Future Technology Devices International Ltd.

Vinco Volt Meter Example Application Note AN_162

Document Reference No.: FT_000364

Version 2.0

Issue Date: 2010-04-15

This application note describes how the Vinco module can be used to generate waveforms with the VNC2 PWM interface and drive a serial oLED display showing the RMS voltage value from the waveform after conversion through the Vinco ADC.

Future Technology Devices International Ltd (FTDI)

Unit 1, 2 Seaward Place, Centurion Business Park, Glasgow, G41 1HH, United Kingdom

Tel.: +44 (0) 141 429 2777 Fax: + 44 (0) 141 429 2758

E-Mail (Support): support1@ftdichip.com

1 Introduction

Vinco is a development module inspired by the Arduino concept and uses the Vinculum II, VNC2 device. Vinco uses a VNC2-64Q package to facilitate 38 GPIO options on 0.1” pitch sockets. Vinco is designed as a prototyping platform for VNC2 based designs and applications.

This application note describes an example of how to use the Vinco module to drive the VNC2 PWM interface and convert the generated waveform into an RMS voltage value for display on a oLED display. The application note also provides “C” source code examples to help the user get started with their own specific application. This source code can be downloaded from the FTDI website at:
http://www.ftdichip.com/Support/SoftwareExamples/VinculumIIProjects/Vinco_Volt_Meter_Demo.zip

Note: Any sample code provided in this note is for illustration purposes and is not guaranteed or supported.

Figure 1.1 - VINCO

1.1 VNC2 Devices

VNC2 is the second of FTDI’s Vinculum family of embedded dual USB host controller devices. The VNC2 device provides USB Host interfacing capability for a variety of different USB device classes including support for BOMS (bulk only mass storage), Printer and HID (human interface devices). For mass storage devices such as USB Flash drives, VNC2 transparently handles the FAT file structure.

Communication with non USB devices, such as a low cost microcontroller, is accomplished via either UART, SPI or parallel FIFO interfaces. VNC2 provides a new, cost effective solution for providing USB Host capability into products that previously did not have the hardware resources available.

VNC2 allows customers to develop their own firmware using the Vinculum II software development tool suite. These development tools provide compiler, assembler, linker and debugger tools complete within an integrated development environment (IDE).

The Vinculum-II VNC2 family of devices are available in Pb-free (RoHS compliant) 32-lead LQFP, 32-lead QFN, 48-lead LQFP, 48-lead QFN, 64-Lead LQFP and 64-lead QFN packages For more information on the ICs refer to http://www.ftdichip.com/Products/ICs/VNC2.htm

1.2 4D Systems Serial oLED Display

This application example uses the 4D Systems serial oLED display uoLED-160-G1SGC to display the voltage levels created by the VNC2 PWM interface on the Vinco prototype via the ADC converter. The display is driven by a 5V power supply and serial RX and TX data lines to control the characters displayed on the display. For more information on 4D Systems displays see:
http://www.4dsystems.com.au/prod.php?id=79

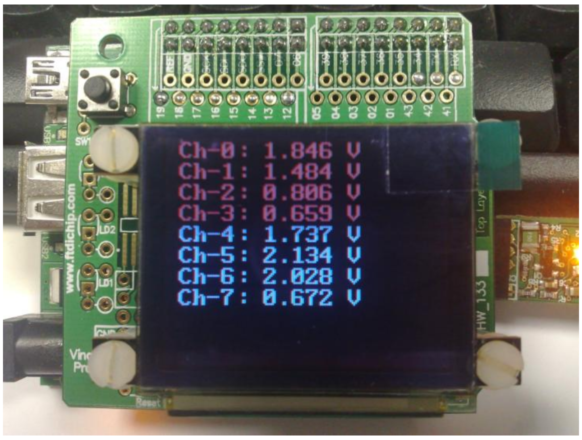


Figure 1.2 – uoLED-160-G1SGC Display on Vinco Proto shield

Table of Contents

2 PWM Interface

Pulse width modulation (PWM) is a common interface on microcontrollers.

VNC2 has 8 separate independent PWM channels. The following section describes the building blocks used to control these PWM channels.

Pre-scaler - This is a programmable counter that reduces the frequency of the system clock (48 MHz, 24 MHz, or 12 MHz) to the desired frequency. The pre-scaler is shared by all 8 PWM channels.

16 Bit Counter Block - This is a programmable counter that determines the period of the PWM signal. The input clock is from the pre-scaler block. The 16 bit counter is shared by all 8 PWM channels.

16 Bit Comparator – Up to 8 comparators can be used per PWM channel. The number of comparators assigned to a PWM channel determines the toggle events (up to 8), which give up to 4 data pulses. Simple duty cycle based pulse width modulation can be programmed by using only 2 comparators. There are a total of 8 comparators in the PWM module.

Control Block – This controls the number of times to repeat the PWM waveform. The control block is shared by all 8 PWM channels.

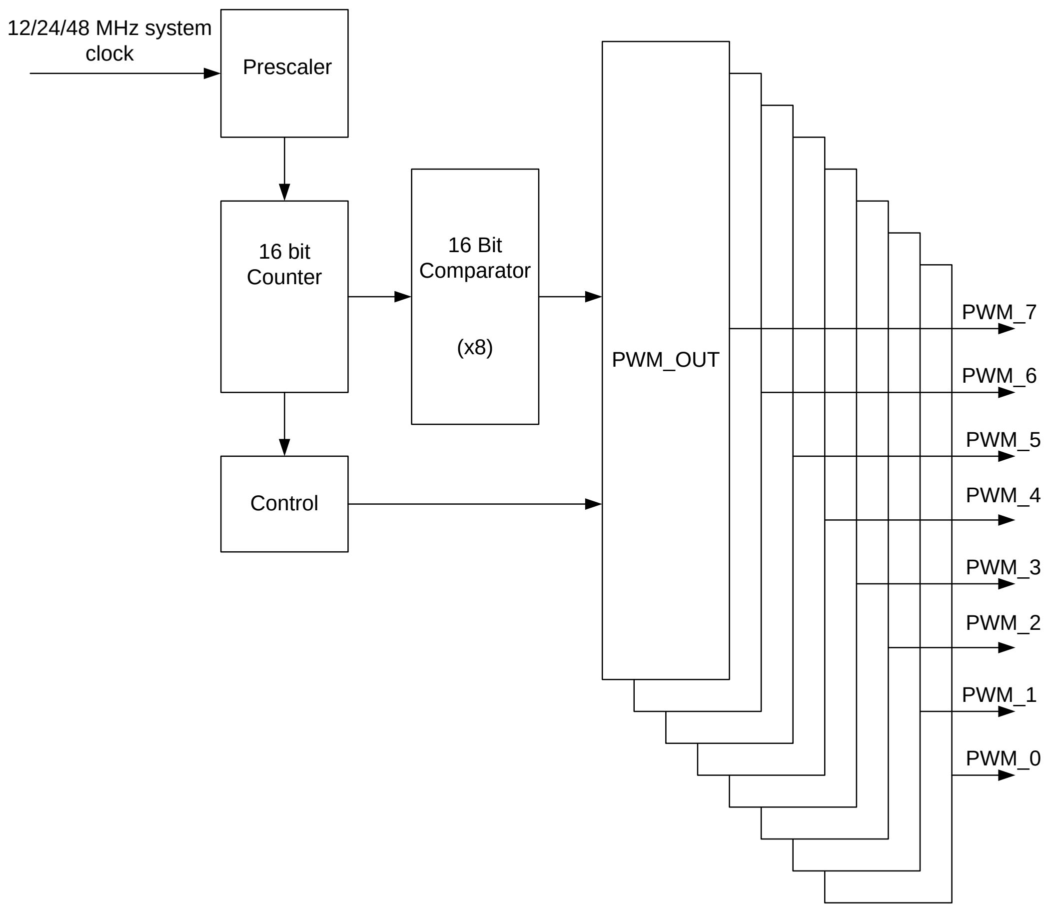


Figure 2.1 – VNC2 PWM Block Diagram

3 Schematic Diagram

This schematic diagram, Figure 3.1, shows the interconnect required on the Vinco Proto PCB for the Vinco to provide waveforms from the PWM interface of the VNC2 into the Vinco ADC then convert this to a serial message to be displayed on the oLED display.

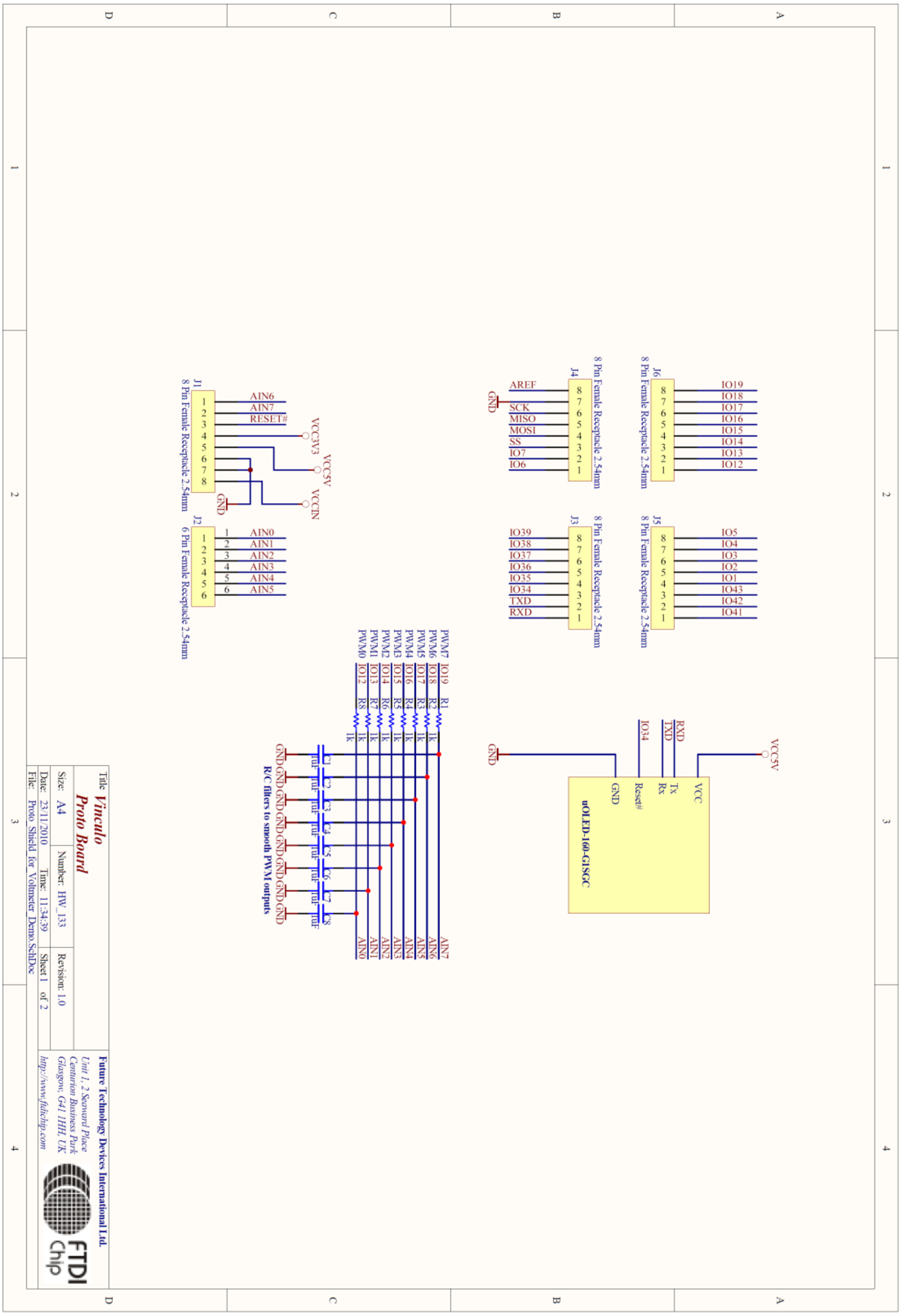


Figure 3.1 – Vinco Volt Meter Demo Block Diagram

4 Interconnect

4.1 Power

The Vinco module may be powered from the USB port on CN3 (5V) or via an external power converter (9V/1A DC) to CN1 (for example the FTDI VNCLO-PSU-UK)

As this application provides power to external circuitry (the oLED display), the Vinco is powered from an external 9V supply.

To ensure this power source is routed to the PCB, JP1 on the Vinco module must be set to the 2-3 position.

Power from the Vinco module is taken from J1 pin 5 to give a +5V supply for the oLED display.

4.2 oLED Control

The 4D Systems oLED display is controlled via a serial interface. One RX data line and one TX data line. as described in Table 4.1.

Signal	Function
RXD	Serial data to the oLED display
TXD	Serial data from the oLED display
RESET	RESET for the display

Table 4.1 - Signal Name and Description – oLED Interface

4.3 Debugger Interface

The purpose of the debugger interface is to provide access to the VNC2 silicon/firmware debugger. The debug interface can be accessed by connecting a *VNC2_Debug/programming_Module* (http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_Vinculum-II.pdf) to the J8 connector. This debug/programming module gives access to the debugger through a USB connection to a PC via the Integrated Development Environment (IDE). The IDE is a graphical interface to the VNC2 software development tool-chain and gives the following debug capabilities through the debugger interface:

- Flash Erase, Write and Program.
- Application debug - application code can have breakpoints, be single stepped and can be halted.
- Detailed internal debug - memory and register read/write access.

The IDE may be downloaded, free of charge, from
http://www.ftdichip.com/Firmware/V2TC/VNC2toolchain.htm

The Debugger Interface, and how to use it, is further described in the following applications Note
Vinculum-II Debug Interface Description

4.3.1 Signal Description - Debugger Interface

Table 4.2 shows the signals and pins description for the Debugger Interface pin header J8

Pin No.	*Name*	*Name On PCB*	*Type*	*Description*
J8-1	IO0	DBG	I/O	Debugger Interface
J8-2	-	[Key]	-	Not connected. Used to make sure that the debug module is connected correctly.
J8-3	GND	GND	PWR	Module ground supply pin
J8-4	RESET#	RST#	Input	Can be used by an external device to reset the VNCL2. This pin is also used in combination with PROG# and the UART interface to program firmware into the VNC2.
J8-5	PROG#	PRG#	Input	This pin is used in combination with the RESET# pin and the UART interface to program firmware into the VNC2.
J8-6	5V0	VCC	PWR Input	5.0V module supply pin. This pin can be used to provide the 5.0V input to the V2DIP2-32 from the debugger interface when the V2DIP2-32 is not powered from the USB connector (VBUs) or the DIL connector pins J1-1 and J3-6.

Table 4.2 - Signal Name and Description – Debugger Interface

5 Source code for the VNC2 writing to oLED Display

The Vinculum II IDE is used to create application code to run on VNC2. This section gives some example source code, and explains its operation, used to drive the OLED display via the Vinco module.

Note the full project can be downloaded at:
http://www.ftdichip.com/Support/SoftwareExamples/VinculumIIProjects/Vinco_Volt_Meter_Demo.zip

5.1 VNC2 Initialisation

When generating firmware for VNC2, the first steps are to enable the Vinculum Operating System (VOS), which controls the VNC2 services and device manager, defines the clock speed the core will use, and defines the VNC2 pins that will be used. This is done in the function labelled *main*. The "*main*" function for this application is shown as follows

```
void main(void)
{
   // GPIO context structure
   gpio_context_t gpio_Ctx;
   // UART context structure
   uart_context_t uart_Ctx;
   // SPI Master context structure
   spimaster_context_t spim_Ctx;

   // call VOS initialisation routines
   vos_init(10, VOS_TICK_INTERVAL, NUMBER_OF_DEVICES);
   vos_set_clock_frequency(VOS_48MHZ_CLOCK_FREQUENCY);

   // Route IO signals
      // GPIO port A bit 0 to pin 39
      vos_iomux_define_output(39,IOMUX_OUT_GPIO_PORT_A_0); // LED1#
      vos_iocell_set_config(39, VOS_IOCELL_DRIVE_CURRENT_4MA,
                            VOS_IOCELL_TRIGGER_NORMAL,
                            VOS_IOCELL_SLEW_RATE_FAST,
                            VOS_IOCELL_PULL_NONE);
      // GPIO port A bit 1 to pin 40
      vos_iomux_define_output(40,IOMUX_OUT_GPIO_PORT_A_1); // LED2#
      vos_iocell_set_config(40, VOS_IOCELL_DRIVE_CURRENT_4MA,
                            VOS_IOCELL_TRIGGER_NORMAL,
                            VOS_IOCELL_SLEW_RATE_FAST,
                            VOS_IOCELL_PULL_NONE);
      // GPIO port A bit 2 to pin 41
      vos_iomux_define_output(41,IOMUX_OUT_GPIO_PORT_A_2); // PWREN#
      vos_iocell_set_config(41, VOS_IOCELL_DRIVE_CURRENT_4MA,
                            VOS_IOCELL_TRIGGER_NORMAL,
                            VOS_IOCELL_SLEW_RATE_FAST,
```

```
                    VOS_IOCELL_PULL_NONE);

// GPIO port A bit 6 to pin 55
vos_iomux_define_output(55,IOMUX_OUT_GPIO_PORT_A_6); // Display RESET#
vos_iocell_set_config(55, VOS_IOCELL_DRIVE_CURRENT_4MA,
                    VOS_IOCELL_TRIGGER_NORMAL,
                    VOS_IOCELL_SLEW_RATE_FAST,
                    VOS_IOCELL_PULL_NONE);

//---------------------------------------
// Configure GPIO Port B as high-impedance inputs
// GPIO port B bit 0 to pin 43
vos_iomux_define_input(43,IOMUX_IN_GPIO_PORT_B_0); // AIN0
vos_iocell_set_config(43, VOS_IOCELL_DRIVE_CURRENT_4MA,
                    VOS_IOCELL_TRIGGER_NORMAL,
                    VOS_IOCELL_SLEW_RATE_FAST,
                    VOS_IOCELL_PULL_UP_75K);
// GPIO port B bit 1 to pin 44
vos_iomux_define_input(44,IOMUX_IN_GPIO_PORT_B_1); // AIN1
vos_iocell_set_config(44, VOS_IOCELL_DRIVE_CURRENT_4MA,
                    VOS_IOCELL_TRIGGER_NORMAL,
                    VOS_IOCELL_SLEW_RATE_FAST,
                    VOS_IOCELL_PULL_UP_75K);
// GPIO port B bit 2 to pin 45
vos_iomux_define_input(45,IOMUX_IN_GPIO_PORT_B_2); // AIN2
vos_iocell_set_config(45, VOS_IOCELL_DRIVE_CURRENT_4MA,
                    VOS_IOCELL_TRIGGER_NORMAL,
                    VOS_IOCELL_SLEW_RATE_FAST,
                    VOS_IOCELL_PULL_UP_75K);
// GPIO port B bit 3 to pin 46
vos_iomux_define_input(46,IOMUX_IN_GPIO_PORT_B_3); // AIN3
vos_iocell_set_config(46, VOS_IOCELL_DRIVE_CURRENT_4MA,
                    VOS_IOCELL_TRIGGER_NORMAL,
                    VOS_IOCELL_SLEW_RATE_FAST,
                    VOS_IOCELL_PULL_UP_75K);
// GPIO port B bit 4 to pin 47
vos_iomux_define_input(47,IOMUX_IN_GPIO_PORT_B_4); // AIN4
vos_iocell_set_config(47, VOS_IOCELL_DRIVE_CURRENT_4MA,
                    VOS_IOCELL_TRIGGER_NORMAL,
                    VOS_IOCELL_SLEW_RATE_FAST,
                    VOS_IOCELL_PULL_UP_75K);
// GPIO port B bit 5 to pin 48
```

```
    vos_iomux_define_input(48,IOMUX_IN_GPIO_PORT_B_5); // AIN5
    vos_iocell_set_config(48, VOS_IOCELL_DRIVE_CURRENT_4MA,
                          VOS_IOCELL_TRIGGER_NORMAL,
                          VOS_IOCELL_SLEW_RATE_FAST,
                          VOS_IOCELL_PULL_UP_75K);
    // GPIO port B bit 6 to pin 49
    vos_iomux_define_input(49,IOMUX_IN_GPIO_PORT_B_6); // AIN6
    vos_iocell_set_config(49, VOS_IOCELL_DRIVE_CURRENT_4MA,
                          VOS_IOCELL_TRIGGER_NORMAL,
                          VOS_IOCELL_SLEW_RATE_FAST,
                          VOS_IOCELL_PULL_UP_75K);
    // GPIO port B bit 7 to pin 50
    vos_iomux_define_input(50,IOMUX_IN_GPIO_PORT_B_7); // AIN7
    vos_iocell_set_config(50, VOS_IOCELL_DRIVE_CURRENT_4MA,
                          VOS_IOCELL_TRIGGER_NORMAL,
                          VOS_IOCELL_SLEW_RATE_FAST,
                          VOS_IOCELL_PULL_UP_75K);

    // UART to Vinco board pins
    vos_iomux_define_output(51,IOMUX_OUT_UART_TXD); // UART Tx
    vos_iocell_set_config(51, VOS_IOCELL_DRIVE_CURRENT_4MA,
                          VOS_IOCELL_TRIGGER_NORMAL,
                          VOS_IOCELL_SLEW_RATE_FAST,
                          VOS_IOCELL_PULL_UP_75K);
    vos_iomux_define_input(52,IOMUX_IN_UART_RXD); // UART Rx
    vos_iocell_set_config(52, VOS_IOCELL_DRIVE_CURRENT_4MA,
                          VOS_IOCELL_TRIGGER_NORMAL,
                          VOS_IOCELL_SLEW_RATE_FAST,
                          VOS_IOCELL_PULL_UP_75K);

  // PWM Outputs
  // All PWM outputs are connected to ADC inputs
  // through 1k/1uF R/C filter.
  vos_iomux_define_output(24, IOMUX_OUT_PWM_0); // PWM_0
  vos_iocell_set_config(24, VOS_IOCELL_DRIVE_CURRENT_4MA,
                        VOS_IOCELL_TRIGGER_NORMAL,
                        VOS_IOCELL_SLEW_RATE_FAST,
                        VOS_IOCELL_PULL_UP_75K);
  vos_iomux_define_output(25, IOMUX_OUT_PWM_1); // PWM_1
  vos_iocell_set_config(25, VOS_IOCELL_DRIVE_CURRENT_4MA,
                        VOS_IOCELL_TRIGGER_NORMAL,
                        VOS_IOCELL_SLEW_RATE_FAST,
```

```
                        VOS_IOCELL_PULL_UP_75K);
vos_iomux_define_output(26, IOMUX_OUT_PWM_2); // PWM_2
vos_iocell_set_config(26, VOS_IOCELL_DRIVE_CURRENT_4MA,
                        VOS_IOCELL_TRIGGER_NORMAL,
                        VOS_IOCELL_SLEW_RATE_FAST,
                        VOS_IOCELL_PULL_UP_75K);
vos_iomux_define_output(27, IOMUX_OUT_PWM_3); // PWM_3
vos_iocell_set_config(27, VOS_IOCELL_DRIVE_CURRENT_4MA,
                        VOS_IOCELL_TRIGGER_NORMAL,
                        VOS_IOCELL_SLEW_RATE_FAST,
                        VOS_IOCELL_PULL_UP_75K);
vos_iomux_define_output(28, IOMUX_OUT_PWM_4); // PWM_4
vos_iocell_set_config(28, VOS_IOCELL_DRIVE_CURRENT_4MA,
                        VOS_IOCELL_TRIGGER_NORMAL,
                        VOS_IOCELL_SLEW_RATE_FAST,
                        VOS_IOCELL_PULL_UP_75K);
vos_iomux_define_output(29, IOMUX_OUT_PWM_5); // PWM_5
vos_iocell_set_config(29, VOS_IOCELL_DRIVE_CURRENT_4MA,
                        VOS_IOCELL_TRIGGER_NORMAL,
                        VOS_IOCELL_SLEW_RATE_FAST,
                        VOS_IOCELL_PULL_UP_75K);
vos_iomux_define_output(31, IOMUX_OUT_PWM_6); // PWM_6
vos_iocell_set_config(31, VOS_IOCELL_DRIVE_CURRENT_4MA,
                        VOS_IOCELL_TRIGGER_NORMAL,
                        VOS_IOCELL_SLEW_RATE_FAST,
                        VOS_IOCELL_PULL_UP_75K);
vos_iomux_define_output(32, IOMUX_OUT_PWM_7); // PWM_7
vos_iocell_set_config(32, VOS_IOCELL_DRIVE_CURRENT_4MA,
                        VOS_IOCELL_TRIGGER_NORMAL,
                        VOS_IOCELL_SLEW_RATE_FAST,
                        VOS_IOCELL_PULL_UP_75K);


    // SPI Master to Vinco board pins
    vos_iomux_define_output(19,IOMUX_OUT_SPI_MASTER_CLK); // SPI_M CLK
    vos_iocell_set_config(19, VOS_IOCELL_DRIVE_CURRENT_4MA,
                            VOS_IOCELL_TRIGGER_NORMAL,
                            VOS_IOCELL_SLEW_RATE_FAST,
                            VOS_IOCELL_PULL_UP_75K);
    vos_iomux_define_output(20,IOMUX_OUT_SPI_MASTER_MOSI); // SPI_M MOSI
    vos_iocell_set_config(20, VOS_IOCELL_DRIVE_CURRENT_4MA,
                            VOS_IOCELL_TRIGGER_NORMAL,
```

```
                           VOS_IOCELL_SLEW_RATE_FAST,
                           VOS_IOCELL_PULL_UP_75K);
    vos_iomux_define_input(22,IOMUX_IN_SPI_MASTER_MISO); // SPI_M MISO
    vos_iocell_set_config(22, VOS_IOCELL_DRIVE_CURRENT_4MA,
                           VOS_IOCELL_TRIGGER_NORMAL,
                           VOS_IOCELL_SLEW_RATE_FAST,
                           VOS_IOCELL_PULL_UP_75K);
    vos_iomux_define_output(23,IOMUX_OUT_SPI_MASTER_CS_0); // SPI_M CS0
    vos_iocell_set_config(23, VOS_IOCELL_DRIVE_CURRENT_4MA,
                           VOS_IOCELL_TRIGGER_NORMAL,
                           VOS_IOCELL_SLEW_RATE_FAST,
                           VOS_IOCELL_PULL_UP_75K);
    // CS# signal for onboard ADC
    vos_iomux_define_output(61,IOMUX_OUT_SPI_MASTER_CS_1); // SPI_M CS1
    vos_iocell_set_config(61, VOS_IOCELL_DRIVE_CURRENT_4MA,
                           VOS_IOCELL_TRIGGER_NORMAL,
                           VOS_IOCELL_SLEW_RATE_FAST,
                           VOS_IOCELL_PULL_UP_75K);


// initialise device drivers
uart_Ctx.buffer_size = VOS_BUFFER_SIZE_128_BYTES;
uart_init(VOS_DEV_UART,&uart_Ctx);

spim_Ctx.buffer_size = VOS_BUFFER_SIZE_64_BYTES;
spimaster_init(VOS_DEV_SPIM,&spim_Ctx);

gpio_Ctx.port_identifier = GPIO_PORT_A;
gpio_init(VOS_DEV_GPIO_A,&gpio_Ctx);

gpio_Ctx.port_identifier = GPIO_PORT_B;
gpio_init(VOS_DEV_GPIO_B,&gpio_Ctx);

pwm_init(VOS_DEV_PWM);
// enable PWM interrupts
vos_enable_interrupts(VOS_PWM_TOP_INT_IEN);

adc_mcp3008_init(VOS_DEV_ADC);

// create threads for firmware application (no parameters)
tcbADC_thread = vos_create_thread(29, SIZEOF_FIRMWARE_TASK_MEMORY, &ADC_thread, 0);
tcbPWM_thread = vos_create_thread(29, SIZEOF_FIRMWARE_TASK_MEMORY, &PWM_thread, 0);
```

```
    // start VOS scheduler
    vos_start_scheduler();

main_loop:
    goto main_loop;
}
```

Note: Starting the VOS scheduler is always the last thing to be done as all configuration must be complete before this starts.

5.2 The PWM function

The PWM function in this example is used to define the waveform output on the PWM pins of the VNC2.

```
void PWM_thread(void)
{
    pwm_ioctl_cb_t pwm_iocb;
    unsigned short count0 = 0x0001;
    unsigned short count1 = 0x002A;
    unsigned short count2 = 0x0055;
    unsigned short count3 = 0x00AA;
    unsigned char direction0 = 1; // up
    unsigned char direction1 = 0; // down
    unsigned char direction2 = 1; // up
    unsigned char direction3 = 0; // down

    // open pwm and get a handle
    hPwm = vos_dev_open(VOS_DEV_PWM);

    // set prescaler value
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_PRESCALER_VALUE;
    pwm_iocb.count.prescaler = 0xF0;
    vos_dev_ioctl(hPwm, &pwm_iocb);

    // set counter value
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_COUNTER_VALUE;
    pwm_iocb.count.value = 0x00B0;
    vos_dev_ioctl(hPwm, &pwm_iocb);

    // set comparator 0 value
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_COMPARATOR_VALUE;
    pwm_iocb.identifier.comparator_number = COMPARATOR_0;
    pwm_iocb.comparator.value = count0;
    vos_dev_ioctl(hPwm, &pwm_iocb);
    // set comparator 1 value
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_COMPARATOR_VALUE;
    pwm_iocb.identifier.comparator_number = COMPARATOR_1;
    pwm_iocb.comparator.value = count1;
    vos_dev_ioctl(hPwm, &pwm_iocb);
    // set comparator 2 value
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_COMPARATOR_VALUE;
    pwm_iocb.identifier.comparator_number = COMPARATOR_2;
    pwm_iocb.comparator.value = count2;
    vos_dev_ioctl(hPwm, &pwm_iocb);
    // set comparator 3 value
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_COMPARATOR_VALUE;
    pwm_iocb.identifier.comparator_number = COMPARATOR_3;
    pwm_iocb.comparator.value = count3;
    vos_dev_ioctl(hPwm, &pwm_iocb);

    // enable comparator 0 for PWM_0 and PWM_4
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_OUTPUT_TOGGLE_ENABLES;
    pwm_iocb.identifier.pwm_number = PWM_0;
    pwm_iocb.output.enable_mask = MASK_COMPARATOR_0;
    vos_dev_ioctl(hPwm, &pwm_iocb);
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_OUTPUT_TOGGLE_ENABLES;
    pwm_iocb.identifier.pwm_number = PWM_4;
    pwm_iocb.output.enable_mask = MASK_COMPARATOR_0;
    vos_dev_ioctl(hPwm, &pwm_iocb);
    // enable comparator 1 for PWM_1 and PWM_5
```

```
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_OUTPUT_TOGGLE_ENABLES;
    pwm_iocb.identifier.pwm_number = PWM_1;
    pwm_iocb.output.enable_mask = MASK_COMPARATOR_1;
    vos_dev_ioctl(hPwm, &pwm_iocb);
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_OUTPUT_TOGGLE_ENABLES;
    pwm_iocb.identifier.pwm_number = PWM_5;
    pwm_iocb.output.enable_mask = MASK_COMPARATOR_1;
    vos_dev_ioctl(hPwm, &pwm_iocb);
    // enable comparator 2 for PWM_2 and PWM_6
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_OUTPUT_TOGGLE_ENABLES;
    pwm_iocb.identifier.pwm_number = PWM_2;
    pwm_iocb.output.enable_mask = MASK_COMPARATOR_2;
    vos_dev_ioctl(hPwm, &pwm_iocb);
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_OUTPUT_TOGGLE_ENABLES;
    pwm_iocb.identifier.pwm_number = PWM_6;
    pwm_iocb.output.enable_mask = MASK_COMPARATOR_2;
    vos_dev_ioctl(hPwm, &pwm_iocb);
    // enable comparator 3 for PWM_3 and PWM_7
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_OUTPUT_TOGGLE_ENABLES;
    pwm_iocb.identifier.pwm_number = PWM_3;
    pwm_iocb.output.enable_mask = MASK_COMPARATOR_3;
    vos_dev_ioctl(hPwm, &pwm_iocb);
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_OUTPUT_TOGGLE_ENABLES;
    pwm_iocb.identifier.pwm_number = PWM_7;
    pwm_iocb.output.enable_mask = MASK_COMPARATOR_3;
    vos_dev_ioctl(hPwm, &pwm_iocb);

    // set initial state
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_INITIAL_STATE;
    pwm_iocb.output.init_state_mask = 0x00; // all low initially
    vos_dev_ioctl(hPwm, &pwm_iocb);

    // set restore state
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_RESTORE_INITIAL_STATE;
    pwm_iocb.output.restore_state_mask = (MASK_PWM_0 | MASK_PWM_1 | MASK_PWM_2 |
MASK_PWM_3 | MASK_PWM_4 | MASK_PWM_5 | MASK_PWM_6 | MASK_PWM_7);    //all reset
    vos_dev_ioctl(hPwm, &pwm_iocb);

    // set mode to continuous
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_NUMBER_OF_CYCLES;
    pwm_iocb.output.mode = 0xB0; // 0 cycles -> continuous
    vos_dev_ioctl(hPwm, &pwm_iocb);

    // enable interrupt
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_ENABLE_INTERRUPT;
    vos_dev_ioctl(hPwm, &pwm_iocb);

    // enable output
    pwm_iocb.ioctl_code = VOS_IOCTL_PWM_ENABLE_OUTPUT;
    vos_dev_ioctl(hPwm, &pwm_iocb);

    do
    {
        // wait on interrupt
        // interrupt will be fired when No. of cycles counted
        pwm_iocb.ioctl_code = VOS_IOCTL_PWM_WAIT_ON_COMPLETE;
        vos_dev_ioctl(hPwm, &pwm_iocb);
        //-----------------------------
        // disable output
        pwm_iocb.ioctl_code = VOS_IOCTL_PWM_DISABLE_OUTPUT;
        vos_dev_ioctl(hPwm, &pwm_iocb);
        //-----------------------------
        // Comparator_0
```

```
// check for change of direction required
if (count0 == 0x00AF)
    direction0 = 0;
if (count0 == 0x0001)
    direction0 = 1;
// update comparator 0 value
if (direction0 == 1)
    count0++; // increment value
else
    count0--; // decrement value

// set value for comparator_0
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_COMPARATOR_VALUE;
pwm_iocb.identifier.comparator_number = COMPARATOR_0;
pwm_iocb.comparator.value = count0;
vos_dev_ioctl(hPwm, &pwm_iocb);
//-----------------------------
// Comparator_1
// check for change of direction required
if (count1 == 0x00AF)
    direction1 = 0;
if (count1 == 0x0001)
    direction1 = 1;
// update comparator 0 value
if (direction1 == 1)
    count1++; // increment value
else
    count1--; // decrement value

// set value for comparator_1
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_COMPARATOR_VALUE;
pwm_iocb.identifier.comparator_number = COMPARATOR_1;
pwm_iocb.comparator.value = count1;
vos_dev_ioctl(hPwm, &pwm_iocb);
//-----------------------------
// Comparator_2
// check for change of direction required
if (count2 == 0x00AF)
    direction2 = 0;
if (count2 == 0x0001)
    direction2 = 1;
// update comparator 0 value
if (direction2 == 1)
    count2++; // increment value
else
    count2--; // decrement value

// set value for comparator_2
pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_COMPARATOR_VALUE;
pwm_iocb.identifier.comparator_number = COMPARATOR_2;
pwm_iocb.comparator.value = count2;
vos_dev_ioctl(hPwm, &pwm_iocb);
//-----------------------------
// Comparator_3
// check for change of direction required
if (count3 == 0x00AF)
    direction3 = 0;
if (count3 == 0x0001)
    direction3 = 1;
// update comparator 0 value
if (direction3 == 1)
    count3++; // increment value
else
```



```
            count3--; // decrement value

        // set value for comparator_3
        pwm_iocb.ioctl_code = VOS_IOCTL_PWM_SET_COMPARATOR_VALUE;
        pwm_iocb.identifier.comparator_number = COMPARATOR_3;
        pwm_iocb.comparator.value = count3;
        vos_dev_ioctl(hPwm, &pwm_iocb);

        // enable output
        pwm_iocb.ioctl_code = VOS_IOCTL_PWM_ENABLE_OUTPUT;
        vos_dev_ioctl(hPwm, &pwm_iocb);
    }while(1);
}
```

5.3 The ADC function

This function is used to read the data from the ADC on the Vinco.

The data is read via the VNC2 SPI port and then sent to the display via the VNC2 UART port.

```
void ADC_thread(void)
{
   // general purpose variables
   unsigned short rgb[8]; // To store voltage reading for each ADC channel
   unsigned short ADC_Val = 0; // Raw ADC reading
   unsigned short i = 0;
   unsigned char j = 0;

   // Open GPIO device port A
   hGpio_A = vos_dev_open(VOS_DEV_GPIO_A);
   gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_SET_MASK;
   gpio_iocb.value = (LED1|LED2|PWREN_n|DISPRST_n);    // set bits 6, 2..0 as outputs, all other as
inputs
   vos_dev_ioctl(hGpio_A, &gpio_iocb);

   // Open GPIO device port B
   hGpio_B = vos_dev_open(VOS_DEV_GPIO_B);
   gpio_iocb.ioctl_code = VOS_IOCTL_GPIO_SET_MASK;
   gpio_iocb.value = 0x00;    // set all as input
   vos_dev_ioctl(hGpio_B, &gpio_iocb);

   // Open UART device
   hUart = vos_dev_open(VOS_DEV_UART);

   // Enable DMA for UART driver
   uart_iocb.ioctl_code = VOS_IOCTL_COMMON_ENABLE_DMA;
   vos_dev_ioctl(hUart,&uart_iocb);

   // Setup transmision paramemters
```

```
//set baud rate
uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_BAUD_RATE;
uart_iocb.set.uart_baud_rate = 230400;
vos_dev_ioctl(hUart,&uart_iocb);
//set flow control
uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_FLOW_CONTROL;
uart_iocb.set.param = UART_FLOW_NONE;
vos_dev_ioctl(hUart,&uart_iocb);
//set data bits
uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_DATA_BITS;
uart_iocb.set.param = UART_DATA_BITS_8;
vos_dev_ioctl(hUart,&uart_iocb);
// set stop bits
uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_STOP_BITS;
uart_iocb.set.param = UART_STOP_BITS_1;
vos_dev_ioctl(hUart,&uart_iocb);
// set parity
uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_PARITY;
uart_iocb.set.param = UART_PARITY_NONE;
vos_dev_ioctl(hUart,&uart_iocb);

// Attach UART to stdio driver
// Formatted strings with voltage readings will be sent
// to UART using 'printf' function.
stdioAttach(hUart);
//------------------------------------------
// Setup SPI Master
// open SPI Master and get a handle
hSPIm = vos_dev_open(VOS_DEV_SPIM);
// enable DMA
spim_iocb.ioctl_code = VOS_IOCTL_COMMON_ENABLE_DMA;
vos_dev_ioctl(hSPIm,&spim_iocb);
// set clock phase
spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_SCK_CPHA;
spim_iocb.set.param = SPI_MASTER_SCK_CPHA_1; // Data will be clocked in to ADC on rising edge
vos_dev_ioctl(hSPIm,&spim_iocb);
// set clock polarity
spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_SCK_CPOL;
spim_iocb.set.param = SPI_MASTER_SCK_CPOL_1; // Clock will be high in idle
vos_dev_ioctl(hSPIm,&spim_iocb);
// set data order
spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_DATA_ORDER;
```

```
spim_iocb.set.param = SPI_MASTER_DATA_ORDER_MSB; // MSB first
vos_dev_ioctl(hSPIm,&spim_iocb);
// set clock rate
spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_SET_SCK_FREQUENCY;
spim_iocb.set.spi_master_sck_freq = 3000000;
vos_dev_ioctl(hSPIm,&spim_iocb);
// Set data delay
spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_SET_DATA_DELAY;
spim_iocb.set.param = 0;
vos_dev_ioctl(hSPIm,&spim_iocb);
// set initial state of chip select 0 pin
spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_SS_0;
spim_iocb.set.param = SPI_MASTER_SS_DISABLE;
vos_dev_ioctl(hSPIm,&spim_iocb);
// set initial state of chip select 1 pin
spim_iocb.ioctl_code = VOS_IOCTL_SPI_MASTER_SS_1;
spim_iocb.set.param = SPI_MASTER_SS_DISABLE;
vos_dev_ioctl(hSPIm,&spim_iocb);

// open ADC driver
hAdc = vos_dev_open(VOS_DEV_ADC);

// attach ADC driver to SPI master
adc_iocb.ioctl_code = VOS_IOCTL_ADC_MCP3008_ATTACH;
adc_attach_info.spi_master_handle = hSPIm;
adc_attach_info.chip_select_identifier = ADC_MCP3008_CHIP_SELECT_1;
adc_iocb.attach_info =  &adc_attach_info;
vos_dev_ioctl(hAdc,&adc_iocb);

//------------------------------------------
// Initialise display
Display_Init();
//------------------------------------------
// loop
do
{
   // Read all 8 channels of ADC in single-ended mode.
   for(j=0;j<8;j++)
   {
      adc_iocb.ioctl_code = VOS_IOCTL_ADC_MCP3008_READ_CHANNEL;
      adc_iocb.mode = ADC_MCP3008_MODE_SINGLE_ENDED;
      adc_iocb.channel = j;
```

```
    vos_dev_ioctl(hAdc,&adc_iocb);
    // Convert the value to mV
    ADC_Val = ((adc_iocb.value*32) / 10);
    rgb[j] = ADC_Val;
}

//------------------------------------------
// Display voltage readings.
for(j=0;j<=7;j++)
{
    // Set font color and size
    // font or background color have 16-bit value as follows:
    //      [R4R3R2R1R0 G5G4G3G2G1G0 B4B3B2B1B0]
    // It is possible to display 65000 colors
    ADC_Val = ((rgb[0]>>5) & 0x1F); // RED value
    ADC_Val = (ADC_Val << 6) | ((rgb[1]>>4) & 0x3F); // GREEN value
    ADC_Val = (ADC_Val << 5) | ((rgb[2]>>5) & 0x1F); // BLUE value
    ADC_buff[0]= GSGC_STRINGTXT; // Command
    ADC_buff[1]= 3;   // Cloumn
    ADC_buff[2]= j+1;   // Row
    ADC_buff[3]= FONT3; // Font
    ADC_buff[4]= ((ADC_Val >> 8) & 0xFF); // Colour MSB
    ADC_buff[5]= (ADC_Val & 0xFF); // Colour LSB
    vos_dev_write(hUart, ADC_buff, 6, NULL);
    // Display ADC readings
    if(rgb[j] < 10)
    {
        printf(" Ch-%u: 0.00%uV", (j & 0x0F), rgb[j]);
    }
    else if(rgb[j] < 100)
    {
        printf(" Ch-%u: 0.0%uV", (j & 0x0F), rgb[j]);
    }
    else if(rgb[j] < 1000)
    {
        printf(" Ch-%u: 0.%uV", (j & 0x0F), rgb[j]);
    }
    else // ADC_Val > 1000
    {
        i = (rgb[j] % 1000);
        if(i < 10)
        {
```

```
            printf(" Ch-%u: %u.00%uV", (j & 0x0F), ((rgb[j] / 1000) & 0xFFFF),(i & 0xFFFF));
          }
          else if(i < 100)
          {
            printf(" Ch-%u: %u.0%uV", (j & 0x0F), ((rgb[j] / 1000) & 0xFFFF),(i & 0xFFFF));
          }
          else // ADC_Val % 1000 > 100
          {
            printf(" Ch-%u: %u.%uV", (j & 0x0F), ((rgb[j] / 1000) & 0xFFFF),(i & 0xFFFF));
          }
        }
        ADC_buff[0]= '\0'; // String termination
        vos_dev_write(hUart, ADC_buff, 1, NULL);
      }
      //-------------------------------------------
      // Set background color
      ADC_Val = ((rgb[3]>>5) & 0x1F); // RED value
      ADC_Val = (ADC_Val << 6) | ((rgb[4]>>4) & 0x3F); // GREEN value
      ADC_Val = (ADC_Val << 5) | ((rgb[5]>>5) & 0x1F); // BLUE value
      ADC_buff[0]= GSGC_BACKGND_REPLACE; // Command
      ADC_buff[1]= ((ADC_Val >> 8) & 0xFF); // Color MSB
      ADC_buff[2]= (ADC_Val & 0xFF); // Color LSB
      vos_dev_write(hUart, ADC_buff, 3, NULL);
      // Wait for ACK
      do
      {
        if(Get_ACK())
        {
          break;
        }
      }while(1);

  } while(1);
}
```

5.4 Display Commands

5.4.1 Display Initialisation

This function will initialize the oLED display.

```
void Display_Init(void)
{
  unsigned char j;
```

```
// Toggle display RESET# pin
j = 0xFF&(~DISPRST_n);
vos_dev_write(hGpio_A, &j, 1, NULL);
vos_delay_msecs(10);
j = 0xFF;
vos_dev_write(hGpio_A, &j, 1, NULL);
vos_delay_msecs(1000);

// Durring power-up display may send garbage.
// Perform a dummy read from the display.
do
{
    // Get Rx queue
    uart_iocb.ioctl_code = VOS_IOCTL_COMMON_GET_RX_QUEUE_STATUS;
    vos_dev_ioctl(hUart, &uart_iocb);
    j = uart_iocb.get.queue_stat; // How much data to read?
    if(j > 0)
    {
        // Read data from UART
        vos_dev_read(hUart, ADC_buff, (j & 0xFF), NULL);
    }
}while(j > 0);

// Try to synchronize baudrate with display.
do
{
    // Turn on LED2#
    j=(~LED2);
    vos_dev_write(hGpio_A, &j, 1, NULL);
    // Send AUTOBAUD command to the display
    ADC_buff[0] = GSGC_AUTOBAUD;
    vos_dev_write(hUart, ADC_buff, 1, NULL);
    vos_delay_msecs(100);
    // Turn off LED2#
    j=0xFF;
    vos_dev_write(hGpio_A, &j, 1, NULL);
    // Check if ACK from display received
    if(Get_ACK())
    {
        break;
    }
}while(1);
```

```
    // Setup the display
    ADC_buff[0] = GSGC_SETOPAQUE;
    ADC_buff[1] = 1;
    vos_dev_write(hUart, ADC_buff, 2, NULL);
    vos_delay_msecs(50);
    // Wait for ACK from display
    do
    {
        if(Get_ACK())
        {
            break;
        }
    }while(1);

    ADC_buff[0] = GSGC_DISPCONT;
    ADC_buff[1] = 2; // Mode = (adjust contrast)
    ADC_buff[2] = 0x0F; // Contrast value (0x0F..0x00)
    vos_dev_write(hUart, ADC_buff, 3, NULL);
    vos_delay_msecs(50);
    // Wait for ACK from display
    do
    {
        if(Get_ACK())
        {
            break;
        }
    }while(1);
}
```

5.4.2 Display Acknowledgement

```
unsigned char Get_ACK(void)
{
    unsigned char i;

    // Check if ACK from display received
    uart_iocb.ioctl_code = VOS_IOCTL_COMMON_GET_RX_QUEUE_STATUS;
    vos_dev_ioctl(hUart, &uart_iocb);
    i = uart_iocb.get.queue_stat; // How many bytes to read?
    if(i>0)
    {
        // Read data from UART
        vos_dev_read(hUart, ADC_buff, (i & 0xFF), NULL);
```

```
        if(ADC_buff[0] == ACK)
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }
}
```

6 Programming Vinco

When Vinco has been connected to the oLED panel and the firmware has been built in the IDE, the next step is to transfer the .ROM file generated by the IDE to the Vinco module. The IDE generates the .ROM file with a single button click of the "Build" button.

Connect the USB port of the VNC2 Debug Module to a PC and load the free FTDI drivers for the FT232R device on the debug module. This will happen automatically via Windows Update if you are connected to their internet. Otherwise refer to the installation guide for your OS:

http://www.ftdichip.com/Support/Documents/InstallGuides.htm

The IDE should now automatically detect the VNC2 debug module.

Connect the other end of the VNC2 Debug Module to the J8 connector of the Vinco.

Use the IDE FLASH button to load the .ROM file into the Vinco. A getting started guide for using the Vinculum IDE may be downloaded from:

http://www.ftdichip.com/Support/Documents/AppNotes/AN_142_Vinculum-II_Tool_Chain_Getting_Started_Guide.pdf

The IDE will report back a successful programming. At this point the VNC2 Debug module may be removed from the Vinco J8 connector.

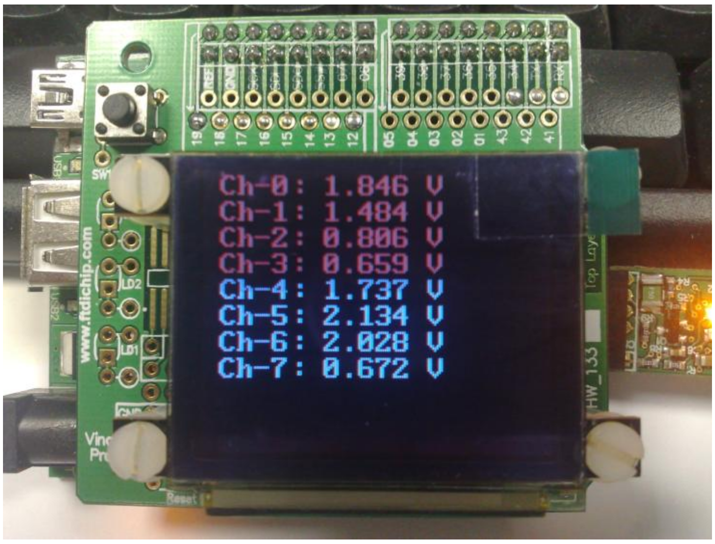
The .rom file can also be downloaded from the following location:

http://www.ftdichip.com/Support/SoftwareExamples/VinculumIIProjects/Vinco_Volt_Meter_Demo.zip

7 Running the firmware

The Vinco may be reset by power cycling the unit and then the firmware will run.

The user will observe text on the oLED being updated as per the firmware code:



It is left to the user to experiment with changing the displayed text by modifying the sample project code to adjust the PWM outputs.

8 Contact Information

Head Office – Glasgow, UK

Future Technology Devices International Limited
Unit 1, 2 Seaward Place,
Centurion Business Park
Glasgow, G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales) sales1@ftdichip.com
E-mail (Support) support1@ftdichip.com
E-mail (General Enquiries) admin1@ftdichip.com
Web Site URL http://www.ftdichip.com
Web Shop URL http://www.ftdichip.com

Branch Office – Taipei, Taiwan

Future Technology Devices International Limited (Taiwan)
2F, No 516, Sec. 1 NeiHu Road
Taipei 114
Taiwan, R.O.C.
Tel: +886 (0) 2 8791 3570
Fax: +886 (0) 2 8791 3576

E-mail (Sales) tw.sales1@ftdichip.com
E-mail (Support) tw.support1@ftdichip.com
E-mail (General Enquiries) tw.admin1@ftdichip.com
Web Site URL http://www.ftdichip.com

Branch Office – Hillsboro, Oregon, USA

Future Technology Devices International Limited (USA)
7235 NW Evergreen Parkway, Suite 600
Hillsboro, OR 97123-5803
USA
Tel: +1 (503) 547 0988
Fax: +1 (503) 547 0987

E-Mail (Sales) us.sales@ftdichip.com
E-Mail (Support) us.support@ftdichip.com
E-Mail (General Enquiries) us.admin@ftdichip.com
Web Site URL http://www.ftdichip.com

Branch Office – Shanghai, China

Future Technology Devices International Limited (China)
Room 408, 317 Xianxia Road,
ChangNing District,
ShangHai, China

Tel: +86 (21) 62351596
Fax: +86(21) 62351595

E-Mail (Sales): cn.sales@ftdichip.com
E-Mail (Support): cn.support@ftdichip.com
E-Mail (General Enquiries): cn.admin1@ftdichip.com
Web Site URL http://www.ftdichip.com

Distributor and Sales Representatives

Please visit the Sales Network page of the FTDI Web site for the contact details of our distributor(s) and sales representative(s) in your country.

Appendix A – References

Application and Technical Notes available at
http://www.ftdichip.com/Support/Documents/AppNotes.htm

Vinco datasheet

http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_Vinculum-II.pdf

VNC2 Debug Module

http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_Vinculum-II.pdf

Vinculum-II IO Cell Description

http://www.ftdichip.com/Support/Documents/AppNotes/AN_137_Vinculum-II%20IO_Cell_Description.pdf

Vinculum-II Debug Interface Description

http://www.ftdichip.com/Support/Documents/AppNotes/AN_138_Vinculum-II_Debug_Interface_Description.pdf

Vinculum-II IO Mux Explained

http://www.ftdichip.com/Support/Documents/AppNotes/AN_139_Vinculum-II%20IO_Mux%20Explained.pdf

Vinculum-II PWM Example

http://www.ftdichip.com/Support/Documents/AppNotes/AN_140_Vinculum-II_PWM_Example.pdf

Vinculum-II Errata Technical Note

http://www.ftdichip.com/Support/Documents/TechnicalNotes/TN_118_VNC2%20Errata%20Technical%20Note.pdf

4D system uoLED -160-G1SGC Display datasheet

(http://www.4dsystems.com.au/downloads/Serial-Display-Modules/uOLED-160-G1(SGC)/Docs/uOLED-160-G1SGC-DS-rev4.pdf)

Appendix B – List of Figures and Tables

List of Figures

List of Tables

Appendix C – Revision History

Version 1.0	First release	7th November 2010
Version 2.0	Change Vinculo brand name to vinco	14th April 2011

Appendix D Legal Disclaimer:

System and equipment manufacturers and designers are responsible to ensure that their systems, and any Future Technology Devices International Ltd (FTDI) devices incorporated in their systems, meet all applicable safety, regulatory and system-level performance requirements. All application-related information in this document (including application descriptions, suggested FTDI devices and other materials) is provided for reference only. While FTDI has taken care to assure it is accurate, this information is subject to customer confirmation, and FTDI disclaims all liability for system designs and for any applications assistance provided by FTDI. Use of FTDI devices in life support and/or safety applications is entirely at the user's risk, and the user agrees to defend, indemnify and hold harmless FTDI from any and all damages, claims, suits or expense resulting from such use. This document is subject to change without notice. No freedom to use patents or other intellectual property rights is implied by the publication of this document. Neither the whole nor any part of the information contained in, or the product described in this document, may be adapted or reproduced in any material or electronic form without the prior written consent of the copyright holder. Future Technology Devices International Ltd, Unit 1, 2 Seaward Place, Centurion Business Park, Glasgow G41 1HH, United Kingdom. Scotland Registered Company Number: SC136640