OULU UNIVERSITY OF
APPLIED SCIENCES

Teemu Kangasharju

SOFTWARE ARCHITECTURE FOR LOW TIER EMBEDDED PRODUCT PLATFORM

SOFTWARE ARCHITECTURE FOR LOW TIER EMBEDDED PRODUCT PLATFORM

Teemu Kangasharju
Master's thesis
Autumn 2012
Degree program in Information Technology
(Master of Engineering)
Oulu University of Applied Sciences

ABSTRACT

OULU UNIVERSITY OF APPLIED SCIENCES

Degree programme	Thesis	Pages / enclosures
Degree Programme in Information Technology	Master's Thesis	59 / 3
Line	Date	
	2012	
Commissioned by	Author	
Lewel Group Finland Oy	Teemu Kangasharju	

Thesis title

SOFTWARE ARCHITECTURE FOR LOW TIER EMBEDDED PRODUCT PLATFORM

The purpose of this thesis is to describe embedded software platform architecture for low tier applications. The point of view of this software platform description is hardware oriented. General trends influence the solutions that have been selected to the final architectural design of the software platform. The Open Source Software (OSS) is becoming more common in the software product development. Many chip manufacturers offer a comprehensive baseline for the development software and the whole embedded system. Development communities and system development companies re-develop the open software components and hardware components to final, type approved products.

Nowadays it is very common that a customer has some product or business idea which should be processed to a final product onto the market. The customer in this case means the organization who wants to purchase an R&D and/or productization project from someone else. Other starting points for this work are low tier and low cost aspects. The processor / microcontroller and Real Time Operating System (RTOS) choices are significant factors when designing a modular and cost efficient embedded system.

The main aim of the thesis is to create architectural design for an embedded product platform. It gives the preliminary solutions to develop modular, configurable, and maintainable embedded systems.

Keywords: Embedded software platform, Software architecture, Low tier product platform, RTOS, Embedded systems, Open source software, Wireless

PREFACE

I graduated as an Engineer of Computer Science from Raahe Institute of Computer Science in 2001. The first couple of years after the graduation I worked at Buscom Oy. Buscom Oy is known as Fara Oy nowadays. My responsibilities were smart card related tasks in software development. I left Buscom Oy at the end of 2006.

I started at Sweco Industry Oy at the beginning of 2007 as software specialist. Sweco Industry is a consultant house. I started in a team that developed an automated test system for digital Application Specific Integrated Circuits (ASIC) in the cellular phone platform development. The work tasks were embedded software mainly. In April 2010 Lewel Group Finland Oy bought the electronic and mechanic departments from Sweco Industry and the employees continued as old workers.

In the autumn of 2010 we started to plan the Fast Prototyping Platform to boost an entire Research and Development (R&D) project selling for customers speeding up the product development. I have been the software architect in the development. Low Tier Device (LTD) platform development started in the autumn of 2011. The engineering work of the LTD Platform has been done as collaboration with Oulu and Kuopio people. This thesis describes the work of the LTD platform software architecture.

I would like to thank all Lewel people which are involved in my daily work. I thank employees which participated to the LTD platform development; Aleksi Ukkola, Mika Tapaninaho, Markus Paldanius, Kari Salo, Juha Kortesalmi, Mikko Alafrantti, Tommi Takkinen and as well as Marko Pylkkänen and Markku Soininen the professionals of Kuopio. They all ensured that the thesis reached its target. I would like to direct special thanks to Hannu Ylönen, the system architect of LTD and the mentor of this thesis in addition, I thank the thesis supervisor Kari Laitinen from OUAS.

Oulu, Finland, August 2012

Teemu Kangasharju

CONTENTS

TERMS AND ABBREVIATIONS

LTD	Low Tier Device
API	Application Programming Interface
LCD	Liquid Crystal Display
ASIC	Application Specific Integrated Circuit
HAL	Hardware Abstraction Layer
BSP	Board Support Package
RTOS	Real Time Operating System
MQX	Freescale RTOS
FFS	Flash File System
MFS	MQX File System
OTG	On The Go
PM	Power Management
HMI	Human Machine Interface
SoC	System on Chip
SDK	Software Development Kit
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
BT	Bluetooth
BLE	Bluetooth Low Energy

PXP	BLE Proximity Profile
GAP	Generic Access Profile
GATT	Generic Attribute Profile
WLAN	Wireless Local Area Network
OSS	Open Source Software
I2C	multi-master serial single-ended computer bus
SPI	Serial Peripheral Interface
SDIO	Secure Digital Input Output
GPL	General Public License
LSB	Least Significant Bit
VLPS	Very Low Power Stop mode
VLPR	Very Low Power Run mode
PGA	Pin Grid Array
FAT	File Allocation Table
GUI	Graphical User Interface
RTC	Real Time Clock
PLL	Phase Lock Loop
FLL	Frequency Lock Loop
MCG	Multipurpose Clock Generation
FBE	FLL Bypassed External

1 INTRODUCTION

This chapter introduces the focus of the thesis. It describes special features of an embedded software development. In addition, the idea of the embedded software platform is presented shortly in this chapter.

1.1 Embedded Software Development

When planning software for embedded systems, it has to be considered from a different point of view than in the general software development. Embedded software often has to deal with memory constraints, critical timing demands, power consumption demands, etc. When constructing embedded systems to some specific areas such as medical, automotive, telecommunications or consumer electronics, these rules and constraints should be applied into development. It is a common trend that embedded systems have to be small, cheap and powerful.

1.2 The Idea of Embedded Software Product Platform

Nowadays it is very common that a customer has some product or business idea which should be processed to final a product onto the market. The customer in this case means the organization who can be a product owner, who wants to purchase an R&D and/or productization project from someone else. They have not necessarily the engineering resources and knowledge to get progress for it. This thesis gives some answers for these demands from the software perspective. The thesis presents one solution to create an embedded software platform.

This Software architecture design is divided into two main software parts: Application Programming Interface (API) and Hardware Abstraction Layer (HAL). These parts should be operated as independently as possible of application functionality and hardware solution. It is obvious the customer's needs influence these layers, but target is that customer specific

needs can be just an API service like some specific measurement technique controlling interface for application. Hardware-wise the product platform can be like a reference design, and the actual research and development work tailors it to customer specific format.

The main focus areas of the thesis work are to plan modularity and maintainable software architecture and to observe and investigate the portability of different wireless protocol stacks. Bluetooth (BT), Bluetooth Low Energy (BLE) and Wireless Local Area Network (WLAN) are the wireless technologies under review. The modularity aspect contains how easily the customer needs can be ported into basic platform set up and how easily it can be configured to another application requirements. Wireless protocol stacks are usually licensed. BLE is quite new technology, are therefore the code portability could be a challenge.

The other starting points for this master's thesis work are low tier and low cost aspects and thus the processor / microcontroller and real time operating system (RTOS) choices are significant factors when designing the modularity and cost efficient embedded system. The design tool choices allow effective designing work. The selections of the processor architecture and preliminary RTOS choice have been done before this thesis. The design processes influence the progressing of the thesis work.

One aim is to create architectural design for an embedded product platform. It gives the preliminary solutions to develop a modular, configurable and maintainable embedded software platform. Another target of this work is to open a wireless protocol stack for portability, giving guidelines for software developing work. In addition, different business areas and their standards are challenges for product designing processes.

1.3 Outline of the Thesis

Chapter 2 describes a technological and business based background of this work. It is followed by the problem research. Chapter 3 presents the requirements for this software

platform, and chapter 4 discusses the actual work. In that chapter the architecture of the software platform and its different software components are described. Chapter 5 contains a discussion about further development possibilities.

2 EMBEDDED SOFTWARE PLATFORM

This chapter describes a technological framework to develop an embedded system. It also presents the technical trends, which are common in the embedded product development nowadays.

2.1 Product Development

Naturally, development of an entire embedded system takes time. A product development of a certain business area or technology branch (as medical, automotive, telecommunications or consumer electronics) has a big impact onto the work amounts of the system development. That is because of the standard requirements of specific technology branches. The requirement of the product life cycle management (PLM) differs widely depending on the business branch. Usually, the intent is that the product has to be launched onto the market as soon as possible.

2.2 Technical Trends

The open source software (OSS) is becoming more common in the software product development. Many chip manufacturers offer a comprehensive baseline for the development software and the whole embedded system development. One business philosophy of them is to boost their electronic component selling. They open their reference designs to all, and the development communities and system development companies re-develop the open software components and hardware components ahead to the final, type approved product. Some good examples of that kind of companies are STMicroelectronics, Texas Instrument (TI) and Freescale Semiconductors. Both sides, the re-developers and component manufactures, can have benefits:

- The chip manufacturers boost their business. As usual, when more and more manufacturers sell their components, components are becoming cheaper.

- The systems development ahead, and in that way the new innovations come up more easily.
- The systems can become more reliable when it is controlled by communities.

Along the new technology development the companies make more collaboration. For example, some parts of the development or evaluation boards are compatible between different manufacturers.

The embedded systems are becoming more powerful and smaller, what is obvious, but on other hand the battery technology development does not progress so rapidly. In many cases the battery size is bigger than the other device assembly. Wireless subsystems are coming into the embedded devices. It sets challenges for minimizing the power consumption of the hand held and battery operated embedded devices.

2.3 Wireless Subsystems

Electronic systems are increasingly becoming wireless. Viewing the battery operated devices in general, the wireless subsystems take a significant part of the entire power consumption of the embedded device. It is an obvious direction of the wireless technology progress. Bluetooth (BT) has evolved to Bluetooth Low energy (BLE).

Another prevailing research direction is to get a higher data transferring rate. BT based systems can be adapted to the entire system via a serial link as the Serial Peripheral Interface (SPI) bus. WLAN has two kinds of physical connection ways such as SPI and Secure Digital Input Output (SDIO). Basically, the SPI connection limits the speed of the data transferring.

Usually, the BT software stacks are licensed strictly. Therefore the stacks are not free although the BT hardware reference designs are available. The solution from the OSS side

can be found for base porting it into the embedded system. Progressing that way it is very important to know the open source licensing policy. The most common license type there is the General Public License (GPL) in the OSS world.

3 RESEARCH PROBLEM

This chapter describes the approach to a principled problem of the product development. In addition, the requirements of the LTD product platform software are shown in this chapter.

3.1 Problem Definition

The problem is surveyed from two directions, the business oriented approach and the product development oriented approach.

3.1.1 Business Oriented Approach

Basically, a customer has some product or business idea which should be processed to the final product onto the market as fast as possible. Briefly, the more ready a product platform is for the customer requirements, the faster the product will be on the market. Many product creation houses are developing their own platforms, and therefore competition is quite tight in this business area, which means that company has to focus on the special knowledge of them. For example, the specialty can be knowledge of some physical phenomenon or specific business sector knowhow. In this case the focus is to do an embedded product platform for low tier applications which do not have for example wireless data transfer features.

The following are competing platforms which are on the market already:

- Navicron Oy – FUSIONSOFTWARE (1).
- Aava Mobile Oy – Aava Core (2).
- ERNI – WHITEspeed (3).

3.1.2 Product Development Oriented Approach

From a technical point of view the platform development needs comprehensive knowledge about the embedded system architectures and their subsystems. An effective and wide partner network is an important part of the system and platform development. Usually, big component manufactures do not collaborate and therefore it is important to create connections to the suppliers who can handle a component distribution effectively. A direct connection with component manufacturers means either big volumes of the end product or co-operation with well-known company.

This software platform work is based on the System on Chip (SoC) of Freescale. Freescale Semiconductors is a big worldwide company. Their business philosophy is a somewhat different from that of their competitors. They open their system giving source codes of their own platform modules and they support all kinds of companies, smaller, bigger and start ups. In that way they boost their component selling. For this kind of a low tier product platform Freescale has, for example, its own open real time operating system (RTOS) named MQX. The kernel of MQX is open, Board Support Package (BSP) level code is open, and that means, a system creator has a base for starting then own platform or product development.

In an ideal situation reprocessing the end product for the customer means that existing hardware and software components are integrated in a customer specific product development project.

An expanded component library and good knowledge of the platform environment offer faster feedback for customer requirements. The following section specifies the requirement for a platform.

3.2 Technical Requirements for LTD Product Platform

The most important features to be planned were that the platform should be purposed to small, portable devices. The platform for the end product should be power consumption, size and cost efficient. The application of the device has not necessarily a support of the display component. The indication of application state can be solved by the LED blink sequence, for example.

The LTD hardware platform must be equipped with the following functional sub modules:

- CPU based on Cortex-M4 architecture. Freescale Kinetis K20
- USB 2.0 interface (LS/FS)
    - On The Go (OTG) support
- PM (Power management)
- Liquid Crystal Display (LCD) I/F via GPIO
    - Three LEDs & buttons
- Optional Bluetooth, BT 4.0. BLE
- I/O connector for external I/O components and peripheral devices

3.2.1 Software Requirements

At the general level, the most important software requirements for LTD are related to the power consumption and code size efficiency. The operation modes of SoC should be used widely to get better efficiency. The RTOS capacity has to utilize as well as possible concerning the tight real time requirements. The software architecture should be modular the way that the new customized software components can be fitted easily to the existing system. In addition, the maintainability should be taken into account. The requirements for the LTD product platform software are:

- Freescale MQX Real Time Operating System support
- Kernel (PSP) and BSP base porting for LTD hardware

- I/O drivers for equipped hardware components of LTD
    - GPIOs, I2C, SPI, Flextimer, Analog to Digital Converters (ADC), Digital to Analog Converters (DAC) etc.
- USB 2.0 interface (LS/FS) support
    - On the Go (OTG) support
    - OTG communication between LTD device and mobile phone (e.g. Samsung Galaxy S2)
    - BLE
- Proximity (PXP) profile support for both monitoring and reporting
- LTD should be worked as monitor and reporter
    - UI
    - Sharp 96*96 graphical LCD support
    - EQUI support
    - Led & Button functionality
- Test Application
    - Tests functions for CPU and peripherals
    - Test function menu is shown in LCD
    - Test function selection by buttons
    - Result of tests are indicated within LCD, LEDs and beeps of buzzer
- Detailed specification for LTD test application is described in the document LTD Test SW Design. (4).

4 LTD SW ARCHITECTURE

This chapter describes the results of this work. The LTD software architecture will be cleared out in more detail in section 4.2.

4.1 Introduction

This section and its sub-sections work as an introduction for the actual work of the architecture design. The point of view of this software platform description is hardware oriented. The planning of the LTD Software Architecture needs a great deal of investigations and studies of the Freescale product family. The Kinetis chip series is quite a new product family. It is based on the ARM Cortex M4 core microcontroller. Freescale has its own TWR (Tower) development platform to evaluate the Kinetis microcontrollers and peripheral devices.

The first steps to create the entire LDT platform have been done in the TWR evaluation environment. The SoC Choice for LTD was Kinetis K20. Different Kinetis SoCs have the same core. Only some IC blocks are activated in different Kinetis configurations as can be seen in figure 1. Every Kinetis family K20 has many package and memory based variants.

One-Stop Enablement Offering—MCU + IDE + RTOS

Freescale Tower System hardware development environment:

- Integrated development environments
  - Eclipse-based CodeWarrior V10.x IDE and Processor Expert
  - IAR Embedded Workbench
  - Keil MDK
  - CodeSourcery Sourcery G++ (GNU)
- Runtime software and RTOS
  - Math, DSP and encryption libraries
  - Motor control libraries
  - Complimentary bootloaders (USB, Ethernet, RF, serial)
  - Complimentary Freescale embedded GUI
  - Complimentary Freescale MQX™
  - Cost-effective Nano™ SSL/Nano™ SSH for Freescale MQX RTOS
  - Micrium uC/OS-III
  - Express Logic ThreadX
  - SEGGER embOS
  - freeRTOS
  - Mocana (security)
- Full ARM ecosystem

Features	Benefits
• ARM® Cortex™-M4 core with DSP instruction support and optional single precision floating point unit • Up to 32-channel DMA. Up to 16 KB of cache. Cross bar switch	• Up to 120 MHz core supporting a broad range of processing bandwidth needs • Peripheral and memory servicing with reduced CPU loading. Optimized bus bandwidth and flash execution performance. Concurrent multi-master bus accesses for increased bus bandwidth
• USB On-The-Go (Full- and High-Speed) with device charger detect	• Optimized charging current/time for portable USB devices, enabling longer battery life. USB low-voltage regulator supplies up to 120 mA off chip at 3.3V to power external components from 5V input
• Memory protection unit • Hardware cyclic redundancy check engine • Independent-clocked COP. External watchdog monitor	• Provides memory protection for all cross bar switch masters, increasing software reliability • Validates memory contents and communication data, increasing system reliability • Prevents code runaway in fail-safe applications. Drives output pin to safe state external components if watchdog event occurs
• Up to four FlexTimers with up to 20 channels • Carrier modulator transmitter • 4-channel, 32-bit periodic interrupt	• General purpose timers with hardware dead-time insertion and quadrature decoding for motor control • Infrared waveform generation for remote control applications • Time base generation for RTOS task scheduler or trigger source for ADC conversion and programmable delay block
• FlexBus external bus interface • Secure digital host controller • NAND flash controller	• Enables the connection of external memories and peripherals (e.g., graphics displays) • Connection to SD, SDIO, MMC or CE-ATA cards for in-application software upgrades, file systems or adding Wi-Fi® or Bluetooth® support • Supports up to 32-bit ECC current and future NAND types with minimal software overhead
• 32 KB–1 MB flash. Up to 128 KB of SRAM • 32 KB–512 KB FlexMemory	• High reliability, fast access program memory with 4-level security protection. Independent flash banks allow concurrent code execution and firmware updating • FlexMemory provides 32B–16 KB of user-segmentable byte write/erase EEPROM. In addition, FlexNVM from 32 KB–512 KB for extra program code, data or EEPROM backup

K20 Family Options

Part Number	Memory					Feature Options									Other	Packages											
	CPU (MHz)	Flash (KB)	Flex NVM (KB)	SRAM (KB)	Cache (KB)	Single Precision Floating Point Unit	Memory Protection Unit	CAN	Secure Digital Host Controller	NAND Flash Controller	External Bus Interface	12-bit DAC	Prog. Gain Amplifier	5V Tolerant I/O		FM	FT	LF	MP	LH	LK	MB	LL	ML	MC	LQ	MD
																32QFN (5x5)	48QFN (7X7)	48LQFP (7X7)	64MAPBGA (5X5)	64LQFP (10X10)	80LQFP (12X12)	81BGA (8X8)	100LQFP (14X14)	104BGA (8X8)	121BGA (8x8)	144LQFP (20x20)	144BGA (13x13)
MK20DN32Vyy5	50	32		8											USB OTG (FS)	√	√	√	√	√							
MK20DN64Vyy5	50	64		16											USB OTG (FS)	√	√	√	√	√							
MK20DN128Vyy5	50	128		16											USB OTG (FS)	√	√	√	√	√							
MK20DN512Vyy10	100	512		128			√	√	√		√	√	√	√	USB OTG (FS)						√	√	√		√	√	√
MK20FN1M0Vyy12	120	1 MB		128	16	√	√	√	√	√	√	√	√	√	USB OTG (FS/HS)											√	√
MK20DX32Vyy5	50	32	32	8											USB OTG (FS)	√	√	√	√	√							
MK20DX64Vyy5	50	64	32	16											USB OTG (FS)	√	√	√	√	√							
MK20DX128Vyy5	50	128	32	16											USB OTG (FS)	√	√	√	√	√							
MK20DX64Vyy7	72	64	32	16				√			√	√	√	√	USB OTG (FS)					√	√	√					
MK20DX128Vyy7	72	128	32	32				√			√	√	√	√	USB OTG (FS)					√	√	√	√	√			
MK20DX256Vyy7	72	256	32	64				√			√	√	√	√	USB OTG (FS)					√	√	√	√	√			
MK20DX128Vyy10	100	128	128	32			√	√	√		√	√	√	√	USB OTG (FS)											√	√
MK20DX256Vyy10	100	256	256	64			√	√	√		√	√	√	√	USB OTG (FS)						√	√	√		√	√	√
MK20FX512Vyy12	120	512	512	128	16	√	√	√	√	√	√	√	√	√	USB OTG (FS/HS)											√	√

yy = package designator

FIGURE 1. K20 based variants

The LTD product platform software is based on the Freescale Kinetis K20 SoC solution. The platform has an MQX RTOS operating system. The collection of the MQX documentation can be found in the project folder located on LTD SW\Freescale_MQX\.

The software is created on the IAR embedded workbench development environment. The basic MQX RTOS packet does not include a Graphical User Interface (GUI) or a USB

OTG stack. The Kinetis K20 Family has many variations of memory, package and feature options.

The architecture of the LTD Software consists of the following layers:

- Application layer
- API Services (RTOS, Tasks) LTD_API + customer specific components
    - Starting point of MQX version is MQX 3.70
- Hardware abstraction layer LTD_HAL BSP

The processor component used in the design is actually a SoC on Cortex M4 CPU. The LTD uses the Freescale Kinetis K20 series processor. Figure 2 presents a top level block diagram of K20. The supported functions are dependent on the selected package. The largest 144 pin package is used in this prototype, because it allows the flexible I/O configuration. The customer application could use smaller pin counts if they are feasible.

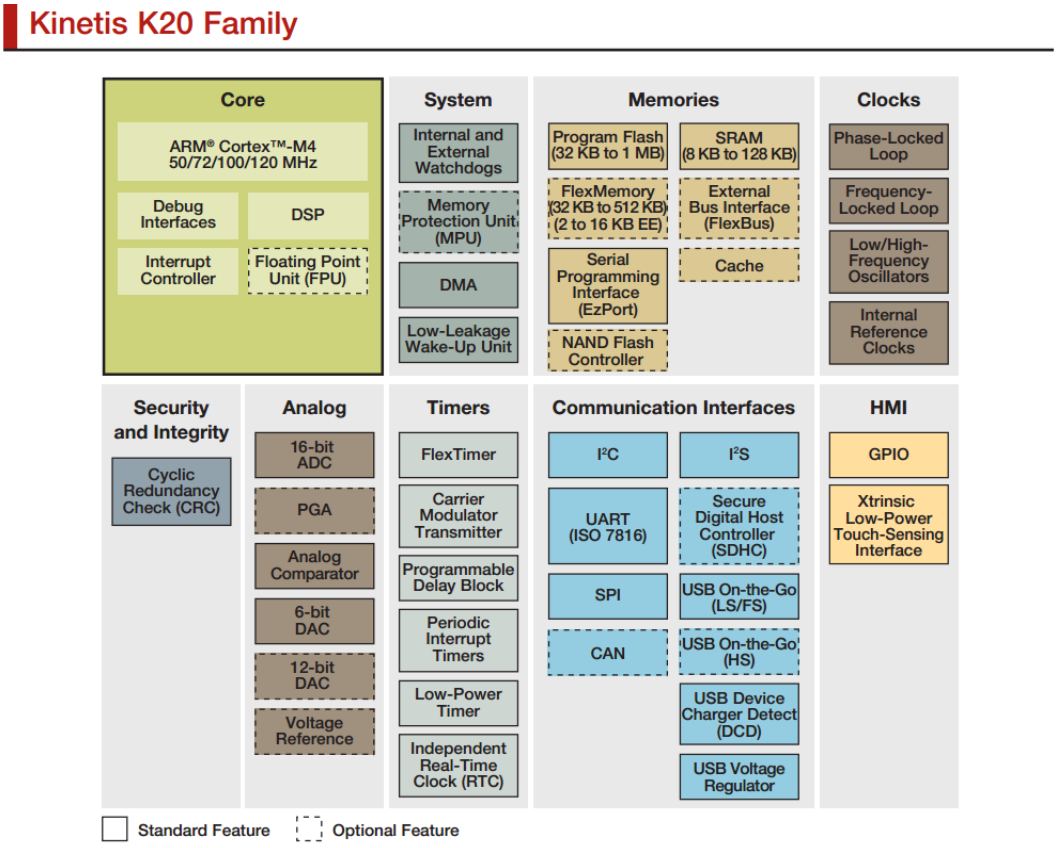

FIGURE 2. Kinetis K20 System on Chip

4.1.1 LTD API Services

The purpose for this Software layer is to handle overall system resources, both hardware and software. For example, the measurement algorithms and device driver logic are built onto these level modules. API services are the interface to build a comprehensive application including RTOS resources.

LDT_API offers the following services:

- General LTD_API services
- MQX 3.7.0 RTOS features
- I/O configurations
- Specific HW initializations

- System recourses
- UI Services
    - Buttons
    - LEDs
    - Display functionality with graphics
- Communication Services
    - Tasks intercommunication
    - USB communication with another host device. It is handled by the USB OTG module.
    - BT Low Energy protocol support. The proximity profile will be supported.
- Flash File System (FFS) handling services
    - File system handling
    - Application specific results can be stored in the flash memory.
- Measurement (imaginary) Services - Customer specific
    - Current pulse generation with Flex timer0 and DAC
    - Current pulse measurement via ADC
- PM Services
    - Operation mode handling
    - Sleep mode handling
    - Voltage monitoring and control

4.1.2 Operating System – MQX

The manufacturer of the used operating system describes it in the following way.

> The Freescale MQX Real-Time Operating System (RTOS) provides real-time performance within a small, configurable footprint. This RTOS is designed to allow you to configure and balance code size with performance requirements. The easy-to-use API and out-of-box experience ensures first-time RTOS users can start developing their application on the day software is installed. For experienced OS developers, it is easy to migrate legacy application code to a Freescale MQX-based platform. The RTOS is tightly integrated with the latest ColdFire® processors from Freescale and provided with commonly used device drivers. The powerful Design and Development Tools are

integrated with CodeWarrior™ tools to provide additional profiling and debugging capability.

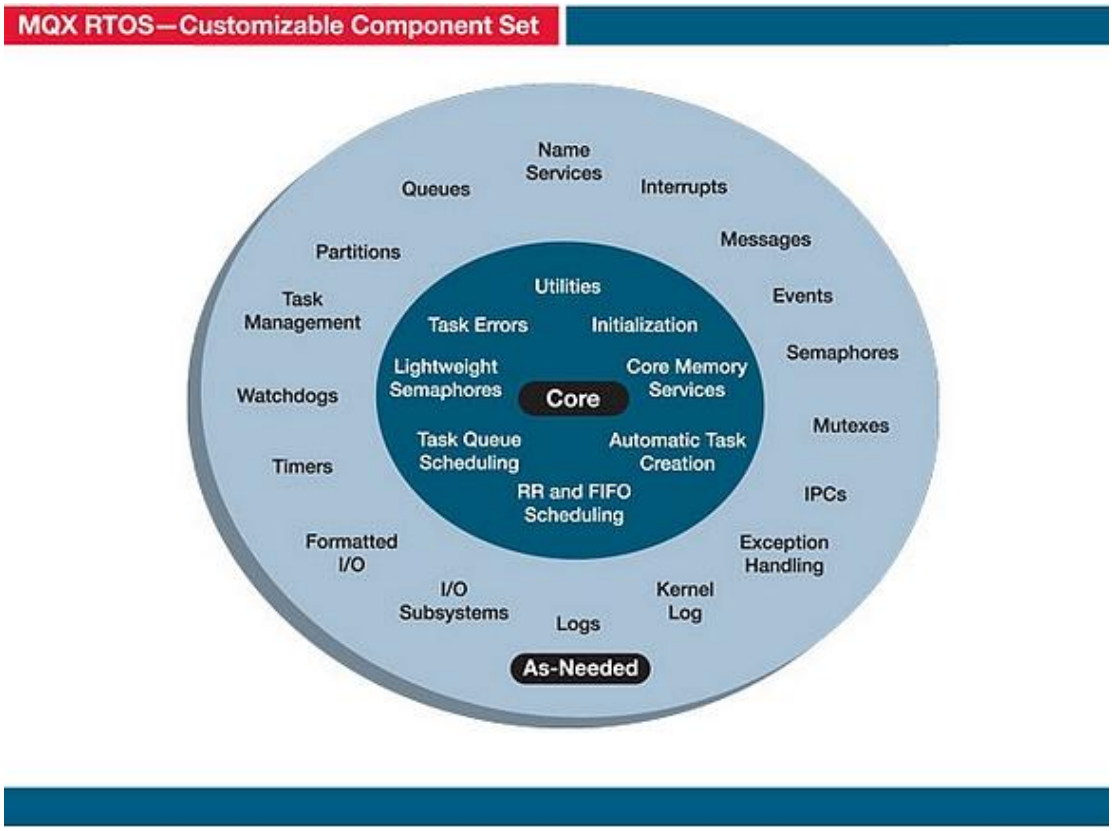

FIGURE 3. MQX RTOS – Customizable Component Set

4.1.3 LTD Hardware Abstraction Layer

Hardware Abstraction Layer (HAL) abstracts physical hardware solution. The I/O drivers of hardware are located in this layer. I/O drivers of MQX are slightly deviated subset of the POSIX standard I/O. LTD product platform has own configurable BSP. Guidelines for BSP porting are presented in the document Board Support Package Porting Guide. (5).

LTD HAL offers I/O drivers for the following subsystems:

- CPU based on Cortex-M4 architecture
- USB 2.0 interface (LS/FS)
    - OTG support
    - Mass Storage device class is supported
- Power management
- User Interface
    - LCD I/F via GPIO
    - Three LEDs & buttons
- Optional Bluetooth, BT 4.0

4.1.4 LTD BSP Configuration

The LTD platform has its own BSP configuration including e.g. clock, processor, memory, fixed I/O lines and boot mode initializations as well as an overall control. The LDT BSP creation is based on the Board Support Package Porting Guide document. (5).

After the design and evaluation stage of developing a product with MQX has been completed, the user often wants to be migrated the application project away from a Freescale evaluation board and to a custom board. Creating a new BSP for that end board is the most advisable move. Subsection 4.4.4 contains a more detailed description the creation of LTD BSP.

4.2 LTD Software Architecture Description

Figure 4 is the core of this thesis. It describes the structure of the LTD platform software. The design of the LTD platform software follows Freescale's implementation and their paradigm using the readymade components. Three software layers are separated clearly as figure 4 presents. The architecture helps to design a modular application. Sharing layers helps to make a more maintainable system.

The LTD software architecture has been designed to be modular, reusable and maintainable as Figure 4 presents. It gives the possibility to fit different kinds of readymade and customized software components to the existing system. The purpose for the LTD API service software layer is to handle overall system resources. For example, the measurement algorithms and device driver logic are built onto these level modules. The API services are the interface to build a comprehensive application using RTOS resources. The MQX RTOS capacity is utilized for the applications which have the tight real time requirements. The software implementation has utilized the operation modes of SoC to achieve a better efficiency. The code size and code running speed optimization has been done having an advantage of the MQX configurability. The green colored blocks are the places where most of the customer specific customization is done. The FFS block means the Freescale MQX File System (MFS). It is an embedded File Allocation Table (FAT) file system compatible with The Microsoft Windows and MS-DOS file systems.

BSP - Hardware Abstraction Layer (HAL) abstracts physical hardware solution. The I/O drivers of the hardware are located on this layer. The I/O drivers of MQX are a slightly deviated subset of the POSIX standard I/O. It is recommended that a new peripheral device is taken into use via the MQX I/O control. All blocks in figure 4 are described deeply in the following sections.

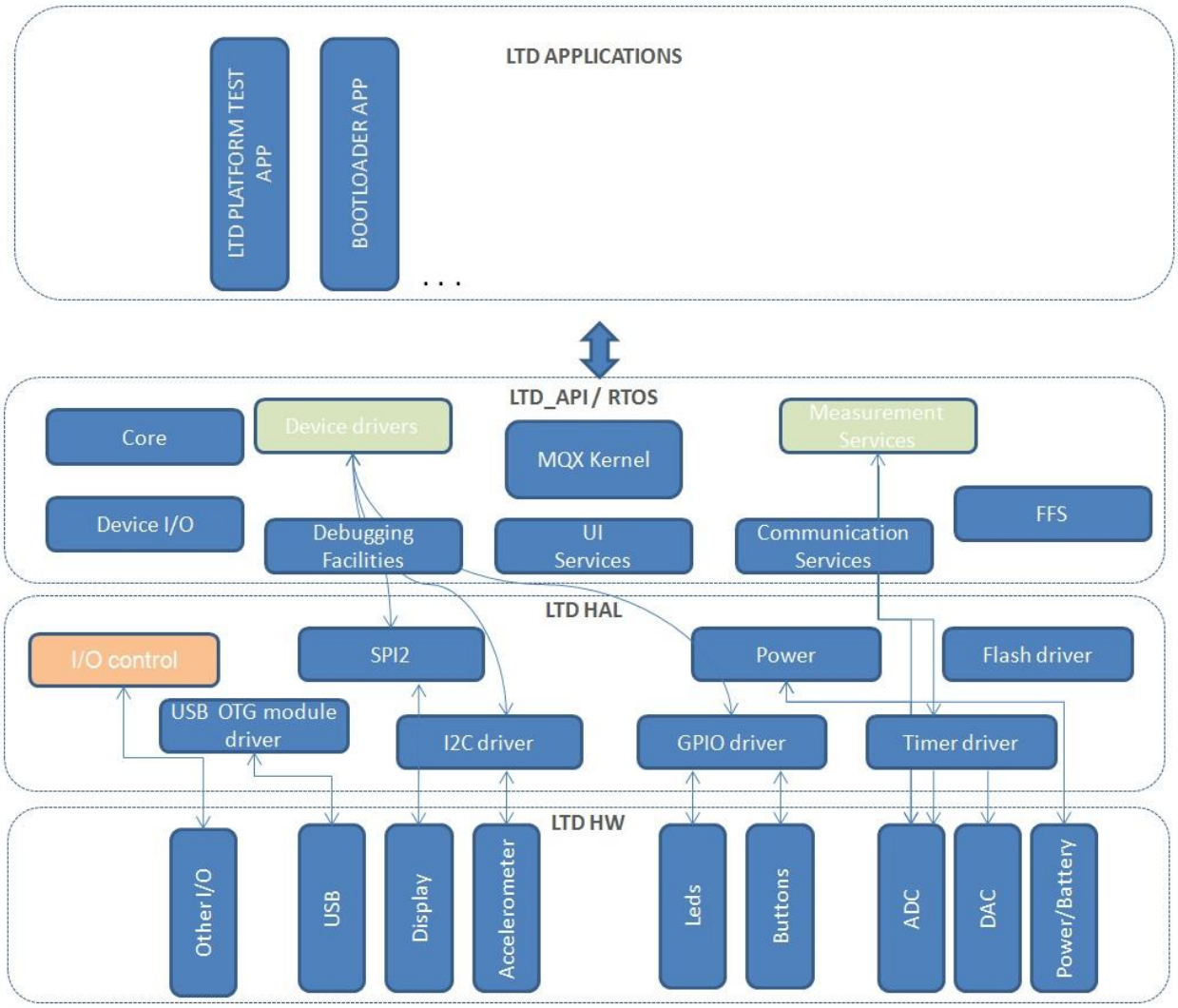

FIGURE 4. LTD SW Architecture

4.2.1 Interface Description

The software Interfaces at the function level are presented in the following sub-sections. There are two significant interfaces:

- HW →HAL. The hardware is abstracted with a HAL layer
- LTD API → APPLICATION. LTD API services applications.

LTD API functions

LTD API functions are listed in Appendix 1. The Listed functions are prototypes of the functions. Inputs and outputs are documented deeply in the source code. The code module headers are as follows:

```
//-----------------------------------------------------------------------------
//
//  RDMMA845 Accelerometer
//  ==================================
//
//  Copyright (C) 2012 Lewel Group Finland
//  All Rights Reserved.
//
//  File:               RDMMA845_accm.c
//  Product:         LTD Platform
//  Purpose:        Interface for RDMMA845 accel
//
//  Last committed:  $Revision: 123 $
//  Last changed:    $Date: 2012-05-08 08:08:37 +0300 (ti, 08 touko 2012) $
//  Last changed by: $Author: teka $
//
//-----------------------------------------------------------------------------
```

LTD Driver level functions

The LTD driver level interface functions can be seen in Appendix 1. The inputs and outputs are documented deeply in the source code. The function headers are as follows:

```
//-----------------------------------------------------------------------------
//  uint_32 I2C_RDMMA845_accm_read(  int i2c_device_address,
//                                   int sensor,
//                                   int length);
//  function read amount of bytes from pointed address
//  input: int i2c_device_address, int sensor, int length
//  output: return status 0 = OK
uint_32 I2C_RDMMA845_accm_read(int i2c_device_address, int sensor, int
length);
```

4.3 Software Component Description

4.3.1 Application Level

Freescale‘s evaluation environment offers a many examples, applications and demo codes for the Kinetis based chips. That is one way to boost and shorten a software design cycle. The applications for the LTD board follow the software structure presented in figure 5.

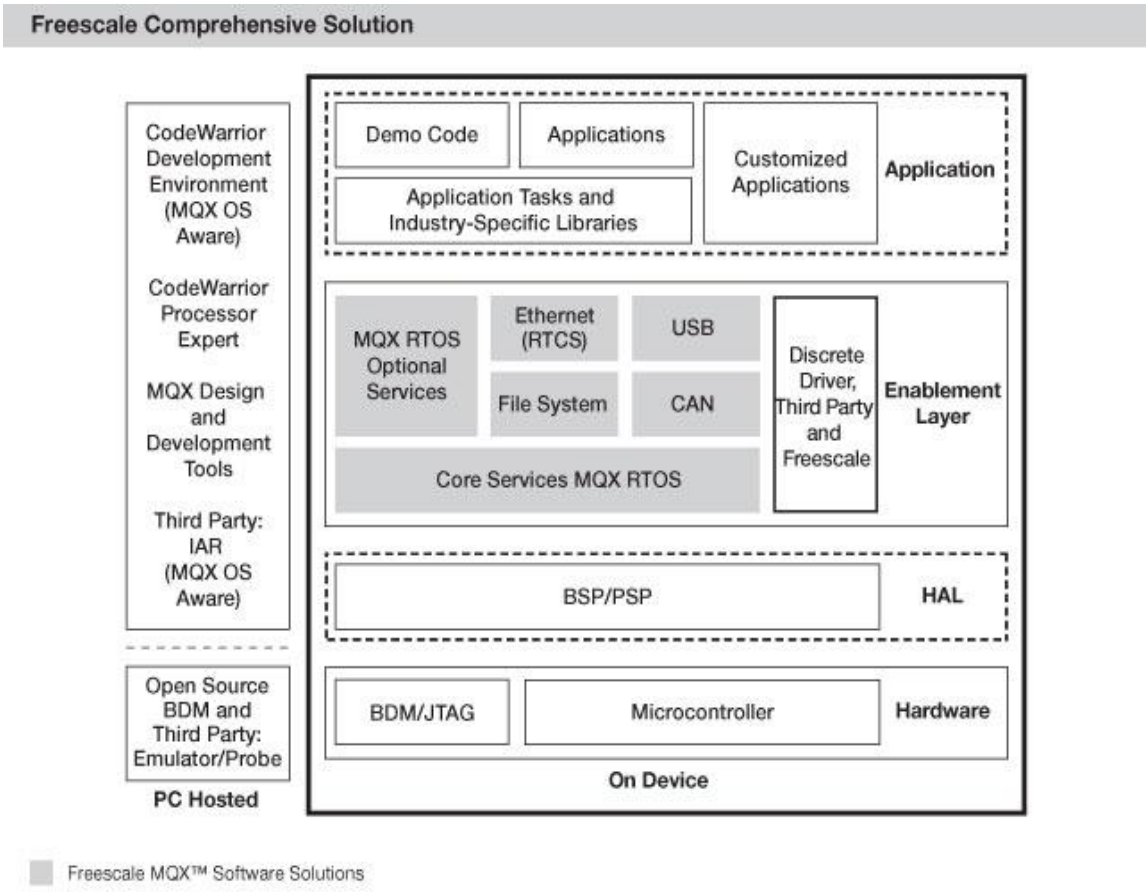

FIGURE 5. Freescale MQX based solution

4.3.2 Low Tier Device API Layer

The LTD API implements the called service level functionality as figure 6 presents. It abstracts the HW solutions and choices. That gives a possibility to use a different HW configuration and an independent application in the future. The requirements of e.g. the measurement services could be customer specific requirements. Most of the customer specific customization is done in the light green colored blocks shown in figure 6.

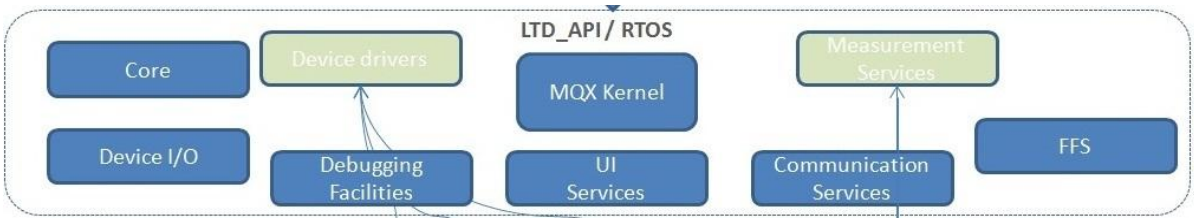

FIGURE 6. LTD API Level

General API Services

The general API service layer offers the functionality for application. In addition to the MQX RTOS and Kinetis core features, this layer could offer higher level algorithm services by the device drivers.

General API services offer the following features:

- Kinetis core features
- Application and HW initialization
- CPU based on Cortex-M4 architecture
- MQX RTOS features
    - Tasks, events, scheduler, semaphores, message handling.
    - See MQX reference manual
    - RTOS customization. MQX is optimized concerning the power consumption and memory wise. Small ram configuration is used.
- Core features

- Application and HW initialization
- Device I/O
    - General I/O initialization.
    - Clock, processor, memory, fixed I/O lines and boot mode initializations and overall control.
    - LTD board related initializations to be done in BSP level. See section 4.3.3.

Most of the MQX related configurations are done in the following files:

..\ Freescale MQX 3.7\mqx\source\bsp\ LTD_BOARD_twrk60n512\user_config.h

..\Freescale MQX 3.7\mqx\source\include\mqx_cnfg.h

..\Freescale MQX 3.7\lib\LTD_BOARD_twrk60n512.iar\small_ram_config.h, Appendix 2.

The LTD BSP configuration is defined in the following header file:

..\Freescale MQX 3.7\mqx\source\bsp\LTD_BOARD_twrk60n512\

LTD_BOARD_twrk60n512.h

Device Driver Services

The device driver services are provided for the application specific functionality. Some specific algorithm is implemented at that level. That kind of services can be as follows:

- Accelerometer functionality via I2C bus
    - open, read, write implementation for 3D accelerometer
- Display functionality is performed via SPI bus
    - Individual task for graphical display is used
    - I/F for application is running: *void show_frame(char *pixel_data);*

Power Management Services

The power management (PM) interface provides services for creating applications that control the power consumption. MQX supports the Kinetis specific lower power modes. The following list contains the main features of the PM services.

- Power mode handling
    - Transition handling between different operation modes
    - RUN, Very Low Power Stop (VLPS) and Very Low Power Run (VLPR) modes are supported
- Boot mode initializations
- Voltage control
    - Battery level monitoring via ADC

User Interface Services

The UI services for creating the application offer the following features:

- Human Machine Interface (HMI) driver handling to be handled via a GPIO driver. Buttons, LEDs etc.
- Display functionality via SPI driver with some dedicated pins
    - Mono color Graphical display support
- eGUI API interface. eGui is Freescale's own free graphical component. The complimentary Freescale embedded graphical user interface (eGUI) allows single chip microcontroller (MCU) systems to implement a graphical user interface and drive the latest generation of the color graphics LCD panels with the integrated display RAM and the simple serial peripheral interface (SPI) or a parallel bus interface.

Communication services

The communication interface offers services to communicate between the devices via USB 2.0 or BLE. JTAG and UART are meant for the debugging purpose. The following list presents the main features of the communication services:

- Serial interface communication via serial driver
- MQX 3.8 has USB HOST and DEVICE stack supporting LS and FS speeds.

    - LTD flash is shown as hard disk in USB communication
    - HS support availability is expected e/o 2012 in MQX.
    - OTG stack support is expected in MQX at 7/2012.
- Protocol definition between host and LTD
- BLE data transfer interface
    - PXP BLE profile is supported. The Profile is done following the specification PXP_SPEC_V10. (6).

FFS services

The Freescale MQX File System (MFS) is an embedded FAT file system compatible with the Microsoft Windows and MS-DOS file systems. It can format, read, write and exchange files with any operating systems running a FAT-12, FAT-16 or FAT-32 files system. It is fully re-entrant and uses the Freescale MQX RTOS file device driver to access disk devices. Also, included with the Freescale MQX RTOS is a Trivial File System (TFS) that can be used in HTTP applications requiring a simple read-only file system.

MFS provides a library of functions. The functions let an embedded application access the file system in a manner that is compatible with the MS-DOS Interrupt 21 functions. All the functions guarantee the application tasks a mutually exclusive access to the file system. MFS is a device driver that an application must install over a lower-level device driver.

Measurement services

The Measurement services module is an imaginary specification for example to serve some specific measurement application. The measurement can be handled by an external I/O measurement board.

The measurement related functions could be as the following functions:

- FlexTimer controlling to give a pulse length definition via DAC
- DAC controlling for example the current or voltage output value creation

- ADC sampling would be used to get the measurement results as feedback.

Debugging Facilities

Debugging is done over the JTAG-connection. The connector includes signals for Trace capturing and programming. JTAG debugging options should be defined in the project options depending on the software work bench.

4.3.3 LTD Hardware Abstraction Layer

The hardware block diagram is described in the LT Engine HW Block Diagram. (7). The LTD HAL development follows based on that HW concept. See the MQX IO Drivers User Guide document for using drivers. (8). See the GPIO mapping in the document LT Engine Hardware Specification. (9, p.28-44). The LTD HAL level implements the HW adaptations as figure 7 presents.

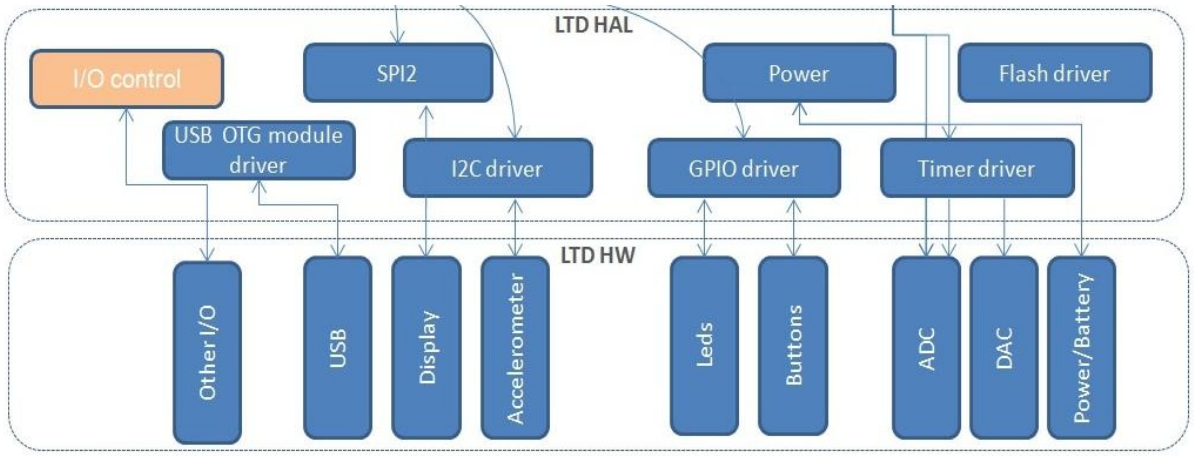

FIGURE 7. LTD Hardware Abstraction Layer (HAL)

Clock Generation

Figure 8 presents the clock generation of the Kinetis K20/K60 based chips. The system starts up by using an internal 32 kHz oscillator. It is used until the external clocks are started. As a default the software configures the clock to use external crystal and enables internal Phase Lock Loop (PLL) in the Frequency Lock Loop (FFL) Bybassed External

(FBE) mode. It is important to notice that as the default Freescale K60 software uses external clock instead crystal.

The clock frequencies in the LTD are:

- System oscillator is 8MHz
- Real Time Clock (RTC) oscillator is 32kHz

The multipurpose Clock Generation (MCG) Control 1 Register (MCG_C1) has settings for the system clock in a normal operation. The recommended operating mode is High-frequency mode 1, low power. The selection is configured in the MCG Control 2 Register (MCG_C2). The clock rate settings are configured for MQX in the bsp\"board".h header file and other clock configurations are in the init_hw.c file. The following configuration must be defined in the LTD_BOARD_twrk60n512.h header file.

```
/* The clock configuration */
#define BSP_CLOCK_SRC      (8000000ul)   // oscillator freq
#define BSP_REF_CLOCK_SRC  (2000000ul)   // must be 2-4MHz

#define BSP_CORE_DIV       (1)
#define BSP_BUS_DIV        (1)
#define BSP_FLEXBUS_DIV    (1)
#define BSP_FLASH_DIV      (2)
#define BSP_USB_DIV        (1)
#define BSP_USB_FRAC       (0)

/*  BSP_CLOCK_MUL from interval 24 - 55 BSP_CORE_CLOCK = 48MHz */
#define BSP_CLOCK_MUL      (24)

#define BSP_REF_CLOCK_DIV  (BSP_CLOCK_SRC / BSP_REF_CLOCK_SRC)
#define BSP_CLOCK          (BSP_REF_CLOCK_SRC * BSP_CLOCK_MUL)

/* CORE CLK, max 100MHz*/
#define BSP_CORE_CLOCK     (BSP_CLOCK / BSP_CORE_DIV)

/* SYSTEM CLK, max 100MHz */
#define BSP_SYSTEM_CLOCK   (BSP_CORE_CLOCK)

/* max 50MHz */
#define BSP_BUS_CLOCK      (BSP_CLOCK / BSP_BUS_DIV)
#define BSP_FLEXBUS_CLOCK  (BSP_CLOCK / BSP_FLEXBUS_DIV)

/* max 25MHz */
   #define BSP_FLASH_CLOCK    (BSP_CLOCK / BSP_FLASH_DIV)
```

	OSC	MCG	SIM
Muliplexers	MCG_C*x*	MCG_C*x*	SIM_SOPT1, SIM_SOPT2
Dividers	—	MCG_C*x*	SIM_CLKDIV*x*
Clock gates	OSC_CR	MCG_C1	SIM_SCGC*x*

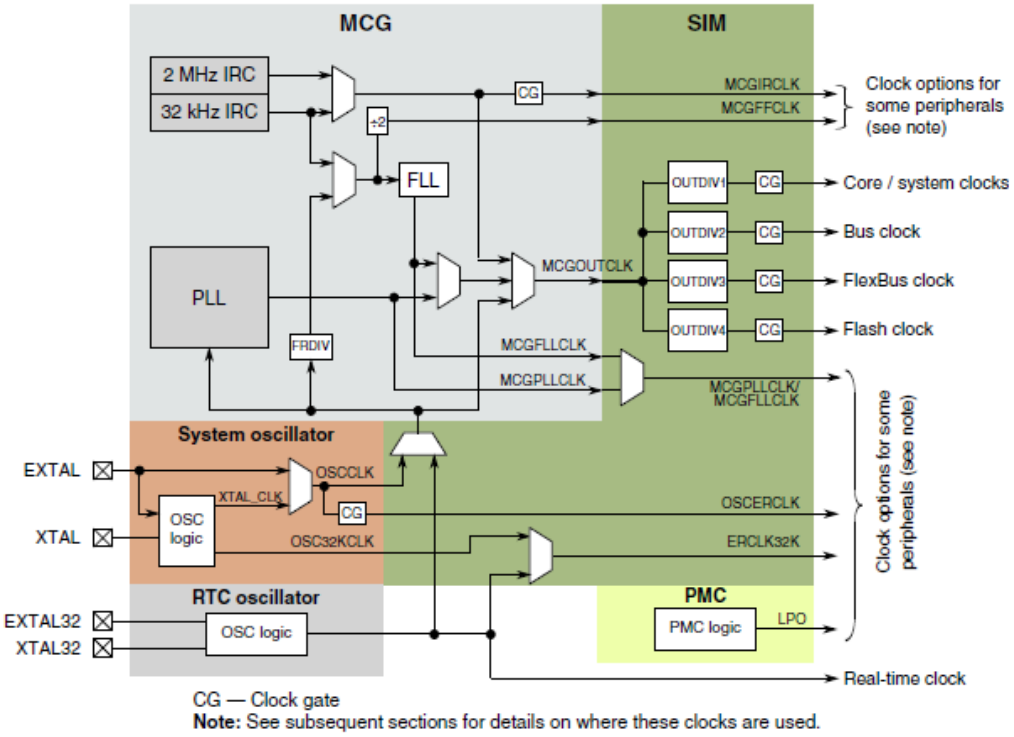

FIGURE 8. Clock Generation of Kinetis

EEPROM

To provide an enhanced EEPROM functionality, the FlexMemory uses a RAM block (FlexRAM), a flash block (FlexNVM), and an EEE state machine. When the EEE functionality is enabled, the FlexRAM becomes the user's EEE memory. The FlexRAM address space is where one accesses all of the EEE data. The detailed description of the Kinestis EEPROM is shown in EEPROM configuration instructions document. (10.)

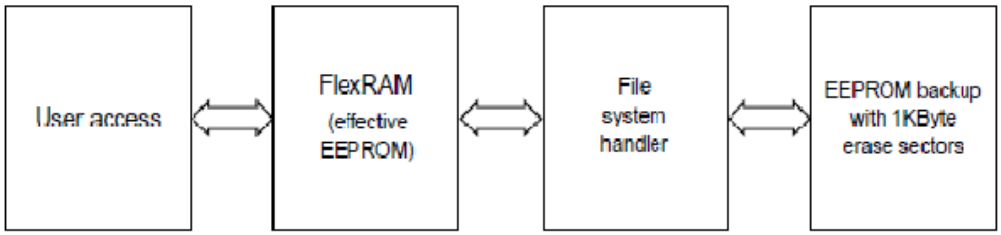

FIGURE 9. Eeprom solution in Kinetis (10).

MQX I/O

The MQX I/O subsystem implementation is a slightly deviated subset of the POSIX standard I/O. It follows the UNIX model of open, close, read, write, and ioctl functions. The I/O subsystem makes calls to I/O device-driver functions. The MQX I/O uses the pointers to the FILE, as returned by fopen(), instead of the file descriptors (FDs). The MQX I/O has three layers as presented in figure 10. (8.)

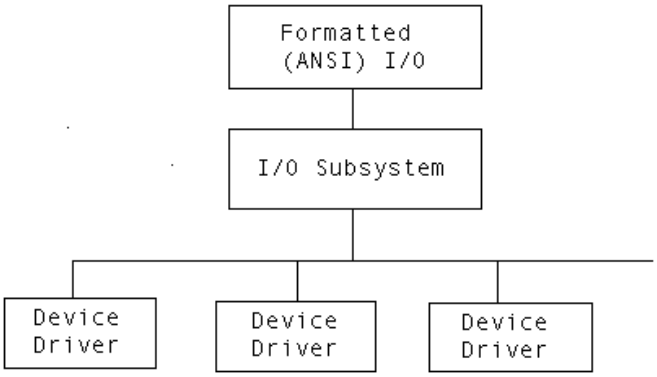

FIGURE 10. MQX I/O layers (8).

Device I/O Control

Figure 11 shows the relationship between a file handle (FILE_STRUCT) that is returned by fopen(), the I/O device structure (allocated when the device is installed), and the I/O driver functions for all I/O device drivers. (8.)

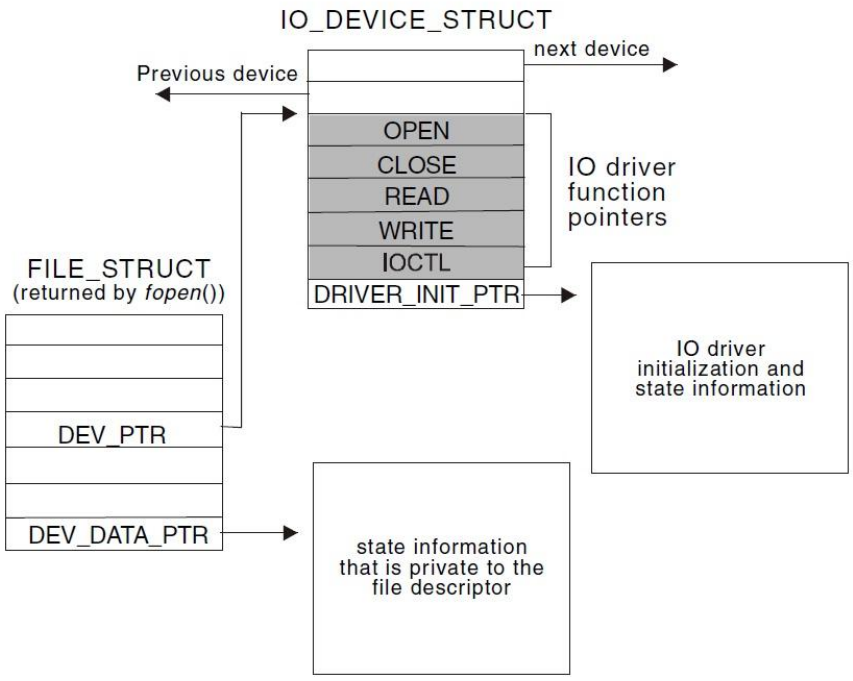

FIGURE 11. I/O Device Structure — I/O Device Drivers (8).

Serial I/O Control

The serial device drivers are complex in that they have a generic driver layer and a low-level standard simple interface to the serial hardware. Figure 12 shows the relationship between a file handle (FILE_STRUCT) that is returned by fopen(), the I/O device structure (allocated when the device is installed), and the upper-level serial-device driver functions. (8.)

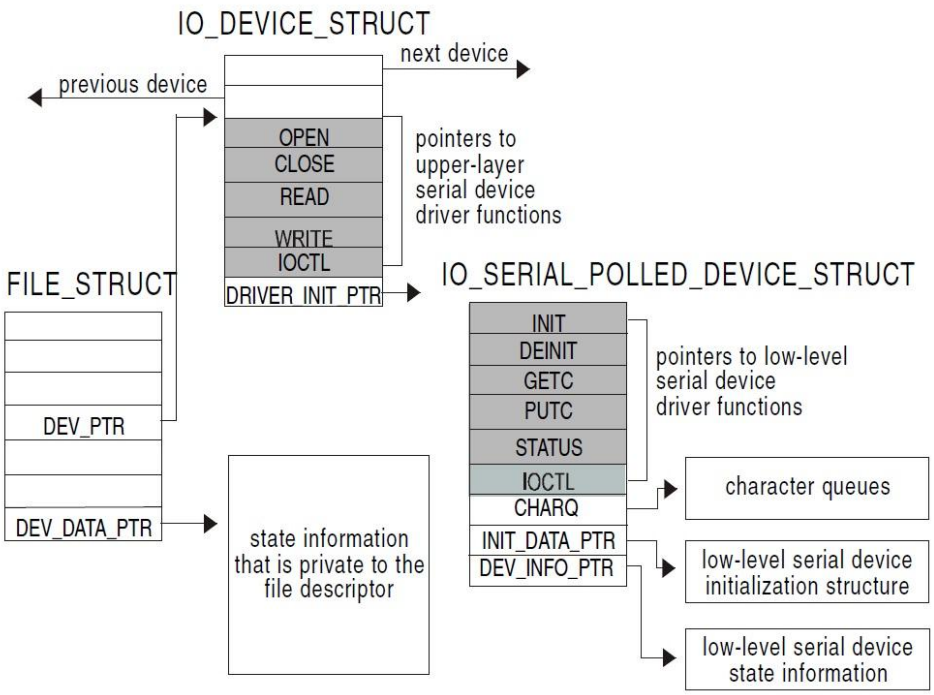

FIGURE 12. I/O Device Structure — Serial-Device Drivers (8).

USB On The Go Driver

The LTD engine supports the USB 2.0 interface in the LS and FS modes. The supported modes are Device, OTG. The design can also support USB Host protocol, but the power supply to VBUS is limited to the level that is required in the OTG-mode when the device is acting as host, i.e. 5V / 8mA rather than 5V / 500mA as required for the standard USB host.

In the Device mode the VBUS voltage is regulated down to the 3.3V inside processor. The data lines of the USB are terminated to that voltage with internal termination resistors.

The USB OTG implementation requires an external component MAX3353E, which supplies the power to VBUS, includes the ESD protection and a possibility to terminate the data lines. MAX3353E is controlled by I2C (I2C0).

TABLE 1. Pin description of USB cable (9).

Pin	Signal
1	VBUS
2	D-
3	D+
4	ID
5	GND

I2C Driver

The chip has three I2C channels. The desired I2C I/O control is taken into use by activating that in the user_config.h header file. The I2C specific commands and the definition are presented in the i2c.h file. The I2C I/O driver's use example is shown below.

```
/* first, initialize the file descriptor */
MQX_FILE_PTR  i2C1_fd = NULL;

/* open I2C1 channel */
i2c1_fd = fopen("i2c1:", NULL);
if (NULL == i2c1_fd)
 {
  printf ("Error opening I2C driver!\n");
  _time_delay (200L);
 _task_block ();
}

/* IOCTL calls specific to I2C */
i2c.h
```

SPI Driver

The chip has three SPI channels, SPI0, SPI1 and SPI2. The desired SPI I/O control is taken into use by activating it in the user_config.h header file. The SPI specific commands and the definition are presented in spi.h. The SPI Driver's use example is shown below.

```
/* first, initialize the file descriptor */
MQX_FILE_PTR  spifd = NULL;

/* open SPI2 channel */
spifd = fopen("spi2:", NULL);
if (NULL == spifd)
{
  printf ("Error opening SPI driver!\n");
  _time_delay (200L);
 _task_block ();
}

/* Set device mode  */
param = SPI_DEVICE_MASTER_MODE;
if (SPI_OK == ioctl (spifd, IO_IOCTL_SPI_SET_TRANSFER_MODE, &param))
{
    printf ("device mode SET OK\n");
}

/* write operation */
if (SPI_OK == ioctl (spifd, IO_IOCTL_SPI_READ_WRITE, &rw_struct))
{
   /* printf ("line data sent OK\n"); */
}

   /* IOCTL calls specific to SPI */
   see spi.h
```

GPIO Driver

The I/O Pins of the CPU can be multiplexed to different functional modules. Each individual I/O pin has a control register to select the multiplexing and to setup the hardware configuration of the pin. The processor has five 32-pin ports as figure 13 shows. Each 32-pin port is assigned one interrupt. The digital filter option has two clock source options: bus clock and 1-kHz LPO. The 1-kHz LPO option gives the user this feature in low power modes. The digital filter is configurable from 1 to 31 clock cycles when enabled. See GPIO mapping of the LTD in the document LT Engine Hardware_Specification. (9, p.28-44).

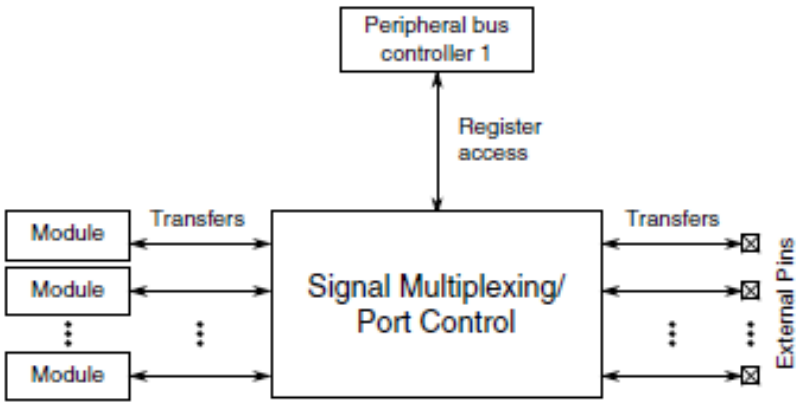

FIGURE 13. Signal Multiplexing and Port Control (12).

GPIO driver in LTD Board Support Package implementation

Three on/off buttons are connected to the GPIO pins. The buttons can be used to simulate a simple UI or used for different control purposes depending on the application's need. One LED is reserved for each control button. If UI is not implemented, the LEDs can be used for example to indicate different operating modes. The TWR MQX GPIO driver implementation is ported to LTD_BOARD BSP. The use of GPIO goes in the same way as the use of the other drivers. See the following GPIO driver use example.

```
/* define gpio pin */
#define LCD_EXTCOMIN (GPIO_PORT_D | GPIO_PIN5)

/* define pin list and default values of them (pin listing is not mandatory if
just one signal is in question) */
uint_32 extcomin_pin[] =
{
  LCD_EXTCOMIN | GPIO_PIN_STATUS_1,
  GPIO_LIST_END
};

/* ...in the function
  first, initialize the file descriptor */
MQX_FILE_PTR  extcomin_fd = NULL;

/* Open gpio port as output ("gpio:write") */
extcomin_fd = fopen ("gpio:write", (char_ptr) &extcomin_pin);
if (NULL == extcomin_fd)
{
   printf ("Error opening extcomin GPIO!\n");
    _time_delay (200L);
    _task_block ();
}
/* GPIO IOCTL commands from io_gpio.h */
```

Analog-to-Digital Converter Driver

The Desired ADC I/O control is taken into use by activating that in the user_config.h header file. The chip has two ADC channels, ADC0 and ADC1. The application specific ADC usability can be specified separately. The use of the ADC channels goes in the same way as the serial bus controlling with the open, read and write functions.

The ADC HW implementation is shown in figures 14 and 15. Figure 14 shows ADC channels with the Pin Grid Array (PGA) integration, and the Interleaved ADC channels are presented in figure 15. The detailed information of the ADC channels is described in the document LT Engine Hardware Specification. (9.)

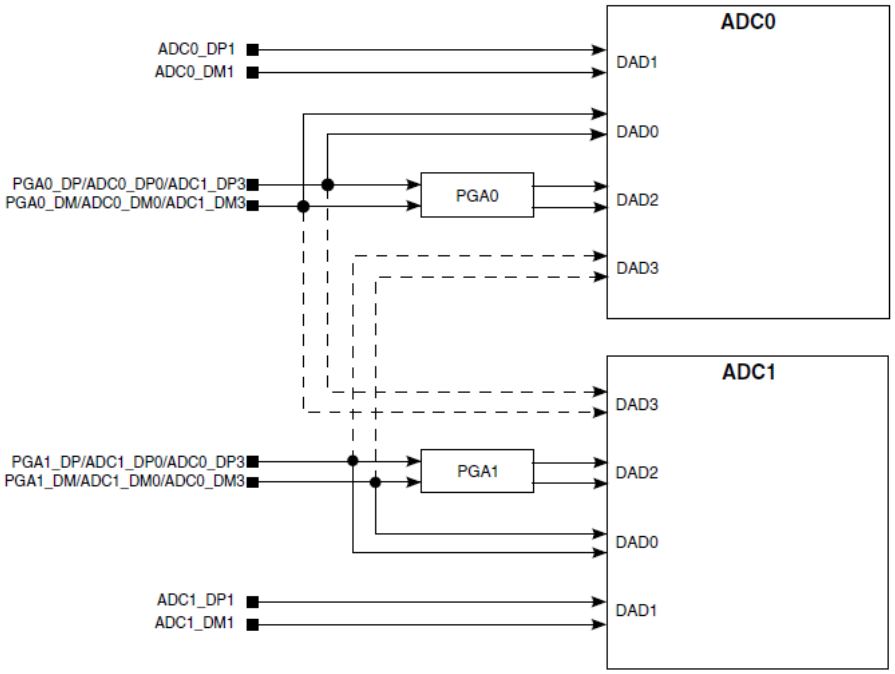

FIGURE 14. ADC Channels with PGA Integration (12).

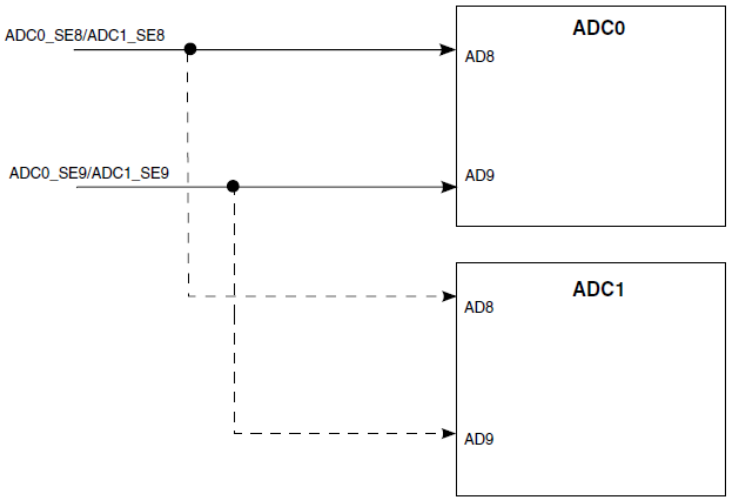

FIGURE 15. Interleaved ADC channels (12).

Digital-to-Analog Converter Driver

There are 2 DAC outputs. The MQX has not a DAC driver implemented as POSIX API. Therefore the DAC usability has to be built with direct register configurations. The application specific DAC usability can be specified separately.

Power Management (PM) Driver

The three primary modes of the operation are run, wait and stop. The WFI instruction invokes both wait and stop modes for the chip. The primary modes are augmented in a number of ways to provide lower power based on the application's needs. The chip level operating modes of the processor are listed in table 2.

TABLE 2. Operation modes of Kinetis (9).

Chip mode	Description	Core mode	Normal recovery method
Normal run	Allows maximum performance of chip. Default mode out of reset; on-chip voltage regulator is on.	Run	-
Normal Wait - via WFI	Allows peripherals to function while the core is in sleep mode, reducing power. NVIC remains sensitive to interrupts; peripherals continue to be clocked.	Sleep	Interrupt
Normal Stop - via WFI	Places chip in static state. Lowest power mode that retains all registers while maintaining LVD protection. NVIC is disabled; AWIC is used to wake up from interrupt; peripheral clocks are stopped.	Sleep Deep	Interrupt
VLPR (Very Low Power Run)	On-chip voltage regulator is in a low power mode that supplies only enough power to run the chip at a reduced frequency. Reduced frequency Flash access mode (1 MHz); LVD off; internal oscillator provides a low power 2 MHz source for the core, the bus and the peripheral clocks.	Run	Interrupt
VLPW (Very Low Power Wait) -via WFI	Same as VLPR but with the core in sleep mode to further reduce power; NVIC remains sensitive to interrupts (FCLK = ON). On-chip voltage regulator is in a low power mode that supplies only enough power to run the chip at a reduced frequency.	Sleep	Interrupt
VLPS (Very Low Power Stop)-via WFI	Places chip in static state with LVD operation off. Lowest power mode with ADC and pin interrupts functional. Peripheral clocks are stopped, but LPTimer, RTC, CMP, TSI, DAC can be used. NVIC is disabled (FCLK = OFF); AWIC is used to wake up from interrupt. On-chip voltage regulator is in a low power mode that supplies only enough power to run the chip at a reduced frequency. All SRAM is operating (content retained and I/O states held).	Sleep Deep	Interrupt
LLS (Low Leakage Stop)	State retention power mode. Most peripherals are in state retention mode (with clocks stopped), but LLWU, LPTimer, RTC, CMP, TSI, DAC can be used. NVIC is disabled; LLWU is used to wake up. **NOTE:** The LLWU interrupt must not be masked by the interrupt controller to avoid a scenario where the system does not fully exit stop mode on an LLS recovery. All SRAM is operating (content retained and I/O states held).	Sleep Deep	Wakeup Interrupt[1]
VLLS3 (Very Low Leakage Stop3)	Most peripherals are disabled (with clocks stopped), but LLWU, LPTimer, RTC, CMP, TSI, DAC can be used. NVIC is disabled; LLWU is used to wake up. SRAM_U and SRAM_L remain powered on (content retained and I/O states held).	Sleep Deep	Wakeup Reset[2]
VLLS2 (Very Low Leakage Stop2)	Most peripherals are disabled (with clocks stopped), but LLWU, LPTimer, RTC, CMP, TSI, DAC can be used. NVIC is disabled; LLWU is used to wake up. SRAM_L is powered off. A portion of SRAM_U remains powered on (content retained and I/O states held).	Sleep Deep	Wakeup Reset[2]

VLLS1 (Very Low Leakage Stop1)	Most peripherals are disabled (with clocks stopped), but LLWU, LPTimer, RTC, CMP, TSI, DAC can be used. NVIC is disabled; LLWU is used to wake up. All of SRAM_U and SRAM_L are powered off. The 32-byte system register file and the 32-byte VBAT register file remain powered for customer-critical data.	Sleep Deep	Wakeup Reset[2]
BAT (backup battery only)	The chip is powered down except for the VBAT supply. The RTC and the 32-byte VBAT register file for customer-critical data remain powered.	Off	Power-up Sequence

1. Resumes normal run mode operation by executing the LLWU interrupt service routine.
2. Follows the reset flow with the LLWU interrupt flag set for the NVIC.

Figure 16 shows the operation mode transitions. As figure 16 presents, the operation mode transition is possible between low power modes. There can be applications which need the low power modes all the time.

- For example, in the VLPS mode some blocks (ADC, DAC etc.) and memories are powered.
- The VLPS power consumption of the microcontroller is about 50uA as mentioned in the K20 Sub-Family Datasheet. (12).

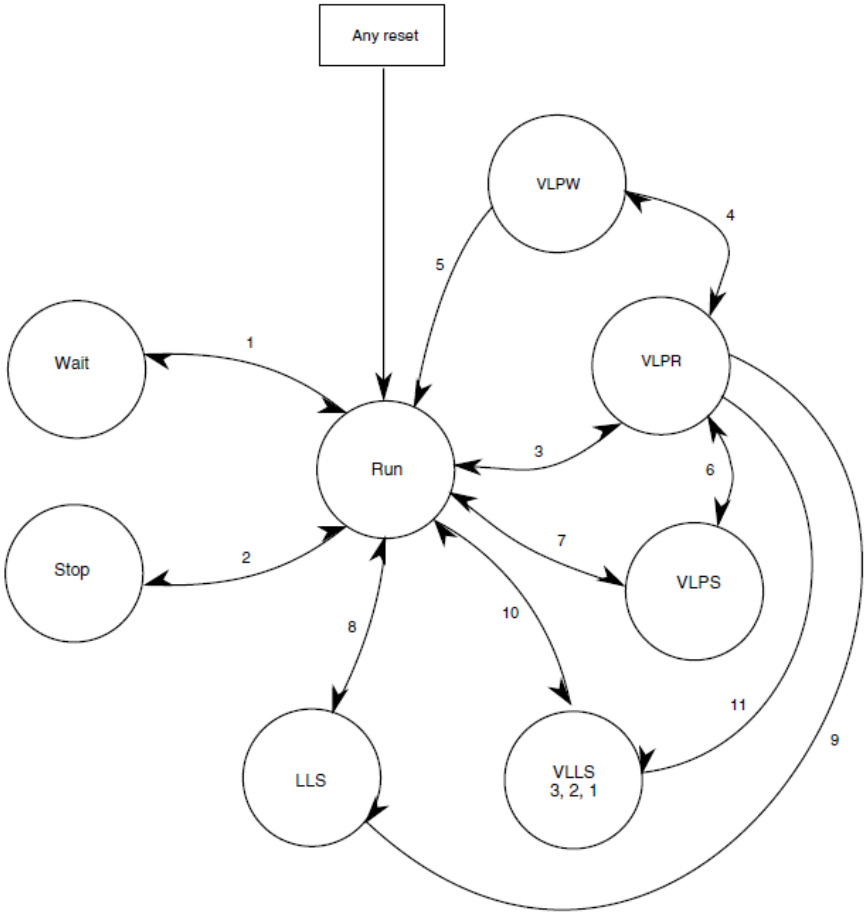

FIGURE 16. Operation mode transitions (12).

See the following example of the VLPS mode entry routine. It puts the processor into the VLPS mode directly from the normal run mode.

If the VLPS mode is entered directly from a normal RUN mode, then the LPWUI bit is forced to 1 by the hardware. This means that when an interrupt occurs the user will exit to normal run mode instead of VLPR. If, however, the VLPS mode is entered from VLPR, the state of the LPWUI bit determines the state the MCU returns to upon the exit from VLPS. If LPWUI is set and an interrupt occurs, one exits to the normal run mode instead of VLPR. If LPWUI is clear and an interrupt occurs, one exits to VLPR.

```
void enter_vlps(char lpwui_value)
{
    /* Write to PMPROT to allow VLPS power modes write one if not all set make
    sure all enabled this write-once bit allows the MCU to enter the very low
    power modes: VLPR, VLPW, and VLPS. */
  MC_PMPROT = MC_PMPROT_AVLP_MASK;

  /* Set the LPLLSM field to 0b010 for VLPS mode - Need to set state of LPWUI
bit 8 */
  if(lpwui_value)
  {
    MC_PMCTRL = (   MC_PMCTRL_LPWUI_MASK    |          // set LPWUI
                 MC_PMCTRL_LPLLSM(2));             // set LPLLSM = 0b10
  }
  else
  {
    MC_PMCTRL = (!MC_PMCTRL_LPWUI_MASK     |      // set LPWUI
                  MC_PMCTRL_LPLLSM(2));            // set LPLLSM = 0b10
  }

  sleep_ltd_disable_ports_partial();

  OSC_CR = 0;

  /* Now execute the stop instruction to go into VLPS */
  /* Set the SLEEPDEEP bit to enable deep sleep mode (STOP) */
  SCB_SCR |= SCB_SCR_SLEEPDEEP_MASK;

  /* WFI instruction will start entry into STOP mode */
  asm("WFI");
}
```

Timer Driver

See the following use example of the low power timer 0. The interrupt is generated when the compare value is reached.

```
void lptmr0_interrupt(int compare_value)
{
  // disable LPT INT
  _cortex_int_init(INT_LPTimer, LPT_INT_PRIORITY, FALSE);

  // Clear global variable
  LPTMR_INTERRUPT = 0;

  // Reset LPTMR module
  lptmr0_clear_registers();

  // install lptmr interrupt service routine
  _int_install_isr(INT_LPTimer, lptmr0_isr, NULL);

  // Enable LPTMR0 Interrupt in NVIC
  _cortex_int_init(INT_LPTimer, LPT_INT_PRIORITY, TRUE);

  // Set compare value
  LPTMR0_CMR = LPTMR_CMR_COMPARE(compare_value);

  // Use LPO clock and bypass prescale
  LPTMR0_PSR = LPTMR_PSR_PCS(0x1) | LPTMR_PSR_PBYP_MASK;

  // Enable LPT interrupt
```

```
    LPTMR0_CSR = LPTMR_CSR_TIE_MASK;

    // Turn on LPTMR and start counting
    LPTMR0_CSR |= LPTMR_CSR_TEN_MASK;

    // Wait for the global variable to be set in LPTMR ISR
    while(LPTMR_INTERRUPT == 0) {}

    DEBUG_PRINT1("Timer should have waited for %d msec\n", compare_value);

    // Turn off LPT to avoid more interrupts
    LPTMR0_CSR &= ~LPTMR_CSR_TEN_MASK;

    // Reset LPTMR module
    lptmr0_clear_registers();
}
```

Flash RW Driver

The Freescale MQX File System (MFS), which is used on the flash memory driver, is an embedded FAT file system compatible with Microsoft Windows and with the MS-DOS file systems. It can format, read, write and exchange files with any operating systems running a FAT-12, FAT-16 or FAT-32 files system. Examples of the lower-level drivers are drivers for memory devices, flash disks, floppy disks, or partition-manager devices. MFS uses the lower-level driver to access the hardware device.

The creation of the file system is possible when the chip packet has Flex NVM memory. See more information about the Flex memory and EEPROM configuring in document EEPROM configuration instructions. (10).

Display Driver

The Sharp 96 * 96 pixel 1.35 inch display is used. The Screen refresh is done by its own display task. `ShowFrame()` function is a interface for higher application layers. The Sharp LCD is controlled by the SPI controller. The SPI controller has to be configured in the way that the LTD is a master. The data is transferred Least Significant Bit (LSB) first. The chip select must stay high during the burst (one line or frame). The frame size has to be set accordingly or the end of the queue flag has to be used and a continuous chip select has to be enabled. Display must be refreshed within a certain interval. The refreshing is done using the EXTCOMIN pin of the display interface. The EXTCOMIN signal is connected to PTE26. The output can be either software controlled or a 1Hz clock output from RTC. The

DISP signal is used to switch on the display ON or OFF. It is connected to PTD15 and it is Software controlled output. The display signal interface is shown in table 3.

TABLE 3. Signals of Display interface (9).

TERMINAL	SYMBOL	I/O	FUNCTION	NOTES
1	SCLK	INPUT	Serial clock signal	
2	SI	INPUT	Serial data input signal	
3	SCS	INPUT	Chip select signal	
4	EXTCOMIN	INPUT	External COM inversion signal input (H: enable)	1
5	DISP	INPUT	Display ON/OFF signal	2
6	VDDA	POWER	Power supply (Analog)	
7	VDD	POWER	Power supply (Digital)	
8	EXTMODE	INPUT	COM inversion select terminal	3
9	VSS	POWER	GND (Digital)	
10	VSSA	POWER	GND (Analog)	

NOTES:
1. EXTCOMIN is HIGH enabled. When LOW, the serial input flag is enabled. See Figure 14 and Figure 15 for recommended circuits.
2. DISP enables/disables the display. All pixels will revert to Normal mode (reflective) when LOW. When DISP = H, data in the pixel memories displays normally.
3. EXTMODE pin must be connected to VDD for HIGH, and to VSS for LOW. See Figure 17 in Interfacing and Signals.

Bluetooth Low Energy Driver

The design is based on the Nordic Semiconductor nRF8001 controller. The operating principle is shown in figure 17. The following is a description of the nRF8001 operational modes:

Sleep mode:

- Power saving mode; all functionality is stopped
- Stored configuration settings are retained in memory
- Dynamic data (like bonding information) is stored in memory

Setup mode:

- nRF8001 configuration and setup:
- Generic Access Profile (GAP) configuration

- Generic Attribute Profile (GATT) service and GATT client configuration
- Hardware configuration
- Default operating mode entered upon reset unless setup is stored in NVM
- All dynamic data is cleared

Active mode:

- Mode used for runtime operation
- Active mode controls three levels of activity:
- Connected; nRF8001 is connected to a peer device, data transfer
- Advertising; nRF8001 is advertising/trying to connect
- Standby; No radio activity, Idle state
- Completing the setup sequence puts nRF8001 in active mode
- Establish a connection with a Bluetooth low energy central device
- Establish a bonded relationship with a Bluetooth low energy central device
- Send and receive data using service pipes
- Save or restore dynamic data such as bonding and pipe status

Test mode:

- Two test features are available: RF PHY and ACI physical connection
- Direct RF PHY Direct Test Mode (DTM)3 for qualification, test and evaluation of RF
- PHY layer performance
- ACI physical connectivity test
- All dynamic data is cleared

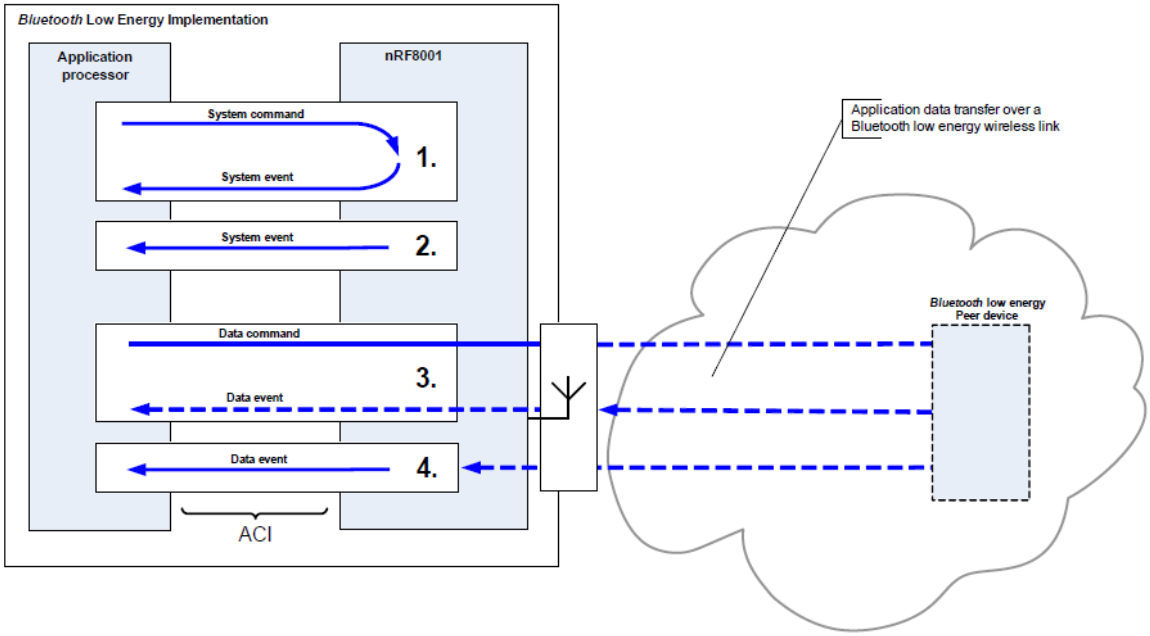

FIGURE 17. BLE operating principle (9).

4.3.4 LTD BSP Creation

The custom BSP creation follows directly the instructions of the Board Support Package Porting Guide. (5.) It is recommended that every different chip package has its own Board Support Package. (11, p.149-154.)

LTD board specific modifications

MK60N512VMD100.h is used in the LTD K20 board definitions. K20 and K60 have a similar memory mapping. The LTD_K20_BOARD pre-processor option is used in the case of K20 chip based the LTD board. The LTD board specific definition is done to the user_config.h, LTD_BOARD_twrk60n512.h, init_bsp.c files. The clock rate settings are configured to the MQX in bsp\"board".h header and the other clock configurations are implemented in the init_hw.c file. See below the LTD board specific modifications for the reference software design.

```
// Display.c
// LTD LCD pin mapping
#define LCD_EXTCOMIN (GPIO_PORT_E | GPIO_PIN26)
#define LCD_EXTCOM_PORT PORTE_BASE_PTR
#define LCD_EXTC_PINNUM 26   /* defined to set ALT to pin control reg */

// LTD specific power enables
#define LCD_DISP     (GPIO_PORT_D | GPIO_PIN15)
#define LCD_2V8      (GPIO_PORT_C | GPIO_PIN12)
#define LCD_5V0      (GPIO_PORT_C | GPIO_PIN13)

// LTD_BOARD_twrk60n512.h
// LTD BOARD Leds
#define LTD_BSP_LED1  (GPIO_PORT_A | GPIO_PIN27)
#define LTD_BSP_LED2  (GPIO_PORT_A | GPIO_PIN28)
#define LTD_BSP_LED3  (GPIO_PORT_A | GPIO_PIN29)

// LTD BOARD buttons
#define LTD_BSP_BUTTON1   (GPIO_PORT_A | GPIO_PIN24)
#define LTD_BSP_BUTTON2   (GPIO_PORT_A | GPIO_PIN25)
#define LTD_BSP_BUTTON3   (GPIO_PORT_A | GPIO_PIN26)
```

5 CONCLUSION

This chapter describes the results of the work. It discusses which could be good further development possibilities.

5.1 Results

The Existing reference designs from Freescale gave a good baseline to create the LTD software platform. This architecture is based on the TRW system design of Freescale. Following the instructions and guidelines made by Freescale helped to create a comprehensive platform solution. It is recommended that every LTD based variant gets its own BSP configuration. (11, p.149-154).

5.1.1 CPU Subsystem

CPU works as planned with 48MHz clock. Two operation modes are supported. They are the RUN and VLPS (Very Low Power Stop) modes. The actual power consumption of different hardware components is quite difficult to verify separately referring to the hardware design of the LTD. The optimization of the hardware design can help to reduce the power consumption.

5.1.2 Low Tier Device Board Support Package

Naturally, the LTD has its own BSP. The LTD board differs in many ways from the reference design. For example, the HW initialization, I/O, GPIO, powering and clock configuration have their own implementations or modifications. MQX configuration got the modifications from baseline as well. The BSP creation is depending on the SW development environment. In this case the IDE Integrated Development Environment (IDE) was the AR embedded work bench version 6.21.

5.1.3 Communication

USB On The Go

The USB OTG implementation faced problems. At the moment the LTD works only in the HOST mode. Freescale promised that MQX has a full OTG support with Kinetis SoCs in in July 2012. The USB communication is tested only with the Samsung Galaxy S2 Android phone.

Bluetooth Low Energy

The LTD BLE demo was done against the Nordic Semiconductor's software development kit (SDK). The LTD worked as slave and the master side was implemented in the PC. The proximity BLE profile was used. The proximity profile is specified to operate in two directions. More information for details of BLE PXP profile is described in the proximity profile specification. (6).

5.1.4 User Interface

The LEDs work as planned via GPIO lines. It is used in the test application to indicate the state of the application run.

The Buttons work as planned via GPIO lines. It is used to control the test application.

The Display works as planned with GPIO lines and dedicated pins. The test application uses the display to present the features are supported by the LTD software.

5.2 Further Development Possibilities

The LTD product platform gives many possibilities to create many kinds of applications. In general, the use of RTOS should be more efficient. An efficient operation mode control makes it possible to create powerful embedded applications. A comprehensive USB OTG

support helps to create more dynamic embedded systems. The bootloader helps to load a new firmware to the device. The bootloader can be handled for example via the USB or Ethernet. Some further development possibilities to improve the LTD product platform are listed below.

5.2.1 USB On The Go

There are some possibilities to handle the USB On the Go feature.

1. Freescale has promised that MQX will get a full OTG support with the Kinetis SoCs in July 2012.

2. The personal Health Care Device class (PHCD) stack has the OTG support which can be ported to the Kinetis K20 SoC.

5.2.2 Power Management

The Optimization of the hardware design can reduce the power consumption. MQX supports low power functions and many of them are only purposed for Kinetis chips. Adding some test points to the assembly of the LTD design would give possibilities to track the software operations and power consumption.

Charging

The first LTD assembly has not included the charging circuitry. The USB charging functionality is quite easy to implement to the LTD design.

Power modes

Now two operation modes are supported. They are RUN and VLPS (Very Low Power Stop) modes. The device has not a many functionalities in the VLPS mode. For example, the DMA data transfer is not usable in this mode. In the VLPR (Very Low Power Run)

mode many blocks are usable but the system clock is decreased down to 2MHz. Appendix 3 contains the comparison of the operation modes from the functionality's point of view.

5.2.3 Bootloader

Bootloader is a small program put into a device that allows the user application codes to be uploaded to the device. Figure 18 present a location of the bootloader code area in the K20 memory space. Bootloader helps to load a new firmware to the LTD based device. Bootloader can use either Ethernet or a USB. See more information about the creating of Ethernet and USB bootloader in documents. Using a bootloader application in MQX RTOS, it should be ensured that RTOS is stopped cleanly when jumping to the actual application. When a bootloader is used, the actual application must be linked in the way that its code is located in a certain memory address. (13.) (14.)

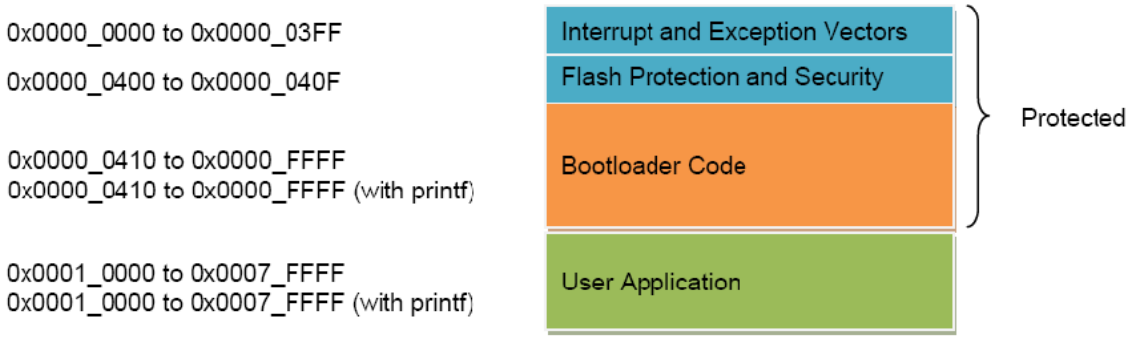

FIGURE 18. Kinetis K20 / K60 bootloader memory map (13).

5.2.4 Bluetooth Low Energy

A further development step for the BLE functionality would make the LTD devices communicate together via the BLE and with proximity profile, in the way that one of device works as monitor and the others as reporters. The reporter is a node which serves the monitor device. The monitor device can work as the gateway to the Internet.

REFERENCES

1. Aava Core Platform. Aava Mobile. 2010.
   Available at: http://aavamobile.com.s143311.gridserver.com/design/. Date of data acquisition. 30. May 2012.

2. Fusion Software. Navicron Oy.
   Available at http://www.navicron.com/products/navicron-sw. Date of data acquisition 30 May 2012.

3. WHITEspeed platform. Erni ltd. 2012.
   Available at: http://www.erni.com/en/news/single-product-news/neues-com-module-whitespeed/. Date of data acquisition 30 May 2012.

4. Paldanius, M. Lewel Group Finland Oy. 2012. LTD Test SW Design.
   D2035SP005_0.2 LTD_Test_SW_Design.doc

5. Board Support Package Porting Guide. Freescale Semiconductor. 2011.
   AN4287_Board Support Package Porting Guide.pdf

6. Proximity profile specification of BLE. Bluetooth.org. 2011.
   PXP_SPEC_V10[1].pdf.
   Available at: https://www.bluetooth.org/apps/content/. Date of data acquisition 13 February 2012.

7. Ylönen, H. Lewel Group Finland Oy. 2011. LT Engine Block Diagram.
   D2035SP009_0.3 LT Engine HW Block Diagram.pdf.

8. MQX IO Drivers User Guide. Freescale Semiconductor. 2011.
   MQXIOUG MQX IO Drivers User Guide.pdf

9. Ylönen, H. Lewel Group Finland Oy. 2011. LT Engine Hardware_Specification. D2035SP005_0.1. LT Engine Hardware_Specification.doc.

10. EEPROM configuration instructions. Freescale Semiconductor. 2011. AN4282_eeprom.pdf

11. MQX User Guide. Freescale Semiconductor. 2011. MQXUG MQX User Guide.pdf

12. K20 Sub-Family Datasheet. Freescale Semiconductor. 2012. Available at: http://cache.freescale.com/files/32bit/doc/data_sheet/K20P48M50SF0.pdf. Date of data acquisition 30 September 2011.

13. USB Bootloader. Freescale Semiconductor. 2011. AN4368_USB_Bootloader.pdf.

14. Ethernet Bootloader. Freescale Semiconductor. 2011. AN4367_Ethernet_Bootloader.pdf.

```
//-----------------------------------------------------------------------------

ltd_flash.h

   extern uint_32 flash_buffer_data_size;
   extern int_32 flash_buffer_init(void);
   extern int_32 flashbuf_seek(int_32 seek_location, int_32 seek_param);
   extern int_32 flashbuf_data_size();

//-----------------------------------------------------------------------------

ltd_pmc_services.h

void enterLTDtoSleep(void);

void enterLTDtoStopMode(void);

boolean is_battery_good(void);

boolean is_100V_good(void);

boolean is_INTVcc_good(void);

uint_8 initialize_voltage_monitoring(void);


//-----------------------------------------------------------------------------

ltd_test_services.h

int button1_pressed(void);

void app_create_tasks(void);

void app_stop_tasks(void);

void app_go_to_sleep(void);

void reset_selections(void);

void flash_led(uint_32);


void do_menu_action(int);

void disp_menu_scr(void);

void disp_button_scr(void);

void disp_led_scr(void);

void disp_buzzer_scr(void);
```

```
void disp_display_scr(void);
void disp_flash_scr(void);
void disp_power_scr(void);
void disp_bt_scr(void);
void disp_usb_scr(void);
void disp_uart_scr(void);
void disp_settings_scr(void);
void disp_info_scr(void);

void do_button_test(int);
void do_led_test(int);
void do_buzzer_test(uint_32, uint_32);
void do_display_test(int);
void do_flash_test(int);
void do_power_test(int);
void do_bt_test(int);
void do_usb_test(int);
void do_uart_test(int);
void do_settings(int);

void read_settings(void);
void write_settings(void);
void led_blink_all_1s(void);

//-----------------------------------------------------------------------------
protelK20.h
#ifdef LTD_ENABLE_DISPLAY_TASK
extern void display_task(uint_32);
```

```
#endif


#ifdef LTD_ENABLE_USB_FILESYSTEM

extern void USB_task(uint_32);

#endif


#ifdef LTD_ENABLE_LOGGING_TASK

extern void Logging_task(uint_32);

#endif


extern void main_task(uint_32);

extern void adc_task(uint_32);

extern void ltd_accm_task(uint_32 initial_data);

extern void Lebt_main(uint_32 initial_data);


//----------------------------------------------------------------------------

Display.h

extern LTD_STATUS init_lcd(void);

extern LTD_STATUS close_lcd(void);

extern void draw_pixel(uint_16 x, uint_16 y, uint_16 color);

extern void draw_string(uint_16 x, uint_16 y, const char* text, struct FONT_DEF
font);

extern void draw_char(uint_16 x, uint_16 y, const char c, struct FONT_DEF font,
uint_16 color);

extern void draw_icon_16(uint_16 x, uint_16 y, uint_16 color, uint_16 icon[]);

extern void draw_lewel_logo(uint_8 lewelBitmaps[]);

extern void draw_horizontal_line(uint_16 y);

extern void draw_vertical_line(uint_16 x);

extern void draw_line(uint_16 x_start, uint_16 y_start, uint_16 x_stop, uint_16
y_stop, uint_16 color);
```

```
extern void draw_line_dotted(uint_16 x0, uint_16 y0, uint_16 x1, uint_16 y1,
uint_16 empty, uint_16 solid, uint_16 color);

extern void draw_circle(uint_16 xCenter, uint_16 yCenter, uint_16 radius, uint_16
color);

extern void draw_circle_points(int cx, int cy, int x, int y, uint_16 color);

extern void draw_circle_filled(uint_16 xCenter, uint_16 yCenter, uint_16 radius,
uint_16 color);

extern void draw_corner_filled(uint_16 xCenter, uint_16 yCenter, uint_16 radius,
drawCornerPosition_t position, uint_16 color);

extern void draw_arrow(uint_16 x, uint_16 y, uint_16 size, drawDirection_t,
uint_16 color );

extern void draw_rectangle(uint_16 x0, uint_16 y0, uint_16 x1, uint_16 y1,
uint_16 color );

extern void draw_rectangle_filled(uint_16 x0, uint_16 y0, uint_16 x1, uint_16 y1,
uint_16 color );

extern void draw_rectangle_rounded( uint_16 x0, uint_16 y0, uint_16 x1, uint_16
y1, uint_16 color, uint_16 radius, drawRoundedCorners_t corners);

extern void draw_triangle(uint_16 x0, uint_16 y0, uint_16 x1, uint_16 y1, uint_16
x2, uint_16 y2, uint_16 color);

extern void draw_triangle_filled(uint_16 x0, uint_16 y0, uint_16 x1, uint_16 y1,
uint_16 x2, uint_16 y2, uint_16 color);

extern void draw_swap(uint_32 a, uint_32 b);

extern void draw_progress_bar(uint_16 x, uint_16 y, uint_16 width, uint_16
height, uint_16 color, uint_8 progress);

extern void draw_button(uint_16 x, uint_16 y, struct FONT_DEF font, uint_16
color, char* text);

extern void clear_screen();

extern void clear_box(uint_16 x_start, uint_16 x_stop, uint_16 y_start, uint_16
y_stop);

extern void clear_pixel(uint_16 x, uint_16 y);

extern uint_8 get_pixel(uint_16 x, uint_16 y);

extern void refresh(void); // Update screen with this method

extern void test_lcd();


//-------------------------------------------------------------------------
```

Flextimer.h

```
void FlexTimer_configuration(_mqx_uint flextimer_module, uint_32 freq);
void FlexTimer_enable_disable(_mqx_uint flextimer_module, boolean enable);
pointer _bsp_get_FlexTimer_base_address(_mqx_uint ftm_index);
static void FTM2_interrupt_handler(pointer param);
void beep(int, uint_32);
```

```
//-----------------------------------------------------------------------------
```

DAC.h

```
void dac0_clk_enable (void);
void dac1_clk_enable (void);
void dac0_1_clk_enable(void);
void DAC12_VreferenceInit(DAC_MemMapPtr dacx_base_ptr,unsigned char Vinselect);
int SET_DACx_BUFFER( DAC_MemMapPtr dacx_base_ptr, byte dacindex, int buffval);
void DACx_register_reset_por_values (DAC_MemMapPtr dacx_base_ptr);
void VREF_Init(void);
void DAC_init(void);
void DAC_write(int output_voltage);
```

```
//-----------------------------------------------------------------------------
```

ltd_acd.h

```
uint8_t ADC_init(void);
uint_8 ADC1_voltage_monitoring_init(void);
uint_16 ADC1_voltage_monitoring_read(int channel);
```

```
//-----------------------------------------------------------------------------
```

ltd_hmi.h

```
uint_32 hmi_init(void);
```

```
uint_32 leds_init(void);

uint_32 buttons_init(void);

uint_32 butt1_init(void);

void leds_close(void);

void buttons_close(void);

void butt1_close(void);

void led_on(uint_32 led_num);

void led_off(uint_32 led_num);

typedef void (*LTD_buttons)(void);

extern void ltd_butt_set_int(LTD_buttons buttIntCB);

extern void ltd_hmi_disable_int(void);

void led_on_off(uint_32 led_num, uint_32 mode);


//-------------------------------------------------------------------------

ltd_pmc.h

void pm_mode_init(uint_32 power_mode);

uint_16 get_battery_level(void);

uint_16 get_INTVcc_level(void);

uint_8 init_voltage_monitoring(void);


//-------------------------------------------------------------------------

max3353_mini_host_mode.h

uint_8 InitializeI2C_max3353();

void close_max3353(void);

void max3353_write_I2C(int i2c_device_address, uchar reg, uchar value);

void max3353_read_I2C(int i2c_device_address, int sensor, int length);

uint_8 max3353_mini_host_mode_init(void);
```

```
uint_8 max3353_vbus_on_off(boolean enable);


//-----------------------------------------------------------------------------
RDMMA845_accm.h
uint_32 I2C_RDMMA845_accm_init(void);
uint_32 I2C_RDMMA845_accm_read(int i2c_device_address, int sensor, int length);
uint_32 I2C_RDMMA845_accm_write(int i2c_device_address, uchar reg, uchar value);
void I2C_RDMMA845_accm_close(void);
uint_32 RDMMA845_accm_read_axels(    int_16 *acc_val_x,
                                     int_16 *acc_val_y,
                                     int_16 *acc_val_z );
uint_32 RDMMA845_accm_to_standby(void);
uint_32 RDMMA845_accm_activate(void);
```

```
#ifndef __small_ram_config_h__
#define __small_ram_config_h__

/* Disable heavy weight components */

#ifndef MQX_USE_IPC
#define MQX_USE_IPC                         0
#endif

#ifndef MQX_USE_LOGS
#define MQX_USE_LOGS                        1
#endif

#ifndef MQX_USE_SEMAPHORES
#define MQX_USE_SEMAPHORES                  1
#endif

#ifndef MQX_USE_SW_WATCHDOGS
#define MQX_USE_SW_WATCHDOGS                0
#endif

#ifndef MQX_USE_TIMER
#define MQX_USE_TIMER                       1
#endif

/* Other configuration */
#ifndef MQX_DEFAULT_TIME_SLICE_IN_TICKS
```

```
#define MQX_DEFAULT_TIME_SLICE_IN_TICKS     1
#endif

#ifndef MQX_LWLOG_TIME_STAMP_IN_TICKS
#define MQX_LWLOG_TIME_STAMP_IN_TICKS       1
#endif

#ifndef MQX_TIMER_USES_TICKS_ONLY
#define MQX_TIMER_USES_TICKS_ONLY           1
#endif

#ifndef MQX_EXTRA_TASK_STACK_ENABLE
#define MQX_EXTRA_TASK_STACK_ENABLE         0
#endif

#ifndef MQX_IS_MULTI_PROCESSOR
#define MQX_IS_MULTI_PROCESSOR              0
#endif

#ifndef MQX_MUTEX_HAS_POLLING
#define MQX_MUTEX_HAS_POLLING               0
#endif

#ifndef MQX_USE_INLINE_MACROS
#define MQX_USE_INLINE_MACROS               0
#endif
```

SMALL RAM CONFIGURATION (small_ram_config.h) APPENDIX 2/3

```
#ifndef MQX_USE_LWMEM_ALLOCATOR
#define MQX_USE_LWMEM_ALLOCATOR             1
#endif

#ifndef MQX_ROM_VECTORS
#define MQX_ROM_VECTORS                     1
#endif

#ifndef MQX_USE_IDLE_TASK
#define MQX_USE_IDLE_TASK                   0
#endif

#ifndef MQX_HAS_TIME_SLICE
#define MQX_HAS_TIME_SLICE                  1
#endif


#ifndef MQX_KERNEL_LOGGING
#define MQX_KERNEL_LOGGING                  1
#endif

#ifndef MQX_SPARSE_ISR_TABLE
#define MQX_SPARSE_ISR_TABLE                1

#ifndef MQX_SPARSE_ISR_SHIFT
#define MQX_SPARSE_ISR_SHIFT                3
#endif
```

```
#endif /* MQX_SPARSE_ISR_TABLE */



#ifndef MQX_TASK_DESTRUCTION
#define MQX_TASK_DESTRUCTION                0
#endif

#ifndef MQX_EXIT_ENABLED
#define MQX_EXIT_ENABLED                    0
#endif

/*
** MFS settings
*/

#ifndef MFSCFG_MINIMUM_FOOTPRINT
#define MFSCFG_MINIMUM_FOOTPRINT            1
#endif

/*
** RTCS settings
*/

#ifndef RTCS_MINIMUM_FOOTPRINT
#define RTCS_MINIMUM_FOOTPRINT              1
#endif
```

```
#ifndef RTCSCFG_ENABLE_LWDNS
#define RTCSCFG_ENABLE_LWDNS                1
#endif


#endif
/* EOF */
```

OPERATION MODE COMPARISON TABLE

APPENDIX 3/1

Modules	STOP	VLPR	VLPW	VLPS	LLS	VLLSx
I^2C[6]	static, address match wakeup	100 kbps	100 kbps	static, address match wakeup	static	OFF
CAN[7]	wakeup	FF	FF	wakeup	static	OFF
I^2S	static	FF	FF	static	static	OFF
TSI	wakeup	FF	FF	wakeup	wakeup[8]	wakeup[8]
FTM	static	FF	FF	static	static	OFF
PIT	static	FF	FF	static	static	OFF
PDB	static	FF	FF	static	static	OFF
LPT	FF	FF	FF	FF	FF	FF
Watchdog	FF	FF	FF	FF	static	OFF
EWM	static	FF	static	static	static	OFF
16-bit ADC	ADC internal clock only	FF	FF	ADC internal clock only	static	OFF
CMP[9]	HS or LS compare	FF	FF	HS or LS compare	LS compare	LS compare
6-bit DAC	static	FF	FF	static	static	static
VREF	FF	FF	FF	FF	static	OFF
12-bit DAC	static	FF	FF	static	static	static
USB FS/LS	static	static	static	static	static	OFF
USB DCD	static	FF	FF	static	static	OFF
USB Voltage Regulator	optional	optional	optional	optional	optional	optional

Modules	STOP	VLPR	VLPW	VLPS	LLS	VLLSx
Ethernet	wakeup	static	static	static	static	OFF
RTC - 32kHz OSC	FF	FF	FF	FF	FF	FF
EzPort	disabled	disabled	disabled	disabled	disabled	disabled
SDHC	wakeup	FF	FF	wakeup	static	OFF
GPIO	wakeup	FF	FF	wakeup	static, pins latched	OFF, pins latched
FlexBus	static	FF	FF	static	static	OFF
CRC	static	FF	FF	static	static	OFF
RNGB	static	FF	static	static	static	OFF
CMT	static	FF	FF	static	static	OFF
NVIC	static	FF	FF	static	static	OFF
Mode Controller	FF	FF	FF	FF	FF	FF
LLWU[10]	static	static	static	static	FF	FF

OPERATION MODE COMPARISON TABLE

Modules	STOP	VLPR	VLPW	VLPS	LLS	VLLSx
Regulator	ON	low power	low power	low power	low power	low power
LVD	ON	disabled	disabled	disabled	disabled	disabled
1kHz LPO	ON	ON	ON	ON	ON	ON
System oscillator (OSC)	ERCLK optional	ERCLK max of 4MHz crystal	ERCLK max of 4MHz crystal	ERCLK max of 4MHz crystal	limited to low range/low power	limited to low range/low power
MCG	static - IRCLK optional; PLL optionally on but gated	2 MHz IRC	2 MHz IRC	static - no clock output	static - no clock output	OFF
Core Clock	OFF	2 MHz max	OFF	OFF	OFF	OFF
System Clock	OFF	2 MHz max	2MHz max	OFF	OFF	OFF
Bus Clock	OFF	2 MHz max	2 MHz max	OFF	OFF	OFF
Flash	powered	1 MHz max access - no pgm	low power	low power	OFF	OFF
Portion of SRAM_U[1]	powered	powered	powered	powered	powered	powered in VLLS3,2
Remaining SRAM_U and all of SRAM_L	powered	powered	powered	powered	powered	powered in VLLS3
FlexMemory[2]	powered	powered[3]	powered	powered	powered	powered in VLLS3, optionally powered in VLLS2
Register File	powered	powered	powered	powered	powered	powered
DMA	static	FF	FF	static	static	OFF
UART[4]	static, wakeup on edge	125 kbps	125 kbps	static, wakeup on edge	static	OFF
SPI[5]	static	1 Mbps	1 Mbps	static	static	OFF