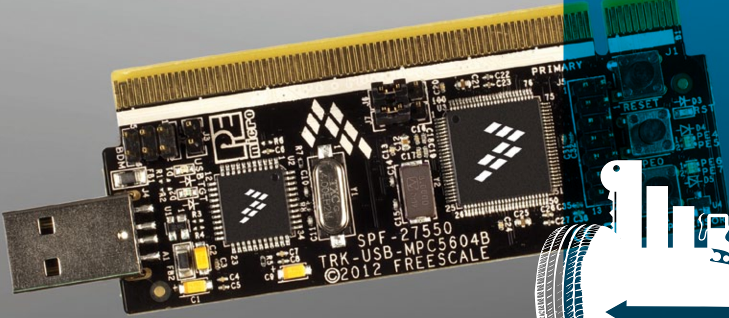


Lab Tutorial

TRK-USB-MPC5604B
StarterTRAK USB Kit

StarterTRAK

Discover the Power of Qorivva

The Freescale StarterTRAK USB kit (TRK-USB-MPC5604B) is designed to allow you to easily and inexpensively test drive our 32-bit Qorivva MCUs. Along with the on-board MPC5604B Qorivva MCU, the StarterTRAK kit features a PCI card edge for compatibility with the Freescale Tower System, which means you can create your own custom development system based on your design needs.

Purpose

The purpose of this document is to introduce you to the MPC5604B device. The guide contains two main examples that provide access to some of the features of the device and the board. The first demonstration provides a simple command line interface to access digital input and output, analog-to-digital converter (ADC) and pulse width modulation (PWM) features of the device whilst the second demonstration reads a value using the temperature sensor on the TRK-USB-MPC5604B board and outputs the value in two different ways.

Initial Setup

Step-by-Step Instructions

1 Configuring Hardware

The hardware contains two jumpers, J6 and J7, for routing the Linflex0 transmit and receive signals either via the OSJTAG virtual serial port or via the Tower connector. For the examples within this lab guide, it is assumed the virtual serial port is used and jumpers J6 and J7 on the TRK-USB-MPC5604B board should be in the 1-2 position.

The same demonstration(s) can also be used in conjunction with Tower System elevators and the TWR-SER board to interface via serial cable. In this case, jumpers J6 and J7 on the TRK-USB-MPC5604B board should be in the 2-3 position.

2 Configuring Software

It is recommended that you download CodeWarrior V2.9 to allow you to write programs for the device, and to give access to the P&E debugger that can load new programs to the flash memory on the device. Additionally, for runtime, you may wish to use the P&E Virtual Serial Toolkit to communicate with the device. See “Using the P&E Serial Tools” for more information on these tools.

Demo 1: An Interactive MPC5604B Shell

The interactive shell for the TRK-USB-MPC5604B board allows you to request that the MPC5604B part perform various functions using the dedicated on-chip hardware.

This program starts the part running from the internal PLL running at 64 MHz and uses the LinFlex0 module to communicate with a host computer over a standard RS232 serial connection. This allows plugging the TRK-USB-MPC5604B board into a USB socket on a computer and controlling it via the P&E OSBDM OSJTAG virtual serial toolkit's terminal utility.

MPC5604B functions illustrated in this demo include:

- Read a general-purpose digital input signal from a pin
- Write a general-purpose digital output to a pin
- Read an analog value from an ADC pin
- Produce a configured PWM output using an eMIOS0 output pin

This lab note explains how these functionalities are provided along with the process of initializing the part for 64 MHz operation and connection over RS232. The implementation of the console, including parsing parameters and executing function calls, is not covered within this lab guide.

In each case, the application accepts a variety of parameters including a number representing a system integration unit lite (SIUL) port number, a pin number (starting with p or P) or a port code (starting with a-e or h).

In addition to the functional instructions, built-in help and port functionality lookup commands are included. Use “help” at the command line to access the online help or “port <port-identifier>” for port information.

The MPC5604B pins, SIUL identifiers and port numbers, along with the features available within this application, are tabulated below.

Startup

This section describes basic startup for the MPC5604B MCU.

The CodeWarrior startup script for the MPC5604B performs basic initialization of the device without the user having to write code. User initialization that must be configured involves setting up the device clocks, enabling required peripherals and choosing a strategy to handle the hardware watchdog.

For the purposes of these demonstrations, the watchdog strategy employed is to simply disable the watchdog hardware that is enabled at reset by default. As explained within the device reference manual, the watchdog control register is unlocked by writing two passwords to the watchdog service register. The code to complete this is shown below:

```
    SWT.SR.R = 0x0000c520; /* key */
    SWT.SR.R = 0x0000d928; /* key */
    SWT.CR.R = 0x8000010A; /* disable WEN */
```

Having completed this, all peripherals are enabled using the following code:

```
/* Enable all modes */
ME.MER.R = 0x000025FF;

/* ME_PCTL[all periph] default points to ME_RUN_PC[0] *
ME.RUNPC[0].R = 0x000000FE; /* Peripheral ON in every mode *

/* Re-enter mode to latch mode settings */
ME.MCTL.R = 0x30005AF0;          /* Mode & Key */
ME.MCTL.R = 0x3000A50F;          /* Mode & Key inverted */
```

No changes are made to the default clock configuration; this means that the device will be running at 16 MHz using the fast internal RC oscillator.

Linflex Configuration for the Console

The console within this laboratory guide is provided over a universal asynchronous receiver/transmitter (UART) connection. The console is contained within the file serialio.c, but the key information within this file relates to the configuration of the Linflex0 module for use in UART mode.

The `void config_sci(void)` function includes a section that initializes the module in this way. The code is commented and presented below to show how this is achieved.

```
// config LINLFEX
LINFLEX_0.LINCR1.B.SLEEP   = 0; // Disable LINFLEX sleep mode
LINFLEX_0.LINCR1.B.INIT    = 1; // LINFLEX in init mode
LINFLEX_0.UARTCR.B.UART    = 1; // Run LINFLEX in UART mode
```

```
    LINFLEX_0.UARTCR.B.TDFL  = 0; // 1byte buffer
    LINFLEX_0.UARTCR.B.RDFL  = 0; // 1byte buffer
    LINFLEX_0.UARTCR.B.RXEN  = 1; // rx enable
    LINFLEX_0.UARTCR.B.TXEN  = 1; // tx enable
    LINFLEX_0.UARTCR.B.PCE   = 0; // no Parity
    LINFLEX_0.UARTCR.B.WL    = 1; // 8bit data

    LINFLEX_0.LINIBRR.B.DIV_M         = 8; // 115200 Baud (at 16 MHz
    LINFLEX_0.LINFBRR.B.DIV_F         = 11;
    LINFLEX_0.LINCR1.B.INIT           = 0; // LINFLEX in run mode

    LINFLEX_0.LINIER.B.DRIE=1;        // enable SCI.RX interrupt
    LINFLEX_0.LINIER.B.DTIE=1;        // enable SCI.TX interrupt
```

Actions

The demonstration application allows for four different actions to be performed, as described in the “Runtime Command Reference” section. These actions are all handled within the “`action.c`” source file and the availability of functionality on pins is stored within an array, `pinoutTable[]`, in “`ports.c`”. This table contains 100 entries, one per pin, containing the pin number, the SIU peripheral configuration register number, the port number, a description and a 32-bit number comprising four 8-bit numbers describing general-purpose input, general-purpose output, ADC and PWM (eMIOS) availability. These entries are each an instance of a `PinDescriptor` struct, as defined in “`ports.h`” and a helper function exists within “`ports.c`” to find the appropriate descriptor.

Reading a General-Purpose Digital Input Signal

In order to read a digital input from a pin, you must select a pin that supports general-purpose input (GPI), configure the pin's input buffer and then read the value. For a list of pins that can be used in this manner, use the "`pinout digin`" command within the application.

The action code to read a GPI is within the `processDiginRequest()` function. This function first attempts to retrieve a pin descriptor from the command and, if a pin is found, compatibility is checked. If compatible, the following code reads the status and print out either "high" or "low" depending on the pin input buffer status.

```
SIU.PCR[pin->siu].R = 0x100;
if (SIU.GPDI[pin->siu].R) {
         printf("high\r\n");
} else {
         printf("low\r\n");
}
```

Writing a General-Purpose Digital Output Signal

Writing a digital output follows the same pattern as reading an input; you simply must choose a pin supporting general-purpose output (GPO) and configure the pin's output buffer before writing the value via the system integration unit (SIU). The command "`pinout digout`" can be used within the application to list supported ports. Pins used for the UART connection to the control application are not made available by the MCU application to ensure control of the device is maintained.

The action code to write this is within the `processDigoutRequest()` function and this follows the same pattern as the GPI action. In this, if a pin is located, the anned for the literal “high” or “low” and this determines the output value. The code for this is shown below.

```
        SIU.PCR[pin->siu].R = 0x200;
        token = strtok(NULL, “ “);
        if (token != NULL && strcmp(token, “high”) == 0) {
                SIU.GPDO[pin->siu].R = 1;
        } else {
                SIU.GPDO[pin->siu].R = 0;
```

It is worth noting that the LEDs sink current via the ports E4-E7 so these ports must be set low in order to light the appropriate LED.

Reading an Analog Value

Reading an analog value again requires the identification of an appropriate pin, the command “`pinout adc`” can be used within the application to list supported pins. Having identified a pin, the action code to read the value must configure the ADC to perform an ADC conversion, start the conversion, wait for completion and then read/display the sampled value.

Prior to reading an ADC value, the ADC must be powered on. This is completed by the following single line in `main.c`:

```
        ADC_0.MCR.B.PWDN = 0;       /* ADC enable */
```

The code for then undertaking the action is within the `processAdcRequest()` function and is shown below:

```
    const struct PinDescriptor *pin;
    char *token = strtok(NULL, “ “);
    pin = findDescriptor(token);
    if (pin == NULL) {
        printf("Pin not found\r\n");
    } else {
        if ((pin->capability & 0x0000FF00) == 0x0000FF00) {
            printf("ADC input is not supported on this pin\r\n");
        } else { // Pin found and supported - set up!
            int adcChannel = (pin->capability & 0xFF00) >> 8;

            SIU.PCR[pin->siu].R = 0x2000; // Pin in analog mode
            ADC_0.NCMR0.R = 0; // Mask disable for all channels
            ADC_0.NCMR1.R = 0; // Mask disable for all channels
            ADC_0.NCMR2.R = 0; // Mask disable for all channels

            if (adcChannel < 16) {
                ADC_0.NCMR0.R = (1 << adcChannel);
                         // Mask enable for channel
            } else if (adcChannel > 31 && adcChannel < 48) {
                ADC_0.NCMR1.R = (1 << (adcChannel - 32));
                         // Mask enable for channel
            } else if (adcChannel > 63) {
                ADC_0.NCMR2.R = (1 << (adcChannel - 64));
                         // Mask enable for channel
            }

            ADC_0.MCR.B.NSTART = 1;               /* ADC Go! */
            while (ADC_0.MSR.B.NSTART);
                         // Wait for scan to complete
            printf("%d\r\n", ADC_0.CDR[adcChannel].B.CDATA);
        }
    }
```

On the MPC5604B device, there are three different registers corresponding to precision, standard and multiplexed ADC channels. These are configured in

NMCR[0..2] and the appropriate registers must be chosen. The function reads the ADC channel corresponding to the pin from the pin descriptor and chooses the appropriate register based on the channel number.

The code then initiates the ADC conversion and prints the result as a decimal value to the console. The ADC reads a 10-bit number corresponding to an analog input between 0 and 5 V. This means the output will be between 0 and 1023.

The temperature sensor on the device is attached to port B5 so using "`adc b5`" within the application will display a number corresponding to temperature.

Generating a PWM Signal

The eMIOS block is highly configurable and can both read and generate various pulse-width modulation (PWM) signals. This demonstration uses the eMIOS only in PWM output mode, driven by the counter bus A generated by channel 23. The application supports definition of the duty cycle of the PWM as a percentage value. Pins available for the PWM output can be listed using the command "`pinout pwm`" in the application.

Before describing the PWM command, the eMIOS initialization code is shown. This code enables the module and configures channel 23 to act as a global counter bus for other channels. This code is shown below:

```
EMIOS_0.MCR.B.MDIS = 0;             /* enable clks to eMIOS*/
EMIOS_0.MCR.B.GPRE= 7;
        /* Divide 8 MHz FXOSC by 7+1 = 8 for 1MHz eMIOS clk */
EMIOS_0.MCR.B.GPREN = 1;            /* Enable eMIOS clock */
EMIOS_0.MCR.B.GTBE = 1;             /* Enable global time base */
```

```
        EMIOS_0.UCDIS.B.CHDIS23 = 0; /* enable ch23*/
        EMIOS_0.CH[23].CADR.R = 1023;
                /* Period will be 1023+1 = 1024 clocks */
        EMIOS_0.CH[23].CCR.B.MODE = 0x50;
                /* Modulus Counter Buffered (MCB) */
        EMIOS_0.CH[23].CCR.B.BSL = 0x3;
                /* Use internal counter */
        EMIOS_0.CH[23].CCR.B.UCPRE=0;
                /* Set channel prescaler to divide by 1 */
        EMIOS_0.CH[23].CCR.B.UCPEN = 1;
                /* Enable prescaler; uses default divide by 1 */
```

Having set these up, the individual action code is similar again to the other action. The code is shown below and can be found in the `process PwmRequest()` function in "`action.c`":

```
   int high;
   const struct PinDescriptor *pin;

   char *token = strtok(NULL, " ");
   high = 100 - atoi(token);

   token = strtok(NULL, " ");
   pin = findDescriptor(token);

   if (pin == NULL) {
        printf("Pin not found\r\n");
   } else {
        if ((pin->capability & 0x000000FF) == 0x000000FF) {
                printf("PWM is not supported on this pin\r\n");
        } else { // Pin found and supported - set up!
                int channel = pin->capability & 0xFF;

          EMIOS_0.UCDIS.B.CHDIS0 = 0;                    /* enable ch0*/
          EMIOS_0.CH[channel].CADR.R = (1023 * high) / 100;
                /* Leading edge when channel's counter bus= % of 1023 */
          EMIOS_0.CH[channel].CBDR.R = 1023;
                /* Trailing edge when channel's counter bus=1023 */
          EMIOS_0.CH[channel].CCR.B.BSL = 0x0;
                /* Use counter bus A */
          EMIOS_0.CH[channel].CCR.B.EDPOL = 1;
                /* Polarity-leading edge sets output/trailing clears */
          EMIOS_0.CH[channel].CCR.B.MODE = 0x60;
                /* Output Pulse Width Modulation Buffered (flag on B1 match) */
          SIU.PCR[pin->siu].R = 0x0600;
                /* Initialize pad for eMIOS output */
       }
    }
```

eMIOS is available on three of the four channels that can drive the LEDs on the TRK-USB-MPC5604B board (ports E4-E6). eMIOS on pin E7 is listed as being available within the reference manual but is not exposed in this application as it is used as a global reference clock.

Runtime Command Reference

The console responds to the following commands.

Command	Description
echo	Display the current echo setting and determines whether user input is echoed back
echo [off|on]	Turn off or on the echo of commands
help	Display a list of commands
help <command>	Display help for the specified command
pinout	Display the device pinout details
pinout <pinspec>	Display information about the specified pin
adc <pinspec>	Use the ADC to read the analog voltage on the specified pin. Output is in decimal between 0 (0 V) and 1024 (5 V)
pwm <high> <pinspec>	Use the eMIOS module to generate a PWM with period high for <high>% of time on the specified pin
digin <pinspec>	Read the digital state of the specified pin
digout <pinspec> <high|low>	Write the specified digital state of the specified pin

ic Interrupt Timer

The periodic interrupt timer (PIT) can automatically trigger the ADC rather than having to start a sample and wait as was seen in Demo 1. In this case, the PIT is initialized using the following code:

```
void PIT2_init(uint32_t LDVAL)
{
      PIT.CH[2].LDVAL.R          = LDVAL;   /* load PIT counter */
      PIT.CH[2].TCTRL.B.TEN     = 1;       /* enable channel */
      PIT.PITMCR.B.MDIS          = 0;       /* enable PIT module */
}
```

The value passed as LDVAL is 0x186A00 which corresponds to 1.6 m decimal. As the PIT decrements from this value once per clock cycle, this will result in 10 PIT timeouts per second giving a 10 Hz sampling rate. When the period has ended, the ADC is configured to automatically sample as described in the next section.

PIT Triggered ADC

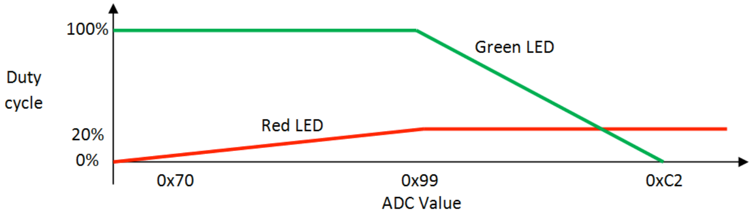
The ADC is configured to automatically trigger on the PIT 10 Hz signal. The configuration in this case is shown below:

```
SIU.PCR[21].R = 0x2000;  // PB5 in analog mode

ADC_0.MCR.B.PWDN = 0;               /* ADC_0 enable */
ADC_0.MCR.B.MODE = 0;    /* Start by SW mode */
ADC_0.MCR.B.JTRGEN = 1;  /* Start by PIT2 */
ADC_0.JCMR0.R = 0x2;                /* Enable ANP1 for injected */
ADC_0.IMR.R = 0x08;                 /* interrupt on end of chain */
ADC_0.CIMR0.R = 0x2;                /* Enable interrupt on channel 2 */
```

Demo 2: The Temperature Sensor

The application is designed to read data from three sources at 10 times per second. The data sources are temperature sensor (analog) and the two buttons (digital) on the TRK-USB-MPC5604B. The application then writes the data out to the Linflex0 in UART mode allowing viewing and charting using the P&E OSBDM OSJTAG virtual serial toolkit's serial grapher utility. Simultaneous to this, the analog value read from the temperature sensor is used to drive the eMIOS channels supplying PWM signals to the green and red LEDs attached to ports E4 and E5 respectively. The digital values read from the switches are also used to set the state of the digital output to the green and red LEDs attached to ports E6 and E7.

The design duty cycles for different input signals are shown in the chart below. Note that the duty cycle for the red LED is given an upper value of 20 percent. This is chosen so that on the centre point where duty cycles are green@100 percent and red@20 percent gives a roughly yellow output from the dual LED. The center point was chosen around room temperature and a variation of 41 samples. This corresponds to 41 * 5 V/1024 = 0.2 V swing. This swing corresponds to around 20 °C as the LM61 temperature sensor is specified as having an output change of 10 mV/°C.



shows the use of features within Demo 1 along with a PIT.

The high-level application code simply initializes all of the required modules for this project and is shown below:

```
int main(void) {
      int i = 0;

      DISABLE_SWT();
      setupSerialAndInterrupts();

      setupEmios();
      setupGpio();
      initADC_0();
      PIT2_init(0x186A00); // 1/10 second

      while(1) {
                i++;
      }
}
```

Many steps here, including disabling the SWT, turning on power, setting up the Linflex in UART mode, and initializing eMIOS, ADC and SIU modules, are described in the previous demonstration. Areas of difference, notably setting up the PIT to trigger the ADC and adding an ADC end of chain interrupt to update the device outputs, are described below.

This code configures the ADC as seen in the first demonstration but additionally sets the JTRGEN flag to enable the PIT trigger. Since this hardware trigger is used, the injected channel mask register (JCMR) is configured instead of NCMR for port B5.

is then set to fire on the end of the chain and the interrupt generation is enabled for the selected channel. This interrupt is configured within the **`setupSerialAndInterrupts()`** function where interrupt 62 (ADC interrupt) calls the **`void ADC_0EocComplete(void)`** function. This function is described in the next section.

The End of Chain Interrupt

The completion of an ADC conversion is the trigger for the part to update the data being sent to the serial grapher application and being displayed on the board LEDs. The function is shown below:

```
// Print out sensor and switch data in format for P&E Grapher Tool
printf(“W%02xZ%02xY%02xX%02x\r\n”, 0, (SIU.GPDI[65].R?0:0xFF),
                                  (SIU.GPDI[64].R?0:0xFF), (value & 0xFF));
printf(“A%02xB%02xC%02xD%02x\r\n”, 0, (SIU.GPDI[65].R?0:0xFF),
                                  (SIU.GPDI[64].R?0:0xFF), (value & 0xFF));

// Map switches to the LEDs
SIU.GPDO[70].R = SIU.GPDI[64].R;
SIU.GPDO[71].R = SIU.GPDI[65].R;

// Map temp sensor to the eMIOS duty cycles
if (value < 0x99) { // GREEN always ON
      EMIOS_0.CH[20].CADR.R = 1023;
      if (value < 0x70) {
                // RED is OFF
                EMIOS_0.CH[21].CADR.R = 0;
      } else {
                // RED varies
                EMIOS_0.CH[21].CADR.R = (1023 * (value – 0x70)) / 200;
      }
```

```
          (value > 0x99) { // RED always ON
        if (value > 0xC2) {          // GREEN is OFF
                EMIOS_0.CH[20].CADR.R = 0;
        } else {                     // GREEN varies
                EMIOS_0.CH[20].CADR.R = (1023 * (0xC2 - value)) / 20;
        }
        EMIOS_0.CH[21].CADR.R = 102;
    } else {
        EMIOS_0.CH[20].CADR.R = 1023;
        EMIOS_0.CH[21].CADR.R = 102;
    }

    // Clear ADC Interrupt Status Register
    ADC_0.ISR.R = 0x08;
```

This code simplifies the ADC value from the channel data register before printing out, to the serial line, the sampled value and the switch statuses. The format of this is defined in the P&E “OSBDM/OSJTAG Virtual Serial Toolkit Resources” document.

There then follows a series of if-else statements that configure the eMIOS outputs according to the duty cycle transfer chart shown previously.

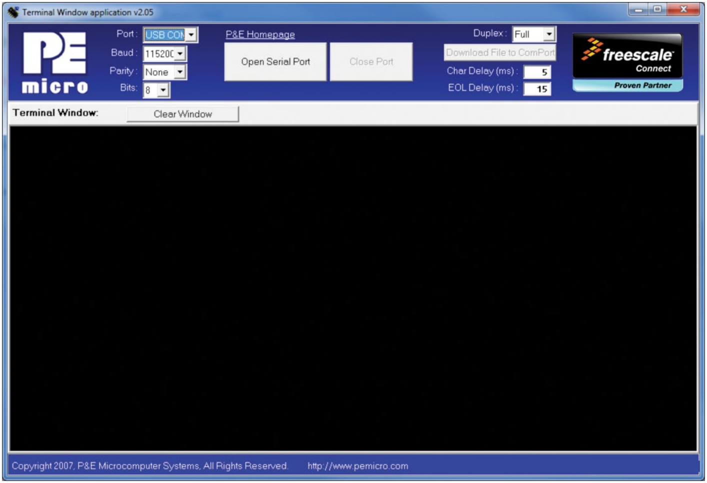
The final instruction in the handler resets the interrupt status register.

Using the P&E Serial Tools

The P&E OSBDM virtual serial toolkit is available from **pemicro.com/osbdm/index.cfm**. This toolkit includes applications that allow you to interact directly, using your PC keyboard and monitor, with the MCU on the TRK-USB-MPC5604B. There are two utilities that are of particular interest for the examples in this laboratory guide.

The Terminal Utility

When the terminal utility is started, the user is presented with the following screen.



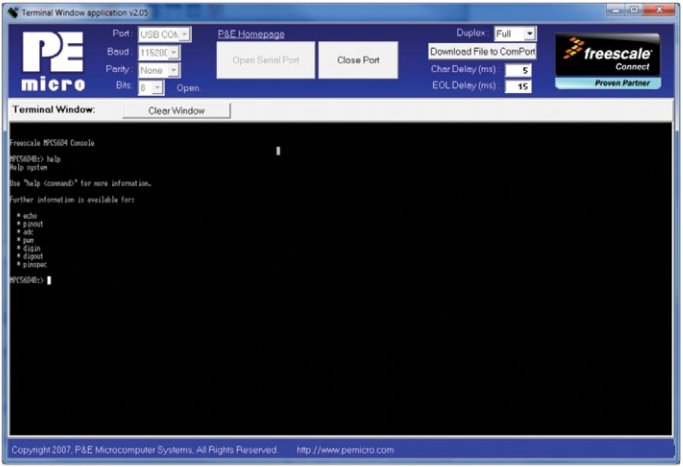
Terminal Window application v2.05

Port : USB COM
Baud : 115200
Parity : None
Bits : 8

Duplex : Full
Char Delay (ms) : 5
EOL Delay (ms) : 15

Copyright 2007, P&E Microcomputer Systems, All Rights Reserved. http://www.pemicro.com

The parameters across the top allow configuration of the connection to the device and the programs within this guide have been written to match the default settings. If the Demo 1 is loaded onto the device, the terminal is connected (using the "Open Serial Port" button) and the part is reset, the user is presented with a command prompt.

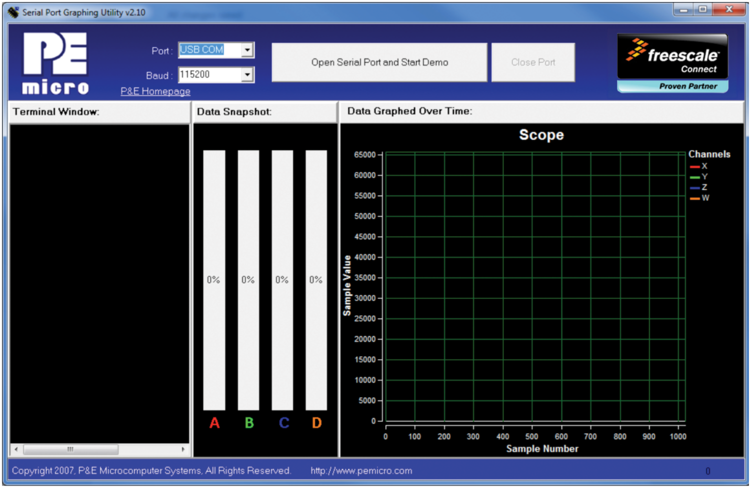


```
Freescale MPC5604 Console

MPC5604B:> help
Help system

Use "help <command>" for more information.

Further information is available for:

  * echo
  * pinout
  * adc
  * pwm
  * digin
  * digout
  * pinspec

MPC5604B:>
```

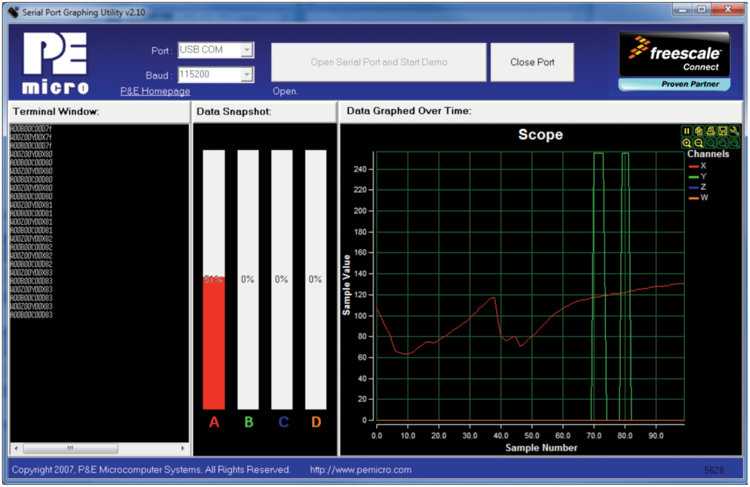
Further information on the commands available is documented within the Demo 1 example.

The Serial Grapher Utility

The serial grapher utility presents the user with a terminal window as well as two bar indicators and a charting tool. Demonstration 2 outputs data over the virtual serial port for display in both the bars and the chart area. The raw data sent is shown in the terminal window. The interface for this tool is shown below.



When Demo 2 is loaded, and the grapher utility is connected, the data from the user switches and the temperature sensor are shown as the last sample on the bar chart and the history on the line chart. Note that only one utility can connect to the device at any given time. You cannot connect the grapher and terminal applications simultaneously, nor can you have multiple instances running.



```
A00B00C00D7f
W00Z00Y00X7f
A00B00C00D7f
W00Z00Y00X80
A00B00C00D80
W00Z00Y00X80
A00B00C00D80
W00Z00Y00X80
A00B00C00D80
W00Z00Y00X81
A00B00C00D81
W00Z00Y00X81
A00B00C00D81
W00Z00Y00X82
A00B00C00D82
W00Z00Y00X82
A00B00C00D82
W00Z00Y00X83
A00B00C00D83
W00Z00Y00X83
A00B00C00D83
W00Z00Y00X83
A00B00C00D83
```

Support

Visit **freescale.com/support** for a list of phone numbers within your region.

Warranty

Visit **freescale.com/warranty** for complete warranty information.

For more information, visit
freescale.com/TRK-USB-MPC5604B

Document Number: MPC5604BLABGD REV 0