ESOS "hello world" program

```
#include "esos.h"
#include "esos_pic24.h"

// HW configuration macros (See Figures 8.3-8.5 in text)
#define CONFIG_LED1()    CONFIG_RB15_AS_DIG_OUTPUT()
#define LED1             _LATB15

// all ESOS applications must provide user_init()
void user_init(void) {
  __esos_unsafe_PutString(HELLO_MSG);

  CONFIG_LED1();

  esos_RegisterTask(heartbeat_LED);
}

ESOS_USER_TASK(heartbeat_LED) {
  ESOS_TASK_BEGIN();
  while (TRUE) {
    LED1 = !LED1;
    ESOS_TASK_WAIT_TICKS(250);
  }
  ESOS_TASK_END();
}
```

Call "hidden" ESOS hardware routine to print `HELLO_MSG`.

`user_init()` must register *at least one* task.

`heartbeat_LED` task runs forever turning `LED1` off and on every 250ms.

Signals end of `heartbeat_LED` task to compiler, but task never actually ends.

ESOS task calls

`ESOS_TASK_YIELD()`
Current task gives up focus until its next opportunity to run in the ESOS scheduler.
`ESOS_TASK_WAIT_UNTIL(cond)`
Current task gives up focus until **`cond`** evaluates to true. ESOS scheduler will query **`cond`** at each opportunity that the current task should have run. Does *not* block the current task at all if **`cond`** evaluates true at initial call.
`ESOS_TASK_WAIT_WHILE(cond)`
Current task gives up focus while **`cond`** evaluates to true. ESOS scheduler will query **`cond`** at each opportunity that the current task should have run. Does *not* block the current task at all if **`cond`** evaluates false at initial call. Equivalent to **`ESOS_TASK_WAIT_UNTIL(!cond)`**
`ESOS_TASK_WAIT_TICKS(u32_duration)`
Current task gives up focus for at least **`u32_duration`** system ticks. ESOS defines system tick as 1 ms.
`ESOS_TASK_WAIT_SEMAPHORE(pstSem, i16_val)`
Current task gives up focus while semaphore **`pstSem`** is not positive. When **`pstSem`** can be signaled with value of **`i16_val`**, the current task will resume execution.

`ESOS_TASK_WAIT_??????`
Many ESOS services provide operations that will cause a task to wait or yield. ESOS services which can potentially cause the current task to lose focus begin with the prefix **`ESOS_TASK_WAIT_`**. Refer to documentation for specific ESOS service for more details.

Echo program

```
#include "esos.h"
#include "esos_pic24.h"

// HW configuration macros (See Figures 8.3-8.5 in text.)
#define CONFIG_LED1()    CONFIG_RB15_AS_DIG_OUTPUT()
#define LED1             _LATB15

// all ESOS applications must provide user_init()
void user_init(void) {
  __esos_unsafe_PutString(HELLO_MSG);
  CONFIG_LED1();
  esos_RegisterTask(heartbeat_LED);
  esos_RegisterTask(echo);
}
```

Print boot message to screen.

`user_init()` registers two user tasks. They start *almost* immediately.

User task `heartbeat_LED` same as previous example.
Code not printed to conserve space.

```
ESOS_USER_TASK(echo) {
  static  uint8   u8_char;

  ESOS_TASK_BEGIN();
  while (TRUE) {
    ESOS_TASK_WAIT_ON_AVAILABLE_IN_COMM();
    ESOS_TASK_WAIT_ON_GET_UINT8(u8_char);
    ESOS_TASK_SIGNAL_AVAILABLE_IN_COMM();
    u8_char++;
    ESOS_TASK_WAIT_ON_AVAILABLE_OUT_COMM();
    ESOS_TASK_WAIT_ON_SEND_UINT8(u8_char);
    ESOS_TASK_SIGNAL_AVAILABLE_OUT_COMM();
  }
  ESOS_TASK_END();
}
```

`echo` uses `u8_char` across wait statements. `static` qualifier ensures value is preserved.

Wait for "in" stream to be free. Wait to get an `uint8`, then free the "in" stream.

Send `u8_char` to the "out" stream once we determine the stream is ours to use. Free it when done.

ESOS communication service

`ESOS_TASK_WAIT_ON_GET_UINT8(u8_in)`
Blocks current task until incoming `uint8` data is available. Results are placed into `u8_in`.
`ESOS_TASK_WAIT_ON_GET_UINT16(u16_in)`
Blocks current task until incoming `uint16` data is available. Results are placed into `u16_in`.
`ESOS_TASK_WAIT_ON_GET_UINT32(u32_in)`
Blocks current task until incoming `uint32` data is available. Results are placed into `u32_in`.
`ESOS_TASK_WAIT_ON_GET_U8BUFFER(pau8_in, u8_len)`
Blocks current task until an array of incoming `uint8` data is available. Reads `u8_len` bytes into the array starting at `pau8_in`.
`ESOS_TASK_WAIT_ON_GET_STRING(psz_in)`
Blocks current task until a zero-terminated string is available. Reads the string into the memory starting at `psz_in`.

`ESOS_TASK_WAIT_ON_SEND_UINT8(u8_out)`
Blocks current task until `u8_out` data can be absorbed by ESOS system.
`ESOS_TASK_WAIT_ON_SEND_UINT16(u16_out)`
Blocks current task until `u16_out` data can be absorbed by ESOS system.
`ESOS_TASK_WAIT_ON_SEND_UINT32(u32_out)`
Blocks current task until `u32_out` data can be absorbed by ESOS system.
`ESOS_TASK_WAIT_ON_SEND_U8BUFFER(pau8_out, u8_len)`
Blocks current task until `u8_len` long array of `uint8` data can be absorbed by ESOS system. Reads `u8_len` bytes from the array starting at `pau8_out`.
`ESOS_TASK_WAIT_ON_SEND_STRING(psz_out)`
Blocks current task until zero-terminated string `psz_out` can be absorbed by ESOS system. Reads the string from memory starting at `psz_out`.
`ESOS_TASK_WAIT_ON_SEND_UINT8_AS_HEX_STRING(u8_out)`
Blocks current task until string data representing `u8_out` can be absorbed by ESOS system. String is ASCII characters representing value `u8_out` in hexadecimal format.
`ESOS_TASK_WAIT_ON_SEND_UINT32_AS_HEX_STRING(u32_out)`
Blocks current task until string data representing `u32_out` can be absorbed by ESOS system. String is ASCII characters representing value `u32_out` in hexadecimal format.

ESOS timer service

`ESOS_USER_TIMER(timername)`
Declares and creates the software timer named **`timername`**. Function can not return values.
`ESOS_TMR_HANDLE esos_RegisterTimer(timername, u32_period)`
Registers the software timer **`timername`** with ESOS. ESOS timer service will call the timer's callback function every **`u32_period`** ticks. Timer **`timername`** will start counting immediately. Returns an **`ESOS_TMR_HANDLE`** to be used with other timer services functions.
`uint8 esos_UnregisterTimer(timer_handle)`
Unregisters the timer denoted by **`timer_handle`**. Returns TRUE if timer was successfully removed from the ESOS timer service; FALSE otherwise.
`ESOS_TMR_HANDLE esos_GetTimerHandle(timername)`
Returns **`ESOS_TMR_HANDLE`** to corresponding to the software timer **`timername`**, or **`ESOS_TMR_FAILURE`** if timer is not currently running. Useful if timer handle was not saved when timer was initially registered with ESOS.
`uint8 esos_ChangeTimerPeriod(timer_handle, u32_period)`
Changes the period of the currently registered timer denoted by **`timer_handle`** to be **`u32_period`** ticks. New period takes effect immediately. Returns TRUE if timer period is successfully changed; FALSE otherwise.

Flash LED via timer service

```
#include "esos.h"
#include "esos_pic24.h"

// HW configuration macros (See Figures 8.3-8.5 in text)
#define CONFIG_LED1()    CONFIG_RB15_AS_DIG_OUTPUT()
#define LED1             _LATB15

// all ESOS applications must provide user_init()
void user_init(void) {
  __esos_unsafe_PutString(HELLO_MSG);
  CONFIG_LED1();
  esos_RegisterTask(echo);

  esos_RegisterTimer(swTimerLED,250);
}
```

← Print boot message to screen.

← Register `echo` task.

← Register `swTimerLED` function as software timer callback. ESOS will call `swTimerLED` every 250 ticks.

User task `echo` same as previous example. Code not printed to conserve space.

```
ESOS_USER_TIMER(swTimerLED) {
  LED1 = !LED1;
}
```

← Toggle `LED1` every time timer callback is run.

ESOS semaphore service

`ESOS_SEMAPHORE(semaphoreName)`
Declares and creates the semaphore **`semaphoreName`**. This macro is usually placed in the global variables area of the application source since sempahores are used for inter-task synchronication.
`ESOS_INIT_SEMAPHORE(semaphoreName, i16_val)`
Initializes the semaphore **`semaphoreName`** with value `i16_val`. Semaphores are often initialized at the start of an application in `user_init()`.
`ESOS_TASK_WAIT_SEMAPHORE(semaphoreName, i16_val)`
Blocks the current task until **`semaphoreName`** has been signaled at least `i16_val` times. After being blocked, current task will continue execution at the next statement.
`ESOS_SIGNAL_SEMAPHORE(semaphoreName, i16_val)`
Signal **`semaphoreName`** `i16_val` times. This function does not have to be called inside of the context of a task since it cannot block.

ESOS task signaling via semaphore

```
// global variables
char psz_CRNL[3]= {0x0D, 0x0A, 0};
char psz_T1[] = "Task 1: ";
char psz_T2[] = "Task 2: ";
ESOS_SEMAPHORE(sem_T2CanRun);

void user_init(void) {
  __esos_unsafe_PutString(HELLO_MSG);
  CONFIG_LED1();
  ESOS_INIT_SEMAPHORE(sem_T2CanRun, 0);
  esos_RegisterTask(task1);
  esos_RegisterTask(task2);
  esos_RegisterTimer(swTimerLED,250);
}

inline uint32 getRandomDelay() {
  return ((esos_GetRandomUint32() & 0x0FFF)|0xFF);
}

ESOS_USER_TASK(task1) {
  static uint8  u8_cnt=0;
  ESOS_TASK_BEGIN();
  while (TRUE) {
    ESOS_TASK_WAIT_ON_AVAILABLE_OUT_COMM();
    ESOS_TASK_WAIT_ON_SEND_STRING(psz_T1);
    ESOS_TASK_WAIT_ON_SEND_UINT8_AS_HEX_STRING(u8_cnt);
    ESOS_TASK_WAIT_ON_SEND_STRING(psz_CRNL);
    ESOS_TASK_SIGNAL_AVAILABLE_OUT_COMM();
    ESOS_SIGNAL_SEMAPHORE(sem_T2CanRun, 1);
    u8_cnt++;
    ESOS_TASK_WAIT_TICKS(getRandomDelay());
  }
  ESOS_TASK_END();
}
```

Declare semaphore and allocate storage.

User timer `swTimerLED` same as previous example.

ESOS function tthat returns a random number

Create random value between 255 and 4095.

counter local to `task1`

Print task counter to screen. Signal semaphore at every pass through loop.

Yield for a random period of time.

ESOS task signalling via semaphore (2)

```
ESOS_USER_TASK(task2) {
  static uint8  u8_cnt=0;
  ESOS_TASK_BEGIN();
  while (TRUE) {
    ESOS_TASK_WAIT_SEMAPHORE(sem_T2CanRun, 5);
    ESOS_TASK_WAIT_ON_AVAILABLE_OUT_COMM();
    ESOS_TASK_WAIT_ON_SEND_STRING(psz_T2);
    ESOS_TASK_WAIT_ON_SEND_UINT8_AS_HEX_STRING(u8_cnt);
    ESOS_TASK_WAIT_ON_SEND_STRING( psz_CRNL );
    ESOS_TASK_SIGNAL_AVAILABLE_OUT_COMM();
    u8_cnt++;
  }
  ESOS_TASK_END();
}
```

counter local to `task2`

Yield until `task1` has signaled five times.

Print task counter to screen.

ESOS task sync via semaphore

Global variables same as previous example, but add:

```
char psz_rv[] = "rendez-vous!";
ESOS_SEMAPHORE(sem_T1CanRun);
```

`user_init()` same as previous example, but add:

```
ESOS_INIT_SEMAPHORE(sem_T1CanRun, 0);
```

Timer `swTimerLED` and function `getRandomDelay()` same as previous example.

```
ESOS_USER_TASK(task1) {
  static uint8  u8_cnt=0;
  ESOS_TASK_BEGIN();
  while (u8_cnt < 10) {
    ESOS_TASK_WAIT_ON_AVAILABLE_OUT_COMM();
    ESOS_TASK_WAIT_ON_SEND_STRING(psz_T1);
    ESOS_TASK_WAIT_ON_SEND_UINT8_AS_HEX_STRING(u8_cnt);
    ESOS_TASK_WAIT_ON_SEND_STRING(psz_CRNL);
    ESOS_TASK_SIGNAL_AVAILABLE_OUT_COMM();
    u8_cnt++;
    ESOS_TASK_WAIT_TICKS(getRandomDelay());
  }
  ESOS_SIGNAL_SEMAPHORE(sem_T2CanRun, 1);
  ESOS_TASK_WAIT_SEMAPHORE(sem_T1CanRun, 1);
  ESOS_TASK_WAIT_ON_SEND_STRING(psz_T1);
  ESOS_TASK_WAIT_ON_SEND_STRING(psz_rv);
  ESOS_TASK_WAIT_ON_SEND_STRING(psz_CRNL);
  ESOS_TASK_END();
}
```

10 times: Print task counter to screen and yield for a random period of time.

Signal arrival at rendez-vous point and wait for `task2`.

Print message that `task1` passed rendez-vous point.

```
ESOS_USER_TASK(task2) {
  static uint8  u8_cnt=0;
  ESOS_TASK_BEGIN();
  while (u8_cnt<10) {
    ESOS_TASK_WAIT_ON_AVAILABLE_OUT_COMM();
    ESOS_TASK_WAIT_ON_SEND_STRING(psz_T2);
    ESOS_TASK_WAIT_ON_SEND_UINT8_AS_HEX_STRING(u8_cnt);
    ESOS_TASK_WAIT_ON_SEND_STRING(psz_CRNL);
    ESOS_TASK_SIGNAL_AVAILABLE_OUT_COMM();
    u8_cnt++;
    ESOS_TASK_WAIT_TICKS(getRandomDelay());
  }
  ESOS_SIGNAL_SEMAPHORE(sem_T1CanRun, 1);
  ESOS_TASK_WAIT_SEMAPHORE(sem_T2CanRun, 1);
  ESOS_TASK_WAIT_ON_SEND_STRING(psz_T2);
  ESOS_TASK_WAIT_ON_SEND_STRING(psz_rv);
  ESOS_TASK_WAIT_ON_SEND_STRING(psz_CRNL);
  ESOS_TASK_END();
}
```

10 times: Print task counter to screen and yield for a random period of time.

Signal arrival at rendez-vous point and wait for `task1`.

Print message that `task2` passed rendez-vous point.

ESOS user flags service

`esos_SetUserFlag(flag)`
Sets a bit in the user flag denoted by **`flag`**. ESOS provides 16 user flags named **`ESOS_USER_FLAG_0`**, **`ESOS_USER_FLAG_1`**, ..., **`ESOS_USER_FLAG_F`**. The **`flag`** input can be the OR of several flag names to set bits in more than one flag simultaneously .
`esos_ClearUserFlag(flag)`
Clears a bit in the user flag denoted by **`flag`**. ESOS provides 16 user flags named **`ESOS_USER_FLAG_0`**, **`ESOS_USER_FLAG_1`**, ..., **`ESOS_USER_FLAG_F`**. The **`flag`** input can be the OR of several flag names to clear bits in more than one flag simultaneously .
`esos_IsUserFlagSet(flag)`
Queries whether a bit in the user flag denoted by **`flag`** is set. ESOS provides 16 user flags named **`ESOS_USER_FLAG_0`**, **`ESOS_USER_FLAG_1`**, ..., **`ESOS_USER_FLAG_F`**. Returns **`TRUE`** if the bit is set; returns **`FALSE`** otherwise.
`esos_IsUserFlagClear(flag)`
Queries whether a bit in the user flag denoted by **`flag`** is clear. ESOS provides 16 user flags named **`ESOS_USER_FLAG_0`**, **`ESOS_USER_FLAG_1`**, ..., **`ESOS_USER_FLAG_F`**. Returns **`TRUE`** if the bit is clear; returns **`FALSE`** otherwise.

ESOS child tasks (1)

Timer `swTimerLED` same as previous example.

```
// Global variables
char psz_CRNL[3]= {0x0D, 0x0A, 0};
char psz_prompt[] = "Enter number 0-9 for echo increment:  ";
char psz_done[9]= {' ','D','O','N','E','!',0x0D, 0x0A, 0};

void user_init(void) {
  __esos_unsafe_PutString( HELLO_MSG );
  CONFIG_LED1();
  esos_RegisterTask(prompter);
  esos_RegisterTimer(swTimerLED, 250);
}

ESOS_USER_TASK(prompter) {
  static uint8              u8_char;
  static ESOS_TASK_HANDLE   th_child;

  ESOS_TASK_BEGIN();
  while(TRUE) {
    ESOS_TASK_WAIT_ON_AVAILABLE_OUT_COMM();
    ESOS_TASK_WAIT_ON_SEND_STRING(psz_prompt);
    ESOS_TASK_WAIT_ON_AVAILABLE_IN_COMM();
    do {
      ESOS_TASK_WAIT_ON_GET_UINT8(u8_char);
    } while((u8_char < '0') | (u8_char > '9'));
    ESOS_TASK_SIGNAL_AVAILABLE_IN_COMM();
    ESOS_TASK_WAIT_ON_SEND_STRING(psz_CRNL);
    ESOS_TASK_SIGNAL_AVAILABLE_OUT_COMM();
    ESOS_ALLOCATE_CHILD_TASK(th_child);
    ESOS_TASK_SPAWN_AND_WAIT(th_child, echo_child, u8_char-'0');
  }
  ESOS_TASK_END();
}
```

Register the "parent" task with ESOS.

Declare storage for the handle to the "child" task `echo_child`.

Prompt the user to type a number 0-9. `prompter` will not procced until it gets a number 0-9.

Have ESOS allocate a child task and initialize handle `th_child`.

`prompter` spawns `echo_child` with arguments. `prompter` is blocked until `echo_child` ends.

ESOS child tasks (1)

Child tasks are declared differently than "normal" user tasks.

```
ESOS_CHILD_TASK(echo_child, uint8 u8_in){
  static uint8  u8_char;
  ESOS_TASK_BEGIN();
  do {
    ESOS_TASK_WAIT_ON_AVAILABLE_IN_COMM();
    ESOS_TASK_WAIT_ON_GET_UINT8(u8_char);
    ESOS_TASK_SIGNAL_AVAILABLE_IN_COMM();
    ESOS_TASK_WAIT_ON_AVAILABLE_OUT_COMM();
    ESOS_TASK_WAIT_ON_SEND_UINT8(u8_char+u8_in);
    ESOS_TASK_SIGNAL_AVAILABLE_OUT_COMM();
  } while (u8_char != '!');
  ESOS_TASK_WAIT_ON_AVAILABLE_OUT_COMM();
  ESOS_TASK_WAIT_ON_SEND_STRING(psz_done);
  ESOS_TASK_SIGNAL_AVAILABLE_OUT_COMM();
  ESOS_TASK_END();
}
```

Child tasks can accept arguments from parents.

Echo incremented characters until user types an '!'.

Print message before ending.

A "child" task *must* eventually end, or the "parent" task will be blocked forever.

ESOS interrupts service

`ESOS_USER_INTERRUPT(desc)`
Declares an user-provided routine to service the interrupt denoted by interrupt descriptor **`desc`**. Interrupt descriptors exist for all interrupts.
`ESOS_REGISTER_PIC24_USER_INTERRUPT(desc, ipl, p2f)`
Registers function **`p2f`** as the ISR for the interrupt denoted by **`desc`**, where the ISR has an interrupt priority level of **`ipl`**. ESOS provides 4 priority levels (highest-to-lowest priority): **`ESOS_USER_IRQ_LEVEL1`**, **`ESOS_USER_IRQ_LEVEL2`**, **`ESOS_USER_IRQ_LEVEL3`**, **`ESOS_USER_IRQ_LEVEL4`**. Functions **`p2f`** should be declared as type **`ESOS_USER_INTERRUPT()`**. Registered interrupts are not enabled.
`ESOS_ENABLE_PIC24_USER_INTERRUPT(desc)`
Enables the declared and registered interrupt dentoed by **`desc`**.
`ESOS_DISABLE_PIC24_USER_INTERRUPT(desc)`
Disables the interrupt dentoed by **`desc`**.
`ESOS_ENABLE_ALL_PIC24_USER_INTERRUPTS()`
Enables all declared and registered user interrupts. Does not affect interrupts used by ESOS.
`ESOS_DISABLE_ALL_PIC24_USER_INTERRUPTS()`
Disables all declared and registered user interrupts. Does not affect interrupts used by ESOS.
`ESOS_IS_PIC24_USER_INTERRUPT_ENABLED(desc)`
Queries the enabled state of the interrupt dentoed by **`desc`**. Returns TRUE if the interrupt is registered and enabled; returns FALSE otherwise.
`ESOS_DOES_PIC24_USER_INTERRUPT_NEED_SERVICING(desc)`
Queries the status flag state of the interrupt dentoed by **`desc`**. Returns TRUE if the interrupt is needs servicing; returns FALSE otherwise. Used most often when polling a peripheral with enabling the peripheral interrupt.
`ESOS_MARK_PIC24_USER_INTERRUPT_SERVICED(desc)`
Clears the status flag state of the interrupt dentoed by **`desc`**. Usually called within the **`ESOS_USER_INTERRUPT`** routine for the interrupt dentoed by **`desc`**.

ESOS user interrupts (1)

```
#include    "esos.h"
#include    "esos_pic24.h"
#include    "esos_pic24_rs232.h"
#include    "pic24_timer.h"

#define CONFIG_LED1()    CONFIG_RB15_AS_DIG_OUTPUT()
#define LED1             _LATB15
#define WAITING_FOR_FALLING_EDGE    ESOS_USER_FLAG_0
#define CAPTURED_FLAG               ESOS_USER_FLAG_1

char psz_CRNL[3]= {0x0D, 0x0A, 0};
char psz_prompt[] = "Press button...";
char psz_r1[] = "Pulse width = ";
char psz_r2[] = "us\n";

volatile UINT32 U32_lastCapture;
volatile UINT32 U32_thisCapture;
volatile int32 u32_delta;

inline void CONFIG_SW1()  {
  CONFIG_RB13_AS_DIG_INPUT();
  ENABLE_RB13_PULLUP();
  CONFIG_INT1_TO_RP(13);
}
```

ESOS user flags to signal between ISR and `main()`

`UINT32` is an union datatype defined in ESOS file `all_generic.h`.

`CONFIG_SW1` from Figure 12.5 with interrupt configuration removed. ESOS interrupt service used to configure interrupts.

User timer `swTimerLED` same as previous example.

```
void  configTimer23(void) {
  T2CON = T2_OFF | T2_IDLE_CON | T2_GATE_OFF
          | T2_32BIT_MODE_ON
          | T2_SOURCE_INT
          | T2_PS_1_1 ;
  PR2 = 0xFFFF;          //maximum period
  PR3 = 0xFFFF;          //maximum period
  TMR3HLD = 0;           //write MSW first
  TMR2  = 0;             //then LSW
  ESOS_MARK_PIC24_USER_INTERRUPT_SERVICED(ESOS_IRQ_PIC24_T3);
  T2CONbits.TON = 1;  //turn on the timer
}
```

`configTimer23()` from Figure 12.5 modified only to use ESOS function to clear `T3IF`.

ESOS user interrupts (2)

```
ESOS_USER_INTERRUPT(ESOS_IRQ_PIC24_INT1) {
    ESOS_MARK_PIC24_USER_INTERRUPT_SERVICED(ESOS_IRQ_PIC24_INT1);
    if (esos_IsUserFlagSet(WAITING_FOR_FALLING_EDGE)) {
        if (esos_IsUserFlagClear(CAPTURED_FLAG)) {
            U32_lastCapture.u16LoWord = TMR2;
            U32_lastCapture.u16HiWord = TMR3HLD;
            _INT1EP = 0;
            esos_ClearUserFlag(WAITING_FOR_FALLING_EDGE);
        }
    } else {
        U32_thisCapture.u16LoWord = TMR2;
        U32_thisCapture.u16HiWord = TMR3HLD;
        u32_delta = U32_thisCapture.u32 -
                    U32_lastCapture.u32;
        esos_SetUserFlag(CAPTURED_FLAG);
        _INT1EP = 1;
        esos_SetUserFlag(WAITING_FOR_FALLING_EDGE);
    }
} //end ESOS_IRQ_PIC24_INT1

ESOS_USER_TASK(task1) {
  static  uint32          u32_pulseWidth;

  ESOS_TASK_BEGIN();
  while (TRUE) {
    ESOS_TASK_WAIT_ON_SEND_STRING(psz_prompt);
    ESOS_TASK_WAIT_UNTIL(esos_IsUserFlagSet(CAPTURED_FLAG));
    u32_pulseWidth = ticksToUs(u32_delta, getTimerPrescale(T2CONbits));
    esos_ClearUserFlag(CAPTURED_FLAG);
    ESOS_TASK_WAIT_ON_SEND_STRING(psz_r1);
    ESOS_TASK_WAIT_ON_SEND_UINT32_AS_HEX_STRING(u32_pulseWidth);
    ESOS_TASK_WAIT_ON_SEND_STRING(psz_r2);
  }
  ESOS_TASK_END();
} // end task1()

void user_init(void) {
  CONFIG_LED1(); CONFIG_SW1(); configTimer23();

  esos_RegisterTask(task1);
  esos_RegisterTimer( swTimerLED, 250 );

  esos_SetUserFlag(WAITING_FOR_FALLING_EDGE);
  _INT1EP = 1;
  ESOS_REGISTER_PIC24_USER_INTERRUPT( ESOS_IRQ_PIC24_INT1,
                  ESOS_USER_IRQ_LEVEL1, _INT1Interrupt );
  ESOS_ENABLE_PIC24_USER_INTERRUPT(ESOS_IRQ_PIC24_INT1);
}
```

Clear interrupt bit.

If `task1` is ready for another capture, record the timer value, and change edge sensitivity.

Capture is complete. Record timer value, compute `u32_delta`, change edge, and signal `task1` via `CAPTURED_FLAG`.

Yield until ISR has signaled success via `CAPTURED_FLAG`.

Let ISR know `task1` used `u32_delta`.

Setup hardware and timer.

Initialize ISR state flag `WAITING_FOR_FALLING_EDGE`.

Register ISR with ESOS and have ESOS enable INT1.

Configure INT1 for falling edges.

ESOS I2C service
Fundamental I2C operations

```
#define __PIC24_I2C1_START()                          \
   do{                                                \
      I2C1CONbits.SEN = 1;                            \
      ESOS_TASK_WAIT_WHILE(I2C1CONbits.SEN);          \
   }while(0)

#define __PIC24_I2C1_RSTART()                         \
  do{                                                 \
    I2C1CONbits.RSEN = 1;                             \
    ESOS_TASK_WAIT_WHILE(I2C1CONbits.RSEN);           \
  }while(0)

#define __PIC24_I2C1_STOP()                           \
  do{                                                 \
    I2C1CONbits.PEN = 1;                              \
    ESOS_TASK_WAIT_WHILE(I2C1CONbits.PEN);            \
  }while(0)

#define __PIC24_I2C1_PUT(byte)                                \
  do{                                                         \
    I2C1TRN = (byte);                                         \
    ESOS_TASK_WAIT_WHILE(I2C1STATbits.TRSTAT);                \
  }while(0)
```

Keep multi-line macros atomic by "protecting" them with `do-while(0)`.

I2C transactions are child tasks ESOS I2C write tranactions (1)

```
ESOS_CHILD_TASK(__esos_pic24_write1I2C1, uint8 u8_addr, uint8 u8_d1) {

  static uint8 u8_tempAddr, u8_tempD1;

  ESOS_TASK_BEGIN();
  u8_tempAddr=u8_addr; u8_tempD1=u8_d1;
  __PIC24_I2C1_START();
  __PIC24_I2C1_PUT(I2C_WADDR(u8_tempAddr));
  __PIC24_I2C1_PUT(u8_tempD1);
  __PIC24_I2C1_STOP();
  ESOS_TASK_END();
}

ESOS_CHILD_TASK(__esos_pic24_write2I2C1, uint8 u8_addr,
                                uint8 u8_d1, uint8 u8_d2) {
  static uint8 u8_tempAddr, u8_tempD1, u8_tempD2;

  ESOS_TASK_BEGIN();
  u8_tempAddr=u8_addr; u8_tempD1=u8_d1; u8_tempD2=u8_d2;
  __PIC24_I2C1_START();
  __PIC24_I2C1_PUT(I2C_WADDR(u8_tempAddr));
  __PIC24_I2C1_PUT(u8_tempD1);
  __PIC24_I2C1_PUT(u8_tempD2);
    PIC24_I2C1_STOP();
```

local copy of arguments

macro to make target address into I2C write address

local copy of arguments

ESOS I2C write transactions (2)

```
ESOS_CHILD_TASK(__esos_pic24_writeNI2C1, uint8 u8_addr,
                                uint8* pu8_d, uint16 u16_cnt) {
  static uint8   u8_tempAddr;
  static uint8*  pu8_tempPtr;
  static uint16  u16_tempCnt, u16_i;

  ESOS_TASK_BEGIN();
  u8_tempAddr=u8_addr; pu8_tempPtr=pu8_d; u16_tempCnt=u16_cnt;
  __PIC24_I2C1_START();
  __PIC24_I2C1_PUT(I2C_WADDR(u8_tempAddr));
  for (u16_i=0; u16_i < u16_tempCnt; u16_i++) {
    __PIC24_I2C1_PUT(*pu8_tempPtr);
    pu8_tempPtr++;
  }
  __PIC24_I2C1_STOP();
  ESOS_TASK_END();
}
```

local copy of arguments

ESOS I2C write transactions (3)

```
#define ESOS_TASK_WAIT_ON_WRITE1I2C1(u8_addr,u8_d1)                \
  ESOS_TASK_SPAWN_AND_WAIT((ESOS_TASK_HANDLE)&__stChildTaskI2C,  \
  __esos_pic24_write1I2C1, (u8_addr), (u8_d1) )

#define ESOS_TASK_WAIT_ON_WRITE2I2C1(u8_addr,u8_d1,u8_d2)          \
  ESOS_TASK_SPAWN_AND_WAIT((ESOS_TASK_HANDLE)&__stChildTaskI2C,  \
  __esos_pic24_write2I2C1, (u8_addr), (u8_d1), (u8_d2) )

#define ESOS_TASK_WAIT_ON_WRITENI2C1(u8_addr,pu8_d,u16_cnt)        \
  ESOS_TASK_SPAWN_AND_WAIT((ESOS_TASK_HANDLE)&__stChildTaskI2C,  \
  __esos_pic24_writeNI2C1, (u8_addr), (pu8_d), (u16_cnt) )
```

I2C transactions are child tasks ESOS I2C read transactions (1)

```
ESOS_CHILD_TASK( __esos_pic24_getI2C1,
                 uint8* pu8_x,
                 uint8 u8_ack2Send) {
  static uint8* pu8_local;
  static uint8 u8_local;

  ESOS_TASK_BEGIN();
  pu8_local = pu8_x;
  u8_local = u8_ack2Send;
  ESOS_TASK_WAIT_WHILE(I2C1CON & 0x1F);
  I2C1CONbits.RCEN = 1;
  ESOS_TASK_WAIT_UNTIL(I2C1STATbits.RBF);
  *pu8_local = I2C1RCV;
  ESOS_TASK_WAIT_WHILE(I2C1CON & 0x1F);
  I2C1CONbits.ACKDT = u8_local;
  I2C1CONbits.ACKEN = 1;
  ESOS_TASK_WAIT_WHILE(I2C1CONbits.ACKEN);
  ESOS_TASK_END();
}
```

desination for received byte

local copy of arguments to preserve across waits

Wait for idle.

Wait for receive.

Save byte at destination's address.

Wait for idle before attempting ACK.

Wait for ACK to complete.

ESOS I2C read transactions (2)

```
ESOS_CHILD_TASK( __esos_pic24_read1I2C1, uint8 u8_addr, uint8* pu8_d) {
  static uint8         u8_tempAddr;
  static uint8*        pu8_tempD1;

  ESOS_TASK_BEGIN();
  u8_tempAddr=u8_addr; pu8_tempD1=pu8_d;
  __PIC24_I2C1_START();
  __PIC24_I2C1_PUT(I2C_RADDR(u8_tempAddr));
  ESOS_TASK_WAIT_ON_GETI2C1(pu8_tempD1, I2C_NAK);
  __PIC24_I2C1_STOP();
  ESOS_TASK_END();
}

ESOS_CHILD_TASK( __esos_pic24_read2I2C1, uint8 u8_addr,
                          uint8* pu8_d1, uint8* pu8_d2) {
  static uint8         u8_tempAddr;
  static uint8*        pu8_tempD1;
  static uint8*        pu8_tempD2;

  ESOS_TASK_BEGIN();
  u8_tempAddr=u8_addr; pu8_tempD1=pu8_d1; pu8_tempD2=pu8_d2;
  __PIC24_I2C1_START();
  __PIC24_I2C1_PUT(I2C_RADDR(u8_tempAddr));
  ESOS_TASK_WAIT_ON_GETI2C1(pu8_tempD1, I2C_ACK);
  ESOS_TASK_WAIT_ON_GETI2C1(pu8_tempD2, I2C_NAK);
  __PIC24_I2C1_STOP();
  ESOS_TASK_END();
}
```

Create local copy of arguments.

macro to make target address into I^2C read address

Create local copy of arguments.

ESOS I2C read transactions (3)

```
ESOS_CHILD_TASK( __esos_pic24_readNI2C1, uint8 u8_addr,
                            uint8* pu8_d, uint16 u16_cnt) {
  static uint8         u8_tempAddr;
  static uint8*        pu8_tempD;
  static uint16        u16_tempCnt, u16_i;

  ESOS_TASK_BEGIN();
  u8_tempAddr=u8_addr; pu8_tempD=pu8_d; u16_tempCnt=u16_cnt;
  __PIC24_I2C1_START();
  __PIC24_I2C1_PUT(I2C_RADDR(u8_tempAddr));
  for (u16_i=0; u16_i < u16_tempCnt-1; u16_i++) {
    ESOS_TASK_WAIT_ON_GETI2C1(pu8_tempD, I2C_ACK);
    pu8_tempD++;
  }
  ESOS_TASK_WAIT_ON_GETI2C1(pu8_tempD, I2C_NAK);
  __PIC24_I2C1_STOP();
  ESOS_TASK_END();
}
```

Create local copy of arguments.

ESOS I2C read transactions (4)

```
#define ESOS_TASK_WAIT_ON_READ1I2C1(u8_addr,u8_d1)                  \
  ESOS_TASK_SPAWN_AND_WAIT((ESOS_TASK_HANDLE)&__stChildTaskI2C,  \
  __esos_pic24_read1I2C1, (u8_addr), &(u8_d1) )

#define ESOS_TASK_WAIT_ON_READ2I2C1(u8_addr,u8_d1,u8_d2)          \
  ESOS_TASK_SPAWN_AND_WAIT((ESOS_TASK_HANDLE)&__stChildTaskI2C,  \
  __esos_pic24_read2I2C1, (u8_addr), &(u8_d1), &(u8_d2) )

#define ESOS_TASK_WAIT_ON_READNI2C1(u8_addr,pu8_d,u16_cnt)        \
  ESOS_TASK_SPAWN_AND_WAIT((ESOS_TASK_HANDLE)&__stChildTaskI2C,  \
  __esos_pic24_readNI2C1, (u8_addr), (pu8_d), (u16_cnt) )
```

ESOS I2C App

```
#include    "esos_pic24.h"
#include    "esos_pic24_rs232.h"
#include    "esos_pic24_i2c.h"
#include    <stdio.h>
```

This application uses `printf` for convenience.

```
#define DS1631ADDR      0x90
#define ACCESS_CONFIG   0xAC
#define CONFIG_COMMAND  0x0C
#define START_CONVERT   0x51
#define READ_TEMP       0xAA

int16 i16_temp;
ESOS_SEMAPHORE(sem_dataReady);
ESOS_SEMAPHORE(sem_dataPrinted);
ESOS_SEMAPHORE(sem_ds1631Ready);
```

continuous 12bit conversion

Declare/allocate the semaphores our tasks will use.

Timer `swTimerLED` same as previous example.

```
void user_init(void) {
  CONFIG_LED1();
  esos_pic24_configI2C1(400);

  ESOS_INIT_SEMAPHORE(sem_ds1631Ready, 0);
  ESOS_INIT_SEMAPHORE(sem_dataReady, 0);
  ESOS_INIT_SEMAPHORE(sem_dataPrinted, 0);

  esos_RegisterTask(start_ds1631);
  esos_RegisterTask(read_ds1631);
  esos_RegisterTask(update);

  esos_RegisterTimer( swTimerLED, 250);
```

400bps I^2C operation

Init the semaphores our tasks use to synchronize.

Register our tasks with the ESOS scheduler.

ESOS I2C app (2)

```
ESOS_USER_TASK(start_ds1631) {
  ESOS_TASK_BEGIN();
  ESOS_TASK_WAIT_TICKS(500);
  ESOS_TASK_WAIT_ON_WRITE2I2C1(DS1631ADDR, ACCESS_CONFIG, CONFIG_COMMAND);
  ESOS_TASK_WAIT_ON_WRITE1I2C1(DS1631ADDR, START_CONVERT);
  ESOS_TASK_WAIT_TICKS(500);
  ESOS_SIGNAL_SEMAPHORE(sem_ds1631Ready, 1);
  ESOS_TASK_END();
}

ESOS_USER_TASK(read_ds1631) {
  static uint8 u8_lo, u8_hi;

  ESOS_TASK_BEGIN();
  ESOS_TASK_WAIT_SEMAPHORE(sem_ds1631Ready, 1);
  while (TRUE) {
    ESOS_TASK_WAIT_ON_WRITE1I2C1(DS1631ADDR, READ_TEMP);
    ESOS_TASK_WAIT_ON_READ2I2C1(DS1631ADDR, u8_hi, u8_lo);
    i16_temp = u8_hi;
    i16_temp = ((i16_temp<<8)|u8_lo);
    ESOS_SIGNAL_SEMAPHORE(sem_dataReady, 1);
    ESOS_TASK_WAIT_TICKS(750);
    ESOS_TASK_WAIT_SEMAPHORE(sem_dataPrinted, 1);
  }
  ESOS_TASK_END();
}

ESOS_USER_TASK(update) {
  float   f_tempC, f_tempF;

  ESOS_TASK_BEGIN();
  while (TRUE) {
    ESOS_TASK_WAIT_SEMAPHORE(sem_dataReady, 1);
    f_tempC = (float) i16_temp;
    f_tempC = f_tempC/256;
    f_tempF = f_tempC*9/5 + 32;
    printf("Temp is: 0x%0X, %4.4f (C), %4.4f (F)\n",
        i16_temp, (double) f_tempC, (double) f_tempF);
    ESOS_SIGNAL_SEMAPHORE(sem_dataPrinted, 1);
  }
  ESOS_TASK_END();
}
```

Give DS1631 time to convert first reading.

Signal `read_ds1631` to begin.

Signal `update` to print data.

Make sure `update` used data.

Do not procced until data is ready.

Application uses `printf` for convenience.

Signal `read_ds1631` that `update` consumed recent data.