1. ARINC 429 is a data communication protocol used primarily in the avionics industry for the transfer of digital data between different systems on commercial aircraft.
2. It is a key part of the ARINC family of standards, developed by the Airlines Electronic Engineering Committee (AEEC) to standardize the avionics interfaces used by various aircraft manufacturers.

Key Features of ARINC 429

1. **Simplex Communication**:
   - **One-way communication**: ARINC 429 is designed for simplex communication, meaning data is transmitted in one direction only from a transmitter to one or more receivers.
   - Each ARINC 429 bus is unidirectional, so a system can either transmit or receive, but not both on the same bus.
2. **Two-wire Differential Signaling**:
   - Uses a twisted pair of wires for differential signaling, which reduces susceptibility to electromagnetic interference (EMI) and allows for reliable data transmission over long distances.
   - The signaling method uses a balanced differential pair, meaning that the signal on one wire is the inverse of the signal on the other, and the receiver interprets the difference.
3. **Fixed Data Word Length**:
   - ARINC 429 transmits data in a fixed 32-bit word format.
   - The 32-bit data word is divided into different fields: Label, Source/Destination Identifier (SDI), Data, Sign/Status Matrix (SSM), and Parity.
4. **Data Word Structure**:
   - **Label (8 bits)**: Identifies the type of data being transmitted and is the first 8 bits of the 32-bit

word. Labels are typically represented in octal form.
    - **SDI (2 bits)**: Source/Destination Identifier, used to identify the source or destination of the data.
    - **Data (19 bits)**: The main content of the message, carrying the actual data to be transmitted.
    - **SSM (2 bits)**: Sign/Status Matrix, providing additional information about the data (e.g., sign, operational status).
    - **Parity (1 bit)**: The last bit is a parity bit used for error checking (Odd parity).
5. **Transmission Speed**:
    - ARINC 429 supports two standard data transmission rates:
        - **Low speed**: 12.5 kbps
        - **High speed**: 100 kbps
    - The data rate is selected based on the requirements of the specific avionics system.
6. **Self-clocking**:
    - ARINC 429 is self-clocking, meaning the timing information is encoded within the data stream, allowing the receiver to extract timing information directly from the received data without requiring a separate clock signal.
7. **Error Detection**:
    - ARINC 429 uses a simple parity bit for error detection. The parity bit ensures that the total number of 1s in the 32-bit word is odd, which helps in detecting single-bit errors.
    - Additional error detection mechanisms, such as reasonableness checking, may be implemented at the application level.
8. **Multiple Receivers**:

- Although ARINC 429 is a simplex protocol, a single transmitter can send data to up to 20 receivers on the same bus. Each receiver can independently receive and process the data.

9. **Low Complexity**:
    - The protocol is designed to be straightforward and easy to implement, which is critical in avionics where reliability and ease of use are paramount.

Functional Characteristics of ARINC 429

1. **Data Transmission:**
    - ARINC 429 transmits data asynchronously in a serial bit stream. Each 32-bit word is sent sequentially, with gaps between words that are at least 4-bit times wide.
    - There are no start or stop bits; the 32-bit word is transmitted as a continuous stream.
2. **Synchronization**:
    - Receivers synchronize on the incoming data stream by identifying the gaps between the 32-bit words. These gaps help the receiver determine where each data word begins and ends.
3. **Label Identification**:
    - The label field in the 32-bit word identifies the type of data being transmitted. For example, a specific label might indicate airspeed, while another label could indicate altitude.
    - Labels are typically defined in octal format, and each type of data has a standardized label to ensure consistency across different systems.
4. **Data Interpretation**:
    - The data field within the 32-bit word can represent various types of information, including numeric values, discrete status bits, or binary-coded decimal (BCD) numbers.

- Receivers interpret the data based on the label and SDI fields, which tell them how to decode and use the information.

5. **System Interoperability**:
   - ARINC 429 ensures interoperability between different avionics systems and components from various manufacturers. By adhering to the standard, systems can reliably communicate with each other, regardless of their origin.
6. **Prioritization and Filtering**:
   - Since ARINC 429 is a simplex protocol, there is no built-in mechanism for prioritizing messages or dealing with bus contention.
   - However, receivers can filter incoming messages based on the label and SDI fields to prioritize or ignore specific types of data.

Applications of ARINC 429

1. **Flight Control Systems**:
   - ARINC 429 is widely used in flight control systems to transmit critical data such as altitude, airspeed, heading, and attitude between sensors, computers, and display systems.
2. **Navigation Systems**:
   - Data from navigation systems, such as GPS and inertial reference systems, is transmitted using ARINC 429 to ensure that all avionics systems have accurate positional information.
3. **Engine Monitoring**:
   - Engine parameters such as fuel flow, temperature, and pressure are transmitted over ARINC 429 for monitoring and control purposes.
4. **Cockpit Display Systems**:

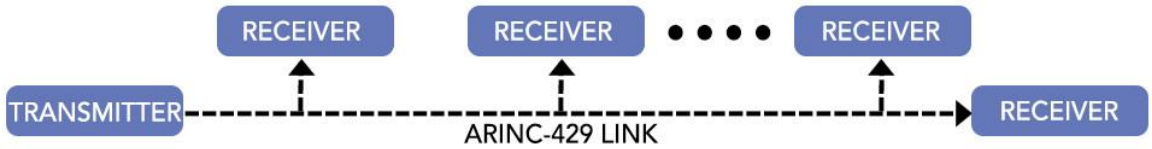
- The protocol is used to send data from various sensors and systems to cockpit displays, providing pilots with the information needed to fly the aircraft.

5. **Communication and Data Management**:
   - ARINC 429 is used to manage communications between avionics systems, ensuring that critical data is shared

ARINC-429 SPECIFICATION:

The ARINC-429 technical specification, originally referred to as the Digital Information Transfer System (DTIS), was published in 1977 to define how avionics systems and components should communicate within commercial aircraft. The Mark 33 Digital Information Transfer System, as it is known today, is still the standard most commonly used by airlines. This specification is used to establish 429 bus communications for word structures, electrical characteristics and other protocols.

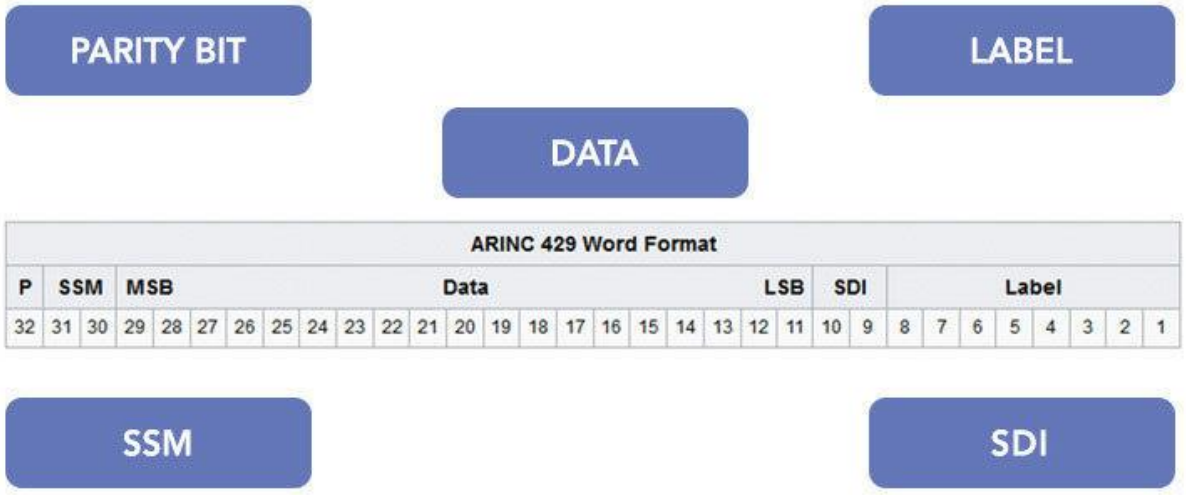
What is unique about ARINC 429 data transfer is its simple one directional flow of bus communications data. A typical data bus offers multidirectional data transfer between various bus points on a single set of wires. Not so with ARINC-429, but this is not taken as a disadvantage to the airlines as it has allowed for long-term operational cost savings and system reliability.



The ARINC-429 specification entails the following:

- Hardware consisting of only a single transmitter source supporting 1 to 20 receivers (also known as “sinks”) on a single wire pair.
- Data transmission is one directional. Additional busses are required for multidirectional data transfer.
- A data transmitter can only talk to a defined number of data receivers on a single bus on one wire pair.
- For multidirectional communication, 2 wire pairs are required for data transmission in opposite directions.
- Transmit and receive channels are different ports.
- Data words are 32 bits (most messages consist of a single data word) broken into 24-bits containing the core information and 8-bits acting as a data label describing the data transmitted.
- Messages are transmitted at either low speed (12.5 kbit/s) or high speed (100 kbit/s) to receiver components.

ARINC-429 WORD FORMAT:



ARINC 429 Word Format																													
P	SSM		MSB										Data								LSB		SDI		Label				
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

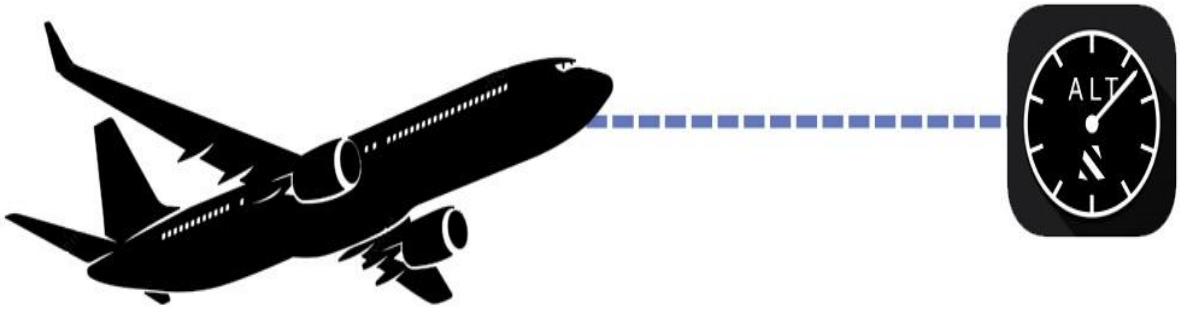
Data is sent over the ARINC-429 bus in a 32-bit word, with each word representing an engineering unit such as altitude or barometric pressure. The different parts of the message are shown in the image above. The 8-bit label is an important aspect. It is used to interpret the other fields of a message – each type of equipment will have a set of standard parameters identified by the label number, regardless of the manufacturer.

For example, Label 372 for any Heading Reference system will provide wind direction and Label 203 for any air data computer will give barometric altitude.

The other bits are reserved for SDI, SSM, data, and parity:

- **SDI (Source Destination Identifiers):** Used by a transmitter connected to multiple receivers to identify which one should process the message. If not needed, the bits may be used for data.
- **Data:** The information that is being communicated
- **SSM (Sign Status Matrix):** Used to indicate sign or direction, and also to test if data is valid
- **Parity (odd):** Used for error detection

ARINC-429 TRANSMISSION : Data is transmitted in a Return-to-Zero (RZ) format with three different states – HIGH(1), NULL, and LOW(0). HIGH state is achieved when the transmission signal jumps from NULL to +10V, then back to zero. LOW is similarly achieved when the signal goes to -10V and back**.**



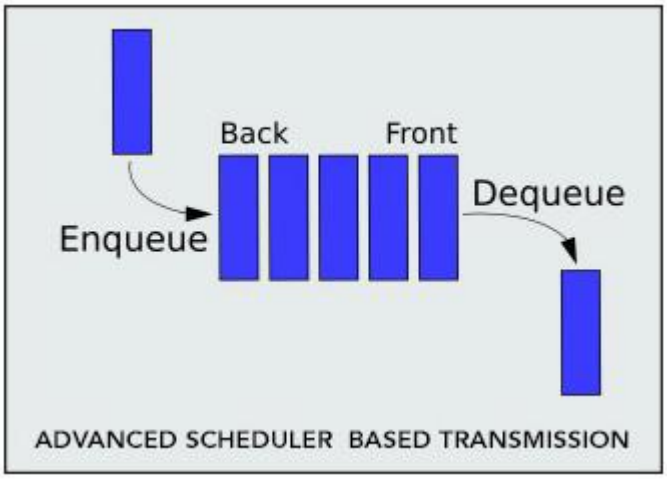
ARINC 429 specifies two speeds for data transmission – low speed of 12.5 kHz with an allowable range of 12 to14.5kHz, and a high speed of 100kHz +/- 1%. While this may be fair for most cases, some systems require a bit more flexibility.

TRANSMISSION INTERVAL :

Typical update rates are set to either 25, 40, or 65 ms. Avionics equipment usually display inoperable after two consecutive frames are missing, which can be strenuous on the software that is running. To ensure consistent messaging success, it is best practice to have the hardware help save the software's bandwidth. The two types of messaging techniques are First-in First-out (FIFO) and the Scheduler.

ARINC 429 SCHEDULER

Entry #	Type	PS / Clk	Time Delay	Recycle	Data
0	ME	01	0	0	123
1	ME	01	0	1	543
2	ME	01	1	1	4324
3	ME	01	4	1	4
4	SE	01	0	1	989
5	SE	01	3	1	654
6	SE	01	0	1	234
7	ME	11	20	1	23



As mentioned above, if you need to send a message, you can use one or both methods:

- A FIFO based method where the command is simply transmitted when you call the FIFO's API.
- Set up an advanced Scheduler that allows for master slave entries, so messages are sent only when another message has been sent.

  You can have messages sent at 3 different frequencies and set for a time delay so a message has different timing than the previous message. You can also be notified if a command was sent as well as have a message be sent once and only once if needed.

For successful real-world ARINC 429 testing, you should have a flexible, capable scheduler.

ADDRESSING SIMULATION, TESTING & CONTROL CHALLENGES

Testing, verifying, simulating, and controlling avionics busses present many challenges.

UEI offers a wide variety of methods to control and verify avionic equipment so your systems run efficiently.

SPECIAL FEATURES:

UEI has developed special features for ARINC 429 to solve critical operational challenges:

Precise Timing - A client needed to be signaled upon receipt of a GPS label. They could do it in software, however timing was an issue. UEI's solution was to use the FPGA for precise timing. In one application, a non-ARINC 429 device need to know when a label was sent. We adapted the system to send a pulse when a label of a particular value was transmitted.

Bus Conversion - Many times, protocols need to be converted. With a 1553 and 429 IO board, one application took in 1553 and send it out 429. Another took in ARINC 664 (AFDX™) and sent the data out in a CAN bus. In flight, a new entry into the Very Light Jet Market required a single Data Acquisition System that not only monitors the analog and digital inputs but also ARINC-429 avionics bus, RS-232 devices with an integrated GPS receiver to log position and velocity data. UEI was able to provide this solution.

24-channel ARINC 429 Interface

DNX-429-516, DNA-429-516, DNR-429-516, DNF-429-516

FEATURES

- DNA-429-516 for use in “Cubes”, DNF-429-516 forFLATRACK and DNR-429-516 for use in RACKtangle™ chassis
- 16 ARINC 429 RX or TX plus 8 dedicated RX channels
- High (100 kHz) or low (12.5 kHz) speed selectable by channel
- Hardware Label filtering and TX scheduler
- Includes support for ARINC-615 protocol
- 350 Vrms Isolation (in 8 groups of 3-channels)
- Guardian Series Diagnostics:
  - On-board 429 RXs allow read-back on TX channel

DESCRIPTION :

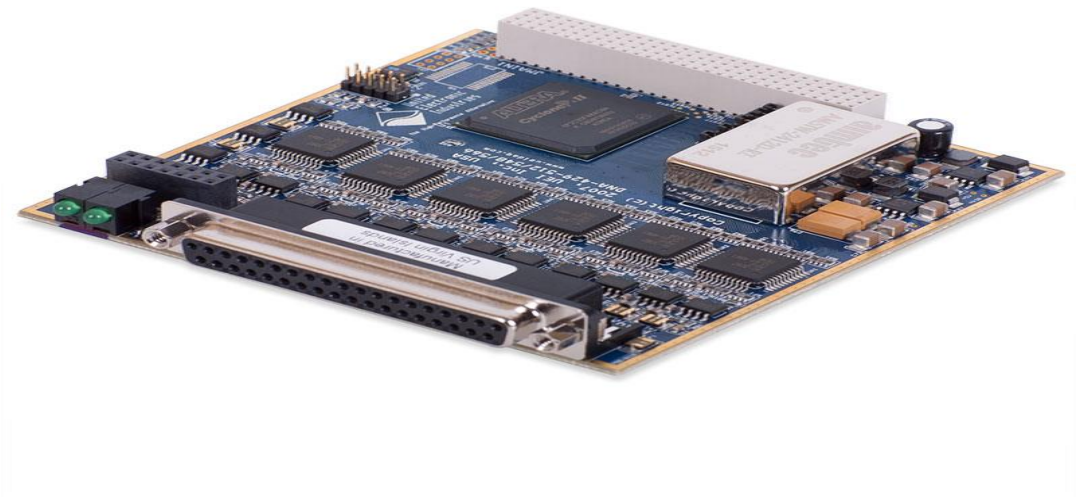
- DESCRIPTION
  - The DNA/DNR/DNF-429-516 are ARINC 429 communications interfaces for UEI’s popular Cube / RACKtangle / FLATRACK I/O chassis respectively. The DNx-429-516 offers 16 TX/RX channels and 8dedicated

RX channels. The 8 dedicated RX channels are a new feature and are available on all 429-516 boards. The TX drivers on the TX/RX channels can be disabled on a channel-by-channel basis. To use a TX/RX channel as RX simply disable the TX driver on that channel.

- The board is fully compatible with the earlier DNx-429-516-024 board and software written for the DNx-429-516-024 version will work without issue on the new board. The new board simply adds 8 dedicated RX channels to the connector that were previously labelled s RSVD on earlier versions of the DNx-429-516.
- All boards are fully compliant with the ARINC 429 spec and support both high speed (100 kHz) and low speed (12.5 kHz) operation. The channel speed is software selectable on a channel by channel basis. The channel speed can be set to frequencies other than 100 and 12.5 kHz to support legacy devices that “push” the ARINC 429 standard.
- Data integrity, even when all channels are set in high speed mode, is assured with the use of 256 word FIFOs on all channels. and in both directions. The board is part of UEI’s Guardian series and provides a diagnostic, on-board ARINC-429 receiver connected to each transmit channel. This allows the application to confirm the correct information has been written to the ARINC-429 bus.
- Channels may be set to transmit asynchronously or based on a hardware controlled scheduler. Each channel supports a transmission table that allows up to 256 unique schedules. Transmission schedule resolution is100 microseconds. There is also a TX mode where a label is transmitted only upon receipt of data from a pre-programmed label. Asynchronous (non-scheduled) data may be sent with three priorities. High priority data is sent immediately upon completion of the current transmission, regardless of scheduled messages. Data sent with standard priority is transmitted during times when no scheduled data is being sent. Finally, the lowest priority is

data streamed from a 256 word FIFO which is sent when no scheduled, high or standard priority data is being transmitted.

- The DNx-429-516 series provides a host of helpful filtering capabilities. The board may be set to only return data from specific labels. Data from up to 255 labels may be read or the board can be set to read data from all labels. A “new data only” filter compares the received label data to the most recent previous reading and only returns data if something has changed. Data may also be filtered based on the SDI bits.
- Software for the DNx-429-516 is included with the board. The UEIDAQ Framework provides a comprehensive, easy to use API supporting all popular Windows programming languages. Factory written and supported drivers are also included for Linux and are available for other popular real-time operating systems including QNX and VxWorks. Finally, the UEIDAQ Framework supplies complete support for those creating applications in all popular data acquisition and control packages, including LabVIEW, MATLAB/Simulink, as well as any application which supports ActiveX or OPC servers.
- The DNx-429-516 board is also fully supported by UEI’s popular UEIPAC series chassis.



ARINC 429 Interface board with 12 RX channels

DNX-429-512, DNA-429-512, DNR-429-512, DNF-429-512

FEATURES:

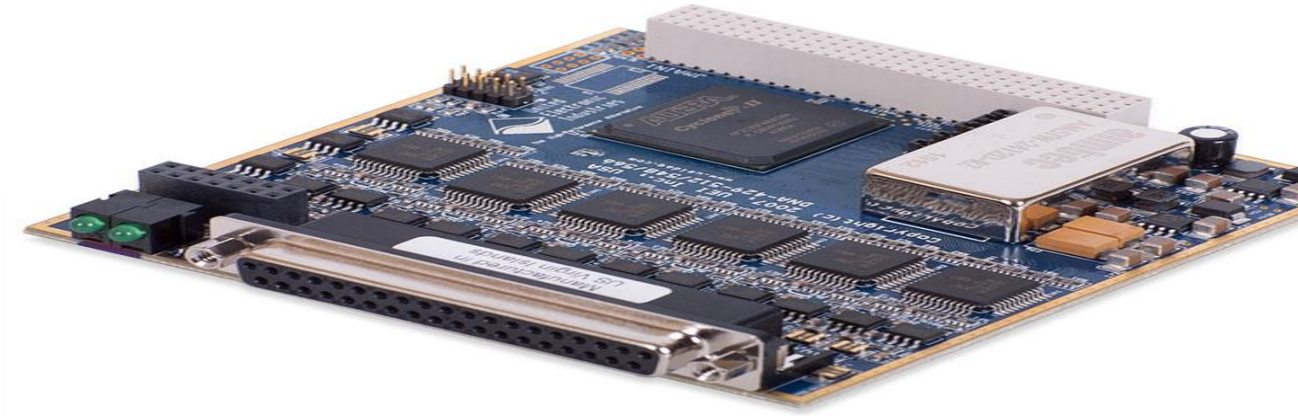
- 12 ARINC 429 RX channels
- High (100 kHz) or low (12.5 kHz) speed selectable by channel
- Hardware Label filtering
- Automatic timestamping of RX data (if desired)
- Powerful API included

- DESCRIPTION:
  - The DNx-429-512 is a 12 channel ARINC-429 receiver interface. The board is fully compliant with the ARINC 429 spec and supports both high speed (100 kHz) and low speed (12.5 kHz) operation. The channel speed is software selectable in banks of either two channels. Data integrity, even when all channels are set in high speed

mode, is assured with the use of 256 word FIFOs on all RX channels.

- Receive channels include the ability to filter on Labels so that only data from selected channels is captured. The filter may be set to forward data from between one and 255 Labels, or may be disabled so all data is captured, regardless of source. The user may also select on a label by label basis whether all data is forwarded or only data which has changed since the last transmission. Each channel may be set such that data Receive filtering is also supported based on the Source/Destination Identifier (bits 9 & 10). Words that match the desired SDI are forwarded while those that do not are ignored. Each received word may be time stamped with the date and time of reception (10 µs resolution). Parity errors may either be flagged and errant data trapped at the board level, or RX channels may be configured to forward data with an illegal parity bit.

- Software for the DNx-429 series is provided as part of the UEI Framework. The framework provides a comprehensive yet easy to use API that supports all popular Windows programming languages as well as supporting programmers using Linux and most real-time operating systems including QNX, INtime, RT Linux and more. Finally, the framework supplies complete support for those creating applications in LabVIEW, MATLAB/Simulink, or any application which supports ActiveX or OPC servers.



ARINC 429 Interface board with 6 TX and 6 RX channels

DNX-429-566, DNA-429-566, DNR-429-566, DNF-429-566

FEATURES

- ARINC 429 Interface board with 6 TX and 6 RX channels
- 6 ARINC 429 TX channels
- 6 ARINC 429 RX channels
- High (100 kHz) or low (12.5 kHz) speed selectable by channel
- Hardware Label filtering
- Hardware TX scheduler (100 µs timing resolution)

- Automatic timestamping of RX data (if desired)
- Powerful API included

DESCRIPTION:

The DNx-429-566 is a 12 channel (6 TX, 6 RX) ARINC-429 communication interface. The board is fully compliant with the ARINC 429 spec and supports both high speed (100 kHz) and low speed (12.5 kHz) operation. The channel speed is software selectable in banks of either two channels. Data integrity, even when all channels are set in high speed mode is assured with the use of 256 word FIFOs on all RX and TX channels.

Receive channels include the ability to filter on Labels so that only data from selected channels is captured. The filter may be set to forward data from between one and 255 Labels, or may be disabled so all data is captured, regardless of source. The user may also select on a label by label basis whether all data is forwarded or only data which has changed since the last transmission. Each channel may be set such that data receive filtering is also supported based on the Source/Destination Identifier (bits 9 & 10). Words that match the desired SDI are forwarded while those that do not are ignored. Each received word may be time stamped with the date and time of reception (10 µs resolution). Parity errors may either be flagged and errant data trapped at the board level, or RX channels may be configured to
forward data with an illegal parity bit.

Transmit channels may be set to transmit asynchronously or based upon a hardware controlled scheduler. Each channel supports a transmission table that allows up to 256 unique

schedules. Transmission schedule resolution is 100 microsecond. There is also a TX mode where a label is transmitted only upon receipt of data from a preprogrammed label.

Asynchronous (non-scheduled) data may be sent with three priorities. High priority data is sent immediately upon completion of the current transmission, regardless of scheduled messages. Data sent with standard priority is transmitted during times when no scheduled data is being sent. Finally, the lowest priority is data streamed from a 256 word FIFO which is sent when no scheduled, high or standard priority data is being transmitted.

Software for the DNx-429-566 is provided as part of the UEI Framework. The framework provides a comprehensive yet easy to use API that supports all popular Windows programming languages as well as supporting programmers using Linux and most real-time operating systems including QNX, INtime, RT Linux and more. Finally, the framework supplies complete support for those creating applications in LabVIEW, MATLAB/Simulink, or any application which supports ActiveX or OPC servers.

Example: ARINC 429 Transmission and Reception in C

1. Definitions and Setup:

```
#include <stdint.h>

#include <stdbool.h>

// Define the bit rate (High Speed: 100 kbps or Low Speed: 12.5 kbps)
```

```
#define ARINC_BIT_RATE_HIGH 100000
#define ARINC_BIT_RATE_LOW 12500
// Example pin definitions for ARINC 429 interface
#define ARINC_TX_PIN  0x01  // Example GPIO pin for transmission
#define ARINC_RX_PIN  0x02  // Example GPIO pin for reception
#define ARINC_WORD_LENGTH 32     // Data word length for ARINC 429
// ARINC 429 word structure
typedef struct {
    uint8_t label;     // 8-bit label
    uint8_t sdi;       // 2-bit Source/Destination Identifier
    uint32_t data;     // 19-bit data field
    uint8_t ssm;       // 2-bit Sign/Status Matrix
    bool parity;       // 1-bit parity
} ARINC429Word;
// Function prototypes
void ARINC429_Init(uint32_t bitRate);
void ARINC429_Transmit(ARINC429Word word);
ARINC429Word ARINC429_Receive(void);
bool CalculateParity(uint32_t word);
```

2. Initialization:

```
void ARINC429_Init(uint32_t bitRate) {
```

```
    // Initialize GPIO pins and set the bit rate for ARINC 429 transmission
    // Setup Timer or UART peripheral for transmission depending on the microcontroller
    // Configure ARINC_TX_PIN as output
    // Configure ARINC_RX_PIN as input
    // Set the appropriate bit rate for ARINC 429 transmission
    // Example: Setup code for a microcontroller peripheral here
    // Initialize UART/Timer for ARINC429 with the specified bit rate
}
```

3.Parity Calculation

ARINC 429 uses odd parity, so we need to ensure that the number of '1' bits in the 31-bit data word plus the parity bit is odd.

```
bool CalculateParity(uint32_t word) {
    uint8_t count = 0;
    for (int i = 0; i < 31; i++) {
        if (word & (1 << i)) {
            count++;
        }
    }
    // Return true if odd parity (odd number of '1' bits)
    return (count % 2) == 0;
```

```
}
```

4.Transmission of ARINC 429 Word

```
void ARINC429_Transmit(ARINC429Word word) {
    uint32_t arincWord = 0;
    // Construct the 32-bit ARINC 429 word
    arincWord |= ((uint32_t)word.label << 24);
    arincWord |= ((uint32_t)word.sdi << 22);
    arincWord |= ((uint32_t)word.data << 3);
    arincWord |= ((uint32_t)word.ssm << 1);
    arincWord |= (uint32_t)CalculateParity(arincWord);
    // Transmit the ARINC 429 word bit by bit
    for (int i = 0; i < ARINC_WORD_LENGTH; i++) {
        bool bit = (arincWord & (1 << (31 - i))) != 0;
        // Set the transmission pin to the corresponding bit value
        // Transmit each bit using the appropriate timing
        // Example: TransmitBit(bit);
    }
    // Ensure minimum idle time between words (4-bit times)
    // Example: delayMicroseconds(4 * bitTime);
}
```

5.Reception of ARINC 429 Word

```
ARINC429Word ARINC429_Receive(void) {
    ARINC429Word receivedWord;
    uint32_t arincWord = 0;
    // Wait and read 32 bits from the ARINC 429 receiver line
    for (int i = 0; i < ARINC_WORD_LENGTH; i++) {
        // Example: bool bit = ReadBit();
        bool bit = false;  // Placeholder for actual bit reading function
        arincWord |= ((uint32_t)bit << (31 - i));
    }
    // Parse the received 32-bit ARINC 429 word
    receivedWord.label = (arincWord >> 24) & 0xFF;
    receivedWord.sdi = (arincWord >> 22) & 0x03;
    receivedWord.data = (arincWord >> 3) & 0x7FFFF;
    receivedWord.ssm = (arincWord >> 1) & 0x03;
    receivedWord.parity = arincWord & 0x01;
    // Validate parity (optional)
    if (CalculateParity(arincWord) != receivedWord.parity) {
        // Handle parity error
    }
    return receivedWord;
```

}

Summary:

Initialization: Configure the transmission and reception pins and set the appropriate bit rate for ARINC 429 communication.

Parity Calculation: Ensure that the parity bit for each 32-bit word maintains odd parity.

Transmission: Build the 32-bit ARINC 429 word using the structure fields, then transmit it bit by bit with correct timing.

Reception: Receive a 32-bit word from the ARINC 429 bus, parse it into the correct fields, and verify the parity.

This example is highly simplified and assumes that you have access to GPIO manipulation functions or hardware peripherals that can handle precise timing (like UART or a timer peripheral). Actual implementations may require more detailed timing control, especially to meet ARINC 429's strict timing and electrical standards.

If you're implementing ARINC 429 on a specific microcontroller or hardware platform, you'll need to adapt the GPIO and timing functions to that platform's specific capabilities.

To implement ARINC 429 communication in C, you'll need to interface with ARINC 429 hardware (e.g., a transceiver or a controller card), as the standard requires specific electrical signaling and timing. Typically, you'd use a dedicated ARINC 429 library or API provided by the hardware manufacturer. Below is a conceptual example of how you might structure C code for transmitting and receiving ARINC 429 data. This example assumes the existence of hypothetical API functions provided by the ARINC 429 hardware vendor.

Example: ARINC 429 Transmission and Reception in

```
#include <stdio.h>

#include <stdint.h>

#include <stdbool.h>

// Hypothetical ARINC 429 hardware API

#include "arinc429.h" // Assume this is the API provided by the hardware vendor

// Function to create an ARINC 429 word

uint32_t create_arinc429_word(uint8_t label, uint8_t sdi, uint32_t data, bool parity) {

    uint32_t word = 0;

    word |= (label & 0xFF) << 24;       // Set the label (8 bits)

    word |= (sdi & 0x03) << 22;         // Set the SDI (2 bits)

    word |= (data & 0x7FFFF) << 3;      // Set the data field (19 bits)

    if (parity) {

        word |= 0x1; // Set the parity bit

    }

    return word;

}

// Function to transmit an ARINC 429 word

void arinc429_transmit(uint32_t word) {
```

```
    // Send the ARINC 429 word using the hardware API
    arinc429_send_word(word);
}
// Function to receive an ARINC 429 word
uint32_t arinc429_receive() {
    uint32_t word = 0;
    // Wait for an ARINC 429 word from the hardware
    if (arinc429_receive_word(&word)) {
        return word;
    } else {
        // Handle error or timeout
        printf("Error: Failed to receive ARINC 429 word\n");
        return 0;
    }
}
// Function to parse an ARINC 429 word
void parse_arinc429_word(uint32_t word) {
    uint8_t label = (word >> 24) & 0xFF;
    uint8_t sdi = (word >> 22) & 0x03;
    uint32_t data = (word >> 3) & 0x7FFFF;
    bool parity = word & 0x1;
```

```
    // Display the parsed information
    printf("Label: 0x%02X\n", label);
    printf("SDI: 0x%01X\n", sdi);
    printf("Data: 0x%05X\n", data);
    printf("Parity: %d\n", parity);
}

int main() {
    // Example of creating, transmitting, and receiving an ARINC 429 word

    // Create an ARINC 429 word (example: Label 0x1A, SDI 0x2, Data 0x12345, Parity true)
    uint32_t tx_word = create_arinc429_word(0x1A, 0x2, 0x12345, true);
    // Transmit the word
    arinc429_transmit(tx_word);
    // Receive an ARINC 429 word
    uint32_t rx_word = arinc429_receive();
    // Parse and display the received word
    parse_arinc429_word(rx_word);
    return 0;
```

}

Key Points:

Creating ARINC 429 Word: The `create_arinc429_word` function assembles a 32-bit ARINC 429 word from the label, SDI, data, and parity inputs.

Transmission: The `arinc429_transmit` function sends the ARINC 429 word using the `arinc429_send_word` function from the hypothetical hardware API.

Reception: The `arinc429_receive` function waits for and receives an ARINC 429 word using the `arinc429_receive_word` function from the hypothetical hardware API.

Parsing: The `parse_arinc429_word` function extracts the label, SDI, data, and parity from the received ARINC 429 word and displays them.

Notes:

- The above code is a conceptual example. Real-world applications would involve using specific libraries/APIs from the hardware manufacturer.
- Error handling, timeouts, and retries should be implemented based on the hardware specifications and the criticality of the data being transmitted/received.

If you are using specific hardware, you'll need to refer to its documentation for the exact API functions and usage.