TI Designs

Real-Time Ethernet Tracer for PRU-ICSS Reference Design

Description

A real-time Ethernet and industrial Ethernet are used in the factory automation floor to control automated production. Monitoring the Ethernet traffic in such areas is an essential maintenance and error analysis tool.

This TI Design is a real-time Ethernet tracer created for the programmable real-time unit and industrial communication subsystem (PRU-ICSS) that is set inside an industrial Ethernet network as a passive tracer device, which enables to monitor and record all Ethernet frames including frame time stamping without modifying the Ethernet traffic.

This TI Design enables customers to build products for network analysis in factory automation and industrial communication

Features

- Monitors and Traces Functions of Industrial Ethernet Frames
- Frame Timestamping in Nanosecond Resolution
- Supports 64-Bit Time Frame Stamping
- Gbit Ethernet PHY Uplink
- Wireshark™ Compatible Capture Format
- PRU Firmware and ARM® Driver in Source Code to Enable Customer Differentiation

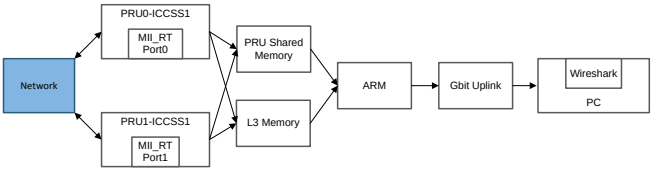
Applications

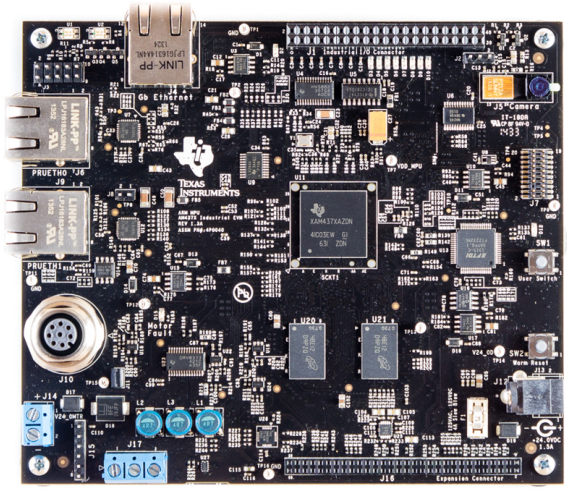
- Factory Automation and Control
- Industrial Communication

Resources

TIDEP0064	Design Folder
AM4379	Product Folder
DP83848	Product Folder
DP83822	Product Folder
TLK105L	Product Folder

TI E2E™ Community — ASK Our E2E Experts





1 System Overview

1.1 What is Real-Time Ethernet?

Real-time Ethernet, which comprises of many systems like programmable logic controller (PLC), remote input/output, actuators, sensors and motor drives, is very popular on the factory floor. Real-time does not mean faster communication cycles but rather the communication cycle is very deterministic and has low jitter. Real-time systems not only depend on the validity of process data but also on its timeliness. This quality results in determinism which is an important factor when differentiating real-time Ethernet from standard Ethernet. The packets of data in industrial settings are required to be sent and received at specific times, and real-time Ethernet needs to be guaranteed that data will be delivered each and every time because a loss of data or delay of data can result in an undesired malfunction of the machine in an industrial environment. Industrial Ethernet protocols like PROFINET®, EtherCAT®, Sercos III, EtherNet/IP™, and Ethernet POWERLINK are very popular in today's real-time Ethernet networks because these industrial Ethernet protocols provide the required functionality and determinism

1.2 What is a Tracer?

A tracer or sniffer is a device that monitors and captures the Ethernet data flowing through the Ethernet network links in real-time. The captured information can be used for diagnostics and network maintenance. The tracer does not modify the Ethernet frames but only captures the Ethernet traffic.

1.3 System Block Diagram

Figure 1 shows the block diagram of the system. The real-time Ethernet tracer can be connected to the network using the two Ethernet ports (MII_RT0 and MII_RT1). The incoming packets are stored inside a frame buffer for each port – the frame buffer is located in L3 memory. All the queue management data of the frame buffer is stored in PRU shared memory. The ARM takes these frames out of the frame queue in a chronological order based on the 64-bit frame timestamp. The timestamp is appended at the end of Ethernet frame and acts as information for the ESL Wireshark frame format. The frames are transmitted by the ARM over the gigabit Ethernet port to the PC where the frames can be captured with the Wireshark.

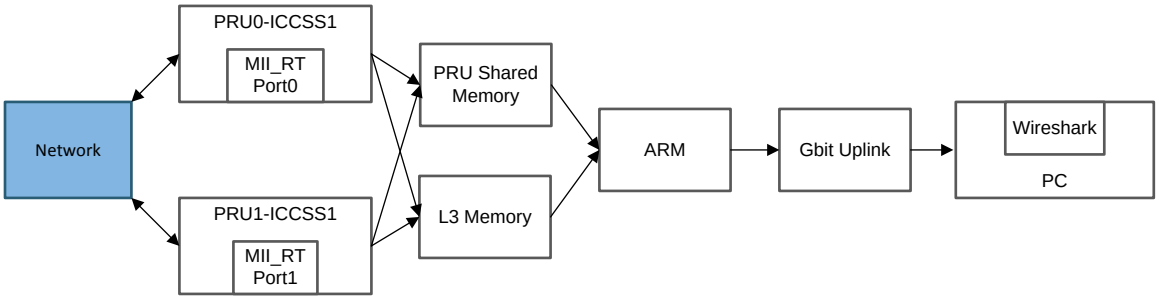


Figure 1. Real-Time Ethernet Tracer Block Diagram

1.4 Key System Level Specifications

- 2-port real-time Ethernet tracer for 100-Mbps full-duplex
- Gbit port Uplink
- Cyclic redundancy check (CRC) error and error nibble detection
- ESL-compatible frame format generation for Wireshark

1.5 Highlighted Products

1.5.1 AM4379 Processor

Up to 1-GHz Sitara™ ARM® Cortex®-A9 32-Bit RISC Processor

- NEON™ single-instruction, multiple data (SIMD) coprocessor and vector floating point (VFPv3) coprocessor
- 32 KB of L1 instruction and 32 KB of data cache
- 256 KB of L2 cache or L3 RAM
- 256 KB of on-chip boot ROM
- 64KB of dedicated RAM
- Emulation and debug - JTAG
- Interrupt controller

PRU-ICSS

- Supports protocols such as EtherCAT, PROFIBUS®, PROFINET, EtherNet/IP, EnDat 2.2, and more
- Two PRU subsystems with two PRU cores each
- 32-bit load and store RISC processor capable of running at 200 MHz
- 12 KB (PRU-ICSS1), 4 KB (PRU-ICSS0) of instruction RAM with single-error detection (parity)
- 8 KB (PRU-ICSS1), 4 KB (PRU-ICSS0) of data RAM with single-error detection (parity)
- Single-cycle 32-bit multiplier with 64-bit accumulator
- Enhanced GPIO module provides shift-in and -out support and parallel latch on external signal
- 12KB (PRU-ICSS1 only) of shared RAM with single-error detection (parity)
- Three 120-byte register banks accessible by each PRU
- Interrupt controller module (INTC) for handling system input events
- Local interconnect bus for connecting internal and external masters to the resources inside the PRU-ICSS
- Peripherals inside the PRU-ICSS
  - One UART port with flow control pins, supports up to 12 Mbps
  - One enhanced capture (eCAP) module
  - Two media independent interfaces (MII) ethernet ports that support industrial ethernet, such as EtherCAT
  - One management data input/ouput (MDIO) port

On-chip memory (shared L3 RAM)

- 256 KB of general-purpose on-chip memory controller (OCMC) RAM
- Accessible to all masters

External memory interfaces (EMIF)

- Double data rate memory (DDR) controllers:
  - LPDDR2: 266-MHz clock (LPDDR2-533 data rate)
  - DDR3 and DDR3L: 400-MHz clock (DDR-800 data rate)
  - 32-bit data bus
  - 2 GB of total addressable space
  - Supports one x32, two x16, or four x8 memory device configurations
- General-purpose memory controller (GPMC)
  - Flexible 8-bit and 16-bit asynchronous memory interface with up to seven chip selects [TM note: NAND and NOR are flash memory types, they are not written out. It basically means not 'and'] (NAND, NOR, Muxed-NOR, SRAM)
  - Uses Bose-Chaudhuri-Hocquenghem (BCH) code to support 4-, 8-, or 16-bit ECC
  - Uses hamming code to support 1-bit error correction coding (ECC)

See the AM4379 data sheet for a complete list of features (SPRS851).

1.5.2 TLK105L Ethernet PHY

- Low power consumption:
  - Single supply: <205 mW PHY, 275 mW with center tap (typical)
  - Dual supplies: <126 mW PHY, 200 mW with center tap (typical)
- Programmable power back off to reduce PHY power up to 20% in systems with shorter cables
- IEEE 1588 start frame delimiter (SFD) indication enables time stamping by a controller or processor
- Low deterministic latency supports IEEE1588 implementation
- Cable diagnostics
- Programmable fast link down modes, <10 µs reaction time
- Variable I/O voltage range: 3.3 V, 2.5 V, 1.8 V
- MAC interface I/O voltage range:
  - MII I/O voltage range: 3.3 V, 2.5 V, 1.8 V
  - MII I/O voltage range: 3.3 V, 2.5 V
- R-fixed TX clock to XI with programmable phase shift
- Auto-MDIX for 10/100 Mbs
- Energy detection mode
- MII and RMII capabilities
- IEEE 802.3u MII
- Error-free 100Base-T operation up to 150 meters under typical conditions
- Error-free 10Base-T operation up to 300 meters under typical conditions
- Serial management interface
- IEEE 802.3u auto-negotiation and parallel detection
- IEEE 802.3u ENDEC, 10Base-T
- Transceivers and filters
- IEEE 802.3u PCS, 100Base-TX transceivers
- Integrated ANSI X3.263 compliant TP-PMD physical sublayer with adaptive equalization and baseline wander compensation
- Programmable LED support link, activity
- 10/100 Mbs packet built in self test (BIST)
- Born–Mayer–Huggins (HBM) electro static discharge (ESD) protection on RD± and TD± of 16 kV
- 32-pin VQFN
  - 5 mm × 5 mm

1.5.3 DP83822 Ethernet PHY

- IEEE 802.3u compliant: 100BASE-FX, 100BASETX and 10BASE-Te
- MII, RMII, and RGMII MAC Interfaces
- Low-power single supply options:
  - 1.8-V average (AVD) < 120 mW
  - 3.3-V AVD < 220 mW
- ±16-kV HBM ESD Protection
- ±8-kV IEC 61000-4-2 ESD Protection
- Start of frame detect for IEEE 1588 time stamp
- Fast link-down timing
- Auto-crossover in force modes
- Operating temperature: –40 to 125°C
- I/O voltages: 3.3 V, 2.5 V and 1.8 V

- Power savings features
  - Energy efficient Ethernet (EEE) IEEE 802.3az
  - Wake-on-LAN (WoL) support with magic packet detection
  - Programmable energy savings modes
- Cable diagnostics
- BIST
- Management data clock (MDC) and MDIO interface

1.5.4 AM437X IDK EVM Hardware Specification

- AM4379 ARM Cortex-A9
- 1-GB DDR3, QSPI-NOR Flash
- Discrete power solution
- EnDat connectivity for motor feedback control
- 24-V power supply
- USB cable for JTAG interface and serial console

Software and tools:

- SYS/BIOS real-time OS
- Starterware base port
- Code Composer Studio™ (CCS) integrated development environment (IDE)
- Application stack for industrial communication protocols
- Sample industrial applications

Connectivity:

- PROFIBUS interface
- CANOpen
- EtherCAT
- EtherNet/IP
- PROFINET
- Sercos III
- IEC61850
- PWM
- Motor axis position feedback
- Up to three-phase motor drive connector
- Sigma-delta decimation filter
- Digital inputs and outputs
- Serial port interface (SPI)
- UART
- JTAG

See the AM437X IDK website for a complete list of features and design resources (http://www.ti.com/tool/TMDXIDK437X).

2 System Design Theory

2.1 MII_RT Configuration

- The configuration for both Ethernet ports and MII_RT ports is done in such a way that the incoming packets are stored in the L2 buffer of MII_RT with auto-forwarding enabled.
- The packets stored in L2 buffer include the frame preamble, SFD, and CRC.

2.2 Ethernet Frame Queue Management

The packets arriving at port0 and port1 are stored in L2 buffer of MII_RT. The PRU copies the packets into a queue for each port, which resides in L3 memory. The queue is a cyclic queue with a read and write pointer for the management of packets inside the queue. Each entry of the queue is called a buffer and each buffer consists of 32 bytes.

The total number of buffer entries can be N inside the queue, which can be configured by the programmer. The packets stored inside the queue take integer number of buffers. If a buffer is not completely occupied by the packet, that buffer is still completely dedicated to that packet, and the next packet will still start to occupy the next buffer.

The management of the queue is done by a buffer descriptor and a queue descriptor. The queue descriptors contain the values for read and write pointers where each pointer value consists of 16 bits. Each buffer descriptor is 32 bits with the upper 16 bits dedicated to packet length storage and the lower 16 bits stores the information for alignment error (bit 3), CRC error (bit 4), and receive error (bit 5). This queue management technique is described in Figure 2.

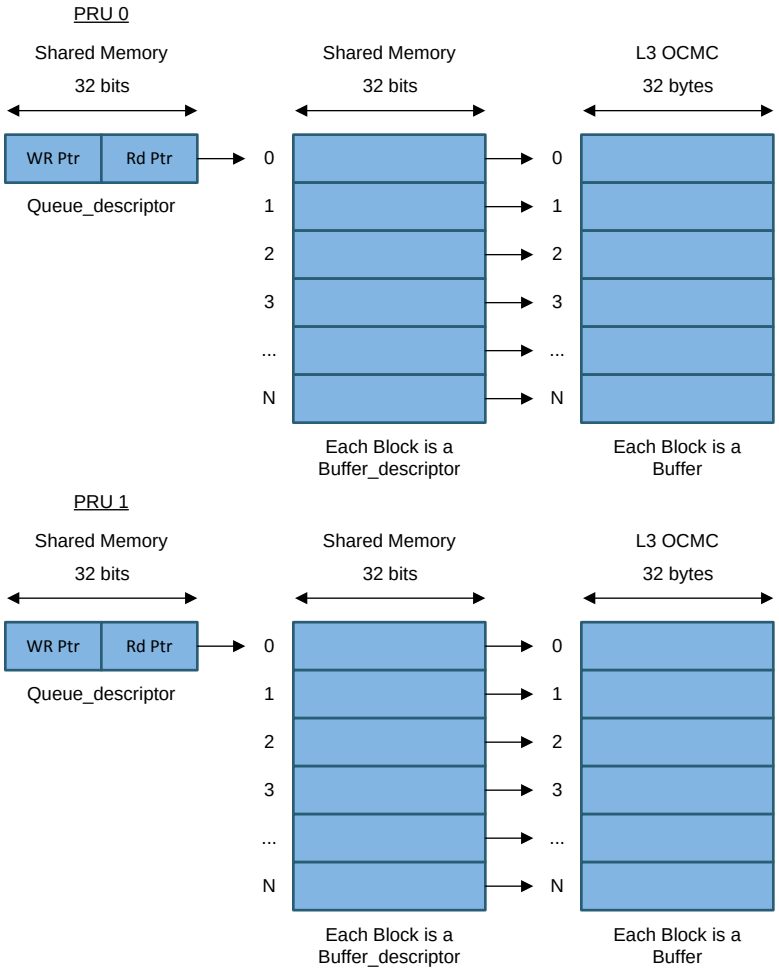


Figure 2. Queue Management for Each PRU

The queue exists inside the L3 memory whereas the buffer descriptors and queue descriptors are stored inside PRU shared memory.

Table 1. Memory Map for Queue Management

NAME	MEMORY LOCATION IN SHARED MEMORY	MEMORY LOCATION IN L3 MEMORY	DESCRIPTION
PRU0 queue descriptor	0x100	-	Contains the value of read and write pointers of queue for port0 handled by PRU0
PRU1 queue descriptor	0x104	-	Contains the value of read and write pointers of queue for port1 handled by PRU1
PRU0 buffer descriptor offset	0x120	-	Buffer descriptor entries for PRU1 are stored starting from this address
PRU1 buffer descriptor offset	0x120 + (number of buffers × 4)	-	Buffer descriptor entries for PRU1 are stored starting from this address
PRU0 queue offset	-	0x00	Starting address for queue for port0
PRU1 queue offset	-	0x00 + (number of buffers × 32)	Starting address for queue for port1

2.2.1 Custom Frame Header

Each packet stored inside the queue is stored with a custom defined header that consists of 32 bytes and occupies one buffer entry. The upper most 4 bytes (bytes 31 to 28) in the queue contain the packet specific information called packet descriptor shown in Table 2. After that the next 4 bytes (bytes 27 to 24) contains the upper 32 bits of timestamp followed by the next 4 bytes (bytes 23 to 20) containing the lower 32 bits of timestamp

Table 2. Custom Header With Packet Descriptor and Timestamp Value

BYTES	BITS	NAME	DESCRIPTION
31 to 28	0 to 14	Reserved	Reserved for future use
	15	Timestamp	Receive packet has 64 bit time stamp appended
	16 to 26	Length	11 bit of total packet length which is put into buffer descriptor as well. This is the actual length of the packet received at the port including preamble
	27, 28	Port	Indicates which port the packet was received on. Port=0 indicates Ethernet PHY 0 and Port=1 indicates Ethernet PHY 1
	29 to 31	Reserved	Reserved for future use
27 to 24	0 to 31	TS_high	Contains the upper 32 bits of the Timestamp value
23 to 20	0 to 31	TS_low	Contains the lower 32 bits of the Timestamp value
19 to 0	all	Reserved	Reserved for future use

2.2.2 ESL Information Format

Each packet stored inside the queue is stored with a custom defined header that consists of 32 bytes and occupies 1 buffer entry. The upper most 4 bytes (bytes 31 to 28) in the queue contains the packet specific information called packet descriptor shown in Table 3. After that the next four bytes (bytes 27 to 24) contains the upper 32 bits of timestamp followed by the next four bytes (bytes 23 to 20) containing the lower 32 bits of timestamp

Table 3. ESL Information

BYTES	NAME	DESCRIPTION
15 to 10	Symbolic MAC	Contains identifier 01 01 05 10 00 00 that is a symbolic MAC address
9	Port designation	Tells the port number on which packet is received
8	Error information	This byte tells the error information. Bits 0..2 are reserved, bit 3 is for alignment error and bit 4 is for CRC error
7 to 0	Timestamp	These bytes represent the 64 bit timestamp in ns

2.3 PRU Firmware

The PRU firmware is designed so that it handles 32 bytes of data as a block. The firmware works in a loop that, depending on the state of the communication, executes one of the receive functions: first block (FB), next block (NB) or last block (LB). PRU firmware also serves pending interrupt controller (INTC) events.

2.3.1 PRU Firmware Main Control Loop

The functionality of the main control loop is described in the form of flowchart in Figure 3.

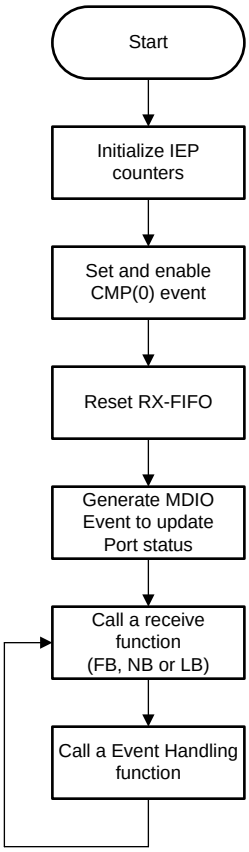


Figure 3. PRU Firmware Main Loop Functionality

2.3.2 Receive Functions (FB, NB, LB)

The receive functions are responsible for storing the incoming packet inside the PRU queues and updating the queue descriptor and buffer descriptor fields. FB checks for start of new frame and initializes the custom header space, NB is called every time there are 32 bytes or more in L2 buffer, and LB upon end of frame event and updates statistics and queue descriptor and buffer descriptor fields. The functionality of these receive functions is described in detail in the following subsections. The logical functionality of how these functions are linked together is shown in Figure 4.

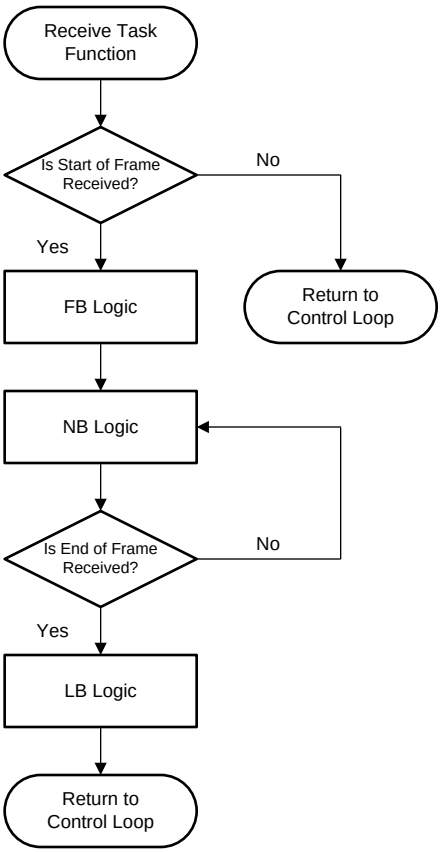


Figure 4. Logical Functionality of Receive Functions

2.3.2.1 First Block (FB)

FB executes in the following order:

- Check if there are required number of bytes in L2 after start of new frame.
- If the above is true, set RCV_ACTIVE_FLAG and RCV_HOST_PORT_FLAG.
- Set up the queue by loading the right write pointer value and buffer offset.
- Calculate free space and continue if space is available; otherwise, clear RCV_HOST_PORT_FLAG and return to control loop.
- Initialize 32 bytes custom header space.
- Load and store the 64-bit timestamp.
- Increment buffer write pointer by 32 bytes and handle buffer wrap-around if there is one.
- Return to control loop

Figure 5 shows the functionality of the FB in the form of a flow chart.

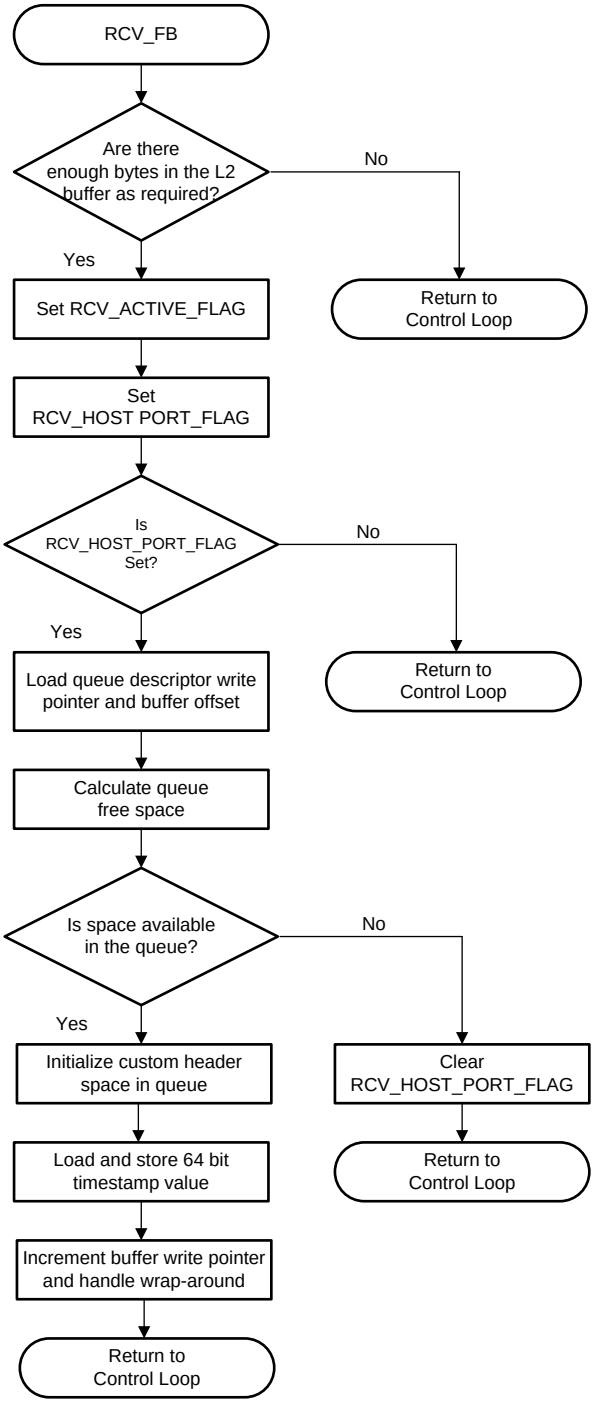


Figure 5. Receive Function FB Functionality

2.3.2.2 Next Block (NB)

NB executes in the following order:

- Check if RCV_HOST_PORT_FLAG is set and if there are 32 bytes or more in L2 buffer.
- Continue if the above is true; otherwise, return to the control loop.
- Transfer 32 bytes from L2 buffer to the PRU internal registers and increment the received packet length by 32.
- Calculate free space and continue if space is available; otherwise, clear RCV_HOST_PORT_FLAG and return to control loop.
- Store 32 bytes of the frame inside the queue according to buffer write pointer.
- Increment buffer write pointer by 32 bytes, and handle buffer wrap-around if there is one.
- Check if there is an end of frame (EOF) event.
- If there is an EOF event, jump to LB receive function; otherwise, return to control loop.

Figure 6 shows the functionality of the NB receive function in the form of a flow chart.

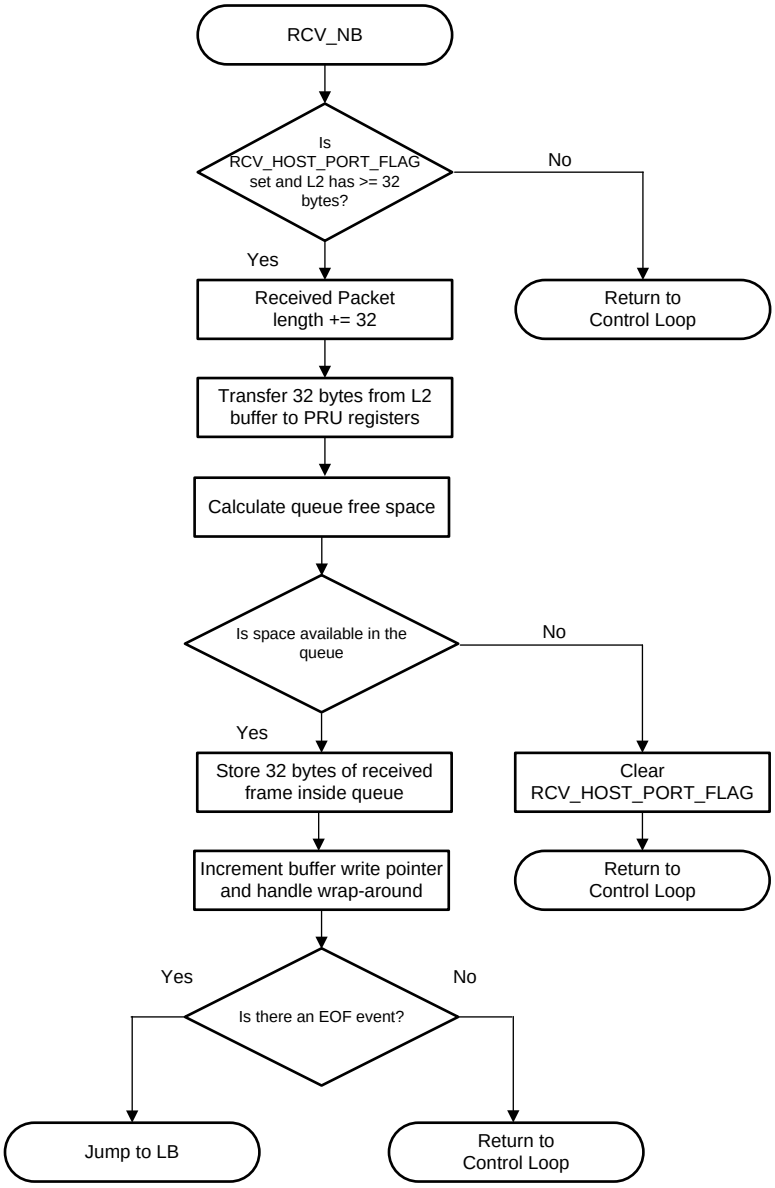


Figure 6. Receive Function NB Functionality

2.3.2.3 Last Block (LB)

LB is called by the NB when an EOF event occurs executes in the following order:

- Check if RCV_HOST_PORT_FLAG is set and continue if it is; otherwise, return to control loop.
- Transfer remaining bytes of received packet frame from the L2 buffer to the PRU internal registers.
- Increment the received packet length by the remaining bytes.
- Calculate free space and continue if space is available; otherwise, clear RCV_HOST_PORT_FLAG and return to control loop.
- Store the remaining bytes of the received packet frame inside the queue according to buffer write pointer .
- Extract frame status information from the L2 buffer and check if there is CRC, alignment, or receive error.
- Set the error bits in a PRU register according to ESL format if there are above mentioned errors and update statistics about received errors per port in shared memory.
- Load the original queue descriptor write pointer and calculate packet descriptor offset inside custom header in queue buffer.
- Update the packet descriptor field inside custom header with the received packet length and the port number on which it was received.
- Append the received packet length with the error information from the PRU register for the received packet and store it in the buffer descriptor.
- Calculate and update the new queue descriptor write pointer value in shared memory.
- Update statistics about received packets per port in shared memory.
- Reset the RX FIFO, and clear all flags in RCV_STATUS_REG.
- Return to control loop.

Figure 7 shows the functionality of the LB Receive Function in the form of a flow chart.

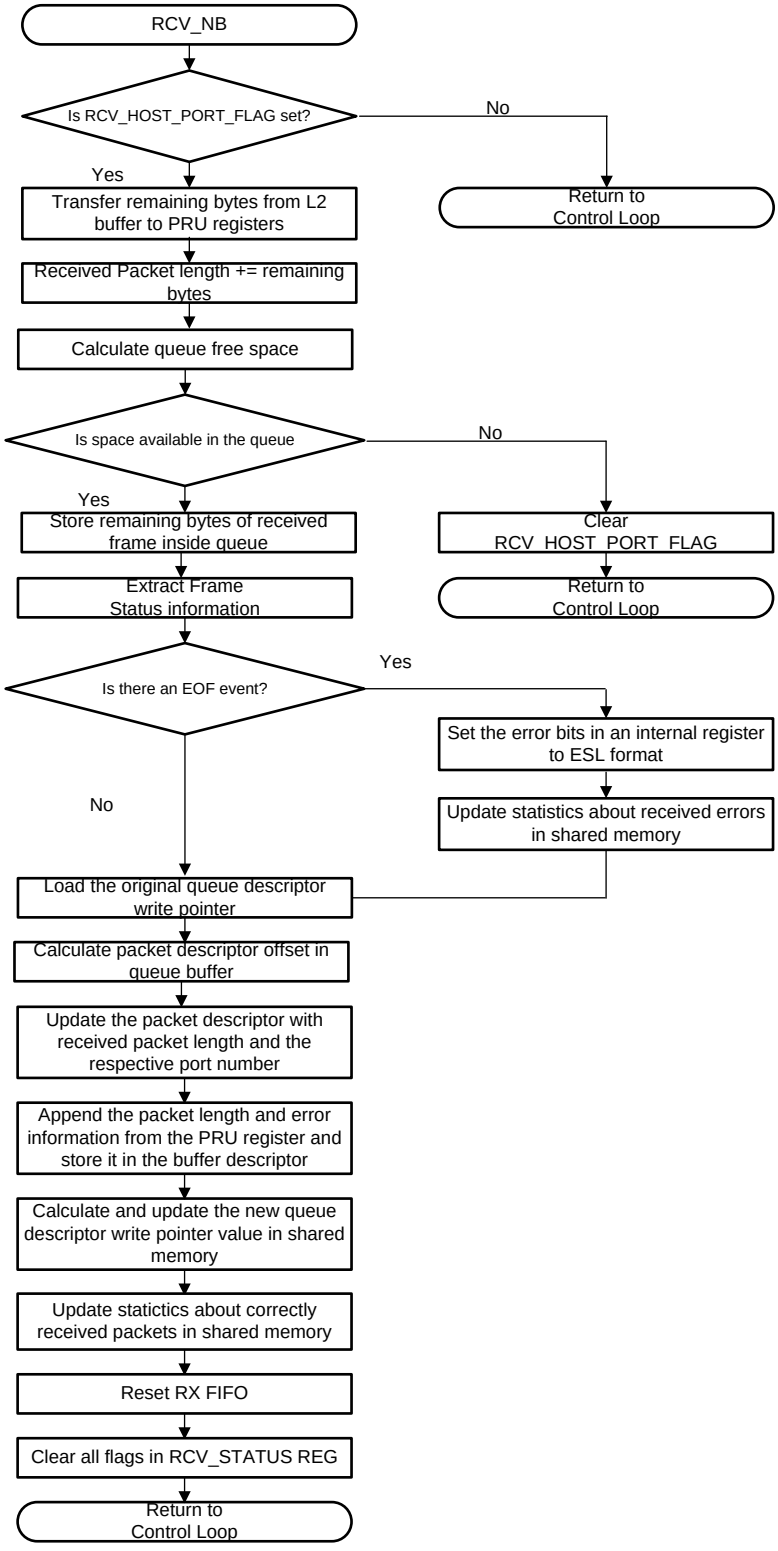


Figure 7. Receive Function Last LB Functionality TM

NOTE: Figure 7 does not match the original picture.

2.3.3 IEP Timestamping With 64-Bit Support

The IEP timer peripheral of PRU-ICSS within the AM437x supports only a 32-bit timer. PRU firmware is used to extend the timestamping to 64 bit. The 32-bit IEP timer wrap-around occurs every 4.29 seconds. This wrap-around is used in providing 64-bit timestamping. The IEP event is generated when there is a wrap-around by setting up the IEP_CMP [0] event. Each PRU updates their IEP high count value, which is stored in shared memory.

The IEP_CMP[0] event is only handled by the PRU0. To update the PRU1 about the event so the PRU1 can update its IEP high count, a PRU2PRU event is defined, which is generated by PRU0 when it handles the IEP_CMP[0] event.

The timestamp is stored in FB and cannot execute until there is a specified amount of bytes in the L2 buffer. The IEP event takes around 1 µs to take effect. If the IEP event arrives when the FB has already been executed and RCV_ACTIVE_FLAG is set, the IEP high count of each PRU is incremented However, if the packet has started arriving but the FB has not executed yet, there are two cases that need to be handled, which are illustrated in Figure 8 .

Case 1: The frame starts arriving just before IEP wrap-around and IEP event occurs before FB.

Case 2: The frame starts arriving just after IEP wrap-around and IEP event occurs before FB.

To handle these cases each PRU firmware IEP event handler first checks if RCV_ACTIVE_FLAG is set. If RCV_ACTIVE_FLAG is set, the IEP high count is incremented, and if RCV_ACTIVE_FLAG is not set, the RCV_ACTIVE_FLAG is checked in case there is something in the L2 buffer, which indicates that new packet has started to arrive though the FB is not yet executed so the RCV_TS_HIGH_UPDATE_FLAG is then set. In FB while storing the timestamp value a decision is made that if RCV_TS_HIGH_UPDATE_FLAG is set and the IEP timer value is less than 1 µs, the packet that arrived before the IEP wrap-around and the IEP high count value is subtracted by one before storing the value.

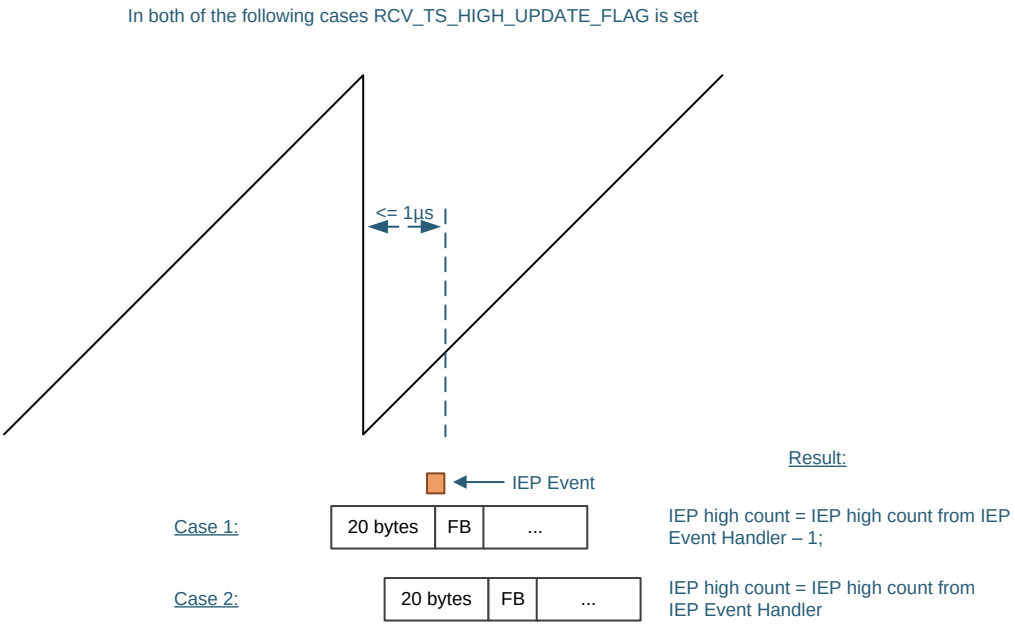


Figure 8. 64-Bit IEP Timestamping Logic

2.3.4 Event Handler

The PRU firmware event handlers are part of the event handling function called from the main control loop called after receive functions FB, NB, or LB. The following events are covered by event handlers.

2.3.4.1 Receive Function Events

The end of frame event is generated by the MII_RT port upon detection, which is handled by EOF event handler. The EOF event handler sets the RCV_EOF_EVENT_FLAG for LB so that LB is called from NB upon detection of this flag.

2.3.4.2 MDIO Events

MDIO events occur whenever there is a change of state for the link of the MII_RT ports. The event handler for MDIO events check which ports are enabled. If both ports are enabled, auto-forwarding for the MII_RT ports is enabled; otherwise, auto-forwarding is disabled

2.3.4.3 IEP Events

IEP events are enabled due to 64-bit timestamping logic. The IEP event is the IEP_CMP [0] event that is set close to the wrap-around of the IEP timer value. The IEP event is handled only by the PRU0. When the IEP event occurs, the PRU0 generates a PRU to the PRU event for the PRU1. After the PRU is generated, the RCV_TS_HIGH_UPDATE_FLAG is set up if the packet is being received and the FB is not executed yet. Finally, IEP high count value is incremented by the event handler.

The PRU1 event handler handles PRU to PRU event generated by the PRU0 due to the IEP timer event. The event handler of PRU1 applies the same logic as PRU0 IEP event handler except for generating an event.

2.4 ARM® Software Application

ARM software is responsible for:

- Initializing the board
- Interrupt mappings
- Board pin-multiplexing
- PRU-ICSS configuration
- MII_RT ports configuration (enabling L2 buffer and auto forwarding)
- Defining and creating tasks
- Initializing L3 memory and shared memory
- Loading queue parameters
- Loading and starting both PRU’s firmware
- Opening gigabit port socket interface
- Running the application loop

2.4.1 Main Application Loop

The main loop in the application task is responsible for loading the PRU firmware and opening the gigabit socket interface. The function RxPktTransfer_ESL returns the packet length of the packet in ARM buffer in the beginning of every iteration of the loop. The length inside ARM buffer includes the length of the packet received by the MII_RT port excluding the preamble and including 16 bytes of additional length because of the ESL information. If there is a packet in the ARM buffer, the RxPktTransfer_ESL function has returned a length greater than zero. This packet is then sent through the NDK socket over the gigabit port. After that the task goes to sleep to allow other tasks to run.

Figure 9 shows the functionality of the application loop in the form of a flowchart.

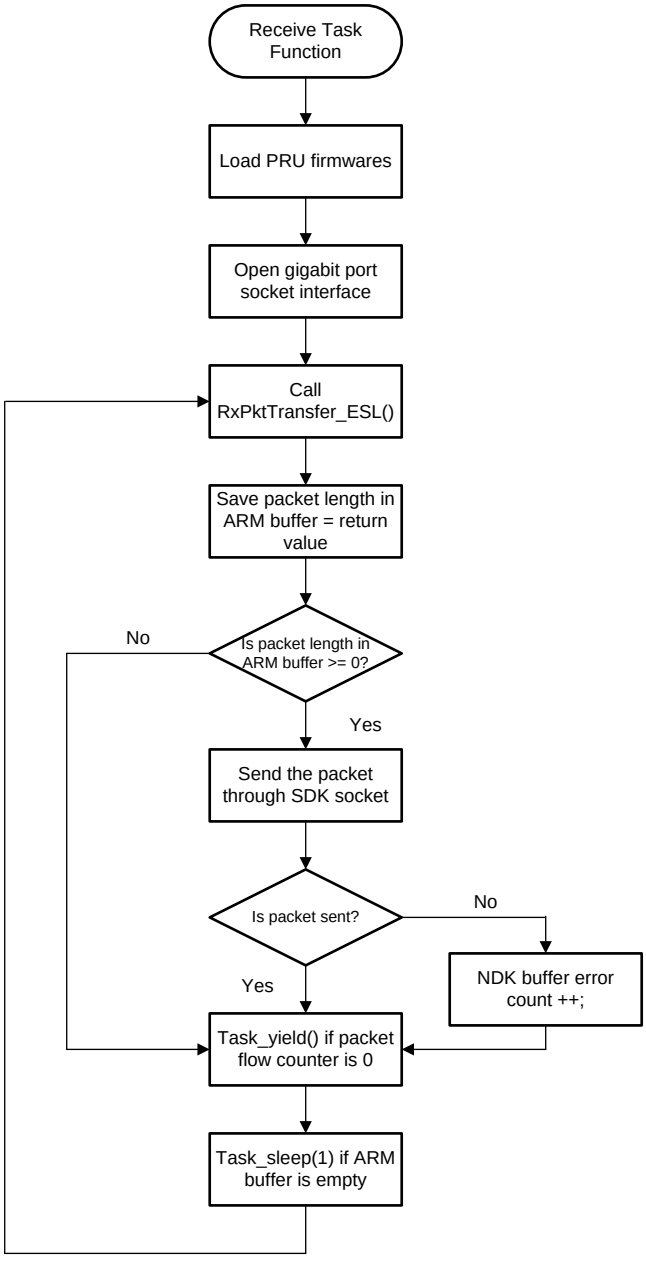


Figure 9. ARM® Application Loop Functionality

2.4.2 RxPktTransfer_ESL Function

This function is called from the main application loop. When the function is called it first checks if there is a packet in queue for both ports by calling the function RxPktInfo, which returns a code. The returned code tells which queue has the packet. The function then checks if the code is 0, which means queue for port 0 has the packet. If the code is not 0, the function then checks if the code is 1, which means one queue for port 1 has the packet. If the code is not 1, the function checks if the code is 2, which mean both queues have packets. If both ports have packets, the function assigns the value of the port number based on the least timestamp value of packet in each port. If both queues are empty, the function exits by returning the value of -1; otherwise, the function Copy_Packet_PRU2ARM_ESL is called with port_number as input parameter, which returns the length of the packet including ESL information, excluding preamble. This length information is then also returned by this function.

Figure 10 shows the functionality of RxPktTransfer_ESL function in the form of a flowchart.

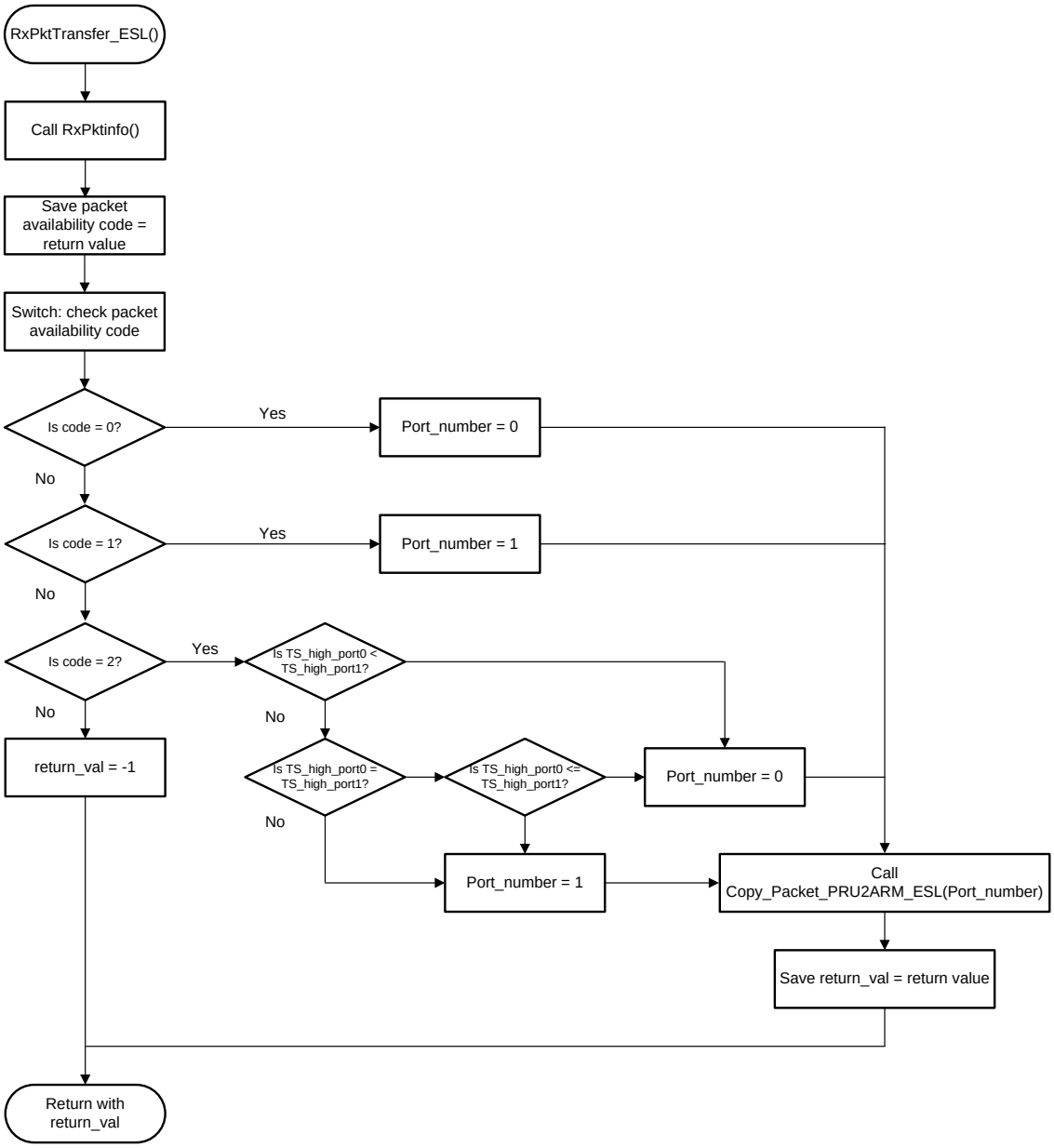


Figure 10. RxPktTransfer_ESL Function

2.4.3 RxPktInfo Function

This function checks which queue for both ports has the packets. The function checks by first loading and storing the read and write pointers of each queue from their queue descriptors and then compares the pointers to see if they are equal, which means the queue is empty; otherwise, there is a packet in the queue. If there is a packet, it sets a packet found flag for that port. Then based on this flag for each port the function decides which value to return as a code that signifies which port has the packet. 0 means port0, 1 means port1, 2 means both ports, and -1 means none of the ports have the packets.

Figure 11 shows the functionality of this function in the form of a flowchart.

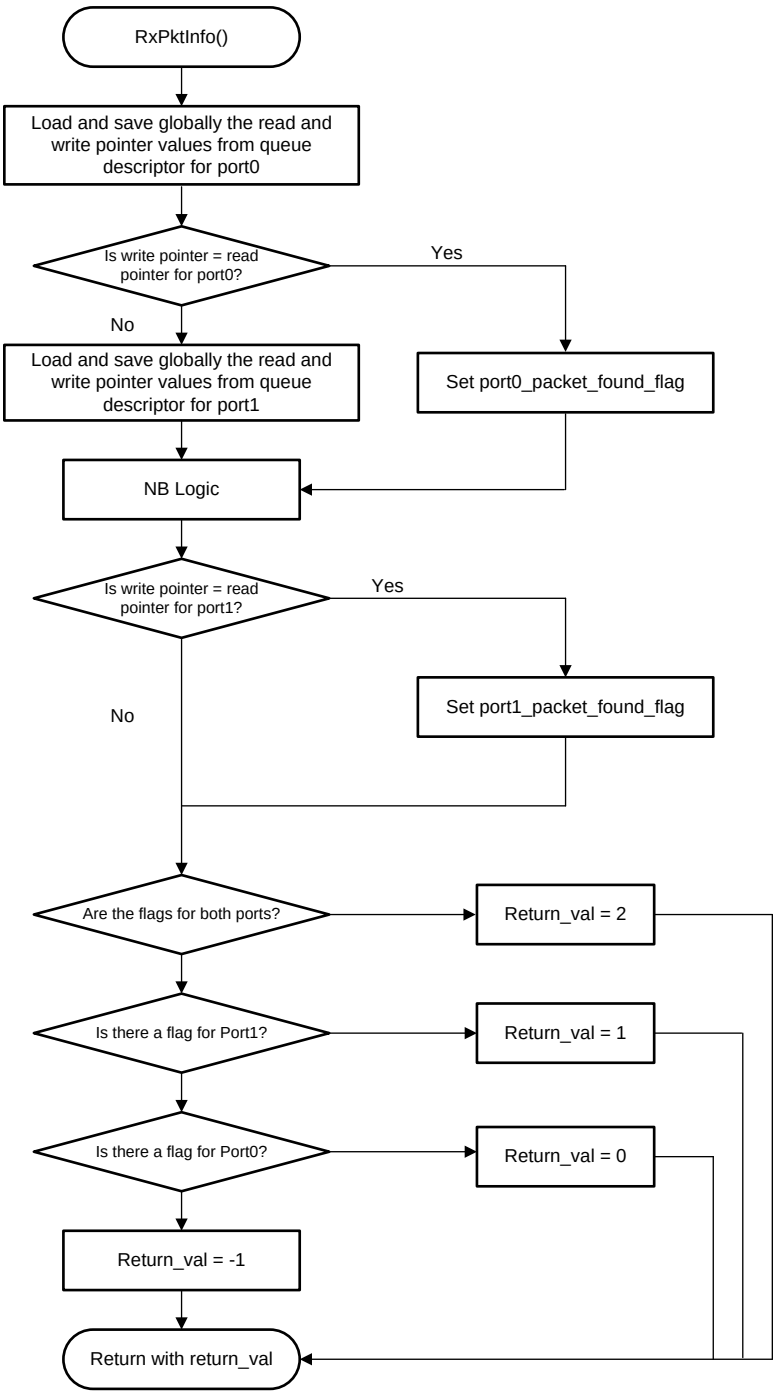


Figure 11. RxPktInfo Function

2.4.4 Copy_Packet_PRU2ARM_ESL Function

This function is actually responsible for copying the packet from queue to the ARM buffer for the port number, which it gets as input parameter. The function first loads the packet length of the packet that was received by the MII_RT port from the buffer descriptor. The function then copies the packet from the queue in L3 memory to the ARM buffer excluding the preamble. Then the function appends the 16 bytes of ESL information with the packet inside the ARM buffer. Lastly, it updates the read pointer in the queue descriptor, handles wrap-around of the read pointer (if there is one), and returns the length of the packet that was copied excluding the preamble and including ESL information.

Figure 12 shows the functionality of this function in the form of a flowchart.

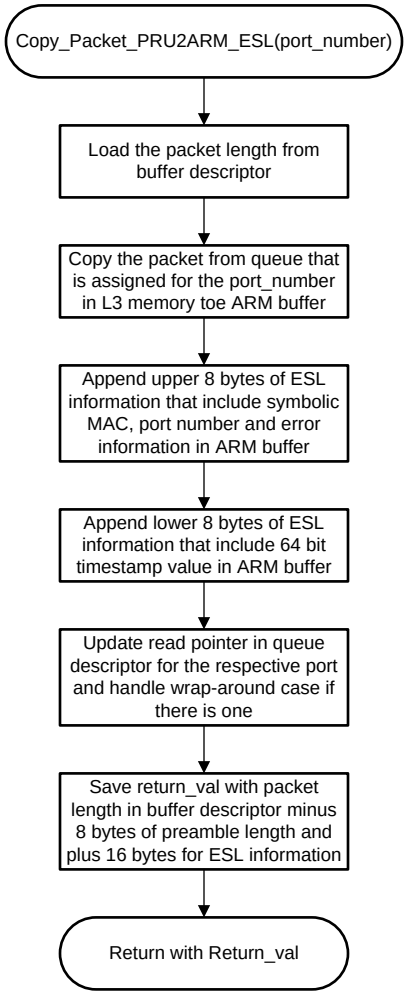


Figure 12. Copy_Packet_PRU2ARM_ESL Function

3 Getting Started Hardware and Software

3.1 Hardware

Figure 13 shows the TMDSIDK437X board.

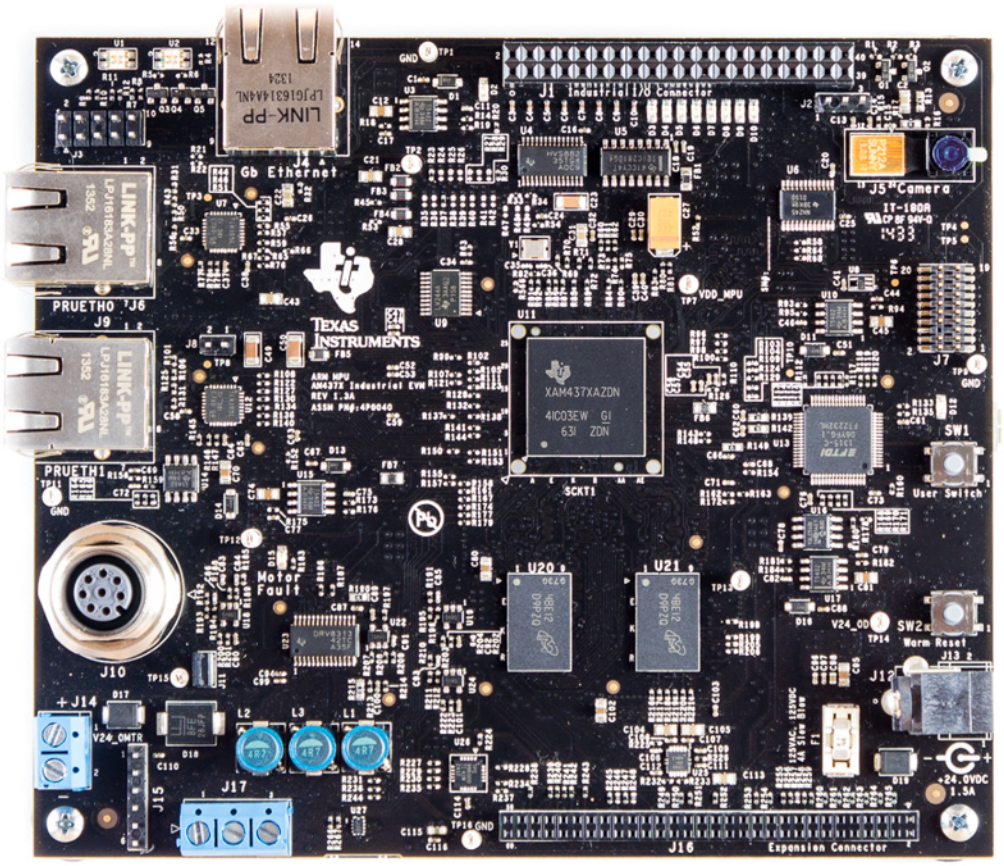


Figure 13. TMDSIDK437X Board

3.2 Software

The following hardware and software are required:

- TMDSIDK437X board
- Code Composer Studio™ (CCS) v6 or higher
- PRU Compiler for CCSv6 (install through CCS add-on)
- RTOS Processor SDK 1.0.3 for AM437x IDK
- Industrial SDK 2.1.1.2
  - SYS-BIOS (refer to Industrial SDK release notes)
  - XDC-Tools (refer to Industrial SDK release notes)
- NDK 2.25.0.09 (refer to NDK release notes and NDK reference guide)
- TI Design TIDEP0064 software download files

After installing the development tools, extract the TIDEP0065 project in the C:/TI folder. Import the PRU and ARM project into CCS from the downloaded source code file folder.

3.2.1 PRU Firmware

Once the project has been imported the PRU firmware can be compiled. The outcome of the PRU Compiler is an .out file. The .out file must get converted into a C-Header file, which is then included by the ARM application. The ARM application loads the PRU firmware header at run time into the PRU core.

Follow these steps to convert the PRU .out file into a C-Header file:

1. Go to the RTE_PRU-ICSS folder.
2. Read the readme.txt in header_gen folder.
   - If required, adopt file names and paths in build_header_pru1.bat.
3. Copy the following files from header_gen to the debug folder under the respective PRU folder.
   - build_header_pru0.bat
   - pru_header.cmd
   - build_header_pru1.bat
4. Open two CMD DOS box and navigate to the PRU0 folder under debug folder in one window and navigate to the PRU1 folder under debug folder in other window.
5. Execute the build_header_pru0.bat for the PRU0 folder and build_header_pru1.bat file for PRU1 folder: This generates the C-Header files and copies this into the ARM project include folder.

For development and testing purposes, download the .out file through JTAG to the PRU core. Note that if the ARM application is reloaded and executed, it will rewrite the PRU firmware into the PRU code.

3.2.2 ARM® Application

The ARM application initializes the PRU-ICSS subsystem, sets the pinmux for the board, configures MII_RT ports, and loads the PRU firmware. In addition, the example application transmits the packets being received on the MII_RT ports through the gigabit port to the PC where they can be seen on the Wireshark.

Compile the ARM project and load the .out file into the ARM code. Execute the application to start Rea-Time Ethernet Tracer to trace the communication on the network. Use Wireshark to validate that the packets are being captured through the gigabit port. Use breakpoints in the ARM software and read out the return variables of RxPktTransfer_ESL Function and other functions that it calls to validate the expected results.

4 Testing and Results

4.1 Test Setup

The test setup included a PROFINET network in which a PLC, acting as master, communicates with multiple slaves. Our real-time Ethernet tracer is placed between the PLC and the first slave as shown in Figure 14.

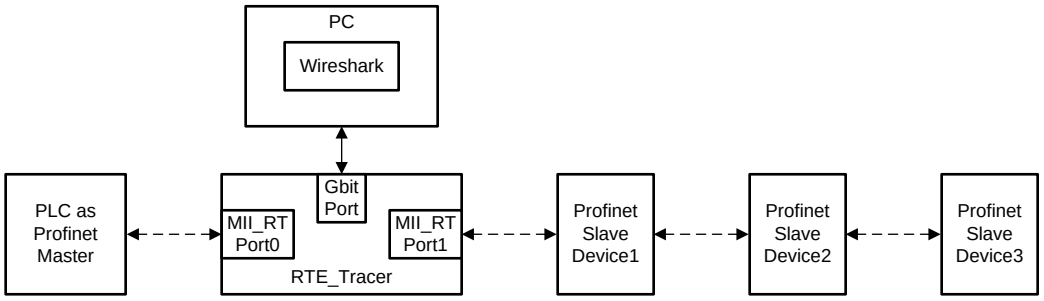


Figure 14. Test Setup Block Diagram

4.2 Wireshark™ Screenshots

Figure 15 shows a screenshot of a Wireshark capture from the test setup.

No.	Time	Source	Destination	Protocol	Length	Info
37959	68.902966275	TexasIns_85:d6:e6	PN-MC_00:01:01	PNIO	80	RTC3, ID:0x0104, Len: 44, Cycle:33287 (Valid,Primary,Problem,Run)
37960	68.903210615	Siemens_13:42:c0	PN-MC_00:01:01	PNIO	80	RTC3, ID:0x0105, Len: 44, Cycle:42824 (Invalid,Primary,Problem,Run)
37961	68.903216250	TexasIns_85:d6:e6	PN-MC_00:01:01	PNIO	80	RTC3, ID:0x0104, Len: 44, Cycle:19189 (Invalid,Backup,Ok,Stop)
37962	68.903460620	Siemens_13:42:c0	PN-MC_00:01:01	PNIO	80	RTC3, ID:0x0105, Len: 44, Cycle: 7961 (Invalid,Backup,Problem,Stop)
37963	68.903466225	TexasIns_85:d6:e6	PN-MC_00:01:01	PNIO	80	RTC3, ID:0x0104, Len: 44, Cycle:62116 (Invalid,Primary,Ok,Run)
37964	68.903710625	Siemens_13:42:c0	PN-MC_00:01:01	PNIO	80	RTC3, ID:0x0105, Len: 44, Cycle:14253 (Invalid,Backup,Ok,Run)
37965	68.903716280	TexasIns_85:d6:e6	PN-MC_00:01:01	PNIO	80	RTC3, ID:0x0104, Len: 44, Cycle:55824 (Invalid,Primary,Problem,Stop)
37966	68.903960590	Siemens_13:42:c0	PN-MC_00:01:01	PNIO	80	RTC3, ID:0x0105, Len: 44, Cycle:36860 (Invalid,Primary,Ok,Stop)
37967	68.903966255	TexasIns_85:d6:e6	PN-MC_00:01:01	PNIO	80	RTC3, ID:0x0104, Len: 44, Cycle:25153 (Invalid,Backup,Problem,Run)
37968	68.904210635	Siemens_13:42:c0	PN-MC_00:01:01	PNIO	80	RTC3, ID:0x0105, Len: 44, Cycle:18190 (Valid,Backup,Problem,Run)
37969	68.904216270	TexasIns_85:d6:e6	PN-MC_00:01:01	PNIO	80	RTC3, ID:0x0104, Len: 44, Cycle:43699 (Valid,Primary,Ok,Stop)
37970	68.904460640	Siemens_13:42:c0	PN-MC_00:01:01	PNIO	80	RTC3, ID:0x0105, Len: 44, Cycle:65375 (Valid,Primary,Problem,Stop)

Figure 15. Screenshot of Wireshark™ Capture

Figure 16 shows the contents of the highlighted packet, which includes the ESL information with its timestamp value and other fields.

```
Frame 37959: 80 bytes on wire (640 bits), 80 bytes captured (640 bits) on interface 0
Ethernet II, Src: TexasIns_85:d6:e6 (d4:f5:13:85:d6:e6), Dst: PN-MC_00:01:01 (01:0e:cf:00:01:01)
   Destination: PN-MC_00:01:01 (01:0e:cf:00:01:01)
   Source: TexasIns_85:d6:e6 (d4:f5:13:85:d6:e6)
   Type: PROFINET (0x8892)
PROFINET Isochronous-Real-Time, RTC3, ID:0x0104, Len:  44, Cycle:33287 (Valid,Primary,Problem,Run)
   FrameID: 0x0104 (0x0100-0x06FF: RED: Real-Time(class=3): non redundant, normal or DFP)
   CycleCounter: 33287
   DataStatus: 0x17 (Frame: Valid and Primary, Provider: Problem and Run)
   TransferStatus: 0x37 (ignore this frame)
PROFINET IO Cyclic Service Data Unit: 44 bytes
User Data (including GAP and RTCPadding): 44 bytes
EtherCAT Switch Link
   Port: 1
   .... 0... .... .... = Alignment Error: no
   ...0 .... .... .... = Crc Error: no
   timestamp: 0x000000100af0022f
```

```
0000  01 0e cf 00 01 01 d4 f5  13 85 d6 e6 88 92 01 04   ........ ........
0010  80 80 80 80 00 00 00 00  00 00 00 00 80 80 00 80   ........ ........
0020  80 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 00  a8 08 35 00 82 07 17 37   ........ ..5....7
0040  01 01 05 10 00 00 40 00  2f 02 f0 0a 10 00 00 00   ......@. /.......
```

Figure 16. Screenshot of Packet Content From Wireshark™ With ESL Information

4.3 Performance Testing and Limitations

Performance testing was done on the RTE tracer TI Design is to find how much time it takes to process a 64-byte packet by the RxPktTransfer_ESL function in which it copies the packet from queue in L3 memory to the ARM buffer. Testing was also completed to determine the time it takes the NDK socket to transmit the packet through the gigabit link using the send function. This testing was done using GPIO toggling where the GPIO pin was enabled as output. The pin was set to *1* for the duration of the execution of each function.

Figure 17 shows the time it takes to process a packet by RxPktTransder_ESL function, which comes out to be 5 µs.

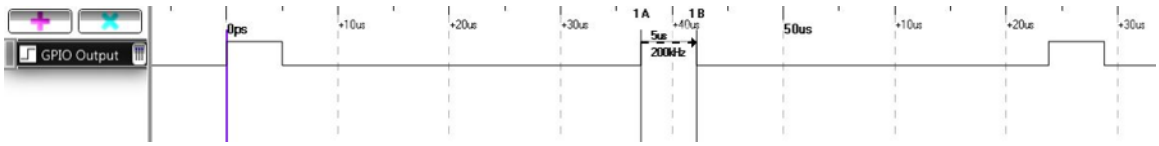


Figure 17. Time to Process a Packet by RxPktTransfer_ESL Function

Figure 18 shows the time it takes to transmit the packet through the NDK socket *send* function.

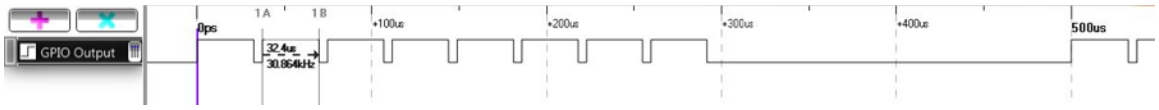


Figure 18. Time to Transmit a Packet Through NDK Socket Send Function

On the cable between the multiple Ethernet packets, there is a 960-ns duration inter-frame gap (IFG). If sent one after another, the shortest packets (64 bytes plus 8 bytes of preamble) can take only the time to send a 72 bytes plus 960 ns for IFG, which comes out to be 6.72 µs. However, this TI design takes roughly 37.4-µs total (5 µs to process and 33.2 µs to send the packet). If such situation arises that the packets are sent one after another without a time difference between the two packets, being less than 37.4 µs, the tracer can drop the packets if many of those packets are received and if the overflow count can be seen to increase.

5 Design Files

5.1 Schematics

To download the schematics, see the design files at TIDEP0064.

5.2 Bill of Materials

To download the bill of materials (BOM), see the design files at TIDEP0064.

5.3 PCB Layout Recommendations

5.3.1 Layout Prints

To download the layer plots, see the design files at TIDEP0064.

5.4 Altium Project

To download the Altium project files, see the design files at TIDEP0064.

5.5 Gerber Files

To download the Gerber files, see the design files at TIDEP0064

5.6 Assembly Drawings

To download the assembly drawings, see the design files at TIDEP0064.

6 Software Files

To download the software files, see the design files at TIDEP0064.

7 Terminology

CCS - Code Composer Studio

ICSS - Industrial Communication Subsystem

PLC - Programmable Logic Controller

PRU - Programmable Real-time Unit

8 About the Author

MUHAMMAD HAISEM KHAN is a master's student at the University of Stuttgart, Germany. He is pursuing specialization in embedded systems under the INFOTECH program at his university. He has significant knowledge in the fields of industrial automation, embedded systems, real-time systems, and real-time programming. As per his curriculum and interests, he was a master's intern in the Factory Automation and Control Team in Texas Instruments Freising, Germany. He was responsible for the implementation of PRU firmware and ARM software in RTE_Tracer project under Thomas' supervision. Haisem acquired his Bachelor's Degree in Electrical (Telecommunication) Engineering from the National University of Sciences & Technology (NUST) in Islamabad, Pakistan.

THOMAS MAUER is a System Engineer in the Factory Automation and Control Team at Texas Instruments Freising. He is responsible for developing reference design solutions for the industrial segment. Thomas brings his extensive experience in industrial communications like Industrial Ethernet and fieldbuses and industrial applications to this role. Thomas earned his degree in Electrical Engineering (Dipl. Ing. (FH)) at the University of Applied Sciences in Wiesbaden, Germany.

IMPORTANT NOTICE FOR TI REFERENCE DESIGNS

Texas Instruments Incorporated ('TI") reference designs are solely intended to assist designers ("Designer(s)") who are developing systems that incorporate TI products. TI has not conducted any testing other than that specifically described in the published documentation for a particular reference design.

TI's provision of reference designs and any other technical, applications or design advice, quality characterization, reliability data or other information or services does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such reference designs or other items.

TI reserves the right to make corrections, enhancements, improvements and other changes to its reference designs and other items.

Designer understands and agrees that Designer remains responsible for using its independent analysis, evaluation and judgment in designing Designer's systems and products, and has full and exclusive responsibility to assure the safety of its products and compliance of its products (and of all TI products used in or for such Designer's products) with all applicable regulations, laws and other applicable requirements. Designer represents that, with respect to its applications, it has all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. Designer agrees that prior to using or distributing any systems that include TI products, Designer will thoroughly test such systems and the functionality of such TI products as used in such systems. Designer may not use any TI products in life-critical medical equipment unless authorized officers of the parties have executed a special contract specifically governing such use. Life-critical medical equipment is medical equipment where failure of such equipment would cause serious bodily injury or death (e.g., life support, pacemakers, defibrillators, heart pumps, neurostimulators, and implantables). Such equipment includes, without limitation, all medical devices identified by the U.S. Food and Drug Administration as Class III devices and equivalent classifications outside the U.S.

Designers are authorized to use, copy and modify any individual TI reference design only in connection with the development of end products that include the TI product(s) identified in that reference design. HOWEVER, NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of the reference design or other items described above may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI REFERENCE DESIGNS AND OTHER ITEMS DESCRIBED ABOVE ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING THE REFERENCE DESIGNS OR USE OF THE REFERENCE DESIGNS, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY DESIGNERS AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS AS DESCRIBED IN A TI REFERENCE DESIGN OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF THE REFERENCE DESIGNS OR USE OF THE REFERENCE DESIGNS, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

TI's standard terms of sale for semiconductor products (http://www.ti.com/sc/docs/stdterms.htm) apply to the sale of packaged integrated circuit products. Additional terms may apply to the use or sale of other types of TI products and services.

Designer will fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of Designer's non-compliance with the terms and provisions of this Notice.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265