Jan/Feb 2022

www.ccsinfo.com
262-522-6500

<BITS & BYTES> Newsletter

Product Spotlight

INSIDE THIS ISSUE:

CAPACITIVE TOUCH DEVELOPMENT KIT

Create human touch applications with the Capacitive Touch Development Kit. Along with a prototyping board, it includes the powerful PCW Integrated Development Environment with compiler support for Microchip's PIC10, PIC12 and PIC16 families and an ICD-U64 in-circuit programmer/debugger that supports C-aware real time debugging. The capacitive touch prototyping board features Microchip's PIC16LF727- part of the new generation of Enhanced Mid-Range family of PIC16 devices with mTouch™ Sensing Solution technology. An exercise tutorial contain 14 example programs that steps the user through capacitive touch applications.

support@ccsinfo.com sales@ccsinfo.com

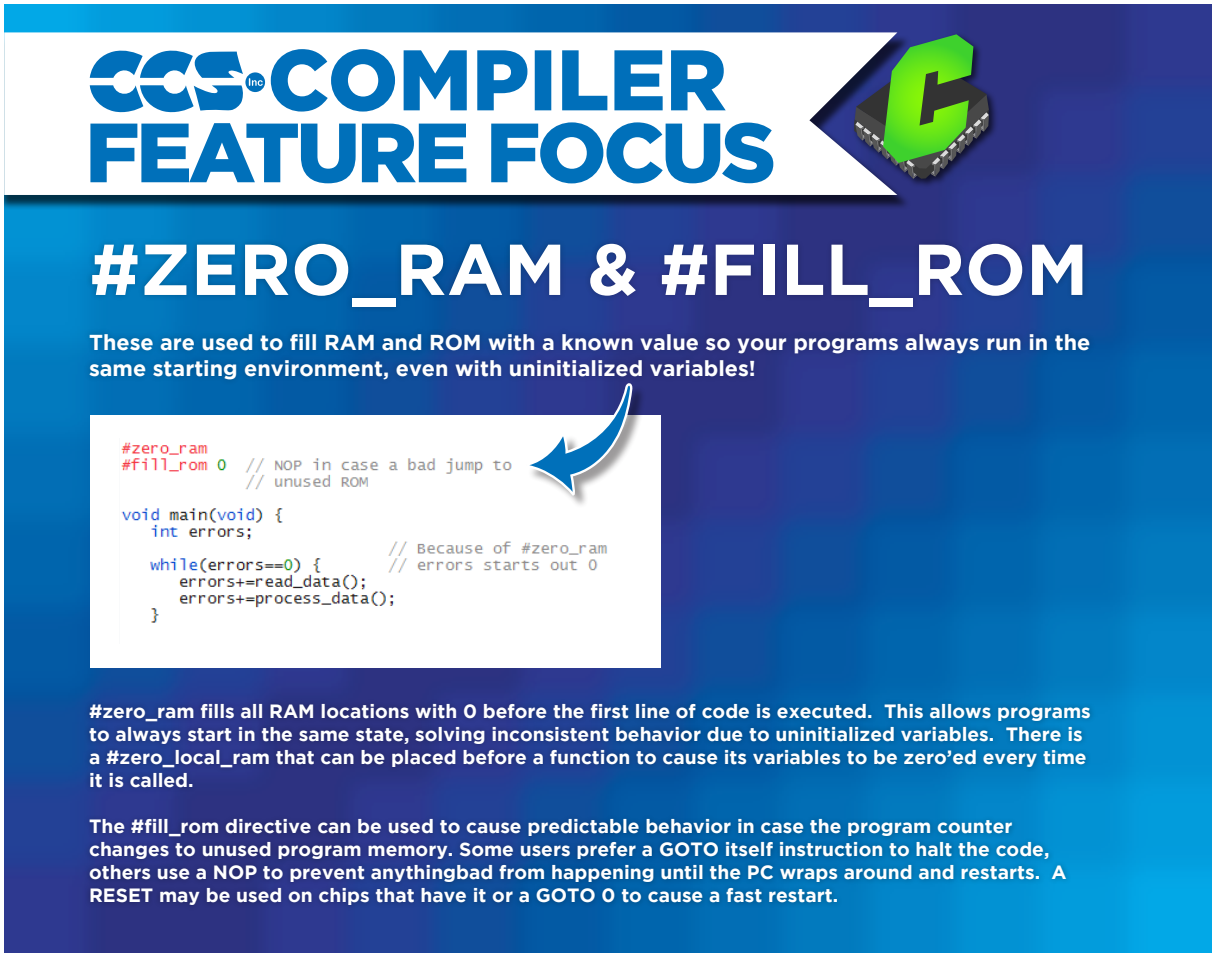
The generation of Enhanced Mid-Range Family of PIC16 devices include mTouch™Sensing technology. CCS demonstrates our uses for Capacitive Touch Development Board and compiler libraries in a fun hands-on application.

The CCS Capacitive Touch board uses the PIC16LF727 device for creating human touch applications and utilizing contact sensitive hardware. Instead of using mechanical switches or buttons that can break, the Capacitive Touch pads are activated by placing a human finger over the pad and the user's natural electrical properties generate the needed response or change.

Cut development time by utilizing specific Capacitive Touch functions built into the CCS C Compiler. The #USE TOUCH_PAD library reduces roughly 500 lines of assembly to 1 line of C code! A full version of the CCS C Compiler is available with the development kit.

The CCS Exercise tutorial contains 14 example programs that step the user through Capacitive Touch applications. Use the 16 on-board programmable capacitive pads and LCD to quickly develop "touch" applications. In addition, the board is equipped with *Tag Connect footprint for ICSP™ programming and a ICD-U64 that can be used with all Flash-supported PIC®MCU devices.

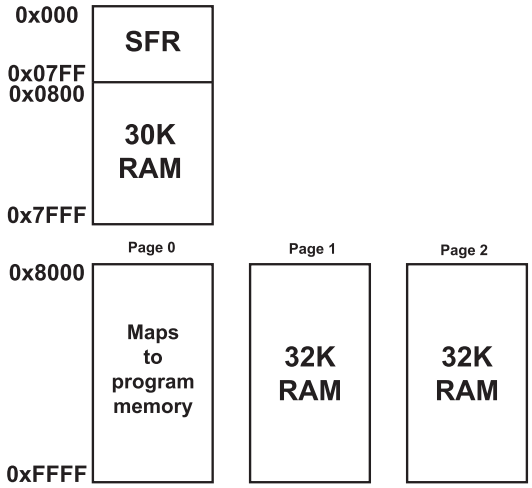
For additional information or to order today, go to: www.ccsinfo.com/Touch



Extended RAM on 24 bit parts

The 24 bit PIC architecture allows for very easy access from most instructions to locations 0x0000 to 0x1FFF. There are some additional instructions to move data from a working register to a location in the 0x2000 to 0x7FFF range. Note that 0x0000 to 0x07FF (or 0x0FFF on some parts) are special function registers, not general RAM.

Some devices have more than 30K of RAM. For these devices a special method is required to access the RAM above 30K. This extended RAM is organized into pages of 32K bytes each. The only way to access that RAM from assembly is to set a page register to identify the RAM page and then use indirect addresses from 0x8000 to 0xFFFF. Note that the first page is used to map program memory into the RAM address space. The other pages are the extended RAM.



From C, the compiler will allocate variables into the first page of extended RAM only. To access additional memory special functions must be used.

The basic functions to access that RAM are:

write_extended_ram(p, addr, ptr, n);

Writes n bytes from ptr to extended RAM page p starting at address addr.

read_extended_ram(p,addr,ptr,n);

Reads n bytes from extended RAM page p starting at address addr to ptr.

The first page is 1.

Example Code:

```
write_extended_ram(1,0x100,WriteData,8); //Writes 8 bytes from WriteData to
                                         //addresses 0x100 to 0x107 of
                                         //extended RAM page 1.

read_extended_ram(1,0x100,ReadData,8);   //Reads 8 bytes from addresses 0x100
                                         //to 0x107 of extended RAM page 1
                                         //to ReadData.
```

How Long Does It Take for Code to Execute

The PIC instructions are very deterministic in the time they take. There are exceptions, but in general a instruction takes 4 clocks (or 2 on some 24 bit chips) and if there is a change in the program counter it take s twice as long. Counting instructions in the LST file is one way to figure out the time code takes. Consider this example from the LST file:

```
.....................    if( a==b ) {
0C82:  MOVF   07,W
0C84:  SUBWF  06,W
0C86:  BNZ   0C90
.....................       a=1;
0C88:  MOVLW  01
0C8A:  MOVWF  06
.....................       b=9;
0C8C:  MOVLW  09
0C8E:  MOVWF  07
.....................    }
```

The IF statement takes instruction times if a==b or 4 otherwise. On a PIC18 this is 16 clocks. So if the chip oscillator (fosc) is 40mhz. Then the instruction time is 4/40000000 or 100ns. This IF statement takes 300ns or 400ns to execute. The two assignments take 400ns so in total if a==b then it takes 700ns or 400ns otherwise.

IDE users can use the code profiling tool to find out how long functions take to execute or to time how long it takes to get from one point in code to another.

Use code like the following To do timing manually:

```
setup_timer_1(t1_internal|t1_div_by_4);    // 1us tick
set_timer1(0);
for(i=1;i<=100;i++) {
   a=b;
}
overhead=get_timer1();

set_timer1(0);
clear_interrupt(int_timer1);

for(i=1;i<=100;i++) {
   a=b+c;
}
time=get_timer1();
time=time-overhead;

if(interrupt_active(int_timer1))
  printf("\r\nOVERFLOW");
printf("\r\nus=%6.2lw\r\n",time);
```

Unsigned 8 bit operations for math operations are quite fast and floating point is very slow. If possible consider fixed point instead of floating point.

For example, instead of “float cost_in_dollars;” do “long cost_in_cents;”. You can also get the compiler to do the math for you by using a declaration like “long fixed(2) cost_in_dollars;”

Consider a lookup table for trig formulas instead of real time calculations (see EX_SINE.C for an example).

Note all times will vary depending on memory banks used and sometimes for multiply, divide and float operations the actual numbers will affect the time.

PIC18 40 MHz 10 MIPS

	int8	int16	int32	float32
+	0.2 us	1.0 us	2.6 us	47.0 us
-	0.2 us	1.0 us	2.6 us	49.6 us
*	9.6 us	42.2 us	109 us	121 us
/	20.0 us	68.6 us	228 us	220 us
sin()				2.18 ms
crc32	84 ms			

PIC24 140 MHz 70 MIPS

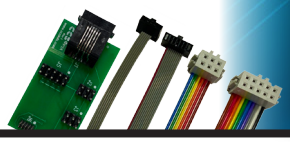
	int8	int16	int32	int48	int64
+	43 ns	14 ns	29 ns	43 ns	57 ns
-	71 ns	14 ns	29ns	43 ns	57 ns
*	85 ns	14 ns	727 ns	1.1 us	1.1 us
/	370 ns	271 ns	9.5 us	10 us	40 us
crc32	2.3 us				

	float32	float48	float64
+	3.4 us	3.5us	3.6us
-	3.4 us	3.5us	3.6 us
*	1.6 us	1.1 us	2.1 us
/	9.1 us	10.3 us	20.4 us
sin()	18 ms	46 ms	103 ms

8-Bit AVR® Support for Programmers

Programming support for all 8-bit AVR® microcontrollers. LOAD-n-GO, Prime8 and ICD-U80 supported. Programming adapter and cables available as separate purchase.

8-bit AVR® Programming Adapter
53505-1867 | $25.00



sales@ccsinfo.com
262-522-6500 EXT 35
www.ccsinfo.com/NL0222

CCS Inc

C Compiler Savings

$25 Off a Full Compiler or Compiler Maintenance

Use Code: Winter2022

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

CCS Inc

Follow Us!