, #255, wait2_2a ;inc vcr1r (high byte if
```

```
            inc R7                     ;necessary then
wait2_2a:   inc R6                     ;low byte)

wait2_2b:   mov C, P1.3                ;move cpr3 to carry
            jnc wait2_2                ;loop if not cpr3

            inc R0                     ;inc count
            cjne R0, #60, wait2_2      ;loop if 2 sec. have not elapsed

            ;now tell vcr 2 to play

            mov DPTR, #512             ;
            mov A, #0                  ;
            movx @DPTR, A              ;load r1-2 with play2
            mov DPTR, #0               ;
            mov A, #8                  ;
            movx @DPTR, A              ;load r3-4 with video1

            ;wait 6 sec to make sure vcr 2 begins playing

wait6_2:    movx A, @DPTR              ;read inputs
            mov P1, A                  ;
            mov C, P1.4                ;mov cpp1 to carry
            jnc wait6_2b               ;if not cpp1 check cpr3

            cjne R6, #255, wait6_2a    ;inc vcr1r (high byte if
            inc R7                     ;necessary, then
wait6_2a:   inc R6                     ;low byte)

wait6_2b:   mov C, P1.3                ;move cpr3 to carry
            jnc wait6_2                ;loop if not cpr3

            inc R0                     ;inc count
            cjne R0, #240, wait6_2     ;loop if 6 sec. have not elapsed

            ;now vcr 2 can be rewound

            mov DPTR, #0               ;
            mov A, #0                  ;
            movx @DPTR, A              ;don't load anything into r1-2
            mov A, #10                 ;
            movx @DPTR, A              ;load r3-4 with rewind2 and video1

            ;update vcr3

            mov A, #240                ;vcr3=vcr3+240
```

```
            add A, 38H              ;
            mov 38H, A              ;
            mov A, 39H              ;
            addc A, #0              ;
            mov 39H, A              ;

            ;The two paths that split as to whether vcr 2
            ;was filled or not when the viewer returned
            ;meet again here.  Both paths just told vcr 2 to
            ;start rewinding.  In the following sections, the
            ;point where vcr 2 needs to stop rewinding is found.

            ;figure out the start point of its recorded section

paths_meet:clr C                    ;clear the carry
            mov A, R4               ;start point=vcr2-135
            subb A, #135            ;
            mov 34H, A              ;
            mov A, R5               ;
            subb A, #0              ;
            mov 35H,A               ;

            ;this point is searched for

paths:      movx A, @DPTR           ;get inputs
            mov P1, A               ;
            mov C, P1.4             ;move cpp1 to carry
            jnc pathsb              ;if not cpp1 check cpr3

            cjne R6, #255, pathsa   ;inc vcr1r (high byte if
            inc R7                  ;necessary, then
pathsa:     inc R6                  ;low byte)

pathsb:     mov C, P1.3             ;move cpr3 to carry
            jnc pathsd              ;if not cpr4 check cpp2

            mov A, 38H              ;inc vcr3 (high byte if
            cjne A, #255, pathsc    ;necessary
            inc 39H                 ;
pathsc:     inc 38H                 ;then low byte)

pathsd:     mov C, P1.5             ;mov cpp2 to carry
            jnc paths               ;loop if not cpp2

            mov A, 3AH              ;inc vcr2r (high byte
            cjne A, #255, pathse    ;if
```

```
          inc 3BH                 ;necessary then
pathse:   inc 3AH                 ;low byte)

          mov A, 34H              ;see if beginning of segment is
                                  ;found (vcr1r=startpt)
          cjne A, 3AH, paths      ;loop if low byte is not equal
          mov A, 35H              ;
          cjne A, 3BH, paths      ;loop if high byte is not equal

          ;the beginning of vcr 2's recorded segment has been
          ;found. To get out of search rewind mode it is signalled
          ;to play.

          mov DPTR, #512          ;
          mov A, #0               ;
          movx @DPTR, A           ;load r1-2 with play2
          mov DPTR, #0            ;
          mov A, #8               ;
          movx @DPTR, A           ;load r3-4 with video1

          ;reset count

          mov R0, #0              ;count=0

          ;vcr 2 is allowed to play for 2 sec. then it is paused

play_for2: movx A, @DPTR          ;get inputs
          mov P1, A               ;
          mov C, P1.4             ;move cpp1 to carry
          jnc playb               ;if not cpp1, check cpr3

          cjne R6, #255, playa    ;inc vcr1r (high byte if
          inc R7                  ;necessary then
playa:    inc R6                  ;low byte)

playb:    mov C, P1.3             ;move cpr3 to carry
          jnc play_for2           ;loop if not cpr3

          inc R0                  ;inc count
          cjne R0, #60, play_for2 ;loop if 2 sec. have not elapsed

          ;vcr 2 should now be paused to wait for the end
          ;of vcr 1's segment to be played out

          mov DPTR, #0            ;
          mov A, #16              ;
```

```
            movx @DPTR, A           ;load r1-2 with pause2
            mov A, #8               ;
            movx @DPTR, A           ;load r3-4 with video1

            ;update vcr3

            mov A, 38H              ;vcr3=vc3+60
            add A, #60              ;
            mov 38H, A              ;
            mov A, 39H              ;
            addc A, #0              ;
            mov 39H, A              ;

            ;At this point vcr 2 is in pause mode, vcr 3 is recording,
            ;and vcr 1 is just about to finish playing its recorded
            ;segment.  vcr 2 will stay in pause mode until 2 sec.
            ;before vcr 1 is finished.

            ;this point is figured next and temporarily stored
            ;in count

            clr C                   ;the carry is cleared
            mov A, 30H              ;count=ptime-60
            subb A, #60             ;
            mov R0, A               ;
            mov A, 31H              ;
            subb A, #0              ;
            mov R1, A               ;

            ;Now this point is looked for:

pause:      movx A, @DPTR           ;get inputs
            mov P1, A               ;
            mov C, P1.3             ;mov cpr3 to carry
            jnc pauseb              ;if not cpr3, check cpp1

            mov A, 38H              ;
            cjne A, #255, pausea    ;inc vcr3 (high byte if
            inc 39H                 ;necessary, then
pausea:     inc 38H                 ;low byte)

pauseb:     mov C, P1.4             ;move cpp1 to carry
            jnc pause               ;loop if not cpp1

            cjne R6, #255, pausec   ;inc vcr1r (high byte if
            inc R7                  ;necessary, then
```

```
pausec:     inc R6                  ;low byte)

            mov A, R6               ;see if play point is reached
            mov P3, R0              ;
            cjne A, P3, pause       ;
            mov A, R7               ;
            mov P3, R1              ;
            cjne A, P3, pause       ;

            ;Play point for vcr2 has been found.  The command for
            ;vcr 2 to play is given next.

            mov DPTR, #512          ;
            mov A, #0               ;
            movx @DPTR, A           ;load r1-2 with play2
            mov DPTR, #0            ;
            mov A, #8               ;
            movx @DPTR, A           ;load r3-4 with video1

            ;reset count

            mov R0, #0              ;count=0

            ;The actual switching doesn't take place here
            ;since it takes play a little time to get up
            ;to speed.  Switching does take place after a
            ;2 sec. delay.

switch:     movx A, @DPTR           ;get inputs
            mov P1, A               ;
            mov C, P1.3             ;mov cpr3 to carry
            jnc switch              ;loop if not cpr3

            inc R0                  ;inc count
            cjne R0, #60, switch    ;loop if 2 sec have not elapsed

            ;The switch point has been reached.  The outputs are
            ;switched and vcr 1 is stopped.

            mov DPTR, #0            ;
            mov A, #1               ;
            movx @DPTR, A           ;load r1-2 with stop1
            mov A, #16              ;
            movx @DPTR, A           ;load r3-4 with video2

            ;update vcr3
```

```
mov  A, 38H             ;vcr3=vcr3+60
add A, #60              ;
mov  A, 39H             ;
addc A, #0              ;

;figure play time of vcr 2 (for vcr 3 to use)

clr C                   ;clear carry
mov  A, R4              ;ptime=vcr2-240
subb A, #240            ;
mov 30H, A              ;
mov  A, R5              ;
subb A, #0              ;
mov 31H, A              ;

;figure rewind point for vcr 3

clr C                   ;clear carry
mov  A, 30H             ;rpoint=(ptime-780-vcr2r)rspeed+495
subb A, #12             ;ptime-780
mov 32H, A              ;
mov  A, 31H             ;
subb A, #3              ;
mov 33H, A              ;
mov  A, 32H             ;ptime-780-vcr2r
subb A, 3AH             ;
mov 32H, A              ;
mov  A, 33H             ;
subb A, 3BH             ;
mov 33H, A              ;
mov  A, 32H             ;(ptime-780-vcr2r)rspeed
mov  B, 3DH             ;
mul  AB                 ;
mov 32H, A              ;
mov  R0, B              ;temp store overflow
mov  A, 33H             ;
mov  B, 3DH             ;
mul  AB                 ;
add A, R0               ;add in overflow
mov 33H, A              ;
mov  A, 32H             ;+495
add A, #239             ;
mov 32H, A              ;
mov  A, 33H             ;
addc A, #1              ;
```

```
        mov 33H, A              ;

        ;the program loops here until the user pushes the stop button

        mov A, 3CH              ;cycle=cycle+1 max 2
        cjne A, #2, cycle1      ;
        mov 3CH, #0             ;
        jmp loop                ;loop

cycle1: inc 3CH                 ;inc cycle
        jmp loop                ;loop


end
```

Appendix B: Hardware Schematics

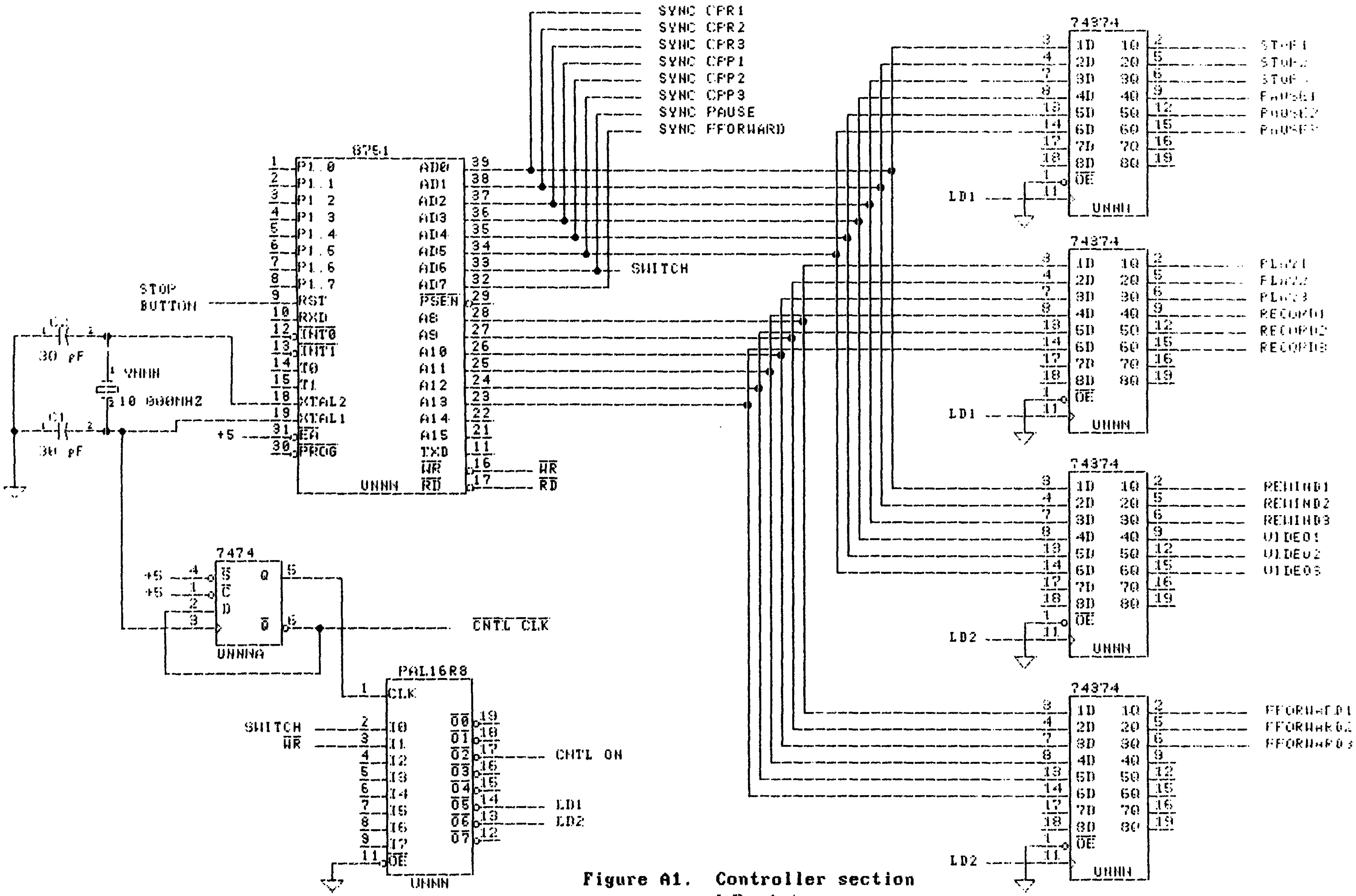


Figure A1. Controller section and Registers

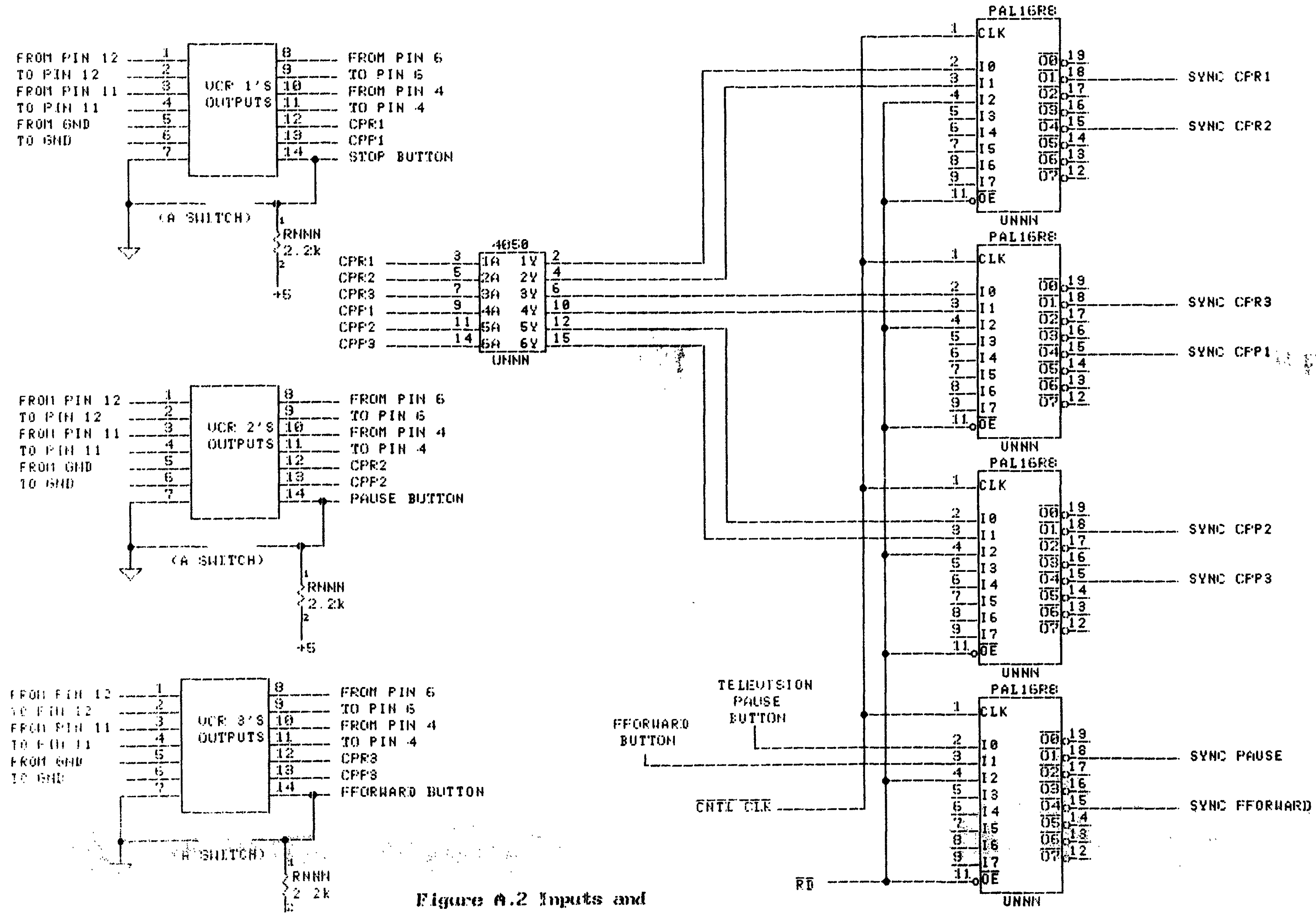


Figure A.2 Inputs and

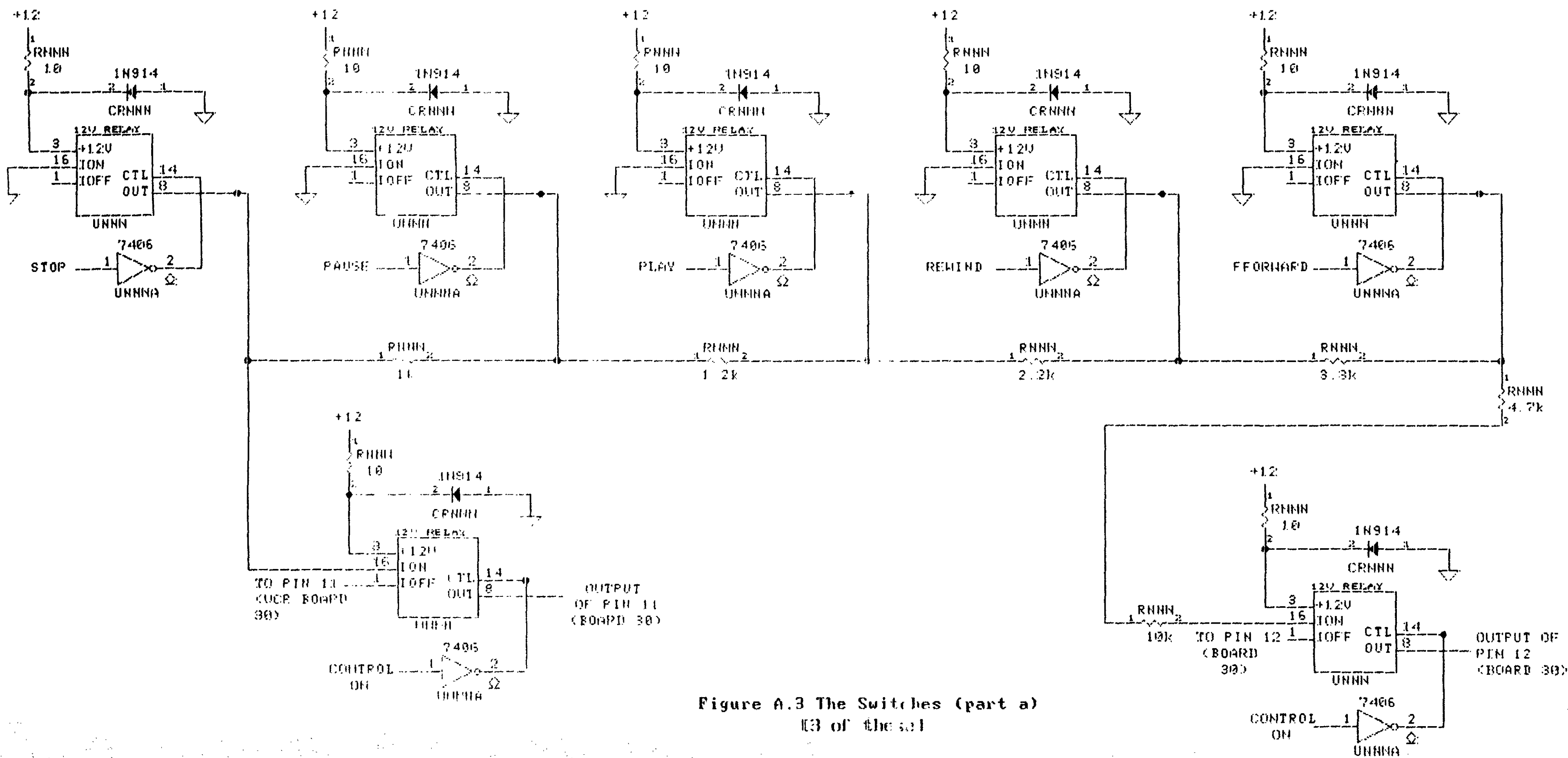
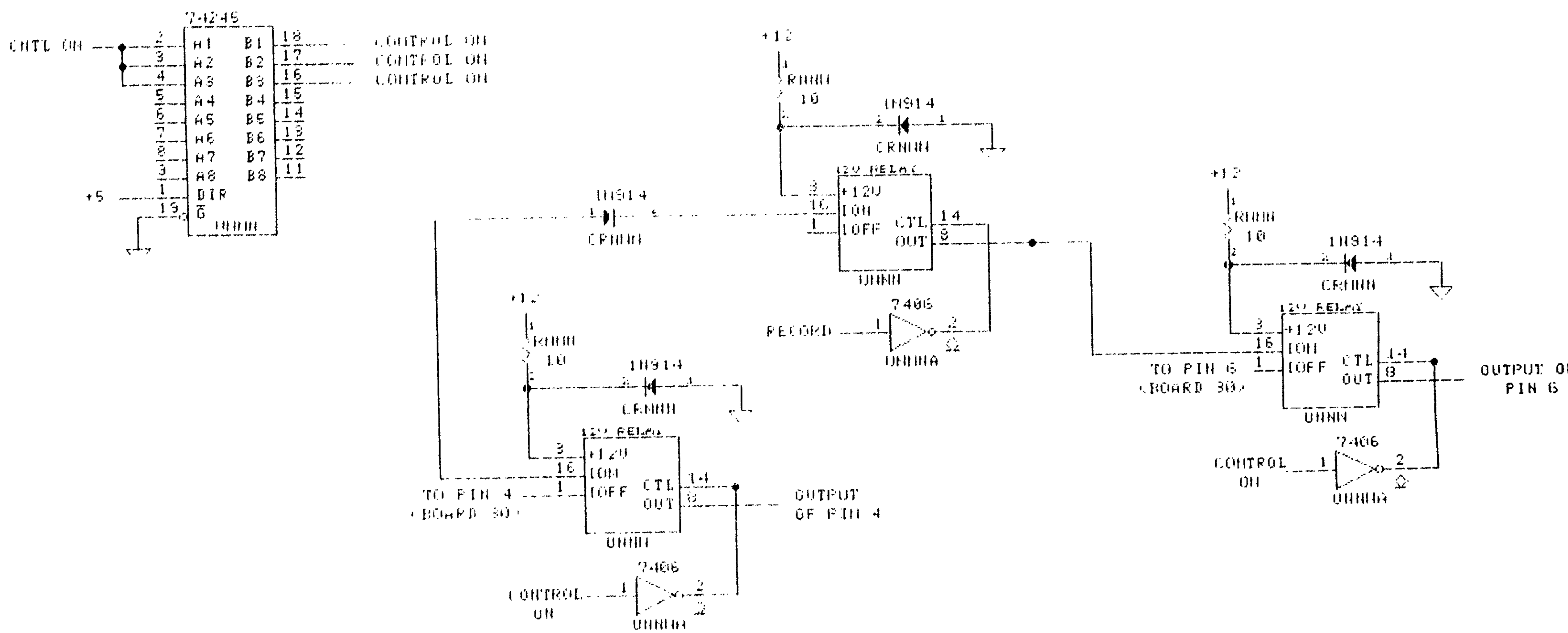


Figure A.3 The Switches (part a)
[3 of these]



CNTL ON
74245
CONTROL ON
CONTROL ON
CONTROL ON
+5
1N914
CRNNN
+12
RNNN
10
12V RELAY
+12V
ION
IOFF
CTL
OUT
UNNN
7406
UNNNA
RECORD
TO PIN 4
(BOARD 30)
OUTPUT
OF PIN 4
CONTROL
ON
TO PIN 6
(BOARD 30)
OUTPUT OF
PIN 6

Appendix C: Pal Programs

```
MODULE vcrcntl
FLAG '-R3';
TITLE 'Control pal for the television pause function
     Mike Truog
     Electronic Publishing Group May 23, 1989'

    vcrpal  DEVICE 'P16R8';

"inputs
    clk       PIN  1;
    switch    PIN  2;
    wr        PIN  3;
    reset     PIN  4;

"outputs
    WRIN     PIN  19 = 'reg, feed_pin';
    SYNCWR   PIN  18 = 'reg, feed_pin';
    CNTLON   PIN  17 = 'reg, feed_pin';
    ST       PIN  16 = 'reg, feed_pin';
    LD1      PIN  14 = 'reg, feed_pin';
    LD2      PIN  13 = 'reg, feed_pin';

" constants

    H, L, X, C, Z = 1, 0, .X., .C., .Z.;
    h, l, x, c, z = 1, 0, .X., .C., .Z.;

Equations

    WRIN := !wr;
    SYNCWR := !wr & !WRIN;
    CNTLON := SYNCWR & switch & !CNTLON & !reset
              # CNTLON & !switch & !reset
              # CNTLON & !SYNCWR & !reset;
    LD1 := !ST & SYNCWR;
    LD2 := ST & SYNCWR;
    ST := !ST & SYNCWR & !reset # ST & !SYNCWR & !reset;


"WRIN and SYNCWR synchronize WR (write from the microprocessor)
"so that it is only seen for one clock pulse.
"STATE is a two state state machine.  It toggles so that
"the first two registers are loaded on one write
"then the second two are recorded on the next write and back again.
```

```
test_vectors 'Test Synchronization and Load Signals'

([ clk, reset, switch, wr ] -> [ WRIN, SYNCWR, CNTLON, LD1, LD2, ST ])

[ C, H, L, H ] -> [ L, L, L, X, X, L ];
[ C, L, L, H ] -> [ L, L, L, L, L, L ];
[ C, L, H, H ] -> [ L, L, L, L, L, L ];
[ C, L, H, L ] -> [ H, H, L, L, L, L ];
[ C, L, H, L ] -> [ H, L, H, H, L, H ];
[ C, L, H, L ] -> [ H, L, H, L, L, H ];
[ C, L, H, L ] -> [ H, L, H, L, L, H ];
[ C, L, L, H ] -> [ L, L, H, L, L, H ];
[ C, L, L, H ] -> [ L, L, H, L, L, H ];

[ C, L, L, L ] -> [ H, H, H, L, L, H ];
[ C, L, L, L ] -> [ H, L, H, L, H, L ];
[ C, L, L, L ] -> [ H, L, H, L, L, L ];
[ C, L, L, H ] -> [ L, L, H, L, L, L ];
[ C, L, L, H ] -> [ L, L, H, L, L, L ];


END  vcrcntl
```

```
MODULE vcrsync
FLAG  '-R3';
TITLE 'Synchronization pal for the television pause function
     Mike Truog
     Electronic Publishing Group May 23, 1989'

    vcrpal2  DEVICE 'P16R8';

"inputs
    clk       PIN  1;
    cpr1      PIN  2;
    cpr2      PIN  3;
    rd    PIN  4;
    reset     PIN  5;

"outputs
    READYR1    PIN   19 = 'reg, feed_pin';
    OUTR1      PIN   18 = 'reg, feed_pin';
    USEDR1     PIN   17 = 'reg, feed_pin';
    READYR2    PIN   16 = 'reg, feed_pin';
    OUTR2      PIN   15 = 'reg, feed_pin';
    USEDR2     PIN   14 = 'reg, feed_pin';

" constants

    H, L, X, C, Z = 1, 0, .X., .C., .Z.;
    h, l, x, c, z = 1, 0, .X., .C., .Z.;

Equations

    READYR1 := !cpr1 # READYR1 & !rd;
    OUTR1 := !READYR1 & !USEDR1 & !reset # OUTR1 & !rd & !reset;
    USEDR1 := OUTR1 & !rd & !reset # USEDR1 & cpr1 & !reset;
    READYR2 := !cpr2 # READYR2 & !rd;
    OUTR2 := !READYR2 & !USEDR2 & !reset # OUTR2 & !rd & !reset;
    USEDR2 := OUTR2 & !rd & !reset # USEDR2 & cpr2 & !reset;

"This pal synchronizes input signals such that the data is
"valid a short time after the signal is first seen, but is
"not enabled until the microprocessor executes a read command.
"Data is held stable for the length of the read plus a short
"time after the read is finished.  The data is then unasserted
"and is not reasserted until after the signal goes low (thus it
"only outputs each signal once.
```

```
test_vectors 'Test Synchronization'

([ clk, cpr2, rd, reset ] -> [ READYR2, OUTR2, USEDR2 ])

[ C, L, L, H ] -> [ H, L, L ];
[ C, L, L, L ] -> [ H, L, L ];
[ C, H, L, L ] -> [ H, L, L ];
[ C, H, L, L ] -> [ H, L, L ];
[ C, H, H, L ] -> [ L, L, L ];
[ C, H, H, L ] -> [ L, H, L ];
[ C, H, H, L ] -> [ L, H, L ];
[ C, H, L, L ] -> [ L, H, H ];
[ C, H, L, L ] -> [ L, H, H ];
[ C, H, H, L ] -> [ L, L, H ];
[ C, H, H, L ] -> [ L, L, H ];
[ C, L, H, L ] -> [ H, L, L ];


[ C, L, H, L ] -> [ H, L, L ];
[ C, H, L, L ] -> [ H, L, L ];
[ C, H, L, L ] -> [ H, L, L ];
[ C, H, L, L ] -> [ H, L, L ];
[ C, H, H, L ] -> [ L, L, L ];
[ C, H, H, L ] -> [ L, H, L ];
[ C, H, H, L ] -> [ L, H, L ];
[ C, H, L, L ] -> [ L, H, H ];
[ C, H, L, L ] -> [ L, H, H ];
[ C, H, H, L ] -> [ L, L, H ];
[ C, H, H, L ] -> [ L, L, H ];
[ C, L, H, L ] -> [ H, L, L ];


[ C, L, H, L ] -> [ H, L, L ];
[ C, H, H, L ] -> [ L, L, L ];
[ C, H, H, L ] -> [ L, H, L ];
[ C, H, L, L ] -> [ L, H, H ];
[ C, H, H, L ] -> [ L, L, H ];
[ C, H, H, L ] -> [ L, L, H ];
[ C, H, L, L ] -> [ L, L, H ];
[ C, L, L, L ] -> [ H, L, L ];


END vcrsync
```

Acknowledgements

To Walter for providing me with a project that I could enjoy the long nights I spent working on it and for not giving up on me.

To Foof for helping me out in every way possible.

To Wad for answering a whole range of stupid questions.

To Scott F. for all the advice during all stages of the design and debugging.

To my parents, who wanting to see me graduate, gave me alot of support and encouragement.

To Alexis for everything. You made the tough times much better.