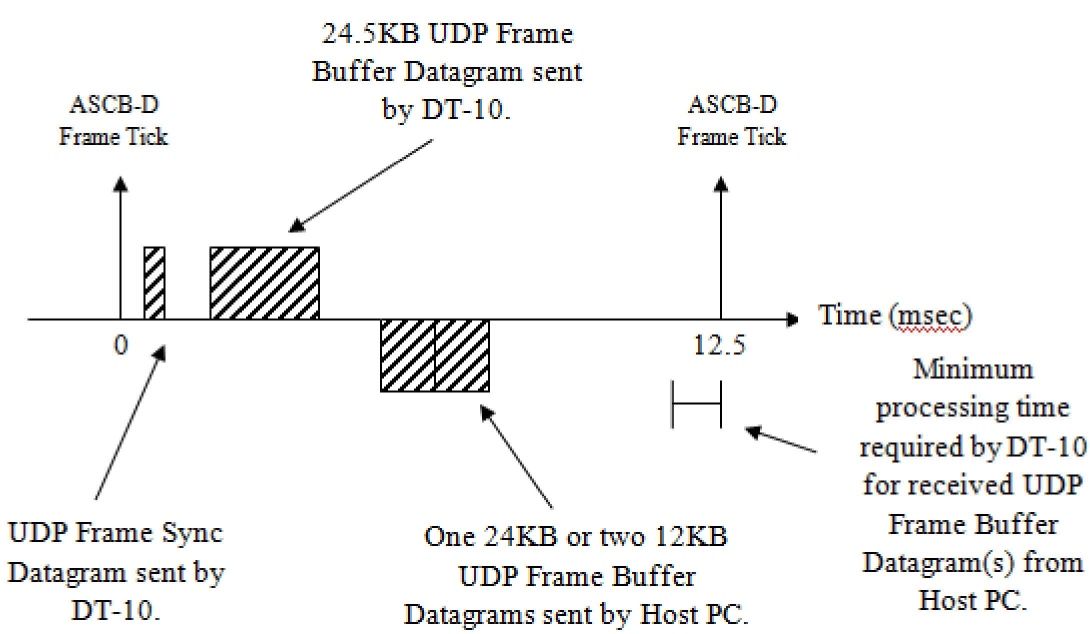
1. Interfacing the DataTap-10 to an ASCB-D Data Generator

Interfacing the DataTap-10 (DT-10) to a host PC acting as an ASCB-D data generator requires only an Ethernet connection between the two. The DT-10 is connected to the two ASCB-D busses and ships out a UDP frame sync datagram over the Ethernet link at the start of each ASCB-D frame. It then proceeds to ship out one large UDP datagram consisting of a 512-byte header and two 12KB buffers of frame buffer data - one for each ASCB-D bus. This datagram will be fragmented in order to meet the Maximum Transfer Unit constraint of the Ethernet link.

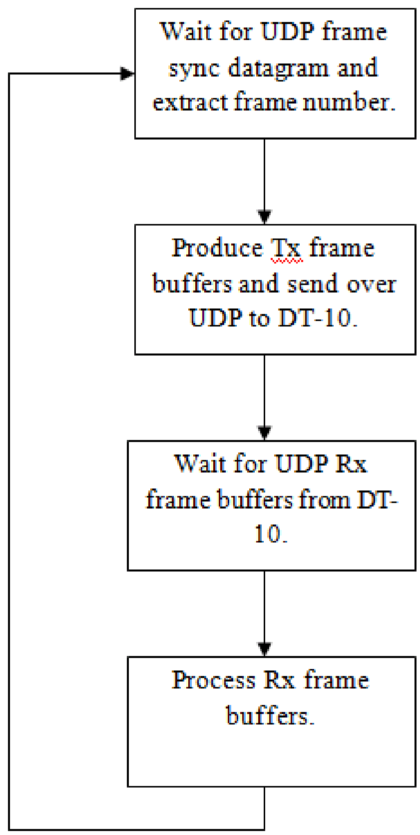
The host PC must first listen for the UDP frame sync datagram transmitted at the start of an ASCB frame. It then produces the frame buffers for ASCB-D transmission based on the frame number extracted from the Frame Sync datagram. Once the frame buffers have been produced, the host PC sends them over UDP to the DT-10. The DT-10 must receive the buffers and have time to process them prior to the start of the next ASCB-D frame. The time it takes for the DT-10 to process the frame buffers is less than 1.5 milliseconds. It is expected that this datagram (or datagrams) will be fragmented in order to meet the Maximum Transfer Unit constraint of the Ethernet link.

The sending and receiving of UDP datagrams over the Ethernet link during a typical ASCB-D frame is shown below.



Once the host PC has sent out the datagram(s) over UDP, it can then wait for the UDP frame buffer datagram sent by the DT-10. This may likely already be queued up in the host PC's IP stack. This datagram represents the data received from the ASCB-D busses. Once the full 24KB of data has been received, the host PC can then consume the received frame buffers. It then waits for the next UDP frame sync datagram to indicate the start of the next ASCB frame.

A flow chart depicting the typical processing steps on the host PC for an ASCB-D frame is shown below.



The only DT-10 configuration steps required by the host PC are to set the NIC IDs. This is described in the following section.

* Note that throughout the following sections, the pilot-side ASCB bus is referred to as ASCB Bus 0 and the copilot-side bus is referred to as ASCB Bus 1.

1.1.Configuring the NIC IDs of the DT-10

The NIC IDs of the DT-10 can be dynamically changed through a UDP command datagram sent from a host PC to the DT-10. A single NIC ID can be assigned for each of the two ASCB busses. A value of 0 for a NIC ID will put the DT-10 into a read-only mode for the associated bus. Reception of a NIC ID command datagram will cause the DT-10 to automatically re-sync to the ASCB bus.

The currently assigned NIC ID values are reflected in any status datagrams sent from the DT-10. A status datagram is sent as an acknowledgment to any command datagrams received, and may be sent periodically depending on the configuration of the DT-10. Once a command datagram is sent to the DT-10, it is recommended that the host PC wait for and verify the returned status datagram due to the unreliability of the UDP protocol. The command datagram should be resent if a status datagram is not received or if the values are not reflected within a reasonable amount of time (i.e. 250 milliseconds).

The command datagrams are described in the following section.

1.2.Command Datagrams

The command datagrams allow dynamic configuration of the DT-10 from a host PC. The UDP port that the DT-10 will listen on for these datagrams is configurable through the system.cfg file located on the microSD card, and defaults to a value of 51001. The command datagram structure for setting the NIC IDs is shown below.

1.2.1. NIC ID Command Datagram Structure

```
typedef struct __attribute__ ((__packed__))
{
   uint32 command;            // Command
   uint32 ascb0_nic_id;       // ASCB 0 NIC ID
   uint32 ascb1_nic_id;       // ASCB 1 NIC ID
   uint08 padding[88];        // Padding

} udp_nic_id_cmd_pkt_t;
```

The command field to allow setting of the NIC IDs must be set to a value of 0x0A, as shown:

31   0
Command (0x0A)
ACSB 0 NIC ID
ASCB 1 NIC ID
Padding

The following is a capture of a NIC ID command datagram starting at the UDP payload:

```
0000: 0a 00 00 00 01 00 00 00 21 00 00 00 00 00 00 00
0010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060: 00 00 00 00
```

Note that the UDP payload must be padded to a length of 100 bytes, as shown, or it will be rejected by the DT-10. Also note that the actual NIC ID value should be written into each NIC ID field of the command datagram and not a bit-field representation, as is returned in the status datagrams.

1.3. Status Datagrams

Status datagrams are automatically sent out of the DT-10 at a rate of 10 Hz, unless the UDP_STS_EN_DEF parameter in the system.cfg file is set to FALSE. A status datagram is always immediately sent out in response to a command datagram received. The UDP destination port of the status datagrams is configurable through the system.cfg file, and defaults to a value of 51000.

The status datagram structure starting at the UDP payload is shown below.

1.3.1. Status Datagram Structure

```
typedef struct __attribute__ ((__packed__))
{
   uint32 tbl_ver;            // Structure/Table Version
   uint32 status;             // Status Field                [1]
   uint32 frame_num;          // Current ASCB Frame Number
   uint32 IRIG1;              // IRIG 1 Field                [2]
```

```
    uint32 IRIG2;              // IRIG 2 Field                 [2]
    uint32 IRIG3;              // IRIG 3 Field                 [2]
    uint32 IRIG4;              // IRIG 4 Field                 [2]
    uint32 cpu0_load;          // CPU 0 Load Percentage
    uint32 board_temp;         // Temp Sensor Reading
    uint32 ascb0_pkt_cnt;      // ASCB 0 Packet Count
    uint32 ascb0_crc_err;      // ASCB 0 CRC Error Count
    uint32 ascb0_runt_err;     // ASCB 0 Runt Packet Count
    uint32 ascb0_sfd_err;      // ASCB 0 SFD Error Count       [3]
    uint32 ascb1_pkt_cnt;      // ASCB 1 Packet Count
    uint32 ascb1_crc_err;      // ASCB 1 CRC Error Count
    uint32 ascb1_runt_err;     // ASCB 1 Runt Packet Count
    uint32 ascb1_sfd_err;      // ASCB 1 SFD Error Count       [3]
    uint32 gpio_inputs;        // GPIO Input State
    uint32 gpio_outputs;       // GPIO Output State
    uint16 adc_val[4];         // ADC Readings
    uint08 timing_nics[4];     // Timing NIC IDs Last Frame
    uint32 nic_synced_to;      // NIC Synced To Last Frame
    uint32 ascb0_asgn_nids;    // ASCB 0 Assigned NIC IDs      [4]
    uint32 ascb1_asgn_nids;    // ASCB 1 Assigned NIC IDs      [4]
    uint08 nic_reg_hdr[144];   // NIC Registry Header String
    uint08 esc_reg_hdr[176];   // ESCAPE Registry Header String
    uint32 fpga_version;       // FPGA version
    uint08 cpu0_sw_date[16];   // CPU 0 SW Compile Date String
    uint08 cpu0_sw_time[16];   // CPU 0 SW Compile Time String
    uint08 cpu1_sw_date[16];   // CPU 1 SW Compile Date String
    uint08 cpu1_sw_time[16];   // CPU 1 SW Compile Time String
    uint08 cpu2_sw_date[16];   // CPU 2 SW Compile Date String
    uint08 cpu2_sw_time[16];   // CPU 2 SW Compile Time String
    uint32 command;            // Echo of Command Received
    uint08 cmd_data[96];       // Echo of Command Data Received
    uint32 ascb0_rcvd_nids[2]; // ASCB 0 Received NIC IDs      [5]
    uint32 ascb1_rcvd_nids[2]; // ASBB 1 Received NIC IDs      [5]

} udp_sts_pkt_t;
```

Where:

[1] The “status” field can have the following bits set:

```
#define C_UDP_STS_EMAC_10          0x00000001
#define C_UDP_STS_EMAC_100         0x00000002
#define C_UDP_STS_EMAC_1000        0x00000004
#define C_UDP_STS_IDIODE_USB       0x00000100
#define C_UDP_STS_IDIODE_936       0x00000200
#define C_UDP_STS_RXSW0_PRIM       0x00010000
#define C_UDP_STS_RXSW1_PRIM       0x00020000
```

[2] The IRIG fields can be decoded using the following table:

Field	[31:16]	[15:12]	[11:8]	[7:4]	[3:0]
IRIG1	0	10^0 mS	10^2 uS	10^1 uS	10^0 uS
IRIG2	0	10^1 Seconds	10^0 Seconds	10^2 mS	10^1 mS
IRIG3	0	10^1 Hours	10^0 Hours	10^1 Minutes	10^0 Minutes
IRIG4	0	0	10^2 Days	10^1 Days	10^0 Days

[3] The “ascb0_sfd_err” and “ascb1_sfd_err” fields indicate the number of start-of-frame delimiter errors on ASCB bus 0 and ASCB bus 1, respectively. This error occurs when the DT-10 detects a valid preamble but no start-of-frame delimiter.

[4] The “ascb0_asgn_nids” and “ascb1_asgn_nids” fields reflect the currently assigned NIC ID values of the DT-10. These are 32-bit fields with each bit representing a specific NIC ID, as shown:

“ascb0_asgn_ids” field definition:

```
0x80000000 – N/A
0x40000000 – NIC ID 31
      …
0x00000004 – NIC ID 3
0x00000002 – NIC ID 2
0x00000001 – NIC ID 1
```

“ascb1_asgn_ids” field definition:

```
0x80000000 – N/A
0x40000000 – NIC ID 63
      …
0x00000004 – NIC ID 35
0x00000002 – NIC ID 34
0x00000001 – NIC ID 33
```

[5] The “ascb0_rcvd_nids” and “ascb1_rcvd_nids” contain a bit-field representation of all NIC IDs received in the previous frame on ASCB Bus 0 and ASCB Bus 1, respectively. These are 64-bit fields allowing indication of all possible NIC IDs on either bus. This provides some error detection as well as the ability to detect if the ASCB cables are swapped. The bit-field representation is shown below:

“ascb0_rcvd_nids[0]” and “ascb1_rcvd_nids[0]”:

```
0x80000000 – N/A
0x40000000 – NIC ID 63
      …
0x00000004 – NIC ID 35
0x00000002 – NIC ID 34
0x00000001 – NIC ID 33
```

"ascb0_rcvd_nids[1]" and "ascb1_rcvd_nids[1]":

```
0x80000000 – N/A
0x40000000 – NIC ID 31
      …
0x00000004 – NIC ID 3
0x00000002 – NIC ID 2
0x00000001 – NIC ID 1
```

The following is a capture of a status datagram starting at the UDP payload:

```
0000   02 00 00 00 04 01 03 00 47 1f 00 00 36 37 00 00  ........G...67..
0010   80 46 00 00 01 00 00 00 01 00 00 00 14 00 00 00  .F..............
0020   00 14 00 00 db f7 00 00 00 00 00 00 00 00 00 00  ................
0030   00 00 00 00 1d f5 00 00 00 00 00 00 00 00 00 00  ................
0040   00 00 00 00 0f 00 00 00 00 00 00 00 00 00 00 00  ................
0050   00 00 00 00 01 02 21 22 00 00 00 01 03 00 00 00  ......!"........
0060   03 00 00 00 00 01 fe ff 50 41 00 00 02 00 00 00  ........PA......
0070   90 00 00 00 20 e6 13 00 50 00 00 00 00 08 02 04  .... ...P.......
0080   00 12 04 1f 00 02 04 1f 00 03 42 01 f0 a8 f6 d9  ..........B.....
0090   33 34 33 33 31 30 36 39 2d 4e 49 43 20 52 65 67  34331069-NIC Reg
00a0   2d 30 30 31 33 45 36 32 30 00 00 00 00 00 00 00  -0013E620.......
00b0   50 6c 61 6e 65 76 69 65 77 20 47 56 49 00 00 00  Planeview GVI...
00c0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00d0   47 56 49 20 50 6c 61 6e 65 76 69 65 77 20 54 49  GVI Planeview TI
00e0   55 20 50 52 4f 44 55 43 45 20 41 4c 4c 00 00 00  U PRODUCE ALL...
00f0   54 26 41 21 00 01 fe ff a0 47 21 00 03 00 00 00  T&A!.....G!.....
0100   b0 00 00 00 c4 e6 13 00 50 01 05 01 00 05 02 04  ........P.......
0110   00 12 04 1f 00 02 04 1f 00 01 42 01 3f 33 19 10  ..........B.?3..
0120   33 34 33 33 31 30 36 39 2d 33 39 2d 32 31 38 2d  34331069-39-218-
0130   30 30 31 33 45 36 43 34 00 00 00 00 00 00 00 00  0013E6C4........
0140   50 6c 61 6e 65 76 69 65 77 20 47 56 49 00 00 00  Planeview GVI...
0150   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0160   47 56 49 20 50 6c 61 6e 65 76 69 65 77 20 54 49  GVI Planeview TI
0170   55 20 50 52 4f 44 55 43 45 20 41 4c 4c 00 00 00  U PRODUCE ALL...
0180   54 49 53 00 00 00 00 00 00 00 00 00 00 00 00 00  TIS.............
0190   54 49 53 50 52 4f 43 00 00 00 00 00 00 00 00 00  TISPROC.........
01a0   54 26 41 21 01 11 27 09 41 70 72 20 20 32 20 32  T&A!..'.Apr  2 2
01b0   30 31 31 00 00 00 00 00 31 34 3a 33 35 3a 32 36  011.....14:35:26
01c0   00 00 00 00 00 00 00 00 41 70 72 20 20 32 20 32  ........Apr  2 2
01d0   30 31 31 00 00 00 00 00 31 34 3a 33 35 3a 32 32  011.....14:35:22
01e0   00 00 00 00 00 00 00 00 41 70 72 20 20 32 20 32  ........Apr  2 2
01f0   30 31 31 00 00 00 00 00 31 34 3a 33 35 3a 32 39  011.....14:35:29
0200   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0210   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0220   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0230   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0240   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0250   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
0260   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0270   03 00 00 00 03 00 00 00 00 00 00 00               ............
```

1.4.Frame Sync Datagrams

A UDP frame sync datagram is sent out by the DataTap-10 at the start of each ASCB-D frame. These datagrams contain the current ASCB frame number and allow a host PC to begin producing the ASCB-D transmission frame buffers. The UDP frame sync datagrams are sent out of the DataTap-10 to the UDP port plus 1 specified in the system configuration file (system.cfg) with the UDP_HOST_PORT_RAW parameter. Sending of these datagrams can be disabled through the UDP_RAW_EN_DEF parameter in the system.cfg file.

The frame sync datagram structure starting at the UDP payload is shown below.

1.4.1. Frame Sync Datagram Structure

```
typedef struct __attribute__ ((__packed__))
{
   uint32 frame_num;        // ASCB Frame Number
   uint32 rfu0;             // Reserved for Future Use
   uint32 ascb0_rcvd_nids;  // ASCB 0 Received NIC IDs    [1]
   uint32 ascb1_rcvd_nids;  // ASCB 1 Received NIC IDs    [1]
   uint08 nic_synced_to;    // NIC ID synced to last frame
   uint08 rx_switch;        // ASCB Rx Switch Settings    [2]

} udp_sync_pkt_t;
```

Where:

[1] The “ascb0_rcvd_nids” and “ascb1_rcvd_nids” contain a bit-field representation of all valid NIC IDs received in the previous frame on ASCB Bus 0 and ASCB Bus 1, respectively. These are 32-bit fields representing only the valid NIC IDs received on each bus. The bit-field representation is shown below:

“ascb0_rcvd_nids”:

```
0x80000000 – N/A
0x40000000 – NIC ID 31
      …
```

```
0x00000004 – NIC ID 3
0x00000002 – NIC ID 2
0x00000001 – NIC ID 1
```

"ascb1_rcvd_nids":

```
0x80000000 – N/A
0x40000000 – NIC ID 63
      …
0x00000004 – NIC ID 35
0x00000002 – NIC ID 34
0x00000001 – NIC ID 33
```

[1] The "rx_switch" field indicates whether the DT-10 is receiving on the primary or backup bus for both ASCB 0 and ASCB 1. This field has the following definitions:

```
// ASCB 0 Rx Switch Setting (1=Primary, 0=Backup)
#define C_UDP_SYNC_RXSW0_PRIM     0x00000001

// ASCB 1 Rx Switch Setting (1=Primary, 0=Backup)
#define C_UDP_SYNC_RXSW1_PRIM     0x00000002
```

The following is a capture of a frame sync datagram along with its decoded fields starting at the UDP payload:

```
0000   08 4d 00 00 61 41 0c 27 03 00 00 00 03 00 00 00
0010   01 03
```

frame_number:	0x00004d08
ascb0_nic_ids:	NIC IDs 1 and 2
ascb1_nic_ids:	NIC IDs 33 and 34
nic_synced_to:	0x01
rx_switch:	Primary for both ASCB 0 and ASCB 1

1.5.Frame Buffer Output Datagrams

The frame buffer UDP datagrams sent from the DataTap-10 ASCB-D consist of a 512-byte header followed by two 12KB buffers of data. The 12KB buffers contain the output of the NIC PDD for each of the two ASCB-D busses. One large 25,088-byte datagram containing all data is sent each ASCB-D frame (every 12.5 milliseconds) after the UDP frame sync datagram is sent. Each large datagram is fragmented into smaller packets to meet the Maximum Transfer Unit constraint of the Ethernet link and will be reassembled by the IP stack residing on the host. These datagrams are sent out of the DataTap-10 to the UDP port specified in the system configuration file (system.cfg) with the UDP_HOST_PORT_RAW parameter. Sending of these datagrams can be disabled through the UDP_RAW_EN_DEF parameter in the system configuration file.

The frame buffer output datagram structure starting at the UDP payload is shown below.

1.5.1. Frame Buffer Output Datagram Structure

```
typedef struct __attribute__ ((__packed__))
{
   uint32 frame_tick;        // ASCB Frame Number
   uint32 unused0;
   uint32 unused1;
   uint32 unused2;

   uint32 IRIG1;             // IRIG Time 1
   uint32 IRIG2;             // IRIG Time 2
   uint32 IRIG3;             // IRIG Time 3
   uint32 IRIG4;             // IRIG Time 4

   uint08 unused3[480];

   uint08 rx2_data[12288];  // RX2 NIC Frame Buffer
   uint08 rx1_data[12288];  // RX1 NIC Frame Buffer

} udp_raw_out_pkt_t;
```

1.6. Frame Buffer Input Datagrams

The data to be transmitted onto the ASCB-D busses is sent to the DT-10 through one 24KB or two 12KB UDP datagrams. The datagrams contain the entire 12KB NIC transmit frame buffer for each bus. The DT-10 takes these frame buffers and transmits them on the appropriate ASCB bus as directed by the NIC Registry, similar to an MAU NIC.

The DT-10 listens for the datagrams based on the port specified by the UDP_LSTN_PORT_RAW parameter in the system.cfg file. The frame buffer data for ASCB Bus 0 is expected on the specified port and the frame buffer data for ASCB Bus 1 is expected on the port above it (specified port + 1). Alternatively, one 24KB datagram may be sent containing the frame buffer data for both busses (ASCB 0 data followed by ASCB 1 data) on the port specified by the UDP_LSTN_PORT_RAW parameter.

MAC Header
IP Header
UDP Header
ASCB 0 NIC Frame Buffer (12KB)

UDP Destination Port = UDP_LSTN_PORT_RAW

MAC Header
IP Header
UDP Header
ASCB 1 NIC Frame Buffer (12KB)

UDP Destination Port = UDP_LSTN_PORT_RAW + 1

The Frame Buffer data can be sent to the DT-10 in the form of two 12KB datagrams sent to successive UDP ports, as shown above.

MAC Header
IP Header
UDP Header
ASCB 0 NIC Frame Buffer (12KB)
ASCB 1 NIC Frame Buffer (12KB)

UDP Destination Port = UDP_LSTN_PORT_RAW

Alternatively, the Frame Buffer data can be sent to the DT-10 in the form of one 24KB datagram, as shown above.