UNIVERSITY OF MIAMI

RIC

(Radio frequency Interface Controller)

ASIC based design of the RF Interface Controller for interfacing
with the Motorola MC 68824 Token Bus Controller
in a Radio Frequency LAN environment

By

Tat Chee Wan

Senior Project/Honors Thesis

Department of Electrical and Computer Engineering

Coral Gables, Florida

December, 1990

Abstract

The requirements for a wireless LAN in a factory environment are discussed. A proposed solution is given, with the aim of adapting existing MAP-based LAN architectures for use in controlling a fleet of Autonomous Guided Vehicles. The design of the RF Interface Controller (RIC) using ASIC methodology for implementing the proposed solution is described. The state machine module listings, circuit schematics, and simulation results are also provided in the appendix.

Acknowledgements

I would like to thank the following people for their help and guidance:

- Dr. M. Kabuka for his direction and suggestions throughout this project.
- Dr. C. Douligeris for his help with various aspects of the Token Bus Protocol.
- Kevin Chase, and all my friends for their support and encouragement throughout this time.
- All others who helped in one way or another, for whom I am grateful.

Table of Contents

List of Figures and Tables

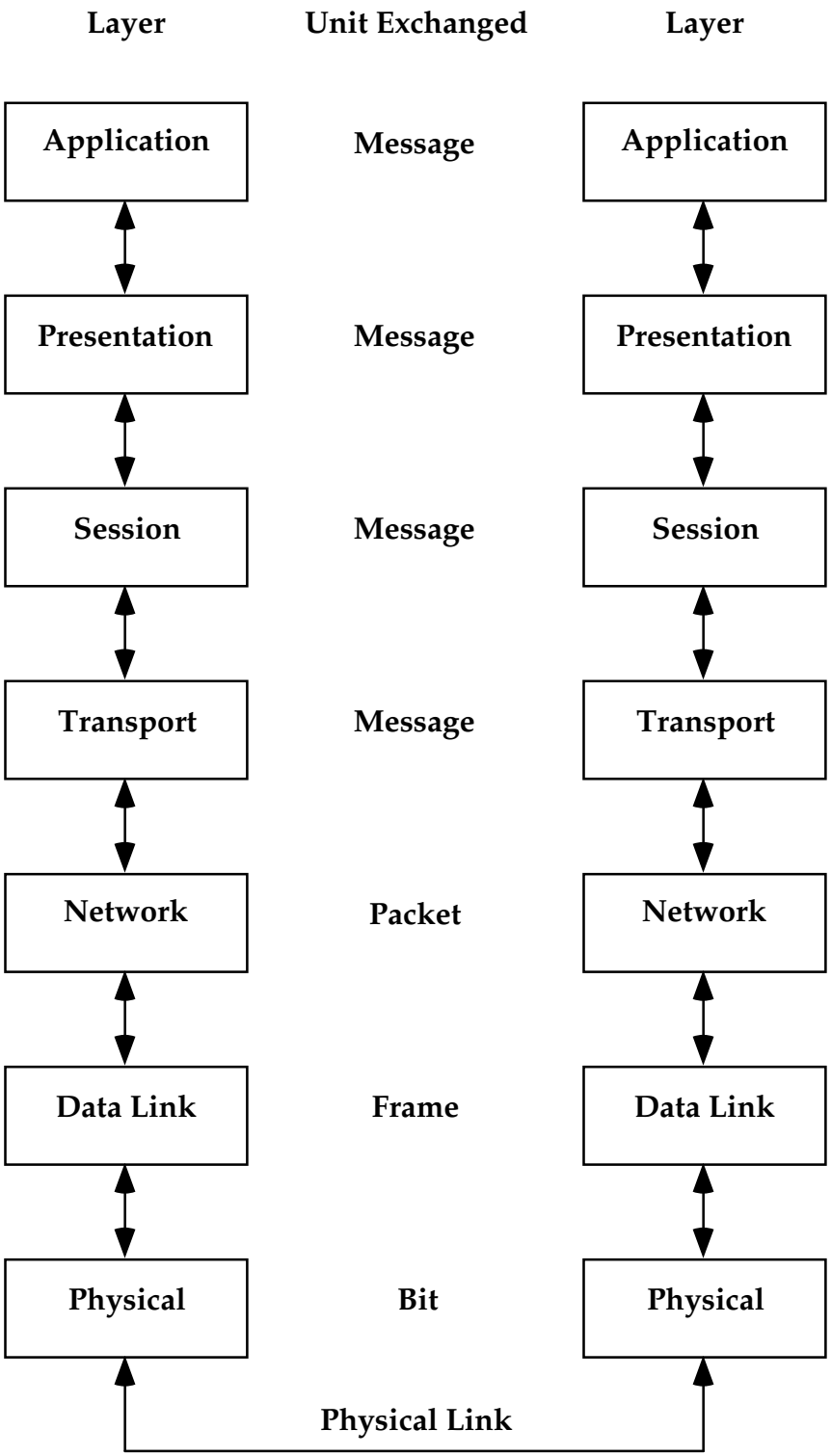
1. Introduction

In the field of automated manufacturing, the need for a communications network which would satisfy both the requirements of dynamic adaptability as well as robustness is critical. This is especially true of distributed real time manufacturing processes. Examples of such needs include synchronizing the actions of two or more robots, and the control of Autonomous Guided Vehicles (AGV) which transport materials from point to point in a manufacturing plant.

Currently, no widespread standard exists for inter-robot and inter-device communications in the manufacturing industry. Various options, including CSMA (Carrier Sense Multiple Access) or Ethernet type local area networks to the token bus based MAP network are available. However, since many of the robots and devices in a manufacturing environment tend to be mobile, the use of coaxial cabling or fiber optics as the physical medium for transmission is not very desirable. As such, most existing LAN systems are not suitable for such situations. An introductory treatment of the issues involved in dealing with networking in a factory environment is given by Rappaport [1].

1.1. OSI Network Reference Model

The *Open System Interconnect* (OSI) basic reference model for communications as specified by the International Standards Organization breaks down the various functions for intersystem communications into seven layers. In Figure 1, the various components are listed and a brief description of each layer is given.



OSI Basic Reference Model for Layered Communications

Figure 1. OSI Basic Reference Model

The *physical layer* is the lowest level in the model, and it specifies the electrical and mechanical aspects of the communications channel as well as the functional control of the data circuits. Its function is to activate, maintain and deactivate the physical connection.

The *data-link layer* serves to establish, maintain and release data links and perform point-to-point error and flow control of data “frames”.

The *network layer* is used for network routing, switching, segmentation, blocking, error recovery, and flow control of packets. It controls whether data is sent up to the application process or down to the physical layer.

The *transport layer* is responsible for providing communications services to the session layer, multiplexing messages over logical connections and maintaining end-to-end reliability control.

The *sessions layer* establishes and terminates logical links between processes and manages dialogue over the links; synchronizing data between application processes.

The *presentation layer* is responsible for converting data to and from a form that is understood by the system, including data encryption and decryption and code conversion.

The *application layer* is the highest layer, and it provides management functions and virtual device services to the programs which use the network services.

From this reference model, we can determine that the physical layer is the segment that needs most modification for use in a factory environment. Since mobility and adaptability is the main criteria, physical cabling is not suitable. Alternatives include inductively coupled communications via the AGV guidance wire [2], Infra-Red (IR) links and Radio-Frequency (RF) transceivers for inter-device transmissions. The use of Inductively coupled communications assumes the existence of guidance wires for AGV movement. In the case of AGVs that use optical navigation techniques, the approach will not be feasible.

1.2. Radio Frequency versus Infra Red Links

IR communications is a relatively new technology, and has the drawback of requiring line-of-sight transmission or relative immobility for error-free communications.

It is also not suitable for places where there are a lot of obstructions or partitions. Existing work in the area includes an IR LAN proposed by Arthur Lessard [3],[4]. RF, on the other hand, is a more mature technology and is better suited for enclosed spaces where IR cannot penetrate. However, it is susceptible to RF interference (RFI), especially in environments where electrical machinery and motors are used. Nevertheless, it is well understood and is therefore preferable.

Another problem with RF communications is that there is often overcrowding of the available frequencies. This is especially acute for radio and television transmission, and is carried over to the frequencies used for communications purposes. FCC regulations regarding RF use is also very stringent, and the use of many mobile robots and other devices in a large factory would require many more channels than which is practical using normal radio communications networks.

1.3. Cellular Radio Communications

One of the solutions for bandwidth overcrowding present in RF communications systems is the use of “Cellular” concepts [5],[6],[7]. This involves dividing up the area to be covered into small enough cells so that frequencies can be reused every few cells since the RF signal does not propagate very much further than the cell boundaries. Each cell has a transceiver station responsible for establishing, maintaining and terminating communications links with whichever device in its vicinity. It is also responsible for coordinating the “handing off” of a communications channel from one cell site to another as the device crosses the cell boundary. Since cell sizes can be adjusted and new cells added to an existing cell configuration by subdividing (or “sectorizing”) cells as needed; the load that a cellular network can handle is limited only by the minimum signal strength required for reliable communications (signal strength above any RFI due to machinery) and the inter-cell interference due to reusing frequencies.

Cellular radio communications is now used mainly in cellular telephone units. However, with the advent of digital cellular telephones, the possibility of integrating cellular PBX and digital computer communications in a single system is made more desirable. The use of a cellular-type link in the manufacturing environment is therefore the best solution for the future.

1.4. Multiple Access Methods for Wireless Communications

The need to partition the RF spectrum into various channels for multiple access gives rise to three general schemes, Time Division Multiple Access (TDMA), Frequency Division Multiple Access (FDMA) and Code Division Multiple Access (CDMA), otherwise known as the “Spread Spectrum” technique. Each system has its advantages and disadvantages, and their respective capabilities are given below.

1.4.1. Frequency Division Multiple Access

FDMA works by subdividing the spectrum into various channels or bands with a specified bandwidth. All television broadcast systems are FDMA in nature. For the cellular environment however, it poses the problem of how to increase the number of simultaneous point-to-point connections while not utilizing more and more bandwidth. This is achieved by making each channel narrower as the total number of required channels increases. Two problems arises directly out of this scheme. The RF modulation unit for each node in the network needs to be changed to reduce their transmission bandwidth, and detection of narrow-band signals require much more sophisticated equipment. As a result, it is not very flexible nor bandwidth efficient in an environment where the load varies significantly since unused spectrum space is tied up in each allocated channel.

1.4.2. Time Division Multiple Access

TDMA is inherently more bandwidth-efficient for a given bandwidth due to its multiplexing nature. An analysis of the channel carrying capacity of a TDMA multiplexed channel vs. a FDMA multiplexed channel would lead to the conclusion that TDMA is more efficient, primarily due to the fact that unused capacity in a FDMA point-to-point link goes to waste while unused slots can always be reassigned in TDMA. Increased load in TDMA is handled by increasing the interval between each slot belonging to a particular point-to-point connection and interleaving more connections per channel.

1.4.3. Code Division Multiple Access

CDMA or Spread Spectrum technology can be categorized into two different approaches — Direct Sequence Spread Spectrum (DSSS) and Frequency Hopping Spread Spectrum (FHSS) [8]. DSSS works by modulating the data signal at a given baud rate with a very wide-band signal at a much higher baud-rate using a pseudo-random

sequence. This “spreads” the original data signal across a much larger spectrum and enables more than one signal to overlay each other at the same frequency. The disadvantage of DSSS is that interference is always present during transmission due to other users. As the number of users increase, the level of interference increases proportionally. There is the issue of synchronization as well. FHSS works by not using a single frequency channel for transmission; instead, it “hops” from one frequency to another while transmitting part of the data signal at each particular frequency. The advantage of spread spectrum is the much lowered vulnerability to tapping and interference. When applied to portable communications, however, it purports to reduce the susceptibility of the channel to fading and noise. It is theoretically possible to transmit in an environment where the Signal to Noise ratio is less than one (ie: the noise level is stronger than the signal).

The FCC has released frequencies in the 902-928 MHz, 2400-2483.5 MHz and 5725-5850 MHz bands [9],[10] for experimental spread-spectrum communications using transceivers of less than 1W with a minimum of 20 dB out-band attenuation. However, those frequencies are considered noisy due to interference from microwave ovens and ham-radio usage. Nevertheless, the noise level decreases rapidly around 1 GHz and becomes less of an issue compared to fading and multipath effects [1]. Commercial systems utilizing spread-spectrum technology for use in the office environment include systems by Agilis Corp. for use with Ethernet Networks, and BICC Technologies for use with Ethernet and Token Ring Networks. The performance of such systems in a mobile factory environment is unknown, as they were designed to operate in offices with relatively stationary equipment.

1.5. Signal Attenuation in a Factory Environment

Various fading phenomena are described in [1]. Path loss due to the presence of obstacles in the transmission path causes attenuation in the transmitted signal strength. The measured path loss is described by:

$$\text{Path loss} \propto d^n, \text{ where } n = 2 \text{ for free space.}$$

Empirically, n varies from 1.5 to 2.8, which is much less than in an enclosed office environment. However, over a distance of 50m, it does introduce from 10 to 40 dB of attenuation to the transmitted signal strength. More importantly, the effects of Rayleigh and Rician fading causes the signal level to undergo what is called fast-fading. This is typically between 30-35 dB over very short distances (1.3 m) [11]. Rayleigh

fading is due to the presence of multiple paths from the transmitter to the receiver. It is also called multipath fading. Rician fading, on the other hand, occurs between stationary transmitters and receivers having a single transmission path. The movement of objects in and out of the transmission path is what causes the fading. Saleh and Valenzuela presented models on the characteristics of indoor fading channels in [12].

2. LAN Protocols and considerations for wireless LANs

The LAN protocol in common use today is based on the IEEE 802 protocols; the main contenders being the 802.3 CSMA/CD (Ethernet) protocols and the 802.4 Token Bus protocol adapted by General Motors called Manufacturing Automation Protocol (MAP). Ethernet is a “non-deterministic” type of protocol. This means that there is a possibility that a node in the network can be blocked by existing traffic and is not able to transmit its packets indefinitely. The possibility increases dramatically as the network load increases. In contrast, Token Bus networks are “deterministic” since a token is passed from node to node to denote access rights and access can be prioritized so that a node can always obtain the token in a worst case time. The efficiency of the Token Bus protocol is also much better than that of Ethernet for a heavily used channel [13]. For a real time factory environment, Ethernet is not suitable. The Token Bus protocol therefore becomes the preferred standard.

In addition, the question of interfacing the mobile robots to the main LAN (which uses standard wiring schemes) running Token Bus has to be resolved. Lessard and Gerla [3] proposed a hybrid CSMA/MLMA method using “satellites” for synchronizing the communications between stationary and mobile nodes in the network. However, this would present a problem with protocol conversion between the Token Bus Network and the CSMA/MLMA network connecting the mobile nodes. Furthermore, the scheme outlined in [3] allows for communications between mobile nodes and the satellites only. Direct communications between mobile nodes is not possible. The alternative is therefore to develop a Token Bus Network for the mobile units while accounting for the issues of spectrum re-use (cellular concepts) and data transmission rate restrictions.

The nature of a LAN is different from that of cellular telephones in that only one channel (the broadcast channel) is required for all the nodes in a particular network, while cellular telephones require a dedicated set of channels (two for each connection) for each node or user in that cell. In that respect, the issue of the number of channels required is less critical for LAN applications. FDMA as it is used in cellular telephone systems is too

restrictive due to the problem of available bandwidth — each channel of 25 kHz or less does not allow data rates greater than in the tens of kilobits range. TDMA simplifies the problem of the simultaneous transmission of data and voice signals. However, it requires the use of fixed data block sizes, which is too restrictive for a LAN environment that can have variable length messages that are usually larger than the block sizes specified for TDMA cellular telephone systems.

From [1],[5] the current upper limit for data rates seem to be in the 100 kbps to 400 kbps range. This requires the use of a channel with bandwidth much greater than that specified for cellular telephone systems. However, existing FCC regulations do not permit the use of high bandwidth channels that are required for the RF LAN. The need for compliance therefore necessitates the use of spread spectrum techniques. Despite the increased complexity of CDMA (spread spectrum), the use of spread spectrum techniques does provide an advantage in dealing with interference and fading [1],[6],[7] that occurs with RF channels.

The modulation technique used has to be bandwidth efficient and has to have a low enough average probability of error in order to support the required transmission rates. The use of multi-level modulation would increase the bit per baud ratio without utilizing additional bandwidth. Such schemes include m-PSK, m-QAM techniques. The average Bit Error Rate (BER) for a RF environment is substantially higher than that achieved in a cable based system. This is due to the problems of Rayleigh and Rician fading, multi-path distortion, delay spread, and RFI [1],[6],[7]. However, for effective communications, a BER of better than 10^{-4} is required. This can be achieved by utilizing antenna diversity, pulse shaping to increase spectral efficiency and making the packets as short as possible.

In order to deal with the above problems, a combination of the various techniques are used. Each “cell” becomes a subnetwork in the LAN, while the RF spectrum is divided into channels in order to provide for several channels per defined subnetwork. Saleh and Cimini [14] described the use of Slow Frequency Hopping (SFH) in a TDMA scheme with Reed-Solomon burst error correction for a radio channel. However, the use of SFH requires complex synchronization. Another strategy use DSSS and vary the carrier frequency from frame to frame in order to account for deep fades. A 5 level QAM with pulse shaping scheme is employed to meet the requirements of the Token Bus standards (using alternative modulation techniques described in the Appendix of Chapter 14 in [15]) and to reduce the transmission baud rate. DSSS is applied to the signal just

prior to transmission in order to provide for multiple access according to FCC regulations. An analysis of the bandwidth efficiency of M-ary DSSS systems is provided by Pahlavan and Chase [16].

3. RF LAN Organization

Before the details of the network can be specified, it is necessary to consider the environment and requirements for which the wireless communications is needed. It is tailored for a manufacturing environment which requires the use of AGVs to transport materials from one part of the plant to another. The scheduling and synchronization of the activities of each AGV is centralized, but each AGV is semi-autonomous in that they require only the specification of a starting point and a destination — the AGV performs the path planning itself. However, “critical zones” exist where there are intersecting paths and where traffic control is necessary. Each critical zone is managed by a “Local Controller” (LC) which issues messages for the AGV to halt or proceed, as well as forward new instructions to each AGV. The AGVs, in turn, provides status updates to the LC as to its location and problems encountered (obstacles, mechanical faults, etc). The various CCs communicate with each other and with the larger factory-wide network via standard LAN technology (coaxial cables). A logical way to partition the network therefore, is to make each Local Controller into its own wireless “subnetwork” which linked together through a high speed backbone. This is similar to the scheme proposed by Lessard and Gerla except that the LC acts as a gateway between the subnet and the main network, while also executing control functions of its own, as opposed to the “satellites” which provide networking functions in a single large network only. Peer to peer communications between AGVs in each subnet is also possible.

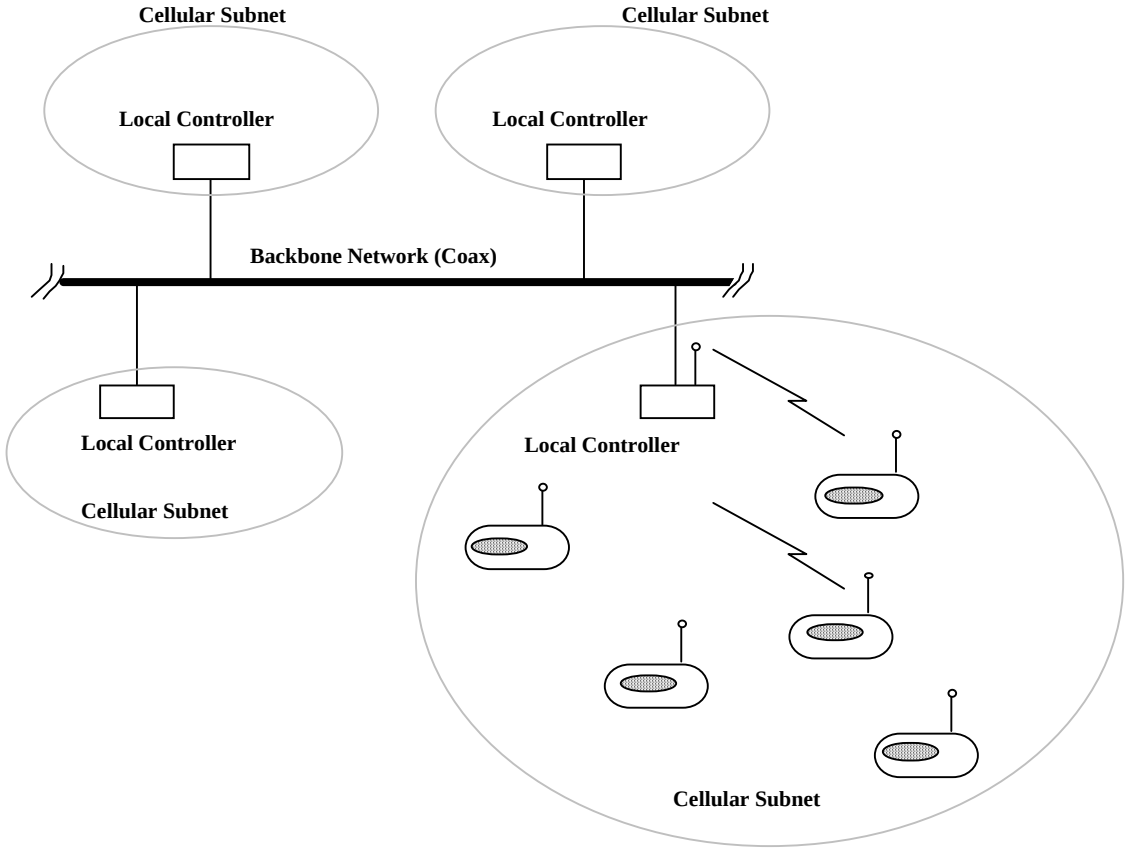


Figure 2. RF LAN organization

Figure 2 is a representation of the layout of the network. A further note is that each AGV transceiver and each LC transceiver has to be of the same power, since each transceiver in a particular subnet has to be able to reach all the other nodes in that subnet.

3.1. Frequency Assignments

Each subnet containing the LC will utilize two channels at any given time — a communications channel for the network itself, and a control channel for broadcasting control information. Individual mobile units currently assigned to that LC will use the information transmitted on the control channel to tune in to the specified communications channel frequency (subnet frequency) and operate as a node in that particular subnet. The frequency of the control channel remains static or quasi-static, while cyclical frequency hopping is used on the communications channel to counteract the effects of deep fades and interference. Frequency switching is done on a frame by frame basis so as to reduce the complexity of the transceiver. DSSS is applied to the signal just prior to transmission; synchronization is provided by the Local Controller. The reason for using static

frequency assignment for the control channel is to facilitate “handing off” of an AGV leaving the critical zone controlled by one LC to the subnet of another LC. The control channel frequencies are “published” so that the AGVs can tune in to the new control channel without knowing the new subnet frequency. Interference on the control channel is expected to be less critical since the only information transmitted is the current frequency/next frequency information as well as frequency switching commands which constitutes short frames that are transmitted repeatedly.

The use of very slow frequency hopping will make the transmission channel more robust against deep fades or severe interference. For even better performance, the receiver circuitry can be made to sense the onset of a fade and force a switch in frequency to one which is not experiencing fading at that point in time. From [14], it is shown that the separation of two adjacent channels should be greater than 4 MHz for a TDMA channel in order to have a better than 10 dB difference in signal strength for a delay spread of 25ns.

3.2. Frame Format for Token Bus Packets

In order to specify the functional behavior of RIC, the Token Bus Protocol needs to be examined. The following information is obtained from [15]:

The token bus standard defines a packet generated by the Medium Access Control (MAC) sublayer as having the following characteristics shown in Figure 3:

Frame Format for Token Bus Protocol

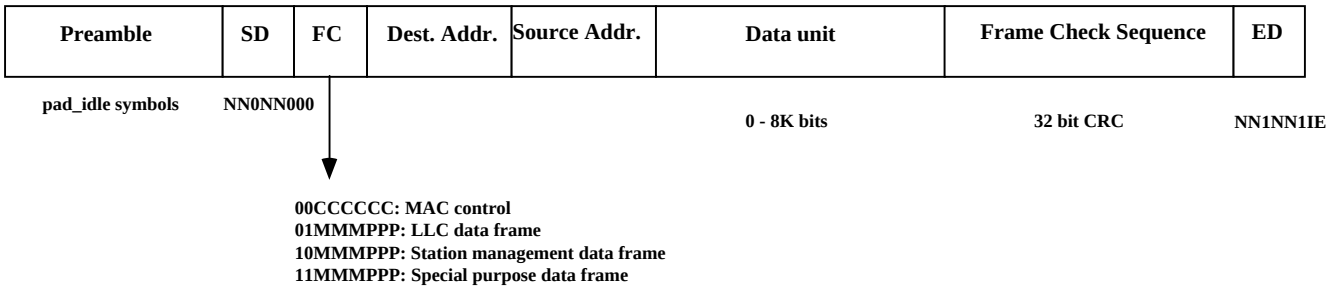


Figure 3. Token Bus Format

When the Token Bus Controller (TBC) is not transmitting data, it transmits “Silence” and the transmitter circuitry is disabled. A Preamble precedes each packet and

contains “Pad idle” symbols which are used for synchronization and for allowing the Token Bus Controller (TBC) to process the previous frame. These symbols are detected by the receiver as out of sequence data symbols. The standard specifies a minimum duration of 2 μs for the Preamble. The maximum amount of Preamble is set by the “jabber” control in the physical layer. For the purpose of the RF LAN, longer silence and preambles may be necessary in order for the AGVs to regain synchronization after channel switching has occurred.

The Start Delimiter (SD) indicates the beginning of the frame. “Non data” (N) symbols are used to indicate this special field. They are distinct from the One and Zero symbols and are used to indicate the beginning and end of the frame.

The Frame Control (FC) field defines the type of packet that is being sent. The defined types are MAC control data for station identification and initialization, LLC data (normal data packets), Station Management data and Special purpose data which is reserved for special use. The control channel will utilize a variation of the Station Management data frame exclusively to broadcast the current and subsequent frequency channel used for transmission. This reduces the switching time overhead for the data channels to a minimum since the synchronization is done with “out band signaling” which separates the data transmission from the control functions of the physical layer.

The Destination Address (DA) and Source Address (SA) are defined as either 16 bit or 48 bit addresses, depending on the size and complexity of the network. This allows the partition of addressing into various subgroups (or zones).

The Data Unit contains the message to be sent. This is followed by a 32 bit Frame Check Sequence (FCS) for the data packets, defined in [15]. The station management data utilizes a different FCS. Instead of the normal 32 bit sequence, a 16 bit sequence, CRC16 [17] is used. Since the frames themselves are very short, transmission time for the station management packets will be reduced by the use of a shorter FCS.

The transmission is ended with an End Delimiter (ED) which contains Non data symbols and an Intermediate bit to signify a continuing packet, and an Error bit which is used to indicate transmission errors between station repeaters. The End Delimiter used for the station management data is slightly different from the normal End Delimiter. It uses the bit pattern NN1NN000 instead of the normal NN1NN100. This serves to differentiate the station management frames from normal data frames.

3.2.1. MAC Symbol Encoding

The Token Bus Standard specify various methods for encoding MAC symbols into Physical (PHY) symbols for transmission. PHY symbols refer to the entities which have been made suitable for transmission on the physical media.

Four methods are specified:

1) The standard One bit per baud 3 symbol duo-binary encoding scheme.
2) Two bit per baud 5 symbol encoding
3) Two bit per baud quadrature 3 symbol encoding
4) Four bit per baud quadrature 5 symbol encoding.

Method (3) involves modulating two method (1) modulated bit streams to form a quadrature output stream. Method (4) is the combination of methods (2) and (3) to give a quadrature output consisting of 4 bits per baud. The one chosen for this project is the 2 bit per baud 5 symbol encoding scheme (method (2)) as it handles a single bit stream and achieves the required objective of increasing information density per transmitted symbol. The PHY symbol rate is therefore half that of the MAC symbol rate.

The notation used to represent MAC to PHY symbol encoding is as follows (Table 1):

MAC symbol pairs	PHY symbols
zero, zero	{0}
zero, one	{1}
one, zero	{3}
one, one	{4}
Non-data, Non-data	{2}
Pad-Idle	{4} {0}
Pseudo-Silence	{2} {0} {4}

Table 1. Token Bus 5 Symbol Encoding Representation

The PHY symbols are numbered according to a duo-binary scheme described in [1]. {0} refers to a minimum amplitude signal, while {4} refers to a maximum amplitude signal. The numbering scheme is arbitrary for QAM modulation since amplitude is not the sole determinant in deciding which signal represents what symbol. However, the notation is kept to maintain consistency with [15]. Pseudo-Silence refers to the symbols that a broadband Token Bus network with a headend remodulator would transmit when it

encounters a Silence symbol. This is to ensure that receiving stations have a clock signal to lock onto. Since the RF LAN would not require a headend remodulator, Pseudo-Silence is not implemented.

The Start and End Delimiter encoding is modified slightly to account for the change in representing Non-data pairs. (A non-data followed by a data bit is not allowed). For example, the SD is encoded as follows:

N, N, 0, N, N, 0, 0, 0 becomes {2} {0} {2} {0}

where the second PHY symbol {0} comprises the first two data bits in the octet.

3.3. Frame Transmission and Reception

The transmission of a Token Bus frame involves the following procedure:

The Token Bus Controller (TBC) is initially transmitting silence (no transmission). When the TBC is ready to transmit a new frame, it will transmit at least four octets (4 * 8 = 32 bits) of Pad-Idle symbols to provide the receiver a clock pulse for synchronization. The Start Delimiter (SD) is then transmitted, causing the “data scrambler” to reset itself. The “data scrambler” randomizes the spectral components of the output signal as well as reduces the possibility of a long stream of ones or zeros. This scrambled bit stream is then fed into the Physical Symbol Converter to generate the corresponding PHY symbols. A “Kicker Inserter” then takes the transmission PHY symbol stream and inserts a “kicker” whenever two octets of identical symbols are present in the output stream. This results in a stream of {0} symbols being transformed in the following manner:

{0} {0} {0} {0} {0} {0} {0} {0} {0} {0} {0} {0} {0} {0} {0} {0}

becomes

{0} {0} {0} {0} {0} {0} {0} {0} {0} {0} {0} {0} {0} {0} {0} {2}.

The resulting transmission stream is then passed to the external RF circuitry for spreading and QAM modulation before transmission.

The opposite occurs for frame reception. Upon detection of the Pad-Idle symbols, the receiver will lock onto the signal and wait for the SD octet. The “kicker deleter” then removes any kickers which were inserted into the data stream. The signal is then fed into

the Physical Symbol Converter to transform the PHY symbols back into MAC symbols. The resultant bit stream is then passed through the “descrambler” to retrieve the original message bits.

3.4. Specifications for RIC

For this project, a total of five frequency channels will be specified for each subnet. Up to four channels can be used for communications in each subnet, while an additional control channel is used for the transmission of channel frequency information from the Local Controller to the mobile units. The control channel implements a subset of the Token Bus Protocol, as a modified Token Bus Packet (with a CRC-16 FCS) is continually transmitted by the Local Controller (master) to all the AGV’s (slaves) in its subnet. Figure 4 indicates the channel switching undergone as transmission progresses in the subnetwork.

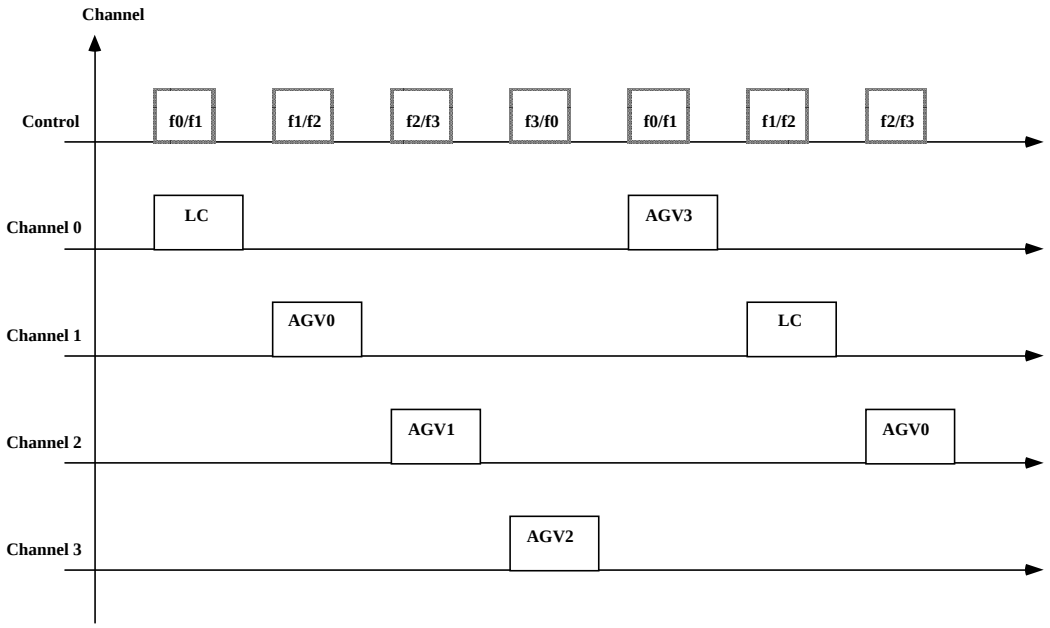


Figure 4. Graph of Channel Frequencies vs. Time

Error-correction coding will not be dealt with in this project. However, it can conceivably be included as a function block in the RF Interface Controller or provided by the RF modulation section.

The design would be rated for a maximum of 500 kbps although actual throughput would probably be much less depending on the available technology. However, the

choice of 500 kbps simplifies the integration of Local Controllers into existing Token Bus networks, since 1 Mbps is a defined speed in the 802.4 standard.

The project involves implementing the physical layer interface between the Token Bus Controller (TBC) and the RF modulation sections. The RF Interface Controller (RIC) will handle frequency switching and channel synchronization. A block diagram of the various components in the controller is given in Figure 5.

Block Diagram of Communications Controller

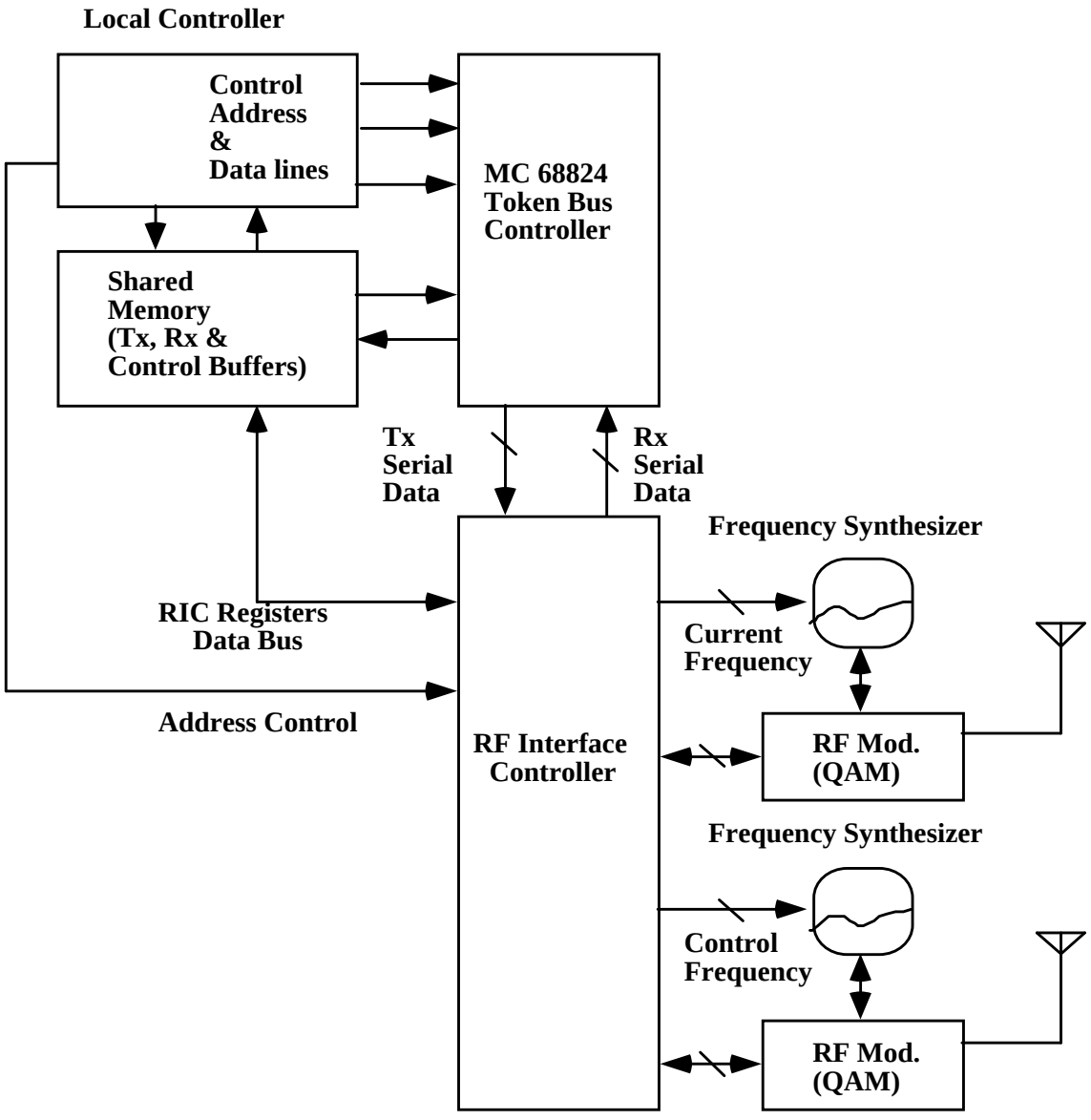


Figure 5. Communications Controller Block Diagram

4. Functional Blocks for RIC

There are five functional blocks in RIC:

The Transmitter section receives data and RIC commands from the TBC, and converts the data into Physical symbols to be transmitted by the RF section.

The Receiver receives data from the RF section and converts it back into a binary bit stream. It also serves as the command decoder and returns status information to the TBC.

The Station Management section has two modes. In master mode, it transmits the frequency channel in use and updates the frequency controls to the RF section as the need arises. In slave mode (used by the AGVs), it receives the frequency channel information and updates the appropriate frequency controls. Frequency selection is done by means of an 8 bit register which is used by a frequency synthesizer circuit to generate the appropriate signals.

The Registers section holds all the registers necessary for the functioning of the RIC, including status information, mode selection as well as channel frequency information and address data for the Station Management section.

The BIST Controller section performs self testing when the $\overline{\text{TEST}}$ signal is activated. When testing is completed, it will indicate whether the RIC has passed the tests.

Figure 6 lists the various modules of the RIC. The figure indicates primary inter-module control signals; clock distribution circuits are not included. In addition, each module except for the BIST controller has its own test circuitry comprising of a Pseudo Random Pattern Generator (PRPG) and an Multiple Input Serial Register (MISR). The functions of each module are further explained in the following sections. Submodules are partitioned into two parts depending on the clock rate. Clock signals used in each part are indicated in the respective section above or below the dashed line separating the modules with different clock rates. Each “primitive” submodule is then described in the sections following the module descriptions. Detailed module interconnections can be found in the circuit diagrams in the appendices.

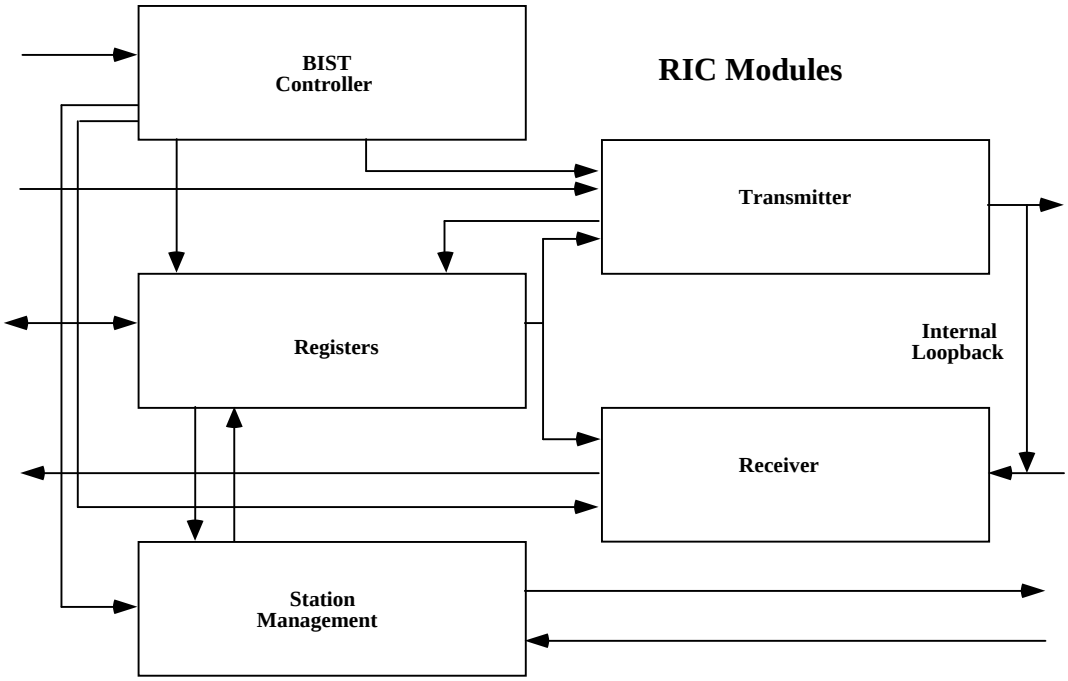


Figure 6. Block Diagram of the various RIC modules

4.1. Control Signals for RIC

The Motorola MC68000 Family Reference [18] describes the use of the MC 68184 Broadband Interface Controller (BIC) with the MC 68824 Token Bus Controller in a coaxial broadband environment. Since many of the functions of the RIC would be similar to that of the BIC, the internal organization of the RIC is therefore derived from the specifications of the BIC. The similarities include the various functional blocks which handles scrambling and descrambling of the input bit stream, the control signals that are used in interfacing with the TBC as well as the radio frequency modulation sections. However, the BIC does not support two bit per baud 5 symbol encoding. It also requires a serial command input interface accessed through the TBC. The RIC employs a parallel interface in order to simplify the design and speed up the initialization process. In addition, the RF LAN uses a dedicated control channel, and the RIC generates and decode station management frames without any intervention from the Token Bus Controller. Figure 7, which is derived from the specifications of the MC 68184, details the control signals for interfacing with the TBC and the radio frequency modulation stages. Note that IMPULSECLK has a frequency half that of the TXCLK and RXCLK. This is due to the two bit per baud encoding scheme. The RF section generates the RXCLK, IMPULSECLK, CTRLRXCLK and CTRLTXCLK signals. TXCLK is usually derived from RXCLK.

In addition, TXCLK2 and RXCLK2, which have the same clock rate as IMPULSECLK, are derived from TXCLK and RXCLK respectively. They are fed to various submodules of RIC which handles the physical symbols. Binary data is clocked into and out of RIC using TXCLK and RXCLK, while physical symbols are clocked using TXCLK2 and RXCLK2. The station management (frequency control) section transmits and receives data using the same baud rate as the physical symbols.

Two types of control signals are defined for RIC. There are those that deal with interfacing with the TBC, and those that deal with interfacing with the RF section. Since active low and active high signals are both present in RIC, there is a need to define exactly what is meant by an active or an inactive signal. The convention used for describing signal levels is to indicate an active high signal as being *asserted* if its logic level is 1, whereas an active low signal is asserted if its logic level is 0. Similarly, an active high signal is *negated* if it has logic level 0, and an active low signal is negated if its logic level is 1. (This is consistent with the convention used in [18]. Figure 7 indicates the signals used for interfacing between the TBC and RF Sections.

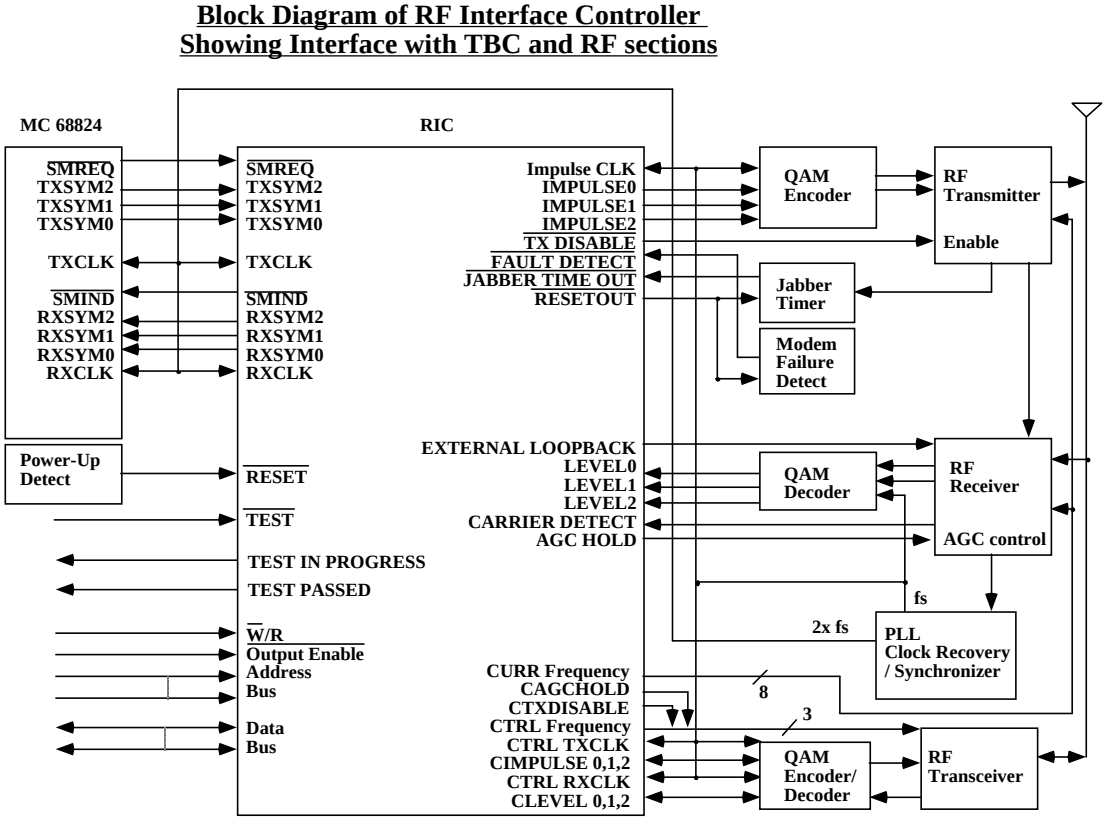


Figure 7. Block Diagram of RIC showing Interface and Control Signals

4.1.1. The Power-up Reset Signal

$\overline{\text{RESET}}$: The $\overline{\text{RESET}}$ line must be asserted (held low) for at least four clock cycles upon system power-up in order to initialize RIC to its Reset state. This is a Hardware Reset, and is distinct from the **Reset** command which is given via TXSYM(2:0) and which performs a Software Reset. When $\overline{\text{RESET}}$ is asserted, the Command/Data Decoder, Command/Data Encoder, BIST Controller, clock division circuitry (for generating TXCLK2 and RXCLK2) are reset. All the other modules are reset using the reset command.

4.1.2. RIC Initialization Sequence

In order for RIC to function properly, $\overline{\text{RESET}}$ is asserted for at least two TXCLKs in order to initialize RIC. The hardware $\overline{\text{RESET}}$ is similar to the software Reset command except that the hardware reset is performed upon power-up and subsequent resets are done using the Reset command. Station Management mode is then entered by asserting the $\overline{\text{SMREQ}}$ line. In response, $\overline{\text{SMIND}}$ will be asserted and ACK present on RXSYM(2:0) when RIC is ready to accept commands (refer to Receiver section for details). Valid commands are Reset, Loopback Disable, Enable Transmitter, and SM mode (register control). The command encodings are given in the Transmitter section.

A Reset will cause $\overline{\text{RESETOUT}}$ to be asserted and the following sequence of events to occur:

(a) any physical error that is present will be cleared.

(b) Bits 0,1, and 2 in the Status Register will latch any error conditions that were present when the reset became effective. Other bits are set or reset to indicate Internal Loopback mode with Loopback Enabled, and Transmitter Disabled.

(c) RIC enters the Internal Loopback mode and all frequency and address registers are cleared. Individual bits in Register 1 are set or reset in order to bring RIC into Internal Loopback mode, and all submodules are enabled. 16 bit addressing is set, and RIC is placed into slave mode.

(d) Transmitter is disabled, causing $\overline{\text{TRANSMIT DISABLE}}$ to be asserted and Silence present on IMPULSE(2:0).

(e) AGCHOLD is negated.

4.1.3. Internal Loopback

RIC enters the Internal Loopback mode after a Reset, and remains in that mode until either the Loopback Disable command is given on TXSYM(2:0) or if the Control Register is modified. When in Internal Loopback mode, the Status Register is set appropriately, and $\overline{\text{TRANSMIT DISABLE}}$ is asserted to disconnect RIC from the RF stages. IMPULSE(2:0) will also be transmitting Silence. The Transmit Clock Synchronizer is bypassed, and the output from the Kicker Inserter is fed directly to the Receiver module using RXCLK2 (derived from RXCLK). The Kicker Deleter and Descrambler is enabled irrespective of the state of Carrier Detect. AGC Hold is not activated in Internal Loopback mode. In addition, the Station Management module is disabled — MAC mode will be activated immediately when $\overline{\text{SMREQ}}$ is negated.

4.1.4. External Loopback

External Loopback mode is activated when the Control Register is initialized to external loopback mode and the Enable Transmitter command has not been passed to RIC. In External Loopback mode, the External Loopback signal is asserted, and Carrier Detect is ignored. This means that $\overline{\text{TRANSMIT DISABLE}}$ is not asserted when Carrier Detect goes low. Station Management is enabled but RIC will enter MAC mode without waiting for the initialization sequence to complete. External Loopback mode allows the external RF circuitry to be tested.

Loopback mode is not entered if any other combinations were present. (Refer to Registers section for details).

4.2. Transmitter Section

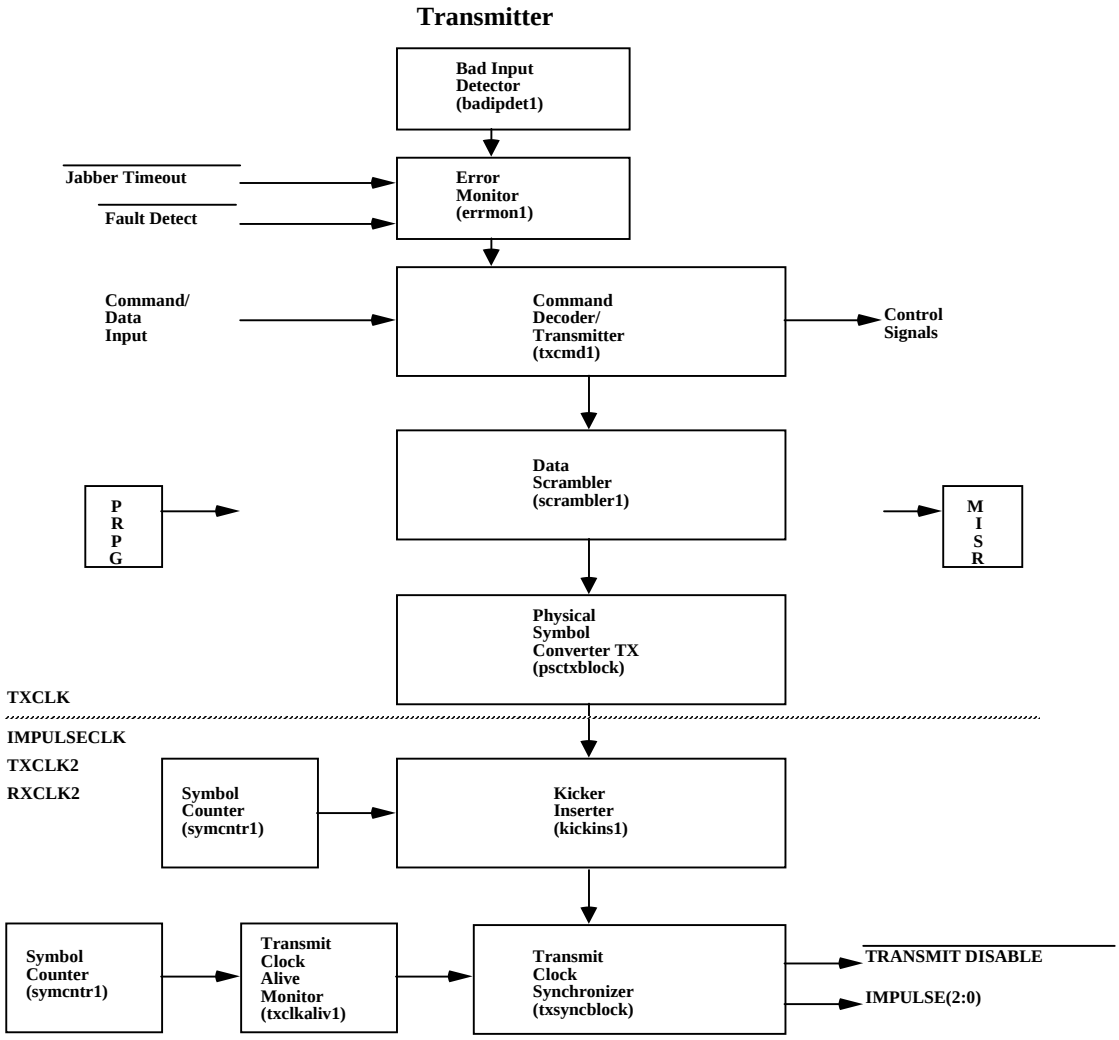


Figure 8. Block Diagram of Transmitter module

Figure 8 indicates the various submodules that make up the Transmitter section. The Command/Data Decoder interprets the input as either commands to the RIC or data to be transmitted to the physical media. Commands on the input lines modify the control signals going to the various modules in RIC. Commands are indicated using $\overline{\text{SMREQ}}$. When in command (Station Management) mode, the Command Encoder in the Receiver module also gives an indication of the current command on the output. The Bad Input Detector detects invalid input sequences on the input and generates an error signal if an error occurs. The Error monitor takes its inputs from the Bad Input Detector, the Jabber Timeout input and the Fault Detect input and forces a Physical error indication on RXSYM(2:0) if any of the three errors occur. This will cause RIC to enter an error state

which is held until a reset command is received by RIC. The error status can then be read from the Status Register (described in the Registers section).

When RIC is in data (MAC) mode, input data is first passed through the Data Scrambler (defined in [15]). It takes the input bit stream and randomizes it by dividing it with the polynomial $(1 + X^{-6} + X^{-7})$ before passing it to the Physical Symbol Converter TX for symbol generation. This block takes two bits and encodes it as one symbol to be used later for QAM transmission. Effectively, the baud rate required and hence the bandwidth for each channel is half of the normal bit oriented transmission technique. The Kicker Inserter mentioned in the Token Bus Packet format section takes care of long streams of identical symbols and inserts a "kicker" wherever necessary.

The Transmit Clock Synchronizer enables the Transmitter RF circuitry to have an arbitrary clock phase with respect to the rest of the Transmitter section. It has the same clock rate as TXCLK2 (generated internally by the RIC) but at an arbitrary phase relationship. This decouples the propagation delays of TXCLK from the RF circuitry. The Transmit Clock Synchronizer also generates $\overline{\text{TRANSMIT DISABLE}}$, which switches the RF stages on and off depending on the respective control signals. The TXCLK Alive monitor ensures that the RF Transmitter does not keep transmitting if the inputs to RIC are disabled accidentally while in the middle of a message. It relies on RXCLK to determine if a clock signal is present on TXCLK.

When data is first present on TXSYM(2:0), it takes about 10 TXCLKs to propagate through the Transmitter and appear at the outputs IMPULSE(2:0). In addition, RIC takes some time to enter MAC mode from Station Management mode after a reset because the Data Channel Frequency needs to be determined (Frequency Acquisition phase) before any transmission can begin.

4.2.1. Physical DATA Request Channel (TBC to Transmitter)

TXCLK: All signals are clocked into RIC with the positive edge of TXCLK. It should not be greater than 1 MHz in order for RIC to function properly.

$\overline{\text{SMREQ}}$: This signal is used to switch RIC to the MAC (data transmission) mode when negated or to Station Management (SM) mode when asserted. In MAC mode, it takes the data inputs on TXSYM(2:0). In SM mode, it takes TXSYM(2:0) as commands for RIC.

TXSYM(2:0): The inputs have the following meaning when in SM mode. ($\overline{\text{SMREQ}}$ = 0) as shown in Table 2. All other combinations are not recognized and RIC will ignore them.

State	TXSYM2	TXSYM2	TXSYM0
Reset	1	1	1
Loopback Disable	1	0	1
Enable Transmitter	0	1	1
SM mode (Register Control)	0	0	0

Table 2. Transmitter Commands

When in MAC mode ($\overline{\text{SMREQ}}$ = 1), the data is encoded as indicated in Table 3.

Symbol	TXSYM2	TXSYM2	TXSYM0
Zero	0	0	0
One	0	0	1
Non-Data	1	0	–
Pad-Idle	0	1	–
Silence	1	1	–

Table 3. Transmitter Input Symbols Encoding

The meaning of the above symbols are defined in Table 1.

4.2.2. RIC Transmitter Control Signals (Transmitter to RF stage)

$\overline{\text{RESETOUT}}$: This signal is asserted when a Reset command is received by RIC, or if the $\overline{\text{RESET}}$ pin is asserted. It generates the reset signal for all the submodules in RIC except for those reset by the $\overline{\text{RESET}}$ signal.

$\overline{\text{JABBER TIMEOUT}}$: This signal is the input from the Jabber Timer circuitry in the RF Module. It is asserted when the transmitter has been transmitting for more then half a second. The Jabber Timer is reset when the $\overline{\text{RESETOUT}}$ line is asserted. If $\overline{\text{JABBER TIMEOUT}}$ is not negated when $\overline{\text{RESETOUT}}$ is negated, then RIC will again indicate an error.

$\overline{\text{FAULT DETECT}}$: This signal is asserted when external modem failure detect circuitry detects a fault and the RIC will stop transmission. It is similar to $\overline{\text{JABBER TIMEOUT}}$ in operation.

EXTERNALLOOPBACK: This signal is asserted when RIC is in external loopback mode. (Transmit Enable has not been issued by TBC, and LS1 and LS2 set up for external loopback).

IMPULSECLK: The Impulse Clock is used to clock out Impulse(2:0) and is half the frequency of TXCLK. However, it can have an arbitrary phase relationship with TXCLK.

IMPULSE(2:0): The output from RIC appears on Impulse(2:0) encoded as 2-bit Physical symbols based on the following table (Table 4):

Physical Symbol	**Impulse2**	**Impulse1**	**Impulse0**
Silence	1	1	0
Impulse0 {0}	0	0	0
Impulse1 {1}	0	0	1
Impulse3 {3}	0	1	0
Impulse4 {4}	0	1	1
Impulse2 {2} Non-Data	1	0	0
Error *	1	0	1

Table 4. Transmitter Output Physical Symbol Encoding

* Error is generated when an invalid sequence is detected by the Physical Symbol Converter (to be defined later). It will be asserted until Silence is received on TXSYM(2:0).

$\overline{\text{TRANSMIT DISABLE}}$: It is asserted (forced low) under any of the following conditions:

(a) After a reset command.

(b) When Transmitter is disable (Transmitter Enable command not received by RIC yet). This is indicated by Bit 4 of Register 0.

(c) When $\overline{\text{SMREQ}}$ is asserted and RIC is in SM mode.

(d) When internal loopback is in effect.

(e) When (CARRIER DETECT & EXTERNAL LOOPBACK) are negated.

(f) when TXCLK has zero frequency (caused by TXCLKALIVE not detecting a transition on TXCLK for 32 RXCLKs or 16 IMPULSECLKS). This is to prevent a remotely connected TBC is disconnected and the RIC is left on unintentionally.

(g) After Silence is transmitted on Impulse(2:0) for more than 10 clock cycles. $\overline{\text{TRANSMIT DISABLE}}$ is negated once data is presented to the RIC.

(h) When a Physical Error is reported by the Error Monitor (RXSYM(2:0) will indicate a Physical Error to the TBC).

$\overline{\text{TRANSMIT DISABLE}}$ allows the RF circuitry time to power up and power down, since it is negated a few clocks before data is presented to the RF transmitter and asserted a few clocks after a frame has been transmitted and only silence is present on IMPULSE(2:0).

4.3. Receiver

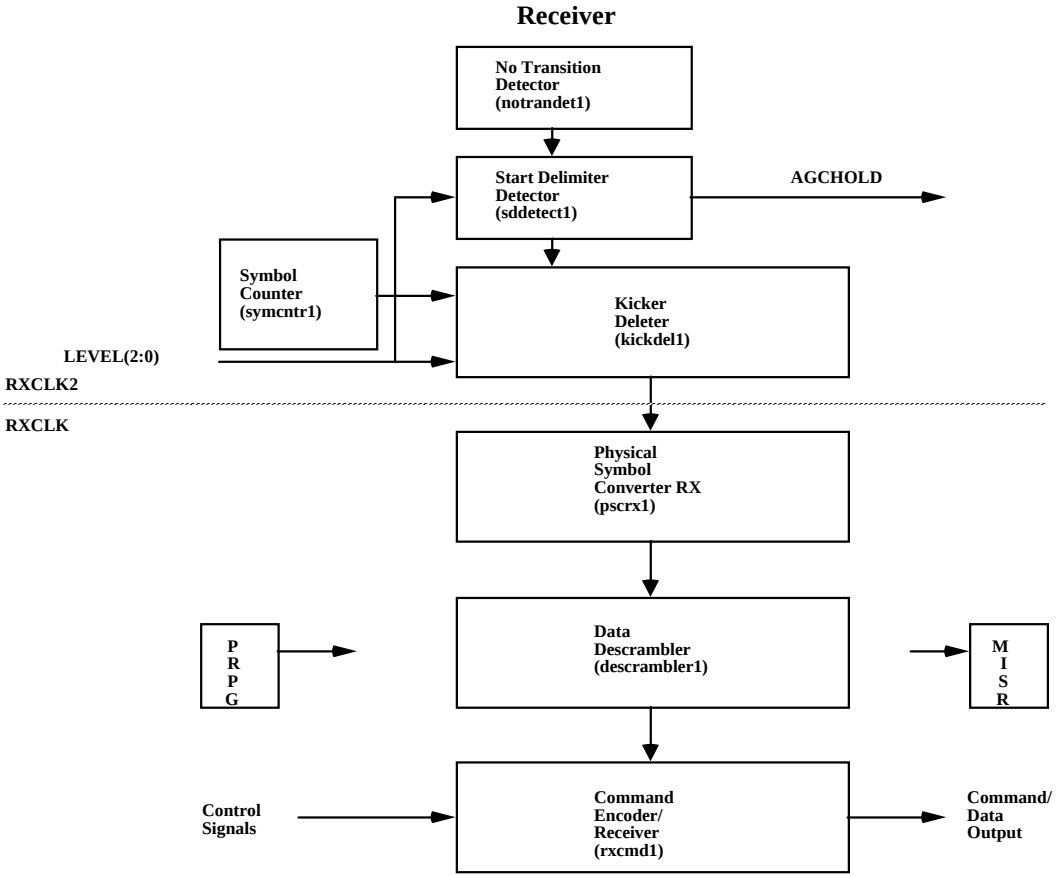


Figure 9. Block Diagram of Receiver

Data coming in on LEVEL(2:0) is monitored until a valid Start Delimiter is detected by the Start Delimiter Detector (sddetect1). It then asserts AGCHOLD for the RF receiver circuitry and enables the control signal SDFOUND which activates the Kickier Deleter (kickdel1) and the Data Descrambler (descrambler1). When SDFOUND is low, the two submodules are disabled as they are used only when data is present on the symbol stream. The No Transition Detector (notrandet1) is used to prevent a “false lock” from occurring where the same symbol is present for more then 48 RXCLKs (or 24 symbols on LEVEL(2:0)). This normally occurs when the automatic gain control in the RF circuitry is held at the wrong level. The Start Delimiter is reset when no transition occurs.

The Kicker Deleter is the counterpart of the Kicker Inserter; it removes any kickers which were inserted into the data stream. The Physical Symbol Converter RX (pscrx1) converts the symbol stream back into the bit stream to be passed to the Data Descrambler. The bit stream then appears on the Command/Data Encoder and is transmitted to the TBC. In command mode, the Command Encoder will ACK or NACK any commands present on the Command Decoder/Transmitter and also indicate errors reported by the error monitor.

4.3.1. Physical DATA Indication Channel (Receiver to TBC)

RXCLK: This signal must be the same frequency as TXCLK. Outputs are synchronized to the positive edge of RXCLK.

$\overline{\text{SMIND}}$: This signal indicates that RIC is in MAC (data transmission) mode when negated or in Station Management (SM) mode when asserted. In MAC mode, RXSYM(2:0) outputs data received. In SM mode, RXSYM(2:0) acts as the response to the commands given to RIC.

RXSYM(2:0): The inputs have the following meaning when in SM mode. ($\overline{\text{SMREQ}}$ = 0) as shown in Table 5. All commands given to RIC will be either ACKed or NACKed depending on whether they are valid commands. RIC will respond with Idle in subsequent clock cycles. Acquire Frequency is the response given when Transmit Enable is initially enabled. It will go away once RIC transmits a control frame (in master mode) or receives a valid control frame (in slave mode).

State	RXSYM2	RXSYM2	RXSYM0
NACK	1	0	0
ACK	0	1	0
Idle Response	0	0	1
Acquire Frequency	0	0	0
Physical Layer Error	1	1	1

Table 5. Receiver Response

When in MAC mode ($\overline{\text{SMREQ}}$ = 1), the data is encoded based on Table 6:

Symbol	RXSYM2	RXSYM2	RXSYM0
Zero	0	0	0
One	0	0	1
Non-Data	1	0	0
Silence	1	1	1
Bad Signal	0	1	1

Table 6. Receiver Output Symbol Encoding Bad Signal is the output when no carrier is present (carrier detect is negated).

4.3.2. RIC Receiver Control Signals (Receiver to RF Stage)

CARRIER DETECT: The Carrier Detect (input) signal is asserted when the RF circuitry detects the presence of a carrier signal in the data channel. AGC Hold is negated whenever CARRIER DETECT is negated. When Carrier Detect is negated and RIC is not in internal loopback mode or SM mode, RXSYM(2:0) is disabled and the Bad Input indication is given to the TBC. In addition, if RIC is not in external loopback $\overline{\text{TRANSMIT DISABLE}}$ is asserted and IMPULSE(2:0) transmits silence.

AGC HOLD: This signal (Automatic Gain Control Hold) (output) is negated whenever CARRIER DETECT is negated. It is asserted after a Start Delimiter on LEVEL(2:0) is detected by the Start Delimiter Detector. AGC HOLD will be negated after one of the following occurs:

(a) 24 inputs with the same values are received on LEVEL(2:0).

(b) Non-datas are detected in the input stream after the Start Delimiter is received (usually signifies the End Delimiter).

(c) If the kicker deleter has been disabled (with the Descrambler Disable bit set) and a kicker (Non-data) is present in the input.

(d) During reset.

LEVEL(2:0): The input to RIC appears on Level(2:0) encoded as 3-bit Physical symbols based on Table 7.

Physical Symbol	Impulse2	Impulse1	Impulse0
Silence	1	1	0
Impulse0 {0}	0	0	0
Impulse1 {1}	0	0	1
Impulse3 {3}	0	1	0
Impulse4 {4}	0	1	1
Impulse2 {2} Non-Data	1	0	0
Error	1	0	1

Table 7. Receiver Input Physical Symbol Encoding

4.4. Registers

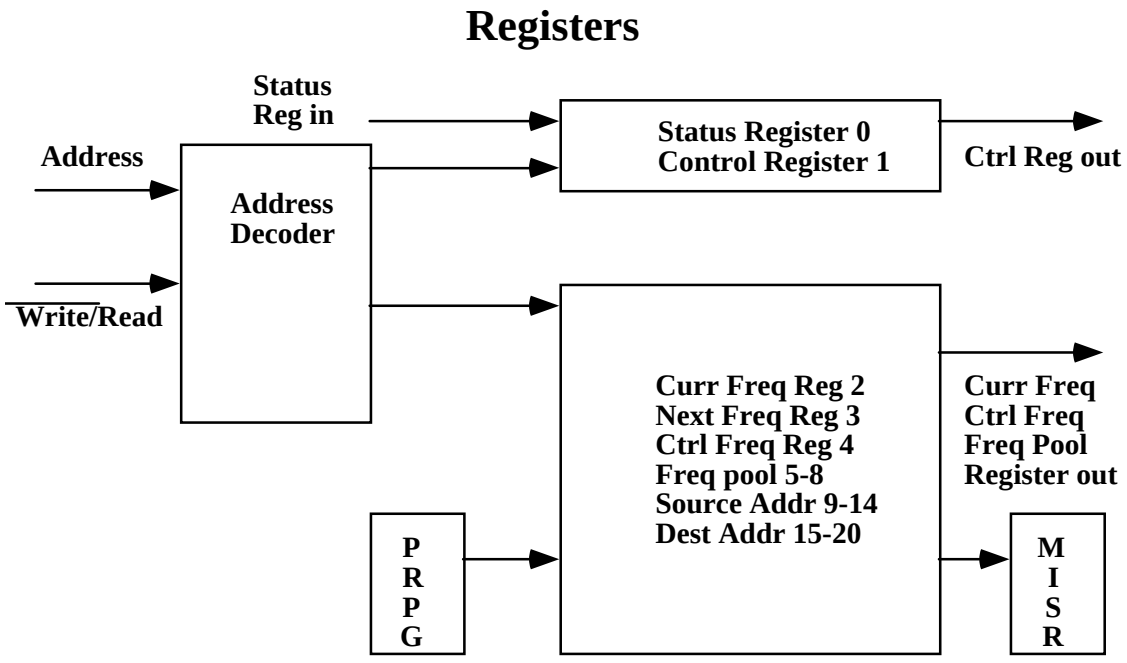


Figure 10. Block Diagram of RIC Registers

The Registers section is operated using $\overline{\text{WRITE}}$ /READ, $\overline{\text{OUTPUT ENABLE}}$, ADDR(4:0), and DATA(7:0). In order to read or write data into the registers, RIC must be in SM mode (TXSYM(2:0) = 000, $\overline{\text{SMREQ}}$ = 0). Register output is available on the bi-directional DATA bus whenever $\overline{\text{WRITE}}$ /READ is high and $\overline{\text{OUTPUT ENABLE}}$ is asserted. To write data into the registers, $\overline{\text{OUTPUT ENABLE}}$ must first be negated and the ADDR and DATA lines stabilized. Then $\overline{\text{WRITE}}$ /READ is held low for one clock cycle in order to write the data into the appropriate register. Note that Registers 0, 2 and 3 are read only registers. Addresses that are out of range (greater than 20) will give invalid results.

4.4.1. RIC Registers Control Signals (external signals)

$\overline{\text{WRITE}}$ /READ: The $\overline{\text{WRITE}}$ /READ line is used to clock in data into the RIC Registers, and is only vaild when the RIC is in SM mode. (TXSYM(2:0) = 000). To write data into the registers, $\overline{\text{OUTPUT ENABLE}}$ is negated first. The address and Datain are first stabilized and then the $\overline{\text{WRITE}}$ /READ line is held low for one clock cycle during which both address and Datain should be stable. $\overline{\text{WRITE}}$ /READ is then brought high again for one clock cycle before the signals are released. Writes to the registers therefore takes three cycles, one to stabilize the data and address lines, one to latch in the value, and another for the value to be held when $\overline{\text{WRITE}}$ /READ is high. Reads from the registers can be accomplished in one clock cycle. Dataout is available as long as $\overline{\text{WRITE}}$ /READ held high and $\overline{\text{SMREQ}}$ and $\overline{\text{OUTPUT ENABLE}}$ are asserted. Dataout is valid one clock cycle after the address stabilizes and continues to be valid for as long as the address is stable.

$\overline{\text{OUTPUT ENABLE}}$: The $\overline{\text{OUTPUT ENABLE}}$ line enables register outputs on DATA(7:0). $\overline{\text{OUTPUT ENABLE}}$ should be negated when data is to be written into the registers.

DATA(7:0): This is the bidirectional data bus which is accessible whenever $\overline{\text{SMREQ}}$ is asserted. It is used to access the RIC Registers.

ADDR(4:0): This is the address control line for accessing the RIC Registers. All Registers are Read/Write except for Register 0,2 and 3 which are read-only. The registers are defined as follows with the address in parenthesis:

Status Register (0): Each bit in the register is a status indicator. This is a read-only register.

7	6	5	4	3	2	1	0
LE	EL	IL	TD	CF	IF	MF	JTO

JTO — Jabber Time Out. This bit is high when a Reset command is received on TXSYM(2:0) during which $\overline{\text{JABBER TIMEOUT}}$ is asserted. It has no meaning before the first Reset command is given.

MF — Modem Failure. This is similar to JTO, except that is indicates that $\overline{\text{FAULT DETECT}}$ is asserted during the last Reset command. It has no meaning before the first Reset command.

IF — Input Failure. This bit indicates that a bad input sequence is given to RIC when in MAC mode. It is the output from the Bad Input Detector. If the Bad Input Detector is disabled, this bit will be low after a reset. It has no meaning before the first Reset command.

CF — Cable Failure. This bit indicates the inverse of the state of the Carrier Detect pin (from the RF module).

TD — Transmit Disable. This bit indicates that the transmitter has been disabled (by a Reset) when high. This bit is set (high) after a reset .

IL — Internal Loopback. This bit indicates that RIC is currently in internal loopback mode when high. This bit is set (high) after a reset.

EL — External Loopback. This bit indicates that RIC is currently in external loopback mode when high. This bit is cleared (low) after a reset.

LE — Loopback Enable. This bit indicates that loopback has been enabled after a reset. If this bit is high and neither Internal Loopback or External Loopback is asserted, then no action will be taken. This bit is set (high)

after a reset . A reset is the only way to return the RIC to a loopback mode after the Loopback Disable command is given on TXSYM(2:0).

Control Register (1): Each bit in the register controls a particular function in RIC.

7	6	5	4	3	2	1	0
MSTR	SYD	BID	DD	SD	LS2	LS1	A48

A48 — 16/48 bit Station Addresses. This bit indicates whether 16 or 48 bit addresses are used for the Address Section of each Token Bus Frame [15]. If it is high, 48 bit addresses are used. This bit is cleared (low) after a reset.

LS1 & LS2 — Loopback Select 1 and Loopback Select 2. These two bits determine which loopback state is in operation when Loopback Enable is asserted. LS1 is set (high) and LS2 is cleared (low) after a reset.

Loopback Select 2	**Loopback Select 1**	**Mode**
0	0	None
0	1	Internal Loopback
1	0	External Loopback
1	1	None

Table 8. Loopback Mode Selection

SD — Scrambler Disable. When high, this bit disables the scrambler encoder and the kicker inserter. SD is cleared (low) after a reset.

DD — Descrambler Disable. The kicker deleter and descrambler are disabled when this bit is high. DD is cleared (low) after a reset.

BID — Bad Input Disable. This bit disables the bad input detector (which detects a single Silence, a single Non-data, and three consecutive Non-datas) when high. This bit is cleared (low) after a reset.

SYD — Synchronizer Disable. This bit disables the Impulse Clock synchronizer when high. It is cleared (low) after a reset.

MSTR — Master mode. This bit is set in order to initialize the Station Management module to Master mode. When low, it acts as a slave and Station Management only receives transmission broadcast on the Control Channel. This bit is cleared (low) after a reset. Once the Transmit Enable command is given, only a Reset command will return Station Management to an Idle state for reprogramming into another state.

Current Frequency Register (2): This register indicates the current data channel frequency. It controls the frequency synthesizer for the RF stage. It is a read-only register.

Next Frequency Register (3): This register indicates the next data channel frequency. It is a read-only register.

Control Frequency Register (4): This register indicates the control channel frequency for the use of the Station Management module. It has to be set to a valid value before the Transmit Enable command is given in order for the Station Management module to transmit on a valid frequency.

Frequency Pool (5-8): These four registers store the frequencies that the data channel will hop through as data transmission and reception progresses. It is accessed in a circular fashion (hence cyclic frequency hopping). Any choice of one, two or four frequencies can be specified. If only one frequency is needed. All four registers are set to the same value. For two frequencies, registers 5 and 7 are set to one frequency, and registers 6 and 8 are set to the other.

Source Address (9-14): The six bytes of the Source Address are arranged with the low order byte first (9) and high order byte last (14). If 16 bit addresses were used, only addresses 9 and 10 will be referenced.

Destination Address (15-20): The six bytes of the Destination Address are arranged with the low order byte first (9) and high order byte last (14). If 16 bit addresses were used, only addresses 9 and 10 will be referenced.

4.5. Station Management

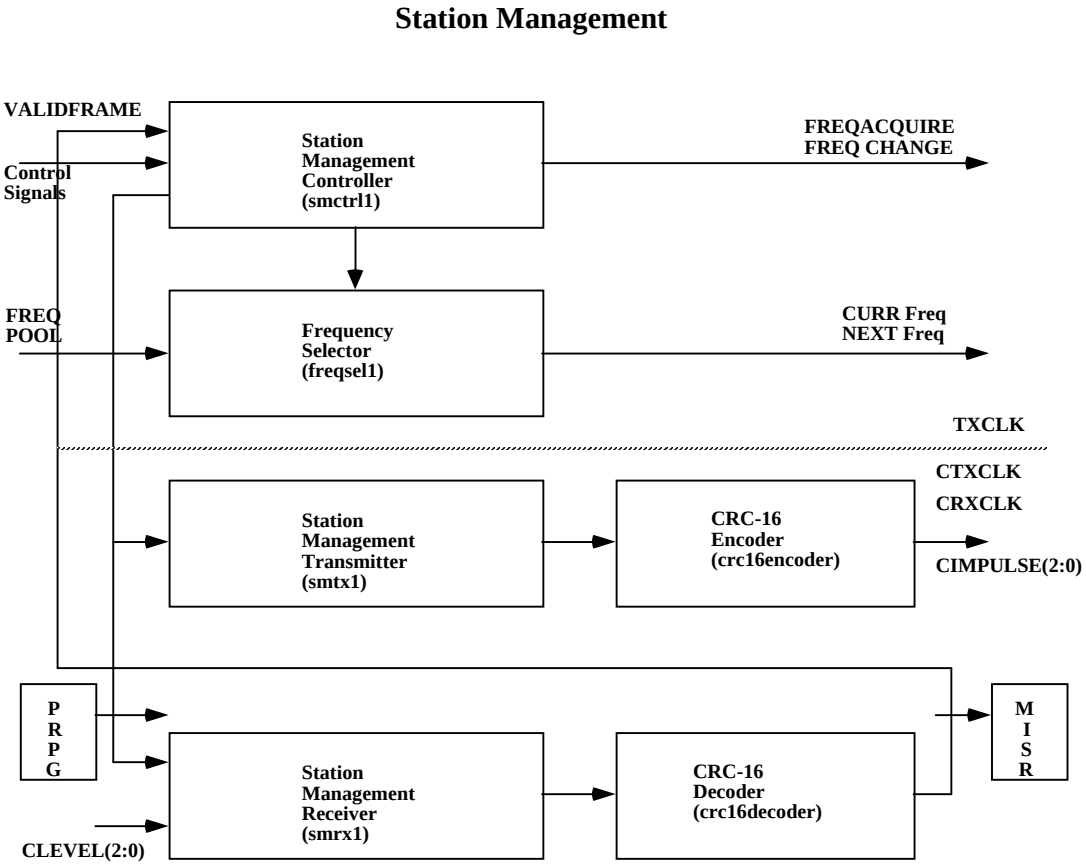


Figure 11. Block Diagram of Station Management module

The Station Management section of the RIC is initialized depending on whether the RIC is operating in Master or Slave mode (bit 7 of Register 1) (Master mode is used for the stationary Local Controller, while slave mode is used for the mobile robots). Source and Destination Addresses also need to be stored into the appropriate registers in order for the station management modules to function properly. The addresses are used to verify that the received frame is intended for the particular subnetwork. When RIC is used in Master mode, the values for the various frequencies of the data channels must first be loaded into the transmission channel frequency table (Registers 5-8) before anything else is done. Once the Transmit Enable command is given via TXSYM(2:0) and RIC is not in Internal Loopback mode, the station management module will be enabled and undergo an initialization sequence before entering MAC (data transmission) mode. In external loopback mode, the station management module will also be enabled, but RIC will be placed in MAC mode immediately without having to wait for the Station

Management module to go through its initialization sequence. In normal operation, MAC mode will not be entered until the initialization sequence completes.

The initialization sequence for master mode involves the transmission of a control frame. Otherwise, a *valid* control frame must be received by the station management receiver before $\overline{\text{SMIND}}$ is negated and RIC enters MAC mode. The Station management section is run asynchronously with respect to the Receiver and Transmitter section of RIC. The Station Management Controller (smctrl1) initiates the Station Management Transmitter (smtx1) and Receiver (smrx1) based on the mode setting and allows them to run independently of the Transmitter and Receiver sections of RIC. Data Channel frequency switching is done when both the Transmitter and Receiver are silent. (Silence is present on both TXSYM and RXSYM). In addition, frequency switching is done by the Frequency Selector submodule (freqsel1) in cyclical fashion based on the four frequency values stored in Registers 5-8. This control frequency is made known to each mobile robot beforehand so that they would know which frequency to tune in to for control information from each Local Controller. Once initialized, the station management transmitter will broadcast the present Data Channel frequency and the next frequency continuously in a station management frame. The two CRC-16 Encoders and Decoder Pairs are used to generate the Frame Check Sequence and verify the correctness of received frames [17],[19]. Since the FCS operates only on data symbols, bit 2 of CIMPULSE(2:0) is zero and an error occurs if the bit is set for any data symbol in the frame. Frame transmission is aborted if a frequency switch is performed in the middle of a station management frame transmission what would invalidate the information already transmitted. Similarly, if the station management receiver determined that the current frequency for the Data channel is not valid, it will force a frequency change, interrupting any data transmission in progress. Once the Station Management module is activated, the mode cannot be changed without first resetting RIC via the RESET command on TXSYM(2:0).

4.5.1. RIC Station Management Control Signals (Station Management to RF stage)

CTRLTXCLK, CTRLRXCLK: These are the clock signals used by the Station Management Control Frame transmitter and receiver circuitry. It should be the same clock speed as IMPULSECLK, although it can operate at the same clock rate as TXCLK and RXCLK.

$\overline{\text{CTXDISABLE}}$: This has the same function as $\overline{\text{TRANSMIT DISABLE}}$ in the Transmitter section. It is asserted when no data is transmitted on CIMPULSE(2:0).

CIMPULSE(2:0): The output from the Station Management transmitter. This signal goes high when $\overline{\text{CTXDISABLE}}$ is asserted. It uses the same encoding as IMPULSE(2:0).

CLEVEL(2:0): This is the input to the Station Management receiver. It uses the same signal encoding as LEVEL(2:0).

CCARDET: This is the control channel carrier detect line. If this line is not asserted when a start delimiter is detected by the receiver circuitry, the frame will be ignored.

CAGCHOLD: The AGC hold line for the control channel is asserted if CCARDET is high when the start delimiter is detected. It will go low again when the frame is complete (end delimiter is received), or if an error occurs.

CURRFREQ(7:0): This is the Data Channel Frequency Control signal. The external RF circuitry takes the 8 bit value and selects the appropriate carrier frequency for transmission.

CTRLFREQ(7:0): The Control Channel Transceiver uses the output from CTRLFREQ(7:0) to set the appropriate carrier frequency for transmission.

4.6. BIST Controller

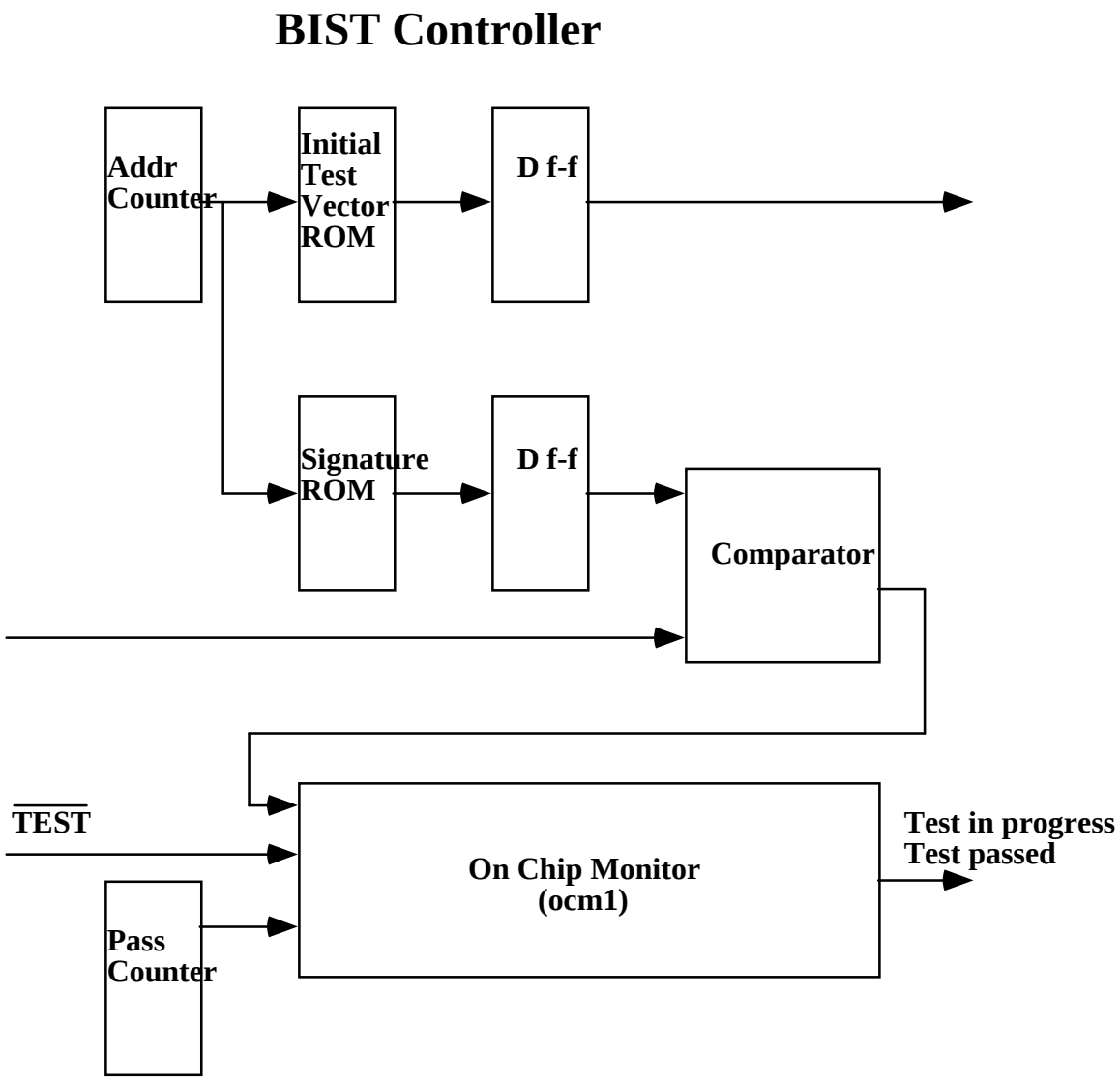


Figure 12. Block Diagram of BIST Controller

Upon assertion of $\overline{\text{TEST}}$, the On Chip Monitor (ocm1) will load the initial test vector into the PRPG for each module. The initial test vector loaded into the PRPG is 10101010101010101010101010101010. Once that is completed, the OCM will then enable the scan paths for all the submodules in each module and scan the test vector generated by the PRPG into the various scan paths. Each submodule takes the first n lines of the PRPG output as its test vector input, where n is the bit width of the submodule. The scan path is preloaded with the test vector for VECTORLEN clock cycles. The system is switched back to normal operation mode for one cycle, and the output from each submodule is clocked into the flip-flops, and then scanned out into the MISR while the

next set of test vectors are scanned in. Once all the test vector sets have been executed, the resultant signature scanned out serially and compared to the signature stored in ROM to determine if the tests were passed. The output from the On Chip Monitor, TESTPASSED will indicate the success or failure of the tests.

4.6.1. RIC BIST Controller Control Signals (external signals)

$\overline{\text{TEST}}$: The $\overline{\text{TEST}}$ line is asserted in order to put the RIC into a self test mode. Once initiated, it will require a $\overline{\text{RESET}}$ signal to return RIC to normal operations. The signal should be asserted for four clock cycles until TESTINPROGRESS is asserted. If it is still present at the end of the test cycle, the TESTPASSED signal will not be asserted. Instead, the TESTINPROGRESS signal will go low once the signature is ready to be read out through the internal SIGNATURE(8:0) net. This mode is used to collect the signature that is used by the signature ROM. This signal is not used in normal operations, but is provided for diagnostics purposes.

TESTINPROGRESS: This signal is asserted when internal self-tests are being executed. (After $\overline{\text{TEST}}$ is asserted).It will be negated once the tests are complete.

TESTPASSED: This signal indicates if the RIC passed the self-tests. It is valid on the negative edge of TESTINPROGRESS. A logic 1 on this line indicates that the RIC passed the self-tests. A logic 0 indicates failure. If $\overline{\text{TEST}}$ remains asserted at the end of the testing cycle, then TESTPASSED will not be asserted. Instead, the signature vector will be scanned out from the MISR for verification.

TESTCLK: This is the clock used for testing RIC. It can be run at a higher speed than the regular system clock. In this way, testing time will be reduced.

4.7. Brief RIC Submodule Descriptions

4.7.1. Bad Input Detect

This module detects invalid input sequences on TXSYM(2:0). These sequences include a single Non-data, three consecutive Non-datas and a single Silence symbol. Invalid sequences cause the bad input detector to generate an error signal that is fed to the Error Monitor.

4.7.2. Error Monitor

This module monitors the Fault Detect, Jabber Timeout and Bad Input Detect lines for any errors that may occur. Whenever any one of the three errors occur, the Error Monitor will flag an error and remain in the error state until the next reset. The error signal is reported by the Command Encoder to the TBC.

4.7.3. Frequency Select

The frequency select module is used by the station management module to select the two consecutive frequency channels given by the four registers in the frequency pool. It starts with frequency 0 and frequency 1, then switches to frequency 1 and frequency 2 when **advance** is asserted, and so on. The module is enabled by **enablefreqsel**, otherwise it will just select the first two frequencies.

4.7.4. Kicker Deleter

The kicker deleter looks at the input from level(2:0) and removes any kickers that are present. Kickers in the input stream are ignored when the ignorekicker line (descrambler disable bit) is asserted. Otherwise, the kicker deleter is enabled when enablekicker is asserted using sdfound.

4.7.5. Kicker Inserter

The kicker inserter accepts input from the Physical Symbol Converter TX and inserts kickers where appropriate. It is synchronized to the octet boundaries using the firstsym line from Symcounter module. No kickers are inserted when nokicker is asserted. The kicker inserter is disabled when enablekicker is negated.

4.7.6. No Transition Detector

The No Transition Detector checks level(2:0) to see if there are 22 identical inputs. If that input sequence is detected, then it will assert notrans, which reset the AGCHOLD signal. This prevents the RF section from holding the signal at the wrong level which can cause the no transition condition.

4.7.7. On Chip Monitor

The On Chip Monitor automates the testing process by setting up the initial test vector for the PRPGs and scans it into the scan path of each submodule, as well as latching the output into the MISR to generate the signature. TESTINPROGRESS is asserted while the tests are being executed. The length of the initial test vector loaded into the PRPG is given by VECTORLEN. The number of cycles required to load the test vector into the scan path is given by TESTLEN. The PRPG parallel loads the test vector into the combinational logic while scanning the test vector into the scan path. This means that the valid test vector is only present at the last iteration given by TESTLEN. The tests are run for NOPASS number of passes. After the required number of passes were executed, the On Chip Monitor will clock the test vector out into the MISR and generate the final signature. The final signature is then compared to the value stored in ROM and the TESTPASSED signal is asserted if the values match, otherwise it will be negated. IF the input signal TEST is asserted till the end of the test cycle, the TESTPASSED signal will not be asserted. Instead, the signature will simply be clocked out in order to collect the actual signature (for storage in ROM).

4.7.8. Physical Symbol Converter TX

The Physical Symbol Converter state machine (psctx1) is the submodule which handles the bit stream to symbol stream (MAC to PHY) conversion. The transmitter block operates at the TXCLK frequency, and the output changes every two clock cycles to synchronize it with the Impulse CLK. Conversely, the receiver block takes PHY symbols and generates two bits per symbol at the RXCLK frequency (which is equal to TXCLK). The State Transition Graphs are given in Figures 13, 14, and 15.

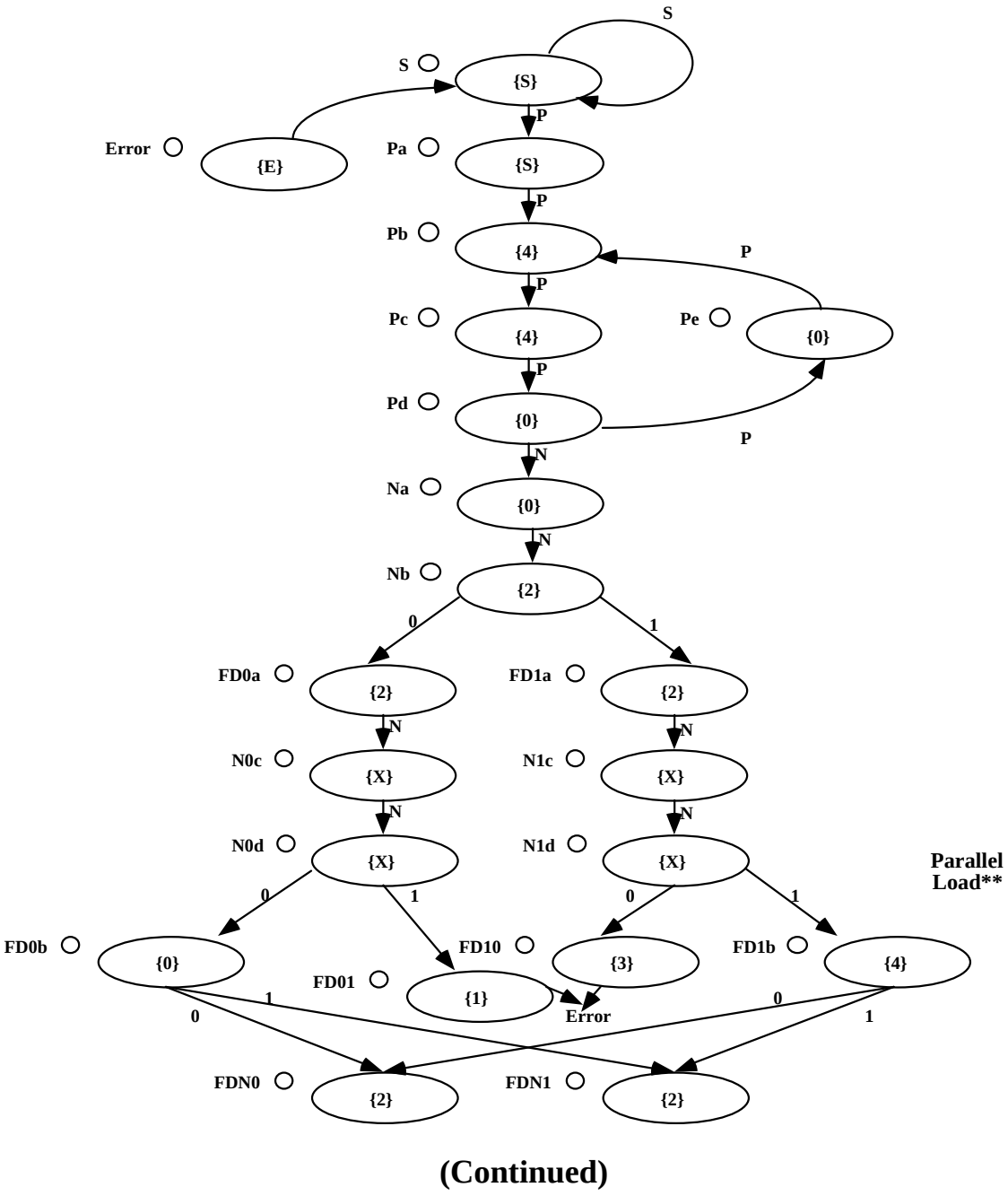


Figure 13. State Diagram of PSC Transmitter

(Continued)

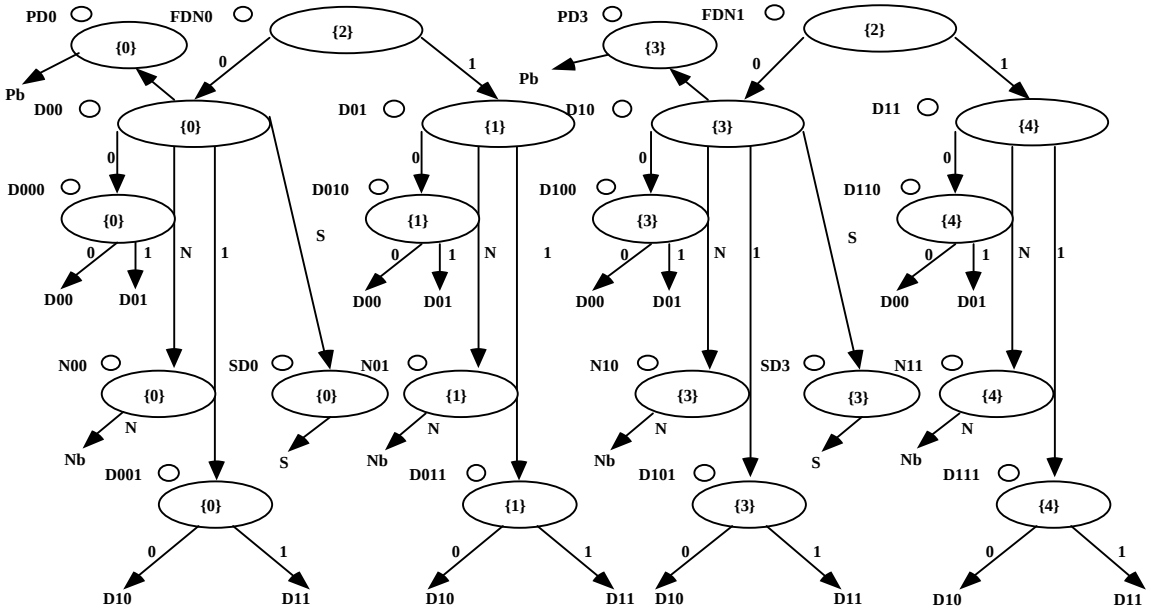


Figure 14. State Machine of PSC Transmitter (continued)

4.7.8.1. PSCTX Block: Reordering the Delimiter Byte Output Symbols

Since the order of the transmitted bits at the output stream is switched for Delimiter bytes, the output from psctx1 is fed through a four bit Shift Register Chain. When the state machine reaches one of the boxed states (Figure 13), new values will be shifted into the flip-flops. This ensures that the symbols are in order (Figure 15). In order for synchronization to be preserved for the silencemode indicator and the enable kicker signals, each signal is fed through a three bit shift register before being passed on to the output.

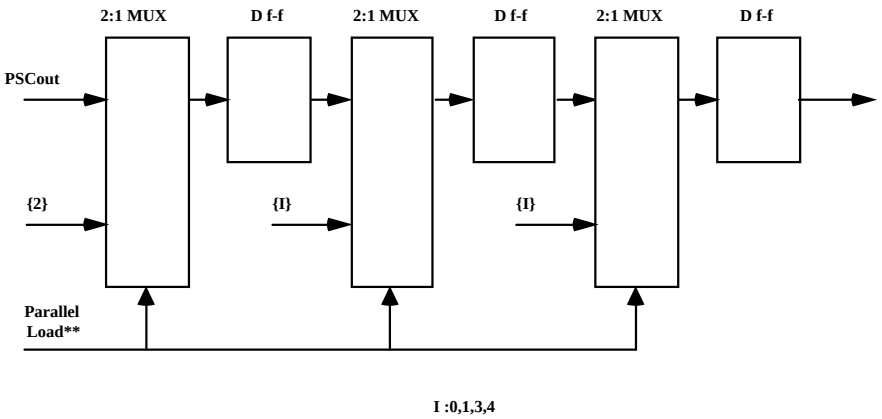


Figure 15. Block Diagram of Shift Register Chain

4.7.9. Physical Symbol Converter RX

Figure 16 diagrams the state transitions for the Physical Symbol Converter RX (pscrx1), which takes the PHY input stream and convert it back into the MAC data stream.

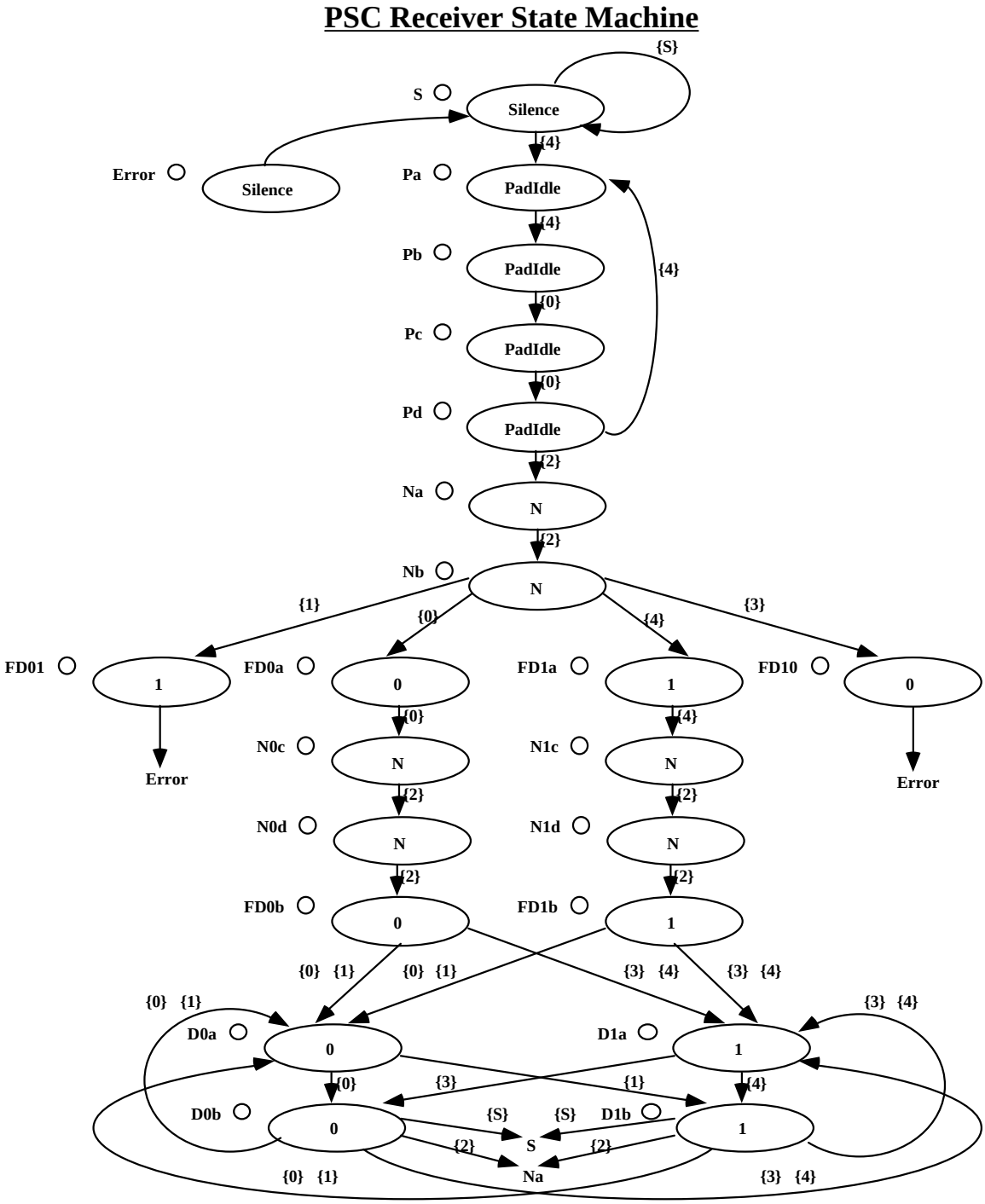


Figure 16. State Diagram of PSC Receiver

4.7.10. Command/Data Encoder (Receiver)

Figures 17 and 18 indicate the control signals and the state diagram for the Command Encoder/Receiver (Rxcmd1). There are only two modes, Station Management

mode and Receive mode. SMindmode is entered whenever Station Management mode is activated or if RIC is acquiring the Data Channel frequencies, or if an error is present in the Physical Error line. Otherwise, it is in the Rxenablemode, waiting for data input. Figure 18 gives a brief diagram of the two modes, and the state $\overline{\text{SMIND}}$ is in for each of them.

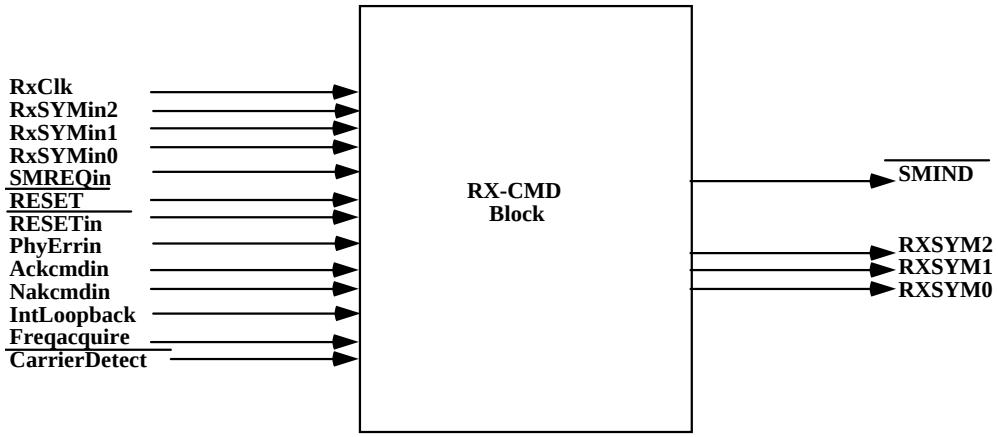


Figure 17. Block Diagram of Command/Data Encoder

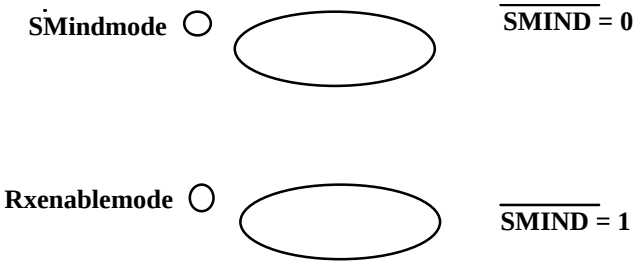


Figure 18. State Diagram of Command/Data Encoder

4.7.11. Command/Data Decoder (Transmitter)

The Command Decoder/Transmitter (txcmd1) submodule takes inputs from the TBC and handles commands and data appropriately. Figure 19 lists the control signals handled by the txcmd1 module as well as the generation of the $\overline{\text{TRANSMIT DISABLE}}$ signal by Txsyncblock. The state diagram for txcmd1 is given in Figure 20. No state transition information is given for the states (except for HResetmode) because all the states are interconnected and any transition is possible given the command on TXSYM(2:0). An ACK is generated every time a state transition occurs, after which Idle is generated for as long as the same command is present. When in MAC mode, data from the TBC is passed onto the scrambler stage after a one clock delay. Figure 19 also

includes the logic diagram for generating the Transmit Disable signal. This signal is generated within the Txsyncblock submodule.

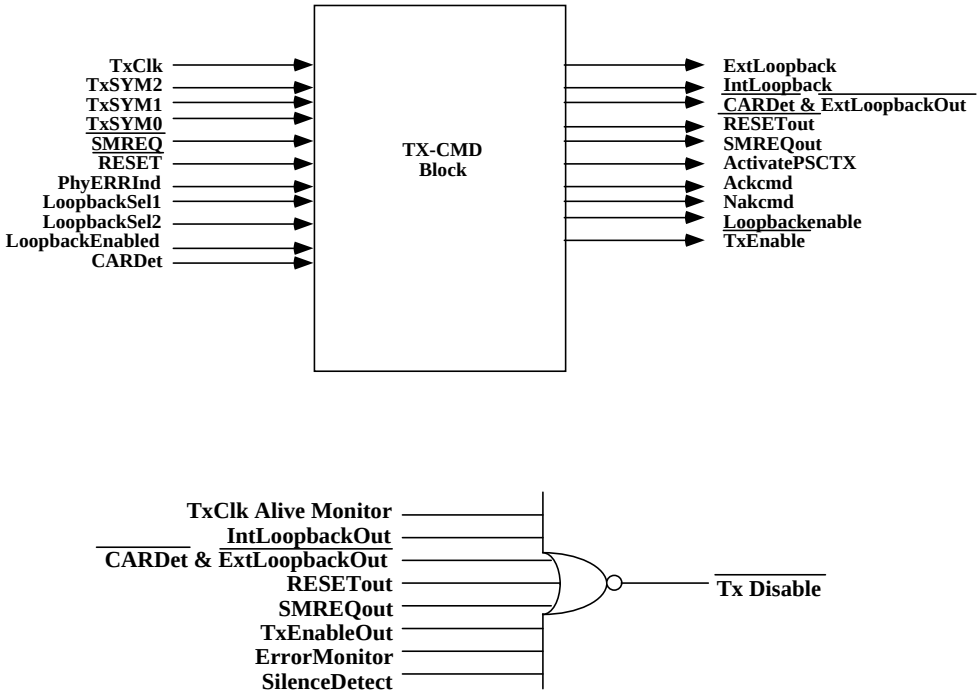


Figure 19. Block Diagram of Command/Data Decoder and Transmit Disable signal

The State diagram indicates the actions taken on the Status and Control registers when RIC is in a particular state. Reset will cause both the Status (R0) and Control (R1) registers to be set to Internal Loopback mode with Transmit Disabled. This was described in the Registers section.

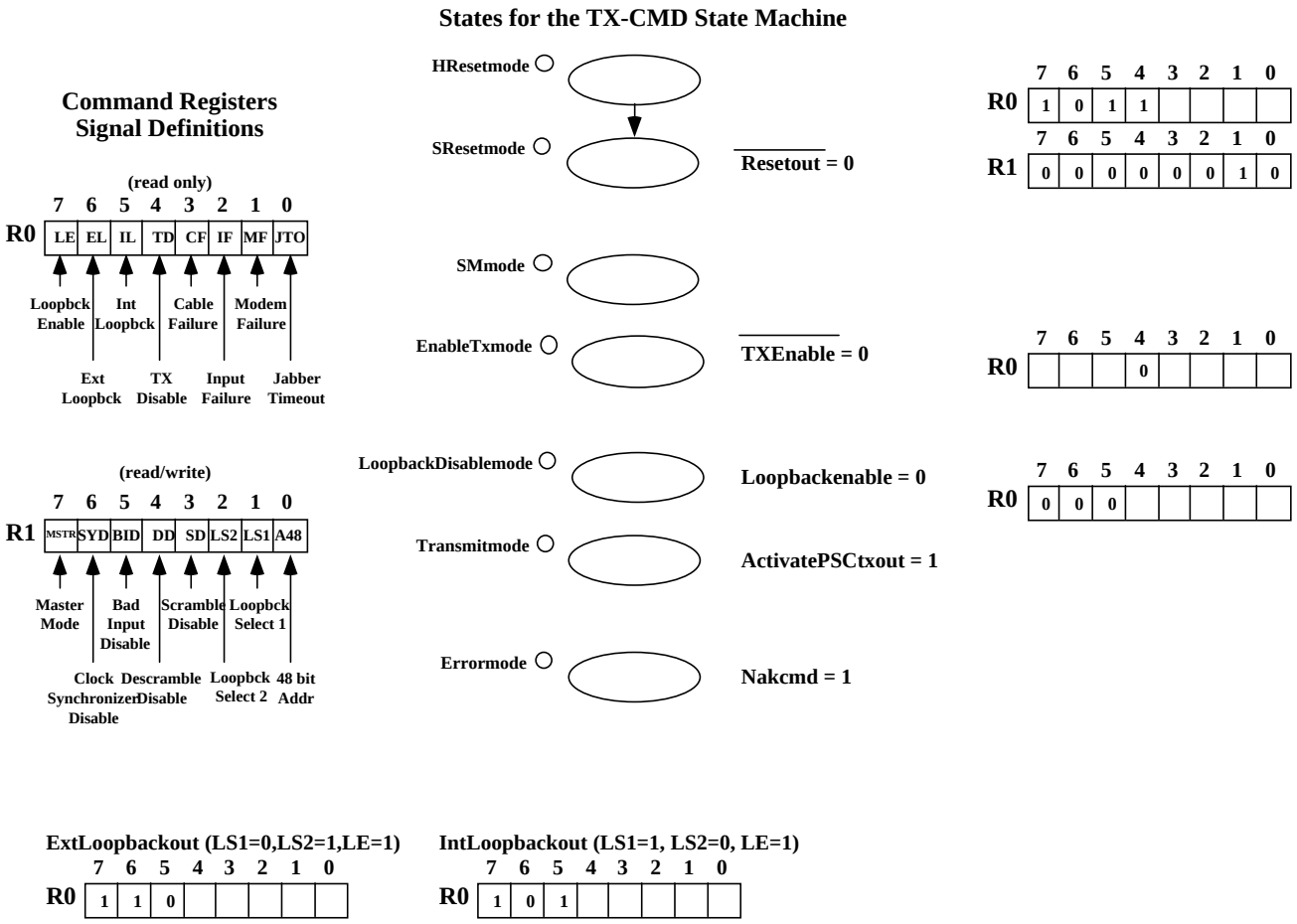


Figure 20. State Diagram of Command/Data Decoder

4.7.12. Start Delimiter Detector

The Start Delimiter Detector monitors LEVEL(2:0) and asserts SDFOUND if a start delimiter is detected. SDFOUND is negated when no transition occurs on LEVEL(2:0) (indicated by No Transition Detector), a Non-data is present in the descrambler output (from the End Delimiter), if CARRIER DETECT goes low when not in internal loopback, and during reset. SDFOUND is asserted when Carrier Detect is active and a Start Delimiter is detected, and negated when an End Delimiter is detected in normal operation. However in internal loopback, Carrier Detect is ignored.

4.7.13. Station Management Controller

The Station Management Controller coordinates the functions of the Station management Transmitter and Receiver based on the Master Mode bit in the Control Register. If RIC is not in Internal Loopback, it will assert freqacquire when RIC tries to enter MAC mode, and will not enable MAC mode until a valid Control Frame is either transmitted or received by the Station Management Module. It also coordinates the frequency switching for the data channel by waiting for silence on both the transmitter and receiver before setting the new data channel frequency. If Silence occurs in both the transmitter and receiver during the transmission of a Control Frame, it will be aborted if part of the frequency information has already been broadcast. Similarly, if the Station Management Receiver receives a valid Control Frame which indicates a Current Frequency different from that stored in the Registers, the data channel frequency will be updated immediately.

4.7.14. Station Management Transmitter

The station management transmitter transmits control frames from the Local Controller operating in Master mode. The control frames have the following format:

- One symbol of Silence (minimum).
- An octet of Preamble
- The Start Delimiter
- The Frame Control indicating that it is a station management frame.
- The Destination Address transmitted least significant dibit (two bit pair) first.
- The Source Address transmitted least significant dibit first.
- The Data Unit, which consists of the Current Frequency and Next Frequency transmitted least significant dibit first.
- The 16 bit FCS (CRC-16) transmitted most significant bit first. The FCS is formed by passing CIMPULSE(0) and CIMPULSE(1) individually through a CRC encoding circuit. The FCS is transmitted as a combinational symbol with CIMPULSE(2) being zero and CIMPULSE(1) and CIMPULSE(2) coming from their individual encoders.
- The End Delimiter, which has the pattern *NN1NN000* or *{2} {1} {2} {0}.*

At the end of the frame, the SM Transmitter asserts the **txendofframe** signal to indicate that the data channel frequency initialization has completed.

4.7.15. Station Management Receiver

The station management receiver on the AGCs operating in slave mode receives control frames from the Local Controller. The frames are assumed to be in error if CLEVEL(2) is non-zero for any bit in the FCS. The **validframe** signal is asserted upon the receipt of a valid station management frame.

4.7.16. Symbol Counter

This submodule is a 3 bit counter (modulo 8 counter) with a firstsym output that indicates the first symbol in an octet. The Kicker Inserter and Deleter both use the firstsym signal to determine if a symbol is the first symbol in an octet.

4.7.17. Transmit Clock Alive Detector

The TXClk Alive Monitor monitors TXClk2 and disables IMPULSE(2:0) if there is no transition for twenty four clocks. This is to prevent a faulty connection between the TBC and the RF section from causing the transmitter to be stuck in the transmit mode indefinitely. It is powered using RXCLK2 and uses Symcounter to register any transitions. TXClk2 is the clock signal for Symcounter. If TXClk2 fails, the counter will stop counting and the TXClk Alive Monitor will detect the error.

4.7.18. Transmit Disable Silence Detector

This module is used by the Txsyncblock to determine when to assert the Transmit Disable signal. It is negated when Silence is detected on the inputs to the RIC, and asserted10 TXCLK cycles after Silence appears at the output after a transmission.

4.7.19. Scrambler and Descrambler Circuits

Figure 21 gives the basic circuitry used for the scrambler and descrambler circuitry. Whenever a Non-data symbol is encountered, all the flip-flops are preset to one. This initializes the scrambler and descrambler to handle the data portion of the input bit stream. Both the scrambler and descrambler have disable lines for control which are not shown in the figure. In addition, the descrambler has a **sdfound** line which enables the descrambler when a start delimiter is detected.

MAC Data Scrambler and Descrambler Circuitry

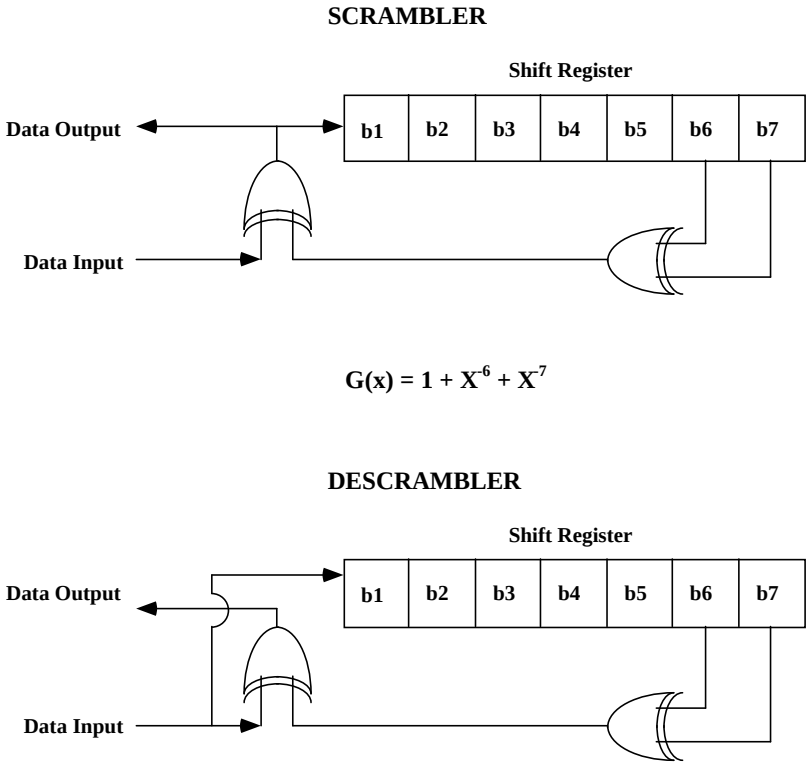


Figure 21. Block Diagram of Scrambler and Descrambler circuits

5. Logic Simulation using QUICKSIM

The following simulation runs were performed on the design:

Rsetup.sim:

The RIC Registers were initialized and the values stored in the registers read back via DATA(7:0).

REsetup.sim:

Various error conditions during initialization and the responses from RIC.

Rirun.sim:

RIC registers were initialized and executed in internal loopback mode. The output comes from RXSYM(2:0)

Rxrun.sim:

RIC registers were initialized and executed in external loopback mode.

Rorun.sim:

Various options for controlling the Scrambler and Descrambler were deactivated to indicate their function in the transmission process.

Rmrun.sim:

RIC is initialized in master mode and a packet was transmitted and received.

Rsrun.sim:

RIC is initialized in slave mode and a packet was transmitted and received.

Rerun.sim:

RIC is initialized in slave mode and various errors are simulated for the transmission and reception of packets.

Rtrun.sim:

RIC is reset and placed into test mode. If the test input is low at the end of the tests, then the signature will be shifted out for observation. Otherwise, the TESTPASSED flag will indicate whether RIC passed the self-tests.

By means of the above simulations, the logical function of RIC could be verified. Various problems were encountered during the process of integrating the submodules. Without the help of a logic simulator, some of these problems would not be detected, since they deal with buffer sizing and timing constraints.

6. RIC Design Tradeoffs and Conclusions

Most of the modules in RIC were defined as FINESSE modules as they consist of state machines which can be specified clearly and concisely while being easily modifiable. The outputs from each state machine are latched to minimize the effects of spikes and hazards in the outputs. However, the use of D flip-flops to latch the output signals introduces delays into the output stream, and this delay adds up to about ten clock cycles in the Transmitter module. In addition, the use of D flip-flops for the Registers means that the load on the data bus as well as the clock distribution circuitry is increased tremendously. As a result, all signals going to and from the registers have to be buffered. This introduces delays into the design and lowers the maximum achievable clock speed. The advantage of using D flip-flops is that the individual registers can be accessed

simultaneously. This is necessary for implementing the status, control and frequency registers.

Testability is implemented with PRPGs and MISRs using Random Test Vector Generation. All inputs to each submodule are multiplexed between normal inputs and test vector inputs to allow for simultaneous testing of all submodules. The BIST circuitry consumes considerable area overhead, but the percentage of fault coverage is unknown at present due to a lack of suitable tools to do fault detection evaluation. However, the ability to do automatic testing will provide for increased reliability if the fault coverage can be accurately determined to be satisfactory.

The use of ASIC design methods greatly simplified the transition from state machine definitions to actual physical layout of the device. Automated circuit synthesis tools (FINESSE) allow for the abstraction of required function from the details of implementation at the expense of computer time. However, the present state of the art still does not provide for complete automation of the design process. The computer tools were not able to perform thorough consistency checking at each step of the design and compilation process. Some errors which were not trapped by the schematic capture tools caused the compilation tools to abort with errors.

Nevertheless, ASIC methodology is definitely an improvement over full-custom logic design at the transistor level. Designs which would take years to finish can now be completed in months. The approach for designing a specific circuit is to first define the top level blocks which perform the required functions. Once that is done, the individual modules which makes up the high level blocks can be designed and compiled into macros or “Superblocks” which is then put together into the final design. The simultaneous top-down and bottom-up approach allows for quick prototyping and changes can be incorporated quickly as the need arises. Once the bottom level modules are defined, they are integrated together in a hierarchical manner to arrive at the top level blocks. The process of integration is currently awkward and the tools available are rather primitive in that no thorough consistency checks are performed at the initial stages of the integration process. Errors would surface, due to misnamed nets and invalid pin connections which would abort the silicon compilation.

The compilation process is a tradeoff between required compilation time and efficiency of silicon layout. In order to achieve quick prototyping time, compilation of the individual lower level modules has to be done first. A flag is then set to designate them as

primitive components so that compilation of the top level design does not re-expand those modules. This will save compilation time at the expense of much greater silicon area.

It can be said that the use of ASIC methodology for chip design has improved the turn-around time involved. The ability to do quick prototyping and simulation of the chip also improves the odds that the fabricated design will function according to specifications.

References

[1] T. Rappaport, "Indoor Radio Communications for Factories of the Future," *IEEE Communications Magazine,* Vol. 27, No. 5, May 1989.

[2] L. Hill, "Improvements in Inductive Communications for Wire-guided AGVs," *Proceedings of the 7th International Conference on Automated Guided Vehicle Systems,* June 1989.

[3] A. Lessard, and M. Gerla, "Wireless Communications in the Automated Factory Environment," *IEEE Network,* Vol. 2, No. 3, May 1988.

[4] A. Lessard, "Wireless Communications for an Automated Factory Environment," Ph.D. Dissertation, University of California Los Angeles, 1988.

[5] D. Cox, "Portable Digital Radio Communications — An Approach to Tetherless Access," *IEEE Communications Magazine,* Vol. 27, No. 7, July 1989.

[6] S. K. Gibson, Cellular Mobile Telephones, Prentice-Hall, Inc., New Jersey, 1987.

[7] G. Calhoun, Digital Cellular Radio, Artech House, Massachusetts, 1988.

[8] M. Kavehrad, and P. McLane, "Spread Spectrum for Indoor Digital Radio," *IEEE Communications Magazine,* Vol. 25, No. 6, June 1987.

[9] D. Newman, Jr., "FCC Authorizes Spread Spectrum," *IEEE Communications Magazine,* Vol. 24, No. 7, July 1986.

[10] K. Pahlavan, "Wireless Intraoffice Networks," *ACM Transactions on Office Information Systems,* Vol.6 No. 3, July 1988.

[11] T. Rappaport, and C. McGillem, "UHF Fading in Factories," *IEEE Journal On Selected Areas In Communications,* Vol. 7, No. 1, January 1989.

[12] A. Saleh, and R. Valenzuela, "A Statistical Model for Indoor Multipath Propagation," *IEEE Journal on Selected Areas in Communcations,* Vol. SAC-5, No. 2, February 1987.

[13] A. Tanenbaum, Computer Networks, 2nd. Ed., Prentice-Hall, Inc. New Jersey, 1987.

[14] A. Saleh, and L. Cimini, Jr., "Indoor Radio Communications Using Time-Division Multiple Access with Cyclical Slow Frequency Hopping and Coding," *IEEE Journal On Selected Areas In Communications,* Vol. 7, No. 1, January 1989.

[15] IEEE Token Passing Bus Standard 802.4—1985, Institute of Electrical and Electronics Engineers, 1985.

[16] K. Pahlavan, and M. Chase, “Spread Spectrum Multiple-Access Performance of Orthogonal Codes for Indoor Radio Communications,” *IEEE Transactions on Communcations*, Vol. 38, No. 5, May 1990.

[17] B. Arazi, A Commonsense Approach to the Theory of Error Correcting Codes, MIT Press, Mass., 1988.

[18] MC 68000 Family Reference, Motorola Inc., 1988

[19] F. Kuo, Ed, Protocols and Techniques for Data Communications Networks, Prentice Hall, Inc., New Jersey, 1981. Chp. 7, p. 261-268

Appendix A: Finesse Listings and QUICKSIM Simulation Files

A.1 Badinput Detect

```
#define Nondata ((badinputdetectenable) & (smreq == 0) & (txsym == ^b10-))
#define Padidle ((badinputdetectenable) & (smreq == 0) & (txsym == ^b01-))
#define Silence ((badinputdetectenable) & (smreq == 0) & (txsym == ^b11-))
#define Data0   ((badinputdetectenable) & (smreq == 0) & (txsym == ^b000))
#define Data1   ((badinputdetectenable) & (smreq == 0) & (txsym == ^b001))

FINESSE badipdet1

INPUT ~resetbadipdet, ~smreq, ~badinputdetectenable, txsym[2:0];
OUTPUT DFF badinput;
INTERNAL DFF state;

SYMBOLIC state { Idle, Nondata1, Nondata2, Silence1, Silence2, Error };

FSM
{

DEFAULT
  { state := Idle;
    badinput := 0;
  }

WHEN resetbadipdet RESET
  {
    state = Idle;
    badinput = 0;
  }

Idle: { IF Nondata
       /* Non data symbol detected */
          { state := Nondata1; }
      ELSE IF Silence
        { state := Silence1; }
        ELSE
            LOOP;
      }

Nondata1: { IF Nondata
            { state := Nondata2; }
          ELSE
            { badinput := 1;
            state := Error; }
        }

Nondata2: { IF Nondata
            { badinput := 1;
            state := Error; }
          ELSE IF Silence
            { state := Silence1; }
          ELSE
            { state := Idle; }
        }

Silence1: { IF Silence
            { state := Silence2; }
          ELSE
            { badinput := 1;
            state := Error; }
        }

Silence2: { IF Nondata
       /* Non data symbol detected */
          { state := Nondata1; }
      ELSE IF Silence
            LOOP;
        ELSE
        { state := Idle; }
      }

Error: { badinput := 1;
       LOOP; }
}
```

```
                                      FINESSE
                                   Status Report

File Name  : badipdet.fin
Module Name: badipdet1

/*** Parsing/Synthesis messages     ***/
(WARNING)   No exit out of state Error
/*** End Parsing/Synthesis messages ***/

/*** Optimization messages    ***/
/*** End Optimization messages      ***/

/*** Structure Generation messages        ***/

      /*** Testability Information  ***/

      INSTANCE    :  g0#badipdet1
      SCANIN NET  :  f#scanin
      OUTPUT NET  :  state(1)

      INSTANCE    :  g1#badipdet1
      SCANIN NET  :  state(1)
      OUTPUT NET  :  state(2)

      INSTANCE    :  g2#badipdet1
      SCANIN NET  :  state(2)
      OUTPUT NET  :  state(3)

      INSTANCE    :  g3#badipdet1
      SCANIN NET  :  state(3)
      OUTPUT NET  :  badinput

      /*** End Testability Information    ***/

      PORT txsym(0) IS NOT REQUIRED.
      OUTPUT f#scanout MERGED WITH badinput.
/*** End Structure Generation messages    ***/

/*** Component Summary messages     ***/

Component Summary
  Finesse Name
    Module Name    Count  Size Information

  Control Block
    c#badipdet1    1      12 rows, 7 inputs, 4 outputs      (PLA)
    dffclr         4


External values used for symbolic variables:

  Symbolic          Symbolic          External
  Variable           Value          Boolean Value
  ---------------------------------------------

  state              Idle           000
                     Nondata1       101
                     Nondata2       010
                     Silence1       011
                     Silence2       001
                     Error          111

/*** End Component Summary messages ***/
```

A.2 Error Monitor

```
FINESSE errmon1

INPUT ~faultdetected, ~jabbertimedout, badinput, ~reseterrmon;
OUTPUT DFF phyerr;
INTERNAL DFF state;

SYMBOLIC state { Idle, Error };

FSM
{

DEFAULT
  { state := Idle;
    phyerr := 0;
  }

WHEN reseterrmon RESET
  {
    state = Idle;
    phyerr = 0;
  }

Idle: { IF ((faultdetected == 1) | (jabbertimedout == 1) | (badinput == 1))
          { phyerr := 1;
            state := Error; }
      ELSE
        LOOP;
      }

Error: { phyerr := 1;
       LOOP; }

}
```

```
                                      FINESSE
                                   Status Report

File Name  : errormonitor.fin
Module Name: errmon1

/*** Parsing/Synthesis messages     ***/
(WARNING)   No exit out of state Error
/*** End Parsing/Synthesis messages ***/

/*** Optimization messages    ***/
/*** End Optimization messages      ***/

/*** Structure Generation messages        ***/
      Removing unneeded DFF phyerr.

      /*** Testability Information  ***/

      INSTANCE    :  g0#errmon1
      SCANIN NET  :  f#scanin
      OUTPUT NET  :  state(1)

      /*** End Testability Information    ***/

      OUTPUT f#scanout COLLAPSED TO INTERNAL NET state(1).
      OUTPUT phyerr COLLAPSED TO INTERNAL NET state(1).
/*** End Structure Generation messages    ***/

/*** Component Summary messages     ***/

Component Summary
  Finesse Name
    Module Name    Count  Size Information

  Control Block
    c#errmon1      1      4 rows, 4 inputs, 1 outputs (PLA)
    dffclr         1


External values used for symbolic variables:

  Symbolic          Symbolic          External
  Variable           Value          Boolean Value
  ---------------------------------------------

  state              Idle           0
                     Error          1

/*** End Component Summary messages ***/
```

A.3 Frequency Select

```
#define WAIT (advance == 0)

FINESSE freqsel1
/* 2 bit frequency select with advance control */

INPUT freq0[7:0], freq1[7:0], freq2[7:0], freq3[7:0];
INPUT enablefreqsel, advance, ~resetfreqsel;
OUTPUT DFF cfreq[7:0], nfreq[7:0], state;

SYMBOLIC state { Idle, one, two, three, four };

FSM
{
DEFAULT
{
  LOOP;
}

WHEN resetfreqsel RESET
  { cfreq = 0;
```

```
    nfreq = 0;
    state = Idle;  }

Idle: { cfreq := freq0;
      nfreq := freq1;
      state := one; }

one: IF (enablefreqsel == 1)
      { IF WAIT
           { cfreq := freq0;
           nfreq := freq1;
           state := one; }
           ELSE
          { cfreq := freq1;
            nfreq := freq2;
            state := two; }
      }
      ELSE
      { state := one;
        cfreq := freq0;
        nfreq := freq1; }

two: IF (enablefreqsel == 1)
      { IF WAIT
          { cfreq := freq1;
          nfreq := freq2;
          state := two; }
          ELSE
          { cfreq := freq2;
          nfreq := freq3;
          state := three; }
      }
      ELSE
      { state := one;
        cfreq := freq0;
        nfreq := freq1; }

three: IF (enablefreqsel == 1)
      { IF WAIT
           { cfreq := freq2;
           nfreq := freq3;
           state := three; }
          ELSE
           { cfreq := freq3;
           nfreq := freq0;
           state := four; }
      }
      ELSE
      { state := one;
        cfreq := freq0;
        nfreq := freq1; }

four: IF (enablefreqsel == 1)
      { IF WAIT
           { cfreq := freq3;
           nfreq := freq0;
           state := four; }
          ELSE
           { cfreq := freq0;
           nfreq := freq1;
           state := one; }
      }
      ELSE
      { state := one;
        cfreq := freq0;
        nfreq := freq1; }

}


                                      FINESSE
                                   Status Report

File Name  : freqsel.fin
Module Name: freqsel1

/*** Parsing/Synthesis messages     ***/
/*** End Parsing/Synthesis messages ***/

/*** Optimization messages    ***/
/*** End Optimization messages      ***/

/*** Structure Generation messages        ***/

      /*** Testability Information  ***/

      INSTANCE    :  g0#freqsel1
      SCANIN NET  :  f#scanin
      OUTPUT NET  :  cfreq(0)

      INSTANCE    :  g1#freqsel1
      SCANIN NET  :  cfreq(0)
      OUTPUT NET  :  cfreq(1)

      INSTANCE    :  g2#freqsel1
      SCANIN NET  :  cfreq(1)
      OUTPUT NET  :  cfreq(2)

      INSTANCE    :  g3#freqsel1
      SCANIN NET  :  cfreq(2)
      OUTPUT NET  :  cfreq(3)

      INSTANCE    :  g4#freqsel1
      SCANIN NET  :  cfreq(3)
      OUTPUT NET  :  cfreq(4)

      INSTANCE    :  g5#freqsel1
      SCANIN NET  :  cfreq(4)
      OUTPUT NET  :  cfreq(5)

      INSTANCE    :  g6#freqsel1
      SCANIN NET  :  cfreq(5)
      OUTPUT NET  :  cfreq(6)

      INSTANCE    :  g7#freqsel1
      SCANIN NET  :  cfreq(6)
      OUTPUT NET  :  cfreq(7)

      INSTANCE    :  g8#freqsel1
      SCANIN NET  :  cfreq(7)
      OUTPUT NET  :  nfreq(0)

      INSTANCE    :  g9#freqsel1
      SCANIN NET  :  nfreq(0)
      OUTPUT NET  :  nfreq(1)

      INSTANCE    :  g10#freqsel1
      SCANIN NET  :  nfreq(1)
      OUTPUT NET  :  nfreq(2)

      INSTANCE    :  g11#freqsel1
      SCANIN NET  :  nfreq(2)
      OUTPUT NET  :  nfreq(3)

      INSTANCE    :  g12#freqsel1
      SCANIN NET  :  nfreq(3)
      OUTPUT NET  :  nfreq(4)

      INSTANCE    :  g13#freqsel1
      SCANIN NET  :  nfreq(4)
      OUTPUT NET  :  nfreq(5)

      INSTANCE    :  g14#freqsel1
      SCANIN NET  :  nfreq(5)
      OUTPUT NET  :  nfreq(6)

      INSTANCE    :  g15#freqsel1
      SCANIN NET  :  nfreq(6)
      OUTPUT NET  :  nfreq(7)

      INSTANCE    :  g16#freqsel1
      SCANIN NET  :  nfreq(7)
      OUTPUT NET  :  state(1)

      INSTANCE    :  g17#freqsel1
      SCANIN NET  :  state(1)
      OUTPUT NET  :  state(2)

      INSTANCE    :  g18#freqsel1
      SCANIN NET  :  state(2)
      OUTPUT NET  :  state(3)

      /*** End Testability Information    ***/

      OUTPUT f#scanout MERGED WITH state(3).
/*** End Structure Generation messages    ***/

/*** Component Summary messages     ***/

Component Summary
  Finesse Name
    Module Name    Count  Size Information

  Control Block
    c#freqsel1     1      150 rows, 37 inputs, 19 outputs   (PLA)
    dffpre         1
    dffclr         18


External values used for symbolic variables:

  Symbolic          Symbolic           External
  Variable           Value           Boolean Value
  ------------------------------------------------

  state              Idle            100
                     one             000
                     two             010
                     three           111
                     four            110

/*** End Component Summary messages ***/
```

A.4 Kicker Deleter

```
#define lvlS ^b110
#define lvl0 ^b000
#define lvl1 ^b001
#define lvl3 ^b010
#define lvl4 ^b011
#define lvl2 ^b100
#define lvlE ^b101

#define ENABLED ((lvlin == prevsym) & (enablekicker == 1) & (ignorekicker == 0))

FINESSE kickdel1

INPUT ~resetkickdel, firstsym, enablekicker, ignorekicker, lvlin[2:0];
OUTPUT DFF lvlout[2:0];
OUTPUT DFF prevsym[2:0];
OUTPUT DFF state;

SYMBOLIC state { Idle, Sym2, Sym3, Sym4, Sym5, Sym6, Sym7, Sym8, Sym9, \
SymA, SymB, SymC, SymD, SymE, SymF, SymN };

FSM
{

DEFAULT
  { state := Idle;
    lvlout := lvlin;
    prevsym := lvlin;
  }

WHEN resetkickdel RESET
  {
    state = Idle;
    lvlout = lvlS;
    prevsym = lvlS;
  }

Idle: { IF ((enablekicker == 1) & (ignorekicker == 0) & (firstsym == 1))
          { lvlout := lvlin;
          state := Sym2; }
        ELSE
          { lvlout := lvlin;
          state := Idle; }
      }

Sym2: { IF ENABLED
            {  state := Sym3; }
        ELSE
          { state := Idle; }
      }

Sym3: { IF ENABLED
            { state := Sym4; }
        ELSE
          { state := Idle; }
      }

Sym4: { IF ENABLED
            { state := Sym5; }
        ELSE
```

```
          { state := Idle; }
      }

Sym5: { IF ENABLED
          { state := Sym6; }
        ELSE
          { state := Idle; }
      }

Sym6: { IF ENABLED
          { state := Sym7; }
        ELSE
          { state := Idle; }
      }

Sym7: { IF ENABLED
          { state := Sym8; }
        ELSE
          { state := Idle; }
      }

Sym8: { IF ENABLED
          { state := Sym9; }
        ELSE
          { state := Idle; }
      }

Sym9: { IF ENABLED
          { state := SymA; }
        ELSE
          { state := Idle; }
      }

SymA: { IF ENABLED
          { state := SymB; }
        ELSE
          { state := Idle; }
      }

SymB: { IF ENABLED
          { state := SymC; }
        ELSE
          { state := Idle; }
      }

SymC: { IF ENABLED
          { state := SymD; }
        ELSE
          { state := Idle; }
      }

SymD: { IF ENABLED
          { state := SymE; }
        ELSE
          { state := Idle; }
      }

SymE: { IF ENABLED
          { state := SymF; }
        ELSE
          { state := Idle; }
      }

SymF: { IF ENABLED
          { state := SymN; }
        ELSE
          { state := Idle; }
      }

SymN: { IF ((lvlin == lvl2) & (enablekicker == 1) & (ignorekicker == 0))
          { lvlout := prevsym;
            state := Idle; }
        ELSE
          { lvlout := lvlin;
          state := Idle; }
      }
}


                                    FINESSE
                                Status Report

File Name  : kickdel.fin
Module Name: kickdel1

/*** Parsing/Synthesis messages     ***/
/*** End Parsing/Synthesis messages ***/

/*** Optimization messages    ***/
/*** End Optimization messages      ***/

/*** Structure Generation messages        ***/

      /*** Testability Information  ***/

      INSTANCE    :  g0#kickdel1
      SCANIN NET  :  f#scanin
      OUTPUT NET  :  lvlout(0)

      INSTANCE    :  g3#kickdel1
      SCANIN NET  :  lvlout(0)
      OUTPUT NET  :  prevsym(0)

      INSTANCE    :  g6#kickdel1
      SCANIN NET  :  prevsym(0)
      OUTPUT NET  :  state(1)

      INSTANCE    :  g7#kickdel1
      SCANIN NET  :  state(1)
      OUTPUT NET  :  state(2)

      INSTANCE    :  g8#kickdel1
      SCANIN NET  :  state(2)
      OUTPUT NET  :  state(3)

      INSTANCE    :  g9#kickdel1
      SCANIN NET  :  state(3)
      OUTPUT NET  :  state(4)

      INSTANCE    :  g1#kickdel1
      SCANIN NET  :  state(4)
      OUTPUT NET  :  lvlout(1)

      INSTANCE    :  g2#kickdel1
      SCANIN NET  :  lvlout(1)
      OUTPUT NET  :  lvlout(2)

      INSTANCE    :  g4#kickdel1
      SCANIN NET  :  lvlout(2)
      OUTPUT NET  :  prevsym(1)

      INSTANCE    :  g5#kickdel1
      SCANIN NET  :  prevsym(1)
      OUTPUT NET  :  prevsym(2)

      /*** End Testability Information    ***/

      OUTPUT f#scanout MERGED WITH prevsym(2).
/*** End Structure Generation messages    ***/

/*** Component Summary messages     ***/

Component Summary
  Finesse Name
    Module Name    Count  Size Information

  Control Block
    c#kickdel1     1      118 rows, 13 inputs, 7 outputs    (PLA)
    dffpre         4
    dffclr         6


External values used for symbolic variables:

  Symbolic          Symbolic           External
  Variable           Value           Boolean Value
  ------------------------------------------------

  state              Idle            0000
                     Sym2            0001
                     Sym3            1001
                     Sym4            0111
                     Sym5            1011
                     Sym6            0011
                     Sym7            1111
                     Sym8            1101
                     Sym9            1110
                     SymA            0010
                     SymB            1010
                     SymC            0110
                     SymD            1000
                     SymE            0101
                     SymF            0100
                     SymN            1100

/*** End Component Summary messages ***/
```

A.5 Kicker Inserter

```
#define impS ^b110
#define imp0 ^b000
#define imp1 ^b001
#define imp3 ^b010
#define imp4 ^b011
#define imp2 ^b100
#define impE ^b101

#define ENABLED ((impin == prevsym) & (enablekicker == 1) & (nokicker == 0))

FINESSE kickins1

INPUT ~resetkickins, firstsym, silencemodein, enablekicker, nokicker, impin[2:0];
OUTPUT DFF impout[2:0];
OUTPUT DFF prevsym[2:0];
OUTPUT DFF silencemodeout, state;

SYMBOLIC state { Idle, Sym2, Sym3, Sym4, Sym5, Sym6, Sym7, Sym8, Sym9, \
SymA, SymB, SymC, SymD, SymE, SymF, SymN };

FSM
{

DEFAULT
  { state := Idle;
    impout := impin;
    prevsym := impin;
    silencemodeout := silencemodein;
  }

WHEN resetkickins RESET
  {
    state = Idle;
    impout = impS;
    prevsym = impS;
    silencemodeout = 1;
  }

Idle: { IF ((enablekicker == 1) & (nokicker == 0) & (firstsym == 1))
          { impout := impin;
            state := Sym2; }
        ELSE
          { impout := impin;
            state := Idle; }
      }

Sym2: { IF ENABLED
            {  state := Sym3; }
        ELSE
          { state := Idle; }
      }

Sym3: { IF ENABLED
            { state := Sym4; }
        ELSE
          { state := Idle; }
      }

Sym4: { IF ENABLED
            { state := Sym5; }
        ELSE
          { state := Idle; }
      }

Sym5: { IF ENABLED
            { state := Sym6; }
        ELSE
          { state := Idle; }
      }

Sym6: { IF ENABLED
            { state := Sym7; }
        ELSE
          { state := Idle; }
      }
```

```
Sym7: { IF ENABLED
          { state := Sym8; }
        ELSE
          { state := Idle; }
      }

Sym8: { IF ENABLED
          { state := Sym9; }
        ELSE
          { state := Idle; }
      }

Sym9: { IF ENABLED
          { state := SymA; }
        ELSE
          { state := Idle; }
      }

SymA: { IF ENABLED
          { state := SymB; }
        ELSE
          { state := Idle; }
      }

SymB: { IF ENABLED
          { state := SymC; }
        ELSE
          { state := Idle; }
      }

SymC: { IF ENABLED
          { state := SymD; }
        ELSE
          { state := Idle; }
      }

SymD: { IF ENABLED
          { state := SymE; }
        ELSE
          { state := Idle; }
      }

SymE: { IF ENABLED
          { state := SymF; }
        ELSE
          { state := Idle; }
      }

SymF: { IF ENABLED
          { state := SymN; }
        ELSE
          { state := Idle; }
      }

SymN: { IF ENABLED
          { impout := imp2;
            state := Idle; }
        ELSE
          { impout := impin;
            state := Idle; }
      }
}
```

```
                                    FINESSE
                                Status Report

File Name  : kickins.fin
Module Name: kickins1

/*** Parsing/Synthesis messages     ***/
/*** End Parsing/Synthesis messages ***/

/*** Optimization messages    ***/
/*** End Optimization messages      ***/

/*** Structure Generation messages        ***/

      /*** Testability Information  ***/

      INSTANCE    :  g0#kickins1
      SCANIN NET  :  f#scanin
      OUTPUT NET  :  impout(0)

      INSTANCE    :  g3#kickins1
      SCANIN NET  :  impout(0)
      OUTPUT NET  :  prevsym(0)

      INSTANCE    :  g7#kickins1
      SCANIN NET  :  prevsym(0)
      OUTPUT NET  :  state(1)

      INSTANCE    :  g8#kickins1
      SCANIN NET  :  state(1)
      OUTPUT NET  :  state(2)

      INSTANCE    :  g9#kickins1
      SCANIN NET  :  state(2)
      OUTPUT NET  :  state(3)

      INSTANCE    :  g10#kickins1
      SCANIN NET  :  state(3)
      OUTPUT NET  :  state(4)

      INSTANCE    :  g1#kickins1
      SCANIN NET  :  state(4)
      OUTPUT NET  :  impout(1)

      INSTANCE    :  g2#kickins1
      SCANIN NET  :  impout(1)
      OUTPUT NET  :  impout(2)

      INSTANCE    :  g4#kickins1
      SCANIN NET  :  impout(2)
      OUTPUT NET  :  prevsym(1)

      INSTANCE    :  g5#kickins1
      SCANIN NET  :  prevsym(1)
      OUTPUT NET  :  prevsym(2)

      INSTANCE    :  g6#kickins1
      SCANIN NET  :  prevsym(2)
      OUTPUT NET  :  silencemodeout

      /*** End Testability Information    ***/

      OUTPUT f#scanout MERGED WITH silencemodeout.

/*** End Structure Generation messages    ***/

/*** Component Summary messages     ***/

Component Summary
  Finesse Name
    Module Name    Count  Size Information

  Control Block
    c#kickins1     1      126 rows, 13 inputs, 7 outputs    (PLA)
    dffpre         5
    dffclr         6


External values used for symbolic variables:

  Symbolic          Symbolic           External
  Variable           Value           Boolean Value
  ------------------------------------------------

  state              Idle            0000
                     Sym2            0001
                     Sym3            1001
                     Sym4            0111
                     Sym5            1011
                     Sym6            0011
                     Sym7            1111
                     Sym8            1101
                     Sym9            1110
                     SymA            0010
                     SymB            1010
                     SymC            0110
                     SymD            1000
                     SymE            0101
                     SymF            0100
                     SymN            1100

/*** End Component Summary messages ***/
```

A.6 No Transition Detector

```
#define lvlS ^b110
#define lvl0 ^b000
#define lvl1 ^b001
#define lvl3 ^b010
#define lvl4 ^b011
#define lvl2 ^b100
#define lvlE ^b101

#define IDENTICAL (lvlin == prevsym)

FINESSE notrandet1

INPUT ~resetnotrandet, lvlin[2:0];
OUTPUT DFF notrans, prevsym[2:0];
OUTPUT DFF state;

SYMBOLIC state { Sym1, Sym2, Sym3, Sym4, Sym5, Sym6, Sym7, Sym8, Sym9, \
SymA, SymB, SymC, SymD, SymE, SymF, Sym10, Sym11, Sym12, Sym13, Sym14, Sym15, \
Sym16, Sym17, Sym18 };

FSM
{

DEFAULT
  { state := Sym1;
    notrans := 0;
  }

WHEN resetnotrandet RESET
  {
    state = Sym1;
    prevsym = lvlS;
    notrans := 0;
  }

Sym1: { IF IDENTICAL
          { state := Sym2; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

Sym2: { IF IDENTICAL
            {  state := Sym3; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }

      }

Sym3: { IF IDENTICAL
            { state := Sym4; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

Sym4: { IF IDENTICAL
            { state := Sym5; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

Sym5: { IF IDENTICAL
            { state := Sym6; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

Sym6: { IF IDENTICAL
            { state := Sym7; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

Sym7: { IF IDENTICAL
            { state := Sym8; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }
```

```
Sym8: { IF IDENTICAL
          { state := Sym9; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

Sym9: { IF IDENTICAL
          { state := SymA; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

SymA: { IF IDENTICAL
          { state := SymB; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

SymB: { IF IDENTICAL
          { state := SymC; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

SymC: { IF IDENTICAL
          { state := SymD; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

SymD: { IF IDENTICAL
          { state := SymE; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

SymE: { IF IDENTICAL
          { state := SymF; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

SymF: { IF IDENTICAL
          { state := Sym10; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

Sym10: { IF IDENTICAL
          { state := Sym11; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

Sym11: { IF IDENTICAL
          { state := Sym12; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

Sym12: { IF IDENTICAL
          { state := Sym13; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

Sym13: { IF IDENTICAL
          { state := Sym14; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

Sym14: { IF IDENTICAL
          { state := Sym15; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

Sym15: { IF IDENTICAL
          { state := Sym16; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

Sym16: { IF IDENTICAL
          { state := Sym17; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

Sym17: { IF IDENTICAL
          { state := Sym18; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }

Sym18: { IF IDENTICAL
          { notrans := 1;
            state := Sym1;
            prevsym := lvlin; }
        ELSE
          { state := Sym1;
            prevsym := lvlin; }
      }
}


                                    FINESSE
                                Status Report

File Name  : notrandet.fin
Module Name: notrandet2

/*** Parsing/Synthesis messages     ***/
/*** End Parsing/Synthesis messages ***/

/*** Optimization messages    ***/
/*** End Optimization messages      ***/

/*** Structure Generation messages        ***/

      /*** Testability Information  ***/

      INSTANCE    :  g0#notrandet2
      SCANIN NET  :  f#scanin
      OUTPUT NET  :  notrans

      INSTANCE    :  g1#notrandet2
      SCANIN NET  :  notrans
      OUTPUT NET  :  prevsym(0)

      INSTANCE    :  g5#notrandet2
      SCANIN NET  :  prevsym(0)
      OUTPUT NET  :  state(2)

      INSTANCE    :  g2#notrandet2
      SCANIN NET  :  state(2)
      OUTPUT NET  :  prevsym(1)

      INSTANCE    :  g3#notrandet2
      SCANIN NET  :  prevsym(1)
      OUTPUT NET  :  prevsym(2)

      INSTANCE    :  g4#notrandet2
      SCANIN NET  :  prevsym(2)
      OUTPUT NET  :  state(1)

      INSTANCE    :  g6#notrandet2
      SCANIN NET  :  state(1)
      OUTPUT NET  :  state(3)

      INSTANCE    :  g7#notrandet2
      SCANIN NET  :  state(3)
      OUTPUT NET  :  state(4)

      INSTANCE    :  g8#notrandet2
      SCANIN NET  :  state(4)
      OUTPUT NET  :  state(5)

      /*** End Testability Information    ***/

      OUTPUT f#scanout MERGED WITH state(5).
/*** End Structure Generation messages    ***/

/*** Component Summary messages     ***/

Component Summary
  Finesse Name
    Module Name    Count  Size Information

  Control Block
    c#notrandet2    1      65 rows, 11 inputs, 9 outputs    (PLA)
    dffpre         6
    dffclr         3


External values used for symbolic variables:

  Symbolic          Symbolic           External
  Variable           Value           Boolean Value
  ------------------------------------------------

  state              Sym1            11101
                     Sym2            01101
                     Sym3            10101
                     Sym4            00101
                     Sym5            11001
                     Sym6            01001
                     Sym7            10001
                     Sym8            00001
                     Sym9            11111
                     SymA            01111
                     SymB            10111
                     SymC            00111
                     SymD            11011
                     SymE            01011
                     SymF            10011
                     Sym10           00011
                     Sym11           11100
                     Sym12           01100
                     Sym13           10100
                     Sym14           00100
                     Sym15           11000
                     Sym16           01000
                     Sym17           10000
                     Sym18           00000

/*** End Component Summary messages ***/
```

A.7 On Chip Monitor

```
#define VECTORLEN 31
#define TESTLEN 50
#define TESTLEN1 49
#define NOPASS 5
#define FINALSIGLEN 31
FINESSE ocm1

INPUT ~resetocm, testmode, count[7:0], sigverified, testpassno[7:0];
OUTPUT JKFF misren, prpgen, testmodeout, ~passctrclr;
OUTPUT JKFF testpass, testinprogress;
OUTPUT DFF testmoduleout, enablectr, advancepasscnt;
OUTPUT DFF state;

SYMBOLIC state { Idle, Setup, Test1, Collectsig, Clearctr,
Finalsig, Checkpass, Diagnose, Exit };

FSM
{

DEFAULT
  { state := Idle;
    testmoduleout := 0;
  }

WHEN resetocm RESET
  {
    state = Idle;
```

```
    testmodeout = 0;
    advancepasscnt = 0;
    passctrclr = 1;
    misren = 0;
    testpass = 0;
    testinprogress = 0;
    prpgen = 0;
    enablectr = 0;
    testmoduleout = 0;
  }

Idle: { IF (testmode == 1)
        { enablectr := 1;
          testmoduleout := 0;
          testmodeout := 0;
          testpass := 0;
          testinprogress := 1;
          passctrclr := 0;
          prpgen := 1;
          state := Setup; }
        ELSE
        LOOP;
      }

Setup: { IF (count != VECTORLEN)
         { enablectr := 1;
         LOOP; }
       ELSE
         { enablectr := 0;
           testmodeout := 1;
           state := Test1; }
        }

Test1: { IF (count != TESTLEN)
         { enablectr := 1;
           testmoduleout := 1;
           state := Test1; }
       ELSE
         { enablectr := 0;
           advancepasscnt := 1;
           testmoduleout := 0;
           misren := 1;
           state := Checkpass; }
      }

Checkpass: { IF (testpassno == NOPASS)
             { passctrclr := 1;
             enablectr := 1;
               testmoduleout := 1;
             state := Collectsig; }
           ELSE
             { state := Test1;
             testmoduleout := 1;
             enablectr := 1;
             advancepasscnt := 0; }
           }

Collectsig: { IF (count != TESTLEN)
              { enablectr := 1;
              testmoduleout := 1;
            LOOP; }
            ELSE
             { enablectr := 0;
               testmoduleout := 1;
             state := Clearctr; }
          }

Clearctr:  { IF (testmode == 0)
/* to activate test passed signal, test line should be negated before this stage
*/
            { enablectr := 1;
              testmoduleout := 0;
              testmodeout := 0;
              state := Finalsig; }
            ELSE IF (testmode == 1)
            { testinprogress := 0;
              testmoduleout := 0;
              testmodeout := 0;
              enablectr := 1;
              state := Diagnose; }
         }

Finalsig: { IF ((count != FINALSIGLEN) & (sigverified == 1))
            { enablectr := 1;
            LOOP; }
          ELSE IF (sigverified == 1)
            { state := Exit;
            enablectr := 0;
            testinprogress := 0;
            testpass := 1; }
          ELSE
            { state := Exit;
            enablectr := 0;
            testpass := 0;
            testinprogress := 0; }
         }

Diagnose: { IF (count != FINALSIGLEN)
           { enablectr := 1;
            LOOP; }
          ELSE
            { state := Exit;
              enablectr := 0; }
           }

Exit: { misren := 0;
      prpgen := 0;
      LOOP; }
}
```

```
                                      FINESSE
                                  Status Report

File Name  : ocm.fin
Module Name: ocm2

/*** Parsing/Synthesis messages     ***/
(WARNING)   No exit out of state Exit
/*** End Parsing/Synthesis messages ***/

/*** Optimization messages    ***/
/*** End Optimization messages      ***/

/*** Structure Generation messages        ***/
/*** End Structure Generation messages    ***/

/*** Component Summary messages     ***/
```

```
Component Summary
  Finesse Name
    Module Name    Count  Size Information

  Control Block
    c#ocm2         1      45 rows, 22 inputs, 15 outputs    (PLA)
    dffpre         1
    dffclr         6
    jkclr          6


External values used for symbolic variables:

  Symbolic          Symbolic           External
  Variable           Value           Boolean Value
  ---------------------------------------------

  state              Idle            0001
                     Setup           1001
                     Test1           0101
                     Collectsig      1101
                     Clearctr        0011
                     Finalsig        1011
                     Checkpass       0111
                     Diagnose        1111
                     Exit            0000

/*** End Component Summary messages ***/
```

A.8 Receiver Physical Symbol Converter

```
#define Lvlis (lvl == ^b110)
#define Lvli0 (lvl == ^b000)
#define Lvli1 (lvl == ^b001)
#define Lvli3 (lvl == ^b010)
#define Lvli4 (lvl == ^b011)
#define Lvli2 (lvl == ^b100)
#define LvliE (lvl == ^b101)

#define Nondata   ^b100
#define Badsignal ^b011
#define Silence   ^b111
#define Data0     ^b000
#define Data1     ^b001

FINESSE pscrx1

INPUT    lvl[2:0], ~resetline;
OUTPUT   DFF rxsym[2:0], silencemode;
INTERNAL DFF  state;
SYMBOLIC state {Sil, Error, Pa, Pb, Pc, Pd, Na, Nb,  FD0a, FD1a, N0c, N1c, N0d,
N1d, \
FD0b, FD1b, FD01, FD10, D0a, D1a, D0b, D1b} ;

FSM
{
DEFAULT
{
  rxsym := Silence;
  silencemode := 0;
  state := Error;
}
WHEN resetline RESET
{
  state = Sil;
  rxsym = Silence;
  silencemode = 1;
}

Sil: { rxsym := Silence;
       silencemode := 1;
       IF Lvli4
         state := Pa;
       ELSE LOOP; }

Error: { rxsym := Silence;
       IF Lvlis
           state := Sil;
       ELSE LOOP; }

Pa: { rxsym := Data1;
      IF Lvli4
        state := Pb;
      ELSE
        state := Error; }

Pb: { rxsym := Data1;
      IF Lvli0
        state := Pc;
      ELSE
        state := Error; }

Pc: { rxsym := Data0;
      IF Lvli0
        state := Pd;
      ELSE
        state := Error; }

Pd: { rxsym := Data0;
      IF Lvli4
        state := Pa;
      ELSE IF Lvli2
        state := Na;
      ELSE IF Lvli0
        state := FD0a;
      ELSE IF Lvli1
        state := FD01;
      ELSE IF Lvli3
        state := FD10;
      ELSE
        state := Error; }

Na: { rxsym := Nondata;
      IF Lvli2
        state := Nb;
      ELSE
        state := Error; }

Nb: { rxsym := Nondata;
      IF Lvli0
        state := FD0a;
      ELSE IF Lvli1
        state := FD01;
      ELSE IF Lvli3
        state := FD10;
      ELSE IF Lvli4
```

```
       state := FD1a;
      ELSE
       state := Error; }

FD0a: { rxsym := Data0;
       IF Lvli0
        state := N0c;
       ELSE
        state := Error; }

FD1a: { rxsym := Data1;
       IF Lvli4
        state := N1c;
       ELSE
        state := Error; }

N0c:  { rxsym := Nondata;
       IF Lvli2
        state := N0d;
       ELSE
        state := Error; }

N0d:  { rxsym := Nondata;
       IF Lvli2
        state := FD0b;
       ELSE
        state := Error; }

N1c:  { rxsym := Nondata;
       IF Lvli2
        state := N1d;
       ELSE
        state := Error; }

N1d:  { rxsym := Nondata;
       IF Lvli2
        state := FD1b;
       ELSE
        state := Error; }

FD01: { rxsym := Data1;
       state := Error; }

FD10: { rxsym := Data0;
       state := Error; }

FD0b: { rxsym := Data0;
       IF Lvli0
        state := D0a;
       ELSE IF Lvli1
        state := D0a;
       ELSE IF Lvli3
        state := D1a;
       ELSE IF Lvli4
        state := D1a;
       ELSE
        state := Error; }

FD1b: { rxsym := Data1;
       IF Lvli0
        state := D0a;
       ELSE IF Lvli1
        state := D0a;
       ELSE IF Lvli3
        state := D1a;
       ELSE IF Lvli4
        state := D1a;
       ELSE
        state := Error; }

D0a: { rxsym := Data0;
       IF Lvli0
        state := D0b;
       ELSE IF Lvli1
        state := D1b;
       ELSE
        state := Error; }

D1a: { rxsym := Data1;
       IF Lvli3
        state := D0b;
       ELSE IF Lvli4
        state := D1b;
       ELSE
        state := Error; }

D0b: { rxsym := Data0;
       IF Lvli0
        state := D0a;
       ELSE IF Lvli1
        state := D0a;
       ELSE IF Lvli3
        state := D1a;
       ELSE IF Lvli4
        state := D1a;
       ELSE IF Lvli2
        state := Na;
       ELSE IF Lvlis
        state := Sil;
       ELSE
        state := Error; }

D1b: { rxsym := Data1;
       IF Lvli0
        state := D0a;
       ELSE IF Lvli1
        state := D0a;
       ELSE IF Lvli3
        state := D1a;
       ELSE IF Lvli4
        state := D1a;
       ELSE IF Lvli2
        state := Na;
       ELSE IF Lvlis
        state := Sil;
       ELSE
        state := Error; }
}


                                        FINESSE
                                    Status Report

File Name  : pscrx.fin
Module Name: pscrx1

/*** Parsing/Synthesis messages     ***/
/*** End Parsing/Synthesis messages ***/

/*** Optimization messages    ***/
/*** End Optimization messages      ***/

/*** Structure Generation messages        ***/

      /*** Testability Information  ***/

      INSTANCE    :  g1#pscrx1
      SCANIN NET  :  f#scanin
      OUTPUT NET  :  state(2)

      INSTANCE    :  g2#pscrx1
      SCANIN NET  :  state(2)
      OUTPUT NET  :  state(3)

      INSTANCE    :  g3#pscrx1
      SCANIN NET  :  state(3)
      OUTPUT NET  :  state(4)

      INSTANCE    :  g4#pscrx1
      SCANIN NET  :  state(4)
      OUTPUT NET  :  state(5)

      INSTANCE    :  g0#pscrx1
      SCANIN NET  :  state(5)
      OUTPUT NET  :  state(1)

      INSTANCE    :  g5#pscrx1
      SCANIN NET  :  state(1)
      OUTPUT NET  :  rxsym(0)

      INSTANCE    :  g6#pscrx1
      SCANIN NET  :  rxsym(0)
      OUTPUT NET  :  rxsym(1)

      INSTANCE    :  g7#pscrx1
      SCANIN NET  :  rxsym(1)
      OUTPUT NET  :  rxsym(2)

      INSTANCE    :  g8#pscrx1
      SCANIN NET  :  rxsym(2)
      OUTPUT NET  :  silencemode

      /*** End Testability Information    ***/

      OUTPUT f#scanout MERGED WITH silencemode.
/*** End Structure Generation messages    ***/

/*** Component Summary messages     ***/

Component Summary
  Finesse Name
    Module Name    Count  Size Information

  Control Block
    c#pscrx1       1      31 rows, 8 inputs, 9 outputs      (PLA)
    dffpre         5
    dffclr         4


External values used for symbolic variables:

  Symbolic          Symbolic           External
  Variable           Value           Boolean Value
  ------------------------------------------------

  state              Sil             00001
                     Error           00000
                     Pa              10000
                     Pb              10001
                     Pc              01111
                     Pd              10010
                     Na              11110
                     Nb              11010
                     FD0a            10011
                     FD1a            01011
                     N0c             11000
                     N1c             11100
                     N0d             11101
                     N1d             11011
                     FD0b            01110
                     FD1b            01010
                     FD01            00011
                     FD10            01100
                     D0a             01001
                     D1a             10100
                     D0b             00110
                     D1b             00010

/*** End Component Summary messages ***/
```

A.9 Transmitter Physical Symbol Converter

```
#define Nondata (txsym == ^b10-)
#define Padidle (txsym == ^b01-)
#define Silence (txsym == ^b11-)
#define Data0   (txsym == ^b000)
#define Data1   (txsym == ^b001)

#define impoS ^b110
#define impo0 ^b000
#define impo1 ^b001
#define impo3 ^b010
#define impo4 ^b011
#define impo2 ^b100
#define impoE ^b101


FINESSE psctx1

INPUT    txsym[2:0];
INPUT    ~resetline;
OUTPUT   DFF enablekicker, load, silencemode;
OUTPUT   DFF imp[2:0];
INTERNAL DFF state;
SYMBOLIC state {Sil, Error, SD0, SD3, Pa, Pb, Pc, Pd, Pe, \
Na, Nb, FD0a, FD1a, N0c, N1c, N0d, N1d, FD0b, FD1b, FD01, FD10, FDN0, FDN1,\
D00, D01, D10, D11, D000, D001, D010, D011, D100, D101, D110, D111, \
N00, N01, N10, N11, PD0, PD3} ;

FSM
{
DEFAULT
{
  enablekicker := 0;
```

```
  load := 0;
  imp := impoS;
  state := Error;
  silencemode := 0;

}
WHEN resetline RESET
{
  state = Sil;
  enablekicker = 0;
  load = 0;
  imp = impoS;
  silencemode = 1;
}

Sil: {  imp := impoS;
      silencemode := 1;
      IF Padidle
        state := Pa;
      ELSE LOOP;  }

Error: { imp := impoE;
       IF Silence
           state := Sil;
       ELSE LOOP; }

SD0: { imp := impo0;
       enablekicker := 1;
       state := Sil; }

SD3: { imp := impo3;
       enablekicker := 1;
       state := Sil; }

PD0: { imp := impo0;
       enablekicker := 1;
       state := Pb; }

PD3: { imp := impo3;
       enablekicker := 1;
       state := Pb; }

Pa:  { imp := impoS;
       IF Padidle
         state := Pb;
       ELSE
         state := Error; }

Pb:  { imp := impo4;
       IF Padidle
         state := Pc;
       ELSE
         state := Error; }

Pc:  { imp := impo4;
       IF Padidle
         state := Pd;
       ELSE
         state := Error; }

Pd:  { imp := impo0;
       IF Padidle
         state := Pe;
       ELSE IF Nondata
         state := Na;
       ELSE
         state := Error; }

Pe: { imp := impo0;
       IF Padidle
         state := Pb;
      ELSE
         state := Error; }

Na: { imp := impo0;
      IF Nondata
         state := Nb;
      ELSE
         state := Error; }

Nb: { imp := impo2;
      enablekicker := 1;
      IF Data0
         state := FD0a;
      ELSE IF Data1
         state := FD1a;
      ELSE
         state := Error; }

FD0a: { imp := impo2;
        enablekicker := 1;
        IF Nondata
         state := N0c;
        ELSE
         state := Error; }

FD1a: { imp := impo2;
        enablekicker := 1;
        IF Nondata
         state := N1c;
        ELSE
         state := Error; }

N0c:  { imp := impoS;
        enablekicker := 1;
        IF Nondata
         state :=  N0d;
        ELSE
         state := Error; }

N0d:  { imp := impoS;
        enablekicker := 1;
        IF Data0
         state := FD0b;
        ELSE IF Data1
         state := FD01;
        ELSE
         state := Error; }

N1c:  { imp := impoS;
        enablekicker := 1;
        IF Nondata
         state := N1d;
        ELSE
         state := Error; }

N1d:  { imp := impoS;
        enablekicker := 1;
        IF Data0
         state := FD10;
        ELSE IF Data1
         state := FD1b;
        ELSE
         state := Error; }

FD01: { imp := impo1;
        enablekicker := 1;
        load := 1; /* parallel load {1} */
        state := Error; }

FD10: { imp := impo3;
        enablekicker := 1;
        load := 1; /* parallel load {3} */
        state := Error; }

FD0b: { imp := impo0;
        enablekicker := 1;
        load := 1; /* parallel load {0} */
        IF Data0
         state := FDN0;
        ELSE IF Data1
         state := FDN1;
        ELSE
         state := Error; }

FD1b: { imp := impo4;
        enablekicker := 1;
        load := 1; /* parallel load {4} */
        IF Data0
         state := FDN0;
        ELSE IF Data1
         state := FDN1;
        ELSE
         state := Error; }

FDN0: { imp := impo2;
        enablekicker := 1;
        IF Data0
         state := D00;
        ELSE IF Data1
         state := D01;
        ELSE
         state := Error; }

FDN1: { imp := impo2;
        enablekicker := 1;
        IF Data0
         state := D10;
        ELSE IF Data1
         state := D11;
        ELSE
         state := Error; }

D00:  { imp := impo0;
        enablekicker := 1;
        IF Data0
         state := D000;
        ELSE IF Data1
         state := D001;
        ELSE IF Nondata
         state := N00;
        ELSE IF Silence
         state := SD0;
        ELSE IF Padidle
         state := PD0; }

D01:  { imp := impo1;
        enablekicker := 1;
        IF Data0
         state := D010;
        ELSE IF Data1
         state := D011;
        ELSE IF Nondata
         state :=  N01;
        ELSE
         state := Error; }

D10:  { imp := impo3;
        enablekicker := 1;
        IF Data0
         state := D100;
        ELSE IF Data1
         state := D101;
        ELSE IF Nondata
         state := N10;
        ELSE IF Silence
         state := SD3;
        ELSE IF Padidle
         state := PD3; }

D11:  { imp := impo4;
        enablekicker := 1;
        IF Data0
         state := D110;
        ELSE IF Data1
         state := D111;
        ELSE IF Nondata
         state := N11;
        ELSE
         state := Error; }

D000: { imp := impo0;
        enablekicker := 1;
        IF Data0
         state := D00;
        ELSE IF Data1
         state :=  D01;
        ELSE
         state := Error; }

D001: { imp := impo0;
        enablekicker := 1;
        IF Data0
         state := D10;
        ELSE IF Data1
         state := D11;
        ELSE
         state := Error; }

D010: { imp := impo1;
        enablekicker := 1;
        IF Data0
         state := D00;
        ELSE IF Data1
         state := D01;
        ELSE
```

```
        state := Error; }

D011: { imp := impo1;
        enablekicker := 1;
        IF Data0
         state := D10;
        ELSE IF Data1
         state := D11;
        ELSE
         state := Error; }

D100: { imp := impo3;
        enablekicker := 1;
        IF Data0
         state := D00;
        ELSE IF Data1
         state := D01;
        ELSE
         state :=  Error; }

D101: { imp := impo3;
        enablekicker := 1;
        IF Data0
         state := D10;
        ELSE IF Data1
         state := D11;
        ELSE
         state := Error; }

D110: { imp := impo4;
        enablekicker := 1;
        IF Data0
         state := D00;
        ELSE IF Data1
         state := D01;
        ELSE
         state := Error; }

D111: { imp := impo4;
        enablekicker := 1;
        IF Data0
         state := D10;
        ELSE IF Data1
         state := D11;
        ELSE
         state := Error; }

N00:  { imp := impo0;
        enablekicker := 1;
        IF Nondata
         state := Nb;
        ELSE
         state := Error; }

N01:  { imp := impo1;
        enablekicker := 1;
        IF Nondata
         state := Nb;
        ELSE
         state := Error; }

N10:  { imp := impo3;
        enablekicker := 1;
        IF Nondata
         state :=  Nb;
        ELSE
         state := Error; }

N11:  { imp := impo4;
        enablekicker := 1;
        IF Nondata
         state := Nb;
        ELSE
         state := Error; }
}
                                      FINESSE
                                   Status Report

File Name  : psctx.fin
Module Name: psctx1

/*** Parsing/Synthesis messages     ***/
/*** End Parsing/Synthesis messages ***/

/*** Optimization messages    ***/
/*** End Optimization messages      ***/

/*** Structure Generation messages        ***/

      /*** Testability Information  ***/

      INSTANCE    :  g1#psctx1
      SCANIN NET  :  f#scanin
      OUTPUT NET  :  state(2)

      INSTANCE    :  g6#psctx1
      SCANIN NET  :  state(2)
      OUTPUT NET  :  enablekicker

      INSTANCE    :  g7#psctx1
      SCANIN NET  :  enablekicker
      OUTPUT NET  :  load

      INSTANCE    :  g9#psctx1
      SCANIN NET  :  load
      OUTPUT NET  :  imp(0)

      INSTANCE    :  g0#psctx1
      SCANIN NET  :  imp(0)
      OUTPUT NET  :  state(1)

      INSTANCE    :  g2#psctx1
      SCANIN NET  :  state(1)
      OUTPUT NET  :  state(3)

      INSTANCE    :  g3#psctx1
      SCANIN NET  :  state(3)
      OUTPUT NET  :  state(4)

      INSTANCE    :  g4#psctx1
      SCANIN NET  :  state(4)
      OUTPUT NET  :  state(5)

      INSTANCE    :  g5#psctx1
      SCANIN NET  :  state(5)
      OUTPUT NET  :  state(6)

      INSTANCE    :  g8#psctx1
      SCANIN NET  :  state(6)
      OUTPUT NET  :  silencemode

      INSTANCE    :  g10#psctx1
      SCANIN NET  :  silencemode
      OUTPUT NET  :  imp(1)

      INSTANCE    :  g11#psctx1
      SCANIN NET  :  imp(1)
      OUTPUT NET  :  imp(2)

      /*** End Testability Information    ***/

      OUTPUT f#scanout MERGED WITH imp(2).
/*** End Structure Generation messages    ***/

/*** Component Summary messages     ***/

Component Summary
  Finesse Name
    Module Name    Count  Size Information

  Control Block
    c#psctx1       1      81 rows, 9 inputs, 12 outputs     (PLA)
    dffpre         8
    dffclr         4


External values used for symbolic variables:

  Symbolic          Symbolic           External
  Variable           Value          Boolean Value
  ------------------------------------------------

  state              Sil             111101
                     Error           000000
                     SD0             111001
                     SD3             110000
                     Pa              100000
                     Pb              101010
                     Pc              101011
                     Pd              011010
                     Pe              011101
                     Na              110010
                     Nb              110101
                     FD0a            010101
                     FD1a            000101
                     N0c             101001
                     N1c             001110
                     N0d             010111
                     N1d             101101
                     FD0b            001001
                     FD1b            010000
                     FD01            001000
                     FD10            011100
                     FDN0            100100
                     FDN1            000011
                     D00             100111
                     D01             010001
                     D10             110111
                     D11             110001
                     D000            001111
                     D001            011111
                     D010            100110
                     D011            001010
                     D100            110110
                     D101            111100
                     D110            101000
                     D111            111000
                     N00             110011
                     N01             101100
                     N10             000111
                     N11             000100
                     PD0             101111
                     PD3             010100

/*** End Component Summary messages ***/
```

A.10 Command/Data Encoder (Receiver)

```
#define NACK       ^b100
#define ACK        ^b010
#define ACQUIRE    ^b000
#define IDLERESP   ^b001
#define PHYERR     ^b111

#define Nondata   ^b100
#define Badsignal ^b011
#define Silence   ^b111
#define Data0     ^b000
#define Data1     ^b001


#define SMREQ    (smreqin == 1)
#define ACQUIREFREQ (freqacquire == 1)

FINESSE rxcmd1

INPUT rxsymin[2:0], phyerrin, nakcmdin, intloopback, ackcmdin, freqacquire;
INPUT ~resetrxcmd, ~carrierlost, ~resetin, smreqin;
OUTPUT DFF ~smind, rxsym[2:0], state;

SYMBOLIC state { SMindmode, Rxenablemode };


FSM
{

DEFAULT
 { LOOP;
   rxsym := IDLERESP;
   smind := 0;
 }

WHEN resetrxcmd RESET
 {
   state = SMindmode;
   rxsym = IDLERESP;
   smind = 1;
 }

SMindmode: { IF (SMREQ | ACQUIREFREQ | (phyerrin == 1) | (resetin == 1))
              { LOOP;
              smind := 1;
```

```
            IF (phyerrin == 1)
              rxsym := PHYERR;
            ELSE IF (nakcmdin == 1)
              rxsym := NACK;
            ELSE IF ((phyerrin == 0) & (ackcmdin == 1))
              rxsym := ACK;
            ELSE IF ((resetin == 0) & (smreqin == 0) & ACQUIREFREQ)
              rxsym := ACQUIRE;
            ELSE
              rxsym := IDLERESP;
            }
          ELSE IF ((smreqin == 0) & (freqacquire == 0) & (phyerrin == 0) &
(resetin == 0))
            { IF ((carrierlost == 1) & (intloopback == 0))
              rxsym := Badsignal;
            ELSE
              rxsym := rxsymin;
              smind := 0;
              state := Rxenablemode; }
          }

Rxenablemode: { IF (SMREQ | ACQUIREFREQ | (phyerrin == 1) | (resetin == 1))
             { smind := 1;
               IF (phyerrin == 1)
                 rxsym := PHYERR;
               ELSE IF (ACQUIREFREQ & (resetin == 0) & (smreqin == 0))
                 rxsym := ACQUIRE;
               ELSE
                 rxsym := IDLERESP;
               state := SMindmode; }
           ELSE
             { smind := 0;
               IF ((carrierlost == 1) & (intloopback == 0))
                 rxsym := Badsignal;
               ELSE
                 rxsym := rxsymin;
               LOOP; }
             }
}
```

```
                                    FINESSE
                                Status Report

File Name  : rxcmd.fin
Module Name: rxcmd2

/*** Parsing/Synthesis messages     ***/
/*** End Parsing/Synthesis messages ***/

/*** Optimization messages    ***/
/*** End Optimization messages      ***/

/*** Structure Generation messages        ***/
      Removing unneeded DFF smind.

      /*** Testability Information  ***/

      INSTANCE    :  g2#rxcmd2
      SCANIN NET  :  f#scanin
      OUTPUT NET  :  rxsym(1)

      INSTANCE    :  g3#rxcmd2
      SCANIN NET  :  rxsym(1)
      OUTPUT NET  :  rxsym(2)

      INSTANCE    :  g4#rxcmd2
      SCANIN NET  :  rxsym(2)
      OUTPUT NET  :  state(1)

      INSTANCE    :  g1#rxcmd2
      SCANIN NET  :  state(1)
      OUTPUT NET  :  rxsym(0)

      /*** End Testability Information    ***/

      OUTPUT smind MERGED WITH state(1).
      OUTPUT f#scanout MERGED WITH rxsym(0).
/*** End Structure Generation messages    ***/

/*** Component Summary messages     ***/

Component Summary
  Finesse Name
    Module Name    Count  Size Information

  Control Block
    c#rxcmd2       1      15 rows, 11 inputs, 4 outputs     (PLA)
    dffpre         1
    dffclr         3


External values used for symbolic variables:

  Symbolic          Symbolic           External
  Variable           Value           Boolean Value
  ------------------------------------------------

  state              SMindmode       0
                     Rxenablemode    1

/*** End Component Summary messages ***/
```

A.11 Start Delimiter Detector

```
#define lvlS ^b110
#define lvl0 ^b000
#define lvl1 ^b001
#define lvl3 ^b010
#define lvl4 ^b011
#define lvl2 ^b100
#define lvlE ^b101

FINESSE sddetect1

INPUT ~resetsddetect, internalloopback, carrierdetect, notrandetect,
nondatain[2:0], lvlin[2:0];
OUTPUT DFF agchold;
OUTPUT DFF sdfound;
OUTPUT DFF state;

SYMBOLIC state { Idle, SDfound1, SDfound2, SDfound3, SDfound4, SDwait };

FSM
{

DEFAULT
  { state := Idle;
    agchold := 0;
    sdfound := 0;
  }

WHEN resetsddetect RESET
  {
    state = Idle;
    agchold = 0;
    sdfound = 0;
  }

Idle: { IF (((internalloopback == 1) | (carrierdetect == 1)) & (lvlin == lvl2))
       /* Non data symbol detected */
          { state := SDfound1; }
        ELSE
            LOOP;
      }

SDfound1: { IF (((internalloopback == 1) | (carrierdetect == 1)) & (lvlin ==
lvl0))
             /* {0} found */
              { state := SDfound2; }
            ELSE
              state := Idle;
        }

SDfound2: { IF (((internalloopback == 1) | (carrierdetect == 1)) & (lvlin ==
lvl2))
           /* Non data symbol detected */
            { state := SDfound3; }
           ELSE
             state := Idle;
         }

SDfound3: { IF (((internalloopback == 1) | (carrierdetect == 1)) & (lvlin ==
lvl0))
            /* {0} found */
              { state := SDfound4;
              sdfound := 1;
              IF (carrierdetect == 1)
              agchold := 1; }
             ELSE
               state := Idle;
        }

SDfound4: { IF ((internalloopback == 1) | (carrierdetect == 1))
        /* skip the Nondata from the descrambler stage */
              { state := SDwait;
              sdfound := 1;
              IF (carrierdetect == 1)
              agchold := 1; }
             ELSE
               state := Idle;
        }


SDwait:   { IF ((nondatain == lvl2) /* Non data symbol found -- exit */
            | ((internalloopback == 0) & (notrandetect == 1)) \
            | ((internalloopback == 0) & (carrierdetect == 0)))
            { state := Idle;
            sdfound := 0;
            agchold := 0; }
          ELSE IF ((internalloopback == 0) & (carrierdetect == 1) & (notrandetect
== 0))
            { LOOP;
            sdfound := 1;
            agchold := 1; }
          ELSE IF (internalloopback == 1)
            /* ignore notrandetect */
            { LOOP;
            sdfound := 1;
            agchold := carrierdetect; }
         }
}
```

```
                                    FINESSE
                                Status Report

File Name  : sddetect.fin
Module Name: sddetect1

/*** Parsing/Synthesis messages     ***/
/*** End Parsing/Synthesis messages ***/

/*** Optimization messages    ***/
/*** End Optimization messages      ***/

/*** Structure Generation messages        ***/
      Removing unneeded DFF sdfound.

      /*** Testability Information  ***/

      INSTANCE    :  g0#sddetect1
      SCANIN NET  :  f#scanin
      OUTPUT NET  :  agchold

      INSTANCE    :  g2#sddetect1
      SCANIN NET  :  agchold
      OUTPUT NET  :  state(1)

      INSTANCE    :  g3#sddetect1
      SCANIN NET  :  state(1)
      OUTPUT NET  :  state(2)

      INSTANCE    :  g4#sddetect1
      SCANIN NET  :  state(2)
      OUTPUT NET  :  state(3)

      /*** End Testability Information    ***/

      OUTPUT f#scanout MERGED WITH state(3).
      OUTPUT sdfound MERGED WITH state(1).
/*** End Structure Generation messages    ***/

/*** Component Summary messages     ***/

Component Summary
  Finesse Name
    Module Name    Count  Size Information

  Control Block
    c#sddetect1    1      19 rows, 12 inputs, 4 outputs     (PLA)
    dffclr         4


External values used for symbolic variables:
```

```
  Symbolic          Symbolic          External
  Variable           Value          Boolean Value
  ------------------------------------------------

  state              Idle           000
                     SDfound1       100
                     SDfound2       010
                     SDfound3       110
                     SDfound4       001
                     SDwait         101

/*** End Component Summary messages ***/
```

A.12 Station Management Controller

```
#define NOCHANGE   0
#define CHANGECURR 1
#define CHANGENEXT 2
#define CHANGEBOTH 3
#define ALLSILENT ((rxsilence == 1) & (txsilence == 1))

FINESSE smctrl1
/* station management control unit */

INPUT master, ~smindin, intloopbackin, extloopbackin;
INPUT txsilence, rxsilence, rxnextfreqin[7:0];
INPUT rxvalidframe, txendofframe, ~startsmtx, ~resetsmctrl;
/* startsmtx is the txen input */
INPUT  rxcurrfreqmatch;
OUTPUT JKFF freqacquire;
OUTPUT JKFF enablesmtx, enablesmrx, enablefreqsel;
OUTPUT DFF  advancefreq, state, ~freqchange[1:0];

SYMBOLIC state { Idle, Initmaster1, Initmaster2,
Msilwait1, Msilwait2, Mcfreq1, Mcfreq2,
Initslave1, Initslave2, Ssilwait1, Ssilwait2, Scfreq
};

FSM
{
DEFAULT
{
  advancefreq := 0;
  freqchange := NOCHANGE;
  freqacquire := 0;
  LOOP;
}

WHEN resetsmctrl RESET
  {
    freqacquire = 1;
    enablesmtx = 0;
    enablesmrx = 0;
    enablefreqsel = 0;
    advancefreq = 0;
    freqchange = NOCHANGE;
    state = Idle;  }

Idle: { IF ((intloopbackin == 1) | (startsmtx == 0))
/* enable station management only if not in internal loopback mode */
          { advancefreq := 0;
          enablesmtx := 0;
            enablesmrx := 0;
          freqacquire := 0;
          /* disable station management */
          state := Idle; }
       ELSE IF ((intloopbackin == 0) & (extloopbackin == 1) & (master == 1))
        { advancefreq := 0;
          freqacquire := 0;
          enablesmtx := 1;
          enablefreqsel := 1;
          state := Initmaster1; }

       ELSE IF ((intloopbackin == 0) & (extloopbackin == 1) & (master == 0))
        { advancefreq := 0;
          freqacquire := 0;
          enablesmrx := 1;
          state := Initslave1; }

       ELSE IF ((intloopbackin == 0) & (extloopbackin == 0) & (master == 1))
        /* wait for freq acquisition only if in operation mode */
        { advancefreq := 0;
          freqacquire := 1;
          enablesmtx := 1;
          enablefreqsel := 1;
          state := Initmaster1; }

       ELSE IF ((intloopbackin == 0) & (extloopbackin == 0) & (master == 0))
        /* wait for freq acquisition only if in operation mode */
        { advancefreq := 0;
          freqacquire := 1;
          enablesmrx := 1;
          state := Initslave1; }
     }

Initmaster1: { advancefreq := 1;
             freqacquire := 1;
             freqchange := CHANGEBOTH;
             state := Initmaster2; }

Initmaster2: { advancefreq := 0;
             freqchange := NOCHANGE;
             IF (txendofframe == 1)
               { freqacquire := 0;
               state := Msilwait1; }
              ELSE
             { freqacquire := 1;
               LOOP; }
            }
Msilwait1: { IF ((smindin == 1) | ALLSILENT)
             { state := Msilwait1; }
           ELSE
             { state := Msilwait2; }
         }

Msilwait2: { IF (smindin == 0)
            { IF ALLSILENT
            /* a frame has been transmitted: switch channel frequency */
            { state := Mcfreq1;
              freqchange := CHANGEBOTH;
              advancefreq := 1;
              enablesmtx := 0; /* abort current sm transmission */ }
            ELSE
            { state := Msilwait2; }
            }
           ELSE
             { state := Idle;
             freqacquire := 1;
             enablesmtx := 0;
             enablefreqsel := 0; }
          }

Mcfreq1: { advancefreq := 0;
         state := Mcfreq2; }

Mcfreq2: { advancefreq := 0;
         enablesmtx := 1;
         IF (ALLSILENT & (smindin == 0))
           LOOP;
         ELSE
           state := Msilwait2; }

/* wait till silence goes away, then wait for silence
to change freq, and loop */

Initslave1: { IF (rxvalidframe == 1)
             { state := Initslave2;
               freqacquire := 0;
               freqchange := CHANGEBOTH; }
            ELSE
            { freqacquire := 1;
              LOOP; }
          }

Initslave2: { freqchange := NOCHANGE;
            state := Ssilwait1; }

Ssilwait1: { IF ((smindin == 1) | ALLSILENT)
              { LOOP;
                IF (rxvalidframe == 1)
                { IF (rxcurrfreqmatch == 1)
                    { freqchange := CHANGENEXT; }
                  ELSE
                    { freqchange := CHANGEBOTH; }
                }
              }
            ELSE
              { state := Ssilwait2;
              IF (rxvalidframe == 1)
                { IF (rxcurrfreqmatch == 1)
                    { freqchange := CHANGENEXT; }
                  ELSE
                    { freqchange := CHANGEBOTH; }
                }
              }
            }

Ssilwait2: IF (smindin == 0)
           { IF (ALLSILENT & (rxnextfreqin != 0))
              { state := Scfreq;
                freqchange := CHANGEBOTH; }
            ELSE
            { LOOP;
              IF (rxvalidframe == 1)
              { IF (rxcurrfreqmatch == 1)
                  { freqchange := CHANGENEXT; }
                ELSE
                  { freqchange := CHANGEBOTH; }
              }
            }
          }
         ELSE
           { state := Idle;
             freqacquire := 1;
             enablesmrx := 0; }

Scfreq: { IF (ALLSILENT & (smindin == 0))
          { LOOP;
            IF (rxvalidframe == 1)
            { IF (rxcurrfreqmatch == 1)
                { freqchange := CHANGENEXT; }
              ELSE
                { freqchange := CHANGEBOTH; }
            }
            ELSE
            freqchange := NOCHANGE;
          }
        ELSE
          state := Ssilwait2; }
}
                                      FINESSE
                                   Status Report

File Name  : smctrl.fin
Module Name: smctrl2

/*** Parsing/Synthesis messages     ***/
/*** End Parsing/Synthesis messages ***/

/*** Optimization messages    ***/
/*** End Optimization messages      ***/

/*** Structure Generation messages        ***/

      /*** Testability Information  ***/

      INSTANCE    :  g1#smctrl2#dff
      SCANIN NET  :  f#scanin
      OUTPUT NET  :  enablesmtx

      INSTANCE    :  g2#smctrl2#dff
      SCANIN NET  :  enablesmtx
      OUTPUT NET  :  enablesmrx

      INSTANCE    :  g3#smctrl2#dff
      SCANIN NET  :  enablesmrx
      OUTPUT NET  :  enablefreqsel

      INSTANCE    :  g4#smctrl2
      SCANIN NET  :  enablefreqsel
      OUTPUT NET  :  advancefreq

      INSTANCE    :  g7#smctrl2
      SCANIN NET  :  advancefreq
      OUTPUT NET  :  state(3)

      INSTANCE    :  g0#smctrl2#dff
      SCANIN NET  :  state(3)
      OUTPUT NET  :  freqacquire

      INSTANCE    :  g5#smctrl2
      SCANIN NET  :  freqacquire
      OUTPUT NET  :  state(1)
```

```
      INSTANCE    :  g6#smctrl2
      SCANIN NET  :  state(1)
      OUTPUT NET  :  state(2)

      INSTANCE    :  g8#smctrl2
      SCANIN NET  :  state(2)
      OUTPUT NET  :  state(4)

      INSTANCE    :  g9#smctrl2
      SCANIN NET  :  state(4)
      OUTPUT NET  :  freqchange(0)

      INSTANCE    :  g10#smctrl2
      SCANIN NET  :  freqchange(0)
      OUTPUT NET  :  freqchange(1)

      /*** End Testability Information    ***/

      OUTPUT f#scanout MERGED WITH freqchange(1).
/*** End Structure Generation messages    ***/

/*** Component Summary messages     ***/

Component Summary
  Finesse Name
    Module Name    Count  Size Information

  Control Block
    c#smctrl2      1      37 rows, 22 inputs, 15 outputs    (PLA)
    dffpre         5
    dffclr         2
    jkpre          1
    jkclr          3


External values used for symbolic variables:

  Symbolic          Symbolic           External
  Variable           Value           Boolean Value
  ------------------------------------------------

  state              Idle            1011
                     Initmaster1     1111
                     Initmaster2     0111
                     Msilwait1       0100
                     Msilwait2       0110
                     Mcfreq1         0001
                     Mcfreq2         0000
                     Initslave1      1001
                     Initslave2      0101
                     Ssilwait1       1000
                     Ssilwait2       1110
                     Scfreq          1100

/*** End Component Summary messages ***/
```

A.13 Station Management Receiver

```
#define lvlS ^b110
#define lvl0 ^b000
#define lvl1 ^b001
#define lvl3 ^b010
#define lvl4 ^b011
#define lvl2 ^b100
#define lvlE ^b101
#define ZCLVL ((clvl[2] == 0) & (enablesmrx == 1))
#define CLVL76 (clvl[1:0] == addr[7:6])
#define CLVL54 (clvl[1:0] == addr[5:4])
#define CLVL32 (clvl[1:0] == addr[3:2])
#define CLVL10 (clvl[1:0] == addr[1:0])

FINESSE smrx1

INPUT ~resetsmrx, enablesmrx, adr16or48bit;
INPUT clvl[2:0], addr[7:0], ccardet, parityin;
OUTPUT JKFF currfreq[7:0], nextfreq[7:0];
OUTPUT JKFF addroffset[4:0], cagchold, fcsenable, ~parity;
OUTPUT DFF validframe, state;

SYMBOLIC state { Idle, Pad1, Pad2, Pad3, Pad4, SD1, SD2, SD3, SD4,
FC1, FC2, FC3, FC4,
DA1, DA2, DA3, DA4, DA5, DA6, DA7, DA8, DA9, DA10, DA11, DA12,
DA13, DA14, DA15, DA16, DA17, DA18, DA19, DA20, DA21, DA22, DA23, DA24,
SA1, SA2, SA3, SA4, SA5, SA6, SA7, SA8, SA9, SA10, SA11, SA12,
SA13, SA14, SA15, SA16, SA17, SA18, SA19, SA20, SA21, SA22, SA23, SA24,
CF1, CF2, CF3, CF4, NF1, NF2, NF3, NF4,
FCS1, FCS2, FCS3, FCS4, FCS5, FCS6, FCS7, FCS8,
FCS9, FCS10, FCS11, FCS12, FCS13, FCS14, FCS15, FCS16,
ED1, ED2, ED3, ED4 };

FSM
{

DEFAULT
  { state := Idle;
    validframe := 0;
  }

WHEN resetsmrx RESET
  {
    state = Idle;
    addroffset = 0;
    cagchold = 0;
    currfreq = 0;
    nextfreq = 0;
    parity = 0;
    fcsenable = 0;
    validframe = 0;
  }

Idle: { IF ((enablesmrx == 1) & (clvl == lvl4))
          { state := Pad1;
          cagchold := 0;
          fcsenable := 0;
          parity := 0; }
        ELSE
        { cagchold := 0;
          LOOP; }
      }

Pad1: { IF (clvl == lvl0)
        { state := Pad2; }
      ELSE
        state := Idle;
      }

Pad2: { IF (clvl == lvl4)
        state := Pad3;
      ELSE
        state := Idle; }

Pad3: { IF (clvl == lvl0)
        state := Pad4;
      ELSE
        state := Idle; }

Pad4: { IF (clvl == lvl2)
        state := SD1;
      ELSE
       state := Idle; }

SD1:  { IF (clvl == lvl0)
        state := SD2;
      ELSE
        state := Idle; }

SD2:  { IF (clvl == lvl2)
        { state := SD3;
          fcsenable := 1; }
      ELSE
        state := Idle; }

SD3:  { IF (clvl == lvl0)
        { state := SD4; }
      ELSE
        state := Idle; }

SD4:  { IF ((clvl == lvl3) & (ccardet == 1))
      /* carrier detect must be high before frame is recognized */
        { cagchold := 1;
          state := FC1; }
      ELSE
        state := Idle; }

FC1:  { IF (clvl == lvl0)
        state := FC2;
      ELSE
        state := Idle; }

FC2:  { IF (clvl == lvl1)
        state := FC3;
      ELSE
        state := Idle; }

FC3:  { IF (clvl == lvl4)
        { addroffset := 15;  /* low addr bits first */
          state := FC4; }
      ELSE
        state := Idle;
      }

FC4:  { IF (CLVL10 & ZCLVL)
        state := DA1;
      ELSE
        state := Idle;
      }

DA1:  { IF (CLVL32 & ZCLVL)
        state := DA2;
      ELSE
        state := Idle; }

DA2:  { IF (CLVL54 & ZCLVL)
        state := DA3;
      ELSE
        state := Idle; }

DA3:  { IF (CLVL76 & ZCLVL)
        { addroffset := 16;
          state := DA4; }
      ELSE
        state := Idle; }

DA4:  { IF (CLVL10 & ZCLVL)
        state := DA5;
      ELSE
        state := Idle; }

DA5:  { IF (CLVL32 & ZCLVL)
        state := DA6;
      ELSE
        state := Idle; }

DA6:  { IF (CLVL54 & ZCLVL)
        state := DA7;
      ELSE
        state := Idle; }

DA7:  { IF (CLVL76 & ZCLVL)
        {
            IF (adr16or48bit == 0) /* 16 bit addr */
            { state := DA24;
              addroffset := 9; }
          ELSE
            { state := DA8;
            addroffset := 17; }
  }
      ELSE
        state := Idle;
      }

DA8:  { IF (CLVL10 & ZCLVL)
        state := DA9;
      ELSE
        state := Idle; }

DA9:  { IF (CLVL32 & ZCLVL)
        state := DA10;
      ELSE
        state := Idle; }

DA10: { IF (CLVL54 & ZCLVL)
        state := DA11;
      ELSE
         state := Idle; }

DA11: { IF (CLVL76 & ZCLVL)
       { addroffset := 18;
         state := DA12; }
      ELSE
        state := Idle; }

DA12: { IF (CLVL10 & ZCLVL)
```

```
        state := DA13;
      ELSE
        state := Idle; }

DA13: { IF (CLVL32 & ZCLVL)
        state := DA14;
      ELSE
        state := Idle; }

DA14: { IF (CLVL54 & ZCLVL)
        state := DA15;
      ELSE
        state := Idle; }

DA15: { IF (CLVL76 & ZCLVL)
        { addroffset := 19;
          state := DA16; }
      ELSE
        state := Idle; }

DA16: { IF (CLVL10 & ZCLVL)
        state := DA17;
      ELSE
        state := Idle; }

DA17: { IF (CLVL32 & ZCLVL)
        state := DA18;
      ELSE
        state := Idle; }

DA18: { IF (CLVL54 & ZCLVL)
        state := DA19;
      ELSE
        state := Idle; }

DA19: { IF (CLVL76 & ZCLVL)
        { addroffset := 20;
          state := DA20; }
      ELSE
        state := Idle; }

DA20: { IF (CLVL10 & ZCLVL)
        state := DA21;
      ELSE
        state := Idle; }

DA21: { IF (CLVL32 & ZCLVL)
        state := DA22;
      ELSE
        state := Idle; }

DA22: { IF (CLVL54 & ZCLVL)
        state := DA23;
      ELSE
        state := Idle; }

DA23: { IF (CLVL76 & ZCLVL)
        { addroffset := 9;  /* high addr bits first */
          state := DA24; }
      ELSE
        state := Idle;
       }

DA24: { IF (CLVL10 & ZCLVL)
        state := SA1;
      ELSE
        state := Idle; }

SA1:  { IF (CLVL32 & ZCLVL)
        state := SA2;
      ELSE
        state := Idle; }

SA2:  { IF (CLVL54 & ZCLVL)
        state := SA3;
      ELSE
        state := Idle; }

SA3:  { IF (CLVL76 & ZCLVL)
        { addroffset := 10;
          state := SA4; }
      ELSE
        state := Idle; }


SA4:  { IF (CLVL10 & ZCLVL)
        state := SA5;
      ELSE
        state := Idle; }

SA5:  { IF (CLVL32 & ZCLVL)
        state := SA6;
      ELSE
        state := Idle; }

SA6:  { IF (CLVL54 & ZCLVL)
        state := SA7;
      ELSE
        state := Idle; }

SA7:  { IF (CLVL76 & ZCLVL)
          { IF (adr16or48bit == 0) /* 16 bit addr */
            { state := SA24; }
          ELSE
            { state := SA8;
              addroffset := 11; }
        }
      ELSE
        state := Idle; }

SA8:  { IF (CLVL10 & ZCLVL)
        state := SA9;
      ELSE
        state := Idle; }

SA9:  { IF (CLVL32 & ZCLVL)
        state := SA10;
      ELSE
        state := Idle; }

SA10: { IF (CLVL54 & ZCLVL)
        state := SA11;
      ELSE
        state := Idle; }

SA11:  { IF (CLVL76 & ZCLVL)
         { addroffset := 12;
           state := SA12; }
      ELSE
        state := Idle; }

SA12:  { IF ( CLVL10 & ZCLVL)
         state := SA13;
       ELSE
         state := Idle; }

SA13:  { IF (CLVL32 & ZCLVL)
         state := SA14;
       ELSE
         state := Idle; }

SA14:  { IF (CLVL54 & ZCLVL)
         state := SA15;
       ELSE
         state := Idle; }

SA15:  { IF (CLVL76 & ZCLVL)
         { addroffset := 13;
           state := SA16; }
       ELSE
         state := Idle; }

SA16:  { IF (CLVL10 & ZCLVL)
         state := SA17;
       ELSE
         state := Idle; }

SA17:  { IF (CLVL32 & ZCLVL)
         state := SA18;
       ELSE
         state := Idle; }

SA18:  { IF (CLVL54 & ZCLVL)
         state := SA19;
       ELSE
         state := Idle; }

SA19:  { IF (CLVL76 & ZCLVL)
         { addroffset := 14;
           state := SA20; }
       ELSE
         state := Idle; }

SA20:  { IF (CLVL10 & ZCLVL)
         state := SA21;
       ELSE
         state := Idle; }

SA21:  { IF (CLVL32 & ZCLVL)
         state := SA22;
       ELSE
         state := Idle; }

SA22:  { IF (CLVL54 & ZCLVL)
         state := SA23;
       ELSE
         state := Idle; }

SA23:  { IF (CLVL76 & ZCLVL)
         state := SA24;
       ELSE
         state := Idle; }

SA24: { IF ZCLVL
        { currfreq[1:0] := clvl[1:0];
          state := CF1; }
      ELSE
        state := Idle; }

CF1:  { IF ZCLVL
        { currfreq[3:2] := clvl[1:0];
          state := CF2; }
      ELSE
        state := Idle; }

CF2:  { IF ZCLVL
        { currfreq[5:4] := clvl[1:0];
          state := CF3; }
      ELSE
        state := Idle; }

CF3:  { IF ZCLVL
        { currfreq[7:6] := clvl[1:0];
          state := CF4; }
      ELSE
        state := Idle; }

CF4:  { IF ZCLVL
        { nextfreq[1:0] := clvl[1:0];
          state := NF1; }
      ELSE
        state := Idle; }

NF1:  { IF ZCLVL
        { nextfreq[3:2] := clvl[1:0];
          state := NF2; }
      ELSE
        state := Idle; }

NF2:  { IF ZCLVL
        { nextfreq[5:4] := clvl[1:0];
          state := NF3; }
      ELSE
        state := Idle; }

NF3:  { IF ZCLVL
        { nextfreq[7:6] := clvl[1:0];
          state := NF4; }
      ELSE
        state := Idle; }

NF4:  { IF ZCLVL
        state := FCS1;
      ELSE
        state := Idle; }

FCS1:  { IF ZCLVL
         state := FCS2;
       ELSE
         state := Idle; }

FCS2:  { IF ZCLVL
         state := FCS3;
       ELSE
         state := Idle; }
```

```
FCS3:  { IF ZCLVL
         state := FCS4;
       ELSE
         state := Idle; }

FCS4:  { IF ZCLVL
         state := FCS5;
       ELSE
         state := Idle; }

FCS5:  { IF ZCLVL
         state := FCS6;
       ELSE
         state := Idle; }

FCS6:  { IF ZCLVL
         { state := FCS7; }
       ELSE
         state := Idle; }

FCS7:  { IF ZCLVL
         { state := FCS8; }
       ELSE
         state := Idle; }

FCS8:  { IF ZCLVL
         { state := FCS9; }
       ELSE
         state := Idle; }

FCS9:  { IF ZCLVL
         { state := FCS10; }
       ELSE
         state := Idle; }

FCS10:  { IF ZCLVL
          { state := FCS11; }
        ELSE
          state := Idle; }

FCS11:  { IF ZCLVL
          { state := FCS12; }
        ELSE
          state := Idle; }

FCS12:  { IF ZCLVL
          { state := FCS13; }
        ELSE
          state := Idle; }

FCS13:  { IF ZCLVL
          { state := FCS14; }
        ELSE
          state := Idle; }

FCS14:  { IF ZCLVL
          { state := FCS15; }
        ELSE
          state := Idle; }

FCS15:  { IF ZCLVL
          { parity := 1; /* is it done here? */
            state := FCS16; }
        ELSE
          state := Idle; }

FCS16: { IF ((parityin == 0) & (clvl == lvl2)) /* CRC is correct */
         { fcsenable := 0;
           parity := 0;
           state := ED1; }
       ELSE
         state := Idle; }

ED1:  { IF (clvl == lvl1)
        state := ED2;
      ELSE
        state := Idle; }

ED2:  { IF (clvl == lvl2)
        state := ED3;
      ELSE
        state := Idle; }

ED3:  { IF (clvl == lvl0)
        state := ED4;
      ELSE
        state := Idle; }

ED4:  { IF (clvl == lvlS)
        { state := Idle;
          validframe := 1; }
      ELSE
        state := Idle; }

}

                                    FINESSE
                                 Status Report

File Name  : smrx.fin
Module Name: smrx1

/*** Parsing/Synthesis messages     ***/
/*** End Parsing/Synthesis messages ***/

/*** Optimization messages    ***/
/*** End Optimization messages      ***/

/*** Structure Generation messages        ***/
      K of JKFF currfreq(0) is GND.
      K of JKFF currfreq(1) is GND.
      K of JKFF currfreq(2) is GND.
      K of JKFF currfreq(3) is GND.
      K of JKFF currfreq(4) is GND.
      K of JKFF currfreq(5) is GND.
      K of JKFF currfreq(6) is GND.
      K of JKFF currfreq(7) is GND.
      K of JKFF nextfreq(0) is GND.
      K of JKFF nextfreq(1) is GND.
      K of JKFF nextfreq(2) is GND.
      K of JKFF nextfreq(3) is GND.
      K of JKFF nextfreq(4) is GND.
      K of JKFF nextfreq(5) is GND.
      K of JKFF nextfreq(6) is GND.
      K of JKFF nextfreq(7) is GND.

      /*** Testability Information  ***/

      INSTANCE    :  g0#smrx1#dff
      SCANIN NET  :  f#scanin
      OUTPUT NET  :  currfreq(0)

      INSTANCE    :  g1#smrx1#dff
      SCANIN NET  :  currfreq(0)
      OUTPUT NET  :  currfreq(1)

      INSTANCE    :  g2#smrx1#dff
      SCANIN NET  :  currfreq(1)
      OUTPUT NET  :  currfreq(2)

      INSTANCE    :  g3#smrx1#dff
      SCANIN NET  :  currfreq(2)
      OUTPUT NET  :  currfreq(3)

      INSTANCE    :  g4#smrx1#dff
      SCANIN NET  :  currfreq(3)
      OUTPUT NET  :  currfreq(4)

      INSTANCE    :  g5#smrx1#dff
      SCANIN NET  :  currfreq(4)
      OUTPUT NET  :  currfreq(5)

      INSTANCE    :  g6#smrx1#dff
      SCANIN NET  :  currfreq(5)
      OUTPUT NET  :  currfreq(6)

      INSTANCE    :  g7#smrx1#dff
      SCANIN NET  :  currfreq(6)
      OUTPUT NET  :  currfreq(7)

      INSTANCE    :  g8#smrx1#dff
      SCANIN NET  :  currfreq(7)
      OUTPUT NET  :  nextfreq(0)

      INSTANCE    :  g9#smrx1#dff
      SCANIN NET  :  nextfreq(0)
      OUTPUT NET  :  nextfreq(1)

      INSTANCE    :  g10#smrx1#dff
      SCANIN NET  :  nextfreq(1)
      OUTPUT NET  :  nextfreq(2)

      INSTANCE    :  g11#smrx1#dff
      SCANIN NET  :  nextfreq(2)
      OUTPUT NET  :  nextfreq(3)

      INSTANCE    :  g12#smrx1#dff
      SCANIN NET  :  nextfreq(3)
      OUTPUT NET  :  nextfreq(4)

      INSTANCE    :  g13#smrx1#dff
      SCANIN NET  :  nextfreq(4)
      OUTPUT NET  :  nextfreq(5)

      INSTANCE    :  g14#smrx1#dff
      SCANIN NET  :  nextfreq(5)
      OUTPUT NET  :  nextfreq(6)

      INSTANCE    :  g15#smrx1#dff
      SCANIN NET  :  nextfreq(6)
      OUTPUT NET  :  nextfreq(7)

      INSTANCE    :  g16#smrx1#dff
      SCANIN NET  :  nextfreq(7)
      OUTPUT NET  :  addroffset(0)

      INSTANCE    :  g17#smrx1#dff
      SCANIN NET  :  addroffset(0)
      OUTPUT NET  :  addroffset(1)

      INSTANCE    :  g18#smrx1#dff
      SCANIN NET  :  addroffset(1)
      OUTPUT NET  :  addroffset(2)

      INSTANCE    :  g19#smrx1#dff
      SCANIN NET  :  addroffset(2)
      OUTPUT NET  :  addroffset(3)

      INSTANCE    :  g20#smrx1#dff
      SCANIN NET  :  addroffset(3)
      OUTPUT NET  :  addroffset(4)

      INSTANCE    :  g21#smrx1#dff
      SCANIN NET  :  addroffset(4)
      OUTPUT NET  :  cagchold

      INSTANCE    :  g22#smrx1#dff
      SCANIN NET  :  cagchold
      OUTPUT NET  :  fcsenable

      INSTANCE    :  g24#smrx1
      SCANIN NET  :  fcsenable
      OUTPUT NET  :  validframe

      INSTANCE    :  g25#smrx1
      SCANIN NET  :  validframe
      OUTPUT NET  :  state(1)

      INSTANCE    :  g26#smrx1
      SCANIN NET  :  state(1)
      OUTPUT NET  :  state(2)

      INSTANCE    :  g27#smrx1
      SCANIN NET  :  state(2)
      OUTPUT NET  :  state(3)

      INSTANCE    :  g28#smrx1
      SCANIN NET  :  state(3)
      OUTPUT NET  :  state(4)

      INSTANCE    :  g29#smrx1
      SCANIN NET  :  state(4)
      OUTPUT NET  :  state(5)

      INSTANCE    :  g30#smrx1
      SCANIN NET  :  state(5)
      OUTPUT NET  :  state(6)

      INSTANCE    :  g31#smrx1
      SCANIN NET  :  state(6)
      OUTPUT NET  :  state(7)

      INSTANCE    :  g23#smrx1#dff
      SCANIN NET  :  state(7)
      OUTPUT NET  :  parity
```

```
     /*** End Testability Information    ***/

     OUTPUT f#scanout MERGED WITH parity.
/*** End Structure Generation messages    ***/

/*** Component Summary messages     ***/

Component Summary
  Finesse Name
    Module Name    Count  Size Information

  Control Block
    c#smrx1        1      189 rows, 22 inputs, 37 outputs   (PLA)
    dffclr         8
    jkpre          1
    jkclr          23


External values used for symbolic variables:

  Symbolic          Symbolic           External
  Variable           Value           Boolean Value
  ------------------------------------------------

  state              Idle            0000000
                     Pad1            1000000
                     Pad2            0100000
                     Pad3            1100000
                     Pad4            0010000
                     SD1             1010000
                     SD2             0110000
                     SD3             1110000
                     SD4             0001000
                     FC1             1001000
                     FC2             0101000
                     FC3             1101000
                     FC4             0011000
                     DA1             1011000
                     DA2             0111000
                     DA3             1111000
                     DA4             0000100
                     DA5             1000100
                     DA6             0100100
                     DA7             1100100
                     DA8             0010100
                     DA9             1010100
                     DA10            0110100
                     DA11            1110100
                     DA12            0001100
                     DA13            1001100
                     DA14            0101100
                     DA15            1101100
                     DA16            0011100
                     DA17            1011100
                     DA18            0111100
                     DA19            1111100
                     DA20            0000010
                     DA21            1000010
                     DA22            0100010
                     DA23            1100010
                     DA24            0010010
                     SA1             1010010
                     SA2             0110010
                     SA3             1110010
                     SA4             0001010
                     SA5             1001010
                     SA6             0101010
                     SA7             1101010
                     SA8             0011010
                     SA9             1011010
                     SA10            0111010
                     SA11            1111010
                     SA12            0000110
                     SA13            1000110
                     SA14            0100110
                     SA15            1100110
                     SA16            0010110
                     SA17            1010110
                     SA18            0110110
                     SA19            1110110
                     SA20            0001110
                     SA21            1001110
                     SA22            0101110
                     SA23            1101110
                     SA24            0011110
                     CF1             1011110
                     CF2             0111110
                     CF3             1111110
                     CF4             0000001
                     NF1             1000001
                     NF2             0100001
                     NF3             1100001
                     NF4             0010001
                     FCS1            1010001
                     FCS2            0110001
                     FCS3            1110001
                     FCS4            0001001
                     FCS5            1001001
                     FCS6            0101001
                     FCS7            1101001
                     FCS8            0011001
                     FCS9            1011001
                     FCS10           0111001
                     FCS11           1111001
                     FCS12           0000101
                     FCS13           1000101
                     FCS14           0100101
                     FCS15           1100101
                     FCS16           0010101
                     ED1             1010101
                     ED2             0110101
                     ED3             1110101
                     ED4             0001101

/*** End Component Summary messages ***/
```

A.14 Station Management Transmitter

```
#define impS ^b110
#define imp0 ^b000
#define imp1 ^b001
#define imp3 ^b010
#define imp4 ^b011
#define imp2 ^b100
#define impE ^b101
#define ZCIMP cimp[2] := 0
#define CIMP76 cimp[1:0] := addr[7:6]
#define CIMP54 cimp[1:0] := addr[5:4]
#define CIMP32 cimp[1:0] := addr[3:2]
#define CIMP10 cimp[1:0] := addr[1:0]

FINESSE smtx1

INPUT ~resetsmtx, enablesmtx, adr16or48bit;
INPUT currfreq[7:0], nextfreq[7:0], addr[7:0];
OUTPUT DFF cimp[2:0];
OUTPUT JKFF addroffset[4:0], txendofframe, fcsenable, ~parity;
OUTPUT DFF state;

SYMBOLIC state { Idle, Pad1, Pad2, Pad3, Pad4, SD1, SD2, SD3, SD4,
FC1, FC2, FC3, FC4,
DA1, DA2, DA3, DA4, DA5, DA6, DA7, DA8, DA9, DA10, DA11, DA12,
DA13, DA14, DA15, DA16, DA17, DA18, DA19, DA20, DA21, DA22, DA23, DA24,
SA1, SA2, SA3, SA4, SA5, SA6, SA7, SA8, SA9, SA10, SA11, SA12,
SA13, SA14, SA15, SA16, SA17, SA18, SA19, SA20, SA21, SA22, SA23, SA24,
CF1, CF2, CF3, CF4, NF1, NF2, NF3, NF4,
FCS1, FCS2, FCS3, FCS4, FCS5, FCS6, FCS7, FCS8,
FCS9, FCS10, FCS11, FCS12, FCS13, FCS14, FCS15, FCS16,
ED1, ED2, ED3, ED4 };

FSM
{

DEFAULT
  { state := Idle;
    txendofframe := 0;
    cimp := impS;
  }

WHEN resetsmtx RESET
  {
    state = Idle;
    cimp = impS;
    addroffset = 0;
    txendofframe = 0;
    parity = 0;
    fcsenable = 0;
  }

Idle: { IF (enablesmtx == 1)
          { state := Pad1;
          fcsenable := 0;
          parity := 0;
          cimp := imp4;}
        ELSE
        { cimp := impS;
            LOOP; }
      }

Pad1: { cimp := imp0;
      state := Pad2; }

Pad2: { cimp := imp4;
      state := Pad3; }

Pad3: { cimp := imp0;
      state := Pad4; }

Pad4: { cimp := imp2;
      state := SD1; }

SD1:  { cimp := imp0;
      state := SD2; }

SD2:  { cimp := imp2;
      state := SD3; }

SD3:  { cimp := imp0;
      fcsenable := 1; /* or is it in next stage? */
      state := SD4; }

SD4:  { cimp := imp3;
      state := FC1; }

FC1:  { cimp := imp0;
      state := FC2; }

FC2:  { cimp := imp1;
      state := FC3; }

FC3:  { cimp := imp4;
      addroffset := 15;  /* low addr bits first */
        state := FC4;
      }

FC4:  { CIMP10;
      ZCIMP;
      state := DA1; }

DA1:  { CIMP32;
      ZCIMP;
      state := DA2; }

DA2:  { CIMP54;
      ZCIMP;
      state := DA3; }

DA3:  { CIMP76;
      ZCIMP;
      addroffset := 16;
      state := DA4; }

DA4:  { CIMP10;
      ZCIMP;
      state := DA5; }

DA5:  { CIMP32;
      ZCIMP;
      state := DA6; }

DA6:  { CIMP54;
      ZCIMP;
      state := DA7; }

DA7:  { CIMP76;
      ZCIMP;
        IF (adr16or48bit == 0) /* 16 bit addr */
        { state := DA24;
          addroffset := 9; }
      ELSE
        { state := DA8;
        addroffset := 17; }
      }
```

```
DA8:  { CIMP10;
      ZCIMP;
      state := DA9; }

DA9:  { CIMP32;
      ZCIMP;
      state := DA10; }

DA10:  { CIMP54;
      ZCIMP;
      state := DA11; }

DA11:  { CIMP76;
      ZCIMP;
      addroffset := 18;
      state := DA12; }

DA12:  { CIMP10;
      ZCIMP;
      state := DA13; }

DA13:  { CIMP32;
      ZCIMP;
      state := DA14; }

DA14:  { CIMP54;
      ZCIMP;
      state := DA15; }

DA15:  { CIMP76;
      ZCIMP;
      addroffset := 19;
      state := DA16; }

DA16:  { CIMP10;
      ZCIMP;
      state := DA17; }

DA17:  { CIMP32;
      ZCIMP;
      state := DA18; }

DA18:  { CIMP54;
      ZCIMP;
      state := DA19; }

DA19:  { CIMP76;
      ZCIMP;
      addroffset := 20;
      state := DA20; }

DA20:  { CIMP10;
      ZCIMP;
      state := DA21; }

DA21:  { CIMP32;
      ZCIMP;
      state := DA22; }

DA22:  { CIMP54;
      ZCIMP;
      state := DA23; }

DA23:  { CIMP76;
      ZCIMP;
      addroffset := 9;  /* high addr bits first */
      state := DA24;
       }

DA24:  { CIMP10;
      ZCIMP;
      state := SA1; }

SA1:  { CIMP32;
      ZCIMP;
      state := SA2; }

SA2:  { CIMP54;
      ZCIMP;
      state := SA3; }

SA3:  { CIMP76;
      ZCIMP;
      addroffset := 10;
      state := SA4; }

SA4:  { CIMP10;
      ZCIMP;
      state := SA5; }

SA5:  { CIMP32;
      ZCIMP;
      state := SA6; }

SA6:  { CIMP54;
      ZCIMP;
      state := SA7; }

SA7:  { CIMP76;
      ZCIMP;
        IF (adr16or48bit == 0) /* 16 bit addr */
        { state := SA24; }
      ELSE
        { state := SA8;
        addroffset := 11; }
      }

SA8:  { CIMP10;
      ZCIMP;
      state := SA9; }

SA9:  { CIMP32;
      ZCIMP;
      state := SA10; }

SA10:  { CIMP54;
      ZCIMP;
      state := SA11; }

SA11:  { CIMP76;
      ZCIMP;
      addroffset := 12;
      state := SA12; }

SA12:  { CIMP10;
      ZCIMP;
      state := SA13; }

SA13:  { CIMP32;
      ZCIMP;
      state := SA14; }

SA14:  { CIMP54;
      ZCIMP;
      state := SA15; }

SA15:  { CIMP76;
      ZCIMP;
      addroffset := 13;
      state := SA16; }

SA16:  { CIMP10;
      ZCIMP;
      state := SA17; }

SA17:  { CIMP32;
      ZCIMP;
      state := SA18; }

SA18:  { CIMP54;
      ZCIMP;
      state := SA19; }

SA19:  { CIMP76;
      ZCIMP;
      addroffset := 14;
      state := SA20; }

SA20:  { CIMP10;
      ZCIMP;
      state := SA21; }

SA21:  { CIMP32;
      ZCIMP;
      state := SA22; }

SA22:  { CIMP54;
      ZCIMP;
      state := SA23; }

SA23:  { CIMP76;
      ZCIMP;
      state := SA24;
       }

SA24: { cimp[1:0] := currfreq[1:0];
      ZCIMP;
      state := CF1; }

CF1:  { cimp[1:0] := currfreq[3:2];
      ZCIMP;
      IF (enablesmtx == 1)
        state := CF2;
      ELSE
        state := Idle; }

CF2:  { cimp[1:0] := currfreq[5:4];
      ZCIMP;
      IF (enablesmtx == 1)
        state := CF3;
      ELSE
        state := Idle; }

CF3:  { cimp[1:0] := currfreq[7:6];
      ZCIMP;
      IF (enablesmtx == 1)
        state := CF4;
      ELSE
        state := Idle; }

CF4:  { cimp[1:0] := nextfreq[1:0];
      ZCIMP;
      IF (enablesmtx == 1)
        state := NF1;
      ELSE
        state := Idle; }

NF1:  { cimp[1:0] := nextfreq[3:2];
      ZCIMP;
      IF (enablesmtx == 1)
        state := NF2;
      ELSE
        state := Idle; }

NF2:  { cimp[1:0] := nextfreq[5:4];
      ZCIMP;
      IF (enablesmtx == 1)
        state := NF3;
      ELSE
        state := Idle; }

NF3:  { cimp[1:0] := nextfreq[7:6];
      ZCIMP;
      IF (enablesmtx == 1)
        state := NF4;
      ELSE
        state := Idle; }

NF4:  { ZCIMP;
      parity := 1;
      IF (enablesmtx == 1)
        state := FCS1;
      ELSE
        state := Idle; }

FCS1: { ZCIMP;
      IF (enablesmtx == 1)
        state := FCS2;
      ELSE
        state := Idle; }

FCS2: { ZCIMP;
      IF (enablesmtx == 1)
        state := FCS3;
      ELSE
        state := Idle; }

FCS3: { ZCIMP;
      IF (enablesmtx == 1)
        state := FCS4;
      ELSE
        state := Idle; }

FCS4: { ZCIMP;
      IF (enablesmtx == 1)
        state := FCS5;
```

```
      ELSE
        state := Idle; }

FCS5: { ZCIMP;
      IF (enablesmtx == 1)
        state := FCS6;
      ELSE
        state := Idle; }

FCS6: { ZCIMP;
      IF (enablesmtx == 1)
        state := FCS7;
      ELSE
        state := Idle; }

FCS7: { ZCIMP;
      IF (enablesmtx == 1)
        state := FCS8;
      ELSE
        state := Idle; }

FCS8: { ZCIMP;
      IF (enablesmtx == 1)
        state := FCS9;
      ELSE
        state := Idle; }

FCS9: { ZCIMP;
      IF (enablesmtx == 1)
        state := FCS10;
      ELSE
        state := Idle; }

FCS10: { ZCIMP;
      IF (enablesmtx == 1)
        state := FCS11;
      ELSE
        state := Idle; }

FCS11: { ZCIMP;
      IF (enablesmtx == 1)
        state := FCS12;
      ELSE
        state := Idle; }

FCS12: { ZCIMP;
      IF (enablesmtx == 1)
        state := FCS13;
      ELSE
        state := Idle; }

FCS13: { ZCIMP;
      IF (enablesmtx == 1)
        state := FCS14;
      ELSE
        state := Idle; }

FCS14: { ZCIMP;
      IF (enablesmtx == 1)
        state := FCS15;
      ELSE
        state := Idle; }

FCS15: { ZCIMP;
      IF (enablesmtx == 1)
        state := FCS16;
      ELSE
        state := Idle; }

FCS16: { cimp := imp2;
      parity := 0;
      fcsenable := 0;
      IF (enablesmtx == 1)
        state := ED1;
      ELSE
        state := Idle; }

ED1:  { cimp := imp1;
      state := ED2; }

ED2:  { cimp := imp2;
      state := ED3; }

ED3:  { cimp := imp0;
      state := ED4; }

ED4:  { cimp := impS;
      txendofframe := 1;
      state := Idle; }

}
```

```
                                      FINESSE
                                   Status Report

File Name  : smtx.fin
Module Name: smtx1

/*** Parsing/Synthesis messages     ***/
/*** End Parsing/Synthesis messages ***/

/*** Optimization messages    ***/
/*** End Optimization messages      ***/

/*** Structure Generation messages        ***/

      /*** Testability Information  ***/

      INSTANCE    :  g0#smtx1
      SCANIN NET  :  f#scanin
      OUTPUT NET  :  cimp(0)

      INSTANCE    :  g3#smtx1#dff
      SCANIN NET  :  cimp(0)
      OUTPUT NET  :  addroffset(0)

      INSTANCE    :  g4#smtx1#dff
      SCANIN NET  :  addroffset(0)
      OUTPUT NET  :  addroffset(1)

      INSTANCE    :  g5#smtx1#dff
      SCANIN NET  :  addroffset(1)
      OUTPUT NET  :  addroffset(2)

      INSTANCE    :  g6#smtx1#dff
      SCANIN NET  :  addroffset(2)
      OUTPUT NET  :  addroffset(3)

      INSTANCE    :  g7#smtx1#dff
      SCANIN NET  :  addroffset(3)
      OUTPUT NET  :  addroffset(4)

      INSTANCE    :  g8#smtx1#dff
      SCANIN NET  :  addroffset(4)
      OUTPUT NET  :  txendofframe

      INSTANCE    :  g9#smtx1#dff
      SCANIN NET  :  txendofframe
      OUTPUT NET  :  fcsenable

      INSTANCE    :  g11#smtx1
      SCANIN NET  :  fcsenable
      OUTPUT NET  :  state(1)

      INSTANCE    :  g12#smtx1
      SCANIN NET  :  state(1)
      OUTPUT NET  :  state(2)

      INSTANCE    :  g13#smtx1
      SCANIN NET  :  state(2)
      OUTPUT NET  :  state(3)

      INSTANCE    :  g14#smtx1
      SCANIN NET  :  state(3)
      OUTPUT NET  :  state(4)

      INSTANCE    :  g15#smtx1
      SCANIN NET  :  state(4)
      OUTPUT NET  :  state(5)

      INSTANCE    :  g1#smtx1
      SCANIN NET  :  state(5)
      OUTPUT NET  :  cimp(1)

      INSTANCE    :  g2#smtx1
      SCANIN NET  :  cimp(1)
      OUTPUT NET  :  cimp(2)

      INSTANCE    :  g10#smtx1#dff
      SCANIN NET  :  cimp(2)
      OUTPUT NET  :  parity

      INSTANCE    :  g16#smtx1
      SCANIN NET  :  parity
      OUTPUT NET  :  state(6)

      INSTANCE    :  g17#smtx1
      SCANIN NET  :  state(6)
      OUTPUT NET  :  state(7)

      /*** End Testability Information    ***/

      OUTPUT f#scanout MERGED WITH state(7).
/*** End Structure Generation messages    ***/

/*** Component Summary messages     ***/

Component Summary
  Finesse Name
    Module Name    Count  Size Information

  Control Block
    c#smtx1        1      107 rows, 33 inputs, 23 outputs   (PLA)
    dffpre         4
    dffclr         6
    jkpre          1
    jkclr          7


External values used for symbolic variables:

  Symbolic          Symbolic          External
  Variable           Value          Boolean Value
  ---------------------------------------------

  state              Idle           1100000
                     Pad1           1100100
                     Pad2           1101100
                     Pad3           0010100
                     Pad4           1110000
                     SD1            1110100
                     SD2            1111101
                     SD3            0011101
                     SD4            0101011
                     FC1            0111001
                     FC2            1111001
                     FC3            0011001
                     FC4            0000111
                     DA1            1100011
                     DA2            0110100
                     DA3            1001111
                     DA4            0010011
                     DA5            1101011
                     DA6            0101101
                     DA7            1001001
                     DA8            0010111
                     DA9            1101110
                     DA10           0111100
                     DA11           1000101
                     DA12           0001011
                     DA13           1111110
                     DA14           0111111
                     DA15           1000001
                     DA16           0001111
                     DA17           1110111
                     DA18           0100100
                     DA19           1000010
                     DA20           0000011
                     DA21           1110110
                     DA22           0110111
                     DA23           1001000
                     DA24           0001110
                     SA1            1101111
                     SA2            0101110
                     SA3            1000011
                     SA4            0011111
                     SA5            1100111
                     SA6            0111110
                     SA7            1000110
                     SA8            0010110
                     SA9            1100110
                     SA10           0100111
                     SA11           1001110
                     SA12           0011011
                     SA13           1110011
                     SA14           0101111
```

```
                    SA15            1001010
                    SA16            0011110
                    SA17            1111111
                    SA18            0100110
                    SA19            1000111
                    SA20            0000110
                    SA21            1111011
                    SA22            0110110
                    SA23            1001011
                    SA24            0011100
                    CF1             0010000
                    CF2             0110000
                    CF3             0001000
                    CF4             0011000
                    NF1             0111000
                    NF2             0101010
                    NF3             0101000
                    NF4             0111010
                    FCS1            1011101
                    FCS2            1011111
                    FCS3            1011110
                    FCS4            1010010
                    FCS5            1011010
                    FCS6            1011100
                    FCS7            1010000
                    FCS8            1010111
                    FCS9            1011000
                    FCS10           1010100
                    FCS11           1010101
                    FCS12           1011001
                    FCS13           1010001
                    FCS14           1010011
                    FCS15           1011011
                    FCS16           0000000
                    ED1             0110001
                    ED2             1110001
                    ED3             0010001
                    ED4             0000100

/*** End Component Summary messages ***/
```

A.15 Symbol Counter

```
FINESSE symcntr1
/* 3 bit counter with RCO output */

INPUT enablecntr;
OUTPUT DFF  state;     /* rco = 1 when count = 0 (first symbol) */
OUTPUT COMB rco;

SYMBOLIC state { zero, one, two, three, four, five, six, seven } \
 { ^b000, ^b001, ^b010, ^b011, ^b100, ^b101, ^b110, ^b111 };

FSM
{
DEFAULT
{
  NEXT;
}

WHEN ~enablecntr RESET
  { state = seven;  }

zero:  { rco = 1; }
one:   { rco = 0; }
two:   { rco = 0; }
three: { rco = 0; }
four:  { rco = 0; }
five:  { rco = 0; }
six:   { rco = 0; }
seven: { rco = 0; }
}
```

```
                                    FINESSE
                                 Status Report

File Name  : symcntr.fin
Module Name: symcntr1

/*** Parsing/Synthesis messages     ***/
(WARNING)   INPUT variables were only referenced in RESET section
/*** End Parsing/Synthesis messages ***/

/*** Optimization messages    ***/
/*** End Optimization messages      ***/

/*** Structure Generation messages        ***/

      /*** Testability Information  ***/

      INSTANCE    :  g0#symcntr1
      SCANIN NET  :  f#scanin
      OUTPUT NET  :  state(1)

      INSTANCE    :  g1#symcntr1
      SCANIN NET  :  state(1)
      OUTPUT NET  :  state(2)

      INSTANCE    :  g2#symcntr1
      SCANIN NET  :  state(2)
      OUTPUT NET  :  state(3)

      /*** End Testability Information    ***/

      OUTPUT f#scanout MERGED WITH state(3).
/*** End Structure Generation messages    ***/

/*** Component Summary messages     ***/

Component Summary
  Finesse Name
    Module Name    Count  Size Information

  Control Block
    c#symcntr1     1      6 rows, 3 inputs, 4 outputs (PLA)
    dffpre         3


External values used for symbolic variables:

  Symbolic          Symbolic           External
  Variable           Value           Boolean Value
  ------------------------------------------------

  state              zero            000
                     one             001
                     two             010
                     three           011
                     four            100
                     five            101
                     six             110
                     seven           111

/*** End Component Summary messages ***/
```

A.16 Transmit Clock Alive Detector

```
#define SAMECNT (prevsym == Clkin)
#define CLK0 ^b000

FINESSE txclkaliv1

INPUT ~resettxclkalive, Clkin[2:0];
OUTPUT DFF txclkdead;
INTERNAL DFF prevsym[2:0];
INTERNAL DFF state;

SYMBOLIC state { Clk1, Clk2, Clk3, Clk4, Clk5, Clk6, Clk7, Clk8, Clk9, \
ClkA, ClkB, ClkC, ClkD, ClkE, ClkF } ;

FSM
{

DEFAULT
  { state := Clk1;
    txclkdead := 0;
    prevsym := Clkin;
  }

WHEN resettxclkalive RESET
  {
    state = Clk1;
    txclkdead = 0;
    prevsym = CLK0;
  }

Clk1: { IF SAMECNT
          { state := Clk2; }
        ELSE
          LOOP;
      }

Clk2: { IF SAMECNT
          {  state := Clk3; }
        ELSE
          { state := Clk1; }
      }

Clk3: { IF SAMECNT
          { state := Clk4; }
        ELSE
          { state := Clk1; }
      }

Clk4: { IF SAMECNT
          { state := Clk5; }
        ELSE
          { state := Clk1; }
      }

Clk5: { IF SAMECNT
          { state := Clk6; }
        ELSE
          { state := Clk1; }
      }

Clk6: { IF SAMECNT
          { state := Clk7; }
        ELSE
          { state := Clk1; }
      }

Clk7: { IF SAMECNT
          { state := Clk8; }
        ELSE
          { state := Clk1; }
      }

Clk8: { IF SAMECNT
          { state := Clk9; }
        ELSE
          { state := Clk1; }
      }

Clk9: { IF SAMECNT
          { state := ClkA; }
        ELSE
          { state := Clk1; }
      }

ClkA: { IF SAMECNT
          { state := ClkB; }
        ELSE
          { state := Clk1; }
      }

ClkB: { IF SAMECNT
          { state := ClkC; }
        ELSE
          { state := Clk1; }
      }

ClkC: { IF SAMECNT
          { state := ClkD; }
        ELSE
          { state := Clk1; }
      }

ClkD: { IF SAMECNT
          { state := ClkE; }
        ELSE
          { state := Clk1; }
      }

ClkE: { IF SAMECNT
          { state := ClkF; }
        ELSE
          { state := Clk1; }
      }

ClkF: { IF SAMECNT
          { txclkdead := 1;
```

```
          LOOP; }
       ELSE
         { state := Clk1; }
     }

}
```

```
                                    FINESSE
                                 Status Report

File Name  : txclkalive.fin
Module Name: txclkaliv1

/*** Parsing/Synthesis messages     ***/
/*** End Parsing/Synthesis messages ***/

/*** Optimization messages    ***/
/*** End Optimization messages      ***/

/*** Structure Generation messages        ***/

     /*** Testability Information  ***/

     INSTANCE    :  g0#txclkaliv1
     SCANIN NET  :  f#scanin
     OUTPUT NET  :  prevsym(0)

     INSTANCE    :  g1#txclkaliv1
     SCANIN NET  :  prevsym(0)
     OUTPUT NET  :  prevsym(1)

     INSTANCE    :  g2#txclkaliv1
     SCANIN NET  :  prevsym(1)
     OUTPUT NET  :  prevsym(2)

     INSTANCE    :  g6#txclkaliv1
     SCANIN NET  :  prevsym(2)
     OUTPUT NET  :  state(4)

     INSTANCE    :  g7#txclkaliv1
     SCANIN NET  :  state(4)
     OUTPUT NET  :  txclkdead

     INSTANCE    :  g3#txclkaliv1
     SCANIN NET  :  txclkdead
     OUTPUT NET  :  state(1)

     INSTANCE    :  g4#txclkaliv1
     SCANIN NET  :  state(1)
     OUTPUT NET  :  state(2)

     INSTANCE    :  g5#txclkaliv1
     SCANIN NET  :  state(2)
     OUTPUT NET  :  state(3)

     /*** End Testability Information    ***/

     OUTPUT f#scanout COLLAPSED TO INTERNAL NET state(3).
/*** End Structure Generation messages    ***/

/*** Component Summary messages     ***/

Component Summary
  Finesse Name
    Module Name    Count  Size Information

  Control Block
    c#txclkaliv1    1      43 rows, 10 inputs, 5 outputs    (PLA)
    dffpre         3
    dffclr         5


External values used for symbolic variables:

  Symbolic          Symbolic          External
  Variable           Value           Boolean Value
  ------------------------------------------------

  state              Clk1            0111
                     Clk2            1111
                     Clk3            0011
                     Clk4            1011
                     Clk5            0101
                     Clk6            1101
                     Clk7            0001
                     Clk8            1001
                     Clk9            0110
                     ClkA            1110
                     ClkB            0010
                     ClkC            1010
                     ClkD            0100
                     ClkE            1100
                     ClkF            0000

/*** End Component Summary messages ***/
```

A.17 Command/Data Decoder (Transmitter)

```
#define RESETCMD   ^b111
#define LOOPBCKDIS ^b101
#define ENABLETX   ^b011
#define SM         ^b00-
#define ERR0       ^b010
#define ERR1       ^b100
#define ERR2       ^b110

#define TXReset      (txsym == RESETCMD)
#define TXLoopbckdis (txsym == LOOPBCKDIS)
#define TXEnabletx   (txsym == ENABLETX)
#define TXSM         (txsym == SM)
#define TXErr0       (txsym == ERR0)
#define TXErr1       (txsym == ERR1)
#define TXErr2       (txsym == ERR2)

#define Silence     ^b111
#define TXErr      (TXErr0 | TXErr1 | TXErr2)

#define INTLBK    loopbackenable := 1; extloopbck = 0; intloopbck = 1;
#define EXTLBK   { loopbackenable := 1; extloopbck := 1; intloopbck := 0; }
#define LBKDIS    loopbackenable := 0; extloopbck = 0; intloopbck = 0;
#define TXEN      txenable := 1;

FINESSE txcmd1

INPUT txsym[2:0], loopbcksel1, loopbcksel2, loopbckenabled;
INPUT cardet, ~resetin, ~smreq, silencemodein;
OUTPUT COMB extloopbck, intloopbck, cardetNextloopbck;
OUTPUT DFF smreqout, activatepsctx, ackcmd, nakcmd, ~resetout;
OUTPUT JKFF loopbackenable, ~txenable;
OUTPUT DFF txsymout[2:0], state;

SYMBOLIC state { HResetmode, SResetmode, Loopbckdisablemode, Enabletxmode, \
SMmode, Transmitmode, CMDErrormode };


FSM
{

DEFAULT
 { LOOP;
   txsymout := txsym;
   extloopbck = ((loopbcksel1 == 0) & (loopbcksel2 == 1) & (loopbckenabled ==
1));
   intloopbck = ((loopbcksel1 == 1) & (loopbcksel2 == 0) & (loopbckenabled ==
1));
   cardetNextloopbck = (((loopbcksel1 == 1) | (loopbcksel2 == 0) |
(loopbckenabled == 0)) & (cardet == 0));
   smreqout := 1;
   activatepsctx := 0;
   nakcmd := 0;
   resetout := 0;

 }

WHEN resetin RESET
 {
   state = HResetmode;
   txsymout = Silence;
   txenable = 0;
   smreqout = 1;
   activatepsctx = 0;
   loopbackenable = 1;
   nakcmd = 0;
   ackcmd = 0;
   resetout = 1;

 }

HResetmode: {   /* clear reset signal input lines */
                  resetout := 0;
               state := SResetmode;
             }

SResetmode: {

            IF ((smreq == 1) & TXReset)
            { INTLBK
              resetout := 1;
              ackcmd := 1;
              txenable := 0;
              state := SResetmode; }
            ELSE IF ((smreq == 1) & TXLoopbckdis)
            { LBKDIS
              smreqout := 1;
              ackcmd := 1;
              state := Loopbckdisablemode; }
            ELSE IF ((smreq == 1) & TXEnabletx)
            { TXEN
              smreqout := 1;
              ackcmd := 1;
              state := Enabletxmode; }
            ELSE IF ((smreq == 1) & TXSM)
            { smreqout := 1;
              ackcmd := 1;
              state := SMmode; }
            ELSE IF ((smreq == 1) & TXErr)
            { nakcmd := 1;
              smreqout := 1;
              state := CMDErrormode; }
            ELSE IF (smreq == 0)
            { activatepsctx := 1;
              txsymout := txsym;
              smreqout := 0;
              state := Transmitmode; }
          }

SMmode: {

            IF ((smreq == 1) & TXReset)
            { resetout := 1;
              INTLBK
              ackcmd := 1;
              txenable := 0;
              state := SResetmode; }
            ELSE IF ((smreq == 1) & TXLoopbckdis)
            { LBKDIS
              ackcmd := 1;
              smreqout := 1;
              state := Loopbckdisablemode; }
            ELSE IF ((smreq == 1) & TXEnabletx)
            { TXEN
              smreqout := 1;
              ackcmd := 1;
              state := Enabletxmode; }
            ELSE IF ((smreq == 1) & TXSM)
            { smreqout := 1;
              state := SMmode; }
            ELSE IF ((smreq == 1) & TXErr)
            { nakcmd := 1;
              smreqout := 1;
              state := CMDErrormode; }
            ELSE IF (smreq == 0)
            { activatepsctx := 1;
              txsymout := txsym;
              smreqout := 0;
              state := Transmitmode; }
          }

Enabletxmode: {

            IF ((smreq == 1) & TXReset)
            { resetout := 1;
              INTLBK
              ackcmd := 1;
              txenable := 0;
              state := SResetmode; }
            ELSE IF ((smreq == 1) & TXLoopbckdis)
            { LBKDIS
              smreqout := 1;
              ackcmd := 1;
              state := Loopbckdisablemode; }
            ELSE IF ((smreq == 1) & TXEnabletx)
            { TXEN
```

```
            smreqout := 1;
            state := Enabletxmode; }
          ELSE IF ((smreq == 1) & TXSM)
          { smreqout := 1;
            ackcmd := 1;
            state := SMmode; }
          ELSE IF ((smreq == 1) & TXErr)
          { nakcmd := 1;
            smreqout := 1;
            state := CMDErrormode; }
          ELSE IF (smreq == 0)
          { activatepsctx := 1;
            smreqout := 0;
            txsymout := txsym;
            state := Transmitmode; }
        }

Loopbckdisablemode: {

          IF ((smreq == 1) & TXReset)
          { resetout := 1;
            INTLBK
            ackcmd := 1;
            txenable := 0;
            state := SResetmode; }
          ELSE IF ((smreq == 1) & TXLoopbckdis)
          { LBKDIS
            smreqout := 1;
            state := Loopbckdisablemode; }
          ELSE IF ((smreq == 1) & TXEnabletx)
          { TXEN
            smreqout := 1;
            ackcmd := 1;
            state := Enabletxmode; }
          ELSE IF ((smreq == 1) & TXSM)
          { smreqout := 1;
            ackcmd := 1;
            state := SMmode; }
          ELSE IF ((smreq == 1) & TXErr)
          { nakcmd := 1;
            smreqout := 1;
            state := CMDErrormode; }
          ELSE IF (smreq == 0)
          { activatepsctx := 1;
            smreqout := 0;
            txsymout := txsym;
            state := Transmitmode; }
        }

Transmitmode: {

          IF ((smreq == 1) & (silencemodein == 1) & TXReset)
          { resetout := 1;
            INTLBK
            ackcmd := 1;
            txenable := 0;
            state := SResetmode; }
          ELSE IF ((smreq == 1) &  (silencemodein == 1) & TXLoopbckdis)
          { LBKDIS
            smreqout := 1;
            ackcmd := 1;
            state := Loopbckdisablemode; }
          ELSE IF ((smreq == 1) &  (silencemodein == 1) & TXEnabletx)
          { TXEN
            smreqout := 1;
            ackcmd := 1;
            state := Enabletxmode; }
          ELSE IF ((smreq == 1) &  (silencemodein == 1) & TXSM)
          { smreqout := 1;
            ackcmd := 1;
            state := SMmode; }
          ELSE IF ((smreq == 1) &  (silencemodein == 1) & TXErr)
          { nakcmd := 1;
            smreqout := 1;
            state := CMDErrormode; }
          ELSE IF ((smreq == 1) & (silencemodein == 0))
          { activatepsctx := 1;
            smreqout := 0;
            txsymout := Silence;
            state := Transmitmode; }
          ELSE IF (smreq == 0)
          { activatepsctx := 1;
            smreqout := 0;
            txsymout := txsym;
            state := Transmitmode; }
        }

CMDErrormode: {

          IF ((smreq == 1) & TXReset)
          { resetout := 1;
            INTLBK
            ackcmd := 1;
            txenable := 0;
            state := SResetmode; }
          ELSE IF ((smreq == 1) & TXLoopbckdis)
          { LBKDIS
            smreqout := 1;
            ackcmd := 1;
            state := Loopbckdisablemode; }
          ELSE IF ((smreq == 1) & TXEnabletx)
          { TXEN
            smreqout := 1;
            ackcmd := 1;
            state := Enabletxmode; }
          ELSE IF ((smreq == 1) & TXSM)
          { smreqout := 1;
            ackcmd := 1;
            state := SMmode; }
          ELSE IF ((smreq == 1) & TXErr)
          { nakcmd := 1;
            smreqout := 1;
            state := CMDErrormode; }
          ELSE IF (smreq == 0)
          { activatepsctx := 1;
            smreqout := 0;
            txsymout := txsym;
            state := Transmitmode; }
        }

}

                                    FINESSE
                                Status Report

File Name  : txcmd.fin
Module Name: txcmd1

/*** Parsing/Synthesis messages     ***/
/*** End Parsing/Synthesis messages ***/

/*** Optimization messages    ***/
/*** End Optimization messages      ***/

/*** Structure Generation messages        ***/

      /*** Testability Information  ***/

      INSTANCE    :  g1#txcmd1
      SCANIN NET  :  f#scanin
      OUTPUT NET  :  activatepsctx

      INSTANCE    :  g2#txcmd1
      SCANIN NET  :  activatepsctx
      OUTPUT NET  :  ackcmd

      INSTANCE    :  g3#txcmd1
      SCANIN NET  :  ackcmd
      OUTPUT NET  :  nakcmd

      INSTANCE    :  g4#txcmd1
      SCANIN NET  :  nakcmd
      OUTPUT NET  :  resetout

      INSTANCE    :  g11#txcmd1
      SCANIN NET  :  resetout
      OUTPUT NET  :  state(2)

      INSTANCE    :  g0#txcmd1
      SCANIN NET  :  state(2)
      OUTPUT NET  :  smreqout

      INSTANCE    :  g5#txcmd1#dff
      SCANIN NET  :  smreqout
      OUTPUT NET  :  loopbackenable

      INSTANCE    :  g6#txcmd1#dff
      SCANIN NET  :  loopbackenable
      OUTPUT NET  :  txenable

      INSTANCE    :  g7#txcmd1
      SCANIN NET  :  txenable
      OUTPUT NET  :  txsymout(0)

      INSTANCE    :  g8#txcmd1
      SCANIN NET  :  txsymout(0)
      OUTPUT NET  :  txsymout(1)

      INSTANCE    :  g9#txcmd1
      SCANIN NET  :  txsymout(1)
      OUTPUT NET  :  txsymout(2)

      INSTANCE    :  g10#txcmd1
      SCANIN NET  :  txsymout(2)
      OUTPUT NET  :  state(1)

      INSTANCE    :  g12#txcmd1
      SCANIN NET  :  state(1)
      OUTPUT NET  :  state(3)

      /*** End Testability Information    ***/

      OUTPUT f#scanout MERGED WITH state(3).
/*** End Structure Generation messages    ***/

/*** Component Summary messages     ***/

Component Summary
  Finesse Name
    Module Name    Count  Size Information

  Control Block
    c#txcmd1       1      42 rows, 12 inputs, 17 outputs    (PLA)
    dffpre         6
    dffclr         5
    jkpre          2


External values used for symbolic variables:

  Symbolic          Symbolic           External
  Variable           Value           Boolean Value
  ------------------------------------------------

  state              HResetmode      101
                     SResetmode      100
                     Loopbckdisablemode  000
                     Enabletxmode    110
                     SMmode          011
                     Transmitmode    001
                     CMDErrormode    111

/*** End Component Summary messages ***/
```

A.18 Transmit Disable Silence Detector

```
#define Txsymsil   (txsymin == ^b11-)
#define Impulsesil (impulsein == ^b110)

FINESSE txdissd1

INPUT ~resettxdissd, txsymin[2:0], impulsein[2:0];
OUTPUT DFF silencedetect;
OUTPUT DFF state;

SYMBOLIC state { Idle, disablesildet, waitforsil, delay1, delay2, delay3,
delay4, delay5 };

FSM
{

DEFAULT
  { state := Idle;
    silencedetect := 0;
  }

WHEN resettxdissd RESET
  {
    state = Idle;
    silencedetect = 1;
  }
```

```
Idle: { IF Txsymsil
        { silencedetect := 1;
          LOOP; }
        ELSE
        { silencedetect := 0;
            state := disablesildet; }
      }

disablesildet: { IF Impulsesil
            { silencedetect := 0;
              LOOP; }
          ELSE
            { silencedetect := 0;
            state := waitforsil; }
        }

waitforsil: { IF Impulsesil
            { silencedetect := 0;
              state := delay1; }
            ELSE
            { silencedetect := 0;
              LOOP; }
            }

delay1: state := delay2;

delay2: state := delay3;

delay3: state := delay4;

delay4: state := delay5;

delay5: { IF Txsymsil
         { silencedetect := 1;
           state := Idle; }
       ELSE
         { silencedetect := 0;
           state := disablesildet; }
      }
}
```

```
                                    FINESSE
                                Status Report

File Name  : txdissd.fin
Module Name: txdissd1

/*** Parsing/Synthesis messages     ***/
/*** End Parsing/Synthesis messages ***/

/*** Optimization messages    ***/
/*** End Optimization messages      ***/

/*** Structure Generation messages        ***/

      /*** Testability Information  ***/

      INSTANCE    :  g2#txdissd1
      SCANIN NET  :  f#scanin
      OUTPUT NET  :  state(2)

      INSTANCE    :  g3#txdissd1
      SCANIN NET  :  state(2)
      OUTPUT NET  :  state(3)

      INSTANCE    :  g0#txdissd1
      SCANIN NET  :  state(3)
      OUTPUT NET  :  silencedetect

      INSTANCE    :  g1#txdissd1
      SCANIN NET  :  silencedetect
      OUTPUT NET  :  state(1)

      /*** End Testability Information    ***/

      PORT txsymin(0) IS NOT REQUIRED.
      OUTPUT f#scanout MERGED WITH state(1).
/*** End Structure Generation messages    ***/

/*** Component Summary messages     ***/

Component Summary
  Finesse Name
    Module Name    Count  Size Information

  Control Block
    c#txdissd1     1      9 rows, 8 inputs, 4 outputs (PLA)
    dffpre         2
    dffclr         2


External values used for symbolic variables:

  Symbolic          Symbolic           External
  Variable           Value           Boolean Value
  ------------------------------------------------

  state              Idle            001
                     disablesildet   111
                     waitforsil      000
                     delay1          100
                     delay2          010
                     delay3          110
                     delay4          011
                     delay5          101

/*** End Component Summary messages ***/
```

A.19 Rsetup.sim

```
    VIEw Sheet
    history 50000
    period trace 100000

    # setup clocks

    SCAle TRace Time 1000
    CLOck Period 1000
    FORCe txclk 1 0 -Repeat
    FORCe txclk 0 500 -Repeat
    FORCe rxclk 1 0 -Repeat
    FORCe rxclk 0 500 -Repeat
    CLOck Period 500
    FORCe testclk 1 0 -Repeat
    FORCe testclk 0 250 -Repeat
    CLOck Period 2000
    FORCe impulseclk 1 0 -Repeat
    FORCe impulseclk 0 1000 -Repeat
    FORCe ctrltxclk 1 0 -Repeat
    FORCe ctrltxclk 0 1000 -Repeat
    FORCe ctrlrxclk 1 0 -Repeat
    FORCe ctrlrxclk 0 1000 -Repeat

    #initial input signal levels

    FORCe clevel 6
    FORCe write#read 1
    FORCe txsym 6
    FORCe smreq 1
    FORCe reset 1
    FORCe level 6
    FORCe test 1
    FORCe jabbertimeout 1
    FORCe faultdetect 1
    FORCe datain 0
    FORCe carrierdetect 1
    FORCe ccardet 1
    FORCe addr 0

    # trace signals

    TRAce txclk rxclk impulseclk reset resetout write#read addr datain dataout
    TRAce smreq txsym smind rxsym transmitdisable
    TRAce impulse externalloopback level agchold
    TRAce currfreq ctrlfreq
    TRAce ctxdisable cimpulse cagchold clevel
    TRAce   signature

    #monitor signals

    mon b txsym smreq impulse rxsym smind level
    SAVe STAte state0 -Replace
    template run 2000

    #hard reset

    forc reset 0
    run
    forc reset 1
    run

    #soft reset (to clear error flags in reg. 0 bits 0-2)

    forc txsym 7
    forc smreq 0
    run

    #setup registers

    forc txsym 0
    run; run
    template run 1000
    forc addr 4
    forc datain 44
    run
    forc write#read 0
    run
    forc write#read 1
    run

    #setup freq pool

    forc addr 5
    forc datain aa
    run
    forc write#read 0
    run
    forc write#read 1
    run
    forc addr 6
    forc datain bb
    run
    forc write#read 0
    run
    forc write#read 1
    run
    forc addr 7
    forc datain cc
    run
    forc write#read 0
    run
    forc write#read 1
    run
    forc addr 8
    forc datain dd
    run
    forc write#read 0
    run
    forc write#read 1
    run
    forc addr 9

    #setup source addr

    forc datain 01
    run
    forc write#read 0
    run
    forc write#read 1
    run
    forc addr a
    forc datain 23
    run
    forc write#read 0
    run
    forc write#read 1
    run
    forc addr b
    forc datain 45
    run
    forc write#read 0
    run
    forc write#read 1
    run
    forc addr c
    forc datain 67
    run
    forc write#read 0
    run
    forc write#read 1
    run
    forc addr d
    forc datain 89
```

```
run
forc write#read 0
run
forc write#read 1
run
forc addr e
forc datain ab
run
forc write#read 0
run
forc write#read 1
run

#setup dest. addr

forc addr f
forc datain fe
run
forc write#read 0
run
forc write#read 1
run
forc addr 10
forc datain dc
run
forc write#read 0
run
forc write#read 1
run
forc addr 11
forc datain ba
run
forc write#read 0
run
forc write#read 1
run
forc addr 12
forc datain 98
run
forc write#read 0
run
forc write#read 1
run
forc addr 13
forc datain 76
run
forc write#read 0
run
forc write#read 1
run
forc addr 14
forc datain 54
run
forc write#read 0
run
forc write#read 1
run

#read data back

forc addr 0
run

forc addr 1
run

forc addr 2
run

forc addr 3
run

forc addr 4
run

forc addr 5
run

forc addr 6
run

forc addr 7
run

forc addr 8
run

forc addr 9
run

forc addr a
run

forc addr b
run

forc addr c
run

forc addr d
run

forc addr e
run

forc addr f
run

forc addr 10
run

forc addr 11
run

forc addr 12
run

forc addr 13
run

forc addr 14
run

template run 1000
```

A.20 Rmaster.sim

```
#scale trace time
scale trace time 8000

#set up master mode
forc addr 1
forc datain 80
run
forc write#read 0
run
forc write#read 1
run
forc txsym 5
run; run; run
forc txsym 3
run; run; run
forc smreq 1
forc txsym 6
run 208000
```

A.21 Rslave.sim

```
#scale trace time
scale trace time 8000

#set up slave mode
forc clevel 6
forc addr 1
forc datain 00
run
forc write#read 0
run
forc write#read 1
run
forc txsym 5
run; run; run
forc txsym 3
run; run; run
forc smreq 1
forc txsym 6
run; run

#channel frequency received

template run 2000
forc clevel 3
run
forc clevel 0
run
forc clevel 3
run
forc clevel 0
run
forc clevel 4
run
forc clevel 0
run
forc clevel 4
run
forc clevel 0
run
forc clevel 2
run
forc clevel 0
run
forc clevel 1
run
forc clevel 3
run
forc clevel 2
run
forc clevel 3
run; run; run
forc clevel 0
run
forc clevel 3
run
forc clevel 1
run
forc clevel 3
run
forc clevel 1
run
forc clevel 0
run; run; run
forc clevel 3
run
forc clevel 0
run
forc clevel 2
run
forc clevel 0
run
forc clevel 2
run; run; run; run
forc clevel 3
run
forc clevel 2
run
forc clevel 3
run
forc clevel 2
run; run
forc clevel 0
run; run; run
forc clevel 1
run
forc clevel 2
run
forc clevel 3
run
forc clevel 0
run
forc clevel 3
run
forc clevel 0
run; run
forc clevel 1
run; run
forc clevel 2
run
forc clevel 3
run
forc clevel 0
run
forc clevel 4
run
```

```
forc clevel 1
run
forc clevel 4
run
forc clevel 0
run
forc clevel 6
run
```

A.22 Rmactx.sim

```
#silence (one octet)
template run 1000
forc txsym 6
run 8000
#pad idle symbols (one octet)
forc txsym 2
run 8000
#start delimiter (one octet)
forc txsym 4
run 2000
forc txsym 0
run
forc txsym 4
run 2000
forc txsym 0
run 3000
#mac 'data' -- frame control, addresses, and data
forc txsym 1
run 4000
forc txsym 0
run 4000
forc txsym 1
run 64000
#end delimiter (one octet)
forc txsym 4
run 2000
forc txsym 1
run
forc txsym 4
run 2000
forc txsym 1
run
forc txsym 0
run 2000
#silence (one octet)
forc txsym 6
run 8000
```

A.23 Rmacrx.sim

```
#silence (one octet)
template run 2000
forc level 6
run 8000
#pad idle symbols (one octet)
forc level 3
run
forc level 0
run
forc level 3
run
forc level 0
run
#start delimiter (one octet)
forc level 4
run
forc level 0
run
forc level 4
run
forc level 0
run
#mac 'data' -- frame control, addresses, and data
forc level 3
run
forc level 2
run
forc level 0
run
forc level 2
run 4000
forc level 3
run
forc level 0
run 4000
forc level 1
run 4000
forc level 3
run
forc level 2
run
forc level 0
run
forc level 3
run
forc level 2
run
forc level 3
run
forc level 1
run
forc level 2
run
forc level 1
run
forc level 0
run
forc level 2
run 4000
forc level 1
run
forc level 0
run 4000
forc level 2
run
forc level 1
run
forc level 3
run
forc level 0
run
forc level 2
run
forc level 3
run
forc level 1
run
forc level 0
run
forc level 1
run
forc level 0
run
forc level 1
run
#end delimiter (one octet)
forc level 4
run
forc level 3
run
forc level 4
run
forc level 0
run
#silence (one octet)
forc level 6
run 8000
```

A.24 Rsetuperr.sim

```
    VIEw Sheet
    #history 50000
    period trace 100000

    # setup clocks

    SCAle TRace Time 1000
    CLOck Period 1000
    FORCe txclk 1 0 -Repeat
    FORCe txclk 0 500 -Repeat
    FORCe rxclk 1 0 -Repeat
    FORCe rxclk 0 500 -Repeat
    CLOck Period 500
    FORCe testclk 1 0 -Repeat
    FORCe testclk 0 250 -Repeat
    CLOck Period 2000
    FORCe impulseclk 1 0 -Repeat
    FORCe impulseclk 0 1000 -Repeat
    FORCe ctrltxclk 1 0 -Repeat
    FORCe ctrltxclk 0 1000 -Repeat
    FORCe ctrlrxclk 1 0 -Repeat
    FORCe ctrlrxclk 0 1000 -Repeat

    #initial input signal levels

    FORCe clevel 6
    FORCe write#read 1
    FORCe txsym 6
    FORCe smreq 1
    FORCe reset 1
    FORCe level 6
    FORCe test 1
    FORCe jabbertimeout 1
    FORCe faultdetect 1
    FORCe datain 0
    FORCe carrierdetect 1
    FORCe ccardet 1
    FORCe addr 0

    # trace signals

    TRAce txclk rxclk impulseclk reset resetout write#read addr datain dataout
    TRAce smreq txsym smind rxsym transmitdisable
    TRAce impulse externalloopback level agchold
    trace carrierdetect
    TRAce currfreq ctrlfreq
    TRAce ctxdisable cimpulse cagchold clevel

    #monitor signals

    mon b txsym smreq impulse rxsym smind level
    #SAVe STAte state0 -Replace
    template run 2000

    #hard reset

    forc reset 0
    run
    forc reset 1
    run

    #soft reset (to clear error flags in reg. 0 bits 0-2)

    forc txsym 7
    forc smreq 0
    run

    #setup registers

    forc txsym 0
    run; run
    template run 1000
    forc addr 4
    forc datain 44
    run
    forc write#read 0
    run
    forc write#read 1
    run

    #setup freq pool

    forc addr 5
    forc datain aa
    run
    forc write#read 0
    run
    forc write#read 1
    run
    forc addr 6
    forc datain bb
    run
    forc write#read 0
    run
    forc write#read 1
    run
    forc addr 7
    forc datain cc
    run
    forc write#read 0
    run
    forc write#read 1
```

```
run
forc addr 8
forc datain dd
run
forc write#read 0
run
forc write#read 1
run
forc addr 9

#setup source addr

forc datain 01
run
forc write#read 0
run
forc write#read 1
run
forc addr a
forc datain 23
run
forc write#read 0
run
forc write#read 1
run
forc addr b
forc datain 45
run
forc write#read 0
run
forc write#read 1
run
forc addr c
forc datain 67
run
forc write#read 0
run
forc write#read 1
run
forc addr d
forc datain 89
run
forc write#read 0
run
forc write#read 1
run
forc addr e
forc datain ab
run
forc write#read 0
run
forc write#read 1
run

#setup dest. addr

forc addr f
forc datain fe
run
forc write#read 0
run
forc write#read 1
run
forc addr 10
forc datain dc
run
forc write#read 0
run
forc write#read 1
run
forc addr 11
forc datain ba
run
forc write#read 0
run
forc write#read 1
run
forc addr 12
forc datain 98
run
forc write#read 0
run
forc write#read 1
run
forc addr 13
forc datain 76
run
forc write#read 0
run
forc write#read 1
run
forc addr 14
forc datain 54
run
forc write#read 0
run
forc write#read 1
run

#read data back

forc addr 0
run

forc addr 1
run

forc addr 2
run

forc addr 3
run

forc addr 4
run

forc addr 5
run

forc addr 6
run

forc addr 7
run

forc addr 8
run

forc addr 9
run

forc addr a
run

forc addr b
run

forc addr c
run

forc addr d
run

forc addr e
run

forc addr f
run

forc addr 10
run

forc addr 11
run

forc addr 12
run

forc addr 13
run

forc addr 14
run

template run 1000
scale trace time 8000
```

A.25 Rregsetup.sim

```
template run 2000
#soft reset (to clear error flags in reg. 0 bits 0-2)

forc txsym 7
forc smreq 0
run 4000

#setup registers

forc txsym 0
run; run
template run 1000
forc addr 4
forc datain 44
run
forc write#read 0
run
forc write#read 1
run

#setup freq pool

forc addr 5
forc datain aa
run
forc write#read 0
run
forc write#read 1
run
forc addr 6
forc datain bb
run
forc write#read 0
run
forc write#read 1
run
forc addr 7
forc datain cc
run
forc write#read 0
run
forc write#read 1
run
forc addr 8
forc datain dd
run
forc write#read 0
run
forc write#read 1
run
forc addr 9

#setup source addr

forc datain 01
run
forc write#read 0
run
forc write#read 1
run
forc addr a
forc datain 23
run
forc write#read 0
run
forc write#read 1
run
forc addr b
forc datain 45
run
forc write#read 0
run
forc write#read 1
run
forc addr c
forc datain 67
run
forc write#read 0
run
forc write#read 1
run
forc addr d
forc datain 89
run
```

```
    forc write#read 0
    run
    forc write#read 1
    run
    forc addr e
    forc datain ab
    run
    forc write#read 0
    run
    forc write#read 1
    run

    #setup dest. addr

    forc addr f
    forc datain fe
    run
    forc write#read 0
    run
    forc write#read 1
    run
    forc addr 10
    forc datain dc
    run
    forc write#read 0
    run
    forc write#read 1
    run
    forc addr 11
    forc datain ba
    run
    forc write#read 0
    run
    forc write#read 1
    run
    forc addr 12
    forc datain 98
    run
    forc write#read 0
    run
    forc write#read 1
    run
    forc addr 13
    forc datain 76
    run
    forc write#read 0
    run
    forc write#read 1
    run
    forc addr 14
    forc datain 54
    run
    forc write#read 0
    run
    forc write#read 1
    run

    #read data back

    forc addr 0
    run

    forc addr 1
    run

    forc addr 2
    run

    forc addr 3
    run

    forc addr 4
    run

    forc addr 5
    run

    forc addr 6
    run

    forc addr 7
    run

    forc addr 8
    run

    forc addr 9
    run

    forc addr a
    run

    forc addr b
    run

    forc addr c
    run

    forc addr d
    run

    forc addr e
    run

    forc addr f
    run

    forc addr 10
    run

    forc addr 11
    run

    forc addr 12
    run

    forc addr 13
    run

    forc addr 14
    run

    template run 1000
```

A.26 Rsd.sim

```
#set up internal loopback with scrambler disabled
forc txsym 0
forc smreq 0
run 4000
forc addr 1
forc datain 8A
run
forc write#read 0
run
forc write#read 1
run
forc txsym 3
run; run; run
forc smreq 1
```

A.27 Rdd.sim

```
#set up internal loopback with descrambler disabled
forc smreq 0
forc txsym 0
run 4000
forc addr 1
forc datain 92
run
forc write#read 0
run
forc write#read 1
run
forc txsym 3
run; run; run
forc smreq 1
```

A.28 Rddsd.sim

```
#set up internal loopback with scrambler disabled
forc txsym 0
forc smreq 0
run 4000
forc addr 1
forc datain 9A
run
forc write#read 0
run
forc write#read 1
run
forc txsym 3
run; run; run
forc smreq 1
```

A.29 Rintloopback.sim

```
#set up internal loopback mode
forc txsym 3
run; run; run
forc smreq 1
```

A.30 Rextloopback.sim

```
#set up external looopback mode and 48 bit addresses (master mode)
forc addr 1
forc datain 85
run
forc write#read 0
run
forc write#read 1
run
forc txsym 3
run; run; run
forc smreq 1
```

A.31 Rtest.sim

```
view sheet
    SCAle TRace Time 1000
    check -nos

    #setup clocks

    CLOck Period 1000
    FORCe txclk 1 0 -Repeat
    FORCe txclk 0 500 -Repeat
    FORCe rxclk 1 0 -Repeat
    FORCe rxclk 0 500 -Repeat

    #use high speed clocks
    CLOck Period 500
    FORCe testclk 1 0 -Repeat
    FORCe testclk 0 250 -Repeat

    CLOck Period 2000
    FORCe impulseclk 1 0 -Repeat
    FORCe impulseclk 0 1000 -Repeat
    FORCe ctrltxclk 1 0 -Repeat
    FORCe ctrltxclk 0 1000 -Repeat
    FORCe ctrlrxclk 1 0 -Repeat
    FORCe ctrlrxclk 0 1000 -Repeat

    #force input signals to known levels

    FORCe clevel 6
    FORCe write#read 1
    FORCe txsym 6
    FORCe smreq 1
    FORCe level 6
    FORCe test 1
    FORCe jabbertimeout 1
    FORCe faultdetect 1
    FORCe datain 0
    FORCe carrierdetect 1
    FORCe ccardet 1
    FORCe addr 0

    #setup probes

    DO /idea/sys/hi/macro/analysis/view_down I$841
    MARK -Rectangle -0.3,2.1,View
    VIEw ARea 1.7,0.4,View
    VIEw ALL
    MARK -Rectangle -1.2,1.7,View
    VIEw ARea 3.2,0.0,View
    PROBe romout 1.4,1.1,View
    PROBe cksigout 1.4,0.7,View
    PROBe addr -0.8,1.5,View
```

```
PROBe compare 3.0,0.9,View
PROBe state 2.6,-0.5,View
TRAce addr romout cksigout compare state
DO /idea/sys/hi/macro/analysis/view_up

#setup traces
trace testclk test testinprogress testpassed signature
trace misrreg misrsm misrtx misrrx

template run 4000
#reset ric
#-- and clear the error flags in the status reg

FORCe reset 0
RUN
FORCe reset 1
RUN

#execute test to obtain signature

forc test 0
run 10000

#enable testpassed signal

#forc test 1
run 376000
```

A.32 REsetup.sim

```
VIEw Sheet
#history 50000
save state state0 -r
period trace 100000

# setup clocks

SCAle TRace Time 1000
CLOck Period 1000
FORCe txclk 1 0 -Repeat
FORCe txclk 0 500 -Repeat
FORCe rxclk 1 0 -Repeat
FORCe rxclk 0 500 -Repeat
CLOck Period 500
FORCe testclk 1 0 -Repeat
FORCe testclk 0 250 -Repeat
CLOck Period 2000
FORCe impulseclk 1 0 -Repeat
FORCe impulseclk 0 1000 -Repeat
FORCe ctrltxclk 1 0 -Repeat
FORCe ctrltxclk 0 1000 -Repeat
FORCe ctrlrxclk 1 0 -Repeat
FORCe ctrlrxclk 0 1000 -Repeat

#initial input signal levels

FORCe clevel 6
FORCe write#read 1
FORCe txsym 6
FORCe smreq 1
FORCe reset 1
FORCe level 6
FORCe test 1
FORCe jabbertimeout 1
FORCe faultdetect 1
FORCe datain 0
FORCe carrierdetect 1
FORCe ccardet 1
FORCe addr 0

# trace signals

TRAce txclk rxclk impulseclk reset resetout write#read addr datain dataout
TRAce  jabbertimeout faultdetect
TRAce smreq txsym smind rxsym transmitdisable
TRAce impulse level agchold
TRAce currfreq ctrlfreq
TRAce ctxdisable cimpulse cagchold clevel

#monitor signals

mon b txsym smreq impulse rxsym smind level
template run 2000

#hard reset

forc reset 0
run
forc reset 1
run

#soft reset (to clear error flags in reg. 0 bits 0-2)

forc txsym 7
forc smreq 0
run
forc txsym 0
run 4000

#single silence

forc txsym 6
forc smreq 1
run 1000
forc txsym 2
run 2000
forc txsym 6
run 8000

#soft reset (to clear error flags in reg. 0 bits 0-2)

forc txsym 7
forc smreq 0
run
forc txsym 0
run 4000

#odd number of non-data's

forc txsym 6
forc smreq 1
run 2000
forc txsym 2
run 8000
forc txsym 4
run 3000
forc txsym 0
run 1000
forc txsym 6
run 8000

#soft reset (to clear error flags in reg. 0 bits 0-2)

forc txsym 7
forc smreq 0
run 8000
forc txsym 0
run 4000

#disable bad input detector
forc txsym 0
forc addr 1
forc datain 20
run 1000
forc write#read 0
run 1000
forc write#read 1
forc addr 0
run 1000

#single silence (ignored)

forc txsym 6
forc smreq 1
run 1000
forc txsym 2
run 4000

#soft reset (to clear error flags in reg. 0 bits 0-2)

forc txsym 7
forc smreq 0
run 8000
forc txsym 0
run 4000

#disable bad input detector
forc txsym 0
forc addr 1
forc datain 20
run 1000
forc write#read 0
run 1000
forc write#read 1
forc addr 0
run 1000

#odd number of non-data's (ignored)

forc txsym 6
forc smreq 1
run 2000
forc txsym 2
run 8000
forc txsym 4
run 3000
forc txsym 0
run 2000
forc txsym 4
run 2000
forc txsym 0
run 3000

#soft reset (to clear error flags in reg. 0 bits 0-2)

forc txsym 7
forc smreq 0
run 8000
forc txsym 0
run 4000

#physical error -- jabber timeout
forc txsym 0
run 4000
forc jabbertimeout 0
run 3000

#soft reset (to clear error flags in reg. 0 bits 0-2)

forc txsym 7
forc smreq 0
run 8000
forc jabbertimeout 1
run
forc txsym 0
run 4000

#physical error -- fault detect
forc txsym 0
run 4000
forc faultdetect 0
run 3000

#soft reset (to clear error flags in reg. 0 bits 0-2)

forc txsym 7
forc smreq 0
forc faultdetect 1
run
forc txsym 0
run 4000
```

A.33 REslave.sim

```
#scale trace time
scale trace time 8000

#set up slave mode
forc clevel 6
forc addr 1
forc datain 00
run
forc write#read 0
run
forc write#read 1
run
forc txsym 5
run; run; run
forc txsym 3
run; run; run
forc smreq 1
forc txsym 6
run; run
```

```
#channel frequency received (crc error)

template run 2000
forc clevel 3
run
forc clevel 0
run
forc clevel 3
run
forc clevel 0
run
forc clevel 4
run
forc clevel 0
run
forc clevel 4
run
forc clevel 0
run
forc clevel 2
run
forc clevel 0
run
forc clevel 1
run
forc clevel 3
run
forc clevel 2
run
forc clevel 3
run; run; run
forc clevel 0
run
forc clevel 3
run
forc clevel 1
run
forc clevel 3
run
forc clevel 1
run
forc clevel 0
run; run; run
forc clevel 3
run
forc clevel 0
run
forc clevel 2
run
forc clevel 0
run
forc clevel 2
run; run; run; run
forc clevel 3
run
forc clevel 2
run
forc clevel 3
run
forc clevel 2
run; run
forc clevel 0
run; run; run
forc clevel 1
run
forc clevel 2
run
forc clevel 3
run
forc clevel 0
run
forc clevel 3
run
forc clevel 0
run; run

#error in reception

forc clevel 3
run; run
forc clevel 2
run
forc clevel 3
run
forc clevel 0
run
forc clevel 4
run
forc clevel 1
run
forc clevel 4
run
forc clevel 0
run
forc clevel 6
run

# used to account for Rslave.sim setup
# commands are ignored by station management module

forc smreq 0
forc txsym 0
```

A.34 REmacrx.sim

```
template run 2000

#first frame -- start delimiter error
#silence (one octet)
forc level 6
run 8000
#pad idle symbols (one octet)
forc level 3
run
forc level 0
run
forc level 3
run
forc level 0
run
#start delimiter (one octet)
forc level 4
run
forc level 0
run
forc level 4
run
forc level 1
run
forc level 3
run
forc level 2
run
forc level 3
run
forc level 1
run
forc level 3
run
forc level 1
run
forc level 0
run; run; run
forc level 3
run
forc level 0
run
forc level 2
run
forc level 0
run
forc level 2
run; run; run; run
forc level 3
run
forc level 2
run
forc level 3
run
forc level 2
run; run
forc level 0
run; run; run
forc level 1
run
forc level 2
run
forc level 3
run
forc level 0
run
forc level 3
run
forc level 0
run; run

#error in reception

forc level 3
run; run
forc level 2
run
forc level 3
run
forc level 0
run
forc level 4
run
forc level 1
run
forc level 4
run
forc level 0
run
forc level 6
run

#second frame -- odd number of pad-idle
#silence (one octet)
forc level 6
run 8000
#pad idle symbols (one octet)
forc level 3
run
forc level 0
run
forc level 3
run
forc level 4
run

#third frame -- invalid end delimiter
#silence (one octet)
forc level 6
run 8000
#pad idle symbols (one octet)
forc level 3
run
forc level 0
run
forc level 3
run
forc level 0
run
#start delimiter (one octet)
forc level 4
run
forc level 0
run
forc level 4
run
forc level 0
run
#mac 'data' -- frame control, addresses, and data
forc level 1
run 8000
forc level 0
run 8000
forc level 1
run 8000
#end delimiter (one octet)
forc level 4
run
forc level 2
run
forc level 4
run
forc level 0
run
#silence (one octet)
forc level 6
run 8000

#fourth frame -- carrierdetect error
#silence (one octet)
forc level 6
run 8000
#pad idle symbols (one octet)
forc level 3
run
forc level 0
run
forc level 3
run
forc level 0
run
#start delimiter (one octet)
forc level 4
run
forc level 0
run
forc level 4
run
forc level 1
run
#mac 'data' -- frame control, addresses, and data
forc level 1
run 8000
forc carrierdetect 0
run 8000
forc level 6
forc carrierdetect 1
run 8000

#fifth frame -- too many identical symbols
#silence (one octet)
forc level 6
run 8000
#pad idle symbols (one octet)
forc level 3
run
forc level 0
run
forc level 3
run
forc level 0
run
#start delimiter (one octet)
forc level 4
run
forc level 0
run
forc level 4
run
forc level 0
run
#mac 'data' -- frame control, addresses, and data
forc level 1
run 55000
forc level 4
run
forc level 3
run
forc level 4
run
forc level 0
run
forc level 6
run 8000
```

A.35 REmactx.sim

```
template run 1000

#first frame -- start delimiter error
#silence (one octet)
forc txsym 6
run 8000
#pad idle symbols (one octet)
forc txsym 2
run 8000
#start delimiter (one octet)
forc txsym 4
run 2000
forc txsym 0
run
forc txsym 4
run 2000
forc txsym 1
run 3000

#second frame -- odd number of pad-idle symbols
forc txsym 6
run 8000
#pad idle symbols (one octet)
forc txsym 2
run 5000
#start delimiter (one octet)
forc txsym 4
run 2000
forc txsym 0
run
forc txsym 4
run 2000
forc txsym 0
run 3000

#third frame -- invalid end delimiter
forc txsym 6
run 8000
#pad idle symbols (one octet)
forc txsym 2
run 8000
#start delimiter (one octet)
forc txsym 4
run 2000
forc txsym 0
run
forc txsym 4
run 2000
forc txsym 0
run 3000
#mac 'data' -- shortened version of data frame
forc txsym 1
run 4000
forc txsym 0
run 4000
forc txsym 1
run 8000
#end delimiter -- 1 extra bit (third bit)
forc txsym 4
run 2000
forc txsym 1
run 2000
forc txsym 4
run 2000
forc txsym 1
run
forc txsym 0
run 2000
#silence (one octet)
forc txsym 6
run 8000
```

A.36 Rirun.sim

```
check -nos
do ~/asic/ric/chip_files/Rsetup.sim
do ~/asic/ric/chip_files/Rintlpbk.sim
do ~/asic/ric/chip_files/Rmactx.sim
run 10000
```

A.37 Rxrun.sim

```
check -nos
do ~/asic/ric/chip_files/Rsetup.sim
do ~/asic/ric/chip_files/Rextlpbk.sim
do ~/asic/ric/chip_files/Rmactx.sim
run 73000
```

A.38 Rorun.sim

```
check -nos
do ~/asic/ric/chip_files/Rsetup.sim
do ~/asic/ric/chip_files/Rsd.sim
do ~/asic/ric/chip_files/Rmactx.sim
run 10000
do ~/asic/ric/chip_files/Rdd.sim
do ~/asic/ric/chip_files/Rmactx.sim
run 10000
do ~/asic/ric/chip_files/Rddsd.sim
do ~/asic/ric/chip_files/Rmactx.sim
run 10000
```

A.39 Rmrun.sim

```
check -nos
do ~/asic/ric/chip_files/Rsetup.sim
do ~/asic/ric/chip_files/Rmaster.sim
do ~/asic/ric/chip_files/Rmactx.sim
do ~/asic/ric/chip_files/Rmacrx.sim
```

A.40 Rsrun.sim

```
check -nos
do ~/asic/ric/chip_files/Rsetup.sim
do ~/asic/ric/chip_files/Rslave.sim
do ~/asic/ric/chip_files/Rmactx.sim
do ~/asic/ric/chip_files/Rmacrx.sim
```

A.41 Rerun.sim

```
check -nos
do ~/asic/ric/chip_files/Rsetuperr.sim
do ~/asic/ric/chip_files/REslave.sim
do ~/asic/ric/chip_files/Rslave.sim
do ~/asic/ric/chip_files/REmactx.sim
do ~/asic/ric/chip_files/Rregsetup.sim
do ~/asic/ric/chip_files/Rslave.sim
do ~/asic/ric/chip_files/REmacrx.sim
```

A.42 Rtrun.sim

```
do ~/asic/ric/chip_files/Rtest.sim
```

Appendix B: ChipCrafter Schematics