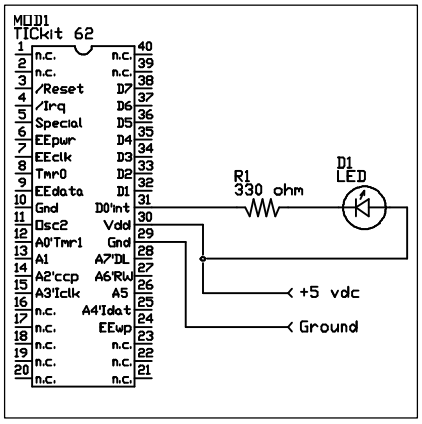
Blinking an LED on the TICkit 62(Simple I/O Example)

Submitted by:

Glenn Clark - Protean Logic Inc.

The programs and drawings shown below are taken from the examples section of the manual. This note serves as a good introduction to I/O on a TICkit controller.

After you get your TICkit up and running the "Hello World..." program, a good second program is a simple program to blink an LED. This assures that you understand basic I/O and how to connect devices electrically to the TICkit. In this example a general purpose I/O pin drives an LED via a current limiting resistor, R1. The output is wired to be low active, which means the LED is lit by outputting a ground level. It is desirable to drive higher current devices at ground level because the internal nature of the TICkit processor can drive higher currents from ground than from +5 vdc. The circuit is shown below.



This circuit is very simple. When your program instructs, the TICkit processor will turn on an internal switch that connects the pin labeled D0 to the ground. This completes a circuit in which current flows from the +5 vdc power supply input, through the forward biased LED, through the 330 ohm current limiting resistor and then through the TICkit processor to the ground of the power supply. No other pin of the TICkit module need to be connected. For the +5 vdc input you can use any regulated power supply. Many people have access to a 5 volt supply. If not, you can use one of the circuits shown in the next section as a power supply. Any of the general purpose outputs can be used for this program (pins labeled D0 through D7 or A0 through A7). They all function in the same way. When writing your programs refer to the pins through their corresponding symbolic names. For example the pin labeled D0 is symbolically referred to as "pin_d0" within a program just as the pin labeled A5 is called "pin_a5". The symbols for the pins are actually numeric constants that evaluate to a number between 0 and 15. Pin D0 is pin number 0, pin D1 is number 1 etc. and pin A0 is pin number 8, pin A1 is pin number 9 and so on. It is usually preferable to refer to constants by symbolic name as it makes the program easier to understand and allows easier modifications later on. Numbers or variables can be used in the pin_high() or pin_low() functions when your application can benefit from a pin reference that is variable. The program for blinking the LED is equally simple. Most of the program is the required FBasic verbiage to inform the compiler of the version of the TICkit and where to start the program. Before showing the final LED blinking program, examine the program below to simply turn on the LED.

```
DEF tic62_c

LIB fbasic.lib
```

```
FUNC none main
BEGIN
    pin_low( pin_d0 )       ; this is the same as pin_low( 0 )
    REP
        debug_on()
    LOOP
ENDFUN
```

The first function this program executes is the pin_low(pin_d0) line. This function makes the specified pin an output and switches it to ground. Once this line executes, the LED is on. The lines at the end of the program are there because of the nature of a controller. The TICkit is a controller computer. This means it presumably controls something. In the above program, the last three lines make the program continually ask to connect to the console. If these lines were not there, the TICkit would have no idea what to do when it finished the function, so it would execute random garbage contained in it's EEprom. This could reset the processor or do virtually anything. By putting the loop at the end of our program, we can be sure that the processor is occupied in the loop and the LED stays on for us to observe. Most control programs are just big loops. They execute the same basic task over and over their entire life. As you write more programs you will see this tendency emerge. Okay, lets make the light blink. This next program does indeed blink the light, but does not give satisfactory results, see if you can discover why:

```
DEF tic62_c

LIB fbasic.lib

FUNC none main
BEGIN
    REP
        pin_low( pin_d0 )
        pin_high( pin_d0 )
    LOOP
ENDFUN
```

Did you figure it out? The pin_d0 will indeed turn the LED on and off, but at so fast of a rate that it appears to be on constantly. This effect is useful for multiplexing, but not for blinking some lights. The correct program needs some delay for both the on state and the off state. If you have more delay in the off state than the on state, the LED will appear dimmer. If you have more delay in the on state than the off state, the LED appears brighter. This is an important concept called pulse width modulation (PWM) that we will discuss in detail later on. The correct program for a 1 second blink rate is as follows:

```
DEF tic62_c

LIB fbasic.lib

FUNC none main
BEGIN
    REP
        pin_low( pin_d0 )  ; turn LED on
        delay( 500 )       ; leave LED on for 500/1000 of a sec.
        pin_high( pin_d0 ) ; turn LED off
        delay( 500 )       ; leave LED off for 500/1000 second.
    LOOP
ENDFUN
```

The delay() function halts the processor for the specified number of milliseconds (1/1000 second). The delay function expects to see a number between 0 and 65535 (the range for a 16 bit word). Feel free to modify this program. Control more LED's, or maybe increase the blink rate by lowering the delays. If the blink rate is less than about 1/30 of a second, the LED appears to be on constantly. At this rate, you can alter the relative on and off delays to observe the effects of PWM. The following code produces a continually glowing LED at about 1/2 brightness.

```
DEF tic62_c

LIB fbasic.lib
```

```
FUNC none main
BEGIN
    REP
        pin_low( pin_d0 )  ; turn LED on
        delay( 15 )        ; leave LED on for 15/1000 of a sec.
        pin_high( pin_d0 ) ; turn LED off
        delay( 15 )        ; leave LED off for 15/1000 second.
    LOOP
ENDFUN
```

There is another way to turn off the output of a pin besides changing the level of its output. You could use the pin_in() function as shown below.

```
DEF tic62_c

LIB fbasic.lib

GLOBAL byte trash ; an 8 bit variable used below

FUNC none main
BEGIN
    REP
        pin_low( pin_d0 )      ; turn LED on
        delay( 500 )           ; leave LED on for 500/1000 of a sec.
        =( trash, pin_in( pin_d0 )) ; turn LED off
        delay( 500 )           ; leave LED off for 500/1000 second.
    LOOP
ENDFUN
```

The pin_in function makes the specified pin an input and reads the level on the pin. A 0 is returned if the level is low (<2.5 volts) or 255 if the level is high (>2.5 volts). In our example, we do not care what the level is on the pin, we just want to turn off the output and make the pin an input. The returned value must be assigned to something though, or the compiler will generate an error because it knows the pin_in function returns a number and expects the program to use that value. Examples of the pin_in() function are used extensively in later examples to read button presses.