TI Designs

High-Availability Seamless Redundancy (HSR) Ethernet for Substation Automation

TI Designs

This TI Design implements a solution for high-reliability, low-latency network communications for substation automation equipment in Smart Grid transmission and distribution networks. It supports the high-availability seamless redundancy (HSR) specification in the IEC 62439 standard and Precision Time Protocol (PTP) specification in IEEE 1588. This solution is a lower-cost alternative to FPGA approaches and provides the flexibility and performance to add features such as IEC 61850 support without additional components.

Design Resources

TIDEP0053	Design Folder
AM3359	Product Folder
TLK110	Product Folder
TPS65910	Product Folder
TMDSICE3359	Tools Folder
SYSBIOSSDK-IND-SITARA	Tools Folder

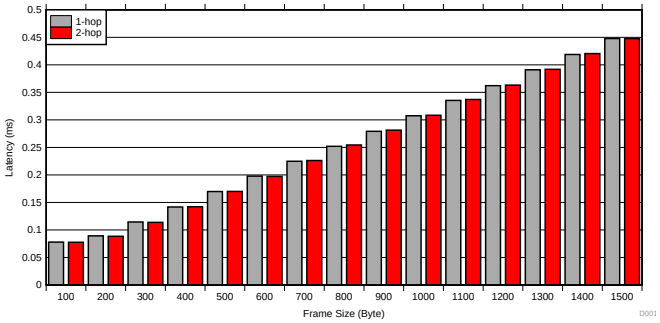
TI E2E™ Community — ASK Our E2E Experts

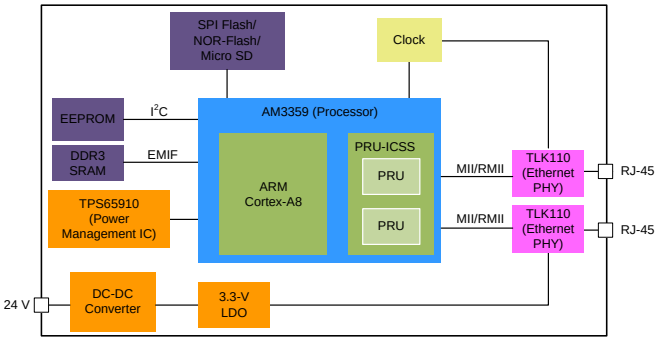
Design Features

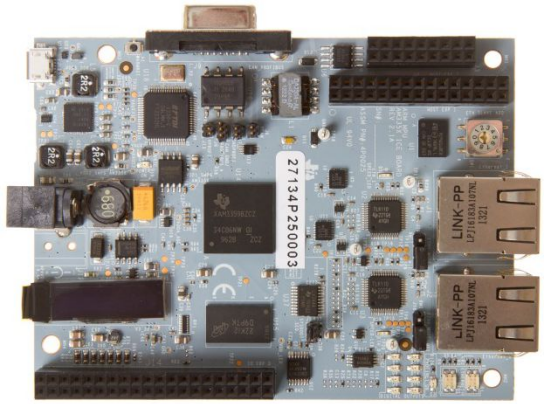
- Compliant to IEC 62439-3 Clause 5 Specification for HSR Ethernet Communications
- IEEE 1588 Peer-to-Peer Transparent Clock Profile for Network Synchronization
- Traffic Filtering Based on VLAN IDs, Multicast and Broadcast Support, and Built-in Storm Prevention Mechanism
- Zero Recovery Time in Case of Network Failure
- Dual-Ported, Full-Duplex 100-Mbps Ethernet
- Fully Programmable Solution Provides Platform for Integration of Additional Applications

Featured Applications

- Substation and Distribution Automation
- Protection Relays
- Smart Grid Communications







1 Background

A substation is a key component of the electricity grid infrastructure, located everywhere from power generation facilities throughout the distribution network to the low-voltage feeders serving residences and businesses. Substations play a key role in transforming voltage levels for transmission and performing important functions such as switching, monitoring, and protecting subsystems to maintain grid efficiency and reliability. Traditional substation systems focused on fault monitoring, which can be manually fixed by switching to backup subsystems.

Consumers, regulators, and grid operators demand ever-greater reliability of electricity delivery. The introduction of automatic switching and protection of subsystems place high demands for the increasing automation of substation operations and communications to monitor grid conditions and communicate that information to grid operators reliably and rapidly.

Operators need to be able to continually monitor the health of their network and take action to maintain its operation with high speed. This leads to the requirement for reliable and low-latency communications between the operator's control center and high-value nodes such as substations.

The International Electro-technical Commission (IEC) has released specifications for industrial Ethernet communications under the IEC 62439 standard. The HSR specification is a static redundancy, Ethernet-based protocol, which supports critical real-time systems that require continuous monitoring.

The IEEE 1588 PTP is designed to provide high-accuracy network time synchronization between subsystems.

2 System Description

This design provides a reliable high-speed HSR communication solution with high-accuracy time synchronization (IEEE 1588) for substation automation. This design implements HSR compliant with IEC 62439-3 Clause 5 and IEEE 1588/PTP v2.

This is a cost-effective alternative to ASIC or FPGA-based Ethernet solutions while delivering equivalent performance. The programmable nature of the solution allows operating different redundancy Ethernet protocols without modifying hardware and adding applications such as IEC 61850 without requiring extra system cost.

Figure 1 shows the overall system architecture. The HSR supports dual-port full duplex Ethernet communication and IEEE 1588 provides high-accuracy time synchronization between network devices. The system includes Ethernet PHY as layer-1, Ethernet MAC, HSR, and IEEE 1588 protocols as layer-2. An application example project, provided in the software development kit (SDK) software package, allows developers to build their own applications on top of those protocols via direct API calls or using NDK (network development kit) libraries to import network stacks. Typically, hard real-time application can be implemented through using direct API call and TCP/IP-based application can be implemented using NDK stacks provided by TI.

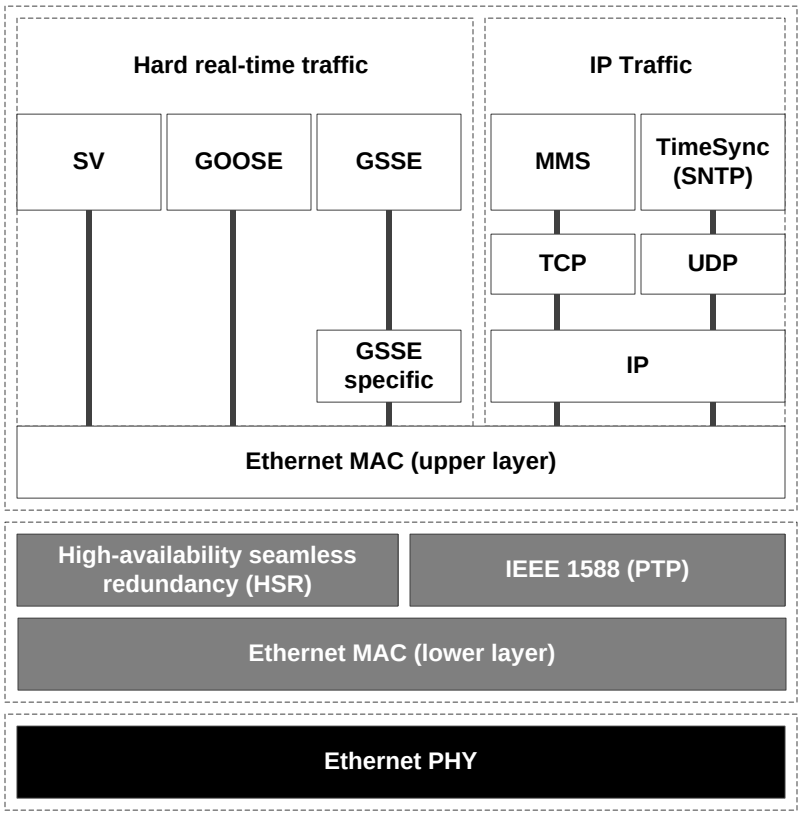


Figure 1. System Architecture

2.1 HSR

HSR is a redundancy protocol for Ethernet networks, standardized as IEC 62439-3 Clause 5, and is selected as one of the redundancy protocols for substation automation in the IEC 61850 standard. HSR is application-protocol independent and can be used by most industrial Ethernet applications that require reliable high-speed communications.

The HSR supports ring topology. Compared to star topology where typical Ethernet is operating, the advantage of ring topology is that there is no requirement on the infrastructure (for example, router) to form networks, which saves installation cost. A disadvantage is that it might cause more delays to reach at the destination if the packet goes through multiple hops.

In this design, the transmission delay over multiple hops was minimized by introducing cut-through mode. The cut-through mode is when a node receives packets that are partially decoded up to the destination address field and, if the final destination is not the node, the packets are forwarded to the TX port. In addition, this design includes built-in HSR supervision, storm prevention mechanism, and VLAN support. Figure 2 shows HSR ring topology and how the packet reaches at destination. Once a packet is generated at Node 1, the packet is distributed in both directions until being consumed by the destination at Node 4. The redundancy provides zero recovery time in case when the packet fails to be delivered in one direction.

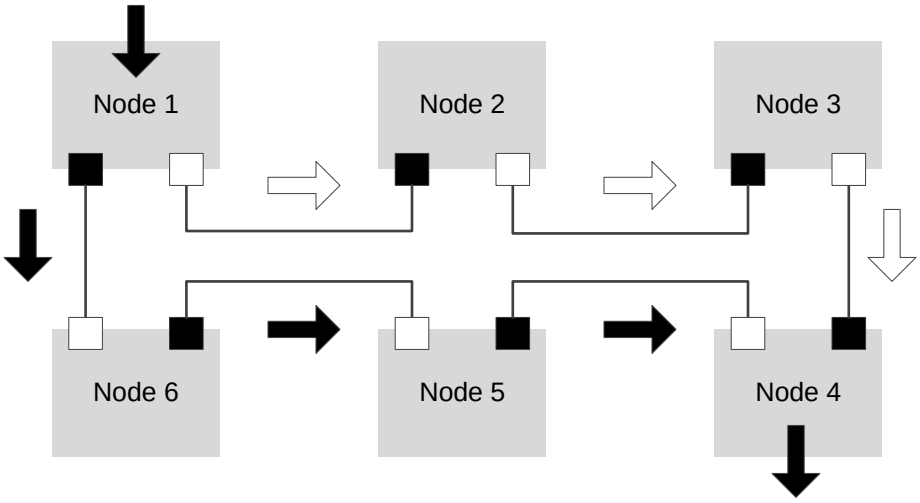


Figure 2. HSR Ring Topology

2.2 IEEE 1588 (PTP)

The IEEE 1588 is a protocol used to provide high-accuracy time synchronization over networks. Originally defined in the IEEE 1588 standard, this is designed to fill a niche not well served by NTP and GPS [2]. In this design with the HSR protocol, IEEE 1588 v2 peer-to-peer transparent clock profile is supported at layer 2 to synchronize network time by measuring mean path delay using peer delay request and response mechanism. The PTP supports transmissions over IEEE 802.3, and only multicast PTP messages shall be used.

2.3 Ethernet PHY

Based on IEEE standard 802.3, the Ethernet PHY is responsible for transmitting and receiving data over Ethernet lines.

3 Block Diagram

Figure 3 shows block diagram. The primary devices for this design are the AM3359, TLK110, and TPS65910. The AM3359 ARM™ Cortex-A8 microprocessor is the host processor used to support HSR, IEEE 1588, and user applications running under a TI RTOS environment. The TLK110 is an Ethernet PHY device and, in this design, two TLK110s are used to create redundant Ethernet communications. The TPS65910 is an integrated power management IC (PMIC) with four DC/DCs, eight LDOs, and an RTC.

This design uses these devices because of the following:

- The PRU-ICSS subsystem allows independent operation for real-time communication stacks.
- The high-performance ARM core allows support for the real-time applications for substation automation.
- The programmable flexible software design allows upgrades to different Ethernet-based redundancy protocols without hardware modification.
- The high-performance Ethernet PHY meets the requirements of IEEE 802.3 with high margins in terms of cross-talk and alien noise.

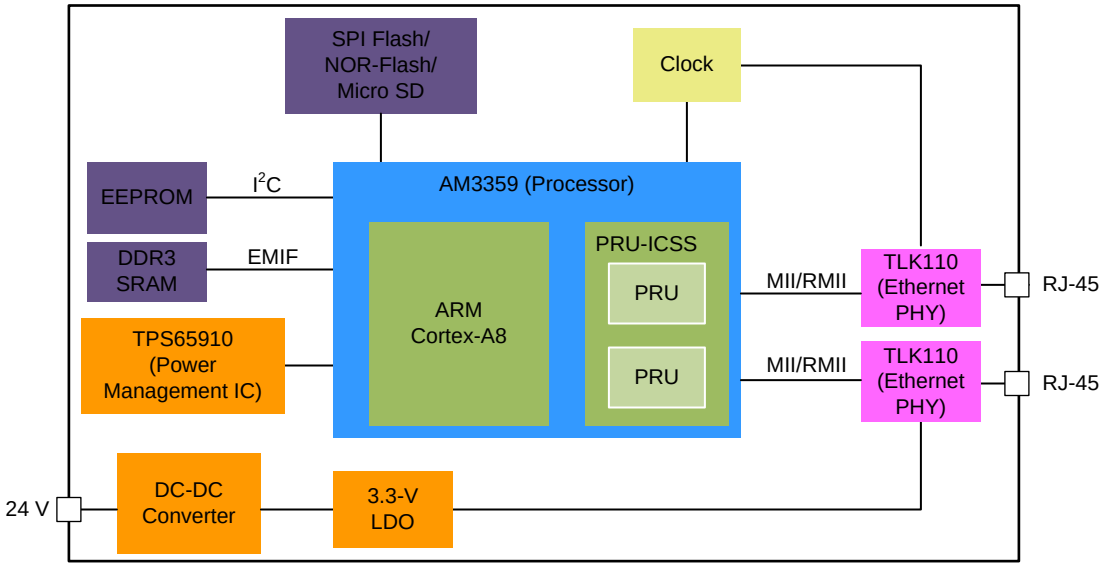


Figure 3. Block Diagram

3.1 AM3359

Based on the ARM Cortex-A8 processor, the AM3359 microprocessors are enhanced with image, graphics processing, peripherals, and industrial interface options for HSR and IEEE 1588. The Programmable Real-time Unit and Industrial Communication Subsystem (PRU-ICSS) is separate from the ARM core allowing for independent operation and clocking for greater efficiency and flexibility. The PRU-ICSS unit contains two PRUs, each of which includes 32-bit RISC processor capable of running at 200 MHz, that support real-time protocol for HSR and IEEE 1588, and supports additional interfaces of media independent interface (MII) and reduced media independent interface (RMII) to connect to the Ethernet PHY devices directly.

Additionally, the programmable nature of the PRU-ICSS, along with its access to pins, events and all system-on-chip (SoC) resources, provides flexibility in implementing fast, real-time responses and specialized data handling [3].

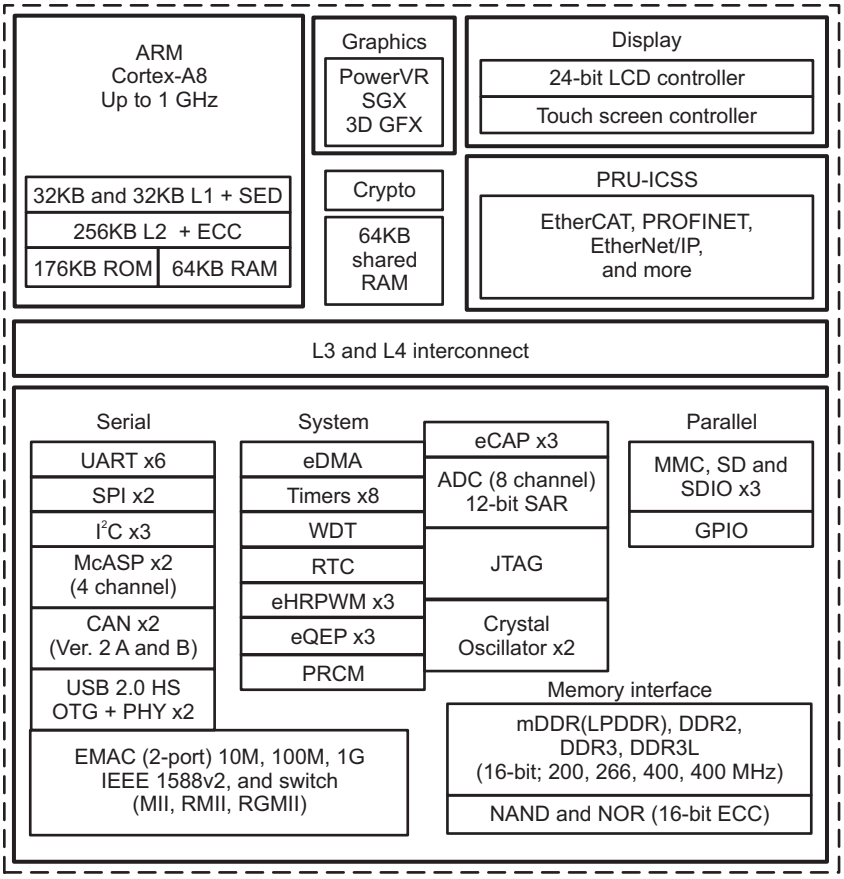


Figure 4. AM3359 Functional Block Diagram [3]

3.2 TLK110

The TLK110 is a single-port Ethernet PHY for 10BASE-TX and 100BASE-TX signaling. This device integrates all the physical-layer functions needed to transmit and receive data on standard twisted-pair cables. The TLK110 supports the standard MII and RMII for direct connection to a media access controller (MAC). The TLK110 is designed for power-supply flexibility, and can operate with a single 3.3-V power supply or with combinations of 3.3-V and 1.55-V power supplies for reduced power operation. The TLK110 uses mixed-signal processing to perform equalization, data recovery, and error correction to achieve robust operation over CAT 5 twisted-pair wiring. This device not only meets the requirements of IEEE 802.3, but maintains high margins in terms of cross-talk and alien noise [4].

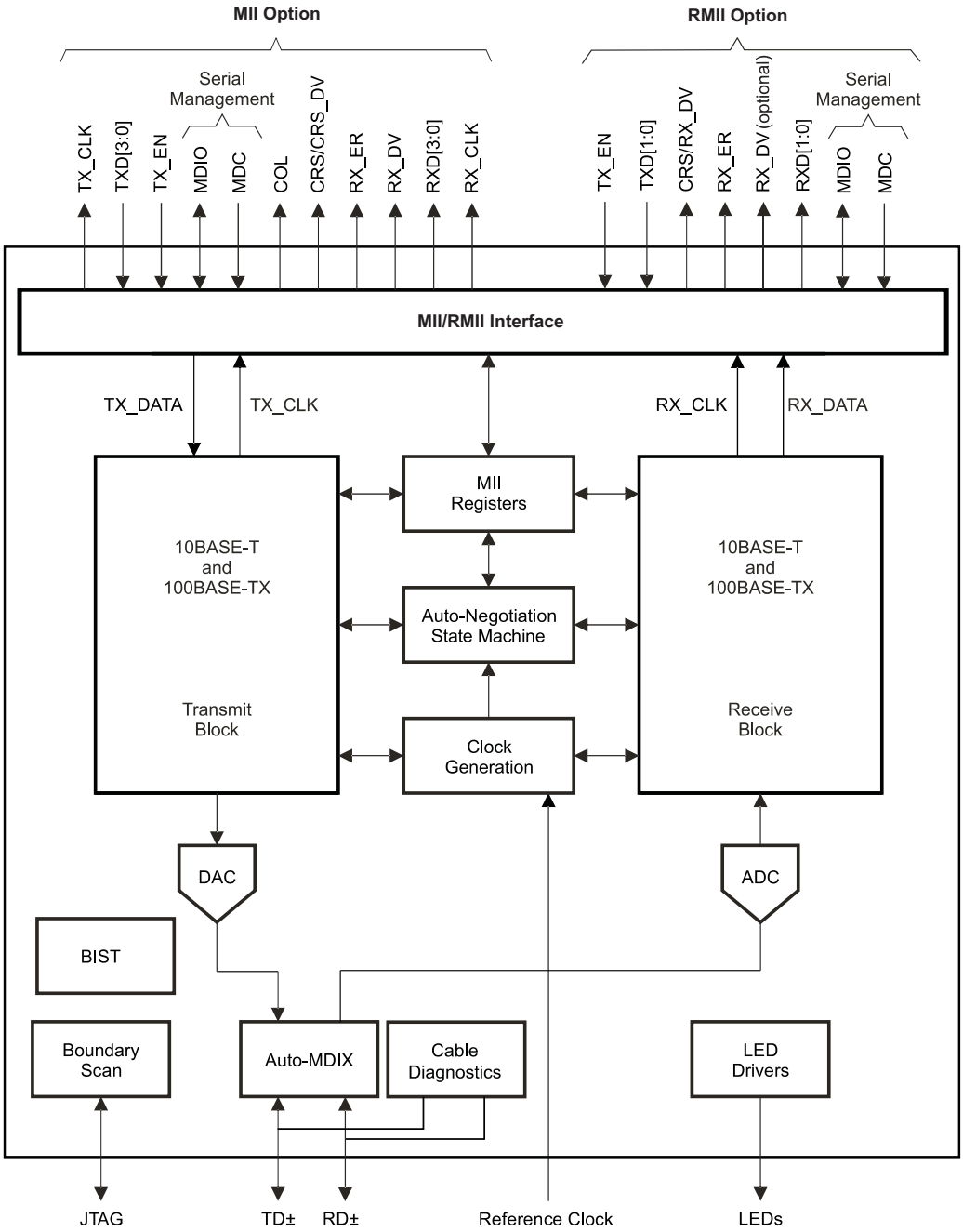


Figure 5. TLK110 Functional Block Diagram [4]

3.3 TPS65910

The TPS65910 is an integrated power-management IC that provides three step-down converters, one step-up converter, and eight LDOs and is designed to support the specific power requirements of OMAP-based applications. Two of the step-down converters power the dual processor cores and are controllable by a dedicated class-3 SmartReflex interface for optimum power savings. The third converter provides power for the I/Os and memory in the system. The device includes eight general-purpose LDOs providing a wide range of voltage and current capabilities. The LDOs are fully controllable by the I^2C interface. The use of the LDOs is flexible; they are intended to be used as follows: Two LDOs are designated to power the PLL and video DAC supply rails on the OMAP-based processors, four general-purpose auxiliary LDOs power other devices in the system, and two LDOs power DDR memory supplies in applications requiring these memories. In addition to the power resources, the device contains an embedded power controller (EPC) to manage the power sequencing requirements of the OMAP systems and an RTC [5].

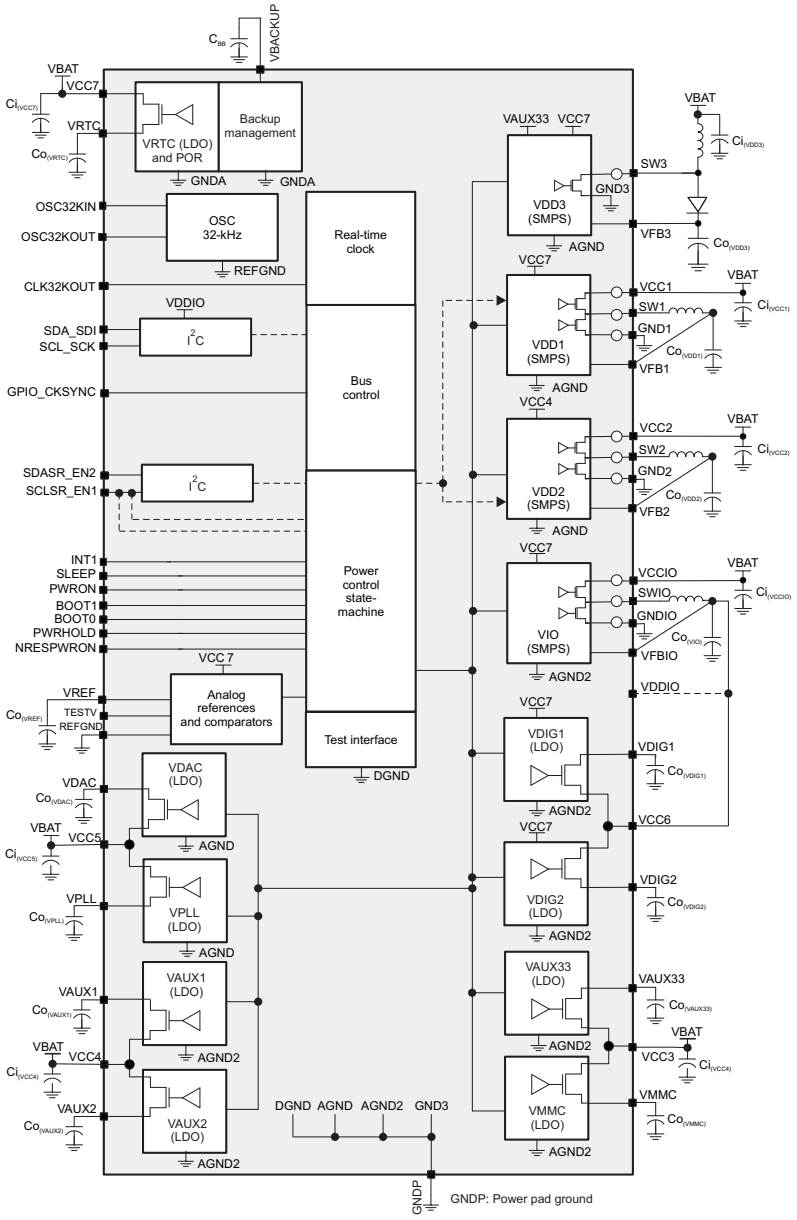


Figure 6. TPS65910 Functional Block Diagram [5]

4 Getting Started Hardware

Figure 7 shows AM3359 Industrial Communication Engine (ICE) EVM revision 2.1A. In addition to the core devices of the AM3359 and TLK110, the EVM includes various flash devices to be supported by AM3359, DDR memory, and power management devices. The EVM is designed to support multiple communication standards by providing various interfaces such as Ethernet, CAN, and RS-485.

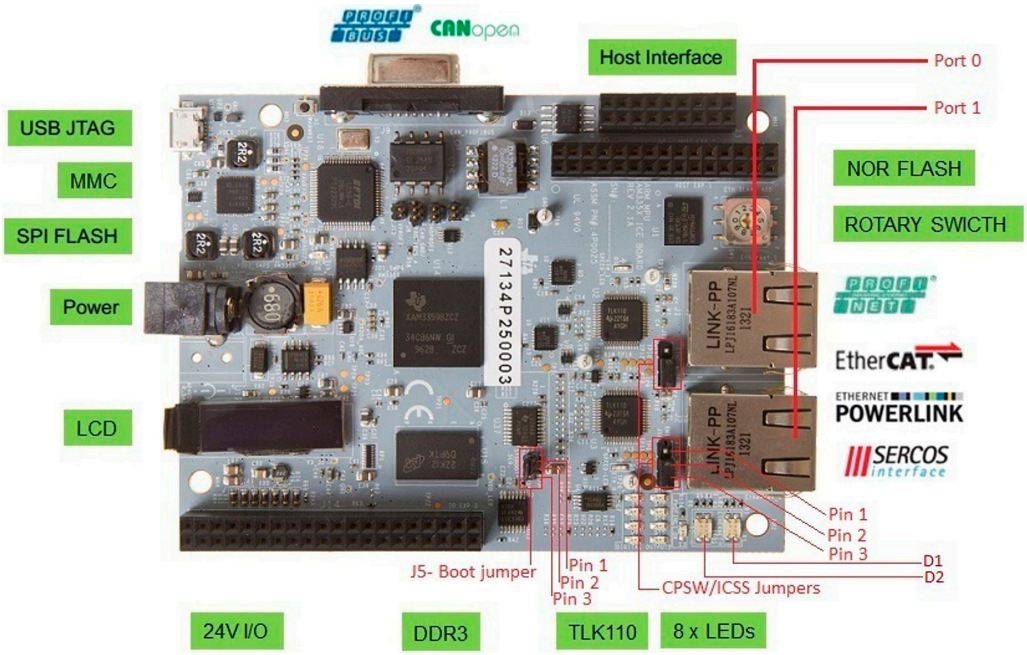


Figure 7. ICE EVM Rev 2.1A

4.1 EVM Configuration

The ICE EVM provides multiple boot options of NOR-flash, SPI-Flash, and MMC/SD boot. The boot mode can be configured with J5. Table 1 summarizes the jumper setting.

Table 1. Boot Options

BOOT MODE	JUMPER (J5) CONFIGURATION
NOR-Flash	Close pin 1-2
SPI-Flash	Close pin 2-3
MMC/SD	Close pin 2-3

4.2 Three-Node Setup Example

Each node has 2 Ethernet ports and there is no specific requirement on which ports have to be connected between nodes. Figure 8 shows an example of a 3-node setup. For testing purpose, nodes can be connected to serial terminal program with the baud rate of 115,200 bps by default.

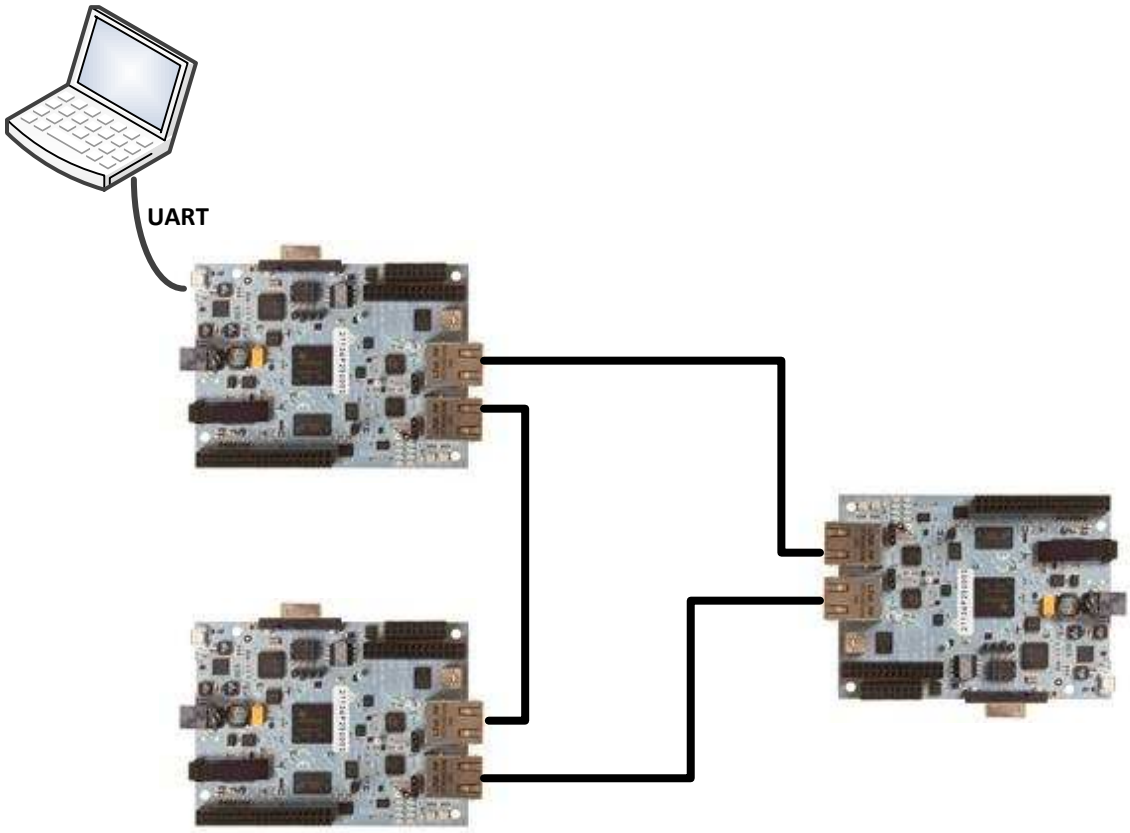


Figure 8. Three-Node Network Setup

5 Getting Started Firmware

This section provides step-by-step procedure to develop an application using HSR and PTP protocols based on the SYSBIOS industrial SDK for Sitara Processor. The first step is to check out the SDK package of "SYSBIOSSDK-IND-SITARA" with the version of v2.1.1 or above. After installing the SDK package, it is required to set an environment variable of "IA_SDK_HOME" to the SDK installation directory. Figure 9 shows the default SDK installation directory.

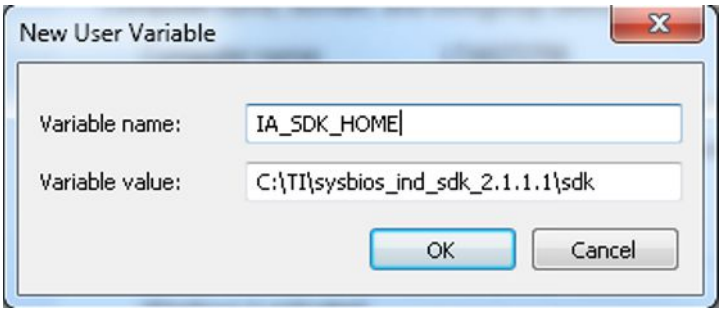


Figure 9. IA_SDK_HOME

For more details on the software architecture and system/tool requirements, see the user's guide in $(IA_SDK_HOME)\docs\.

5.1 Overview

The baseline for application development is the hsr_app CCS project in $(IA_SDK_HOME)\examples\hsr_prp_app\hsr\. The example project initializes the low-level drivers, and creates and runs some example tasks, which can be modified depending on the application requirements.

The top-level hsr_app project is built with three primary library projects: sys_bios_driver (in $(IA_SDK_HOME)\os_drivers\), ptp_lib (in $(IA_SDK_HOME)\protocols\ptp\), and hsr_lib (in $(IA_SDK_HOME)\protocols\hsr_prp\hsr\). The pre-built libraries are available in the SDK package; therefore, it is not necessary to re-build the projects if no modification is needed. The pre-built binaries are libhsr_lib.a (available in $(IA_SDK_HOME)\protocols\hsr_prp\hsr\lib\am335x\), libptp_lib.a (available in $(IA_SDK_HOME)\protocols\ptp\lib\am335x\), and libsys_bios_driver.a (available in $(IA_SDK_HOME)\os_drivers\lib\am335x\).

The sys_bios_driver project provides driver libraries for accessing device peripheral modules from a SYSBIOS application. This includes the PRU-ICSS NDK network interface management unit (NIMU) driver, PRU-ICSS low-level driver (LLD) and ICSS-EMAC LLD. The PRU-ICSS LLD is a low-level driver for the applications and drivers to interface with the PRU subsystem to be used to initialize PRU, load firmware, and configure PRU-ICSS interrupt controller. The hsr_lib project provides software stacks for the HSR protocol and the ptp_lib project implements PTP stacks.

Section 5.2 covers how to build the hsr_app CCS project, Section 5.3 covers how to modify the example project with external APIs for application development, and Section 5.4 covers how to flash and debug the applications.

5.2 Building hsr_app CCS Project

The hsr_app example project includes lower-level driver initialization, PTP and HSR protocol configurations, and task creations. There are three tasks running in the example project: taskPruss, taskLedBlink, and taskRedDebug. The taskPruss is a one-shot task to load HSR firmware into the DDR memory and then initialize necessary NDK stacks (in the example project, SNMP). The taskLedBlink is responsible for controlling LEDs and taskRedDebug is a task to print statistics and node tables periodically.

Follow these steps to build the hsr_app CCS project:

1. Open the CCS project in $(IA_SDK_HOME) \examples\hsr_prp_app\hsr\.
2. Set the build configuration to “am335x_debug” and then build.

Once the project is built, three types of binaries (hsr_app.out, hsr_app.bin, and hsr_app_ti.bin) can be found under $(IA_SDK_HOME)\examples\hsr_prp_app\hsr_app\am335x_debug\. The hsr_app.out binary can run in debug mode, and the hsr_app_ti.bin binary can be flashed into SPI_Flash to run as standalone mode.

5.3 Implementing Applications Based on the hsr_app Example Project

This section covers how to develop applications with the given hsr_app example project. Section 5.3.1 includes a list of key APIs to transmit and receive data. Section 5.3.2 covers code examples of the API usages. More details on the HSR software architecture and APIs can be found in the *ICSS EMAC LLD developers guide* [7] and the details on the NDK stacks in the NDKTCPIP product page [8].

5.3.1 List of APIs

Here is a list of key APIs for packet transmission and reception. The API to obtain the MAC address is shown here because the address should be parts of the Ethernet MAC header.

5.3.1.1 Transmit Packet

Table 2. Transmit Packet

TITLE	DESCRIPTION
Syntax	int32_t ICSS_EmacTxPacket(ICSSEMAC_Handle icssEmacHandle,
	const uint8_t* srcAddress,
	int32_t portNumber,
	int32_t queuePriority,
	int32_t lengthOfPacket);
Description	API to transmit a Packet. This function uses ICSS_EmacTxPacketEnqueue API to copy the packet to TX queue.
Parameters	@param icssEmacHandle [IN] handle to ICSS_EMAC Instance.
	@param srcAddress [IN] the TX buffer pointer where the frame to be transmitted resides. The TX frame is ETHERNET MAC PDU frame.
	@param portNumber [IN] Port on which frame has to be transmitted. Valid value is 1 (ICSS_EMAC_PORT_0) for HSR
	@param queuePriority [IN] Queue number in which frame will be queued for transmission. Valid values are 0 (ICSS_EMAC_QUEUE1) to 3 (ICSS_EMAC_QUEUE4)
	@param lengthOfPacket [IN] MAC PDU frame length in bytes
Return value	@retval 0 on success, <0 on failure

5.3.1.2 Receive Packet

Table 3. Receive Packet

TITLE	DESCRIPTION
Syntax	int32_t ICSS_EmacRxPktGet(ICSSEMAC_Handle icssEmacHandle,
	uint32_t destAddress,
	int32_t queueNumber,
	int32_t* port,
	int32_t* more);
Description	Retrieves a frame from a host queue and copies it in the allocated stack buffer. The API copies the packet from Host Queue to any other memory location. The host queue resides in L3OCMCRAM and this function can be used to transfer the packet to the high level buffers.
Parameters	@param icssEmacHandle [IN] handle to ICSS_EMAC Instance.
	@param destAddress [IN] Base address of data buffer where received frame has to be stored
	@param queueNumber [IN] Receive queue from which frame has to be copied
	@param port [OUT] Returns port number on which frame was received. The user who calls this function can validate the Port number from which the Packet was received
	@param more [OUT] Returns more which is set to 1 if there are more frames in the queue. The user need not wait for RX interrupt to process the next packet
Return value	@retval Length of the MAC PDU frame received in number of bytes or -1 on Failure

5.3.1.3 Get Ethernet MAC Address

Table 4. Get Ethernet MAC Address

TITLE	DESCRIPTION
Syntax	void SOCCtrlGetPortMacAddr(uint32_t portNum, uint8_t *pMacAddr);
Description	This API reads the Ethernet MAC address.
Parameters	@param [IN] portNum Port number. (For HSR, valid value is 1)
	@param [OUT] pMacAddr Ethernet Address
Return Value	NULL

5.3.2 Examples of API Usage

This subsection shows examples of API usage to transmit and receive packets.

5.3.2.1 Transmit Packet

There are two ways to transmit packets: one is to call the TX API directly and one is to use NDK stack lib [8]. The NDK stack lib provides an easy way to build a network-enabled application with no additional effort on either building the network stacks or interfacing to the underlying HSR protocol. If the application is time-critical and network stacks are not necessary, it is recommended to call the TX API directly.

Figure 10 shows an example of transmitting packet with TX API of ICSS_EmacTxPacket. The port number (the third argument) means which port will be used for transmission. For the HSR protocol, this is ignored because the HSR transmits a packet through both ports. The valid values for the queue priority (the fourth argument) can be from 0 (ICSS_EMAC_QUEUE1) to 3 (ICSS_EMAC_QUEUE4). The lower value stands for a higher priority and transmission occurs in the order of priority. The packet length (the fifth argument) should be MAC PDU frame size.

The second argument points to the address where TX packet is located. The TX packet is in forms of Ethernet MAC frame that consists of Ethernet MAC header and payload. The Ethernet MAC header includes 6-byte destination address, 6-byte source address, and 2-byte Ethernet type. The 4-byte CRC is not required at the application level. The underlying HSR protocol will append the CRC with the HSR tag sub-fields in the MAC frame. In this example, 2-byte Ethernet type is set to 0x0800 (IPv4). The 6-byte source address can be obtained via SOCCtrlGetPortMacAddr.

The Ethernet type can be set to VLAN (virtual LAN)-tagged frame with optional 802.1Q header. Using VLAN tag allows prioritized packet reception based on the priority code point (PCP) subfield in 802.1Q header. In the HSR firmware, when a packet is received, the 3-bit PCP subfield of a VLAN tag is read and the packet is copied to the appropriate RX queue based on fixed mapping which maps two levels (out of eight) of QoS to one queue (out of four in total). When the Ethernet type is set to 0x0800, the packet is received at RX queue #3 (the lowest priority) at all times.

```
/**A dummy packet*/
uint8_t dummyPkt[ETHERNET_FRAME_SIZE_60] =
{0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x08,0x00,
0x45,0x00,0x00,0x2E,0x00,0x00,0x40,0x00,0x40,0x00,0x3A,0xD1};

Void taskSendPacket(UArg a0, UArg a1)
{
    /*wait for system to initialize*/
 /*Send packet in a loop every 500ms*/
 Task_sleep(5000);
 while(1)
 {
    ICSS_EmacTxPacket(emachandle, dummyPkt, ICSS_EMAC_PORT_0, 1, ETHERNET_FRAME_SIZE_60);
    Task_sleep(500);
 }
}
```

Figure 10. Transmit Packet Example

5.3.2.2 Receive Packet

Similar to TX, there are two ways to receive packets: one is to hook RX callback directly and one is to use the NDK stack lib [8]. Using the NDK stack lib allows the application to hook the RX callback with the NDK lib stacks automatically. If the application is time-critical and no need of network stacks, it is recommended to use the RX callback directly. An example of how to use the direct RX callback is given in Figure 11.

There are three steps to hook the RX callback directly. Typically, the first two steps can be done in the initialization routine. In step 1, the ethPrioQueue provides the HSR firmware to determine whether the RX interrupts pass to the NDK lib stacks or the RX callback if the callback is valid. The RX maintains four RX queues and in this example the ethPrioQueue is set to 4, which means that the HSR firmware notifies the packet reception to the NDK stack lib when receives the packet at queue #4. Otherwise, if the packet is received at queue #1 to #3, the notification goes to the RX callback directly. If the RX callback is NULL, the packets will be consumed without notifying to the application level. The ethPrioQueue can be any values. The ethPrioQueue can be set to 5 (or above) if all the packet receptions have to be handled in the RX callback directly or it can be set to 0 if all the packet receptions have to be handled by the NDK stacks. This provides you an effective way to implement your application to handle the mixed traffic of hard real-time traffic and TCP/IP traffic.

Step 2 is to hook the RX callback and the code is provided in the hsr_app example. Step 3 is to copy the RX packet into a local buffer when an interrupt occurs. In step 3, it is mandatory to call ICSS_EmacRxPktGet to flush the RX queue. Otherwise, it will block subsequent packet receptions. The hsr_app example includes the callback function to receive PTP packets and the following example code shows an example to handle both application frames and PTP frames through address comparison. Find the reference codes in icss_eip_driver.c in the ethernetip_adapter example project.

```
Step 1: Configure queue priority to decide whether RX interrupts go to NDK or RX callback.

ICSSEMAC_InitConfig* switchEmacCfg;
switchEmacCfg->ethPrioQueue = ICSS_EMAC_QUEUE4;

Step 2: Hook RX callback function
    /*Packet processing callback*/
    ((((ICSSEMAC_Object*)emachandle->object)->callBackHandle)->rxRTCallBack)->callBack =
(ICSS_EmacCallBack)processPTPFrames;
    ((((ICSSEMAC_Object*)emachandle->object)->callBackHandle)->rxRTCallBack)->userArg
=emachandle;

Step 3: Parse RX frames when interrupts occur.
void processPTPFrames(uint32_t *queue_number, void *userArg)
{
    int16_t size;
    int32_t port;
    int32_t more;
    volatile uint8_t *bytePtr;
    ICSSEMAC_Handle ptpIcssEmacHandle = (ICSSEMAC_Handle)userArg;

       uint8_t *dst_addr;
       uint32_t status;

    dst_addr=rxBuffer;
       size=ICSS_EmacRxPktGet(ptpIcssEmacHandle, (uint32_t)rxBuffer, *queue_number,
&port, &more);

       if(COMPARE_MAC(dst_addr, ptpMAC)) //PTP frame processing
       {
       bytePtr = (uint8_t *)((((ICSSEMAC_HwAttrs *)
                             ptpIcssEmacHandle->hwAttrs)->emacBaseAddrCfg)-
>sharedDataRamBaseAddr +
                             PTP_ANNOUNCE_MSG_RCVD);

       if(*bytePtr == 1)
       {
              /*clear the bit*/
              *bytePtr = 0;
              processPTPFrame(ptpIcssEmacHandle, rxBuffer, port - 1, size);
       }
       }
       else  //Application Frame processing
   {

   }
}
```

Figure 11. Receive Packet Example

5.4 Flashing Binaries to SPI Flash on ICE v2.1 Using CCS

This section covers how to flash hsr_app_ti.bin binary to SPI flash using CCS tool. More flashing options can be found in the *SYSBIOS Industrial SDK Getting Started Guide* [6]. Note that, to use the SPI boot option, to the user must configure J5 jumper to close pin 1-2 in the EVM.

5.4.1 Launching and Debugging Application in CCS

This section covers the step-by-step procedure to run the application in debug mode. For the product-related questions or bug reports, contact the TI e2e forum (http://e2e.ti.com/support/arm/).

1. Connect the USB cable to the ICE EVM.
2. Select View → Target Configurations. Right click on the required configuration in the list and select "Launch Selected Configuration".
3. Right click on the Cortex A9/A8 listed in Debug view and select Connect (Click View → Debug to view debug window if not visible).
4. Select "System Reset" by clicking Run → Reset → System Reset.
5. Select "Suspend" by clicking Run → Suspend.
6. Load GEL file and execute initialization script.
   (a) Click Tools → GEL Files
   (b) Remove the existing GEL file (TMDXICE3359.gel) and load the GEL file of "TMDXICE3359_v2_1A.gel" in $(IA_SDK_HOME)\ tools\gel\AM335x.
   (c) Once the GEL file is loaded, the scripts available will be shown in Menu → Scripts.
   (d) Click the initialization script by selecting Menu → Scripts → AM335x System Initialization → AM3359_ICE_Initialization
7. Once the initialization is completed, Select Run → Load → Load Program.

Once Step 7 is completed, the application binary (for example, hsr_app.out) can be loaded to run in the CCS debug mode.

NOTE: For a new ICEv2 EVM bring-up, select CPU RESET (HW) by Run → Reset → CPU RESET (HW) between Step 5 and 6. This is a one-time requirement.

5.4.2 Erasing and Flashing Binaries to SPI Flash on ICE v2.1

1. Complete Step 1 to 7 in Section 5.4.1.
2. Load the pre-built "isdk_spi_flasher.out" in $(IA_SDK_HOME)\tools\flasher\spi_flasher\.
3. Run the application.
4. Follow the steps the CCS console window shows. For more details, see Section "Flashing Binaries to SPI Flash on ICE V1/V2 Using CCS" in the *SYSBIOS Industrial SDK Getting Started Guide* [6].
5. For bootloader, the pre-built bootloader binary (for example, bootloader_boot_mcspi_a8host_release_ti.bin for SPI flash) can be found in $(IA_SDK_HOME)\starterware\binary\bootloader\bin\am335x-evm\gcc\.

6 Test Setup

Figure 12 shows the test setup with four nodes and each node has two Ethernet connections: each per adjacent node. For these experiments, the hsr_app project application was modified to measure the performance of delivery ratio and latency. For the target TX and RX, a PC is attached to each node to configure test modes and parameters with a serial terminal program. In addition, the underlying firmware generates background traffic such as supervision frames to discover neighbors and IEEE 802.3 encapsulated PTP frames to synchronize networks.

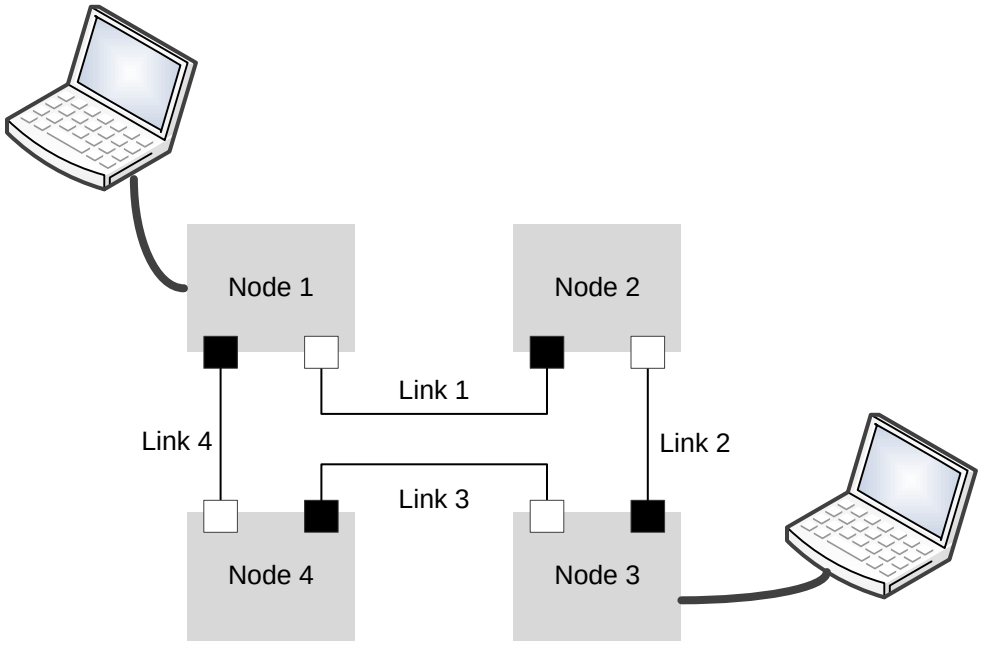


Figure 12. Test Setup (Four Nodes)

7 Test Data

The goal of these experiments is to evaluate that TI HSR/PTP solution to meets the performance requirement for substation automation. Table 5 summarizes the performance requirement for substation automation. The required communication recovery time means the time duration in which a network recovers failure and the application recovery tolerated delay (or grace time) is the time duration in that the substation tolerates an outage of the automation system. The sampled values (SV) are sampled at a nominal value of 4 kHz. Therefore, the target application recovery tolerated delay for SV is 500 µs (= 2 × ¼ kHz).

Table 5. Recovery Delay Demands as Shown in IEC 61850-5

COMMUNICATING PARTNERS	SERVICE	APPLICATION RECOVERY TOLERATED DELAY	REQUIRED COMMUNICATION RECOVERY TIME
SCADA to IED, client-server	IEC 61850-8-1	800 ms	400 ms
IED to IED interlocking	IEC 61850-8-1	12 ms (with T_{min} set to 4 ms)	4 ms
IED to IED, reverse blocking	IEC 61850-8-1	12 ms (with T_{min} set to 4 ms)	4 ms
Protection trip excluding busbar protection	IEC 61850-8-1	8 ms	4 ms
Busbar protection	IEC 61850-9-2 on station bus	< 1 ms	Bumpless
Sampled values	IEC 61850-9-2 on process bus	Less than two consecutive samples	Bumpless

7.1 Latency

The goal of this experiment is to validate if the latency performance meets the requirement for substation automation applications. To measure the latency, one node is configured as TX from 100 to 1500 bytes in increments of 100 bytes, and another node is configured as echo-back mode to send the RX packet back to the TX packet. The round-trip delay is measured at the originator of the packet by calculating the time gap between TX and RX. Then, the latency is calculated by a half of the round-trip delay. The latency measurement was performed five times and the latency was averaged. The latency is considered as one-way delay based on the definition in the IEC/TR 61850-90-4.

To validate the impact of number of hops on the overall latency, the latency is measured over 1-hop and 2-hop network to compare the gap. In Figure 10, to create 2-hop network, Node 1 is configured as TX mode and Node 3 is configured as echo-back RX mode. Similarly, to create 1-hop network, Node 1 is configured as TX mode and Node 2 is configured as echo-back RX mode. The latency is measured at Node 1.

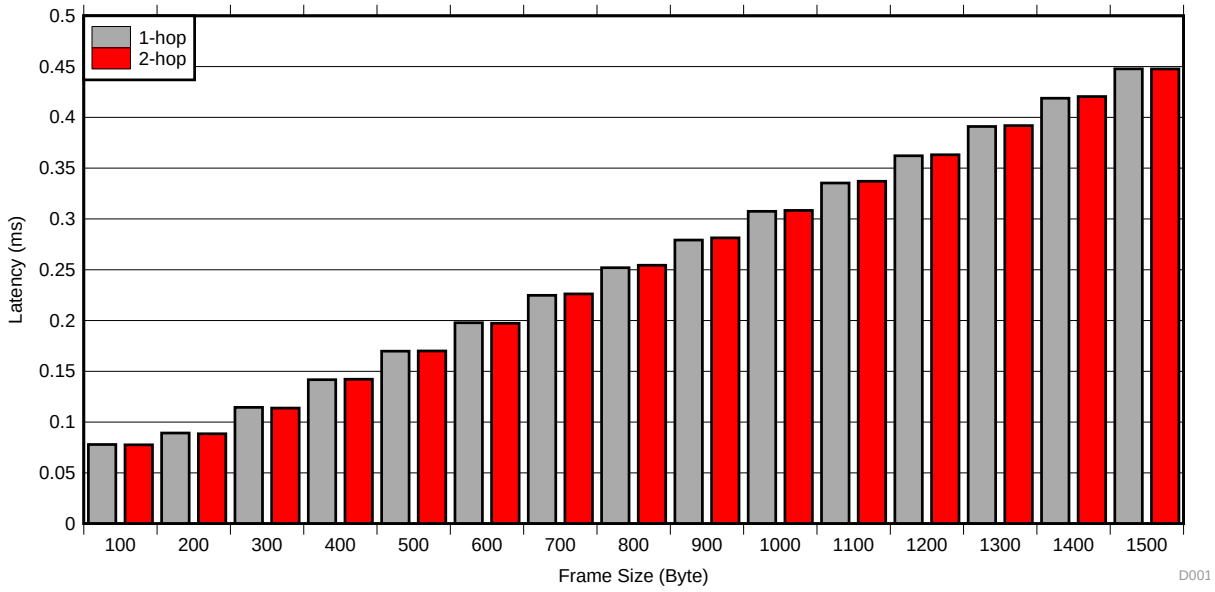


Figure 13. Latency Performance

Figure 13 shows latency performance as a function of frame size. The X-axis shows frame size in bytes and the Y-axis shows latency in milliseconds. The result shows that latency performance with maximum frame size of 1500 bytes over 2-hop meets the delay requirement (≤ 500 µs) for sampled values application. In addition, the result shows that the additional delay incurred by a single hop is very negligible due to the cut-through mechanism implemented in our HSR firmware. From this experiment, the worst case delay gap shows 2.4 µs.

Table 6 shows timing profile for TX and RX processing. The TX processing time is measured by comparing the timestamps at the time before/after calling the TX API. The RX processing time is calculated by (1-hop latency-TX processing time-propagation delay), where the propagation delay is calculated by frame size in bit divided by the PHY capacity rate (100 Mbps). The timing profile shows that the RX processing time is consistently 10 µs greater than the TX processing time regardless of the frame size. This can determine the minimal time gap between consecutive transmissions to ensure that RX completes the RX frame processing.

Table 6. TX and RX Processing Time

FRAME SIZE (BYTES)	TX PROCESSING TIME (ms)	RX PROCESSING TIME (ms)
100	0.03	0.04
200	0.03	0.04
300	0.04	0.05
400	0.05	0.06
500	0.06	0.07
600	0.07	0.08
700	0.08	0.09
800	0.09	0.10
900	0.10	0.11
1000	0.11	0.12
1100	0.12	0.13
1200	0.13	0.14
1300	0.14	0.15
1400	0.15	0.16
1500	0.16	0.17

7.2 Delivery Ratio

The goal of this experiment is to verify zero network recovery time that is one of requirements for substation automation. For this purpose, the delivery ratio performance was measured while emulating the link failures by disconnecting a link intentionally in the middle of data transmissions.

For this experiment, Node 1 is configured as TX mode with 10,000 packet transmissions, 1528-byte frame size, and 1-ms packet interval. The other nodes are configured as RX mode. During the experiment, Link 1 is disconnected to emulate link failure. To validate the impact of hops and packet types on delivery ratio performance, various experiments were performed with different hops and unicast or broadcast traffic. Each experiment emulated link failure by disconnecting Link 1.

Each experiment captured the number of TX packets at Node 1 and the number of RX packets at the other nodes. The delivery ratio is calculated by the number of TX packets divided by the number of RX packets.

Table 7 shows delivery ratio performance over various scenarios. For all scenarios, the result shows a 100% delivery ratio even with link failure, which implies that link failure is recovered immediately. This is expected because redundant communication recovers the link failure immediately.

Table 7. Delivery Ratio Performance

TEST SCENARIO	DELIVERY RATIO (%)
Unicast, 1-hop	100
Unicast, 2-hop	100
Broadcast, every nodes in the network	100

8 Design Files

8.1 Schematics

To download the schematics, see the design files at TIDEP0053.

8.2 Bill of Materials

To download the bill of materials (BOM), see the design files at TIDEP0053.

8.3 Layer Prints

To download the layer prints, see the design files at TIDEP0053.

8.4 Assembly Drawings

To download the assembly drawings, see the design files at TIDEP0053.

9 Software Files

To download the software files, see the design files at TIDEP0053.

10 References

1. Wikipedia, *High-availability Seamless Redundancy* (https://en.wikipedia.org/wiki/High-availability_Seamless_Redundancy)
2. Wikipedia, *Precision Time Protocol* (https://en.wikipedia.org/wiki/Precision_Time_Protocol)
3. Texas Instruments, *AM335x Sitara™ Processors*, AM3359 Datasheet (SPRS717)
4. Texas Instruments, *PHYTER® Industrial Temperature 10/100Mbs Ethernet Physical Layer Transceiver*, TLK110 Datasheet (SLLS901)
5. Texas Instruments, *TPS65910x Integrated Power-Management Unit Top Specification*, TPS65910 Datasheet (SWCS046)
6. Texas Instruments, *SYSBIOS Industrial SDK Getting Started Guide*, TI Wiki (http://processors.wiki.ti.com/index.php/SYSBIOS_Industrial_SDK_Getting_Started_Guide)
7. Texas Instruments, *ICSS EMAC LLD developers guide*, TI Wiki (http://processors.wiki.ti.com/index.php/ICSS_EMAC_LLD_developers_guide)
8. Texas Instruments, TI-RTOS Networking, NDKTCPIP Product Page (http://www.ti.com/tool/ndktcpip)
9. University of Brescia, *IEC61850 One World, One Technology, One Standard* (PDF)

11 About the Author

WONSOO KIM is a system applications engineer at Texas Instruments, where he is responsible for providing technical support and training on communication software and system, driving system solutions for Smart Grid, and working on defining future requirements in roadmap. He received the Ph.D. degree in electrical and computer engineering from the University of Texas at Austin, Austin, TX.

IMPORTANT NOTICE FOR TI REFERENCE DESIGNS

Texas Instruments Incorporated ("TI") reference designs are solely intended to assist designers (“Buyers”) who are developing systems that incorporate TI semiconductor products (also referred to herein as “components”). Buyer understands and agrees that Buyer remains responsible for using its independent analysis, evaluation and judgment in designing Buyer’s systems and products.

TI reference designs have been created using standard laboratory conditions and engineering practices. **TI has not conducted any testing other than that specifically described in the published documentation for a particular reference design.** TI may make corrections, enhancements, improvements and other changes to its reference designs.

Buyers are authorized to use TI reference designs with the TI component(s) identified in each particular reference design and to modify the reference design in the development of their end products. HOWEVER, NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY THIRD PARTY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT, 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 components 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 such information 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 ARE PROVIDED "AS IS". TI MAKES NO WARRANTIES OR REPRESENTATIONS WITH REGARD TO THE REFERENCE DESIGNS OR USE OF THE REFERENCE DESIGNS, EXPRESS, IMPLIED OR STATUTORY, INCLUDING ACCURACY OR COMPLETENESS. TI DISCLAIMS ANY WARRANTY OF TITLE AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, QUIET ENJOYMENT, QUIET POSSESSION, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO TI REFERENCE DESIGNS OR USE THEREOF. TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY BUYERS AGAINST ANY THIRD PARTY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON A COMBINATION OF COMPONENTS PROVIDED IN A TI REFERENCE DESIGN. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, SPECIAL, INCIDENTAL, CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ON ANY THEORY OF LIABILITY AND WHETHER OR NOT TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT OF TI REFERENCE DESIGNS OR BUYER’S USE OF TI REFERENCE DESIGNS.

TI reserves the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques for TI components are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

Reproduction of significant portions of TI information in TI data books, data sheets or reference designs is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards that anticipate dangerous failures, monitor failures and their consequences, lessen the likelihood of dangerous failures and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in Buyer’s safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed an agreement specifically governing such use.

Only those TI components that TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components that have ***not*** been so designated is solely at Buyer's risk, and Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

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