ECE 4723/6723 Embedded Systems

Lecture

Embedded Systems Operating System (ESOS)

Part 2

Reading:

Coding conventions

- "Code is read much more often than it is written."

  -- *Guido van Rossum*

- Team-written code should not be distinguishable from code from one person
  - Your southern accent is cool. Be proud!
  - But, code is no place for your "accent" to show through
- Conventions mean less time trying to figure out what someone else is doing.
  - In industry, repeated violations of conventions will get you fired!

Coding conventions

- ECE 4723/6723 will use coding conventions for all code
  - ECE 4723/6723 C language coding conventions
    - Available at the class website
  - Python PEP 8
    - Available at www.python.org
- Consider adopting a "code-beautifier" in your tool-flow
  - Astyle, PythonTidy, PyLint, etc.

Conventions are not about stifling your creativity. They are all about increasing your productivity!

Don't write *C* code like this

Syntactically correct (& functioning) code is not necessarily good code

```
main(int argc, char** argv)
  {
  while (*argv != argv[1] && (*argv = argv[1])
          && (argc = 0) || (*++argv && (**argv
          && ((++argc)[*argv] &&
          (**argv <= argc[*argv] ||
          (**argv += argc[*argv] -=
          **argv = argc[*argv] - **argv))
          && --argv || putchar(**argv) &&
          ++*argv--) || putchar(10))))
                    ;
  }
```

C language variable names

Prefix	***Data Type***	***Prefix***	***Data Type***
b	boolean	st	structure
u8	unsigned 8-bit integer	sz	zero-terminated string
u16	unsigned 16-bit integer	fn	function
u32	unsigned 32-bit integer	*Composite prefix examples*	
i8	signed 8-bit integer	pu16	pointer to an unsigned 16-bit integer
i16	signed 16-bit integer	ai8	array of signed 8-bit integers
i32	signed 32-bit integer	pfn	pointer to a function
f	floating point	pst	pointer to a structure
d	double precision float	psz	pointer to a zero-terminated string
p	pointer	apfn	array of “function pointers'”
a	array	apai8	array of pointers to byte arrays

All C language variables should be named

`prefix_somethingDescriptiveAndUseful`

C language variable names

- b_busy
  - boolean, or bit
- u8_numOfApples
  - unsigned 8-bit (byte) count of items
- u32_lightYears
  - unsigned 32-bit value
- i16_temperature
  - signed 16-bit value
- f_priceOfTeaInChina
  - floating-point value
- d_pi
  - double precision floating-point value
- sz_lastName
  - zero-terminated string
- u32_identifier
  - unsigned 32-bit integer
- st_time
  - clock time structure
- fn_fourthRoot
  - function name
- psz_owner
  - pointer to zero-terminated string
- af_accountBalances
  - array of floating-point values
- pst_bufferDescriptor
  - pointer to a structure
- apst_activeTasks
  - array of pointers of structures

ESOS threads

- ESOS "threading" is based on protothreads
  - ANSI C code from Adam Dunkels (*www.sics.se*)
- Protothreads are a mixture of the event-driven and "true" threads
  - stackless, non-preemptive threading
  - can be driven by an event-handler
  - With protothreads, we can write blocking waits, inside an event-handler
    - ESOS_TASK_WAIT_UNTIL() – conditional blocking

ESOS thread-based implementation

```
ESOS_USER_TASK( radio_wake_thread ) {
  ESOS_TASK_BEGIN();
  while( TRUE ) {
1   radio_on();
    timer_set(&st_timer, T_AWAKE);
2   ESOS_TASK_WAIT_UNTIL( timer_expired(&st_timer) );
    timer_set(&st_timer, T_SLEEP);
    if(!communication_complete()) {
3, 4  ESOS_TASK_WAIT_UNTIL(
          communication_complete() ||
    timer_expired(&st_timer))
    }
5   if(!timer_expired(&st_timer)) {
      radio_off();
      ESOS_TASK_WAIT_UNTIL( timer_expired(&st_timer) );
6   }
  }
  ESOS_TASK_END();
}
```

Limitations on ESOS "threads"

- Automatic variables (stack variables) not saved across a blocking wait
  - Limitation inherited from the event-driven model
  - Programmer must manually save automatic variables
  - However, static local variables still work as expected

You <u>must</u> "`static`" ALL variables you want to preserve across any task yielding/waiting statement!!!!!

ESOS "thread" functions (1)

- Blocking and yielding
  - ESOS_TASK_YIELD()
  - ESOS_TASK_WAIT_UNTIL(condition)
  - ESOS_TASK_WAIT_WHILE(cond)
  - ESOS_TASK_WAIT_TICKS(u32_duration)
  - ESOS_TASK_WAIT_SEMAPHORE(pstSem,i16_val)
  - ESOS_TASK_WAIT_THREAD(pfnThread,...)
  - ESOS_TASK_SPAWN_AND_WAIT(pstChild, pfnChild,...)
- Thread life cycle management
  - ESOS_TASK_SLEEP(pstTask)
  - ESOS_TASK_WAKE(pstTask)
  - ESOS_TASK_KILL(pstTask)
  - ESOS_TASK_RESTART(pstTask)
  - ESOS_TASK_EXIT(pstTask)

ESOS "thread" functions (2)

- Child tasks
  - ESOS_TASK_WAIT_THREAD(pstTask, pfnThread,...)
  - ESOS_TASK_SPAWN_AND_WAIT(pstTask, pstChild, pfnChild,...)
  - ESOS_CHILD_TASK_ALLOCATE(pstName)
- Semaphore task functions
  - ESOS_TASK_SEM_INIT(pstSem, u8Val)
  - ESOS_TASK_SEM_WAIT(pstTask, pstSem)
  - ESOS_TASK_SEM_SIGNAL(pstTask, pstSem)

ESOS services: Thread (Task) Management

- pstTask esos_RegisterTask(pfn)
  - *returns* a pstTask
  - registers the task pfn with ESOS and places it in the ESOS scheduler
  - task pfn will run at next opportunity
- uint8 esos_UnregisterTask(pfn)
  - ends the task and remove it from ESOS task scheduler
  - return TRUE is successful, FALSE otherwise
- uint8 esos_GetNumberRegisteredTasks()
  - returns number of tasks currently registered with ESOS system
  - tasks may not be "running" but yielded/waiting

ESOS services

- ESOS provides services BEYOND task mgmt
  - system clock (called the "tick")
  - communications system
    - UART, I2C, SPI
  - user interrupts
  - "soft-timers"

  - Others services under development
    - CAN
    - PWM
    - servo & motor control
    - generic analog sensor (ADC)

Other ESOS services – ESOS Timers

- uint32 esos_GetSystemTick()
  - *returns a uint32 corresponding to the number of <u>MILLISECONDS</u> since the ESOS system started*
- ESOS_TMR_HANDLE esos_RegisterTimer (pfn, uint32 u32_period)
  - *registers function pfn as an ESOS timer function with period u32_period (in ticks)*
- uint8 esos_UnregisterTimer (ESOS_TMR_HANDLE)
  - *removes the timer from the timer service*
- ESOS_TMR_HANDLE esos_GetTimerHandle (pfn)
  - *gets the timer handle based on the pfn*
- uint8 esos_ChangeTimerPeriod (ESOS_TMR_HANDLE, uint32 u32_period)
  - *changes the running timer's period*

Other ESOS services – User Flags

- esos_SetUserFlag(uint16 u16_mask)
  - sets the user-defined flag denoted by u16_mask
- esos_ClearUserFlag(uint16 u16_mask)
  - clears the user-defined flag denoted by u16_mask
- esos_IsUserFlagSet(uint16 u16_mask)
  - returns TRUE if the user flag u16_mask is *SET*
- esos_IsUserFlagClear(uint16 u16_mask)
  - returns TRUE if the user flag u16_mask is *CLEAR*
- PROVIDED user flag masks
  - ESOS_USER_FLAG_0, ESOS_USER_FLAG_1, ..., ESOS_USER_FLAG_E, ESOS_USER_FLAG_F
  - user can/should #define their own names to map to these masks. Don't #define your own constant values!

Other ESOS services – User Interrupts (1)

- esos_RegisterUserInterrupt(uint16 u16_irqHandle, uint8 u8_pLevel, pfn)
  - registers user function pfn as ISR for u16_irqHandle at priority level u8_pLevel
  - IRQ handles given in esos_p24_irq.h
    - ESOS_IRQ_PIC24_T2, ESOS_IRQ_PIC24_T3, ESOS_IRQ_PIC24_AD1, ESOS_IRQ_PIC24_I2C1, etc.
- esos_UnregisterUserInterrupt(uint16 u16_irqHandle)
  - unregisters the ISR from the ESOS system

Do <u>NOT</u> manipulate IRQ hardware enables, flags, or priorities directly! ESOS provides access for you!

Other ESOS services – User Interrupts (2)

- esos_EnableUserInterrupt(u16_irqHnd)
  - enables the IRQ denoted by the handle
  - user should have registered an ISR already
- esos_DisableUserInterrupt(u16_irqHnd)
  - disables the IRQ denoted by the handle
- esos_DisableAllUserInterrupts()
  - a global IRQ disable for the user interrupts
    - NOTE: *HW IRQs will continue if ESOS uses them internally*
- esos_EnableAllUserInterrupts()
  - a global IRQ enable for the user interrupts

Do NOT manipulate IRQ hardware enables, flags, or priorities directly! ESOS provides access for you!

Other ESOS services – User Interrupts (3)

- esos_IsUserInterruptEnabled(u16_irqHnd)
  - checks user IRQ enabled status
- esos_DoesUserInterruptNeedServicing (u16_irqHnd)
  - returns TRUE if the IRQ u8IrqIndex is requesting service
    - NOTE: *You will not use this serice as often as you think*
- esos_MarkUserInterruptServiced (u16_irqHnd)
  - marks user IRQ as being serviced

Do NOT manipulate IRQ hardware enables, flags, or priorities directly! ESOS provides access for you!

Other ESOS services – Communications System (1)

- Fairly complete set of comm API methods
- Communications is inherent blocking
  - Thus, communications are done in child tasks
- ESOS has two comm streams: *IN* & *OUT*
  - NOTE: *streams named from ESOS's viewpoint*
- API:
  - ESOS_TASK_WAIT_ON_AVAILABLE_x_COMM()
    - blocks until the communications stream is available for your task's use
  - ESOS_TASK_SIGNAL_BUSY_x_COMM()
    - notify other tasks that you now control the comm stream
  - ESOS_TASK_RELEASE_x_COMM()
    - notify other tasks that the comm stream is available

Other ESOS services – Communications System (2)

- ESOS_TASK_WAIT_ON_GET_UINT8(u8_in)
  - reads 8 bits from "in" stream into u8_in
- ESOS_TASK_WAIT_ON_GET_U8BUFFER(pau8_in, u8_size)
  - reads up to 256 bytes into pau8_in
- ESOS_TASK_WAIT_ON_SEND_UINT8(u8_out)
  - puts a byte into the "out" stream
- ESOS_TASK_WAIT_ON_SEND_UINT8_AS_HEX_STRING(u8_out)
  - puts byte to "out" stream as a human-readable hex
- ESOS_TASK_WAIT_ON_SEND_U8BUFFER(pu8_out, u8_size)
  - sends up to 256 bytes of pu8_out to "out" stream

Other ESOS services – Communications System (3)

- ESOS_TASK_WAIT_ON_SEND_UINT32_AS_HEX_STRING(u32_out)
  - sends a 32-bit number to "out" as a hex string
- ESOS_TASK_WAIT_ON_SEND_STRING(psz_out)
  - sends a sz to "out" stream

***** <u>WARNING</u> ** <u>WARNING</u> ** <u>WARNING</u> ** <u>WARNING</u> *****
Only one task can access the "stream" at a time, if you want your data to come out FIFO.

You can easily write other child tasks to facilitate common communications activities!

Other ESOS services – Communications System (4)

- Other helpful ESOS communications functions/macros:
  - GET_ESOS_COMM_IN_DATA_LEN()
  - IS_ESOS_COMM_GOT_IN_DATA()
  - IS_ESOS_COMM_GOT_EXACTLY_DATA_BYTES(x)
  - IS_ESOS_COMM_GOT_AT_LEAST_DATA_BYTES(x)
  - FLUSH_ESOS_COMM_IN_DATA()
  - IS_ESOS_COMM_READY_OUT_DATA()

Never forget that ESOS communications are child tasks.

They will <u>block their parent</u> while they accomplish their function!

FYI: ESOS execution timeline IRQs can/will occur at anytime!

init ESOS structures/starts ESOS system tick

init ESOS communications system

ESOS will call user_init()

(The user *MUST* provide this function, and it must "register" *at least one* user task!)

ESOS's while(1) loop

Execute/call registered user tasks

(Order of execution is "random" and user should make *no* assumptions.)

ESOS does some user task housekeeping

user_init()

```
void user_init(void) {
  // config our GPIO to the LEDs
  CONFIG_LED1(); LED1 = 1;  // config our LED (set direction/"on")
  CONFIG_LED2(); LED2 = 1;  // config our LED (set direction/"on")

  T2CON = T2_IDLE_CON + T2_PS_1_256 + T2_SOURCE_INT;
  PR2 = MS_TO_TICKS(500, 256);     // 500 ms interrupt interval
  TMR2  = 0;                       // clear T2's count
  T2CONbits.TON = 1;               // turn on the timer

  esos_RegisterUserInterrupt( ESOS_IRQ_PIC24_T2, ESOS_USER_IRQ_LEVEL4,
   _T2Interrupt );

  esos_RegisterTask( blink_LED2 );
  esos_RegisterTask( upper_case );
  esos_EnableUserInterrupt( ESOS_IRQ_PIC24_T2 );

} // end user_init()
```

_T2Interrupt()

```
#define CONFIG_LED1()        CONFIG_RB15_AS_DIG_OUTPUT()
#define LED1                 _LATB15
#define CONFIG_LED2()        CONFIG_RB0_AS_DIG_OUTPUT()
#define LED2                 _LATB0


void _ISR _T2Interrupt (void) {
  LED1 = !LED1;
  esos_MarkUserInterruptServiced( ESOS_IRQ_PIC24_T2 );
}
```

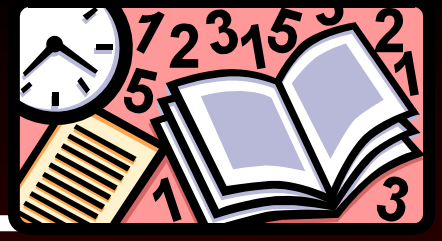
blink_LED2() task

```
ESOS_USER_TASK( blink_LED2 ) {

  ESOS_TASK_BEGIN( );
  while (TRUE) {
    ESOS_TASK_WAIT_TICKS( 500 );
    LED2 = !LED2;
  } //end while(TRUE)
  ESOS_TASK_END( );
} // end Blink_LED2()
```

upper_case() task

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

  ESOS_TASK_BEGIN( );
  while (TRUE) {
    ESOS_TASK_WAIT_ON_AVAILABLE_IN_COMM();
    ESOS_TASK_SIGNAL_BUSY_IN_COMM();
    ESOS_TASK_WAIT_ON_GET_UINT8( u8_char );
    ESOS_TASK_RELEASE_IN_COMM();
    if ((u8_char >= 'a') && (u8_char <= 'z') )
        u8_char = u8_char - 'a' + 'A';
    ESOS_TASK_WAIT_ON_AVAILABLE_OUT_COMM();
    ESOS_TASK_SIGNAL_BUSY_OUT_COMM();
    ESOS_TASK_WAIT_ON_SEND_UINT8( u8_char);
    ESOS_TASK_RELEASE_OUT_COMM();
  } // endof while(TRUE)
  ESOS_TASK_END( );
} // end upper_case()
```

References



- New PIC24 users:
  - Read Chapters 1-9 in R/B/J
- Read Chapter 14 in R/B/J
  - *You may want to build a few of the examples from Chapter 14.*
- D/L PIC24 libraries and ESOS from bitbucket
- see online ESOS dox @
  - www.ece.msstate.edu/courses/ece3724/
- Read ECE4723 C language coding conventions
- Read Ganssle's Chapters 1-3
  - (if you have the book)