µEZ™ Overview

The Rapid Development Platform

Muse

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

Overview

- What is µEZ™?
- µEZ™ RTOS Engine
- µEZ™ Four Tier Hierarchy
- Reusable HAL and Device Drivers
- LPC2478 & LPC2362 Example
- FDI and Community Support Network
- Micrium Comparison
- Field Upgradability
- 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

FDI Future Designs, Inc. Your Development Partner

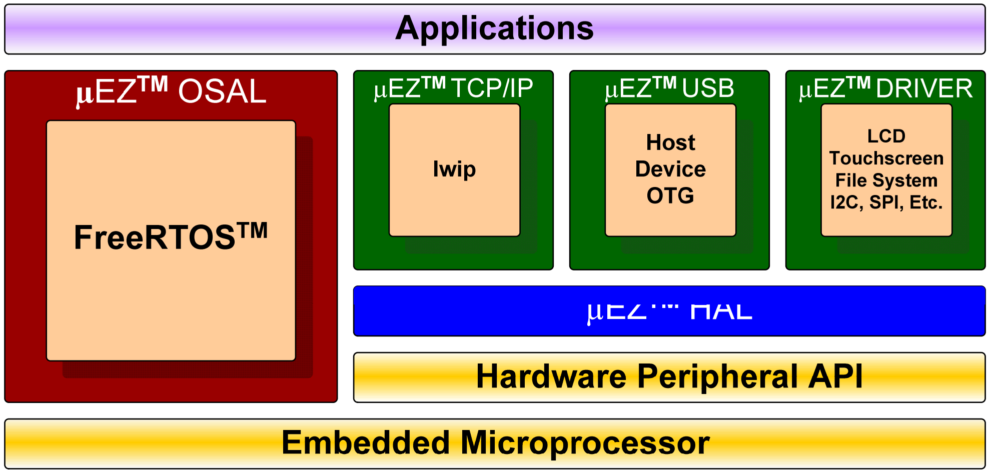
µEZ™ Value Proposition

- 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, ***<u>but</u>***
  - –COM Drivers & Stacks are part of the 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

ARM®

What is µEZ™ ?

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



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

What is µEZ™ ?

- 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
  - 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

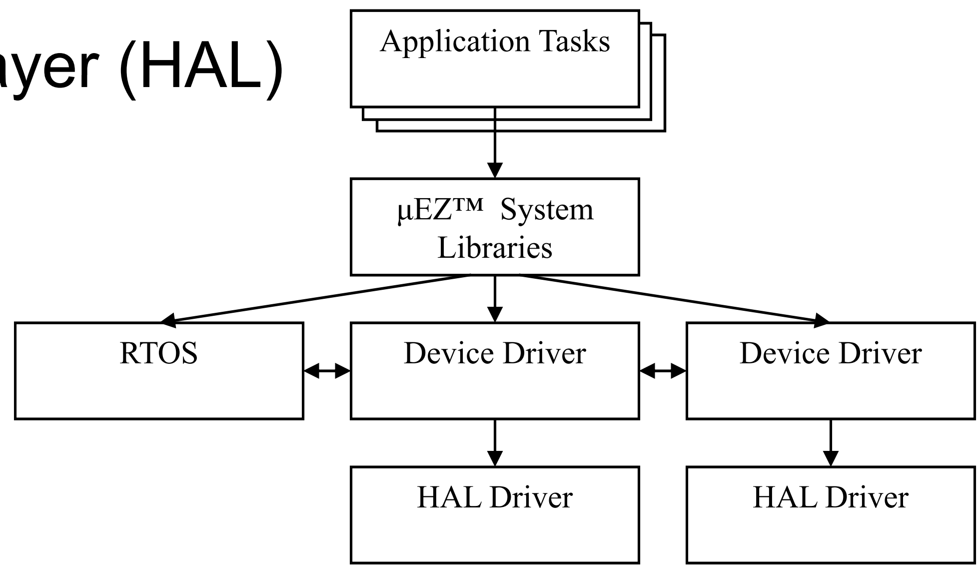
Preliminary data based on uEZ™ V 0.11

The µEZ™ Engine – RTOS

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

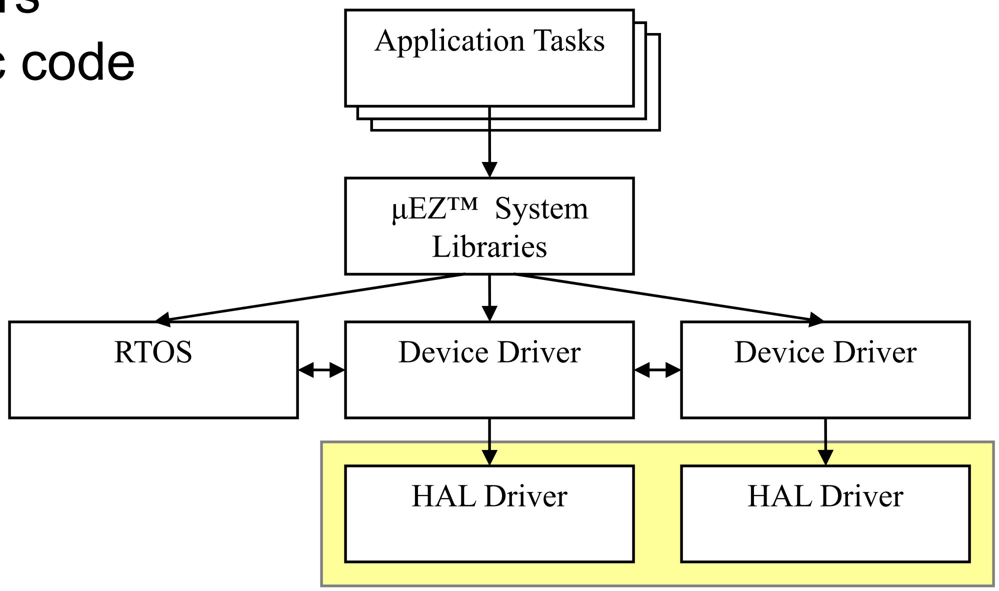
µEZ™ Four Tier Hierarchy

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



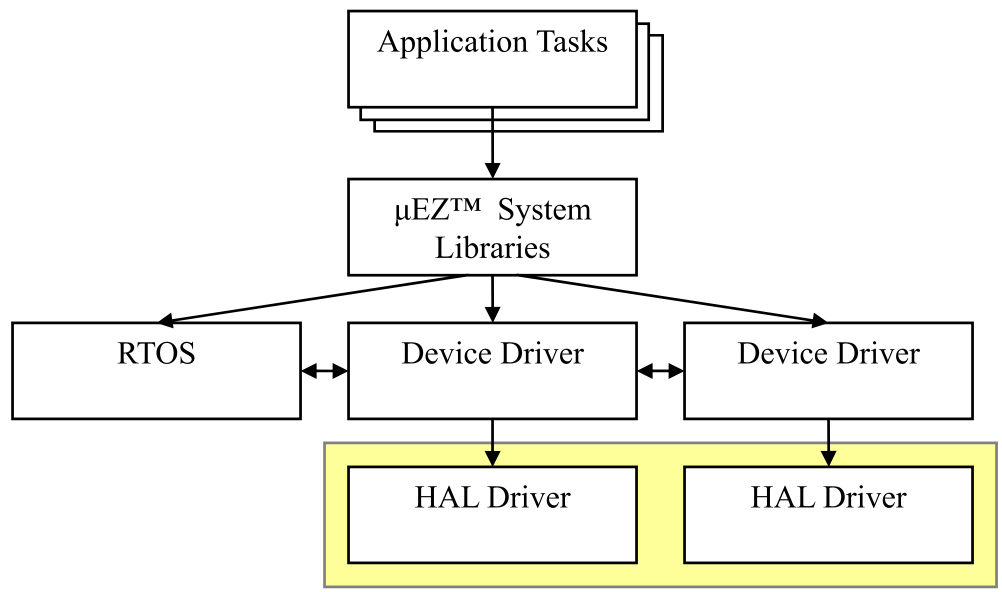
Building Up: HAL Drivers

- 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



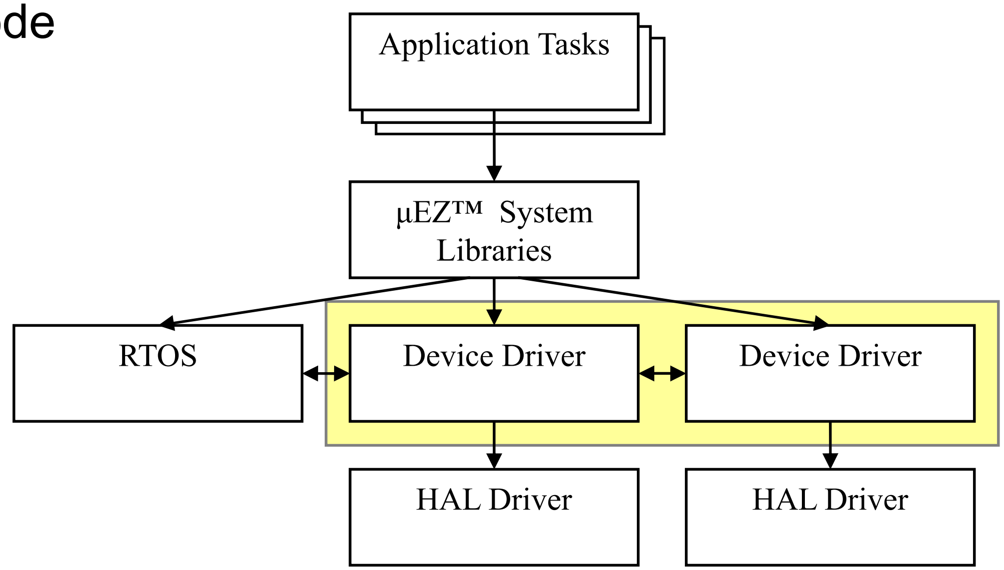
Example – LPC2478 HAL Package

- LPC2478 BSP Package
  - ADC
  - Ethernet MAC
  - GPIO
  - I2C
  - Interrupts
  - LCD Controller
  - PWM
  - RTC
  - SPI/SSP
  - Timers
  - UARTs
  - USB Device Controller
  - USB Host



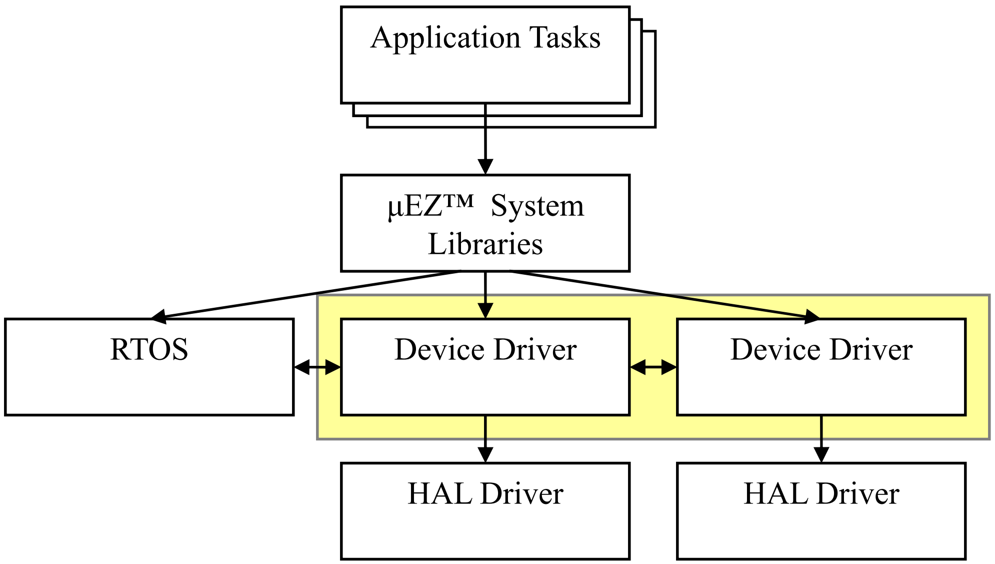
Building Up: Device Drivers

- 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



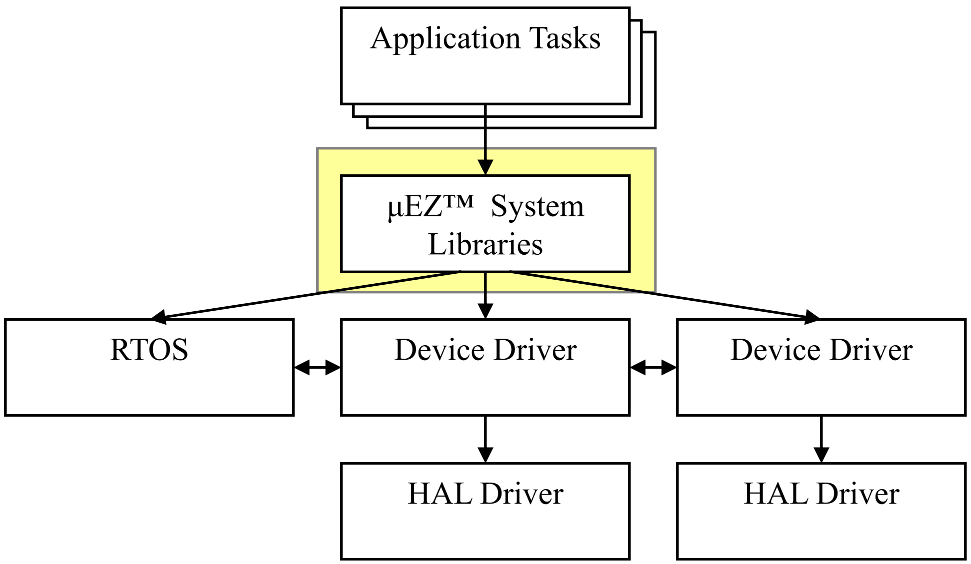
Example – Platform Device Drivers

- CARRIER Device Drivers Package
  - Accelerometer (I2C)
  - Temperature Sensor (I2C)
  - ADC
  - Backlight (PWM)
  - Buttons (I2C)
  - EEPROM (I2C)
  - I2C
  - LCD
  - LED (I2C)
  - Mass Storage (SPI/USB)
  - RTC (I2C)
  - UART
  - SPI/SSP
  - USB Device



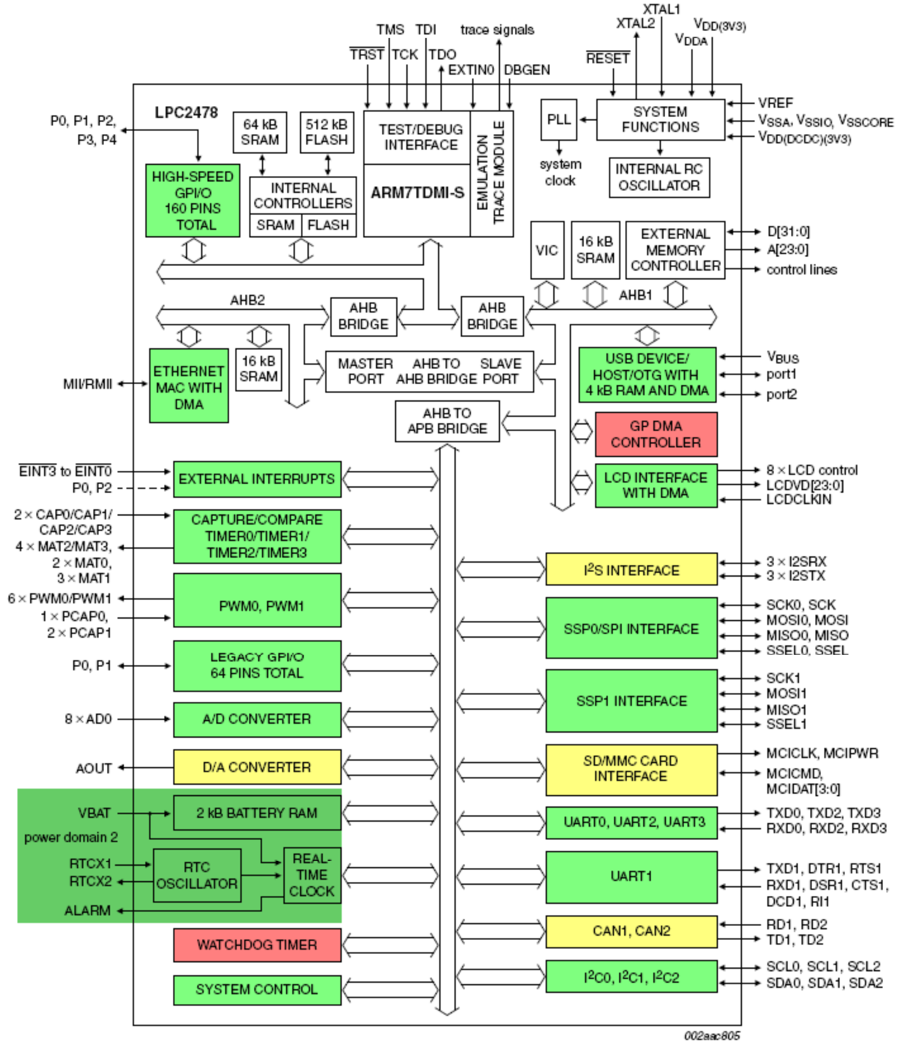
µEZ™ System Libraries

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



µEZ™ LPC2478 Support

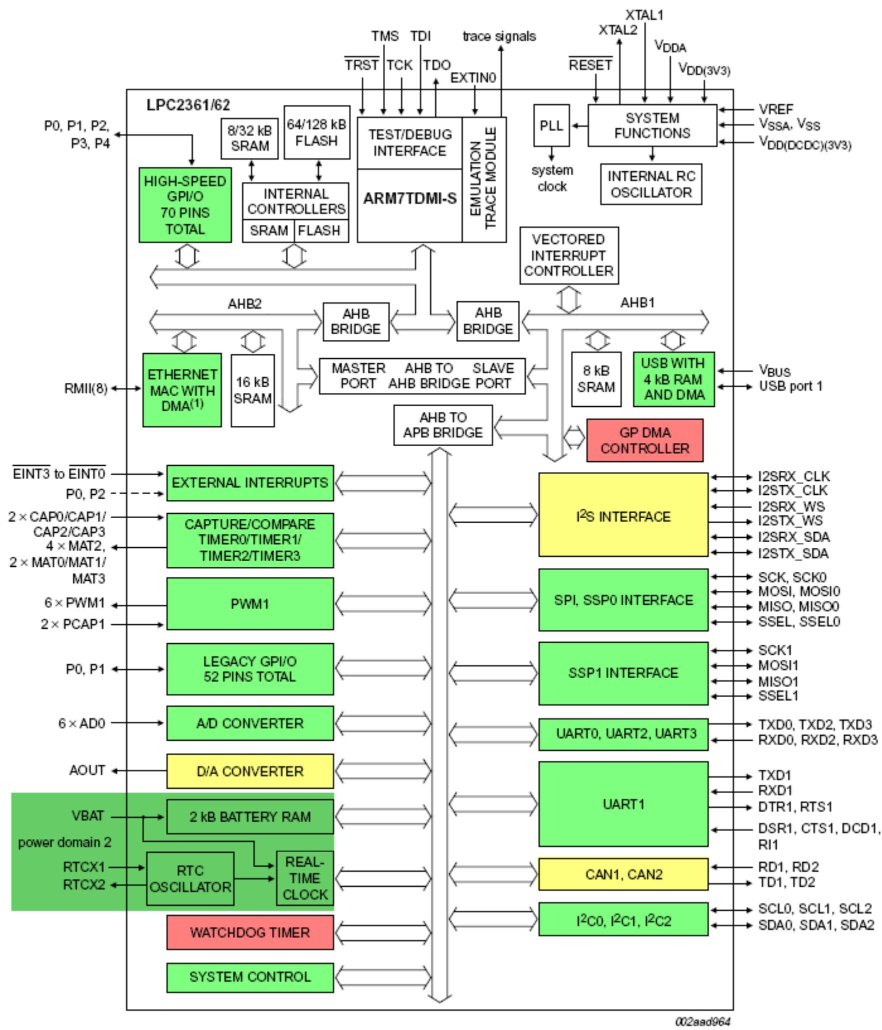
- **Available Now**
  - GPIO
  - A/D
  - PWM
  - RTC
  - USB
  - SSP
  - SPI
  - UART
  - I2C
- **Coming Soon**
  - D/A
  - I2S
  - MMC Card
  - CAN
- **Future**
  - Watchdog
  - GP DMA



LPC2478

µEZ™ LPC2362 Support

- **Available Now**
  - GPIO
  - A/D
  - PWM
  - RTC
  - USB
  - SSP
  - SPI
  - UART
  - I2C
- **Coming Soon**
  - D/A
  - I2S
  - CAN
- **Future**
  - Watchdog
  - GP DMA
- Reuse LPC2478 Source Code!



LPC2362

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 µEZ™ Comparison

Module	Brand M Flash size	Brand M RAM	uEZ Flash size	uEZ RAM
BSP	8,503	32	20736	8406
uC/LCD	384	6	4,296	772
App tasks	8,697	4,133	12,893	3392
uC/USB Host	26,565	10,669	6520	313
uC/USB Device	7,410	513	10093	1187
uC/LIB	19,744	228	13,455	388
uC/OS	8,898	7,584	22,249	9,868
uC/HTTP	4,236	6,696	921	600
uC/TCP-IP	77,634	24,531	56,895	23,868
uC/FS	17,124	565	20,091	1,156
Total	**179,195**	**54,957**	**154,232**	**48,234**

Preliminary data based on uEZ™ V 1.0

Field Upgradability and µEZ™

- Primary Boot Loader Options
  - JTAG
  - ISP Flash
- Secondary Boot Loader
  - Stand-alone µEZ™ 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 µEZ™

- I2S Audio HAL and Device Driver
- Nano-X Graphics Library
- USB – switching support for Host/Device on one port
- Ongoing Processor Support
  - Cortex-M3 (LPC17xx Family)
  - ARM9 (LPC3250)
- Improved Make System

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

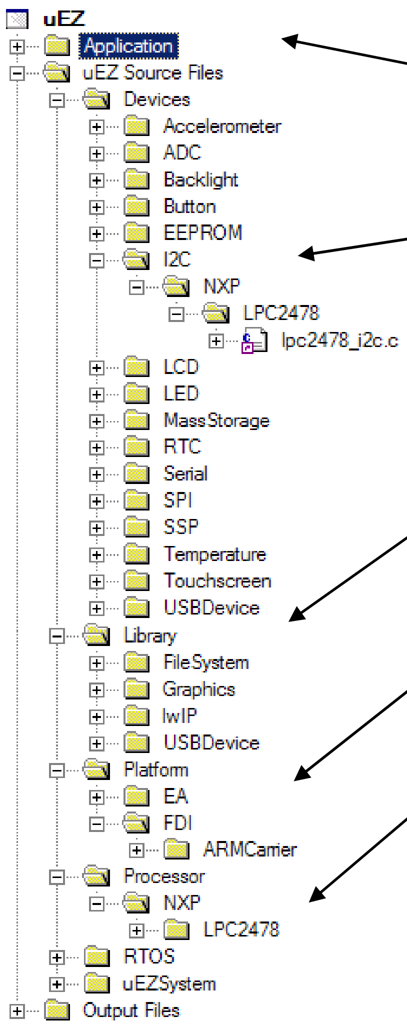
Example µEZ™ System Library – I2C Bus

- 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



- Project files are organized by major functionality
  - Application
    - Customer and Application specific files
  - Devices
    - Devices are stored in Interface -> Manufacturer -> instance folders
  - Library
    - Libraries are grouped by Type (e.g. Graphics, Network, etc.) and then subdivided by project library
  - Platform
    - Manufacturer -> Product
  - Processor
    - Manufacturer -> Family -> Product
  - RTOS
    - Folder per RTOS
  - µEZ™System
    - Generic µEZ™ support files

Smaller is Better – µEZ™ Compile Options

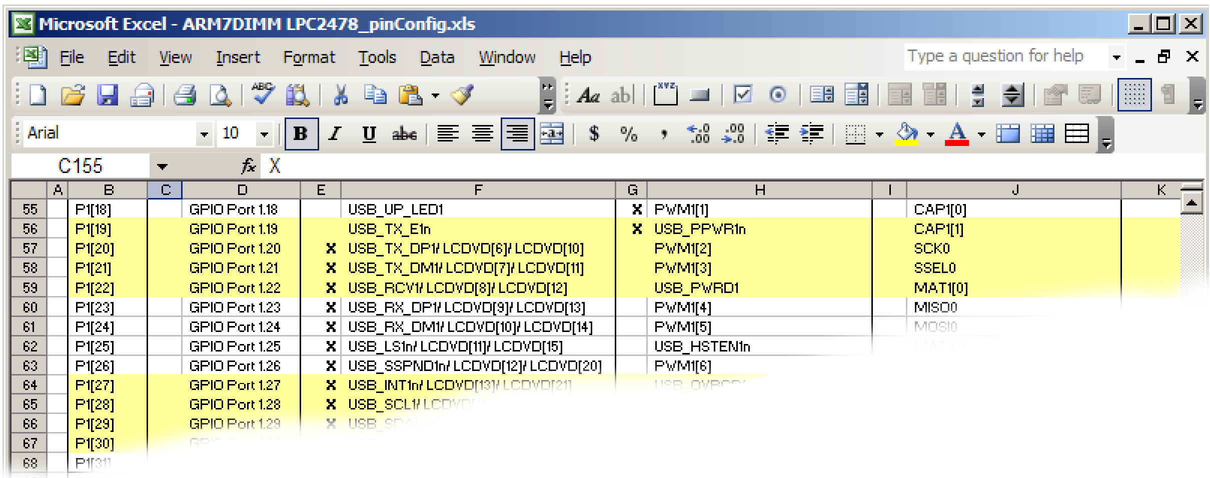
- Full customization of all components
  - #defines are used to enable/disable components
- Three configuration files are used
  - Application Configuration File
  - Platform Configuration File
  - Processor Configuration File
- Each layer of configuration can override the lower level
  - Application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

Startup

- Bootloader (outside of µEZ™ 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 utilty
  - 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_E1n	X	USB_PPWR1n		CAP1[1]	
57		P1[20]		GPIO Port 1.20	X	USB_TX_DP1/ LCDVD[6]/ LCDVD[10]		PWM1[2]		SCK0	
58		P1[21]		GPIO Port 1.21	X	USB_TX_DM1/ LCDVD[7]/ LCDVD[11]		PWM1[3]		SSEL0	
59		P1[22]		GPIO Port 1.22	X	USB_RCV1/ LCDVD[8]/ LCDVD[12]		USB_PWRD1		MAT1[0]	
60		P1[23]		GPIO Port 1.23	X	USB_RX_DP1/ LCDVD[9]/ LCDVD[13]		PWM1[4]		MISO0	
61		P1[24]		GPIO Port 1.24	X	USB_RX_DM1/ LCDVD[10]/ LCDVD[14]		PWM1[5]		MOSI0	
62		P1[25]		GPIO Port 1.25	X	USB_LS1n/ LCDVD[11]/ LCDVD[15]		USB_HSTEN1n			
63		P1[26]		GPIO Port 1.26	X	USB_SSPND1n/ LCDVD[12]/ LCDVD[20]		PWM1[6]			
64		P1[27]		GPIO Port 1.27	X	USB_INT1n/ LCDVD[13]/ LCDVD[21]					
65		P1[28]		GPIO Port 1.28	X	USB_SCL1/					
66		P1[29]		GPIO Port 1.29	X						
67		P1[30]									
68											

- 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
```

Thank You