Ethernet Interface Documentation Package

John Seamons and Andreas Bechtolsheim
Computer Science Department
Stanford University

June 30, 1980

This set of documents describes a 3 Megabit Ethernet Interface developed by the Computer Science Department at Stanford University. The interface is designed to interconnect host computers to the Ethernet via a 16-bit parallel port.

The package contains:

- * Function and Interface Description
- * Logic Schematics
- * Prototype Layout
- * Timing Diagrams
- * State Machine Graphs
- * State Machine Code

ANFORD CSD	Project SUNet	Reference Ethernet Interface Clocks	File ei.acover.sil	Designer John Seamons	Rev 3	Date 30-jun-80	Page 1

```
COMMENT ⊗   VALID 00007 PAGES
C REC  PAGE   DESCRIPTION
C00001 00001
C00002 00002    Ethernet Interface Documentation
C00005 00003    Ethernet Receiver
C00008 00004    Ethernet Transmitter
C00010 00005    Ethernet Controller
C00012 00006    DMA Controller:
C00015 00007    Controller Function:
C00018 ENDMK
C⊗;
```

Ethernet Interface Documentation

The Ethernet interface described in the following connects an Ethernet transceiver to one of three system interfaces:

1) A 8/16 bits parallel port.
2) The GPIB General Purpose Interface Bus.
3) A RS-422 high-speed serial line.

The interface is designed for full-duplex operation.
Thus loopback (sending to oneself) is possible for diagnostic purposes.

The Ethernet interface itself consists of three parts:
a Receiver, a Transmitter, and a Controller.

The receiver converts serial data from the Ethernet into 16-bit parallel words.
A Bit-vector address filter, which is loaded from the controller, accepts only a selected set of destination addresses.
The receiver also straps the Ethernet leader bit and checks the CRC.

The transmitter translates 16-bit parallel data into a serial data stream.
It prefixes the data stream with a leader bit and appends the CRC code which it computes on the fly.
In addition, the transmitter jams the Ethernet if a collision is detected either by the transceiver or the receiver.

The controller performs the other functions necessary for a system level Ethernet interface: packet buffering, retransmission, and initialization.
The controller interfaces the Ethernet to either a 8/16 bit parallel port, a GPIB port, or a high-speed serial line port.
These ports can be easily interfaced to almost any host computer.
The controller is implemented as a fast 8-bit microcomputer system with four DMA channels, Timer, and 4 kBytes of buffer memory.

If desired, the controller's functionality can be replaced by a "host" mini- or microcomputer system. However, the timing characteristics of the Ethernet (one 16-bit word every 5.44 usec, retransmission within 37 usec) need to be carefully considered for alternative controller implementations.

Ethernet Receiver

Receiver Interface:

R.Data <0:15>	Output	Parallel Data.
R.Read0\	Input	Read Byte 0 (Bits <0:7>). } Tie together for
R.Read1\	Input	Read Byte 1 (Bits <8:15>). } 16 bit operation.
R.Read\	Input	Clear flags. Tie to R.Read0/1.
R.Ready\	Output	New data ready.
R.End\	Output	End of Message. Cleared by R.Ack.
R.Abort\	Output	Abort message in progress. Cleared by R.Ack.
R.Ack\	Input	Clear R.End and R.Abort.
R.Enable	Input	Initializes receiver state machine. Assert after initialization is complete.
R.FA <0:7>	In/Out	Filter Address. Shift Register Output if R.Enable is asserted.
R.FDin	Input	Filter Data In
R.FWE\	Input	Filter Write Enable

Receiver Functions:

If R.Enable non asserted, then receiver state machine is initialized.
Address filter can be loaded via R.FA, R.FDin, and R.FWE\.
Address filter should be loaded before interface is enabled.

After R.Enable is asserted, receiver state machine will enter
flush state until carrier idle.
Receiver then listens continously to Ethernet.
Phase Decoder acquires synchronization on Start Bit.
Receiver state machine places data into shift register.
At the end of the first word it checks the address in the address vector.
If Ethernet carrier drops, receive state machine checks
CRC and Bit-count. If both are correct, R.End flag is set.
Otherwise and in the case of collision after the first R.Ready
R.Abort is set.
Microcomputer system should throw away aborted messages.
(Number of aborted messages is expected to be very small.)

Response time requirements:

R.Ready to R.Read:	5440 nsec (16 bit times).
R.End to R.Ready:	6460 nsec (19 bit times).

Ethernet Transmitter

Transmitter Interface:

T.Data <0:15>	Input	Parallel Data.
T.Write0\	Input	Write Byte 0 (Bits <0:7>). } Tie together for
T.Write1\	Input	Write Byte 1 (Bits <8:15>). } 16 bit operation.
T.Write\	Input	Clear flags. Tie to Write0/1.
T.Ready\	Output	Ready to accept data. Cleared by T.Write.
T.Sent\	Output	Message transmitted. Cleared by T.Ack.
T.Abort\	Output	Message collided. Cleared by T.Ack.
T.Ack\	Input	Clear T.Sent and T.Abort.
T.Enable	Input	Enable transmit data driver to transceiver. Can be used to terminate transmissions immediately. Should be held inactive after power-up to allow transmitter state machine to autoinitialize.

Transmitter Functions:

Transmitter is activated by writing first data word.
Transmitter then attempts to aquire Ethernet.
Timeout function for aquisition must be provided externally.
Transmission is terminated with CRC code if no more data is available.
Transmission is aborted upon collision reported by Transceiver or Phase Decoder.
Retransmission function and algorithm must be provided externally.

Response time requirements:

T.Write to T.Ready:	5440 nsec (16 bit times).
T.Abort to T.Write:	37 usec (1 retransmission slot time).

Ethernet Controller

Controller Interface:

The controller supports one of three bidirectional data ports:
a parallel port, a GPIB port, and a SIO port.
For a particular system, only one of these ports can be used.
The particular port in use will be called "local port".

The interface is implemented via messages consisting of one 16-bit command word that is optionally followed by a fixed number of data words. The length of the data field is limited to 512 words (1024 bytes).

Message Summary:

Host to Ethernet Controller:

Initialize
Load Address Filter with <n> address
What is your fixed address?
Send following packet with <n> words, <words>*n

Ethernet Controller to Host:

My fixed address is <n>
Timeout on network aquisition
I failed to send a packet despite 8 retries
I received following packet with <n> words, <words>*n

DMA Controller:

All data transfers are handled by the 9517 DMA Controller.
This DMA chip has four channels allocated as follows:

Channel 0 Receiver
Channel 1 Receiver EOP Pointer
Channel 2 Transmitter
Channel 3 Local Port

```
Chan 0: Autoinitialization implements cyclical buffer.
        Save all receiver data in buffer (including bad messages)
Chan 1: Autoinitialization implements cyclical buffer.
        Save current address and R.Abort on R.End.
Chan 2: Transfer fixed number of data words from buffer to transmitter.
Chan 3: Transfer fixed number of data words between local port and buffer.
```

The first two channels allow to receive back-to-back packets while the buffer memory can be simultaneously unloaded to the local port.

Both the channel for Receiver Data and EOP Address perform their task without intervention of the microprocessor and deposit their data items into cyclical queues in buffer memory.
In contrast, the local port Read/Write and Transmitter Data channel are explicitely started under microcomputer control with a definite wordcount, which is the length of the packet to be transferred.
The Microcomputer is interrupted on EOP of a local port transfer and on SENT from the transmitter. Note that SENT is set whenever the transmitter terminated its transmission, which might be on collision and on data underflow in addition to regular completion.

Controller Function:

Upon initialization (either hardware or command), the 8085 controller:
1) reads the fixed address and loads it into the address filter,
2) initializes the Ports
3) initializes the DMA device
4) enables the receiver and the transmitter
5) enables interrupts
6) halts.
All further functions are interrupt driven (described below).

Interrupt routines:

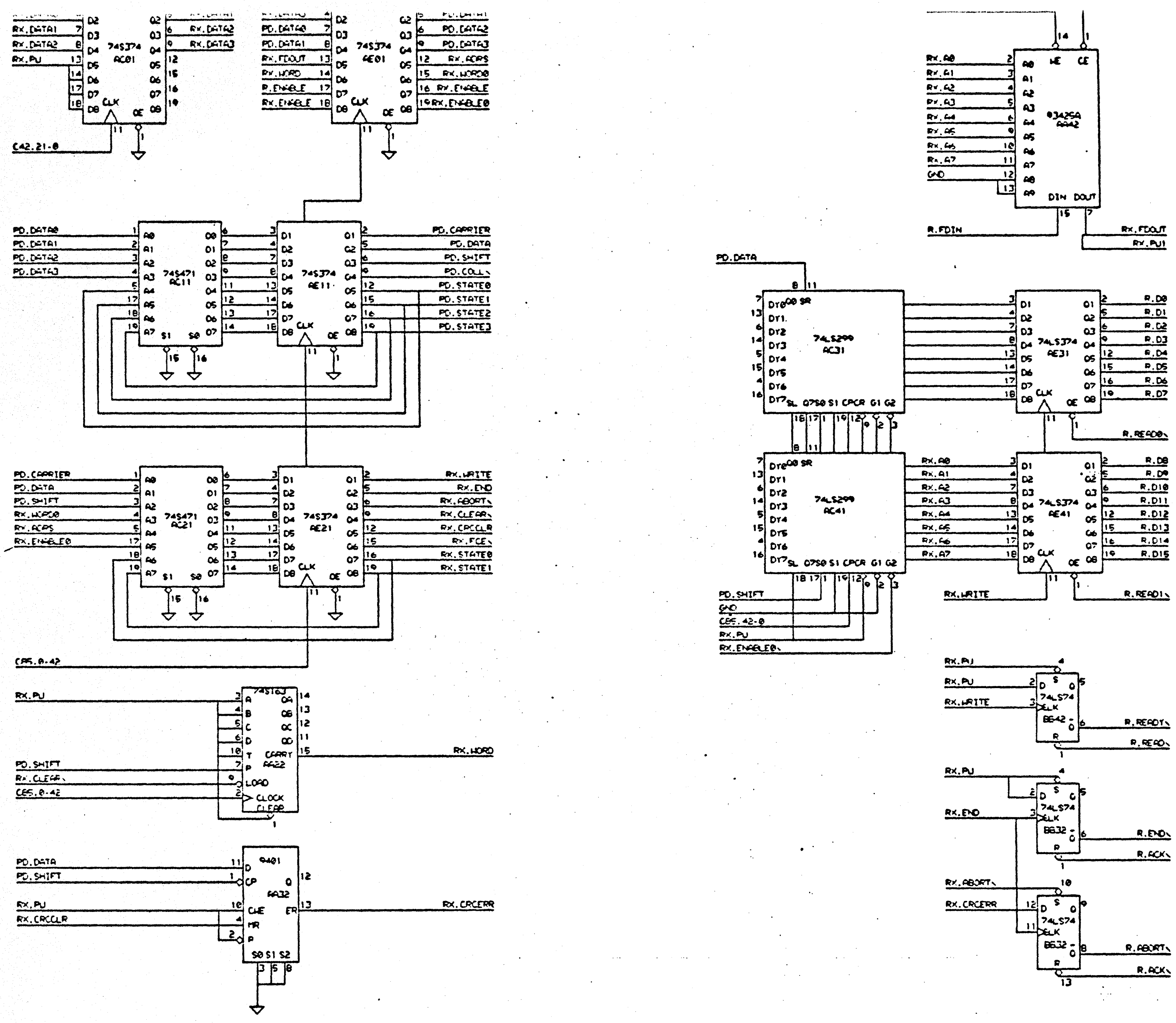
Trap (edge and level, non-maskable)
not used because of difficulty to synchronize DMA device manipulation.

RST 7.5 (edge level)
TO (Timer: Count=0)
Timer is started on network aquisition for timeout and
on collision for retransmission.
At other times timer is running continously for random number generation.
RST 7.5 Interrupt is disabled then.

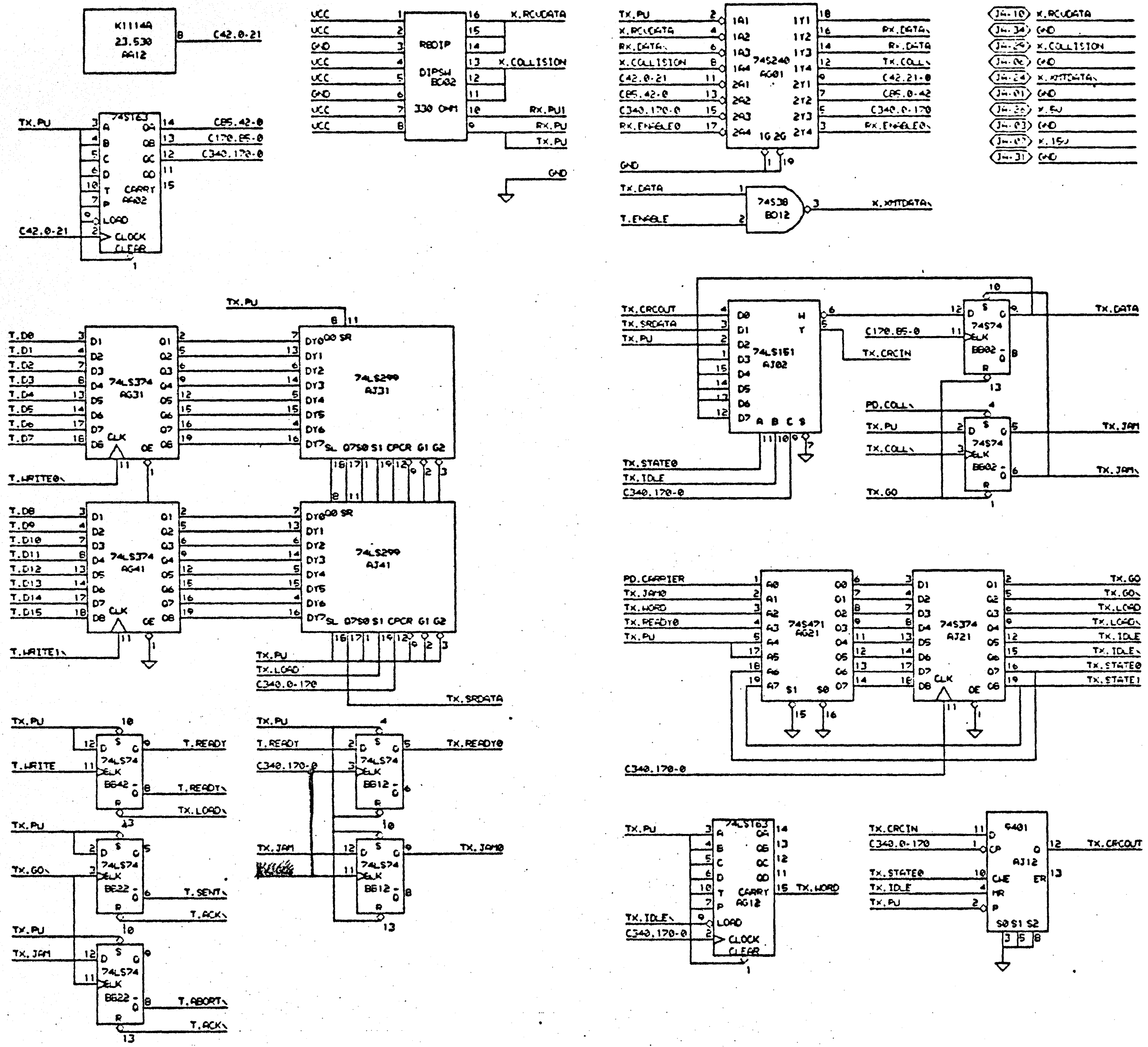
RST 6.5 (level)
RX: Message arrived (Pointer and Status was queued by Channel 1)
Reset Interrupt request
Check DMA Channel 1 current address to see how many pointers are pending.
Note that if last interrupt was asynchronous, no new message arrived.
Future Interrupts will again be handled synchronously.
If Abort then throw packet away else put on received queue.
Send message to host that packet arrived.

RST 5.5 (level)
TX: Message sent.
Check Collision Status.
Check whether DMA controller terminated transmission.
Clear request.
If Collision then if collision count < 8 then retry
else send host message about failure.
If port idle then write message header to port else put on queue.

INTR: (level)
If port idle, then Intr ← Write Interrupt request else
Intr ← EOP channel 3.
Check whether there are any pending messages for host.
Reenter idle state. (Checks on pending messages from host).



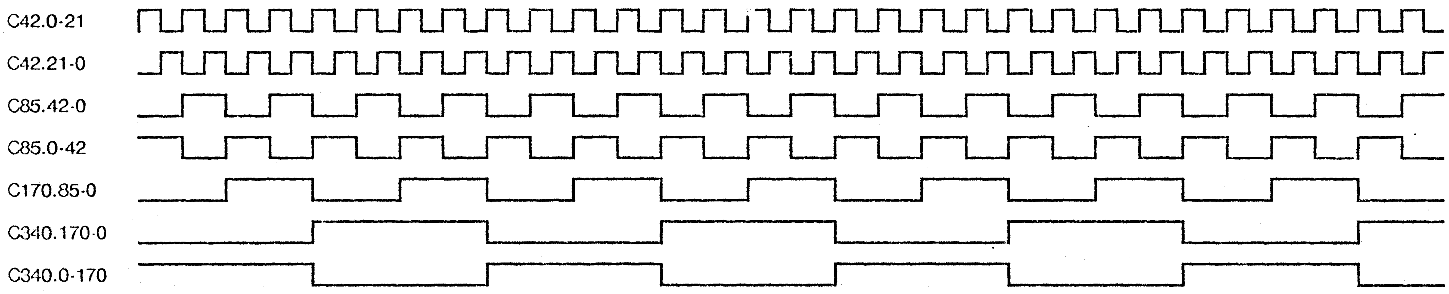
ETHERNET INTERFACE	RECEIVER	03-JUL-80 12:25	E0[SUN, AVB]



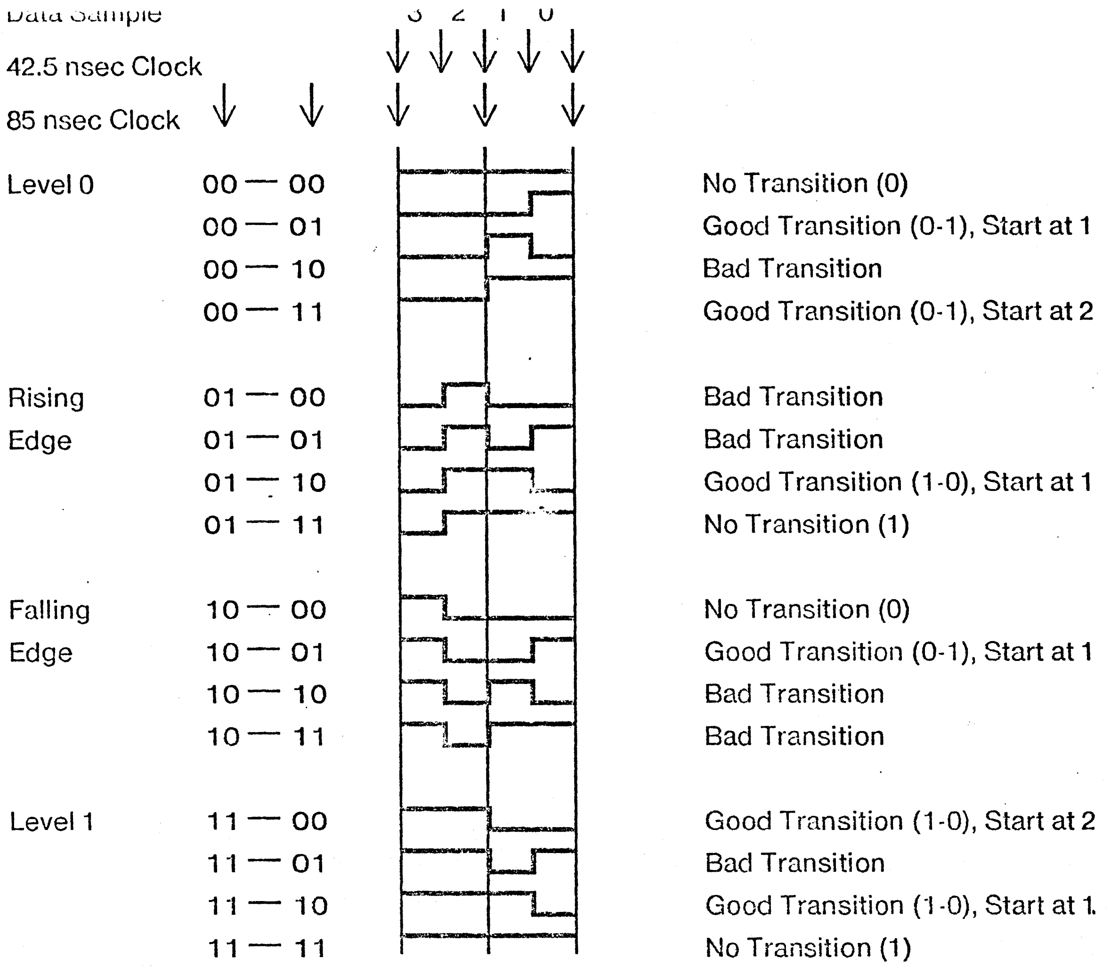
ETHERNET INTERFACE TRANSMITTER 03-JUL-80 12:30 E1[SUN, AVB

```
                                     E.PRT[SUN,AVB]        03-JUL-80  1231

PART NUMBER     DIPTYPE        COUNT  DESCRIPTION                       LOCATIONS

N/A             74S163         2                                        AA02, AA22
                K1114A         1      FREQ:23.530                       AA12
                9401           2                                        AA32, AJ12
                93425A         1                                        AA42
                74S374         5                                        AC01, AE01, AE11, AE21, AJ21
                74S471         3                                        AC11, AC21, AG21
                74LS299        4                                        AC31, AC41, AJ31, AJ41
                74LS374        4                                        AE31, AE41, AG31, AG41
                74S240         1                                        AG01
                74LS163        1                                        AG12
                74LS151        1                                        AJ02
                74S74          1                                        BB02
                74LS74         4                                        BB12, BB22, BB32, BB42
                DIPSW          1                                        BD02
                74S38          1                                        BD12
```

```
BF-->:  . . . . : . . . . : . . . . : . . . . : . . . . : . . . . : . . . . : . . . . : . . . . : . . . . :
        ----------------    -------------
BE-->:  !DIPSW          ! : !74S38        ! . : . . . . : . . . . : . . . . : . . . . : . . . . : . . . . :
        !BD02           !   !BD12         !
BD-->:  ! 1/1           ! : ! 1/4  *      ! . : . . . . : . . . . : . . . . : . . . . : . . . . : . . . . :
        \---------------    \------------       -------------      --------------       --------------
BC-->:  !74S74        ! . : !74LS74       ! . : !74LS74       ! . : !74LS74       ! . : !74LS74       ! . :
        !BB02         !     !BB12         !     !BB22         !     !BB32         !     !BB42         !
BB-->:  ! 2/2         ! . : ! 2/2         ! . : ! 2/2         ! . : ! 2/2         ! . : ! 2/2         ! . :
        \---------------    \------------     --\---------------- --\---------------- --\-----------------
BA-->:  !74LS151        ! : !9401         ! . !74S374             !74LS299            !74LS299            !
        !AJ02           !   !AJ12         !   !AJ21               !AJ31               !AJ41               !
AJ-->:  ! 1/1           ! : ! 1/1         ! . ! 1/1               ! 1/1               ! 1/1               !
      --\-----------------  \--------------   \------------------ \------------------ \-------------------
AH-->!74S240              ! !74LS163        ! !74S471             !74LS374            !74LS374            !
     !AG01                ! !AG12           ! !AG21               !AG31               !AG41               !
AG-->! 1/2  *             ! ! 1/1           ! ! 1/1               ! 1/1               ! 1/1               !
     \------------------- --\---------------- \------------------ \------------------ \-------------------
AF-->!74S374              !74S374             !74S374             !74LS374            !74LS374            !
     !AE01                !AE11               !AE21               !AE31               !AE41               !
AE-->! 1/1                ! 1/1               ! 1/1               ! 1/1               ! 1/1               !
     \------------------- \------------------ \------------------ \------------------ \-------------------
AD-->!74S374              !74S471             !74S471             !74LS299            !74LS299            !
     !AC01                !AC11               !AC21               !AC31               !AC41               !
AC-->! 1/1                ! 1/1               ! 1/1               ! 1/1               ! 1/1               !
     \------------------- \------------------ \------------------ \------------------ \-------------------
AB-->:  !74S163         ! : !K1114A       ! . : !74S163        ! : !9401         ! . : !93425A       ! :
        !AA02           !   !AA12         !     !AA22          !   !AA32         !     !AA42          !
AA-->:  ! 1/1           ! : ! 1/1         ! . : ! 1/1          ! : ! 1/1         ! . : ! 1/1          ! :
       \---------------    \------------       \--------------     \------------       \--------------
```

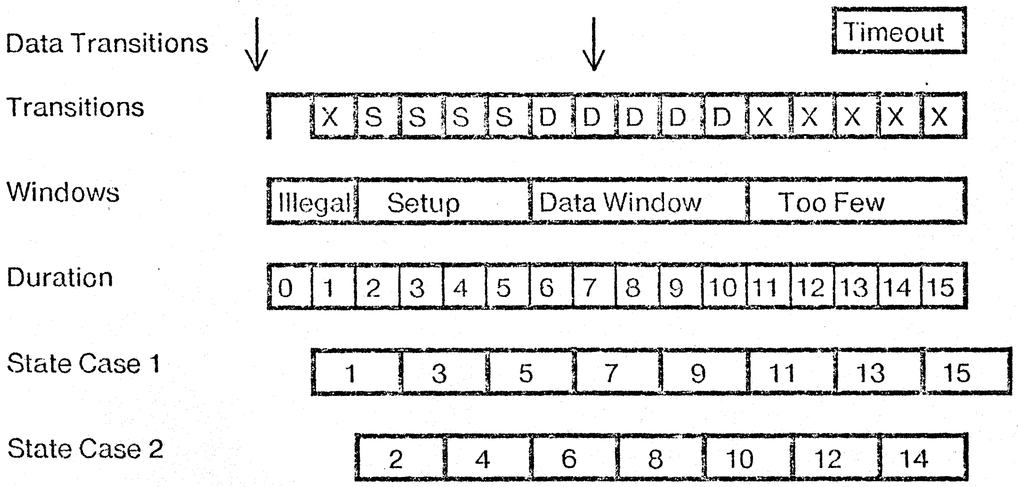


ANFORD	Project	Reference	File	Designer	Rev	Date	Page
CSD	SUNet	Ethernet Interface Clocks	ei.clocks.sil	John Seamons	3	30-jun-80	1



Level 0	00 — 00	No Transition (0)
	00 — 01	Good Transition (0-1), Start at 1
	00 — 10	Bad Transition
	00 — 11	Good Transition (0-1), Start at 2
Rising Edge	01 — 00	Bad Transition
	01 — 01	Bad Transition
	01 — 10	Good Transition (1-0), Start at 1
	01 — 11	No Transition (1)
Falling Edge	10 — 00	No Transition (0)
	10 — 01	Good Transition (0-1), Start at 1
	10 — 10	Bad Transition
	10 — 11	Bad Transition
Level 1	11 — 00	Good Transition (1-0), Start at 2
	11 — 01	Bad Transition
	11 — 10	Good Transition (1-0), Start at 1.
	11 — 11	No Transition (1)

The Shift Register samples RX.Data every 42.5 nsec.
Since the Phase Decoder State Machine is clocked at 85 nsec, each 85 nsec window has 4 possible cases (Level 0, Rising Edge, Falling Edge, Level 1). Thus there are 16 possible transitions between consecutive state machine states, which are shown above.



Transitions		X	S	S	S	S	D	D	D	D	D	X	X	X	X	X
Duration	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

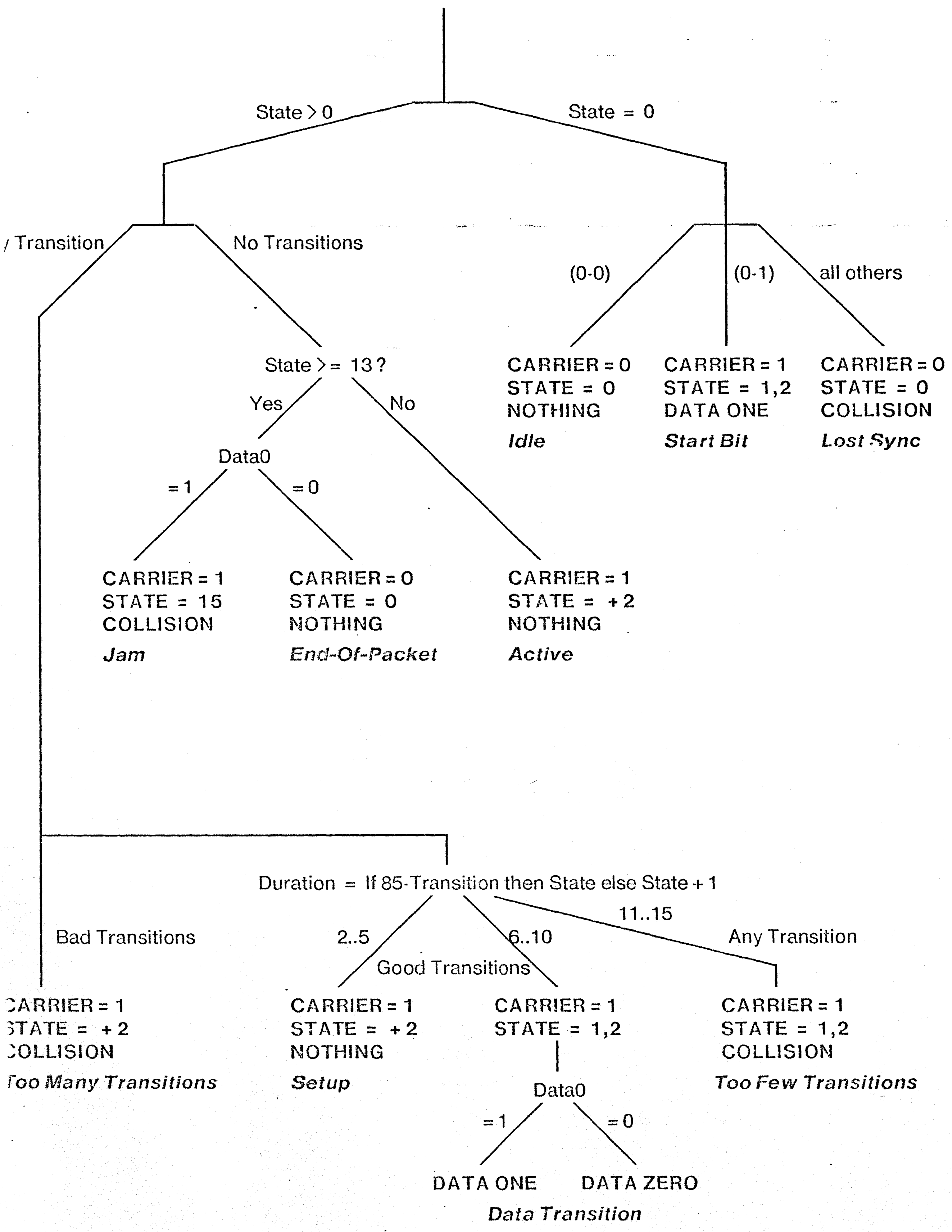
Windows: Illegal | Setup | Data Window | Too Few

State Case 1: 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15

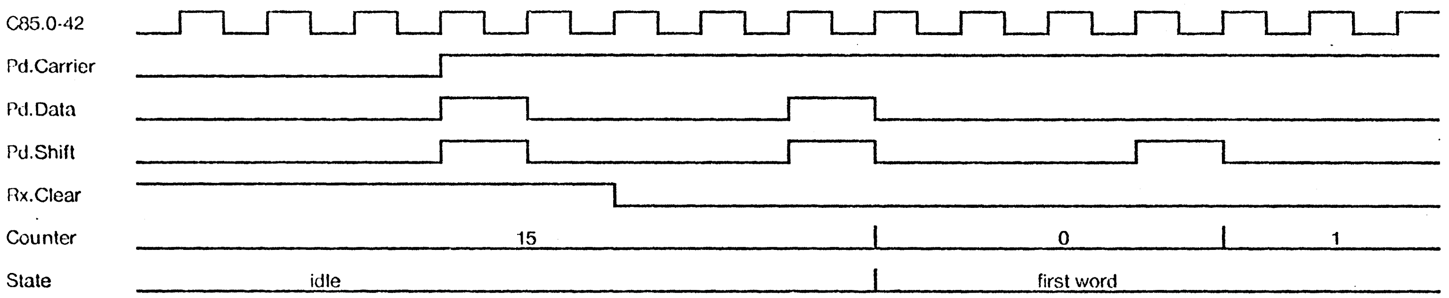
State Case 2: 2 | 4 | 6 | 8 | 10 | 12 | 14

State 0 means no carrier.
State measures time between data transitions in 42.5 nsec units.
State starts at 1 if data transition falls in between 85 nsec clock, starts on 2 if data transition on 85 nsec boundary.
Duration of pulse is computed from current state, adjusted by +1 if transition falls into middle of 85 nsec cycle.

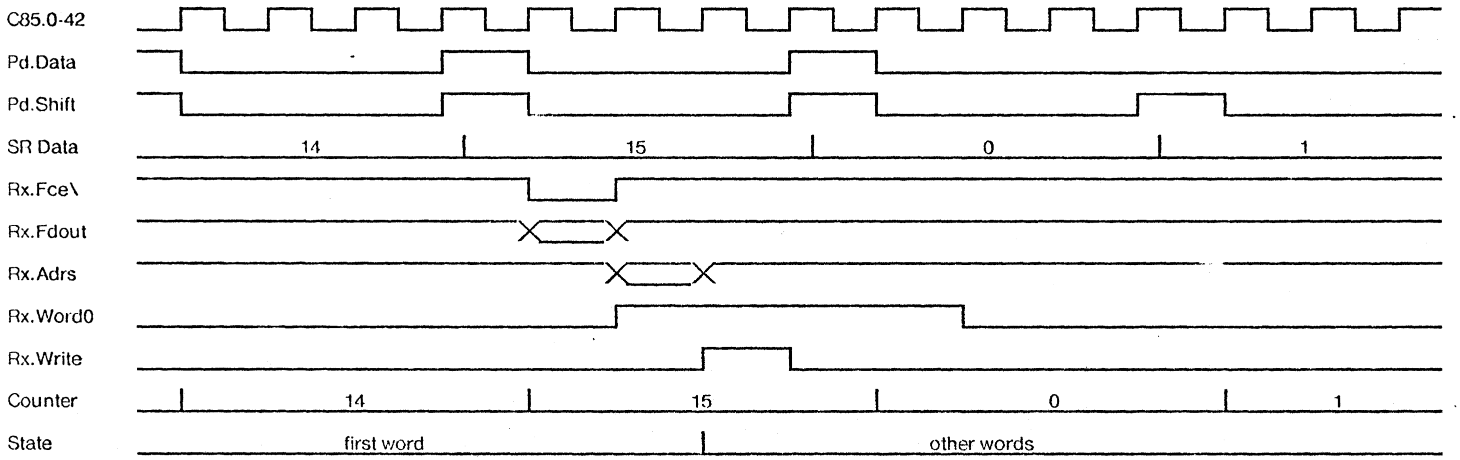
ANFORD CSD	Project	Reference	File	Designer	Rev	Date	Page
	SUNet	Phase Decoder SR	ei.pdsr.sil	John Seamons	3	30-jun-80	1



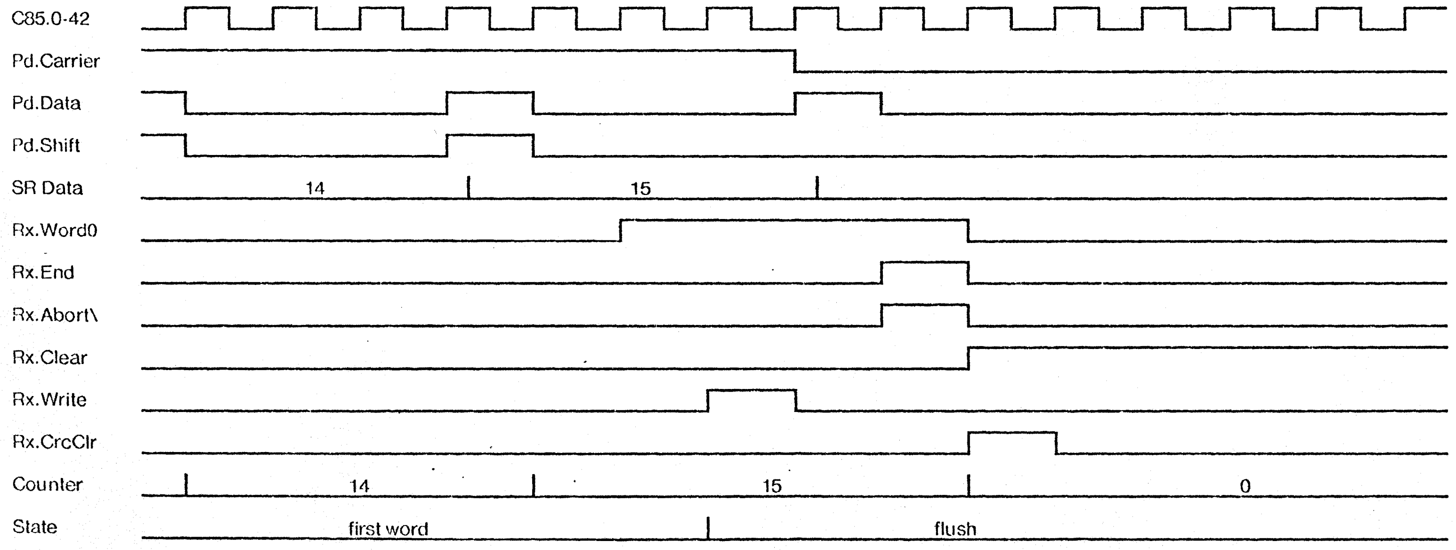
ANFORD	Project	Reference	File	Designer	Rev	Date	Page
CSD	SUNet	Phase Decoder FSM	ei.pdfsm.sil	John Seamons	3	30-jun-80	1



Start Bit

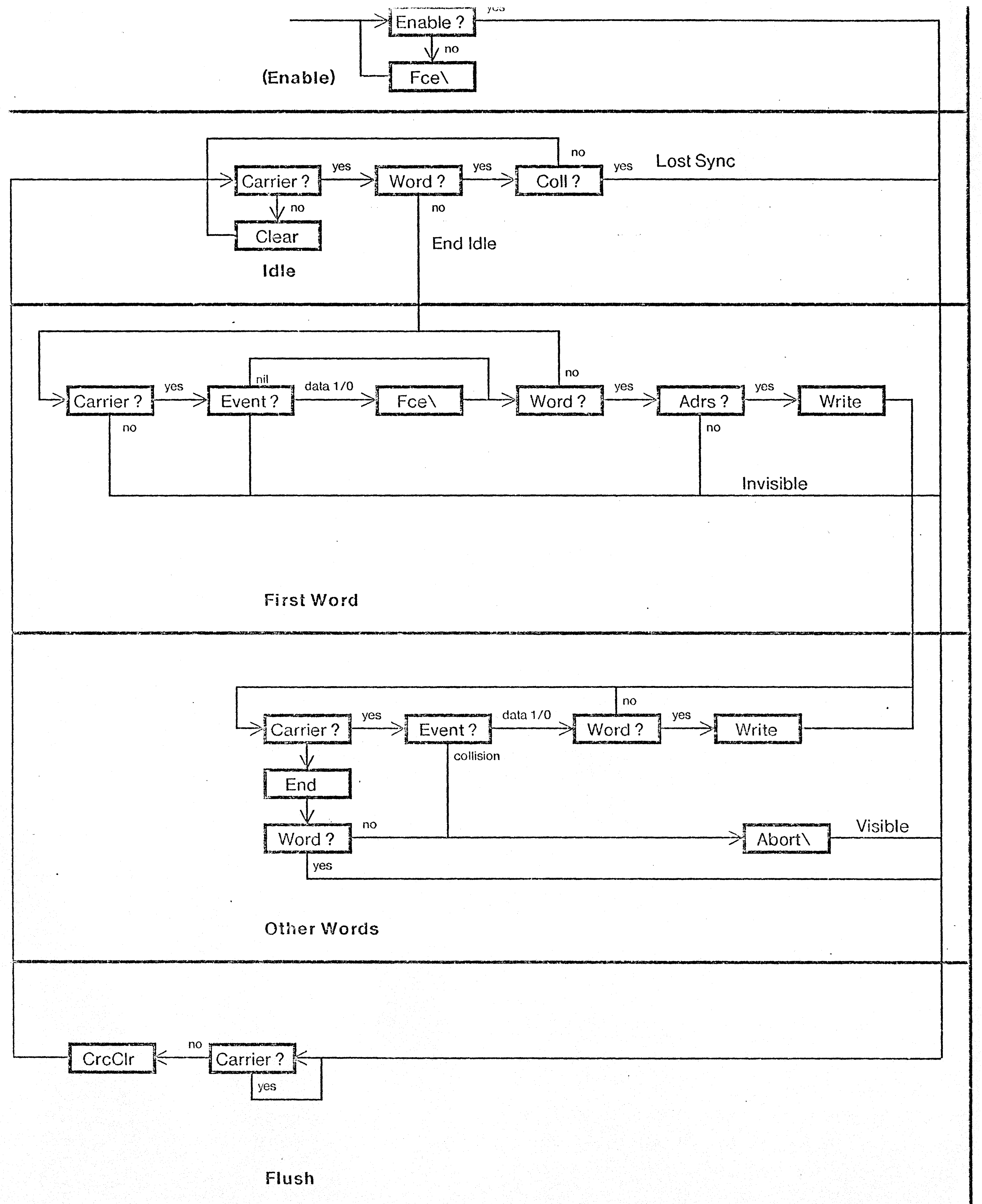


SR Full

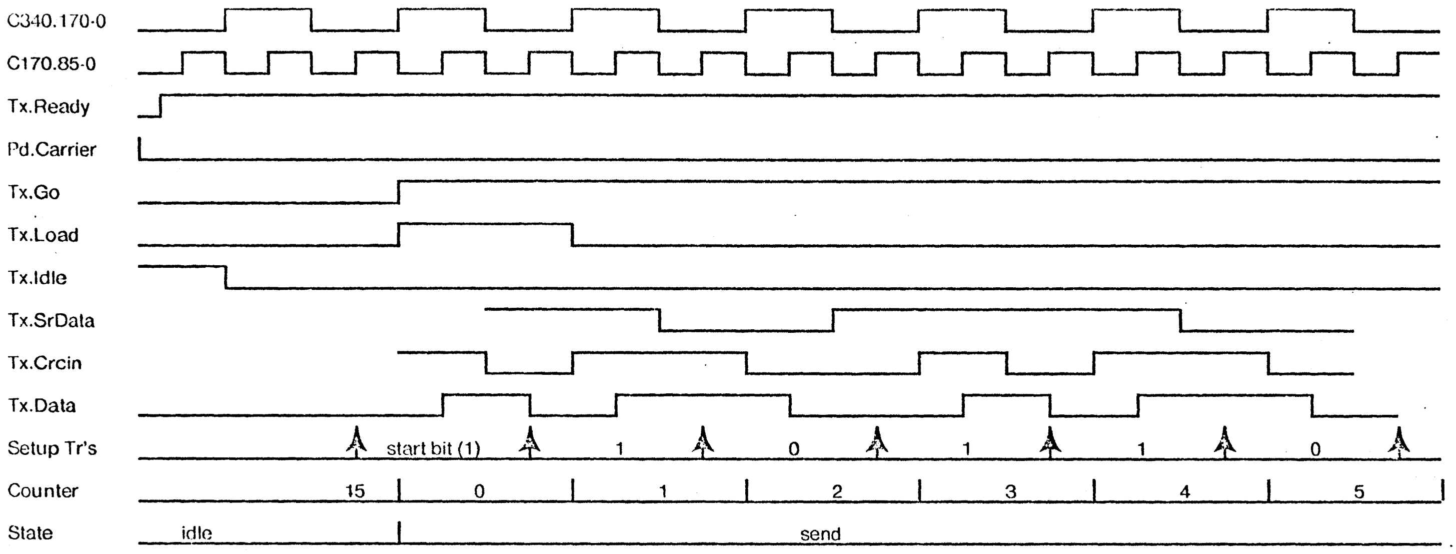


End of Packet

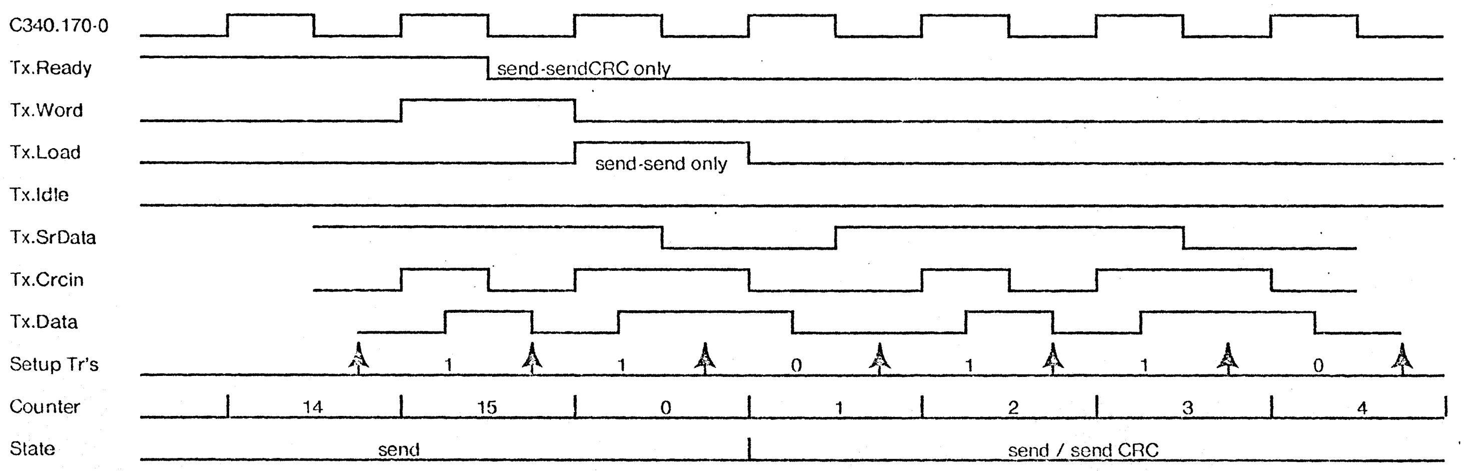
TANFORD	*Project*	*Reference*	*File*	*Designer*	*Rev*	*Date*	*Page*
CSD	SUNet	Receiver Timing	ei.rxtiming.sil	John Seamons	3	30-jun-80	1



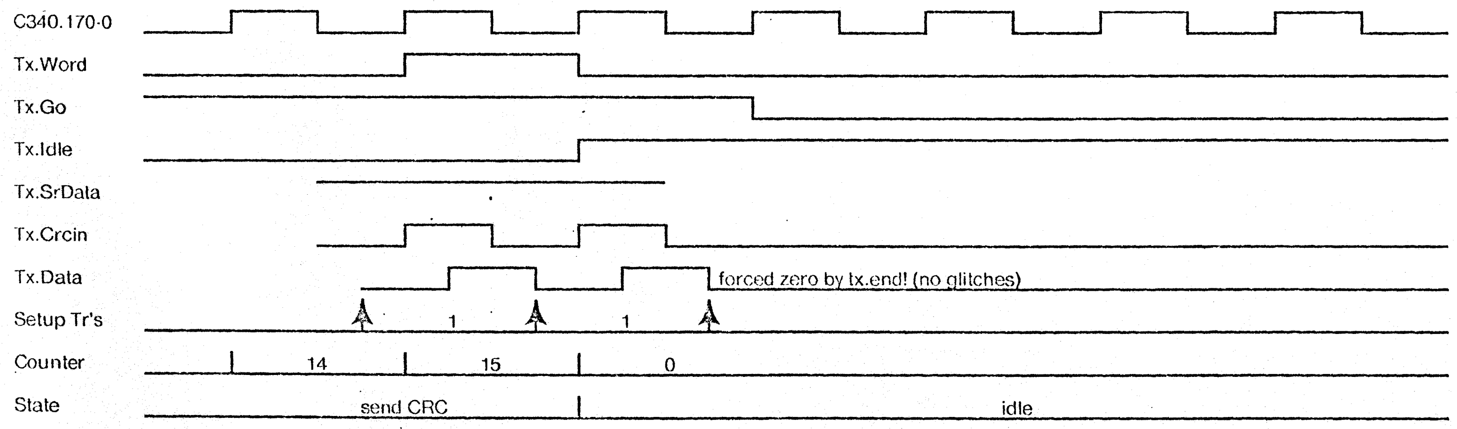
ANFORD	Project	Reference	File	Designer	Rev	Date	Page
CSD	SUNet	Receiver FSM	ei.rxfsm.sil	John Seamons	3	30-jun-80	1



Idle

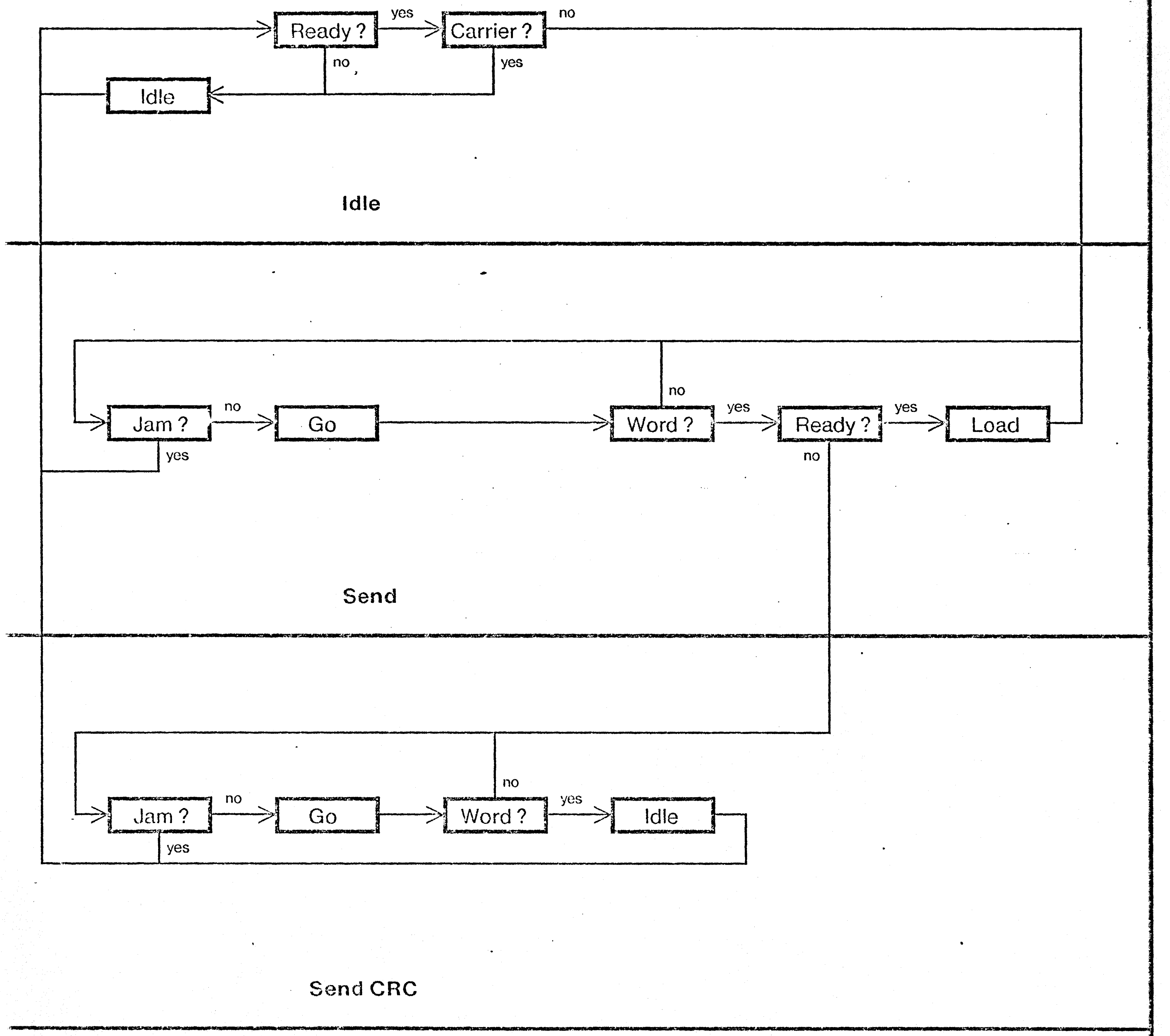


Send / Send CRC



Shut Down

TANFORD CSD	*Project* SUNet	*Reference* Transmitter Timing	*File* ei.txtiming.sil	*Designer* John Seamons	*Rev* 3	*Date* 30-jun-80	*Page* 1



ANFORD CSD	Project	Reference	File	Designer	Rev	Date	Page
	SUNet	Transmitter FSM	ei.txfsm.sil	John Seamons	3	30-jun-80	1

```
begin "prmgen"
require "prom.sai" source!file;
$256;

define

data0   =[a0],
data1   =[a1],
data2   =[a2],
data3   =[a3],
state0  =[a4],
state1  =[a5],
state2  =[a6],
state3  =[a7],

$carrier=[d0],
$data   =[d1],
$shift  =[d2],
$coll_i =[d3],
$state0 =[d4],
$state1 =[d5],
$state2 =[d6],
$state3 =[d7],
state              =[((state0 + state1 + state2 + state3) div d4)],
active             =[(state≠0)],
onetransition      =[(¬data2 ∧ data0)],                 comment X0 → X1;
zerotransition     =[(data2 ∧ ¬data0)],                 comment X1 → X0;
idletransition     =[(¬data3 ∧ ¬data2 ∧ ¬data1 ∧ ¬data0)], comment 00 → 00;
goodtransition     =[(onetransition v zerotransition)],
notransition       =[((¬data2 ∧ ¬data1 ∧ ¬data0)       comment X0 → 00;
                    v (data2 ∧ data1 ∧ data0))],        comment X1 → 11;
badtransition      =[(¬goodtransition ∧ ¬notransition)],   comment all others;
_85_transition     =[(data0 ∧ data1 v ¬data0 ∧ ¬data1)],   comment in sync with 85;
duration           =[(if _85_transition then state else (state+1))],

idle               =[(¬active ∧ idletransition)],
startbit           =[(¬active ∧ onetransition)],
lostsync           =[(¬active ∧ ¬(idletransition v onetransition))],
toomany            =[(active ∧ badtransition)],
setup              =[(active ∧ goodtransition ∧ (2≤duration≤5))],
datatrans          =[(active ∧ goodtransition ∧ (6≤duration≤10))],
toofew             =[(active ∧ (goodtransition v badtransition) ∧ (11≤duration≤15))],
timeout            =[(active ∧ notransition ∧ (13≤state≤15))],
jam                =[(timeout ∧ data0)],
eop                =[(timeout ∧ ¬data0)],

data_zero          =[(zerotransition ∧ datatrans)],
data_one           =[((onetransition ∧ datatrans) v startbit)],
collision          =[(lostsync v jam v toomany v toofew)],
start              =[(startbit v lostsync v jam v datatrans v toofew)],
reset              =[(idle v eop)],
nextstate          =[(if reset then 0
                      else if (start ∧ ¬_85_transition) then 1
                      else if (start ∧ _85_transition) then 2
                      else (state+2))];

prombegin

bit($carrier,      (nextstate≠0));
bit($data,         (collision v data_one));
bit($shift,        (data_zero v data_one));
bit($coll_i,       (¬collision));
bit($state0,       (nextstate land d0));
bit($state1,       (nextstate land d1));
bit($state2,       (nextstate land d2));
bit($state3,       (nextstate land d3));

if (idle ∧ (collision v data_zero v data_one))
        then error ("illegal idle state!");
if (collision ∧ (idle v data_zero v data_one))
        then error ("illegal collision state!");
if (data_zero ∧ (idle v collision v data_one))
        then error ("illegal data_zero state!");
if (data_one ∧ (idle v collision v data_zero))
        then error ("illegal data_one state!");

promend;
writeprom("pd",0);
end;
```

```
begin "prmgen"
require "prom.sai" source!file;
$256;

define

carrier         =[a0],
data            =[a1],
shift           =[a2],
word            =[a3],
adrs            =[a4],
enable          =[a5],
state0          =[a6],
state1          =[a7],

$write          =[d0],
$end            =[d1],
$abort_i        =[d2],
$clear_i        =[d3],
$crcclr         =[d4],
$fce_i          =[d5],
$state0         =[d6],
$state1         =[d7],
init            =[(¬enable)],
state           =[((state1 + state0) div d6)],
idle            =[(state=0)],
first_word      =[(state=1)],
other_words     =[(state=2)],
flush           =[(state=3)],
event           =[((shift + data) div d1)],
nil             =[(event=0)],
collision       =[(event=1)],
d_zero          =[(event=2)],
d_one           =[(event=3)],
datatr          =[(carrier ∧ (d_zero v d_one v nil))],
visible         =[(other_words ∧ ((¬carrier ∧ ¬word) v (carrier ∧ collision)))],
invisible       =[(first_word ∧ ((¬carrier) v (carrier ∧ collision) v
                  (datatr ∧ word ∧ ¬adrs)))],
end_idle        =[(idle ∧ carrier ∧ ¬word)],
lostsync        =[(idle ∧ carrier ∧ collision)],

nextstate       =[(
                if (       ! idle;
                (idle ∧ ¬carrier) v
                (idle ∧ carrier ∧ ¬(collision v end_idle)) v
                (flush ∧ ¬carrier)
                ) then 0 else
                if (    ! first_word;
                (first_word ∧ datatr ∧ ¬word) v
                (end_idle)
                ) then 1 else
                if (    ! other_words;
                (first_word ∧ datatr ∧ word ∧ adrs) v
                (other_words ∧ datatr)
                ) then 2 else
                if (    ! flush;
                (flush ∧ carrier) v
                (lostsync) v
                (visible) v
                (invisible) v
                (other_words ∧ ¬carrier ∧ word)
                ) then 3 else
                error(cvs($$adrs)&": didn't assign any state!")
                )];

prombegin

bit($write,     (¬init ∧ ((first_word ∧ datatr ∧ word ∧ adrs) v
                        (other_words ∧ datatr ∧ word))));
bit($end,       (¬init ∧ ((visible) v (other_words ∧ ¬carrier ∧ word))));
bit($abort_i,   ¬(init v (visible)));
bit($clear_i,   ¬(init v (idle ∧ ¬carrier)));
bit($crcclr,    (init v (flush ∧ ¬carrier)));
bit($fce_i,     ¬(init v (first_word ∧ carrier ∧ (d_zero v d_one))));
bit($state0,    (init v (nextstate land d0)));
bit($state1,    (init v (nextstate land d1)));

promend;
writeprom("rx",0);
end;
```

```
spare          =[(spare0 ∧ spare1)],
state          =[((state1 + state0) div d6)],
idle           =[(state=0)],
send           =[(state=1)],
sendcrc        =[(state=2)],
start          =[(idle ∧ ready ∧ ¬carrier)],
reload         =[(send ∧ ¬jam ∧ word ∧ ready)],
nextstate      =[(if (¬spare ∨ (state=3)) then 0
                else if (       ! idle;
                (idle ∧ (¬ready ∨ carrier)) ∨
                ((send ∨ sendcrc) ∧ jam) ∨
                (sendcrc ∧ ¬jam ∧ word)
                ) then 0 else
                if (    ! send;
                (start) ∨
                (reload) ∨
                (send ∧ ¬jam ∧ ¬word)
                ) then 1 else
                if (    ! sendcrc;
                (send ∧ ¬jam ∧ word ∧ ¬ready) ∨
                (sendcrc ∧ ¬jam ∧ ¬word)
                ) then 2 else
                error(cvs($$adrs)&": didn't assign any state!")
                )];

prombegin

bit($go,       (spare∧ ((send ∨ sendcrc) ∧ ¬jam)));
bit($go_i,     (spare∧ ¬((send ∨ sendcrc) ∧ ¬jam)));
bit($load,      (spare∧ reload));
bit($load_i,    (spare∧ ¬reload));
bit($idle,      (spare∧ (idle ∧ ¬start)));
bit($idle_i,    (spare∧ ¬(idle ∧ ¬start)));
bit($state0,    (spare∧ (nextstate land d0)));
bit($state1,    (spare∧ (nextstate land d1)));

promend;
writeprom("tx",0);
end;
```

```
begin "prmgen"
require "prom.sai" source!file;
$256;

define

carrier         =[a0],
jam             =[a1],
word            =[a2],
ready           =[a3],
spare0          =[a4],
spare1          =[a5],
state0          =[a6],
state1          =[a7],

$go             =[d0],
$go_i           =[d1],
$load           =[d2],
$load_i         =[d3],
$idle           =[d4],
$idle_i         =[d5],
$state0         =[d6],
$state1         =[d7],
```