



# Freescal**MQX™** RTOS

## Introduction

Stanley Huang, Sr. MCU FAE

M A R . 2 0 1 5



External Use

Freescal, the Freescal logo, ARMv6, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, Qorliva, SafeAssure, the SafeAssure logo, StarCore, Symphony and VortiQa are trademarks of Freescal Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Atheros, BeasX, BeasStack, CoreNet, Farris, Layerscape, MagniV, MXC, Platform in a Package, QorIQ Qonverge, QorIQ Engine, Ready Play, SMARTMOS, Tower, TurboLink, UMEMS, Vybrid and Xtrinsic are trademarks of Freescal Semiconductor, Inc. All other product or service names are the property of their respective owners. © 2014 Freescal Semiconductor, Inc.



# Agenda



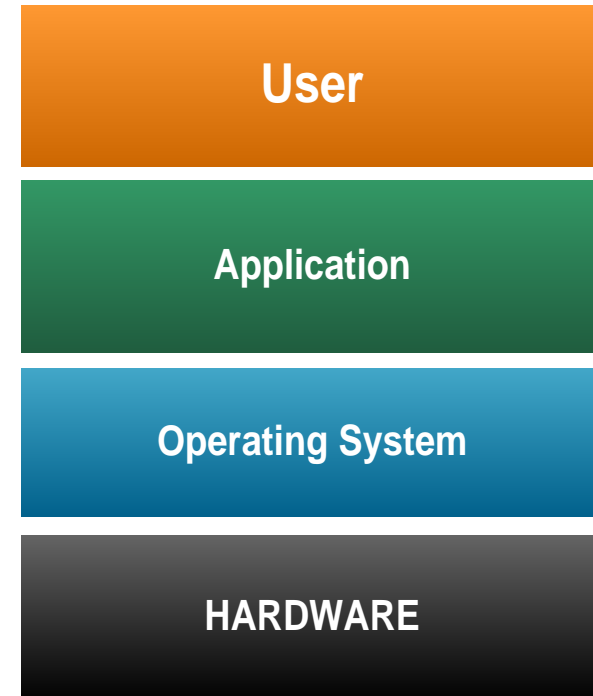
- **What is an RTOS?**
- **MQX Basics: Tasks**
- **MQX Basics: Scheduling**
- **MQX Basics: Task Synchronization**
  - Semaphores
  - Events and Messages
- **MQX Intermediate**
  - Libraries
  - Interrupts
  - BSP
- **Additional Resources**
- **Review**



# What is an RTOS

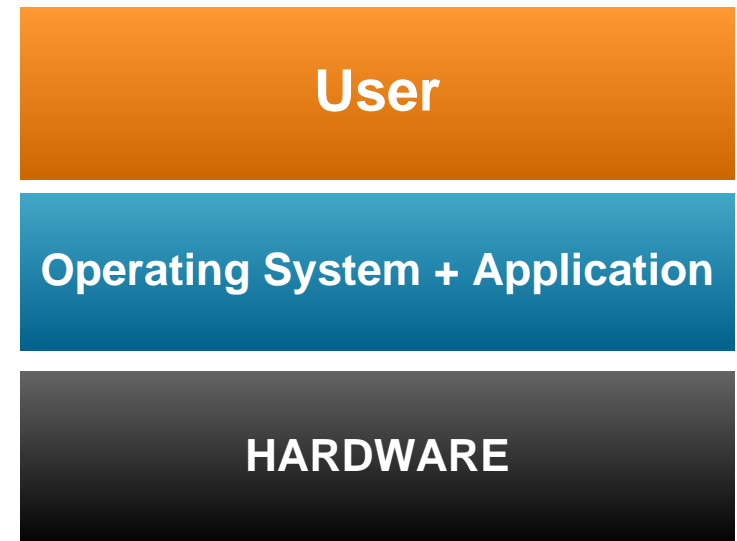
# Operating Systems

- The term “**operating system**” can be used to describe the collection of software that manages a system’s hardware resources
- This software might include a file system module, a GUI and other components
- Often times, a “**kernel**” is understood to be a subset of such a collection
- **Characteristics**
  - ❑ Resource management
  - ❑ Interface between application and hardware
  - ❑ Library of functions for the application



# Real Time Operating Systems

- Fusion of the application and the OS to one unit.
- Code of the OS and the application mostly reside in ROM.
- A real-time operating system (RTOS) manages the time of a microprocessor or microcontroller.
- Features of an RTOS:
  - ❑ Allows multi-tasking
  - ❑ Scheduling of the tasks with priorities
  - ❑ Synchronization of the resource access
  - ❑ Inter-task communication
  - ❑ Time predictable
  - ❑ Interrupt handling



# Why use an RTOS?

- Plan to use drivers that are available with an RTOS
- Would like to spend your time developing application code and not creating or maintaining a scheduling system
- Multi-thread support with synchronization
- Portability of application code to other CPUs
- Resource handling
- Add new features without affecting higher priority functions
- Support for upper layer protocols such as:
  - ❑ TCP/IP, USB, Flash Systems, Web Servers,
  - ❑ CAN protocols, Embedded GUI, SSL, SNMP

# Freescal MQX

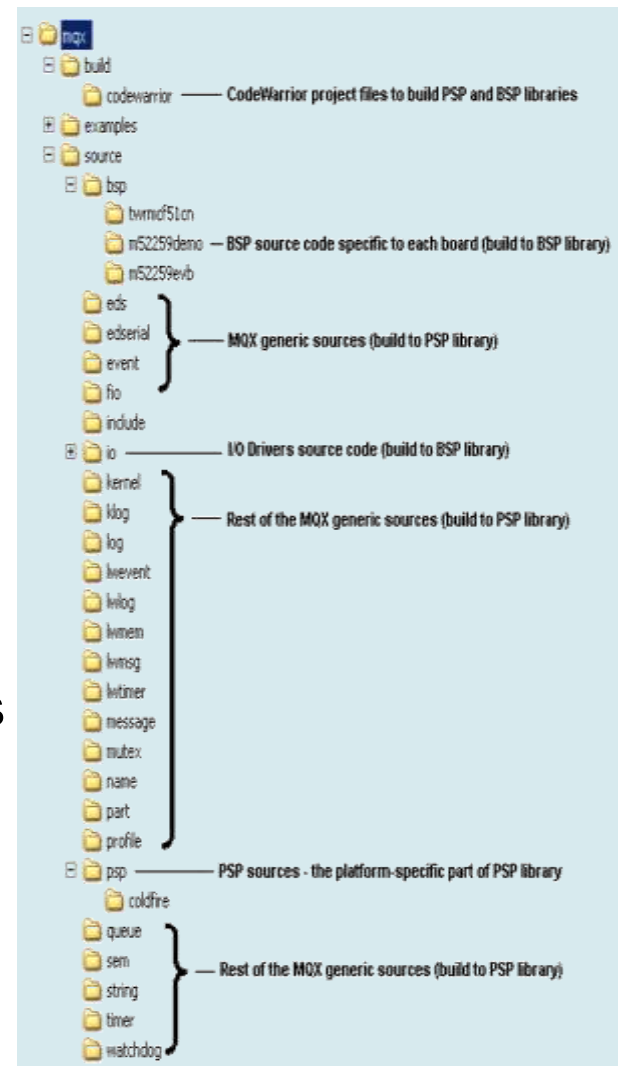
- We will be using Freescale MQX to demonstrate these RTOS concepts.
- Freescale MQX Software can be downloaded:
  - ❑ <http://www.freescale.com/mqx>
- Default Freescale MQX folder:
  - ❑ C:\Freescale\Freescale\_MQX\_4\_0

# MQX Directory Structure

- Described in the MQX Release Notes

- Folders are:

- ☐ config
- ☐ demo
- ☐ doc
- ☐ lib
- ☐ mqx
- ☐ tools
- ☐ And then the RTCS, USB, and MFS stacks

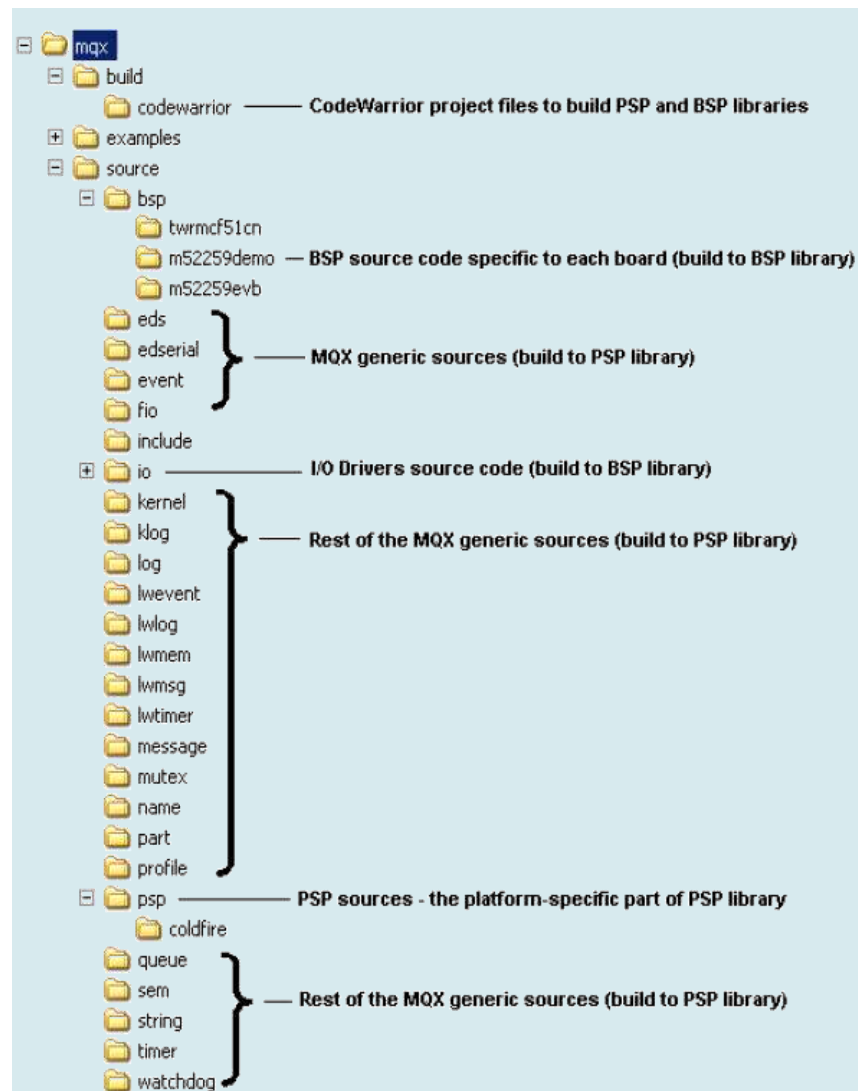


# MQX Directory Structure (Cont.)

- The “mqx” directory is heart of MQX

- **Folders are:**

- ☐ build
- ☐ examples
- ☐ source
  - bsp
  - io
  - psp
  - MQX API source



# Agenda

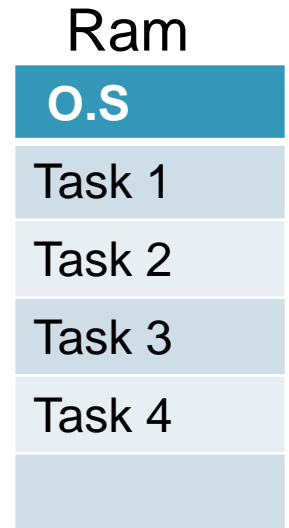
- What is an RTOS?
- ➔ • **MQX Basics: Tasks**
- **MQX Basics: Scheduling**
- **MQX Basics: Task Synchronization**
  - Semaphores
  - Events and Messages
- **MQX Intermediate**
  - Libraries
  - Interrupts
  - BSP
- **Additional Resources**
- **Review**



## MQX Basics: Tasks

# MQX RTOS Tasks

- A system consists of multiple tasks
- Tasks take turns running
- Only one task is active (has the processor) at any given time
- MQX manages how the tasks share the processor (context switching)
- Task Context
  - Data structure stored for each task, including registers and a list of owned resources



# Typical Task Coding Structure

```
void mytask(uint_32 startup_parameter)  {  
    /* Task initialization code */  
    ....  
    while (1) {  
        /* Task body */  
        ....  
        ....  
    }  
}
```



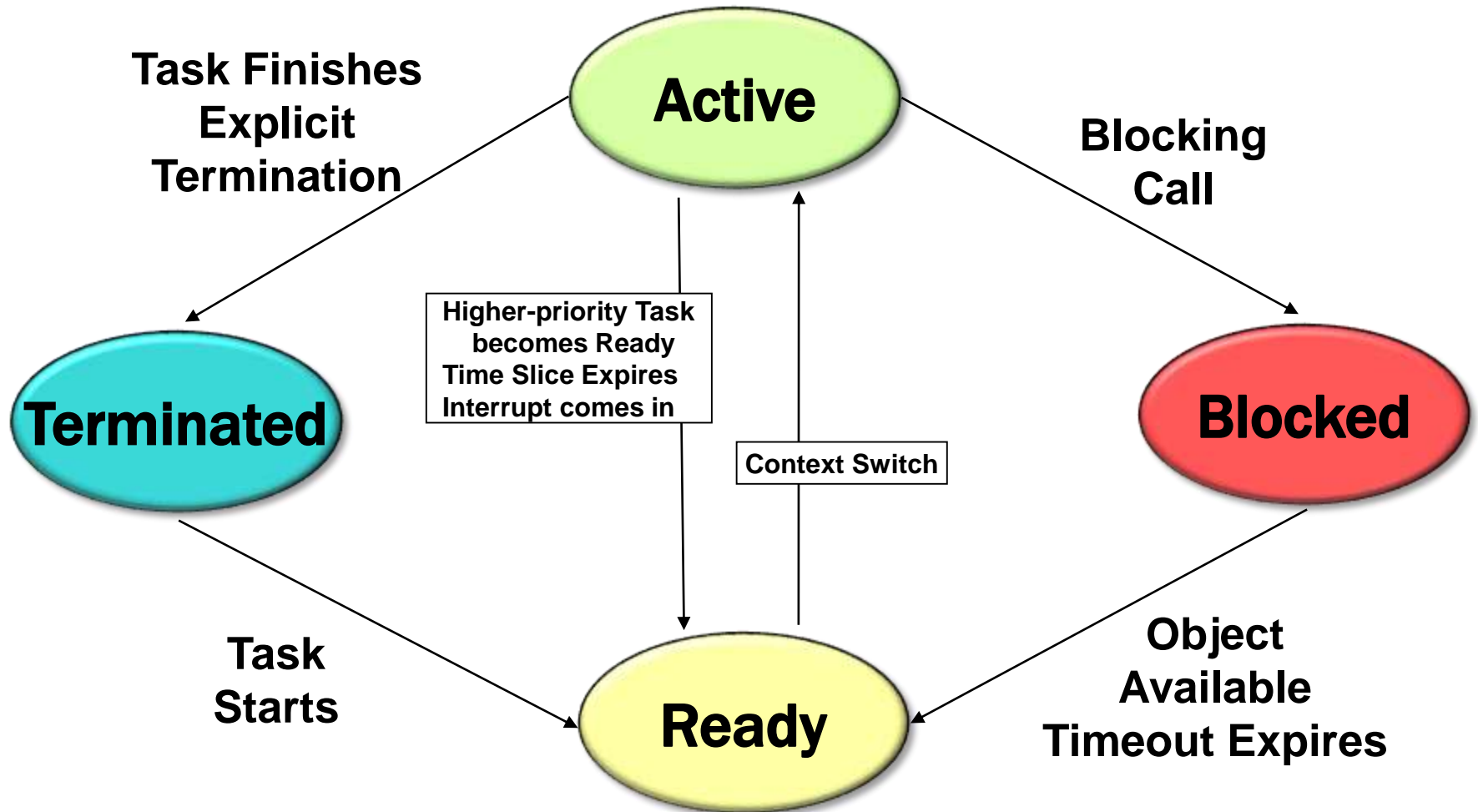
# Task States

- **A task is in one of these logical states:**


- ❑ blocked
  - the task is blocked and therefore not ready
  - it's waiting for a condition to be true
- ❑ active
  - the task is ready and is running because it's the highest-priority ready task
- ❑ ready
  - the task is ready, but it's not running because it isn't the highest-priority ready task
- ❑ terminated
  - the task has finished all its work, or was explicitly destroyed



# Task States



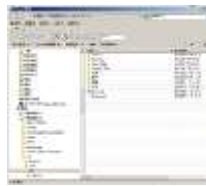
# Agenda

- What is an RTOS?
- MQX Basics: Tasks
-  • MQX Basics: Scheduling
- MQX Basics: Task Synchronization
  - Semaphores
  - Events and Messages
- MQX Intermediate
  - Libraries
  - Interrupts
  - BSP
- Additional Resources
- Review



## MQX Basics: Scheduling

# Priorities



1



2



3



4



5

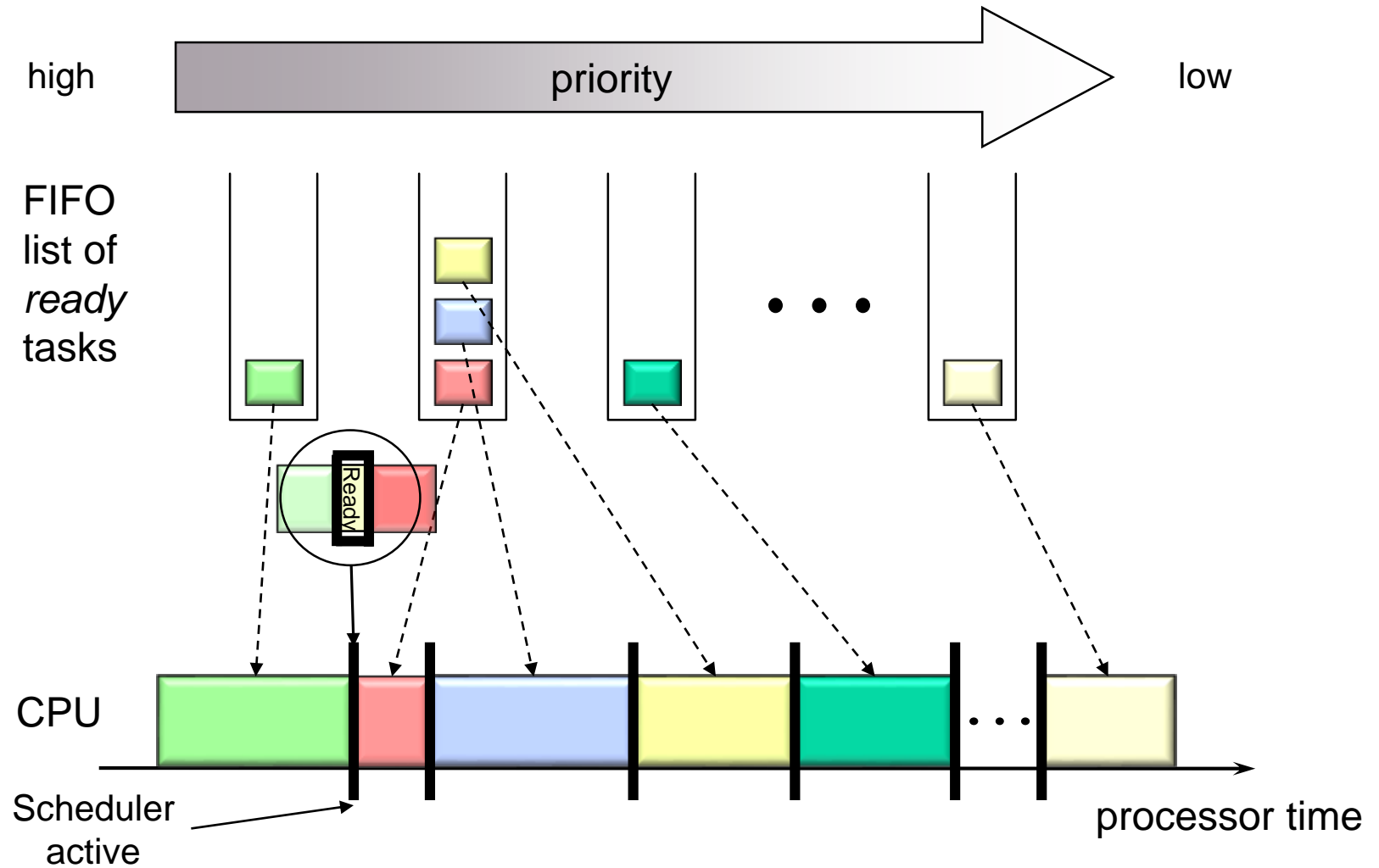
- Priorities run from 0 to N
  - Priority 0 means interrupts disabled, 1 is most important task
- N is set by the highest priority number in the MQX\_Template\_List
  - Idle task runs at N+1
- MQX creates one ready queue for each priority up to the lowest priority (highest number)
  - So must make sure priorities are consecutive
- Able to change priority of a task during runtime
  - `_task_set_priority()`
- Any tasks at priority below 6 means it masks certain levels of interrupts. So user tasks should start at 7 or above.

# Scheduler

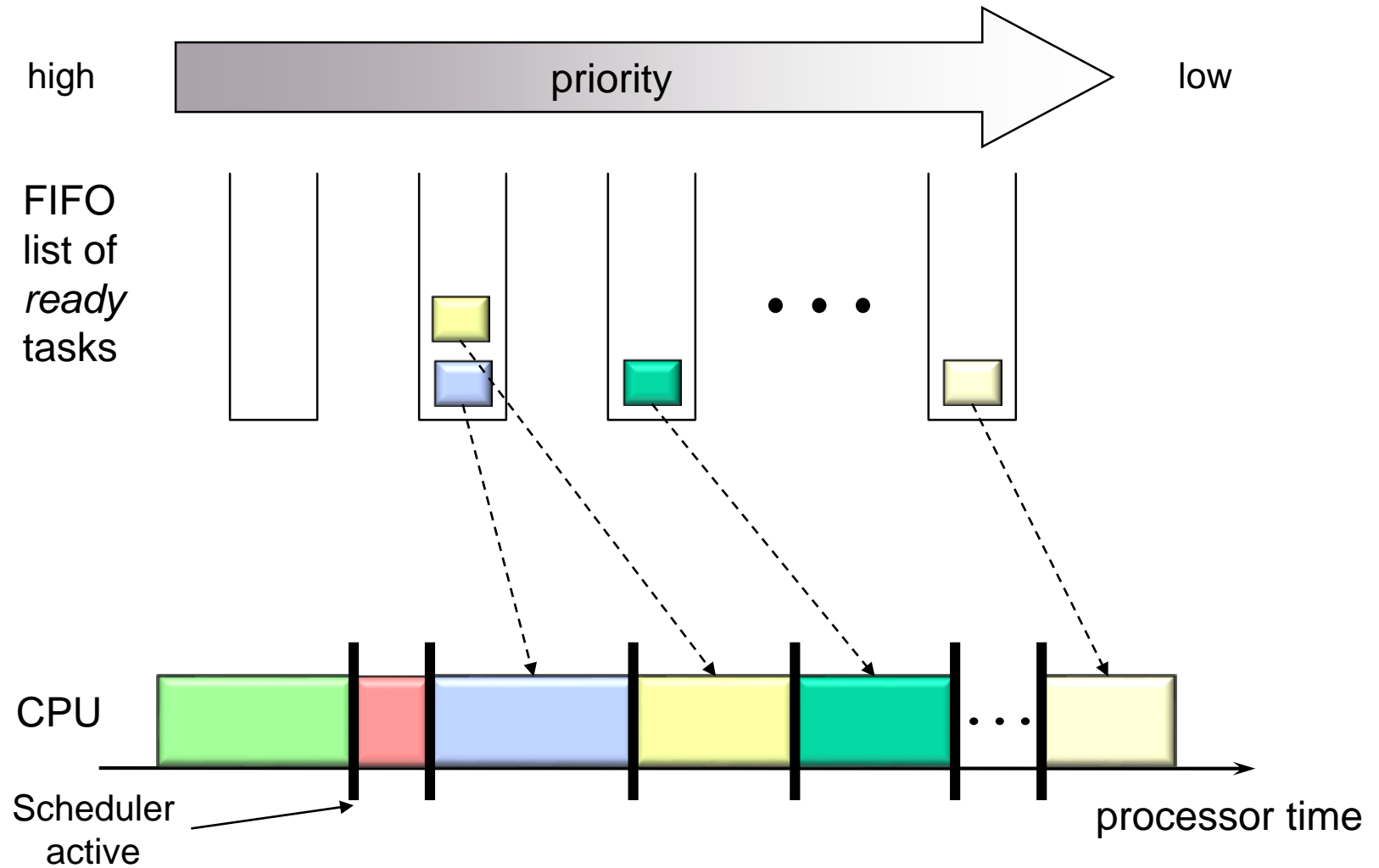
- **Common Scheduling Configurations:**

- ❑ FIFO (also called priority-based preemptive)
  - The active task is the highest-priority task that has been ready the longest
- ❑ Round Robin
  - The active task is the highest-priority task that has been ready the longest without consuming its time slice

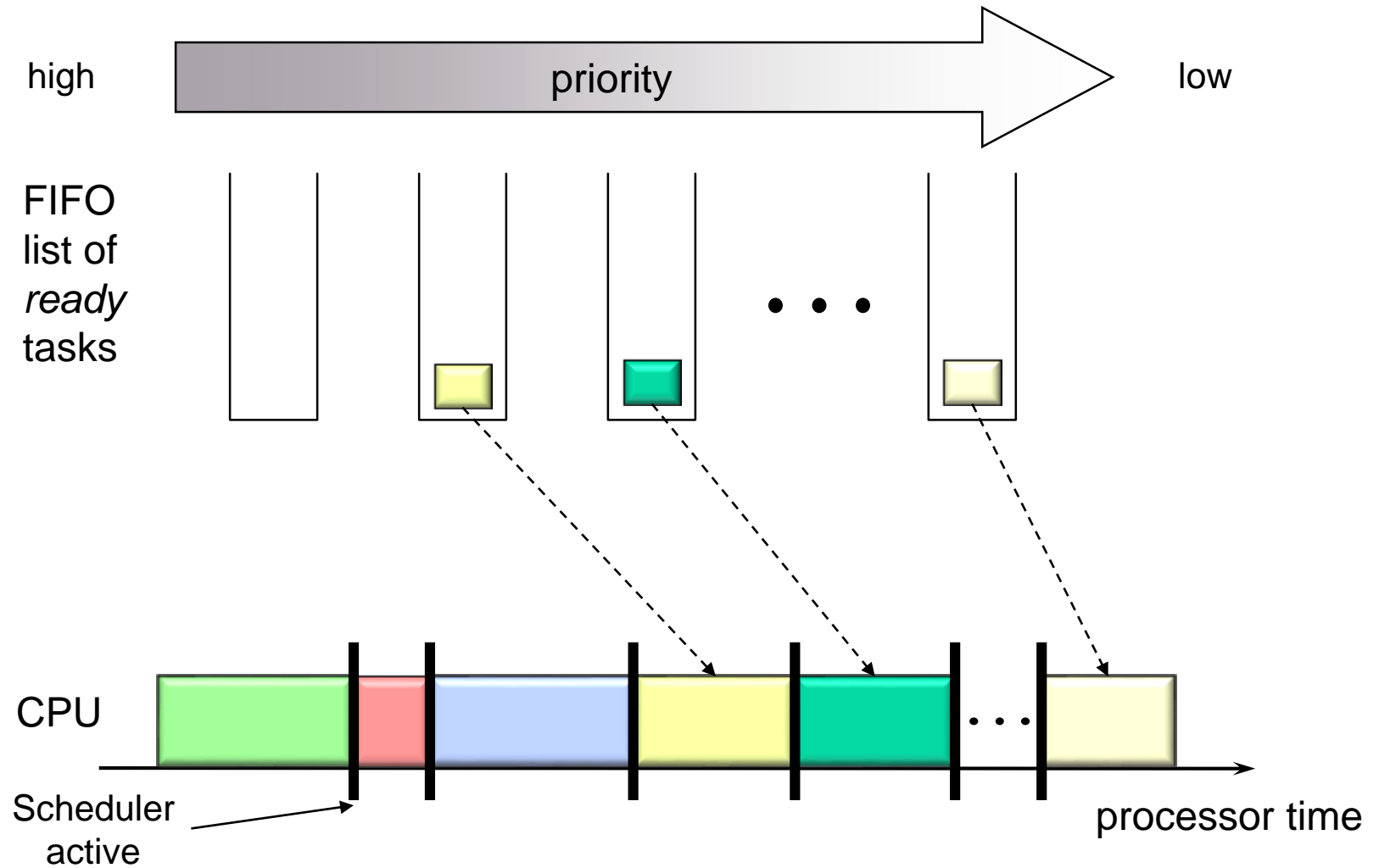
# Priority Based FIFO Scheduling



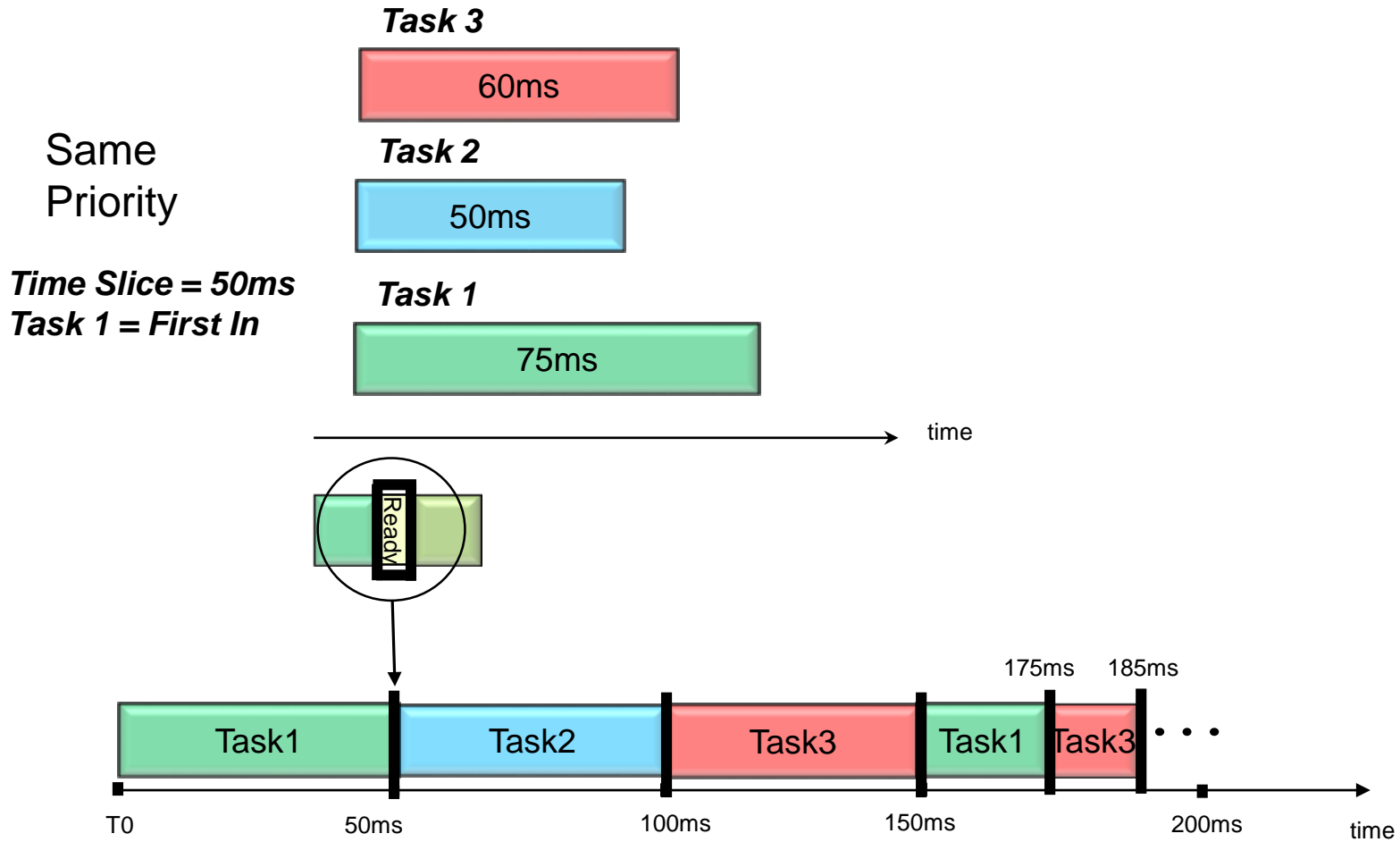
# Priority Based FIFO Scheduling



# Priority Based FIFO Scheduling

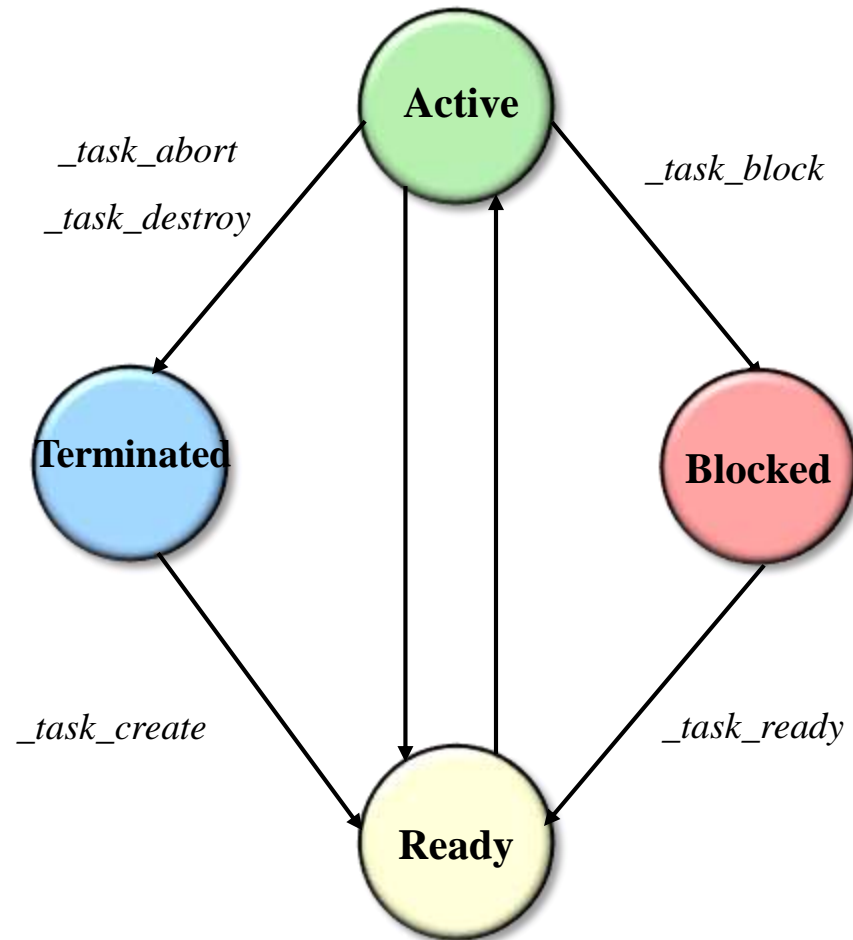


# Round-Robin Scheduling



# MQX Tasks

- Tasks can be automatically created when MQX Starts; also, any task can create another task by calling `_task_create()` or `_task_create_blocked()`
- The function `_task_create()` puts the child task in the ready state and the scheduler puts the higher priority task to run
- If `_task_create_blocked` is used the task is not ready until `_task_ready()` is called



# Creating a Task

- **When creating a task you have to:**

- ❑ Make the task prototype and index definition

```
#define INIT_TASK 5
extern void init_task(uint_32);
```

- ❑ Add the task in the Task Template List

```
TASK_TEMPLATE_STRUCT MQX_template_list[] =
{
    { TASK_INDEX, TASK, STACK, TASK_PRIORITY,
      TASK_NAME, TASK_ATTRIBUTES, CREATION_PARAMETER,
      TIME_SLICE}
}
```

Using the init\_task example:

```
TASK_TEMPLATE_STRUCT MQX_template_list[] =
{
    {INIT_TASK, init_task, 100, 9, "init",
      MQX_AUTO_START_TASK, 0, 0},
}
```

# Creating a Task (Continue)

- TASK\_INDEX: is usually a Define with an index number.
- TASK: Refers to the function name; C compiler takes the address pointer of the function name.
- STACK is the defines stack size.
- TASK\_PRIORITY; the lower number, the higher priority. Task with priority 0 disables all the interrupts ,Priorities 0 to 8 are used by the OS Kernel.
- TASK\_NAME is a string that helps to identify the task. It is also used to get the task ID.
- TASK\_ATTRIBUTES.
  - Auto start — when MQX starts, it creates one instance of the task.
  - DSP — MQX saves the DSP co-processor registers as part of the task's context.
  - Floating point — MQX saves floating-point registers as part of the task's context.
  - Time slice — MQX uses round robin scheduling for the task. Default is FIFO.
- CREATION\_PARAMETER: is the parameter to be passed to this task, when created.
- TIME\_SLICE: Time slice (in milliseconds) used for the task when using round-robin scheduling.

# Creating a Task (Continue)

- **When creating a task you have to:**

- ☐ Make the task definition

```
void init_task(void)
{
    /* Put the Task Code here */
}
```

- ☐ During execution time, create the task using

```
task_create()
```

(if it is not an autostart task)

# MQX\_Template\_List

```
{ WORLD_ID, world_task, 150, 9,  
  "world_task",  
  MQX_AUTO_START_TASK, 0, 0},
```

```
{ HELLO_ID, hello_task, 100, 8,  
  "hello_task",  
  MQX_TIME_SLICE_TASK, 0, 100},
```

```
{ LED_ID, led_task, 125, 10,  
  "LED Task",  
  MQX_AUTO_START_TASK |  
  MQX_TIME_SLICE_TASK, 0, 50},
```

# MQX - Task Management Example

```
{INIT_TASK,  
init_task, 100, 11,  
"init",  
MQX_AUTO_START_TASK,  
0, 0},
```

```
{TASK_A,  
Task_A, 100, 10,  
"Task A",  
0,  
0, 0},
```

```
{TASK_B,  
Task_B, 100, 9,  
"Task B",  
0,  
0, 0},
```

```
void init_task(void)  
{  
    _task_create(0, TASK_A, 0);  
    ...  
    _task_ready(Task_B);  
    ...  
}
```

```
void Task_A(void)  
{  
    ...  
    _task_create_blocked(0, TASK_B, 0);  
    ...  
    _task_abort(TASK_A);  
}
```

```
void Task_B(void)  
{  
    ...  
    _task_abort(TASK_B);  
}
```

init\_task is  
created when  
MQX starts

CPU Time

# Agenda

- What is an RTOS?
- MQX Basics: Tasks
- MQX Basics: Scheduling
- ➔ • **MQX Basics: Task Synchronization**
  - Semaphores
  - Events and Messages
- **MQX Intermediate**
  - Libraries
  - Interrupts
  - BSP
- **Additional Resources**
- **Review**



## MQX Basics: Task Synchronization

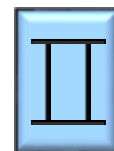
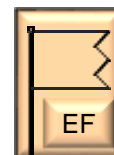
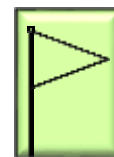
# Competence Condition

- What happens if two tasks access the same resource at the same time?
  - ❑ We call this “competence condition”. When two or more tasks read or write on share a resource at a certain moment
- Why the “competence condition” can be a problem
  - ❑ Memory corruption
  - ❑ Wrong results
  - ❑ Unstable application
  - ❑ Device conflicts



# Why Synchronization?

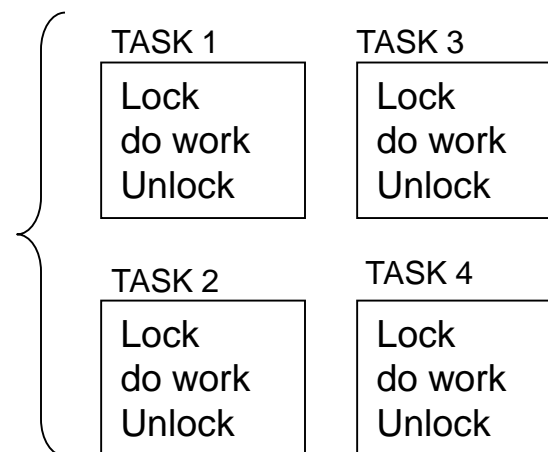
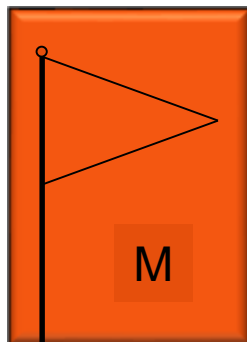
- **Synchronization may be used to solve:**
  - ☐ Mutual Exclusion
  - ☐ Control Flow
  - ☐ Data Flow
- ☐ **Synchronization Mechanisms include:**
  - ☐ Semaphores
  - ☐ Events
  - ☐ Mutexs
  - ☐ Message Queues
- The correct **synchronization mechanism** depends on the synchronization issue being addressed



# Mutual Exclusion

- Allowing only one task at a time to access a shared resource
- Resource may be devices, files, memory, drivers, code...
- Mutual exclusion locks the resource

Protected Resource



# Control Flow

- Control Flow provides a mechanism for a task or ISR to resume execution of one or more other tasks
  - Mutual exclusion is used to prevent another task from running
  - Control Flow is used to allow another task to run
- Not all synchronization objects are suitable for control flow:
  - Good: Non-strict Semaphores, Events, Messages, Task Qs
  - Bad: Mutexes, Strict Semaphores

# Agenda

- What is an RTOS?
- MQX Basics: Tasks
- MQX Basics: Scheduling
- MQX Basics: Task Synchronization
  - - Semaphores
  - Events and Messages
- MQX Intermediate
  - Libraries
  - Interrupts
  - BSP
- Additional Resources
- Review

# Semaphores

- A semaphore is a protocol mechanism offered by most multitasking kernels. Semaphores are used to:
  - ❑ Control access to a shared resource (mutual exclusion)
  - ❑ Signal the occurrence of an event
  - ❑ Allow two tasks to synchronize their activities
- A semaphore is a token that your code acquires in order to continue execution
- If the semaphore is already in use, the requesting task is suspended until the semaphore is released by its current owner



# How Semaphores Work

- **A semaphore has:**
  - ❑ counter — maximum number of concurrent accesses
  - ❑ queue — for tasks that wait for access
- **If a task waits for a semaphore**
  - ❑ if counter  $> 0$ 
    - counter is decremented by 1
    - task gets the semaphore and can do work
  - else
    - task is put in the queue
- **If a task releases (post) a semaphore**
  - ❑ if at least one task is in the semaphore queue
    - appropriate task is readied, according to the queuing policy
  - else
    - counter is incremented by 1

# Agenda

- What is an RTOS?
- MQX Basics: Tasks
- MQX Basics: Scheduling
- MQX Basics: Task Synchronization
  - Semaphores
  - Events and Messages
- MQX Intermediate
  - Libraries
  - Interrupts
  - BSP
- Additional Resources
- Review



# Events

```
If (EventBit == 0x03)
:  
Else  
:
```

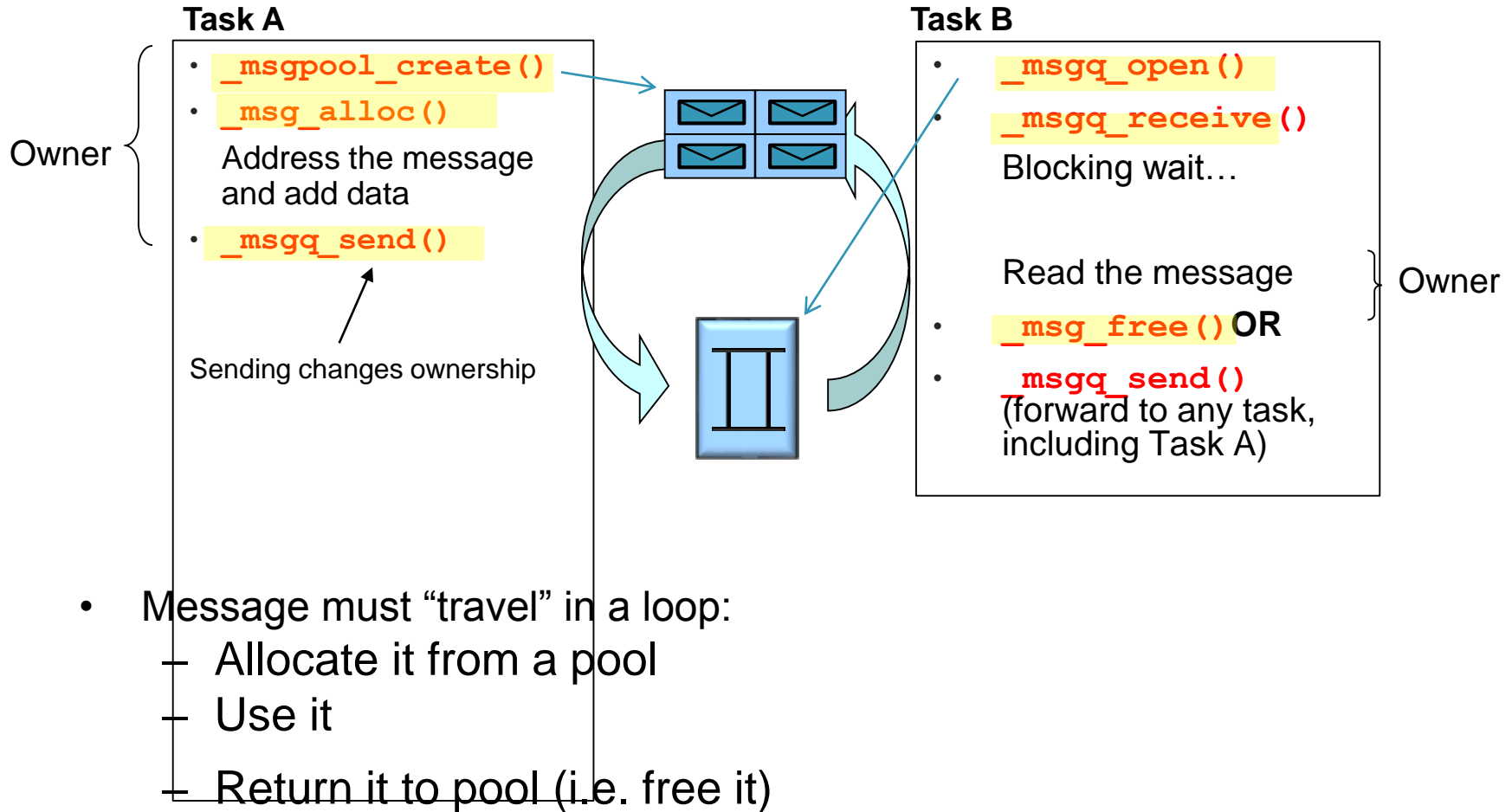
- Tasks can wait for a combination of **event bits** to become **set**. A task can set or clear a combination of event bits.
- Events can be used to **synchronize a task** with another task or with an ISR.
- The event component consists of event groups, which are groupings of event bits.
  - **32 event bits per group** (mqx\_unit)
- Tasks can wait for **all** or **any** set of event bits in an event group (with an *optional timeout*)
- Event groups can be identified by name or by index (fast event groups)

# Messages Passing

- Tasks can communicate with each other by exchanging messages  
(e.g Clipboard in windows, or mobile phone)
- Tasks send messages to message queues, and receive messages from message queues
- Messages can be assigned a priority or marked urgent
- Tasks can send broadcast messages

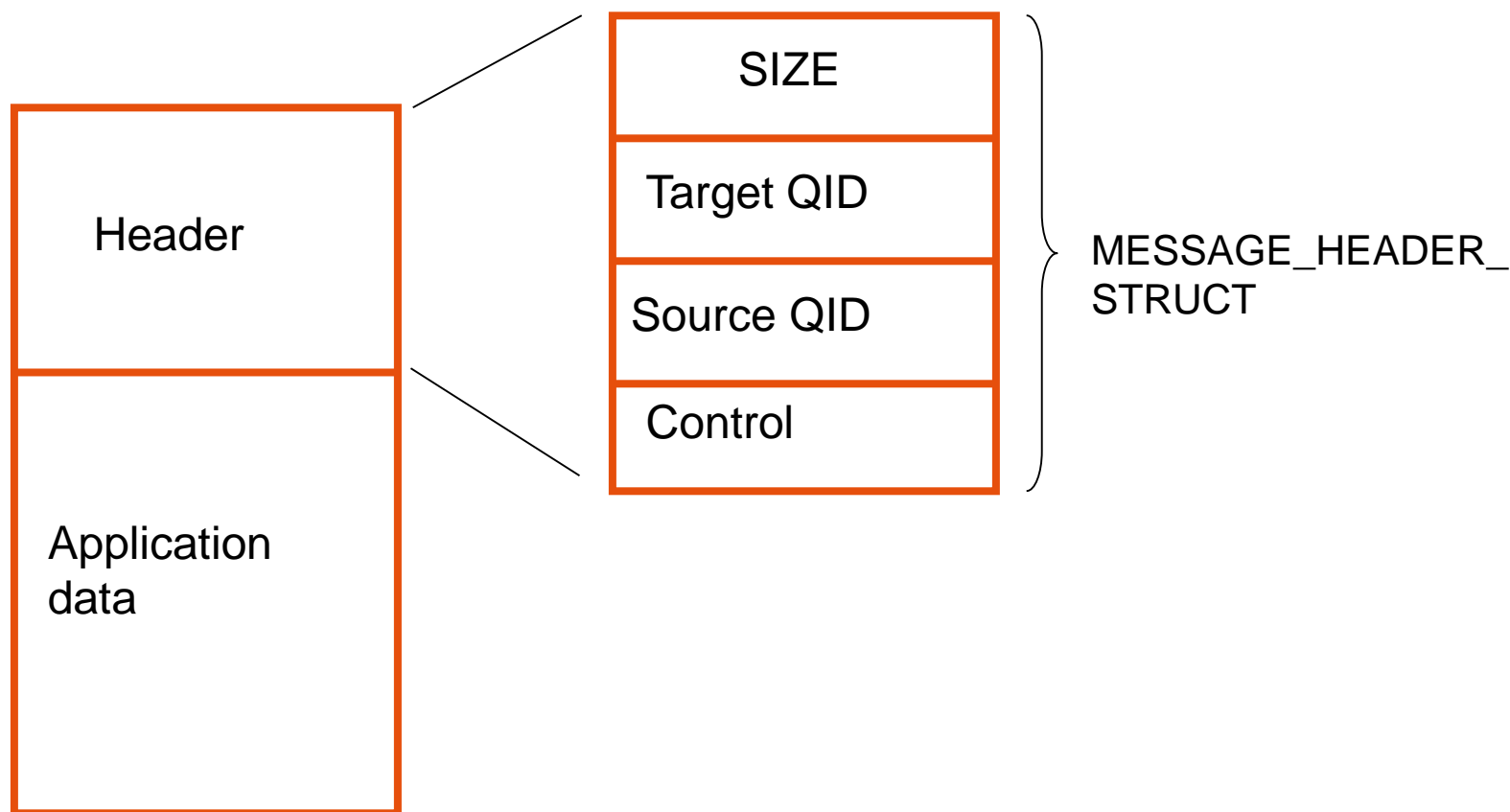


# Message passing example

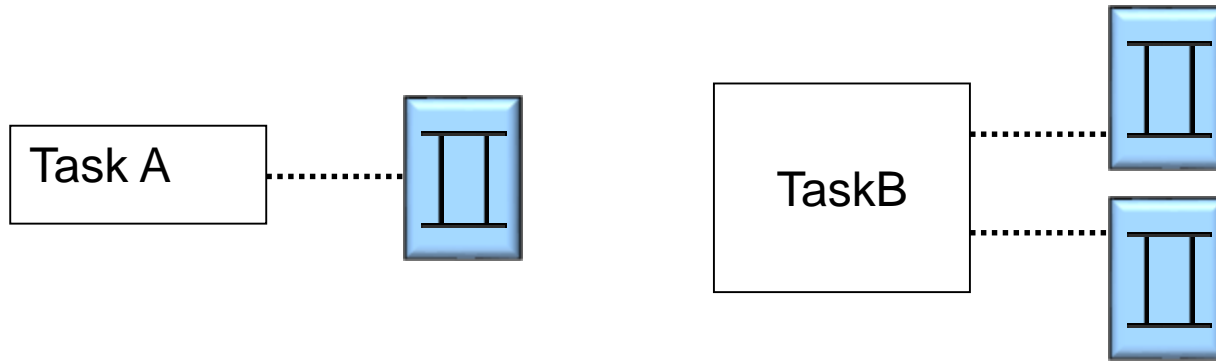


# Message Format

- Messages are areas of memory divided into a header and a data area
- Application data is user-defined



# Message Queues




- Each task can have one (or more) messages queues associated with it
- Messages are always addressed to queues, not tasks
- Queues are identified by `_queue_id`
  - This is a combination of queue number and CPU number
- Create a queue using `_msgq_open()`

# MQX Interrupts

- Embedded systems are based on ISR
- Usually an ISR is used for signal an event
- The most common actions on an ISR are:
  - Post a semaphore
  - Send a message
  - Set an event
  - Clear an error condition
- **Important:** ISRs are not tasks
- **Remember:** ISR should be short and should not use blocking functions.

# Agenda

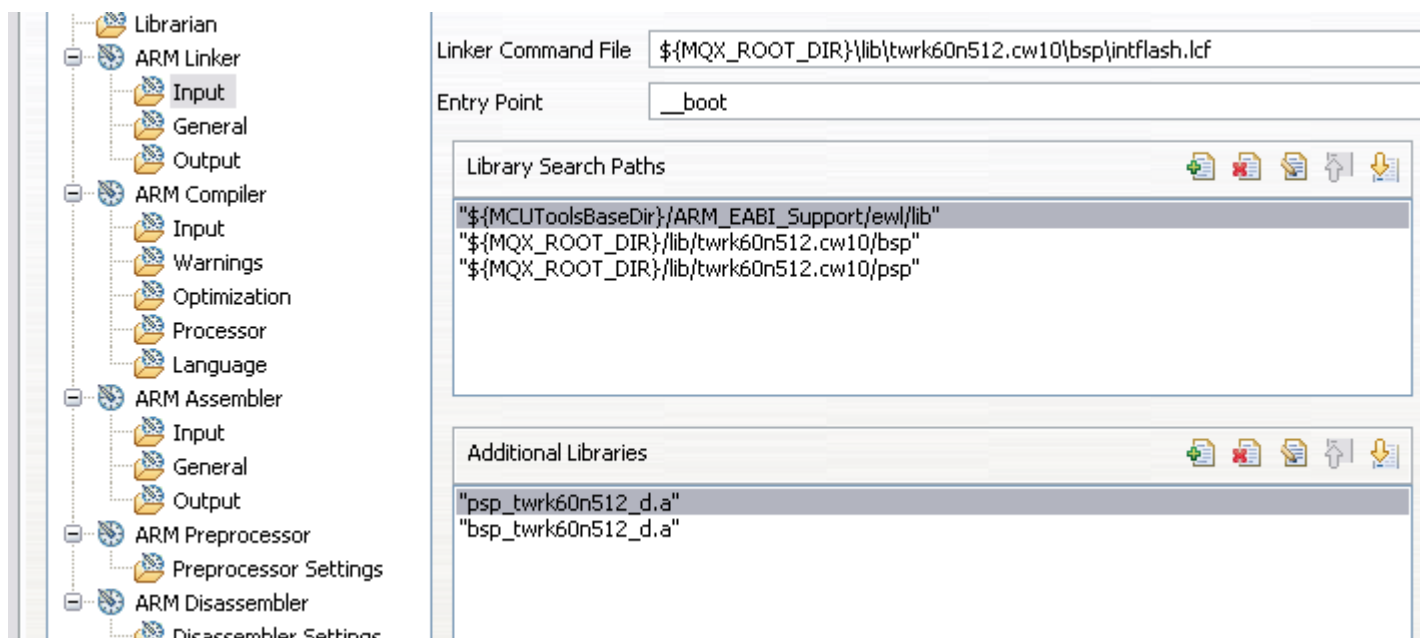
- What is an RTOS?
- MQX Basics: Tasks
- MQX Basics: Scheduling
- MQX Basics: Task Synchronization
  - Semaphores
  - Events and Messages
-  • **MQX Intermediate**
  - Libraries
  - Interrupts
  - BSP
- Additional Resources
- Review



## MQX Intermediate

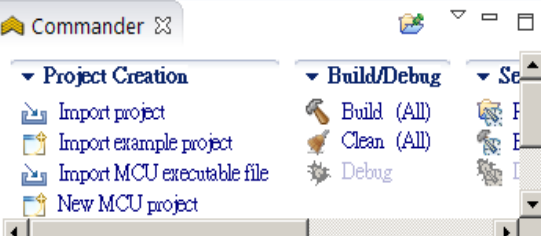
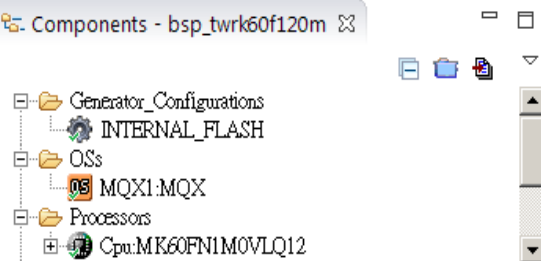
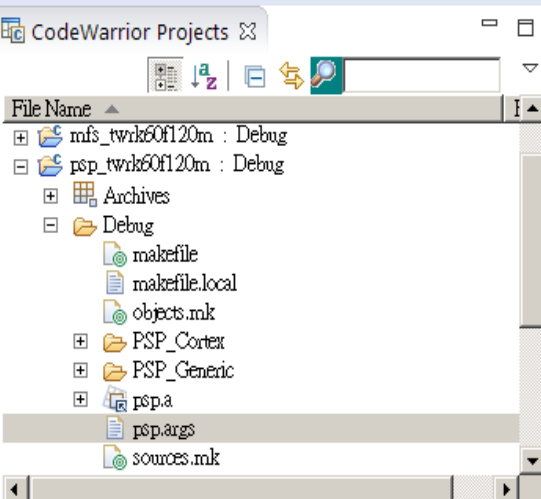
# Re-Compiling Libraries

- MQX is structured so that the applications get the MQX information from pre-compiled libraries located in the **lib** folder (ending in .a)



# Re-Compiling Libraries

- Anytime a change is made to **user\_config.h** the libraries should be re-compiled. This over-writes all the files in lib for that board.
- Anytime a change is made in the library source code, the library should be re-compiled.
- To re-compile the libraries, open up the library projects for the board.



sai\_dma\_ksai.c user\_config.h

Type filter text here

Parameter	Value	Defined in	User's comment
Scheduler Options			
MQX_HAS_TICK	true	<input type="checkbox"/> mqx_cfg.h	
MQX_DEFAULT_TIME_SLICE_IN_TICKS	true	<input type="checkbox"/> small_ram_config.h	
MQX_HAS_TIME_SLICE	false	<input type="checkbox"/> small_ram_config.h	
MQX_HAS_DYNAMIC_PRIORITIES	true	<input type="checkbox"/> mqx_cfg.h	
MQX_USE_IDLE_TASK	true	<input checked="" type="checkbox"/> user_config.h	
Synchronization Objects Options			
MQX_USE_EVENTS	true	<input type="checkbox"/> mqx_cfg.h	
MQX_USE_LWEVENTS	true	<input type="checkbox"/> mqx_cfg.h	
MQX_USE_MESSAGES	true	<input type="checkbox"/> mqx_cfg.h	
MQXCFG_ENABLE_MSG_TIMEOUT_ERROR	false	<input type="checkbox"/> mqx_cfg.h	
MQX_USE_LWMSGQ	true	<input type="checkbox"/> mqx_cfg.h	
MQX_USE_MUTEXES	true	<input type="checkbox"/> mqx_cfg.h	
MQX_MUTEX_HAS_POLLING	false	<input type="checkbox"/> small_ram_config.h	
MQX_USE_SEMAPHORES	false	<input type="checkbox"/> small_ram_config.h	
Other MQX Objects Options			
MQX_USE_LOGS	false	<input type="checkbox"/> small_ram_config.h	
MQX_USE_LWLOGS	true	<input type="checkbox"/> mqx_cfg.h	
MQX_LWLOG_TIME_STAMP_IN_TICKS	true	<input type="checkbox"/> small_ram_config.h	

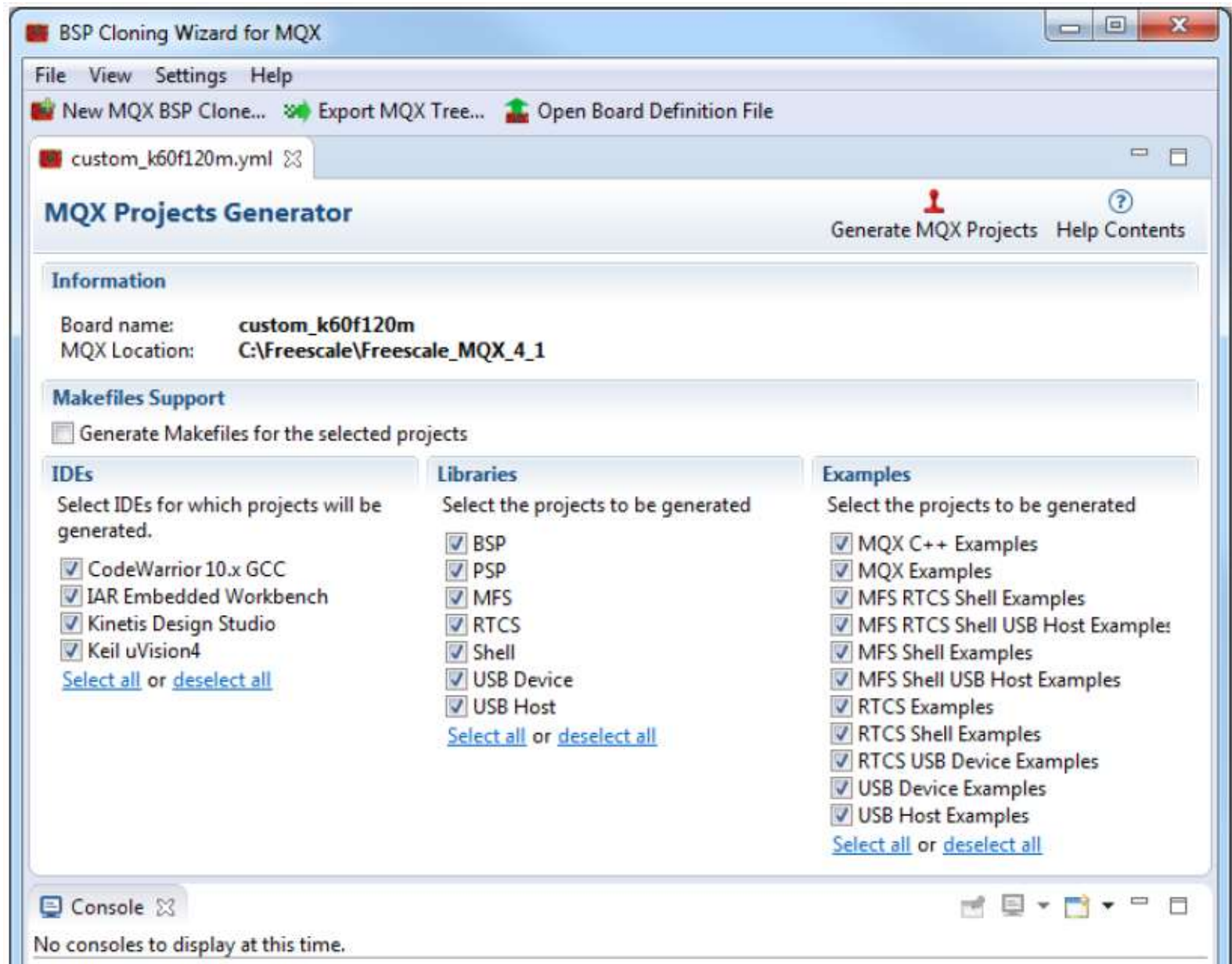
BSP MQX RTCS MFS Shell USBH user\_config.h

Problems Console Target Tasks Memory Search Progress

Processor Expert

User working directory = C:\ProgramData\Processor Expert\mqx\_pes\_00  
 Internal cache directory = C:\ProgramData\Processor Expert\PECache\8209a496  
 Processor Expert license file = not used (no license file)  
 Mar 19, 2015 1:37:12 PM Successfully started Processor Expert service  
 CDE: no license found, using Community Edition License

# Clone wizard



# MQX Boot-Up

- What is a Board Support Package (BSP)?
  - Startup Code for the processor
  - Initialize memory, clock, interrupt controller
  - All the drivers needed with the proper settings



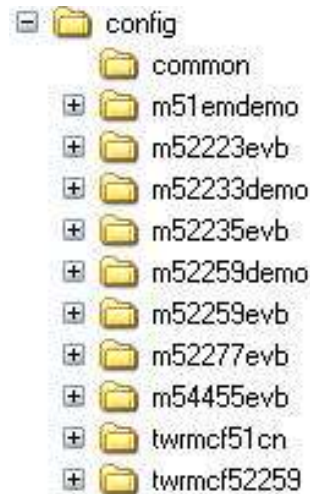
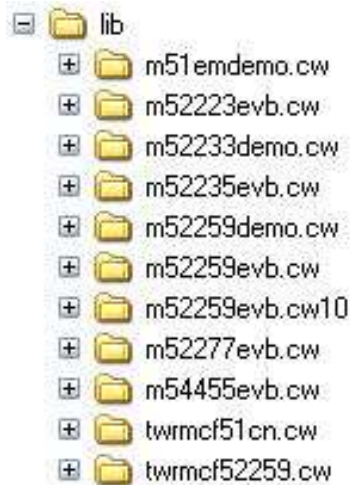
# MQX Board Support Package

- Initializes microprocessor and board
  - PLL and clocks, memory interface, core registers
- Defines board specific parameters
  - Clocks, memory parameters, interrupt usage, driver parameters/enabling, MQX limits, IO pin definitions, ENET interfaces, etc.
- Presents board-specific API to I/O drivers and application
  - Timer ISR functions used by MQX scheduler, I/O pin initializations
- Installs and initializes device drivers (selected by *user\_config.h*)

# Board-specific Folders in MQX

Lib, Config and MQX folders contain folders for each board

- The \lib\ folder contains the Library outputs
- The \config\ folder has the user\_config.h file and the Build\_libs projects
- The \mqx\source\bsp\ folder has the source code specific to any board



# BSP Source Directory Overview

- Key BSP project files

- `bsp.h` and `<board_name>.h` – main BSP header files, configuration constants etc.
- `<target_name>.lcf` – linker command files (one for each memory configuration)
- `init_hw.c` (*bsp\_init.c in older BSPs*) – PLL, memory, FlexBus initialization
- `init_bsp.c` – BSP timer, watchdog and IO drivers initialization
- `gpio_init.c` – IO pin initialization routines called by IO drivers
- `vectors.c` – startup code (boot vector) and initial interrupt table

- Parameter Files

- `init_<dev>.c` – default initialization parameters structures for given IO device driver
- `enet_ini.c` – ethernet specific initialization structures and functions
- `mqx_init.c` – default MQX initialization parameters structure

# MQX RTOS Source Code

## MQX RTOS

### "PSP" library compiled as mqx.a

e.g. /mqx/build/psp\_m52259evb.mcp -> /lib/m52259evb.cw/mqx/mqx.a

`mqx/source/*` (except *bsp*, *psp* and *io*)

**Generic RTOS Kernel**  
(platform- and device-independent ANSI C Code)

`mqx/source/psp/<platform>`

#### PSP: Platform-specific Code

(e.g. ColdFire-specific low-level routines, assembler-optimized parts of scheduler, interrupt context save/restore, cache control, ...)

#### PSP: Device-dependent Code

(e.g. mcf5225.h register structures and macros)

### "BSP" library <board>.a

e.g. /lib/m52259evb.cw/mqx/m52259evb.a

`mqx/source/io/*`

#### IO Drivers

(each driver is typically split to device-independent code and code specific to device or family of devices)

`mqx/source/bsp/<board>`

#### BSP: Device- and Board-specific code

(startup, vector table, device, memory and board initialization, starting \_mqx, installing IO drivers)

`config/<board>`

**User Configuration**



Included by all MQX files


Linked library files and related public header files



`lib/<board>/mqx`  
`lib/<board>/*`

**Binary Libraries**  
linked by apps.

# Agenda

- What is an RTOS?
- MQX Basics: Tasks
- MQX Basics: Scheduling
- MQX Basics: Task Synchronization
  - Semaphores
  - Events and Messages
- MQX Intermediate
  - Libraries
  - Interrupts
  - BSP
-  • Additional Resources
- Review



## Additional Resources

# Kinetis MQX Quick Start Demos

- Source code and lab guide available online for both K40 SLCD and K60 Web server demos (IAR and CW10.1)
  - <http://freescale.com/twr-k40x256>
  - <http://freescale.com/twr-k60n512>
- Showcases Ethernet, SLCD, SD Card, USB, I2C, ADC, TSI, RNG, UART, RTC, Flash, and GPIO features on Kinetis.
- TWR-K40X256
  - Display seconds, hours and minutes, potentiometer, and temperature
  - <http://youtu.be/4sSRHyYyilA>
- TWR-K60N512
  - Interactive web server and touch memory game
  - <http://youtu.be/gkL4n2b5RU4>

Learn more at: [www.freescale.com/MQX](http://www.freescale.com/MQX)



# Watch the K60 quick start video

- (a) Assembly
- (b) OS console
- (c) SD card access & File system.
- (d) USB mouse
- (e) Ethernet Web server



# Further Reading and Training

- **Webinnar at [www.freescale.com/tower](http://www.freescale.com/tower)**

- Introduction to Tower, CodeWarrior 10, and MQX
- TWR-K60N512 and TWR-K40X256 Quick Start Demos

- **Videos: [www.freescale.com/mqx](http://www.freescale.com/mqx)**

- Getting started with MQX
- And more

- **vFTF technical session videos [www.freescale.com/vftf](http://www.freescale.com/vftf)**

- Introducing a modular system, Serial-to-Ethernet V1 ColdFire® MCU and Complimentary MQX™ RTOS
- Writing First MQX Application
- Implementing Ethernet Connectivity with the complimentary Freescale MQX™ RTOS

## Further Reading and Training (Continue)

- **MQX Release Notes**
- **MQX User's Guide**
- **Writing First MQX Application (AN3905)**
- **Using MQX: RTCS, USB, and MFS (AN3907)**
- **How to Develop I/O Drivers for MQX (AN3902)**
- **IP Camera and USB Snapshot with MQX (AN4022)**
- **Supporting New Toolchains with Freescale MQX RTOS (4190)**
- **Motor Control Under the Freescale MQX Operating System (AN4254)**
- **MQX Board Support Package Porting Guide (AN4287)**



[www.Freescale.com](http://www.Freescale.com)