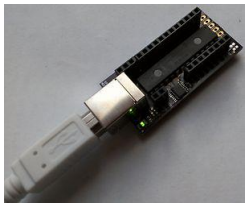
USB to SPI / I2C / ADC / GPIO Converter & Analyser

Devices

Two flavours of the device are available:

1. 28-pin PDIP IC-only package. This allows the device to be seamlessly integrated directly onto a host system PCB.

2. USB-incorporated package. This allows a user to easily communicate with target devices using a convenient USB connection directly to a host PC.



This datasheet relates to the **USB-incorporated package**.

The datasheet for the IC-only package may be found by following this link.

Contact **devices@burningimage.net** for pricing and availability.

Summary

This device is an SPI/I2C/ADC/GPIO master and is controlled via the USB port. It allows all of the following, directly from your PC:

- Communicate with **any SPI or I2C device** (EEPROMs, SD cards, RTCs, etc.). SPI and I2C devices may be communicated with simultaneously – they do not share pins.
- 8 chip select lines, allowing the hard-wiring of the device to 8 different SPI devices. These are multi-functional lines – they can also be used as GPIO digital outputs (illuminating LEDs etc.).
- 4 analog (ADC) input pins. When queried from the PC, the device will communicate the voltage on any of the four input pins (see example below).
- 5 digital input pins. When queried from the PC, the device will communicate the logic level on any one of the 4 digital inputs.
- I2C search function, for scanning an I2C bus for connected devices.
- Multiple serial baud rates.
- +3V3 and +5V outputs for directly powering target devices.
- Binary interface for high speed embedded communications.

Driver Installation

If you are using Windows Vista, Windows 7 or Windows XP then drivers should be automatically installed when you plug the lead into your USB port, assuming that you are connected to the internet. Bear in mind that this can take several minutes, so ensure that you don't unplug the lead before installation is complete.

If the drivers do not automatically install then unplug the device and download and install the following file:

http://www.ftdichip.com/Drivers/CDM/CDM20824_Setup.exe

Once the above file has installed, plug the device back in and the latest drivers will be installed.

Connections

The supplied cable should be plugged into a spare USB port on your PC.

- Pins **AD0** to **AD3** are analogue inputs. Any voltage up to 3V3 may be connected to these pins. The device will return the voltage sensed on the pins when queried with the appropriate command (see command list).
- Pins **D0** to **D5** are digital inputs. They will return either '1' or '0' depending on the logic level on the pin.
- **MISO**, **CLK** and **MOSI** are SPI bus related pins.
- **SCL** and **SDA** are I2C bus related pins. These are pulled up to 3V3, as per the I2C specification.
- **CS0** to **CS7** are digital outputs. These are designed for use as Chip Select pins for the SPI bus, but because each one can be driven independently, they can be used for any purpose (25mA each max current).

Power Supply

The USB converter generates its own internal 3V3 voltage from the USB port of your PC, so no additional power supply is required.

If you wish to use this 3V3 to power a target device (an EEPROM for example) then you can connect your target to the socket on the board labelled '3V3' (max 45mA). You can also use the '5V' socket to power a device that requires 5V (max 450mA).

Commands

There are two command sets available for this device. A 'standard' command set, based around human readable commands, intended for use with a terminal program such as Hyperterm or Tera Term.

The second command set is a binary command set. This allows fast and efficient communication with the USB converter device, intended for use within embedded applications.

Both are described in the following sections.

Note that the binary command set is only available for firmware versions >= 3.00. If you already own a device then it may be sent back to the supplier for a free firmware upgrade.

Standard Command List

The device automatically boots with the standard command set selected.

Each command must be sent to the device terminated with a 'CR' (character 10). Tera Term (see below) performs this action when the 'enter' key is pressed on a keyboard.

- **ACKI2C** – sends an I2C ACK *(only for firmware versions 1.51 and lower. Version 2.00 incorporates this into the READI2C command).*
- **AD0? –** returns the current voltage on the AD0 pin. Other valid commands are AD1?, AD2? and AD3?
- **BAUD <BAUDRATE> - *(advanced)* -** changes the UART baud rate (*only for firmware versions 1.30 and higher).* Possible values are 9600, 19200, 57600 and 115200. The change is permanent – the value persists when the device is powered off. Thus, it's **very important** that the new baud rate is remembered; otherwise it would be necessary to try all four baud rates until communication is successful. Adjusting this setting is recommended for advanced users only.
  Example: *BAUD 9600* – set the baud rate to 9600bps.
- **CLOSEI2C** – closes the I2C bus.
- **CLS** – clears the terminal screen (*only for firmware versions 1.30 and higher).*
- **CS <PIN>** - Takes the relevant CS output low.
  Example*: CS 1* – sets the CS1 pin low.
- **D0? -** returns the current logic level on the D0 pin. Other valid commands are D1?, D2?, D3? and D4?
- **NACKI2C** – sends an I2C NACK *(only for firmware versions 1.51 and lower. Version 2.00 incorporates this into the READI2CN command).*
- **NOCS <PIN>** - Takes the relevant CS output high (i.e. chip not selected)[1]
- **OPENI2C** – prepares the I2C bus for use. This command must be sent before communication is initiated over the I2C port (*only for firmware versions 1.51 and lower).*
- **OPENSPI <MODE>** - opens the SPI bus in the specified mode.
  Example: *OPENSPI 0*

[1] All CS pins are high upon device reset.

- **READI2C** – reads a byte from the I2C bus. Only call this if there is data to be read. Will timeout if no data is ready to be received from the target *(firmware versions 2.00 and above automatically send an ACK to the target device once the data has been read).*
- **READI2CN** – as above, but a NAK is sent to the target device once the data has been read *(only for firmware versions 2.00 and higher).*
- **READI2CPAGE <BYTECOUNT>** - reads a number of bytes from the target I2C device. An ACK is sent to the target device after each byte is read, except for the last byte, after which a NACK is sent (*only for firmware versions 2.10 and higher)*.
  Example: READI2CPAGE 04 – reads four bytes from the target I2C device.
- **RESTARTI2C** – sends an I2C RESTART.
- **SCANI2C** – scan the I2C bus and return the address of any device found. Note that some I2C devices support communication using multiple addresses *(only for firmware versions 1.50 and higher).*
- **STARTI2C** – sends an I2C START.
- **STOPI2C** – sends an I2C STOP.
- **VER** – returns the version of the firmware installed in the device.
- **WRITEI2C <BYTE> -** writes a byte to the I2C bus and waits for an acknowledge. If the target device does not respond then the device will reset. The device will return either 'ACK' or 'NACK' depending on what was received from the target device.
  Example: *WRITEI2C D7* – writes 0xd7 to the I2C target device.
- **WRITEI2CPAGE <BYTECOUNT> <BYTES>** - writes multiple bytes to the target I2C device (*only for firmware versions 2.10 and higher)*.
  Example: WRITEI2CPAGE 04 a0005678 – writes four bytes to the I2C device.
- **WRITESPI <BYTE>** - writes a byte to the target SPI device and returns what was read from the device.[2]
- **WRITESPIPAGE <BYTECOUNT> <BYTES>** - writes multiple bytes to the target SPI device and returns what was read from the device (*only for firmware versions 1.10 and higher)*.
  Example: *WRITESPIPAGE 05 030000FFFF* – writes five bytes to the target SPI device and returns the five bytes that were read back.

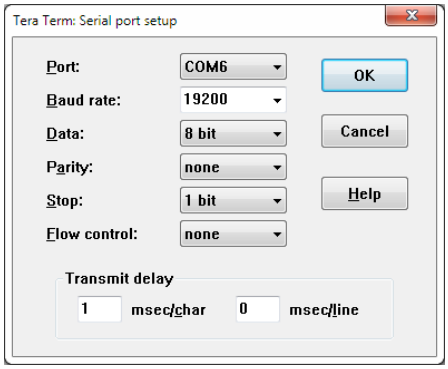
Standard Communication

It is recommended that Tera Term be used to communicate via PC with the UART to SPI/I2C converter. Tera Term may be downloaded for free from the following location:

http://logmett.com/index.php?/download/tera-term-474-freeware.html

It is important that the serial port is correctly configured to communicate with the UART to SPI/I2C bridge device. The serial mode is 8-N-1 and the default baud rate is 19200bps. In Tera Term, click Setup -> Serial Port and check that the settings are as below (your 'Port' may be a different number):

[2] There is no 'READSPI' command, as this isn't relevant in the SPI protocol. To read a byte from the SPI device, simply send dummy data, for example *WRITESPI FF*. The device will respond with what was read back from the SPI device.



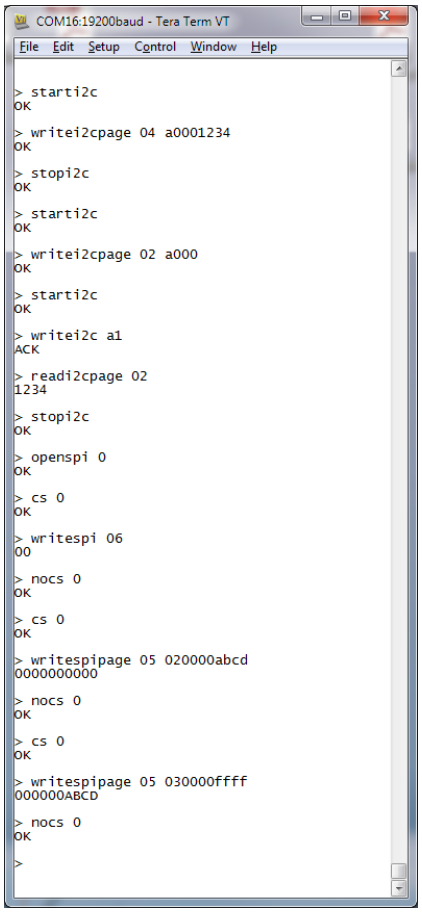
If you wish to make your own batch files then it is important that the transmit delay is set to 1 msec/char, as above.

Batch Files

In order to save typing in every command for repetitive tasks, it may be a good idea to make a batch file. This is a simple task. Simply open notepad (or any other text editor) on your PC and type in the string of commands you wish to send, with each command on a new line. Then when you are connected to the UART to SPI/I2C converter, simply 'drag' the text file onto the Tera Term window and Tera Term will send the commands, one after the other, to the device. This could be useful for running predetermined routines, for example writing data to or erasing an EEPROM.

Standard Command Set Example Use

See below for an example of the device in use, communicating with an I2C and an SPI device.



Binary Command List

In order to use the binary command set, follow this procedure:

1. Apply power to the device (plug the device into a USB port). At this point, the device will be constantly sending data, waiting for the user to press any key to use the standard command set.
2. Send byte **0xAC**. This will put the device in 'binary' mode, allowing the use of the binary command list below. The device will stop sending data and start to listen for binary commands.

Each command is given in hex format and should be sent using a binary communication method (i.e. not via an ASCII terminal program). The device replies to commands with 0x01 to indicate that the command was received and processed.

- **0x50** – sends an I2C START. Device replies with 0x01.
- **0x51** – sends an I2C STOP. Device replies with 0x01.
- **0x52** – sends an I2C RESTART. Device replies with 0x01.
- **0x53 [aa] [bb ... bb]** – write I2C data. Device replies with 0x01.
    - aa = number of bytes to write
    - bb = bytes to send
      e.g. *53 02 a0 11* or *53 04 a0 33 66 44*
- **0x54 [aa]** – read I2C data. Device replies with 0x01 followed by data read.
    - aa = number of bytes to read
      e.g. *54 02* (read two bytes)
- **0x55** – scan the I2C bus for connected devices. Returns 0x01, followed by a list of address discovered, terminated with 0xFF.
    - Example data returned: *01 a0 a2 a4 ff* (devices discovered on address a0, a2 and a4).
    - Note that a single I2C IC may respond to several addresses.
- **0x60 [aa]** – read ADC. Device replies with 0x01 followed by a 10-bit ADC value represented in two bytes (0x0000 to 0x03FF). The maximum value, 0x3FF equates to 3.3V.
    - aa = ADC channel to read
      e.g. *60 00* (read ADC channel 0)
- **0x61 [aa]** – read digital line. Device replies with 0x01 followed by 0x01 or 0x00 depending on whether the selected line is high or low.
    - aa = digital channel to read
      e.g. *61 02* (read digital channel 2)
- **0x70 [aa]** – open SPI ready for communication. Device replies with 0x01.
    - aa = SPI mode to initialise in
      e.g. *70 01* (open in SPI mode 1)
- **0x71 [aa] [bb ... bb]** – write SPI data. Device replies with 0x01, followed by data read back.
    - aa = number of bytes to write
    - bb = bytes to write
      e.g. *71 05 02 00 00 ab cd* (write five bytes)

- **0x80 [aa]** – drive CS output LOW. Device replies with 0x01.
    - aa = the CS output to drive LOW.
      e.g. *80 00*
- **0x81 [aa]** – drive CS output HIGH. Device replies with 0x01.
    - aa = the CS output to drive HIGH.
      e.g. *81 00*
- **0x90 –** return the current version. Device replies with 0x01 followed by the version multiplied by 100.
    - Example data returned: *01 01 2C* (for version 3.00)

It is important to note that if an invalid command is sent (for example specifying that 5 bytes are written to SPI, but providing only 4), the device will time out and reset, requiring that **0xAC** is written to the device again in order to re-initiate the binary mode interface.

Raspberry Pi Compatibility

This device is compatible with the Raspberry Pi, as well as almost any other Linux distribution.

When connected to one of the Pi's USB ports, it is mapped under /dev, which allows simple low level access for communication. If you don't have any other USB Serial converters connected to the Pi, the device will most likely be mapped to /dev/ttyUSB0.

In order to communicate with the device, it is first necessary to set a few parameters, using the 'stty' command. The full stty command to type into the console once the device is plugged in is:

```
stty -F /dev/ttyUSB0 19200 ignbrk -brkint -icrnl -imaxbel -opost -onlcr -isig -icanon -iexten -echo -echoe -echok -echoctl -echoke
```

Once this command has been issued, it is possible to read and write to the device at /dev/ttyUSB0 using standard Linux commands.

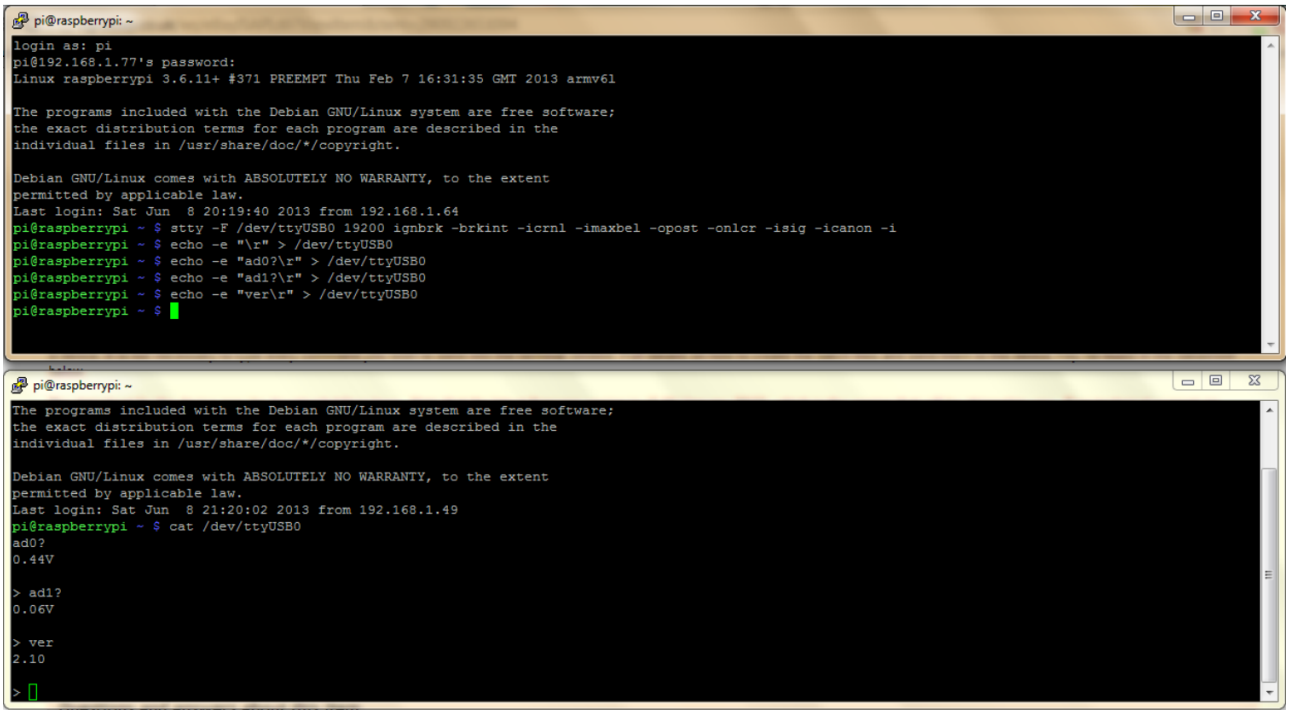
Raspberry Pi Example

Follow the example below in order to prove communications with the device on the Pi:

1. Open two terminal windows.
2. In the first, type the full 'stty' command given above.
3. Now, in the second terminal window, type `cat /dev/ttyUSB0`. This window will now constantly be refreshed with the data being sent from the device. If the device has just been plugged in then the window now probably says "Press any key to start…".
3. Back in the first terminal window, type `echo -e "\r" > /dev/ttyUSB0`. This will send the carriage return character (\r) to the device, to initiate communication.
4. Still in the first terminal window, type `echo -e "ad0?\r" > /dev/ttyUSB0.` This sends the 'ad0?' command, followed by a carriage return (\r). You should see the response to this command in the second terminal window.

The above is just an example to prove that the device is working correctly on your Pi. The world is your oyster with regards what you do with it now that you have such free access.

An example of the device being used on the Pi (with two terminal windows as described above) is given below:


pi@raspberrypi: ~
login as: pi
pi@192.168.1.77's password:
Linux raspberrypi 3.6.11+ #371 PREEMPT Thu Feb 7 16:31:35 GMT 2013 armv6l
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Jun 8 20:19:40 2013 from 192.168.1.64
pi@raspberrypi ~ $ stty -F /dev/ttyUSB0 19200 ignbrk -brkint -icrnl -imaxbel -opost -onlcr -isig -icanon -i
pi@raspberrypi ~ $ echo -e "\r" > /dev/ttyUSB0
pi@raspberrypi ~ $ echo -e "ad0?\r" > /dev/ttyUSB0
pi@raspberrypi ~ $ echo -e "ad1?\r" > /dev/ttyUSB0
pi@raspberrypi ~ $ echo -e "ver\r" > /dev/ttyUSB0
pi@raspberrypi ~ $
pi@raspberrypi: ~
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Jun 8 21:20:02 2013 from 192.168.1.49
pi@raspberrypi ~ $ cat /dev/ttyUSB0
ad0?
0.44V
> ad1?
0.06V
> ver
2.10
>