PDIUSBD11 – USB Peripheral with I^2C Serial Interface

Features

The PDIUSBD11 USB Device with Serial Interface from Philips Semiconductor allows almost any microcontroller the option of having a USB Interface. Being a full speed device, it allows USB transfer modes including Control, Bulk and Interrupt. It doe not support Isochronous. It's I^2C Interface can be clocked at a maximum of 1Mbit/s with a theoretical maximum transfer of 568KB/s[1], thus makes communication between the microcontroller and PDIUSBD11 quite a bit slower than the 12Mbits/s achievable with a full speed USB device.

The PDIUSBD11 is a 3.3v device with 5V tolerant I/O. Don't let this put you off when considering your design. A small low powered 3.3Volt regulator is all that is needed to interface to your 5V logic. The I/O pins are open drain, thus using pull up resistors to 5V, a 0 to 5v logic output is obtainable. Unlike other USB peripheral IC's such as National's USBN9602 which require a 48Mhz crystal, the PDIUSBD11 uses a in-built PLL to derive it's internal 48MHz from a 12MHz Crystal. Not only does this make it cheaper as 48MHz crystals are hard to obtain, but it also helps reduce EMI.

But with all these positives, there must be some negatives. Lack of documentation is one. Philips gives no sample circuits in their data sheet, makes little effort to describe any supporting passive components around it and assumes you will consult the USB Specification for most data. After numerous contacts, and reading between the lines a basic circuit can be sought. What is more worrying is the software. Philips has left out critical initialisation information regarding the disabling of the HUB, and has repeatedly specified wrong commands for the clearing of interrupts. It would be almost impossible to get the PDIUSBD11 going from just the data sheet alone.

Oscillator

As discussed, the clock is generated by a 12MHz Crystal. The data sheet suggests that no external components other than the crystal are needed. This is true in many cases, however Guy Jaumotte from Philip Semiconductor states they are not needed, but are used to guarantee start-up. It's definitely much easier to add them rather than argue and have oscillator problems later down the track. When designing a board, it's wise to place pads for the capacitors even if you do not add them during manufacture.

USB Termination Resistors

The data sheet would suggest some series termination resistors are required to connect the transceiver to the USB Cable. This is the classic case where Philips expects you to look up the USB Spec, with little knowledge of what's actually in the PDIUSBD11. 22ohm ±1% resistors have been used in this example, but it's suggested that anything from 22ohms to 44ohms can be used. Their purpose is impedance matching of the bus. The analog input pins, have an internal pull down resistor of approximately 15K and a software selectable pull up resistor to D+, known as SoftConnect™.

V_{BUS}

V_{BUS} is used to detect a connection to the USB Bus. Without a presence, the PDIUSBD11 doesn't generate any interrupts or returns a status. V_{BUS} is also used to enable SoftConnect should SoftConnect be enabled using the Set Mode command. The PDIUSBD11, being 5V tolerant will allow the V_{BUS} pin to be connected directly to the 5V Bus power supply. However being a 3.3V device, it's wise to use a voltage divider network to obtain 3.3v for the V_{BUS} Pin. However as we will discover complications will result in suspend.

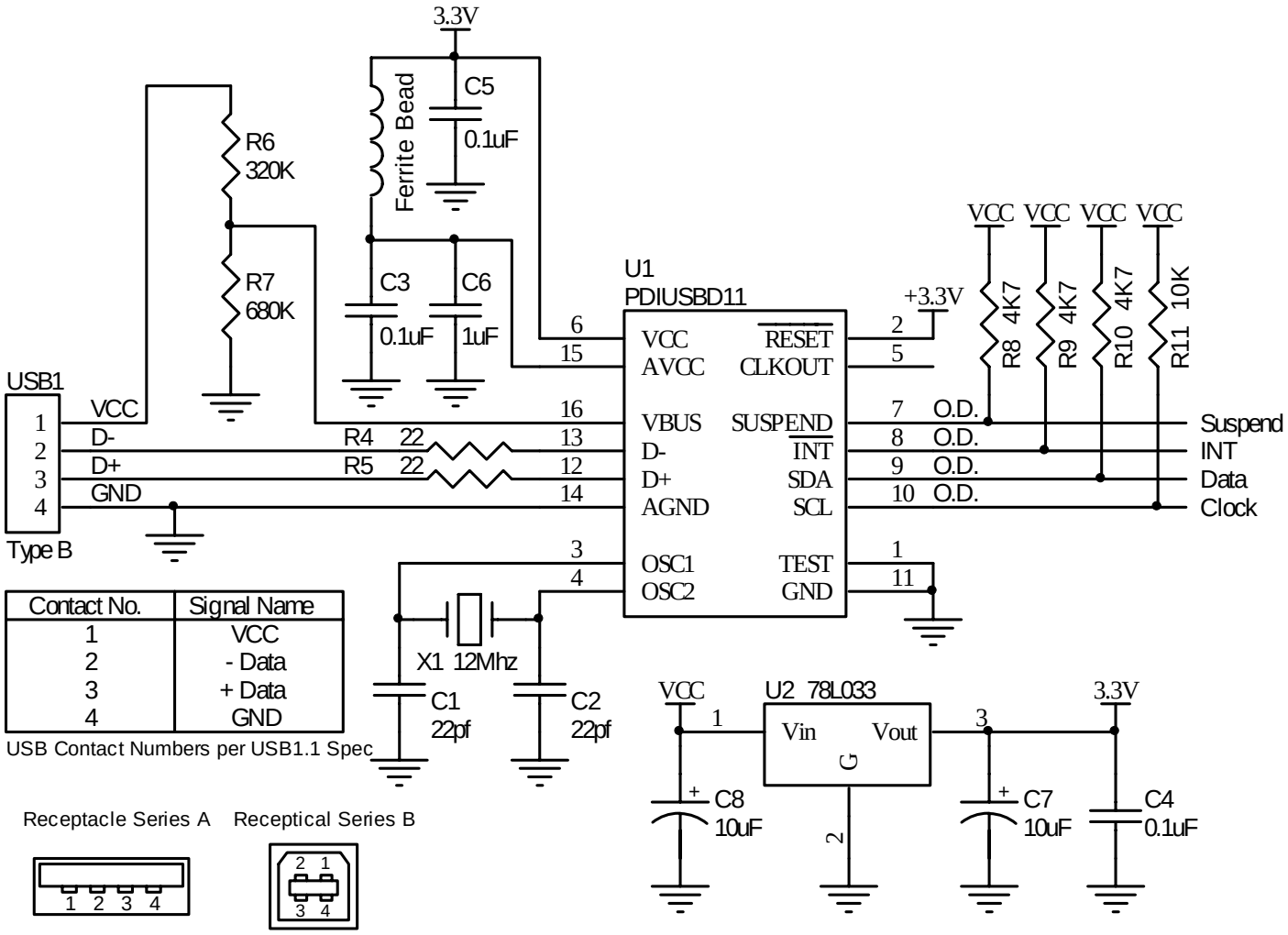
"The PDIUSBD11 is 5.5 Volt tolerant (see Data sheet) so you can connect directly the Bus power supply line to the VBUS pin. However the device Vcc is 3.3 Volt so it make sense to connect the Bus power line via a divider network. Any combination of resistor will do providing that take into account the maximum voltage drop allowed on Vbus(See USB specs), the minimum voltage that is recognised as a HIGH in 3.3Volt technology (approx 2Volt) and the maximum consumption allowed in suspend mode (500microAmp). In a Self power mode, the pin consumes 1 microAmp and you have to take into account the bus divider network consumption [Vbus - 1.5KOhm - D+ - 15KOhm - GND]"

Guy.Jaumotte@Philips.com

PDIUSBD11 Pin Descriptions

Pin	Pin Name	Type	Description
1	Test	Input	For normal operation connect to ground.
2	Reset_N	Schmitt Trigger Input	Reset. PDIUSBD11 provides internal reset circuit, thus may be tied directly to VCC should an external reset not be needed.
3	XTAL1	Input	Connect to 12Mhz Crystal. While the PDIUSBD11 is designed not to require any additional circuitry, a capacitor (~22pF) to ground will help guarantee start-up.
4	XTAL2	Output	Connect to 12Mhz Crystal. While the PDIUSBD11 is designed not to require any additional circuitry, a capacitor (~22pF) to ground will help guarantee start-up.
5	ClkOut	3mA Output	Programmable Clock Output. This output defaults to 4MHz which can be used to clock a microcontroller. This pin will stabilise after 1mS from power-on and 320μS after suspend. If this output is used to control a microcontroller, care should be taken to ensure the μC comes out of reset after 1mS by using a suitably timed RC network.
6	V_{CC}	Power	Connect to a 3.3V supply with a tolerance of ± 0.3Volts.
7	Suspend	Bi-Directional OD 6mA Sink	This pin is bi-directional, not an output only as the PDIUSBD11 data sheet states. This line can be tied low to prevent the PDIUSBD11 from going into suspend, or pulled low during suspend to wake up the PDIUSBD11.
8	INT_N	OD Out 6mA Sink	Interrupt Output. Interrupt is level sensitive and will go low on an interrupt occurring and return high once **all** interrupts have been cleared. Most microcontrollers will accept only an Edge Sensitive Interrupt, thus care should be taken at the end of the Interrupt Service Routine, to check that all interrupts have been dealt with before returning from interrupt. The other option is to poll the INT_N Pin.
9	SDA	OD I/O 6mA Sink	I^2C Serial Data. Bi-directional pin. Use a pull up resistor if talking directly to a μC or follow the I^2C Specifications for connection to an I^2C Bus.
10	SCL	OD I/O 6mA Sink	I^2C Serial Clock. Bi-directional pin. Use a pull up resistor if talking directly to a μC or follow the I^2C Specifications for connection to an I^2C Bus.
11	GND	Power	Ground.
12	DP	AI/O	USB D+ Connection. Series termination resistors (22Ω ± 1%) are required for impedance matching of USB Bus. The USB Spec 1.1 states that the impedance of each driver is required to be between 28 and 44Ω. The PDIUSBD11's Drive Output Resistance is 29Ω to 44Ω max provided 22Ω ±1% series resistors are used.
13	DM	AI/O	USB D- Connection. Series termination resistors (22Ω ± 1%) are required for impedance matching of USB Bus. The USB Spec 1.1 states that the impedance of each driver is required to be between 28 and 44Ω. The PDIUSBD11's Drive Output Resistance is 29Ω to 44Ω max provided 22Ω ±1% series resistors are used.
14	AGND	Power	Analog ground
15	AV_{CC}	Power	Analog Supply 3.3V ± 0.3Volts. Normally connect to V_{CC} through some suppression to isolate any digital noise.
16	V_{BUS}	Input	USB Power Sense. Full speed devices are identified by pulling D+ to 3.3V ± 0.3 Volts via a 1.5kΩ ± 5% resistor. This is build into the PDIUSBD11 as part of Philip's SoftConnect™ Technology. However any USB device is not allowed to supply power to the USB Data lines when the power has been removed, Thus the requirement of the PDIUSBD11 to sense the USB Power line. While this pin is 5V tolerant, it is normally connected to V_{BUS} via a divider network to reduce it to 3.3volts. Recommended values are 680kΩ & 320kΩ.

Example Schematic



SoftConnect™

Note that the USB specification calls for a 1.5kΩ ± 5% (1425Ω to 1575Ω) pull up resistor on D+. Philips specifies a SoftConnect pull up resistor with a range of 1.1kΩ minimum and a maximum of 1.9kΩ which has a tolerance closer to ± 30%, than the USB Spec's 5%. Philips specifies this in their data sheet ensuring the design engineer that V_{SE} voltage specification can still be meet and the end designer lies with the option of using it or not.

Essentially if you use SoftConnect, it is out of spec and not USB compliant. You could use you own termination resistor, but you have to find a 3.3V source to connect it too, bearing in mind that the V_{BUS} is 5 volts. However as you will need a 3.3 Volt ± 0.3V supply for the PDIUSBD11 in bus powered designs this should not be a problem. The other consequence of using an external pull up resistor is that it can only be connected when V_{BUS} is present. The USB specification states that no power can be applied to the data lines in the absence of $V_{BUS.}$This is once again not a problem for bus powered designs as you can still connect it to your regulated 3.3V supply which is derived from V_{BUS}

Self Powered or Bus Powered? Current Budgeting

The PDIUSBD11 doesn't mention much about power consumption. Power consumption is big business with USB, if you draw too much current, you violate the USB Spec. The PDIUSBD11 draws around 25mA during normal operation. The data sheet specifies no minimum, maximum or typical values for power consumption in fully operational nor suspends states!

A USB device specifies its power consumption expressed in 2mA units in the configuration descriptor. A device cannot increase its power consumption, greater than what it specifies during enumeration, even if it loses external power. There are three classes of USB functions, low power bus powered functions, high power bus powered functions, and self powered functions.

Low power bus powered functions draw all it's power from the V_{BUS} and cannot draw any more than one unit load. USB defines a unit load as 100mA. High power bus powered functions draw all it's power from the bus and cannot draw more than one unit load until they have been configured, after which it can then drain 5 unit loads (500mA Max)

Self power functions may draw up to 1 unit load from the bus and derive the rest of it's power from an external source. Should this external source fail, it must have provisions in place to draw no more than 1 unit load from the bus. Self powered functions are easier to design to spec as there is not so much of an issue with power consumption.

Bus Powered

During initialisation and enumeration the maximum power drain that USB 1.1 permits is 100mA. As the PDIUSBD11 consumes approximately 25mA, there is a 75mA excess for the microcontroller and support circuitry. A low powered device must be capable of operating on a minimum 4.40V to maximum 5.25v at the plug of the USB device.

However during suspend, additional constrains come into force. The maximum suspend current is proportional to the unit load. For a 1 unit load devices (default) the maximum suspend current is 500uA. This includes current from the pull up and pull down resistors on the bus. The PDIUSBD11 drains approximately 210uA during suspend. Of this approximately 200uA is due to the internal pull up resistor on D+. This leaves approximately 290uA to play with.

However this is dependent on what we have done with V_{BUS}. If you have used the 680KΩ/320KΩ voltage divider then you are sinking 5µA into the divider network. The input leakage current of V_{BUS} being a digital I/O pin is 5µA max. Guy Jaumotte suggests this figure is closer to 1µA typical. Another consideration is the required 3.3V regulator for bus powered designs.

Suspend Mode

Every USB device must support suspend. However this is another area the data sheet avoids.

The PDIUSBD11 can enter suspend mode in various ways such as,

- Selective Suspend – Host sends a suspend command to the attached port.
- Global Suspend – The host suspends its self.
- No activity on bus for more than 3 SOF ~ 3mS.

Each 1mS a SOF (Start of Frame) packet should be sent on the USB. This is the responsibility of the host. When the bus goes into suspend, the SOF every 1mS will seize to exist. The PDIUSBD11 will wait for 3mS without the presence of a SOF. It will then allow its Suspend pin to go high, signalling to the microcontroller that it is about to enter the suspend state and to stop finishing any processing and go to sleep. The entire USB device (PDIUSBD11, MicroController and Support Circuitry) must not drain anymore than 500uA from V_{BUS} when in suspend. This of course is not so much of an issue for Self Powered devices. 1mS after the suspend pin goes low, the ClockOut will go into Lazy Clock Output if this feature is selected. These features are selected by the Configuration Byte as detailed below.

Configuration Byte		Selected Configuration	Current Consumed
No Lazy Clock (Bit 1)	Clock Running (Bit 2)		
0	0	ClockOut will switch to Lazy Clock Mode (Frequency 30KHz ± 40%) 1mS after suspend pin goes high. Internal Clock, Crystal Oscillator and PLL will stop during suspend, consuming less power.	~2.5mA (External µC will run at slower speed, reducing current if connected)
0	1	ClockOut will switch to Lazy Clock Mode (Frequency 30KHz ± 40%) 1mS after suspend pin goes high. Internal Clock. Crystal Oscillator and PLL are always running regardless of suspend mode.	~25mA External µC will run at slower speed, reducing current if connected)
1	0	Internal Clock, Crystal Oscillator and PLL is stopped during suspend as a result, ClockOut seizes to run.	~210µA (Assuming SoftConnect™ Active)
1	1	ClockOut stays at its original speed. The Internal Clock, Crystal Oscillator and PLL are always running regardless of suspend mode.	~25mA (+ Full load of attached µC)

The PDIUSBD11 can come out of suspend in two ways

- Pull Suspend Pin Low.
- Resume Signal from Bus (Downstream to Device)

The data sheet is rather misleading in this area. With the suspend pin specified as an output only and the resume command saying “This command is normally issued when the device is in suspend”, one would assume you would simply send the Resume Command. This is not the case.

If the microcontroller wishes to wake up the PDIUSBD11, it pulls the suspend pin low. The PDIUSBD11 will come out of suspend, but can’t talk on the Bus as there are no SOF Packets every 1mS. To wake up the Host, the Send Resume command is then sent to the PDIUSBD11.

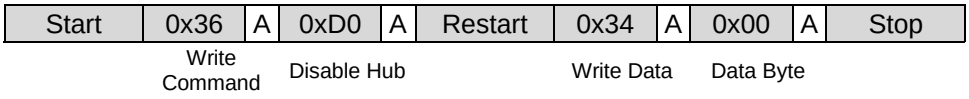
I^2C Protocol Interface

The PDIUSBD11 uses the Philips I^2C protocol which can be a little daunting if you have never used it before. Unlike other serial buses such as SPI and Microwire which have individual chip selects, I^2C sends an address down the bus after a start condition. Only the device which has the matching address will respond to the following commands until either a restart condition or a stop/start condition is generated again. As I^2C is multi-master, a restart condition will allow the host to re-issue an address without giving up its control of the bus. Sending a stop condition means the device no longer requires the bus and should another device want to talk on the bus, it can generate a start condition and take over as master.

Instead of the PDIUSBD11 having only one address as you would expect, the PDIUSBD11 has “three” addresses for simplicity. This allows information about the data to follow e.g. is it command/data or read/write to be efficiently encoded into the address. This is not entirely correct in the context of the I^2C specification, as it has two addresses (Command/Data) and a direction bit (LSBit) which specifies read/write operations as shown below.

Function	I^2Caddress	D	Combined “Address”
Command Write	0011 011	0	0x36
Data Write	0011 010	0	0x34
Data Read	0011 010	1	0x35

An example Write Command Cycle (Disable Hub Address) is shown below.

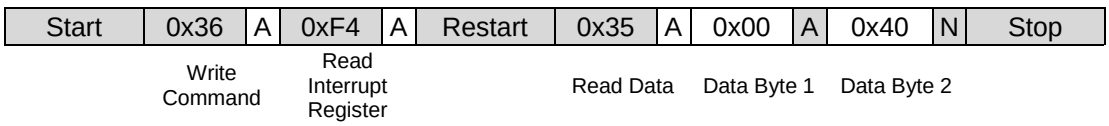


Multiple commands can be sent after a write command.

Key	
	Master to PDIUSBD11
	PDIUSBD11 to Master
A	Ack
N	Negative Ack

Complicating matters further, when reading the PDIUSBD11, it must place its serial line into input mode. However as multiple reads can be performed after the one “Command Read” address, it has no way of telling when to release the bus. Therefore the master must acknowledge all bytes except for the last byte which it must return an negative acknowledge.

An example Read Command Cycle (Read Interrupt Register) is shown as follows,



Initialisation

The wafer of the PDIUSBD11 is the same in both the HUB (PDIUSBH11A) and Device/Function chips (PDIUSBD11). What Philips Semiconductor omits from the data sheet is that you need to disable the hub before you can use the embedded function. Failure to do so, will result in setup packets being received on the HUB's default port and not the embedded functions. They do however mention the initialisation in their FAQ but have once again missed it in their latest revision of the data sheet, dated 22 July 1999. (Before that, one could off assumed they lost the original/editable data sheet!)

"The PDIUSBD11 is effectively the embedded function 1 of a PDIUSBH11A(HuB). We have taken the silicon of the PDIUSBH11A and bounded the embedded function 1. This has for consequence that the HUB is still present and active inside the PDIUSBD11. The Hub part MUST be disabled at power-on and AFTER Bus reset by sending the command 0xD0(Set Address(Hub)) and writing the data 0x00(Address 0 disabled). The same can be done for the hub endpoints." Guy.Jaumotte@Philips.com

Another thing you must note, is that the HUB is re-enabled on a Bus Reset. It is therefore, necessary to disable the hub and enable the Embedded Function every time a Bus Reset Interrupt occurs.

The recommended Initialisation sequence is,

```
/* Disable Hub Function in PDIUSBD11 */
SET_HUB_ADDRESS(0xD0) to 0x00

/* Set Address to zero (default) and enable function */
SET_ADDRESS_ENABLE(0xD1) to 0x80

/* Enable function generic endpoints */
SET_ENDPOINT_ENABLE(0xD8) to 0x02

/* Set Mode - Enable SoftConnect */
SET_MODE(0xF3) to 0x97; /* Embedded Function, SoftConnect, Clk Run, */
                        /* No LazyClk, Remote Wakeup */
    and byte 2 to 0x0B; /* CLKOut = 4MHz */
```

Notes :

(1) http://www-eu3.semiconductors.com/usb/products/interface/pdiusbd11/faq/#2.1 PDIUSBD11 FAQ

(2) SoftConnect™ is a patent pending technology from Philips Semiconductors

PDIUSBD11 Command Summary

Set Address / Enable

Command **0xD0** Set Hub Address
0xD1 Set Embedded Function 1's Address
0xD2 *Embedded Function Two (PDIUSBH11)*
0xD3 *Embedded Function Three (PDIUSBH11)*

Data Command followed by a Write of One Data Byte with the format below.

7	6	5	4	3	2	1	0
Enable	**Address**						

This command will enable the desired function (bit 7) and set its address. A '1' in bit 7 enables the function. The low seven bits are used to set the function's address. When first powered up, an address of zero is used, until the Host issues the Set Address Device Request (Chapter 9 USB Spec) during enumeration.

The PDIUSBD11 contains the same silicon than the Philips PDIUSBH11A HUB. As a result, the HUB powers up enabled and thus needs to be **turned off at initialisation** and **after** a Bus Reset.

Set Endpoint Enable

Command **0xD8** Set Endpoint Enable

Data Command followed by a Write of One Data Byte with the format below

7	6	5	4	3	2	1	0	
X	X	X	X	0	0	0	0	Power on Reset
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	**Generic Endpoint Enable**	Reserved	

When the function is enabled, only its Default Control Pipe (Endpoint 2 & 3) are enabled. During enumeration, your device will describe to the host the type of Generic Endpoints it wishes to use in the form of an Endpoint Descriptor. Later during enumeration the host will send the Set Configuration Device Request (Chapter 9 USB Spec). At this point you can enable your Generic Endpoints. Note that many devices will enable the Generic Endpoints at Power Up and this does not effect the functionality of the device.

Endpoint No.	Endpoint Index	Endpoint Type	Direction
0	**2** **3**	**Control / Default**	**Out** **In**
1	**5** **4**	**Generic**	**Out** **In**
2	**6** **7**	**Generic**	**Out** **In**
3	**8** **9**	**Generic**	**Out** **In**

Note : Endpoint 1's endpoint indexes are swapped. This is not an error – Index 5 is EP1 Out, Index 4 is EP1 In

Set Mode

Command **0xF3** Set Mode

Data Command followed by writing two data bytes with format below,

Byte 1 - Configuration

7	6	5	4	3	2	1	0	
1	0	0	0	1	1	0	1	Power on Reset
Embedded Function Mode	X	X	**Soft Connect**	**Debug Mode**	**Clock Running**	**No Lazy Clock**	**Remote Wakeup**	

Remote Wakeup — Setting this bit enables the Remote Wakeup Feature. A bus reset will enable this function.

No Lazy Clock — Clearing this bit, ensures the clock will not switch to lazy clock mode (~30kHz) 1ms after Suspend. This value does not change on a Bus Reset.

Clock Running	The setting of this bit will ensure the clock & phase lock loop is always running even in suspend. Use this mode with self powered devices. Bus powered devices will need to set this bit to 0 to ensure the maximum load specifications during suspend is meet.
Debug Mode	Setting this bit will cause all errors and negative acknowledgments to be reported. If Clear only OK and babbling is shown.
Soft Connect	Setting this bit will enable the pull up resistor on D+ to be connected when V_{BUS} is present. This value is not changed by reset.
Embedded Function Mode / Future Mode	For normal operation, set this bit. The PDIUSBH11 uses this bit to enable multiple embedded functions. It has not yet be determined if this will work with the hub disabled. See the PDIUSBH11A data sheet for more details.

Byte 2 – Clock Divider

7	6	5	4	3	2	1	0	
X	X	0	0	0	0	1	1	Power on Reset
X	X	Clock Divisor						

Byte 2 sets the frequency of the clock output. Should you desire either a default 4MHz clock or do not wish to use the clock then this byte can be ignored. The power on value is 3 giving a default clock out of 4MHz which is quite common for many microcontrollers. Of course faster microcontrollers can power up on 4Mhz and then set the Mode to run at their full speed. The expected clock frequency is 48MHz/(N+1) where N is the clock divisor.

The PDIUSBD11 data sheet shows only the low nibble being used. However after accidental playing, it was found to be the same than the PDIUSBH11A, using the lowest 6 bits for the divisor. Thus care must be taken with the extra two bits. They are NOT don't cares as the data sheet would suggest.

Read Interrupt Register

Command **0xF4** Read Interrupt Register **Data** Read Two Data Bytes

Interrupt Register Byte 1

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	Power on Reset
EndPoint Index 7	Endpoint Index 6	Endpoint Index 5	Endpoint Index 4	Control In Endpoint	Control Out Endpoint	Reserved (Hub)	Reserved (Hub)	

Interrupt Register Byte 2

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	Power on Reset
Reserved	Bus Reset	Reserved	Reserved	Reserved	Reserved	Endpoint Index 8	Endpoint Index 9	

After an interrupt has occurred a read of this command will show what event caused the interrupt. Each bit is cleared by reading the "Read Last Transaction Status" [3]. However should a Bus Reset Interrupt occur, reading the Interrupt Register (0xF4) will clear this flag.

The Interrupt pin on the PDIUSBD11 is an active low signal which gets pulled low after an interrupt has occurred and remains their until all interrupts are cleared. Most microcontrollers will accept an Edge Sensitive Interrupt, however few will accept a Level Sensitive Interrupt. There are two ways of approaching this problem. What Philips does in many of their examples is simply poll the interrupt register continuously and branch to a handler should an interrupt be pending. This opens up the use of the INT pin on the microcontroller to other possibilities.

The other option is to generate an interrupt on the falling edge and when the first handler has finished, before returning from the interrupt, check if any other interrupts are still pending and if so handle these. This way, not all you idle cycles are being taken up with polling the interrupt register.

Note [3] : The PDIUSBD11 Data Sheet would suggest that the Interrupt Register is cleared by reading the "Read Endpoint Status Command". This is an error that Philips Semiconductors acknowledges in their FAQ. Please refer to http://www-eu3.semiconductors.com/usb/products/interface/pdiusbd11/faq.html#4.1 – "4.1 How is the Interrupt Flag cleared?" Also note that this error is still not fixed in their latest revision of the Data Sheet dated 22nd July 1999

Select Endpoint

Command 0x02 Select Control Out Endpoint
0x03 Select Control In Endpoint
0x04 Select Generic Endpoint 1 IN
0x05 Select Generic Endpoint 1 OUT
0x06 Select Generic Endpoint 2 OUT
0x07 Select Generic Endpoint 2 IN
0x08 Select Generic Endpoint 3 OUT
0x09 Select Generic Endpoint 3 IN

Data Read One Byte – Optional

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	Power on Reset
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Buffer Full	

This command will select the desired endpoint (Set the Internal Pointer) for a subset of commands. Changing endpoints will reset the pointer. An optional byte can be read to determine if the Endpoint Buffer selected is full or empty. This is seldom used in the interest of efficiency, as the Read Endpoint Status command will indicate if the Buffer is full plus other information. It bit 1 is set, then the Buffer is Full.

Read Last Transaction Status

Command 0x42 Read Last Transaction Status for the Control Out Endpoint
0x43 Read Last Transaction Status for the Control In Endpoint
0x44 Read Last Transaction Status for the Generic Endpoint 1 IN
0x45 Read Last Transaction Status for the Generic Endpoint 1 OUT
0x46 Read Last Transaction Status for the Generic Endpoint 2 OUT
0x47 Read Last Transaction Status for the Generic Endpoint 2 IN
0x48 Read Last Transaction Status for the Generic Endpoint 3 OUT
0x49 Read Last Transaction Status for the Generic Endpoint 3 IN

Data Read One Byte

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	Power on Reset
Previous Status not Read	Data 0/1 Packet	Setup Packet	Error Code				Data Receive Transmit Success	

This command is intended for debugging. It will return a byte showing the status of the last transaction on the requested Endpoint without resetting any internal pointers set by the Set Endpoint Command.

Code	Error
0000	No Error
0001	PID Encoding Error
0010	PID Unknown
0011	Unexpected Packet
0100	Token CRC Error
0101	Data CRC Error
0110	Time Out Error
0111	Babble Error
1000	Unexpected End of Packet
1001	Sent or Received NAK
1010	Sent Stall
1011	Overflow Error
1101	BitStuff Error
1111	Wrong DATA PID

Read Endpoint Status

Command 0x82 Read Control OUT Endpoint Status **Data** Read One Byte
0x83 Read Control IN Endpoint Status
0x84 Read Generic Endpoint 1 IN Endpoint Status
0x85 Read Generic Endpoint 1 OUT Endpoint Status
0x86 Read Generic Endpoint 2 OUT Endpoint Status
0x87 Read Generic Endpoint 2 IN Endpoint Status
0x88 Read Generic Endpoint 3 OUT Endpoint Status
0x89 Read Generic Endpoint 3 IN Endpoint Status

7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	Power on Reset
Reserved	Reserved	Buffer Full	Data 0/1 Packet	Stalled	Setup Packet	Reserved	Reserved	

Read Buffer / Write Buffer

Command **0xF0** Read Buffer **Data** Read up to 10 Bytes
0xF0 Write Buffer **Data** Write up to 10 Bytes

The same command is sent to Read or Write Data. The desired operation is selected by the data phase. The PDIUSBD11 contains an area of linear RAM segmented into Endpoint buffers. The Read or Write Commands will not set the PDIUSBD11's Internal RAM pointer to the start of the particular 8 byte buffer. This is done using the Select Endpoint Command.

After a byte has been written or read the internal pointer is incremented. Beware that there is no protection from reading or writing into the next endpoint's buffer.

The Data in the Buffer has the following format.

Offset 0	Offset 1	Offset 2	Offset 3	….	Offset 9
Reserved Undefined Value	Number of bytes to follow	Byte 1	Byte2		Byte 8

Clear Buffer

Command **0xF2** **Data** None

After a packet has been received the buffer full flag is set and the PDIUSBD11 will issue NAK to additional packets send to the endpoint until the Buffer Clear Flag is cleared. Therefore once data has been received it should be read and on completion of reading the data the clear buffer command should be issued to enable subsequent packets to be received. Failure to do so will inhibit an more packets being received on this endpoint.

Validate Buffer

Command **0xFA** **Data** None

Once data has been written to a IN Buffer, the Validate Buffer command should be set. This tells the PDIUSBD11 that the data is complete and should be sent when the next IN Token is received.

Set Endpoint Status

Command 0x42 Set Control OUT Endpoint Status
0x43 Set Control IN Endpoint Status
0x44 Set Generic Endpoint 1 IN Endpoint Status
0x45 Set Generic Endpoint 1 OUT Endpoint Status
0x46 Set Generic Endpoint 2 OUT Endpoint Status
0x47 Set Generic Endpoint 2 IN Endpoint Status
0x48 Set Generic Endpoint 3 OUT Endpoint Status
0x49 Set Generic Endpoint 3 IN Endpoint Status

Data Write one byte with the following format

7	6	5	4	3	2	1	0	
X	X	X	X	X	X	X	0	Power on Reset
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Stalled	

This command can be used to stall endpoints. Endpoints can be stalled, if they are not in use or if a command is not supported, among other reasons. A Setup Packet will be received regardless if the endpoint is stalled or not. Should the endpoint be stalled when it receives a Setup Packet, another Set Endpoint Status command will need to be sent to stall the endpoint again.

If a Zero is written to un-stall an endpoint, even if the endpoint is already un-stalled, the buffer is cleared and If the endpoint is an IN endpoint, the PDIUSBD11 will send a DATA 0 PID to the host. If the endpoint is an OUT Endpoint the PDIUSBD11 will wait for a DATA0 PID. This procedure is the same should a Setup Packet un-stall the Endpoint.

The Set Endpoint Status shares the same command numbering than the Read Last Transaction Status. The data phase will determine which command is sought after.

Acknowledge Setup

Command 0xF1 **Data** None

When a Setup Packet is received, the PDIUSBD11 will clear the Control IN Endpoint Buffer, and disable the Validate Buffer and Clear Buffer commands until the packet is acknowledged by the controller, by sending the Acknowledge Setup Command to both IN & OUT Control Endpoints.

This prevents the Setup packet from being overridden and any packets being sent back to the host.

Send Resume

Command 0xF6 **Data** None

This command will send the resume signal upstream to the hub or host. This can be used to wake the host up.

Read Current Frame Number

Command 0xF5 **Data** Read One or Two Bytes

The Read Current Frame Number can be used to return the current 16 Bit Frame Number of the last SOF received successfully. The LSByte is returned first, followed by the MSByte.

Third Release 6th April 2002
Second Release 20th December 2001
First Release 22nd January 2000 - Draft