ICN12

I2C to UART Bridge, ADC,DAC and I/O

Firmware version 1.6

Updated 12 Sept. 2018 Corrected chip markings

Updated 24 Sept. 2018 Address address to commands to allow same device on same bus.

Introduction

This is an I2C to UART bridge, designed to give an extra UART to a microcontroller when only I2C is available. It is an I2C device and thus is configured and controlled by that interface.

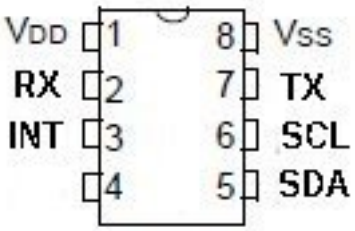
Features

- Configurable I2C of any address
- Defaults to 115200 Baud, configurable
- INT pin indicates when there is characters in the UART buffer
- CTS output for some hardware handshaking
- 80 byte UART input buffer (type A 32 byte)
- Operates on 3.3V or 5V
- 1 channel DAC 64 steps
- 4 channel 10 bit ADC with precision voltage reference, up to 4.095 volts
- 5V I/O pins and ADC

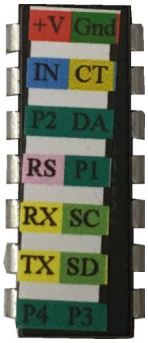
Pinout

There are several options for the device, the 8pin DIL provides a basic UART bridge whilst all of the other devices are based on a 14 pin device that has extra I/O that can be configured for ADC, or GPIO.

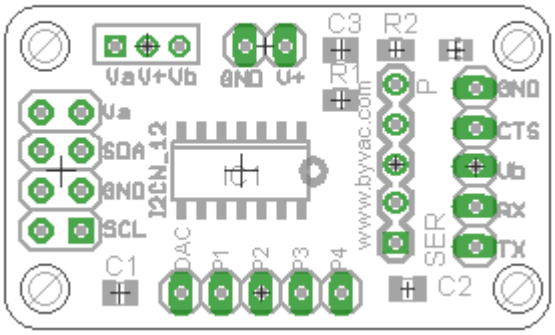
The PCB board versions have all of the extra I/O one is a general purpose boar and the other is designed as a WeMos (D! Type) shield.

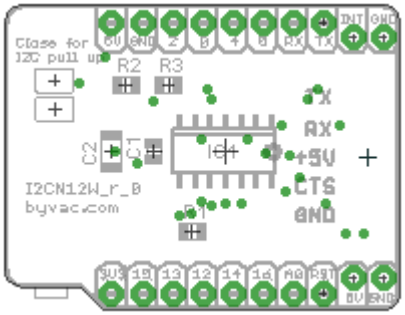


Type A



Type B





Types A and W

- VDD Power voltage, this can be either 3.3V or 5V see note
- RX Serial input
- TX Serial output
- CTS This is an output that will go low[1] when the UART buffer becomes full and can be used to tell the host device to stop transmitting
- SDA I2C data
- SCL I2C clock
- INT This pin will go high when there is one or more bytes in the UART buffer otherwise it will be low.
- DAC Digital to analogue output
- Pn General purpose I/O that can be used for digital input, digital output or analogue to digital input
- ADC 10 bit analogue to digital on 4 channels

Note:

[1] By default will be high when the buffer is not full, can be changes in EEPROM

Any pin must not exceed the voltage on VDD so if a 3.3V power supply is used then the signal pins must also be 0 – 3.3V

DAC

This is an output with a voltage swing controlled by a command from about 0.3V to VDD. The resolution is 32, 0 being 0.3V and 31 being VDD.

Using values greater then 31 will give unpredictable results.

When switching off it will power down the internal peripheral but may not alter the voltage on the output pin.

GPIO

There are 4 GPIO lines that can set set to output high or low, or input. Output is set in one go by issuing the output command followed by either high or low.

Input is carried out by using the read input command, the port will be automatically configured for input when using this command.

ADC

The ADC has a resolution of 10 bits (0 to 1023) and is referenced from a precision voltage reference. The reference can be set to VDD, 1.024V, 2.048V, 4.096V To use the last voltage VDD must be 5V.

The ADC takes about 100us which matches the interface and so is not suitable for very high speed applications.

Any GPIO can be used for ADC, when a read command is issued the port automatically configures for analogue input.

Status Flag

							BF[1]

[1] bit cleared when read

BF Indicates if the buffer has become full at some point and it is likely that input serial bytes will have been lost.

EEPROM Locations

Adr	Default	Notes
0	0x55	System use
1	0x48	I2C address
2	0	Baud rate bits 31:24
3	1	Baud rate bits 23:16
4	0xc2	Baud rate bits 15:8
6	0	Baud rate bits 7:0
7	1	Command trigger
8	1	CTS buffer not full value

14	0x48	I2C address copy
240	0x48	I2C address copy

I2C Commands and Operation

Default device address is 0x48 (72) 8 bit address this translates to the following:

Eight bit addressing where the read write flag is part of the address:

Write address 72

Read address 73

Seven bit addressing where there are read and write i2c commands (Arduino, MicroPython)

I2c Address 36

By default anything written to the I2C address will be echoed on the TX pin. So for example i2c.write('hello') will appear on the TX pin. I2c.write will use the write address of the device (0x48).

Reading from the i2c will retrieve the contents of the UART buffer (if there is anything in there), so for example, using the read address 0x49 (0x24 7 bit), i2c.read(n) where n is the number of bytes to read will get the contents of the UART buffer. If there are no bytes in the buffer 0 is returned.

In short then, writing to i2c is the same as writing to the UART and reading from i2c is the same as reading the UART buffer.

Commands

In order to get information such as how many bytes there are in the UART buffer, or to clear the buffer a general call is used, followed by a trigger byte followed by the device address (7 bit).

The general call is address 0 and a command sequence is 0+trigger+adr followed by the command and any other data. For commands that return information a normal read is used after the command sequence.

Both devices A and B

Applies to B device only

The default trigger is 0x1a

Example of clearing the buffer would be:

<address 0><trigger><7 bit device address><1>

Example of writing a 1 to an output port would be:

<address 0><trigger><7 bit device address><1>

Example of reading the number of bytes in a UART buffer requires writing the command and then reading the number of bytes

<address 0><trigger><7 bit device address><5>

read 1 byte

Command	Range	Notes
<n/a>	N/a	**EEPROM reset** This will restore the default EEPROM settings which has the I2C address. No effect will take place until the device is reset. This can take a few ms and so some master I2C devices will need to take this into account if a timeout is to be avoided. This is a special command in that it can be called regardless of the state of the EEPROM, in other words it does not rely on the I2C address or trigger value Example Warning this will reset all I2CN12 devices on the bus i2c.write(0,0x55)
1	N/a	**Clear the UART Buffer** Example i2c.write(0,trigger,7bit addr,1)
5	N/a	**Gets number of bytes in UART buffer** Example i2c.write(0,trigger,7bit addr,5) i2c.read(1)
6	N/a	**Get Status** Gets status byte see text. Example i2c.write(0,trigger,7bit addr,6) i2c.read(1)
10	B 1 to 8	**Set Baud Rate Temporary** Sets the Baud rate to a new value, this will be forgotten at reset. Tos set the Baud rate so it is remembered after reset, set the EEPROM values. Rate value Meaning 1 1200 2 2400 3 4800 4 9600 5 19200 6 38400 7 57600 8 115200 Example, to set the baud rate to 9600 i2c.write(0,trigger,7bit addr,10,4)
40	V = 0 to 31	**Sets DAC** Sets the voltage on the DAC pin, range is from about 0.3V to VDD with a resolution from 0 to 31. Example

		i2c.write(0,trigger,40,v)

41	Ch 1 to 4	**Get ADC Value**
		Returns ADC value for a particular channel. Two bytes are returned forming a 16 bit value, high byte first.
		Example
		i2c.write(0,trigger,7bit addr,13,n) // n is 1 to 4
		i2c.read(2)

42	0 to 4	**Set Vref for ADC** By default the voltage reference is set to VDD. Applies to all channels, this can be changed using this command where n is: 0 = VDD 1 = 1.024V 2 = 2.048V 3 = 4.096V (Vdd must be 5V to use this) Example i2c.write(0,trigger,7bit addr,42,n)
50	P= 1 to 4 value 0 or 1	**Set Port Pn to output and set value** Sends 0 or 1 to Pn that will be automatically configured as an output when using this command. Example, write 1 to port P2 i2c.write(0,trigger,7bit addr,50,2,1)

51	P = 1 to 4	**Get Value from Pn**
		Get the value on Pn, (0 or 1) Pn will be automatically configured as an input when using this command.
		Example
		i2c.write(0,trigger,7bit addr,51,n) // where n is 1 to 4
		i2c.read(1)

60	Adr 6-255	**Change I2C address** The address must be an even value, this is the 8 bit or full address where an even address is write and an odd address is read. It will not take effect until reset. The same effect can be achieved by writing to the EEPROM Example i2c.write(0,trigger,7bit addr,60,adr)
61	Adr 0-255	**Read EEPROM** This will read a single value from an address in EEPROM Example i2c.write(0,trigger,7bit addr,61,adr) i2c.read(1) Send the trigger, command and then the address to be read.
62	Adr 0-255 data 0-	**Write EEPROM** Writes a single byte to the given address Example

	255	i2c.write(0,trigger,7bit addr,62,adr,data)
63		**Gets device ID as 2 bytes** The first byte is the high byte of a 16 bit number and the second byte is the low byte Example i2c.write(0,trigger,7bit addr,63) i2c.read(2)
64		**Gets firmware version as 2 bytes** Example i2c.write(0,trigger,7bit addr,64) i2c.read(2)
65	N/a	**Sleep** Places device in sleep mode Example i2c.write(0,trigger,7bit addr,65)
66	N/a	**Reset** Resets the device as at first switch on. Depending on the I2C master this will likely cause a time out as there will be no reply from the device and tso this may cause an I2C error from the master Example i2c.write(0,trigger,7bit addr,66)

Baud rate

This is held in EEPROM as 4 bytes as it is a 32 bit number the following table has some pre-calculated values

Rate	Adr 2	Adr 3	Adr 4	Adr 5
1200	0	0	4	0xb0
2400	0	0	9	0x60
4800	0	0	0x12	0xc0
9600	0	0	0x25	0x80
14400	0	0	0x38	0x40
28800	0	0	0x70	0x80
57600	0	0	0xe1	0
115200	0	1	0xc2	0

Read Write Examples, using I2C primitives

Write

```
i2c_start()
i2c_putc(0x48) // write address
```

```
i2c_putc('H') // etc. ad many bytes are as required
i2c_stop()
```

Read

```
i2c_start()
i2c_putc(0x49) // read address
i2c_getc(0) // read 1 byte ACK
i2c_getc(0) // read 1 byte ACK
i2c_getc(1) // read 1 byte NACK last byte
i2c_stop()
```

Because of the way I2C works, the buffer is loaded by the device BEFORE is is clocked out by the master and so the device needs to know the last byte being read otherwise it will place a byte in the buffer that will never be read, effectively loosing a byte from the buffer.

Command

Get the number of bytes from the buffer using command 5

```
i2c_start()
i2c_putc(0) // general call address
i2c_putc(trigger) // send trigger
i2c_putc(0x24) // send 7 bit device address
i2c_putc(5) // command
i2c_stop()
i2c_start()
i2c_putc(0x49) // read address
i2c_getc(1) // number of bytes
i2c_stop()
```