

μEZ® Overview

The Rapid Development Platform



Muse

μEZ® is a registered trademark of Future Designs, Inc.

Overview

- ▶ What is μ EZ?
- ▶ μ EZ RTOS Engine
- ▶ μ EZ Four Tier Hierarchy
- ▶ Reusable HAL and Device Drivers
- ▶ LPC2478 , LPC1788, & RX62N Examples
- ▶ FDI and Community Support Network
- ▶ Brand “M” RTOS Comparison
- ▶ Field Upgradeability
- ▶ Future Enhancements
- ▶ Developer Details



Customer Dilemma

- ▶ 8-bit migration customers require free / low cost tools and FTTM
- ▶ These customers also want to use the cool new MCU features
 - Like USB, Ethernet, Touch Screen LCD, ***but***
- ▶ Most customer ENG resources are either
 - 8-bit HW guys
 - PC/API level SW guys
- ▶ Quickly get crushed by OS and Driver complexity
- ▶ Also want single chip uC solutions where possible
- ▶ Linux doesn't ***really*** solve their problems
 - Complex OS, steep learning curve
 - Requires complex HW / memory
 - Dedicated Host development environment
- ▶ Linux works for some customers, but not most 8-bit guys



μEZ® Value Proposition

- ▶ μEZ is a Low Cost Tools Solution
 - Enables the 8-bit migration to ARM
- ▶ Migration customers require free or low cost tools
 - μEZ / FreeRTOS and Crossworks = less than \$1500
- ▶ Migration customers want cool features like USB, ***but***
 - COM Drivers & Stacks are part of the μEZ package
- ▶ Think of μEZ as **“Linux Light”**
- ▶ μEZ enables the single chip MCU solution
 - Saves as much as \$40 in HW cost /complexity
 - No BGAs, no fine pitch PCBs
 - No short external memory life cycles
- ▶ Developed by FDI, but an open source community project like Linux



80C51/PIC

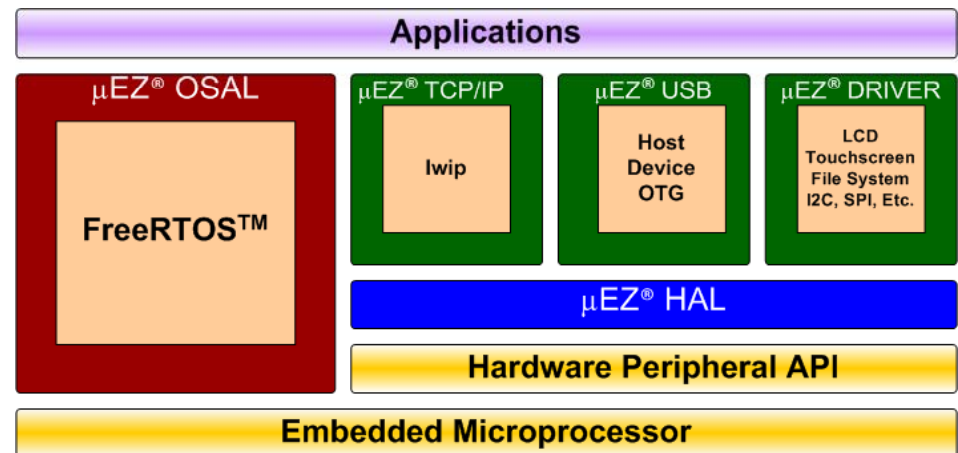


ARM[®]
RENESAS

What is μ EZ[®] ?

μ EZ provides underlying RTOS and processor abstraction, enabling the application programmer to focus on the value added features of their product.

μ EZ is an **optional** platform that enhances portability of application code to multiple ARM platforms with high reusability.



What is μ EZ[®] ?

- ▶ Has three primary components:
 - Operating System Access Layer (OSAL)
 - Sub-system Drivers
 - Hardware Abstraction Layer (HAL)
- ▶ Providing:
 - Cross Processor and Cross Platform Portability
 - RTOS Independence
 - Hardware and Device Driver Abstraction
 - Library of Reusable Code – stop reinventing the wheel!
 - Rapid Application Development
 - Standardized interfaces across similar drivers
 - Open Source
- ▶ Customizable
 - by the end customer, FDI, Open Source community

μEZ Components



- ▶ RTOS – FreeRTOS
 - Tasks, Semaphores, Mutexes, Queues
- ▶ FileSystem – FATFS
 - FAT16/FAT32
 - SDCard
 - Flash Drive
- ▶ USB-Device
 - HID
 - Mass Storage Devices
- ▶ USB-Host
 - OHCI
 - Bulk Device
- ▶ TCP-IP – lwIP
 - TCP/IP
 - UDP
 - BSD Socket / Netconn Interfaces
 - SNMP
 - ICMP
 - DHCP Client
 - SLIP
 - PPP
- ▶ Graphics – SWIM
 - Windows
 - Fonts
 - Drawing primitives

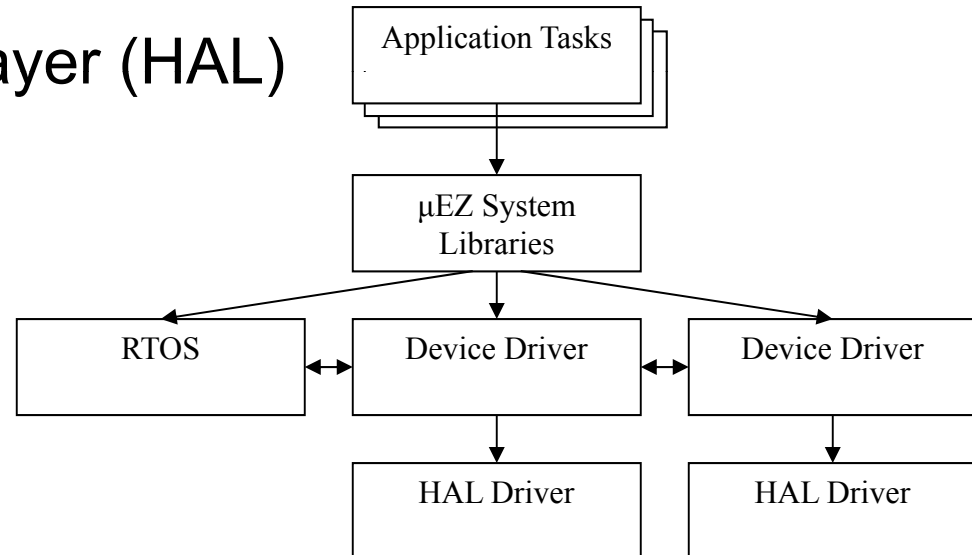
Based on uEZ V 1.08

The μ EZ Engine – RTOS

- ▶ Support for Multiple RTOS
 - FreeRTOS, SafeRTOS
 - Micrium
 - Others
- ▶ Operating System Access Layer (OSAL)
 - Common μ EZ Interface to RTOS
- ▶ μ EZ requires only basic RTOS features
 - Tasks
 - Semaphores
 - Mutexes
 - Queues
 - Basic Memory Management

μEZ Four Tier Hierarchy

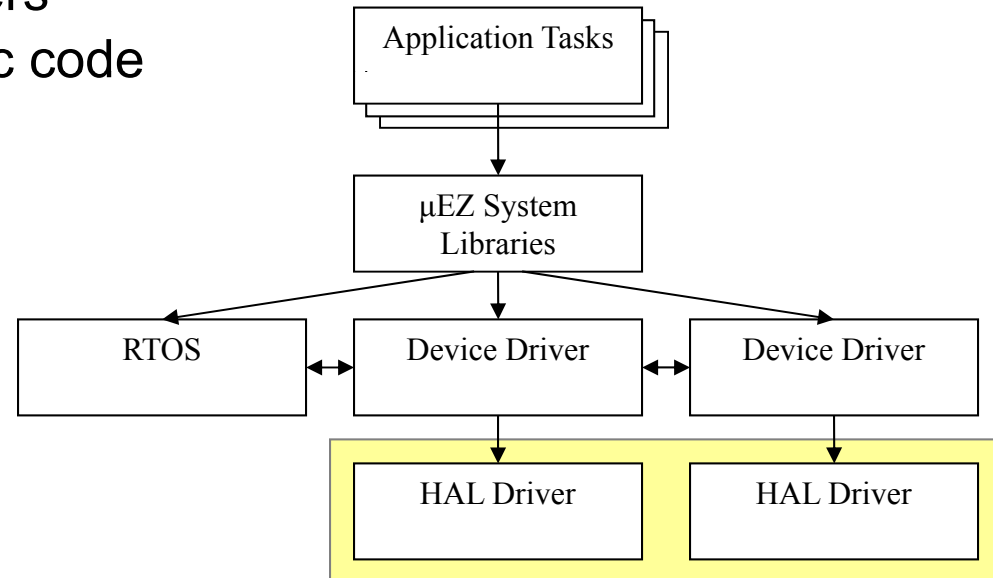
- ▶ Application Program
- ▶ μEZ System Libraries
- ▶ Device Drivers
- ▶ Hardware Abstraction Layer (HAL)



Building Up: HAL Drivers

▶ Hardware Abstraction Layer (HAL) Drivers

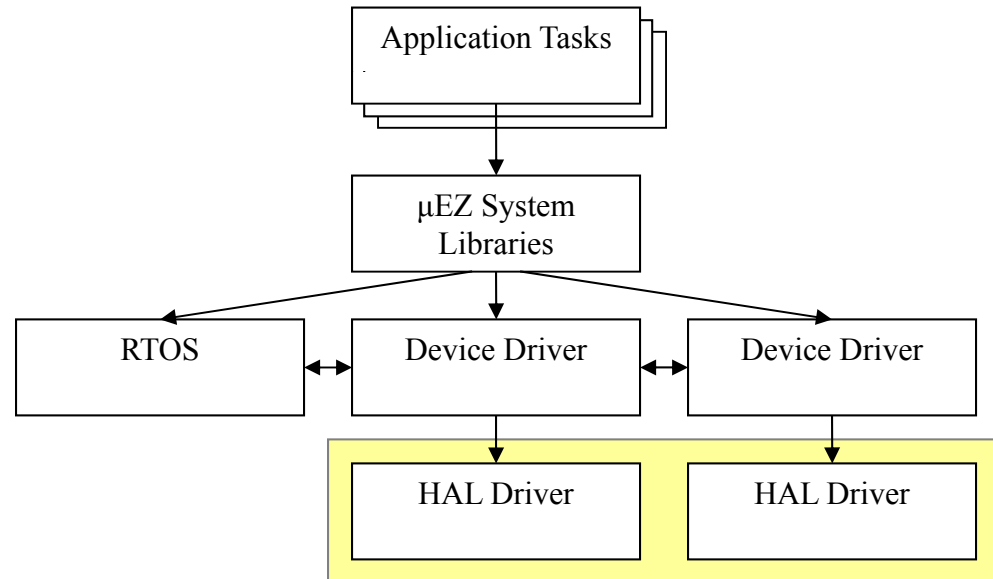
- Direct access to processor peripherals and hardware
- Does *not* interface with RTOS
- Standard structure to all HAL Drivers
- Registry of all HAL Drivers
- Lowest level and specific code



Example – LPC2478 HAL Package

▶LPC2478 BSP Package

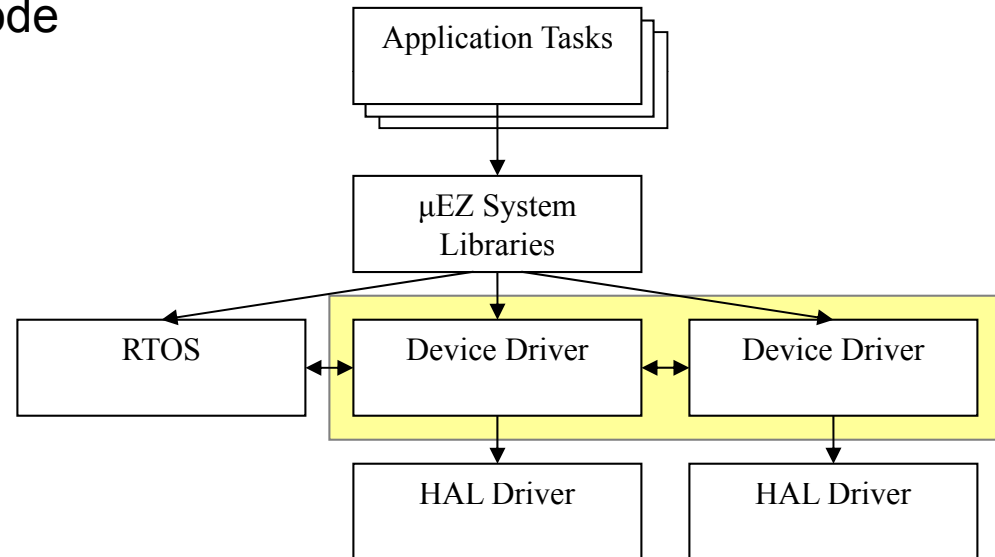
- ADC
- Ethernet MAC
- GPIO
- I2C
- Interrupts
- LCD Controller
- PWM
- RTC
- SPI/SSP
- Timers
- UARTs
- USB Device Controller
- USB Host



Building Up: Device Drivers

▶ Device Drivers

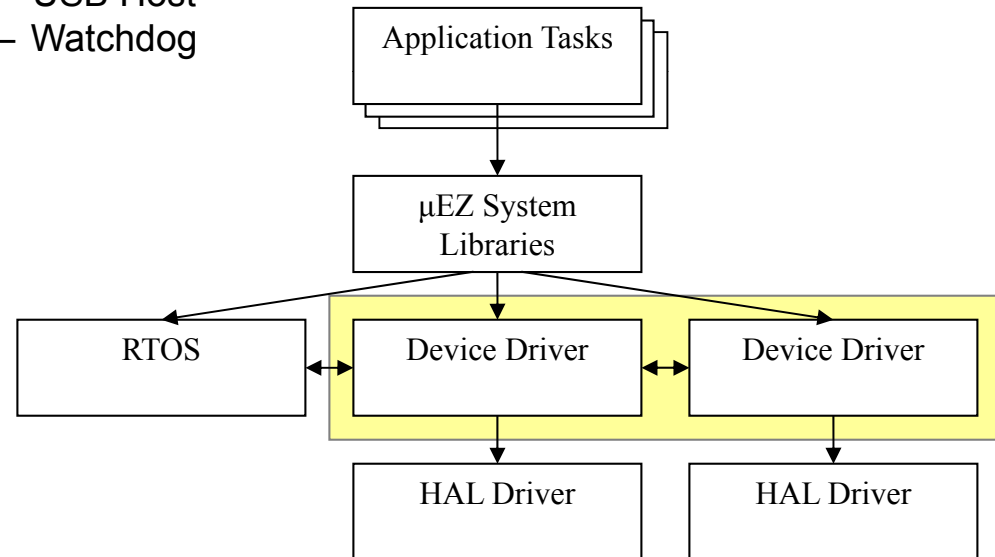
- Connect the HAL Drivers to the RTOS
- Manage multiple callers, blocking, and queuing
- Standard structure for all Device Drivers
- Registry of all Device Drivers
- Mid-level highly portable code



Example – Platform Device Drivers

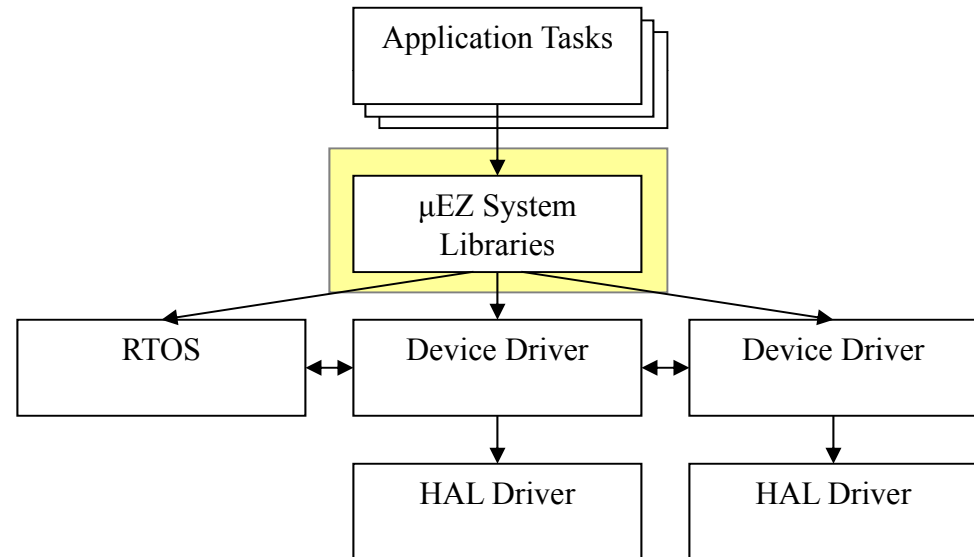
▶ CARRIER Device Drivers Package

- Accelerometer (I2C)
- ADC
- Amplifier (I2C)
- Backlight (PWM)
- Buttons (I2C)
- Character Display
- DAC
- EEPROM (I2C)
- Flash Memory
- I2C
- ISC
- Keypad
- LCD
- LED (I2C)
- Mass Storage (SPI/USB)
- PWM
- RTC (I2C)
- Serial UART
- Temperature Sensor (I2C)
- Tone Generator
- Touchscreen
- SPI/SSP
- USB Device
- USB Host
- Watchdog



μEZ System Libraries

- ▶ Portable code on top of portable device drivers
- ▶ Wrappers for commonly used low level functions
 - GPDMA
 - I2C
 - I2S
 - SPI
 - SSP
 - UART/Serial
- ▶ High Level Libraries
 - TCP/IP Stack
 - HTTP Server
 - FAT File System
 - USB Host
 - USB Device Drivers (HID)
 - Graphics Library (SWIM)
 - Customer specific



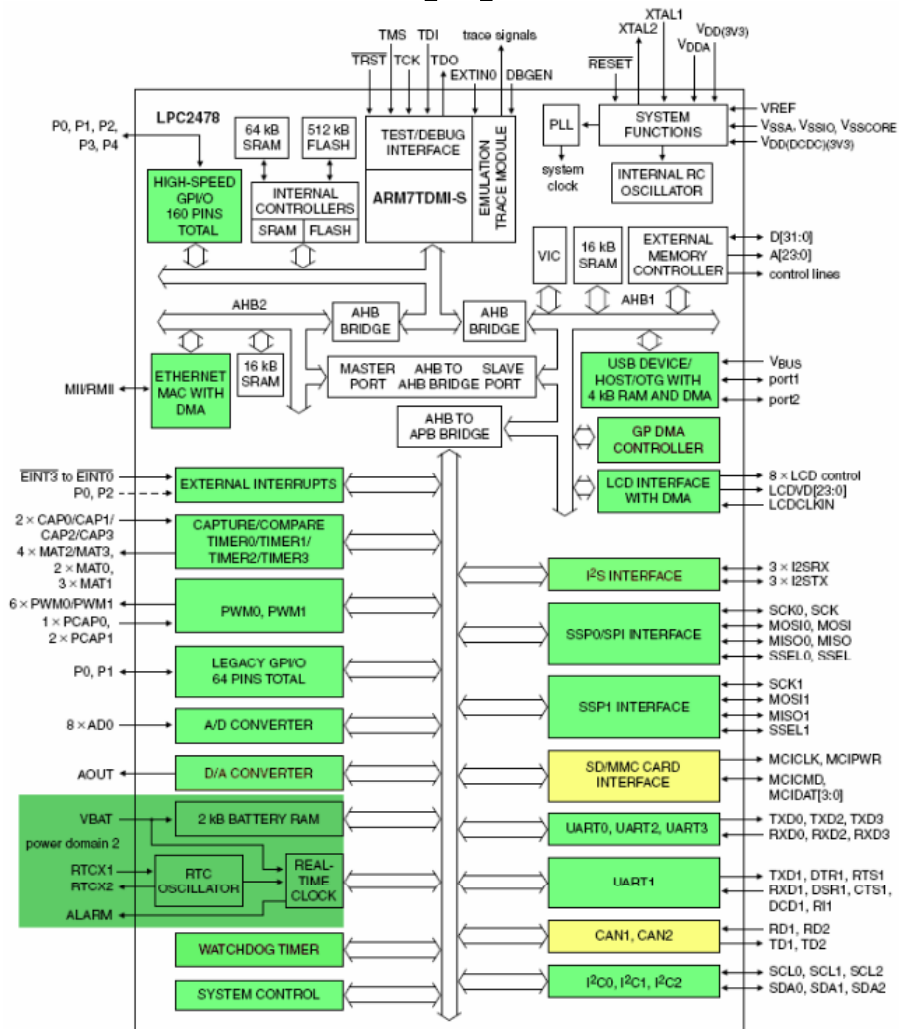
μEZ LPC2478 Support

Available Now

- GPIO
- A/D
- D/A
- PWM
- RTC
- USB
- SSP
- SPI
- UART
- I2C
- I2S
- GP DMA
- Watchdog

Coming Soon

- MMC Card
- CAN



LPC2478

μEZ LPC1788 Support

Available Now

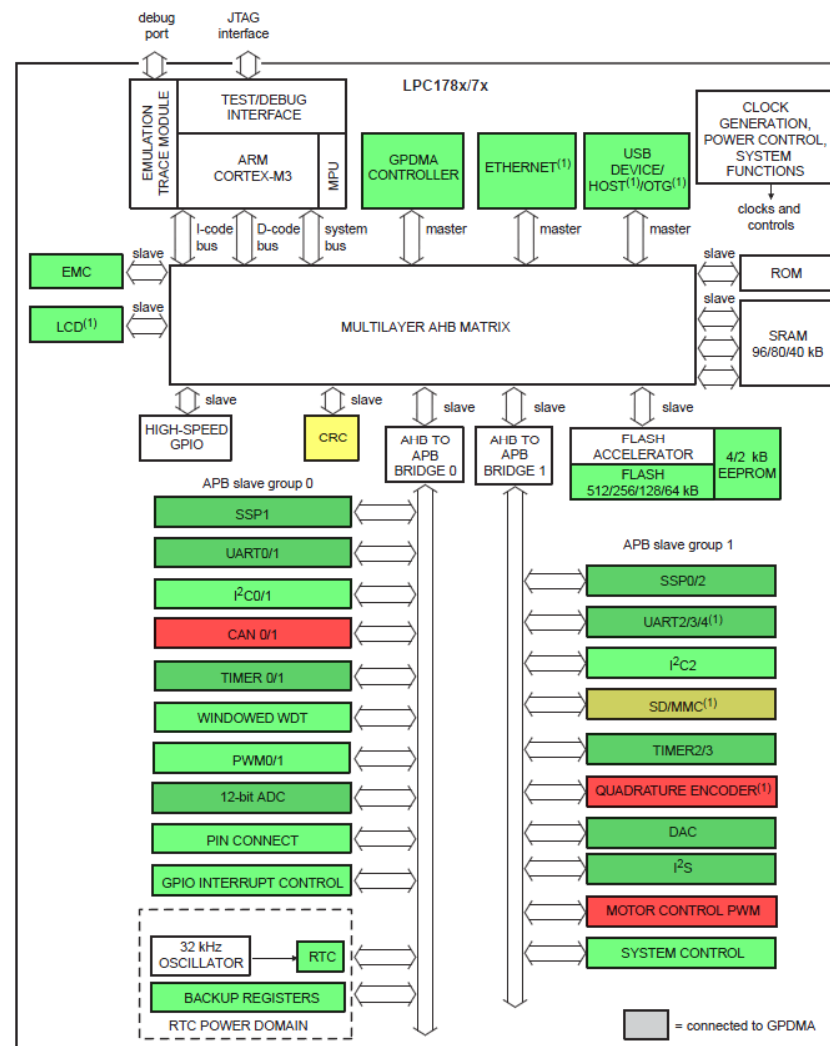
- A/D
- D/A
- EEPROM
- Ethernet
- GPDMA
- GPIO
- I2C
- I2S
- LCD
- PWM
- RTC
- SDRAM via EMC
- SSP
- Timers
- UART
- USB Device & Host
- Watchdog

Coming Soon

- CRC
- SD/MMC

Future

- CAN
- Quadrature
- Motor Control



002aaf528

FDI and Community Support

- ▶ μEZ is Open Source
- ▶ Source provided on www.sourceforge.net/projects/uez
- ▶ Forums for discussion
- ▶ Bug tracking
- ▶ Enhancement submission
- ▶ Contract Services

Brand M RTOS and μ EZ[®] Comparison

| Module | Brand M Flash size | Brand M RAM | μ EZ Flash size | μ EZ RAM |
|---------------|-----------------------|----------------|------------------------|-----------------|
| BSP | 8,503 | 32 | 32,696 | 8,521 |
| uC/LCD | 384 | 6 | 4,117 | 212 |
| App tasks | 8,697 | 4,133 | 16,848 | 4,332 |
| uC/USB Host | 26,565 | 10,669 | 10,000 | 2,153 |
| uC/USB Device | 7,410 | 513 | 9,975 | 4,201 |
| uC/LIB | 19,744 | 228 | 11,553 | 340 |
| uC/OS | 8,898 | 7,584 | 16,708 | 8,512 |
| uC/HTTP | 4,236 | 6,696 | 945 | 1,088 |
| uC/TCP-IP | 77,634 | 24,531 | 58,654 | 19,907 |
| uC/FS | 17,124 | 565 | 25,426 | 1,062 |
| Total | 179,195 | 54,957 | 186,922 | 52,070 |

Preliminary data based on μ EZ v1.08

Field Upgradability and μ EZ

- ▶ Primary Boot Loader Options
 - JTAG
 - ISP Flash

- ▶ Secondary Boot Loader
 - Stand-alone μ EZ based FAT FS Download developed for customer
 - Developing requirements for an optional integrated module
 - Options under consideration
 - Serial - X-modem or other protocol
 - TCP/IP - TFTP is most common method
 - FAT File Download to Flash
 - USB
 - Micro SD card
 - All options have trade-offs
 - May offer all options over time

Future Enhancements for μ EZ

- ▶ Nano-X Graphics Library
- ▶ Ongoing Processor Support
 - Cortex-M3 from other vendors (STMicro, Atmel, TI)
 - ARM9 (LPC3250)
 - Renesas RX62N/RX63N support
- ▶ Improved Make System
- ▶ Wireless Networking

μEZ® Developer Details

The Rapid Development Platform



Muse

Developer Details

- ▶ Examples
- ▶ Project Layout
- ▶ Initialization Basics

Object Oriented Interfaces

▶ Goals

- ANSI-C compatible
- Easy to understand
- One-to-one correspondence with hardware
- Unique workspace per instance with each API in its own workspace or 'box'
- Runtime configuration (e.g. more than one type of LCD)
- Common interfaces to layers above HAL and Devices

Object Oriented Interfaces

▶ Interface Structure – stored once in ROM (OOP Class)

```
typedef struct {
    const char *iName;
    TUInt16 iVersion;
    T_uezError (*InitializeWorkspace)(void *aWorkspace);
    TUInt32 iWorkspaceSize;

    <<<list of pointers to functions>>>
} T_halInterface;
```

- Unique name “Interface:Manufacturer:UniqueID” (e.g. I2C:NXP:LPC2478)
- Version of API (e.g., 0x100 = 1.00)
- Initialization routine to create workspace
- Size of workspace for memory management

▶ Workspace Structure – stored in memory per instance (OOP Instance)

```
typedef struct {
    T_halInterface *iInterface;
    <<< specific members to this driver go here >>>
} T_halWorkspace;
```

- Pointer to interface provides link to functions
- This workspace structure is passed to all interface functions (*this* pointer)

Example HAL Interface – I2C Bus

▶ HAL_I2CBus

```
typedef void (*I2CRequestCompleteCallback)(
    void *aCallbackWorkspace,
    I2C_Request *iRequest);

typedef struct {
    // Header
    T_halInterface iInterface;

    // Functions
    void (*RequestRead)(
        void *aWorkspace,
        I2C_Request *iRequest,
        void *aCallbackWorkspace,
        I2CRequestCompleteCallback aCallbackFunc);
    void (*RequestWrite)(
        void *aWorkspace,
        I2C_Request *iRequest,
        void *aCallbackWorkspace,
        I2CRequestCompleteCallback aCallbackFunc);
} HAL_I2CBus;
```

- Each call handles a single read or write with all parameters in I2C_Request
- When I2C is complete, uses callback (from within interrupt)
- These routines can be interrupt driven or polled, but must assume interrupt

Example Device Interface – I2C Bus

▶ DEVICE_I2C_BUS

```
typedef struct {  
    // Header  
    T_uezDeviceInterface iDevice;  
  
    // Functions  
    T_uezError (*ProcessRequest)(void *aWorkspace, I2C_Request *aRequest);  
} DEVICE_I2C_BUS;
```

- ProcessRequest function handles the I2C request using RTOS features and blocks the calling thread until the request is complete
- Each call handles a single read and/or write with all parameters in I2C_Request
- The interrupt driven code is handled internally by the I2C Bus device driver
- Allows the caller to focus on the requirements of the request and not the low level details

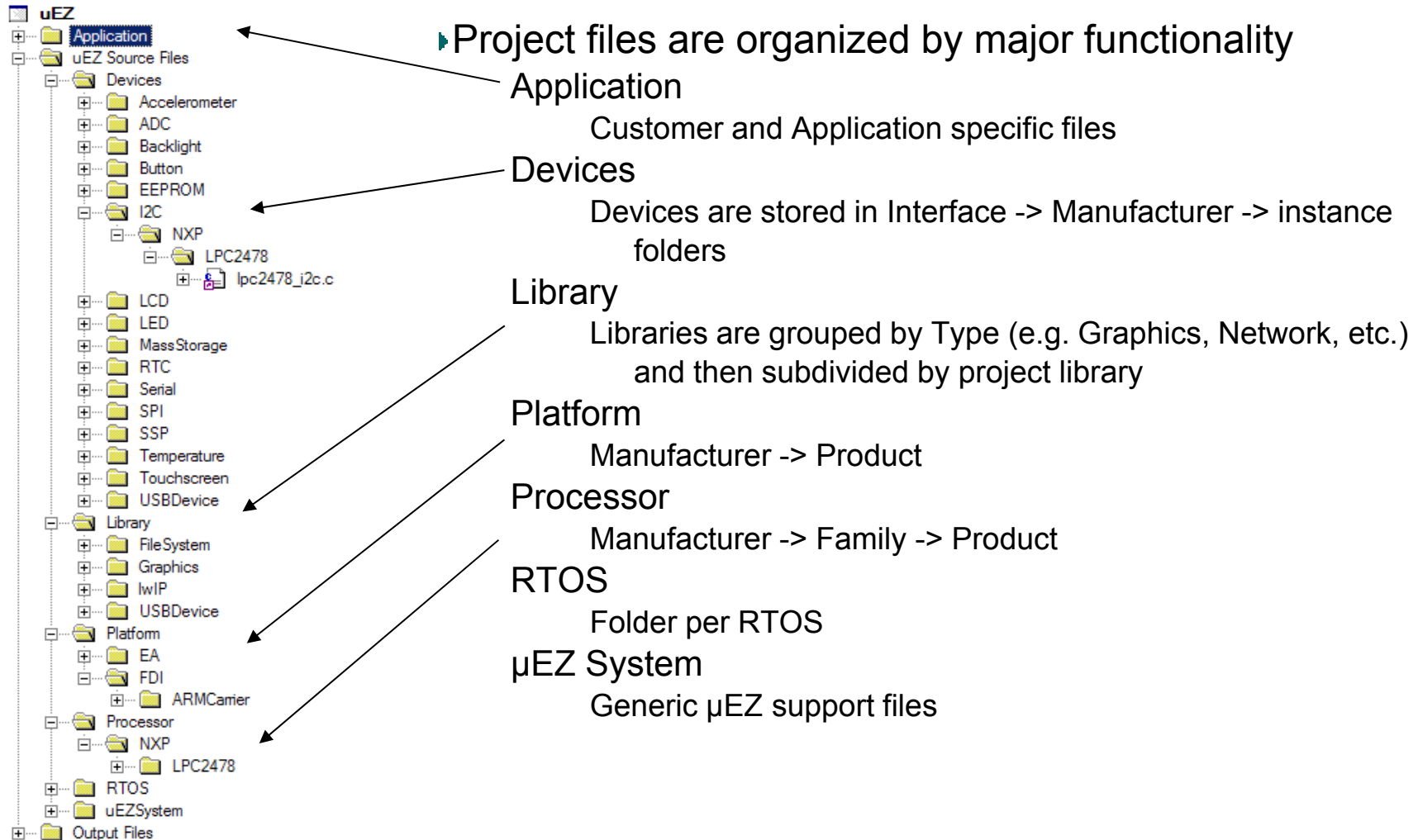
Example μ EZ System Library – I2C Bus

▶ uEZI2C.h

```
T_uezError UEZI2COpen(  
    const char *const aName,  
    T_uezDevice *aDevice);  
T_uezError UEZI2CClose(T_uezDevice aDevice);  
T_uezError UEZI2CRead(  
    T_uezDevice aDevice,  
    TUInt8 aAddress,  
    TUInt32 aSpeed,  
    TUInt8 *aData,  
    TUInt8 aDataLength,  
    TUInt32 aTimeout);
```

- Refer to I2C devices by general name (e.g. “I2C0”, “I2C1”, “I2C2”, etc.) even if the specific underlying hardware is different
- Simple open/close mechanism allows for easy access to device
- Standard command for making read and write commands

μEZ Project Layout



Smaller is Better – μ EZ Compile Options

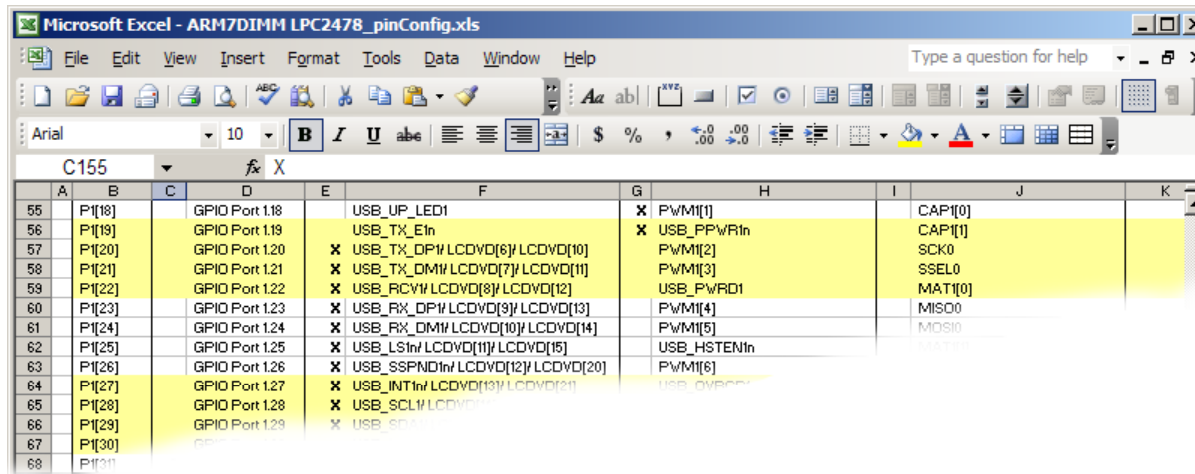
- ▶ Full customization of all components
 - #defines are used to enable/disable components
- ▶ Four configuration files are used
 - Build Configuration File
 - Application Configuration File
 - Platform Configuration File
 - Processor Configuration File
- ▶ Each layer of configuration can override the lower level
 - Application Configuration File controls everything in one place

Startup

- ▶ Bootloader (outside of μ EZ or internal to processor)
- ▶ Bootstrap (startup.s)
- ▶ uEZBSPInit()
 - Pin Configuration
 - Interrupts Initialization
 - SDRAM/Memory Initialization
 - Processor HAL Drivers Registration and Initialization
 - Platform Device Driver Registration and Initialization
 - RTOS initialized and started
- ▶ Main task created starting with main()

Startup – Pin Configuration

- ▶ Pins are should be set to power up defaults at startup
- ▶ PinsToH conversion utility
 - Translates .csv spreadsheet file of pins into ARM7 compatible format



| A | B | C | D | E | F | G | H | I | J | K |
|----|--------|----------------|--------------------------------|---|-------------|---|---|---------|---|---|
| 55 | P1[18] | GPIO Port 1.18 | USB_UP_LED1 | X | PWM1[1] | | | CAP1[0] | | |
| 56 | P1[19] | GPIO Port 1.19 | USB_TX_Eln | X | USB_PPWRIn | | | CAP1[1] | | |
| 57 | P1[20] | GPIO Port 1.20 | USB_TX_DPn/LCDVD[6]/LCDVD[10] | | PWM1[2] | | | SCK0 | | |
| 58 | P1[21] | GPIO Port 1.21 | USB_TX_DMn/LCDVD[7]/LCDVD[11] | | PWM1[3] | | | SSEL0 | | |
| 59 | P1[22] | GPIO Port 1.22 | USB_RX_DPn/LCDVD[8]/LCDVD[12] | | USB_PwRD1 | | | MAT[0] | | |
| 60 | P1[23] | GPIO Port 1.23 | USB_RX_DMn/LCDVD[9]/LCDVD[13] | | PWM1[4] | | | MISO0 | | |
| 61 | P1[24] | GPIO Port 1.24 | USB_RX_DMn/LCDVD[10]/LCDVD[14] | | PWM1[5] | | | MOSI0 | | |
| 62 | P1[25] | GPIO Port 1.25 | USB_LSIn/LCDVD[11]/LCDVD[15] | | USB_HSTENIn | | | MSTR0 | | |
| 63 | P1[26] | GPIO Port 1.26 | USB_SSFn/LCDVD[12]/LCDVD[20] | | PWM1[6] | | | | | |
| 64 | P1[27] | GPIO Port 1.27 | USB_INTIn/LCDVD[13]/LCDVD[21] | | USB_OVCFn | | | | | |
| 65 | P1[28] | GPIO Port 1.28 | USB_SCLn/LCDVD[14]/LCDVD[22] | | | | | | | |
| 66 | P1[29] | GPIO Port 1.29 | USB_SDAIn/LCDVD[15]/LCDVD[23] | | | | | | | |
| 67 | P1[30] | GPIO Port 1.30 | | | | | | | | |
| 68 | P1[31] | GPIO Port 1.31 | | | | | | | | |

- ▶ Example pin configuration (GPIO pin P1.18 high = default backlight off):

```
#define PINCFG_P1_18 0 // GPIO Port 1.18
//#define PINCFG_P1_18 1 // USB_UP_LED1
//#define PINCFG_P1_18 2 // PWM1[1]
//#define PINCFG_P1_18 3 // CAP1[0]
#define PINSET_P1_18 1 // Set
#define PINCLR_P1_18 0
#define PINDIR_P1_18 1 // Output
#define PINMODE_P1_18 0 // Pull Up
```

For More Information
On uEZ® or the uEZ® GUI Family



uEZ GUI Family - www.uezgui.com

uEZ Source Code and Documentation
www.sourceforge.net/projects/uez

Other LCD sizes or uEZ GUI Customization
sales@teamfdi.com