APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

# ViSi Pin Counter and Pulse Out

| | |
|---|---|
| DOCUMENT DATE: | **13th April 2019** |
| DOCUMENT REVISION: | **1.1** |

## Description

This Application note is intended to demonstrating to the user the set-up, initialization and operation of the built-in pin counter and pulse output feature of the Diablo16 display module.

- The target screen can be any of the following Diablo16 touch display modules:

| | | |
|---|---|---|
| gen4-uLCD-24D Series | gen4-uLCD-28D Series | gen4-uLCD-32D Series |
| gen4-uLCD-35D Series | gen4-uLCD-43D Series | gen4-uLCD-50D Series |
| gen4-uLCD-70D Series | | |
| uLCD-35DT | uLCD-43D Series | uLCD-70DT |

Visit www.4dsystems.com.au/products to see the latest display module products that use the Diablo16 processor.

- 4D Programming Cable / µUSB-PA5/µUSB-PA5-II for non-gen4 displays (uLCD-xxx)
- 4D Programming Cable & gen4-IB / gen4-PA / 4D-UPA, for gen-4 displays (gen4-uLCD-xxx)
- micro-SD (µSD) memory card
- Workshop 4 IDE (installed according to the installation document)

- When downloading an application note, a list of recommended application notes is shown. It is assumed that the user has read or has a working knowledge of the topics presented in these recommended application notes.

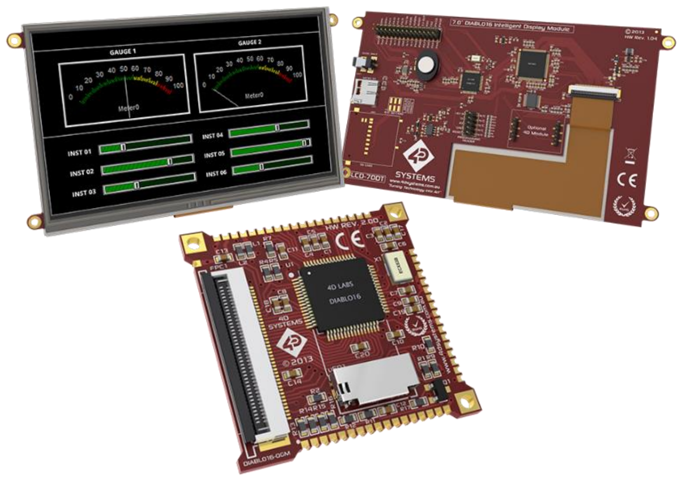
## Content

## Application Overview

This document is focused on the fundamental usage of the built-in pin counter and pin output pulse feature of the DIABLO16 Embedded Graphics Processor. A pulse is a simple transition from a logic 0 to a logic 1 or vice versa. The DIABLO16 pin counter feature enables detection of the transition in either ways. Diablo16 OGM and Diablo16 display module has a total of 6 pin counter channels. These pin counters are supported in 6 particular GPIO terminals.

Also included in this application is the pulse out feature of the DIABLO16 embedded graphics processor. The pulse out is a square wave signal generator that can has a variable duty cycle. The DIABLO16 has a straight forward function to generate this pulse which can be used to momentarily turn on and off an isolated output. The pulse out feature of the DIABLO16 is a 'one-shot' square-wave signal.

## The DIABLO16 Embedded Graphics Processor



Driving the display and peripherals is the DIABLO16 embedded graphics processor, a very capable and powerful chip which enables stand-alone functionality, programmed using the 4D Systems Workshop 4 IDE Software. The Workshop IDE enables graphic solutions to be constructed rapidly and with ease due to its design being solely for 4D's graphics processors.

The DIABLO16 Processor offers considerable FLASH and RAM upgrades over the PICASO processor, and also provides map-able functions such as I2C, SPI, Serial, PWM, Pulse Out, and Quadrature Input, to various GPIO, and also provide up to 4 Analogue Input channels.

## Setup Procedure

For instructions on how to launch Workshop 4, how to open a **ViSi** project, and how to change the target display, kindly refer to the section "**Setup Procedure**" of the application note

**ViSi Getting Started - First Project for Picaso and Diablo16**

## Create a New Project

For instructions on how to create a new **ViSi** project, please refer to the section "**Create a New Project**" of the application note

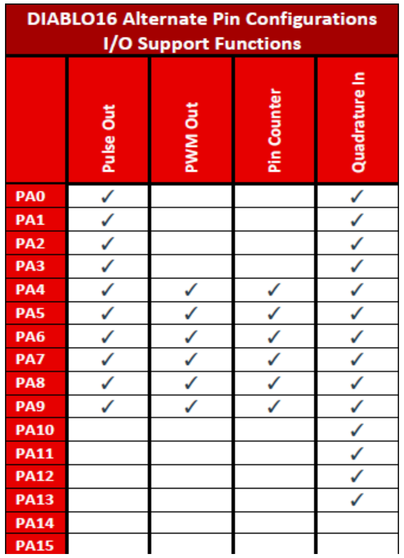
**ViSi Getting Started - First Project for Picaso and Diablo16**

## Design the Project

To create a simple program that will be able activate and initialize the DIABLO16 pulse input and pulse output, we will need to use some commands enlisted in the DIABLO 4DGL Internal Functions.

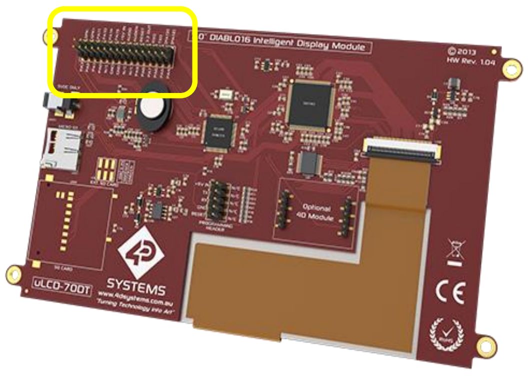
The DIABLO16 embedded graphics processor has a total of 6 and 10 configurable GPIO for the pin counter and pin pulse out feature, respectively. Referring to the chart below, we will see that these output pins can be used alternately with other processor I/O function support.

These general purpose I/O pins for the PWM output does not follow any arrangements. The pins may be used readily, after configuring the function setup, according to the user's needs. These I/O pins are located similar to the image.

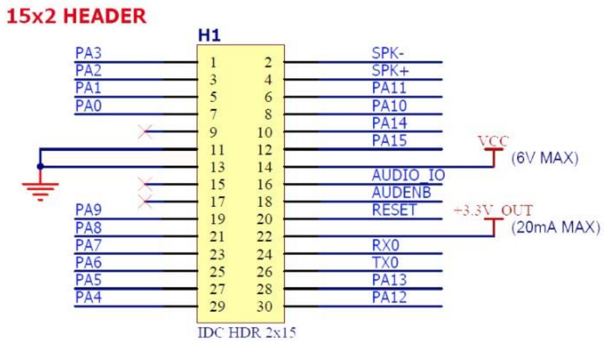


**DIABLO16 Alternate Pin Configurations I/O Support Functions**

| | Pulse Out | PWM Out | Pin Counter | Quadrature In |
|---|---|---|---|---|
| PA0 | ✓ | | | ✓ |
| PA1 | ✓ | | | ✓ |
| PA2 | ✓ | | | ✓ |
| PA3 | ✓ | | | ✓ |
| PA4 | ✓ | ✓ | ✓ | ✓ |
| PA5 | ✓ | ✓ | ✓ | ✓ |
| PA6 | ✓ | ✓ | ✓ | ✓ |
| PA7 | ✓ | ✓ | ✓ | ✓ |
| PA8 | ✓ | ✓ | ✓ | ✓ |
| PA9 | ✓ | ✓ | ✓ | ✓ |
| PA10 | | | | ✓ |
| PA11 | | | | ✓ |
| PA12 | | | | ✓ |
| PA13 | | | | ✓ |
| PA14 | | | | |
| PA15 | | | | |



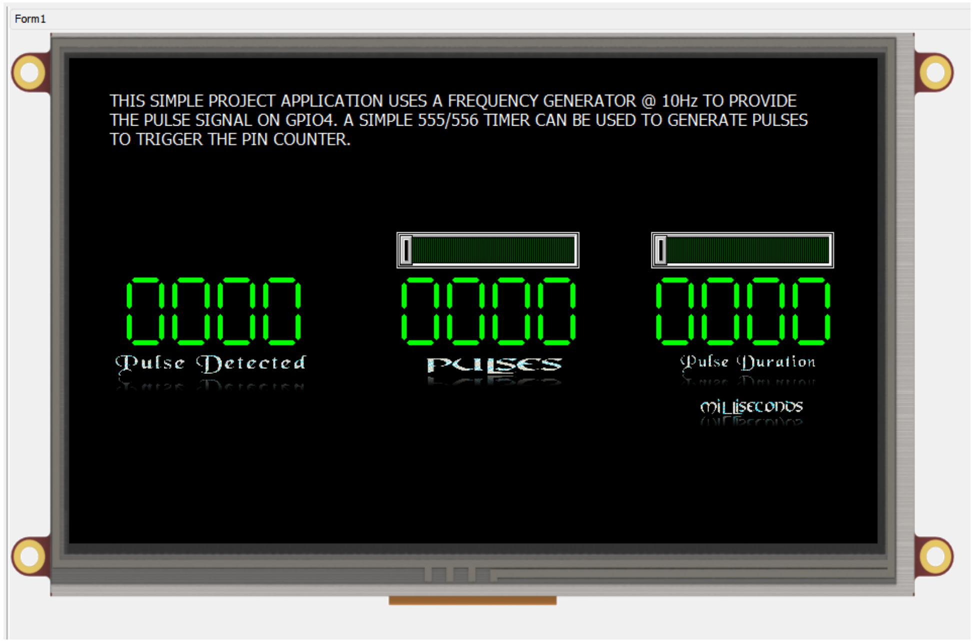
The DIABLO16 configurable I/O are a group of 3.3 volts TTL level terminals but these are tolerant to a maximum of 5 volts. Anything greater than or less than the specified operating voltage may prohibit proper communication or even damage the embedded graphics processor.



The 2x15 male header pin assignment of the DIABLO16 70DT. The previous table on pin function configuration option are lumped together with several other specific purpose pins.

## The ViSi - based application project

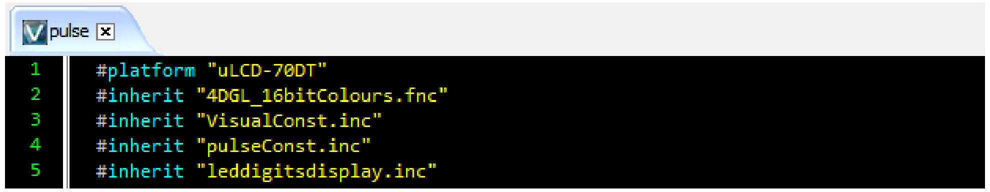
For this application project a slider and three LED digits will be needed. Add objects by navigating to the Layout the objects similar to the one below.


Form1
THIS SIMPLE PROJECT APPLICATION USES A FREQUENCY GENERATOR @ 10Hz TO PROVIDE THE PULSE SIGNAL ON GPIO4. A SIMPLE 555/556 TIMER CAN BE USED TO GENERATE PULSES TO TRIGGER THE PIN COUNTER.
0000
Pulse Detected
0000
PULSES
0000
Pulse Duration
milliseconds

After all the objects have been laid-out, let's continue with the other half which involves the coding of the project. This will be presented in a sectional manner so as not to create confusion with the project. For an in-depth detail of the functions used in this application note please refer to the DIABLO16 Internal Functions Reference Manual.

## The Include Section

This project starts with the identification of the platform being used as declared by the #platform function. For the program to be able to function properly files are included herein using the #inherit function.



```
#platform "uLCD-70DT"
#inherit "4DGL_16bitColours.fnc"
#inherit "VisualConst.inc"
#inherit "pulseConst.inc"
#inherit "leddigitsdisplay.inc"
```

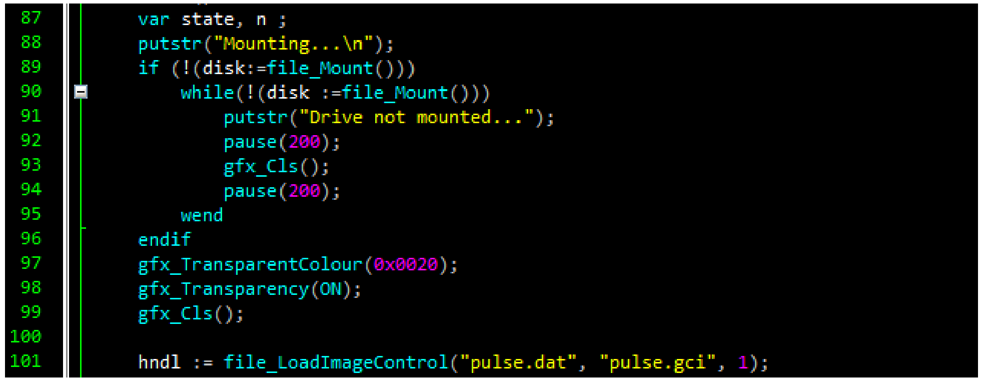
In this application note, pulseConst.inc, contains all the information about the objects that are used in the project. Meanwhile, the leddigitsdisplay.inc contains the function for the proper operation of the led digits objects.

## The main program

The main program for this project contains several sections: the mounting of the micro-SD card, the initial displaying and image touch setup for objects, the repeat-forever loops which contains the pulse counter detection condition and touch conditions. Also, the main program calls out sub-routine functions that perform a particular function.

## The micro-SD initialization

Let's start with the initialization of the uSD card. The uSD card contains all the image information about the objects used in the project. The object information and data are saved under a *.DAT and a *.GCI filename extension which is copied to the uSD during project compilation. Mounting of the disk in this application note was done using the following set of program statements.

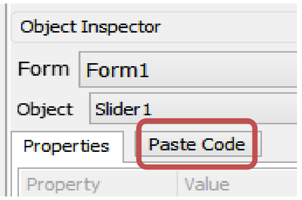


```
    var state, n ;
    putstr("Mounting...\n");
    if (!(disk:=file_Mount()))
        while(!(disk :=file_Mount()))
            putstr("Drive not mounted...");
            pause(200);
            gfx_Cls();
            pause(200);
        wend
    endif
    gfx_TransparentColour(0x0020);
    gfx_Transparency(ON);
    gfx_Cls();

    hndl := file_LoadImageControl("pulse.dat", "pulse.gci", 1);
```

When starting a new project in the ViSi environment these set of statements are already included in the coding area. The last part of this set of statements uses a function file_LoadImageControl() to call on the object data/information files on the uSD drive. This initializes the data to be called in using the variable 'hndl'.

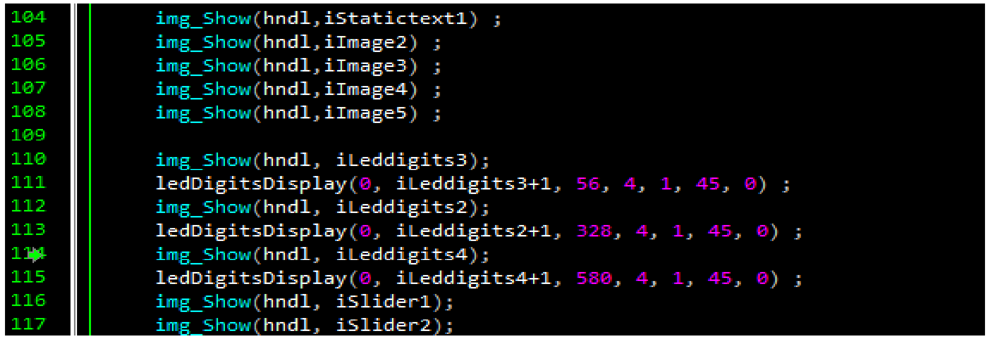
Having been able to load and initialize the uSD drive, the processor is now able to access the information stored therein. As mentioned from the previous section, the filenames with an extension of DAT and GCI has the image data and information.

Therefore, the next part of the main program is to display all the objects that were placed on the Workshop IDE form viewer. To do so, a special button from the Object Inspector can help reduce the time of coding of this part. The 'Paste Code" simply pastes object code into the coding area.



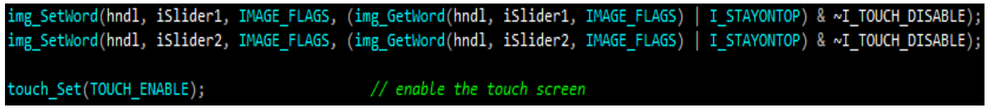
## The initial image display and image touch setup segment

In this part of the program, the img_Show() function calls out the object image and information found in the microSD drive. This set of statements displays every object that were included in the application project. The displaying of the images is directly done using the img_Show() function.



```
        img_Show(hndl,iStatictext1) ;
        img_Show(hndl,iImage2) ;
        img_Show(hndl,iImage3) ;
        img_Show(hndl,iImage4) ;
        img_Show(hndl,iImage5) ;

        img_Show(hndl, iLeddigits3);
        ledDigitsDisplay(0, iLeddigits3+1, 56, 4, 1, 45, 0) ;
        img_Show(hndl, iLeddigits2);
        ledDigitsDisplay(0, iLeddigits2+1, 328, 4, 1, 45, 0) ;
        img_Show(hndl, iLeddigits4);
        ledDigitsDisplay(0, iLeddigits4+1, 580, 4, 1, 45, 0) ;
        img_Show(hndl, iSlider1);
        img_Show(hndl, iSlider2);
```

The statements in this segment of the program displays all the images and slider. We have displayed all the slider and LED digit images that were used to provide the value for the width of the modulated pulse. Moving to the next part of the main program, this segment is all related to the image touch detection setup.

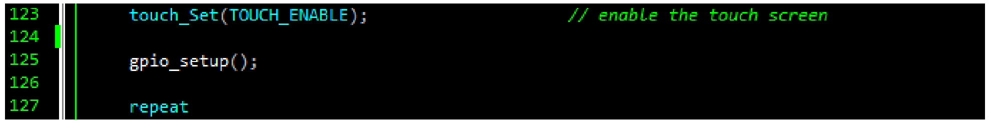


```
img_SetWord(hndl, iSlider1, IMAGE_FLAGS, (img_GetWord(hndl, iSlider1, IMAGE_FLAGS) | I_STAYONTOP) & ~I_TOUCH_DISABLE);
img_SetWord(hndl, iSlider2, IMAGE_FLAGS, (img_GetWord(hndl, iSlider2, IMAGE_FLAGS) | I_STAYONTOP) & ~I_TOUCH_DISABLE);

touch_Set(TOUCH_ENABLE);                    // enable the touch screen
```

This statements uses the img_SetWord() function. The primary objective of this set of statement is to enable the touch detection for the slider image. The slider images on this project serves as input objects which utilizes the touch feature of the device. At the end, of this segment we would notice that the touch feature of the device was enabled using the touch_Set(TOUCH_ENABLE) statement.

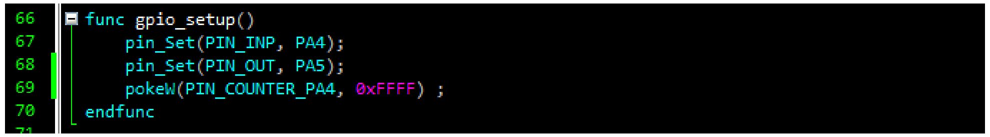
## The GPIO setup sub-routine

The purpose of this sub-routine is to simplify the presentation of the program statements in this document. When using the GPIO pins for a special function, it is always best that the direction of data is assigned.



```
    touch_Set(TOUCH_ENABLE);                    // enable the touch screen

    gpio_setup();

    repeat
```

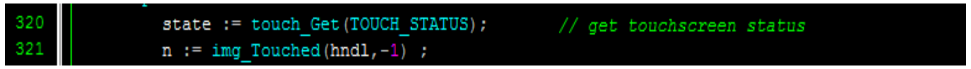
Below is the sub-routine being called in by the gpio_setup() function. It can be observed that the GPIO PA4 is being set as an input pin while the PA5 is set as output. The pins are assigned respectively using the pin_Set() function. The pokeW() function at the end of this routine is used to trigger the pin counter of PA4 to count single transitions as denoted by the 0xFFFF to 0x0000 rollover transition.



```
func gpio_setup()
    pin_Set(PIN_INP, PA4);
    pin_Set(PIN_OUT, PA5);
    pokeW(PIN_COUNTER_PA4, 0xFFFF) ;
endfunc
```

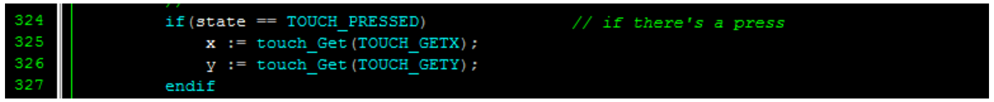
## The repeat-forever image touch detect loop

At this end part of the main program, the routine was to detect any activity on the touch screen. Please refer to the image on the next page. Three touch states were included in the repetitive routine: the detection for a pressed state, a released state, and a moving state. Prior to the touch detection, a variable ‘n’ is assigned to store temporary image touch detection result. The img_Touched() function checks the object being touched and return the name of the object enlisted in the variable ‘hndl’. Deploy



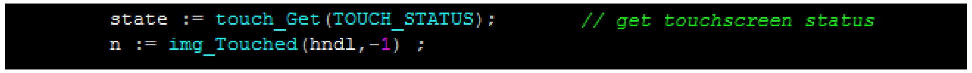
```
    state := touch_Get(TOUCH_STATUS);          // get touchscreen status
    n := img_Touched(hndl,-1) ;
```

Moving to the touch detection routines, when a touch status of ‘pressed’ is detected the value of the coordinates are saved on the variables x and y.



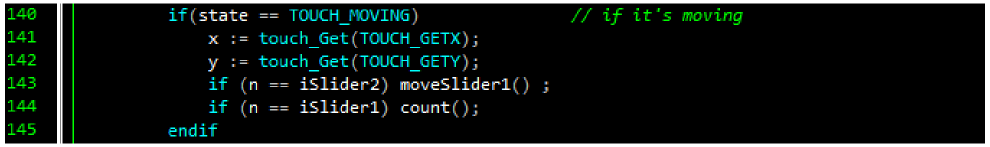
```
        if(state == TOUCH_PRESSED)               // if there's a press
            x := touch_Get(TOUCH_GETX);
            y := touch_Get(TOUCH_GETY);
        endif
```

The most significant segment of this routine is the moving touch state, it is in this conditional loop that the image touch detection is made use. If a touch was detected over the slider image, a sub-routine or a function is called upon and executed by the processor.



```
    state := touch_Get(TOUCH_STATUS);          // get touchscreen status
    n := img_Touched(hndl,-1) ;
```

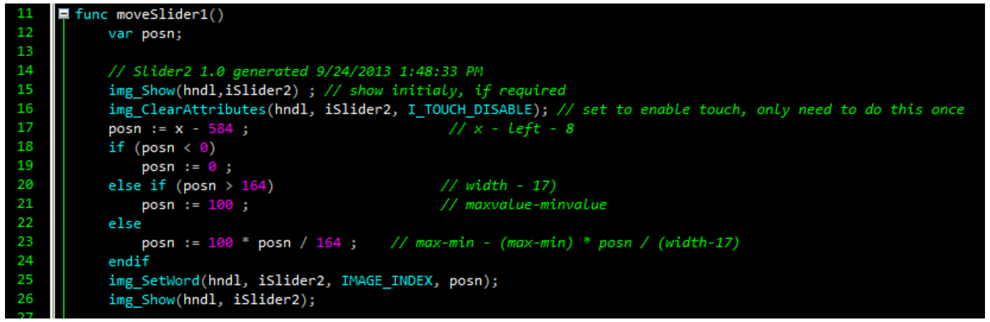
Let us take the above statement as an example. From the start of the repeat-forever loop, the img_Touched() function saves the result of an image touch to a variable ‘n’, this is then checked in the touch moving conditional statements. If it proves to be equal to one of the conditions then the sub-routine will be executed. For this statement a moveSlider1() sub-routine is being called and executed. Referring to the image following this section, we -can see that if the touched image is equivalent to one of the if-conditions then a particular sub-routine is called in.



```
        if(state == TOUCH_MOVING)                // if it's moving
            x := touch_Get(TOUCH_GETX);
            y := touch_Get(TOUCH_GETY);
            if (n == iSlider2) moveSlider1() ;
            if (n == iSlider1) count();
        endif
```

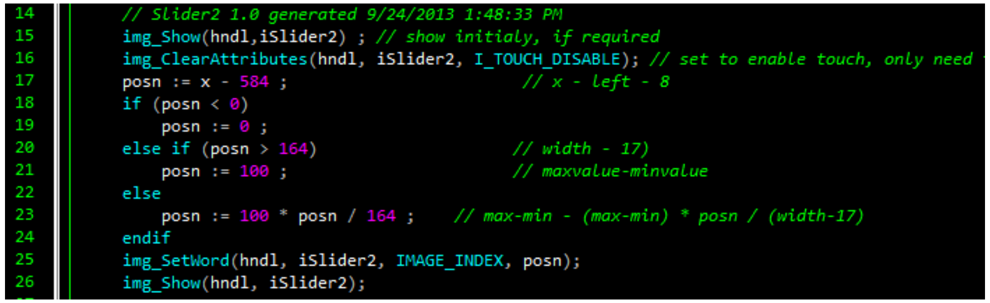
## The moveSlider1() sub-routine

Whenever the touch detection results to the slider1 image being touched, the processor is directed to run the statements contained in the sub-routine.



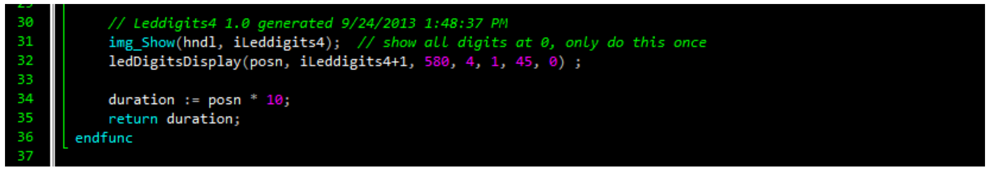
```
func moveSlider1()
    var posn;

    // Slider2 1.0 generated 9/24/2013 1:48:33 PM
    img_Show(hndl,iSlider2) ; // show initialy, if required
    img_ClearAttributes(hndl, iSlider2, I_TOUCH_DISABLE); // set to enable touch, only need to do this once
    posn := x - 584 ;                         // x - left - 8
    if (posn < 0)
        posn := 0 ;
    else if (posn > 164)                      // width - 17)
        posn := 100 ;                         // maxvalue-minvalue
    else
        posn := 100 * posn / 164 ;     // max-min - (max-min) * posn / (width-17)
    endif
    img_SetWord(hndl, iSlider2, IMAGE_INDEX, posn);
    img_Show(hndl, iSlider2);
```

The first part of the sub-routine includes the slider codes that are automatically generated using the 'Paste Code' tab on the object inspector window. This is also true with the LED digits objects. The paste code function is a very good tool to minimize the coding of the objects.



```
    // Slider2 1.0 generated 9/24/2013 1:48:33 PM
    img_Show(hndl,iSlider2) ; // show initialy, if required
    img_ClearAttributes(hndl, iSlider2, I_TOUCH_DISABLE); // set to enable touch, only need
    posn := x - 584 ;                         // x - left - 8
    if (posn < 0)
        posn := 0 ;
    else if (posn > 164)                      // width - 17)
        posn := 100 ;                         // maxvalue-minvalue
    else
        posn := 100 * posn / 164 ;     // max-min - (max-min) * posn / (width-17)
    endif
    img_SetWord(hndl, iSlider2, IMAGE_INDEX, posn);
    img_Show(hndl, iSlider2);
```

From the previous set of statements the resulting mathematical value of the slider position is stored in the variable 'posn'. In turn, this value is displayed using the Leddigits4 object. The frame index value is pointed by any change in the value of 'posn'.

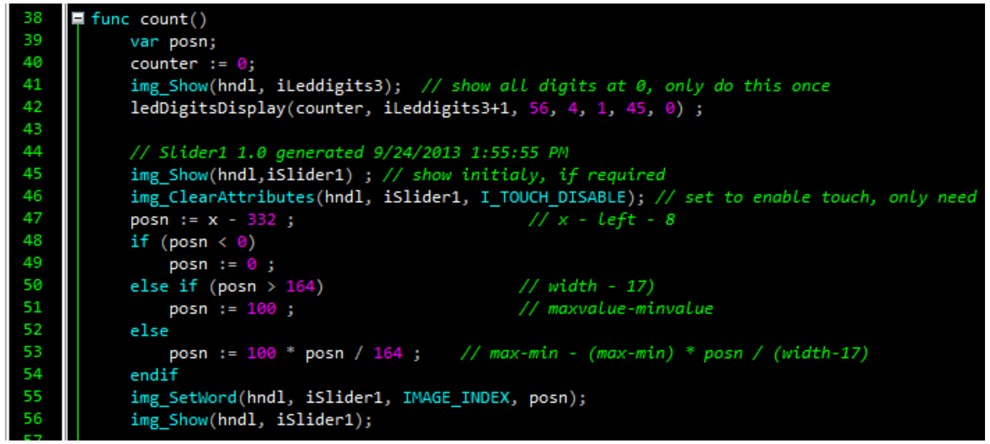


```
    // Leddigits4 1.0 generated 9/24/2013 1:48:37 PM
    img_Show(hndl, iLeddigits4);  // show all digits at 0, only do this once
    ledDigitsDisplay(posn, iLeddigits4+1, 580, 4, 1, 45, 0) ;

    duration := posn * 10;
    return duration;
endfunc
```

At the end of the subroutine, a processed value is returned. The returned value is used as the time-base for the pulse out square wave signal width. The result of the change in the returned value will be seen in the succeeding sections of this documentation.

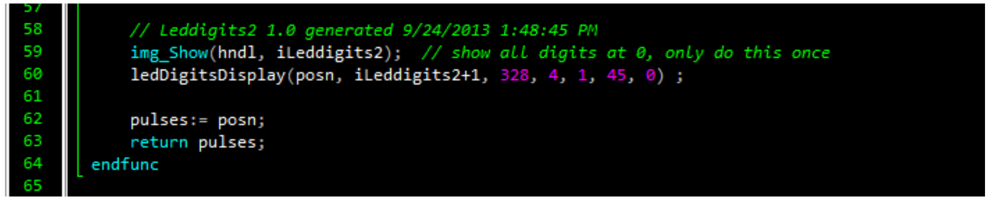
## The count() sub-routine

Whenever the result of the image touch results to the slider1 being pressed or moved the processor is directed to execute the count() subroutine. This sub-routine contains the 'Paste Code' generated statements for the slider1 and Led digits to display.



```
func count()
    var posn;
    counter := 0;
    img_Show(hndl, iLeddigits3);  // show all digits at 0, only do this once
    ledDigitsDisplay(counter, iLeddigits3+1, 56, 4, 1, 45, 0) ;

    // Slider1 1.0 generated 9/24/2013 1:55:55 PM
    img_Show(hndl,iSlider1) ; // show initialy, if required
    img_ClearAttributes(hndl, iSlider1, I_TOUCH_DISABLE); // set to enable touch, only need
    posn := x - 332 ;                         // x - left - 8
    if (posn < 0)
        posn := 0 ;
    else if (posn > 164)                      // width - 17)
        posn := 100 ;                         // maxvalue-minvalue
    else
        posn := 100 * posn / 164 ;     // max-min - (max-min) * posn / (width-17)
    endif
    img_SetWord(hndl, iSlider1, IMAGE_INDEX, posn);
    img_Show(hndl, iSlider1);
```

The resulting value is saved on the local variable 'posn'. This value is again used in the Led digits statement to point the frame index and display the current value.



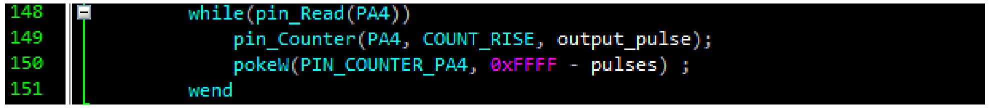
```
    // Leddigits2 1.0 generated 9/24/2013 1:48:45 PM
    img_Show(hndl, iLeddigits2);  // show all digits at 0, only do this once
    ledDigitsDisplay(posn, iLeddigits2+1, 328, 4, 1, 45, 0) ;

    pulses:= posn;
    return pulses;
endfunc
```

Notice at the end of the count() sub-routine that a value pulse is returned. This value will be used to point to the pin counter limit. Whenever the pin count reaches this value then the pulse out will be triggered.

The use of the values provided by the count() and the moveSlider() subroutines will be shown in the succeeding section of this document.

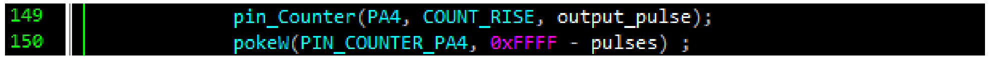
## The pin counter detect condition

The detection of the transition for the pulse input to the GPIO pin PA5 can be set to follow a rising edge or a falling or both.



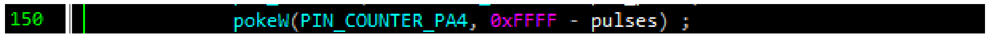
```
        while(pin_Read(PA4))
            pin_Counter(PA4, COUNT_RISE, output_pulse);
            pokeW(PIN_COUNTER_PA4, 0xFFFF - pulses) ;
        wend
```

For this application, the input pulse detection set was set for a low logic to high logic transition using the argument COUNT_RISE. Referring to the statements shown above, the while(pin_read()) detects the status of the GPIO PA4. Any transition detected will execute the statements enclosed in the while-wend condition loop.



```
            pin_Counter(PA4, COUNT_RISE, output_pulse);
            pokeW(PIN_COUNTER_PA4, 0xFFFF - pulses) ;
```

The pin_Counter() function directs the processor to assign the PA4 GPIO to detect all rising edge transitions up to the number of pulses set using the pokeW() function from the setup_gpio() during the initial run.



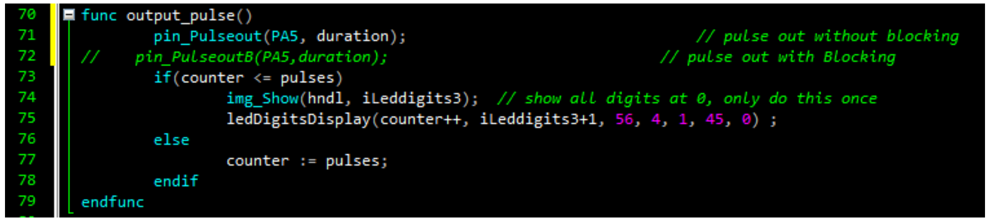
```
            pokeW(PIN_COUNTER_PA4, 0xFFFF - pulses) ;
```

The pokeW() function is used to write values to the PIN_REGISTER_PA4 a value between 0xFFFF and 0x0000. The number of pulses subtracted from the 0xFFFF gives the number for the counter before rolling over to zero. The PIN_COUNTER_PAx register needs to be re-armed each time after the pin counter rolls over to zero. Each time a transition is detected on the GPIO the counter is increased moving to the value 0xFFFF.

## The pulse_out() sub-routine

Each time that a logic transition is detected on the PA4, the output_pulse() sub-routine is called and executed. Each time this routine is called the pulse detected counter LED digits is increased.

From the previous sections of this documents, recall that the slider-related sub-routines return a value each time there are changes in the slider values. These values which are returned using the global variables are utilized in the output_pulse() sub-routine. The 'duration' give the width of the square wave pulse signal generated on GPIO PA5. Likewise, the limit of the pulse detection is limited to the value given by the count() sub-routine.

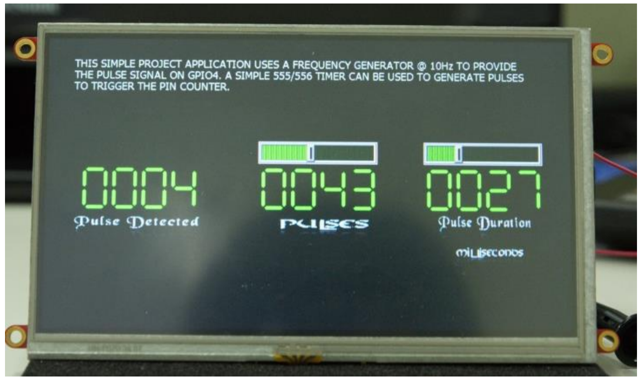


```
func output_pulse()
        pin_Pulseout(PA5, duration);                                // pulse out without blocking
//    pin_PulseoutB(PA5,duration);                               // pulse out with Blocking
        if(counter <= pulses)
                img_Show(hndl, iLeddigits3);  // show all digits at 0, only do this once
                ledDigitsDisplay(counter++, iLeddigits3+1, 56, 4, 1, 45, 0) ;
        else
                counter := pulses;
        endif
endfunc
```

## Running the project

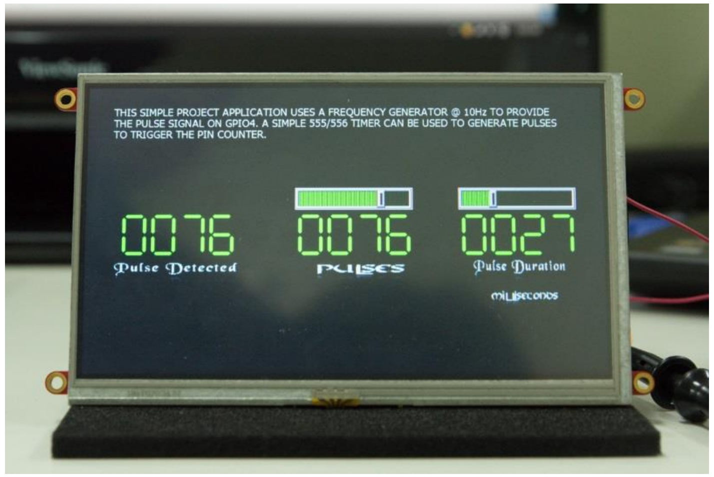
Compile and download the program to the display module. Having been able to complete this step, the next step that needs to be done is to provide the pulsating input on the GPIO PA4. Inputs to this pin are limited to an approximate value of 1Khz.

For the sole purpose of demonstration, a digital arbitrary waveform generator is used to provide the logic transitions. Furthermore, to clearly visualize the output of the GPIO PA5 – that is the pulse output, an oscilloscope will be used to view this waveform.

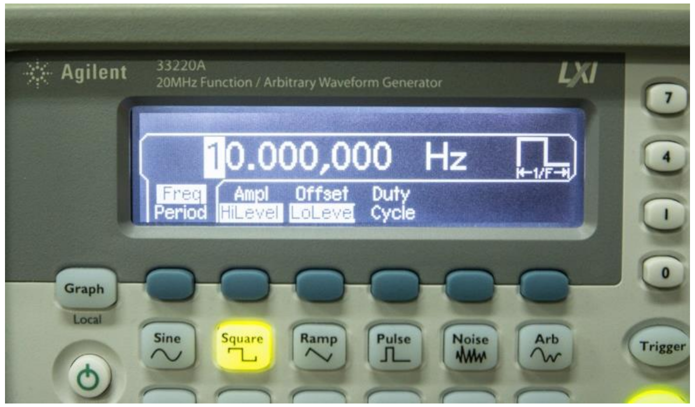

THIS SIMPLE PROJECT APPLICATION USES A FREQUENCY GENERATOR @ 10Hz TO PROVIDE THE PULSE SIGNAL ON GPIO4. A SIMPLE 555/556 TIMER CAN BE USED TO GENERATE PULSES TO TRIGGER THE PIN COUNTER.
0004
Pulse Detected
0043
PULSES
0027
Pulse Duration
milliseconds

This application starts with the selection of the pulse counting limit. Every time the pin counter GPIO detects a transition from logic 0 to logic 1 the pulse detected digits is incremented until it reaches the pulse limit. Also, the duty cycle of the pulse output is controlled using the slider on the right most side. Note that the duration set is in milliseconds and that it is multiplied by ten.

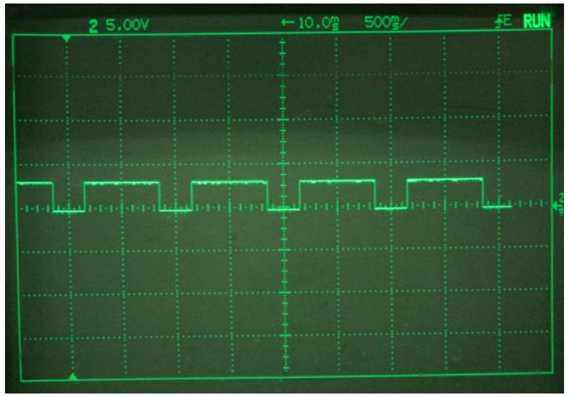
Pulse counting is terminated when the counting limit is set. Changing the counting limit value will reset the counter to the preset value. This triggers the pin counter to start all over again.


THIS SIMPLE PROJECT APPLICATION USES A FREQUENCY GENERATOR @ 10Hz TO PROVIDE THE PULSE SIGNAL ON GPIO4. A SIMPLE 555/556 TIMER CAN BE USED TO GENERATE PULSES TO TRIGGER THE PIN COUNTER.
0076
Pulse Detected
0076
PULSES
0027
Pulse Duration
milliseconds

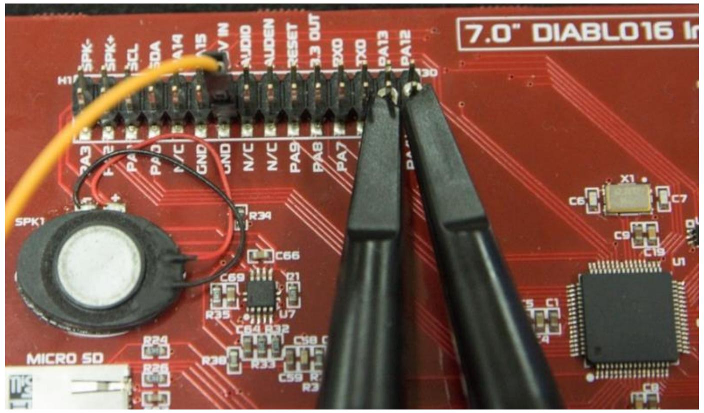
The pulse generator used in this application is an Agilent Arbitrary Waveform Generator set to a frequency of 10 Hz. Set with an output voltage of 0-5 volt dc peak voltage square wave output.



The resulting output is monitored using an oscilloscope. Referring to the image below, we can see that the output pulse is generated each time the pulse is detected. This means that for a count limit of ten, the resulting output pulse will be generated ten times. The generated output pulse will have a pulse duration from the preset value.



For the connection of the generator and the oscilloscope, the probe were placed adjacent to each other with their ground connected to the display module.



## Proprietary Information

## Disclaimer of Warranties & Limitation of Liability