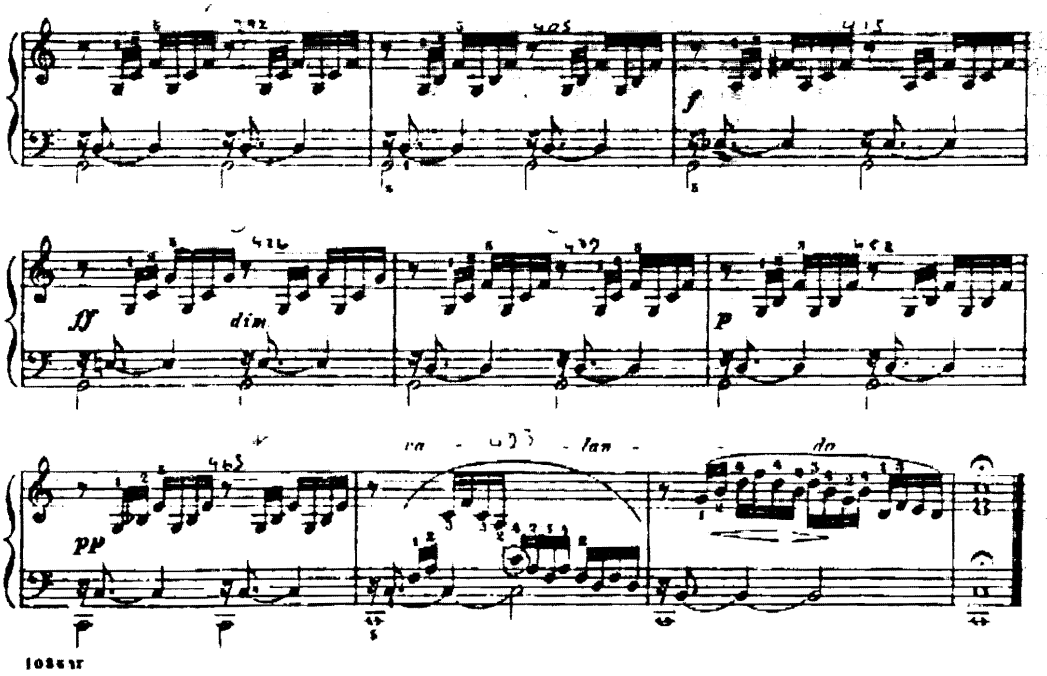
Spring 5-1991

# Multi-Channel Music Synthesis Using an ASIC

Lewis Thornberry

# Multi-channel Music Synthesis Using an ASIC




Andrew B. Gardner & Lewis Thornberry
ECE-494: Introduction to ASIC Design
April 23, 1991
Professor: Dr. Don Bouldin

TABLE OF CONTENTS

## ABSTRACT

The high speed and multiple channel requirements necessary for music synthesis suggest the use of an application-specific integrated circuit (ASIC). While a microprocessor is capable of controlling such a system, the parallel architecture and precision timing and I/O capabilities of an ASIC make it an attractive alternative, particularly when high-quality results are desired (like accurate note frequencies).

Inputs to this particular ASIC are eight bits wide. The data structure of the system is not complex, and the options available include: tempo control (fast or slow), variable note duration (in multiples of a $16^{th}$ note), and selectable pitch (including a range of three octaves). Notes may be played indefinitely without a pause by setting an endnote bit. The output of this circuit is a variable frequency square wave that can be used in conjunction with drivers to control a speaker element.

## INTRODUCTION

The music synthesis ASIC designed provided a practical introduction to many VLSI design concepts: simulation methods, CAD tools, and space limitations. An effort to reduce the complexity of the chip and data encoding scheme clearly resulted in a tradeoff in 'extra' functions. Music in the form of notes from a three-octave range are encoded in a memory device (ROM or RAM). The music synthesis system sequentially addresses the memory to provide an appropriate input signal to the ASIC, in turn providing an appropriate output signal for music generation (the variable frequency square wave). Rest notes and tempo were considered and implemented.

Pin assignment was not a problem in this particular project; if a larger chip size were available, an improved data loading scheme would have to be developed to accomodate multiple channels (the current method requires eight pins of input per channel... obviously four or more channels becomes unwieldy in terms of package size). Because of the low frequency range of human hearing this ASIC will not be overly taxed with speed requirements. Rather the space limitations of the TinyChip forced difficult decisions on which functions (tempo, special sound effects, number of available notes, number of tempos,...) to include, and prohibited the addition of an internal ROM containing the count map (making song encoding easier) and additional channels.

One major consideration in the design of this project was exactly how sound would be produced. Different notes are recognized by the human ear based on frequency and waveform quality. Since the latter variable is rather ambiguous and not easily addressable, the primary function of this ASIC was to generate an appropriate frequency square wave. The accuracy of this frequency was directly related to the size of the duration counter used in the circuit. To obtain 'quality' sound, each note's frequency must be within a 3% tolerance (or 3-cent tolerance in music jargon). There still existed an option, however, concerning the final output stage of the system: it was possible to have the ASIC generate n-bit words at a variable frequency, with each word representing a digitized waveform. Or the ASIC could clock a ROM containing a digitized waveform. These words (from the ASIC or ROM) could then be decoded by a digital-to-analog converter (DAC) which

would drive a speaker element. While the waveform quality would undoubtedly improve, the difficulty of this impementation was deferred.

The last important consideration for this project was the complexity of the supporting system. It appears that while the ASIC will require several auxiliary integrated circuits (see Figure 1 in Appendix I), these will be easily connected. Still, the music synthesizer chip designed is hardly a stand-alone component! The alternative to an ASIC or microprocessor is a direct implementation of logic using SSI/MSI/LSI IC's. This hardly seems feasible, though, since this circuit required approximately 100 devices and the support of a clever finite state machine controller.

## SYSTEM LEVEL DESIGN

As mentioned in the introduction, there is more than one possible implementation of this music synthesis system. The proposed stand-alone system is shown in Figure 1. Logic Works was used to generate the logic of the ASIC. This logic was combined with a finite state machine controller to produce the chip. Due to pin limitations, a maximum of three music channnels can be implemented on the TinyChip using the current data encoding scheme.

For every song he intends to generate, the user is asked to produce a Song Prom, a programmable memory that contains a sequence of note codes. These codes are addressed by the ASIC's address select lines. The codes are then used to address a value in the subsequent Count Map, a pre-packaged memory containing frequency-related count values that are loaded into the ASIC. A Count Map is required for each individual channel the user intends to produce; he therefore controls the width of his programmable memory.

The data field that the user must master is simply composed of a six-bit field for each note: five of the bits determine the specific frequency of the note (or rest) (two octaves are available). The sixth bit is an endnote bit. The purpose of this endnote is to tell the ASIC whether to turn the note off at the end of a 16thnote cycle. By leaving the note on (and playing it again), a continuous note can be produced of long duration.

The address-select and ROMs feed the input signal (the encoded song) into the ASIC. An additional chip requirement is the external clock, which is necessary to generate note timing as ø1 and ø2 are too fast for audio frequencies. The clock, a 100 kHz signal, also feeds a configuration of four decade counters. These are used to extract the 100 Hz and 10 Hz frequencies the ASIC needs to keep tempo.

Since the ASIC is not a high current-output device, a line drive will interface the ASIC to the speaker element. It is anticipated that the power rating, dynamic range, and dynamic impedance of the speaker will not present much challenge to interfacing. With the current system setup, it is still possible to use the output of the ASIC to clock a ROM containing a digitized waveform. The ROM output word could be decoded and used by a DAC.

## CHIP-LEVEL DESIGN

## HARDWARE

The ASIC is designed to support a finite state machine, or controller, and its peripheral random logic. These functions are illustrated in Figure 2. While the FSM is actually programmed with special software, the surrounding schematics are developed using Logicworks.

### REPEATABLE CHANNEL PROCESSING

Of these schematics, perhaps the most crucial element is the repeatable channel processing logic that is shown within dotted lines. With only minor changes to the control program, and simple cut and paste operations in Logicworks, this block can be repeated as long as space remains within the chip (and the necessary pins are available). Of course, each repitition of the grouped functions adds to the overall parallel processing effectiveness of the ASIC, and is thus the main advantage of the automated approach. Unfortunately, it is clear that there is no room on the TinyChip for even a second channel of logic. Perhaps the design could be further minimized in order to achieve multiple channels.

The operation of the logic is simple. The 8-bit input that corresponds to the note to be played is loaded into the counter, where it is tested for zero by the FSM. The controller will increment the count if zero has not been reached, and will toggle the output when zero has been reached. After expiration of the count, the same count value is reloaded into the counter for incrementing. This cycle continues until the note's duration has expired, at which time a new count value appears at the input.

### THE IMPLICIT REST GENERATION

The Implicit Rest Generator of Figure 4 determines when to finally turn off a note's ringing. This mimics music in the real world, where blocks of time are actually composed of both musical notes and various unrecorded rests between notes. The user has direct control over the Implicit Rest Generator. Since each data entry corresponds to the duration of a 1/16th note, the pogrammer may implement a series of notes separated by pauses (such as 4 sixteenth notes with the Endnote bit asserted in each), or one long continuous note (such

as one quarter note implemented as 4 continuous sixteenth notes). In other words, the Endnote bit allows the user to program any continuous note length, with a resolution of the sixteenth note.

Key to the generation of the implicit rest is the 5 Bit Counter, which need not be repeated, but can be tapped by all the channels. Its input is the 1kHz tap of the trio of decade counters in the external circuitry. The two bits that are tapped at the output of the counter correspond to the two available tempos. These bits go high, for the first time in the playing of a note, when around 75-80% of the note's duration has already passed (see CLOCKING). They are gated with the endnote bits and tempo bit of each separate channel processing block. This boolean sum of products is latched as the value IREST, or implicit rest. This line is utilized in the Rest Generator. The FSM clears the counter and latch values at the beginning of a new note.

REST GENERATION

This subset of logic, detailed in Figure 5, serves as the connection between the controller, which is entirely ignorant of rests, and the two rest control lines, IREST and REST. REST, or note rest, is asserted when the user inputs the code for rests (see SIMULATION section). The channel input lines are combined in boolean fashion to detect the REST signal, and the two rest control lines are then added together to produce an output enable line. This enable line, when asserted, prevents the CHTOGGLE output of the FSM from propagating through to the chip's output. The output is instead pulled low for the duration of the note.

TEMPO GENERATION

Also on-chip is the Tempo Generator of Figure 3. This logic generates the control line that determines when a note's duration has expired, and thus orders the playing of the next note. CN, or note clock, is derived from the 10Hz input. The slow clock is divided by four internally to produce two tempo options, which the user selects from with the TEMPO input line. The user pulls TEMPO high for fast operation, where whole notes last around 1.6 seconds, or grounds the line for the slower speed, where whole notes last as long as 6.4 seconds.

## CLOCKING

The counting is clocked by a100kHz toggle clock, or CT. It will generate note periods with a resolution of 0.02msec. By using an 8 bit counter, we designed for the generation of frequencies between 50 kHz and 195 Hz, which certainly includes the two octave range that we targeted.

How accurate are our generated frequencies? Due to pin limitations, we were forced to sacrifice some accuracy. Since the Count Map feeds only 8 bits to the ASIC, one of those 8 bits must contain the Endnote data that the user programmed in the Song Prom. We chose to make the LSB of the Count Map outputs equal to the Endnote bit. Therefore, we have limited the imposed error to plus or minus 0.02msec per period. This is in addition to the intrinsic error associated with our limited resolution.

Analysis shows that most of the notes are within 1% of their ideal values. The only severe errors occur in the last few notes at the top of the frequency range. The human ear is most likely to detect error when two similar notes are played in sequence. Assuming that the worst case results in the playing of B5 and C6, which are 987.7Hz and 1041.67Hz, respectively, the user will hear:

(1/980.4)/0.02msec ==> 51 counts ==> (1/51*0.02msec)
or 980 Hz, and
(1/1041.67)/0.02msec ==> 48 counts + endnote ==> (1/49*0.02msec),
or 1020 Hz.

That is about a 70% reduction in step between notes, and may be noticed by the music expert. But with this application, in which low quality speakers produce low quality sound, it is unlikely that at such high frequencies, the quality will be noticeably reduced for the average listener.

We discussed earlier how the 100kHz toggle clock is tapped by four decade counters to produce a 10Hz note clock, and also how the note clock is again tapped in a two bit counter to provide 2 choices for tempo. Now let us describe how the decision to enable the IREST line is made.

It is obvious that IREST can not be made immediately available at the beginning of the note's duration. That would make the note

sound like a rest note. The idea is to allow the note to play for most of its alloted time before enabling the IREST to bring the channel's output low. This is done by tapping the external decade counters at the 100Hz signal, for the third clock input. This clock feeds into a 5 Bit Duration Counter. When the fourth bit of that counter goes high, the elapsed time has been 8*1/10Hz, or 0.08 seconds, which is nearly 80% of the duration of a note in fast tempo. It is also true that all five bits go high at once every 32 counts. This boolean tap provides an elapsed time of 0.32 seconds, which is 80% of the slow tempo's duration of 0.4 sec. We therefore conclude that combining these two Duration Counter taps with the tempo and endnote bits provides an Implicit Rest function for each tempo.

## FIRMWARE

We have spared our controller from the nightmare of rests. Nor do we ask that it produce count values that correspond to user specified notes. In turn, the FSM worked flawlessly, requiring the correction of only one error in the entire program. Contrast that to the Logicworks effort, which required countless revisions. A good design strategy may be to accomplish as much as one comfortably can in the hardware development language, and to implement the more challenging tasks in logic. Doing this will produce a majority of the design in a small portion of the total design time, and will keep the difficult debugging sessions focussed only on the understandable schematics.

A major task that our FSM performs is the constant manipulation and sampling of the counter. If the counter is at zero, the program must be ready to toggle the CHTOGGLE output, after which it must perform a LOAD command. The controller must therefore check each channel's counter, and follow channel-specific instructions in channel-specific states. Theoretically, it will be possible for the FSM to fail at this elementary task if multiple counters go to zero simultaneously. This challenge would have to be addressed if we had more than one channel.

The FSM is also in charge of clearing the CT and CN latches after these clocks are detected. This requires two additional lines, CLRN and CLRT. These lines are only asserted during the state immediately following the detection of the clocks. CLRN also clears the random logic's flip flops.

A final task that the FSM must perform is the maintenance of the address select lines. The controller has a 2 bit output field that feeds two chip output pads, ADDR1 and ADDR2. Two bits are required because four functions are implemented: clear, remove clear, increment, and hold.

## INPUT/OUTPUT

## PIN ASSIGNMENTS

The chip's pin assignments (see Figure 6) were changed quite frequently during the design. The initial four channel design with internal Count Maps was discarded when it was found that such secondary control is more easily implemented external to the chip. This made the 8 bit Count Map outputs become chip input pins, thereby increasing the I/O requirement for each channel to 9 pins. This, in turn, left us with hopes of fitting 2 or 3 channels on the 25 available pins, because we had to have pins for PHI1 and PHI2, Vdd and GND, the 3 clocks, TEMPO, INITIAL, and the address select field.

If the first two channels are 9 pins each (remember to count the output pin), then that leaves only 7 pins for the third channel. If we were indeed designing three channels, then perhaps the third channel would be defined over only one octave (a bass clef, perhaps), so that it could function with just a 6 bit counter. Or perhaps we would design an extensive testing scheme, using all 7 bits to verify functionality after fabrication. Both of these schemes require more logic, and cannot be implemented on our crowded TinyChip.

## USER DATA FORMAT

The user is asked to familiarize himself with the data entry scheme for the Music Synthesis. He must produce a 6 bit field containing frequency and duration information for each note. The first five bits correspond to the 25 notes available in the two octaves of interest, and also include a code for the musical rest. Of course, the sixth bit is the all-important Endnote bit. The following table provides a listing of the note codes and the respective Count Map outputs. Note that the Count Map's LSB is the Endnote bit.

| NOTE | 5BIT CODE | ENDNOTE | MAP OUTPUT |
|---|---|---|---|
| C4 | 00000 | 0 | 01000000 |
| | | 1 | 01000001 |
| C#4 | 00001 | 0 | 01001010 |
| | | 1 | 01001011 |
| D4 | 00010 | 0 | 01010100 |
| | | 1 | 01010101 |
| D#4 | 00011 | 0 | 01011110 |
| | | 1 | 01011111 |

| | | | |
|---|---|---|---|
| E4 | 00100 | 0 | 01100110 |
| | | 1 | 01100111 |
| F4 | 00101 | 0 | 01110000 |
| | | 1 | 01110001 |
| F#4 | 00110 | 0 | 01111000 |
| | | 1 | 01111001 |
| G4 | 00111 | 0 | 01111110 |
| | | 1 | 01111111 |
| G#4 | 01000 | 0 | 10000110 |
| | | 1 | 10000111 |
| A4 | 01001 | 0 | 10001100 |
| | | 1 | 10001101 |
| A#4 | 01010 | 0 | 10010100 |
| | | 1 | 10010101 |
| B4 | 01011 | 0 | 10011010 |
| | | 1 | 10011011 |
| C5 | 01100 | 0 | 10011110 |
| | | 1 | 10011111 |
| C#5 | 01101 | 0 | 10100100 |
| | | 1 | 10100101 |
| D5 | 01110 | 0 | 10101010 |
| | | 1 | 10101011 |
| D#5 | 01111 | 0 | 10101110 |
| | | 1 | 10101111 |
| E5 | 10000 | 0 | 10110010 |
| | | 1 | 10110011 |
| F5 | 10001 | 0 | 10110110 |
| | | 1 | 10110111 |
| F#5 | 10010 | 0 | 10111010 |
| | | 1 | 10111011 |
| G5 | 10011 | 0 | 10111110 |
| | | 1 | 10111111 |
| G#5 | 10100 | 0 | 11000010 |
| | | 1 | 11000011 |
| A5 | 10101 | 0 | 11000110 |
| | | 1 | 11000111 |
| A#5 | 10110 | 0 | 11001000 |
| | | 1 | 11001001 |
| B5 | 10111 | 0 | 11001100 |
| | | 1 | 11001101 |
| C5 | 11000 | 0 | 11001110 |
| | | 1 | 11001111 |
| REST | 11111 | X | 00100000 |

## SIMULATION DISCUSSION

THOR provided us with two easy-to-use methods of testing our chip's functionality. The interactive mode was useful in the early stages of testing, as it allowed us to step through the initialization of the ASIC, controlling all the inputs very carefully. The batch mode provided an easy way to generate the long periods of time needed to test the chip's response to the assertions of IREST, REST, and CN. Each simulation led to error correction and eventually confirmed successful operation of the circuitry.

Figure A: In interactive mode, we see the toggling of CHTOGGLE and subsequent toggling of the output line. Notice how LOAD forces 11110000 into the counter and CHZERO detects 00000000.

Figure B: A close-up of the start up sequence shows the predicted state sequence 0-1-2-3 followed by the count and wait 3-9-3-9 sequence.

Figure C: A close-up of the assertion of CHZERO. Note that after state 5 detects CHZERO, it moves to State 6. CN is not detected there, so the 3-9-3-9 sequence starts anew.

Figure D: This batch mode overview is not entirely realistic; there would be many more toggles of the output line. There is ample evidence here that the IREST line is effective at TEMPO=1; the output line goes low and stays low after IREST goes high. The next note is chosen at 10000.

Figure E: This close-up of the IREST transition shows that the FSM is completely ignorant of the implicit rest, as it continues to follow the 3-9-3-9 and 5-a-5-a sequences for counting.

Figure F: This overview displays that at the slow tempo, the DETECT line properly enables the IREST line. Here we see some harmless delay between assertion of DETECT and assertion of IREST; the implicit rest is useful after almost any arbitrary percentage of the initial note period has elapsed. Most other delays in this circuit are equally harmless.

Figure G: A close-up (TEMPO=0) of the assertion of IREST, which again goes high on the falling edge of DETECT.

Figure H: The Rest Code, 20hex, has been entered, and notice that REST stays high, thereby grounding the output. Again, the FSM continues as if a regular note were playing.

## CONCLUSION

The development of this project is almost complete; the final routing and switch-level simulation remains due to difficulties with the Sun1 operating system. It is anticipated that the chip will fit the TinyChip size constraints (indicated by rules-of-thumb mentioned in lecture), and that the circuit will function as predicted by THOR since the highest frequency in the circuit is a 100kHz external clock (and there are no abrupt logic transitions). Complete simulation was not possible, however, since the thorough monitoring of a $16^{th}$ note's load, play, and stop cycles would require too many time steps. Rather a short artificial note was played quickly and successfully.

The size of the five-bit duration counter and the eight-bit frequency counter limited the number of channels to one. It should be noted that a larger frequency counter could produce more accurate notes; the current note scale suffers some frequency degradation above middle C. Of course these counters could be moved outside of the ASIC and/or shared between channels. Still, it appears that one-channel is the practical limitation for a TinyChip using this design.

Possible improvements for this project include a streamlined controller and algorithm for note-generation, more specific standard cells (i.e. a NAND-gate without the optional inverter) for more efficient space utilization, and a serial data-load scheme to reduce the number of pins per channel. The unused pins in this project could be used for an additional channel (with a different input scheme, or less precise note frequencies/range) or for testing or monitoring.

The difficulty in the design of this chip seemed to lay in organizing this large task and mastering the CAD tools necessary for completion. Later chips would require far less time to take from the "designed-on-paper" stage to the "tested-and-ready-to-go" stage. It was evident that the use of standard cells speeded the completion of this project: the parts were guaranteed to work, the schematic capture of Logic Works was easy to use, and the high-level functions of the standard cells made it easy to go from initial design to implementation.

# APPENDIX I
# (Figures)

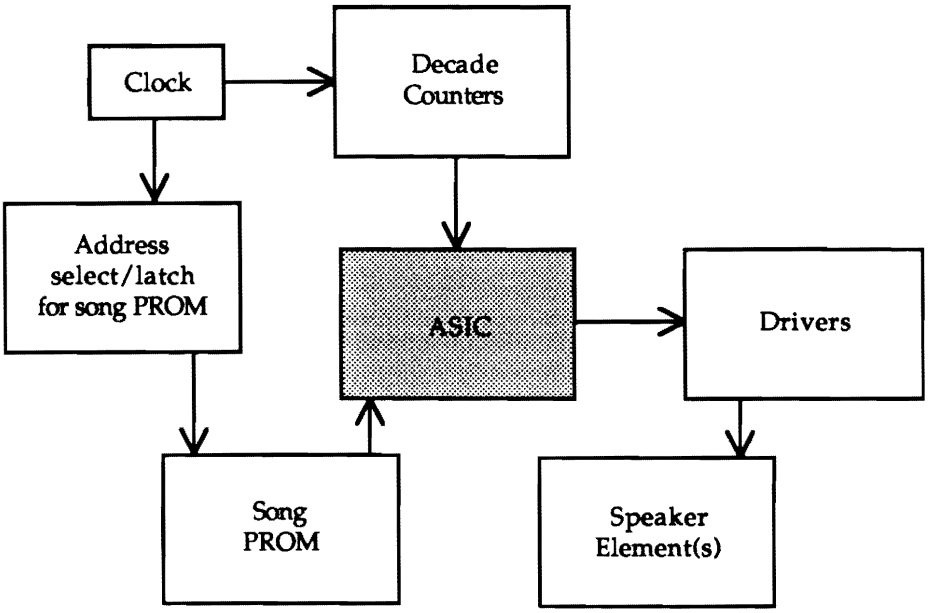



**Figure 1. System Overview**

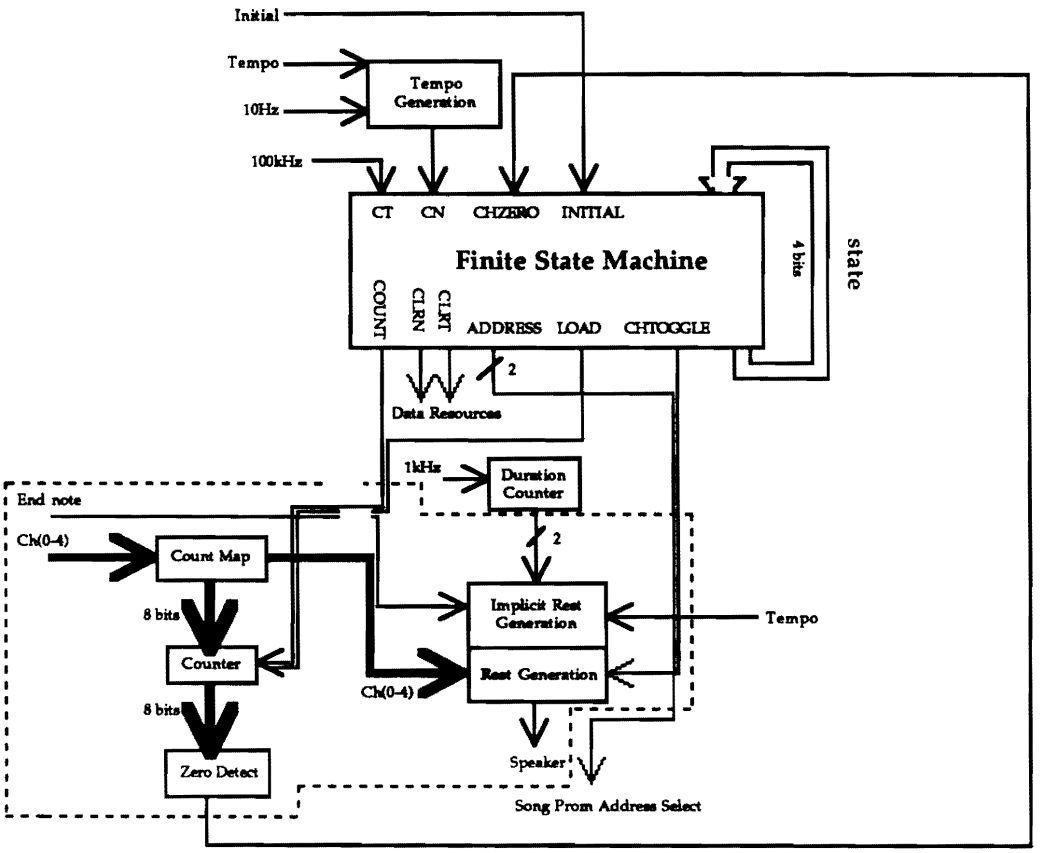



**Figure 2. Chip-level hardware block diagram.**

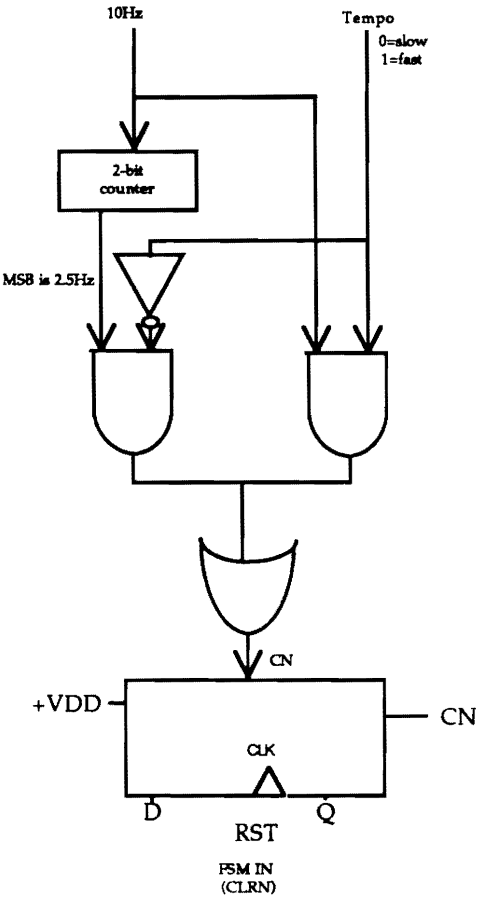



**Figure 3. Tempo Generation Schematic.**

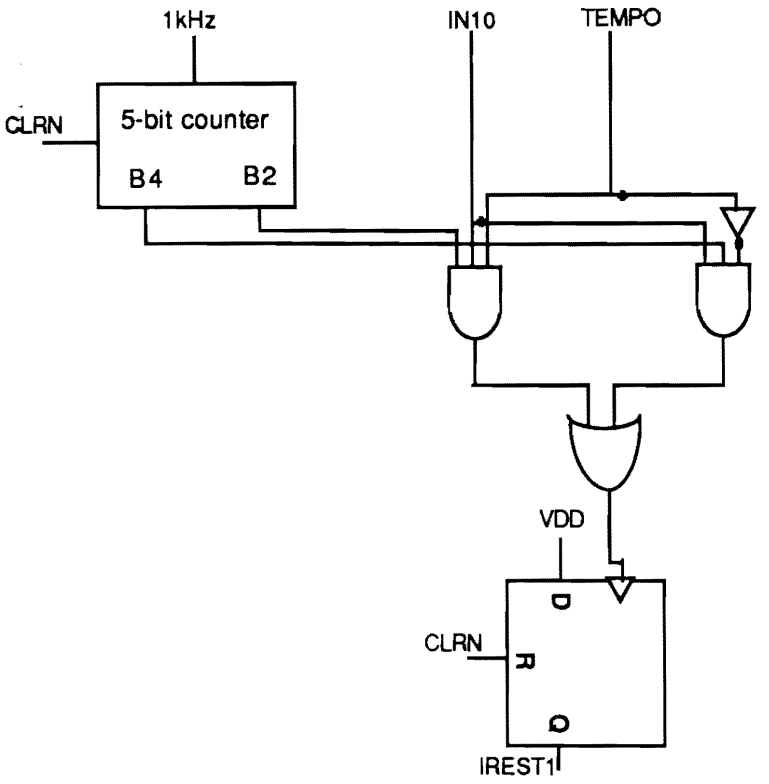



**Figure 4. Implicit Rest Generation Schematic.**

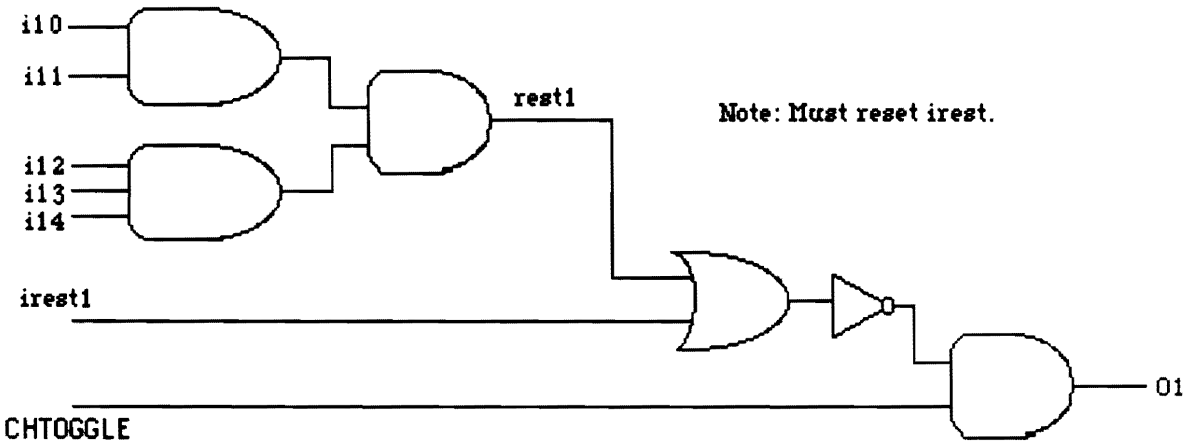



**Figure 5. Rest Generation Schematic.**

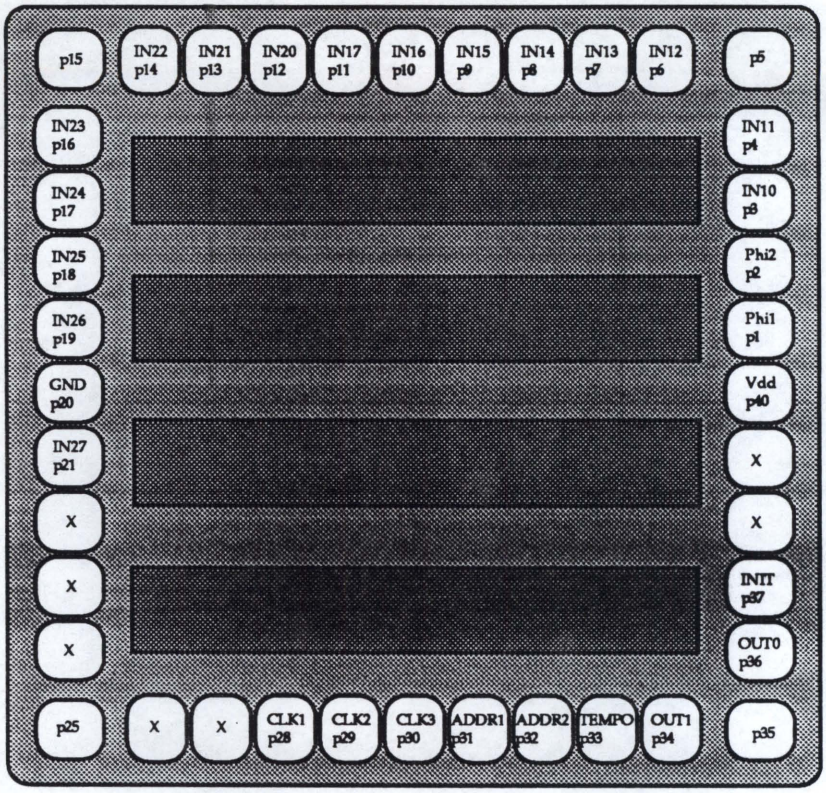


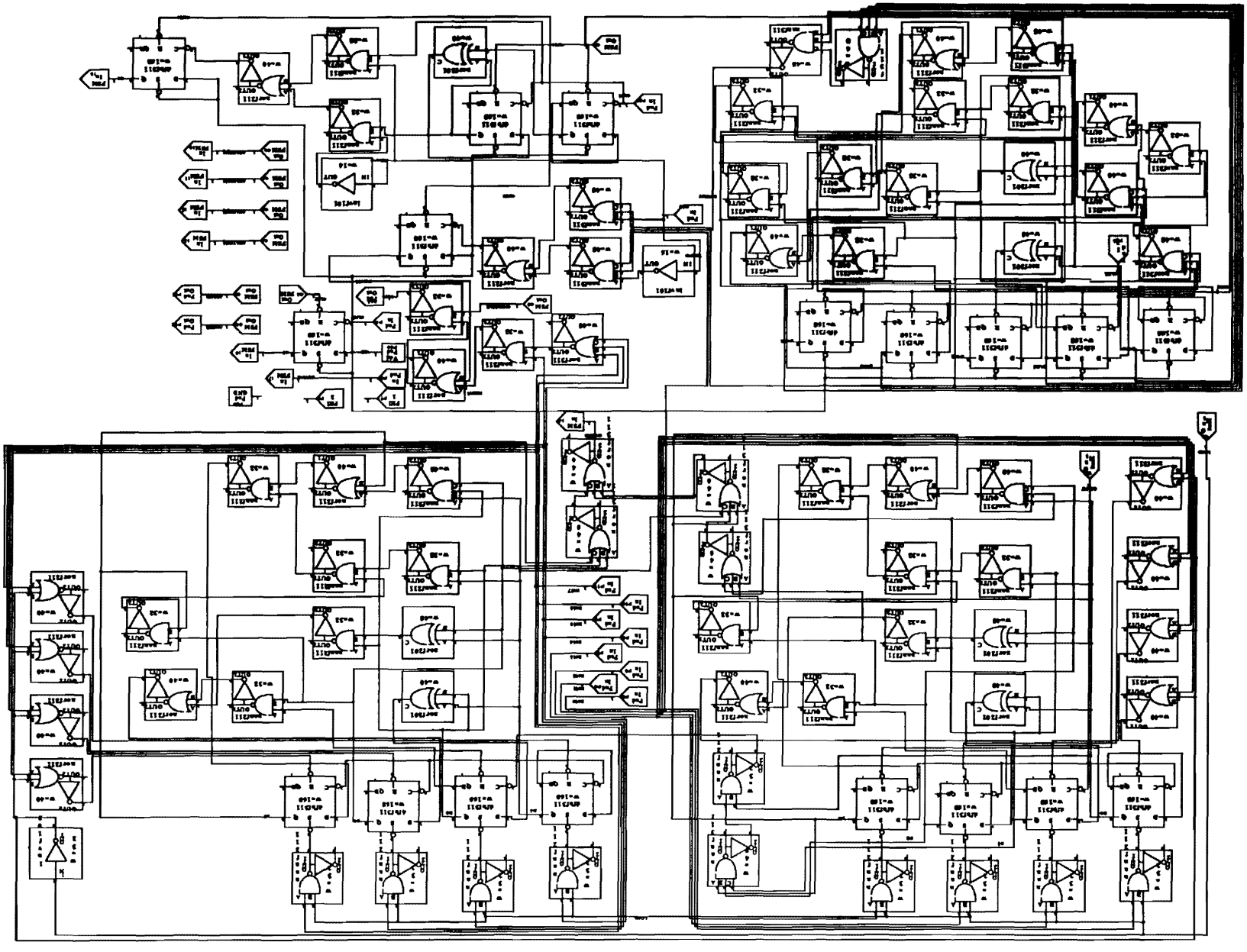

**Figure 6. ASIC Pin Assignment.**

# APPENDIX II
# (CAD files/Simulation results)

```
MODEL CONTROLLER
        ADDRESS<1:0>,           ! FSM 10-9
        LOAD,                   ! FSM 8
        COUNT,                  ! FSM 7
        CHTOGGLE,                       ! FSM 6
        CLRN,                   ! FSM 5
        CLRT,                   ! FSM 4
        NEXTSTATE<3:0>  ! FSM 3-0
=
        INITIAL,                        ! FSM 7
        CT,                             ! FSM 6
        CN,                             ! FSM 5
        CHZERO,                 ! FSM 4
        PRESENTSTATE<3:0>;      ! FSM 3-0
CONSTANT S0 = 0, S1 = 1, S2 = 2, S3 = 3, S4 = 4, S5 = 5, S6 = 6,
                S7 = 7, S8 = 8, S9 = 9, S10 = 10;
ROUTINE MAIN;
        NEXTSTATE = PRESENTSTATE;
        IF INITIAL THEN BEGIN
                NEXTSTATE = S0;
                ADDRESS = 3; LOAD = 0; COUNT = 0;
                CHTOGGLE = 0; CLRN = 0; CLRT = 0;
        END
        ELSE BEGIN
                SELECT PRESENTSTATE FROM
                [ S0 ]: BEGIN
                        ADDRESS = 3 ; LOAD = 0 ; COUNT = 0 ;
                        CHTOGGLE =  0; CLRN =  1; CLRT = 0 ;
                        IF CN THEN BEGIN
                                NEXTSTATE =  S1 ;
                        END
                        ELSE BEGIN
                                NEXTSTATE = S0;
                        END;
                 END;
                [S1]: BEGIN
                        ADDRESS = 2; LOAD = 0; COUNT = 0;
                        CHTOGGLE = 0; CLRN = 0; CLRT = 0;
                        NEXTSTATE = S2;
            END;
                [ S2 ]: BEGIN
                        ADDRESS = 2; LOAD = 1 ; COUNT = 0 ;
                        CHTOGGLE = 0 ; CLRN = 1 ; CLRT = 1 ;

                        IF  CT THEN BEGIN
                                NEXTSTATE =  S3 ;
                        END
                        ELSE BEGIN
                                NEXTSTATE = S2;
                        END;
                 END;
                [S3]: BEGIN
                        ADDRESS = 2 ; LOAD = 0 ; COUNT = 1 ;
                        CHTOGGLE =  0; CLRN = 1 ; CLRT = 0 ;
                        IF CHZERO THEN BEGIN
                                NEXTSTATE = S4  ;
                        END
                        ELSE BEGIN
                                NEXTSTATE = S9;
                        END;
                 END;
                [S4]: BEGIN
                        ADDRESS = 2 ; LOAD = 1 ; COUNT = 0 ;
                        CHTOGGLE = 1 ; CLRN = 1 ; CLRT = 1 ;
                        IF CT THEN BEGIN
                                NEXTSTATE = S5  ;
```

```
                        END
                        ELSE BEGIN
                                NEXTSTATE = S4;
                        END;
                END;
               [S5]: BEGIN
                        ADDRESS = 2 ; LOAD = 0 ; COUNT = 1 ;
                        CHTOGGLE = 1 ; CLRN = 1 ; CLRT = 0 ;
                        IF CHZERO THEN BEGIN
                                NEXTSTATE = S6  ;
                        END
                        ELSE BEGIN
                                NEXTSTATE = S10;
                        END;
                END;
               [S6]: BEGIN
                        ADDRESS = 2 ; LOAD = 0 ; COUNT = 0 ;
                        CHTOGGLE = 1 ; CLRN = 1 ; CLRT = 0 ;
                        IF CN THEN BEGIN
                                NEXTSTATE = S7  ;
                        END
                        ELSE BEGIN
                                NEXTSTATE = S2;
                        END;
                END;
               [S7]: BEGIN
                        ADDRESS = 0 ; LOAD = 0 ; COUNT = 0 ;
                        CHTOGGLE = 1 ; CLRN = 0 ; CLRT = 0 ;
                        NEXTSTATE = S8;
                END;
               [S8]: BEGIN
                        ADDRESS = 1 ; LOAD = 0 ; COUNT = 0 ;
                        CHTOGGLE = 1 ; CLRN = 0 ; CLRT = 0 ;
                        NEXTSTATE = S2;
                END;
               [S9]: BEGIN
                        ADDRESS = 2 ; LOAD =  0; COUNT = 0 ;
                        CHTOGGLE = 0 ; CLRN = 1 ; CLRT = 1 ;
                        IF CT THEN BEGIN
                                NEXTSTATE = S3  ;
                        END
                        ELSE BEGIN
                                NEXTSTATE = S9;
                        END;
                END;
               [S10]: BEGIN
                        ADDRESS = 2 ; LOAD = 0 ; COUNT = 0 ;
                        CHTOGGLE = 1 ; CLRN = 1 ; CLRT = 1 ;
                        IF CT THEN BEGIN
                                NEXTSTATE =  S5 ;
                        END
                        ELSE BEGIN
                                NEXTSTATE = S10;
                        END;
                END;
                ENDSELECT;
        END;
ENDROUTINE;
ENDMODEL;
```



cells

| | |
|---|---|
| D00001 | nanf211 |
| D00002 | nanf211 |
| D00003 | norf211 |
| D00004 | nanf311 |
| D00005 | nanf311 |
| D00006 | norf211 |
| D00007 | invf101 |
| D00008 | dfbf311 |
| D00009 | dfbf311 |
| D00010 | xorf201 |
| D00011 | nanf211 |
| D00012 | nanf211 |
| D00013 | invf101 |
| D00014 | norf211 |
| D00015 | dfbf311 |
| D00016 | dfbf311 |
| D00017 | dfbf311 |
| D00018 | dfbf311 |
| D00019 | xorf201 |
| D00020 | xorf201 |
| D00021 | nanf211 |
| D00022 | nanf211 |
| D00023 | nanf211 |
| D00024 | nanf211 |
| D00025 | nanf211 |
| D00026 | nanf211 |
| D00027 | norf211 |
| D00028 | norf211 |
| D00029 | nanf311 |
| D00030 | dfbf311 |
| D00031 | dfbf311 |
| D00032 | dfbf311 |
| D00033 | dfbf311 |
| D00034 | xorf201 |
| D00035 | xorf201 |
| D00036 | nanf211 |
| D00037 | nanf211 |
| D00038 | nanf211 |
| D00039 | nanf211 |
| D00040 | nanf211 |
| D00041 | nanf211 |
| D00042 | norf211 |
| D00043 | norf211 |
| D00044 | nanf311 |
| D00045 | dfbf311 |
| D00046 | nanf211 |
| D00047 | nanf211 |
| D00048 | nanf211 |
| D00049 | nanf211 |
| D00050 | nanf211 |
| D00051 | nanf211 |
| D00052 | nanf211 |
| D00053 | nanf211 |
| D00054 | nanf311 |
| D00055 | nanf211 |
| D00056 | norf211 |
| D00057 | norf211 |
| D00058 | norf211 |
| D00059 | norf211 |
| D00060 | norf211 |
| D00061 | norf211 |
| D00062 | norf211 |
| D00063 | norf211 |
| D00064 | invf103 |
| D00065 | norf311 |

```
D00066  norf311
D00067  norf311
D00068  norf311
D00069  dfbf311
D00070  dfbf311
D00071  dfbf311
D00072  dfbf311
D00073  xorf201
D00074  xorf201
D00075  nanf211
D00076  nanf211
D00077  nanf211
D00078  nanf211
D00079  nanf211
D00080  nanf211
D00081  norf211
D00082  norf211
D00083  nanf311
D00084  dfbf311
D00085  dfbf311
D00086  nanf311
D00087  nanf311
D00088  nanf211
D00089  norf211
D00090  nanf211
D00091  dfbf311
D00092  norf311
D00093  nanf311
D00094  nanf311
I0      FSMIn
I1      FSMIn
I2      FSMIn
I3      FSMIn
I4      FSMIn
I5      FSMIn
I6      FSMIn
I7      FSMIn
O0      FSMOut
O1      FSMOut
O2      FSMOut
O3      FSMOut
O4      FSMOut
O5      FSMOut
O6      FSMOut
O7      FSMOut
O8      FSMOut
O9      FSMOut
O10     FSMOut
P1      PHI1
P2      PHI2
P3      PadIn
P4      PadIn
P6      PadIn
P7      PadIn
P8      PadIn
P9      PadIn
P10     PadIn
P11     PadIn
P20     PadGND
P28     PadIn
P29     PadIn
P30     PadIn
P31     PadOut
P32     PadOut
P33     PadIn
P36     PadOut
```

```
P37     PadIn
P40     PadVdd
nets
ADDR1 O9-1 P31-1
ADDR2 O10-1 P32-1
B0 D00018-5 D00019-1 D00020-1 D00022-1 D00029-1 D00055-2
D00065-1
B1 D00019-2 D00021-1 D00022-2 D00029-2 D00030-5 D00054-3
D00065-2
B2 D00016-5 D00020-2 D00024-2 D00025-1 D00029-3 D00054-2
D00065-3
B3 D00017-5 D00025-2 D00028-2 D00054-1 D00066-1
B4 D00033-5 D00034-1 D00035-1 D00037-1 D00044-1 D00066-3
B5 D00034-2 D00036-1 D00037-2 D00044-2 D00045-5 D00067-2
B6 D00031-5 D00035-2 D00039-2 D00040-1 D00044-3 D00067-1
B7 D00032-5 D00040-2 D00043-2 D00067-3
CHTOGGLE D00002-2 O6-1
CHZERO D00068-5 I4-1
CLK1 D00008-2 D00009-2 D00012-2 P28-1
CLK2 D00070-2 D00071-2 D00072-2 D00084-2 D00085-2 P29-1
CLK3 D00069-2 P30-1
CLRN D00008-3 D00009-3 D00015-3 D00070-3 D00071-3 D00072-3
D00084-3 D00085-3 D00091-3 O5-1
CLRT D00069-3 O4-1
CN D00015-5 I5-1
COUNT D00016-2 D00017-2 D00018-2 D00030-2 O7-1
CT D00069-5 I6-1
DETECT D00004-3 D00094-4
DUR0 D00072-5 D00073-1 D00074-1 D00076-1 D00083-1 D00086-1
D00094-2
DUR1 D00073-2 D00075-1 D00076-2 D00083-2 D00084-5 D00086-2
D00094-1
DUR2 D00005-3 D00070-5 D00074-2 D00078-2 D00079-1 D00083-3 D00086-3
D00093-3
DUR3 D00071-5 D00079-2 D00082-2 D00087-2 D00088-1
D00093-2
DUR4 D00085-5 D00087-3 D00089-2 D00093-1
IN10 D00004-2 D00005-2 D00046-1 D00056-1 P3-1
IN11 D00053-1 D00057-1 P4-1
IN12 D00047-1 D00058-1 P6-1
IN13 D00048-1 D00059-1 P7-1
IN14 D00049-1 D00063-1 D00092-3 P8-1
IN15 D00001-1 D00052-1 D00062-1 P9-1
IN16 D00050-1 D00061-1 D00092-2 P10-1
IN17 D00051-1 D00060-1 D00092-1 P11-1
INITIAL I7-1 P37-1
IREST1 D00003-2 D00091-5
LOAD D00046-2 D00047-2 D00048-2 D00049-2 D00050-2 D00051-2
D00052-2 D00053-2 D00064-1 O8-1
OUT0 D00002-3 P36-1
REST1 D00001-3 D00003-1
S00001 D00002-1 D00003-4
S00002 D00004-4 D00006-1
S00003 D00004-1 D00007-2
S00004 D00005-4 D00006-2
S00005 D00008-5 D00010-2
S00006 D00008-1 D00008-4
S00007 D00009-5 D00010-1 D00011-2
S00008 D00009-1 D00010-3
S00009 D00011-3 D00014-1
S00010 D00012-3 D00014-2
S00011 D00011-1 D00013-2
S00012 D00014-3 D00015-2
S00013 D00019-3 D00030-1
S00014 D00020-3 D00021-2
S00015 D00021-3 D00027-1
```

S00016 D00022-3 D00023-2
S00017 D00023-4 D00026-1
S00018 D00024-3 D00027-2
S00019 D00023-1 D00025-3
S00020 D00016-1 D00027-3
S00021 D00026-2 D00028-3
S00022 D00028-1 D00029-4
S00023 D00024-1 D00030-4
S00024 D00018-1 D00018-4
S00025 D00034-3 D00045-1
S00026 D00035-3 D00036-2
S00027 D00036-3 D00042-1
S00028 D00037-3 D00038-2
S00029 D00038-4 D00041-1
S00030 D00039-3 D00042-2
S00031 D00038-1 D00040-3
S00032 D00032-1 D00041-3
S00033 D00031-1 D00042-3
S00034 D00041-2 D00043-3
S00035 D00043-1 D00044-4
S00036 D00039-1 D00045-4
S00037 D00033-1 D00033-4
S00038 D00017-1 D00026-3
S00039 D00017-6 D00048-4
S00040 D00018-6 D00046-4
S00041 D00016-6 D00047-4
S00042 D00030-6 D00053-4
S00043 D00031-6 D00050-4
S00044 D00033-6 D00049-4
S00045 D00045-6 D00052-4
S00046 D00032-6 D00051-4
S00047 D00031-2 D00032-2 D00033-2 D00045-2 D00055-3
S00048 D00054-4 D00055-1
S00049 D00018-3 D00056-3
S00050 D00030-3 D00057-3
S00051 D00016-3 D00058-3
S00052 D00017-3 D00059-3
S00053 D00032-3 D00060-3
S00054 D00033-3 D00063-3
S00055 D00031-3 D00061-3
S00056 D00045-3 D00062-3
S00057 D00056-2 D00057-2 D00058-2 D00059-2 D00060-2
D00061-2 D00062-2 D00063-2 D00064-2
S00058 D00065-4 D00066-2
S00059 D00066-4 D00068-1 D00068-2
S00060 D00067-4 D00068-3
S00061 D00073-3 D00084-1
S00062 D00074-3 D00075-2
S00063 D00075-3 D00081-1
S00064 D00076-3 D00077-2
S00065 D00077-4 D00080-1
S00066 D00078-3 D00081-2
S00067 D00077-1 D00079-3
S00068 D00070-1 D00081-3
S00069 D00080-2 D00082-3
S00070 D00082-1 D00083-4
S00071 D00078-1 D00084-4
S00072 D00072-1 D00072-4
S00073 D00071-1 D00080-3
S00074 D00086-4 D00087-1 D00088-2
S00075 D00089-3 D00090-2
S00076 D00088-3 D00089-1
S00077 D00085-1 D00090-3
S00078 D00006-3 D00091-2
S00079 D00001-2 D00092-5
S00080 D00093-4 D00094-3

S00081 D00087-5 D00090-1
STATE[0] I0-1 O0-1
STATE[1] I1-1 O1-1
STATE[2] I2-1 O2-1
STATE[3] I3-1 O3-1
TEMPO D00005-1 D00007-1 D00012-1 D00013-1 P33-1
VDD D00008-6 D00009-6 D00015-1 D00015-6 D00069-1 D00069-6
D00070-6 D00071-6 D00072-6 D00084-6 D00085-6 D00091-1
D00091-6 P40-1

```
(g = ONE) (n = vdd) (o=Vdd);
(g = ZERO) (n = gnd)  (o = GND);
(g = ONE) (n  = IN10)  (o = IN10);
(g = ZERO) (n = IN11) (o = IN11);
(g  = ZERO) (n=IN12) (o=IN12);
(g=ZERO) (n=IN13) (o=IN13);
(g=ONE) (n=IN14) (o=IN14);
(g=ONE)  (n=IN15) (o=IN15);
(g=ONE)  (n=IN16) (o=IN16);
(g=ONE) (n=IN17) (o=IN17);
(g=ONE)  (n=TEMPO)  (o=TEMPO);
(g =  CLOCK) (n=INITIAL) (o=INITIAL) (s=3)  (vs=0,10,11000);
(g = CLOCK) (n = CL_PHI1) (o = PHI1) (s = 3) (vs = 0,1,4);
(g = CLOCK) (n = CL_PHI2) (o = PHI2) (s = 3) (vs = 2,1,4);
(g = CLOCK) (n = CLK1) (o = CLK1) (s = 3) (vs = 40,40,10500);
(g = CLOCK) (n = CLK2) (o = CLK2) (s = 3) (vs = 0,500,1000);
(g = CLOCK) (n = CLK3) (o = CLK3) (s = 3) (vs = 0,5,10);
(m = HEXOUT) (n = STATE) (i = STATE[3-0]);
(m = BINOUT) (n = CLRT) (i = CLRT);
(m = BINOUT) (n = CLRN) (i = CLRN);
(m = BINOUT) (n = CHTOGGLE) (i = CHTOGGLE);
(m = BINOUT) (n = COUNT) (i = COUNT);
(m = BINOUT) (n = LOAD) (i = LOAD);

(f=nanf211)(n=D00001)
    (i= IN15,
        S00079)
    (do=0,0)
    (o= unc,
        REST1)
        ;

(f=nanf211)(n=D00002)
    (i= S00001,
        CHTOGGLE)
    (do=0,0)
    (o= unc,
        OUT0)
        ;

(f=nanf211)(n=D00011)
    (i= S00011,
        S00007)
    (do=0,0)
    (o= unc,
        S00009)
        ;

(f=nanf211)(n=D00012)
    (i= TEMPO,
        CLK1)
    (do=0,0)
    (o= unc,
        S00010)
        ;

(f=nanf211)(n=D00021)
    (i= B1,
        S00014)
    (do=0,0)
    (o= unc,
        S00015)
        ;

(f=nanf211)(n=D00022)
    (i= B0,
```

```
        B1)
    (do=0,0)
    (o= unc,
        S00016)
        ;

(f=nanf211)(n=D00023)
    (i= S00019,
        S00016)
    (do=0,0)
    (o= S00017,
        unc)
        ;

(f=nanf211)(n=D00024)
    (i= S00023,
        B2)
    (do=0,0)
    (o= unc,
        S00018)
        ;

(f=nanf211)(n=D00025)
    (i= B2,
        B3)
    (do=0,0)
    (o= unc,
        S00019)
        ;

(f=nanf211)(n=D00026)
    (i= S00017,
        S00021)
    (do=0,0)
    (o= unc,
        S00038)
        ;

(f=nanf211)(n=D00036)
    (i= B5,
        S00026)
    (do=0,0)
    (o= unc,
        S00027)
        ;

(f=nanf211)(n=D00037)
    (i= B4,
        B5)
    (do=0,0)
    (o= unc,
        S00028)
        ;

(f=nanf211)(n=D00038)
    (i= S00031,
        S00028)
    (do=0,0)
    (o= S00029,
        unc)
        ;

(f=nanf211)(n=D00039)
    (i= S00036,
        B6)
    (do=0,0)
```

```
    (o= unc,
        S00030)
        ;

(f=nanf211)(n=D00040)
    (i= B6,
        B7)
    (do=0,0)
    (o= unc,
        S00031)
        ;

(f=nanf211)(n=D00041)
    (i= S00029,
        S00034)
    (do=0,0)
    (o= unc,
        S00032)
        ;

(f=nanf211)(n=D00046)
    (i= IN10,
        LOAD)
    (do=0,0)
    (o= S00040,
        unc)
        ;

(f=nanf211)(n=D00047)
    (i= IN12,
        LOAD)
    (do=0,0)
    (o= S00041,
        unc)
        ;

(f=nanf211)(n=D00048)
    (i= IN13,
        LOAD)
    (do=0,0)
    (o= S00039,
        unc)
        ;

(f=nanf211)(n=D00049)
    (i= IN14,
        LOAD)
    (do=0,0)
    (o= S00044,
        unc)
        ;

(f=nanf211)(n=D00050)
    (i= IN16,
        LOAD)
    (do=0,0)
    (o= S00043,
        unc)
        ;

(f=nanf211)(n=D00051)
    (i= IN17,
        LOAD)
    (do=0,0)
    (o= S00046,
        unc)
```

```
        ;

(f=nanf211)(n=D00052)
    (i= IN15,
        LOAD)
    (do=0,0)
    (o= S00045,
        unc)
        ;

(f=nanf211)(n=D00053)
    (i= IN11,
        LOAD)
    (do=0,0)
    (o= S00042,
        unc)
        ;

(f=nanf211)(n=D00055)
    (i= S00048,
        B0)
    (do=0,0)
    (o= unc,
        S00047)
        ;

(f=nanf211)(n=D00075)
    (i= DUR1,
        S00062)
    (do=0,0)
    (o= unc,
        S00063)
        ;

(f=nanf211)(n=D00076)
    (i= DUR0,
        DUR1)
    (do=0,0)
    (o= unc,
        S00064)
        ;

(f=nanf211)(n=D00077)
    (i= S00067,
        S00064)
    (do=0,0)
    (o= S00065,
        unc)
        ;

(f=nanf211)(n=D00078)
    (i= S00071,
        DUR2)
    (do=0,0)
    (o= unc,
        S00066)
        ;

(f=nanf211)(n=D00079)
    (i= DUR2,
        DUR3)
    (do=0,0)
    (o= unc,
        S00067)
        ;
```

```
(f=nanf211)(n=D00080)
    (i= S00065,
        S00069)
    (do=0,0)
    (o= unc,
        S00073)
        ;

(f=nanf211)(n=D00088)
    (i= DUR3,
        S00074)
    (do=0,0)
    (o= unc,
        S00076)
        ;

(f=nanf211)(n=D00090)
    (i= S00081,
        S00075)
    (do=0,0)
    (o= unc,
        S00077)
        ;

(f=norf211)(n=D00003)
    (i= REST1,
        IREST1)
    (do=0,0)
    (o= unc,
        S00001)
        ;

(f=norf211)(n=D00006)
    (i= S00002,
        S00004)
    (do=0,0)
    (o= S00078,
        unc)
        ;

(f=norf211)(n=D00014)
    (i= S00009,
        S00010)
    (do=0,0)
    (o= S00012,
        unc)
        ;

(f=norf211)(n=D00027)
    (i= S00015,
        S00018)
    (do=0,0)
    (o= S00020,
        unc)
        ;

(f=norf211)(n=D00028)
    (i= S00022,
        B3)
    (do=0,0)
    (o= S00021,
        unc)
        ;

(f=norf211)(n=D00042)
    (i= S00027,
```

```
        S00030)
    (do=0,0)
    (o= S00033,
        unc)
        ;

(f=norf211)(n=D00043)
    (i= S00035,
        B7)
    (do=0,0)
    (o= S00034,
        unc)
        ;

(f=norf211)(n=D00056)
    (i= IN10,
        S00057)
    (do=0,0)
    (o= S00049,
        unc)
        ;

(f=norf211)(n=D00057)
    (i= IN11,
        S00057)
    (do=0,0)
    (o= S00050,
        unc)
        ;

(f=norf211)(n=D00058)
    (i= IN12,
        S00057)
    (do=0,0)
    (o= S00051,
        unc)
        ;

(f=norf211)(n=D00059)
    (i= IN13,
        S00057)
    (do=0,0)
    (o= S00052,
        unc)
        ;

(f=norf211)(n=D00060)
    (i= IN17,
        S00057)
    (do=0,0)
    (o= S00053,
        unc)
        ;

(f=norf211)(n=D00061)
    (i= IN16,
        S00057)
    (do=0,0)
    (o= S00055,
        unc)
        ;

(f=norf211)(n=D00062)
    (i= IN15,
        S00057)
    (do=0,0)
```

```
    (o= S00056,
        unc)
        ;

(f=norf211)(n=D00063)
    (i= IN14,
        S00057)
    (do=0,0)
    (o= S00054,
        unc)
        ;

(f=norf211)(n=D00081)
    (i= S00063,
        S00066)
    (do=0,0)
    (o= S00068,
        unc)
        ;

(f=norf211)(n=D00082)
    (i= S00070,
        DUR3)
    (do=0,0)
    (o= S00069,
        unc)
        ;

(f=norf211)(n=D00089)
    (i= S00076,
        DUR4)
    (do=0,0)
    (o= S00075,
        unc)
        ;

(f=nanf311)(n=D00004)
    (i= S00003,
        IN10,
        DETECT)
    (do=0,0)
    (o= unc,
        S00002)
        ;

(f=nanf311)(n=D00005)
    (i= TEMPO,
        IN10,
        DUR2)
    (do=0,0)
    (o= unc,
        S00004)
        ;

(f=nanf311)(n=D00029)
    (i= B0,
        B1,
        B2)
    (do=0,0)
    (o= unc,
        S00022)
        ;

(f=nanf311)(n=D00044)
    (i= B4,
        B5,
```

```
        B6)
    (do=0,0)
    (o= unc,
        S00035)
        ;

(f=nanf311)(n=D00054)
    (i= B3,
        B2,
        B1)
    (do=0,0)
    (o= unc,
        S00048)
        ;

(f=nanf311)(n=D00083)
    (i= DUR0,
        DUR1,
        DUR2)
    (do=0,0)
    (o= unc,
        S00070)
        ;

(f=nanf311)(n=D00086)
    (i= DUR0,
        DUR1,
        DUR2)
    (do=0,0)
    (o= unc,
        S00074)
        ;

(f=nanf311)(n=D00087)
    (i= S00074,
        DUR3,
        DUR4)
    (do=0,0)
    (o= S00081,
        unc)
        ;

(f=nanf311)(n=D00093)
    (i= DUR4,
        DUR3,
        DUR2)
    (do=0,0)
    (o= unc,
        S00080)
        ;

(f=nanf311)(n=D00094)
    (i= DUR1,
        DUR0,
        S00080)
    (do=0,0)
    (o= unc,
        DETECT)
        ;

(f=invf101)(n=D00007)
    (i= TEMPO)
    (do=0)
    (o= S00003)
        ;
```

```
(f=invf101)(n=D00013)
    (i= TEMPO)
    (do=0)
    (o= S00011)
        ;

(f=dfbf311)(n=D00008)
    (i= S00006,
        CLK1,
        CLRN,
        Vdd)
    (do=0,0)
    (o= S00005,
        S00006)
    (s= 3)
        ;

(f=dfbf311)(n=D00009)
    (i= S00008,
        CLK1,
        CLRN,
        Vdd)
    (do=0,0)
    (o= S00007,
        unc)
    (s= 3)
        ;

(f=dfbf311)(n=D00015)
    (i= Vdd,
        S00012,
        CLRN,
        Vdd)
    (do=0,0)
    (o= CN,
        unc)
    (s= 3)
        ;

(f=dfbf311)(n=D00016)
    (i= S00020,
        COUNT,
        S00051,
        S00041)
    (do=0,0)
    (o= B2,
        unc)
    (s= 3)
        ;

(f=dfbf311)(n=D00017)
    (i= S00038,
        COUNT,
        S00052,
        S00039)
    (do=0,0)
    (o= B3,
        unc)
    (s= 3)
        ;

(f=dfbf311)(n=D00018)
    (i= S00024,
        COUNT,
        S00049,
        S00040)
```

```
    (do=0,0)
    (o= B0,
        S00024)
    (s= 3)
        ;

(f=dfbf311)(n=D00030)
    (i= S00013,
        COUNT,
        S00050,
        S00042)
    (do=0,0)
    (o= B1,
        S00023)
    (s= 3)
        ;

(f=dfbf311)(n=D00031)
    (i= S00033,
        S00047,
        S00055,
        S00043)
    (do=0,0)
    (o= B6,
        unc)
    (s= 3)
        ;

(f=dfbf311)(n=D00032)
    (i= S00032,
        S00047,
        S00053,
        S00046)
    (do=0,0)
    (o= B7,
        unc)
    (s= 3)
        ;

(f=dfbf311)(n=D00033)
    (i= S00037,
        S00047,
        S00054,
        S00044)
    (do=0,0)
    (o= B4,
        S00037)
    (s= 3)
        ;

(f=dfbf311)(n=D00045)
    (i= S00025,
        S00047,
        S00056,
        S00045)
    (do=0,0)
    (o= B5,
        S00036)
    (s= 3)
        ;

(f=dfbf311)(n=D00069)
    (i= Vdd,
        CLK3,
        CLRT,
        Vdd)
```

```
    (do=0,0)
    (o= CT,
        unc)
    (s= 3)
        ;

(f=dfbf311)(n=D00070)
    (i= S00068,
        CLK2,
        CLRN,
        Vdd)
    (do=0,0)
    (o= DUR2,
        unc)
    (s= 3)
        ;

(f=dfbf311)(n=D00071)
    (i= S00073,
        CLK2,
        CLRN,
        Vdd)
    (do=0,0)
    (o= DUR3,
        unc)
    (s= 3)
        ;

(f=dfbf311)(n=D00072)
    (i= S00072,
        CLK2,
        CLRN,
        Vdd)
    (do=0,0)
    (o= DUR0,
        S00072)
    (s= 3)
        ;

(f=dfbf311)(n=D00084)
    (i= S00061,
        CLK2,
        CLRN,
        Vdd)
    (do=0,0)
    (o= DUR1,
        S00071)
    (s= 3)
        ;

(f=dfbf311)(n=D00085)
    (i= S00077,
        CLK2,
        CLRN,
        Vdd)
    (do=0,0)
    (o= DUR4,
        unc)
    (s= 3)
        ;

(f=dfbf311)(n=D00091)
    (i= Vdd,
        S00078,
        CLRN,
        Vdd)
```

```
    (do=0,0)
    (o= IREST1,
        unc)
    (s= 3)
        ;

(f=xorf201)(n=D00010)
    (i= S00007,
        S00005)
    (do=0)
    (o= S00008)
        ;

(f=xorf201)(n=D00019)
    (i= B0,
        B1)
    (do=0)
    (o= S00013)
        ;

(f=xorf201)(n=D00020)
    (i= B0,
        B2)
    (do=0)
    (o= S00014)
        ;

(f=xorf201)(n=D00034)
    (i= B4,
        B5)
    (do=0)
    (o= S00025)
        ;

(f=xorf201)(n=D00035)
    (i= B4,
        B6)
    (do=0)
    (o= S00026)
        ;

(f=xorf201)(n=D00073)
    (i= DUR0,
        DUR1)
    (do=0)
    (o= S00061)
        ;

(f=xorf201)(n=D00074)
    (i= DUR0,
        DUR2)
    (do=0)
    (o= S00062)
        ;

(f=invf103)(n=D00064)
    (i= LOAD)
    (do=0)
    (o= S00057)
        ;

(f=norf311)(n=D00065)
    (i= B0,
        B1,
        B2)
    (do=0,0)
```

```
    (o= S00058,
        unc)
        ;

(f=norf311)(n=D00066)
    (i= B3,
        S00058,
        B4)
    (do=0,0)
    (o= S00059,
        unc)
        ;

(f=norf311)(n=D00067)
    (i= B6,
        B5,
        B7)
    (do=0,0)
    (o= S00060,
        unc)
        ;

(f=norf311)(n=D00068)
    (i= S00059,
        S00059,
        S00060)
    (do=0,0)
    (o= unc,
        CHZERO)
        ;

(f=norf311)(n=D00092)
    (i= IN17,
        IN16,
        IN14)
    (do=0,0)
    (o= unc,
        S00079)
        ;

(f=fri3_core_controller_FSM)(n=FSM)
    (i= PHI1,
        PHI2,
        STATE[0-3],
        CHZERO,
        CN,
        CT,
        INITIAL)
    (do=0,0,0,0,0,0,0,0,0,0,
        0)
    (o= STATE[0-3],
        CLRT,
        CLRN,
        CHTOGGLE,
        COUNT,
        LOAD,
        ADDR1,
        ADDR2)
    (s= 8)
        ;

(m=analyzer)(n=scope)
(i=     ADDR1,
        ADDR2,
        OUT0,
        B0,
```

```
        B1,
        B2,
        B3,
        B4,
        B5,
        B6,
        B7,
        DUR0,
        DUR1,
        DUR2,
        DUR3,
        DUR4,
        DETECT,
        LOAD,
        COUNT,
        CHTOGGLE,
        CN,
        CLRN,
        CLRT,
        CHZERO,
        IREST1,
        REST1,
        STATE[3-0])
    (s= 3)
        ;
```

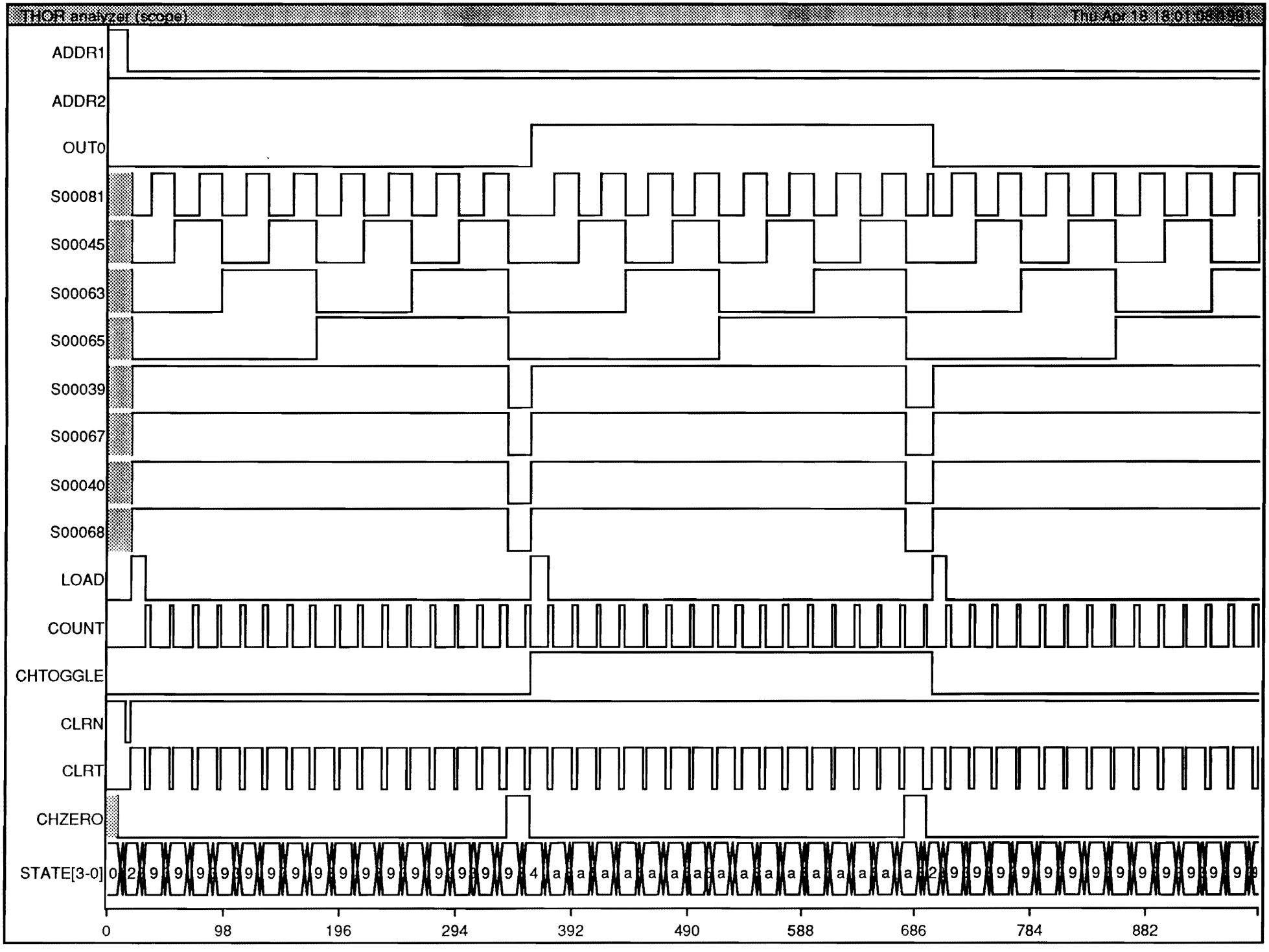

THOR analyzer (scope)
Thu Apr 18 18:01:08 1991
ADDR1
ADDR2
OUT0
S00081
S00045
S00063
S00065
S00039
S00067
S00040
S00068
LOAD
COUNT
CHTOGGLE
CLRN
CLRT
CHZERO
STATE[3-0]
0
98
196
294
392
490
588
686
784
882

FIGURE A

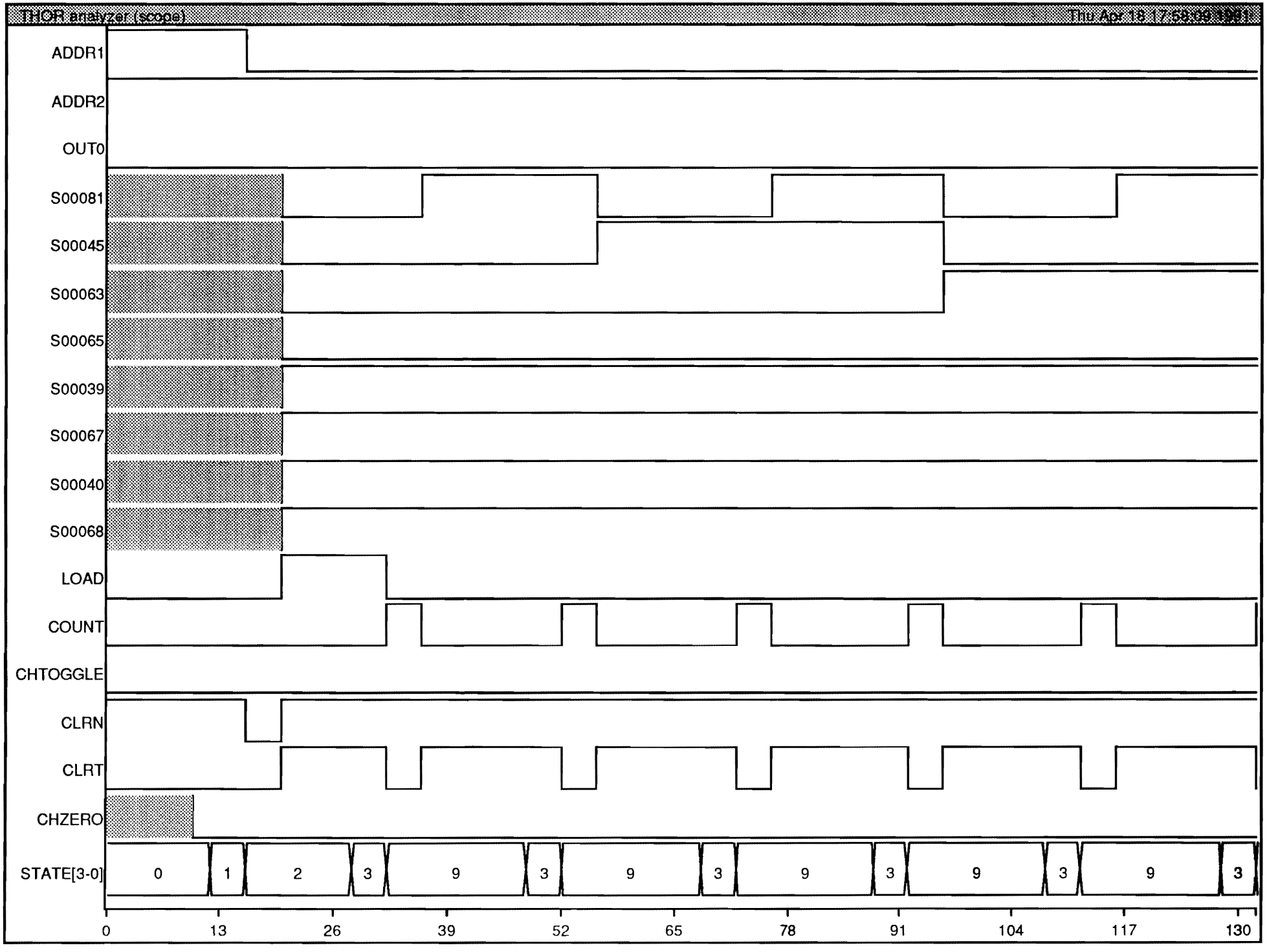
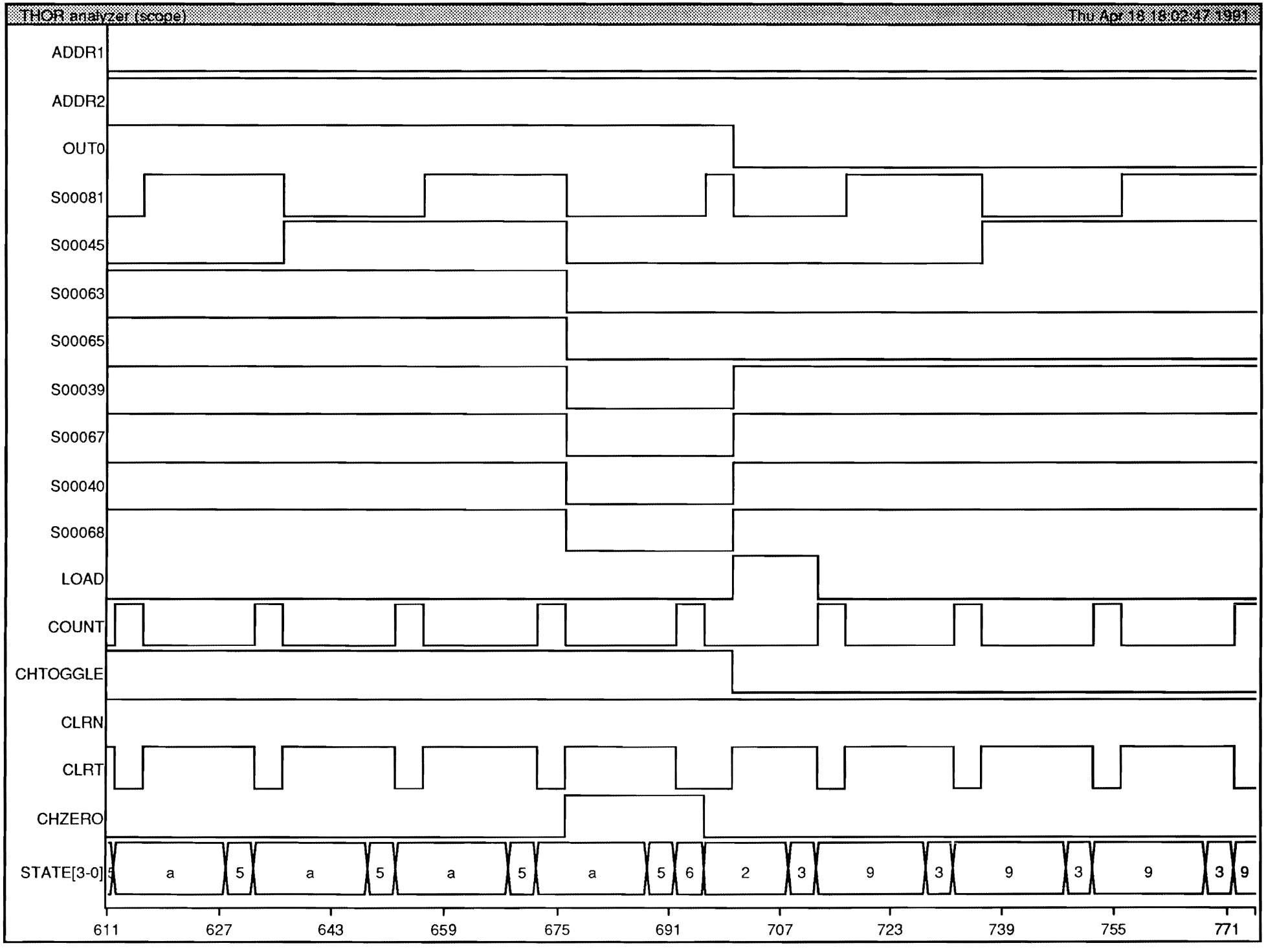

THOR analyzer (scope)
Thu Apr 18 17:59:09 1991
ADDR1
ADDR2
OUT0
S00081
S00045
S00063
S00065
S00039
S00067
S00040
S00068
LOAD
COUNT
CHTOGGLE
CLRN
CLRT
CHZERO
STATE[3-0]
0 1 2 3 9 3 9 3 9 3 9 3 9 3
0 13 26 39 52 65 78 91 104 117 130

FIGURE 8

FIGURE C


THOR analyzer (scope)
Thu Apr 18 18:02:47 1991
ADDR1
ADDR2
OUT0
S00081
S00045
S00063
S00065
S00039
S00067
S00040
S00068
LOAD
COUNT
CHTOGGLE
CLRN
CLRT
CHZERO
STATE[3-0]
a
5
6
2
3
9
611
627
643
659
675
691
707
723
739
755
771

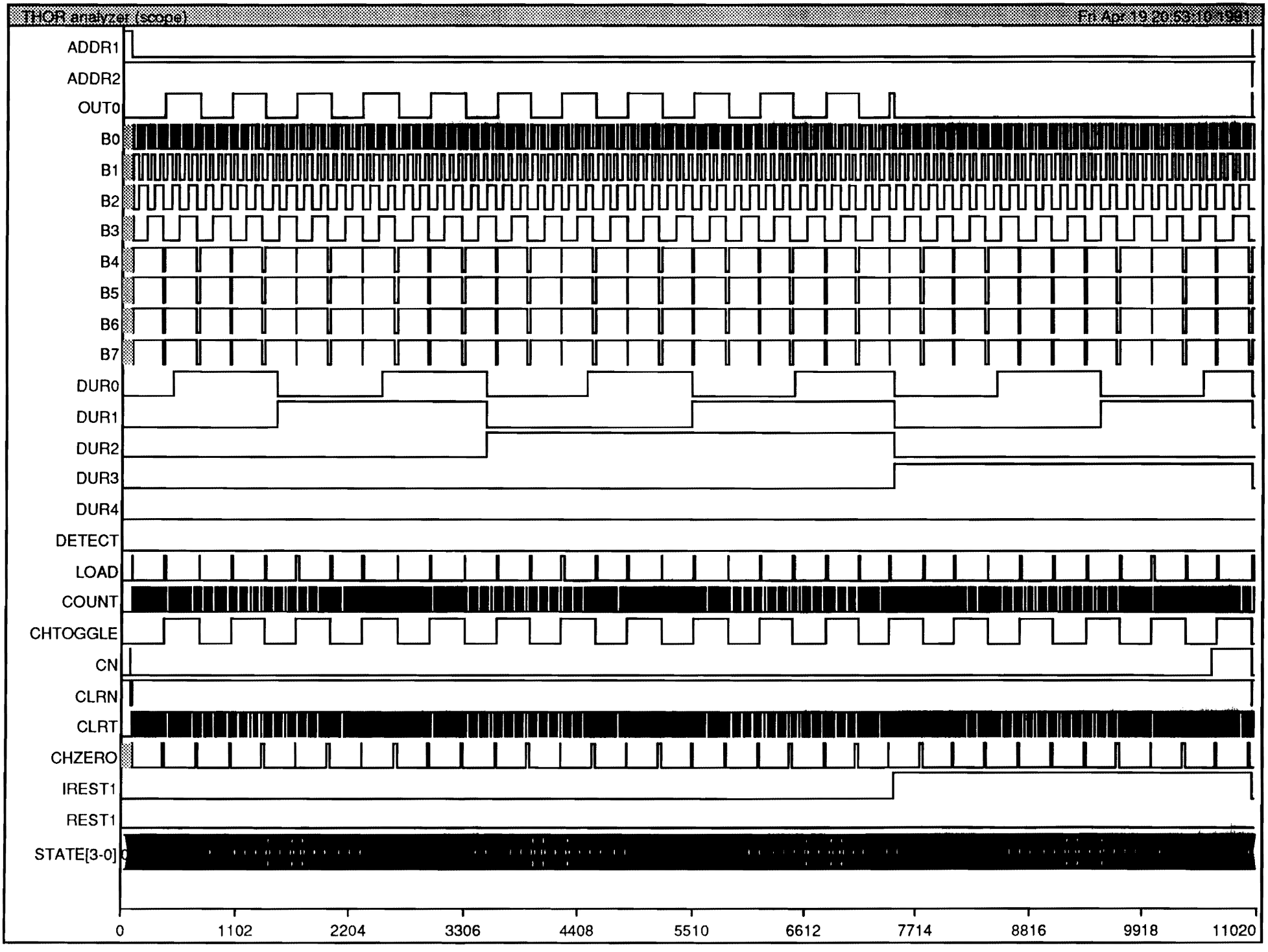

THOR analyzer (scope)
Fri Apr 19 20:59:10 1991
ADDR1
ADDR2
OUT0
B0
B1
B2
B3
B4
B5
B6
B7
DUR0
DUR1
DUR2
DUR3
DUR4
DETECT
LOAD
COUNT
CHTOGGLE
CN
CLRN
CLRT
CHZERO
IREST1
REST1
STATE[3-0]
0
1102
2204
3306
4408
5510
6612
7714
8816
9918
11020

FIGURE D

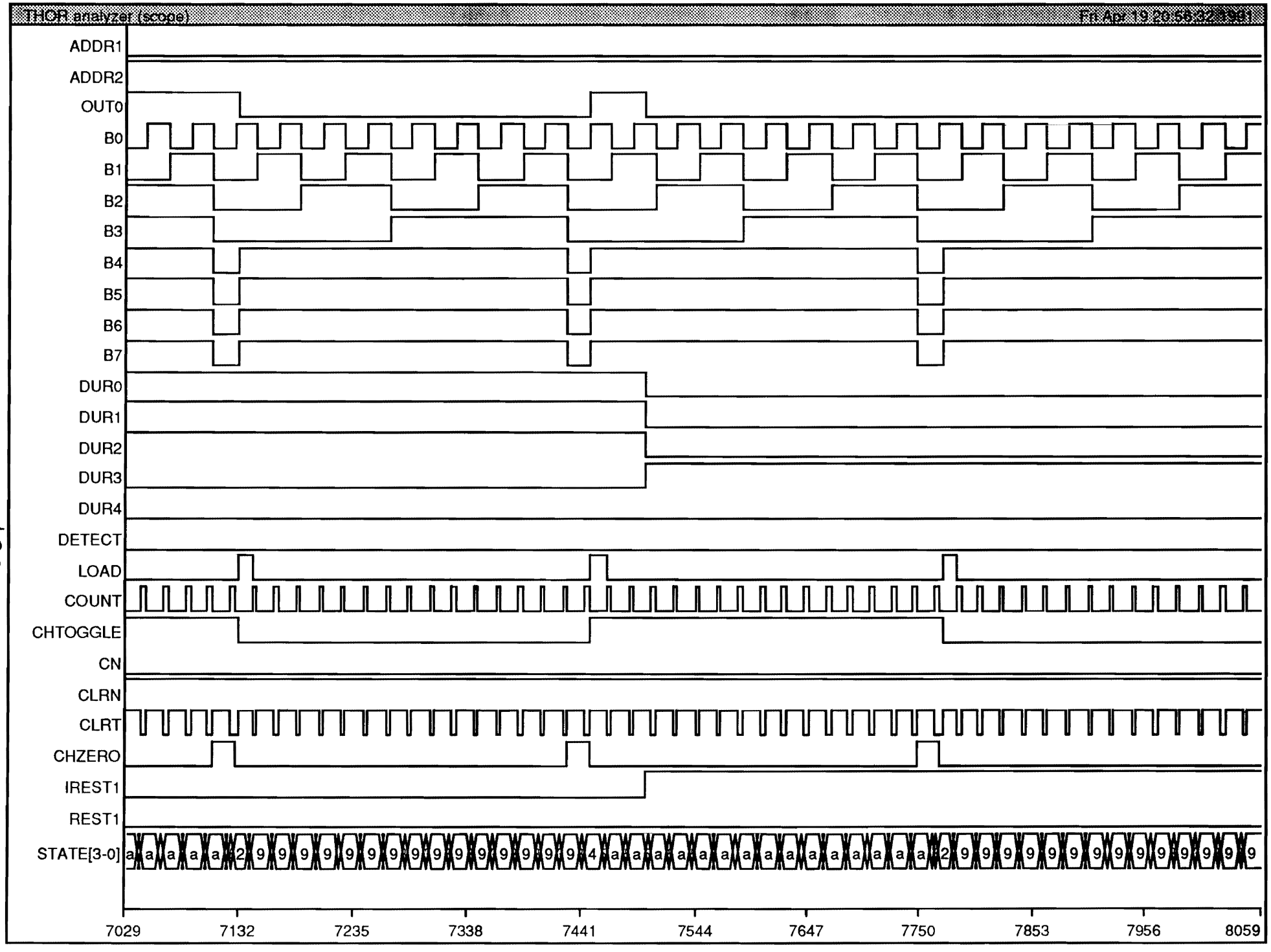

THOR analyzer (scope)
Fri Apr 19 20:56:32 1991
ADDR1
ADDR2
OUT0
B0
B1
B2
B3
B4
B5
B6
B7
DUR0
DUR1
DUR2
DUR3
DUR4
DETECT
LOAD
COUNT
CHTOGGLE
CN
CLRN
CLRT
CHZERO
IREST1
REST1
STATE[3-0]
7029
7132
7235
7338
7441
7544
7647
7750
7853
7956
8059

FIGURE E

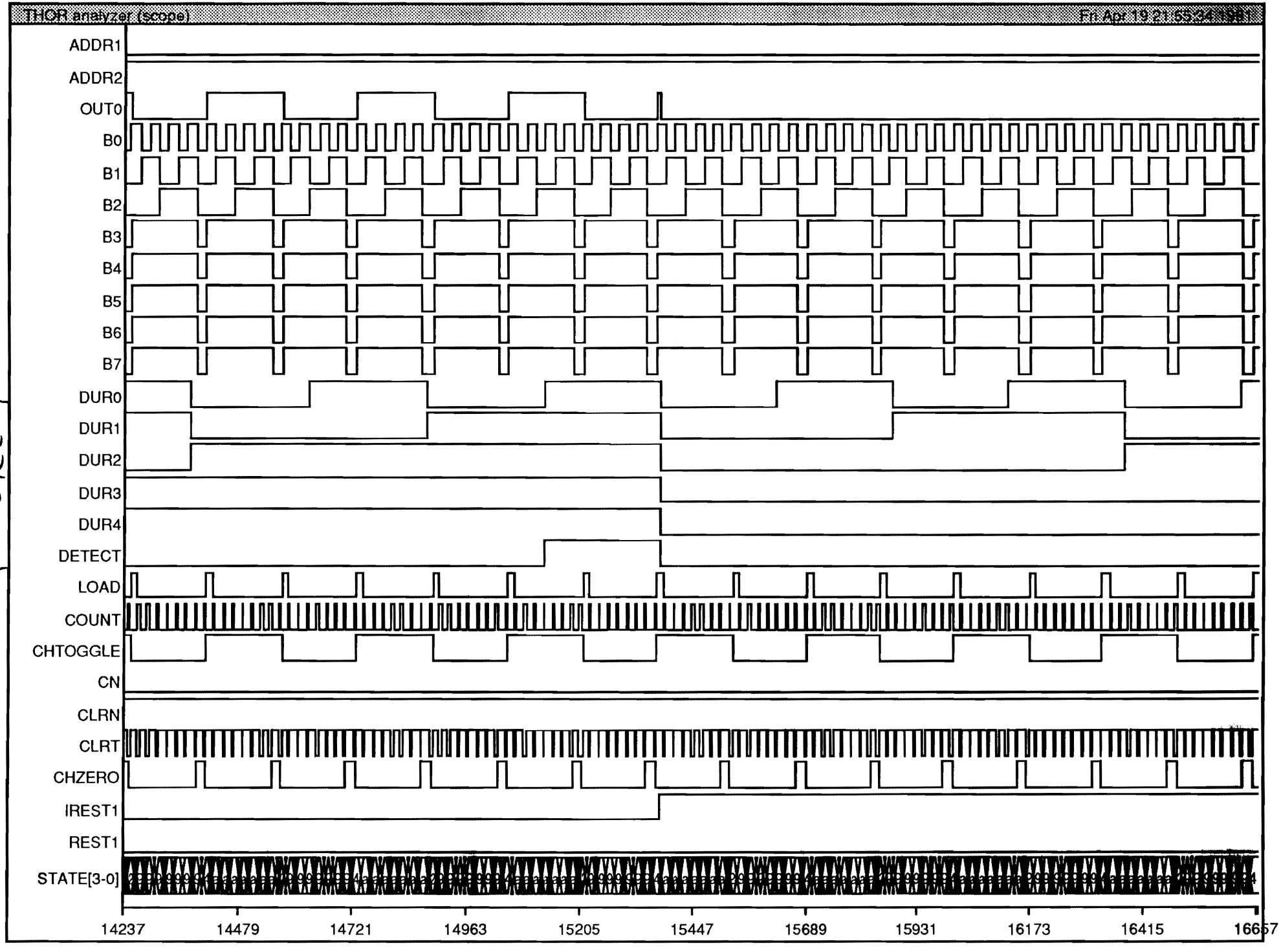

THOR analyzer (scope)
Fri Apr 19 21:55:34 1991
ADDR1
ADDR2
OUT0
B0
B1
B2
B3
B4
B5
B6
B7
DUR0
DUR1
DUR2
DUR3
DUR4
DETECT
LOAD
COUNT
CHTOGGLE
CN
CLRN
CLRT
CHZERO
IREST1
REST1
STATE[3-0]
14237
14479
14721
14963
15205
15447
15689
15931
16173
16415
16657

FIGURE F

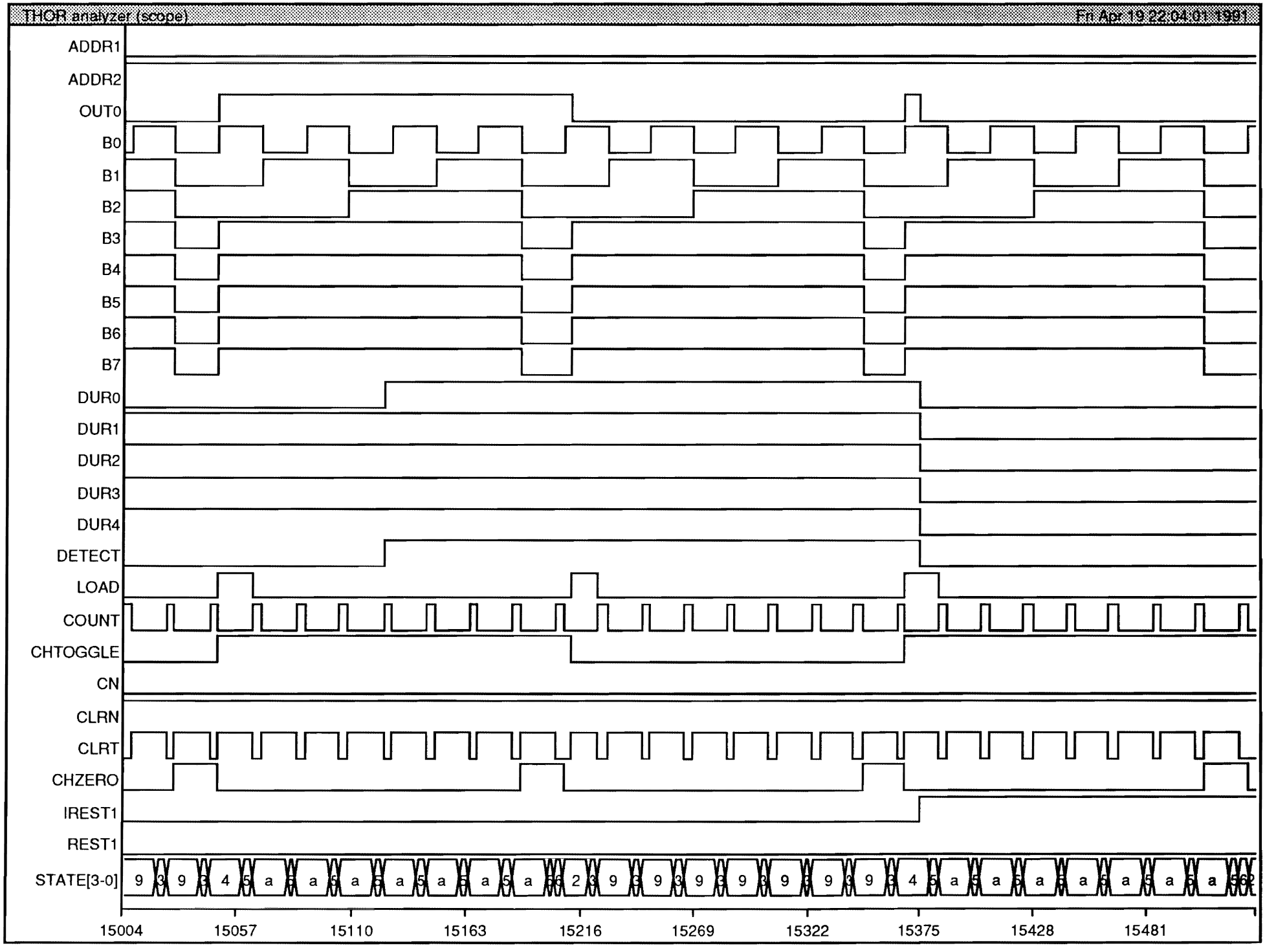

THOR analyzer (scope)
Fri Apr 19 22:04:01 1991
ADDR1
ADDR2
OUT0
B0
B1
B2
B3
B4
B5
B6
B7
DUR0
DUR1
DUR2
DUR3
DUR4
DETECT
LOAD
COUNT
CHTOGGLE
CN
CLRN
CLRT
CHZERO
IREST1
REST1
STATE[3-0]
15004
15057
15110
15163
15216
15269
15322
15375
15428
15481

FIGURE 6

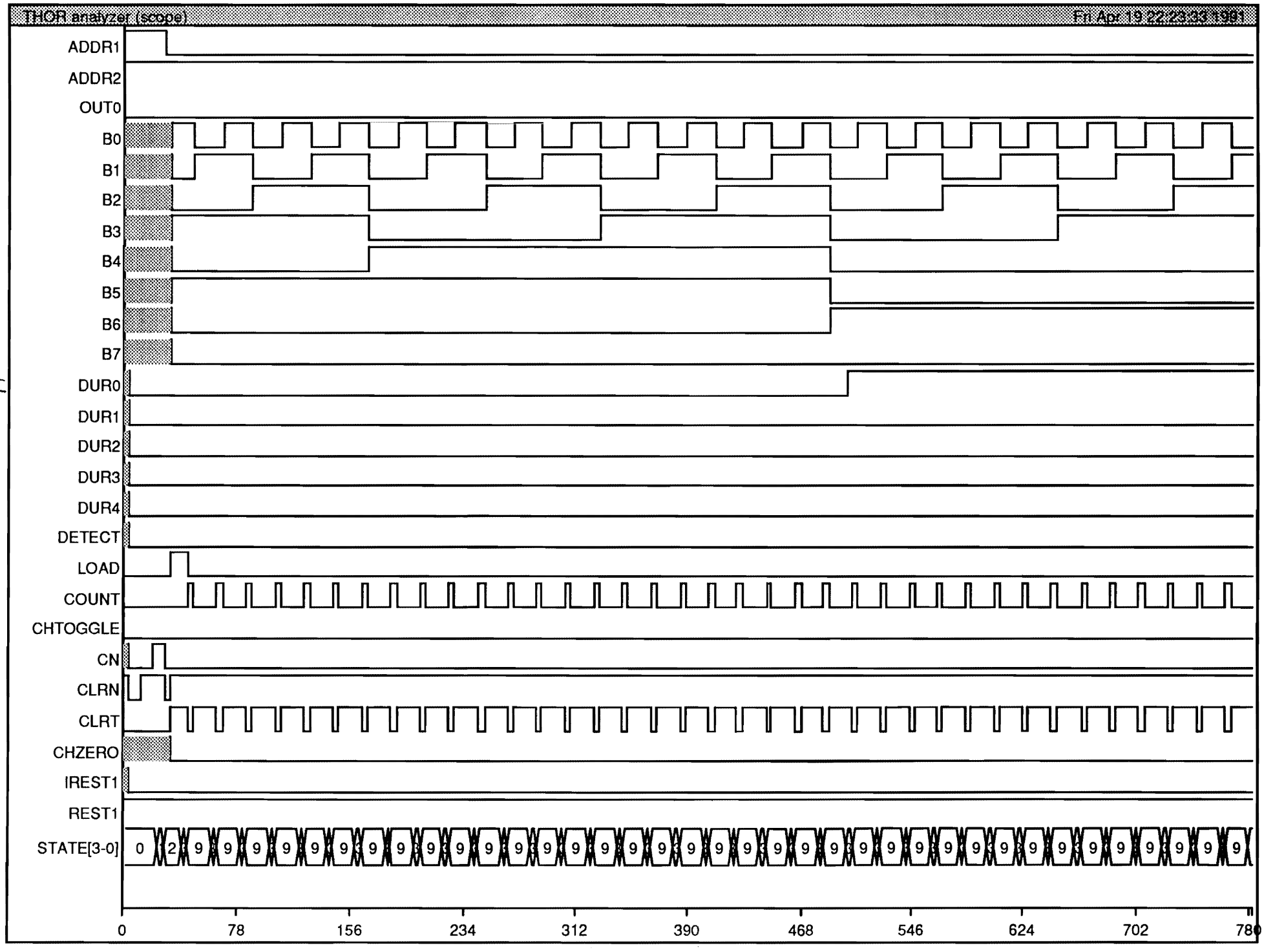

THOR analyzer (scope)
Fri Apr 19 22:23:33 1991
ADDR1
ADDR2
OUT0
B0
B1
B2
B3
B4
B5
B6
B7
DUR0
DUR1
DUR2
DUR3
DUR4
DETECT
LOAD
COUNT
CHTOGGLE
CN
CLRN
CLRT
CHZERO
IREST1
REST1
STATE[3-0]
0
78
156
234
312
390
468
546
624
702
780

FIGURE H