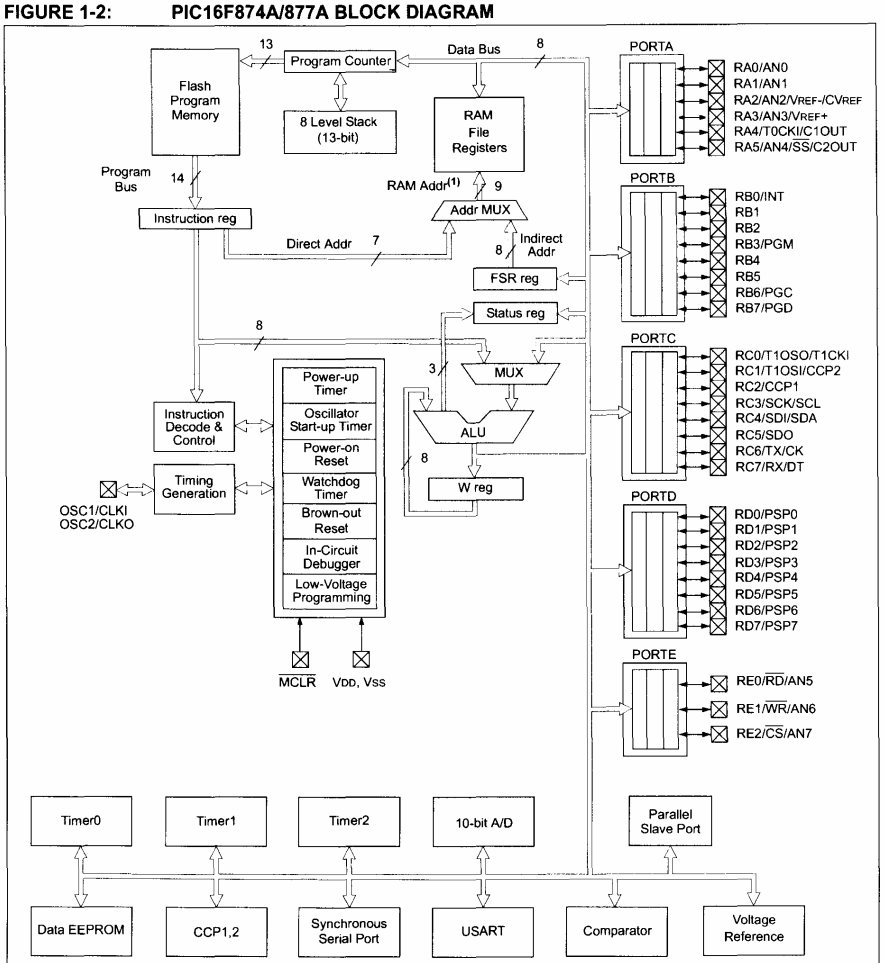
Summary of CCS C compiler Access to PIC internal peripherals

This document will provide summary function calls to operate the PIC 16F877a internal peripherals as described in ECET 431. The format is simple. The internal blocks are listed on the block diagram and the commands available are displayed along with a simple example. NOTE: This guide is not meant to be a comprehensive guide to the commands, but rather a way of organizing the commands. **To look at the exact syntax, one should look at the hardcopy compiler manuals in BELK 370, or at the help files that are in the CCS compiler**. The peripherals include:

1. Digital I/O
2. Timers
3. Capture and Compare (CCP)
    a. Capture mode
    b. Compare Mode
    c. PWM Mode
4. A/D conversion
5. Analog Comparator
6. USART (Universal Sychronous/Asynchronous Receiver/Transmitter)
7. SSP(Synchronous Serial Port)
8. PSP (Parallel Slave Port)
9. Internal Data EEPROM

FIGURE 1-2: PIC16F874A/877A BLOCK DIAGRAM



Device	Program Flash	Data Memory	Data EEPROM
PIC16F874A	4K words	192 Bytes	128 Bytes
PIC16F877A	8K words	368 Bytes	256 Bytes

Note 1: Higher order bits are from the Status register.

1. Digital I/O

On the PIC 16F877a, there are 5 digital I/O ports. They may be accessed as an entire port, or as individual pins on the port. The commands available are:

output_low(pin) // this will set the pin (either explicit or alias) to 0
output_high(pin) // this will set the pin (explicit (ie PIN_A3) or alias (MY_LED)) to 1
output_float(pin) // sets the pin to input mode, causing pin to float high if open collector
output_bit(pin, value) // sets the pin to value 0or 1
input(pin) // reads the value of pin ex "x=input(PIN_A2)"
output_X(value) // for PORT X (A-E) outputs "value" to the entire port
input_X() // reads the value of PORT X (A-E)
port_b_pullups(value) // allows the use of pullup resistors on individual pins of port B
set_tris_X(value) // allows the user to set the data direction register for port "X" by sending an 8 bit number to the TRIS register, bit 1 indicates input, bit 0 indicates output

2. Timers

For mode fields, use 1 from group1 or'd symbol(|) with 1 from group 2

setup_timerX(mode) // activates the timer and sets the mode of operation.

Timer0: syntax: **setup_timer_0(mode)**
Group1: RTCC_INTERNAL, RTCC_EXT_L_TO_H, RTCC_EXT_H_TO_L
Group 2: RTCC_DIV_2, RTCC_DIV_4, RTCC_DIV_8, RTCC_DIV_16, RTCC_DIV_32, RTCC_DIV_64, RTCC_DIV_128, RTCC_DIV_256

Timer1: syntax: **setup_timer_1(mode)**
Group1: T1_DISABLED, T1_INTERNAL, T1_EXTERNAL, T1_EXTERNAL_SYNC
Group2: T1_DIV_BY_1, T1_DIV_BY_2, T1_DIV_BY_4, T1_DIV_BY_8

Timer2: syntax: **setup_timer_2(mode, period, postscale)**
Group1: T2_DISABLED
Goup2: T2_DIV_BY_1, T2_DIV_BY_4, T2_DIV_BY_16

set_timerX(value) // sets timer register to an 8 or 16 bit value depending on timer
get_timerX() // retrieves timer value from timerX (0,1,2) and stores it in a desired
setup_counters(rtcc_state, ps_state) // older command to set up WDT and RTCC
valid rtcc_state: RTCC_INTERNAL, RTCC_EXT_L_TO_H, RTCC_EXT_H_TO_L
valid ps_state: RTCC_DIV_2, RTCC_DIV_4, RTCC_DIV_8, RTCC_DIV_16, RTCC_DIV_32, RTCC_DIV_64, RTCC_DIV_128, RTCC_DIV_256, WDT_18MS, WDT_36MS, WDT_72MS, WDT_144MS, WDT_288MS, WDT_576MS, WDT_1152MS, WDT_2304MS
setup_wdt(mode) // timer must be enabled, timer setup, and periodically reset

modes: WDT_18MS, WDT_36MS, WDT_72MS, WDT_144MS, WDT_288MS, WDT_576MS, WDT_1152MS, WDT_2304MS

example:

```
#fuses WDT

main()
{
setup_wdt(WDT_144MS);
while(1)
{
     restart_wdt();
    // my code goes here
}}
```

restart_wdt() // restarts WDT to avoid system reset

Example for simple timer use:

```
long time;

setup_timer_1(T1_INTERNAL  | T1_DIV_BY_1);  // sets up timer
set_timer1(0); // intitializes timer to 0

time=get_Timer1(); // gets the timer_1 value
printf("Time in ticks is %lu\r\n", time);
```

3. Capture and Compare (CCP)

Setup_ccpX(mode) // sets up the capture and compare unit "X" for the desired mode:

Valid mode values are:
CCP_OFF

Capture mode:
CCP_CAPTURE_FE
CCP_CAPTURE_RE
CCP_CAPTURE_DIV_4
CCP_CAPTURE_DIV_16

Compare mode:
CCP_COMPARE_SET_ON_MATCH
CCP_COMPARE_CLR_ON_MATCH
CCP_COMPARE_INT
CCP_COMPARE_RESET_TIMER

PWM Mode:
CCP_PWM

Set_pwmX_duty(value) // sets the duty cycle to the PWM"X" to the 10 but number in "value"

a. Example: Capture mode

```
long rise,fall,pulse_width;

#int_ccp2
void isr()
{
  rise = CCP_1;
  fall = CCP_2;

  pulse_width = fall - rise;     // CCP_1 is the time the pulse went high
}                                // CCP_2 is the time the pulse went low
                                 // pulse_width/(clock/4) is the time

                                 // In order for this to work the ISR
                                 // overhead must be less than the
                                 // low time.  For this program the
                                 // overhead is 45 instructions.  The
                                 // low time must then be at least
                                 // 9 us.

void main()
{
  printf("\n\rHigh time (sampled every second):\n\r");
  setup_ccp1(CCP_CAPTURE_RE);    // Configure CCP1 to capture rise
  setup_ccp2(CCP_CAPTURE_FE);    // Configure CCP2 to capture fall
```

```
  setup_timer_1(T1_INTERNAL);    // Start timer 1

  enable_interrupts(INT_CCP2);   // Setup interrupt on falling edge
  enable_interrupts(GLOBAL);

  while(TRUE) {
    delay_ms(1000);
    printf("\r%lu us ", pulse_width/5 );
  }
}
```

b. Example: Compare Mode

```
void main()
{
  setup_ccp1(CCP_COMPARE_CLR_ON_MATCH);     // Configure CCP1 in COMPARE mode
  setup_timer_1(T1_INTERNAL);               // Set up timer to instruction clk

  while(TRUE)
  {
     while(input(PIN_B0)) ;                 // Wait for keypress

     setup_ccp1(CCP_COMPARE_SET_ON_MATCH); // Configure CCP1 to set
     CCP_1=0;                              // C2 high now
     set_timer1(0);

     CCP_1 = 500;                          // Set high time limit
                                           // to 100 us
                                           // limit is time/(clock/4)
                                           // 500 = .0001*(20000000/4)

     setup_ccp1(CCP_COMPARE_CLR_ON_MATCH); // Configure CCP1 in COMPARE
                                           // mode and to pull pin C2
                                           // low on a match with timer1

     delay_ms(1000);                       // Debounce - Permit only one pulse/sec
  }
}
```

c. Example: PWM Mode

```
void main() {
  char selection;
  byte value;


  printf("\r\nFrequency:\r\n");
  printf("    1) 19.5 khz\r\n");
  printf("    2) 4.9 khz\r\n");
  printf("    3) 1.2 khz\r\n");

  do {
    selection=getc();
  } while((selection<'1')||(selection>'3'));
```

```
 setup_ccp1(CCP_PWM);   // Configure CCP1 as a PWM

      //   The cycle time will be (1/clock)*4*t2div*(period+1)
             //   In this program clock=10000000 and period=127 (below)
      //   For the three possible selections the cycle time is:
                        //     (1/10000000)*4*1*128 =  51.2 us or 19.5 khz
                        //     (1/10000000)*4*4*128 = 204.8 us or 4.9 khz
                        //     (1/10000000)*4*16*128= 819.2 us or 1.2 khz

 switch(selection) {
  case '1' : setup_timer_2(T2_DIV_BY_1, 127, 1);
          break;
  case '2' : setup_timer_2(T2_DIV_BY_4, 127, 1);
          break;
  case '3' : setup_timer_2(T2_DIV_BY_16, 127, 1);
          break;
 }



setup_port_a(ALL_ANALOG);
setup_adc(adc_clock_internal);
set_adc_channel( 0 );
printf("%c\r\n",selection);

while( TRUE ) {
  value=read_adc();

  printf("%2X\r",value);

  set_pwm1_duty(value);         // This sets the time the pulse is
                          // high each cycle.  We use the A/D
                          // input to make a easy demo.
                          // the high time will be:
                          //  if value is LONG INT:
                          //    value*(1/clock)*t2div
                          //  if value is INT:
                          //    value*4*(1/clock)*t2div
                          // for example a value of 30 and t2div
                          // of 1 the high time is 12us
                          // WARNING:  A value too high or low will
                          //          prevent the output from
                          //          changing.
 }

}
```

4. A/D conversion

setup_adc_ports(value) //sets up the pins to be analog, digital, or both

valid values:
NO_ANALOGS, ALL_ANALOG
AN0_AN1_AN2_AN4_AN5_AN6_AN7_VSS_VREF
AN0_AN1_AN2_AN3_AN4
AN0_AN1_AN2_AN4_VSS_VREF
AN0_AN1_AN3
AN0_AN1_VSS_VREF
AN0_AN1_AN4_AN5_AN6_AN7_VREF_VREF
AN0_AN1_AN2_AN3_AN4_AN5
AN0_AN1_AN2_AN4_AN5_VSS_VREF
AN0_AN1_AN4_AN5_VREF_VREF
AN0_AN1_AN4_VREF_VREF
AN0_AN1_VREF_VREF
AN0
AN0_VREF_VREF
ANALOG_RA3_REF
A_ANALOG
A_ANALOG_RA3_REF
RA0_RA1_RA3_ANALOG
RA0_RA1_ANALOG_RA3_REF
ANALOG_RA3_RA2_REF
ANALOG_NOT_RE1_RE2
ANALOG_NOT_RE1_RE2_REF_RA3
ANALOG_NOT_RE1_RE2_REF_RA3_RA2
A_ANALOG_RA3_RA2_REF
RA0_RA1_ANALOG_RA3_RA2_REF
RA0_ANALOG
RA0_ANALOG_RA3_RA2_REF

setup_adc(mode) // sets up the way that the ADC system works
valid mode values:
ADC_OFF
ADC_CLOCK_DIV_2
ADC_CLOCK_DIV_4
ADC_CLOCK_DIV_8
ADC_CLOCK_DIV_16
ADC_CLOCK_DIV_32
ADC_CLOCK_DIV_64
ADC_CLOCK_INTERNAL

set_adc_channel(channel) // specifies channel to be read by read_adc command

read_adc(mode) // read a/d measurement
mode here is optional, valid values are:
ADC_START_AND_READ (default), ADC_START_ONLY, ADC_READ_ONLY

ADC example:

```
 cutoff 128
 neutral_zone 25

main()

{
int reading;

setup_adc_ports(RA0_ANALOG);
setup_adc(ADC_CLOCK_INTERNAL);
set_adc_channel(0);

while(TRUE)
{
reading=read_adc();
printf("%d \r\n", reading);
if(reading<(cutoff-neutral_zone/2))
light_one_led(GREEN);
else if (reading > (cutoff+neutral_zone/2))
light_one_led(RED);
else
light_one_led(YELLOW);
}
}
```

5. Analog Comparator

Note: the comparator is used to generate an interrupt, and the interrupt service routine is used to perform the task based on the comparison

setup_comparator(mode) // sets up the comparator by which pins are compared
C1- and C1+ and C2- and C2+
Valid mode values:
A0_A3_A1_A3
A0_A3_A1_A2_OUT_ON_A4_A5
A0_A3_A1_A3_OUT_ON_A4_A5
NC_NC_NC_NC
A0_A3_A1_A2
A0_A3_NC_NC_OUT_ON_A4
A0_VR_A1_VR
A3_VR_A2_VR

Example: note: uses Vref tool as well

```
#INT_COMP
void isr()  {

  safe_conditions=FALSE;
  printf("WARNING!!  Voltage level is above 3.6 V.   \r\n");
```

```
}

main()  {

  printf("\r\nRunning voltage test...\r\n\n");

  setup_comparator(A1_VR_OUT_ON_A2);
  setup_vref(VREF_HIGH|15);
  enable_interrupts(INT_COMP);
  enable_interrupts(GLOBAL);

  while(TRUE)
  {
    if(safe_conditions)
      printf("Voltage level is below 3.6 V.          \r\n");
    safe_conditions=TRUE;
    delay_ms(500);
  }
}
```

6. Voltage Reference

setup_vref(mode | value) // sets up the voltage reference for analog compare and analog out on Pin A2 if so desired
valid mode values:
FALSE
VREF_LOW
VREF_HIGH
Value may be any int 0-15

7. USART (Universal Sychronous/Asynchronous Receiver/Transmitter)

value=getc() // waits for a character from RS232 and returns it in value
putc(chardata) // puts the character in chardata out the rs232 xmit pin
fgetc(stream) // same as getc except uses an input stream
gets(stringptr) // gets string from rs232 rcv and puts in string pointed to by stringptr
puts(string) // sends string output to RS232
fgets(stringptr, stream) // same as gets() except uses stream input
fputc(chardata, stream) // same as putc() but reroutes to the stream output
fputs(string, stream) // same as puts() but to stream identifier
printf(cstring, values) // prints out on RS232 values formatted by cstring info
value=kbhit() // returns true if character if waiting in buffer
fprintf(stream, cstring, values) // same as printf but on stream identifier
set_uart_speed(baud, [stream]) // sets the baud rate of the USART, stream optional
perror(string) // prints string out on stdoutput on system error
getchar() // same as getc()
putchar() // same as putc()

8. SSP(Synchronous Serial Port)

The SSP actually controls two protocols: SPI and I2C

SPI Commands:

setup_spi(mode) // sets up the SSP for SPI operations
mode = (groups may be or'd (|))
group1: SPI_MASTER, SPI_SLAVE, SPI_SS_DISABLED,
group 2: SPI_L_TO_H, SPI_H_TO_L,
group 3: SPI_CLK_DIV_4, SPI_CLK_DIV_16, SPI_CLK_DIV_64, SPI_CLK_T2

value = spi_read(data) // returns value read by SPI port. IF data is added, it is clocked out on the read.

spi_write(data) // transmits 8 bit int data onto SPI lines
result= spi_data_is_in() // returns true if SPI data has been received

I2C Commands:

NOTE: all i2c must use the compiler directive #use i2c

#use i2c(options) // options include (separated by commas):

MASTER
SLAVE
SCL=pin
SDA=pin
ADDRESS==nn
FAST
SLOW
RESTART_WDT
FORCE_HW

i2c_start() // begins an i2c transaction in master mode, waits for i2c_write()

i2c_stop() // stops i2c transaction in master mode
data=i2c_read([ack]) // reads single byte (data) over the i2c bus
i2c_write(data) // sends a single byte over the i2c interface
result=i2c_poll() // returns 1 if data is in buffer, ready for i2c_read()

Example I2C: (master side only: slave side waits for clock from master)

#define DAL_SCL PIN_B0

```
#define DAL_SDA PIN_B1
#endif

#define read_temp    read_full_temp // for backwards compatability

#use i2c(master, sda=DAL_SDA, scl=DAL_SCL)


void temp_config(BYTE data) {

  i2c_start();
  i2c_write(0x90);
  i2c_write(0xac);
  i2c_write(data);
  i2c_stop();
}


void init_temp() {
  output_high(DAL_SDA);
  output_high(DAL_SCL);
  i2c_start();
  i2c_write(0x90);
  i2c_write(0x51);
  i2c_stop();
  temp_config(0xc);
}


signed long read_full_temp() {  // Returns hundreths of degrees F (-67 to 257)
  signed int  datah, datal;
  signed long data;

  i2c_start();
  i2c_write(0x90);
  i2c_write(0xaa);
  i2c_start();
  i2c_write(0x91);
  datah=i2c_read();
  datal=i2c_read(0);
  i2c_stop();

  data=(signed long)datah*100;
  data=data+(((datal >> 4 )*(long)50)/16);
  data=data*9;
  data = (data / 5) + 3200;

  return(data);
}
```

9. PSP (Parallel Slave Port)

setup_psp(mode) // setup the parallel slave port for operation Port D for data and Port E for Control
valid mode values are:
PSP_ENABLED
PSP_DISABLED

result=**psp_input_full()**
result=**psp_output_full()**
result=**psp_overflow()**

all three return TRUE or FALSE depending upon condition

example:

```
#define BUSY_LINE    PIN_C0
#define BUFFER_SIZE  96


int next_in = 0;
int next_out = 0;
short data_lost = TRUE;

#int_psp
void psp_isr()  {

  if(psp_overflow())
    data_lost=TRUE;

  if(psp_input_full()) {
    write_bank(2, next_in++, input_D());
    if(next_in == BUFFER_SIZE)
      next_in = 0;
    if(next_in == next_out)
      output_high(BUSY_LINE);
  }
}

void main() {
  setup_adc_ports(NO_ANALOGS);
  setup_psp(PSP_ENABLED);
  enable_interrupts(GLOBAL);
  enable_interrupts(INT_PSP);
  output_low(BUSY_LINE);
  printf("Waiting for print data... \r\n\n");

  while(true)
  {
    if(next_in!=next_out)
    {
      putc(read_bank(2,next_out));
      if(++next_out==BUFFER_SIZE)
        next_out=0;
      if(data_lost) {
```

```
        printf("\r\nData Lost!!!\r\n");
        data_lost = FALSE;
      }
      output_low(BUSY_LINE);
    }
  }
}
```

10. Internal Data EEPROM

read_eeprom(address) // given 8 bit address, reads 8 bit value
write_eeprom(address, value) // writes 8 bit value to 8 bit address