MY-BASIC Customizations
for
MuntsOS Embedded Linux

7 February 2019

by Philip Munts
President, Munts Technologies
http://tech.munts.com

Introduction

This document describes the customizations made to **MY-BASIC** for the MuntsOS Extension Package. These customizations are compiled in; no action is required to enable them.

The extension package installs the **MY-BASIC** interpreter at **`/usr/local/bin/basic`**.

To run the interpreter in interactive mode, enter the following at a MuntsOS command line:

```
basic
```

You can also provide the name of a BASIC source program text file to load and run:

```
basic hello.bas
```

Sample programs are available at: http://git.munts.com/libsimpleio/basic/programs

References

https://github.com/paladin-t/my_basic

http://git.munts.com/libsimpleio

Operating System Services

delay(usecs)

This service pauses program execution for the specified number of microseconds.

Example Program

```
while true
  print "Tick";
  delay(1000000)
  print "Tock";
  delay(1000000)
wend
```

Analog to Digital Converter Services

These services allow reading from Linux kernel ADC input pins using **libsimpleio**.

fd = libsimpleio.adc_open(chip, channel)

This service opens an ADC input pin.

The **`chip`** and **`channel`** parameters select the ADC input pin.

This service returns a Linux file descriptor number that will be used as a handle for all of the other ADC services.

libsimpleio.adc_close(fd)

This service closes an ADC input pin.

The **`fd`** parameter must be a file descriptor number previously returned by **`libsimpleio.adc_open()`**.

sample = libsimpleio.adc_read(fd)

This service reads a single integer sampled data value from an ADC input pin.

The **`fd`** parameter must be a file descriptor number previously returned by **`libsimpleio.adc_open()`**.

This service returns the integer sampled data value.

ADC Example Program

```
fd = libsimpleio.adc_open(0, 0)

while true
  print "Sample: "
  print libsimpleio.adc_read(fd);
  delay(1000000)
wend
```

Digital to Analog Converter Services

These services allow writing to Linux kernel DAC output pins using **libsimpleio**.

fd = libsimpleio.dac_open(chip, channel)

This service opens a DAC output pin.

The **`chip`** and **`channel`** parameters select the DAC output pin.

This service returns a Linux file descriptor number that will be used as a handle for all of the other DAC services.

libsimpleio.dac_close(fd)

This service closes a DAC output pin.

The **`fd`** parameter must be a file descriptor number previously returned by **`libsimpleio.dac_open()`**.

libsimpleio.dac_write(fd, sample)

This service writes a single integer sampled data value to a DAC output pin.

The **`fd`** parameter must be a file descriptor number previously returned by **`libsimpleio.adc_open()`**.

The **`sample`** parameter must be an integer value within the acceptable range for the particular DAC hardware (usually **`0`** to $2^{Resolution}-1$). An ordinary 12-bit DAC with **single-ended** outputs will usually have an acceptable range of **`0`** to $2^{12}-1$ or **`0`** to **`4095`** while an exotic 12-bit DAC with **true differential** outputs might have an acceptable range of -2047 to 2047.

DAC Example Program

```
fd = libsimpleio.dac_open(0, 0)

while true
  for n = 0 to 4095
    dac_write(fd, n)
  next n
wend
```

General Purpose Input/Output Services

These services allow manipulating Linux kernel GPIO pins using **libsimpleio**.

fd = libsimpleio.gpio_open(chip, channel, dir, state)

This service opens a GPIO pin.

The **`chip`** and **`channel`** parameters select the GPIO pin. The **`dir`** parameter selects the data direction (**`0`**=input, **`1`**=output). The **`state`** parameter selects the initial state for an output pin (**`0`**=off or low, **`1`**=on or high).

This service returns a Linux file descriptor number that will be used as a handle for all of the other GPIO services.

libsimpleio.gpio_close(fd)

This service closes a GPIO pin.

The **`fd`** parameter must be a file descriptor number previously returned by **`libsimpleio.gpio_open()`**.

state = libsimpleio.gpio_read(fd)

This service reads from a GPIO pin.

The **`fd`** parameter must be a file descriptor number previously returned by **`libsimpleio.gpio_open()`**.

This service returns the state of the GPIO pin (**`0`**=off or low, **`1`**=on or high)

libsimpleio.gpio_write(fd, state)

This services writes to a GPIO pin.

The **`fd`** parameter must be a file descriptor number previously returned by **`libsimpleio.gpio_open()`**.

The **`state`** parameter indicates the value written to the GPIO pin (**`0`**=off or low, **`1`**=on or high).

GPIO Example Program

```
fd = libsimpleio.gpio_open(0, 26, 1, 0)

while true
  libsimpleio.gpio_write(fd, NOT libsimpleio.gpio_read(fd))
wend
```

Pulse Width Modulated Output Services

These services allow controlling Linux kernel PWM output pins using **libsimpleio**.

fd = libsimpleio.pwm_open(chip, channel, period, ontime)

The **`chip`** and **`channel`** parameters select the PWM output pin.

The **`period`** parameter sets the PWM pulse period in nanoseconds.

The **`ontime`** parameter sets the initial PWM pulse width in nanoseconds.

libsimpleio.pwm_close(fd)

This service closes a PWM output pin.

The **`fd`** parameter must be a file descriptor number previously returned by **`libsimpleio.pwm_open()`**.

libsimpleio.pwm_write(fd, ontime)

This services writes to a PWM output pin.

The **`fd`** parameter must be a file descriptor number previously returned by **`libsimpleio.pwm_open()`**.

The **`ontime`** sets the PWM pulse width in nanoseconds.

PWM Example Program

```
fd = libsimpleio.pwm_open(0, 0, 10000000, 0)

while true
  for ontime = 0 to 10000000 step 10000
    libsimpleio.pwm_write(fd, ontime)
    delay(5000)
  next ontime
wend
```