SOFTWARE DESIGN DOCUMENT FOR AFC004

2 SDD Approvals:

(Approvals are handled through Archangel's Document Management System) Document Approval Record: [Document Approval Record]

2 SDD Change History Log:

Revision	Description	Date	Author
[Revision History - Letter]	[Revision History - Issue]	[Revision History - Date]	[Revision History - Author]

Table of Contents

LIST OF FIGURES

List of Tables

1 Overview

The AFC004 is a serial data concentrator and scheduler built for the Eclipse 550. Acting as an AHRS and MSU replacement for Eclipse's obsolete XBOW AHRS system, the AFC004 is a unique solution that concentrates ARINC429 AHRS data (ARINC 705) from an AHR75 and air data (ARINC 706) through a serial RS422 port into a single blended and scheduled ARINC429 data stream. The AFC004 features a single software configuration item, the IOP. Operational software (software used by the customer) developed in the AFC004's IOP shall follow the DO178C Level A lifecycle standard. A Level D code partition is implemented for factory use only.

2 Code Segments

Peripheral code- handles all microcontroller peripherals.
Common segments – handles floating point calculations, run time startup routines, trig libraries, and numerous data structure implementations.
ARINC429 code: generic ARINC429 message processing library, ARINC429 hardware driver, accompanying data structure definitions, and non-generic modules for project-specific tasks.
RS422 code: generic UART controllers, 16-bit CRC function, and non-generic modules for processing and transmitting serial data.
Maintenance code: Level D partitioned segment of code that handles factory level operations and programming
Configuration block: Variables or data stored in program memory used as configurable IOP parameters.

3 Code Partitioning

The IOP's SCI is divided into two code partitions: Level A code and Level D code. Level A code features operating code such that failure could result in catastrophic loss of life. All standard communications, message processing, and scheduling will conform to DO178C Level A requirements. Level D code is designated for factory use only. These code segments will appear on the final product; however, the operational code shall have no effect on the product for the customer. When the unit boots, the discrete strapping inputs are read. Only a valid strapping configuration can put the IOP into maintenance mode. Once in maintenance mode, the IOP requires a constant valid stream of data from the maintenance computer to remain in maintenance mode. This way, even if the unit was accidentally placed into maintenance mode at boot, unless a valid sequence of data is received from an external computer, the IOP shall exit maintenance mode. Once the unit enters the main operating code, entering maintenance mode is not possible without a reboot.

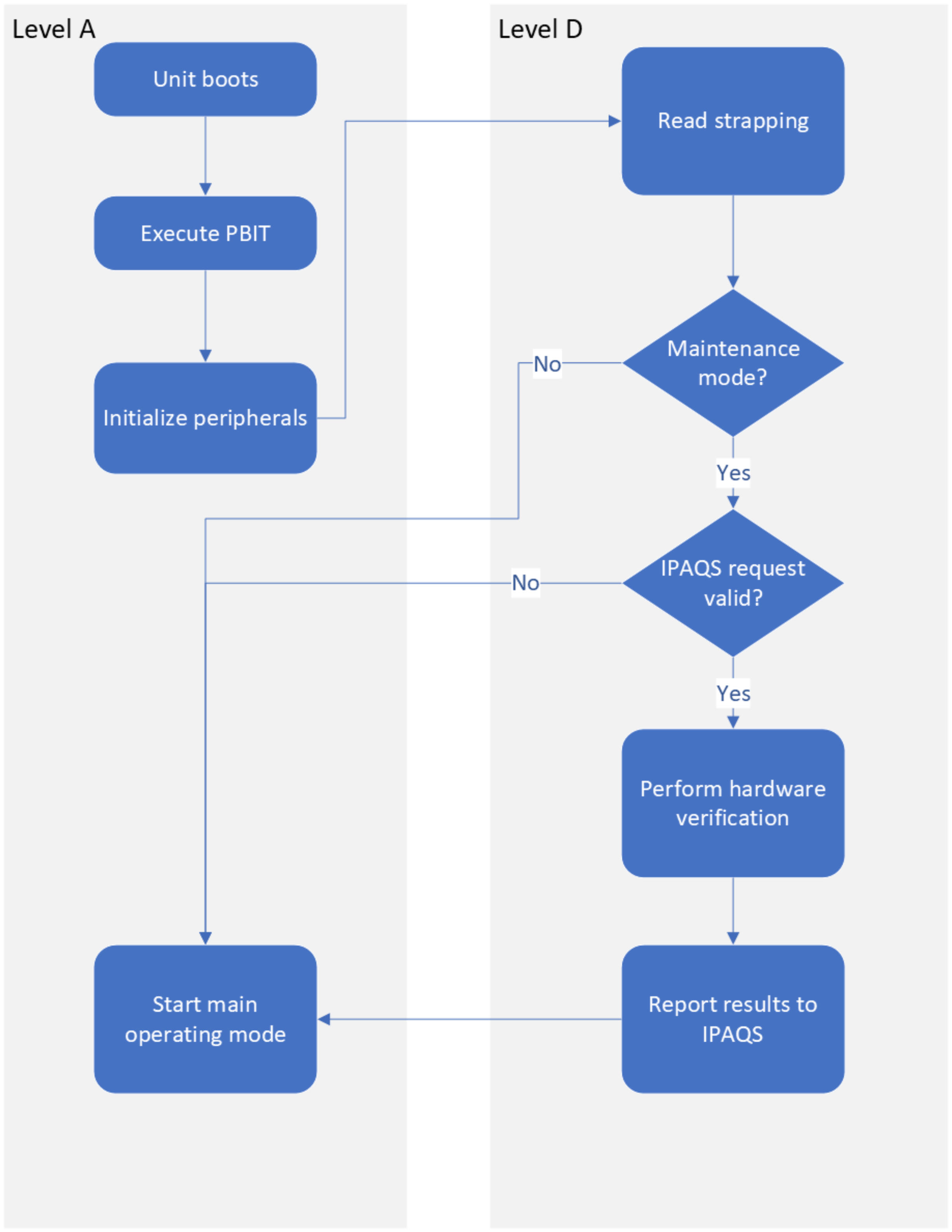


FIGURE 1 AFC004 CODE PARTITIONING DIAGRAM

4 Safety Considerations

4.1 Power-On Built-In-Tests

Outside of the standard DO178-C Level A certification basis, the IOP's software design is centered around addressing additional mission-critical safety concerns. The following power-on built-in-test (PBIT) and continuous built-in-test (CBIT) are implemented.

At program startup, and before entering the main operating loop, the IOP runs a series of internal operating checks. A composite value, known as the boot-fault status, is formed as the result of the tests described below. *All* tests must pass for the program to enter the main operating loop. **Figure 2 Power-on built in test flowchart** describes the process flow of the power on built in tests.

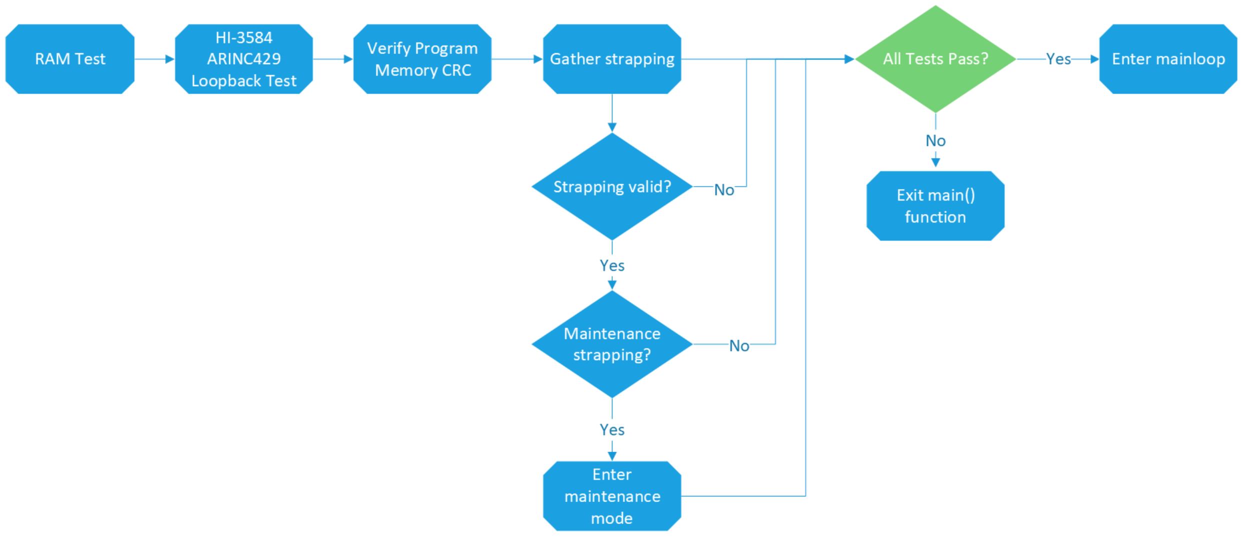


FIGURE 2 POWER-ON BUILT IN TEST FLOWCHART

4.1.1 Program memory CRC validation

During compilation, the executable hex file is appended with the 32bit CRC residue of the operation code. At startup, when the CRC is calculated over the entire program memory region, the result will equal zero so long as the stored program has not been damaged or modified. If the resultant CRC of the program memory does not equal zero, the CRC tests fails.

4.1.2 ARINC Loop-back self test

The IOP connects to two independent HI-3584 ARINC429 transceivers. The loopback test configures the transceivers to internally connect the transmitter to the receivers. During the test, each of two receivers must successfully read a known transmitted value 50 times in a row. If 50 successive reads are not completed, the loopback test fails. If both transceivers do not pass the respective loopback tests, the IOP shall not enter operating mode.

4.1.3 RAM test

The IOP initiates a test on the microcontroller’s RAM. The test performs a series of write and readback tests on various addresses of RAM. Successive matches must be read for the test to pass.

4.2 Continuous Built-in-test

At runtime, the IOP monitors the status of the program’s execution by controlling a discrete-fault-out pin to a monostable vibrator. This one-shot circuit will light an LED should an operating fault occur.

5 Software Architecture

The AFC004's IOP utilizes a dsPIC30F6014A microcontroller for performing all control logic. Since this microcontroller can set a running timer with millisecond granularity, interface with the HI-3584 ARINC429 transceiver, and receive interrupt-driven RS422 data via the UART peripherals, the dsPIC30F6014A is a suitable microcontroller for the AFC004's system architecture.

5.1 Main Operating Loop Logic

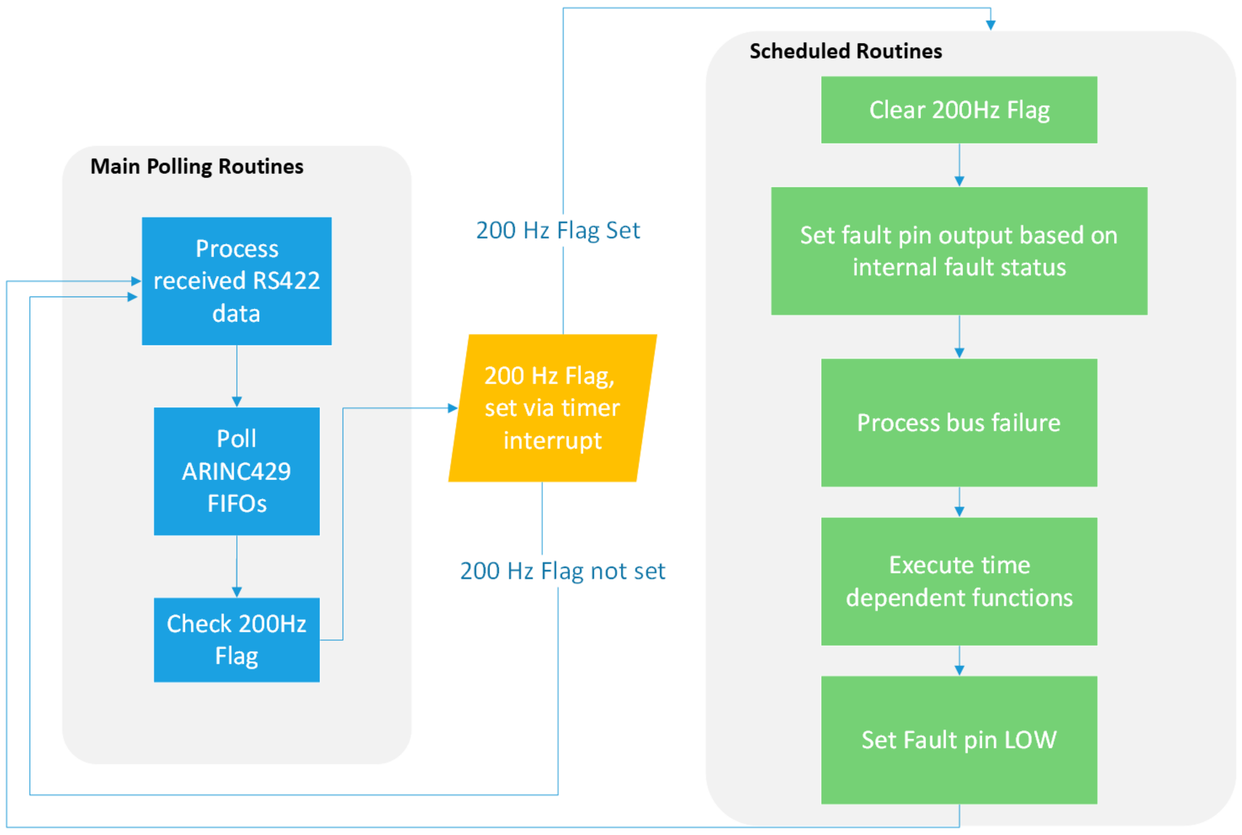


FIGURE 3 IOP CONTROL FLOW LOGIC

The main event loop of the dsPIC30F6014A IOP opcode is depicted in **Error! Reference source not found.**. Each time the main event loop is executed, the 200 Hz flag is polled. The 200 Hz flag is set inside an interrupt routine for a timer peripheral configured to produce an interrupt every 5 milliseconds (i.e., 200 Hz). If the 200 Hz flag is set, the IOP executes the scheduled routines block on this pass through the main event loop.

5.2 Scheduling Method

The AFC004 transmits all processed data at predefined, scheduled intervals. (For more information regarding the transmitted messages and respective time intervals, refer to *ICD-AFC004*.) The master scheduler operates with a base time resolution of 5 milliseconds. Upon each entry into the 200 Hz routine block, the dsPIC30F6014A will first clear the 200 Hz flag and then increment a counter variable, rateCounter, by 1. This variable acts as a down sampling counter for controlling the execution time of functions at various frequencies (10, 16.67, 20 and 50 Hz). Modulo operations are performed on the rateCounter variable to schedule desired functions at specific frequencies. For example, for an execution rate of 16.67 Hz (60 ms period), a modulo operation is performed on the rateCounter variable using 12 as the divisor. The modulus is then compared to a specific modulus comparison value for each operating block frequency. The modulus comparison value controls the staggering of the operating block frequencies to avoid overlapping execution. **Error! Reference source not found.** depicts the down sampling modulo operation parameters implemented in the IOP.

TABLE 1 EXAMPLE MODULUS DIVISION FOR CREATING A 16 HZ (60MS) FREQUENCY

Elapsed Time (ms)	rateCounter variable	rateCounter % 4 Result	Execute?
0	0	0	Yes
5	1	1	No
10	2	2	No
15	3	3	No
20	4	4	No
25	5	5	No
30	6	6	No
35	7	7	No
40	8	8	No
45	9	9	No
50	10	10	No
55	11	11	No
60	12	0	Yes
65	13	1	No
70	14	2	No
75	15	3	No
80	16	4	No
85	17	5	No
90	18	6	No
95	19	7	No
100	20	8	No
105	21	9	No
110	22	10	No

115	23	11	No
120	24	0	Yes
125	25	1	No

5.2.1 Staggered Frequency Design

To maintain maximal periodicity of each block frequency, the IOP must execute a given block within 5 ms. Staggering is implemented to ensure that at most only one block frequency is executed per 200 Hz cycle.

Consider the case where both the 50 Hz and 16.67 Hz block frequencies were called using the same modulus value. Every third execution of the 50 Hz commands would coincide with the execution of the 16.67 Hz commands. Invariably this would cause either the 50 Hz or 16.67 Hz commands, depending on execution order, to be executed later and thereby disrupt the periodicity of command execution. To counter this, a modulus comparison value of 0 is used for all 50 Hz commands and a modulus comparison value of 2 is used for all 16.67 Hz commands. This modulus change generates a phase shift of the execution intervals, ensuring that the 50Hz commands are never called concurrently with the 16.67 Hz commands during any cycle and thus greatly improving periodicity.

The 50 Hz commands involve numerous processor-intensive floating point and trigonometric calculations (the dsPIC30F6014A lacks an FPU), so it is particularly important to stagger the 50 Hz function calls so as not to coincide with other execution block frequencies. Using similar methodology, the 20 Hz and 10 Hz modulus comparison values are set to avoid conflicting with each other and all other block frequencies. In this way, a maximum of one block frequency is executed on any given 200 Hz cycle.

TABLE 2 AFC004 MODULUS DOWN SAMPLING AND STAGGER VALUES

Modulo Op Divisor	Frequency (Hz)	Period (ms)	Modulus
4	50	20	0
10	20	50	7
12	16	60	2
20	10	100	3

5.2.2 ARINC429 Bandwidth Limitations

The AFC004 transmits ARINC429 data to the PFD using a single 100 kBaud ARINC429 channel. Due to the amount of data being transmitted, the different message frequencies used, and the limited bandwidth of the ARINC429 channel, there is not a known strategy that would maintain ideal periodicity with all messages as some overlap would be inevitable. Therefore, it is important that a strategy is chosen that most closely maintains maximal periodicity with all messages. In addition to the above concerns regarding staggered block execution, the ARIN429 bandwidth limitations are a factor to be considered in selecting the modulus values. Modulus values were determined that optimized for both considerations. With the selected strategy, the sole frequency block effected by the ARINC429 bandwidth limitations is the 20 Hz block. On approximately 13.3% of the transmissions of the 20 Hz messages, the messages are delayed by ~2ms. This is well within design specifications of 10 ms maximum latency.

5.2.3 Other Timing Factors

Incoming ARINC429 data from the AHR75 must be time stamped to ensure that data is actively being received (no messages are lost) and to verify that the periodicity of the received data is within specification. There is no clock sync between the AFC004 and the AHR75. This means that the data received from the AHR75 could arrive at any time.

All ARINC429 communications take place using polling, as opposed to via interrupt. For a sufficiently accurate timestamp to be assigned, it is important that the AHR75 ARINC429 receive routines are executed often enough that no more than ~1.5 ms has expired since the last execution.

The 50 Hz block takes longer to execute than the other frequency blocks (how long?). In addition to a call within the main event loop, the AHR75 ARINC receive routines are called periodically inside the 50 Hz routine between time-consuming calculations. This guarantees that all AHR75 ARINC receive data is timestamped within 1.5 ms of receipt by the ARINC transceiver. Without these extra calls, up to 4 ms could pass before an AHR75 ARINC429 message is timestamped (worst case scenario).

5.3 ARINC429 Communications

5.3.1 ARINC429 Software Library Design

ASI's ARINC429 software library is used for all ARINC429 requirements. This includes processing received ARINC429 messages, assembling ARINC429 messages for transmitting, and retrieving valid messages based on receive interval.

5.3.1.1 ARINC429 Data Structures

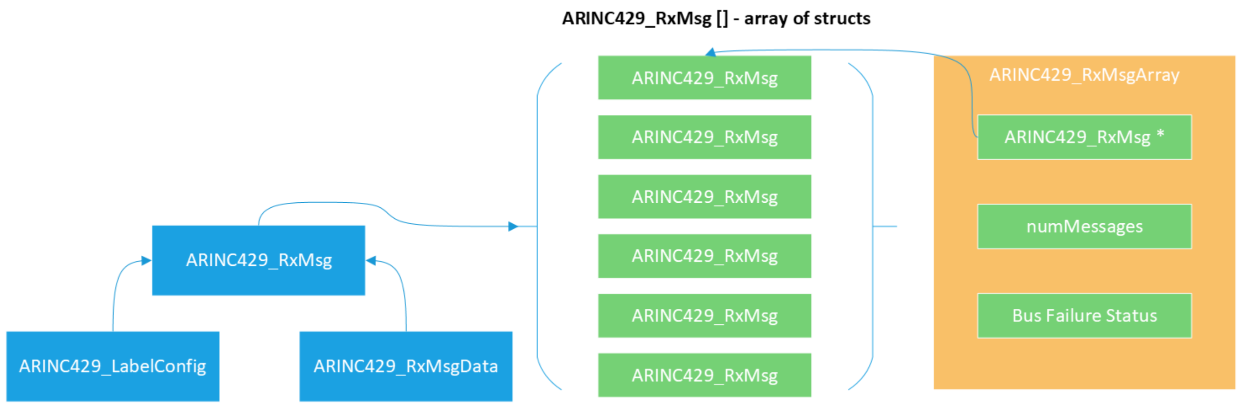


FIGURE 4 ARINC429 PRIMARY DATA STRUCTURES

A single ARINC429 received message (ARINC429_RxMsg) contains a constant label configuration and a data field structure. Similarly grouped ARINC429 receive messages can be grouped into an array of ARINC429_RxMsg structs. A single structure, ARINC429_RxMsgArray, provides an interface to the array of received messages. The ARINC429_RxMsgArray structure contains a pointer to the chosen array of ARINC429_RxMsg structs, the number of messages within the array of structs (typically calculated using the sizeof()), and fields regarding bus failure and timeout calculations.

For this ARINC429 library, message labels must be defined in hex-flipped format. These hex-flipped labels can be hand calculated, or they can be entered using the FormatLabelNumber macro, which allows the user to input the standard octal format while the macro converts to hex-flipped format. The ARINC429_RxMsg [] array can be defined as in inline struct declaration. This allows for more efficient code organization.

5.3.1.2 Processing Received Messages

When ARINC429 messages are popped from the receive FIFO, the raw ARINC429 message is processed into an ARINC429_RxMsgArray. Processing a received message involves timestamping, extracting, and converting to engineering data, storing SSM bits, storing SDI bits. The primary function to process received messages is ARINC429_ProcessReceivedMessage(). This function takes as input one ARINC429 message and a pointer to an ARINC429_RxMsgArray. ARINC429_ProcessReceivedMessage will perform a search through all ARINC429 messages stored in the RxMsgArray and check for a matching label. If a matching label is found, the message is processed, saved, and timestamped.

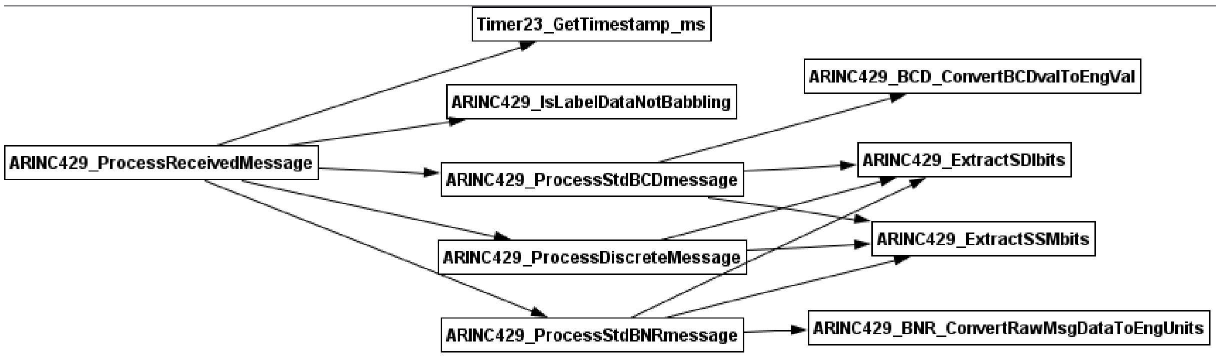

Timer23_GetTimestamp_ms
ARINC429_BCD_ConvertBCDvalToEngVal
ARINC429_IsLabelDataNotBabbling
ARINC429_ProcessReceivedMessage
ARINC429_ProcessStdBCDmessage
ARINC429_ExtractSDIbits
ARINC429_ExtractSSMbits
ARINC429_ProcessDiscreteMessage
ARINC429_ProcessStdBNRmessage
ARINC429_BNR_ConvertRawMsgDataToEngUnits

FIGURE 5 CALL GRAPH FOR ARINC429_PROCESSRECEIVEDMESSAGE

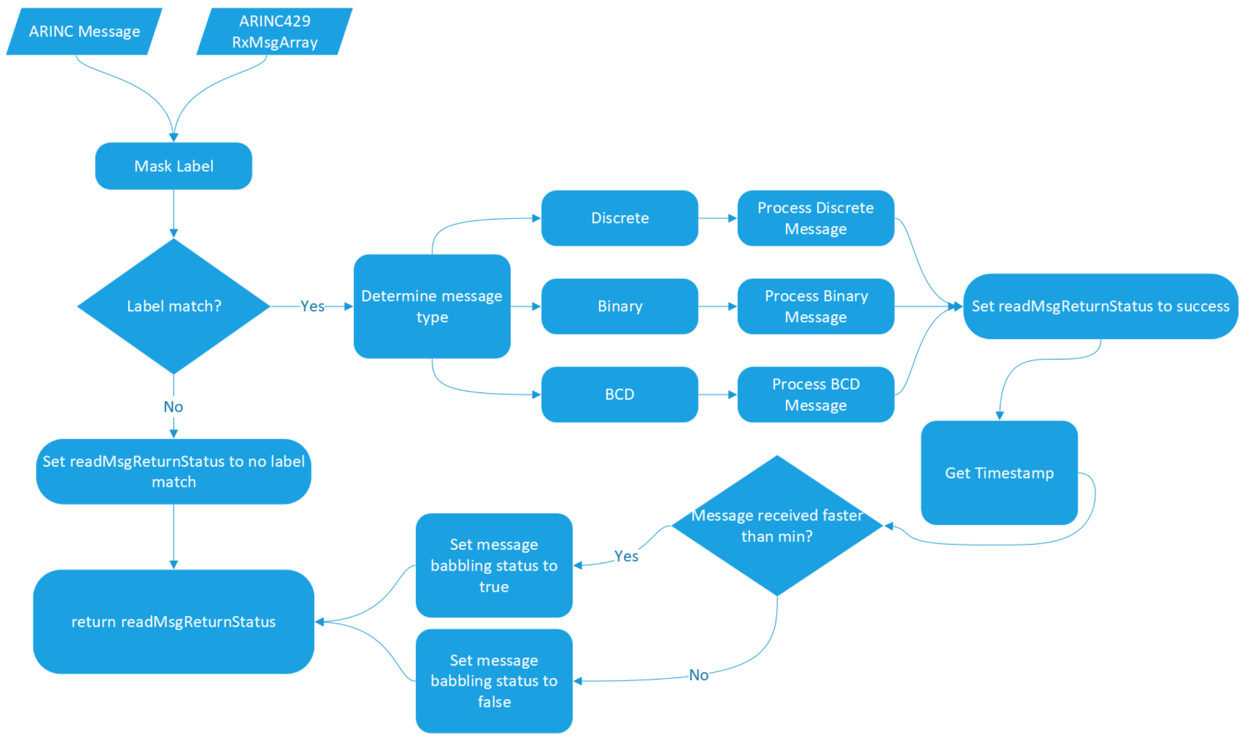

ARINC Message
ARINC429 RxMsgArray
Mask Label
Label match?
Yes
No
Determine message type
Discrete
Binary
BCD
Process Discrete Message
Process Binary Message
Process BCD Message
Set readMsgReturnStatus to success
Get Timestamp
Message received faster than min?
Yes
No
Set message babbling status to true
Set message babbling status to false
Set readMsgReturnStatus to no label match
return readMsgReturnStatus

FIGURE 6 ARINC429_PROCESSRECEIVEDMESSAGE LOGIC DIAGRAM

5.3.1.3 Assembling ARINC429 Messages

ARINC429 messages are assembled based on message type. The user declares an ARINC429_TxMsg, then manually sets the SDI, SSM, data, and message configuration. Based on the message type, the user passes as input the declared ARINC429_TxMsg and a pointer to the desired uint32_t variable which will be the assembled ARINC429 message.

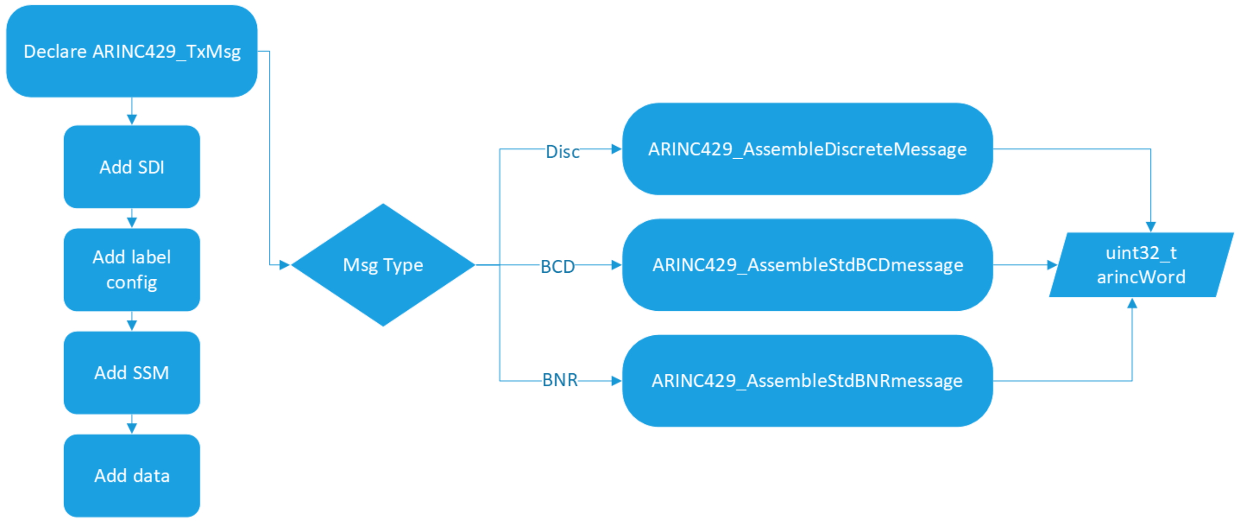


Figure 7 ARINC429 Message Assembly

5.3.1.4 ARINC Timing Considerations

The time sensitive nature of AHRS data combined with the AFC004's scheduling requirements make precise control over timing a must. All ARINC429 words that leave the AFC004 must have been received with a specified minimum and maximum transmit interval. When messages are received and processed, a timestamp is added to each individual message. If messages are received faster than the specified minimum receive interval, the message's isNotBabbling status is set to false. By design, any functions that have authority to transmit ARINC429 messages can either set the babbling message's SSM bits to failure warning, or simply not transmit the babbling message. This ensures that only ARINC429 messages received in a valid time interval shall leave the AFC004.

To access ARINC429 message data, the function ARINC429_GetLatestLabelData is used. This function is the primary interface to processed ARINC429 data. This function ensures that any received message data returned has the staleness flag checked. When a message is processed and received, the current 1ms timestamp is attached. Data fetched by GetLatestLabelData will get the current timestamp and compare to the message's last valid receive timestamp. Messages that have not received a valid message past the maximum receive interval are marked as stale. This function only modifies the staleness flag and has no authority to modify data. It is up to the user to handle all further modifications and decisions based on this status.

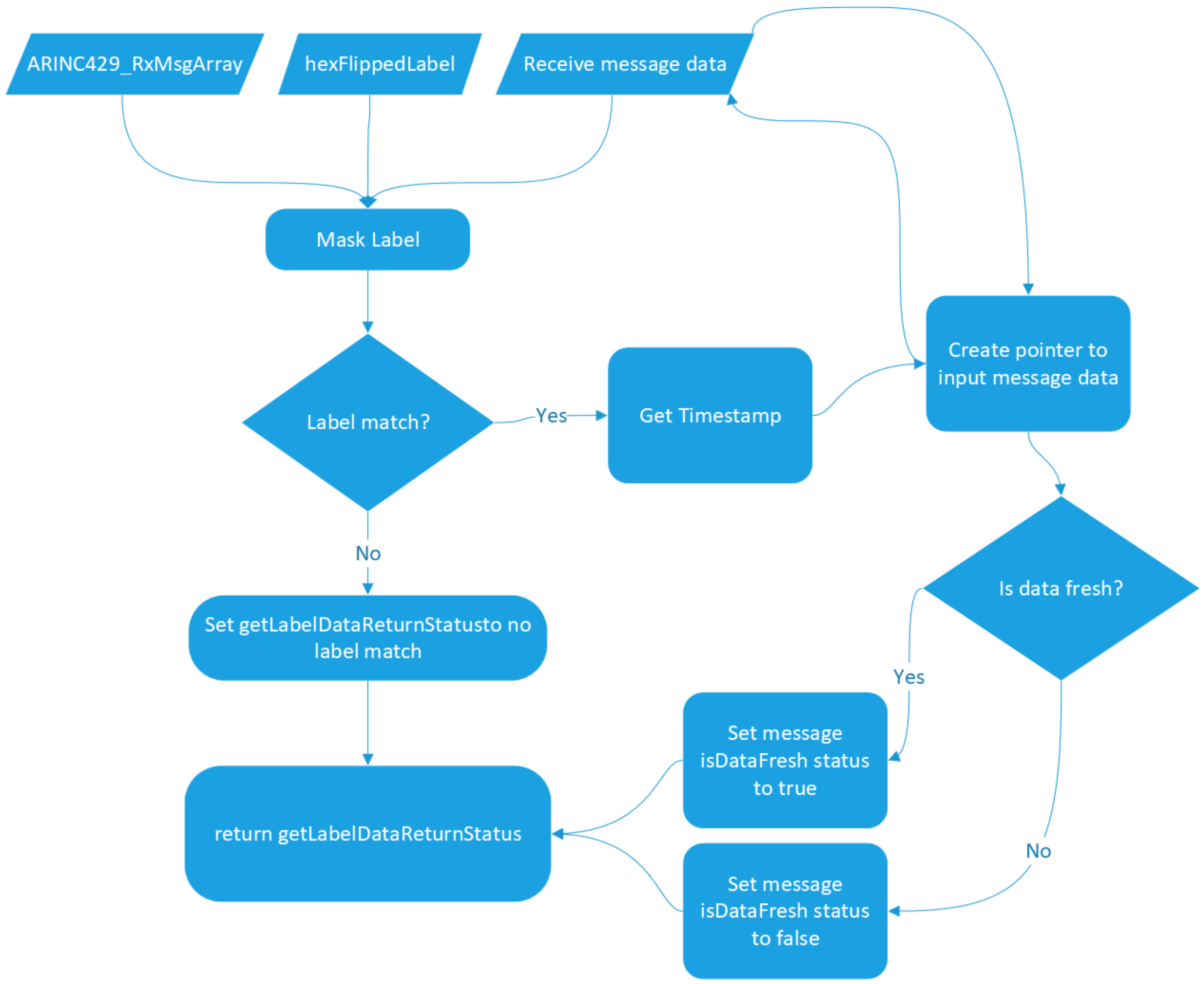

ARINC429_RxMsgArray
hexFlippedLabel
Receive message data
Mask Label
Label match?
Yes
Get Timestamp
Create pointer to input message data
No
Set getLabelDataReturnStatusto no label match
Is data fresh?
Yes
Set message isDataFresh status to true
return getLabelDataReturnStatus
No
Set message isDataFresh status to false

FIGURE 8 LOGIC DIAGRAM OF ARINC429_GETLATESTLABELDATA

5.3.2 HI-3584 Hardware Driver

The IOP connects to two independent ARINC429 transceivers. Both transceivers share a parallel data bus. The IOP handles all request and transmissions to each individual transceiver.

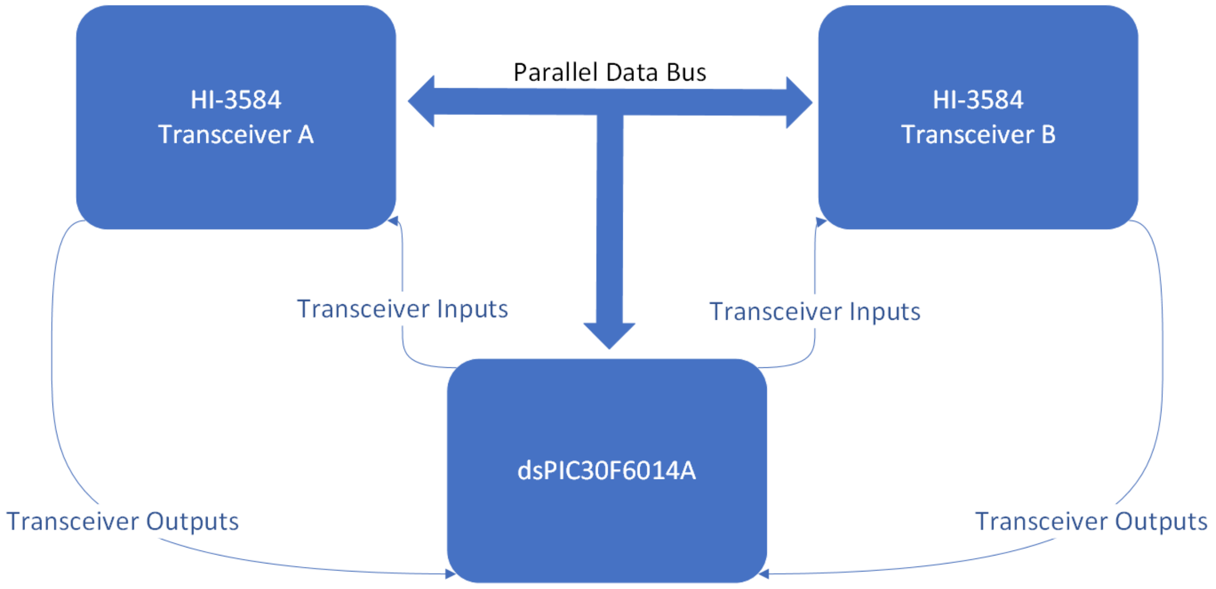


Figure 9 Dual ARINC429 Transceiver Configuration with shared data bus.

5.3.2.1 AFC004 Label Filtering

The HI-3584 ARINC429 transceivers are configured with hardware label filtering. The user defines the ARINC429_RxMsgArray with labels of interest. This ARINC429_RxMsgArray is used as an input to the respective transceiver's label filter. All labels stored in the ARINC429_RxMsgArray are used to setup the receiver label filter. An ARINC429_RxMsgArray must 16 or fewer labels of interest to use label filtering. When the labels are sent to the transceiver configuration register, all labels are read back and compared to the written values. If the readback values do not match the written values, the label filtering configuration bits are turned off.

5.3.2.2 Transmitting messages

ARINC429 messages can be transmitted by one of two methods: TransmitLatestARINCMsgIfValid and ARINC429_HI3584_txvrA_TransmitWord/ ARINC429_HI3584_txvrB_TransmitWord. Using TransmitLatestARINCMsgIfValid allows the user to input an ARINC429_RxMsgArray, an octal label, and transceiver channel. Based on these inputs, the function will search the input message array for a matching label. If a label match is found, the function checks the message's staleness and babbling flags. Only valid messages are transmitted. Messages that are stale or babbling are not transmitted. Calling ARINC429_HI3584_txvrA_TransmitWord or ARINC429_HI3584_txvrB_TransmitWord directly allows the user to transmit an ARINC429 message regardless of timing.

5.3.2.3 Reading receive FIFOs

DownloadMessagesFromARINCtxvrArx2 and DownloadMessagesFromARINCtxvrBrx2 is the interface between the HI-3584 hardware driver and the ARINC429 message processing library. These functions pop all available data from the receive FIFOs and process the respective messages into the input ARINC429_RxMsgArray. All processed messages are timestamped.

The HI-3584 receiver features a parity flag. The 32nd bit will be set to 1 if a parity error is detected in hardware. If a received message has a parity error, the message is discarded.

5.4 AFC004 ARINC Word Calculations and Formulas

5.4.1 Newly Calculated Words

5.4.1.1 Turn Rate

Turn rate is calculated by taking the derivative of magnetic heading. The magnetic heading is placed into an IIR differentiator with configurable coefficient values. The IIR differentiator must yield 10 successful magnetic heading values to be considered valid. This spooling period before the filter is considered stable, will output the turn rate engineering data, but the SSM bits shall be failed. Receiving an invalid magnetic heading will reset the filter's good count to zero and fail all turn rate messages until the filter's good count is greater than or equal to 10.

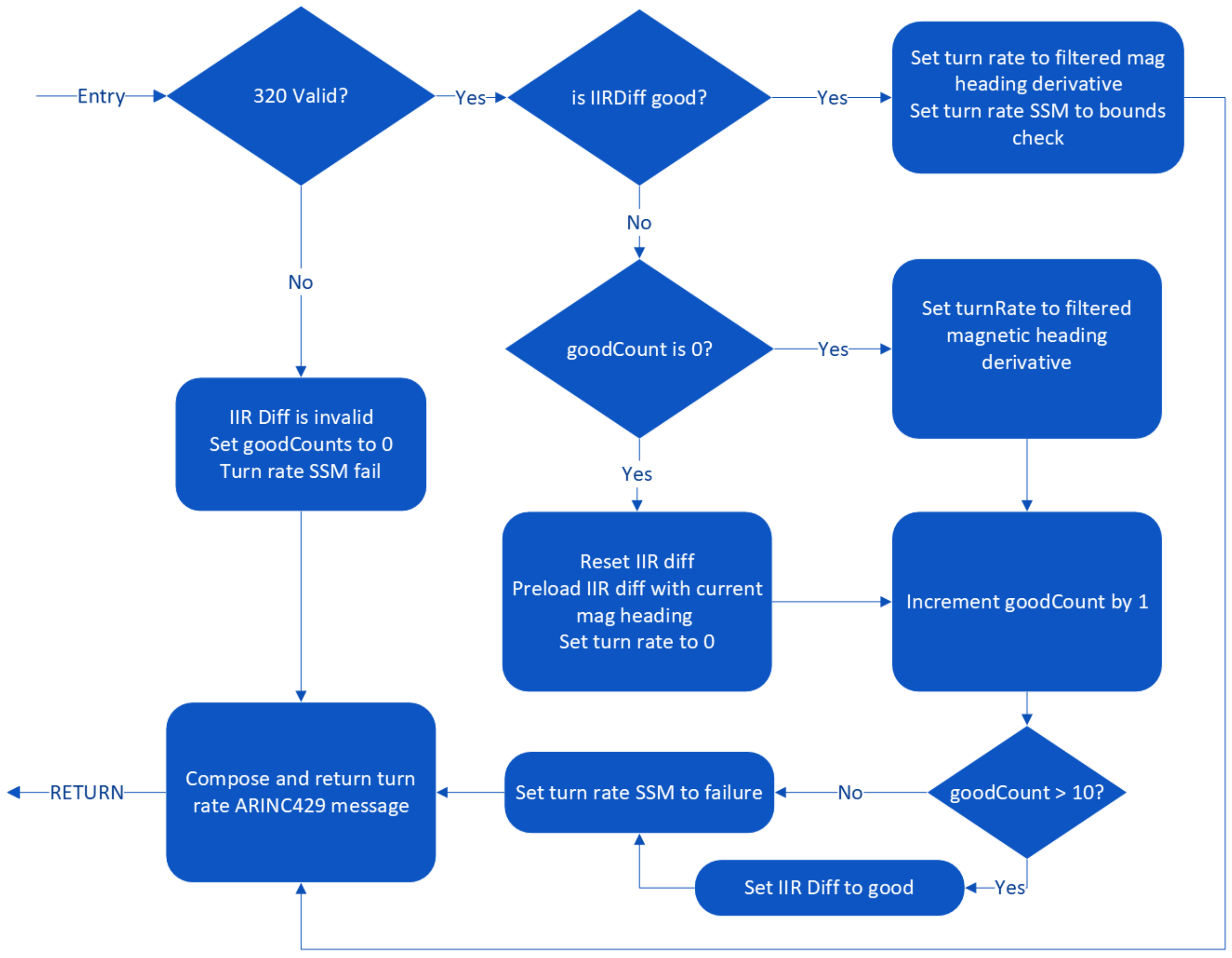

Entry
320 Valid?
Yes
is IIRDiff good?
Yes
Set turn rate to filtered mag heading derivative
Set turn rate SSM to bounds check
No
No
goodCount is 0?
Yes
Set turnRate to filtered magnetic heading derivative
IIR Diff is invalid
Set goodCounts to 0
Turn rate SSM fail
Yes
Reset IIR diff
Preload IIR diff with current mag heading
Set turn rate to 0
Increment goodCount by 1
RETURN
Compose and return turn rate ARINC429 message
Set turn rate SSM to failure
No
goodCount > 10?
Set IIR Diff to good
Yes

Figure 10 Turn Rate Logic Diagram

5.4.1.2 Slip Angle

Slip angle is calculated as: $slip = \tan^{-1}(\frac{lateral\ acceleration}{normal\ acceleration})$. Normal acceleration enters a standard IIR filter. The filter must receive 10 valid normal acceleration words to become valid. If the IIR filter is invalid, the SSM bits of slip angle are set to failure. If lateral acceleration SSM is received as failure, the slip angle's SSM bits will be set to failure, regardless of the IIR status filter. However, unlike normal acceleration, lateral acceleration does not require 10 valid messages to recover slip angle from an invalid SSM.

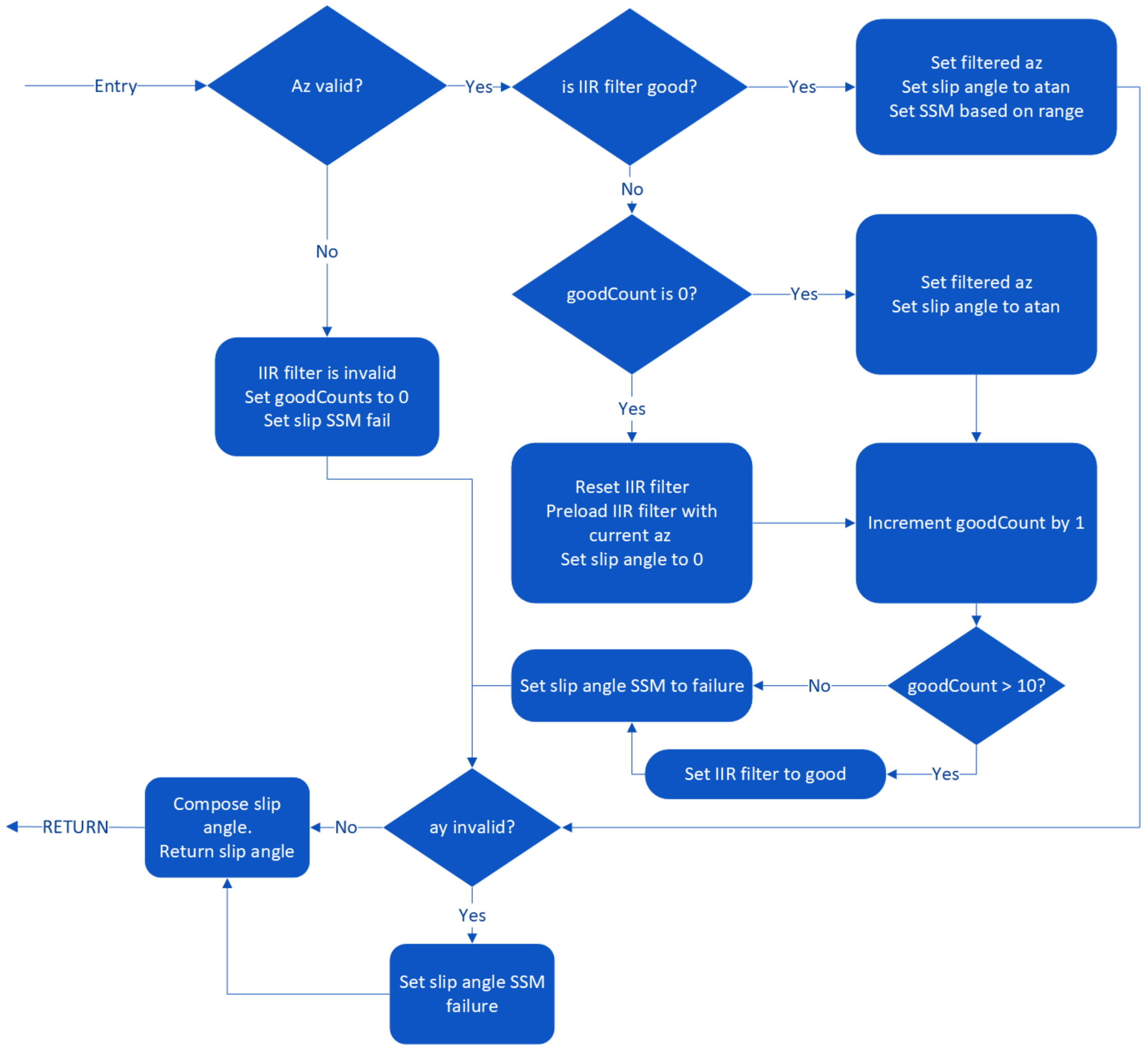


FIGURE 11 SLIP ANGLE LOGIC DIAGRAM

5.4.1.2.1 AHRS Status Words

AHR75 label 270 and 271 are used to calculate Eclipse's AHRS status label 272, 274, and 275. Each function takes as input the AHRS message array. Refer to ICD-AFC004 to AHRS status word definitions.

5.4.1.3 Modified Words

Connected to Archangel's AHR75, the AFC004 is configured to receive standardized ARINC705 AHRS data. Based on Eclipse's non-standard ARINC429 message configurations, the IOP must modify the words listed in **Table 3 Modified ARINC-705 Words**

TABLE 3 MODIFIED ARINC-705 WORDS

Label Name	Octal Label	Modification
Magnetic Heading	320	Change sig bits and resolution. Keep data
Pitch Angle	324	Change sig bits and resolution. Keep data
Roll Angle	325	Change sig bits and resolution. Keep data
Body Lateral Acceleration	332	Change polarity of data.
Body Normal Acceleration	333	Add +1.0 to engineering data.

Modified ARINC429 words are always transmitted. Despite the modification, all modified words require the original word to have been received from the AHR75 within the specified time interval. If a received message is babbling or stale, the modified message will be transmitted with the SSM bits set to failure.

5.4.1.4 Transmitted As-Is

Requiring no modification, the words listed in **Table 4 ARINC-705 Words Transmitted As-Is** are transmitted "as is". When scheduled to transmit, the IOP shall check the staleness and babbling flags of each message. If a message is stale or babbling, it is not transmitted. Only valid messages are transmitted.

TABLE 4 ARINC-705 WORDS TRANSMITTED AS-IS

Label Name	Octal Label
Body Pitch Rate	326
Body Roll Rate	327
Body Yaw Rate	330
Body Longitudinal Acceleration	331

5.5 RS422 Communications

The AFC004 interfaces with an external air data computer via RS422. The RS422 line is received asynchronously by the IOP via the UART peripheral. The RS422 messages contain ARINC429 data words in serial format. Eclipse's RS422 message format is predefined and described in **Figure 12 Eclipse Aviation RS422 Message Format.**

Header	Destination	Source	Command	Data	CRC-16
1 Byte	1 Byte	1 Byte	1 Byte	0-249 Bytes	2 Bytes

FIGURE 12 ECLIPSE AVIATION RS422 MESSAGE FORMAT

The RS422 message header is always 0xEA. The destination and source are hardware device codes defined by Eclipse. Command field is a unique code to each RS422 message's purpose. Data is a variable length field, which ranges from 0-

249 bytes long. Each message is appended with a 16 bit CRC residue. All received messages must yield a CRC calculation of 0 for the packet to be considered valid.

Individual data bytes are received at the UART peripheral on interrupt and pushed to a circular buffer. During the main operating loop, both receive circular buffers are searched for valid messages. If a valid message is found, the data portion is extracted and processed into respective ARINC429 messages.

RS422 messages are transmitted to the ADC at predefined scheduled intervals. The IOP composes a formatted message to transmit and pushes the data to the respective circular transmit buffer, where data is transmitted via interrupt.

5.5.1 RS422 Data Structures

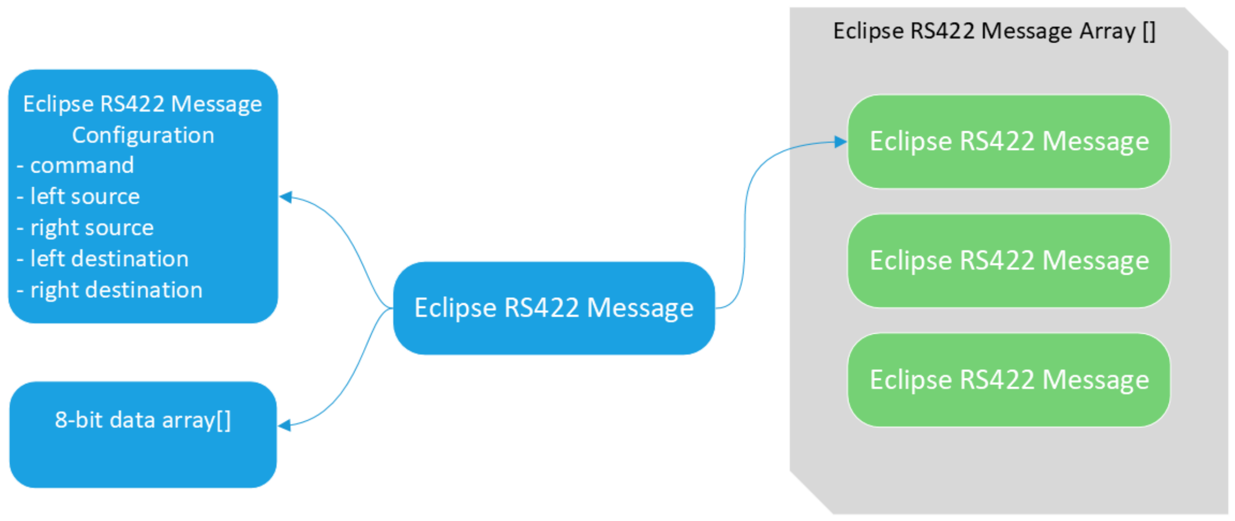


FIGURE 13 OVERVIEW OF RS422 MESSAGE DATA STRUCTURES

RS422 data structures involve a message configuration and data. The message configuration denotes each message's designated command, source, and destination identifiers, and length (length denotes size of cmd + data fields, in bytes). Device numbers are listed in the **Table 5 Eclipse RS422 Device Numbers**

TABLE 5 ECLIPSE RS422 DEVICE NUMBERS

Device Code	Value
Left AHRS	0x81
Right AHRS	0x82
Left ADC	0x85
Right ADC	0x86
Left PFD	0x51
Right PFD	0x52

The command field is specific to a message's unique purpose. The command fields used by the IOP are listed below in **Table 6 Eclipse RS422 Command Table**

TABLE 6 ECLIPSE RS422 COMMAND TABLE

Command Name	Value
Ground Maintenance	0x02
ADC Computed Data	0x30
ADC Status	0x31
AHRS Current Data	0x32
Software Version	0xF8
Hardware Serial Number	0xFA

5.5.2 Processing Received RS422 Messages

Data is received via RS422 on interrupt and stored in a receive circular buffer. During the mainloop, the receive circular buffer is repeatedly searched for valid messages. For every byte in the receive circular buffer, a search is performed to find a message header (0xEA). If a message header is found, all input messages are used to search the receive circular buffer for a matching command, source, destination, and length. If a match is found, the CRC is calculated over the length of the expected receive message. If a valid CRC is calculated, the data portion of the received message is flushed into a linear destination buffer, and the function returns a true status. This signals that ARINC429 messages are set to be processed.

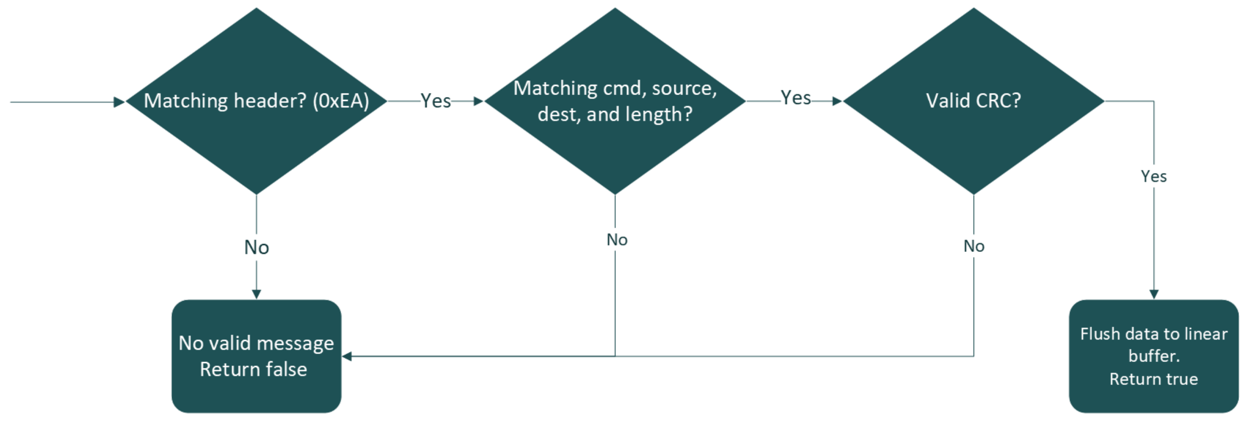

Matching header? (0xEA)
Yes
Matching cmd, source, dest, and length?
Yes
Valid CRC?
Yes
No
No
No
No valid message Return false
Flush data to linear buffer. Return true

Figure 14 EclipseRS422_ProcessNew Logic Diagram

5.5.3 Conversion from RS422 serial stream to ARINC429 data

When a valid RS422 message is found, the contents of the data portion are extracted. These data bytes contain ARINC429 messages (4 bytes long) broken down into individual bytes. Individual data bytes are concatenated together to form a single 32bit ARINC429 word. This word is processed through the function ARINC429_ProcessReceivedMessage, along with the corresponding ARINC429_RxMsgArray. It is important to note that all ARINC429 RS422 messages received by the IOP are predefined. For example, when using command 0x30, the air data computer transmits to the IOP the exact same 20 ARINC messages every time. Because of this, ARINC429_RxMsgArrays can be defined to match the specification of the words to be received.

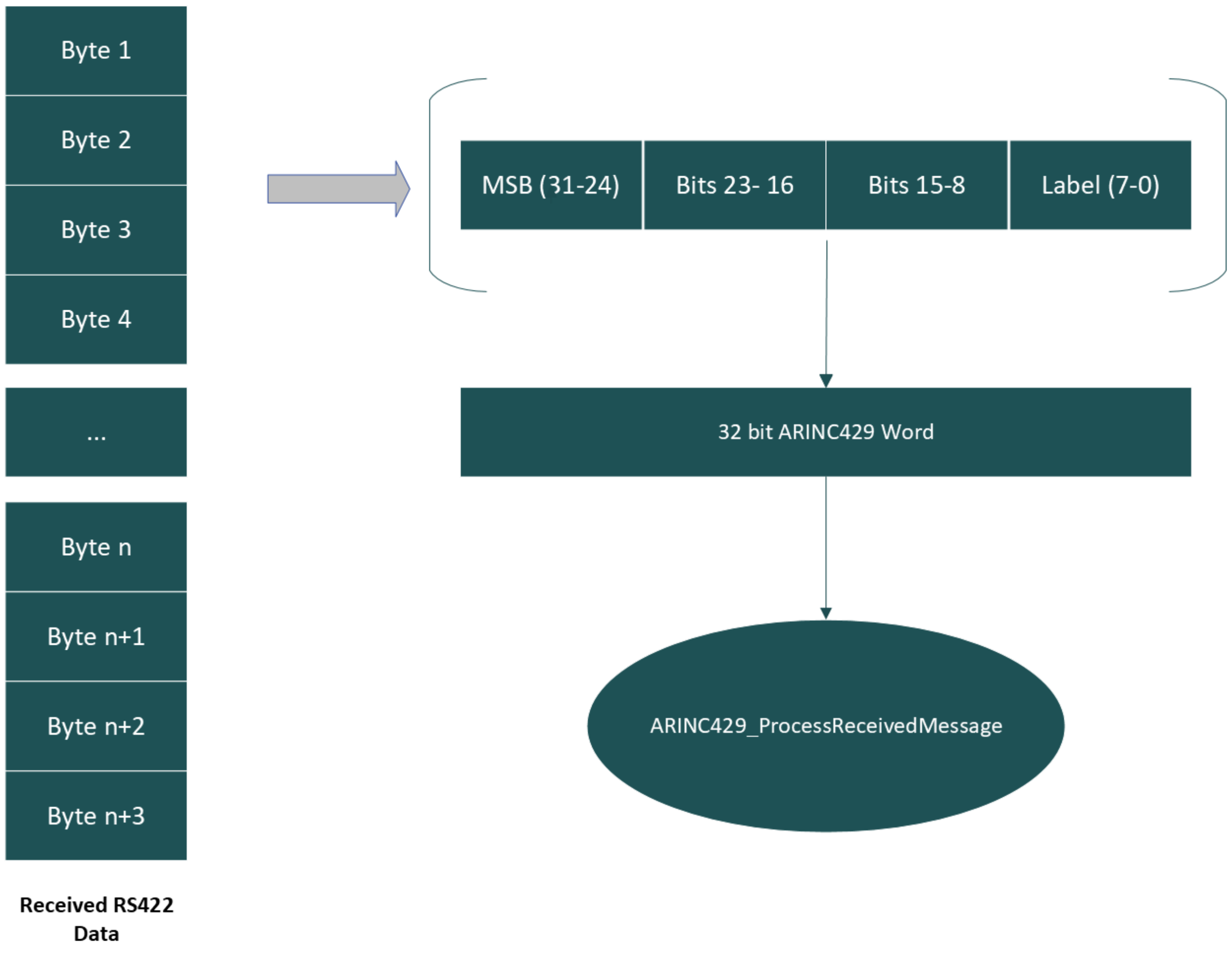


FIGURE 15 CREATION OF ARINC429 WORDS FROM RS422 DATA

5.5.4 Eclipse RS422 Transmit Message Construction

All RS422 messages transmitted to Eclipse's subsystems must match Eclipse's predefined RS422 message format. When constructing RS422 messages to transmit, the function EclipseRS422_ConstructTxMsg handles all formatting and data transfer. The function accepts as input an EclipseRS422msg (which contains the message configuration), a transmit circular buffer pointer, an array of ARINC429 words, the number of ARINC words to process, and the size of the message to transmit.

First, the input ARINC429 words are modified to calculate the odd parity of each message. Then, based on the RS422 message configuration, the header, destination, source, length, and command fields are populated. The data portion is populated by decomposing 4 byte ARINC429 messages into individual serial bytes. Once all ARINC words have been populated, the 16 bit CRC is calculated and appended to the end of the message. The message to transmit is then flushed into the desired transmit circular buffer.

Null data can be provided as input to EclipseRS422_ConstructTxMsg with the parameter arincArray, so long as the input parameter numArincWords is also zero. This is used for sending software and hardware version requests. Here, since no ARINC data is used, the message contains an empty data field. Because of this, the function

EclipseRS422_ConstructTxMsg can generically compose all desired RS422 transmit message types.

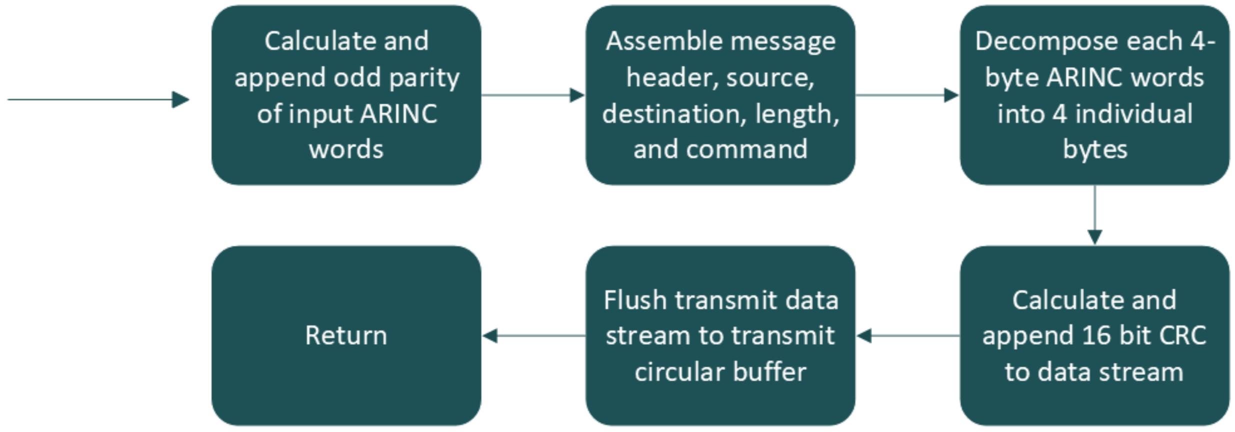

Calculate and append odd parity of input ARINC words
Assemble message header, source, destination, length, and command
Decompose each 4-byte ARINC words into 4 individual bytes
Calculate and append 16 bit CRC to data stream
Flush transmit data stream to transmit circular buffer
Return

Figure 16 EclipseRS422_ConstructTxMsg Logic Diagram

5.6 Bus Timeout Detection

Each bus has a respective receive message array associated with the expected messages to receive. Using the 100Hz frequency flag, every entry into the 100Hz section increments a counter attached to the predefined receive message arrays. When a valid message is received for each respective message array, the running counter is reset. This way, if a valid message is not received in the predefined counter interval, a timeout is detected.

5.7 Peripheral Software

5.7.1 UART Modules

The UART peripherals are designed to operate as non-blocking drivers. Received data is pushed to the UART's designated receive circular buffer. Data to transmit is first pushed to the UART's designated transmit circular buffer, then manually controlled to start transmission. The UART receive interrupt config is set to generate an interrupt when 4 bytes (FIFO is full) occupy the receive FIFO. To ensure that all available data resides in the receive circular buffer at the time of processing, the function UART(x)_ReadToRxCircBuff should be called. This function manually trips the UART receive interrupt flag. This ensures that in cases where 1, 2, or 3 bytes rest in the receive FIFO, the FIFO can be fully flushed at the time of processing.

As they are currently written, the UART drivers can work as a generic library so long as the circular buffer module is used alongside. Further development should make the circular buffer module type independent. Implementation-specific software must still be written to process serial data. However, the hardware abstraction layer is entirely generic, so long as hardware flow control is not required.

5.7.1.1 Circular Buffer Module

A static linear array of bytes (uint8_t) must first be declared. A circular buffer instance can be defined by creating a circBuffer_t type object and setting the data field equal to the statically declared linear array. In this module, head is the read pointer, and tail is the write pointer. Cb_reset should be called after initialization. Data can be pushed to the circular buffer using cb_push. Data can be read by calling cb_pop. The user can peek at a specific index by calling cb_peek. Optimized functions exist which allow for bulk data transfers. The function cb_advanceTail advances the circular buffer's tail by an input amount. To transfer a block of data *out* of a circular buffer, the function cb_flushOut copies data from a circular buffer and writes to a linear destination array. To transfer a block of data *into* a circular buffer, the function cb_flushIn transfers a linear destination array into a circular buffer.

5.7.2 Timer Modules

Two timers are used by the IOP. The Timer 4 module serves as the system's 100 Hz counter. The Timer 4 period register is configured to reset every 10 milliseconds and generate an interrupt. This interrupt sets a global 100 Hz flag to active. In the IOP's mainloop, the 100 Hz flag is repeatedly polled. If the IOP reads the 100 Hz flag is active, the IOP clears the 100 Hz flag, then executes all subsequent scheduled tasks.

Timer 2 and Timer 3 are combined to form a running 32 bit timer. This module uses the largest possible 32 bit period register. No interrupts are implemented. This 32 bit timer acts as a running 1 millisecond timer. During initialization, the user inputs the scale factor that equals the number of instructions, after prescale, that take 1 millisecond to complete. This allows a constant conversion between instruction counts and time.

5.8 Data Flow

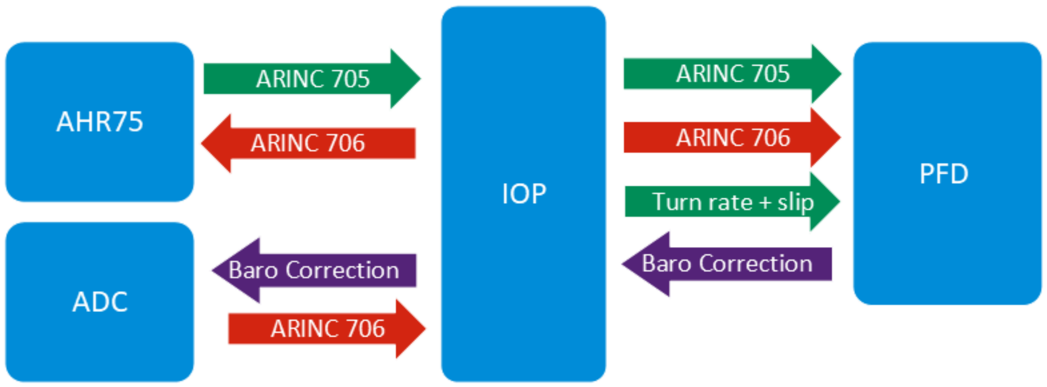


Figure 17 IOP Data Flow Diagram

ARINC-705 data comes from the AHR75 via ARINC-429 communications. The ARINC-706 originates from the external Air Data Computer which communicates to the IOP via RS422. These RS422 data bytes are converted into ARINC-429 words then processed and saved. During the primary scheduling routines, the ARINC-705 and ARINC-706 data are transmitted to the primary flight display. Along with the ARINC705 words, the IOP calculates and transmits slip angle and turn rate. The IOP is configured to receive barometric correction from the primary flight display, then shuttle that received baro correction to the ADC via RS422. The IOP transmits to the AHR75 TAS, CAS, and AoA via low speed ARINC429.

5.8.1 Inputs and Outputs

The table below describes the data flow elements with respect to the IOP.

Table 7 Data Inputs and Outputs

Data Element	Direction	Format
ARINC-705	INPUT (from AHR75)	ARINC 429 High Speed
ARINC-705	OUTPUT (to PFD)	ARINC 429 High Speed

ARINC-706	INPUT (from ADC)	RS422
Baro Correction	INPUT (from PFD)	ARINC 429 High Speed
Baro Correction	OUTPUT (to ADC)	RS422
Turn rate + slip	OUTPUT (to PFD)	ARINC 429 High Speed
ARINC-706	OUTPUT (to PFD)	ARINC 429 High Speed
ARINC-706	OUTPUT (to AHR75)	ARINC 429 Low Speed

5.9 Interrupt Operations

5.9.1 Trap Interrupts

Three trap interrupts are implemented to ensure that erroneous processor conditions do not contribute to sending corrupt or misleading data.

Table 8 List of Trap Interrupts

Trap Interrupt	**Function**
v_AddressError	Enters infinite loop in the event of an address error
v_OscillatorFail	Performs system reset if system PLL is unlocked
v_StackError	Enters infinite loop in the event of a stack error

5.9.2 Implementation Interrupts

For main application code, the IOP features 3 interrupt sources, listed in the table below.

Table 9 List of Interrupts Used in Main Code

Interrupt Source	Priority	Function
Timer4	5	Sets a 100 Hz system flag.
UART1 Receive	3	Pushes receive serial data to circular buffer.
UART1 Transmit	4	Transmits serial data from transmit circular buffer.

5.10 Resource Management

5.10.1 Timing Considerations

The primary 200Hz (5ms) scheduled function block shall remain below 80% of the allotted time. If functions executed within the 5 ms period take longer than 5 ms, latency issues could arise, timing could become staggered, and undefined behavior could result. To combat this, the 5 ms function is tested to ensure the timing limitation of 4 ms execution is achieved. At the beginning of the 5 ms function, a digital output pin is set high. Upon exit of the 5 ms function, the same digital output pin is set back to low. Using an oscilloscope, the pin is probed and the pulse width measured to ensure that the positive pulse width is never greater than 4 ms. After running for a period of 30 minutes, the maximum positive wave length (i.e., cycle time) is 1.72 ms.

To further combat the possibility of latency issues, the AHRS ARINC429 receive FIFO is periodically polled within the 5 ms function. This ensures that no receive ARINC429 message could sit stale in the receive FIFO for any period longer than 1.5 ms.

5.10.2 Memory Organization

The IOP uses static memory allocation only. No dynamic memory allocation is used.

The IOP uses a configuration block of memory stored in program memory. This reserved sector of memory allows changing configuration variables without modifying the executable code.

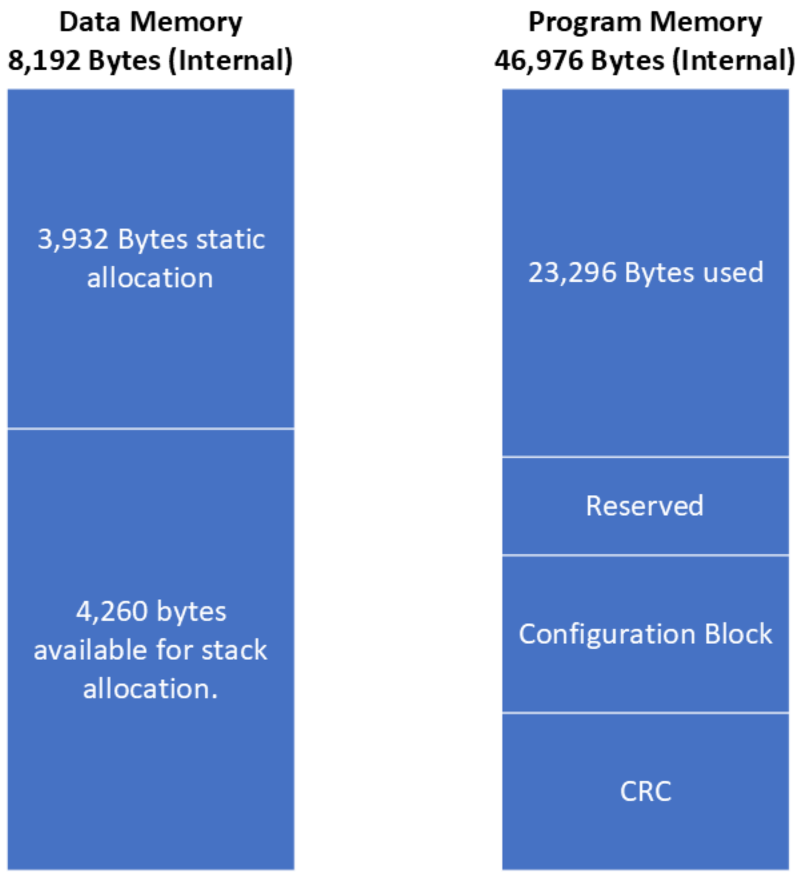


FIGURE 18 MEMORY ORGANIZATION FOR IOP

5.11 Deterministic State

At startup, the IOP must be place into a deterministic state. This is accomplished by calling the hardware reset configuration function, and by setting all unused microcontroller pins to digital zero outputs. The IOP’s pin assignments are described in Appendix C.

5.12 Additional Considerations

5.12.1 Deactivated code

The IOP shall contain no segments of deactivate/debug code.

5.12.2 Use of previously developed software

The IOP uses previously developed software from a previously certified SCI from the AHR150A. The baseline version of this reused code is described in Appendix A.

5.12.3 Multiple Version Dissimilar Software

No multiple version dissimilar software will be used.

5.12.4 User modifiable software

No user modifiable software will be used.

5.12.5 Software loading

Software shall be loaded in factory only using the in-house IPAQS system.

Low-Level Software Requirements

Requirements are developed using ASI work instruction *731-W-007-E ASI Requirements Development Work Instruction.*

Reliability Requirements

REL.0104.S.IOP.1

High level requirement description: The IOP ***shall*** verify stored code and configuration data in non-volatile memory, as part of Cold Boot BIT to facilitate detection of possible erroneous stored code and/or possible erroneous non-volatile memory.

REL.0104.S.IOP.1.001

Reused ARH150A requirement: **REL.0135.S.COM.7.003**

REL.0104.S.IOP.1.002

Reused ARH150A requirement: **REL.0135.S.COM.14.002**

REL.0104.S.IOP.1.003

Reused ARH150A requirement: **REL.0135.S.COM.7.003.D01**

REL.0104.S.IOP.1.004

Reused ARH150A requirement: **REL.0135.S.COM.7.005**

REL.0104.S.IOP.2

High level requirement description: The IOP shall test volatile memory as part of Cold Boot power on BIT to facilitate detection of a possible erroneous volatile memory.

REL.0104.S.IOP.2.001

Reused ARH150A requirement: **REL.0135.S.COM.8.001**

REL.0104.S.IOP.2.002

Reused ARH150A requirement: **REL.0135.S.COM.8.002**

REL.0104.S.IOP.3

High level requirement description: The IOP shall detect a possible overflow error and prevent a possible divide by zero error

REL.0104.S.IOP.3.001

Reused ARH150A requirement: **REL.0135.S.COM.10.004**

REL.0104.S.IOP.3.002

Reused ARH150A requirement: **REL.0135.S.COM.10.005**

REL.0104.S.IOP.3.003

Reused ARH150A requirement: **REL.0135.S.COM.10.006**

REL.0104.S.IOP.4

High level requirement description: The IOP shall filter sensor data.

REL.0104.S.IOP.4.001

Reused ARH150A requirement: **REL.0135.S.COM.15.001**

REL.0104.S.IOP.4.002

Reused ARH150A requirement: **REL.0135.S.COM.15.001.D02**

REL.0104.S.IOP.4.003

Reused ARH150A requirement: **REL.0135.S.COM.15.001.D03**

REL.0104.S.IOP.5

High level requirement description: The IOP shall initialize the system to a deterministic state at boot as defined in the ICDAFC004

REL.0104.S.IOP.5.001

Reused ARH150A requirement: **REL.0135.S.COM.16.001**

REL.0104.S.IOP.6

High level requirement description: A time base shall be used by the IOP to process data and detect errors.

REL.0104.S.IOP.6.001

Reused ARH150A requirement: **REL.0135.S.COM.17.001**

REL.0104.S.IOP.6.002

Reused ARH150A requirement: **REL.0135.S.COM.17.002**

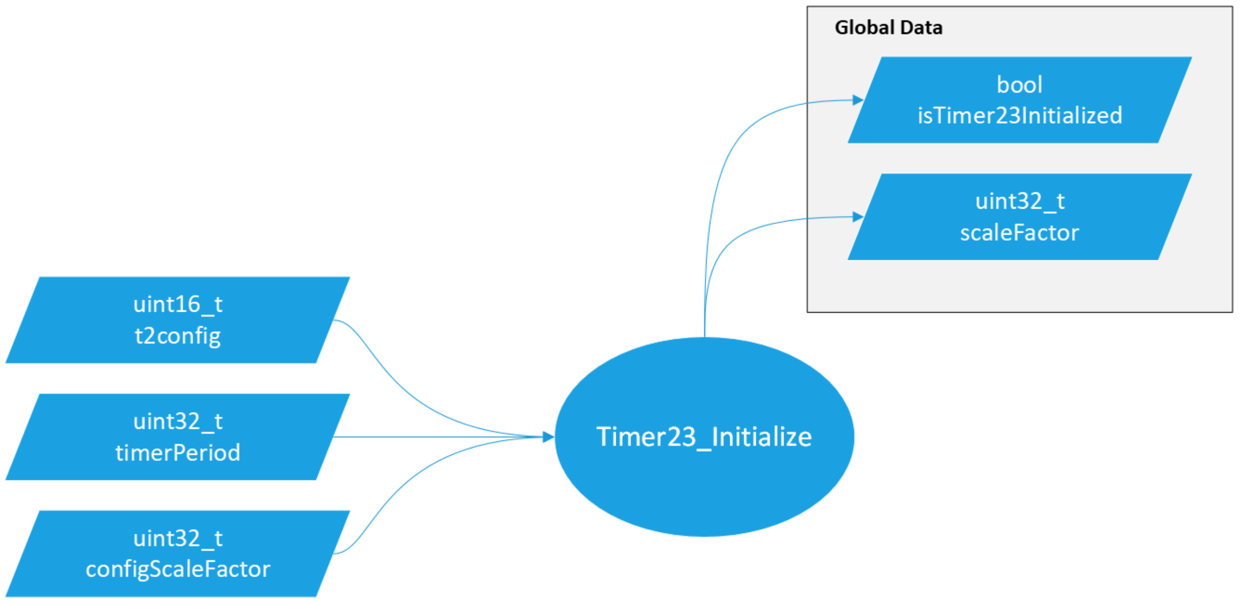
REL.0104.S.IOP.6.003

Reused ARH150A requirement: **REL.0135.S.COM.17.003**

REL.0104.S.IOP.7

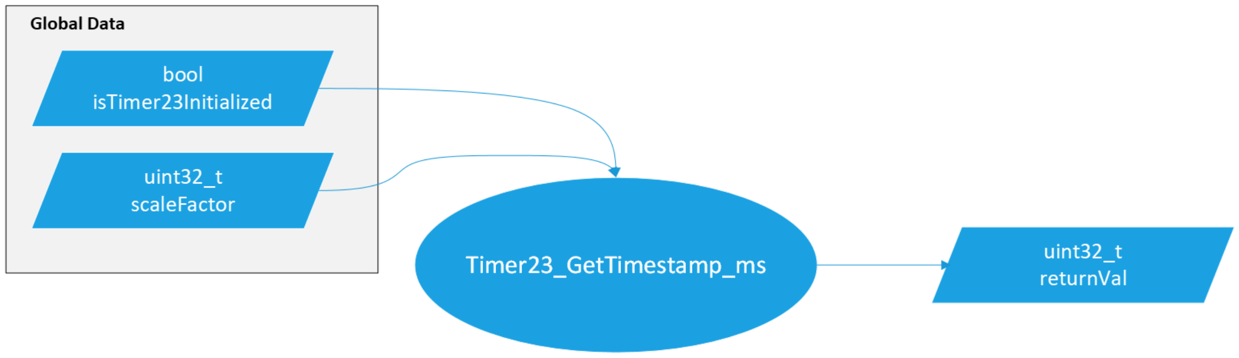
High level requirement description: The IOP ***shall*** start a running timer with 1 millisecond granularity

REL.0104.S.IOP.7.001



Description	The IOP ***shall*** initialize the 32bit timer pair using timer 2 and timer 3.
Function call:	Timer23_Initialize
Input parameters:	uint16_t **t2config**, uint32_t **timerPeriod**, uint32_t **configScaleFactor**
Global data:	uint32_t **scaleFactor**, bool **isTimer23Initialized**
Function static data:	None
Requirement	IF 0 equals **configScaleFactor** RETURN Set **scaleFactor** equal to **configScaleFactor** Set **T2CON** to **t2config** Set **PR3** to ((**timerPeriod** BITWISE-AND **MS_WORD_MASK**) RIGHT-SHIFT 16) Set **PR2** to **timerPeriod** BITWISE-AND **LS_WORD_MASK** Set **isTimer23Initialized** to true Set **IEC0bits.T3IE** to 0 Set **IEC0bits.T2IE** to 0 RETURN

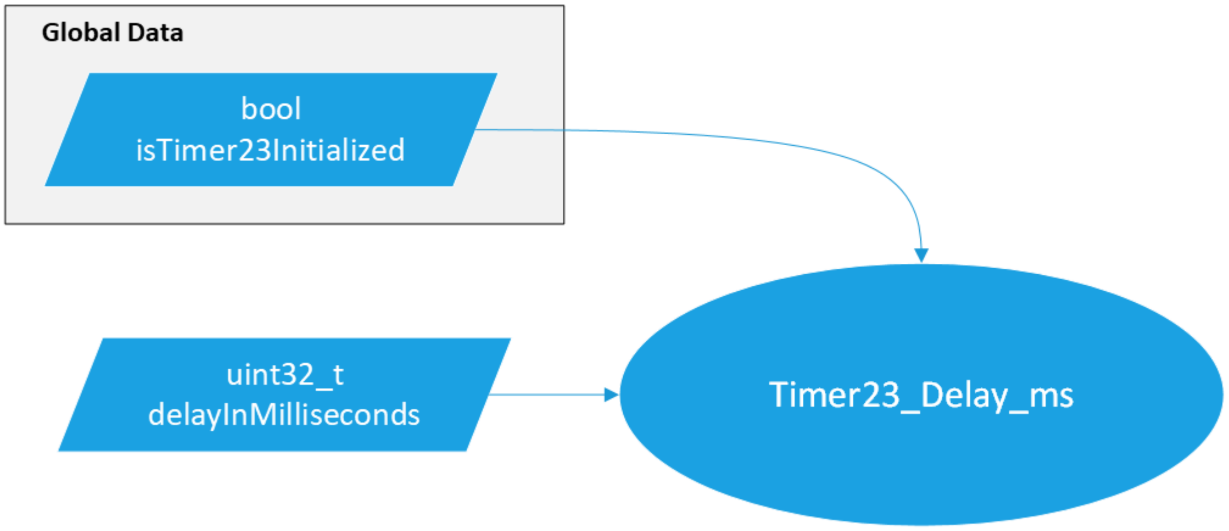
REL.0104.S.IOP.7.002



Description	The IOP ***shall*** get the current timestamp in milliseconds
Function call	Timer23_GetTimestamp_ms
Input parameters	None
Global data:	uint32_t **scaleFactor**, bool **isTimer23Initialized**
Function static data:	None

Requirement	Declare **returnVal**
	IF true equals **isTimer23Initialized** AND 0 does not equal **scaleFactor**
	    Set **lsWord** equal to **TMR2**
	    Set **msWord** equal to **TMR3HLD**
	    Set **returnVal** equal to ((**msWord** LEFT-SHIFT 16) BITWISE-OR **lsWord**))/ **scaleFactor**
	ELSE
	    Set **returnVal** equal to 0
	RETURN **returnVal**

REL.0104.S.IOP.7.003



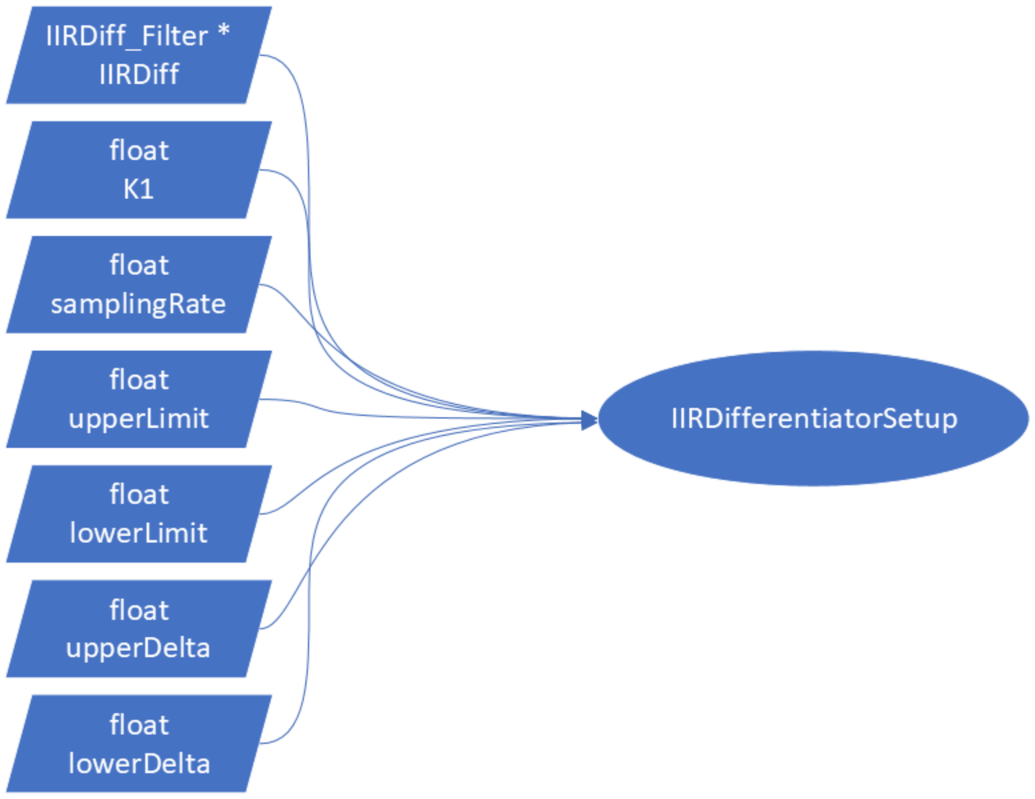
Description	The IOP ***shall*** enter a delay, with millisecond granularity, at the user specified millisecond delay. A maximum delay of 1 second shall be enforced
Function call	Timer23_Delay_ms
Input parameters	uint32_t **delayInMilliseconds**
Global data:	bool **isTimer23Initialized**
Function static data:	None

Requirement	IF false equals **isTimer23Initialized**
	    RETURN
	IF **delayInMilliseconds** is greater than **MAX_DELAY_MS**
	      Set **delayInMilliseconds** to **MAX_DELAY_MS**
	Set **startTimestamp** to Timer23_GetTimestamp_ms ()
	Declare **currentTimestamp**
	WHILE TRUE
	    Set **currentTimestamp** to Timer23_GetTimestamp_ms ()
	    IF (**currentTimestamp** - **startTimestamp**) is greater than **delayInMilliseconds**
	        BREAK
	RETURN

REL.0104.S.IOP.8

High level requirement description: The IOP ***shall*** use an IIR Differentiator to filter and differentiate incoming data.

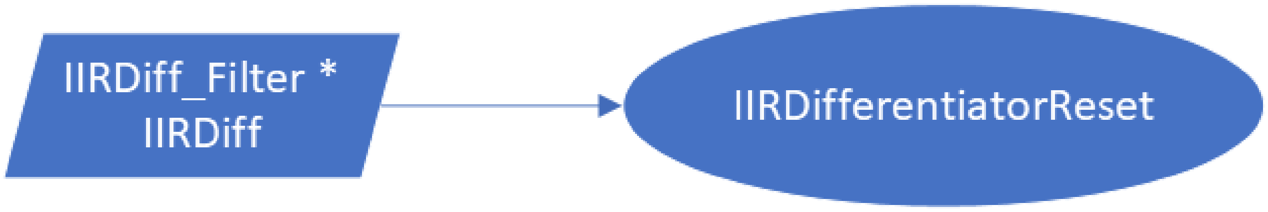
REL.0104.S.IOP.8.001



Description	The IOP ***shall*** setup an IIR Differentiator based on input parameters
Function call:	IIRDifferentiatorSetup
Input parameters:	IIRDiff_Filter * **IIRDiff**, float K1, float **samplingRate**, float **upperLimit**, float **lowerLimit**, float **upperDelta**, float **lowerDelta**
Global data:	None
Function static data:	None
Requirement	IF NULL equals IIRDiff RETURN Set IIRDiff.config.k1 equal to K1 Set IIRDiff.config.k2 equal to 1.0f – K1 Set IIRDiff.config.samplingRate_Hz equal to samplingRate Set IIRDiff.config.upperDelta equal to upperDelta Set IIRDiff.config.lowerDelta equal to lowerDelta

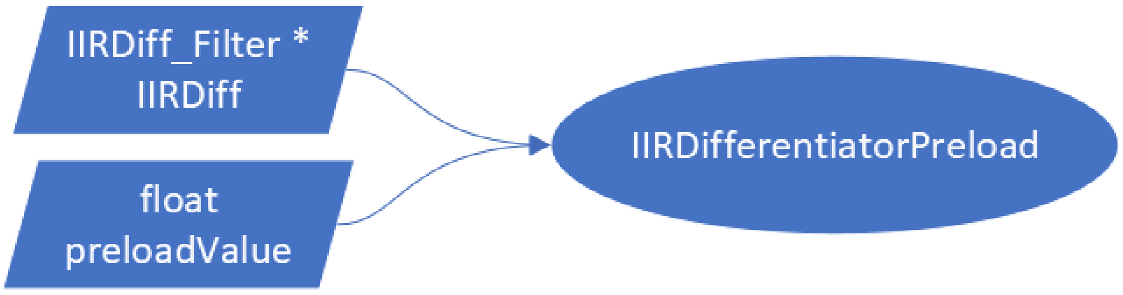
	Set IIRDiff.config.upperLimit equal to upperLimit Set IIRDiff.config.lowerLimit equal to lowerLimit RETURN

REL.0104.S.IOP.8.002



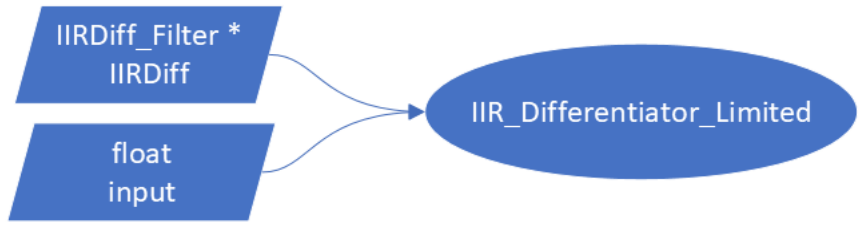
Description	The IOP ***shall*** reset an IIRDiff filter
Function call:	IIRDifferentiatorReset
Input parameters:	IIRDiff_Filter * IIRDiff
Global data:	None
Function static data:	None
Requirement	Set IIRDiff.pastOutputOfDiff equal to 0.0f Set IIRDiff.pastInputOfDiff equal to 0.0f Set IIRDiff.preloadValue equal to DIFF_PRELOAD_FLAG_SET RETURN

REL.0104.S.IOP.8.003



Description	The IOP ***<u>shall</u>*** preload an input value into an existing IIRDiff_Filter
Function call:	IIRDifferentiatorPreload
Input parameters:	float preloadValue, IIRDiff_Filter * IIRDiff
Global data:	None
Function static data:	None
Requirement	Set IIRDiff.pastInputOfDiff equal to preloadValue Set IIRDiff.pastOutputOfDiff equal to 0.0f Set IIRDiff.preloadValue equal to DIFF_PRELOAD_FLAG_CLEAR RETURN

REL.0104.S.IOP.8.004



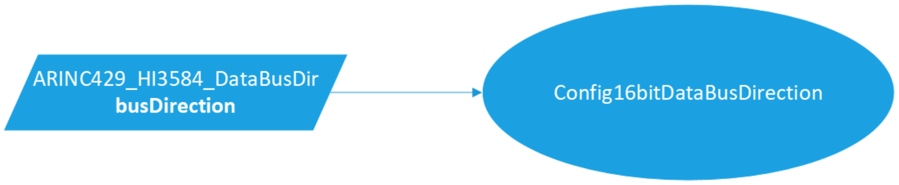
Description	The IOP ***shall*** perform an IIR filtered differentiation based on an input value. The upper and lower limit of the input data is controlled between the filter's configurable values for upper and lower limit.
Function call:	IIR_Differentiator_Limited
Input parameters:	float **input**, IIRDiff_Filter * **IIRDiff**
Global data:	None
Function static data:	None
Requirement	Declare **filteredOutput** Declare **valueToAdd** IF **IIRDiff.preloadValue** equals DIFF_PRELOAD_FLAG_SET Call IIRDifferentiatorPreload(**input**, **IIRDiff**) Set **valueToAdd** equal to input - **IIRDiff.pastInputOfDiff** WHILE **valueToAdd** is greater than **IIRDiff.config.upperLimit** **valueToAdd** plus-equals **IIRDiff.config.lowerDelta** WHILE **valueToAdd** is less than **IIRDiff.config.lowerLimit** **valueToAdd** plus-equals **IIRDiff.config.upperDelta** Set **filteredOutput** equal to **IIRDiff.config.k2** * **IIRDiff.config.samplingRate_Hz** * valueToAdd + **IIRDiff.config.k1** * **IIRDiff.pastOutputOfDiff** Set **IIRDiff.pastOutputOfDiff** equal to **filteredOutput** SEt **IIRDiff.pastInputOfDiff** equal to RETURN **filteredOutput**

Interface Requirements

INT1.0101.S.IOP.1

High level requirement description: The IOP ***shall*** transmit and receive ARINC429 data words through the HI-3584 chip.

INT1.0101.S.IOP.1.001



Description	The IOP ***shall*** configure the parallel ARINC429 data bus direction as input or output, depending on the input direction value.
Function call:	Config16bitDataBusDirection
Input parameters:	ARINC429_HI3584_DataBusDir **busDirection**
Global data:	None
Function static data:	None
Requirement	Declare **trisVal** IF (**busDirection** equals **ARINC429_HI3584_DATA_DIR_INPUT**) Set **trisVal** equal to 1 ELSE Set **trisVal** equal to 0 Set **DB00_TRIS** equal to **trisVal** Set **DB01_TRIS** equal to **trisVal** Set **DB02_TRIS** equal to **trisVal** Set **DB03_TRIS** equal to **trisVal** Set **DB04_TRIS** equal to **trisVal** Set **DB05_TRIS** equal to **trisVal** Set **DB06_TRIS** equal to **trisVal** Set **DB07_TRIS** equal to **trisVal** Set **DB08_TRIS** equal to **trisVal** Set **DB09_TRIS** equal to **trisVal** Set **DB10_TRIS** equal to **trisVal** Set **DB11_TRIS** equal to **trisVal** Set **DB12_TRIS** equal to **trisVal** Set **DB13_TRIS** equal to **trisVal** Set **DB14_TRIS** equal to **trisVal** Set **DB15_TRIS** equal to **trisVal** RETURN

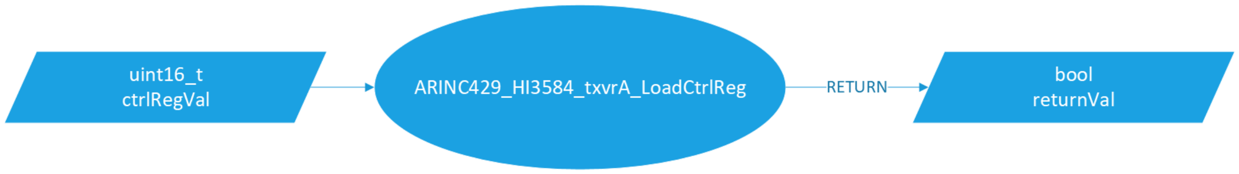
INT1.0101.S.IOP.1.002
Description: The IOP ***<u>shall</u>*** read the data from the 16 bit data bus interfaced with the ARINC devices
Function call: ReadDataFrom16bitDataBus
Input parameters: None
Requirement reuse: INT1.0102.S.IOP.6.003.D02

INT1.0101.S.IOP.1.003
Description: The IOP shall write the data on to the 16 bit data bus interfaced with the ARINC devices.
Function call: WriteDataTo16bitDataBus
Input parameters: uint16_t **dataBusWriteValue**
Requirement reuse: INT1.0102.S.IOP.6.003.D01

INT1.0101.S.IOP.1.004

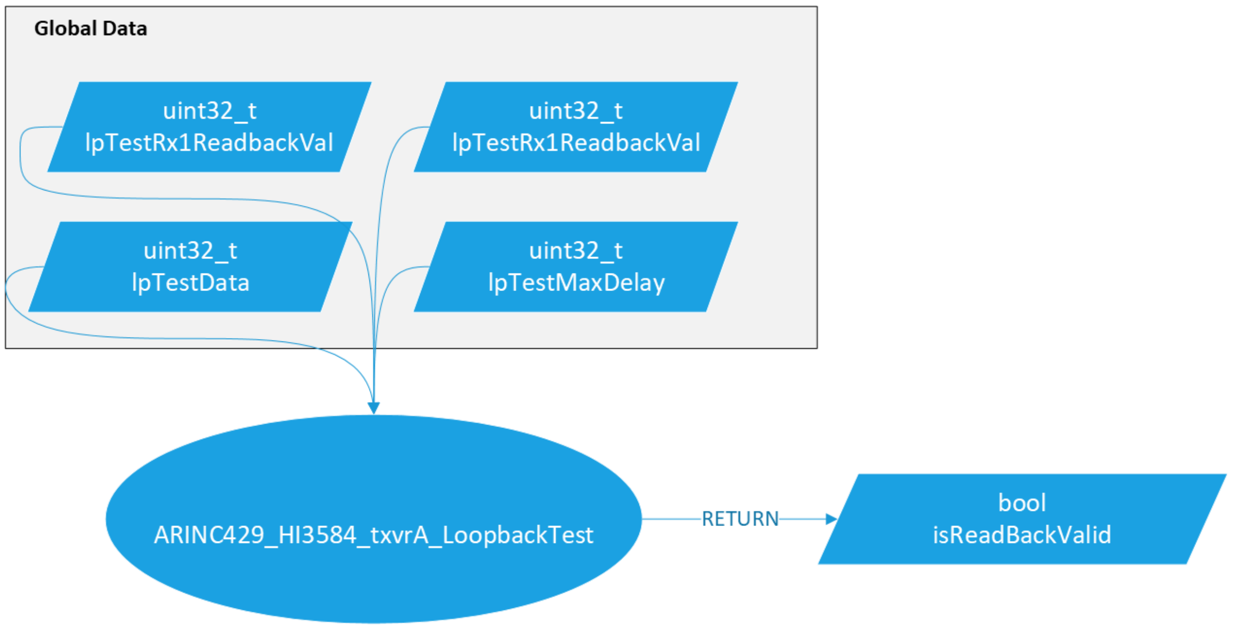
Description	The IOP ***shall*** initialize ARINC429 Transceiver A
Function call:	ARINC429_HI3584_txvrA_Initialize
Input parameters:	None
Global data:	None
Function static data:	None
Requirement	Set **ARINC429_HI3584_TXVRA_SEL_TRIS** equal to 0 Set **ARINC429_HI3584_TXVRA_SEL** equal to 0 Set **ARINC429_HI3584_TXVRA_EN1_TRIS** equal to 0 Set **ARINC429_HI3584_TXVRA_EN1** equal to 1 Set **ARINC429_HI3584_TXVRA_EN2_TRIS** equal to 0 Set **ARINC429_HI3584_TXVRA_EN2** equal to 1 Set **ARINC429_HI3584_TXVRA_PL1_TRIS** equal to 0 Set **ARINC429_HI3584_TXVRA_PL1** equal to 1 Set **ARINC429_HI3584_TXVRA_PL2_TRIS** equal to 0 Set **ARINC429_HI3584_TXVRA_PL2** equal to 1 Set **ARINC429_HI3584_TXVRA_ENTX_TRIS** equal to 0 Set **ARINC429_HI3584_TXVRA_ENTX** equal to 1 Set **ARINC429_HI3584_TXVRA_CWSTR_TRIS** equal to 0 Set **ARINC429_HI3584_TXVRA_CWSTR** equal to 1 Set **ARINC429_HI3584_TXVRA_RSR_TRIS** equal to 0 Set **ARINC429_HI3584_TXVRA_RSR** equal to 1 Set **ARINC429_HI3584_TXVRA_DR1_TRIS** equal to 1 Set **ARINC429_HI3584_TXVRA_DR2_TRIS** equal to 1 Set **ARINC429_HI3584_TXVRA_FFT_TRIS** equal to 1 Call Config16bitDataBusDirection (**ARINC429_HI3584_DATA_DIR_INPUT**) RETURN

INT1.0101.S.IOP.1.005



Description	The IOP ***shall*** load the ARINC429 Transceiver A control register with the input value. The function returns true if the control register readback was successful, false if readback was unsuccessful
Function call:	ARINC429_HI3584_txvrA_LoadCtrlReg
Input parameters:	uint16_t **ctrlRegVal**
Global data:	None
Function static data:	None
Requirement	Set **ARINC429_HI3584_TXVRA_SEL** equal to 0 Set **ARINC429_HI3584_TXVRA_CWSTR** equal to 0 Call Config16bitDataBusDirection (**ARINC429_HI3584_DATA_BUS_DIR_OUTPUT**) Call WriteDataTo16bitDataBus(**ctrlRegVal**) Set **ARINC429_HI3584_TXVRA_CWSTR** equal to 1 Set **readBack** equal to ARINC429_HI3584_txvrA_ReadBackControlRegister () IF (**ctrlRegVal** equals **readBack**) RETURN true ELSE RETURN false

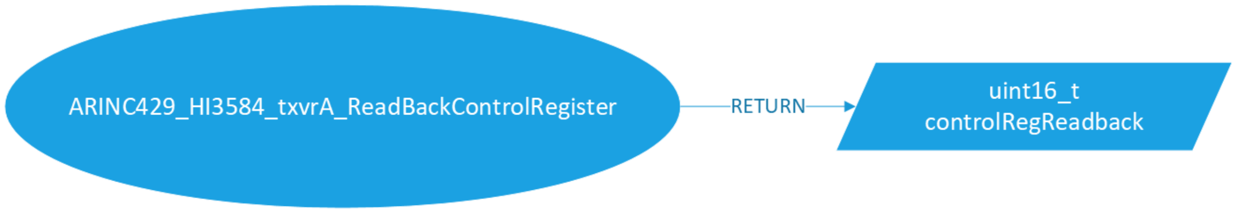
INT1.0101.S.IOP.1.006


Global Data
uint32_t lpTestRx1ReadbackVal
uint32_t lpTestRx1ReadbackVal
uint32_t lpTestData
uint32_t lpTestMaxDelay
ARINC429_HI3584_txvrA_LoopbackTest
RETURN
bool isReadBackValid

Description	The IOP ***shall*** conduct a hardware-level loopback test on ARINC429 transceiver A. The function returns true if the loopback test was successful, false if unsuccessful
Function call:	ARINC429_HI3584_txvrA_LoopbackTest
Input parameters:	None
Global data:	uint32_t lpTestRx1ReadbackVal, uint32_t lpTestRx2ReadbackVal, uint32_t lpTestData, uint32_t lpTestMaxDelay
Function static data:	None
Requirement	Set currentCtrRegValue equal to ARINC429_HI3584_txvrA_ReadBackControlRegister( ) Set status equal to ARINC429_HI3584_txvrA_LoadCtrlReg(0x8000) Set currFIFOflushCount equal to 0 WHILE currFIFOflushCount is less than or equal to txvrRxFIFOsize Call ARINC429_HI3584_txvrA_rx1_ReadWord( ) Increment currFIFOflushCount by 1 Set currFIFOflushCount equal to 0 WHILE currFIFOflushCount is less than or equal to txvrRxFIFOsize ARINC429_HI3584_txvrA_rx2_ReadWord( ) Increment currFIFOflushCount by 1 Set counter equal to Set rx1readback equal to lpTestRx1ReadbackVal Set rx2readback equal to lpTestRx2ReadbackVal

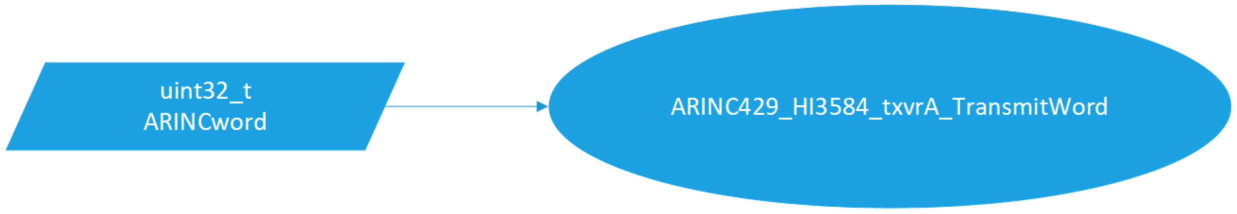
```
WHILE ((counter is less than lpTestNumCycles) AND
        (((lpTestRx1ReadbackVal equals rx1readback) AND (lpTestRx2ReadbackVal
         equals rx2readback)) OR (1 equals counter)))
   Call ARINC429_HI3584_txvrA_TransmitWord( lpTestData )
   Set delayCounter equal to 0
   WHILE (((1 equals ARINC429_HI3584_TXVRA_DR1) OR (1 equals ARINC429_HI3584_TXVRA_DR2)) AND (delayCounter is less than lpTestMaxDelay))
      Increment delayCounter by 1
   Set rx1readback equal to ARINC429_HI3584_txvrA_rx1_ReadWord( )
   Set rx2readback equal to ARINC429_HI3584_txvrA_rx2_ReadWord( )
   Increment counter by 1
IF ((lpTestRx1ReadbackVal equals rx1readback) AND (lpTestRx2ReadbackVal equals rx2readback))
   Set status BITWISE-AND equals true
ELSE
   Set status BITWISE-AND equals false
Call ARINC429_HI3584_txvrA_LoadCtrlReg( currentCtrRegValue )
RETURN status
```

INT1.0101.S.IOP.1.007


ARINC429_HI3584_txvrA_ReadBackControlRegister
RETURN
uint16_t controlRegReadback

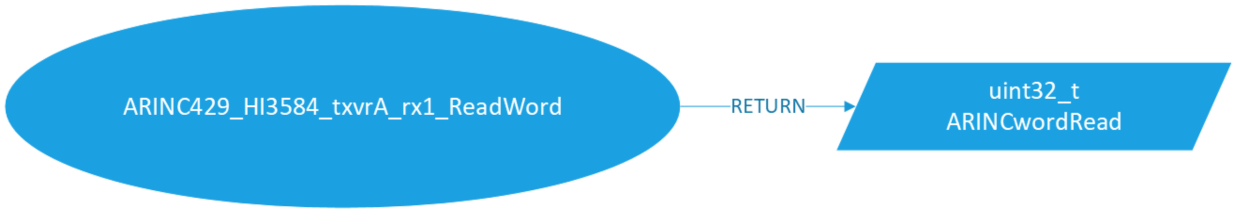
Description	The IOP ***shall*** read back the current control register value of ARINC429 transceiver A
Function call:	ARINC429_HI3584_txvrA_ReadBackControlRegister
Input parameters:	None
Global data:	None
Function static data:	None
Requirement	Call Config16bitDataBusDirection(**ARINC429_HI3584_DATA_DIR_INPUT**) Set **ARINC429_HI3584_TXVRA_SEL** equal to 1 Set **ARINC429_HI3584_TXVRA_RSR** equal to 0 Set **controlRegReadback** equal to ReadDataFrom16bitDataBus() Set **ARINC429_HI3584_TXVRA_RSR** equal to 1 Set **ARINC429_HI3584_TXVRA_SEL** equal to 0 RETURN **controlRegReadback**

INT1.0101.S.IOP.1.008



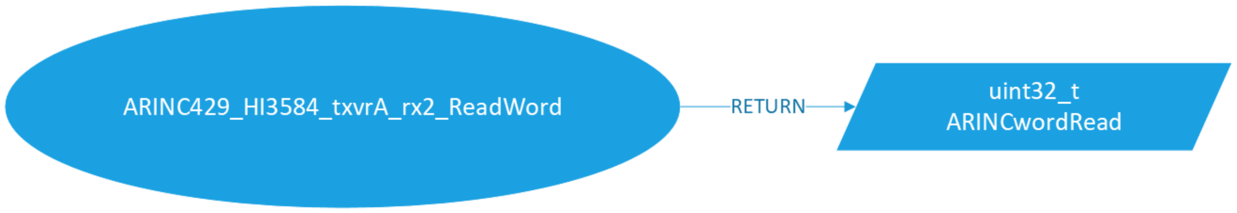
Description	The IOP ***shall*** transmit an ARINC429 word through ARINC429 transceiver A.
Function call:	ARINC429_HI3584_txvrA_TransmitWord
Input parameters:	uint32_t **ARINCword**
Global data:	None
Function static data:	None
Requirement	Call Config16bitDataBusDirection (**ARINC429_HI3584_DATA_BUS_DIR_OUTPUT**) Call WriteDataTo16bitDataBus(**ARINCword** BITWISE-AND 0xFFFF) Set **ARINC429_HI3584_TXVRA_PL1** equal to 0 Call Nop instruction Set **ARINC429_HI3584_TXVRA_PL1** equal to 1 Call WriteDataTo16bitDataBus ((**ARINCword** RIGHT-SHIFT by 16) BITWISE-AND 0xFFFF) Set **ARINC429_HI3584_TXVRA_PL2** equal to 0 Call Nop instruction Set **ARINC429_HI3584_TXVRA_PL2** equal to 1 Call Config16bitDataBusDirection (**ARINC429_HI3584_DATA_DIR_INPUT**) RETURN

INT1.0101.S.IOP.1.009



Description	The IOP ***shall*** read the ARINC429 word in ARINC429 transceiver A receiver 1 FIFO and return the value.
Function call:	ARINC429_HI3584_txvrA_rx1_ReadWord
Input parameters:	None
Global data:	None
Function static data:	None
Requirement	Call Config16bitDataBusDirection (**ARINC429_HI3584_DATA_DIR_INPUT**) Set **ARINC429_HI3584_TXVRA_EN1** equal to 1 Set **ARINC429_HI3584_TXVRA_EN2** equal to 1 Set **ARINC429_HI3584_TXVRA_SEL** equal to 0 Set **ARINC429_HI3584_TXVRA_EN1** equal to 0 Set **ARINCwordRead** equal to ReadDataFrom16bitDataBus () Set **ARINC429_HI3584_TXVRA_EN1** equal to 1 Set **ARINC429_HI3584_TXVRA_SEL** equal to 1 Set **ARINC429_HI3584_TXVRA_EN1** equal to 0 Set **ARINCwordRead** BITWISE-OR-equals (ReadDataFrom16bitDataBus ()) << 16), type-casted to uint32_t Set **ARINC429_HI3584_TXVRA_EN1** equal to 1

INT1.0101.S.IOP.1.010

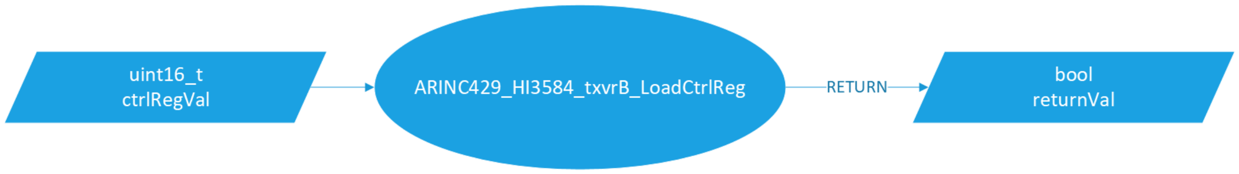


Description	The IOP ***shall*** read the ARINC429 word in ARINC429 transceiver A receiver 2 FIFO and return the value
Function call:	ARINC429_HI3584_txvrA_rx2_ReadWord
Input parameters:	None
Global data:	None
Function static data:	None
Requirement	Call Config16bitDataBusDirection (**ARINC429_HI3584_DATA_DIR_INPUT**) Set **ARINC429_HI3584_TXVRA_EN1** equal to 1 Set **ARINC429_HI3584_TXVRA_EN2** equal to 1 Set **ARINC429_HI3584_TXVRA_SEL** equal to 0 Set **ARINC429_HI3584_TXVRA_EN2** equal to 0 Set **ARINCwordRead** equal to ReadDataFrom16bitDataBus () Set **ARINC429_HI3584_TXVRA_EN2** equal to 1 Set **ARINC429_HI3584_TXVRA_SEL** equal to 1 Set **ARINC429_HI3584_TXVRA_EN2** equal to 0 Set **ARINCwordRead** BITWISE-OR-equals (ReadDataFrom16bitDataBus ()) << 16), type-casted to uint32_t Set **ARINC429_HI3584_TXVRA_EN2** equal to 1

INT1.0101.S.IOP.1.011

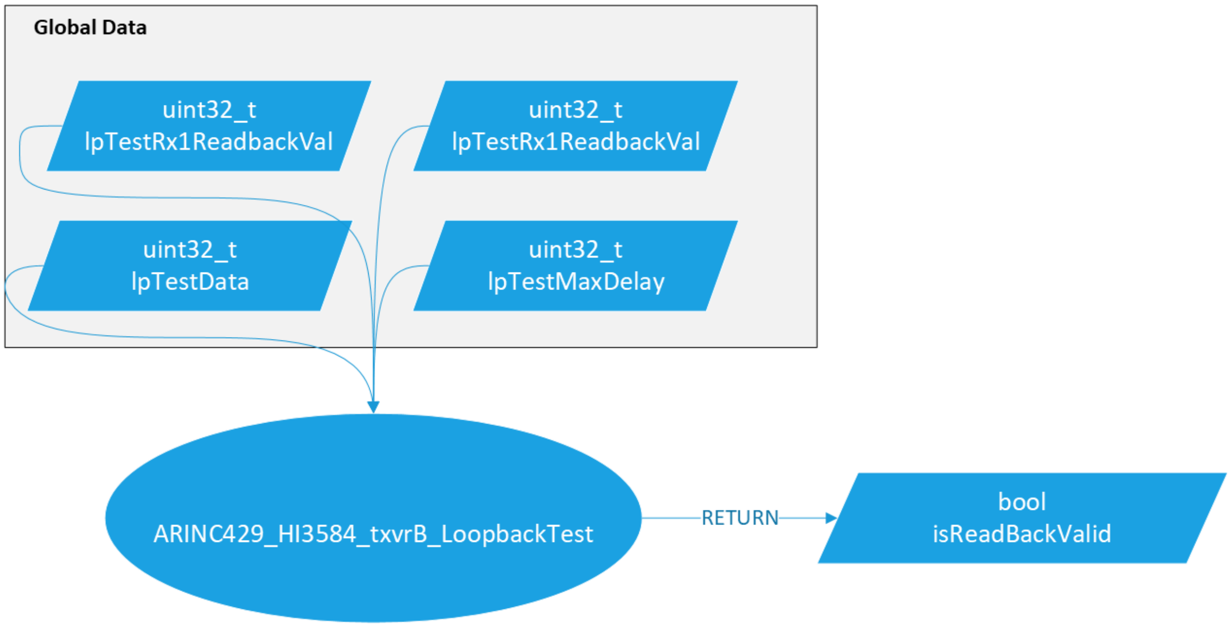
Description	The IOP ***shall*** initialize ARINC429 Transceiver B
Function call:	ARINC429_HI3584_txvrB_Initialize
Input parameters:	None
Global data:	None
Function static data:	None
Requirement	Set **ARINC429_HI3584_TXVRB_SEL** equal to 0 Set **ARINC429_HI3584_TXVRB_EN1_TRIS** equal to 0 Set **ARINC429_HI3584_TXVRB_EN1** equal to 1 Set **ARINC429_HI3584_TXVRB_EN2_TRIS** equal to 0 Set **ARINC429_HI3584_TXVRB_EN2** equal to 1 Set **ARINC429_HI3584_TXVRB_PL1_TRIS** equal to 0 Set **ARINC429_HI3584_TXVRB_PL1** equal to 1 Set **ARINC429_HI3584_TXVRB_PL2_TRIS** equal to 0 Set **ARINC429_HI3584_TXVRB_PL2** equal to 1 Set **ARINC429_HI3584_TXVRB_ENTX_TRIS** equal to 0 Set **ARINC429_HI3584_TXVRB_ENTX** equal to 1 Set **ARINC429_HI3584_TXVRB_CWSTR_TRIS** equal to 0 Set **ARINC429_HI3584_TXVRB_CWSTR** equal to 1 Set **ARINC429_HI3584_TXVRB_RSR_TRIS** equal to 0 Set **ARINC429_HI3584_TXVRB_RSR** equal to 1 Set **ARINC429_HI3584_TXVRB_DR1_TRIS** equal to 1 Set **ARINC429_HI3584_TXVRB_DR2_TRIS** equal to 1 Set **ARINC429_HI3584_TXVRB_FFT_TRIS** equal to 1 Call Config16bitDataBusDirection (**ARINC429_HI3584_DATA_DIR_INPUT**) RETURN

INT1.0101.S.IOP.1.012



Description	The IOP ***shall*** load the ARINC429 Transceiver B control register with the input value. The function returns true if the control register readback was successful, false if readback was unsuccessful.
Function call:	ARINC429_HI3584_txvrB_LoadCtrlReg
Input parameters:	uint16_t **ctrlRegVal**
Global data:	None
Function static data:	None
Requirement	Set **ARINC429_HI3584_TXVRB_SEL** equal to 0 Set **ARINC429_HI3584_TXVRB_CWSTR** equal to 0 Call Config16bitDataBusDirection(**ARINC429_HI3584_DATA_BUS_DIR_OUTPUT**) Call WriteDataTo16bitDataBus(**ctrlRegVal**) Set **ARINC429_HI3584_TXVRB_CWSTR** equal to 1 Set **readBack** equal to ARINC429_HI3584_txvrB_ReadBackControlRegister() IF (**ctrlRegVal** equals **readBack**) return TRUE ELSE Return FALSE

INT1.0101.S.IOP.1.013


Global Data
uint32_t lpTestRx1ReadbackVal
uint32_t lpTestRx1ReadbackVal
uint32_t lpTestData
uint32_t lpTestMaxDelay
ARINC429_HI3584_txvrB_LoopbackTest
RETURN
bool isReadBackValid

Description	The IOP ***shall*** conduct a hardware-level loopback test on ARINC429 transceiver B. The function returns true if the loopback test was successful, false if unsuccessful.
Function call:	ARINC429_HI3584_txvrB_LoopbackTest
Input parameters:	None
Global data:	
Function static data:	None
Requirement	Set currentCtrRegValue equal to ARINC429_HI3584_txvrB_ReadBackControlRegister( ) Set status equal to ARINC429_HI3584_txvrB_LoadCtrlReg(0x8000) Set currFIFOflushCount equal to 0 WHILE currFIFOflushCount is less than or equal to txvrRxFIFOsize Call ARINC429_HI3584_txvrB_rx1_ReadWord( ) Increment currFIFOflushCount by 1 Set currFIFOflushCount equal to 0 WHILE currFIFOflushCount is less than or equal to txvrRxFIFOsize ARINC429_HI3584_txvrB_rx2_ReadWord( ) Increment currFIFOflushCount by 1 Set counter equal to Set rx1readback equal to lpTestRx1ReadbackVal Set rx2readback equal to lpTestRx2ReadbackVal WHILE ((counter is less than lpTestNumCycles) AND (((lpTestRx1ReadbackVal equals rx1readback) AND (lpTestRx2ReadbackVal equals rx2readback)) OR (1 equals counter)))

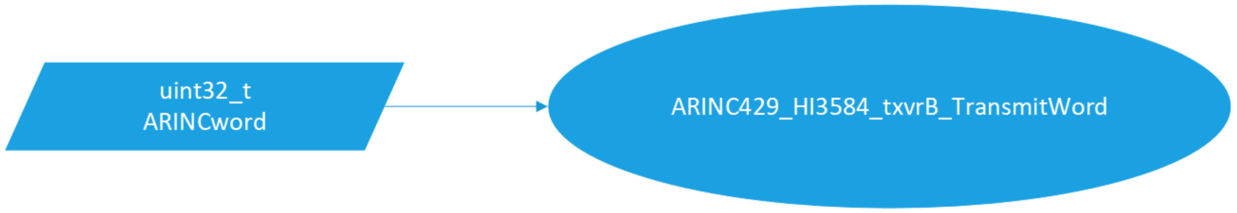
```
   Call ARINC429_HI3584_txvrB_TransmitWord( lpTestData )
   Set delayCounter equal to 0
   WHILE (((1 equals ARINC429_HI3584_TXVRB_DR1) OR (1 equals
ARINC429_HI3584_TXVRB_DR2)) AND (delayCounter is less than
lpTestMaxDelay))
       Increment delayCounter by 1
   Set rx1readback equal to ARINC429_HI3584_txvrB_rx1_ReadWord( )
   Set rx2readback equal to ARINC429_HI3584_txvrB_rx2_ReadWord( )
   Increment counter by 1
IF ((lpTestRx1ReadbackVal equals rx1readback) AND (lpTestRx2ReadbackVal
equals rx2readback))
   Set status BITWISE-AND equals true
ELSE
   Set status BITWISE-AND equals false
Call ARINC429_HI3584_txvrB_LoadCtrlReg( currentCtrRegValue )
RETURN status
```

INT1.0101.S.IOP.1.014

ARINC429_HI3584_txvrB_ReadBackControlRegister
RETURN
uint16_t
controlRegReadback

Description	The IOP ***shall*** read back the current control register value of ARINC429 transceiver B.
Function call:	ARINC429_HI3584_txvrB_ReadBackControlRegister
Input parameters:	None
Global data:	None
Function static data:	None
Requirement	Call Config16bitDataBusDirection(**ARINC429_HI3584_DATA_DIR_INPUT**) Set **ARINC429_HI3584_TXVRB_SEL** equal to 1 Set **ARINC429_HI3584_TXVRB_RSR** equal to 0 Set **controlRegReadback** equal to ReadDataFrom16bitDataBus( ) Set **ARINC429_HI3584_TXVRB_RSR** equal to 1 Set **ARINC429_HI3584_TXVRB_SEL** equal to 0 RETURN **controlRegReadback**

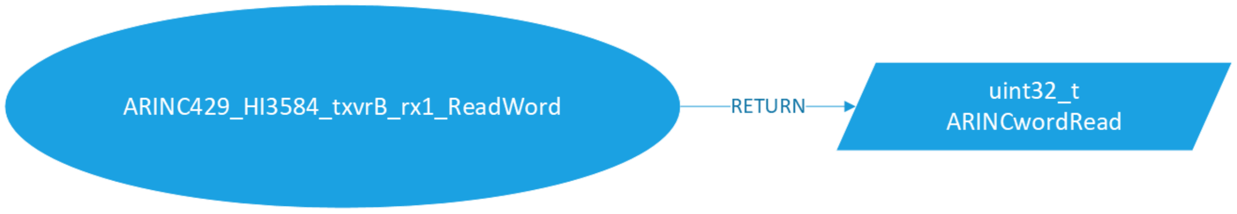
INT1.0101.S.IOP.1.015



Description	The IOP ***shall*** transmit an ARINC429 word through ARINC429 transceiver B
Function call:	ARINC429_HI3584_txvrB_TransmitWord
Input parameters:	uint32_t **ARINCword**
Global data:	
Function static data:	
Requirement	Call Config16bitDataBusDirection (ARINC429_HI3584_DATA_BUS_DIR_OUTPUT) Call WriteDataTo16bitDataBus(ARINCword BITWISE-AND 0xFFFF) Set ARINC429_HI3584_TXVRB_PL1 equal to 0 Call Nop instruction Set ARINC429_HI3584_TXVRB_PL1 equal to 1 Call WriteDataTo16bitDataBus ((ARINCword RIGHT-SHIFT by 16) BITWISE-AND 0xFFFF)

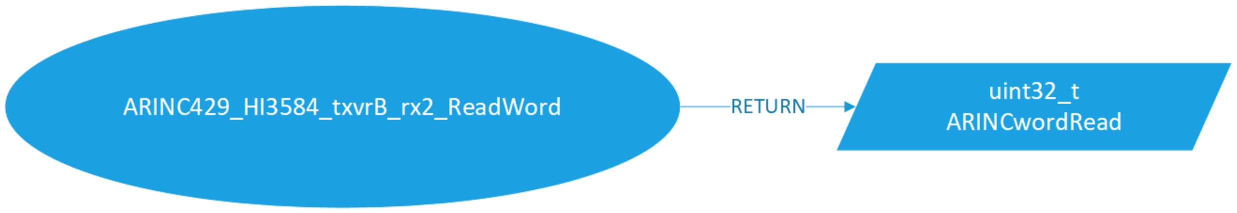
	Set ARINC429_HI3584_TXVRB_PL2 equal to 0 Call Nop instruction Set ARINC429_HI3584_TXVRB_PL2 equal to 1 Call Config16bitDataBusDirection (ARINC429_HI3584_DATA_DIR_INPUT) RETURN

INT1.0101.S.IOP.1.016



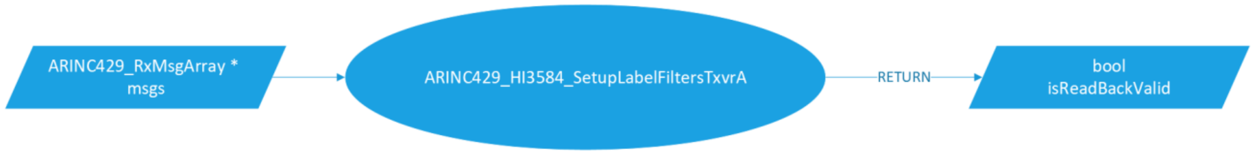
Description	The IOP ***shall*** read the ARINC429 word in ARINC429 transceiver B receiver 1 FIFO and return the value.
Function call:	ARINC429_HI3584_txvrB_rx1_ReadWord
Input parameters:	None
Global data:	None
Function static data:	None
Requirement	Call Config16bitDataBusDirection (ARINC429_HI3584_DATA_DIR_INPUT) Set ARINC429_HI3584_TXVRB_EN1 equal to 1 Set ARINC429_HI3584_TXVRB_EN2 equal to 1 Set ARINC429_HI3584_TXVRB_SEL equal to 0 Set ARINC429_HI3584_TXVRB_EN1 equal to 0 Set ARINCwordRead equal to ReadDataFrom16bitDataBus ( ) Set ARINC429_HI3584_TXVRB_EN1 equal to 1 Set ARINC429_HI3584_TXVRB_SEL equal to 1 Set ARINC429_HI3584_TXVRB_EN1 equal to 0 Set ARINCwordRead BITWISE-OR-equals (ReadDataFrom16bitDataBus ()) << 16), type-casted to uint32_t Set ARINC429_HI3584_TXVRB_EN1 equal to 1

INT1.0101.S.IOP.1.017



Description	The IOP ***shall*** read the ARINC429 word in ARINC429 transceiver B receiver 2 FIFO and return the value.
Function call:	ARINC429_HI3584_txvrB_rx2_ReadWord
Input parameters:	None
Global data:	None
Function static data:	None
Requirement	Call Config16bitDataBusDirection (ARINC429_HI3584_DATA_DIR_INPUT) Set ARINC429_HI3584_TXVRB_EN1 equal to 1 Set ARINC429_HI3584_TXVRB_EN2 equal to 1 Set ARINC429_HI3584_TXVRB_SEL equal to 0 Set ARINC429_HI3584_TXVRB_EN2 equal to 0 Set **ARINCwordRead** equal to ReadDataFrom16bitDataBus () Set ARINC429_HI3584_TXVRB_EN2 equal to 1 Set ARINC429_HI3584_TXVRB_SEL equal to 1 Set ARINC429_HI3584_TXVRB_EN2 equal to 0 Set ARINCwordRead BITWISE-OR-equals (ReadDataFrom16bitDataBus ()) << 16), type-casted to uint32_t Set ARINC429_HI3584_TXVRB_EN2 equal to 1

INT1.0101.S.IOP.1.018


ARINC429_RxMsgArray * msgs
ARINC429_HI3584_SetupLabelFiltersTxvrA
RETURN
bool isReadBackValid

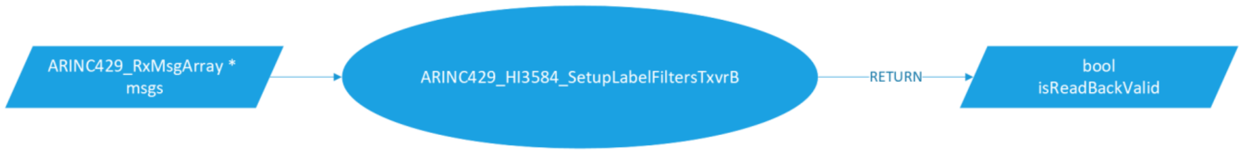
I

Description	The IOP ***shall*** setup ARINC429 receiver A with receive label filters from an input ARINC429 message array.
Function call:	ARINC429_HI3584_SetupLabelFiltersTxvrA
Input parameters:	ARINC429_RxMsgArray * msgs
Global data:	None
Function static data:	None

Requirement	IF NULL equals **msgs** OR **msgs.numMsgs** is greater than **MAX_NUM_REGOCNIZED_LABELS**
	    RETURN false
	Declare **rxLabelsTxrA**[MAX_NUM_REGOCNIZED_LABELS]
	Declare **counter**
	FOR counter equals zero, counter less than **msgs.numMsgs**, increment counter by 1
	   Set **rxLabelsTxrA**[**counter**] equal to **msgs.rxMsgs[counter].msgConfig.label**
	FOR counter less than MAX_NUM_REGOCNIZED_LABELS, increment counter by 1
	   Set **rxLabelsTxrA**[**counter**] equal to 0
	Set **currentControlReg** equal to ARINC429_HI3584_txvrA_ReadBackControlRegister()
	Set **maxNumRetries** equal to 3
	Set **retryCounter** equal to 0
	Declare **isReadBackValid**
	WHILE **retryCounter** is less than **maxNumRetries**
	   Set **isReadBackValid** equal to true
	   Set **ARINC429_HI3584_TXVRA_SEL** equal to 1
	   Call **ARINC429_HI3584_txvrA_LoadCtrlReg**(0x02)
	   Call Config16bitDataBusDirection(**ARINC429_HI3584_DATA_BUS_DIR_OUTPUT**)
	   FOR **counter** equals 0, **counter** less than **MAX_NUM_REGOCNIZED_LABELS**, increment **counter** by 1
	      Set **ARINC429_HI3584_TXVRA_PL2** equal to 0
	      Call **NOP** instruction 4 times
	      Call WriteDataTo16bitDataBus(**rxLabelsTxrA**[**counter**])
	      Set **ARINC429_HI3584_TXVRA_PL2** equal to 1
	      Call **NOP** instruction 5 times
	   Call Config16bitDataBusDirection(**ARINC429_HI3584_DATA_DIR_INPUT**)
	   Declare **readBackValue**
	   FOR **counter** equals 0, **counter** less than **MAX_NUM_REGOCNIZED_LABELS**, increment **counter** by 1

```
        Set ARINC429_HI3584_TXVRA_EN2 equal to 0
        Call NOP instruction 5 times
        Set readBackValue equal to ReadData From16bitDataBus()
        Set isReadBackValid to BITWISE-AND equals (readBackValue equals
rxLabelsTxrA[counter]);
        Set ARINC429_HI3584_TXVRA_EN2 equal to 1
        Call NOP instruction 4 times
    IF isReadBackValid
        BREAK
    Increment retryCounter by 1
IF false equals isReadBackValid
    Set currentControlReg equal to BITWISE-AND-EQUALS 0x0008
Call ARINC429_HI3584_txvrA_LoadCtrlReg(currentControlReg)
RETURN isReadBackValid
```

INT1.0101.S.IOP.1.019


ARINC429_RxMsgArray * msgs
ARINC429_HI3584_SetupLabelFiltersTxvrB
RETURN
bool isReadBackValid

Description	The IOP ***shall*** setup ARINC429 receiver B with receive label filters from an input ARINC429 message array.
Function call:	A RINC429_HI3584_SetupLabelFiltersTxvrB
Input parameters:	ARINC429_RxMsgArray * msgs
Global data:	None
Function static data:	None

Requirement	IF NULL equals **msgs** OR **msgs.numMsgs** is greater than **MAX_NUM_REGOCNIZED_LABELS**
	    RETURN false
	Declare **rxLabelsTxrB**[MAX_NUM_REGOCNIZED_LABELS]
	Declare **counter**
	FOR counter equals zero, counter less than **msgs.numMsgs**, increment counter by 1
	   Set **rxLabelsTxrB**[**counter**] equal to **msgs.rxMsgs[counter].msgConfig.label**
	FOR counter less than MAX_NUM_REGOCNIZED_LABELS, increment counter by 1
	   Set **rxLabelsTxrB** [**counter**] equal to 0
	Set **currentControlReg** equal to ARINC429_HI3584_txvrB_ReadBackControlRegister()
	Set **maxNumRetries** equal to 3
	Set **retryCounter** equal to 0
	Declare **isReadBackValid**
	WHILE **retryCounter** is less than **maxNumRetries**
	   Set **isReadBackValid** equal to true
	   Set **ARINC429_HI3584_TXVRB_SEL** equal to 1
	   Call **ARINC429_HI3584_txvrB_LoadCtrlReg**(0x02)
	   Call Config16bitDataBusDirection(**ARINC429_HI3584_DATA_BUS_DIR_OUTPUT**)
	   FOR **counter** equals 0, **counter** less than **MAX_NUM_REGOCNIZED_LABELS**, increment **counter** by 1
	      Set **ARINC429_HI3584_TXVRB_PL2** equal to 0
	      Call **NOP** instruction 4 times
	      Call WriteDataTo16bitDataBus(**rxLabelsTxrB** [**counter**])
	      Set **ARINC429_HI3584_TXVRB_PL2** equal to 1
	      Call **NOP** instruction 5 times
	   Call Config16bitDataBusDirection(**ARINC429_HI3584_DATA_DIR_INPUT**)

```
   Declare readBackValue
   FOR counter equals 0, counter less than MAX_NUM_REGOCNIZED_LABELS,
increment counter by 1
      Set ARINC429_HI3584_TXVRB_EN2 equal to 0
      Call NOP instruction 5 times
      Set readBackValue equal to ReadDataFrom16bitDataBus()
      Set isReadBackValid to BITWISE-AND equals (readBackValue equals
rxLabelsTxrB[counter]);
      Set ARINC429_HI3584_TXVRB_EN2 equal to 1
      Call NOP instruction 4 times
   IF isReadBackValid
      BREAK
   Increment retryCounter by 1
IF false equals isReadBackValid
   Set currentControlReg equal to BITWISE-AND-EQUALS 0x0008
Call ARINC429_HI3584_txvrB_LoadCtrlReg(currentControlReg)
RETURN isReadBackValid
```

INT1.0101.S.IOP.2

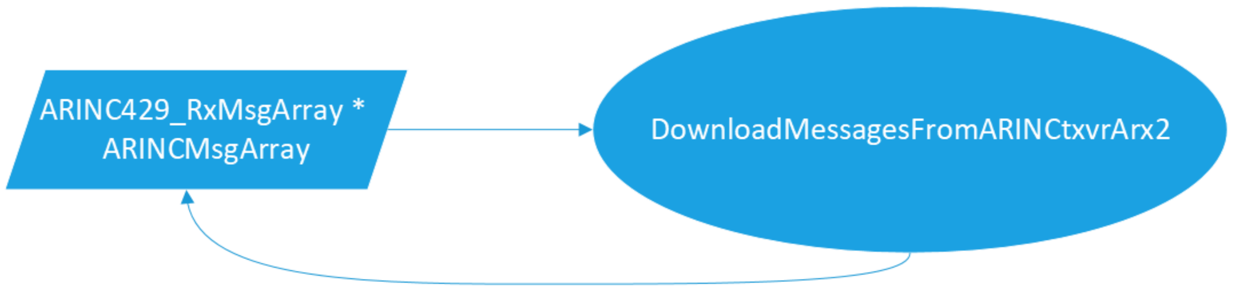
High level requirement description: The IOP shall use the following trigonometric functions to process data:
1. Sin
2. Cos
3. ArcTan
Requirement reuse: INT1.0102.S.IMU.4.015

INT1.0101.S.IOP.2.001
Description: The IOP shall use the following trigonometric functions to process data
1. Sin
2. Cos
3. ArcTan
Requirement reuse: INT1.0102.S.IMU.4.015

INT1.0101.S.IOP.3

High level requirement description: The IOP ***shall*** download received ARINC429 messages from the receive ARINC429 FIFOs

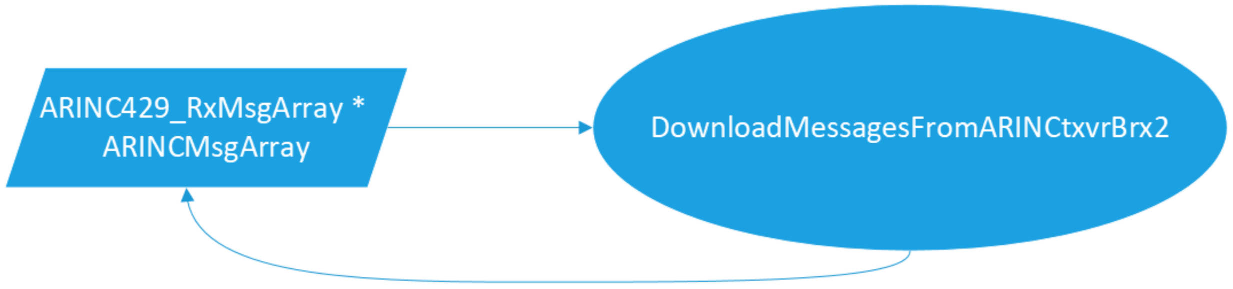
INT1.0101.S.IOP.3.001


ARINC429_RxMsgArray * ARINCMsgArray
DownloadMessagesFromARINCtxvrArx2

Description	The IOP ***shall*** download ARINC429 messages from transceiver A receiver 2
Function call:	DownloadMessagesFromARINCtxvrArx2
Input parameters:	ARINC429_RxMsgArray * **ARINCMsgArray**
Global data:	None
Function static data:	None

Requirement	IF NULL equals **ARINCMsgArray**
	    RETURN
	Set **numWordsProcessed** to 0
	Declare **thisARINCRxMsg**
	WHILE ((**ARINC429_HI3584_TXVRA_DR2** equals 0) AND (**numWordsProcessed** is less than **MAX_NUM_RX_MSGS**))
	    thisARINCRxMsg equals ARINC429_HI3584_txvrA_rx2_ReadWord()
	    IF (**thisARINCRxMsg** BITWISE-AND 0x80000000)
	        ;
	    ELSE IF (ARINC429_READ_MSG_SUCCESS equals ARINC429_ProcessReceivedMessage(**ARINCMsgArray**, **thisARINCRxMsg**))
	        Set **ARINCMsgArray.currentCounts** equal to 0
	    ELSE
	        ;
	    Increment **numWordsProcessed** by 1
	RETURN

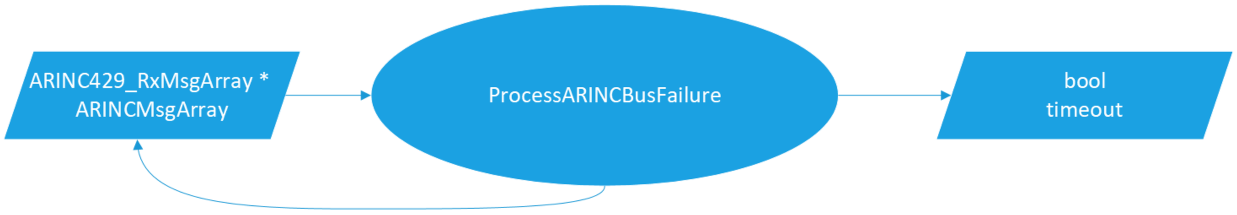
INT1.0101.S.IOP.3.002


ARINC429_RxMsgArray * ARINCMsgArray
DownloadMessagesFromARINCtxvrBrx2

Description	The IOP ***shall*** download ARINC429 messages from transceiver B receiver 2
Function call:	DownloadMessagesFromARINCtxvrBrx2
Input parameters:	ARINC429_RxMsgArray * **ARINCMsgArray**
Global data:	None
Function static data:	None

Requirement	IF NULL equals **ARINCMsgArray**
	    RETURN
	Set **numWordsProcessed** to 0
	Declare **thisARINCRxMsg**
	WHILE ((**ARINC429_HI3584_TXVRB_DR2** equals 0) AND (**numWordsProcessed** is less than MAX_NUM_RX_MSGS))
	    thisARINCRxMsg equals ARINC429_HI3584_txvrB_rx2_ReadWord()
	    IF (**thisARINCRxMsg** BITWISE-AND 0x80000000)
	        ;
	    ELSE IF (ARINC429_READ_MSG_SUCCESS equals ARINC429_ProcessReceivedMessage(**ARINCMsgArray**, **thisARINCRxMsg**))
	        Set **ARINCMsgArray**.**currentCounts** equal to 0
	    ELSE
	        ;
	    Increment **numWordsProcessed** by 1
	RETURN

INT1.0101.S.IOP.3.003

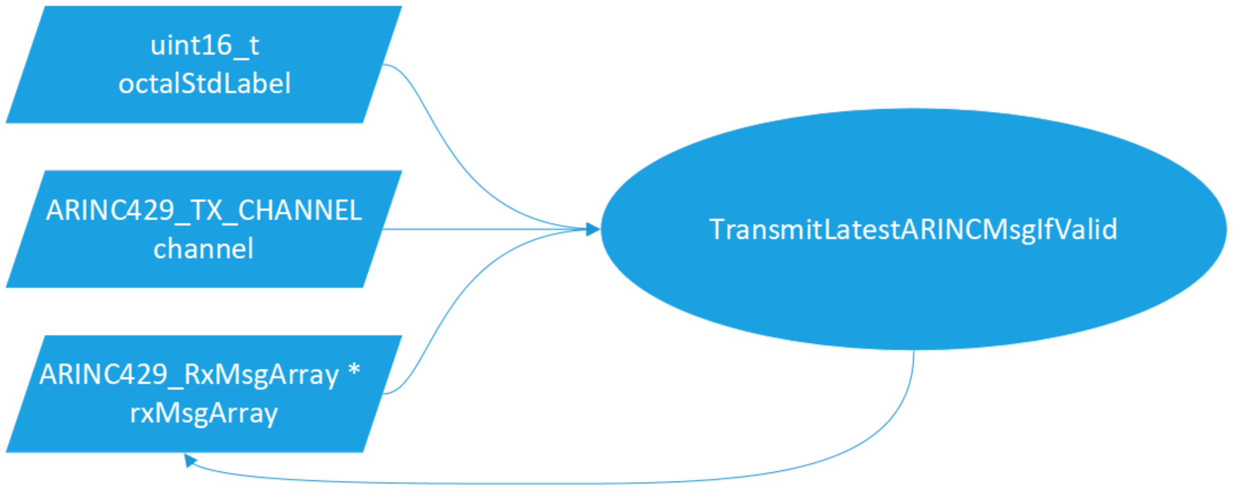


Description	The ***IOP*** shall check the status of a specified ARINC receive bus and decide if the bus has timed out
Function call:	ProcessARINCBusFailure
Input parameters:	ARINC429_RxMsgArray * **ARINCMsgArray**
Global data:	None
Function static data:	None

Requirement	Increment **ARINCMsgArray.currentCounts** by 1
	IF **ARINCMsgArray.currentCounts** is greater than or equal to **ARINCMsgArray.maxBusFailureCounts**
	    RETURN true
	ELSE
	    RETURN false

INT1.0101.S.IOP.3.004

Description: The IOP ***shall*** transmit the latest desired ARINC429 word if the message was valid
Function call: TransmitLatestARINCMsgIfValid
Input parameters: ARINC429_RxMsgArray * **rxMsgArray**, uint16_t **octalStdLabel**, ARINC429_TX_CHANNEL **channel**
Global data: MAX_OCTAL_LABEL_VALUE


uint16_t octalStdLabel
ARINC429_TX_CHANNEL channel
ARINC429_RxMsgArray * rxMsgArray
TransmitLatestARINCMsgIfValid

Description	The IOP ***shall*** transmit the latest desired ARINC429 word if the message was valid
Function call:	TransmitLatestARINCMsgIfValid
Input parameters:	ARINC429_RxMsgArray * **rxMsgArray**, uint16_t **octalStdLabel**, ARINC429_TX_CHANNEL **channel**
Global data:	MAX_OCTAL_LABEL_VALUE
Function static data:	None

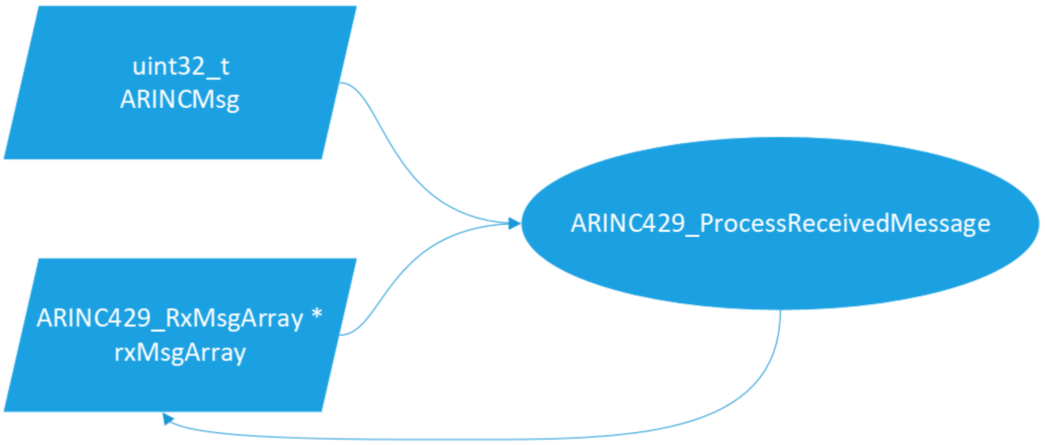
Requirement	IF NULL equals **rxMsgArray** OR **octalStdLabel** is greater than **MAX_OCTAL_LABEL_VALUE**
	    RETURN
	Declare **data**
	Set **hexFlippedLabel** to FormatLabelNumber(**octalStdLabel**)
	Set **readStatus** to ARINC429_GetLatestLabelData(**rxMsgArray**, **hexFlippedLabel**, &**data**)
	IF true equals **data.isDataFresh** AND true equals **data.isNotBabbling** AND ARINC429_GET_LABEL_DATA_MSG_SUCCESS equals **readStatus**
	    SWITCH(**channel**)
	        case A429_CHANNEL_A:
	            Call ARINC429_HI3584_txvrA_TransmitWord(**data.rawARINCword**)
	            BREAK
	        case A429_CHANNEL_B:
	            Call ARINC429_HI3584_txvrB_TransmitWord(**data.rawARINCword**)
	            BREAK
	        default:

	BREAK RETURN

INT1.0101.S.IOP.4

High level requirement description: The IOP ***shall*** process ARINC-429 Messages (process is defined as extracting label, engineering data, sig bits).

INT1.0101.S.IOP.4.001

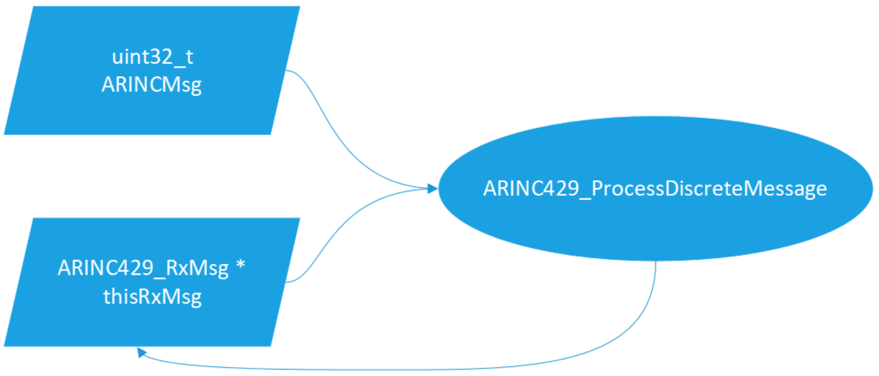


Description	The IOP ***shall*** process a received ARINC429 message, based on message type (BNR, BCD, DISC) and store the data in the proper receive message array.
Function call:	ARINC429_ProcessReceivedMessage
Input parameters:	ARINC429_RxMsgArray * **rxMsgArray**, uint32_t **ARINCMsg**
Global data:	None
Function static data:	None

Requirement	
	```
	IF (NULL equals rxMsgArray OR NULL equals rxMsgArray.rxMsgs
	    RETURN ARINC429_READ_MSG_ERROR
	Set msgLabel to (ARINCMsg BITWISE-AND ARINC429_LBL_MASK)
	Set readMsgReturnStatus to ARINC429_READ_MSG_SUCCESS
	Set count to 0
	Set labelmatchFound equal to false
	WHILE ( count is less than rxMsgArray.numMsgs AND count is less than maxNumRxMsgsInArray AND false equals labelMatchFound )
	    IF rxMsgArray.rxMsgs[count].msgConfig.label equals msgLabel
	        Set thisRxMsg equal to the address of (rxMsgArray.rxMsgs[count])
	        SWITCH (thisRxMsg.msgConfig.msgType)
	            case ARINC429_STD_BNR_MSG
	                Set readMsgReturnStatus equal to ARINC429_ProcessStdBNRmessage (thisRxMsg, ARINCMsg)
	                BREAK
	            case ARINC429_STD_BCD_MSG
	                Set thisRxMsg.data.rawARINCword equal to ARINCMsg
	                Set readMsgReturnStatus equal to ARINC429_ProcessStdBCDmessage(thisRxMsg, ARINCMsg)
	```

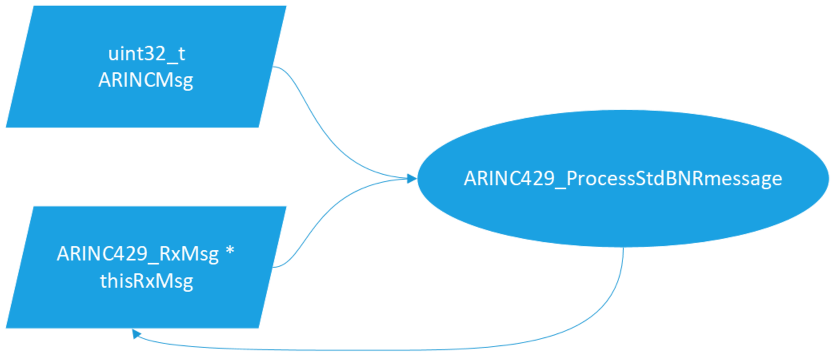
```
                BREAK
            case ARINC429_DISCRETE_MSG
                Set thisRxMsg.data.rawARINCword equal to ARINCMsg
                Set readMsgReturnStatus equal to
ARINC429_ProcessDiscreteMessage( thisRxMsg, ARINCMsg )
                BREAK
            default:
                Set readMsgReturnStatus to ARINC429_READ_MSG_ERROR
                BREAK
        IF ARINC429_READ_MSG_SUCCESS equals readMsgReturnStatus
            Set timestamp_now_ms equal to Timer23_GetTimestamp_ms()
            Set thisRxMsg.data.isNotBabbling equal to
ARINC429_IsLabelDataNotBabbling( timestamp_now_ms, thisRxMsg)
            Set thisRxMsg.data.sysTimeLastGoodMsg_ms equal to
timestamp_now_ms
        Set labelMatchFound equal to true
    ELSE
        Increment count by 1
IF false equals labelMatchFound
    Set readMsgReturnStatus equal to
ARINC429_READ_MSG_ERROR_NO_MATCHING_LABEL
RETURN readMsgReturnStatus
```

INT1.0101.S.IOP.4.002



Description	The IOP ***shall*** process discrete ARINC429 messages.
Function call:	ARINC429_ProcessDiscreteMessage
Input parameters:	ARINC429_RxMsg * **thisRxMsg**, uint32_t **arincMsg**
Global data:	None
Function static data:	None
Requirement	IF(NULL == thisRxMsg) RETURN ARINC429_READ_MSG_ERROR_INVALID_ARGUMENT IF thisRxMsg.msgConfig.numDiscreteBits is less than 1 OR thisRxMsg.msgConfig.numDiscreteBits is greater than ARINC429_DISCRETE_MSG_MAX_NUM_BITS RETURN ARINC429_WRITE_MSG_ERROR_INVALID_MSG_CONFIG Set thisRxMsg.data.engDataFloat to 0.0f Set thisRxMsg.data.engDataInt to 0 Set thisRxMsg.data.isEngDataInBounds to false Set discreteBits to arincMsg RIGHT-SHIFTED by 10 Set discreteBits BITWISE-AND equals (UINT32_MAX RIGHT-SHIFTED by (NUM_BITS_IN_UINT32 - thisRxMsg.msgConfig.numDiscreteBits)) Set thisRxMsg.data.discreteBits to discreteBits Set thisRxMsg.data.SM to ARINC429_ExtractSSMbits(arincMsg) Set thisRxMsg.data.SDI to ARINC429_ExtractSDIbits(arincMsg) RETURN ARINC429_READ_MSG_SUCCESS

INT1.0101.S.IOP.4.003

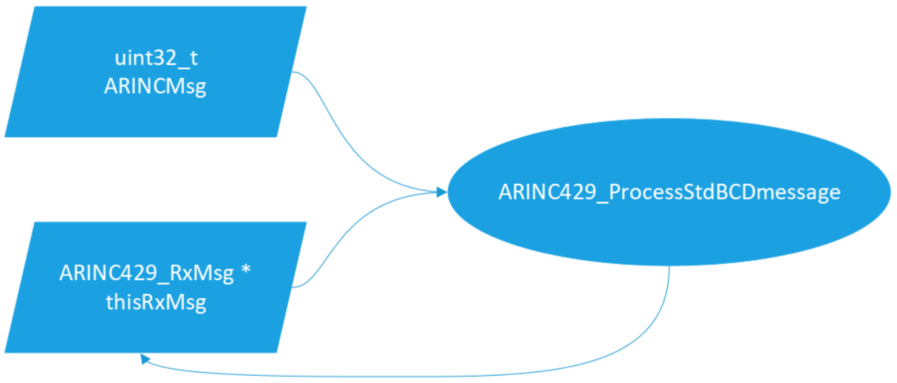


Description	The IOP ***shall*** process binary ARINC429 messages.
Function call:	ARINC429_ProcessStdBNRmessage
Input parameters:	ARINC429_RxMsg * **thisRxMsg**, uint32_t **arincMsg**
Global data:	None
Function static data:	None

Requirement	
	```
	IF ( NULL equals thisRxMsg)
	    RETURN ARINC429_READ_MSG_ERROR_INVALID_ARGUMENT

	Set thisRxMsg.data.rawARINCword equal to ARINCMsg
	Set rawDataField equal to ARINCMsg RIGHT-SHIFTED by
	(ARINC429_BNR_MAX_DATA_FIELD_SHIFT - thisRxMsg.msgConfig.numSigBits)
	Set rawDataField BITWISE-AND equals (UINT32_MAX >>
	(NUM_BITS_IN_UINT32 - thisRxMsg.msgConfig.numSigBits - 1))
	Declare readStatus
	Declare dataEng
	IF EXIT_FAILURE equals ARINC429_BNR_ConvertRawMsgDataToEngUnits(
	thisRxMsg.msgConfig.numSigBits,
	                                     thisRxMsg.msgConfig.resolution,
	                                     &dataEng,
	                                     rawDataField )
	    Set readStatus equal to ARINC429_READ_MSG_ERROR
	ELSE
	    Set thisRxMsg.data.engDataFloat equal to dataEng
	    Declare calcValue
	    IF ( dataEng is less than 0.0f)
	        Set calcValue to dataEng - 0.5
	    ELSE
	        Set calcValue to dataEng + 0.5
	    Set calcValue equal to clamp(calcValue, INT32_MIN, INT32_MAX)
	    Set thisRxMsg.data.engDataInt equal to calcValue (type-casted to int32)
	    IF (thisRxMsg.msgConfig.numDiscreteBits is greater than 0)
	        Set discreteBits equal to ARINCMsg RIGHT-SHIFTED by
	ARINC429_BNR_BCD_MSG_DISCRETE_BITS_SHIFT_VAL
	```

	```
	        Set discreteBits to BITWISE-AND equals (UINT32_MAX >>
	(NUM_BITS_IN_UINT32 - thisRxMsg.msgConfig.numDiscreteBits))
	        Set thisRxMsg.data.discreteBits equal to discreteBits
	    ELSE
	        Set thisRxMsg.data.discreteBits equal to 0
	    Set thisRxMsg.data.SM equal to ARINC429_ExtractSSMbits( ARINCMsg )
	    IF thisRxMsg.msgConfig.numSigBits is less than or equal to
	ARINC429_BNR_STD_MSG_NUM_SIGBITS_18
	        Set thisRxMsg.data.SDI equal to ARINC429_ExtractSDIbits(ARINCMsg)
	    ELSE
	        Set thisRxMsg.data.SDI equal to 0
	    Set readStatus equal to ARINC429_READ_MSG_SUCCESS
	RETURN readStatus
	```

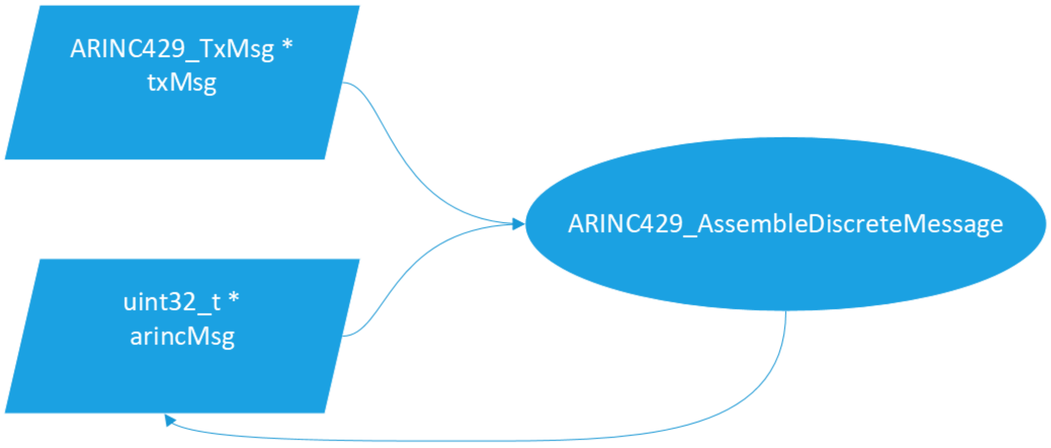
INT1.0101.S.IOP.4.004



Description	The IOP ***shall*** process BCD ARINC429 messages.
Function call:	ARINC429_ProcessStdBCDmessage
Input parameters:	ARINC429_RxMsg * **thisRxMsg**, uint32_t **arincMsg**
Global data:	None
Function static data:	None
Requirement	IF (NULL equals thisRxMsg) RETURN ARINC429_READ_MSG_ERROR_INVALID_ARGUMENT IF (thisRxMsg.msgConfig.numSigDigits is less than 1 OR thisRxMsg.msgConfig.numSigDigits is greater than 5 OR (thisRxMsg.msgConfig.numSigDigits * 4 - 1 + thisRxMsg.msgConfig.numDiscreteBits) is greater than 19 RETURN ARINC429_WRITE_MSG_ERROR_INVALID_MSG_CONFIG Set bcdData to arincMsg BITWISE-AND ARINC429_BCD_DATAFIELDMASK Set bcdData equal to RIGHT-SHIFT-EQUALS (ARINC429_BCD_STD_MSG_DATA_FIELD_SHIFT + ARINC429_BCD_BITS_PER_DIGIT * (ARINC429_BCD_STD_MSG_MAX_NUM_SIGDIGITS - thisRxMsg- >msgConfig.numSigDigits)) Declare dataEng IF (EXIT_FAILURE equals ARINC429_BCD_ConvertBCDvalToEngVal(thisRxMsg.msgConfig.numSigDigits, thisRxMsg.msgConfig.resolution, &dataEng, bcdData) RETURN ARINC429_READ_MSG_ERROR_INVALID_MESSAGE Set thisRxMsg.data.engDataFloat equal to dataEng Declare calcValue IF (dataEng is less than 0.0f) Set calcValue equal to dataEng - 0.5f ELSE Set calcValue equal to dataEng + 0.5f Set calcValue equal to clamp(calcValue, INT32_MIN, INT32_MAX) Set thisRxMsg.data.engDataInt equal to calcValue, type-casted to int32_t

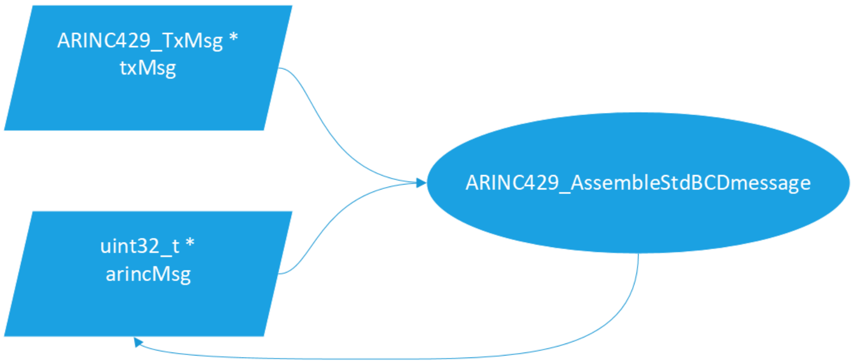
```
IF ( thisRxMsg.msgConfig.numDiscreteBits is greater than 0)
    Set discreteBits equal to arincMsg RIGHT-SHIFTED by ARINC429_BNR_BCD_MSG_DISCRETE_BITS_SHIFT_VAL
    Set discreteBits equal to BITWISE-AND equals (UINT32_MAX RIGHT-SHIFTED by (NUM_BITS_IN_UINT32 - thisRxMsg.msgConfig.numDiscreteBits)
    Set thisRxMsg.data.discreteBits equal to discreteBits
ELSE
    Set thisRxMsg.data.discreteBits equal to 0
Set thisRxMsg.data.SM equal to ARINC429_ExtractSSMbits(arincMsg)
Set thisRxMsg.data.SDI equal ARINC429_ExtractSDIbits(arincMsg)
RETURN ARINC429_READ_MSG_SUCCESS
```

INT1.0101.S.IOP.4.005



Description	The IOP ***shall*** assemble ARINC429 discrete messages.
Function call:	ARINC429_AssembleDiscreteMessage
Input parameters:	ARINC429_TxMsg * **txMsg**, uint32_t * **arincMsg**
Global data:	None
Function static data:	None
Requirement	IF (NULL equals txMsg OR NULL equals arincMsg OR NUL equals txMsg.msgConfig) RETURN ARINC429_WRITE_MSG_ERROR_INVALID_ARGUMENT IF ((txMsg.msgConfig.numDiscreteBits is less than 1) OR (txMsg.msgConfig.numDiscreteBits is greater than ARINC429_DISCRETE_MSG_MAX_NUM_BITS)) RETURN ARINC429_WRITE_MSG_ERROR_INVALID_MSG_CONFIG Declare discreteData_shifted Set discreteData equal to txMsg.discreteBits Set discreteData BITWISE-AND equals (UINT32_MAX RIGHT-SHIFTED by (NUM_BITS_IN_UINT32 - txMsg.msgConfig.numDiscreteBits)) Set discreteData_shifted equal to discreteData LEFT-SHIFTED by (ARINC429_DISCRETE_MSG_MAX_DATA_FIELD_SHIFT - txMsg.msgConfig.numDiscreteBits + 1) Set arincMsgTemp equal to txMsg.msgConfig.label Set arincMsgTemp BITWISE-OR-equals discreteData_shifted Set arincMsgTemp BITWISE-OR-equals (txMsg.SDI BITWISE-AND ARINC429_SDI_FIELD_LIMIT_MASK) LEFT-SHIFTED by ARINC429_SDI_FIELD_SHIFT_VAL Set arincMsgTemp BITWISE-OR-equals (txMsg.SM BITWISE-AND ARINC429_SSM_FIELD_LIMIT_MASK) LEFT-SHIFTED by ARINC429_SSM_FIELD_SHIFT_VAL), type-casted to uint32_t Set the dereferenced input pointer arincMsg equal to arincMsgTemp RETURN ARINC429_WRITE_MSG_SUCCESS

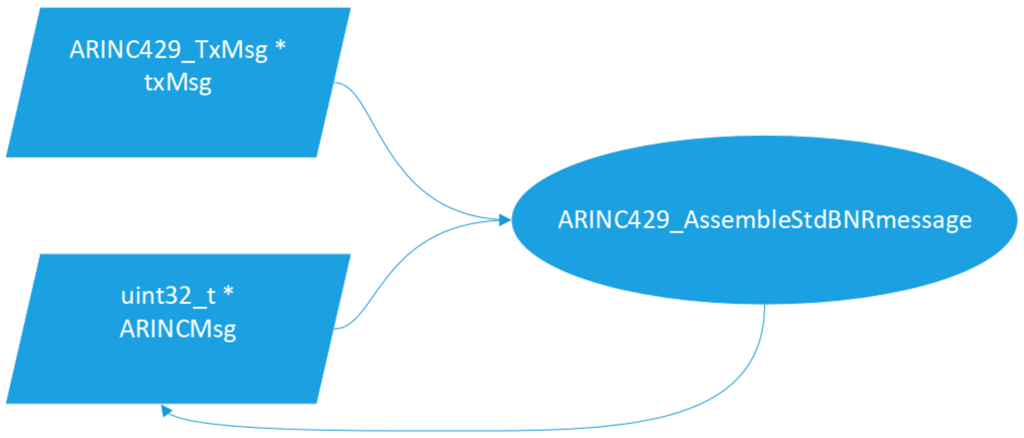
INT1.0101.S.IOP.4.006



Description	The IOP ***shall*** assemble binary coded decimal ARINC429 messages.
Function call:	ARINC429_AssembleStdBCDmessage
Input parameters:	ARINC429_TxMsg * **txMsg**, uint32_t * **arincMsg**
Global data:	None
Function static data:	None
Requirement	IF NULL equals txMsg OR NULL equals txMsg.msgConfig OR NULL equals arincMsg RETURN ARINC429_WRITE_MSG_ERROR_INVALID_ARGUMENT IF txMsg.msgConfig.numSigDigits is less than 1 OR txMsg.msgConfig.numSigDigits is greater than ARINC429_BCD_STD_MSG_MAX_NUM_SIGDIGITS OR (txMsg.msgConfig.numSigDigits * 4 - 1) + txMsg.msgConfig.numDiscreteBits is greater than ARINC429_BCD_STD_DATA_MAX_DATA_FIELD_SIZE RETURN ARINC429_WRITE_MSG_ERROR_INVALID_MSG_CONFIG IF txMsg.engData is less than 0 RETURN ARINC429_WRITE_MSG_ERROR_INVALID_MSG_DATA Set dataField equal to 0 Set isDataClipped equal to false IF EXIT_FAILURE equals ARINC429_BCD_ConvertEngValToBCD(txMsg.msgConfig.numSigDigits, txMsg.msgConfig.resolution, ARINC429_BCD_STD_MSG_MAX_NUM_BITS_MSC, txMsg.engData, &dataField, &isDataClipped) RETURN ARINC429_WRITE_MSG_ERROR Declare writeMsgReturnStatus IF true equals isDataClipped Set writeMsgReturnStatus equal to ARINC429_WRITE_MSG_SENT_DATA_CLIPPED

```
ELSE
    Set writeMsgReturnStatus equal to ARINC429_WRITE_MSG_SUCCESS
Set dataField_shifted equal to dataField LEFT-SHIFTED by
(ARINC429_BCD_STD_MSG_DATA_FIELD_SHIFT +
        ARINC429_BCD_BITS_PER_DIGIT *
(ARINC429_BCD_STD_MSG_MAX_NUM_SIGDIGITS -
txMsg.msgConfig.numSigDigits))
Set dataField_shifted BITWISE-AND-equals ARINC429_BCD_DATAFIELDMASK
Declare discreteBits_shifted
IF txMsg.msgConfig.numDiscreteBits is greater than 0
    Set discreteBits equal to txMsg.discreteBits
    Set discreteBits BITWISE-AND-equals (UINT32_MAX RIGHT-SHIFTED by
(NUM_BITS_IN_UINT32 - txMsg.msgConfig.numDiscreteBits))
    Set discreteBits_shifted equal to (discreteBits LEFT-SHIFTED by
ARINC429_BNR_BCD_MSG_DISCRETE_BITS_SHIFT_VAL)
ELSE
    Set discreteBits_shifted equal to 0
Set arincMsgTemp equal to txMsg.msgConfig.label
Set arincMsgTemp equal to BITWISE-OR-equals dataField_shifted
Set arincMsgTemp equal to BITWISE-OR-equals discreteBits_shifted
Set arincMsgTemp equal to BITWISE-OR-equals(txMsg.SDI BITWISE-AND
ARINC429_SDI_FIELD_LIMIT_MASK) LEFT-SHIFTED by
ARINC429_SDI_FIELD_SHIFT_VAL
Set arincMsgTemp equal to BITWISE-OR-equals ((txMsg.SM BITWISE-AND
ARINC429_SSM_FIELD_LIMIT_MASK) LEFT-SHIFTED by
ARINC429_SSM_FIELD_SHIFT_VAL)
Set the dereferenced input point arincMsg equal to arincMsgTemp
RETURN writeMsgReturnStatus
```

INT1.0101.S.IOP.4.007

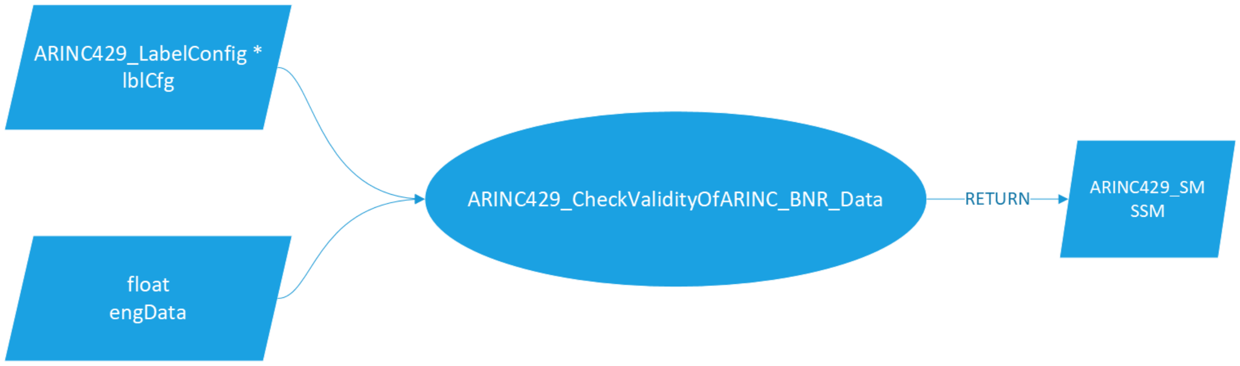


Description	The IOP ***shall*** assemble binary ARINC429 messages.
Function call:	ARINC429_AssembleStdBNRmessage
Input parameters:	ARINC429_TxMsg * **txMsg**, uint32_t * **arincMsg**
Global data:	None
Function static data:	None

Requirement	
	```
	IF NULL equals txMsg.msgConfig OR NULL equals ARINCMsg
	    RETURN ARINC429_WRITE_MSG_ERROR_INVALID_ARGUMENT
	Set dataField equal to 0
	Set isDataClipped to false
	Declare writeMsgReturnStatus
	IF EXIT_FAILURE equals ARINC429_BNR_ConvertEngValToRawBNRmsgData(
	txMsg.msgConfig.numSigBits,
	                                        txMsg.msgConfig.resolution,
	                                        txMsg.engData,
	                                        &dataField,
	                                        &isDataClipped)
	    Set writeMsgReturnStatus equal to ARINC429_WRITE_MSG_ERROR
	ELSE
	    IF (true equals isDataClipped)
	        Set writeMsgReturnStatus equal to
	ARINC429_WRITE_MSG_SENT_DATA_CLIPPED
	    ELSE
	        Set writeMsgReturnStatus equal to ARINC429_WRITE_MSG_SUCCESS
	    Set dataField_Shifted equal to dataField LEFT-SHIFTED by
	(ARINC429_BNR_MAX_DATA_FIELD_SHIFT - txMsg.msgConfig.numSigBits)
	    IF (ARINC429_BNR_STD_MSG_NUM_SIGBITS_20 equals
	txMsg.msgConfig.numSigBits)
	          Set dataField_Shifted to BITWISE-AND-equals
	ARINC429_BNR_STD_MSG_DATAFIELDMASK_20SIGBITS
	```

```
    ELSE IF (ARINC429_BNR_STD_MSG_NUM_SIGBITS_19 equals txMsg.msgConfig.numSigBits)
        Set dataField_Shifted to BITWISE-AND-equals ARINC429_BNR_STD_MSG_DATAFIELDMASK_19SIGBITS
    ELSE
        Set dataField_Shifted to BITWISE-AND-equals ARINC429_BNR_STD_MSG_DATAFIELDMASK_UPTO18SIGBITS
    Declare discreteBits_shifted
    IF txMsg.msgConfig.numDiscreteBits is greater than 0
        Set discreteBits to txMsg.discreteBits
        Set discreteBits BITWISE-AND-equals (UINT32_MAX >> (NUM_BITS_IN_UINT32 - txMsg.msgConfig.numDiscreteBits))
        Set discreteBits_shifted equal to discreteBits LEFT-SHIFTED by ARINC429_BNR_BCD_MSG_DISCRETE_BITS_SHIFT_VAL
    ELSE
        Set discreteBits_shifted equal to 0
    Set arincMsgTemp equal to txMsg.msgConfig.label
    Set arincMsgTemp BITWISE-OR-equals dataField_Shifted
    Set arincMsgTemp BITWISE-OR-equals discreteBits_shifted
    IF (txMsg.msgConfig.numSigBits is less than or equal to ARINC429_BNR_STD_MSG_NUM_SIGBITS_18)
        Set arincMsgTemp to BITWISE-OR-equals (txMsg.SDI BITWISE-AND ARINC429_SDI_FIELD_LIMIT_MASK) LEFT-SHIFTED by ARINC429_SDI_FIELD_SHIFT_VAL
    Set arincMsgTemp to BITWISE-OR-equal (txMsg.SM BITWISE-AND ARINC429_SSM_FIELD_LIMIT_MASK) LEFT-SHIFTED by ARINC429_SSM_FIELD_SHIFT_VAL), typecasted to uint32_t
    Set the dereferenced input pointer ARINCMsg equal to arincMsgTemp
RETURN writeMsgReturnStatus
```

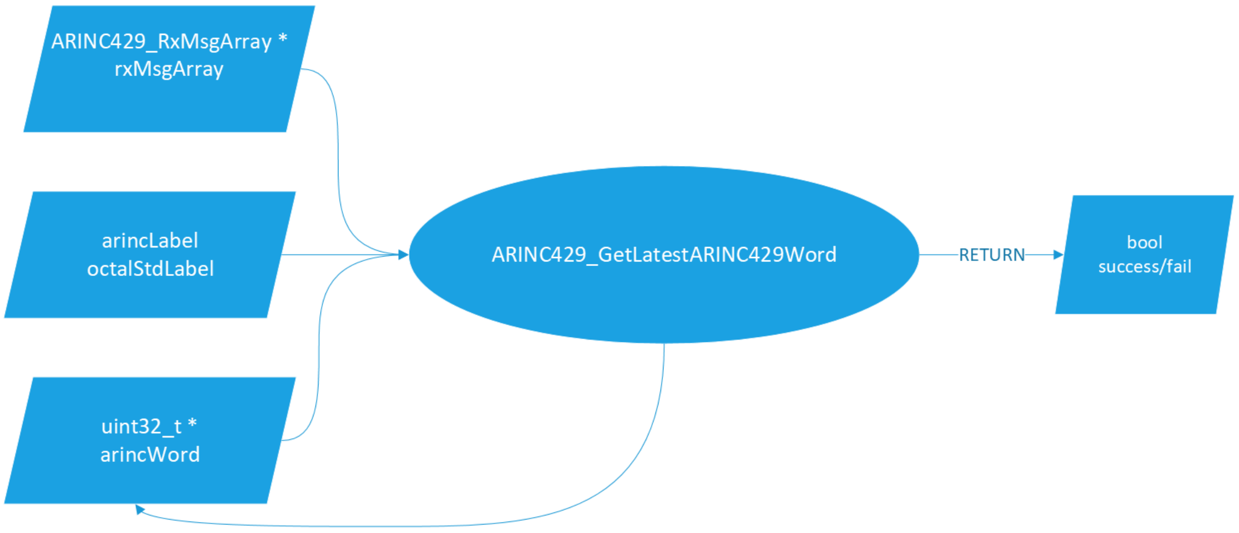
INT1.0101.S.IOP.4.008


ARINC429_LabelConfig * lblCfg
float engData
ARINC429_CheckValidityOfARINC_BNR_Data
RETURN
ARINC429_SM SSM

Description	The IOP ***shall*** check if a binary ARINC429 message's engineering data is within bounds.
Function call:	**ARINC429_CheckValidityOfARINC_BNR_Data**
Input parameters:	ARINC429_LabelConfig * **lblCfg**, float **engData**
Global data:	None
Function static data:	None

Requirement	IF engData is less than lblCfg.minValidValue) OR (engData is greater than lblCfg.maxValidValue))
	    RETURN ARINC429_SSM_BNR_FAILURE_WARNING
	ELSE
	    RETURN ARINC429_SSM_BNR_NORMAL_OPERATION

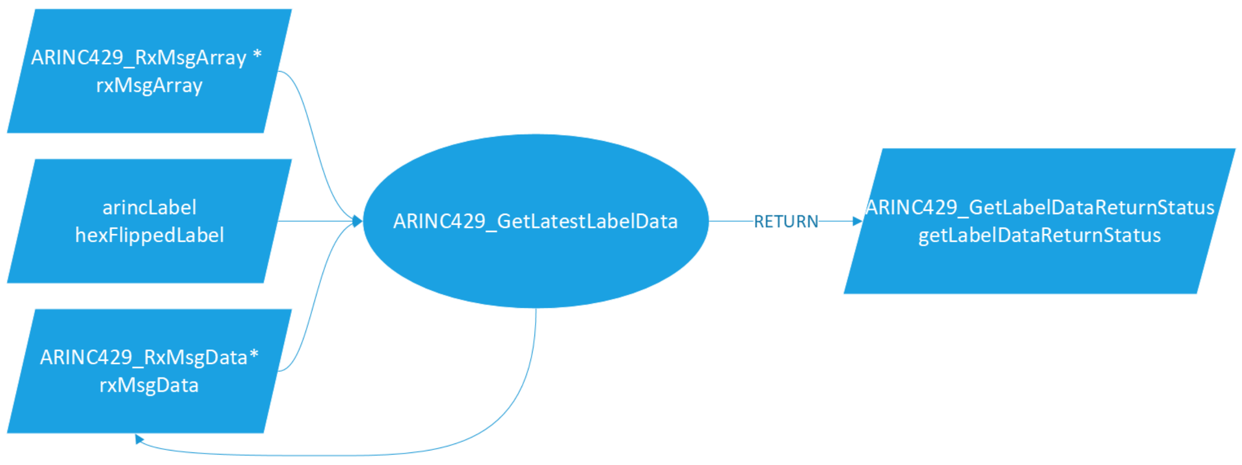
INT1.0101.S.IOP.4.009



Description	The IOP ***shall*** return the latest ARINC429 word if the receive message is not babbling and is fresh.
Function call:	ARINC429_GetLatestARINC429Word
Input parameters:	ARINC429_RxMsgArray * **rxMsgArray**, arincLabel **octalStdLabel**, uint32_t * **arincWord**
Global data:	None
Function static data:	None

Requirement	IF (NULL equals rxMsgArray or 0 equals OctalStdLabel OR OctalStdLabel is greater than MAX_OCTAL_LABEL_VLAUE or NULL equals arincWord
	    RETURN false
	Declare data;
	Set status equal to ARINC429_GetLatestLabelData(rxMsgArray, FormatLabelNumber(octalStdLabel), &data)
	IF ((ARINC429_GET_LABEL_DATA_MSG_SUCCESS equals status) AND
	        (true equals data.isDataFresh) AND
	        (true equals data.isNotBabbling))
	    Set the dereferenced value arincWord equal to data.rawARINCword;
	    RETURN true
	ELSE
	    RETURN false

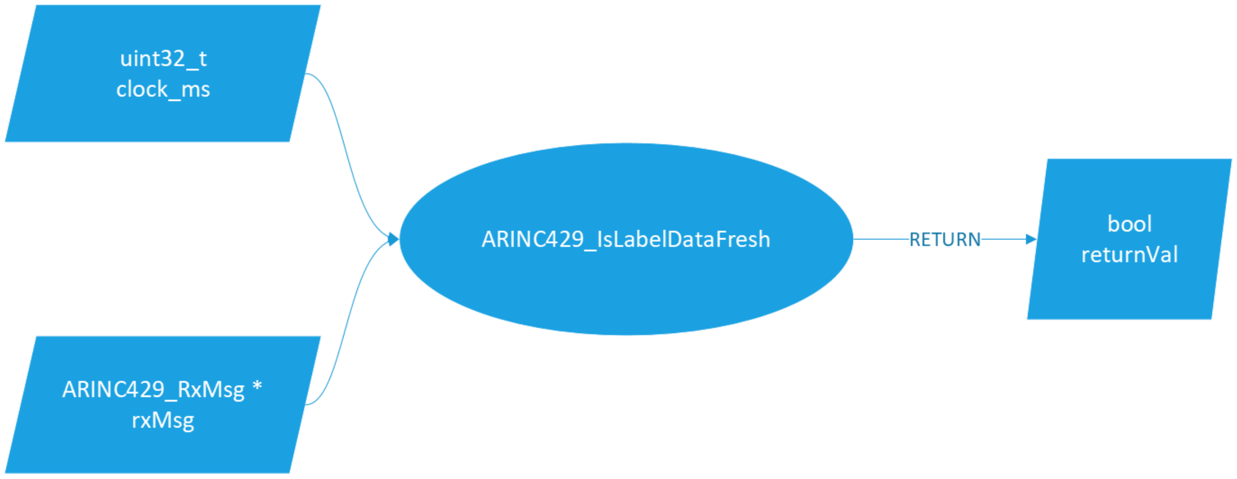
INT1.0101.S.IOP.4.010


ARINC429_RxMsgArray * rxMsgArray
arincLabel hexFlippedLabel
ARINC429_RxMsgData* rxMsgData
ARINC429_GetLatestLabelData
RETURN
ARINC429_GetLabelDataReturnStatus getLabelDataReturnStatus

Description	The IOP ***shall*** search an ARINC429 receive message array for the latest desired label data.
Function call:	ARINC429_GetLatestLabelData
Input parameters:	ARINC429_RxMsgArray * **rxMsgArray**, arincLabel **hexFlippedLabel**, uint32_t * ARINC429_RxMsgData * rxMsgData
Global data:	None
Function static data:	None
Requirement	

```
Declare getLabelDataReturnStatus
IF ( NULL equals rxMsgArray OR NULL equals rxMsgData)
    Set getLabelDataReturnStatus to ARINC429_GET_LABEL_DATA_ERROR_INVALID_ARGUMENT
ELSE
    Set count to 0
    Set labelMatchFound to false
    WHILE ( count is less than rxMsgArray.numMsgs AND count is less than maxNumRxMsgsInArray)
        IF ( rxMsgArray.rxMsgs[count].msgConfig.label equals hexFlippedLabel )
            Set labelMatchFound equal to true
            Set the dereferenced value rxMsgData equal to rxMsgArray.rxMsgs[count].data
            Set current_time_ms equal to Timer23_GetTimestamp_ms()
            Set rxMsgData.isDataFresh equal to ARINC429_IsLabelDataFresh(current_time_ms, &(rxMsgArray.rxMsgs[count])
            Set getLabelDataReturnStatus equal to ARINC429_GET_LABEL_DATA_MSG_SUCCESS
            BREAK
        Increment count by 1
    IF (false equals labelMatchFound)
        Set getLabelDataReturnStatus equal to ARINC429_GET_LABEL_DATA_ERROR_NO_MATCHING_LABEL
RETURN getLabelDataReturnStatus
```

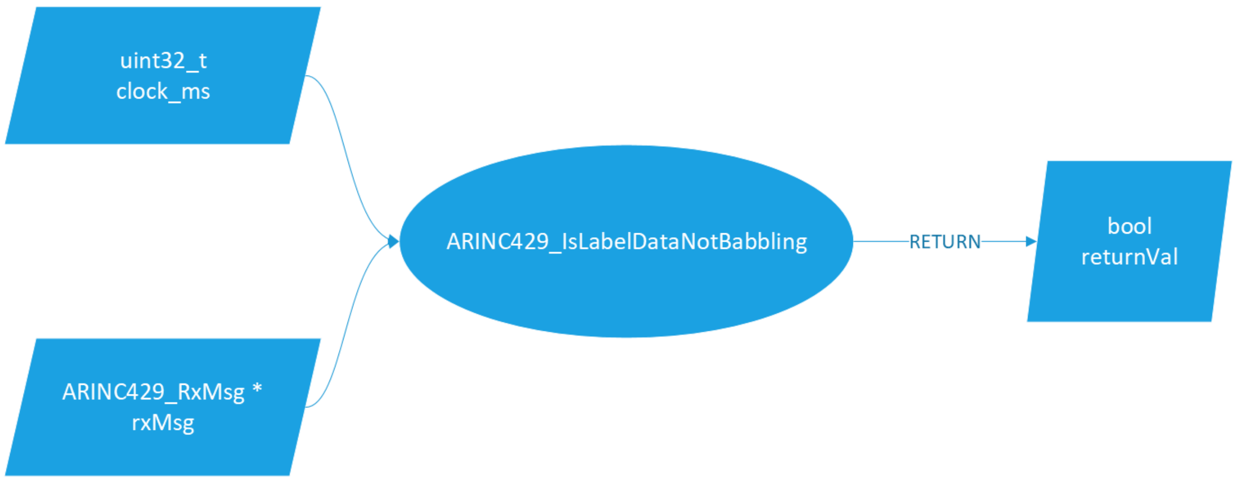
INT1.0101.S.IOP.4.011



Description	The IOP ***<u>shall</u>*** determine if an ARINC429 received message is fresh
Function call:	ARINC429_IsLabelDataFresh
Input parameters:	uint32_t **clock_ms**, ARINC429_RxMsg * **rxMsg**
Global data:	None
Function static data:	None

Requirement	IF (NULL equals rxMsg)
	    RETURN false
	Set elapsedTime_ms to clock_ms - rxMsg.data.sysTimeLastGoodMsg_ms
	Set returnVal to (elapsedTime_ms less than or equal to rxMsg.msgConfig.maxTransmitInterval_ms)
	RETURN returnVal

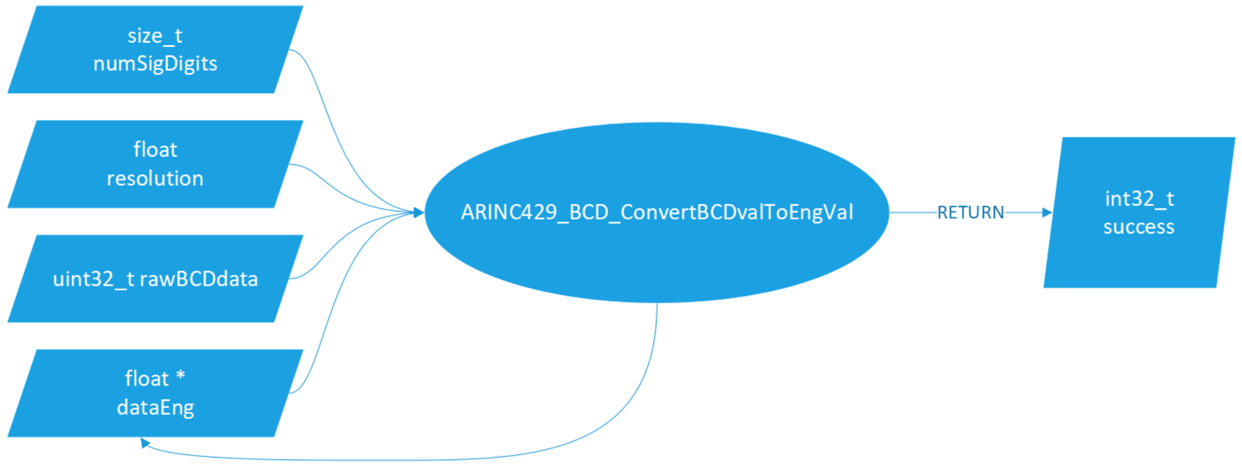
INT1.0101.S.IOP.4.012



Description	The IOP shall determine if an ARINC429 received message is not babbling.
Function call:	ARINC429_IsLabelDataNotBabbling
Input parameters:	uint32_t **clock_ms,** ARINC429_RxMsg * **rxMsg**
Global data:	None
Function static data:	None

Requirement	IF NULL equals rxMsg
	    RETURN false
	Set elapsedTime to clock_ms - rxMsg.data.sysTimeLastGoodMsg_ms
	Set returnVal to (elapsedTime greater than or equal to rxMsg->msgConfig.minTransmitInterval_ms)
	RETURN returnVal

INT1.0101.S.IOP.4.013

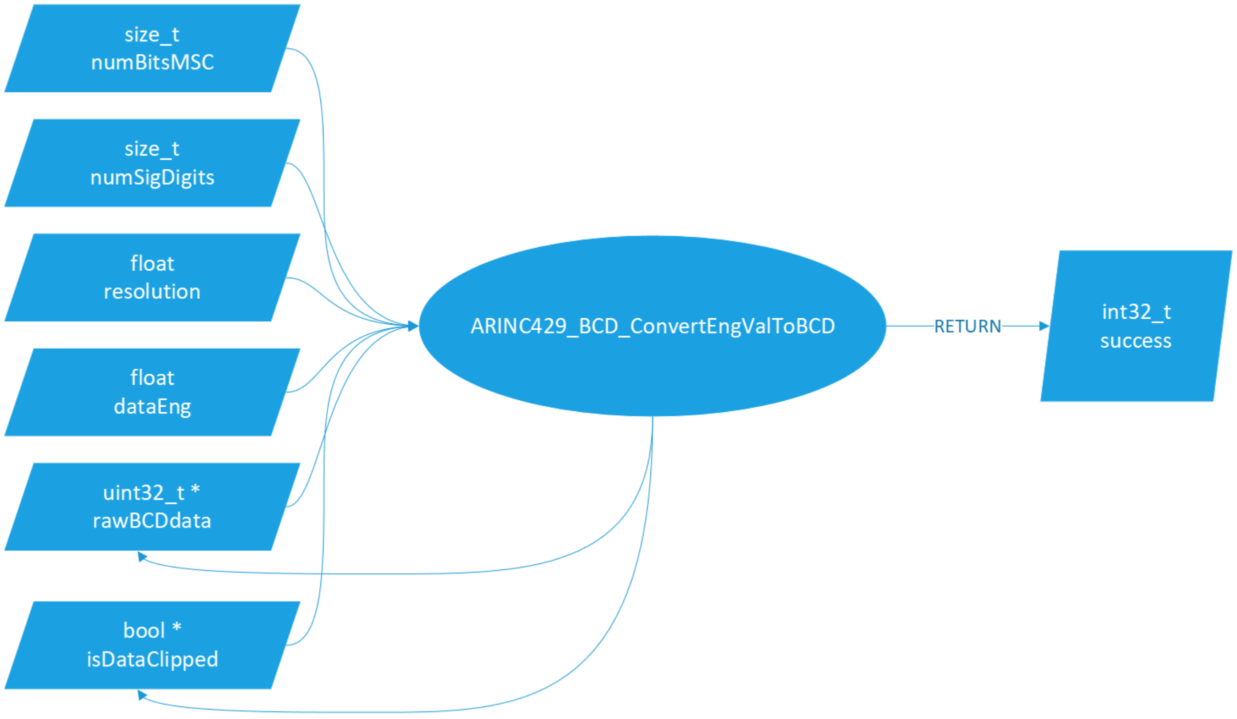


Description	The IOP ***shall*** convert BCD ARINC429 messages data value to engineering units
Function call:	ARINC429_BCD_ConvertBCDvalToEngVal
Input parameters:	size_t **numSigDigits**, float **resolution**, float * **dataEng**, uint32_t **rawBCDdata**
Global data:	None
Function static data:	None

Requirement	
	```
	Declare success
	IF NULL equals dataEng OR numSigDigits is less than 1 OR numSigDigits is greater than ARINC429_BCD_STD_MSG_MAX_NUM_SIGDIGITS
	    Set success to EXIT_FAILURE
	ELSE
	    Set calcValue to 0
	    Set tempVal equal to rawBCDdata
	    Set count to 0
	    Set multVal to 1
	    Declare thisDigit
	    WHILE ( tempVal is greater than 0 AND count is less than numSigDigits
	        Set thisDigit to tempVal BITWISE-AND 0xF
	        IF ( thisDigit is greater than ARINC429_BCD_MAX_DIGIT_VAL )
	            BREAK
	        Set calcValue plus-equals multVal * thisDigit
	        Set tempVal RIGHT-SHIFT-EQUALS ARINC429_BCD_BITS_PER_DIGIT
	        Set multVal times-equals 10
	        Increment count by 1
	    IF ( 0 equals tempVal)
	        Set success to EXIT_SUCCESS
	    ELSE
	        Set success to EXIT_FAILURE
	    IF ( 0 equals tempVal)
	        Set the value of the input pointer dataEng equal to calcValue * resolution, type-casted to float
	```

	```
	   ELSE
	      Set the value of the input pointer dataEng equal to 0
	RETURN success
	```

INT1.0101.S.IOP.4.014

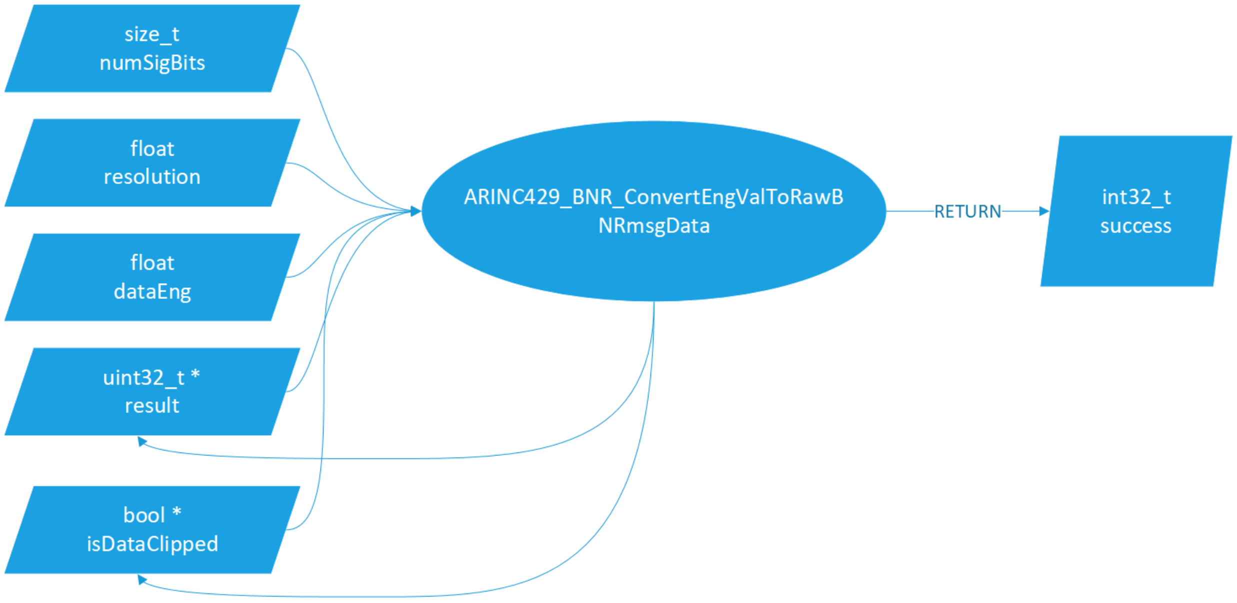


Description	The IOP ***shall*** convert BCD ARINC429 message's engineering value to BCD format
Function call:	ARINC429_BCD_ConvertEngValToBCD
Input parameters:	size_t **numBitsMSC**, size_t **numSigDigits**, float **resolution**, float **dataEng**, uint32_t * **rawBCDdata**, bool * **isDataClipped**
Global data:	None
Function static data:	None

Requirement	Set success to EXIT_SUCCESS
	IF (NULL equals rawBCDdata OR numSigDigits is less than 1 OR numSigDigits is greater than 5 OR numBitsMSC is less than 1 OR numBitsMSC is greater than 4
	    Set success to EXIT_FAILURE
	ELSE
	    Declare calcValue
	    IF resolution does not equal 0.0f
	        Set calcValue equal to dataEng/resolution
	    ELSE
	        Set calcValue equal to 0.0f
	    Set tempValue equal to min (calcValue + 0.5f, UINT32_MAX) typecasted to uint32_t
	    Set asBCD to 0
	    Set count to 0
	    Declare thisDigit
	    WHILE (tempValue is greater than 0 AND count is less than numSigDigits)
	        Set thisDigit equal to tempValue MODULUS 10

	```
	        IF ( numSigDigits equals (count + 1) AND thisDigit is greater than
	(UINT32_MAX RIGHT-SHIFTED by (NUM_BITS_IN_UINT32 - numBitsMSC)
	            BREAK
	        Set asBCD plus-equals thisDigit LEFT-SHIFTED by
	(ARINC429_BCD_BITS_PER_DIGIT * count)
	        Set tempValue divide-equals 10
	        Increment count by 1
	    IF ( 0 equals tempValue)
	        Set the value of the input pointer isDataClipped equal to false
	    ELSE
	        Set the value of the input pointer isDataClipped equal to true
	        Set asBCD equal to 0
	        Set count to 0
	        WHILE ( count is less than numSigDigits)
	            IF ( count does not equal (numSigDigits - 1)
	                Set thisDigit equal to ARINC429_BCD_MAX_DIGIT_VAL
	            ELSE
	                Set thisDigit equal to (UINT32_MAX RIGHT-SHIFTED by
	(NUM_BITS_IN_UINT32 - numBitsMSC))
	                asBCD plus-equals thisDigit LEFT-SHIFTEd by
	ARINC429_BCD_BITS_PER_DIGIT * count
	                Increment count by 1
	              Set the value of the input pointer rawBCDdata to asBCD
	RETURN success
	```

INT1.0101.S.IOP.4.015

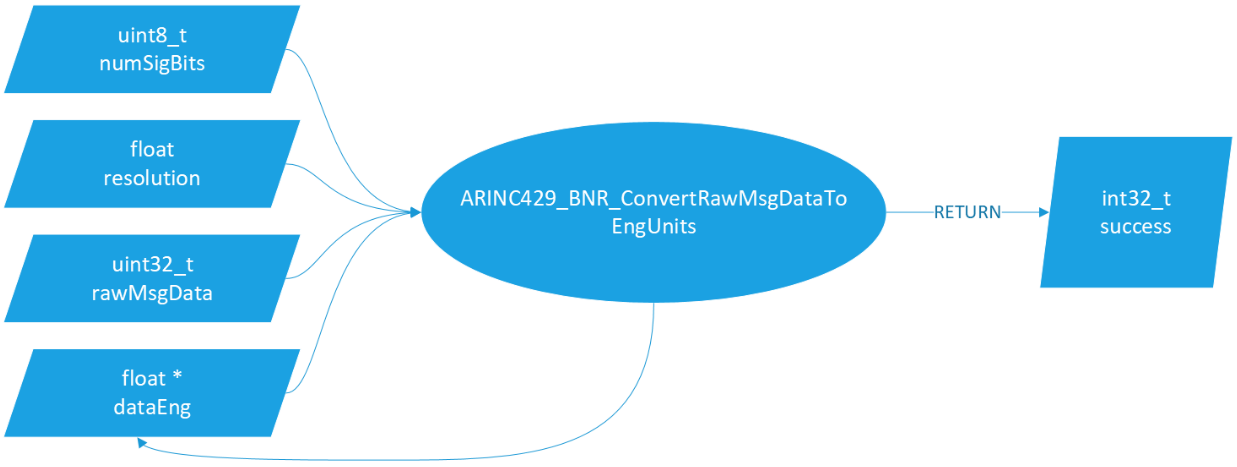


Description	The IOP ***shall*** convert ARINC429 engineering data to raw binary message data
Function call:	ARINC429_BNR_ConvertEngValToRawBNRmsgData
Input parameters:	size_t numSigBits, float resolution, float dataEng, uint32_t * result, bool * isDataClipped
Global data:	None
Function static data:	None

Requirement	Set success to EXIT_SUCCESS
	IF (NULL equals result OR NULL equals isDataClipped OR numSigBits is less than 1 OR numSigBits is greater than 20
	    RETURN EXIT_FAILURE
	ELSE
	    IF (resolution does not equal 0)
	        Set calcValue to dataEng/resolution
	    ELSE
	        Set calcValue to 0
	    IF (calcValue is less than 0)
	        calcValue plus-equals -0.5
	    ELSE
	        calcValue plus-equals 0.5
	    Set calcValueAsInt to calcValue, typecasted to int32_t
	    Set calcValueAsUint to calcValue, typecasted to uint32_t
	    Set isClipped to false
	    Set datafieldOvfCheckMaskVal to UINT32_MAX LEFT-SHIFTED by numSigBits
	    IF (calcValueAsUint BITWISE-AND INT32_SIGN_BIT_MASK)
	        IF (calcValueAsUint BITWISE-AND datafieldOvfCheckMaskVal) does not equal datafieldOvfCheckMaskVal
	            Set calcvalueAsUint equal to 1 LEFT-SHIFTED numSigBits
	            Set the value of the input pointer isClipped to true

	```
	    ELSE
	        IF (calcValueAsUint BITWISE-AND datafieldOvfCheckMaskVal
	            Set calcValueAsUint to UINT32_MAX RIGHT-SHIFT by (NUM_BITS_IN_UINT32 - numSigBits)
	            Set the value of the input pointer isClipped to true
	    Set the value of the input pointer isDataClipped equal to isClipped
	    Set the value of the input pointer result equal to calcValueAsUint
	RETURN success
	```

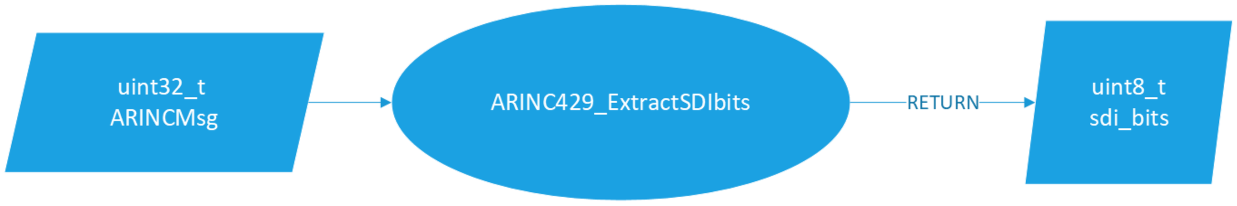
INT1.0101.S.IOP.4.016



Description	The IOP ***shall*** convert binary ARINC429 raw message data to engineering units.
Function call:	ARINC429_BNR_ConvertRawMsgDataToEngUnits
Input parameters:	uint8_t numSigBits, float resolution, uint32_t rawMsgData, float * dataEng
Global data:	None
Function static data:	None

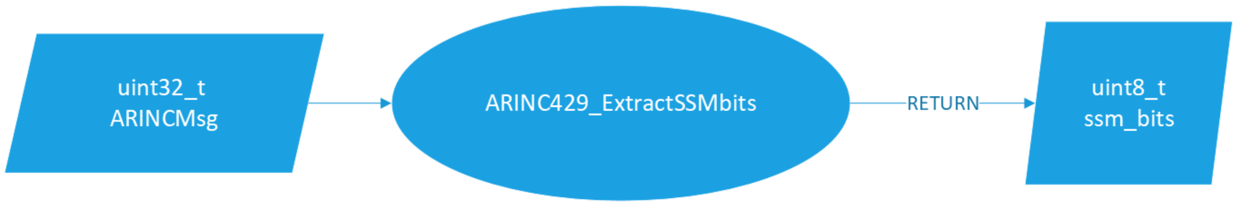
Requirement	Set success equal to EXIT_SUCCESS
	IF NULL equasls dataEng OR numSigBits is less than 1 OR numSigBits is greater than ARINC429_BNR_STD_MSG_MAX_NUM_SIGBITS
	    Set success equal to EXIT_FAILURE
	ELSE
	    Set rawMsgData_SignExt to rawMsgData
	    IF (0 does not equal 0x1 LEFT-SHIFTED by numSigBits BITWISE-AND rawMsgData)
	        Set rawMsgData_SignExt to BITWISE-OR-EQUALS (UINT32_MAX LEFT-SHIFTED by numSigBits)
	    Set msgDataAsInt to rawMsgData_SignExt, type-casted to int32_t
	    Set the value of the input pointer dataEng to msgDataAsInt * resolution, type-casted to float
	RETURN success

INT1.0101.S.IOP.4.017



Description	The IOP ***shall*** extract the SDI bits from an ARINC429 message
Function call:	ARINC429_ExtractSDIbits
Input parameters:	uint32_t ARINCMsg
Global data:	None
Function static data:	None
Requirement	RETURN (ARINCMsg RIGHTSHIFT ARINC429_SDI_FIELD_SHIFT_VAL) BITWISE-AND ARINC429_SDI_FIELD_LIMIT_MASK

INT1.0101.S.IOP.4.018

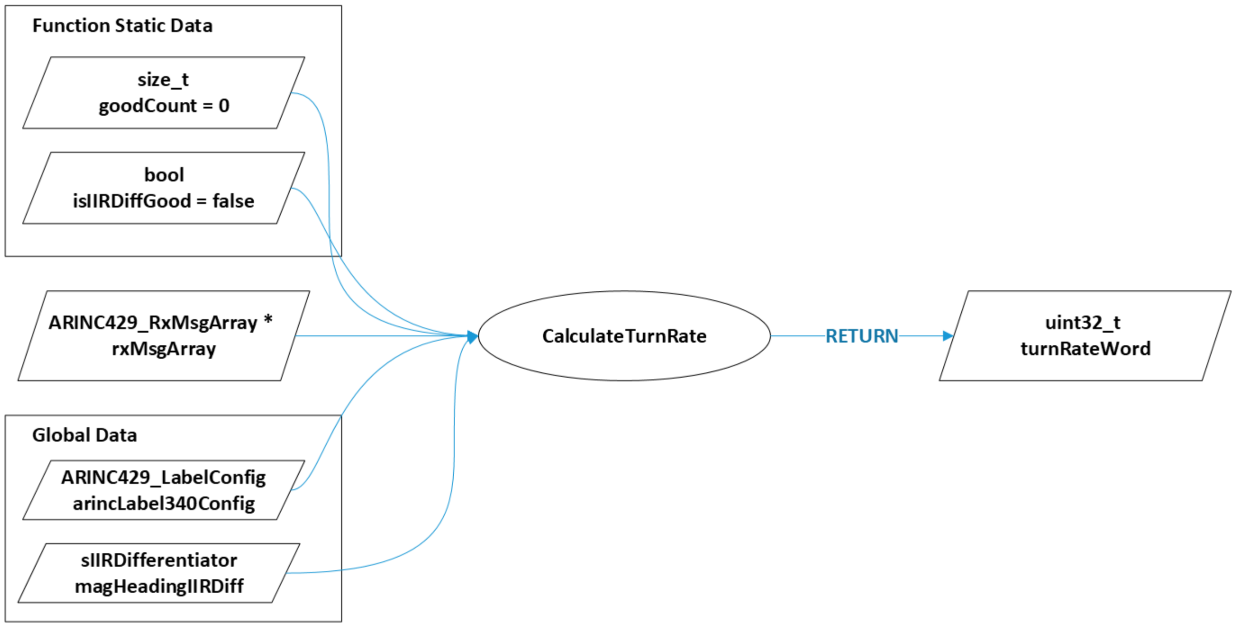


Description	The IOP ***shall*** extract the SSM bits from an ARINC429 message
Function call:	ARINC429_ExtractSSMbits
Input parameters:	uint32_t ARINCMsg
Global data:	None
Function static data:	None
Requirement	RETURN (ARINCMsg RIGHTSHIFT ARINC429_SSM_FIELD_SHIFT_VAL) BITWISE-AND ARINC429_SSM_FIELD_LIMIT_MASK

INT1.0101.S.IOP.5

High level requirement description: The IOP ***shall*** calculate the following ARINC429 words: magnetic heading, pitch angle, roll angle, body normal acceleration, body lateral acceleration, baro correction, status 272, status 274, status 275, turn rate, and slip angle.

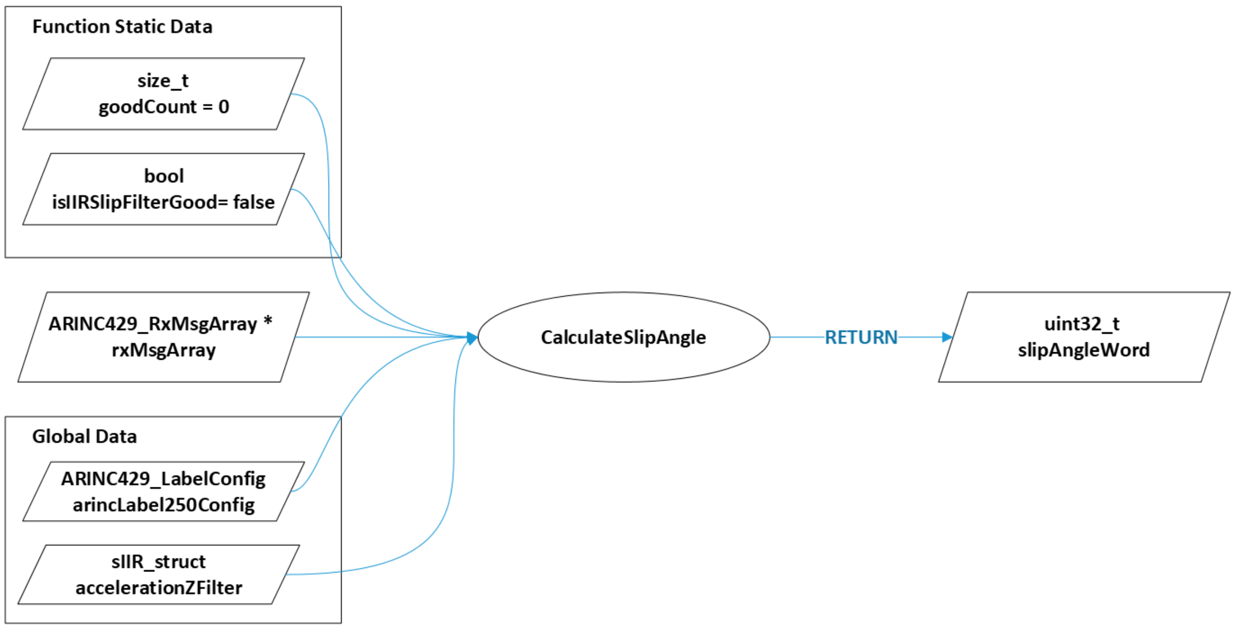
INT1.0101.S.IOP.5.001


Function Static Data
size_t goodCount = 0
bool isIIRDiffGood = false
ARINC429_RxMsgArray * rxMsgArray
CalculateTurnRate
RETURN
uint32_t turnRateWord
Global Data
ARINC429_LabelConfig arincLabel340Config
sIIRDifferentiator magHeadingIIRDiff

Description	The IOP ***shall*** compose the turn rate ARINC429 word.
Function call:	CalculateTurnRate
Input parameters:	ARINC429_RxMsgArray * **rxMsgArray**
Global data:	ARINC429_LabelConfig **arincLabel340Config**, sIIRDifferentiator **magHeadingIIRDiff**
Function static data:	bool **isIIRDiffGood**, size_t **goodCount**
Requirement	IF NULL equals rxMsgArray RETURN 0 Declare static bool isIIRDiffGood equal to false. Declare static size_t goodCount equal to 0 Declare goodThreshold equal to 10 Declare magHeadingData Set status equal to ARINC429_GetLatestLabelData(rxMsgArray, FormatLabelNumber(320), &magHeadingData) Declare turnRateWord Declare txMsgTurnRate Declare turnRate_dps IF ARINC429_GET_LABEL_DATA_MSG_SUCCESS equals status AND true equals magHeadingData.isDataFresh AND true equals magHeadingData.isNotBabbling AND

	```
	        ARINC429_SSM_BNR_NORMAL_OPERATION equals
	magHeadingData.SM
	   IF true equals isIIRDiffGood
	        Set turnRate_dps equal to IIR_Differentiator_Limited (
	magHeadingData.engDataFloat, &magHeadingIIRDiff )
	        Set txMsgTurnRate.SM equal to
	ARINC429_CheckValidityOfARINC_BNR_Data( turnRate_dps,
	&arincLabel340Config )
	   ELSE
	        IF 0 equals goodCount
	           Call v_IIRDifferentiatorReset( &magHeadingIIRDiff )
	           Call v_IIRDifferentiatorPreload(
	magHeadingData.engDataFloat,&magHeadingIIRDiff )
	           Set turnRate_dps equal to 0.0f
	        ELSE
	           Set turnRate_dps equal to IIR_Differentiator_Limited (
	magHeadingData.engDataFloat, &magHeadingIIRDiff )
	        Increment goodCount by 1
	        IF goodCount is greater than or equal to goodThreshold
	           Set isIIRDiffGood equal to true
	        Set txMsgTurnRate.SM equal to
	ARINC429_SSM_BNR_FAILURE_WARNING
	ELSE
	   Set isIIRDiffGood equal to false
	   Set goodCount equal to 0
	   Set turnRate_dps equal to magHeadingIIRDiff. pastOutputOfDiff
	   Set txMsgTurnRate.SM equal to ARINC429_SSM_BNR_FAILURE_WARNING
	Set txMsgTurnRate.msgConfig = &arincLabel340Config
	Set txMsgTurnRate.SDI = magHeadingData.SDI
	Set txMsgTurnRate.engData = turnRate_dps
	Call ARINC429_AssembleStdBNRmessage( &txMsgTurnRate, &turnRateWord )
	RETURN turnRateWord
	```

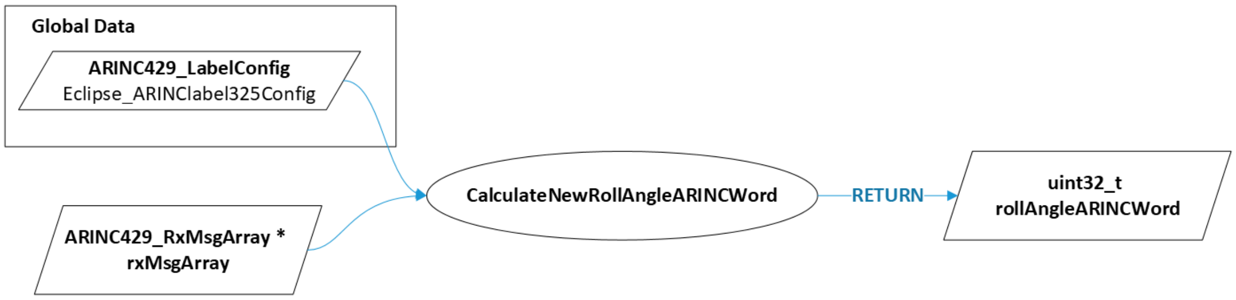
INT1.0101.S.IOP.5.002


Function Static Data
size_t goodCount = 0
bool isIIRSlipFilterGood= false
ARINC429_RxMsgArray * rxMsgArray
Global Data
ARINC429_LabelConfig arincLabel250Config
sIIR_struct accelerationZFilter
CalculateSlipAngle
RETURN
uint32_t slipAngleWord

Description	The IOP ***shall*** compose the slip angle ARINC429 word.
Function call:	CalculateSlipAngle
Input parameters:	ARINC429_RxMsgArray * **rxMsgArray**
Global data:	ARINC429_LabelConfig **arincLabel250Config**, sIIR_struct **accelerationZFilter**
Function static data:	size_t **goodCount**, bool **isIIRSlipFilterGood**
Requirement	IF NULL equals rxMsgArray RETURN 0 Declare static bool isIIRSlipFilterGood equals false Declare static size_t goodCount equals 0 Set goodThreshold equal to 10 Declare ayData Set readStatusAY equal to ARINC429_GetLatestLabelData(rxMsgArray, FormatLabelNumber(331), &ayData) Declare azData; Set readStatusAZ equal to ARINC429_GetLatestLabelData(rxMsgArray, FormatLabelNumber(333), &azData) Declare slipAngleWord Declare txMsgSlipAngle Set txMsgSlipAngle.msgConfig equal to &arincLabel250Config Set txMsgSlipAngle.SDI equal to azData.SDI Declare slipAngleInDegrees Declare filteredAZ IF ARINC429_GET_LABEL_DATA_MSG_SUCCESS equals readStatusAY AND true equals ayData.isDataFresh AND true equals ayData.isNotBabbling AND

```
        ARINC429_SSM_BNR_NORMAL_OPERATION equals ayData.SM))
    Set isAYdataValid equal to true
ELSE
    Set isAYdataValid equal to false
IF ARINC429_GET_LABEL_DATA_MSG_SUCCESS equals readStatusAZ AND
        true equals azData.isDataFresh AND
        true equals azData.isNotBabbling AND
        ARINC429_SSM_BNR_NORMAL_OPERATION equals azData.SM
    IF true equals isIIRSlipFilterGood
        Set filteredAZ equal to f32_IIRFilter( azData.engDataFloat, &accelerationZFilter )
        Set slipAngleInDegrees equal to radToDeg( f32_ArcTan2( -ayData.engDataFloat, filteredAZ + 1.0f ) )
        Set txMsgSlipAngle.SM equal to ARINC429_CheckValidityOfARINC_BNR_Data( slipAngleInDegrees, &arincLabel250Config )
    ELSE
        Set txMsgSlipAngle.SM equal to ARINC429_SSM_BNR_FAILURE_WARNING
        IF 0 equals goodCount
            Call v_IIRReset( &accelerationZFilter )
            Call v_IIRPreload( azData.engDataFloat, &accelerationZFilter )
            Set slipAngleInDegrees equal to 0
        ELSE
            Set filteredAZ equal to f32_IIRFilter( azData.engDataFloat, &accelerationZFilter )
            Set slipAngleInDegrees equal to radToDeg( f32_ArcTan2( -ayData.engDataFloat, filteredAZ ) )
        Increment goodCount by 1
        IF goodCount is greater than goodThreshold
            Set isIIRSlipFilterGood equal to true
ELSE
    Set slipAngleInDegrees equal to 0
    Set txMsgSlipAngle.SM equal to ARINC429_SSM_BNR_FAILURE_WARNING
    Set isIIRSlipFilterGood equal to false
    Set goodCount equal to 0
IF false equals isAYdataValid
    Set txMsgSlipAngle.SM equal to ARINC429_SSM_BNR_FAILURE_WARNING
Set txMsgSlipAngle.engData equal to slipAngleInDegrees
Call ARINC429_AssembleStdBNRmessage( &txMsgSlipAngle, &slipAngleWord )
RETURN slipAngleWord
```

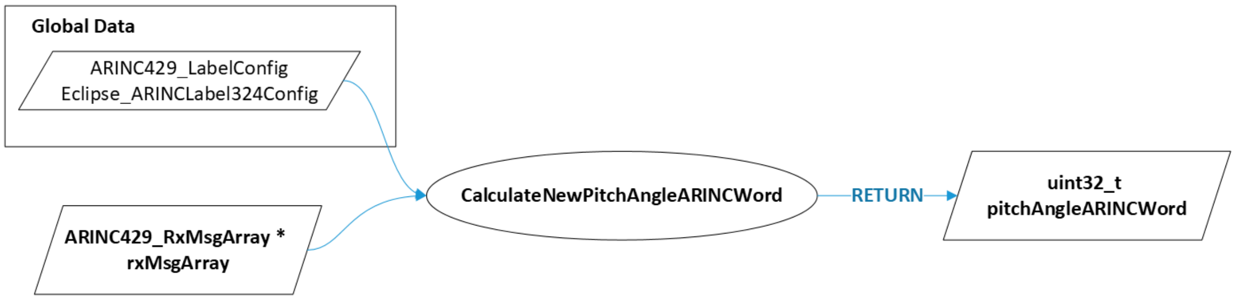
INT1.0101.S.IOP.5.003



Description	The IOP shall calculate the roll angle ARINC429 word based on Eclipse's message configuration.
Function call:	CalculateNewRollAngleARINCWord
Input parameters:	ARINC429_RxMsgArray * **rxMsgArray**
Global data:	ARINC429_LabelConfig **Eclipse_ARINClabel325Config**
Function static data:	None

Requirement	IF NULL equals rxMsgArray
	    RETURN 0
	Declare rollData
	Set status equal to ARINC429_GetLatestLabelData(rxMsgArray, FormatLabelNumber(325), &rollData)
	Declare rollAngleARINCWord
	Declare txMsgRollAngle
	Set txMsgRollAngle.msgConfig equal to &Eclipse_ARINClabel325Config
	Set txMsgRollAngle.SDI equal to rollData.SDI
	Set txMsgRollAngle.engData equal to rollData.engDataFloat
	IF ARINC429_GET_LABEL_DATA_MSG_SUCCESS equals status AND
	        true equals rollData.isDataFresh AND
	        true equals rollData.isNotBabbling
	    Set txMsgRollAngle.SM equal to rollData.SM
	ELSE
	    Set txMsgRollAngle.SM equal to ARINC429_SSM_BNR_FAILURE_WARNING
	Call ARINC429_AssembleStdBNRmessage(&txMsgRollAngle, &rollAngleARINCWord)
	RETURN rollAngleARINCWord

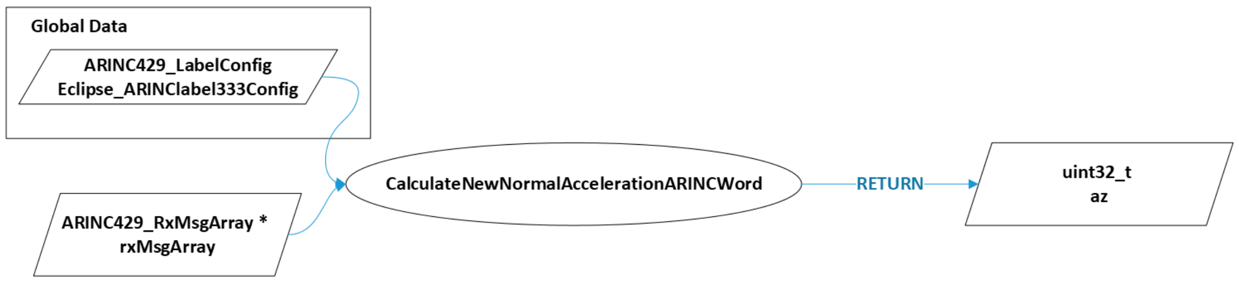
INT1.0101.S.IOP.5.004


Global Data
ARINC429_LabelConfig Eclipse_ARINCLabel324Config
ARINC429_RxMsgArray * rxMsgArray
CalculateNewPitchAngleARINCWord
RETURN
uint32_t pitchAngleARINCWord

Description	The IOP ***shall*** calculate the pitch angle ARINC429 word based on Eclipse's message configuration
Function call:	CalculateNewPitchAngleARINCWord
Input parameters:	ARINC429_RxMsgArray * **rxMsgArray**
Global data:	ARINC429_LabelConfig **Eclipse_ARINCLabel324Config**
Function static data:	None

Requirement	IF NULL equals rxMsgArray
	    RETURN 0
	Declare pitchData
	Set status equal to ARINC429_GetLatestLabelData(rxMsgArray, FormatLabelNumber(324), &pitchData)
	Declare pitchAngleARINCWord
	Declare txMsgPitchAngle
	Set txMsgPitchAngle.msgConfig equal to &Eclipse_ARINCLabel324Config
	Set txMsgPitchAngle.SDI equal to pitchData.SDI
	Set txMsgPitchAngle.engData equal to pitchData.engDataFloat
	IF ARINC429_GET_LABEL_DATA_MSG_SUCCESS equals status AND
	    true equals pitchData.isDataFresh AND
	    true equals pitchData.isNotBabbling
	    Set txMsgPitchAngle.SM equal to pitchData.SM
	ELSE
	    Set txMsgPitchAngle.SM equal to ARINC429_SSM_BNR_FAILURE_WARNING
	Call ARINC429_AssembleStdBNRmessage(&txMsgPitchAngle, &pitchAngleARINCWord)
	RETURN pitchAngleARINCWord

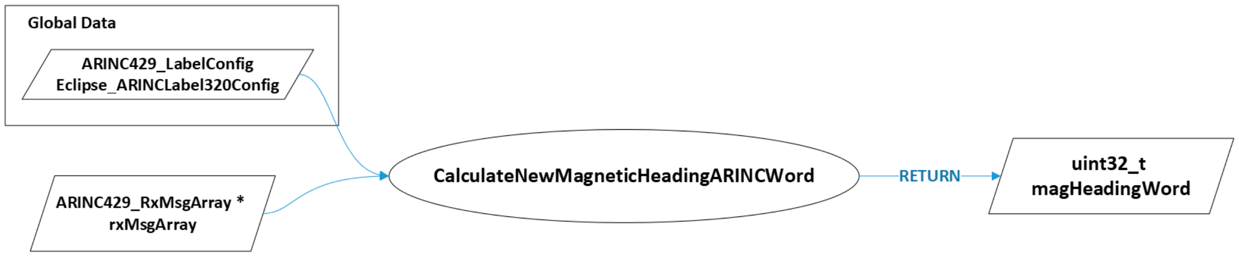
INT1.0101.S.IOP.5.005


Global Data
ARINC429_LabelConfig Eclipse_ARINClabel333Config
ARINC429_RxMsgArray * rxMsgArray
CalculateNewNormalAccelerationARINCWord
RETURN
uint32_t az

Description	The IOP ***shall*** calculate a new normal acceleration ARINC429 word based on the received normal acceleration
Function call:	CalculateNewNormalAccelerationARINCWord
Input parameters:	ARINC429_RxMsgArray * **rxMsgArray**
Global data:	ARINC429_LabelConfig **Eclipse_ARINClabel333Config**
Function static data:	None

Requirement	IF NULL equals rxMsgArray
	    RETURN 0
	Declare bodyNormAccelData
	Set status equal to ARINC429_GetLatestLabelData(rxMsgArray, FormatLabelNumber(333), &bodyNormAccelData)
	Declare az
	Declare txMsgNormAcc
	Set txMsgNormAcc.msgConfig equal to &Eclipse_ARINClabel333Config
	Set txMsgNormAcc.SDI equal to bodyNormAccelData.SDI
	Set azOffset equal to bodyNormAccelData.engDataFloat + 1.0f
	Set txMsgNormAcc.engData equal to azOffset
	IF ARINC429_GET_LABEL_DATA_MSG_SUCCESS equals status AND true equals bodyNormAccelData.isDataFresh AND true equals bodyNormAccelData.isNotBabbling
	    IF ARINC429_SSM_BNR_NORMAL_OPERATION equals bodyNormAccelData.SM
	        Set txMsgNormAcc.SM equal to ARINC429_CheckValidityOfARINC_BNR_Data(azOffset, &Eclipse_ARINClabel333Config)
	    ELSE
	        Set txMsgNormAcc.SM equal to bodyNormAccelData.SM
	ELSE
	    Set txMsgNormAcc.SM equal to ARINC429_SSM_BNR_FAILURE_WARNING
	Call ARINC429_AssembleStdBNRmessage(&txMsgNormAcc, &az)
	RETURN az

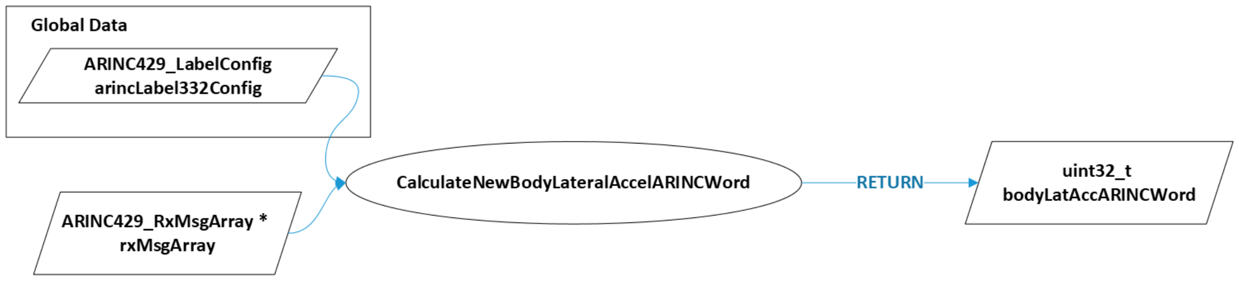
INT1.0101.S.IOP.5.006


Global Data
ARINC429_LabelConfig Eclipse_ARINCLabel320Config
ARINC429_RxMsgArray * rxMsgArray
CalculateNewMagneticHeadingARINCWord
RETURN
uint32_t magHeadingWord

Description	The IOP ***shall*** calculate a new magnetic heading ARINC word using the same data from the received word. The new magnetic heading word shall be formatted to match Eclipse's message configuration
Function call:	CalculateNewMagneticHeadingARINCWord
Input parameters:	ARINC429_RxMsgArray * **rxMsgArray**
Global data:	ARINC429_LabelConfig **Eclipse_ARINClabel320Config**
Function static data:	None
Requirement	IF NULL == rxMsgArray RETURN 0 Declare magHeadingData Set magHeadReadStatus equal to ARINC429_GetLatestLabelData(rxMsgArray, FormatLabelNumber(320), &magHeadingData) Declare lbl271Data Set lbl271ReadStatus equal to ARINC429_GetLatestLabelData(rxMsgArray, FormatLabelNumber(271), &lbl271Data) Declare magHeadingWord Declare txMsgMagHeading Set txMsgMagHeading.msgConfig equal to &Eclipse_ARINCLabel320Config Set txMsgMagHeading.SDI equal to magHeadingData.SDI Set txMsgMagHeading.engData equal to magHeadingData.engDataFloat IF ((ARINC429_GET_LABEL_DATA_MSG_SUCCESS equals magHeadReadStatus) AND magHeadingData.isDataFresh AND magHeadingData.isNotBabbling AND (ARINC429_GET_LABEL_DATA_MSG_SUCCESS equals lbl271ReadStatus) AND lbl271Data.isDataFresh AND lbl271Data.isNotBabbling AND (ARINC429_SSM_DIS_NORMAL_OPERATION equals lbl271Data.SM)) IF lbl271Data.rawARINCword BITWISE-AND AHRS_LABEL_271_MSU_FAIL_MASK

	Set txMsgMagHeading.SM equal to ARINC429_SSM_BNR_FAILURE_WARNING ELSE Set txMsgMagHeading.SM equal to magHeadingData.SM ELSE Set txMsgMagHeading.SM = ARINC429_SSM_BNR_FAILURE_WARNING Call ARINC429_AssembleStdBNRmessage(&txMsgMagHeading, &magHeadingWord) RETURN magHeadingWord

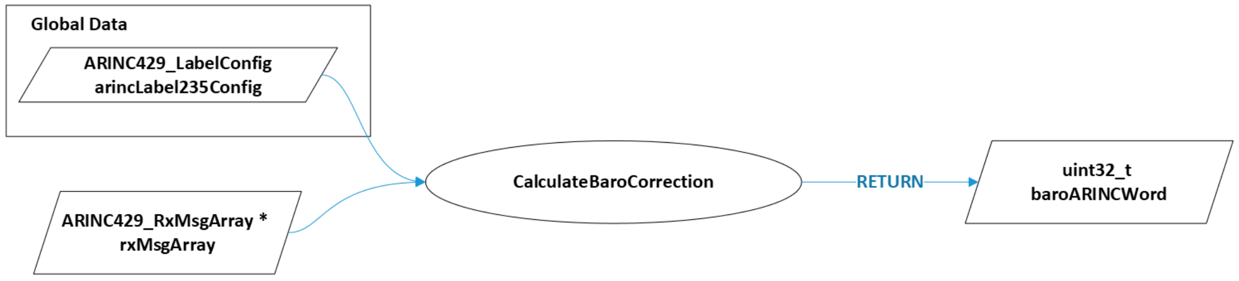
INT1.0101.S.IOP.5.007


Global Data
ARINC429_LabelConfig arincLabel332Config
ARINC429_RxMsgArray * rxMsgArray
CalculateNewBodyLateralAccelARINCWord
RETURN
uint32_t bodyLatAccARINCWord

Description	The IOP ***shall*** calculate the body lateral acceleration ARINC429 word based on Eclipse's message configuration.
Function call:	CalculateNewBodyLateralAccelARINCWord
Input parameters:	ARINC429_RxMsgArray * **rxMsgArray**
Global data:	ARINC429_LabelConfig **arincLabel332Config**
Function static data:	None
Requirement	IF NULL equals rxMsgArray RETURN 0 Declare bodyLatAccelData; Set status equal to ARINC429_GetLatestLabelData(rxMsgArray, FormatLabelNumber(332), &bodyLatAccelData); Declare bodyLatAccARINCWord Declare txMsgbodyLatAcc Set txMsgbodyLatAcc.msgConfig equal to &arincLabel332Config Set txMsgbodyLatAcc.SDI equal to bodyLatAccelData.SDI Set txMsgbodyLatAcc.engData equal to –(bodyLatAccelData.engDataFloat) IF ARINC429_GET_LABEL_DATA_MSG_SUCCESS equals status AND true equals bodyLatAccelData.isDataFresh AND true equals bodyLatAccelData.isNotBabbling Set txMsgbodyLatAcc.SM equal to bodyLatAccelData.SM ELSE Set txMsgbodyLatAcc.SM equal to ARINC429_SSM_BNR_FAILURE_WARNING

	Call ARINC429_AssembleStdBNRmessage(&txMsgbodyLatAcc, &bodyLatAccARINCWord) RETURN bodyLatAccARINCWord

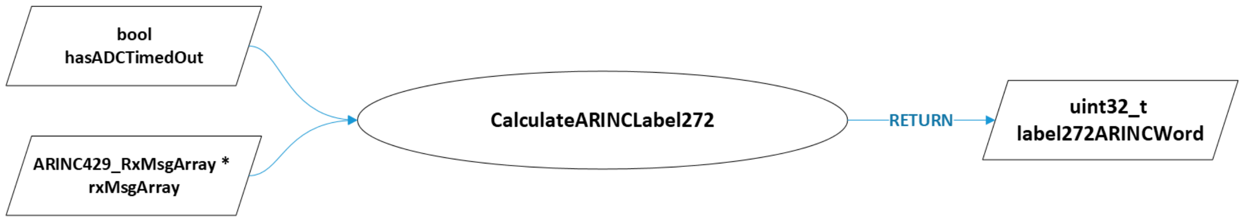
INT1.0101.S.IOP.5.008


Global Data
ARINC429_LabelConfig arincLabel235Config
ARINC429_RxMsgArray * rxMsgArray
CalculateBaroCorrection
RETURN
uint32_t baroARINCWord

Description	The IOP ***shall*** return the baro correction ARINC429 word based on input values
Function call:	CalculateBaroCorrection
Input parameters:	ARINC429_RxMsgArray * **rxMsgArray**
Global data:	ARINC429_LabelConfig arincLabel235Config
Function static data:	None

Requirement	Declare baroData
	Declare baroARINCWord
	Set status equal to ARINC429_GetLatestLabelData(rxMsgArray, FormatLabelNumber(235), &baroData)
	Declare baroMsg
	Set baroMsg.msgConfig equal to &arincLabel235Config;
	IF ARINC429_GET_LABEL_DATA_MSG_SUCCESS equals status AND
	        true equals baroData.isDataFresh AND
	        true equals baroData.isNotBabbling AND
	        ARNIC429_SSM_BCD_PLUS equals baroData.SM
	   Set baroMsg.engData equal to baroData.engDataFloat
	   Set baroMsg.SDI equal to baroData.SDI
	   Set baroMsg.SM equal to ARNIC429_SSM_BCD_PLUS
	ELSE
	   Set baroMsg.engData equal to 0.0f
	   Set baroMsg.SDI equal to0
	   Set baroMsg.SM equal to ARNIC429_SSM_BCD_NO_COMPUTED_DATA
	Call ARINC429_AssembleStdBCDmessage(&baroMsg, &baroARINCWord)
	RETURN baroARINCWord

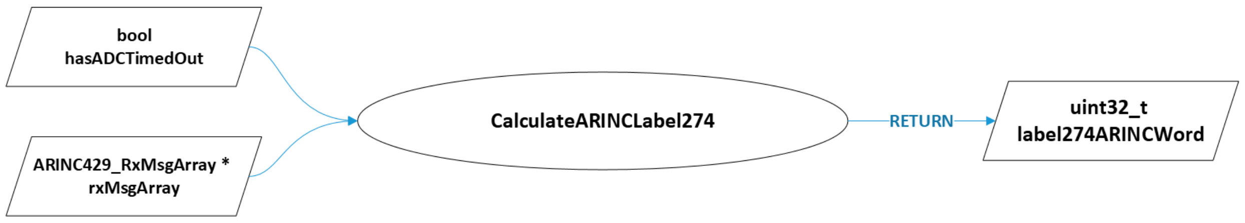
INT1.0101.S.IOP.5.009



Description	The IOP ***shall*** compose the AHRS Status Label 272 ARINC429 word
Function call:	CalculateARINCLabel272
Input parameters:	ARINC429_RxMsgArray * **rxMsgArray,** bool **hasADCTimedOut**
Global data:	None
Function static data:	None

Requirement	IF NULL equals rxMsgArray
	    RETURN 0
	Declare lbl271Data
	Set status equal to ARINC429_GetLatestLabelData(rxMsgArray, FormatLabelNumber(271), &lbl271Data)
	Set label272ARINCWord equal to 0x0000005D
	IF (lbl271Data.isDataFresh &&
	      lbl271Data.isNotBabbling &&
	      (ARINC429_GET_LABEL_DATA_MSG_SUCCESS == status) &&
	      (ARINC429_SSM_DIS_NORMAL_OPERATION == lbl271Data.SM))
	   Set label272ARINCWord to BITWISE-OR-EQUAL (lbl271Data.rawARINCword & AHRS_STATUS_SDI_SSM_MASK)
	   IF true euqals hasADCTimedOut
	      Set label272ARINCWord to BITWISE-OR-EQUAL AHRS_272_BIT_25_SET
	   IF lbl271Data.rawARINCword BITWISE-AND AHRS_LABEL_271_MSU_FAIL_MASK
	      Set label272ARINCWord to BITWISE-OR-EQUAL 0xC00u
	ELSE
	   Set label272ARINCWord to BITWISE-OR equals A429_DISC_SSM_FAIL_MASK
	RETURN label272ARINCWord

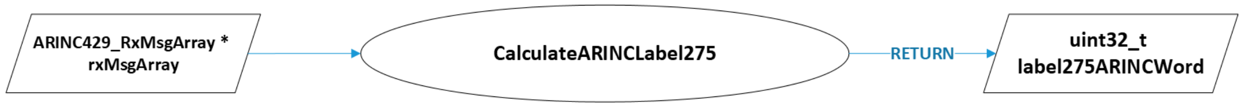
INT1.0101.S.IOP.5.010



Description	The IOP ***shall*** compose the AHRS Status Label 274 ARINC429 word
Function call:	CalculateARINCLabel274
Input parameters:	ARINC429_RxMsgArray * **rxMsgArray,** bool **hasADCTimedOut**
Global data:	None
Function static data:	None

Requirement	
	```
	IF NULL equals rxMsgArray
	  return 0
	Set label274ARINCWord equal to 0x0000003Du
	Declare lbl271Data;
	Set status271 equal to ARINC429_GetLatestLabelData( rxMsgArray, FormatLabelNumber( 271 ), &lbl271Data )
	Declare lbl270Data;
	Set status270 equal to ARINC429_GetLatestLabelData( rxMsgArray, FormatLabelNumber( 270 ), &lbl270Data )
	IF (lbl271Data.isDataFresh AND
	    lbl271Data.isNotBabbling AND
	    (ARINC429_SSM_DIS_NORMAL_OPERATION equals lbl271Data.SM)AND
	    (ARINC429_GET_LABEL_DATA_MSG_SUCCESS equals status271) AND
	    lbl270Data.isDataFresh AND
	    lbl270Data.isNotBabbling AND
	    (ARINC429_SSM_DIS_NORMAL_OPERATION equals lbl270Data.SM) AND
	    (ARINC429_GET_LABEL_DATA_MSG_SUCCESS equals status270))
	  label274ARINCWord to BITWISE-OR equals (lbl271Data.rawARINCword & AHRS_STATUS_SDI_SSM_MASK)
	  IF lbl271Data.rawARINCword BITWISE-AND AHRS_LABEL_271_MSU_FAIL_MASK
	    Set label274ARINCWord to BITWISE-OR equals 0x10000000u
	  IF (lbl270Data.rawARINCword & AHRS_LABEL_270_CAL_MASK)
	    Set label274ARINCWord to BITWISE-OR equals 0x800u
	  IF true equals hasADCTimedOut
	    Set label274ARINCWord to BITWISE-OR equals 0x1000u;
	ELSE
	  Set label274ARINCWord to BITWISE-OR equals A429_DISC_SSM_FAIL_MASK
	RETURN label274ARINCWord
	```

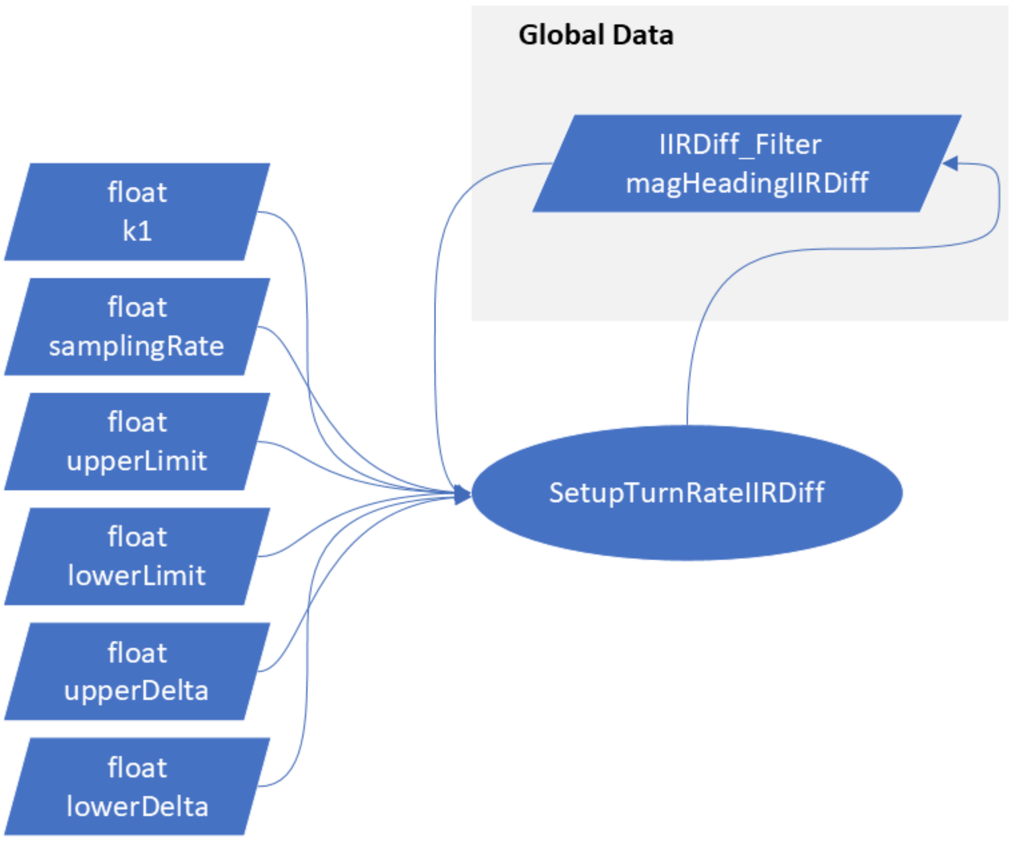
INT1.0101.S.IOP.5.011



Description	The IOP ***shall*** compose the AHRS Status Label 275 ARINC429 word
Function call:	CalculateARINCLabel275
Input parameters:	ARINC429_RxMsgArray * **rxMsgArray**
Global data:	None
Function static data:	None

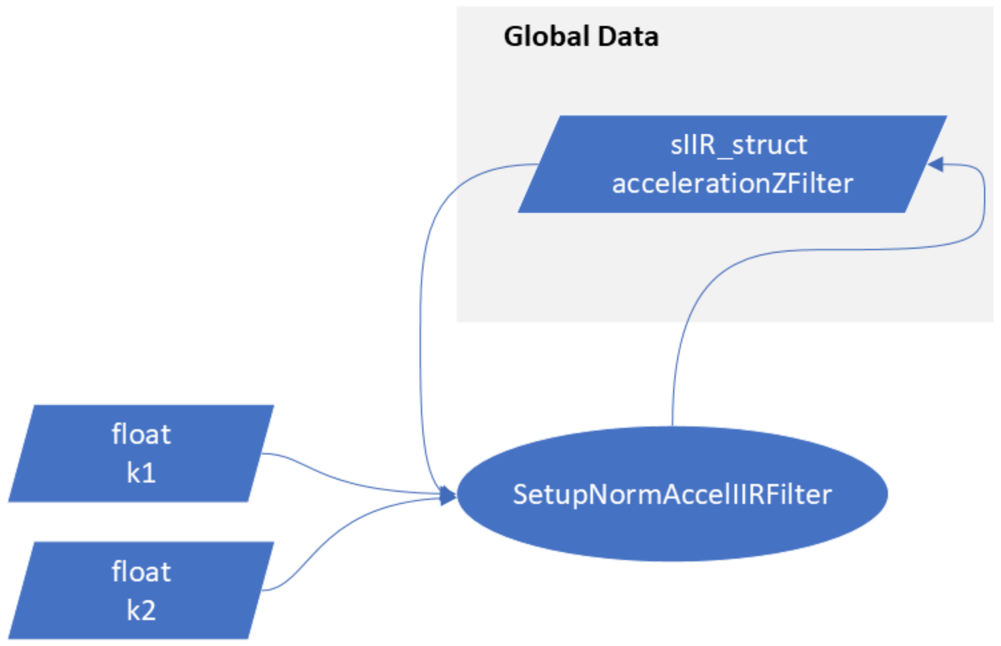
Requirement	
	```
	IF NULL equals rxMsgArray
	   return 0
	Declare lbl271Data
	Set status equal to ARINC429_GetLatestLabelData( rxMsgArray, FormatLabelNumber( 271 ), &lbl271Data )
	Declare flightPathAccelData
	Set fpaStatus equals to ARINC429_GetLatestLabelData( rxMsgArray, FormatLabelNumber( 323 ), &flightPathAccelData )
	Set label275ARINCWord equal to 0x000040BD
	IF ((ARINC429_GET_LABEL_DATA_MSG_SUCCESS equals status) AND
	      lbl271Data.isNotBabbling AND
	      lbl271Data.isDataFresh AND
	      (ARINC429_SSM_DIS_NORMAL_OPERATION equals lbl271Data.SM) AND
	      (ARINC429_GET_LABEL_DATA_MSG_SUCCESS equals fpaStatus) AND
	      flightPathAccelData.isDataFresh AND
	      flightPathAccelData.isNotBabbling)
	   Set label275ARINCWord to BITWISE-OR equals ((lbl271Data.rawARINCword BITWISE-AND AHRS_STATUS_SDI_SSM_MASK))
	   IF (lbl271Data.rawARINCword & AHRS_LABEL_271_MSU_FAIL_MASK)
	      Set label275ARINCWord to BITWISE-OR equals 0x400000
	   IF ARINC429_SSM_BNR_NORMAL_OPERATION does not equal flightPathAccelData.SM
	      Set label275ARINCWord to BITWISE-OR equals 0x3000000u
	   ELSE
	      Set label275ARINCWord to BITWISE-OR equals 0x2000000u
	   Set label275ARINCWord to BITWISE-OR equals A429_DISC_SSM_FAIL_MASK
	RETURN label275ARINCWord
	```

5.12.5.2 INT1.0101.S.IOP.5.012



Description	The IOP ***shall*** setup an IIR differentiator for magnetic heading based on configuration data.
Function call:	SetupTurnRateIIRDiff
Input parameters:	float **k1**, float **samplingRate**, float **upperLimit**, float **lowerLimit**, float **upperDelta**, float **lowerDelta**
Global data:	IIRDiff_Filter **magHeadingIIRDiff**
Function static data:	None
Requirement	Call IIRDifferentiatorSetup(&**magHeadingIIRDiff**, **k1**,**samplingRate**, **upperLimit**, **lowerLimit**, **upperDelta**, **lowerDelta**);

5.12.6.1 INT1.0101.S.IOP.5.013



Description	The IOP ***shall*** setup the IIR filter for normal acceleration based on configuration data.
Function call:	SetupNormAccelIIRFilter
Input parameters:	float k1, float k2
Global data:	sIIR_struct **accelerationZFilter**
Function static data:	None
Requirement	Call v_IIRSetup (&sIIR_struct accelerationZFilter, k1, k2)

5.12.6.2 INT1.0101.S.IOP.5.014

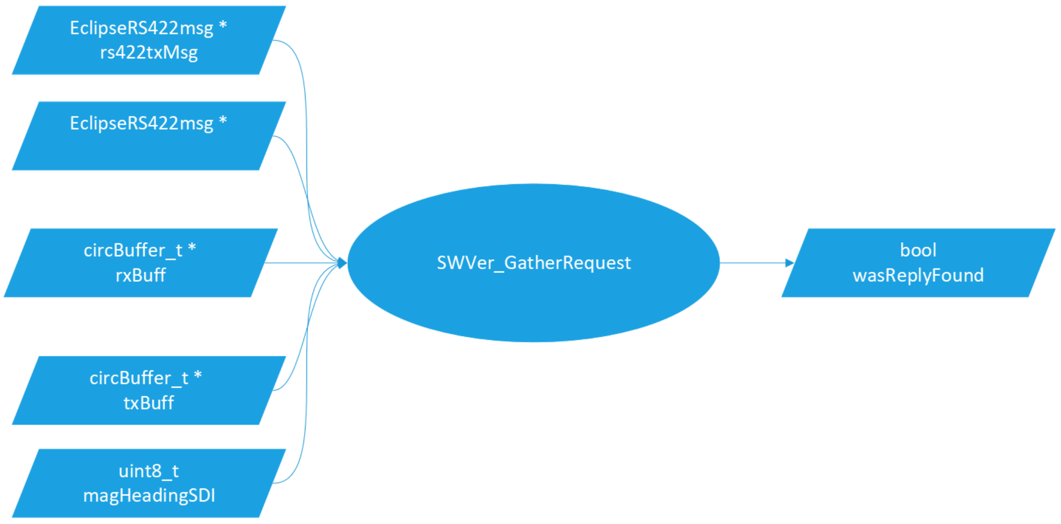
Description	The IOP ***shall*** use a floating point library for all floating point conversions and calculations.

Note: See

NT1.0101.S.IOP.6

High level requirement description: The IOP ***shall*** transmit the Eclipse specified ARINC429 Label 376, Software Version

INT1.0101.S.IOP.6.001

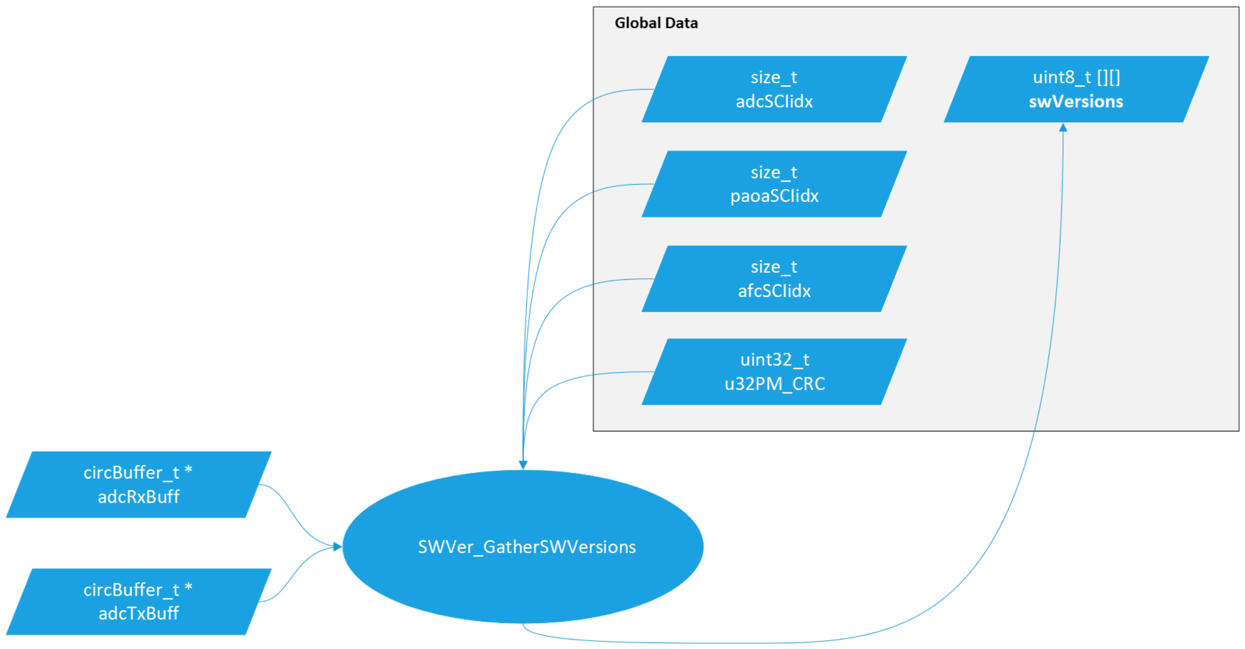


Description	The IOP ***shall*** send a version request message to an Eclipse RS422 subsystem.
Function call:	SWVer_GatherRequest
Input parameters:	EclipseRS422msg * **rs422txMsg**, **EclipseRS422msg** * rs422rxMsg, circBuffer_t * **txBuff**, circBuffer_t * **rxBuff**, uint8_t **magHeadingSDI**
Global data:	None
Function static data:	None

Requirement	Implementation:
	IF (NULL equals txBuff OR NULL equals rxBuff OR NULL equals rs422txMsg OR NULL equals rx422rxMsg OR magHeadingSDI is greater than 0x03
	    RETURN false
	Set maxNumRequestRetries equal to 10
	Set requestCount equal to 0
	Declare wasReplyFound
	WHILE (requestCount is less than maxNumRequestRetries)
	    Declare dummyData
	    Set wasReplyFound equal to false
	    Call EclipseRS422_ConstructTxMsg(rx422txMsg, txBuff, NULL, 0, magHeadingSDI, ECLIPSE_RS422_VERSION_REQUEST_TXMSG_LENGTH)
	    Call UART1_TxStart()
	    Call Timer23_Delay_ms(5)
	    Call UART1_ReadToRxCircBuff()
	    Set wasReplyFound equal to (Eclipse422_ProcessNewMessage(rxBuff, 1, rs422rxMsg, &dummyData)
	    IF (true equals wasReplyFound)
	        BREAK
	    ELSE

	Increment requestCount by 1 RETURN wasReplyFound

INT1.0101.S.IOP.6.002



Description	The IOP ***shall*** gather all Eclipse Aviation formatted software version numbers from the attached RS422 subsystems
Function call:	SWVer_GatherSWVersions
Input parameters:	circBuffer_t * **adcRxBuff**, circBuffer_t * **adcTxBuff**
Global data:	uint8_t [][]**swVersions**, size_t **afcSCIidx**, size_t **adcSCIidx,** size_t **paoaSCIidx,** uint32_t u32PM_CRC
Function static data:	None

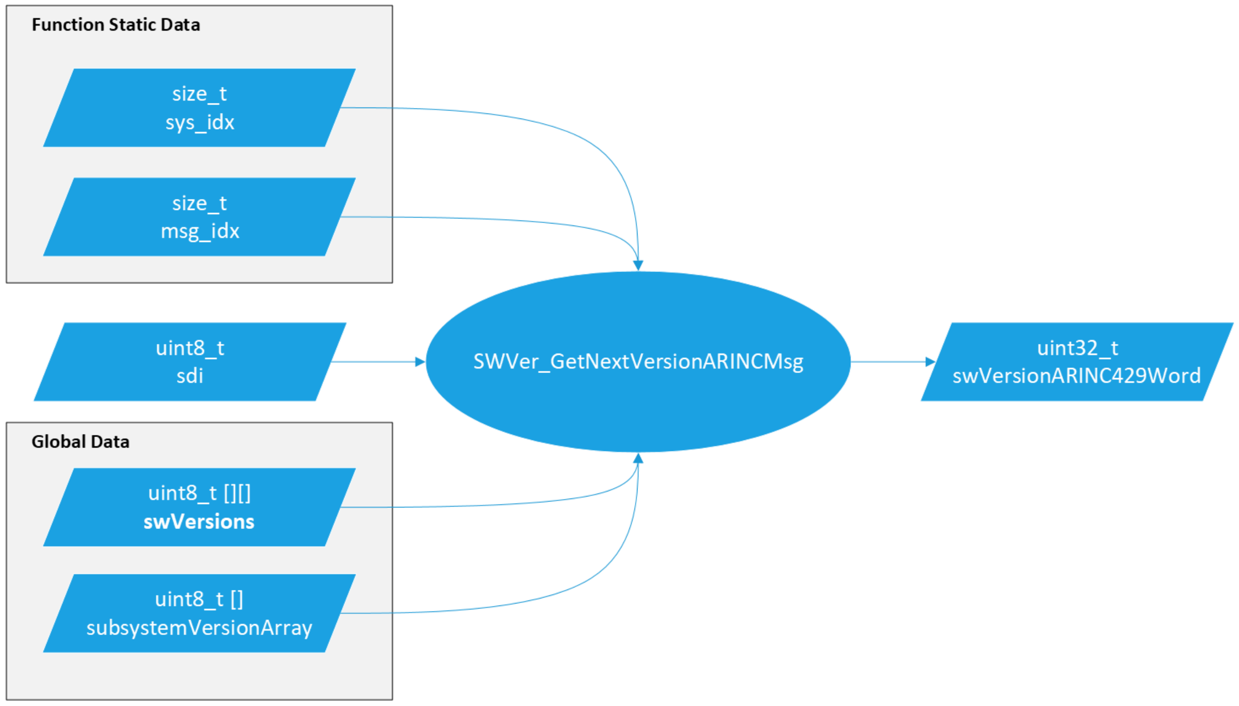
Requirement	
	```
	IF NULL equals adcTxBuff) OR NULL equals adcRxBuff
	    RETURN
	    /* Zero the local array */
	Declare x
	Declare y
	FOR (x equals 0; x is less than NUM_AFC004_SCI; increment x by 1)
	    FOR (y equals 0; y is less than  NUM_BYTES_PER_SCI_VERSION; increment y by 1)
	        Set swVersions[x][y] equal to 0

	Declare EclipseRS422msgConfig swVerADCRequestCfg
	Set swVerADCRequestCfg.cmd equal to SOFTWARE_VERSION_CMD,
	Set swVerADCRequestCfg.leftSource equal to LEFT_AHRS,
	Set swVerADCRequestCfg.rightSource equal to RIGHT_AHRS,
	Set swVerADCRequestCfg.leftDestination equal to LEFT_ADC,
	Set swVerADCRequestCfgrightDestination equal to RIGHT_ADC,
	```

```
Set swVerADCRequestCfg.length equal to
ECLIPSE_RS422_VERSION_REQUEST_MSG_LENGTH

Declare EclipseRS422msgConfig swVerADCReplyCfg
Set swVerADCReplyCfg.cmd equal to SOFTWARE_VERSION_CMD,
Set swVerADCReplyCfg.leftSource equal to LEFT_ADC,
Set swVerADCReplyCfg.rightSource equal to RIGHT_ADC,
Set swVerADCReplyCfg.leftDestination equal to LEFT_AHRS,
Set swVerADCReplyCfg.rightDestination equal to RIGHT_AHRS,
Set swVerADCReplyCfg.length equal to
ECLIPSE_RS422_ADC_SWVERSION_REPLY_MSG_LENGTH

Declare EclipseRS422msgConfig hwVerADCRequestCfg
Set hwVerADCRequestCfg.cmd equal to HARDWARE_SERIAL_NUMBER_CMD,
Set hwVerADCRequestCfg.leftSource equal to LEFT_AHRS,
Set hwVerADCRequestCfg.rightSource equal to RIGHT_AHRS,
Set hwVerADCRequestCfg.leftDestination equal to LEFT_ADC,
Set hwVerADCRequestCfg.rightDestination equal to RIGHT_ADC,
Set hwVerADCRequestCfg.length equal to
ECLIPSE_RS422_VERSION_REQUEST_MSG_LENGTH

Declare EclipseRS422msgConfig hwVerADCReplyCfg
Set hwVerADCReplyCfg.cmd equal to HARDWARE_SERIAL_NUMBER_CMD,
Set hwVerADCReplyCfg.leftSource equal to LEFT_ADC,
Set hwVerADCReplyCfg.rightSource equal to RIGHT_ADC,
Set hwVerADCReplyCfg.leftDestination equal to LEFT_AHRS,
Set hwVerADCReplyCfg.rightDestination equal to RIGHT_AHRS,
Set hwVerADCReplyCfg.length equal to
ECLIPSE_RS422_ADC_HWVERSION_REPLY_MSG_LENGTH

Declare
adcSwVersionReplyData[ECLIPSE_RS422_ADC_SWVERSION_REPLY_MSG_LENGTH
- 1]
Declare
adcHwVersionReplyData[ECLIPSE_RS422_ADC_HWVERSION_REPLY_MSG_LENGTH
- 1]
Declare
adcSwVersionRequestData[ECLIPSE_RS422_VERSION_REQUEST_TXMSG_LENGTH]
Declare
adcHwVersionRequestData[ECLIPSE_RS422_VERSION_REQUEST_TXMSG_LENGTH]

Declare EclipseRS422msg swVersionRequestADCMsg
Set swVersionRequestADCMsg.msgConfig equal to &swVerADCRequestCfg,
Set swVersionRequestADCMsg.data equal to adcSwVersionRequestData

Declare EclipseRS422msg swVersionReplyADCMsg
```

```
Set swVersionReplyADCMsg.msgConfig equal to &swVerADCReplyCfg,
Set swVersionReplyADCMsg.data equal to adcSwVersionReplyData

Declare EclipseRS422msg hwVersionRequesADCtMsg
Set hwVersionRequesADCtMsg.msgConfig equal to &hwVerADCRequestCfg,
Set hwVersionRequesADCtMsg.data equal to adcHwVersionRequestData

Declare EclipseRS422msg hwVersionReplyADCMsg
Set hwVersionReplyADCMsg.msgConfig equal to &hwVerADCReplyCfg,
Set .data equal to adcHwVersionReplyData

IF (true equals SWVer_GatherRequest( &swVersionRequestADCMsg,
                        &swVersionReplyADCMsg,
                        adcTxBuff,
                        adcRxBuff,
                        0x01 ))
   Call memcpy( &(swVersions[adcSCIidx][0]), &adcSwVersionReplyData,
ECLIPSE_RS422_SWVERSION_DATA_LENGTH )
   Set the uint8_t pointer pitotAOASwverion equal to (adcSwVersionReplyData +
ECLIPSE_RS422_SWVERSION_DATA_LENGTH)
   Call memcpy( &(swVersions[paoaSCIidx][0]), pitotAOASwverion,
ECLIPSE_RS422_SWVERSION_DATA_LENGTH )
IF (true equals SWVer_GatherRequest( &hwVersionRequesADCtMsg,
                        &hwVersionReplyADCMsg,
                        adcTxBuff,
                        adcRxBuff,
                        0x01 ))
   Call memcpy( &(swVersions[adcSCIidx][ECLIPSE_RS422_HWVERSION_OFFSET]),
&adcHwVersionReplyData, ECLIPSE_RS422_HWVERSION_DATA_LENGTH )
   Set the uint8_t pointer pitotAOAHWVerion equal to adcHwVersionReplyData +
ECLIPSE_RS422_HWVERSION_DATA_LENGTH
   Call memcpy(
&(swVersions[paoaSCIidx][ECLIPSE_RS422_HWVERSION_OFFSET]),
pitotAOAHWVerion, ECLIPSE_RS422_HWVERSION_DATA_LENGTH )

Declare crcCounter
Declare msgNibble
FOR (crcCounter equals 0; crcCounter less than NUM_CHARS_IN_32BIT_CRC;
Increment crcCounter by 1)
   Set msgNibble equal to ((((u32PM_CRC LEFT-SHIFT 4) * crcCounter) BITWISE-
AND (0xF0000000)) RIGHT-SHIFT 28), typecasted to uint8_t
   Set swVersions[afcSCIidx][crcCounter] equal to asciiConverter( msgNibble )
FOR (crcCounter equals 0; crcCounter less than NUM_BYTES_IN_32BIT_CRC;
Increment crcCounter by 1)
```

	Set swVersions[afcSCIidx][crcCounter + ECLIPSE_RS422_SWVERSION_CRC_POS_OFFSET] equal to ((u32PM_CRC RIGHT-SHIFT (8 * crcCounter)) BITWISE-AND 0xFF), typecasted to uint8_t

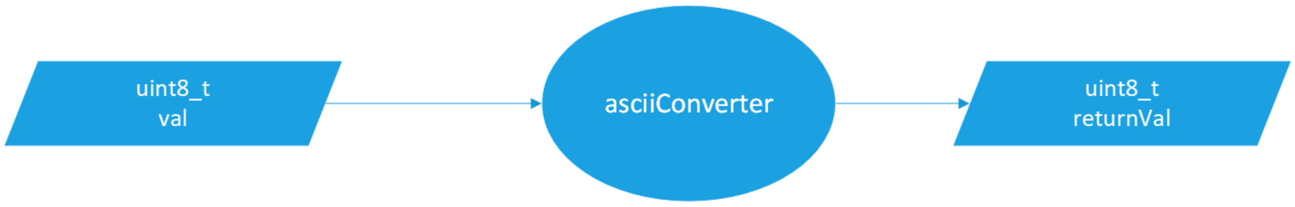
INT1.0101.S.IOP.6.003



Description	The IOP ***shall*** compose an ARINC429 message that matches Eclipse's software version format.
Function call:	SWVer_GetNextVersionARINCMsg
Input parameters:	uint8_t **sdi**
Global data:	uint8_t[][] **swVersions**, uint8_t **subsystemVersionArray**[],
Function static data:	size_t **msg**_idx, size_t **sys**_idx
Requirement	Declare a static variable msg_idx and set to 0 Declare a static variable sys_idx and set to 0 Set swVersionARINC429Word equal to ARINC429_SWVERSION_LABEL Set subSys equal to subsystemVersionArray[sys_idx], typecased to uint32_t Set msgSubIdx equal to msg_idx, typecasted to uint32_t Set data equal to swVersion[sys_idx][msg_idx], type-castesd to uint32_t Set swVersionARINC429Word BITWISE-OR-equals (sdi LEFT-SHIFTED by ARINC429_SDI_SHIFT_VAL) Set swVersionARINC429Word BITWISE-OR-equals (subSys << ARINC429_SUBSYS_IDX_SHIFT_VAL) Set swVersionARINC429Word BITWISE-OR-equals (msgSubIdx << ARINC429_MSGSUB_IDX_SHIFT_VAL) Set swVersionARINC429Word BITWISE-OR-equals (data << ARINC429_SWVER_DATA_SHIFT_VAL) Increment msg_idx by 1 IF (MAX_MSG_IDX_VALUE equals msg_idx) Set msg_idx to 0

	Increment sys_idx by 1
	IF (MAX_SYS_IDX_VALUE equals sys_idx)
	Set sys_idx equal to 0
	RETURN swVersionARINC429Word

INT1.0101.S.IOP.6.004



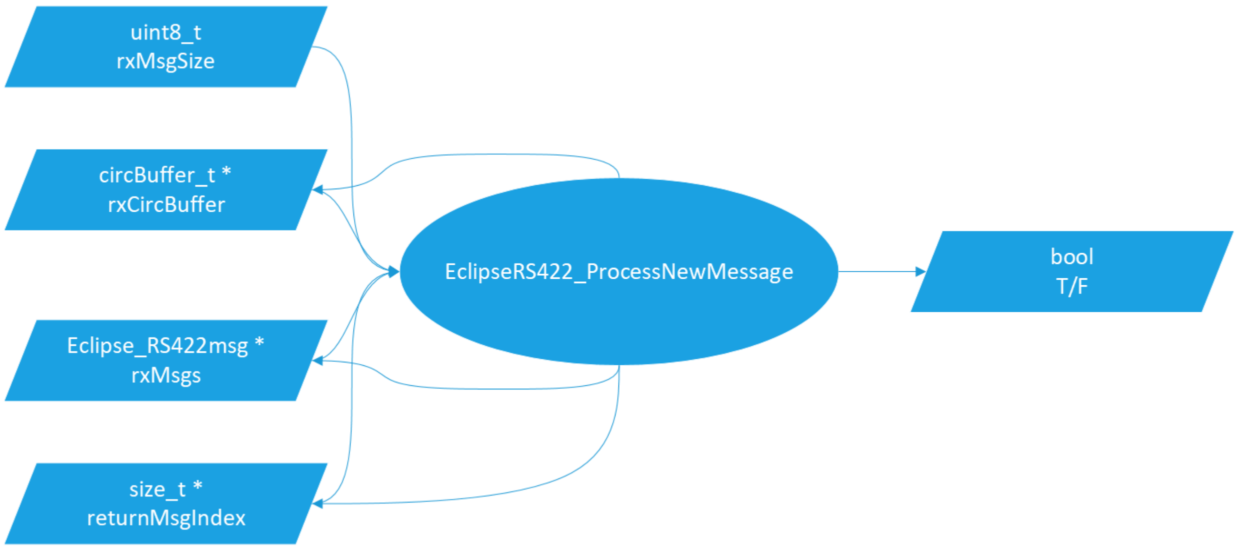
Description	The IOP ***shall*** convert a char value into an ASCII value
Function call:	asciiConverter
Input parameters:	uint8_t **val**
Global data:	None
Function static data:	None

Requirement	IF **val** is less than 10
	RETURN **val** + 48
	ELSE
	RETURN **val** + 55

INT1.0102.S.IOP.1

High level requirement description: The IOP ***shall*** process all RS422 serial to/from ARINC429 communications.

INT1.0102.S.IOP.1.001



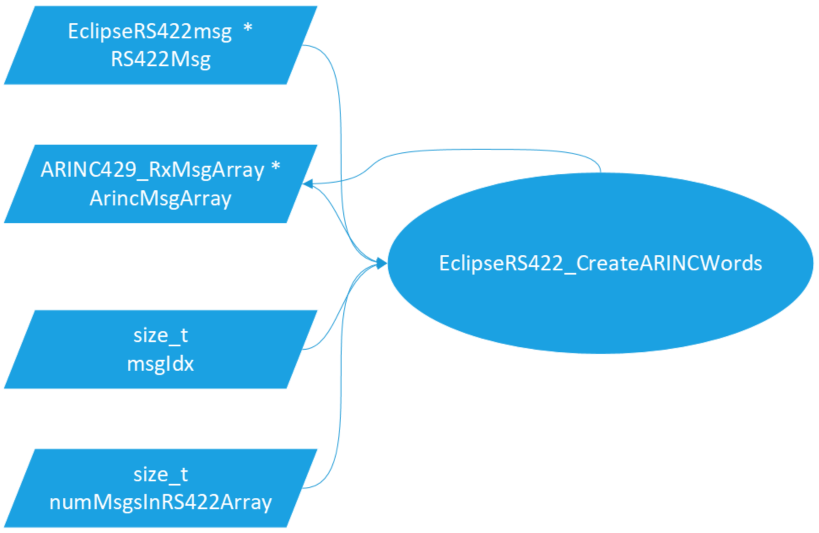
Description	The IOP ***shall*** search a receive circular buffer for a matching Eclipse Aviation RS422 message. If a match is found, flush the received data to a linear buffer.
Function call:	EclipseRS422_ProcessNewMessage
Input parameters:	circBuffer_t ***rxCircBuffer**, uint8_t **rxMsgsSize**, EclipseRS422msg * **rxMsgs**, size_t ***returnMsgIndex**
Global data:	None
Function static data:	None

Requirement	IF NULL equals rxCircBuffer OR NULL equals rxMsgs
	    RETURN
	WHILE true:
	    Set incomingSize to cb_bytesUsed(rxCircBuffer)
	    Set wasMsgStartFound to false
	    Set msgIndex to 0
	    Declare dataByte_index
	    FOR (dataByte_index equals 0, dataByte_index is less than incomingSize, increment dataByte_index by 1)
	        IF (ECLIPSE_RS422_MESSAGE_PREAMBLE equals cb_peek(rxCircBuffer, dataByte_index)
	            IF (dataByte_index + ECLIPSE_RS422_MESSAGE_MIN_TOTAL_LENGTH is greater than incomingSize)
	                BREAK
	            FOR (msgIndex equals 0, msgIndex is less than rxMsgsSize, increment msgIndex by 1)
	                IF
	                    Set wasMsgStartFound equal to true
	                    BREAK

	See pseudocode below

```
        IF ( true equals wasMsgStartFound )
            BREAK
    Call cb_advanceTail ( rxCircBuffer, dataByte_index)
    IF ( false equals wasMsgStartFound)
        BREAK
    Set incomingSize equal to cb_bytesUsed( rxCircBuffer)
    IF ( incomingSize is less than rxMsgs[msgIndex].msgConfig.length +
ECLIPSE_RS422_MESSAGE_LENGTH_HEADER_AND_CRC
        BREAK
    Declare calculatedCRC
    Set totalMsgLength equal to rxMsgs[msgIndex].msgConfig.length +
ECLIPSE_RS422_MESSAGE_LENGTH_HEADER_AND_CRC
    IF ( rxCircBuffer.tail is less than rxCircBuffer.head)
        Set calculatedCRC equal to CRC16_Calculate16bitCRC ( rxCircBuffer.data
+ rxCircBuffer.tail, totalMsgLength, ECLIPSE_RS422_CRC_SEED_VALUE)
    ELSE
        Set bytesToVerify equal to min(totalMsgLength, rxCircBuffer.capacity –
rxCircBuffer.tail )
        Set calculateCRC equal to CRC16_Calculate16bitCRC ( rxCircBuffer.data +
rxCircBuffer.tail, bytesToVerify , ECLIPSE_RS422_CRC_SEED_VALUE)
        Set bytesRemaining to totalMsgLength – bytesToVerify
        IF ( bytesRemaining is greater than 0 )
            Set calculatedCRC equal to
CRC16_Calculate16bitCRC(rxCircBuffer.data, totalMsgLength – bytesToVerify,
calculatedCRC)
    IF ( 0x0 does not equal calculatedCRC)
        call cb_advanceTail(rxCircBuffer, 1)
        RETURN false
    ELSE
        call cb_advanceTail(rxCircBuffer,
ECLIPSE_RS422_MESSAGE_DATA_START_OFFSET)
        IF ( NULL does not equal rxMsgs[msgIndex].data )
            call cb_flushOut( rxCircBuffer, rxMsgs[msgIndex].data,
rxMsgs[msgIndex].msgConfig.length – 1)
        Set rxMsgs[msgIndex].timeStamp_counts equal to 0
        call cb_advanceTail(rxCircBuffer, 2)
        Set the value of the input pointer returnMsgIndex equal to msgIndex
        RETURN true
RETURN false
```

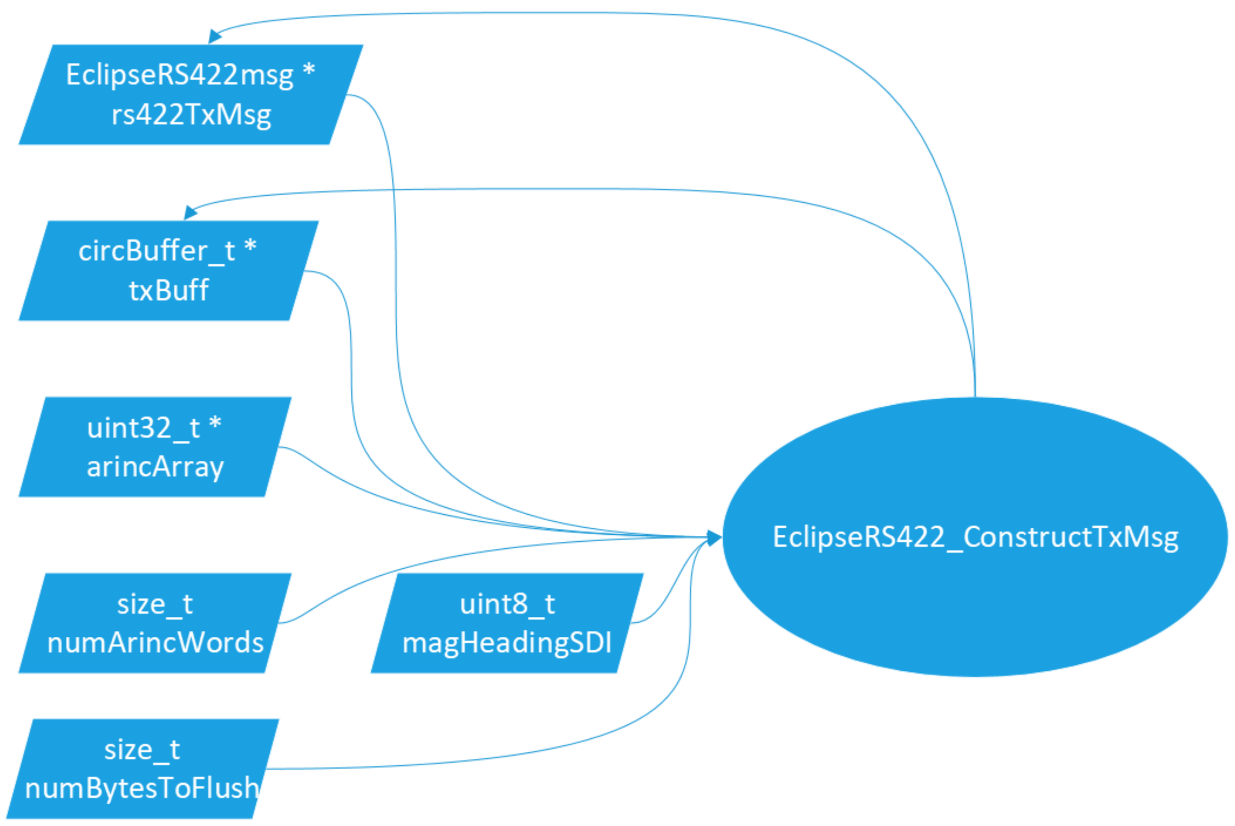
INT1.0102.S.IOP.1.002


EclipseRS422msg * RS422Msg
ARINC429_RxMsgArray * ArincMsgArray
size_t msgIdx
size_t numMsgsInRS422Array
EclipseRS422_CreateARINCWords

Description	The IOP ***shall*** create ARINC429 words from received RS422 serial data.
Function call:	EclipseRS422_CreateARINCWords
Input parameters:	EclipseRS422msg* **RS422Msg**, ARINC429_RxMsgArray * **ArincMsgArray**, size_t **msgIdx**, size_t **numMsgsInRS422Array**
Global data:	None
Function static data:	None

Requirement	IF (NULL equals RS422Msg OR NULL equals ArincMsgArray OR msgIdx is greater than numMsgsInRS422Array)
	    RETURN
	Set data pointer variable to RS422msg[msgIdx].data
	Declare arincWord
	Declare counter
	FOR (counter equals 0, counter less than RS422Msg[msgIdx].msgConfig.length – 1, counter increment)
	    Set arincWord equal to ((data[0]) BITWISE-OR (data[1]) LEFT-SHIFT ARINC_BYTE_ONE_OFFSET) BITWISE OR
	            (data[2]) LEFT-SHIFT ARINC_BYTE_TWO_OFFSET) BITWISE-OR (data[3]) LEFT-SHIFT ARINC_BYTE_THREE_OFFSET)
	      data plus-equals NUM_BYTES_ARINC_MSG;
	      Call ARINC429_ProcessReceivedMessage(ArincMsgArray, arincWord);
	RETURN

INT1.0102.S.IOP.1.003

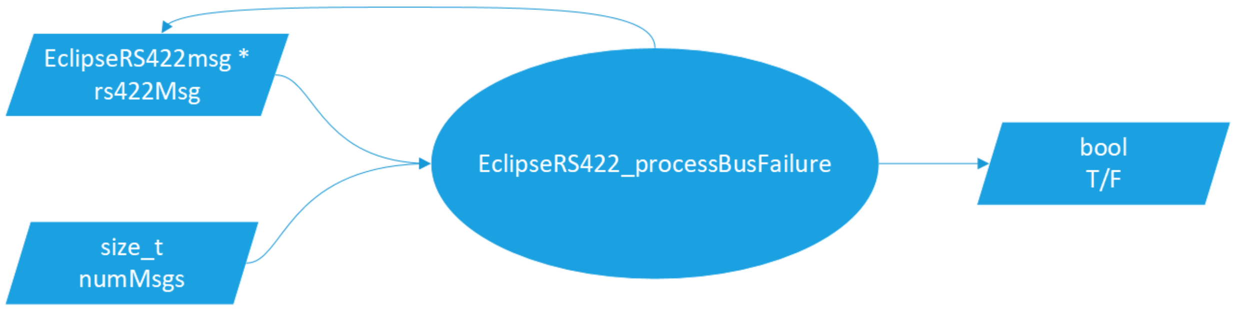


Description	The IOP ***shall*** compose a formatted Eclipse RS422 transmit message based on input configuration
Function call:	EclipseRS422_ConstructTxMsg
Input parameters:	EclipseRS422msg * **rx422TxMsg**, circBuffer_t * **txBuff**, uint32_t * **arincArray**, size_t **numArincWords**, size_t **numBytesToFlush**, uint8_t **magHeadingSDI**
Global data:	None
Function static data:	None
Requirement	IF NULL equals rs422TxMsg OR NULL equals txBuff OR magHeadingSDI is greater than 0x03 OR numBytesToFlush is greater than SIZE_OF_TX_CIRCBUFF_BYTES OR numArincWords is greater than MAX_NUM_ARINC_WORDS_CONSTRUCT_MSG RETURN IF NULL equals arincArray AND 0 does not equal numArincWords RETURN Declare counter Set the value of thisARINCWord equal to arincArray Set tempARINCWord equal to the dereferenced value of thisARINCWord FOR (counter equals 0, counter is less than numArincWords, increment counter by 1) Set tempARINCWord BITWISE-AND-EQUALS 0x7FFFFFFF Set tempARINCWord BITWISE-XOR tempARINCWord RIGHT-SHIFT by 1

	```
	    Set tempARINCWord BITWISE-XOR tempARINCWord RIGHT-SHIFT by 2
	    Set tempARINCWord equal to (tempARINCWord BITWISE-AND 0x11111111) * 0x11111111
	    IF 0 equals ((tempARINCWord RIGHT-SHIFTED by 28) BITWISE-AND 1)
	        Set thisARINCWord BITWISE-OR-equals ARINC_PARITY_SET;
	    Increment thisARINCWord by 1
	    Set tempARINCWord equal to the dereferenced value of thisARINCWord
	Set rs422TxMsg.data[ECLIPSE_RS422_MESSAGE_PREAMBLE_INDEX] to ECLIPSE_RS422_MESSAGE_PREAMBLE
	SWITCH (magHeadingSDI)
	    case (0x01)
	        Set rs422TxMsg.data[ECLIPSE_RS422_MESSAGE_DESTINATION_INDEX] to rs422TxMsg.msgConfig.leftDestination
	        Set rs422TxMsg.data[ECLIPSE_RS422_MESSAGE_SOURCE_INDEX] to rs422TxMsg.msgConfig.leftSource
	        BREAK
	    case (0x10)
	        Set rs422TxMsg->data[ECLIPSE_RS422_MESSAGE_DESTINATION_INDEX] to rs422TxMsg->msgConfig->rightDestination
	    Case (0x03) Set rs422TxMsg->data[ECLIPSE_RS422_MESSAGE_SOURCE_INDEX] to rs422TxMsg->msgConfig->rightSource
	        BREAK
	    default:
	        rs422TxMsg.data[ECLIPSE_RS422_MESSAGE_DESTINATION_INDEX] to rs422TxMsg.msgConfig.leftDestination
	        rs422TxMsg.data[ECLIPSE_RS422_MESSAGE_SOURCE_INDEX] to rs422TxMsg.msgConfig.leftSource
	        BREAK
	Set rs422TxMsg.data[ECLIPSE_RS422_MESSAGE_LENGTH_INDEX] to rs422TxMsg.msgConfig.length
	Set rs422TxMsg.data[ECLIPSE_RS422_MESSAGE_CMD_INDEX] to rs422TxMsg.msgConfig.cmd
	Set the pointer txMsgBuff equal to rs422TxMsg.data + ECLIPSE_RS422_MESSAGE_DATA_START_OFFSET
	Declare arincWordCounter
	FOR (arincWordCounter equals 0, arincWordCounter is less than numArincWords, increment arincWordCounter by 1)
	    Set txMsgBuff[3] to ((arincArray[arincWordCounter] RIGHT-SHIFTED by ARINC_BYTE_THREE_OFFSET) BITWISE-AND LS_BYTE_BITMASK)
	    Set txMsgBuff[2] to ((arincArray[arincWordCounter] RIGHT-SHIFTED by ARINC_BYTE_TWO_OFFSET) BITWISE-AND LS_BYTE_BITMASK)
	    Set txMsgBuff[1] to ((arincArray[arincWordCounter] RIGHT-SHIFTED by ARINC_BYTE_ONE_OFFSET) BITWISE-AND LS_BYTE_BITMASK)
	    Set txMsgBuff[0] to (arincArray[arincWordCounter] BITWISE-AND LS_BYTE_BITMASK)
	```

	Set txMsgBuff to plus-equals NUM_BYTES_ARINC_MSG; Set crc = CRC16_Calculate16bitCRC(rs422TxMsg.data, txMsgBuff - rs422TxMsg.data, ECLIPSE_RS422_CRC_SEED_VALUE) Set txMsgBuff[0] equal to ((crc RIGHT-SHIFT by NUM_BITS_IN_BYTE) BITWISE-AND LS_BYTE_BITMASK) Set txMsgBuff[1] equal to crc BITWISE-AND LS_BYTE_BITMASK Call cb_flushIn(txBuff, rs422TxMsg.data, numBytesToFlush) RETURN

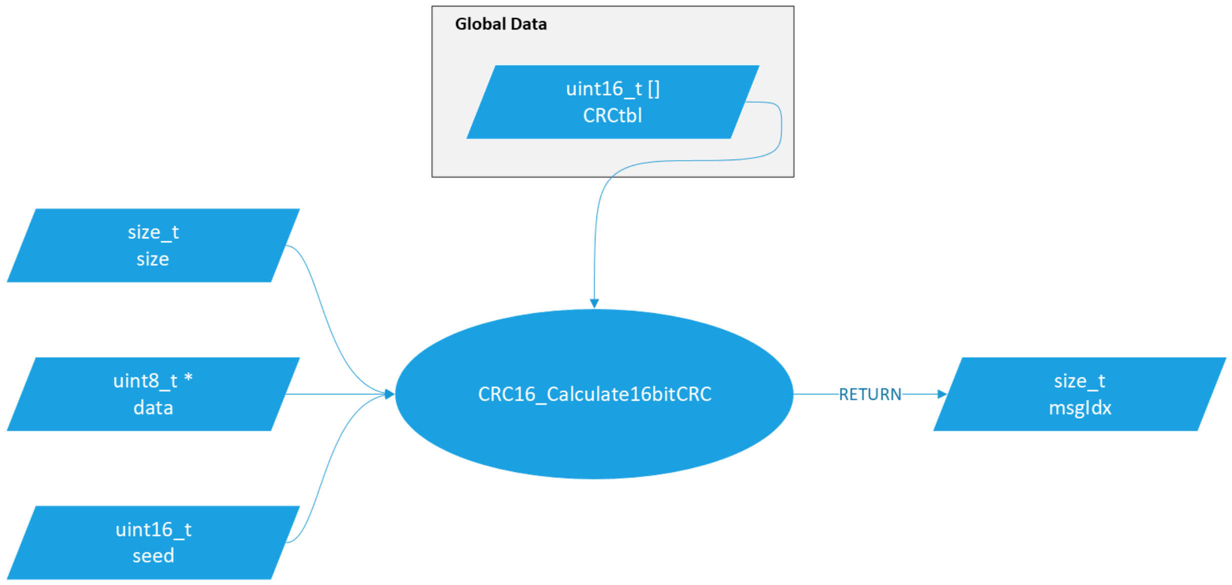
INT1.0102.S.IOP.1.004



Description	The IOP ***shall*** return a true/false status representing the bus failure status of an RS422 bus
Function call:	EclipseRS422_processBusFailure
Input parameters:	EclipseRS422msg * **rs422Msg**, size_t **numMsgs**
Global data:	None
Function static data:	None

Requirement	IF NULL equals RS422Msg RETURN true
	Declare msgIdx
	Set numInvalidMsgs to 0
	FOR (msgIdx equals 0, msgIdx is LESS THAN numMsgs, msgIdx increments by 1)
	    RS422Msg[msgIdx].timeStamp_counts increment by 1
	    IF (RS422Msg[msgIdx].timeStamp_counts is greater than or equal to RS422Msg[msgIdx].timeStamp_max_counts)
	        Increment numInvalidMsgs by 1
	IF numInvalidMsgs equals numMsgs
	    RETURN true
	ELSE
	    RETURN false

INT1.0102.S.IOP.1.005

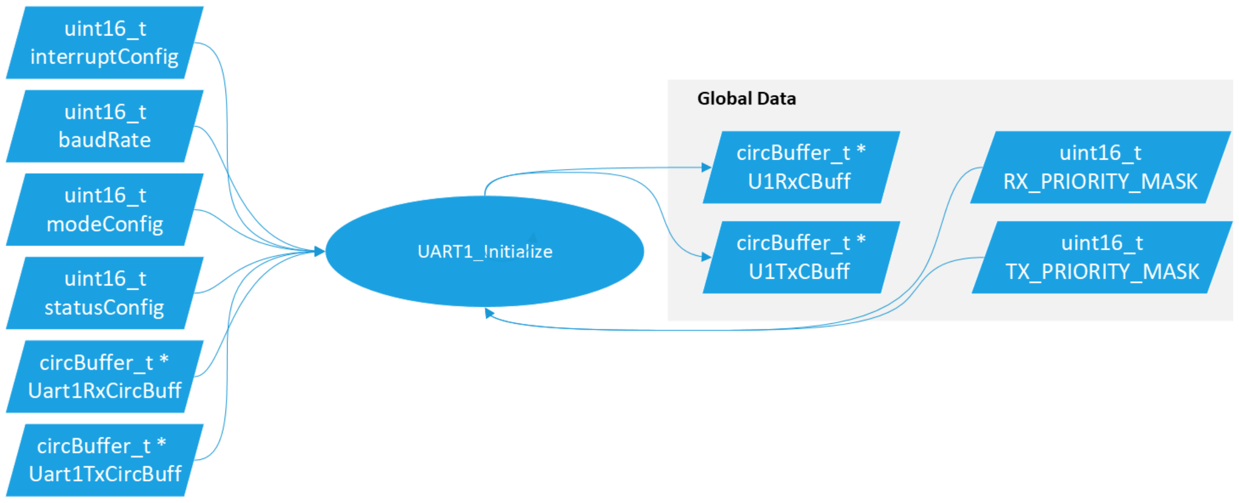


Description	The IOP shall calculate the 16-bit CRC of all incoming and outgoing RS422 messages.
Function call:	CRC16_Calculate16bitCRC
Input parameters:	uint8_t * **data**, size_t **size**, uint16_t **seed**
Global data:	uint16_t CRCtbl[]
Function static data:	None
Requirement	Declare **i** Set **crc** equal to **seed** FOR i equal 0, i is less than size, increment i by 1 Set **crc** equal to (**crc** LEFT-SHIFT 8) BITWISE_OR CRCtbl[((**crc** RIGHT-SHIFT 8) BITWISE-OR data[**i**]) BITWISE-AND 0x00FF] RETURN **crc**

INT1.0102.S.IOP.2

High-level requirement description: The IOP ***shall*** use the UART1 peripheral to transfer serial data to/from the ADC subsystem

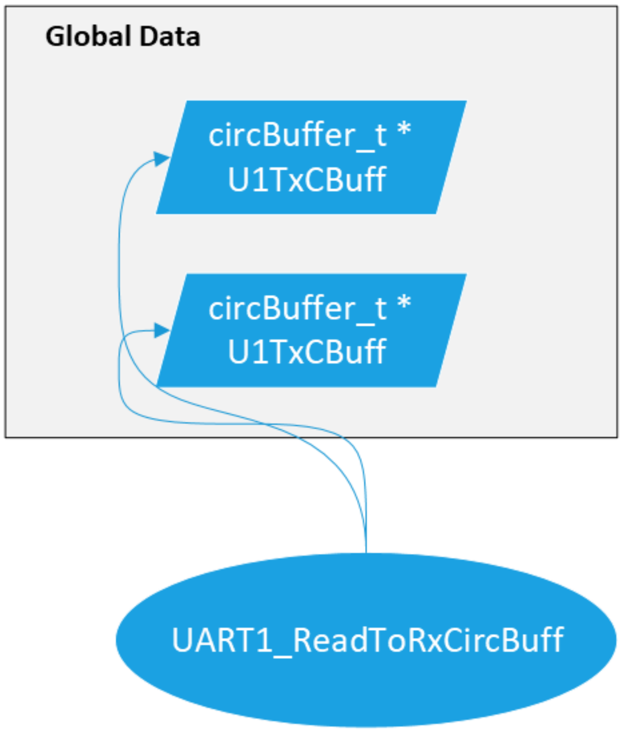
INT1.0102.S.IOP.2.001



Description	The IOP ***shall*** initialize the UART1 peripheral with inputs from configuration data
Function call:	UART1_Initialize
Input parameters:	circBuffer_t * **Uart1RxCircBuff**, circBuffer_t * **Uart2TxCircBuff** ,uint16_t **interruptConfig**, uint16_t **modeConfig**, uint16_t **statusConfig**, uint16_t **baudRate**
Global data:	circBuffer_t * **U1RxCBuff**, circBuffer_t * **U1TxCBuff**, uint16_t **RX_PRIORITY_MASK**, uint16_t **TX_PRIORITY_MASK**
Function static data:	None
Requirement	Set **TRISFbits.TRISF3** to 0 Set **TRISFbits.TRISF2** to 1 Set **U1RxCBuff** equal to input pointer **uart1RxCircBuff** Set **U1TxCBuff** equal to input pointer **uart1TxCircBuff** Call cb_reset(**U1RxCBuff**) Call cb_reset(**U1TxCBuff**) Set **IFS0bits.U1RXIF** to 0 Set **IFS0bits.U1TXIF** to 0 Set **IPC2bits.U1RXIP** to (RX_PRIORITY_MASK bitmask AND **interruptConfig**) Set **IPC2bits.U1TXIP** to (RX_PRIORITY_MASK bitmask AND **interruptConfig**) RIGHTSHIFT 4 Set **U1BRG** to input parameter **baudRate** Set **U1MODE** to input parameter **modeConfig** Set **U1STA** to input parameter **statusConfig** Set **U1STAbits.URXISEL** to 3

	Set **IEC0bits.U1RXIE** to 1 Set **IEC0bits.U1TXIE** to 1 RETURN

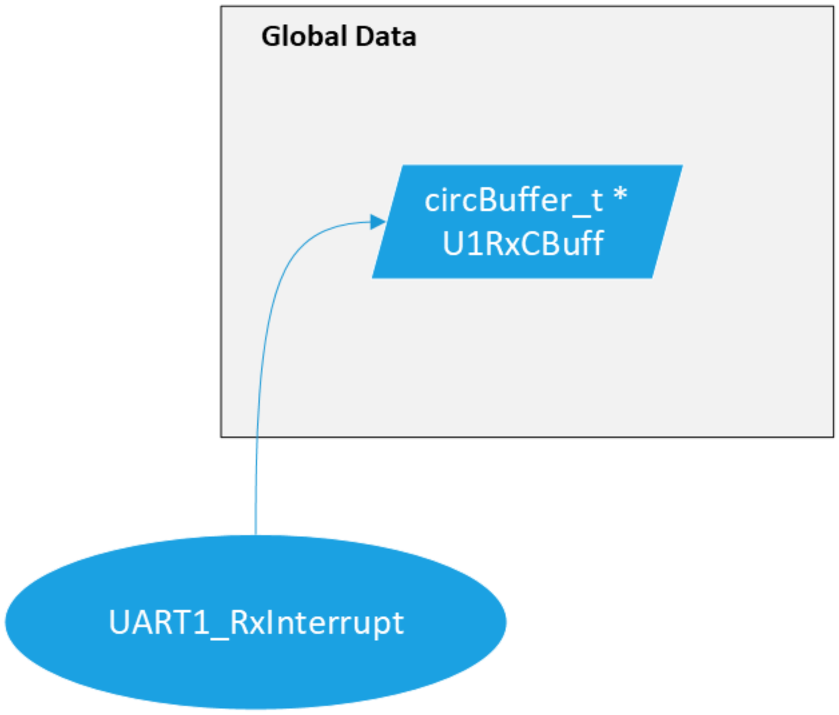
INT1.0102.S.IOP.2.002



Description	The IOP ***shall*** read data to the UART1 receive circular buffer by manually tripping the UART1 Rx interrupt flag
Function call:	UART1_ReadToRxCircBuff
Input parameters:	None
Global data:	circBuffer_t * **U1RxCBuff**, circBuffer_t * **U1TxCBuff**
Function static data:	None

Requirement	IF (**U1RxCBuff.head** > **U1RxCBuff.capacity** OR **U1RxCBuff.tail** > **U1RxCBuff.capacity**)
	    Call cb_reset(**U1RxCBuff**)
	ELSE
	    Set **ISF0bits.U1RXIF** to 1
	RETURN

INT1.0102.S.IOP.2.003

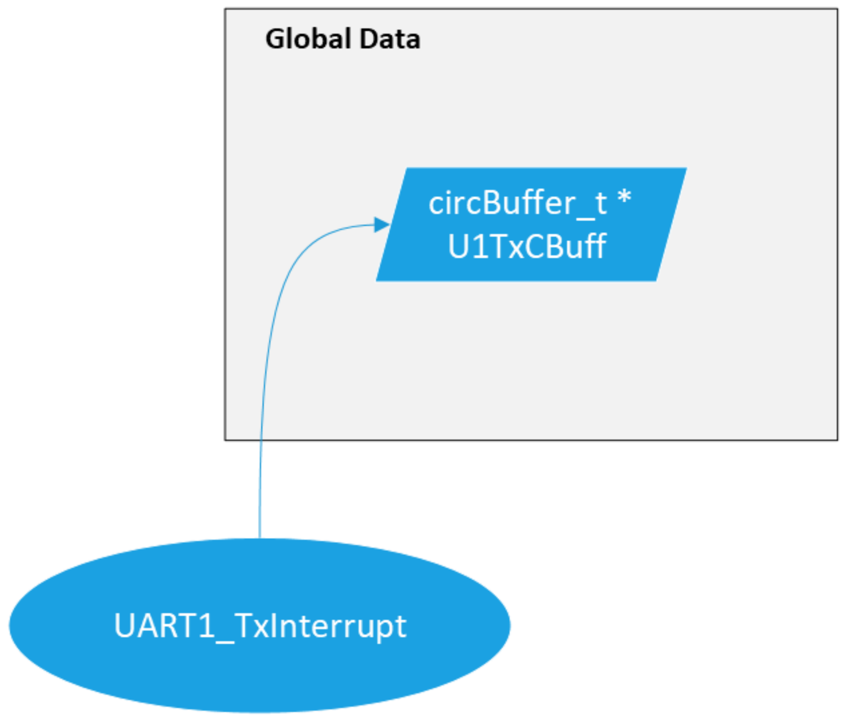


Description	The IOP ***shall*** push received data from the UART1 rx peripheral to the receive circular buffer.
Function call:	UART1_RxInterrupt
Input parameters:	None
Global data:	circBuffer_t * **U1RxCBuff**
Function static data:	None
Requirement	WHILE (**U1STAbits.URXDA** equals 1) call cb_push(**U1RxCBuff**, **U1RXREG**) Set **ISF0bits.U1RXIF** to 0 Set **U1STAbits.OERR** to 0 RETURN

Description	The IOP ***shall*** manually trip the UART1 Transmit interrupt flag
Function call:	UART1_TxStart
Input parameters:	None
Global data:	None
Function static data:	None

Requirement	IF 1 does not equal **IEC0bits.U1TXIE**:
	    Set **IEC0bits.U1TXIE** to 1
	Return

INT1.0102.S.IOP.2.005



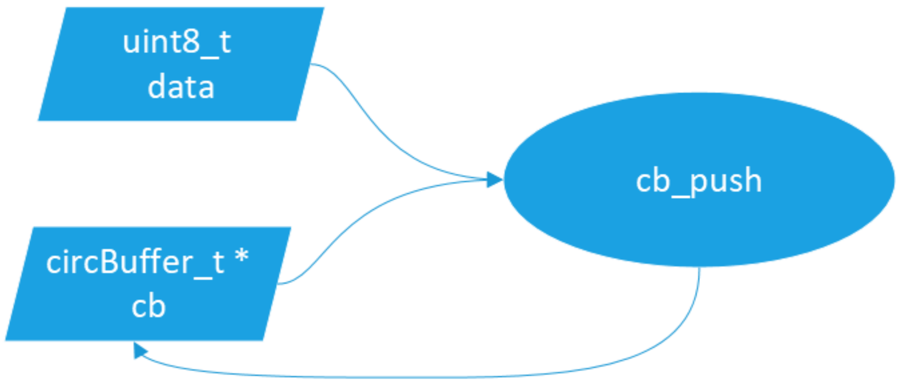
Description	The IOP ***shall*** service the transmit interrupt signal from UART1
Function call:	UART1_TxInterrupt
Input parameters:	None
Global data:	circBuffer_t * **U1TxCBuff**
Function static data:	None

Requirement	IF **U1TxCBuff.tail** >= **U1TxCBuff.head**
	    Set **IECObits.U1TXIE** to 0
	ELSE
	    Set **IFS0bits.U1TXIF** to 0
	    WHILE (**U1STAbits.UTXBF** equals 0 AND **U1TxCBuff.tail** does not equal **U1TxCBuff.head**)
	        U1TXREG = cb_pop (**U1TxCBuff**)
	RETURN

INT1.0102.S.IOP.3

High level requirement description: The AFC004 ***shall*** store received and transmit serial data in circular buffers.

INT1.0102.S.IOP.3.001

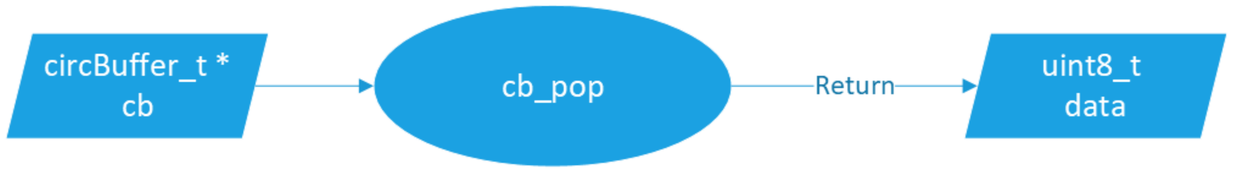


Description	The IOP ***shall*** push data to a circular buffer.
Function call:	cb_push
Input parameters:	circBuffer_t * **cb**, uint8_t **data**
Global data:	None
Function static data:	None

Requirement	Set **offset** to **cb.head** + 1
	IF **offset** is greater than or equal to **cb.capacity**
	    Set **offset** to 0
	IF **offset** does not equal **cb.tail**
	    Set **cb.data[cb.head]** to **data**
	    Set **cb.head** to **offset**
	RETURN

INT1.0102.S.IOP.3.002

Description: The IOP ***shall*** pop data from a circular buffer
Function call: cb_pop
Input parameters: circBuffer_t * cb
Global Data: None

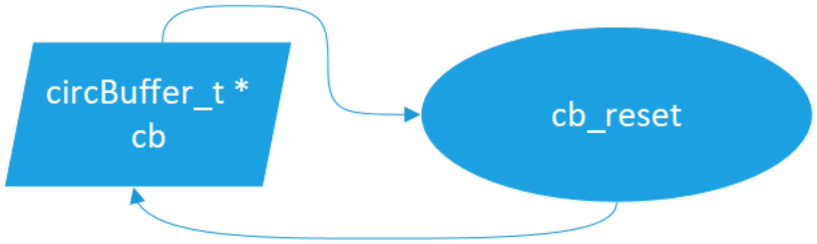


Description	The IOP ***shall*** pop data from a circular buffer
Function call:	cb_pop
Input parameters:	circBuffer_t * cb
Global data:	None
Function static data:	None

Requirement	Declare **data**
	IF **cb**.**head** does not equal **cb**.**tail**
	    Set **data** to **cb**.**data**[**cb**.**tail**]
	    Increment **cb**.**tail** by 1
	    IF **cb**.**tail** is greater than or equal to **cb**.**capacity**
	        Decrement **cb**.**tail** by **cb**.**capacity**
	ELSE Set **data** to 0
	RETURN **data**

INT1.0102.S.IOP.3.003

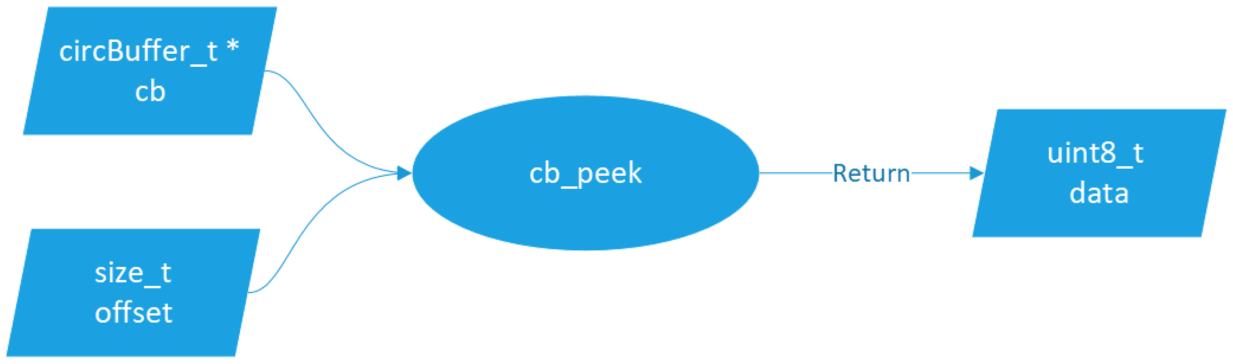
Description: The IOP ***shall*** reset a circular buffer.
Function call: cb_reset
Input Parameters: circBuffer_t * cb
Global Data: None



Description	The IOP ***shall*** reset a circular buffer's head and tail to zero.
Function call:	cb_reset
Input parameters:	circBuffer_t * cb
Global data:	None
Function static data:	None
Requirement	Set **cb.tail** to 0 Set **cb.head** to 0 RETURN

INT1.0102.S.IOP.3.004

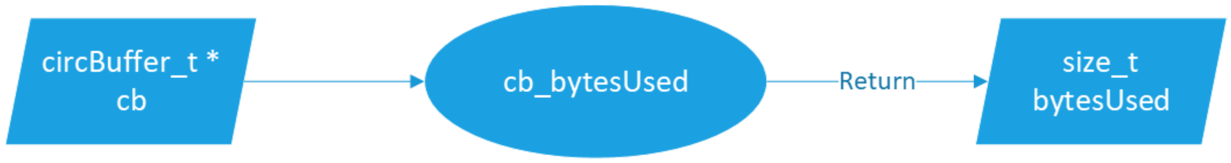
Description: The IOP ***shall*** peek at a circular buffer and return the value at the input offset



Description	The IOP ***shall*** return the value circular buffer's data at the input offset value
Function call:	cb_peek
Input parameters:	size_t **offset**, circBuffer_t * **cb**
Global data:	None
Function static data:	None

Requirement	Set **index** equal to **cb.tail** + **offset**
	IF **index** is greater than or equal to **cb.capacity**
	    Decrement **index** by **cb.capacity**
	RETURN **cb.data[index]**

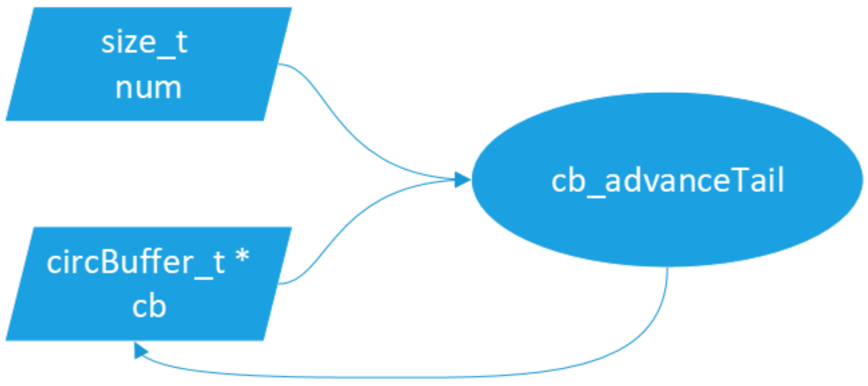
INT1.0102.S.IOP.3.005



Description	The IOP ***<u>shall</u>*** return the total number of bytes occupied by a circular buffer
Function call:	cb_bytesUsed
Input parameters:	circBuffer_t * cb
Global data:	None
Function static data:	None

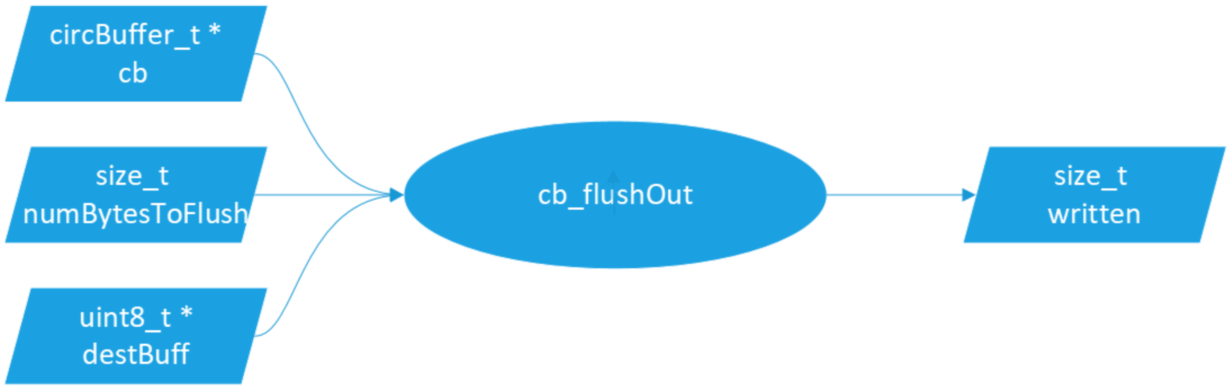
Requirement	IF **cb.head** is greater than or equal to **cb.tail**
	    RETURN **cb.head – cb.tail**
	ELSE
	    RETURN **cb.capacity** – (**cb.tail** – **cb.head**)

INT1.0102.S.IOP.3.006



Description	The IOP ***shall*** advance the circular buffer tail by a specified amount of indices
Function call:	cb_advanceTail
Input parameters:	circBuffer_t * cb, size_t num
Global data:	None
Function static data:	None
Requirement	Declare **maxAdvance** Declare **advance** IF cb.tail is greater than cb.head Set maxAdvance equal to cb.head + cb.capacity – cb.tail ELSE Set maxAdvance equal to cb.head – cb.tail Set advance equal to min(num, maxAdvance) IF (cb.tail is greater than or equal to cb.capacity Decrement cb.tail by cb.capacity RETURN

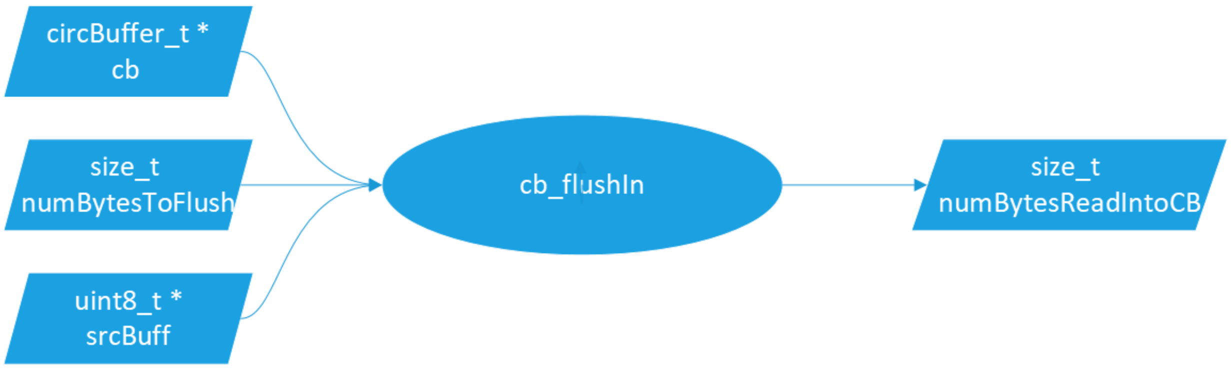
INT1.0102.S.IOP.3.007



Description	The IOP ***shall*** flush data out of a circular buffer to a destination linear source buffer.
Function call:	cb_flushOut
Input parameters:	circBuffer_t * cb, uint8_t * destBuff, size_t numBytesToFlush
Global data:	None
Function static data:	None
Requirement	IF (NULL equals cb OR NULL equals cb.data) RETURN 0 IF (cb.head is greater than cb.capacity OR cb.tail is greater than cb.capacity) Call cb_reset(cb) RETURN 0 Set sz equal to cb_bytesUsed(cb) IF (0 equals sz) RETURN 0 Set contiguous to min(cb.capacity – cb.tail, sz) Set wrapped to sz-contiguous Set cbAddr to cb.data + cb.tail Set maxCount to min(contiguous, numBytesToFlush) FOR (count equals 0, count is less than maxCount, count increments by 1) Set destBuff[count] equal to cbAddr[count] Set written to maxCount call cb_advanceTail(cb, maxCount) IF (written does not equal contiguous OR wrapped equals 0) RETURN written Set maxCount to min(wrapped, numBytesToFlush – contiguous) Set cbAddr to cb.data FOR (count equals 0, count is less than maxCount, count increments Set destBuff[count + contiguous] to cbAddr[count] written plus-equals maxCount call cb_advanceTail(cb, maxCount)

	RETURN written

INT1.0102.S.IOP.3.008



Description	The IOP ***shall*** flush data from a linear source buffer into a circular buffer
Function call:	cb_flushIn
Input parameters:	circBuffer_t * cb, uint8_t srcBuff, size_t numBytesToFlush
Global data:	None
Function static data:	None

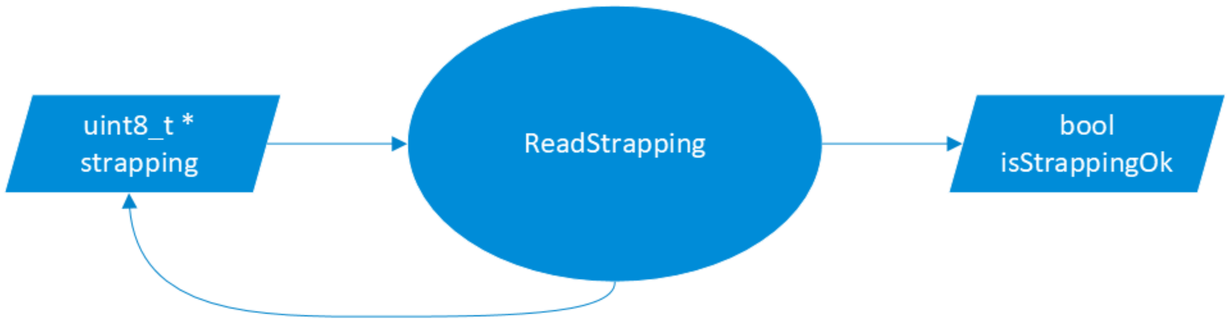
Requirement	
	```
	IF (NULL equals cb OR NULL equals cb.data)
	    RETURN 0
	IF ( cb.head is greater than cb.capacity OR cb.tail is greater than cb.capacity)
	    Call cb_reset(cb)
	    RETURN 0
	declare numBytesReadIntoCB
	IF ( cb.head is less than cb.tail )
	    Set numBytesReadIntoCB equal tomin(cb.tail -cb.heal – 1, numBytesToFlush)
	    Set cbAddr equal to cb.data + cb.head
	    FOR (idx equals 0, idx is less than numBytesReadIntoCB, idx increment)
	        Set cbAddr[idx] equal to srcBuff[idx]
	    Set cb.head to plus-equals numBytesReadIntoCB
	ELSE
	    Set blockSize equal to cb.capacity -cb.head
	    IF ( 0 equals cb.tail)
	        decrement blockSize by 1
	    Set numBytesReadIntoCB equal to min(blockSize, numBytesToFlush)
	    Set cbAddr equal to cb.data + cb.head
	    FOR (idx equals 0, idx less than numBytesReadIntoCB, idx increment)
	        Set cbAddr[idx] equal to srcBuff[idx]
	    cb.head plus-equals numBytesReadIntoCB
	    IF (cb.head is greater-than or equal-to cb.capacity)
	        Set cb.head to 0
	        IF (cb.tail does not equal 0)
	            Set numBytesToRead2ndStep equal to min(cb.tail – 1, numBytesToFlush – numBytesReadIntoCB)
	            Set cbAddr equal to cb.data
	```

	FOR (idx equals 0, idx less than numBytesToRead2ndStep, idx incremenet) Set cbAddr[idx] equal to srcBuff[idx + numBytesReadIntoCB] Set numBytesReadIntoCB to plus-equals numBytesToRead2ndStep Set cb.head to plus-equals numBytesToRead2ndStep RETURN numBytesReadIntoCB

INT1.0103.S.IOP.1

High level requirement description: The IOP ***shall*** enter the desired operating mode based on strapping input configuration.

5.12.6.3 INT1.0103.S.IOP.1.001



Description	The IOP ***shall*** read discrete strapping inputs at startup and return the resultant strapping value.
Function call:	ReadStrapping
Input parameters:	uint8_t * strapping
Global data:	None
Function static data:	None
Requirement	Set numReads equal to 10 Set *strapping equal to 0 Set strap1 equal to STRAP1_Get Set strap2 equal to STRAP2_Get Set strap3 equal to STRAP3_Get Set strapParity equal to STRAP_PARITY_Get Declare index Set isStrappingOK equal to true FOR (index equals 1; index is less than numReads; increment index by 1) Call Timer23_Delay_ms(10) IF STRAP1_Get does not equal strap1 OR STRAP2_Get does not equal strap2 OR STRAP3_Get does not equal strap3 OR STRAP_PARITY_Get does not equal strapParity Set isStrappingOK equal to false;

	```
	    break
	IF true equals isStrappingOK
	        IF 0x1 equals (strap1 + strap2 + strap3 + strapParity) BITWISE-AND 0x1
	  Set *strapping equal to (strap1 LEFT-SHIFT by 2) + (strap2 LEFT-SHIFT by 1) + strap3
	        ELSE
	  Set isStrappingOK equal to false
	RETURN isStrappingOK
	```

Power Requirements

High level requirement description: The IOP shall enter the desired operating mode based on strapping input configuration.

PCS.0402.S.IOP.2.001

Description	The IOP shall initialize the IOP's unused pins to a deterministic state.
Function call:	ConfigureUnusedPinsAsOutputs
Input parameters:	None
Global data:	None
Function static data:	None
Requirement	TRISBbits.TRISB2 equal to 0 LATBbits.LATB2 equal to 0 TRISBbits.TRISB3 equal to 0 LATBbits.LATB3 equal to 0 TRISBbits.TRISB4 equal to 0 LATBbits.LATB4 equal to 0 TRISBbits.TRISB5 equal to 0 LATBbits.LATB5 equal to 0 TRISBbits.TRISB15 equal to 0 LATBbits.LATB15 equal to 0 TRISDbits.TRISD15 equal to 0 LATDbits.LATD15 equal to 0 TRISGbits.TRISG12 equal to 0 LATGbits.LATG12 equal to 0 TRISGbits.TRISG13 equal to 0 LATGbits.LATG13 equal to 0 TRISAbits.TRISA6 equal to 0 LATAbits.LATA6 equal to 0 TRISDbits.TRISD5 equal to 0 LATDbits.LATD5 equal to 0 TRISDbits.TRISD7 equal to 0 LATDbits.LATD7 equal to 0 TRISDbits.TRISD1 equal to 0 LATDbits.LATD1 equal to 0 TRISCbits.TRISC14 equal to 0 LATCbits.LATC14 equal to 0 TRISFbits.TRISF6 equal to 0 LATFbits.LATF6 equal to 0 TRISFbits.TRISF7 equal to 0 LATFbits.LATF7 equal to 0 TRISFbits.TRISF8 equal to 0 LATFbits.LATF8 equal to 0

Derived IOP Requirements

These derived requirements were reused from SDD-1-AHR150A-K section 4.4.1.

Derived Req. #	Derived Requirement Description
INT1.0102.S.IMU.2.001.D04	The IMU ***shall*** do following math functions: 1. Single-precision floating-point addition operation. 2. Single-precision floating-point subtraction operation. 3. Single-precision, floating-point round utility. 4. Single-precision, floating-point pack utility. 5. Single-precision, floating-point unpack utility 6. Single-precision floating-point division operation. 7. Compare two floating point numbers. 8. Convert floating-point to 32-bit, unsigned integer. 9. Convert 32-bit, signed integer to floating-point. 10. Unpack two floating-point operands. 11. Propagate a NaN, return a quiet NaN, common exit from floating-point operations 12. Single-precision multiplication operation.

APPENDIX A: List of Modules

Newly developed modules

Source File (.c)	Version	Header File (.h)	Version
main.c	17	N/A	
configBits.c	15	N/A	
AFC004MessageConfig.c	16	N/A	
IOPConfig.c	16	IOPConfig.h	15
maintenanceMode.c	17	maintenanceMode.h	15
SoftwareVersion.c	15	SoftwareVersion.h	15
Timer23.c	15	Timer23.h	15
CRC16bit.c	15	CRC16bit.h	15
EclipseRS422messages.c	16	EclipseRS422messages.h	15
ARINC.c	15	ARINC.h	15
ARINC_common.c	15	ARINC_common.h	15
ARINC_HI3584.c	16	ARINC_HI3584.h	15
ArincDownload.c	15	ArincDownload.h	15
		ARINC_typedefs.h	16
calculateNewARINCLabels.c	15	calculateNewARINCLabels.h	15
COMIIRDifferentiator.c	15	COMIIRDifferentiator.h	15

Reused Modules

Source File (.c)	Version	Header File (.h)	Version
COMTrigModule.c	6360	COMTrigModule.h	6121
COMSystemTimer.c	6008	COMSystemTimer.h	6008
COMIIRFilter.c	6024	COMIIRFilter.h	6008
COMReadProgramMemory.c	6187	COMReadProgramMemory.h	6008
COMCRCModule.c	6284	COMCRCModule.h	6008
COMHardwareResetConfiguration	6061	COMHardwareResetConfiguration*	6017
COMdsPICunusedISRs.c	6413	COMdsPICunusedISRs.h*	6008
COMDSPicNonVolatileMemRead.s	6187	COMDSPicNonVolatileMemRead.h	6187
COMfpack.s	6298		
COMfunpack.s	5889		
COMRAMTest.s	6023	COMRAMTest.h	6008
COMVerifyNonVolatileMemoryCRC.c	6322	COMVerifyNonVolatileMemoryCRC.h	6187
		COMDefines.h	7717
		COMTypedefs.h	6017
crt0.s	6176		
divsf3.s	5889		
fcompare.s	5889		
feqltle.s	6146		
fgtge.s	5889		
fixsfsi.s	6360		
fisunssfsi.s	5889		
floatsisf.s	5889		

fne.s	5889		
funpack2.s	5889		
futils.s	5889		
mulsf3.s	5889		
addsf3.s	5889		
libm.inc	5889		
COMlibm.inc	5889		
COMDSPicNonVolatileMemRead.inc	6008		

APPENDIX B: Microcontroller pin assignments

All unused pins are configured as digital outputs. ARINC429 DB pins are configured as input or output, based on the desired functionality.

Pin	Direction	Function	Pin	Direction	Function
1	OUTPUT	FAULT	41	OUTPUT	Mx Tx
2	I/O	A429 DB0	42	INPUT	Mx Rx
3	I/O	A429 DB1	43	OUTPUT	UNUSED
4	I/O	A429 DB2	44	OUTPUT	UNUSED
5	I/O	A429 DB3	45	OUTPUT	UNUSED
6	INPUT	STRAP 1	46	OUTPUT	A429A EN1
7	INPUT	STRAP 2	47	OUTPUT	A429A EN2
8	INPUT	STRAP 3	48	---	3.3V
9	INPUT	MCLR	49	INPUT	CLK +
10	INPUT	STRAP P	50	OUTPUT	CLK -
11	---	GND	51	---	GND
12	---	3.3V	52	OUTPUT	A429A PL1
13	I/O	A429 DB4	53	OUTPUT	A429A PL2
14	I/O	A429 DB5	54	OUTPUT	A429A ENTX
15	OUTPUT	UNUSED	55	OUTPUT	A429A CWSTR
16	OUTPUT	UNUSED	56	OUTPUT	A429A RSR
17	OUTPUT	UNUSED	57	OUTPUT	A429B SEL
18	OUTPUT	UNUSED	58	OUTPUT	A429A TX Slope
19	INPUT	ISCPC	59	OUTPUT	A429B TX Slope
20	I/O	ICSPD	60	OUTPUT	UNUSED
21	I/O	A429 DB6	61	OUTPUT	UNUSED
22	I/O	A429 DB7	62	INPUT	A429A DR2
23	I/O	A429 DB8	63	OUTPUT	A429B EN1
24	I/O	A429 DB9	64	OUTPUT	A429B EN2
25	---	3.3V	65	INPUT	A429A FFT
26	---	GND	66	INPUT	A429B DR1
27	I/O	A429 DB10	67	OUTPUT	UNUSED
28	I/O	A429 DB11	68	INPUT	A429B DR2
29	I/O	A429 DB12	69	OUTPUT	UNUSED
30	I/O	A429 DB13	70	---	GND
31	---	GND	71	---	3.3V
32	---	3.3 V	72	OUTPUT	A429B PL1
33	I/O	A429 DB14	73	OUTPUT	A429B PL2
34	I/O	A429 DB15	74	OUTPUT	A429B ENTX
35	OUTPUT	A429 A SEL	75	OUTPUT	A429B CWSTR
36	OUTPUT	UNUSED	76	OUTPUT	UNUSED
37	OUTPUT	UNUSED	77	INPUT	A429B FFT
38	INPUT	A429 A DR1	78	OUTPUT	A429B RSR
39	INPUT	RS422 Rx	79	OUTPUT	UNUSED
40	OUTPUT	RS422 Tx	80	OUTPUT	UNUSED

APPENDIX C: Project specific settings

Item	Setting
MPLABX IDE Version	5.5
Microcontroller	dsPIC30F6014A
Compiler Toolchain	XC16 v1.36
Debug Tool	ICD3
Optimization level	0
Device family pack	dsPIC30F_DFP 1.4.104

6