May 2021

<BITS & BYTES> Newsletter

www.ccsinfo.com
262-522-6500

Product Spotlight

USB DEVELOPMENT KIT

This kit enables users to begin USB interface development with Microchip's PIC® PIC18 family. The development kit includes the powerful PCWH Integrated Development Environment with compiler support for Microchip's PIC® PIC10, PIC12, PIC16 and PIC18 families and an ICD-U64 in-circuit programmer/debugger that supports C-Aware real time debugging.

INSIDE THIS ISSUE:

Programmable Switch Mode Controller (PSMC)

By: CCS Staff

Pulse Width Modulation (PWM) capability was first added to the PIC® microcontroller line using the Capture/Compare/PWM (CCP) unit. Since then we have seen the enhanced CCP (ECCP) and a variety of power and motor control PWM modules each with its own features. The PWM module is called the Programmable Switch Mode Controller (PSMC). This is the most sophisticated PWM yet and is on chips like the PIC16F1789.

The PSMC allows for standard PWM, complementary PWM, shutdown control and deadband control like some of the older modules. It also allows for high resolution on the duty and frequency, as well variable frequency. It can for example do 3 phase 6 step PWM.

Input pins, comparator outputs and CCP triggers can be used not only to control shutdown, but to control if the PWM is running, or to start or stop a cycle.

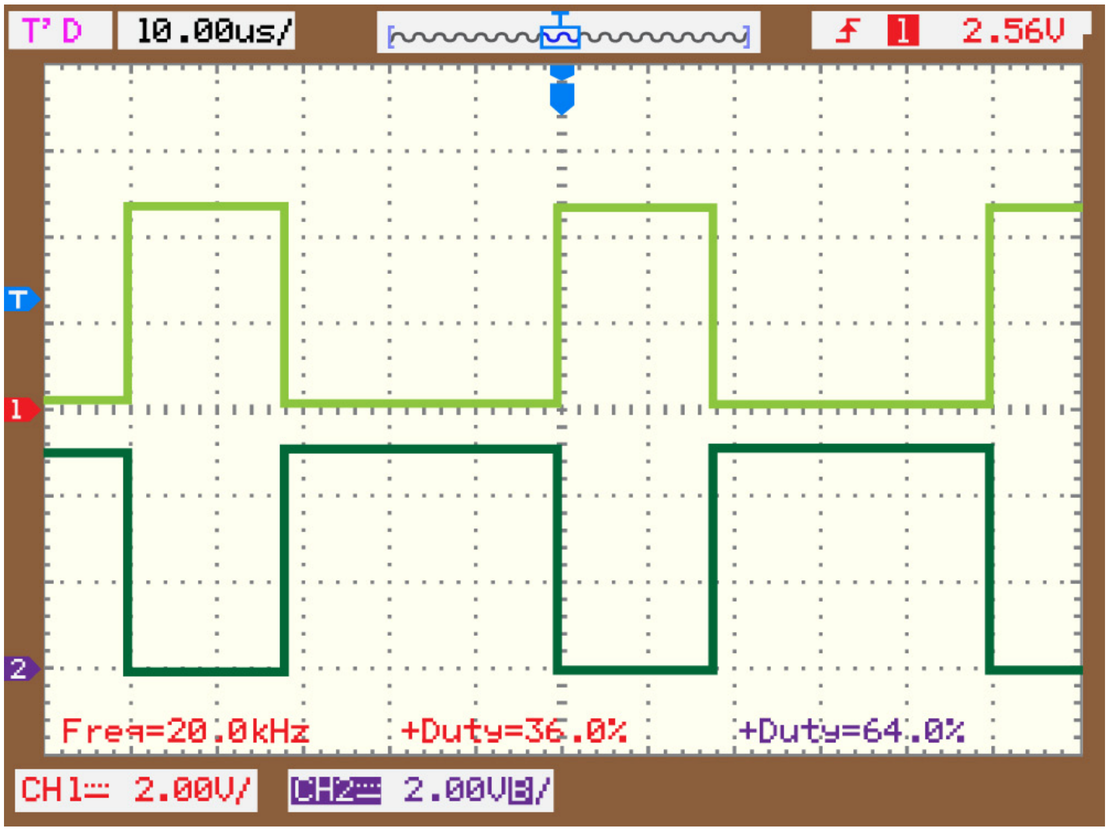
Some PIC® MCU devices have as many as 4 independent units and the units can optionally be synchronized with each other.

The compiler functions are:

```
setup_psmc(unit, mode, period, period_time,
rising_edge, rise_time,
falling_edge, fall_time);

psmc_pins(unit, pins_used, pins_active_low);

psmc_duty(unit, fall_time);

psmc_deadband(unit, rising_edge, falling_edge);

psmc_blanking(unit, rising_edge, rise_time, falling_edge, fall_time);

psmc_shutdown(unit, option, source, pins_high);

psmc_freq_adjust(unit, freq_adjust);

psmc_modulation(unit, options);

psmc_sync(slave_unit, master_unit, options);
```

The following is a short example program that shows off some of the features:

```
#include <16f1789.h>
#use delay(osc=20mhz)

#define us(time) (int16)(time*(getenv("CLOCK")/1000000))

void main(void) {
setup_psmc(1, PSMC_ECCP_BRIDGE_FORWARD,
PSMC_EVENT_TIME | PSMC_SOURCE_FOSC | PSMC_DIV_2, us(100),
PSMC_EVENT_TIME, us(10),
PSMC_EVENT_TIME, us(35));

psmc_deadband(1, us(2), us(4));
psmc_modulation(1, PSMC_MOD_IN_PIN);
psmc_pins(1, PSMC_A | PSMC_B | PSMC_C | PSMC_D);

setup_adc(ADC_CLOCK_INTERNAL);
setup_adc_ports(sAN0);
set_adc_channel(0);

while(TRUE) {
psmc_duty(1, us(((read_adc()*(int16)10)/25)) );
}
}
```



Other examples are included with the compiler download and begin with EX_PSMC_

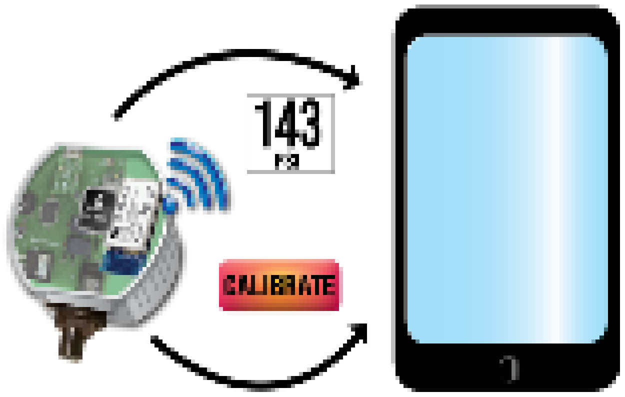
PIC® MCU, are registered trademarks of Microchip Technology Inc. in the U.S. and other countries.

Introduction to the EZ App Lynx Library

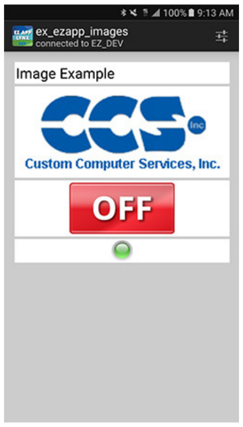
By: CCS Staff

The EZ App Lynx compiler library, Android Application, and iOS Application can add some neat capabilities to your PIC® MCU project. With a Bluetooth® interface, this allows a user's embedded project to interface to a smart device

The technique used to do this does not require to write an app for the phone. In the above diagram, the micro sends the text to display on the initially blank screen. It then requests a button be put on the screen. The generic app available from CCS simply does anything the PIC® MCU tells it to.



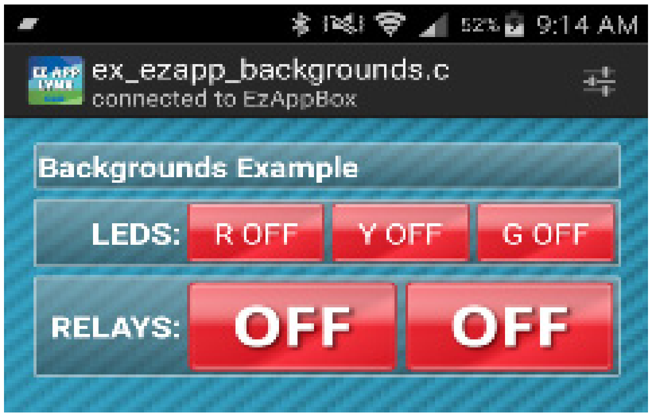
The same certified app available in the smart device stores (for free) can be used for any number of PIC® MCU programs. Each program makes the display and operation it's own. Graphics and logs can be sent to the smart device or you can required the app download the images from the web.



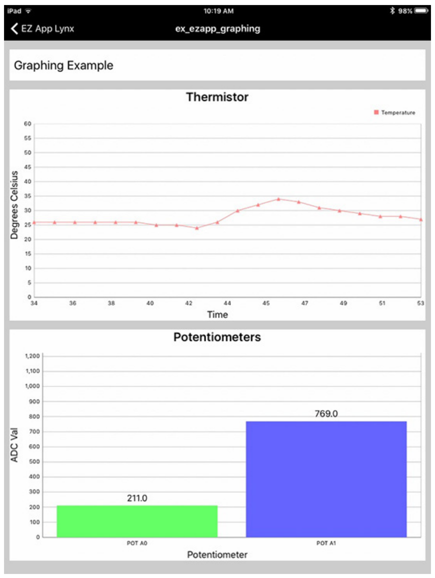
At the PIC® MCU, a simple library is used call EZ APP LYNX and is included with all IDE compilers. An example program looks like this:

```
   void main(void) {
      ezapp_field_index_t pot;
      EZAppInit();          //EZ App Lynx API

       pot = EZAppAddFieldAnalogValueScaled(
              "POT_A1",
              EZAPP_ANALOG_TYPE_GAS_GAUGE,
              0,      //min
              1023,   //max
              330,    //scaling
              2       //decimal points, 3.30
             );

  for(;;) {
     EZAppTask();    //EZ App Lynx API to keep Bluetooth up
     EZAppSetValue(pot, read_adc());
             }
   }
```

The library has a number of buttons, indicators, sliders and much more. Here is an example inserting a background on the smart device display:



Charts and graphics are easy to do as well:
An easy way to get started is using the development kit that includes a plug in Bluetooth® module, ready to start communicating with a smart device.



More information about EZ App Lynx, visit: http://www.ccsinfo.com/content.php?page=ez-app

PIC® MCU, are registered trademarks of Microchip Technology Inc. in the U.S. and other countries.

The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by Bluegiga Technologies is under license. Other trademarks and trade names are those of their respective owners.

ADC2

By: CCS Staff

Microchip has some chips out that they call ADC2. This is the ADC module with a computational unit. In addition to legacy mode of operation, the ADC2 module can be setup in Accumulate, Average, Burst Average or Low-Pass Filter modes of operation. When setup for Accumulate mode with each trigger of the ADC, the conversion result is added to the accumulator and the ADCNT register is incremented. When setup for Average mode with each trigger of the ADC the conversion result is added to the accumulator and when the specified number of triggers has been done, the average will be preformed on the accumulator. When setup for Burst Average mode, it is similar to Average mode the difference being that when the ADC is triggered, it will perform the specified number of conversions and then perform the average calculation. When setup for Low-Pass Filter mode with each trigger of the ADC, the conversion result is sent through the filter.

To support these modes of operation, the CCS C Compiler's built-in setup_adc() function has an optional 2nd and 3rd parameter to set some parameters when using these the new modes of operation. When using Accumulate, Average or Burst Average modes the 2nd parameter sets how much the accumulated value is divided by after each conversion. When using Low-Pass Filter mode the 2nd parameter sets the cut-off frequency of the filter. The 3rd parameter to the function sets the number of samples to be done before performing a threshold comparison for Average, Burst Average and Low-Pass Filter modes of operation.

In addition to the updates to the setup_adc() function, the built-in functions adc_write() and adc_read() have been added to write and read some the other registers used by the ADC2 module. For example when setup for Low-Pass Filter mode, the filtered result is stored in the ADFLTRH and ADFLTRL registers. After the calculation is completed the adc_read() function can be then be used to read the filtered result, for example:

```
      Result = adc_read(ADC_FILTER);
```

Another feature of the ADC2 module is the ability to set a trigger source to start a conversion. To support this feature the built-in function set_adc_trigger() has been added to the CCS C Compiler. For example the following can be used to setup the ADC2 module to trigger the conversion when Timer 2 period match occurs:

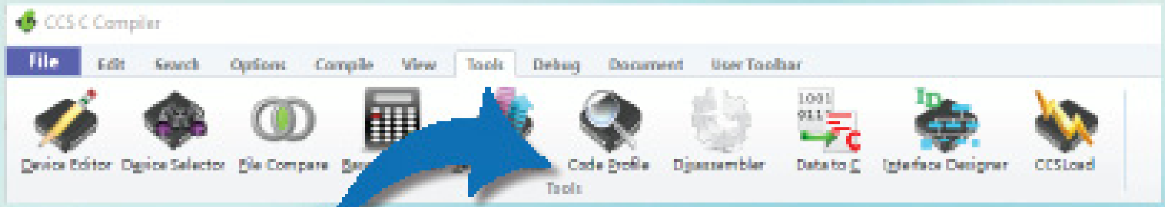
```
      set_adc_trigger(ADC_TRIGGER_TIMER2);
```

The following is an example using the CCS C Compiler to setup and use the ADC2 module for Low-Pass Filter mode, see ex_lowpass_filter_adc2.c in the PICC\Examples folder for entire example:

```
   setup_timer_2(T2_CLK_INTERNAL | T2_DIV_BY_128, 155, 10);
                 //~10ms period, 100ms interrupt

   setup_adc_ports(ADC_PIN, VSS_VDD);
   setup_adc(ADC_LOW_PASS_FILTER_MODE | ADC_CLOCK_INTERNAL |
             ADC_TAD_MUL_255 | ADC_THRESHOLD_INT_END_OF_CALCULATION,
             FILTER_CUT_OFF_FREQ, ADC_READINGS);

   set_adc_channel(ADC_CHANNEL);
   set_adc_trigger(ADC_TRIGGER_TIMER2);

   while(TRUE)
   {
      if(interrupt_active(INT_AD_THRESHOLD))
      {
         FilteredResult= adc_read(ADC_FILTER);

         clear_interrupt(INT_AD_THRESHOLD);
      }
   }
```

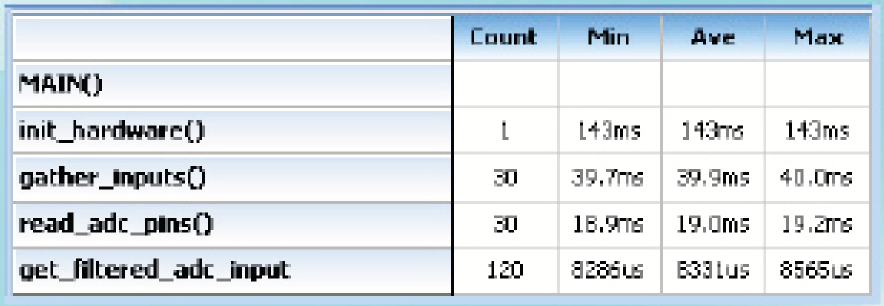
Conditioning ADC data has become a standard requirement for dealing with analog voltages. Even a small amount of noise can disrupt your application. The ADC2 modules can save a lot of processing time to automate the most common filtering operations. See the ex_adc2_trigger.c example program in the compiler example directory for a full program.

CCS COMPILER FEATURE FOCUS



Code Profiler

The Code Profile feature in the Tool Menu runs a program and provides statistics about how often functions are called, how much time is spent in them, the call order and much more.



	Count	Min	Ave	Max
MAIN()				
init_hardware()	1	143ms	143ms	143ms
gather_inputs()	30	39.7ms	39.9ms	40.0ms
read_adc_pins()	30	18.9ms	19.0ms	19.2ms
get_filtered_adc_input	120	8286us	8331us	8565us

USB Development Kit

Includes

- USB Prototyping Board
- In-Circuit Debugger/ Programmer
- Exercise Tutorial
- 9V AC Adapters and Cables

sales@ccsinfo.com
262-522-6500 EXT 35
www.ccsinfo.com/NL521

Spring into Coding Projects with a C Compiler

$25 Off a Full Compiler or Compiler Maintenance

UseCode:Spring2021

More than 25 years experience in software, firmware and hardware design and over 500 custom embedded C design projects using a Microchip PIC® MCU device. We are a recognized Microchip Third-Party Partner.

www.ccsinfo.com

Follow Us!