...
...

Process Model Attributes

attribute	*value*	*type*	*default value*
sync bandwidth	promoted	double	0.001 (%)
T_Req	promoted	double	4.0 (sec)
station_address	promoted	integer	0
ring_id	promoted	integer	0
promiscuous	promoted	toggle	disabled
encap_decap	promoted	toggle	enabled
GWY	promoted	integer	0

Header Block

```
/* Define a timer structure used to implement */
/* the TRT and THT timers.  The primitives defined to */
/* operate on these timers can be found in the */
/* function block of this process model. */
typedef struct
	{
	int			enabled;
	double			start_time;
	double			accum;
	double			target_accum;
	} FddiT_Timer;


/* Declare certain primitives dealing with timer.s */
double			fddi_timer_remaining ();
FddiT_Timer*	fddi_timer_create ();
double			fddi_timer_value ();

/* Scratch strings for trace statements */
char			str0 [512], str1 [512];


/* define constants particular to this implementation */
#define FDDI_MAX_STATIONS		512

 /*Multicasting frp_id instat*/
#define			UPDATE_INSTAT	0

/* define possible values for the frame control field */
#define FDDI_FC_FRAME			0
#define FDDI_FC_TOKEN			1

/* define possible service classes for frames */
#define FDDI_SVC_ASYNC			0
#define FDDI_SVC_SYNC			1

/* define input stream indices */
#define FDDI_LLC_STRM_IN		1
#define FDDI_PHY_STRM_IN		0

/* define output stream indices */
#define FDDI_LLC_STRM_OUT		1
#define FDDI_PHY_STRM_OUT		0

/* define token classes */
```

```
#define FDDI_TK_NONRESTRICTED      0
#define FDDI_TK_RESTRICTED         1

/* Ring Constants */
#define FDDI_TX_RATE               1.0e+08
#define FDDI_SA_SCAN_TIME          28.0e-08

/* Token transmission time: based on 6 symbols plus 16 symbols of preamble */
#define FDDIC_TOKEN_TX_TIME        88.0e-08


/* Codes used to differentiate remote interrupts */
#define FDDIC_TRT_EXPIRE           0
#define FDDIC_TK_INJECT            1


/* Define symbolic expressions used on transition */
/* conditions and in executive statements. */
#define TRT_EXPIRE                                                                  \
    (op_intrpt_type () == OPC_INTRPT_REMOTE && op_intrpt_code () == FDDIC_TRT_EXPIRE)

#define TK_RECEIVED                                     \
    phy_arrival &&                                      \
    frame_control == FDDI_FC_TOKEN

#define RC_FRAME                                        \
    phy_arrival &&                                      \
    frame_control == FDDI_FC_FRAME

#define FRAME_ARRIVAL                                   \
    op_intrpt_type () == OPC_INTRPT_STRM &&             \
    op_intrpt_strm () == FDDI_LLC_STRM_IN

#define STRIP     my_address == src_addr

#define   FORWARD                                       \
    (dest_addr == my_address) ||                        \
    (promiscuous == OPC_TRUE)

#define DECAP     encap_decap == OPC_TRUE

#define   REPEAT dest_addr != my_address


/* Define the maximum value for ring_id.  This is the      */
/* maximum number of FDDI rings that can exist in a        */
/* simulation.  Note that if this number is changed,       */
/* the initialization for fddi_claim_start below must      */
/* also be modified accordingly.                           */
#define FDDI_MAX_RING_ID              60

/* Declare the operative TTRT value 'T_Opr' which is the final          */
/* negotiated value of TTRT.  This value is shared by all stations      */
/* on a ring so that all agree on its value.                            */
double        fddi_t_opr [FDDI_MAX_RING_ID];
#define       Fddi_T_Opr                (fddi_t_opr [ring_id])

/* This flag indicates that the negotiation for the final TTRT       */
/* has not yet begun.  It is statically initialized here, and        */
```

```
/* is reset by the first station which modifies T_Opr.                */
/* Initialize to 1 for all rings.                                     */
int          fddi_claim_start [FDDI_MAX_RING_ID] = {1,1,1,1,1,1,1,1,1,1,1,1,
                                                    1,1,1,1,1,1,1,1,1,1,1,1,
                                                    1,1,1,1,1,1,1,1,1,1,1,1,
                                                    1,1,1,1,1,1,1,1,1,1,1,1,
                                                    1,1,1,1,1,1,1,1,1,1,1,1};

#define      Fddi_Claim_Start              (fddi_claim_start [ring_id])


/* Declare station latency parameters.                                */
/* These are true globals, so they do not need to be arrays.          */
double       Fddi_St_Latency;
double       Fddi_Prop_Delay;

/* Declare globals for Token Acceleration Mechanism.          */
/* Hop delay and token acceleration are true globals.         */
double       Fddi_Tk_Hop_Delay;
int          Fddi_Tk_Accelerate = 1;

/* These are actually values shared by all nodes on a ring,        */
/* so they must be defined as arrays.                              */
double       fddi_tk_block_base_time [FDDI_MAX_RING_ID];
#define      Fddi_Tk_Block_Base_Time       (fddi_tk_block_base_time [ring_id])

int          fddi_tk_block_base_station [FDDI_MAX_RING_ID];
#define      Fddi_Tk_Block_Base_Station    (fddi_tk_block_base_station [ring_id])

int          fddi_tk_blocked [FDDI_MAX_RING_ID];
#define      Fddi_Tk_Blocked               (fddi_tk_blocked [ring_id])

int          fddi_min_station [FDDI_MAX_RING_ID];
#define      Fddi_Min_Station              (fddi_min_station [ring_id])

int          fddi_num_stations [FDDI_MAX_RING_ID] = {0,0,0,0,0,0,0,0};
#define      Fddi_Num_Stations             (fddi_num_stations [ring_id])

int          fddi_num_registered [FDDI_MAX_RING_ID];
#define      Fddi_Num_Registered           (fddi_num_registered [ring_id])

Objid        fddi_address_table [FDDI_MAX_RING_ID][FDDI_MAX_STATIONS];
#define      Fddi_Address_Table            (fddi_address_table [ring_id])


/* Station addresses may begin at any integer value; however for the            */
/* arrays that are indexed by station address, a normalization is               */
/* defined by shifting the addresses into the range [0, num_stations-1].        */
/* The following two macros are used to convert to and from normalized         */
/* address form.                                                                */
#define FDDI_ADDR_NORM(a)        (a - Fddi_Min_Station)
#define FDDI_ADDR_DENORM(a)      (a + Fddi_Min_Station)


/* Event handles for the TRT are maintained at a global level to          */
/* allow token acceleration mechanism to adjust these as necessary        */
/* when blocking and reinjecting the token. TRT_handle simply             */
/* represents the TRT for the local MAC                                   */
Evhandle     fddi_trt_handle [FDDI_MAX_RING_ID][FDDI_MAX_STATIONS];
```

...
...

```
#define     Fddi_Trt_Handle          (fddi_trt_handle [ring_id])
#define     TRT_handle               Fddi_Trt_Handle [FDDI_ADDR_NORM (my_address)]

/* Similarly, the TRT data structure is maintained on a global level. */
FddiT_Timer*    fddi_trt [FDDI_MAX_RING_ID] [FDDI_MAX_STATIONS];
#define         Fddi_Trt             (fddi_trt [ring_id])
#define         TRT                  Fddi_Trt [FDDI_ADDR_NORM (my_address)]

/* Registers to record the expiration time of each TRT when token is blocked. */
double      fddi_trt_exp_time [FDDI_MAX_RING_ID] [FDDI_MAX_STATIONS];
#define     Fddi_Trt_Exp_Time        (fddi_trt_exp_time [ring_id])

/* the 'Late_Ct' flag is declared on a global level so that it can be          */
/* set at the time where the token is injected back into the ring.            */
int         fddi_late_ct [FDDI_MAX_RING_ID] [FDDI_MAX_STATIONS];
#define     Fddi_Late_Ct             (fddi_late_ct [ring_id])
#define     Late_Ct                  Fddi_Late_Ct [FDDI_ADDR_NORM (my_address)]


/* Convenient macro for setting TRT for a given station and absolute time. */
#define TRT_SET(station_id,abs_time)                                                        \
fddi_timer_set (Fddi_Trt [FDDI_ADDR_NORM (station_id)], abs_time - op_sim_time());          \
Fddi_Trt_Handle [FDDI_ADDR_NORM (station_id)] = op_intrpt_schedule_remote (abs_time,
        FDDIC_TRT_EXPIRE, Fddi_Address_Table [FDDI_ADDR_NORM (station_id)]);
```

State Variable Block

```
/* Ring ID for this station. */
int                 \ring_id;

/* Timer-Related state variables */
FddiT_Timer*        \THT;
double              \T_Req;


/* Priority indirection table mapping integer */
/* priority classes to THT thresholds. */
double              \T_Pri [8];

/* Initialization and Identification */
Objid               \my_objid;
int                 \spawn_token;
int                 \my_address;

/* Packet representing token */
Packet*             \tk_pkptr;

/* Synchronous Bandwidth Allocation */
double              \sync_bandwidth;
double              \sync_pc;

/* Restricted Token Mode Variables */
int                 \restricted;
int                 \res_peer;
```

```
/* Token Acceleration. */
int                         \tk_registered;

/* Interface to LLC */
Ici*                        \to_llc_ici_ptr;


/* Tracing (not part of state but making it an SV              */
/* allows it to be used in the FB).                            */
int                         \tk_trace_on;

/* Flags for promiscuous and encapsulation modes */
int             \promiscuous;
int             \encap_decap;

int                 \MC[100];
int                 \upd,\is_gwy;
```

Temporary Variable Block

```
/* Packets and ICI's */
Packet*             mac_frame_ptr;
Packet*             pdu_ptr;
Packet*             pkptr;
Packet*             tmp_pkptr;
Packet*             data_pkptr;
Ici*                ici_ptr;

/* Packet Fields and Attributes */
int                 req_pri, svc_class, req_tk_class;
int                 frame_control, src_addr, dest_addr;
int                 pk_len, pri_level;

int             grp;

/* Token - Related */
int                 tk_usable, res_station, tk_class;
int                 current_tk_class;
double              accum_sync;

/* Timer - Related */
double              tx_time, timer_remaining, accum_bandwidth;
double              tht_value;

/* Miscellaneous */
int                 i;
int                 spawn_station, phy_arrival;
char                error_string [512];
int                 num_frames_sent, num_bits_sent;
```

Function Block

```
/** The procedures defined in this section serve **/
/** to simplify the code in the main body of the **/
/** process model by providing primitives for timer **/
/** manipulation. **/


fddi_timer_disable (timer_ptr)
	FddiT_Timer*		timer_ptr;
	{
	/* if the timer is already disabled, do nothing */
	if (timer_ptr->enabled)
		{
		/* disable the timer */
		timer_ptr->enabled = 0;

		/* reassign the accumulated time so far */
		timer_ptr->accum = op_sim_time () - timer_ptr->start_time;
		}
	}


fddi_timer_enable (timer_ptr)
	FddiT_Timer*		timer_ptr;
	{
	/* if the timer is already enabled, simply return */
	if (!timer_ptr->enabled)
		{
		/* reenable the timer */
		timer_ptr->enabled = 1;

		/* set the start time to the current time */
		/* less the accumulated time so far */
		timer_ptr->start_time = op_sim_time () - timer_ptr->accum;
		}
	}


fddi_timer_expired (timer_ptr)
	FddiT_Timer*		timer_ptr;
	{
	if (fddi_timer_remaining (timer_ptr) <= 0.0)
		return 1;
	else return 0;
	}


double
fddi_timer_remaining (timer_ptr)
	FddiT_Timer*		timer_ptr;
	{
	/* if the timer is enabled, update the accumulated time */
	if (timer_ptr->enabled)
		{
		timer_ptr->accum = op_sim_time () - timer_ptr->start_time;
		}

	/* return the timer remaining before expiration */
```

```
	/* a non-positive value indicates an expired timer */
	return (timer_ptr->target_accum - timer_ptr->accum);
	}


double
fddi_timer_value (timer_ptr)
	FddiT_Timer*		timer_ptr;
	{
	/* if the timer is enabled, update the accumulated time */
	if (timer_ptr->enabled)
		{
		timer_ptr->accum = op_sim_time () - timer_ptr->start_time;
		}

	return (timer_ptr->accum);
	}


fddi_timer_set_value (timer_ptr, value)
	FddiT_Timer*		timer_ptr;
	double			value;
	{
	timer_ptr->accum = value;
	}


fddi_timer_copy (from_timer_ptr, to_timer_ptr)
	FddiT_Timer*		from_timer_ptr;
	FddiT_Timer*		to_timer_ptr;
	{
	op_prg_mem_copy (from_timer_ptr, to_timer_ptr,
		sizeof (FddiT_Timer));
	}


fddi_timer_set (timer_ptr, duration)
	FddiT_Timer*		timer_ptr;
	{
	/* clear out accumulated time */
	timer_ptr->accum = 0.0;

	/* assign the timer duration */
	timer_ptr->target_accum = duration;

	/* assign the current time */
	timer_ptr->start_time = op_sim_time ();

	/* enable the timer */
	timer_ptr->enabled = 1;
	}


FddiT_Timer*
fddi_timer_create ()
	{
	FddiT_Timer*		timer_ptr;

	/* allocate memory for a timer structure */
```

```
	timer_ptr = (FddiT_Timer*) op_prg_mem_alloc (sizeof (FddiT_Timer));

	/* initialize the timer in the disabled mode */
	fddi_timer_init (timer_ptr);

	/* return the timer's address */
	return (timer_ptr);
	}


fddi_timer_init (timer_ptr)
	FddiT_Timer*		timer_ptr;
	{
	/* the timer is initially disabled */
	timer_ptr->enabled = 0;

	/* the accumulated time is zero */
	timer_ptr->accum = 0.0;

	/* the target accumulated time is infinite */
	timer_ptr->target_accum = VOS_DOUBLE_INFINITY;

	/* the start time is now */
	timer_ptr->start_time = op_sim_time ();
	}


fddi_station_register (address)
	int				address;
	{
	/* Count stations and find the minimum address on the ring. */
	FIN (fddi_station_register (address))

	/* Keep track of total number of stations on the ring */
	Fddi_Num_Stations++;

	/* If this is the first station registration, or if this		*/
	/* station address is the lowest seen yet, update the			*/
	/* minimum station ID.											*/
	if (Fddi_Num_Stations == 1 || address < Fddi_Min_Station)
		Fddi_Min_Station = address;

	FOUT
	}


fddi_station_address_map (address, objid)
	Objid			objid;
	int				address;
	{
	/* Fill an entry in the table which maps station		*/
	/* addresses to OPNET object ID's						*/
	FIN (fddi_station_address_map (address, objid))

	Fddi_Address_Table [FDDI_ADDR_NORM (address)] = objid;
	FOUT
	}
```

```
fddi_tk_register ()
	{
	/* Register the station's intent to use the token.		*/
	/* This should be done whenever an unregistered		*/
	/* station obtains new data to transmit.			*/
	FIN (fddi_tk_register ())

	/* increase the number of registered stations */
	Fddi_Num_Registered++;

	/* if the token is currently blocked, unblock it */
	if (Fddi_Tk_Blocked && Fddi_Tk_Accelerate)
		{
		fddi_tk_unblock ();
		}

	FOUT
	}

fddi_tk_deregister ()
	{
	/* Cancel the station's intent to use the token.		*/
	/* This should be done whenever a registered		*/
	/* station exhausts its transmittable data.		*/
	FIN (fddi_tk_deregsister ())

	/* decrease the number of registered stations */
	Fddi_Num_Registered--;

	FOUT
	}


fddi_tk_indicate_no_data (token, address, delay)
	Packet*		token;
	int			address;
	double		delay;
	{
	FIN (fddi_tk_indicate_no_data (token, address, delay))

	/* The calling station is indicating that it has captured			*/
	/* the token, but has no data to send.  If no other stations		*/
	/* have data to send either, the token may be blocked to gain		*/
	/* simulation efficiency.							*/

	if (Fddi_Num_Registered == 0 && Fddi_Tk_Accelerate)
		{
		fddi_tk_block (token, address);
		}
	else{
		/* If the token cannot be blocked, send it into the ring.		*/
		op_pk_send_delayed (token, FDDI_PHY_STRM_OUT,
			delay + Fddi_Prop_Delay);
		}

	FOUT
	}
```

```
fddi_tk_block (token, address)
	Packet*		token;
	int		address;
	{
	int		i;

	FIN (fddi_tk_block (token, address))

	/* Record the address of the blocking station and blocking time. */
	Fddi_Tk_Block_Base_Time = op_sim_time ();
	Fddi_Tk_Block_Base_Station = address;

	if (tk_trace_on == OPC_TRUE)
		{
		sprintf (str0, "Blocking Token: station (%d), time (%.9f)",
				Fddi_Tk_Block_Base_Station, Fddi_Tk_Block_Base_Time);
		op_prg_odb_print_major (str0, OPC_NIL);
		}

	/* Indicate that the token is blocked */
	Fddi_Tk_Blocked = 1;

	/* discard the token packet; another one will be */
	/* created when the token is unblocked. */
	op_pk_destroy (token);

	/* Cancel TRT timers at all MAC interfaces; otherwise these */
	/* timers may continue to expire during the idle period,		*/
	/* generating unnecessary events.					*/
	if (tk_trace_on == OPC_TRUE)
		{
		sprintf (str0, "Canceling timers for (%d) stations", Fddi_Num_Stations);
		op_prg_odb_print_major (str0, OPC_NIL);
		}
/*	Fddi_Num_Stations = 6;*/
/*printf("No of times thru loop: %d\n",Fddi_Num_Stations);  getchar();*/
	for (i = 0; i < Fddi_Num_Stations; i++)
		{
		/* Retain the time at which the TRT would have expired;		*/
		/* this is used for calculations when the token is		*/
		/* reinjected into the ring.					*/
		Fddi_Trt_Exp_Time [i] = op_ev_time (Fddi_Trt_Handle [i]);

		/* Cancel the TRT expiration event. */
		op_ev_cancel (Fddi_Trt_Handle [i]);
		}

	FOUT
	}


fddi_tk_unblock ()
	{
	double		elapsed_time, first_tk_rx, last_tk_rx;
	double		tk_lap_time, next_time, current_time;
	double		dbl_num_hops, num_tk_rx, floor (), ceil ();
	int		i, num_hops, next_station;
```

...
...

```
FIN (fddi_tk_unblock ())

/* reset the blocking indicator */
Fddi_Tk_Blocked = 0;

/* Get the current time, used for many calculations below */
current_time = op_sim_time ();

if (tk_trace_on == OPC_TRUE)
	{
	sprintf (str0, "Unblocking token for ring (%d)", ring_id);
	op_prg_odb_print_major (str0, OPC_NIL);
	}

/* For all stations on the ring, adjust TRT timer and Late_Ct flag. */
for (i = 0; i < Fddi_Num_Stations; i++)
	{
	if (tk_trace_on == OPC_TRUE)
		{
		sprintf (str0, "adjusting state of station (%d)", FDDI_ADDR_DENORM (i));
		op_prg_odb_print_minor ("", str0, OPC_NIL);
		}
	/* Calculate number of hops separating station i from block base station.	*/
	/* In special case where i is the base station, the token must run a full	*/
	/* lap before returning.	*/
	if (i != FDDI_ADDR_NORM (Fddi_Tk_Block_Base_Station))
		{
		num_hops = (i - FDDI_ADDR_NORM (Fddi_Tk_Block_Base_Station)) % Fddi_Num_Stations;
		if (num_hops < 0)
			num_hops = Fddi_Num_Stations + num_hops;
		}
	else num_hops = Fddi_Num_Stations;

	/* Calculate first time at which token would have been received by station i.	*/
	/* Note that initial release of token from base station takes a different	*/
	/* amount of time than repeating of token by other stations. Thus, the first	*/
	/* hop is assumed, and the base time is augmented by the time required to	*/
	/* complete it.	*/
	first_tk_rx = Fddi_Tk_Block_Base_Time + FDDIC_TOKEN_TX_TIME + Fddi_Prop_Delay +
		(num_hops - 1) * Fddi_Tk_Hop_Delay;

	if (tk_trace_on == OPC_TRUE)
		{
		sprintf (str0, "station is (%d) hops from base", num_hops);
		sprintf (str1, "first receipt of token would be at (%.9f)", first_tk_rx);
		op_prg_odb_print_minor (str0, str1, OPC_NIL);
		}

	/* Case 1: the token would not yet have been received by station i. */
	if (first_tk_rx > current_time)
		{
		/* Case 1a: the TRT at station i would not yet have expired. */
		if (Fddi_Trt_Exp_Time [i] > current_time)
			{
			/* Late_Ct remains at its original value; only the TRT needs	*/
			/* to be started again, with the same expiration time.	*/
			TRT_SET (FDDI_ADDR_DENORM (i), Fddi_Trt_Exp_Time [i])

			if (tk_trace_on == OPC_TRUE)
```

```
                    {
                    sprintf (str0, "Restoring TRT to previous exp. time (%.9f)", Fddi_Trt_Exp_Time [i]);
                    op_prg_odb_print_minor ("Token would not be received and TRT not expired", str0, OPC_N
                    }
                }
            /* Case 1b: the TRT at station i would have expired. */
            else
                {
                /* Late_Ct would have been set; also the timer would have been rescheduled          */
                /* for an entire TTRT at the time of expiration.                                    */
                Fddi_Late_Ct [i] = 1;
                TRT_SET (FDDI_ADDR_DENORM (i), (Fddi_T_Opr + Fddi_Trt_Exp_Time [i]))

                if (tk_trace_on == OPC_TRUE)
                    {
                    sprintf (str0, "Restoring TRT to proper exp. time (%.9f)", Fddi_T_Opr + Fddi_Trt_Exp_Time [
                    op_prg_odb_print_minor ("Token would not be received and TRT would have expired", str
                    }
                }
            }

        /* Case 2: the token would have been received (perhaps more than once). */
        else
            {
            /* Calculate the number of times the token would have been received */
            /* not including the first receipt.                                  */
            tk_lap_time = Fddi_Tk_Hop_Delay * Fddi_Num_Stations;
            num_tk_rx = floor ((current_time - first_tk_rx) / tk_lap_time);

            /* Calculate the latest time at which the token would have been received. */
            last_tk_rx = first_tk_rx + (num_tk_rx * tk_lap_time);

            /* Clear Late_Ct and schedule timer to expire at last receipt of token        */
            /* plus one full TTRT.                                                        */
            Fddi_Late_Ct [i] = 0;
            TRT_SET (FDDI_ADDR_DENORM (i), (last_tk_rx + Fddi_T_Opr))

            if (tk_trace_on == OPC_TRUE)
                {
                sprintf (str0, "token received (%g) times, last receipt at (%.9f)",
                    num_tk_rx + 1.0, last_tk_rx);
                sprintf (str1, "Restoring TRT to proper exp. time (%.9f)",
                    last_tk_rx + Fddi_T_Opr);
                op_prg_odb_print_minor ("Token would have been received; Late_Ct is cleared",
                    str1, str0, OPC_NIL);
                }
            }
        }

    /* compute the time since the token was blocked */
    elapsed_time = current_time - Fddi_Tk_Block_Base_Time;

    /* compute the number of hops completed on the ring. For the first hop      */
    /* the token is transmitted directly, not repeated. For all remaining      */
    /* hops, the delay is the station latency plus the propagation delay.      */
    /* Thus, the first hop is assumed, and the remaining time for additional   */
    /* hops is computed beginning at the time where the token enters the       */
    /* base station’s downstream neighbor.                                     */
    dbl_num_hops = 1.0 +
```

```
            (elapsed_time - FDDIC_TOKEN_TX_TIME - Fddi_Prop_Delay) / Fddi_Tk_Hop_Delay;

    /* If the token was unblocked in less time than it would have taken to      */
    /* be fully transmitted by the base station, dbl_num_hops will be           */
    /* negative. However, 1 full hop would still be required before the         */
    /* token could be used, since the station had already committed to          */
    /* issuing the token. Thus, the actual of number of hops should never       */
    /* be less than 1. If it is, round it to 1.                                 */
    if (dbl_num_hops < 1.0)
        dbl_num_hops = 1.0;
    else
        {
        /* In all other cases, round the number of hops up to the nearest       */
        /* integer value. If already an integer, then leave as is.              */
        dbl_num_hops = ceil (dbl_num_hops);
        }

    /* Obtain an integer equivalent of dbl_num_hops. */
    num_hops = dbl_num_hops;

    /* Based on the number of hops and the base station, compute the       */
    /* next station where the token will appear.                           */
    next_station =
        FDDI_ADDR_DENORM ((num_hops + FDDI_ADDR_NORM (Fddi_Tk_Block_Base_Station)
            ) % Fddi_Num_Stations);

    /* Compute the time at which the token will appear there.         */
    /* Again, assume the first hop occurred, and measure time         */
    /* from there forward.                                            */
    next_time = Fddi_Tk_Block_Base_Time + (FDDIC_TOKEN_TX_TIME + Fddi_Prop_Delay) +
            (dbl_num_hops - 1.0) * Fddi_Tk_Hop_Delay;

    if (tk_trace_on == OPC_TRUE)
        {
        sprintf (str0, "Re-introducing token at station (%d), at time (%.9f)",
             next_station, next_time);
        op_prg_odb_print_minor (str0, OPC_NIL);
        }

    /* reinject the token at that station */
    fddi_tk_inject (next_station, next_time);

    FOUT
    }

fddi_tk_inject (address, arv_time)
    int             address;
    double          arv_time;
    {
    /* Re-insert the token into the ring after an idle period. */
    FIN (fddi_tk_inject (address, arv_time))

    /* The token is recreated and reinserted onto the ring */
    /* at the specified station which is not necesssarily the */
    /* station now requesting the token. */
    /* The station which will reinsert the token is */
    /* asked to do so by means of a remote interrupt. */
    op_intrpt_schedule_remote (arv_time, FDDIC_TK_INJECT,
            Fddi_Address_Table [FDDI_ADDR_NORM (address)]);
```

```
	FOUT
	}


fddi_load_frame_attrs (dest_addr_ptr, svc_class_ptr, pri_level_ptr)
	int			*dest_addr_ptr, *svc_class_ptr, *pri_level_ptr;
	{
	Packet			*pkptr;

	FIN (fddi_load_frame_attrs (dest_addr_ptr, svc_class_ptr, pri_level_ptr))

	/* obtain next packet in queue */
	pkptr = op_subq_pk_access (0, OPC_QPOS_HEAD);

	/* extract the fields of interest */
	op_pk_nfd_get (pkptr, "dest_addr", dest_addr_ptr);
	op_pk_nfd_get (pkptr, "svc_class", svc_class_ptr);

	/* only read priority level if frame is asynchronous */
	if (*svc_class_ptr == FDDI_SVC_ASYNC)
		op_pk_nfd_get (pkptr, "pri", pri_level_ptr);

	FOUT
	}
```

Diagnostic Block

```
/* Print out values of timers, and late token counter.			*/
/* Also print out data about restricted mode.				*/
/* Also indicate if this station is operating in promiscuous mode.	*/
/* (This code may be executed by the simulation debugger		*/
/* by invoking the command 'prodiag').					*/

sprintf (str0, "Timers (count upwards):  TRT (%.9g), THT (%.9g)",
	fddi_timer_value (TRT), fddi_timer_value (THT));
sprintf (str1, "Late_ct (%d)", Late_Ct);
op_prg_odb_print_major (str0, str1, OPC_NIL);

if (restricted)
	sprintf (str0, "token is in restricted dialog with (%d)\n", res_peer);
else sprintf (str0, "token is unrestricted\n");
op_prg_odb_print_major (str0, OPC_NIL);

if (promiscuous == OPC_TRUE)
	{
	sprintf (str0, "station is operating in promiscuous mode\n");
	op_prg_odb_print_major (str0, OPC_NIL);
	}

if (encap_decap != OPC_TRUE)
```

```
    {
    sprintf (str0, "station is not operating in encapsulation mode\n");
    op_prg_odb_print_major (str0, OPC_NIL);
    }
```

forced state **INIT**			
attribute	*value*	*type*	*default value*
name	INIT	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **INIT**

```
/* Now that all stations have been counted and minimum address          */
/* determined (done in CLAIM state), construct a table mapping          */
/* addresses to MAC module object ID's.                                 */
fddi_station_address_map (my_address, my_objid);


/* Obtain the station latency for tokens and frames. */
/* Default value is set at 100 nanoseconds. */
Fddi_St_Latency = 100.0e-09;
op_ima_sim_attr_get (OPC_IMA_DOUBLE, "station_latency", &Fddi_St_Latency);

/* Obtain the propagation delay separating stations. */
/* This value is given in seconds with default value 3.3 microseconds. */
Fddi_Prop_Delay = 3.3e-06;
op_ima_sim_attr_get (OPC_IMA_DOUBLE, "prop_delay", &Fddi_Prop_Delay);

/* Derive the Delay for a 'hop' of a freely circulating packet. */
Fddi_Tk_Hop_Delay = Fddi_Prop_Delay + Fddi_St_Latency;


/* The T_Pri [] state variable array supports priority */
/* assignments on a station by station basis by */
/* establishing a correspondence between integer priority */
/* levels assigned to frames and the maximum values of the */
/* token holding timer (THT) which would allow packets to be */
/* sent.  Eight levels are supported here, but this can easily */
/* be changed by redimensioning the priority array. */
/* By default all levels are identical here, allowing */
/* any frame to make use of the token, so that in fact */
/* priority levels are not used in the default case. */
for (i = 0; i < 8; i++)
    {
    T_Pri [i] = Fddi_T_Opr;
    }
```

...
...

```
/* Create the token holding timer (THT) used to restrict the */
/* asynchronous bandwidth consumption of the station */
THT = fddi_timer_create ();

/* Create the token rotation timer (TRT) used to measure the */
/* rotations of the token, detect late tokens and initialize */
/* the THT timer before asynchronous tranmsmissions. */
TRT = fddi_timer_create ();

/* Set the TRT timer to expire in one TTRT */
TRT_SET (my_address, op_sim_time () + Fddi_T_Opr);

/* Initialize the Late_Ct variable which keeps track. */
/* of the number of TRT expirations. */
Late_Ct = 0;

/* initially the ring operates in nonrestricted mode */
restricted = 0;


/* Create an Interface Control Information structure */
/* to use when delivering received frames to the LLC. */
to_llc_ici_ptr = op_ici_create ("fddi_mac_ind");

/* The 'tk_registered' variable indicates if the station */
/* has registered its intent to use the token. */
tk_registered = 0;


/* Determine if the model is to make use of the token */
/* 'acceleration' mechanism.  If not, every passing of the */
/* token will be explicitly modeled, leading to a large  */
/* number of events being scheduled when the ring is idle */
/* (i.e, no stations have data to send). */
op_ima_sim_attr_get (OPC_IMA_INTEGER, "accelerate_token",
	&Fddi_Tk_Accelerate);


/* Determine if the mac layer will operate in "promiscuous mode". */
/* If enabled, this process receives every packet placed on */
/* the ring and sends a copy of it to the higher layer. */
op_ima_obj_attr_get (my_objid, "promiscuous", &promiscuous);

/* Determine if the mac layer will encapsulate and decapsulate */
/* packets.  If disabled, encapsulation and decapsulation of packets */
/* will not occur.  The process will assume that packets received  */
/* from the higher layer are well-formed fddi packets.  This process */
/* will send the entire fddi packet to the higher layer. */
op_ima_obj_attr_get (my_objid, "encap_decap", &encap_decap);


/* Obtain the synchronous bandwidth assigned */
/* to this station.  It is expressed as a */
/* percentage of TTRT, and then converted to seconds */
op_ima_obj_attr_get (my_objid, "sync bandwidth", &sync_pc);
sync_bandwidth = sync_pc * Fddi_T_Opr;
```

...
...

```
/* Only one station in the ring is selected to */
/* introduce the first token.  That station is */
/* the station on the ring which has a station address */
/* that is offset from the lowest station on the ring */
/* by the value of the 'spawn_station' simulation */
/* attribute. Test if this station is it. */
/* If so, set the 'spawn_token' flag. */
op_ima_sim_attr_get (OPC_IMA_INTEGER, "spawn station", &spawn_station);
spawn_token = (spawn_station == FDDI_ADDR_NORM(my_address));


/* If the station is to spawn the token, create */
/* the packet which represents the token. */
if (spawn_token)
	{
	tk_pkptr = op_pk_create_fmt ("fddi_mac_tk");

	/* assign its frame control field */
	op_pk_nfd_set (tk_pkptr, "fc", FDDI_FC_TOKEN);

	/* the first token issued is non-restricted */
	op_pk_nfd_set (tk_pkptr, "class", FDDI_TK_NONRESTRICTED);

	/* The transition will be made into the ISSU_TK */
	/* state where the tk_usable variable is used. */
	/* In case any data has been generated, prset */
	/* this variable to one. */
	tk_usable = 1;
	}


/* When sending packets the variable accum_bandwidth is */
/* used as a scheduling base.  Init this value to zero. */
/* This statement is required in case this is the spawning */
/* station, and the next state entered is ISSUE_TK */
accum_bandwidth = 0.0;
```

transition **INIT -> IDLE**

attribute	*value*	*type*	*default value*
name		string	tr
condition	default	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

...
...

transition **INIT -> ISSUE_TK**			
attribute	*value*	*type*	*default value*
name		string	tr
condition	spawn_token	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

unforced state **IDLE**			
attribute	*value*	*type*	*default value*
name	IDLE	string	st
enter execs	(empty)	textlist	(empty)
exit execs	(See below.)	textlist	(See below.)
status	unforced	toggle	unforced

exit execs **IDLE**

```
/* Determine if a trace is activated for the FDDI model */
tk_trace_on = op_prg_odb_ltrace_active ("fddi_tk");

/* Trap packets arriving from physical layer so that their          */
/* FC field can be extracted before evaluating conditions           */
if (op_intrpt_type () == OPC_INTRPT_STRM && op_intrpt_strm () != FDDI_LLC_STRM_IN)
	{
	/* Acquire the arriving packet. */
	pkptr = op_pk_get (FDDI_PHY_STRM_IN);

	/* Determine the type of packet by extracting */
	/* the frame control field. */
	op_pk_nfd_get (pkptr, "fc", &frame_control);

	/* Physical layer arrival flag is set. */
	phy_arrival = 1;
	}
else{
	/* The interrupt is not due to a physical layer arrival. */
	phy_arrival = 0;

	/* If the interrupt is a remote interrupt with specified code,  it signifies          */
	/* the reinsertion of the token into the ring after an idle period. This only         */
	/* occurs if the token acceleration mechanism is active.                              */
	if (op_intrpt_type () == OPC_INTRPT_REMOTE && op_intrpt_code () == FDDIC_TK_INJECT)
		{
		/* create a new token */
		tk_pkptr = op_pk_create_fmt ("fddi_mac_tk");

		/* assign its frame control field */
		op_pk_nfd_set (tk_pkptr, "fc", FDDI_FC_TOKEN);

		/* the token is non-restricted */
		op_pk_nfd_set (tk_pkptr, "class", FDDI_TK_NONRESTRICTED);

		/* insert it into the ring */
		op_pk_send (tk_pkptr, FDDI_PHY_STRM_OUT);
		}
  if (op_intrpt_type () == OPC_INTRPT_STAT &&
```

```
    op_intrpt_stat () == UPDATE_INSTAT )
  {
   grp = (int)op_stat_local_read(UPDATE_INSTAT);
   if(grp > 99) op_sim_end("Problem in ETH_MAC IGMP","","","");
   if(MC[grp] == 1) MC[grp] = -1;
      else MC[grp] = 1;
 }

   }
```

transition **IDLE -> RCV_TK**			
attribute	*value*	*type*	*default value*
name		string	tr
condition	TK_RECEIVED	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **IDLE -> FR_RCV**			
attribute	*value*	*type*	*default value*
name		string	tr
condition	RC_FRAME	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **IDLE -> TRT_EXP**			
attribute	*value*	*type*	*default value*
name		string	tr
condition	TRT_EXPIRE	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **IDLE -> ENCAP**			
attribute	*value*	*type*	*default value*
name		string	tr
condition	FRAME_ARRIVAL	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **IDLE -> IDLE**			
attribute	*value*	*type*	*default value*
name		string	tr
condition	default	string	

executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **ISSUE_TK**

attribute	*value*	*type*	*default value*
name	ISSUE_TK	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **ISSUE_TK**

```
/* If the token is sent without having been used, and the station */
/* has no data to send, then indicate this fact to the */
/* token acceleration mechanism which may have an */
/* oppurtunity to block the token. */
if (!tk_usable && op_q_stat (OPC_QSTAT_PKSIZE) == 0.0)
	{
	/* Note that if the token cannot be blocked, */
	/* this procedure will forward the token physically. */
	fddi_tk_indicate_no_data (tk_pkptr, my_address, accum_bandwidth);
	}
else{
	if (tk_trace_on == OPC_TRUE)
		{
		sprintf (str0, "Issuing token. accum_bw (%.9f), prop_del (%.9f)",
			accum_bandwidth, Fddi_Prop_Delay);
		op_prg_odb_print_major (str0, OPC_NIL);
		}

	/* Send out the token packet using the accumulated */
	/* consumed bandwidth as a scheduling base. */
	/* In the case of the initial spawning of the token */
	/* this will be zero; otherwise this variable will */
	/* reflect the bandwidth consumed since the last capture */
	/* of the usable token. Propagation delay is also accounted for. */
	op_pk_send_delayed (tk_pkptr, FDDI_PHY_STRM_OUT,
			accum_bandwidth + Fddi_Prop_Delay);
	}
```

transition **ISSUE_TK -> IDLE**

attribute	*value*	*type*	*default value*
name		string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

...
...

forced state **RCV_TK**

attribute	*value*	*type*	*default value*
name	RCV_TK	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **RCV_TK**

```
/* The arriving packet, when received in the IDLE state */
/* is placed in the variable 'pkptr'.  Since it is now  */
/* known that it is a token, it can be placed in 'tk_pkptr. */
tk_pkptr = pkptr;

/* Load the token's class into the temporary variable 'tk_class.' */
op_pk_nfd_get (pkptr, "class", &tk_class);

/* If the token is restricted, determine for which station. */
if (tk_class == FDDI_TK_RESTRICTED)
	{
	/* Place the station address in the variable 'res_station' */
	/* which may factor in to the determination of token usability. */
	op_pk_nfd_get (tk_pkptr, "res_station", &res_station);
	}


/* Determine if the token is usable:                    */
/* assume by default that it is not.                    */
/* Subsequent conditions may override this. */
tk_usable = 0;


/* The token can only be usable if there are frames enqueued */
if (op_q_stat (OPC_QSTAT_PKSIZE) > 0.0)
	{
	/* examine the attributes of the packet at the */
	/* head of the queue. */
	fddi_load_frame_attrs (&dest_addr, &svc_class, &pri_level);

	/* If synchronous data is queued, the token is */
	/* necessarily usable, regardless of timing conditions. */
	if (svc_class == FDDI_SVC_SYNC)
		{
		tk_usable = 1;
		}
	else{
		/* Otherwise, if asynchronous data is queued, it must */
		/* meet several criteria for the token to be usable.  */

		/* The token is only usable only if it is early. */
		if (Late_Ct == 0)
			{
			/* The token's class must be nonrestricted, unless */
			/* this station is involved in the restricted transfer. */
			if (tk_class == FDDI_TK_NONRESTRICTED ||
				res_station == my_address ||
				restricted)
				{
```

```
                    /* Test the frame's priority assignment against the current TRT */
                    /* This test uses the priority indirection table T_Pri */
                    /* so that only packets whose T_Pri [pri_level] exceeds */
                    /* the TRT can be transmitted.  In other words, by */
                    /* assigning lower values to T_Pri for a given priority */
                    /* level, packets of that level will be further restricted */
                    /* from using the ring bandwidth. */
                    if (T_Pri [pri_level] >= fddi_timer_value (TRT))
                        {
                        tk_usable = 1;
                        }
                    }
                }
            }
        }



/* If the token is usable, timers must be readjusted. */
if (tk_usable)
    {
    /* The timer adjustment depends on whether the token is early or late. */
    if (Late_Ct == 0)
        {
        /* Transfer the contents of TRT into THT. */
        fddi_timer_copy (TRT, THT);

        /* Disable the THT timer. */
        fddi_timer_disable (THT);

        /* Reset TRT to time the next rotation. */
        op_ev_cancel (TRT_handle);
        TRT_SET (my_address, op_sim_time () + Fddi_T_Opr);
        }
    else{
        /* If the token is late, set the THT to its expired */
        /* value, and disable it.  This will prevent any */
        /* asynchronous transmissions from occuring. */
        fddi_timer_set_value (THT, Fddi_T_Opr);
        fddi_timer_disable (THT);

        /* clear the Late token counter (note that TRT is not modified,                */
        /* so that less than a full TTRT remains before TRT expires again.)            */
        Late_Ct = 0;
        }
    }


/* If the token is not usable, different adjustments are made. */
else{
    /* Again, the adjustments depend on the lateness of the token */
    if (Late_Ct == 0)
        {
        /* If the token is not late, the TRT is reset to time the next rotation. */
        op_ev_cancel (TRT_handle);
        TRT_SET (my_address, op_sim_time () + Fddi_T_Opr);
        }
    else{
        /* clear the Late token counter (note that TRT is not modified, */
```

...
...

```
        /* so that less than a full TTRT remains before TRT expires again. */
        Late_Ct = 0;
        }

    /* also, account for the time needed by the token                    */
    /* to traverse the station, since it is about to be sent.            */
    /* Note: station latency is not inclusive of token                   */
    /* transmission time, but only of the time required to               */
    /* process and repeat the token's symbols.                           */
    accum_bandwidth = Fddi_St_Latency;
    }
```

transition **RCV_TK -> ISSUE_TK**

attribute	*value*	*type*	*default value*
name		string	tr
condition	default	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **RCV_TK -> TX_DATA**

attribute	*value*	*type*	*default value*
name		string	tr
condition	tk_usable	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **FR_RCV**

attribute	*value*	*type*	*default value*
name	FR_RCV	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **FR_RCV**

```
/* A frame has been received from the physical layer. Note that */
/* at this time, only the leading edge of the frame has arrived. */

/* Extract the frame's source address (this will be used to */
/* determine whether or not to strip the frame from the ring). */
op_pk_nfd_get (pkptr, "src_addr", &src_addr);
```

transition **FR_RCV -> FR_STRIP**

attribute	*value*	*type*	*default value*
name		string	tr

condition	STRIP	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **FR_RCV -> FR_REPEAT**

attribute	*value*	*type*	*default value*
name		string	tr
condition	default	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **FR_STRIP**

attribute	*value*	*type*	*default value*
name	FR_STRIP	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **FR_STRIP**

```
/* Destroy the frame which has now circulated the entire ring. */
op_pk_destroy (pkptr);
```

transition **FR_STRIP -> IDLE**

attribute	*value*	*type*	*default value*
name		string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **FR_REPEAT**

attribute	*value*	*type*	*default value*
name	FR_REPEAT	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **FR_REPEAT**

```
/* Extract the destination address of the frame. */
op_pk_nfd_get (pkptr, "dest_addr", &dest_addr);

/* record total size of the frame (including data) */
pk_len = op_pk_total_size_get (pkptr);

/* If the incoming frame is destined for this station, */
/* or if this station is operating in promiscuous mode, */
```

```
/* then this packet (or its decapsulated data contents) */
/* will be forwarded to the higher layer. */
if (FORWARD)
	{
	/* If the original frame is to be repeated on the ring, */
	/* make a copy of the original. Otherwise the original will be used. */
	if (REPEAT)
		tmp_pkptr = op_pk_copy (pkptr);
	else
		tmp_pkptr = pkptr;

	/* If decapsulation is to be performed, retrieve the data */
	/* contents of the frame. */
	if (DECAP)
		{
		/* decapsulate the data contents of the frame */
		op_pk_nfd_get (tmp_pkptr, "info", &data_pkptr);

		/* The source and destination address are placed in the */
		/* LLC's ICI before delivering the frame's contents. */
		op_ici_attr_set (to_llc_ici_ptr, "src_addr", src_addr);
		op_ici_attr_set (to_llc_ici_ptr, "dest_addr", dest_addr);
		op_ici_install (to_llc_ici_ptr);

		/* Note that the standard specifies that the original */
		/* frame should be passed along until the originating station */
		/* receives it, at which point it is stripped from the ring. */
		/* However, in the simulation model, there is no interest */
		/* in letting the frame continue past its destination unless */
		/* group addresses are used, so that the same frame could be */
		/* destined for several stations.  Here the frame is stripped */
		/* for efficiency as it reaches the destination; if the model */
		/* is modified to include group addresses, this should be changed */
		/* so that the frame is copied and the original repeated. */
		/* Logic is already present for stripping the frame at the origin. */
		op_pk_destroy (tmp_pkptr);
		}
	/* If decapsulation is not to be performed, the frame is sent to the */
	/* higher layer "as-is". */
	else
		data_pkptr = tmp_pkptr;

	/* Because, as noted in the FR_RCV state, only the */
	/* frame's leading edge has arrived at this time, the */
	/* complete frame can only be delivered to the higher */
	/* layer after the frame's transmission delay has elapsed. */
	/* (since decapsulation of the frame data contents has occured, */
	/* the original MAC frame length is used to calculate delay) */
	tx_time = (double) pk_len / FDDI_TX_RATE;
	if(MC[dest_addr-1000] == 1 || is_gwy == 1 || dest_addr == 1500 || dest_addr == my_address)op_pk_send_delayed (data_pkptr,
	  else op_pk_destroy(data_pkptr);
	}

if (REPEAT)
	{
	/* Repeat the original frame on the ring and account for */
	/* the latency through the station and the propagation delay */
	/* for a single hop. */
	/* (Only the originating station can strip the frame). */
```

...
...

```
        op_pk_send_delayed (pkptr, FDDI_PHY_STRM_OUT,
            Fddi_St_Latency + Fddi_Prop_Delay);
        }

/* If not repeating, the original packet has already been sent */
/* to the higher layer or destroyed. */
```

transition **FR_REPEAT -> IDLE**			
attribute	*value*	*type*	*default value*
name		string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **TRT_EXP**			
attribute	*value*	*type*	*default value*
name	TRT_EXP	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **TRT_EXP**

```
/* The timer is reset and allowed to continue running. */
TRT_SET (my_address, op_sim_time () + Fddi_T_Opr);

/* The late token counter is incremented.  This will          */
/* prevent this station from making any asynchronous          */
/* transmissions when it next captures the token.             */
Late_Ct++;
```

transition **TRT_EXP -> IDLE**			
attribute	*value*	*type*	*default value*
name		string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **ENCAP**			
attribute	*value*	*type*	*default value*
name	ENCAP	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

...
...

enter execs **ENCAP**

```
/* A frame has arrived from a higher layer */
/* If in encap_decap mode, then encapsulation */
/* is to be performed. Place the frame in 'pdu_ptr'. */
if (encap_decap == OPC_TRUE)
	{
	pdu_ptr = op_pk_get (op_intrpt_strm ());

	/* Also get the interface control information */
	/* associated with the new frame. */
	ici_ptr = op_intrpt_ici ();
	if (ici_ptr == OPC_NIL)
		{
		sprintf (error_string, "Simulation aborted; error in object (%d)",
			op_id_self ());
		op_sim_end (error_string, "fddi_mac: required ICI not received", " ", " ");
		}

	/* Extract the requested service class */
	/* (e.g, synchronous or asynchronous). */
	if (op_ici_attr_exists (ici_ptr, "svc_class"))
		op_ici_attr_get (ici_ptr, "svc_class", &svc_class);
	else svc_class = FDDI_SVC_ASYNC;

	/* Extract the destination address. */
	op_ici_attr_get (ici_ptr, "dest_addr", &dest_addr);

	/* If the frame is asynchronous, the priority and */
	/* requested token class parameter may be specified. */
	if (svc_class == FDDI_SVC_ASYNC)
		{
		/* Extract the requested priority level. */
		if (op_ici_attr_exists (ici_ptr, "pri"))
			op_ici_attr_get (ici_ptr, "pri", &req_pri);
		else req_pri = 0;

		/* Extract the token class (restricted or non-restricted). */
		if (op_ici_attr_exists (ici_ptr, "tk_class"))
			op_ici_attr_get (ici_ptr, "tk_class", &req_tk_class);
		else req_tk_class = FDDI_TK_NONRESTRICTED;
		}


	/* Compose a mac frame from all these elements. */
	mac_frame_ptr = op_pk_create_fmt ("fddi_mac_fr");
	op_pk_nfd_set (mac_frame_ptr, "svc_class", svc_class);
	op_pk_nfd_set (mac_frame_ptr, "dest_addr", dest_addr);
	op_pk_nfd_set (mac_frame_ptr, "src_addr", my_address);
	op_pk_nfd_set (mac_frame_ptr, "info", pdu_ptr);

	if (svc_class == FDDI_SVC_ASYNC)
		{
		op_pk_nfd_set (mac_frame_ptr, "tk_class", req_tk_class);
		op_pk_nfd_set (mac_frame_ptr, "pri", req_pri);
		}


	/* Assign the frame control field, which in the model */
```

...
...

```
        /* is used to distinguish between tokens and ordinary */
        /* frames on the ring. */
        op_pk_nfd_set (mac_frame_ptr, "fc", FDDI_FC_FRAME);
        }

else
        {
        /* No encapsulation is to be performed. */
        /* get the whole frame from the stream. */
        mac_frame_ptr = op_pk_get (op_intrpt_strm ());

        /* However, the frame must be "stamped" with                  */
        /* this station's source address.  Otherwise,                 */
        /* there is no guarantee that the frame will                  */
        /* be eventually stripped from the ring.                      */
        op_pk_nfd_set (mac_frame_ptr, "src_addr", my_address);
        }

/* Enqueue the frame at the tail of the queue. */
op_subq_pk_insert (0, mac_frame_ptr, OPC_QPOS_TAIL);


/* if this station has not yet registered its intent to */
/* use the token, it may do so now since it has data to send */
if (!tk_registered)
        {
        fddi_tk_register ();
        tk_registered = 1;
        }
```

transition **ENCAP -> IDLE**			
attribute	*value*	*type*	*default value*
name		string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **TX_DATA**			
attribute	*value*	*type*	*default value*
name	TX_DATA	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

...
...

enter execs **TX_DATA**

```
/* In this state, frames are transmitted until the */
/* token is no longer usable.  Frames are taken from */
/* the single input queue in FIFO order. */

/* Reset the accumulator used to keep track of bandwidth */
/* consumed by the transmissions.  Because all the transmissions */
/* are scheduled to happen at the appropriate times, but */
/* these schedulings occur instantly, this accumulator serves */
/* as the scheduling base for the transmissions. */
/* In other words, each successively transmitted frame */
/* is delayed relative to the previous one by the time which */
/* the latter took to send.  At the end of transmission (e.g, */
/* when the token is no longer usable), this accumulator */
/* serves to delay the forwarding of the token. */
accum_bandwidth = 0.0;


/* Note that, because all tranmsmissions are */
/* scheduled, the value of the THT timer will not progress */
/* between schedulings (these all happen in zero time), and so  */
/* the variable 'tht_value' is used to emulate the timer's progress. */
tht_value = fddi_timer_value (THT);


/* Reset an accumulator which reflects the consumed */
/* synchronous bandwidth. */
accum_sync = 0.0;

/* Reset counters for transmitted frames and bits. */
num_frames_sent = 0;
num_bits_sent = 0;

/* The transmission sequence must end if the input queue */
/* becomes exhausted.  Other termination conditions are */
/* embedded in the loop. */
while (op_q_stat (OPC_QSTAT_PKSIZE) > 0.0)
	{
	/* Obtain the next frame that would be transmitted. Note that		*/
	/* the frame is not yet actually removed since, based on its		*/
	/* properties, it is possible that it would not be suitable for		*/
	/* transmission.							*/
	pkptr = op_subq_pk_access (0, OPC_QPOS_HEAD);

	/* Obtain the frame's service class. */
	op_pk_nfd_get (pkptr, "svc_class", &svc_class);

	/* Synchronous and asynchronous frames are treated differently. */
	if (svc_class == FDDI_SVC_SYNC)
		{
		/* Obtain the frame's length, and compute */
		/* the time required to transmit it. */
		pk_len = op_pk_total_size_get (pkptr);
		tx_time = (double) pk_len / FDDI_TX_RATE;

		/* Check if synchronous bandwidth allocation for this */
		/* station would be exceeded if the transmission were to occur. */
		if (accum_sync + tx_time > sync_bandwidth)
			{
```

```
                /* The frame could not be sent without exceeding the allocated          */
                /* synchronous bandwidth. Leave the frame enqueued and exit the         */
                /* transmission loop.                                                   */
                break;
                }
            else{
                /* Actually dequeue the frame so that it can be sent.           */
                pkptr = op_subq_pk_remove (0, OPC_QPOS_HEAD);

                /* Send the frame into the ring after other frames have completed. */
                /* Also, account for its propagation delay; because the token propagation */
                /* delay and the frame propagation delay must be consistent, and the */
                /* token propagation delay is specified as a ring parameter (i.e, stations */
                /* are assumed to be equally spaced), the ring is intended to run with */
                /* the "delay" attributes of point-to-point links set at zero. */
                op_pk_send_delayed (pkptr, FDDI_PHY_STRM_OUT, accum_bandwidth + Fddi_Prop_Delay);

                /* increase the consumed bandwidth to reflect this */
                /* transmission.  Also increase synchronous consumption. */
                accum_bandwidth += tx_time;
                accum_sync += tx_time;

                /* Increase counters for transmitted bits and frames. */
                num_frames_sent++;
                num_bits_sent += pk_len;
                }
            }
        else{
            /* The request enqueued at the head of the queue is                 */
            /* asynchronous. It may only be honored if the THT timer            */
            /* has not expired. If it has expired, simply exit the              */
            /* transmission loop, leaving the frame where it is.                */
            if (tht_value >= Fddi_T_Opr)
                break;
            else
                {
                /* Obtain the priority assignment of the frame. */
                op_pk_nfd_get (pkptr, "pri", &pri_level);

                /* If the packet's assigned priority level is too low for it to         */
                /* be serviced, then exit the leave it in place and quit the loop.      */
                if (T_Pri [pri_level] < tht_value)
                    break;

                /* Obtain the frame's length, and compute the time */
                /* which would be required to transmit it. */
                pk_len = op_pk_total_size_get (pkptr);
                tx_time = (double) pk_len / FDDI_TX_RATE;

                /* Determine the requested token class to be */
                /* released after this frame is transmitted. */
                op_pk_nfd_get (pkptr, "tk_class", &tk_class);

                /* If the station is in restricted mode, then it may */
                /* exit this mode if the class is now nonrestricted */
                /* or if the restricted peer is not the addressee. */
                if (restricted)
                    {
                    /* Determine the destination address for the new packet. */
```

```
                op_pk_nfd_get (pkptr, "dest_addr", &dest_addr);

                if (tk_class == FDDI_TK_NONRESTRICTED || res_peer != dest_addr)
                    {
                    /* Exit restricted mode */
                    restricted = 0;

                    /* Modify the token to reflect the mode change. */
                    op_pk_nfd_set (tk_pkptr, "class", FDDI_TK_NONRESTRICTED);
                    }
                }
            else{
                /* Determine the class of the current captured token. */
                op_pk_nfd_get (tk_pkptr, "class", &current_tk_class);

                /* When not in restricted mode, this mode may be entered */
                /* if the passed packet has the appropriate token class requested, */
                /* and the token is not already restricted. */
                if (tk_class == FDDI_TK_RESTRICTED && current_tk_class != FDDI_TK_RESTRICTED)
                    {
                    /* Enter restricted mode. */
                    restricted = 1;

                    /* Store the address of the resticted peer station. */
                    op_pk_nfd_get (pkptr, "dest_addr", &res_peer);

                    /* Modify the token to reflect the mode change. */
                    op_pk_nfd_set (tk_pkptr, "class", FDDI_TK_RESTRICTED);
                    op_pk_nfd_set (tk_pkptr, "res_station", res_peer);
                    }
                }

            /* Extract the frame from the queue so that it can be sent.                  */
            pkptr = op_subq_pk_remove (0, OPC_QPOS_HEAD);

            /* Send the frame once previous transmissions have completed. */
            /* Account for propagation delay as well. */
            op_pk_send_delayed (pkptr, FDDI_PHY_STRM_OUT, accum_bandwidth + Fddi_Prop_Delay);

            /* Increment THT emulation variable, and consumed bandwidth accumulator. */
            tht_value += tx_time;
            accum_bandwidth += tx_time;

            /* Increase counters for transmitted bits and frames. */
            num_frames_sent++;
            num_bits_sent += pk_len;
            }
        }
    }


/* Since the token is about to be sent, its transmission time              */
/* must be reflected in the accumulated bandwidth. This is not             */
/* done in the ISSUE_TK state because when the token is merely             */
/* repeated, full transmission delay is not required, only                 */
/* a small delay for repeating.                                            */
accum_bandwidth += FDDIC_TOKEN_TX_TIME;

/* If the station has no more data to send (synchronous or */
```

```
/* asynchronous), it should indicate this to the token acceleration */
/* mechanism by deregistering its interest in the token. */
if (tk_registered && op_q_stat (OPC_QSTAT_PKSIZE) == 0.0)
	{
	tk_registered = 0;
	fddi_tk_deregister ();
	}
```

transition **TX_DATA -> ISSUE_TK**			
attribute	*value*	*type*	*default value*
name		string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

unforced state **CLAIM**			
attribute	*value*	*type*	*default value*
name	CLAIM	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(See below.)	textlist	(See below.)
status	unforced	toggle	unforced

enter execs **CLAIM**

```
upd = 0;
/* Obtain this station's object id which is used */
/* to access the station's attribute assignments. */
my_objid = op_id_self ();
op_ima_obj_attr_get(my_objid,"GWY",&is_gwy);

/* Using the object id, obtain the ring id.			*/
/* The ring id is used by macros defined in the			*/
/* header block to obtain "ring-global" values,			*/
/* values shared by all stations on a ring.			*/
op_ima_obj_attr_get (my_objid, "ring_id", &ring_id);

/* Initialize ring variable values if this is the first */
/* station to start up in this ring.					*/
if (Fddi_Num_Stations == 0)
	{
	Fddi_Tk_Blocked = 0;
	Fddi_Num_Registered = 0;
	Fddi_Min_Station = 0;
	}

/* Obtain the station's address .  This is an attribute */
/* of this process. Addressing is simplified by */
/* simply using integers, and only one mode. */
```

...
...

```
/* This mode is 16 bit addressing unless the */
/* packet format 'fddi_mac_fr' is modified. */
op_ima_obj_attr_get (my_objid, "station_address", &my_address);

/* Register the station's object id in a global table. */
/* This table is used by the mechanism which improves */
/* simulation efficiency by 'jumping over' idle periods */
/* rather than circulating an unusable token. */
fddi_station_register (my_address);

/* Determine if token passing activities are to be traced. */
tk_trace_on = op_prg_odb_ltrace_active ("fddi_tk");

/* Using the object id, obtain the value of 'T_Req', */
/* the value of TTRT requested by this station. */
op_ima_obj_attr_get (my_objid, "T_Req", &T_Req);

/* The lowest value of T_Req becomes T_Opr for the ring as a whole. */
if (T_Req < Fddi_T_Opr || Fddi_Claim_Start)
	{
	/* The T_Req for this station is lower than any other to date */
	/* so it is installed in the T_Opr variable. */
	Fddi_T_Opr = T_Req;

	/* The flag indicating that the claim process is just */
	/* beginning may now be cleared. */
	Fddi_Claim_Start = 0;
	}


/* Request a self interrupt from the Simulation Kernel at the current */
/* time so that after all stations have executed their claim states, */
/* they can proceed with initializations.  This is necessary */
/* because some initializations are based in the value of T_Opr */
/* and it must therefore be known that all stations have settled */
/* on a final value. */
op_intrpt_schedule_self (op_sim_time (), 0);
```

exit execs **CLAIM**

```
 if (op_intrpt_type () == OPC_INTRPT_STAT &&
      op_intrpt_stat () == UPDATE_INSTAT )
   {
    grp = (int)op_stat_local_read(UPDATE_INSTAT);
    if(grp > 99) op_sim_end("Problem in ETH_MAC IGMP","","","");
    if(MC[grp] == 1) MC[grp] = -1;
       else MC[grp] = 1;
   upd = 1;
   }
```

...
...

transition **CLAIM -> INIT**			
attribute	*value*	*type*	*default value*
name		string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **CLAIM -> CLAIM**			
attribute	*value*	*type*	*default value*
name	tr_105	string	tr
condition	upd	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline