# PC Data Transfer

## Protocol 1

### Data From PC

Protocol 1 command and control data is transferred in the C0 C1 C2 C3 & C4 bytes of the EP6 messages

|  |  |  |
| --- | --- | --- |
| **C0** | **Data** | **Function Calls** |
| Bit 0 | PTT | SetMOX(bool Mox); |
| 0000000x | Atlas bus controls; DDC sample rate; TX mode; open collector outputs; ADC dither & random; Alex atten; Alex RX ant, RX out, TX ant; Duplex on/off; DDC count; time stamp on/off | (Decoded for Orion mk2: Orion.v line 2445) Atlas bus controls ignored  Alex Atten bits ignored  Preamp bit is ignored Set P1SampleRate(int Rate); SetClassEPA(bool IsClassE); (ignored in P1 Orion)  SetOCOutputs(int Bits);  SetADCOptions(bool Dither, bool Random); (PGA is always 0 in P1 Orion)  SetAlexRXAnt(int bits);  SetAlexRXOut(bool Enable);  SetAlexTXAnt(int bits);  SetDuplex (bool Enabled); (Affects how DDCs are used: if number of DDCs==0 and duplex == 0), DDC0 gets set to TX frequency  SetNumP1DDC(int Count);  EnablePPSStamp(bool Enabled); // ignored by Orion mk2 and this hardware. |
| 0000001x | TX frequency (Hz) | SetDUCFrequency(0, int Value, false); |
| 0000010x | DDC0 frequency (Hz) | SetDDCFrequency(0, int Value, false); |
| 0000011x | DDC1 frequency (Hz) | SetDDCFrequency(1, int Value, false); |
| 0000100x | DDC2 frequency (Hz) | SetDDCFrequency(2, int Value, false); |
| 0000101x | DDC3 frequency (Hz) | SetDDCFrequency(3, int Value, false); |
| 0000110x | DDC4 frequency (Hz) | SetDDCFrequency(4, int Value, false); |
| 0000111x | DDC5 frequency (Hz) | SetDDCFrequency(5, int Value, false); |
| 0001000x | DDC6 frequency (Hz) | SetDDCFrequency(6, int Value, false); |
| 0001001x | TX drive level; mic boost; mic/line in;Apollo controls; manual/auto filter select; Alex RX1 filters; Alex disable T/R relay; Alex TX filters; set apollo bits Note TX drive level is used to set analogue (PWM DAC) drive level AND 0.5dB step digital attenuator level. Orion uses ROMs for this! | For Orion mk2: see orion.v line 2465  SetTXDriveLevel(int Level);  SetMicBoost(bool EnableBoost);  SelectMicLineInput(bool IsLineIn);  EnableAlexManualFilterSelect(bool IsManual);  AlexManualRXFilters(int Bits, int RX);  DisableAlexTRRelay(bool IsDisabled); if this bit is 0, C122\_TR\_Relay is set to 0. If this bit == 1, C122\_TR\_Relay driven by PTT output.  AlexManualTXFilters(int Bits);  SetApolloBits(bool EnableFilter, bool EnableTuner, bool StartAutoTune);  SelectFilterBoard(bool IsApollo);  Ignore apollo controls; orionmk2 chooses its register outputs between apollo & alex, but I have no hw for apollo |
| 0001010x | mic tip/ring select; mic bias; mic PTT; codec line in gain; puresignal enable; ADC1 atten; this puresignal bit is ignored! Uses C2.6 when C0=0010010x | OrionMicOptions(bool MicTip, bool EnableBias, bool EnablePTT);  SetCodecLineInGain(int Gain);  EnablePureSignal(bool Enabled); // ignored by orion mk2  SetADCAttenuator(0, int Atten, bool Enabled); // if not enabled, set atten=0 |
| 0001011x | ADC2 atten; ADC3 atten; CW keys reversed; keyer speed, keyer mode, keyer weight, keyer spacing | SetADCAttenuator(1, int Atten, bool Enabled); SetADCAttenuator(2, int Atten, bool Enabled);  SetCWKeyerReversed(bool Reversed);  SetCWKeyerSpeed(int Speed);  SetCWKeyerMode(int Mode);  SetCWKeyerWeight(int Weight);  SetCWKeyerEnabled(bool Enabled); |
| 0001100x 0001101x | Reserved – no data |  |
| 0001110x | ADC assignment; ADC atten during TX | SetDDCADC(0, ADC);  SetDDCADC(1, ADC);  SetDDCADC(2, ADC);  SetDDCADC(3, ADC);  SetDDCADC(4, ADC);  SetDDCADC(5, ADC);  SetDDCADC(6, ADC);  SetADCAttenDuringTX(Atten); |
| 0001111x | CW enable; CW sidetone volume; CW PTT delay | EnableCW (bool Enabled);  SetCWSidetoneVol(int Volume);  SetCWPTTDelay(int Delay); |
| 0010000x | CW hang time, CW sidetone frequency; | SetCWHangTime(int HangTime);  SetCWSidetoneFrequency(int Frequency); |
| 0010001x | PWM min, max pulse width | SetMinPWMWidth(int Width);  SetMaxPWMWidth(int Width); |
| 0010010x | RX2 filters; transverter enable; puresignal enable | AlexManualRXFilters(int Bits, 1);  SetXvtrEnable(bool Enabled); //CTRL\_TRSW set to XVTR\_Enable && FPGA\_PTT  EnablePureSignal(bool Enabled); // this bit selects DDC4 to TX frequency rather than its programmed frequency, and selects its input data to be TX DAC data  FPGA\_PTT = (C0 bit 0 | CW Keyer PTT) && Debounced\_IO5 (IO5 is a TX enable) |
|  |  |  |

### Data To PC

|  |  |  |
| --- | --- | --- |
| C0 | Data | Function Calls |
| C0 bits 2:0 | always carry PTT, Dash, Dot bits | GetPTTInput();  GetKeyerDashInput();  GetKeyerDotInput(); |
| 00000xxx | ADC0 overflow; IO1-IO4; | GetADCOverflow(int ADC);  GetUserIOBits(); |
| 00001xxx | Exciter fwd power, RF fwd power; | GetAnalogueIn(5);  GetAnalogueIn(1); |
| 00010xxx | Reverse power, AIN3 | GetAnalogueIn(2);  GetAnalogueIn(3); |
| 00011xxx | AIN4, 13.8v supply; | GetAnalogueIn(4);  GetAnalogueIn(6); |
| 00100xxx | ADC overflows | GetADCOverflow(1);  GetADCOverflow(2);  GetADCOverflow(3);  GetADCOverflow(4); |

## Protocol 2

### Data From PC

|  |  |  |
| --- | --- | --- |
| **Packet, Port** | **Data** | **Function Calls** |
| General Packet to SDR (1024)  Note that other packets also come to port 1024 – Discovery, Erase, Program, Set IP Address | Port addresses  Wideband enable  Wideband samples per packet  Wideband sample size  Wideband update rate  Wideband packets per frame  Min, max envelope PWM  Enables: time stamp, VITA49, VNA mode, freq or phase word  Enable hardware timer  Endian & I/Q data format  Alex Enable  PA enable | Port addresses aren’t a hardware setting  SetWidebandEnable(int ADC);  SetWidebandSampleCount(int Samples);  SetWidebandSampleSize(int Bits);  SetWidebandUpdateRate(int Period\_ms);  SetWidebandPacketsPerFrame(int Count);  SetMinPWMWidth(int Width);  SetMaxPWMWidth(int Width);  EnableTimeStamp(bool Enabled);  EnableVITA49(bool Enabled);  SetFreqPhaseWord(bool IsPhase);  SetDataEndian(int Bits);  SetAlexEnable(int Alex);  SetPAEnabled(bool Enabled); |
| DDC Specific | DDC Enables  ADC, sample rate, sample rate for each, dither  DDC  DDC synch | Some of these are done differently from protocol 1! |
| TX Specific | Number of DACs  Enables – EER, CW, CK key swap, iambic, sidetone, mode A/B, strict spacing, Break-in  Keyer speed, keyer weight, CW hang delay, RF delay  DUC sample rate, sample size, phase shift  Line in, mic boost, mic PTT, mic tip/ring, mic bias  Line in gain | SetTXDACCount(int Count);  SetClassEPA(bool enabled);  EnableCW(bool Enabled);  SetCWKeyerReversed(bool Reversed)  SetCWKeyerSpeed(int Speed)  SetCWKeyerMode(int Mode)  SetCWKeyerWeight(int Weight)  SetCWKeyerEnabled(bool Enabled)  SetCWSidetoneEnabled(bool Enabled);  SetCWBreakInEnabled(bool Enabled);  SetCWKeyerSpeed(int Speed);  SetCWKeyerWeight(int Weight);  SetCWHangTime(int HangTime);  SetCWPTTDelay(int Delay);  SetDUCSampleRate(int Bits);  SetDUCSampleSize(int Bits);  SetDUCPhaseShift(int Value);  SetMicBoost(bool EnableBoost);  SetMicLineInput(bool IsLineIn);  SetOrionMicOptions(bool MicTip, bool EnableBias, bool EnablePTT);  SetCodecLineInGain(int Gain); |
| High Priority Data | Run/Standby  PTT  Enable host CW, dash, dot;  DDC phase words;  DUC phase words;  DUC drive level;  Transverter enable;  Audio mute enable;  Open collector outputs;  User output bits;  Alex output bits;  RX step attenuators | SetOperateMode(bool IsRunMode);  SetMOX(bool Mox);  SetCWKeys(bool CWXMode, bool Dash, bool Dot);  SetDDCFrequency(int DDC, int Value, bool IsDeltaPhase);  SetDUCFrequency(int DUC, int Value, bool IsDeltaPhase);  SetTXDriveLevel(int DAC, int Level);  SetXvtrEnable(bool Enabled);  SetTXMute(bool IsMuted);  SetOCOutputs(int Bits);  SetUserOutputBits(int Bits);  SetAlexOutputBits(int Alex, int Bits);  SetADCAttenuator((int ADC, int Atten, bool Enabled); |
| DDC Audio | Left/right audio samples packet |  |
| DUC I/Q | TX IQ samples (can include envelope data if EET or EER mode selected) |  |

### Data To PC

|  |  |  |
| --- | --- | --- |
| **Packet, Port** | **Data** | **Function Calls** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Data Setting Function Calls

void SetDDCFrequency(int DDC, int Value, bool IsDeltaPhase)

void SetDUCFrequency(int DUC, int Value, bool IsDeltaPhase) // only accepts DUC=0

void SetMOX(bool Mox)

void Set P1SampleRate(int Rate)

void SetClassEPA(bool IsClassE)

void SetOCOutputs(int Bits)

void SetADCOptions(bool Dither, bool Random)

void SetAlexRXAnt(int bits)

void SetAlexRXOut(bool Enable)

void SetAlexTXAnt(int bits)

void EnableAlexManualFilterSelect(bool IsManual)

void AlexManualRXFilters(int Bits, int RX)

void DisableAlexTRRelay(bool IsDisabled)

void AlexManualTXFilters(int Bits)

void SetDuplex (bool Enabled)

void SetNumP1DDC(int Count)

void EnablePPSStamp(bool Enabled)

void SetTXDriveLevel(int Dac, int Level)

void SetMicBoost(bool EnableBoost)

void SetMicLineInput(bool IsLineIn)

void SetOrionMicOptions(bool MicTip, bool EnableBias, bool EnablePTT)

void SetCodecLineInGain(int Gain)

void EnablePureSignal(bool Enabled)

void SetADCAttenuator(int ADC, int Atten, bool Enabled)

void SetADCAttenDuringTX(int Atten)

void SetCWKeyerReversed(bool Reversed)

void SetCWKeyerSpeed(int Speed)

void SetCWKeyerMode(int Mode)

void SetCWKeyerWeight(int Weight)

void SetCWKeyerEnabled(bool Enabled)

void SetDDCADC(int DDC, int ADC)

void EnableCW (bool Enabled)

void SetCWSidetoneVol(int Volume)

void SetCWPTTDelay(int Delay)

void SetCWHangTime(int HangTime)

void SetCWSidetoneFrequency(int Frequency)

void SetCWSidetoneEnabled(bool Enabled)

void SetCWBreakInEnabled(bool Enabled);

void SetMinPWMWidth(int Width)

void SetMaxPWMWidth(int Width)

void SetXvtrEnable(bool Enabled)

void SetWidebandEnable(int ADC);

void SetWidebandSampleCount(int Samples);

void SetWidebandSampleSize(int Bits);

void SetWidebandUpdateRate(int Period\_ms);

void SetWidebandPacketsPerFrame(int Count);

void EnableTimeStamp(bool Enabled);

void EnableVITA49(bool Enabled);

void SetFreqPhaseWord(bool IsPhase);

void SetDataEndian(int Bits);

void SetAlexEnabled(int Alex);

void SetPAEnabled(bool Enabled);

void SetTXDACCount(int Count);

void SetDUCSampleRate(int Bits);

void SetDUCSampleSize(int Bits);

void SetDUCPhaseShift(int Value);

void SetOperateMode(bool IsRunMode);

void SetCWKeys(bool CWXMode, bool Dash, bool Dot);

void SetTXMute(bool IsMuted);

void SetUserOutputBits(int Bits);

bool GetPTTInput();

bool GetKeyerDashInput();

bool GetKeyerDotInput();

bool GetADCOverflow(int ADC);

int GetUserIOBits();

int GetAnalogueIn();

# Software Monitored Data

|  |  |  |
| --- | --- | --- |
| Data Item | Register Address | Action Needed |
| DDC0/1 Config |  |  |
| DDC2/3 Config |  |  |
| DDC4/5 Config |  |  |
| DDC6/7 Config |  |  |
| DDC8/9 Config |  |  |
| RX Test DDS Frequency |  |  |
| Processor LED |  |  |
| Status |  |  |
| Version Register |  |  |
| ADC Overflow |  |  |
| FIFO Monitor 0 |  |  |
| FIFO Monitor 1 |  |  |
| FIFO Monitor 2 |  |  |
| FIFO Monitor 3 |  |  |
| CW Keyer Ramp BRAM |  |  |
| Config Prom SPI |  |  |
| Codec I2C |  |  |
| XADC |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# AlexSettings

For P2, the complete registers are copied across. For P1, the mapping is as follows

|  |  |  |  |
| --- | --- | --- | --- |
| **16 bit Alex TX register:** | | | |
| **Bit** | **Function** | **Notes** | **Source** |
| 0 | 0 |  | const 0 |
| 1 | 0 |  | const 0 |
| 2 | TXRX\_STATUS | unsure | TXRX\_Relay strobe |
| 3 | LED D6 | Yellow LED | RX2\_GROUND: from C0=0x24: C1[7] |
| 4 | LPF0 | 20-30m LPF | LPF[0] : from C0=0x12: C4[0] |
| 5 | LPF1 | 40-60m LPF | LPF[1] : from C0=0x12: C4[1] |
| 6 | LPF2 | 80m LPF | LPF[2] : from C0=0x12: C4[2] |
| 7 | LPF3 | 160m LPF | LPF[3] : from C0=0x12: C4[3] |
| 8 | ANT1 |  | Gate from C0=0:C4[1:0]=00 |
| 9 | ANT2 |  | Gate from C0=0:C4[1:0]=01 |
| 10 | ANT3 |  | Gate from C0=0:C4[1:0]=10 |
| 11 | TXRX\_RELAY | T/R relay. 1=TX | TXRX\_Relay strobe |
| 12 | LED-D7 | Red LED | TXRX\_Relay strobe |
| 13 | LPF4 | 6m LPF | LPF[4] : from C0=0x12: C4[4] |
| 14 | LPF5 | 10-10m LPF | LPF[5] : from C0=0x12: C4[5] |
| 15 | LPF6 | 15-17m LPF | LPF[6] : from C0=0x12: C4[6] |

(Note bit 112 isn’t a CPU register and is provided by a hardwired TX/RX signal)

|  |  |  |  |
| --- | --- | --- | --- |
| **32 bit Alex RX register: (bits 15:0 are RX1; bits 31:16 are RX2)** | | | |
| **Bit** | **Function** | **Notes** | **Source** |
| 0 | YELLOWLED |  | const 0 |
| 1 | 13HPF | 10-22MHz BPF | BPF[0]: from C0=0x12: C3[0] |
| 2 | 20HPF | 22-35MHz BPF | BPF[1]: from C0=0x12: C3[1] |
| 3 | 6MLNA | 50MHz BPF&LNA | 10/6M LNA: from C0=0x12: C3[6] |
| 4 | 9.5HPF | 6-10MHz BPF | BPF[2]: from C0=0x12: C3[2] |
| 5 | 6.5HPF | 2.5-6MHz BPF | BPF[3]: from C0=0x12: C3[3] |
| 6 | 1.5HPF | 1-2.5MHz BPF | BPF[4]: from C0=0x12: C3[4] |
| 7 | N/A |  | const 0 |
| 8 | XVTR RELAY | Transverter in | Gated C122\_Transverter. True if C0=0: C3[6:5]=11 |
| 9 | EXT1 RELAY | Ext 1 in | Gated C122\_Rx\_2\_in. True if C0=0: C3[6:5]=10 |
| 10 | N/A |  | const 0 |
| 11 | RX BYPASS RELAY | PS select: Selects main or RX\_BYPASS\_OUT | Gated C122\_Rx\_1\_in True if C0=0: C3[6:5]=01 |
| 12 | HPF\_BYPASS | RX1 Filter bypass | BPF[5]: from C0=0x12: C3[5] |
| 13 | N/A |  | const 0 |
| 14 | RX MASTER IN RELAY | (selects main, or transverter/ext1) | Gated. True if C0=0: C3[6:5]=11 or C0=0: C3[6:5]=10 |
| 15 | REDLED |  | const 0 |
| 16 | YELLOWLED 2 |  | const 0 |
| 17 | 13HPF 2 | 10-22MHz BPF | BPF2[0]: from C0=0x24: C1[0] |
| 18 | 20HPF 2 | 22-35MHz BPF | BPF2[1]: from C0=0x24: C1[1] |
| 19 | 6MLNA 2 | 50MHz BPF & LNA | 10/6M LNA2: from C0=0x24: C1[6] |
| 20 | 9.5HPF 2 | 6-10MHz BPF | BPF2[2]: from C0=0x24: C1[2] |
| 21 | 6.5HPF 2 | 2.5-6MHz BPF | BPF2[3]: from C0=0x24: C1[3] |
| 22 | 1.5HPF 2 | 1-2.5MHz BPF | BPF2[4]: from C0=0x24: C1[4] |
| 23 | N/A |  | const 0 |
| 24 | RX2\_GROUND | When 1, RX2 i/p disconnected | RX2\_GROUND: from C0=0x24: C1[7] |
| 25 | N/A |  | const 0 |
| 26 | N/A |  | const 0 |
| 27 | N/A |  | const 0 |
| 28 | HPF\_BYPASS 2 | RX2 filter bypass | BPF2[5]: from C0=0x24: C1[5] |
| 29 | N/A |  | const 0 |
| 30 | N/A |  | const 0 |
| 31 | REDLED 2 |  | const 0 |