*TI Designs*

# *ARM MPU With Integrated HIPERFACE DSL® Master Interface*

## TI Designs

Position feedback systems are used in many applications such as machinery, motor drives, robotics, elevators, and windmills. HIPERFACE DSL® based position encoders support a 2-wire interface to the sensor, which can be integrated in the motor cable. This design guide describes how to implement a HIPERFACE DSL protocol on the AM437x IDK in conjunction with the TIDA-00177 transceiver card.

## Design Resources

| | |
|---|---|
| TIDEP0035 | Design Folder |
| TIDA-00177 | Tools Folder |
| AM4379 | Product Folder |
| SN65HVD78 | Product Folder |
| AM437x IDK | Tools Folder |
| Industrial SDK | Software Folder |

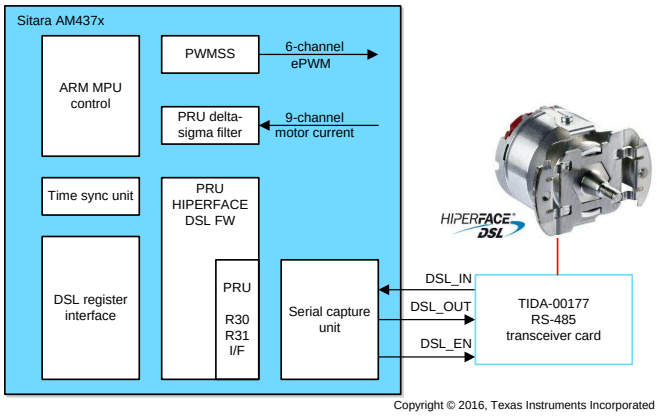
TI E2E™ Community — ASK Our E2E Experts

## Design Features

- HIPERFACE DSL Master Protocol With Register Compatible Interface to Existing FPGA IP Core
- Programmable Approach Using ICSS_L Concurrently With DS Filter and Industrial Ethernet (Single Chip Drive)
- Supports Cable Length of up to 100 m
- Line Delay Compensation
- 8× Oversampling With Sample Edge Detection
- Line Diagnostics — Quality Monitor

## Featured Applications

- Servo Drive
- Position Feedback Module
- Robotics
- Elevators
- AC Drives






An IMPORTANT NOTICE at the end of this TI reference design addresses authorized use, intellectual property matters and other important disclaimers and information.

# 1 HIPERFACE DSL Technology

HIPERFACE DSL is a motor feedback system interface for servo drives developed and maintained by the company SICK Stegmann GmbH. The communication interface consists of a physical layer interface and a data protocol to exchange information between position encoder and servo drive. This design supports cable lengths of up to 100 m by using an RS-485 based transceiver. No analog components are required for the encoder interface as all information is exchanged over a digital protocol. HIPERFACE DSL uses only two wires for powering the remote position sensor and bidirectional communication. The 2-wire approach saves cost by integrating the position feedback wires in the motor cable.

Position feedback data is part of the motor control algorithm. Typical cycle frequencies of industrial motor control are 8 kHz, 10 kHz, and 16 kHz. The minimum time to read position data from the HIPERFACE DSL encoder is 11.52 µs in a free-running mode. When position measurement is synchronized to a 16-kHz pulse width modulation (PWM) for motor control, the length of the protocol package is 12.50 µs. In this example, there are exactly five packages per motor control cycle. For higher PWM frequencies in the 50-kHz range, the protocol length can vary and goes up to 27 us with one HIPERFACE DSL packet per motor control cycle.

The primary function of a position sensor is to continue measuring and communicating the motor position, which is split into single-turn information and multi-turn information. Motor feedback systems with HIPERFACE DSL support up to 23 bits of resolution per revolution and up to 12 bits of multi-turn information. In addition to position data, there are more communication channels for parameter exchange, safe position data, and sensor hub data.

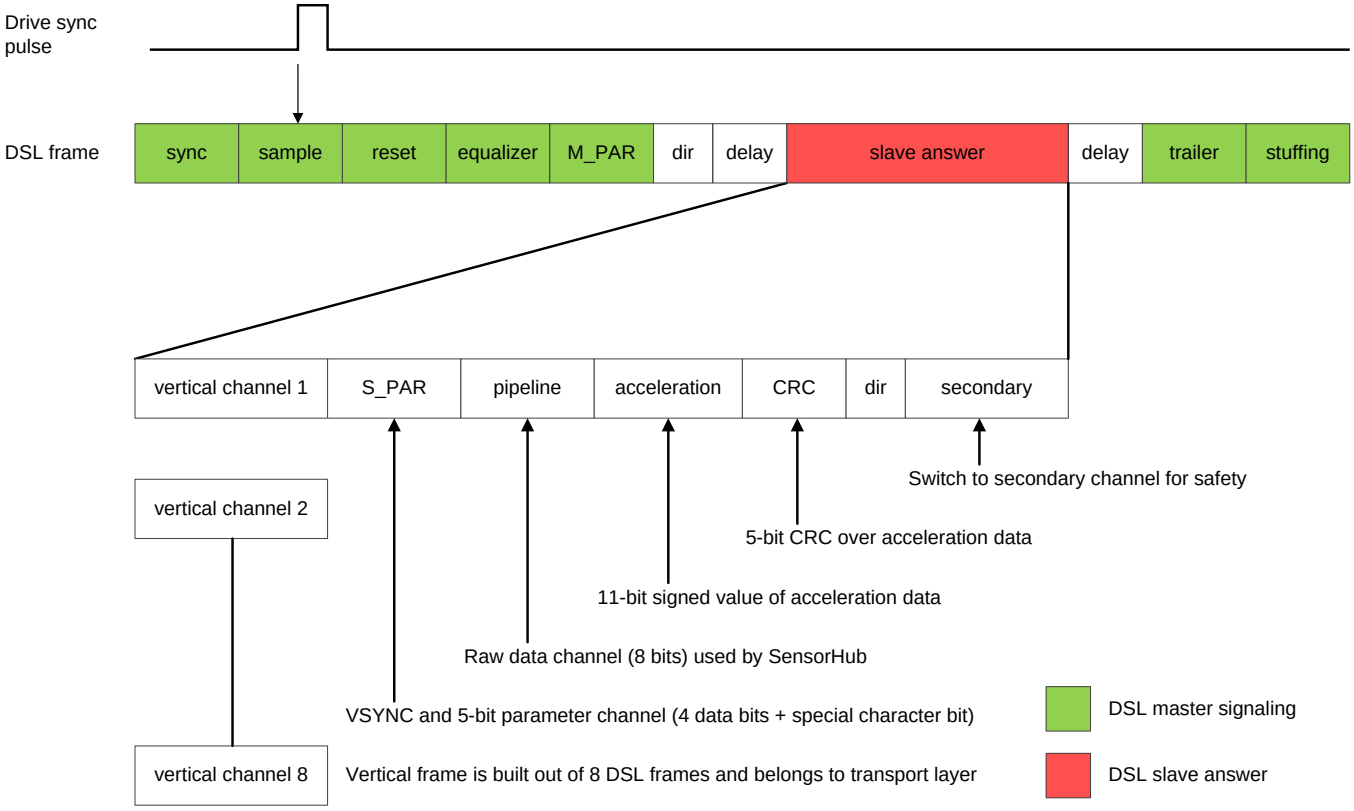



**Figure 1. HIPERFACE DSL Frame**

The physical interface operates at a raw bit rate of 9.375 Mbit/s for all cable lengths and operating states. After the start-up sequence, the data is encoded using a 8- or 10-bit line code to achieve DC balance. Each packet contains information about the increments of rotation speed and fast position. Safe position data is collected over eight packets and compared with fast position data. The parameter channel is an additional communication option that provides access to configuration and diagnostic data in form of short and long messages. In cases where remote position sensors connect more sensing devices, a separate Sensor Hub channel is available. The position encoder with functional safety support uses a secondary communication channel with a separate host interface.

Figure 1 summarizes the HIPERFACE DSL protocol over a single data line in half duplex mode. Synchronization with motor control cycle is through an external sync pulse. The sample window of master side communication embeds the sync pulse asynchronously and serves as a sampling trigger on the slave side. Each DSL frame starts with a fixed sync pattern followed by the sample window. The length of the sample window varies and ensures insertion of sync pulse edge within 5-bit time slots. A reset pattern and equalization window is sent by the master before information is first exchanged with the slave using the M_PAR field.

The number of '1' and '0' bits resulting from the previous frame's ending (trailer, stuffing) and inserting a sync pulse unbalances the DC line. The equalization window compensates the uneven distribution of energy on the wire. In cases where no external sync pulse is available for interim frames, the master interface inserts the sync pulse based on the measured frequency of the external sync pulse.

**Table 1. Master Information**

| NAME | DESCRIPTION |
|---|---|
| m_par | Parameterization data |
| m_par | Special character bit |
| reserved | |
| msync | Vertical frame synchronizer |
| first | Signals the first H-Frame of a drive cycle |

The master information field has multiple functions. Table 1 summarizes the coding of the master parameter sent to the slave with every packet. The first bit indicates the start of a new horizontal frame. To synchronize vertical frames, which deliver a fast position with frames containing a safe position, the msync bit is used. The single bit after parameterization data indicates special information in the data nibble. This bit defines a state machine for initializing the communication protocol before position data is transferred.

When the special character bit is not set, the lower four bits represent a nibble of a longer message. After the master information field, the direction on the wire changes from transmit to receive. There is no data expected during this switch bit, and it allows the physical layer to switch the direction from transmitter to receiver. The SN65HVD78 RS-485 transceiver has a max enable time of 30 ns for the output driver. The delay through the transceiver, wire, and slave response time is much longer even for short wires; during the switch bit, the line is not driven by either the master or the slave.

Depending on the wire length, the slave response is delayed by up to 10 bits, which is equal to 100 m of cable. The slave response is always 61 bits of information. Figure 1 gives more details on the slave answer, which is split into:

- Vertical channel information, which spans over eight frames
- S_PAR salve parameter channel with sync bit
- Pipeline channel, which is raw data format used by Sensor Hub
- Acceleration data and cyclic redundancy code (CRC)
- Direction bit (dir) used on sensor side to switch to second encoder
- Secondary channel for safety

After 73 bit times, the slave response window is over and the master starts sending two more fields called trailer and stuffing. Eight bits of trailer is used to terminate the current frame with a fixed pattern. A stuffing field can vary between 0 and 156 bits to align with motor control cycle time.

**Table 2. DSL Frame Length for Given PWM Cycle**

| MOTOR CONTROL FREQUENCY | LENGTH OF PWM CYCLE | LENGTH OF PROTOCOL FRAMES | DSL FRAMES PER CYCLE |
|---|---|---|---|
| 8 kHz | 125 µs | 12.5 µs | 10 |
| 16 kHz | 62.5 µs | 12.5 µs | 5 |

Table 2 shows the relation between the motor cycle and DSL frame length. The stuffing field is only in increments of 6 to keep line balancing. In order to match the motor cycle time with DSL frames, the number of stuffing bits can vary for the interim frames. For one 8-kHz PWM cycle, there are 10 DSL frames. Distributing stuffing bits and variable bits in the extra window make it an exact match with the cycle time. The maximum length of a DSL frame is 263 bits or 28 µs.

Find further details on the HIPERFACE DSL protocol and analog transceiver card used in this project at:

- HIPERFACE DSL Manual: https://www.sick.com/media/pdf/7/07/607/IM0056607.PDF
- HIPERFACE DSL Encoder Interface: http://www.ti.com/lit/ug/tidua76a/tidua76a.pdf

## 2 System Description

The master side implementation of HIPERFACE DSL is typically part of motor control hardware also called drive or inverter. The basic functions of the motor control hardware are the PWM output for the 3-phase motor, current sensing of three motor phases, and position sensing, which gives information about rotor position. A motor control algorithm receives speed and position targets from the application and then calculates the PWM phases in a closed loop using position and speed information from the position encoder.

The Sitara™ AM437x system on chip has all the functions for the motor control application and connections to an automation system through Industrial Ethernet. For real-time interfaces, the device uses dedicated intelligent peripheral called Industrial Communication Subsystem (ICSS). The programmable real-time unit (PRU) inside the ICSS provides the intelligence needed to implement real-time interfaces with protocols. In case of delta-sigma current measurement, the PRU provides a flexible SINC-3 decimation filter system directly connected to isolated modulators such as the AMC1305.

Another PRU is used for the HIPERFACE DSL master in the motor control application shown in Figure 2. An important aspect for the motor control application is the synchronization between the PWM output, current measurement, and position measurement. A dedicated time synchronization unit inside the ICSS triggers the various events in real time with a 5-ns resolution.

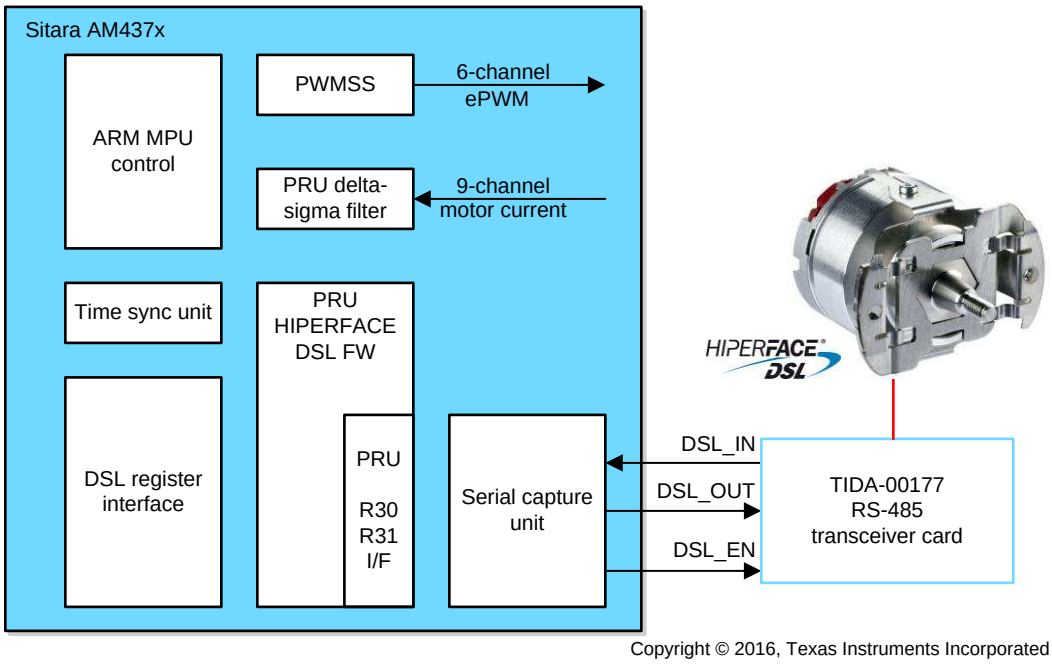





**Figure 2. Sitara AM437x Motor Controller With HIPERFACE DSL Position Feedback**

The PRUs inside the ICSS have different operating modes. For serial interfaces, the AM437x has a hardware extension to the PRU called a serial capture unit (SCU). Figure 2 shows the direct interface from the SCU to the external RS-485 transceiver. There are only three signals between the SCU and transceiver hardware. DSL_IN and DSL_OUT are the data lines for receive and transmit, which share the same wire on the cable to the encoder. As only one data line is active at a time, the DSL_EN signal is used to switch the direction on the RS-485 transceiver.

The PRU-ICSS can operate in different modes. For Industrial Ethernet, the register R30 and R31 are connected to FIFOs in Ethernet interface. In GPIO mode, external pins are directly connected to R30 and R31 register which allows for raw mode with sampling resolution of 5 ns. The SCU is used for serial interfaces with up to three channels per PRU. It supports only half duplex operation, which means there is either receive interface or transmit interface active.

## 2.1 Serial Transmit Channel

Figure 3 shows the connections for one out of three transmit channels when the serial interface mode is selected and the receiver is not active. PRU registers are 32 bits wide. The first 8 bits of R30 transfer data through optional bit-swap hardware into a transmit FIFO. The FIFO size is 4 bytes and does a parallel-to-serial conversion based on clock control and status. The HIPERFACE DSL baud rate is 9.375 Mbit/s and transfers one bit every 106.67 ns. In order to have an exact match of the bit rate, the source clock for the PRU and the TX clock generator is set to 225 MHz. Dividing 225 MHz by 24 provides the exact bit clock of 9.375 Mb. The input clock in the two fractional dividers can be selected between two clock sources. One is the ocp_clk, which is also used by the PRU core. The second clock source is the uart_clk, which typically runs 192 MHz for serial fieldbus application over UART.

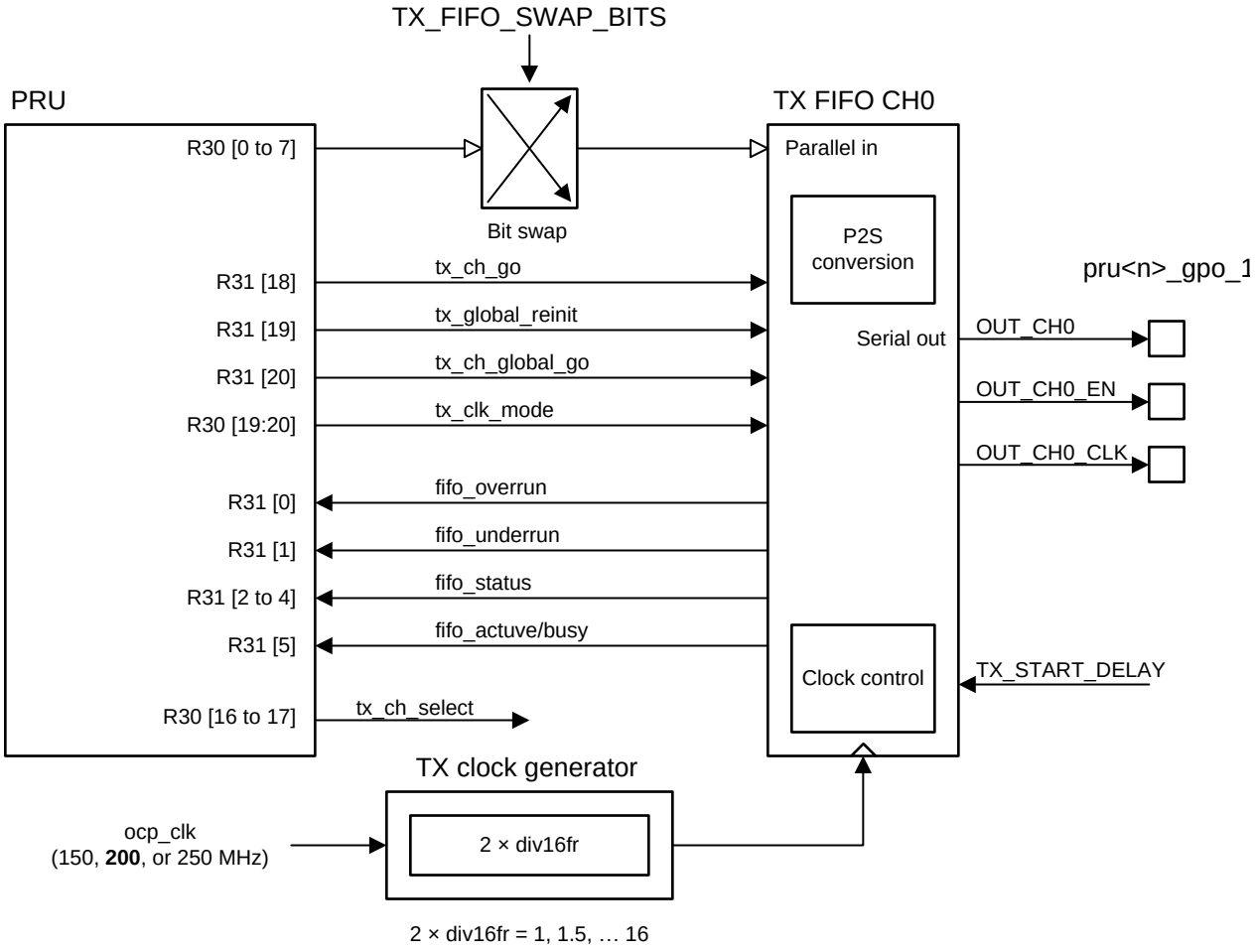



**Figure 3. Transmit FIFO for DSL_OUT Signal**

Basic TX FIFO operation is set through memory mapped configuration registers:

- Clock source for TX clock generator
- Divider setting for clock generator
- TX FIFO bit swap operation, default is MSB (bit 7) first
- TX frame size which defined the timing when OUT_CH0_EN is set
- TX wire delay counter

After setting up the configuration register, the PRU R30/R31 interface to TX FIFO defines the operation. There is only one 8-bit data path going to three channels. Register R30 bits 16 and 17 select which channel the data is sent. The FIFO has two modes of operation. Preload and go is used to transfer frames with less than 32 bits. Continuous mode sends frames bigger than 32 bits. In continuous mode, PRU firmware needs to keep up with the line rate. The FIFO contains both a write pointer and a read pointer, which are incremented whenever data is written to the FIFO and whenever data is read from the FIFO, respectively. When the pointers reach the last address in the FIFO, they will circle back to the first address.

Each interface has dedicated control and status register bits in R30/R31. The FIFO input tx_global_reinit resets all the FIFO pointers and causes the OUT_CH0_CLK for each channel to be held high and de-asserts the OUT_CH0_EN output. At each FIFO write strobe pulse, data will be pushed into the FIFO. The tx_channel_go bit signals the start of the wire delay compensation counter. Data is read from the FIFO at the Tx clock rate, after the TX wire delay and test delay compensation have been met. For HIPERFACE DSL implementation, there is no transmit delay function required as wire delay is only seen during slave answer. For single channel operation, the individual tx_channel_go control bits are used. As an option, all three channels can be started using the tx_ch_global_go bit.

The tx_fifo_status[2:0] indicates the number of bytes remaining in the FIFO (empty means 0 bytes, near empty is 1 or 2 bytes, near full is 3 bytes, full is 4 bytes). Two error flags, overrun (ovr) and underrun (unr), also exist. An overrun occurs whenever data is pushed to an address where data already exists, but has not yet been read. An underrun occurs whenever an address of the FIFO that does not contain data is read. The underrun will not occur if software specifies a TX_FRAME_SIZE. Only if TX_FRAME_SIZE is 0 can an underrun can occur. It is up to the firmware to keep the FIFO from running empty. Once the FIFO runs empty, the hardware will assume end of the last transmission. Any new writes to FIFO will not be sent until the PRU firmware initiates another tx_channel_go.

Tx_clk_mode defines various states of OUT_CH0_CLK when transmission stopped. For the HIPERFACE DSL, the clock output is not used. It can be observed for debug purpose. Important for the PRU firmware is to know when last bit of TX FIFO is sent to the wire. The fifo_active or busy bit provides this real-time status to the PRU. The firmware then changes from transmit mode to receive mode.

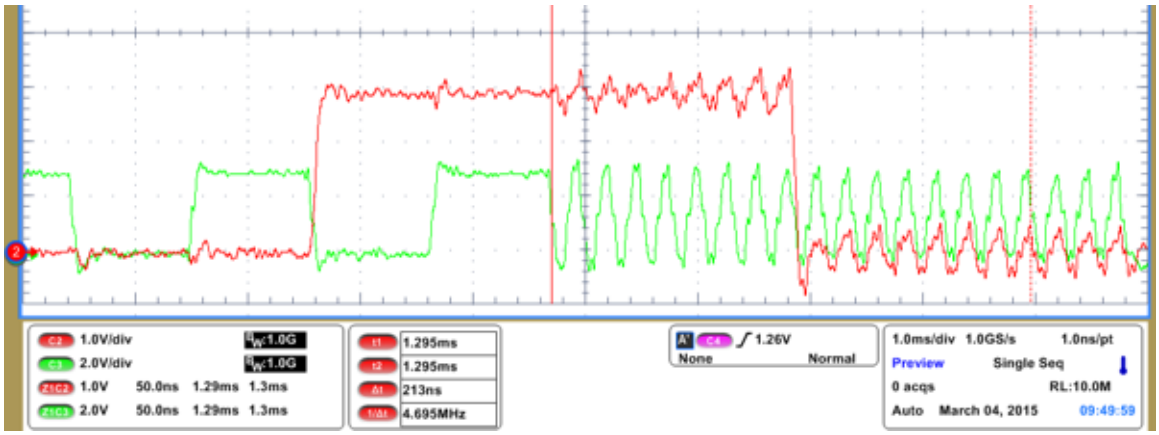



**Figure 4. Overclocking Example (Green Line: Clock, Red Line: Data Out)**

For enhanced debug and tuning capability, the FIFO operation can be dynamically switched in terms of TX clock generation. In theory, it is possible to send each bit with a different bit rate assuming PRU firmware does real-time monitoring and control of TX FIFO. By setting the TX divider to 3, the output data is generated with an 8× oversampled clock in case of the HIPERFACE DSL. This sets the resolution of the outgoing edge to 13.3 ns. This feature is used to asynchronously embed an external sync pulse into the EXTRA window of the HIPERFACE DSL frame. Figure 4 shows an example of dynamic change of TX clock.

## 2.2 Serial Receive Channel

The serial receive channel also operates over a mixture of memory mapped configuration register and PRU register R30/R31 interface. Each receive channel has its own set of R30/R31 interface including data, control, and status information. Figure 5 gives an overview of the receive channel 0 interface. Only channel 0 shares the data interface with status bits from the TX interface of channel 0. Channel 1 and 2 data bits are mapped to higher R30 register bits. With the rx_en bit going high, the receive channel is selected and multiplexer on R30[7 to 0] switches to receive data from the RX FIFO.

The receive channel samples data on CH0_IN pin with a programmable sampling clock. The RX clock generator defines the sampling clock. Similar to the transmit interface, the receive clock is set to support a 9,375-Mb data rate. As the phase and time of incoming data bits are dependent on the slave hardware, transceiver hardware and wires up to 100 m, oversampling of the serial bit stream is required. The sampling frequency for HIPERFACE DSL slave answer is set to 75 MHz (8× oversampling). The RX_SAMPLE_SIZE is configured to 8 bits, meaning that after 8 bits are received the data is available for PRU to read from the 32-bit deep FIFO. This data remains constant for eight clock cycles, and the PRU firmware must read it during this time. Otherwise, the data will overflow. If an overflow occurs, an overflow flag will be set to signal that a val bit has been continuously asserted for longer than one data frame. Both status bits for overflow and valid can be cleared directly by the PRU using clr_ovf and clr_val bits.

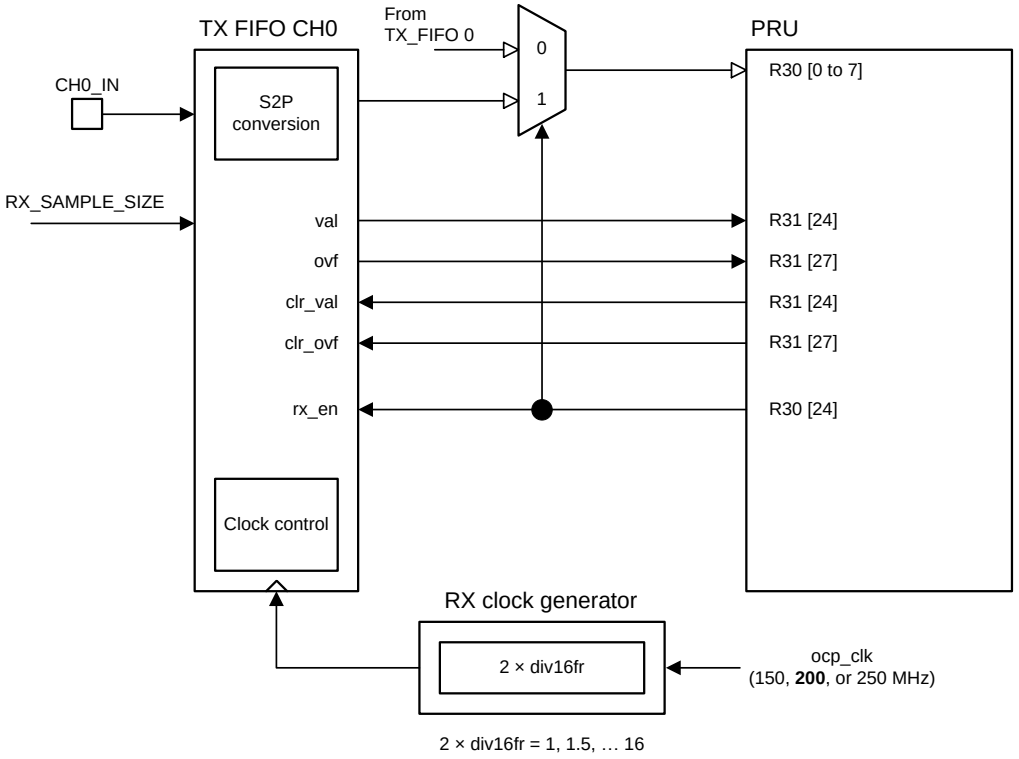



**Figure 5. Receive FIFO for DSL IN**

With rx_en switch to the TX channel, all flags and the receive FIFO are cleared. The memory mapped register settings for receive channel are:

- Receive clock source selection of ocp_clk or uart_clk
- Receive clock divider setting, for example div by 3 to support 9.375 Mbit/s
- Receive sample size between 4 and 8 bits
- Receive frame size for keeping TX clock running, not used for HIPERFACE DSL
- Automatic RX_EN assertion by hardware after rx_en_count (16 bit)

Dynamically switching the receive clock is not required as the incoming bit stream from an encoder is always oversampled with a factor of 8.

## 2.3 Synchronization With Motor Cycle

The AM437x has multiple options to synchronize HIPERFACE DSL with the motor control cycle. The primary question is about the timing master in the system and whether the motor control cycle resides on the same device or not.

Looking at the HIPERFACE DSL as a standalone peripheral, the sync pulse from drive application is generated by external source which has no timing and phase relation to the Sitara AM437x. In this case a second sampling interface of AM437x will be used to synchronize the DSL frame with the motor cycle. The AM437x has two ICSS instances with different configurations. ICSS1 is typically used for Industrial Ethernet where all signals including external time latch input of the timer modules inside ICSS is available. With a lower configuration set in terms of memory and interfaces (no Ethernet), ICSS0 supports the motor side communication with encoder interfaces and delta-sigma decimation filters. Using ICSS0 for the HIPERFACE DSL, the best interface for the sync pulse input is another serial channel using a higher sampling rate and oversampling of 1. When ICSS1 is available, the latch input of the timer module provides a 5-ns sampling resolution with a hardware time stamp.

With the PWM subsystem on the AM437x, there is no need for external sync pulse as on-chip signals can be used to capture the time reference. The timer in ICSS can be configured that is gets reset with every PWM cycle representing the drive synchronization pulse. The timer module then provides a relative time base to the DSL frame, and the PRU can synchronize the DSL frame towards this time reference using a software PLL algorithm. In case both peripherals, ICSS and PWM, run the same clock source, there is a direct phase relationship and no software tuning is required.

## 3 System Design

On the AM437x, the HIPERFACE DSL function is implemented on ICSS0 to leave ICSS1 for Industrial Ethernet functions. The instruction memory on one PRU of ICSS0 is 4 kB or 1000 instructions. To implement an equivalent data link layer and transport layer as the reference IP-core for the HIPERFACE DSL on FPGA, a code overlay scheme is required. The datalink layer is split into two code sections for initialization and normal operation.

The PRU by default does not support code overlay from external master. Only if PRU is halted can the program memory be written by an external master. The state machine of the HIPERFACE DSL is modified in a way that when the protocol changes from initialization to operational code, a few bytes of program memory are loaded using EDMA transfer to PRU memory and a final transfer to continue PRU operation. The event mapping on AM437x does not allow interrupts from ICSS0 to trigger EDMA events. Therefore, routing ICSS0 events into ICCS1 interrupt controller provides an event path to EDMA as shown in Figure 6. The PRU sends the interrupt to interrupt controller channel 7. This channel is configured to generate host interrupt 7, which then connects to host event channel 8 on the interrupt controller of ICSS1. Host interrupt 8 of ICSS1 can now be mapped to trigger an EDMA transfer. Find more details of the ICSS interrupt controller in the Technical Reference Manual of the AM437x (SPRUHL7).

Figure 7 shows the workflow of loading new firmware to PRU code. In the first step, the PRU writes metadata for code overlay to PRU data memory. This data is used by DMA to transfer the new program code into PRU instruction memory. Next, the PRU issues an interrupt over the event chain shown in Figure 6. Immediately after sending the interrupt, the PRU halts the operation through the PRU control register. The DMA transfers new code as defined by metadata. A second DMA transfer, which is chained with the first transfer, re-enables the PRU execution through a configuration register.

The firmware consists of two layers. There is the data link layer, which establishes a communication link to the slave, monitors the connection quality, and prepares the data. On top of the data link layer, the transport layer processes the data and determines what information is sent over the parameter channel. Figure 9 illustrates the relationship between the two layers. The datalink layer assembles the information from the different channels and puts the data symbol by symbol to the channel buffers. The channel buffers are large enough to carry the data of a whole V-Frame for each channel. The transport layer controls the data sent over the parameter channel by setting the symbol to send for the next H-Frame in the parameter channel buffer. This buffer can carry only one symbol. Both layers have direct access to the register interface that is provided to the higher layers.

The HIPERFACE DSL specifies a state machine for the master. The implementation on the AM437x ICSS0 features an additional state for loading new firmware to the PRU. Figure 8 depicts the modified state machine. This implementation exhibits three code sections in the firmware. The first section is for initializing the state machine up to the LOADFW state. The second section contains datalink functionalities that are needed for the startup phase and for the normal workflow. The third section contains the transport layer functionalities. After the state machine is initialized, the initialization section is overwritten by the section that contains the transport layer functionality. The transport layer code is copied to the PRU memory during the LOADFW state.

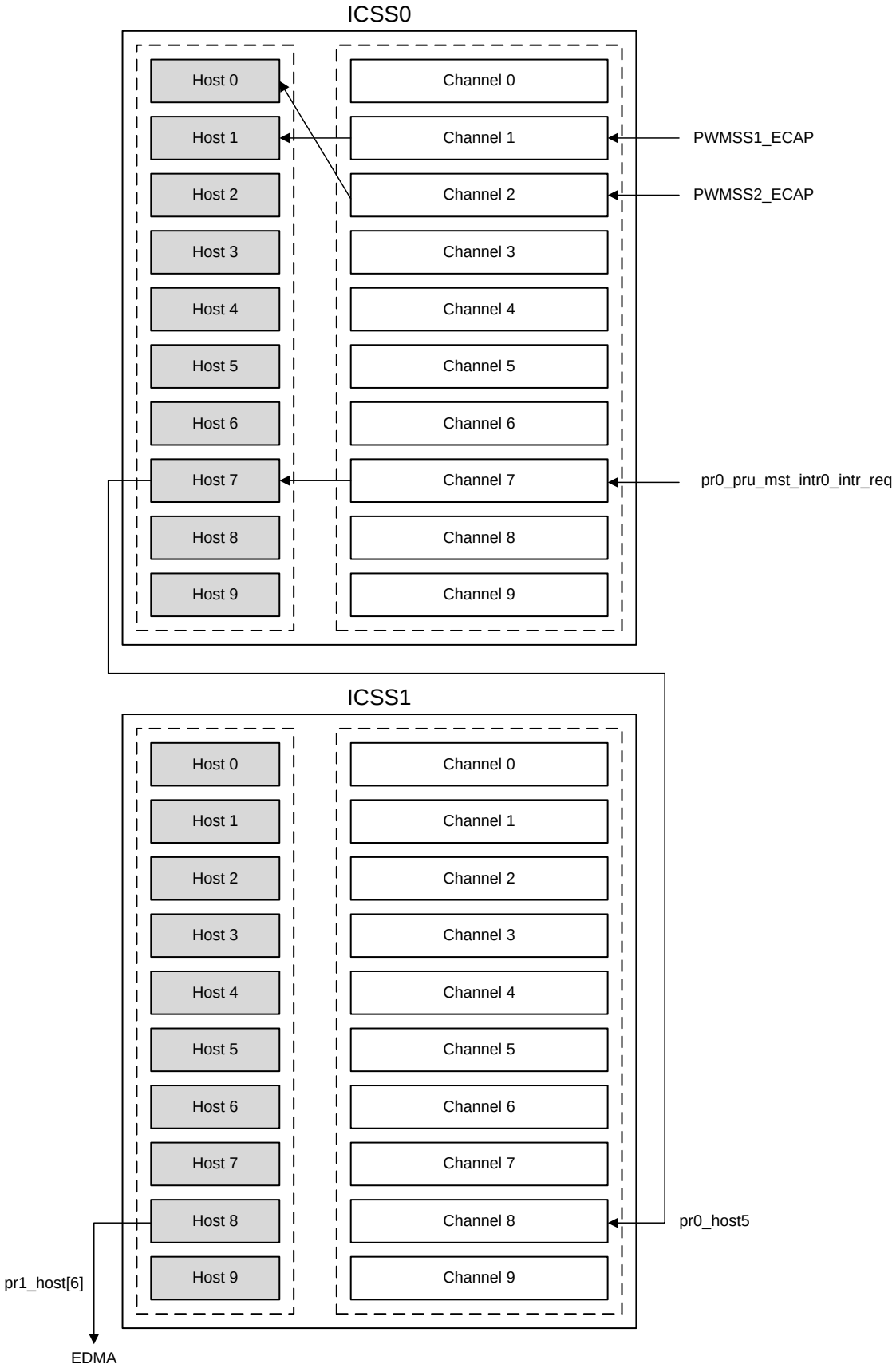



**Figure 6. Interrupt Routing for EDMA**

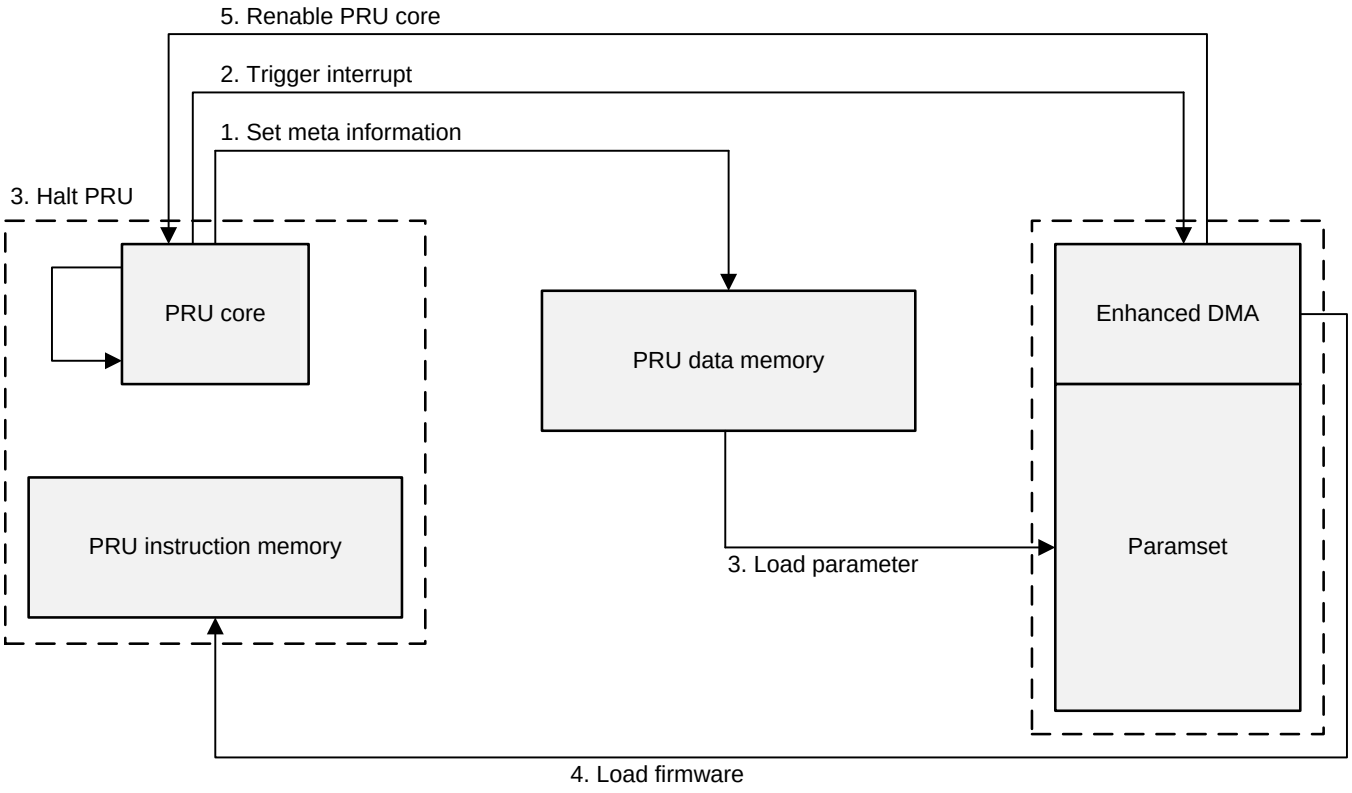



**Figure 7. Workflow of Loading New Firmware**

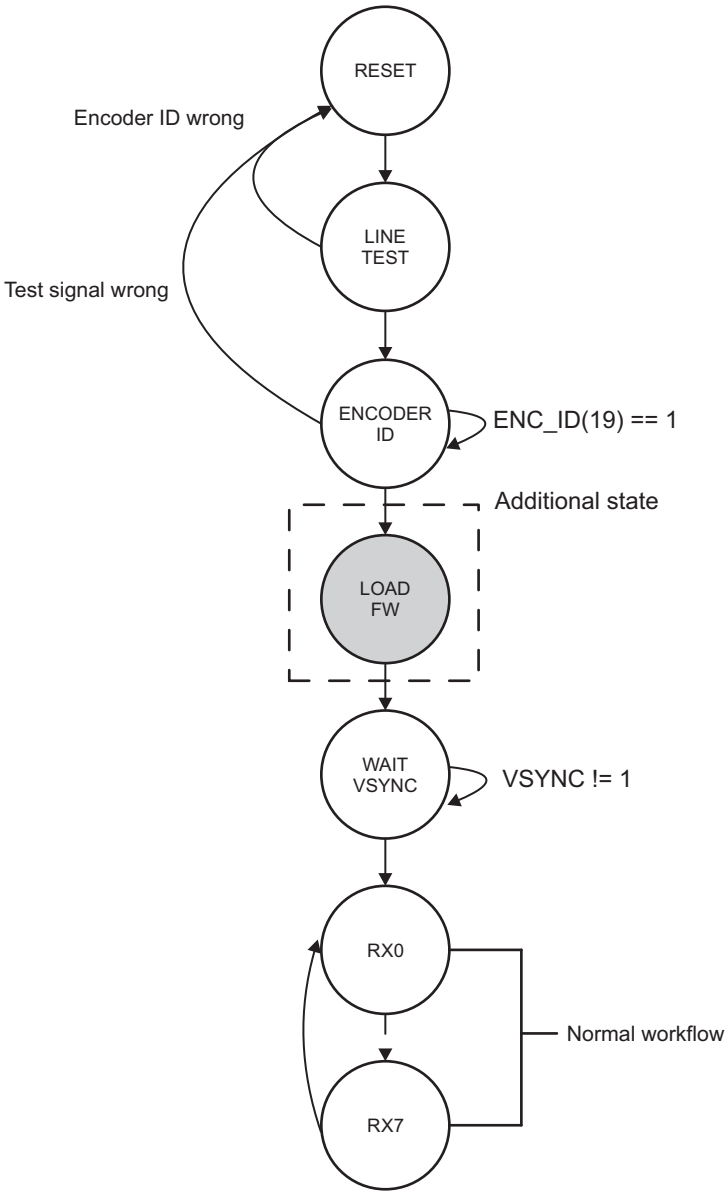



**Figure 8. Modified State Machine**

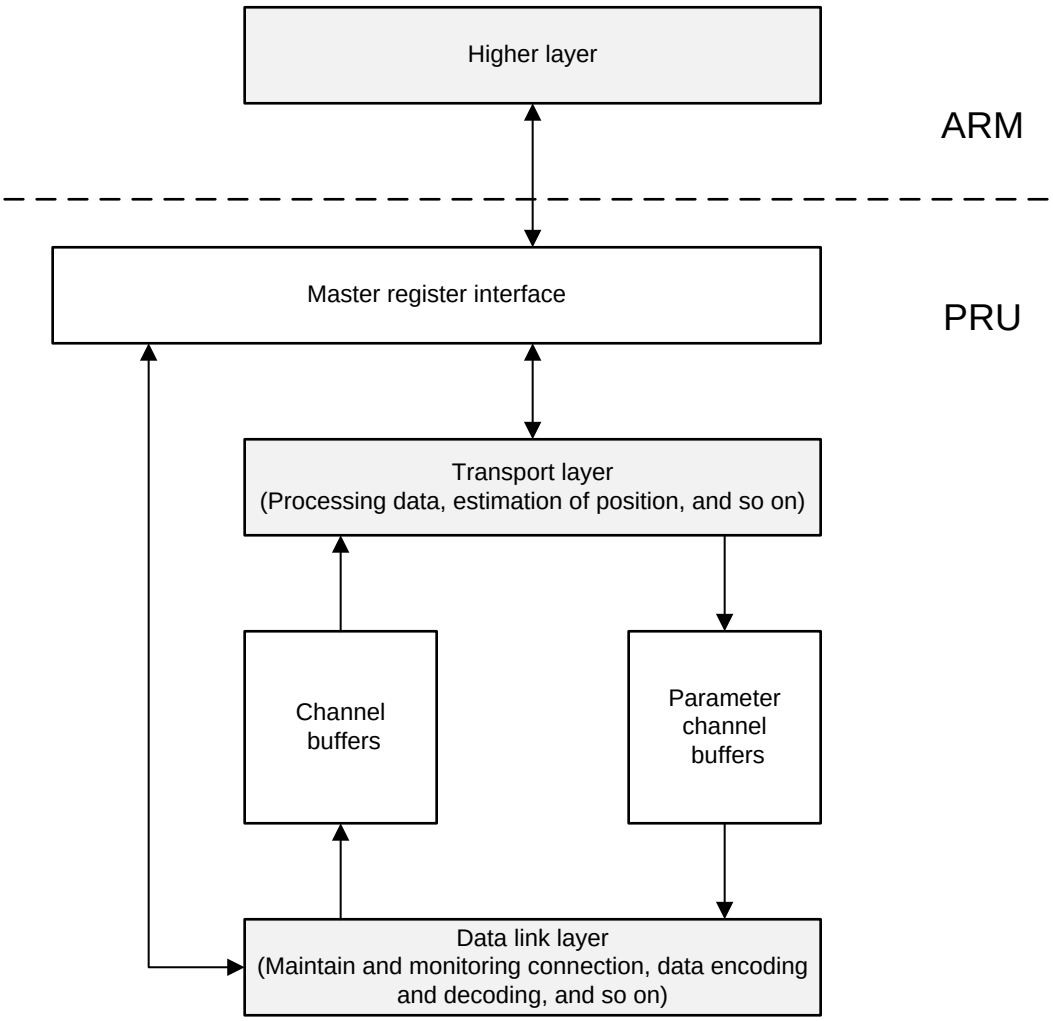



**Figure 9. Firmware Architecture**

### 3.1 Data Link Layer

The data link layer is responsible for handling the communication link to the slave. This includes

- Sampling
- Cable delay compensation
- DC line balancing
- Encoding and decoding data
- Monitoring of the connection quality

During the initialization phase of the HIPERFACE DSL, the receiver of the master detects the sampling edge of an 8× oversampled bit window. Figure 10 shows the transition from transmit to receive with a maximum line delay of 1.26 µs, which was measured connecting a 100-m cable between the encoder and master interface. The blue line represents OUT_CH0_EN and switches the direction on the RS-485 transceiver. The red line is CH0_IN showing multiple phases. The first phase is the end of M_PAR field send by the master over OUT_CH0. After the direction switch to the receive path, it takes about 40 ns until the RS-485 transceiver is settled. As there is no receive signal from the encoder at this time, the CH0_IN goes into a high state for the duration of wire delay. After 1.262 µs, the encoder test pattern arrives, which starts with logic 0.

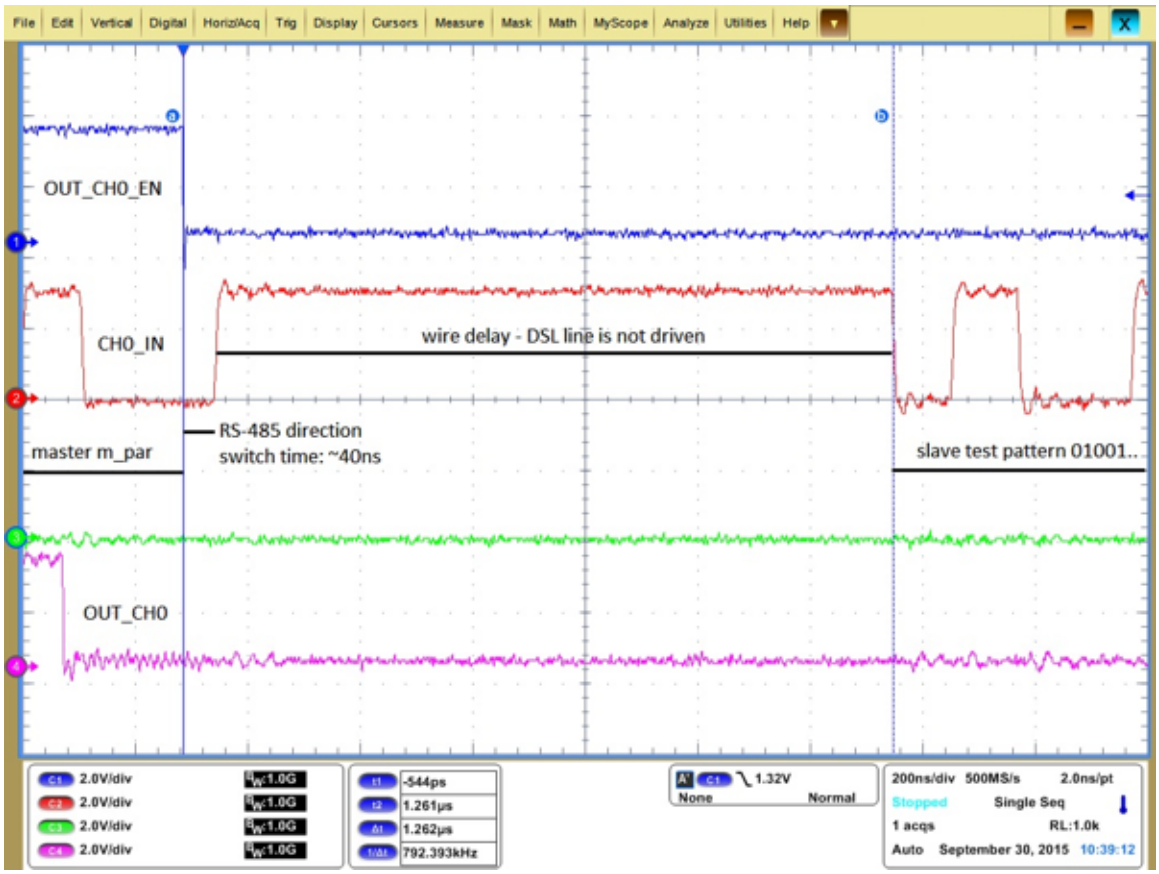



**Figure 10. Slave Delay With 100-m Cable**

The receive channel starts sampling at ~70 ns after the direction switch on the transceiver. In both cases, short and long wires, the line is not driven from the encoder and CH0_IN is driven high. A high state on CH0_IN starts the capture on the receive FIFO with 8× oversampling rate. PRU firmware continues receiving all ones during line delay phase. In case the received oversampled 8 bits contains a bit transition, the sampling edge detection function is called. This function uses a look-up table to return the sampling edge bit position based on the number of 1's detected in the oversampled bit window. With the example shown in Figure 11 the sampling edge is a +4-bit position after 0 to 1 transitions. In case the resulting edge is > 8, the result is subtracted by 8 to stay within the sampling window.

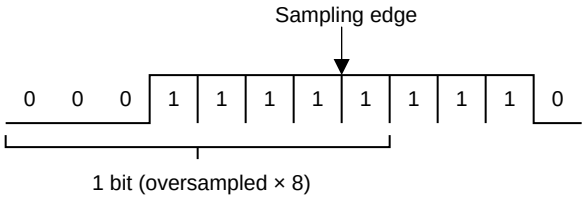



**Figure 11. Sampling Edge Detection on First Bits of Test Pattern**

During the learn state, the encoder sends a 61-bit long test pattern, which is sampled into the PRU register. One bit is masked undefined, which indicates a line switch for two encoders on the slave side. For each cycle in the learn state, the received test pattern is compared with the known test pattern to calculate the delay of the slave response. A maximum delay of 11 bits is possible to fit the 61-bit test pattern plus a switch bit into the 73-bit window of the slave response. Only if all test patterns sent during the learn state are detected, the state machine continues with an ID request state. In case the test pattern is not detected, the HIPERFACE DSL master resets the state machine and starts the initialization from the beginning.

The transition of DSL data line from transmit to receive is critical in terms of the DC level. Figure 12 shows the hysteresis of the input voltage over the receiver output voltage. The output voltage goes into a low state when input voltage difference exceeds –135 mV.

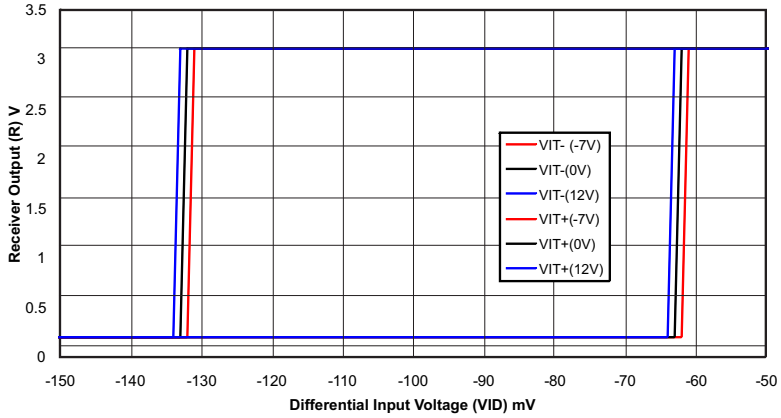



**Figure 12. Input Voltage Hysteresis of SN65HVD78 Transceiver**

While developing the HIPERFACE DSL firmware, a false detection of a signal level occurred due to DC line unbalancing. Figure 13 shows the failing case between curser lines a and b. The differential lines (green and pink lines) show high DC unbalances after the direction switch (yellow line). The input voltage swing at vertical cursor line ‘a’ goes above –135 mV. This drives the output voltage to a low level. With a differential voltage level of –60 mV, the output voltage goes back to a high voltage level. Another false detection happens at vertical curser line 'b'. At the time when the encoder drives the wire, there were already two false detections due to line unbalance from the transmit phase. The failing condition could be solved by proper settings of four equalization bits. The firmware counts the number of unbalanced bits sent during the EXTRA window and compensates uneven distribution of zeroes and ones through equalization bits.

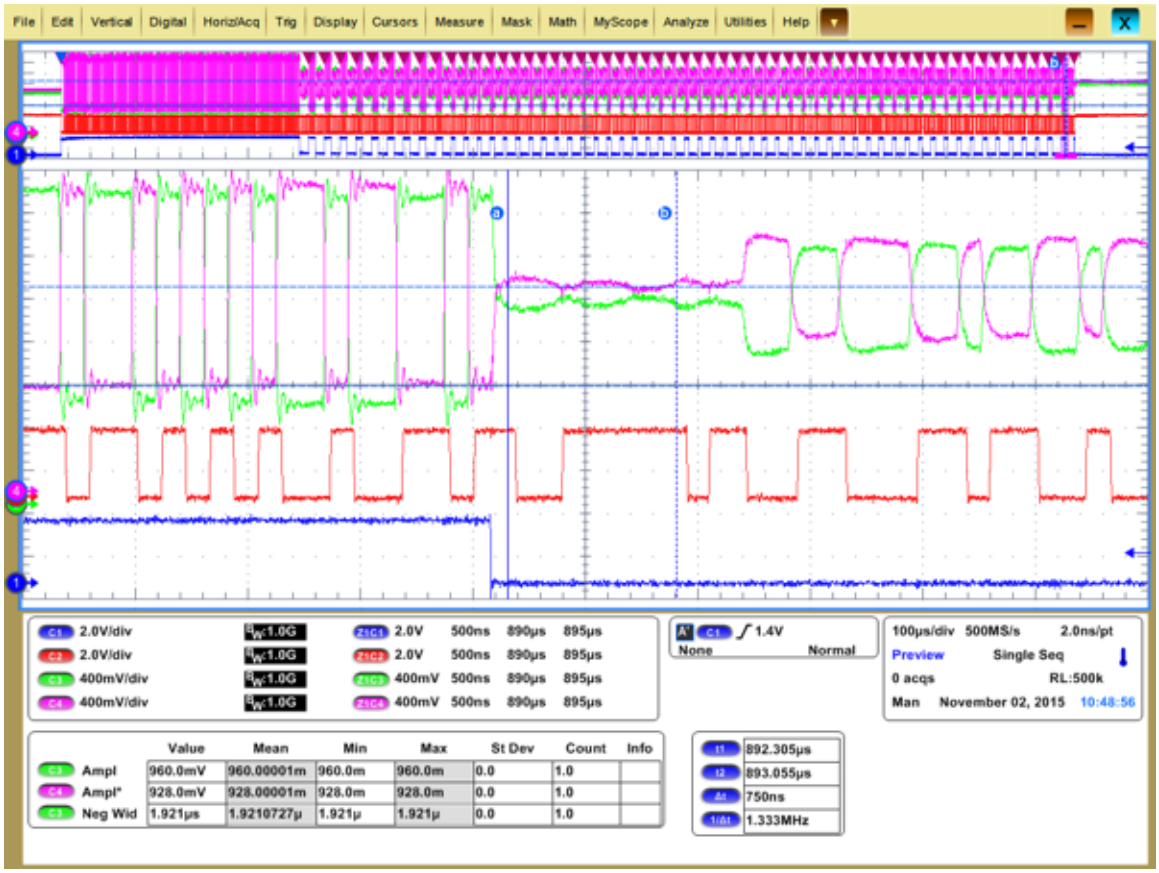



**Figure 13. Failing Condition With Unbalanced DC Line**

The data link layer has the responsibility to decode and encode the data according to an 8b/10b scheme[3]. The 8b/10b encoding/decoding is split into two parts, 3b/4b and 5b/6b encoding and decoding. Each encoding and decoding process is performed by using a look-up table (LUT). The HIPERFACE DSL assumes a transmission with LSB first. Therefore, in the encoding procedure, the index of the LUT is in MSB first order, while the LUT entries are in LSB first order (and vice versa when decoding data). This way, the firmware does not need to handle the reversing of the bit order. When encoding the data, the firmware handles the sending of the correct polarity of the sub-blocks using the measured line disparity. During the receive state, the firmware detects byte errors and special characters by checking the received encoded data according to the article *A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code*[3].

There are several quality measures for the communication link between the master and the slave. The data link layer handles all of these measures, with exception to the CRC verification. A list of the measures is given in the following:

- Received Signal Strength Indication (RSSI)
- Edges
- Vertical synchronization
- Byte error detection

The RSSI is calculated by determining the number of samples between two edges during a bit period. The samples that form the longest sequence between two edges represent the stable bit period, which is used to calculate the RSSI. Instead of calculating the stable period in the firmware, a pre-calculated LUT is used to speed up the process. First, the edges in a bit period are determined, which is performed by an XOR operation (Figure 14). The searched RSSI value is looked up in the table by using the result of the XOR operation as the index.

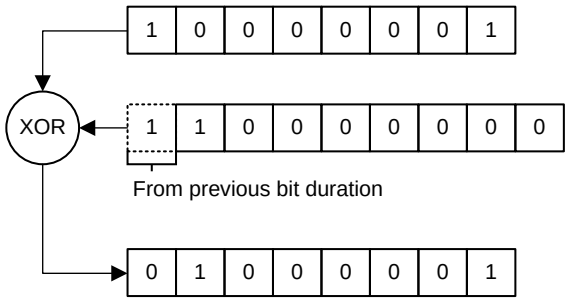



**Figure 14. Calculation of Edges in a Bit Period**

Edges are a measure of the line signal integrity. The 8 bits of edges represent bit position for transition of slave response bits. A clean transmission line will result in only one transition point in a bit window. The sampling point has enough margins for safe bit detection in this case. In case all bits are set in edges, a safe bit sampling is no longer possible and a new initialization is triggered. Edges, RSSI, and DELAY information are available to the user through register interface. Edges are evaluated only during the initialization phase whereas RSSI is continuously updated.

The 8b/10b line code allows for special characters and error detection. For example, the character K29.7 is used to invalidate the position or acceleration data in case of encoder internal error. Another error indicator of the line code is disparity mismatch. Only a packet with either 0, +2, or –2 disparity are allowed. Non-zero blocks must alternate in polarity. Certain combinations of output signal states of the encoder are invalid. For example, a = b = c = d or e = i = f = g= h. PRU firmware checks on this error conditions.

A 16-bit CRC verification of the data is used on multiple occasions. It is used for the vertical channel, secondary channel, and messages. To distribute the computation load equally over all H-Frames, the firmware calculates a running CRC for those data. The algorithm uses a LUT with 256 entries and 2 bytes per entry, whereas each entry is the 16-bit CRC for the corresponding LUT index. The basic approach for the calculation of the 16-bit CRC is shown as C code in the following:

```
uint16_t calc_crc(uint8_t *data, uint32_t size) {
        uint16_t crc = 0;
        uint32_t i;
        for(i = 0; i < size; ++i) {
                crc = ((*data) << 8) ^ crc;
                crc = lut[crc>>8] ^ (crc << 8);
        }
        return (crc ^ 0xff);
}
```

**Figure 15. CRC Algorithm**

The for-loop is split in such a way that only one iteration is performed per H-Frame. Only the current CRC value needs to be saved between two iterations.

## 3.2 Transport Layer

The transport layer processes the channel information, which was prepared by the data link layer. This includes the calculation of the fast position as well as the handling of messages. Fast position is transferred over the vertical channel using the VSYNC bit in slave parameter channel.

During normal workflow, at times the received ΔΔPosition data cannot be used for calculations because of either a transmission error or an internal slave error. To check for a transmission error, the transport layer checks if the data link layer detected a byte error and verifies the CRC in the acceleration channel. If no transmission error occurred, the transport layer searches for two K29.7 characters to recognize an internal slave error. In case one of the data verifications fails, the estimation algorithm shown in Figure 16 is executed.

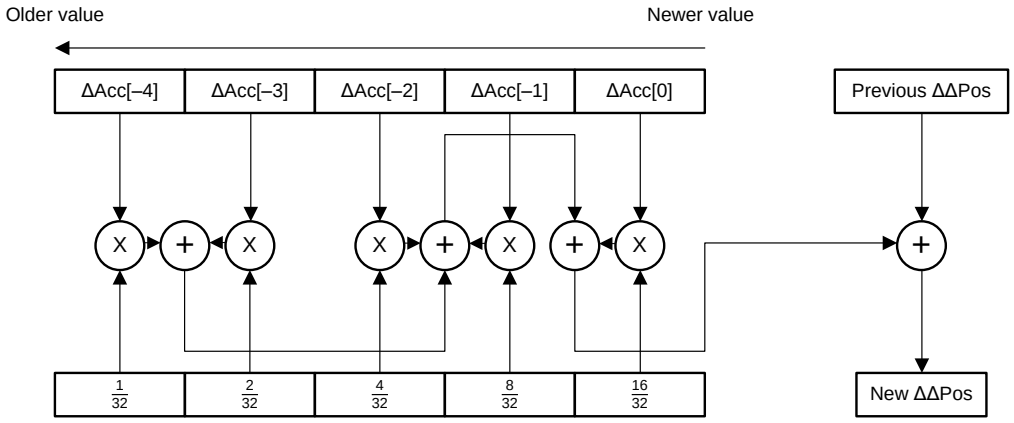



**Figure 16. Estimation Algorithm for ΔΔPosition**

The transport layer saves a history consisting of the five most recent ΔAcceleration. Based on this data, a new ΔΔPosition is calculated by applying a weighted mask to the recent ΔAccelerations. The calculated ΔΔPosition is added to the previous ΔΔPosition. The result is the estimated ΔΔPosition used for the fast position calculation. In case of continuous errors, a threshold of for example eight causes the data link abort and resets the state machine to its initialization state.

Acceleration data consists of 11 bits of signed value and 5 bits of CRC. There is a maximum value for the acceleration data, which depends on the frame duration of horizontal frame. Data from the acceleration channel is processed and validate in transport layer. The horizontal channel sends absolute position data to keep the alignment of position data.

The HIPERFACE DSL provides a quality monitor (QM), which indicates transmission quality of the data link layer and transport layer. The maximum value of 15 is reported when during normal operation without any errors detected. The following error conditions reduce the value of the quality monitor.

**Table 3. QM Adjustment by Error Type**

| ERROR TYPE | QM ADJUSTMENT |
|---|---|
| Wrong VSYNC = 1 | –4 |
| Wrong VSYNC = 0 | –8 |
| Low RSSI | –8 |
| 8B10B coding error | –1 |
| Unexpected special character | –2 |
| Vertical channel error | –8 |
| Secondary channel error | –8 |

If the QM is below the max value of 15 and the safe channel is transferred with no CRC error, the QM is incremented by 1. Correct synchronization in the safe channel also adds 1 to the QM value. A QM value of 0 resets the HIPERFACE DSL connection.

## 3.3 Host Interface

Interactions between the PRU firmware for the HIPERFACE DSL and the host CPU on the AM437x are mainly through a register interface. The firmware is developed in a way that it provides register compatible interface to existing FPG IP core. The original register description is listed in the HIPERFACE DSL Manual referenced at the end of Chapter 1. Table 3 shows the register interface map of the original IP core. The implementation of the register interface described in this design supports all registers and bit field except the ones marked in red color. All relevant register information for position measurement, estimator, QM, range checker, and events are implemented. The pipe channel for sensor hub support is not supported in this design.

**Table 4. Master Register Interface**

| ADDR | DESIGNATION | BIT 7 | BIT 6 | BIT 5 | BIT 4 | BIT 3 | BIT 2 | BIT 1 | BIT 0 | VALUE AT RESET |
|---|---|---|---|---|---|---|---|---|---|---|
| 00h | SYS_CTRL | PRST | MRST | FRST | LOOP | PRDY | SPPE | SPOL | OEN | 0000 0000 |
| 01h | SYNC_CTRL | ES | | | | | | | | 0000 0001 |
| 03h | MASTER_QM | LINK | — | — | — | Quality monitoring | | | | 0—– 0000 |
| 04h | EVENT_H | INT | SUM | SCE | — | POS | VPOS | DTE | PRST | 000– 0000 |
| 05h | EVENT_L | — | — | MIN | ANS | — | QMLW | FREL | FRES | —00 –000 |
| 06h | MASK_H | — | MSUM | MVRT | — | MPOS | MVPOS | MDTE | MPRST | —00 0000 |
| 07h | MASK_L | — | — | MMIN | MANS | — | MQMLW | MFREL | MFRES | —00 –000 |
| 08h | MASK_SUM | MSUM7:0 | | | | | | | | 0000 0000 |
| 09h | EDGES | Bit sampling pattern | | | | | | | | 0000 0000 |
| 0Ah | DELAY | RSSI | | | | Cable delay | | | | 0000 0000 |
| 0Bh | VERSION | Coding | | IP Core version number | | | | | | 0101 0110 |
| 0Dh | ENC_ID2 | — | SCI | | | | ENC_ID19:16 | | | –000 0000 |
| 0Eh | ENC_ID1 | ENC_ID15:8 | | | | | | | | 0000 0000 |
| 0Fh | ENC_ID0 | ENC_ID7:0 | | | | | | | | 0000 0000 |
| 10h | POS4 | Fast position, byte 4 | | | | | | | | 0000 0000 |
| 11h | POS3 | Fast position, byte 3 | | | | | | | | 0000 0000 |
| 12h | POS2 | Fast position, byte 2 | | | | | | | | 0000 0000 |
| 13h | POS1 | Fast position, byte 1 | | | | | | | | 0000 0000 |
| 14h | POS0 | Fast position, byte 0 | | | | | | | | 0000 0000 |
| 15h | VEL2 | Speed, byte 2 | | | | | | | | 0000 0000 |
| 16h | VEL1 | Speed, byte 1 | | | | | | | | 0000 0000 |
| 17h | VEL0 | Speed, byte 0 | | | | | | | | 0000 0000 |
| 18h | SUMMARY | SUM7:0 | | | | | | | | 0000 0000 |
| 19h | VPOS4 | Safe position, byte 4 | | | | | | | | 0000 0000 |
| 1Ah | VPOS3 | Safe position, byte 3 | | | | | | | | 0000 0000 |
| 1Bh | VPOS2 | Safe position, byte 2 | | | | | | | | 0000 0000 |

**Table 4. Master Register Interface (continued)**

| ADDR | DESIGNATION | BIT 7 | BIT 6 | BIT 5 | BIT 4 | BIT 3 | BIT 2 | BIT 1 | BIT 0 | VALUE AT RESET |
|---|---|---|---|---|---|---|---|---|---|---|
| 1Ch | VPOS1 | Safe position, byte 1 | | | | | | | | 0000 0000 |
| 1Dh | VPOS0 | Safe position, byte 0 | | | | | | | | 0000 0000 |
| 1Eh | VPOSCRC_H | CRC of the safe position, byte 1 | | | | | | | | 0000 0000 |
| 1Fh | VPOSCRC_L | CRC of the safe position, byte 0 | | | | | | | | 0000 0000 |
| 20h | PC_BUFFER0 | Parameters channel, byte 0 | | | | | | | | 0000 0000 |
| 21h | PC_BUFFER1 | Parameters channel, byte 1 | | | | | | | | 0000 0000 |
| 22h | PC_BUFFER2 | Parameters channel, byte 2 | | | | | | | | 0000 0000 |
| 23h | PC_BUFFER3 | Parameters channel, byte 3 | | | | | | | | 0000 0000 |
| 24h | PC_BUFFER4 | Parameters channel, byte 4 | | | | | | | | 0000 0000 |
| 25h | PC_BUFFER5 | Parameters channel, byte 5 | | | | | | | | 0000 0000 |
| 26h | PC_BUFFER6 | Parameters channel, byte 6 | | | | | | | | 0000 0000 |
| 27h | PC_BUFFER7 | Parameters channel, byte 7 | | | | | | | | 0000 0000 |
| 28h | PC_ADD_H | LID | LDIR | LOFF | LIND | LIEN | | | LADD9:8 | 1000 0000 |
| 29h | PC_ADD_L | LADD7:0 | | | | | | | | 0000 0000 |
| 2Ah | PC_OFF_H | LID | LOFFADD14:8 | | | | | | | 1000 0000 |
| 2Bh | PC_OFF_L | LOFFADD7:0 | | | | | | | | 0000 0000 |
| 2Ch | PC_CTRL | — | — | — | — | — | — | — | LSTA | ——— ——0 |
| 2Dh | PIPE_S | — | — | — | — | POVR | PEMP | PERR | PSCI | ——— 0000 |
| 2Eh | PIPE_D | Sensor Hub FIFO, output | | | | | | | | 0000 0000 |
| 2Fh | PC_DATA | "Short message" data | | | | | | | | 0000 0000 |
| 38h | ACC_ERR_CNT | — | — | — | Acc. error threshold or counter | | | | | ——0 0000 |
| 39h | MAXACC | Acc. resolution | | Acc. Mantissa | | | | | | 0000 0000 |
| 3Ah | MAXDEV_H | Max. position, byte 1 | | | | | | | | 1111 1111 |
| 3Bh | MAXDEV_L | Max. position, byte 0 | | | | | | | | 1111 1111 |
| 3Fh | DUMMY | No data | | | | | | | | ——— ——— |

There are registers in the master interface with different meanings for read and write operation. Such a combination is not support by PRU-ICSS based implementation. Therefore, this implementation features additional registers to support full functionality of the master register interface. The difference of the modified interface and the original interface is given in Table 5. Register 0x40 to 43h are added to support the write operation with different meanings from the read operation.

**Table 5. Deviation From Original Specification**

| ADDRESS | NAME | ACCESS MODE | VALUE ON RESET | DESCRIPTION |
|---|---|---|---|---|
| 0x38 | ACC_ERR_CNT | R | 0 | Counts number of subsequent acceleration errors |
| 0x3a | MAXDEV_H | R | 0 | Maximum deviation between fast and safe position |
| 0x3b | MAXDEV_L | R | 0 | Maximum deviation between fast and safe position |
| 0x40 | SLAVE_REG_CTRL | W | 0xff | Used to initiate short messages. |
| 0x41 | ACC_ERR_CNT_TRESH | W | 0x1f | Sets threshold for number of subsequent acceleration errors |
| 0x42 | MAXDEV_THRESH_H | W | 0xff | Sets threshold for maximum deviation between fast and safe position |
| 0x43 | MAXDEV_THRESH_L | W | 0xff | Sets threshold for maximum deviation between fast and safe position |

# 4 Software

The software described in this TI Design is planned to be available in a future release of industrial software for Sitara processors.

## *4.1 Known Limitations*

The PRU firmware for the HIPERFACE DSL was tested over various cable lengths without any errors over 24 hours using the SICK Stegmann Encoder. Missing features and tests are:

- External SYNC pulse support
- Pipeline channel
- Register bits marked red in Table 4
- System test with real power stage and high phase current in motor cable

# 5 Design Files

## 5.1 Schematics

To download the schematics, see the design files at TIDRC79.

## 5.2 Bill of Materials

To download the bill of materials (BOM), see the design files at TIDRC80.

## 5.3 Layer Plots

To download the layer plots, see the design files at TIDRC78.

## 5.4 CAD Project

To download the CAD project files, see the design files at TIDRC77.

## 5.5 Gerber Files

To download the Gerber files, see the design files at TIDC823.

## 5.6 Assembly Drawings

To download the Gerber files, see the design files at TIDRC76.

# 6 References

1. SICK Stegmann, *HIPERFACE DSL® Manual*, (PDF)
2. Texas Instruments, *Two-Wire Interface to a HIPERFACE DSL® Encoder*, TIDA-00177 Design Guide (TIDUA76)
3. Franaszek, P. A. and A. X. Widmer, *A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code*. IBM Journal of Research and Development 27, 5, 440-451; 1983 (PDF)

# 7 About the Author

**THOMAS LEYRER** is a systems architect at Texas Instruments responsible for Industrial Communication solutions.

## Revision B History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

| Changes from A Revision (April 2016) to B Revision | Page |
|---|---|
| • Deleted Section 4: Software Files and replaced with Section 4: Software | 22 |

## Revision A History

| Changes from Original (December 2015) to A Revision | Page |
|---|---|
| • Changed from preview page | 1 |

## 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