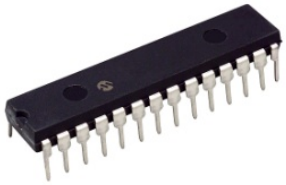
UART (RS232) to SPI / I2C / ADC / GPIO Converter Bridge

Summary

This device is an SPI/I2C/ADC/GPIO master and is controlled via the UART. It allows all of the following:



- 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 value 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.
- Binary interface for high speed embedded communications.

Devices are shipped worldwide. Contact **devices@burningimage.net** for a quote.

Pinout

The IC is a standard 28-pin 300mil (skinny) plastic, dual in-line package. A package drawing is available on request.

N/C	1	28	CS7
AD0	2	27	CS6
AD1	3	26	CS5
AD2	4	25	CS4
AD3	5	24	CS3
D0	6	23	CS2
D1	7	22	CS1
GND	8	21	CS0
D2	9	20	VDD
D3	10	19	GND
D4	11	18	RX
MISO	12	17	TX
CLK	13	16	MOSI
SCL	14	15	SDA

Figure 1: IC Pinout

Connections

- Pins **AD0** to **AD3** are analogue inputs. Any voltage up to VDD may be connected to these pins. The IC will return the value 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.
- **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 max current).
- **RX** and **TX** should be connected to the Tx and Rx pins respectively of the UART receiver. The default baud rate of the UART port is 19200bps.
- **VDD** and **both GND** pins should be connected to a voltage supply as appropriate. VDD must be between 2.3V and 5.5V.
- **N/C** may be left floating.

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 to the IC, intended for use within embedded applications. It is expected that most users of this device will use the binary command set, but both are equally supported.

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.

Binary Command List

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

1. Apply power to the device.
2. Send byte **0x62 ('b')**. This will put the device in 'binary' mode, allowing the use of the binary command list below. The device will 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.

The device works in real-time – if the maximum (115200) baud rate is configured, 14,400 bytes may be sent via the I2C or SPI bus per second.

- **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 immediately with data read (0x01 is not returned).
  - 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.
- **0x56 [aa]** – I2C device ping. Device replies 0x01 if I2C device with address aa is present, 0x00 if it is not.
  - aa = I2C device address to ping
- **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 VDD.
  - 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)

If an invalid command is sent (for example specifying that 5 bytes are written to SPI, but providing only 4) then the device will send **0xEE** to indicate that an error has occurred.

Example Use

In order to write and then subsequently read back out a byte to an I2C 24xx01 EEPROM, send the following commands:

Write data

1.	0x50	I2C START
2.	0x53 0x03 0xA0 0x00 0x12	I2C WRITE address: 0xA0: put 0x12 in EEPROM loc. 0x00
3.	0x51	I2C STOP

Read data

1.	0x50	I2C START
2.	0x53 0x02 0xA0 0x00	I2C WRITE address: 0xA0. Set EEPROM loc. pointer to 0x00
3.	0x50	I2C START
4.	0x53 0x01 0xA1	I2C WRITE set device address: 0xA1
5.	0x54 0x01	I2C READ 1 byte (0x12 is returned)
6.	0x51	I2C STOP

Standard Command List

In order to use the standard command set, open up a terminal program, power on the device and send three '+' characters when prompted. From that point on, the device is programed to use the standard command set. To switch to the binary command set, the device must be power-cycled.

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 referenced to 3V3 *(and ADC count for firmware > 3.10)* 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.

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

Example: *OPENSPI 0*

- **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.

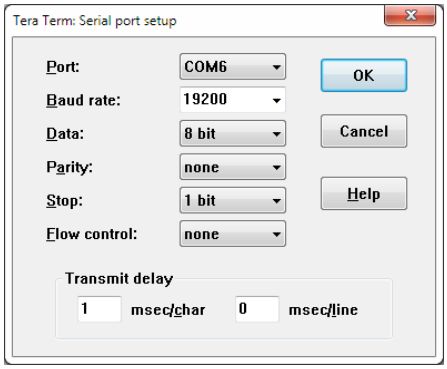
Standard Communication

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-472-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.

Note that the voltage levels used for the UART communication are TTL logic levels. These logic levels are the standard for interfacing with microcontrollers and (for example) USB/UART converter devices. If you wish to connect the IC directly to a PC's COM port then a level translator (such as a MAX232) will ideally need to be employed. If a MAX232-type device isn't available then most modern COM ports will work with a basic logic inverter in its place, but it isn't recommended for long-term use.

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 Figure 2 for an example of the IC in use, communicating with an I2C and an SPI device.

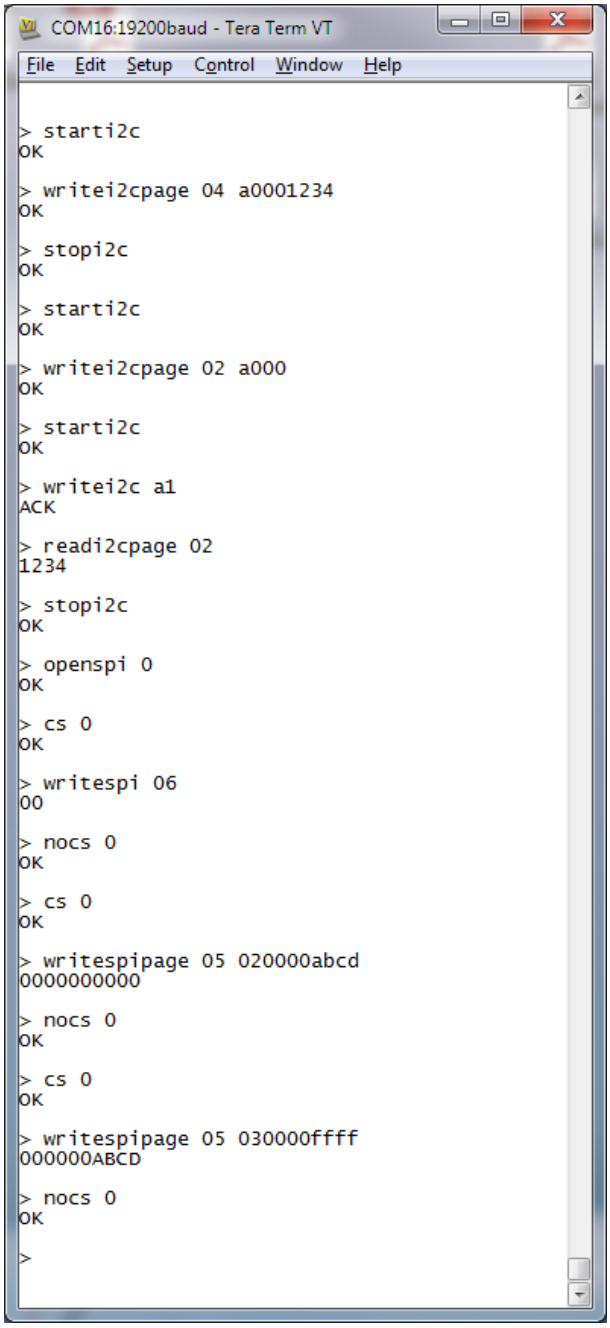


Figure 2: IC in use