MMPP Traffic Generator for the Testing of the SCAR II Fast Packet Switch

Prepared for
William Ivancic, James Budinger and Jorge Quintana
of the
Space Electronics Branch
NASA Lewis Research Center

by

William A. Chren, Jr.
Associate Professor of Engineering
Grand Valley State University
January 16, 1995

Table of Contents

I. Introduction

A prototype MMPP Traffic Generator (TG) has been designed for testing of the COMSAT-supplied SCAR II Fast Packet Switch. By generating packets distributed according to a Markov-Modulated Poisson Process (MMPP) model, it allows the assessment of the switch performance under traffic conditions that are more realistic than could be generated using the COMSAT-supplied Traffic Generator Module. The MMPP model is widely believed to model accurately real-world superimposed voice and data communications traffic.

The TG was designed to be as much as possible of a "drop-in" replacement for the COMSAT Traffic Generator Module. The latter fit on two Altera EPM7256EGC 192-pin CPLDs and produced traffic for one switch input port. No board changes are necessary because it has been partitioned to use the existing board traces. The TG, consisting of parts "TGDATPROC" and "TGRAMCTL" must merely be reprogrammed into the Altera devices of the same name. However, the '040 controller software must be modified to provide TG initialization data. This data will be given in Section II.

I.1 High Level Description

Figure 1 depicts the inputs and outputs of the TG. Tables 1 and 2 list their functions and correspondence to signal names in the COMSAT documentation. Active low signals are indicated by the 'BAR' suffix (for TG signals) and by the 'backslash I' suffix (for COMSAT signals). Signal names were preserved with minor exceptions. Numbers in COMSAT signal names indicate the generator index (one of eight possible). This is not the case in our design, even though the numbers were retained in some cases.

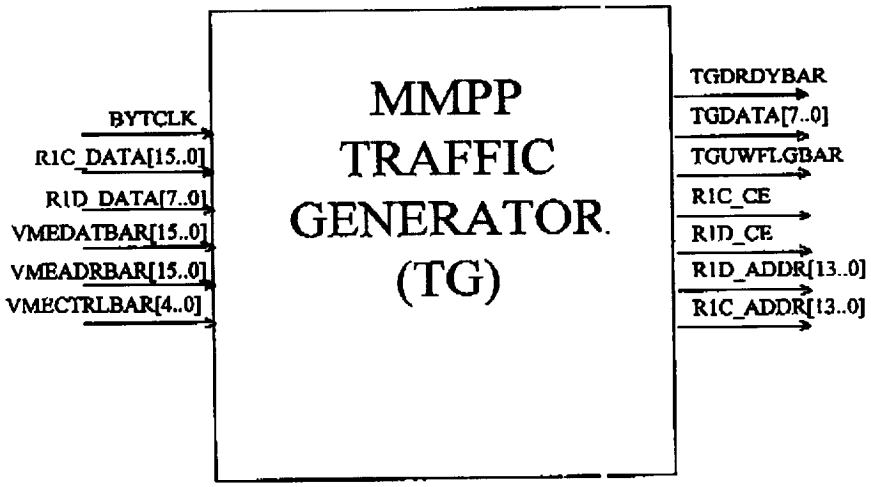

Figure 1: Traffic Generator Block Diagram

Table 1: Inputs to the Traffic Generator

Inputs	Function	Corresponding SCAR II Name
BYTCLK	20 Mhz byte clock (system clock)	TG*BYTCLK (TG Module 6/20/94 page 2)
R1C_DATA[15..0]	Memory bank C data. Contains the state dwell and idle packet counts, statistically generated according to the user specified distribution	same (Channel 1 Traffic Generator, 11/10/94 page 2)
R1D_DATA[7..0]	Memory bank D data. Contains the route numbers statistically distributed according to the user specified distribution	same (Channel 1 Traffic Generator, 11/10/94 page 3)
VMEDATBAR[15..0]	VME Slave data output lines	V/MEDAT[15..0]\I (Channel 1 Traffic Generator, 11/10/94 page 3)
VMEADRBAR[15..0]	VME Slave address output lines	V/MEADR[15..0]\I (Channel 1 Traffic Generator, 11/10/94 page 3)
VMECTRLBAR[4..0]	VME Slave control output lines	V/MECTRL[4..0]\I (Channel 1 Traffic Generator, 11/10/94 page 3)

Table 2: Outputs of the Traffic Generator

Outputs	Function	Corresponding SCAR II Name
TGDATA[7..0]	output data to ECL serialization logic	TG1DATA[7..0]\I (TG Module 11/10/94 page 3)
TGDRDYBAR	output data ready flag	TG1DRDY\I (TG Module 11/10/94 page 3)
TGUWFLGBAR	frame boundary indicator	TG1UWFLG\I (TG Module 11/10/94 page 3)
R1C_CE	memory bank C enable	same (Channel 1 Traffic Generator, 11/10/94 page 2)
R1D_CE	memory bank D enable	same (Channel 1 Traffic Generator, 11/10/94 page 2)
R1C_ADDR[13..0]	memory bank C address	same (Channel 1 Traffic Generator, 11/10/94 page 2)
R1D_ADDR[13..0]	memory bank D address	same (Channel 1 Traffic Generator, 11/10/94 page 2)

I.2 External Interfaces

An overview of the environment in which the TG will reside is given in Figure 2. All external interfaces function identically to the COMSAT design. The TG is controlled by the COMSAT dual '040 controller, which in turn is controlled by an ethernet-connected workstation. Direct control of the TG is performed by a 7256 CPLD called the VME Slave, which translates the '040 commands to the appropriate data and control signals required by the TG. Figure 3 depicts the interfaces to the memory banks and the VME Slave. Details of the type of data and control will be given below in Section II. Note that although the TG requires no modification of the COMSAT hardware, the workstation and/or '040 controller software must be modified to initialize the TG, as discussed below.

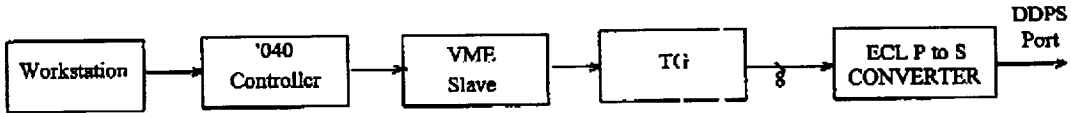

Figure 2: TG Environment

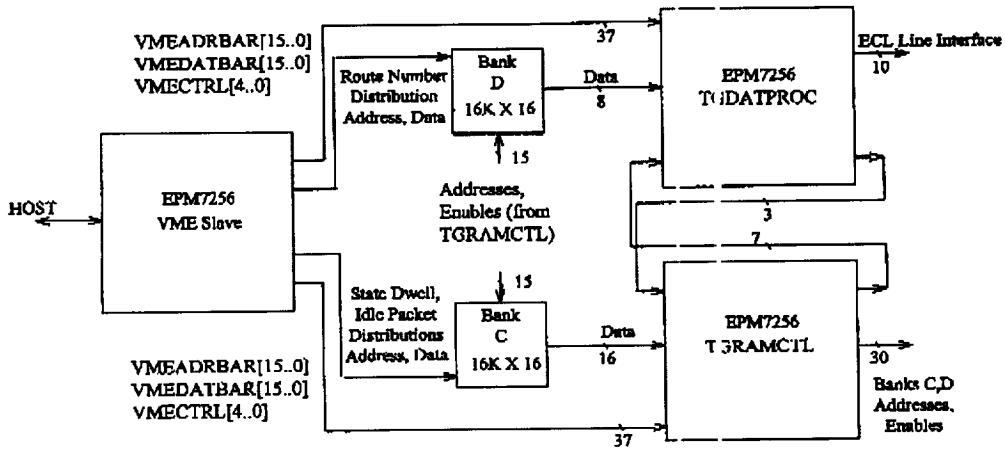

Figure 3: Interfaces to Banks C, D and VME Slave

I.3 TG Operation

The normal operating sequence of the TG consists of an initialization phase and a packet generation phase. During the initialization phase the VME slave controls the TG and must perform the seven functions listed in Table 3. For ready reference, Table 4 lists the combinations of signals VMEADRBAR[12..9], VMECTRLBAR[4..0] and VMEDATBAR[15..0] required.

Table 3: VME Slave Responsibilities During TG Initialization

VME Slave Initialization Functions

1. Send reset command to TG: assert VMEADRBAR9 and VMECTRLBAR4 for at least one period of BYTCLK.

2. Load the frame marker: assert VMEADRBAR10 and VMEADRBAR1 for at least one BYTCLK period with the frame marker on VMEDATBAR[7..0]

3. Load the test length count: assert VMEADRBAR10 and VMEADRBAR0 for at least one BYTCLK period with the test length count on VMEDATBAR[15..0]

4. Load the LFSR initialization count: assert VMEADRBAR10, VMEADRBAR1 and VMEADRBAR0 for at least one BYTCLK period with the initialization count on VMEDATBAR[15..0]

5. Load the congestion control throttles: assert VMEADRBAR11 and the combinations in Table 3b for *at least two BYTCLKL periods* with the throttle settings on VMEDATBAR[15..0]. Odd (even) throttles must be placed on lower (upper) byte of VMEDATBAR.

6. Load the congestion control enable bits: assert VMEADRBAR12 for *at least one BYTCLKD period* with the enable bits on VMEDATBAR[7..0]. Port numbers and bit numbers in VMEDATBAR[7..0] correspond.

7. Enable the TG and begin the test: assert VMEADRBAR9 and VMECTRLBAR3 for at least one period of BYTCLK

Table 4: VME Signal Combinations During Initialization

VMEADR BAR[12..0]	VME CTRL BAR [4..0]	VMEDAT BAR[15..0]	Function
xxx1xxxxxxxxx	1xxxx	xxxxxxxxxxxxxxxx	Reset TG
xx1xxxxxxxx1x	xxxxx	xxxxxxxxdddddddd	Load the frame marker
xx1xxxxxxxxx1	xxxxx	dddddddddddddddd	Load the test length count
xx1xxxxxxxx10	xxxxx	dddddddddddddddd	Load the LFSR initialization count
x1xxxxxxxxx00	xxxxx	eeeeeeeeoooooooo	Load the congestion control

		(o=odd, e=even)	throttles for ports 1 and 2
x1xxxxxxxxx01	xxxxx	eeeeeeeeoooooooo (o=odd, e=even)	Load the congestion control throttles for ports 3 and 4
x1xxxxxxxxx10	xxxxx	eeeeeeeeoooooooo (o=odd, e=even)	Load the congestion control throttles for ports 5 and 6
x1xxxxxxxxx11	xxxxx	eeeeeeeeoooooooo (o=odd, e=even)	Load the congestion control throttles for ports 7 and 8
1xxxxxxxxxxxx	xxxxx	xxxxxxxxdddddddd	Load congestion control enable bits
xxx1xxxxxxxxx	x1xxx	xxxxxxxxxxxxxxxx	Enable the TG and begin the test

After the TG is enabled (the final step in the initialization phase), it enters the packet generation phase. This phase lasts as long as necessary to generate the total (i.e., busy and idle) number of packets specified by the VME Slave during initialization step 3. During the packet generation phase any steps in Table 3 can be executed at any time (e.g., step 1 would reset the TG). However, step 4 will not have any effect because the LFSR initialization count is ignored during packet generation. Once the required number of packets has been generated, the TG enters a reset state called PORSET (see Figure 15), where it stays until the receipt of an enable command from the VME Slave.

I.3.1 Frame and Packet Format

The frame and packet structure used is the same as that used by COMSAT. A frame consists of a one-byte frame marker followed by four 58-byte packets (see SCAR Program Phase II Critical Design Review 4/14/94). A packet consists of a five-byte header and a 53-byte payload, with the header shown in Figure 4. All bits of the header are set to zero except for the route number, which is a variate distributed according to a completely-arbitrary, user-programmable pdf whose inverse distribution is stored in memory bank D. Note that this feature (in conjunction with the congestion control throttle mechanism) gives us exceptionable flexibility in dealing with congestion and modeling traffic sources. Idle packets have a route number of all zeroes. The remaining 53 bytes of the packet consist of all zeroes except for a time stamp in the two bytes immediately following the header. The stamp is a count of the number of BYTCLK transitions since the receipt of the enable signal.

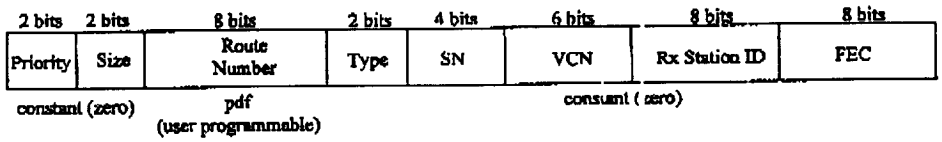

Figure 4: Packet Header Format

I.3.2 The MMPP and the Packet Generation Process

Packets are generated according to a Markov-Modulated Poisson Process (MMPP). This stochastic process is widely used in network performance research to model the bursty and correlative aspects of traffic which contains a mix of voice, video and data.

The MMPP has two states: "busy" and "idle". The former is characterized by higher rates of busy packet generation than the latter. The interarrival times between busy packets in either state are Poisson distributed. The times spent in either of the states are geometrically distributed. The number of packets (busy and idle) generated in either state will henceforth be referred to as the "state dwell time". The state diagram for the MMPP is given in Figure 5, where λ_1, λ_2 refer respectively to the Poisson parameters for the busy and idle state interarrival times, and α, β refer to the probabilities of leaving the busy and idle states, respectively.

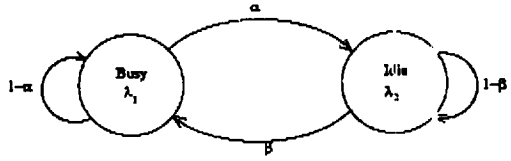

Figure 5: State Diagram for the MMPP

The TG manages the state dwell time using a counter which stores the total number m ($0 \le m \le 65535$) of packets to be generated in the state. The counter is loaded upon state entry with an integer-valued geometric deviate, chosen from either the busy state geometric distribution or the idle state geometric distribution, depending on the type of state being entered. Both types of deviate are stored in memory bank C, whose layout is discussed below.

The packet interarrival times are modeled by another counter (called the "idle packet counter") which contains the number n ($0 \le n \le 65535$) of consecutive idle packets that is to be generated before the next busy packet. The counter is loaded upon state entry and after each busy packet generated while remaining in the same state. The load values are integer-valued Poisson deviates, chosen from either the busy state Poisson distribution or the idle state Poisson distribution, depending on the state the TG is in. Both types of deviate are stored in memory bank C, whose layout will now be discussed.

The generation of the required deviates is done using the "inverse distribution" method, in which variates from an arbitrary distribution are obtained by evaluating the inverse of the distribution at uniformly-distributed values. Memory banks C and D store the inverse distribution values in look-up table fashion. A 32-bit LFSR generates uniform variates which address the banks. This look-up table approach is preferable to computational circuitry for this application because the memory is already available with the COMSAT hardware. This approach is also faster and user-programmable.

Memory bank C is laid out as shown in Table 5. Bank D is used for the route number distribution. Its map is not shown because it is trivial. Note that the distributions of the dwell times and idle packet counts are represented to 12-bit accuracy (i.e., memory address width), instead of the 14 used for the route number distribution.

Table 5: Map for Memory Bank C

Address R1C_ADDR[13..0]	Data
11ddddddddddd	Busy state dwell times
10ddddddddddd	Busy state idle packet counts
01ddddddddddd	Idle state dwell times
00ddddddddddd	Idle state idle packet counts

A high-level block diagram of the TG is shown in Figure 6, where it can be seen

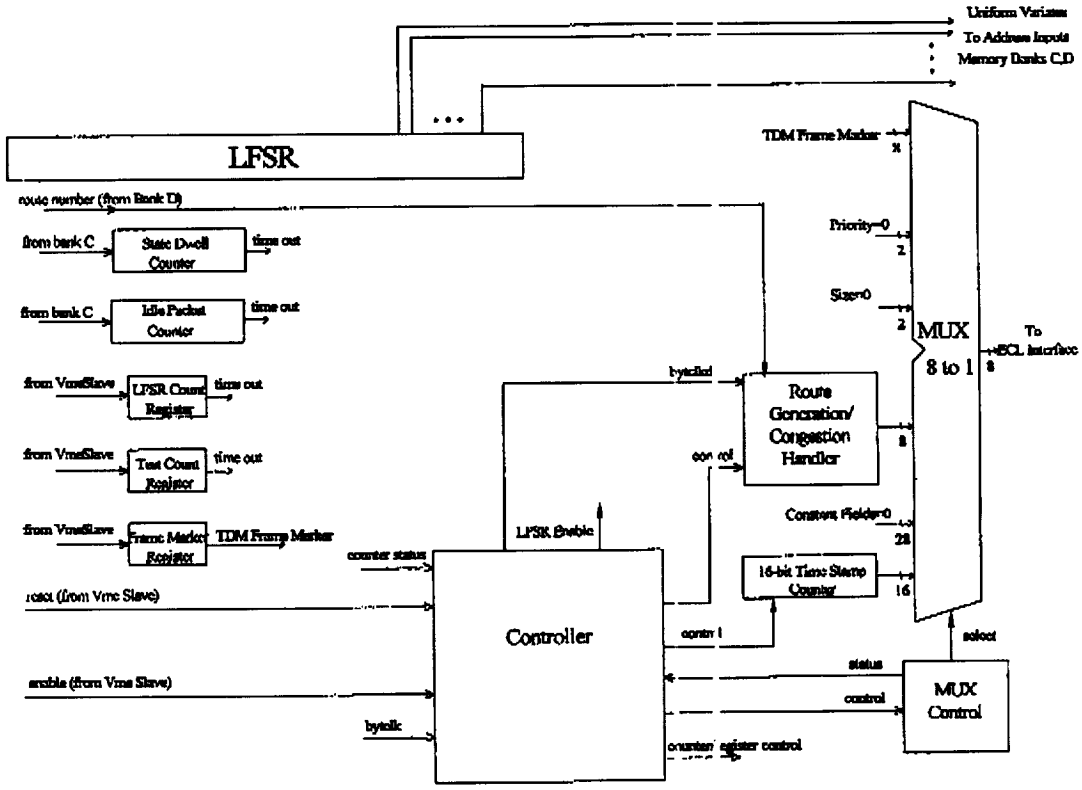

Figure 6: High-Level Block Diagram of the TG

that the TG generates all 58 bytes of a packet simultaneously in parallel. Conversion to byte-serial format (required by the ECL logic) is performed by an output multiplexer MUX which gives very high speed. The complicated parts of the design (e.g., the route number generator/congestion control handler ROUTGEN2) with long critical paths run

at half clock (signal BYTCLKD) because simulation has shown that they are too slow for full speed and cannot be simplified. Fortunately, the commencement of new route number generation/congestion control takes place as soon as the previous route number is output to the ECL interface. This fact allows as many as 57 byte times to be used for the process, which is more than enough.

I.3.3 Simulation Results

Figure A1 on the next page shows the bursty, stochastic nature of a sample packet output stream. The two bottom traces (ROUTSENT and active-low IDLESEND_BAR) indicate packet (idle or busy) times and idle packet times, respectively, for a simulation sequence which begins in the busy state for seven packets, dwells in the idle state for eight packets, and then returns to the busy state for another 7 packets. The difference in idle packet density between the two states is clear from the two traces.

Figure A2 shows the same simulation with the congestion control bit enabled for output port 3, and its throttle set to 1. The "pre-congestion control" route number (i.e., the byte accessed from memory bank D) is FF. By observing the trace of the output (TGDATA[7..0]) between the vertical lines it can be seen that the route fields in successive packets alternate between FF and 7F, indicating correct congestion control on output port 3 with a throttle of 1.

II. TG Design Details

The partitioning of the TG between the two CPLDs is shown in Figure 7.

Figure 7: Partitioning of the TG

II.1 TGDATPROC Details

The upper module (TGDATPROC) functions are given in Table 6. Tables 7 and 8 list the functions of the inputs and outputs, respectively.

Table 6: Functions of the TGDATPROC Module

TGDATPROC Functions
1. Generate packet bytes for ECL serialization logic
2. Compute packet time stamps
3. Generate packet route numbers with congestion control
4. Store congestion control throttle numbers
5. Store frame marker

Table 7: TGDATPROC Inputs and their Functions

Inputs	Function
VMEDATBAR[15..0]	initializing data from VME Slave (frame marker, congestion control enable bits and throttle settings for channels)
VMEADRBAR[15..0]	addresses for setting of frame marker, congestion bits and throttle settings
BYTCLKD	BYTCLK divided-by-2 for congestion control (see ROUTGEN2 module below)
BUSYSEND_BAR	initializes busy packet generation
RID_DATA[7..0]	memory bank D data output lines which contain the statistically-generated routing number (before congestion processing)
IDLESEND_BAR	initializes idle packet generation
EN_ROUTE_BAR	starts the route number generation
RESET_BAR	global reset for TG (see CNTRLER below)
BYTCLK	system clock (20 Mhz)
TMSTMPENA_BAR	starts the timer (used to stamp the outgoing packets)
ENADNCNTR_BAR	starts the packet generation

Table 8: TGDATPROC Outputs and their Functions

Outputs	Function
TGDRDYBAR	output data ready flag
TGDATA[7..0]	output data to ECL serialization logic
ROUTDONE	controller signal to acknowledge the completion of the route number congestion control adjustment process
ROUTSENT	controller signal to acknowledge the sending of the route number in the output stream
TGUWFLGBAR	frame boundary indicator

The schematic of TGDATPROC is shown in Figure 8. The ROUTGEN2 schematic is shown in Figure 9. ROUTGEN2 is responsible for throttle storage, congestion control and the generation of the appropriately modified route number

accessed from memory bank D, which generates route nu nbers according to the user-specified statistical distribution.

The congestion control scheme is as follows. The VME Slave loads throttle settings for each of the eight switch output ports in ROUTGEN2 at any time, including reset. Throttle value n causes an idle packet to be inserted every n successive busy packets destined for that port. Throttle control is enabled only when the throttle enable bit for that port is asserted. The throttle enable bits are loaded from the VME Slave at any time, and stored in ROUTGEN2.

This process is implemented by a loopback scheme. Each bit of the routing number (accessed from the memory bank D) is sequentially tested. If a particular bit is set, then it is reset only if that port's throttle is enabled and the number of consecutive busy packets destined for that port that have been already sent equals the throttle setting. The number of packets already sent is called the "throttle count" and is maintained in ROUTGEN2 by a down counter. If reset is performed, the throttle count is restored to its original (VME Slave-provided) value. If not, the count is decremented. The route number computed by this process is output from the TG.

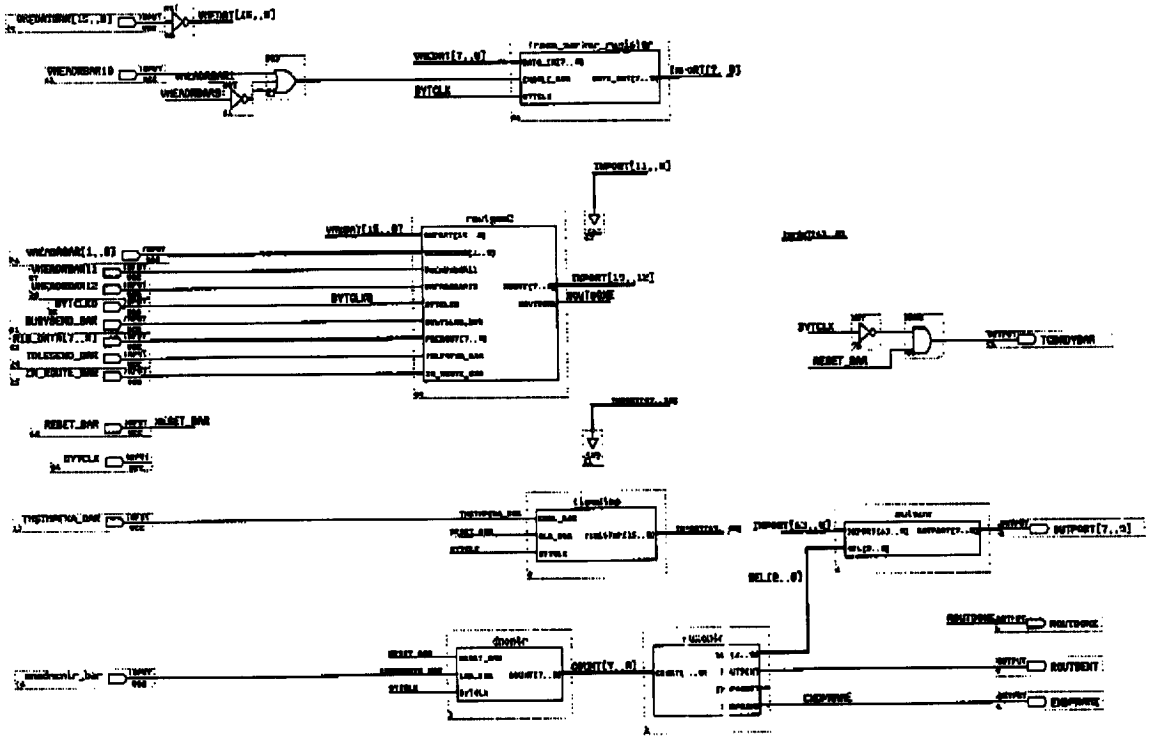

Figure 8: Schematic of the TGDATPROC Module

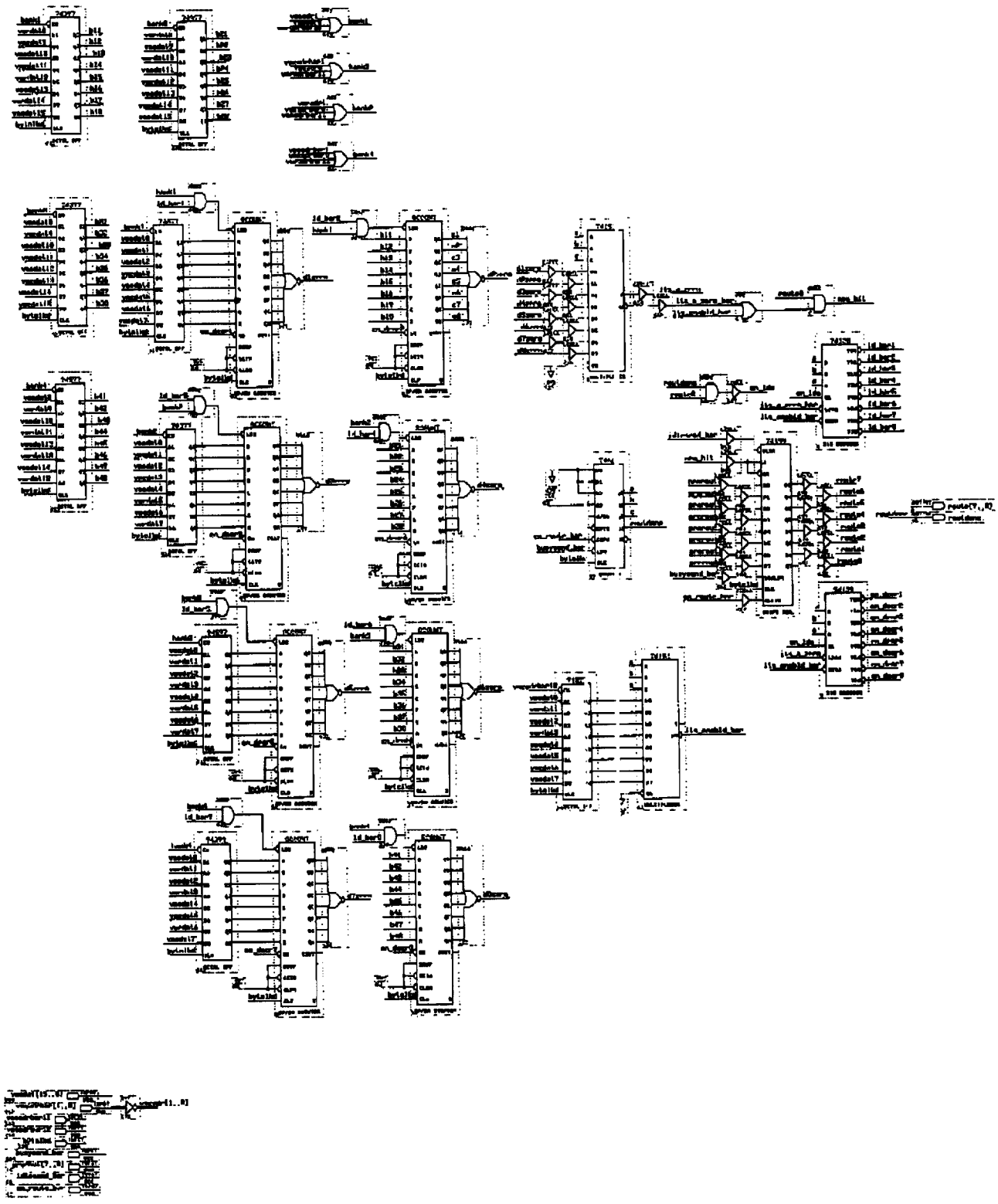

Figure 9: Schematic of the ROUTGEN2 Module

Other TGDATPROC subsystems are straightforward except for MUXCNTRL, whose text listing is shown in Figure 10. MUXCNTRL is designed to convert the output of the downcounter DNCNTR to appropriate select signals for the output multiplexer OUTMUX. The design is not trivial because of the necessity to insert the frame header every four packets.

```
SUBDESIGN muxcntrl
(
        count[7..0]                             : INPUT;
        sel[2..0], routsent                     : OUTPUT;
        endpacket, endframe                     : OUTPUT;
)
BEGIN
        CASE count[] IS
                WHEN 0 => sel[] = 7;
                                endframe = VCC;
                                endpacket = VCC;
                WHEN 52, 110, 168, 226 => sel[] = 6;
                WHEN 53, 111, 169, 227 => sel[] = 5;
                WHEN 54, 112, 170, 228 => sel[] = 4;
                WHEN 55, 113, 171, 229 => sel[] = 3;
                                routsent = VCC;
                WHEN 56, 114, 172, 230 => sel[] = 2;
                WHEN 57, 115, 173, 231 => sel[] = 1;
                WHEN 58, 116, 174 => sel[] = 7;
                                endpacket = VCC;
                WHEN 232 => sel[] = 0;
                WHEN OTHERS => sel[] = 7;
        END CASE;
END;
```

Figure 10: Module OUTMUX Design

II.2 TGRAMCTL Details

The lower module (TGRAMCTL) functions are given in Table 9. Tables 10 and 11 list the functions of the inputs and outputs, respectively.

Table 9: Functions of the TGRAMCTL Module

TGRAMCTL Functions
1. Generate uniform variates for addresses to memory banks C and D, thus procuring suitably distributed state dwell times and idle packet counts
2. Control the movement between and the time in the Markov states ("bursty" and "quiet"). Also control the generation of idle packet sequences
3. Control the generation of busy packets
4. Control the initialization of the TG, and store the LFSR initialization count.
5. Store the test length count
6. Generate a clock signal BYTCLKD with half the frequency of BYTCLK for use by ROUTGEN2, which due to its loopback structure is slow.
7. Perform all other control functions for the TG, for example reset, initialization, and shutdown

Table 10: TGRAMCTL Inputs and their Functions

Inputs	Function
BYTCLK	system clock (20 Mhz)
VMEADRBAR[15..0]	addresses for storing LFSR initialization and length of test counts, as well as resetting and enabling the TG
VMECTRLBAR[4..0]	control bits for resetting and enabling the TG
ENDFRAME	flag indicating that the last byte in a frame has just been sent to the ECL serialization circuitry. Used by CNTRLER to perform an orderly reset sequence
ROUTDONE	flag which when asserted indicates that the route number is being processed by ROUTGEN2 according to the loopback congestion control procedure discussed above.
R1C_DATA[15..0]	memory bank C data output lines which contain (at disjoint times) either the statistically-generated Markov state dwell times, or the statistically-generated idle packet sequence counts
ROUTSENT	flag indicating that the route number byte has just been passed to the ECL serialization circuitry. Used by CNTRLER to begin preparing another idle or busy packet
VMEDATBAR[15..0]	initializing data from VME Slave (test count length and LFSR initialization count)

Table 11: TGRAMCTL Outputs and thei: Functions

Outputs	Function
R1C_CE	Memory bank C enable line
R1D_CE	Memory bank D enable line
R1D_ADDR[13..0]	Address lines for memory bank D. The lower eight of these lines are connected to selected LFSR outputs and thus contain uniform variates for generating the statistically-distributed routing numbers
R1C_ADDR[13..0]	Address lines for memory bank C. These lines are connected to selected LFSR outputs and thus contain uniform variates for generating the statistically-distributed Markov state dwell times and corresponding idle packet counts
RESET_BAR	TG reset signal generated by CNTRLER
TMSTMPENA_BAR	enable signal for the time stamp counter, generated by CNTRLER.
ENADNCNTR_BAR	enable signal for the down counter which sequences MUXCNTRL and OUTMUX in TGDATPROC; generated by CNTRLER
IDLESEND_BAR	enable signal for the generation of an idle packet. Generated by CNTRLER and sent to ROUTGEN2 in TGDATPROC
EN_ROUTE_BAR	enable signal to begin the loopback congestion control procedure discussed above on the routing number within the ROUTGEN2 module in TGDATPROC; generated by CNTRLER
BUSYSEND_BAR	enable signal for the generation of a busy packet. Generated by CNTRLER and sent to ROUTGEN2 in TGDATPROC
BYTCLKD	BYTCLK divided-by-2 for congestion control (see ROUTGEN2 module above)

The schematic of TGRAMCTL is shown in Figure 11. The LFSR schematic is

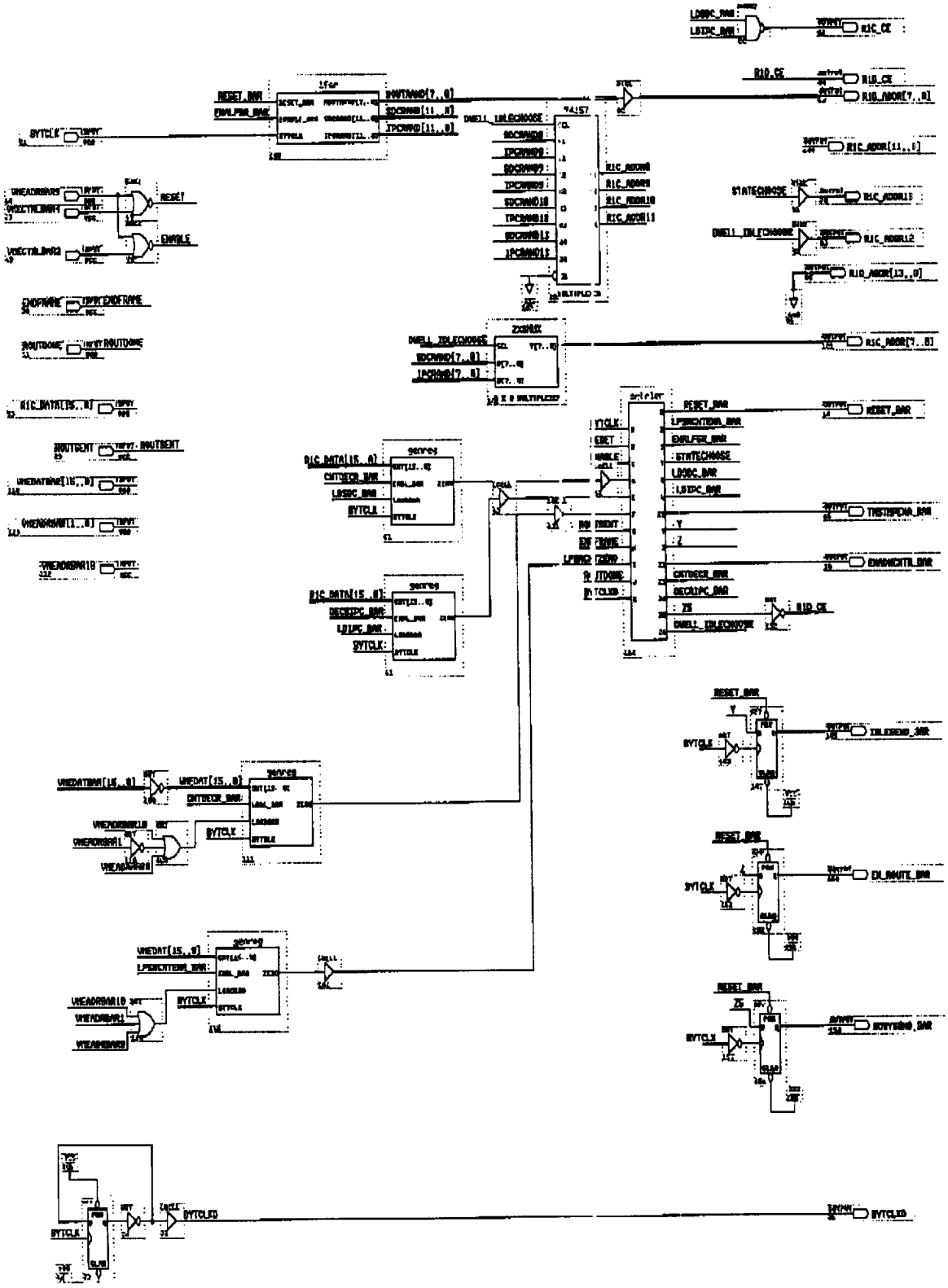

Figure 11: Schematic of the TGRAMCTL Module

is shown in Figure 12. It is 32 bits long and has a period of $2^{32}-1$. The tap locations which are used to form the lower 12 bits of the C memory bank address (see discussion below) and the lower eight bits of the D memory bank address (see discussion below) are as indicated in the figure.

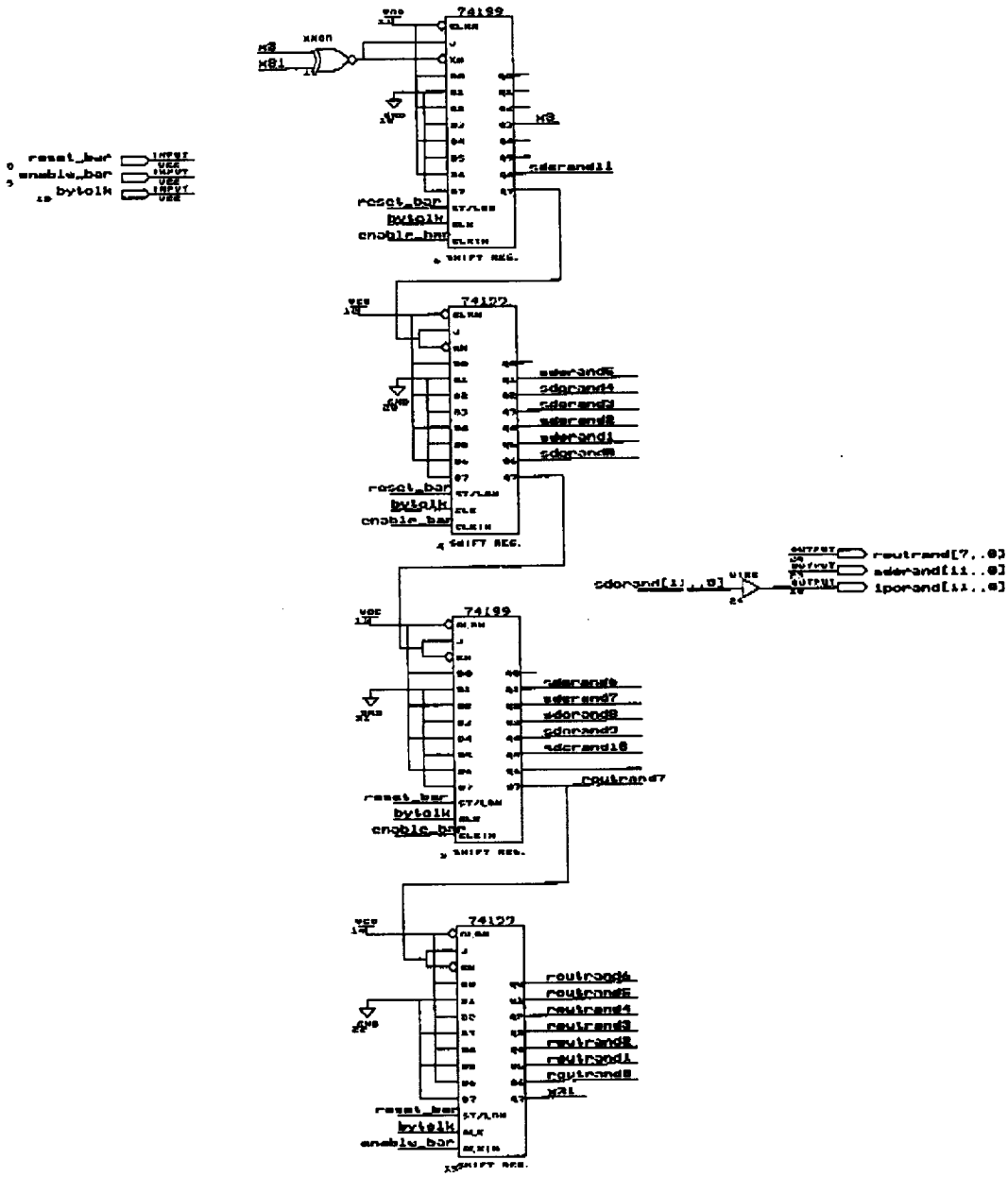

Figure 12: LFSR Schematic Diagram

The four GENREG modules are identical "generic" 16-bit down counters whose schematic will not be shown because of its simplicity. They function as the Markov state dwell time counter, idle packet counter, test length counter and LFSR initialization counter, respectively reading top-to-bottom on the TGRAMCTL schematic.

The text for the CNTRLER module is given in Figure 13. This module is a Finite State Machine (FSM) which is the controller for the TG. A Mealy type architecture (i.e., outputs change in response to input changes without changing state) was chosen because it is easier to understand and debug. The state diagram for CNTRLER is given in Figure 14, and the dictionary for its output symbols is given in Table 12. The output PRELOC in the table represents output l in Figure 13.

```
SUBDESIGN cntrler
%                                                                    %
%Renaming of inputs and outputs:                                     %
%a=bytclk                                                           %
%b=reset                                                            %
%c=enable                                                           %
%d=tosdc                                                            %
%e=ipczero                                                          %
%f=tstcntzero                                                       %
%g=routsent                                                         %
%h=endframe                                                         %
%i=lfsrcntzero                                                      %
%j=routdone                                                         %
%k=bytclkd                                                          %
%l=toggle control for location                                      %
%n=reset_bar                                                        %
%r=lfsrcntena_bar                                                   %
%s=enalfsr_bar                                                      %
%t=statechoose                                                      %
%v=ldsdc_bar                                                        %
%w=ldipc_bar                                                        %
%xi=tmstmpena_bar                                                   %
%y=idlesend_bar                                                     %
%z=en_routc_bar                                                     %
%z1=enadncntr_bar                                                   %
%z3=cntdecr_bar                                                     %
%z4=decripc_bar                                                     %
%z5=busysend_bar                                                    %
%z6=dwell_idlechoose                                                %
%                                                                   %

(
        a, b, c, d, e, f                    : INPUT;
        g, h, i, j, k                       : INPUT;
```

```
        n, r, s                                  : OUTPUT;
        t, v, w, xi, y                           : OUTPUT;
        z, z1, z3, z4, z5, z6                    : OUTPUT;
)
VARIABLE
        ss : MACHINE WITH STATES (porset, init, Apre, As, Bs1, Bs2, Bs, Cpre,
Csend, Cs, Cpost, Cwait, Ds, Ds1, Dsend, preot);
        l : NODE;


BEGIN
        ss.clk = a;
        ss.reset = b;

        t = TFFE(VCC, a, VCC, VCC, l);

        TABLE
%  current     current                              next          current %
%  state       input                                state         output  %
%                                                                       %
ss,       c,d,e,f,g,h,i,j,k =>    ss,     l,  n,r,s,v,w,xi,y,z,z1,z3,z4,z5,z6;

porset,  0,x,x,x,x,x,x,x,x =>  porset, 0,  0,1,0,1,1,1, 1,1, 1, 1, 1, 1, 1;
porset,  1,x,x,x,x,x,x,x,x =>  init,    0,  1,1,1,1,1,1,0, 1,1, 0, 1, 1, 1, 1;
init,    x,x,x,1,x,x,x,x,x =>  preot,   0,  1,1,1,1,1,1,1, 1,1, 0, 1, 1, 1, 1;
init,    x,x,x,0,x,x,0,x,x =>  init,    0,  1,0,0,1,1,1,0, 1,1, 0, 1, 1, 1, 1;
init,    x,x,x,0,x,x,1,x,x =>  Apre,    1,  1,1,0,1,1,1,0, 1,1, 0, 1, 1, 1, 1;
Apre,    x,x,x,x,x,x,x,x,x =>  As,      0,  1,1,1,1,0,1,0, 1,1, 0, 1, 1, 1, 1;
As,      x,1,x,x,x,x,x,x,x =>  Apre,    1,  1,1,0,1,1,1,0, 1,1, 0, 1, 1, 1, 1;
As,      x,0,x,x,x,x,x,x,x =>  Bs1,     0,  1,1,0,1,1,1,0, 1,1, 0, 1, 1, 1, 1;
Bs1,     x,x,x,x,x,x,x,x,x =>  Bs2,     0,  1,1,1,1,1,0,0, 1,1, 0, 1, 1, 1, 0;
Bs2,     x,x,x,x,x,x,x,x,x =>  Bs,      0,  1,1,1,1,1,1,0, 1,1, 0, 1, 1, 1, 1;
Bs,      x,x,0,x,x,x,x,x,0 =>  Cpre,    0,  1,1,1,1,1,1,0, 0,1, 0, 1, 1, 0, 1;
Bs,      x,x,0,x,x,x,x,x,1 =>  Bs,      0,  1,1,1,1,1,1,0, 0,1, 0, 1, 1, 1, 0, 1;
Bs,      x,x,1,x,x,x,x,x,0 =>  Cwait,   0,  1,1,1,1,1,1,0, 1,0, 0, 1, 1, 1, 0, 1;
Bs,      x,x,1,x,x,x,x,x,1 =>  Bs,      0,  1,1,1,1,1,1,0, 1,0, 0, 1, 1, 1, 0, 1;
Cpre,    x,x,x,x,x,x,x,x,x =>  Csend,   0,  1,1,1,1,1,1,0, 0,0, 0, 1, 1, 1, 1, 1;
Csend,   x,x,x,x,x,x,x,1,x =>  Csend,   0,  1,1,1,1,1,1,0, 0,0, 0, 1, 1, 1, 1, 1;
Csend,   x,x,x,x,x,x,x,0,x =>  Cs,      0,  1,1,1,1,1,1,0, 1,1, 0, 1, 1, 1, 1, 1;
Cs,      x,x,x,x,0,x,x,x,x =>  Cs,      0,  1,1,1,1,1,1,0, 1,1, 0, 1, 1, 1, 1, 1;
Cs,      x,x,x,x,1,x,x,x,x =>  Cpost,   0,  1,1,1,1,1,1,0, 1,1, 0, 0, 0, 1, 1, 1;
Cpost,   x,x,x,1,x,x,x,x,x =>  preot,   0,  1,1,1,1,1,1,1, 1,1, 0, 1, 1, 1, 1, 1;
Cpost,   x,1,x,0,x,x,x,x,x =>  Apre,    1,  1,1,0,1,1,1,0, 1,1, 0, 1, 1, 1, 1, 1;
Cpost,   x,0,1,0,x,x,x,x,0 =>  Cwait,   0,  1,1,1,1,1,1,0, 1,0, 0, 1, 1, 1, 0, 1;
Cpost,   x,0,1,0,x,x,x,x,1 =>  Cpost,   0,  1,1,1,1,1,1,0, 1,0, 0, 1, 1, 1, 0, 1;
```

```
Cpost,  x,0,0,0,x,x,x,x,0 =>  Cpre,   0,  1,1,1,1,1,0, 0,1, 0, 1, 1, 0, 1;
Cpost,  x,0,0,0,x,x,x,x,1 =>  Cpost,  0,  1,1,1,1,1,0, 0,1, 0, 1, 1, 0, 1;
Cwait,  x,x,x,x,x,x,x,x,x =>  Dsend,  0,  1,1,1,1,1,0, 1,0, 0, 1, 1, 1, 1;
Dsend,  x,x,x,x,x,x,x,1,x =>  Dsend,  0,  1,1,1,1,1,0, 1,0, 0, 1, 1, 1, 1;
Dscnd,  x,x,x,x,x,x,x,0,x =>  Ds,     0,  1,1,1,1,1,0, 1,1, 0, 1, 1, 1, 1;
Ds,     x,x,x,x,0,x,x,x,x =>  Ds,     0,  1,1,1,1,1,0, 1,1, 0, 1, 1, 1, 1;
Ds,     x,x,x,x,1,x,x,x,x =>  Ds1,    0,  1,1,1,1,1,0, 1,1, 0, 1, 1, 1, 1;
Ds1,    x,x,x,1,x,x,x,x,x =>  preot,  0,  1,1,1,1,1,1, 1,1, 0, 1, 1, 1, 1;
Ds1,    x,1,x,0,x,x,x,x,x =>  Apre,   1,  1,1,0,1,1,0, 1,1, 0, 1, 1, 1, 1;
Ds1,    x,0,x,0,x,x,x,x,x =>  Bs1,    0,  1,1,0,1,1,0, 1,1, 0, 1, 1, 1, 1;
preot,  x,x,x,x,x,1,x,x,x =>  porset, 0,  1,1,1,1,1,1, 1,1, 1, 1, 1, 1, 1;
preot,  x,x,x,x,x,0,x,x,x =>  preot,  0,  1,1,1,1,1,1, 1,1, 0, 1, 1, 1, 1;

        END TABLE;
END;
```

Figure 13: Text of the CNTRLER Module Design

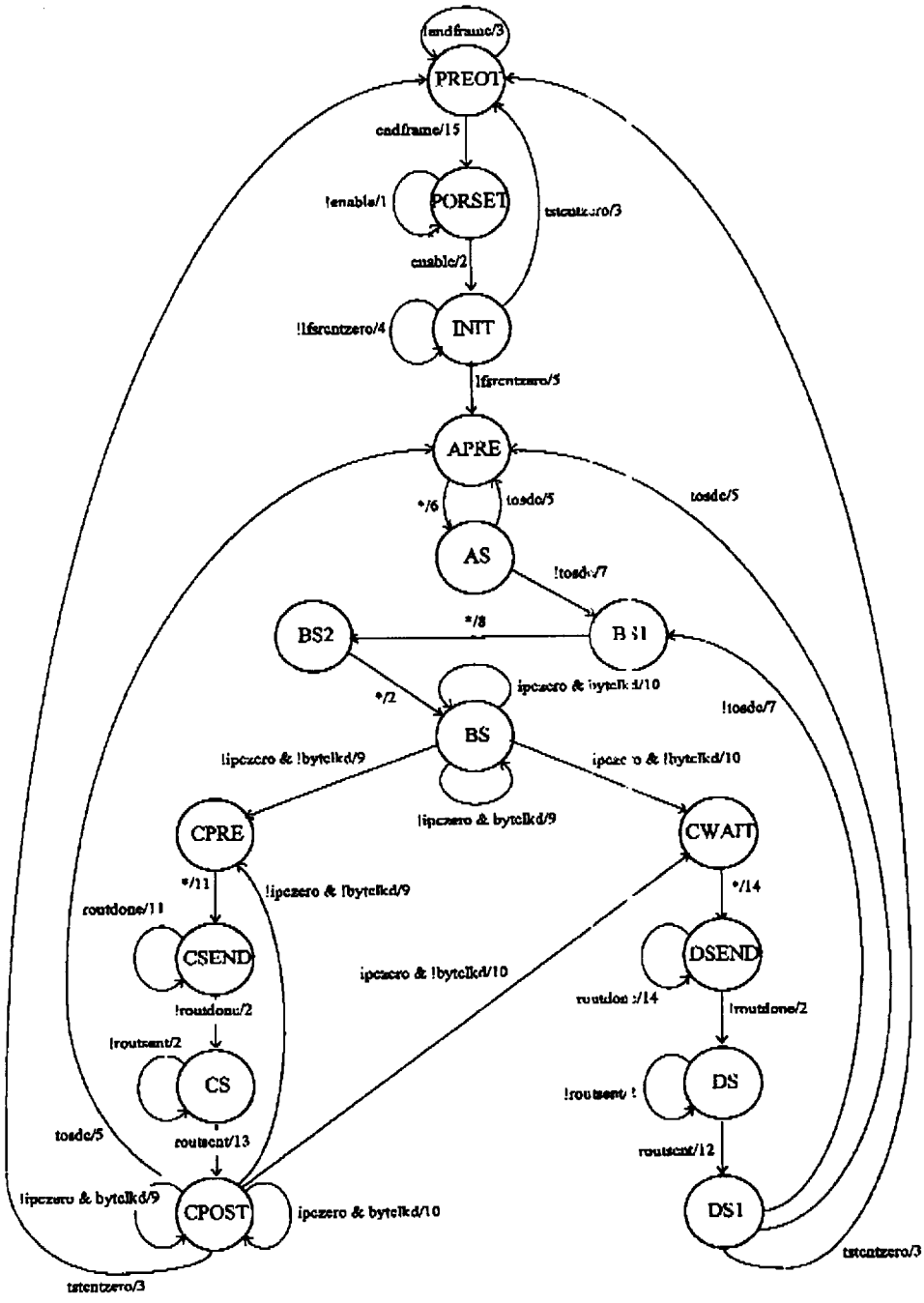

Figure 14: State Diagram for the CNTRLER Module

Table 12: Dictionary of the Output Codes in Figure 10

Code	Signals Asserted
1.	RESET_BAR, ENALFSR_BAR
2.	TMSTMPENA_BAR, ENADNCNTR_BAR
3.	ENADNCNTR_BAR
4.	LFSRCNTENA_BAR, ENALFSR_BAR, TMSTMPENA_BAR, ENADNCNTR_BAR
5.	PRELOC, ENALFSR_BAR, TMSTMPENA_BAR, ENADNCNTR_BAR
6.	LDSDC_BAR, TMSTMPENA_BAR, ENADNCNTR_BAR
7.	ENALFSR_BAR, TMSTMPENA_BAR, ENADNCNTR_BAR
8.	LDIPC_BAR, TMSTMPENA_BAR, ENADNCNTR_BAR, DWELL_IDLECHOOSE
9.	TMSTMPENA_BAR, IDLESEND_BAR, ENADNCNTR_BAR, BUSYSEND_BAR
10.	TMSTMPENA_BAR, ENROUTE_BAR, ENADNCNTR_BAR, BUSYSEND_BAR
11.	TMSTMPENA_BAR, IDLESEND_BAR, ENROUTE_BAR, ENADNCNTR_BAR
12.	TMSTMPENA_BAR, ENADNCNTR_BAR, CNTDECR_BAR
13.	TMSTMPENA_BAR, ENADNCNTR_BAR, CNTDECR_BAR, DECRIPC_BAR
14.	TMSTMPENA_BAR, ENROUTE_BAR, ENADNCNTR_BAR
15.	none

II.3 Future Work

Opportunities for future work exist with the present implementation, as well as with a standard-cell ASIC implementation.

II.3.1 Opportunities With the Present TG Implementation

There are three major enhancements that can be made. The first is the incorporation of different packet priorities and lengths. Memory bank D could store the user-programmable probability distributions for the priorities and lengths, in addition to the route number distribution that it now contains. The loss (2 bits) in pdf resolution would be inconsequential because the present value (14 bits) exceeds our needs.

The second major enhancement is the widening of the Time Stamp Counter to 40 bits. Its present length of 16 bits was chosen because it allowed the design to fit in the

7256 CPLDs. However, the length of the longest test is less than a second. Widening the counter would increase the test length to over 15 hours, which is ample enough to study long-term effects of congestion handling procedures on many different mixes of packet types and priorities. The LFSR and its initialization counter should also be widened, in order to guarantee long-term uniformity of the generated addresses.

The third major enhancement is the inclusion of extra states in the Markov process. Three or more states may allow more accurate modeling of traffic and more thorough testing of the switch. The literature on multi-state MMPPs is scanty and this presents an opportunity for significant research and publication.

These enhancements will require bigger CPLDs. The utilization percentages for the present design (88% and 47%) are high enough that further modifications might make routing/fitting/placement very difficult. Routing/fitting of the present design already is tight as evidenced by the fact that it consumed 40 man-hours and dictated placement of the clock division flip-flop on the device other than the one that needed it. The effect of the enhancements on the speed of the TG will be minimal because of the parallel architecture.

II.3.2 Opportunities for the Development of a Standard-Cell TG Implementation

A standard-cell ASIC implementation of the TG would allow the incorporation of the enhancements mentioned above (as well as others) without routing or board trace/space problems. It would also allow the testing of future switches which require higher speeds, and the multi-state Markov process would generate more realistic traffic.