UI Module

The user interface (UI) module provides a software interface to the three user light-emitting diodes (LEDs) and the three user tactile switches on the board.

The module defines two structures, _LED and _SW, which serve as software objects corresponding respectively to the user LEDs and tactile switches. It provides three statically defined _LED structures (led1, led2, and led3) and three statically defined _SW structures (sw1, sw2, and sw3) that interface to the LEDs and tactile switches with corresponding silkscreened labels on the board.

The module provides the following functions/methods:

```
  void init_ui(void)

    Initializes the UI module including the led1, led2, and led3 _LED
    structures and the sw1, sw2, and sw3 _SW strucutres.  You must call
    this function one time before calling any of the other functions
    provided by the UI module.

  void led_on(_LED *self)

    Turns on the user LED corresponding to the one pointed to by self.

  void led_off(_LED *self)

    Turns off the user LED corresponding to the one pointed to by self.

  void led_toggle(_LED *self)

    Toggles the user LED corresponding to the one pointed to by self.

  void led_write(_LED *self, uint16_t val)

    Turns on or off the user LED corresponding to the one pointed to by self
    depending on the value of val.  If val is zero, the LED is turned off.
    Otherwise, it is turned on.

  uint16_t led_read(_LED *self)

    Returns 1 if the user LED corresponding to the one pointed to by self is
    on and 0 if it is off.

  uint16_t sw_read(_SW *self)

    Returns 1 if the tactile switch corresponding to the one pointed to by
    self is not depressed and 0 if it is depressed.  Note that these input
    signals are active low.
```

Pin Module

The pin module provides a software interface to the input/output (I/O) pins on the analog (A0...A5) and digital (D0...D13) headers on the board.

The module defines the _PIN structure, which serves as a software object corresponding to an I/O pin. It provides two statically defined arrays of pin objects, A[] (six elements) and D[] (14 elements), which correspond to the analog and digital headers, respectively. Note that all of the I/O pins in the analog header can be configured as digital pins.

The module provides the following functions/methods:

```
  void init_pin(void)

    Initializes the pin module including the A[0]...A[5] and D[0]...D[13]
    _PIN structures.  Initially, the A[] pins are configured as analog input
    pins and the D[] pins are configured as digital input pins.  You must
    call this function one time before calling any of the other functions
    provided by the pin module.
  void pin_digitalIn(_PIN *self)

    Configure the I/O pin corresponding to the one pointed to by self to be a
    digital input pin.

  void pin_digitalOut(_PIN *self)

    Configure the I/O pin corresponding to the one pointed to by self to be a
    digital output pin.

  void pin_analogIn(_PIN *self)

    If possible, configure the I/O pin corresponding to the one pointed to by
    self to be an analog input pin.  If not possible, the function returns
    without altering the configuration of the given pin.

  void pin_set(_PIN *self)

    If the I/O pin corresponding to the one pointed to by self is a digital
    output pin, set its value high (i.e., to 1).  Otherwise, do nothing.

  void pin_clear(_PIN *self)

    If the I/O pin corresponding to the one pointed to by self is a digital
    output pin, set its value low (i.e., to 0).  Otherwise, do nothing.

  void pin_toggle(_PIN *self)

    If the I/O pin corresponding to the one pointed to by self is a digital
    output pin, toggle its value.  Otherwise, do nothing.

  void pin_write(_PIN *self, uint16_t val)

    If the I/O pin corresponding to the one pointed to by self is a digital
    output pin, set its value depending on val.  If val is zero, set the pin
    low, otherwise set it high.

    If the I/O pin corresponding to the one pointed to by self has been
    configured to produce a pulse-width-modulated (PWM) signal, set its duty
    cycle to the value of val, interpreted as a 16-bit fixed-point fraction
    (i.e., the most-significant bit is the 1/2 digit, the second
    most-significant bit is the 1/4 digit, etc.).

    If the I/O pin corresponding to the one pointed to by self has been
    configured to produce a servo control signal, set the pulse width
    depending on the value of val, interpreted as a 16-bit fixed-point
    fraction (0 corresponds to the minimum pulse width and 1 corresponds
    to the maximum pulse width).

  uint16_t pin_read(_PIN *self)

    If the I/O pin corresponding to the one pointed to by self is a digital
    I/O pin, return its value (i.e., 0 or 1).

    If the I/O pin corresponding to the one pointed to by self is an analog
    input pin, perform an analog-to-digital conversion and return the value
    as a 16-bit fixed-point fraction (i.e. the 10-bit result from the ADC is
    left justified within the 16-bit value returned).

    If the I/O pin corresponding to the one pointed to by self has been
    configured to produce a pulse-width-modulated (PWM) signal, return the
    current duty cycle as a 16-bit fixed-point fraction.

    If the I/O pin corresponding to the one pointed to by self has been
    configured to produce a servo control signal, return the current pulse
    width as a 16-bit fixed-point fraction (0 corresponds to the minimum
    pulse width and 1 corresponds to the maximum pulse width).
```

Timer Module

The timer module provides a software interface to the PIC's five 16-bit timer
peripherals, which can be used to measure accurately time intervals, to serve
as a timing reference for other peripherals (e.g., an output compare

peripheral), or to schedule a callback function to be executed at a regular interval.

The module defines the _TIMER structure, which serves as a software object corresponding to a hardware timer peripheral. It also provides five statically defined timer objects (timer1, timer2, timer3, timer4, and timer5), which correspond to the PIC's five 16-bit hardware timer peripherals.

The module provides the following functions/methods:

void init_timer(void)

Initializes the timer module including the timer1, timer2, timer3, timer4, and timer5 _TIMER structures. You must call this function one time before calling any of the other functions provided by the timer module.

void timer_setPeriod(_TIMER *self, float period)

Stops the timer corresponding to the one pointed to by self and sets its period to the value specified. Here, the value of period is given in seconds. If the period specified is too large (i.e., greater than 2^24*TCY), does nothing.

float timer_period(_TIMER *self)

Returns the period (in seconds) of the timer corresponding to the one pointed to by self.

void timer_setFreq(_TIMER *self, float freq)

Stops the timer corresponding to the one pointed to by self and sets its frequency to the value specified by freq. Here, the value of freq is given in Hertz. If specified frequency is too low (i.e., less than FCY/2^24), does nothing.

float timer_freq(_TIMER *self)

Returns the frequency (in Hertz) of the timer corresponding to the one pointed to by self.

void timer_start(_TIMER *self)

Starts the timer corresponding to the one pointed to by self.

void timer_stop(_TIMER *self)

Stops the timer corresponding to the one pointed to by self.

float timer_time(_TIMER *self)

Returns the current value (in seconds) of the timer corresponding to the one pointed to by self.

uint16_t timer_read(_TIMER *self)

Returns the current value (in timer ticks) of the the timer corresponding to the one pointed to by self.

uint16_t timer_flag(_TIMER *self)

Returns the value of the interrupt flag of the timer corresponding to the one pointed to by self. The interrupt flag indicates if the timer has expired (e.g., the interval being timed has elapsed).

void timer_lower(_TIMER *self)

Lowers the interrupt flag of the timer corresponding to the one pointed to by self. Note that the interrupt flag must be lowered in software before the next interval elapses if the that interval is not to be missed.

void timer_enableInterrupt(_TIMER *self)

Enables the interrupt of the timer corresponding to the one pointed to by self.

```
void timer_disableInterrupt(_TIMER *self)
```

Disables the interrupt of the timer corresponding to the one pointed to by self.

```
void timer_every(_TIMER *self, float interval,
                 void (*callback)(_TIMER *self))
```

Uses the interrupt facility of the timer corresponding to the one pointed to by self to trigger the execution of the provided callback function periodically at the interval specified (in seconds) until canceled by a call to timer_cancel().

The callback function must take a single argument, which is a pointer to a _TIMER object and must not return anything. It should execute in a relatively short amount of time (e.g., it should certainly take less time to execute than the interval specified).

```
void timer_after(_TIMER *self, float delay, uint16_t num_times,
                 void (*callback)(_TIMER *self))
```

Uses the interrupt facility of the timer corresponding to the one pointed to by self to trigger the execution of the provided callback function the number of times specified by num_times at the interval specified by delay (in seconds).

The callback function must take a single argument, which is a pointer to a _TIMER object and must not return anything. It should execute in a relatively short amount of time (e.g., it should certainly take less time to execute than the interval specified).

```
void timer_cancel(_TIMER *self)
```

Cancels the execution of a callback function associated with the interrupt facility of the timer module corresponding to the one pointed to by self. Stops the timer and disables its interrupt.

OC Module

The output compare (OC) module provides a software interface to the PIC's nine output compare peripherals, which are capapble of producing a wide variety of digital output waveforms that encode infomation in various ways in time (e.g., pulse-width-modulated (PWM) signals and hobby servo control signals). Each of the OC peripherals has its own internal 16-bit hardware timer or it can make use of one of the five timer modules, which have a wider dynamic range through a prescalar mechanism. The OC module makes use of the pin and timer modules; consequently, you must include their header files and call their initialization functions if you want to use the OC module.

The OC module defines the _OC structure, which serves as a software object corresponding to a hardware OC peripheral. The module also statically defines nine OC objects (oc1, oc2, ... oc9), which each correspond to and control one of the PIC's nine OC peripherals.

The module provides the following functions/methods:

```
void init_oc(void)
```

Initializes the OC module including the oc1, oc2, ..., oc9 _OC structures. You must call this function one time before calling any of the other functions provided by the OC module.

```
void oc_pwm(_OC *self, _PIN *out, _TIMER *timer, float freq, uint16_t duty)
```

Configures the OC peripheral corresponding to the one pointed to by self to produce a PWM signal on the pin corresponding to the one pointed to by out with the frequency (in Hertz) specified by freq and the initital duty cycle specified (as a 16-bit fixed point fraction) by duty. If a pointer to a timer object is specified that timer is used as a timebase for the OC module. To use the OC peripheral's internal timer, specify NULL for the timer parameter. The pin_write() and pin_read() functions of the pin

```
    module are used respectively to set and to get the duty cycle thereafter.

    If the specified pin is not a remappable pin or if the pin is in use by
    another module, nothing happens.

  void oc_servo(_OC *self, _PIN *out, _TIMER *timer, float interval,
                float min_width, float max_width, uint16_t pos)

    Configures the OC peripheral corresponding to the one pointed to by self
    to produce a hobby servo control signal on the pin corresponding to the
    one pointed to by out.  The timer corresponding to the one pointed to by
    timer is used to time the period of the control signal, which is specified
    (in seconds) by interval (a typical value is 20e-3).  The minimum pulse
    width (in seconds) is specified by min_width (a typical value is 1e-3) and
    the maximum pulse width (in seconds) is specified by max_wdith (a typical
    value is 2e-3).  The initial position is specified (as a 16-bit fixed-point
    fraction with 0 corresponding to min_width and 1 corresponding to
    max_width) by pos.  The pin_write() and pin_read() functions of the pin
    module are used respectively to set and to get the pulse width thereafter.

    If the specified pin is not a remappable pin or if the pin is in use by
    another module, nothing happens.

  void oc_free(_OC *self)

    Disables the OC peripheral corresponding to the one pointed to by self and
    reconfigures the pin that it had been using to be a digital output (set to
    a low value).

UART Module
-----------
The universal asynchronous receiver/transmitter (UART) module provides a
software interface to the PIC's four hardware UART peripherals operating with
8-bit data transmission.  The UART peripherals can be used to communicate with
a host PC via a virtual COM (i.e. serial) port or with another embedded
processer that also has a UART peripheral.

The module defines the _UART structure, which serves as a software object
corresponding to a hardware UART peripheral.  The module also provides four
statically defined UART objects (uart1, uart2, uart3, and uart4), which each
correspond to and control one of the PIC's four UART peripherals.

The UART module arranges things so that output written to stdout and stderr
via functions in the stdio standard library (e.g., printf()) are sent to a
UART peripheral (i.e. uart1 operating at 19,200 baud with no parity and one
stopbit) and can be displayed on a serial terminal application running on a
host PC connected to the board via the supplied USB-UART cable plugged into
the audio jack on the board.  It is also easy to redirect stdout or stderr
to another UART module by simply making either the supplied _stdout or _stderr
variable point to another of the UART objects.

The module provides the following funcitons/methods:

  void init_uart(void)

    Initializes the UART module including the uart1, uart2, uart3, and uart4
    _UART structures.  Calls init_pin() to initialize the pin module.  Defines
    pin objects corresponding to those connected to the tip and sleve of the
    audio jack.  Opens uart1 to operate at 19,200 baud with no party and one
    stop bit using the pins connected to the audio jack.  Points _stdout and
    _stderr at uart1.  You must call this function one time before calling any
    of the other functions provided by the UART module.


  void uart_open(_UART *self, _PIN *TX, _PIN *RX, _PIN *RTS, _PIN *CTS,
                 float baudrate, int8_t parity, int16_t stopbits,
                 uint16_t TXthreshold, uint8_t *TXbuffer, uint16_t TXbufferlen,
                 uint8_t *RXbuffer, uint16_t RXbufferlen)

    Configures the UART corresponding to the one pointed to by self to make
    use of the transmit and receive pins specified by TX and RX at the baud
    rate specified by baudrate with the specified parity (use 'N' for none, 'E'
    for even, or 'O' for odd) and number of stop bits (1 or 2).  Enables data
```

transmission.

If RTS and CTS pins are provided, the UART is configured to use hardware flow control using the specified request-to-send (RTS) and clear-to-send (CTS) pins specified. If no hardware flow control is desired, you should pass NULL for RTS and CTS.

If any of the specified pins is not a remappable pin or if any of them is in use by another module, nothing happens.

If TXbuffer is provided, software transmit buffering is enabled using the buffer specified and the UART peripheral's transmit interrupt facility. Transmission is initiated if at least TXthreshold bytes are in the buffer. If no software transmit buffering is desired, you should pass NULL for TXbuffer and 0 for TXbufferlen and TXthreshold.

If RXbuffer is provided, software receive buffering is enabled using the buffer specified and the UART peripheral's receive interrupt facility. If no software receive buffering is desired, you should pass NULL for RXbuffer and 0 for RXbufferlen.

```
void uart_close(_UART *self)
```

Disables the UART corresponding to the one pointed to by self and reconfigures the pins that were being used as digital I/Os (RX/CTS are made digital inputs and TX/CTS are made digital outputs set high).

```
void uart_putc(_UART *self, uint8_t ch)
```

Send the character specified by ch via the UART corresponding to the one pointed to by self. Note that this function does not return (i.e. it blocks execution) until the character is transmitted.

```
uint8_t uart_getc(_UART *self)
```

Receive a character from the UART corresponding to the one pointed to by self and return its value. Note that this function does not return until a character is received (i.e., it blocks execution).

```
void uart_flushTxBuffer(_UART *self)
```

Flush the transmit buffer of the UART corresponding to the one pointed to by self. If software buffering is not enabled, nothing happens.

```
void uart_puts(_UART *self, uint8_t *str)
```

Transmit all of the characters in the null-terminated string pointed to by str via the UART corresponding to the one pointed to by self.

```
void uart_gets(_UART *self, uint8_t *str, uint16_t len)
```

Receive a string into the buffer pointed to by str of no more than len-1 characters via the UART corresponding to the one pointed to by self, terminate it with a null character, and return. Received characters are echoed via uart_putc(). Simple editing via the backspace/delete keys are supported. The escape key clears the buffer and the return/enter key signals the end of the string.