

Introduction

The purpose of this module is to introduce you to the concepts and operations of OSAL and HAL. These basic software tools distance your code from the hardware, making code re-use and debugging among other things, much easier.

Objectives

- Review OSAL components and APIs
- Look at the HAL APIs

*** this page left blank by order of the fire marshal ***

Module Topics

OSAL and HAL.....	6-1
<i>Module Topics.....</i>	<i>6-3</i>
<i>Why Use an OS?</i>	<i>6-5</i>
<i>Tasks</i>	<i>6-6</i>
<i>Events and Messaging.....</i>	<i>6-8</i>
<i>Scheduling.....</i>	<i>6-9</i>
<i>Callback and Interrupts.....</i>	<i>6-10</i>
<i>Timers, Clock and Memory.....</i>	<i>6-11</i>
<i>NV Memory and Power.....</i>	<i>6-12</i>
<i>HAL API's.....</i>	<i>6-13</i>

*** a blank page without a smart-aleck comment. Oh, wait ... ***

Why Use an OS?

Why Use an OS?

- ◆ Separates S/W development from H/W development
- ◆ OSAL supports:
 - ◆ Task registration, initialization and starting
 - ◆ Task synchronization
 - ◆ Message exchange between tasks
 - ◆ Interrupt handling
 - ◆ Timers
 - ◆ Memory allocation
 - ◆ Power management

OSAL/HAL ...

2

OSAL / HAL

OSAL provides scheduling, memory management and messaging features.

HAL provides easy programming access to hardware and isolates the software from the hardware specifics

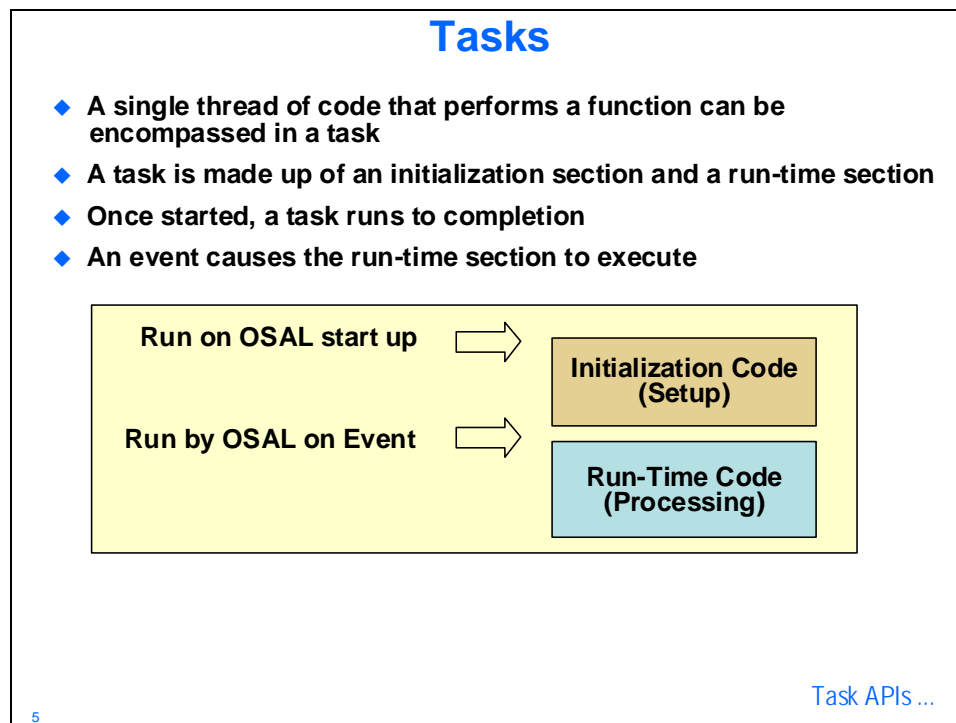
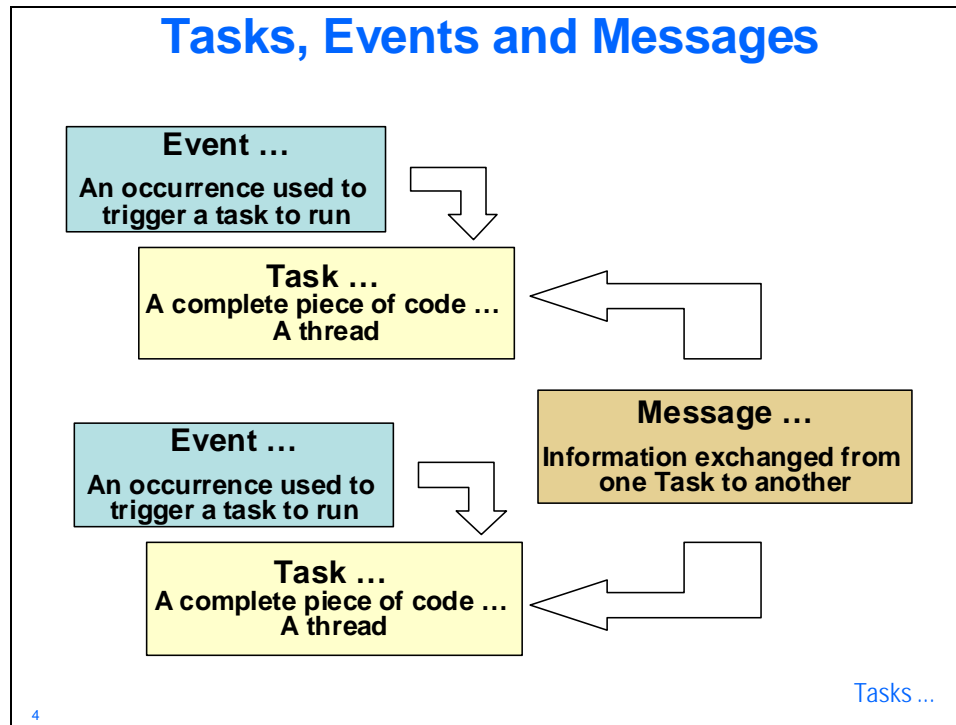
Because of its very limited functionality, the OSAL is not, strictly speaking, considered an Operating System

OSAL = Operating System Abstraction Layer
HAL = Hardware Abstraction Layer

Tasks, Events and Messages ...

3

Tasks



Task APIs

`osal_set_event()`

sets the event flags for a task – runs task

`osal_init_system()`

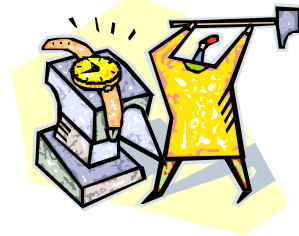
creates the tasks defined in the task table

`osal_start_system()`

starts the OSAL main loop

`osal_self()`

returns the ID of the current task



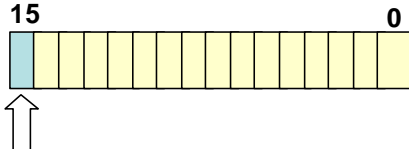
Events ...

Events and Messaging

Events

Definition: An action to be completed by a Task.

event_flag, used for defining the type of event, is 16 bits long



SYS_EVENT_MSG (Reserved for OSAL)

- A special event type used for inter-task communication

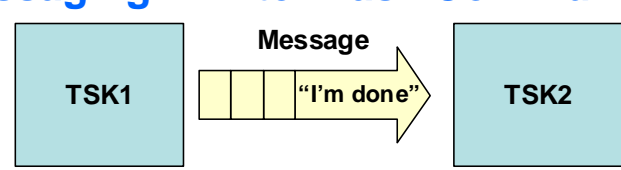
Event other bit position is managed by the application

```
#define TRANSMITAPP_SEND_MSG_EVT    0x0001
#define TRANSMITAPP_RCVTIMER_EVT    0x0002
#define TRANSMITAPP_SEND_ERR_EVT    0x0004
```

Messaging ...

7

Messaging ... Inter-Task Communication



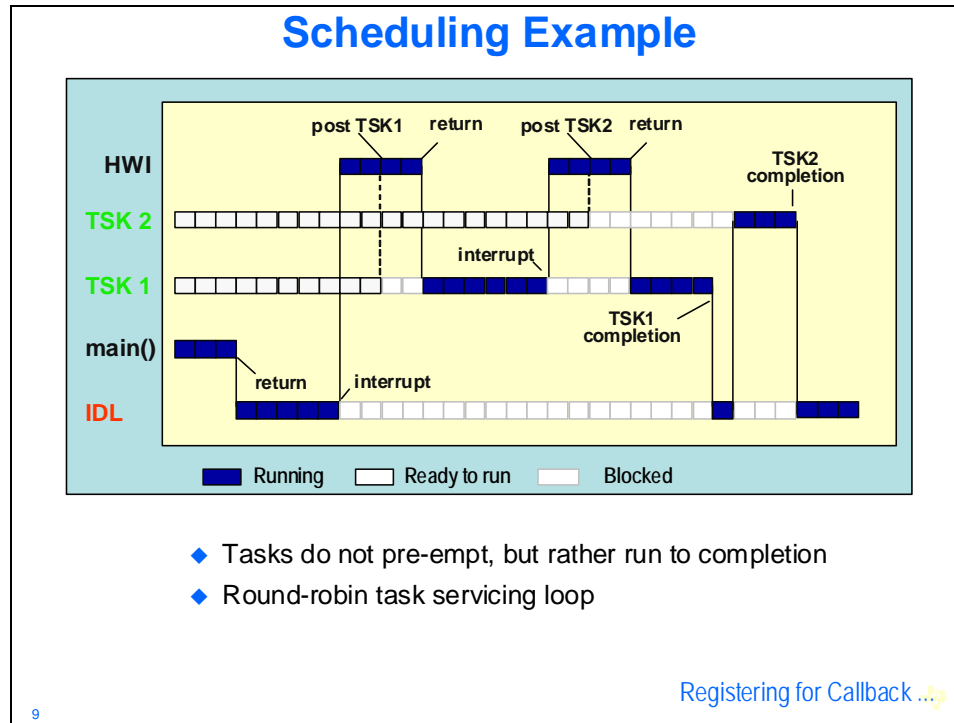
- ◆ Managed by the OSAL
- ◆ Messaging used for triggering over the air communication as well as internal command messaging between tasks.
- ◆ Commands between tasks use `osal_msg_send()` to create **SYS_EVENT_MSG** events

```
osal_msg_allocate()
    allocates a buffer for message transfer between tasks
osal_msg_deallocate()
    deallocates buffer
osal_msg_send()
    place a message in the buffer
osal_msg_receive()
```

Scheduling ...

8

Scheduling



Callback and Interrupts

Registering For Callback

- ◆ The Callback mechanism is natively built into the stack (and isn't strictly an OSAL feature)
- ◆ Callbacks allow "information" to flow from lower layer stack code up to dependent modules
- ◆ Callbacks are manifested through SYS_EVENT_MSG events
- ◆ Registration takes place through an API that is task (module) dependent
- ◆ Example callbacks include
 - ◆ Receiving Messages (de-multiplexing)
 - ◆ Key Events (HAL specific)
 - ◆ Binding Management Messages
 - ◆ Other Management Message Notification
 - ◆ Receiving MT messages & commands



```
// Register for all key events - This app will handle all key events
RegisterForKeys( GenericApp_TaskID );
CODE for handling it is in process event.
case KEY_CHANGE:
    GenericApp_HandleKeys( ((keyChange_t *)MSGpkt)->state, ((keyChange_t *)MSGpkt)->keys );
    break;
```

Interrupts ...

10

Interrupts

- ◆ This API enables a task to interface with external interrupts.
- ◆ The functions in the API allow a task to associate a specific service routine with each interrupt.
- ◆ Inside the service routine, events may be set for other tasks.

```
osal_int_enable()
    enable identified interrupt
osal_int_disable()
    disable identified interrupt
```



Timers and Clock ...

11

Timers, Clock and Memory

Timers and Clock

osal_start_timer()
start a timeout period (mS) and trigger event

osal_start_timerEx()
start a timeout period for a specific task

osal_stop_timer()
stop indicated timer

osal_GetSystemClock()
reads the system clock



osal_setClock()
initializes the devices real-time clock

osal_getClock()
retrieves the time

osal_ConvertUTCTime()
time in seconds since 0:0:0 on 1 Jan 2000 UTC



- ◆ You must use OSAL_CLOCK compiler flag to use the OSAL clock

Memory ...

12

Memory APIs

- ◆ **Standard C malloc() and free() operations have little knowledge of hardware specifics**
- ◆ **OSAL provides the same and extended functions for the user to manage dynamic memory from the heap**
- ◆ **These functions should be used exclusively**

osal_mem_alloc()
dynamically allocates a buffer

osal_mem_free()
returns allocated buffer to heap

- ◆ **Take care to free any memory allocated. This usually takes place after reading the message**
- ◆ **Note that garbage collection is NOT provided**

NV Memory ...

13

NV Memory and Power

Non Volatile Memory

- ◆ Access to persistent data storage (FLASH)
- ◆ Define an identifier for the persistent data structure (ZComDef.h), and then use the provided APIs to read and write to/from this structure
- ◆ On startup, the NV_RESTORE compile option allows restoration of dynamic data (like the distributed short address, neighbor and routing information, security keys, binding information, etc...)

```
osal_nv_item_init()
    init an item in non-volatile memory
osal_nv_read()
    read item
osal_nv_write()
    write item
osal_offsetof()
    calculate memory offset
```

Power ...

14

Power Management

- ◆ Notifies OSAL when it's safe to turn off the receiver, external hardware and put the MCU to sleep
- ◆ The device can be set to always_on or battery power
- ◆ OSAL determines whether to sleep based on both the task and device state
- ◆ Task default state is to **conserve** power
- ◆ You must use **POWER_SAVING** compiler flag to use this feature

```
osal_pwrmgr_state()
    changes or sets the devices power savings mode
osal_pwrmgr_task_state()
    change a task's power state
```



HAL APIs ...

15

HAL API's

HAL APIs

- ◆ **The Hardware Abstraction Layer offers these services:**
 - ◆ ADC
 - ◆ LCD
 - ◆ LED
 - ◆ KEY
 - ◆ SLEEP
 - ◆ TIMER
 - ◆ UART
 - ◆ PA/LNA
- ◆ **Supporting files: hal.h, onboard.c and onboard.h**
- ◆ **See Z-Stack HAL Porting Guide (SWRA199) and Z-Stack HAL Driver API Guide (SWRA193) to port the HAL to your target hardware**



Function Calls ...

16

HAL Function Calls

◆ Initialization Function Calls

Initialize a service and /or setup optional parameters for platform-specific data.

◆ Service Access Function Calls

These function calls can directly access hardware registers to get/set certain value of the hardware (i.e. ADC) or control the hardware components (i.e. LED).

◆ Callback Function Calls

These functions must be implemented by the application and are used to pass events that generated by the hardware (interrupts, counters, timers...) or by polling mechanism (UART poll, Timer poll...) to an upper layer. If these functions execute in the context of the interrupt, they must be efficient and not perform CPU-intensive operations or use critical sections.

◆ Services

HAL drivers provide Timer, GPIO, LEDs, Switched, UART, and ADC service for MAC and upper layers. Not all the features in the service are available in every platform. Features in each service can be configured for different platforms through an initialization function.

ADC ...

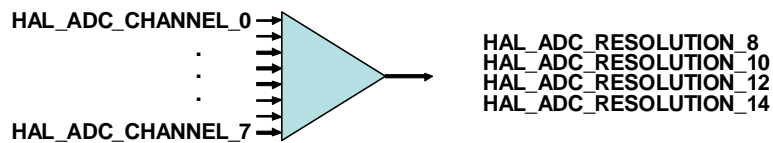
17

HAL ADC API's

- ◆ 8, 10, 12 and 14-bit analog to digital conversion on 8 channels

HalAdcInit()
initializes ADC

HalAdcRead()
reads value from specified channel at specified resolution



18

LCD ...

HAL LCD API's

HalLcdInit()
initializes LCD

HalLcdWriteString()
writes a text string to the LCD

HalLcdWriteValue()
writes a 32-bit value to the LCD

HalLcdWriteScreen()
writes 2 lines of text to the LCD

HalLcdWriteStringValue()
writes a string followed by a 16-bit value to the LCD

HalLcdWriteStringValueValue()
Write two 16-bit values back to back to the LCD

HalLcdDisplayPercentBar()
Simulate a percentage bar on the LCD



19

LED ...

HAL LED API's

HalLedInit()

initializes LEDs

HalLedSet()

sets the given LEDs ON, OFF, BLINK, FLASH or TOGGLE

HalLedBlink()

blinks LEDs based on provided parameters

HalLedGetState()

returns the current state of the LEDs

HalLedEnterSleep()

stores current LED state and turns off LEDs

HalLedExitSleep()

restores pre-sleep state of LEDs

Note: This is a good example of simple GPIO for your target board code.



KEY ...

20

HAL KEY API's

- ◆ Debounced key, switch and joystick service
- ◆ Polling or interrupt driven. Callback available.

HalKeyInit()

initializes keys

HalKeyConfig()

selects either polling (100mS) or interrupt servicing

HalKeyRead()

reads current state based on polling or interrupt

HalKeyEnterSleep()

stops interrupt processing of keys

HalKeyExitSleep()

re-enables interrupt processing of keys



SLEEP ...

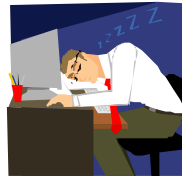
21

HAL Sleep API's

HalSleep()
sets the low power mode of the MAC

HalSleepWait()
performs a blocking wait

- ◆ You must use **POWER_SAVING** compiler flag to use this feature



TIMER ...

22

HAL Timer APIs

- ◆ Each h/w platform places certain timer limitations

HalTimerInit()
initializes timers with specified parameters

HalTimerConfig()
configures channels in different modes

HalTimerStart()
starts the specified timer

HalTimerStop()
stops the specified timer

HalTimerTick()
used for timer polling

HalTimerInterruptEnable()
enables/disables specified timer interrupt



UART ...

23

HAL UART API's

HalUARTInit()
initializes UART

HalUARTOpen()
opens a ports with the specified configuration

HalUARTClose()
closes and turns off UART

HalUARTRead()
reads a buffer from the UART

HalUARTWrite()
writes a buffer to the UART

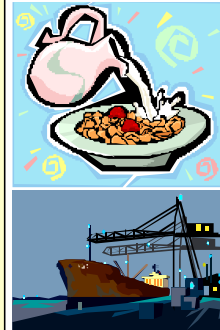
HalUARTIoctl()
performs get/set/flush type operations on a port

HalUARTPoll()
simulates polling the UART

Hal_UART_RxBuffLen()
returns the number of bytes in the Rx buffer

Hal_UART_TxBuffLen()
returns the number of bytes in the Tx buffer

Hal_UART_FlowControlSet()
enables/disables UART flow control



PALNA ...

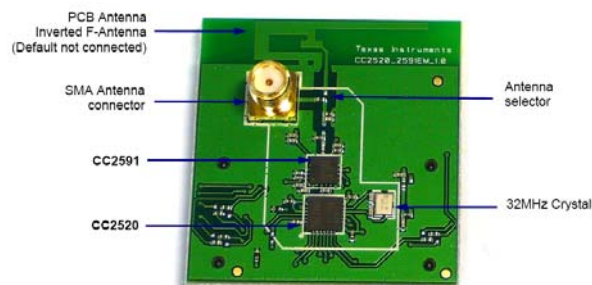
24

HAL PALNA API's

- ◆ Control CC2591 Range Extender Power Amp/ Low Noise Amp functions (if present)

HAL_PA_LAN_RX_LGM()
sets RX low gain mode

HAL_PA_LNA_RX_HGM()
sets RX high gain mode



25

