Getting Started With the Stellaris® EK-LM4F120XL LaunchPad Workshop

Version 1.10
Agenda

Introduction to ARM® Cortex™-M4F and Peripherals

- Code Composer Studio
- Introduction to StellarisWare, Initialization and GPIO
- Interrupts and the Timers
  - ADC12
  - Hibernation Module
- USB
- Memory
- Floating-Point
- BoosterPacks and grLib
- Synchronous Serial Interface
  - UART
  - μDMA
Stellaris® Roadmap

**ARM Cortex-M3**
- LM3S9000
- LM3S8000
- LM3S6000
  - Fixed Point
  - ENET MAC & PHY
  - USB & CAN options
  - RTP Feb ‘13 (TMX Now)
  - ARM Cortex-M3
  - Floating-Point

**ARM Cortex-M4F**
- LM4F23x
- LM4F13x
- RTP Feb ‘13 (TMX Now)
  - USB H/D/OTG + CAN
  - 80 MHz
  - 256K Flash / 32K SRAM
  - Low-power hibernate
  - 2 x 1 Msps 12-bit ADCs
  - Motion control options

- LM4F21x
- LM4F12x
- LM4F11x
- RTP Feb ‘13 (TMX Now)
  - 80 MHz
  - 256K Flash / 32K SRAM
  - Low-power hibernate
  - 2 x 1 Msps 12-bit ADCs
  - Up to 2 x CAN
  - Motion control options

- LM4F29x
- TMS / RTP 2H13
  - Ethernet + USB + CAN
  - 120 MHz
  - 1MB Flash, 256KB SRAM
  - 10/100 ENET MAC + PHY
  - USB H/D/OTG w/FS PHY & HS ULPI
  - Up to 2 x CAN
  - Parallel Bus Interface (EPI)
  - Crypto

- LM4F29x
- TMS / RTP 2H13
  - USB + CAN
  - 120 MHz
  - 1MB Flash, 256KB SRAM
  - USB H/D/OTG w/FS PHY & HS ULPI
  - Up to 2 x CAN
  - Parallel Bus Interface (EPI)
  - Crypto
Stellaris® LM4F120 Series MCUs

Connectivity features:
- CAN, USB Device, SPI/SSI, I2C, UARTs

High-performance analog integration
- Two 1 MSPS 12-bit ADCs
- Analog and digital comparators

Best-in-class power consumption
- As low as 370 µA/MHz
- 500µs wakeup from low-power modes
- RTC currents as low as 1.7µA

Solid roadmap
- Higher speeds
- Larger memory
- Ultra-low power
M4 Core and Floating-Point Unit

- **32-bit ARM® Cortex™-M4 core**
- **Thumb2 16/32-bit code**: 26% less memory & 25% faster than pure 32-bit
- **System clock frequency** up to 80 MHz
- **100 DMIPS @ 80MHz**
- **Flexible clocking system**
  - Internal precision oscillator
  - External main oscillator with PLL support
  - Internal low frequency oscillator
  - Real-time-clock through Hibernation module
- **Saturated math for signal processing**
- **Atomic bit manipulation**. Read-Modify-Write using bit-banding
- **Single Cycle multiply and hardware divider**
- **Unaligned data access** for more efficient memory usage
- **Privileged and unprivileged modes**
  - Limits access to MPU registers, SysTick, NVIC & possibly memory/peripherals
- **IEEE754 compliant single-precision floating-point unit**
- **JTW and Serial Wire Debug** debugger access
  - ETM available through Keil and IAR emulators
LM4F120H5QR Memory

256KB Flash memory
- Single-cycle to 40MHz
- Pre-fetch buffer and speculative branch improves performance above 40 MHz

32KB single-cycle SRAM with bit-banding

Internal ROM loaded with StellarisWare software
- Stellaris Peripheral Driver Library
- Stellaris Boot Loader
- Advanced Encryption Standard (AES) cryptography tables
- Cyclic Redundancy Check (CRC) error detection functionality

2KB EEPROM (fast, saves board space)
- Wear-leveled 500K program/erase cycles
- 10 year data retention
- 4 clock cycle read time

Peripherals...
**LM4F120H5QR Peripherals**

**Battery-backed Hibernation Module**
- Internal and external power control (through external voltage regulator)
- Separate real-time clock (RTC) and power source
- VDD3ON mode retains GPIO states and settings
- Wake on RTC or Wake pin
- 16 32-bit words of battery backed memory
- 5 µA Hibernate current with GPIO retention. 1.7 µA without

**Serial Connectivity**
- USB 2.0 (Device)
- 8-UART
- 4-I2C
- 4-SSI/SPI
- CAN
Two 1MSPS 12-bit SAR ADCs

- Twelve shared inputs
- Single ended and differential measurement
- Internal temperature sensor
- 4 programmable sample sequencers
- Flexible trigger control: SW, Timers, Analog comparators, GPIO
- VDDA/GNDA voltage reference
- Optional hardware averaging
- 2 analog and 16 digital comparators
- µDMA enabled

0 - 43 GPIO

- Any GPIO can be an external edge or level triggered interrupt
- Can initiate an ADC sample sequence or µDMA transfer directly
- Toggle rate up to the CPU clock speed on the Advanced High-Performance Bus
- 5-V-tolerant in input configuration
- Programmable Drive Strength (2, 4, 8 mA or 8 mA with slew rate control)
- Programmable weak pull-up, pull-down, and open drain

New Pin Mux GUI Tool: www.ti.com/StellarisPinMuxUtility
LM4F120H5QR Peripherals

Memory Protection Unit (MPU)
- Generates a Memory Management Fault on incorrect access to region

Timers
- 2 Watchdog timers with separate clocks
- SysTick timer. 24-bit high speed RTOS and other timer
- Six 32-bit and Six 64-bit general purpose timers
- PWM and CCP modes
- Daisy chaining
- User enabled stalling on CPU Halt flag from debugger for all timers

32 channel µDMA
- Basic, Ping-pong and scatter-gather modes
- Two priority levels
- 8, 16 and 32-bit data sizes
- Interrupt enabled

Nested-Vectored Interrupt Controller
- 7 exceptions and 65 interrupts with 8 programmable priority levels
- Tail-chaining
- Deterministic: always 12 cycles or 6 with tail-chaining
- Automatic system save and restore
Stellaris® LaunchPad

- ARM® Cortex™-M4F
  64-pin 80MHz LM4F120H5QR
- On-board USB ICDI
  (In-Circuit Debug Interface)
- Micro AB USB Device port
- Device/ICDI power switch
- BoosterPack XL pinout also supports existing Booster Packs
- 2 user pushbuttons
- Reset button
- 3 user LEDs (1 tri-color device)
- Current measurement test points
- 16MHz Main Oscillator crystal
- 32kHz Real Time Clock crystal
- 3.3V regulator
- Support for multiple IDEs:
Lab 1: Hardware and Software Setup

- Install the software
- Review the kit contents
- Connect the hardware
- Test the QuickStart application

USB Emulation Connection
Agenda

Introduction to ARM® Cortex™-M4F and Peripherals

**Code Composer Studio**

- Introduction to StellarisWare, Initialization and GPIO
- Interrupts and the Timers
  - ADC12
  - Hibernation Module
- USB
- Memory
- Floating-Point
- BoosterPacks and grLib
- Synchronous Serial Interface
  - UART
  - µDMA
# Development Tools for Stellaris MCUs

<table>
<thead>
<tr>
<th>Feature</th>
<th>Mentor Embedded</th>
<th>IAR Systems</th>
<th>ARM KEIL</th>
<th>TCS composer Studio</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Eval Kit License</strong></td>
<td>30-day full function. Upgradeable</td>
<td>32KB code size limited. Upgradeable</td>
<td>32KB code size limited. Upgradeable</td>
<td>Full function. Onboard emulation limited</td>
</tr>
<tr>
<td><strong>Compiler</strong></td>
<td>GNU C/C++</td>
<td>IAR C/C++</td>
<td>RealView C/C++</td>
<td>TI C/C++</td>
</tr>
<tr>
<td><strong>Debugger / IDE</strong></td>
<td>gdb / Eclipse</td>
<td>C-SPY / Embedded Workbench</td>
<td>µVision</td>
<td>CCS/Eclipse-based suite</td>
</tr>
<tr>
<td><strong>Full Upgrade</strong></td>
<td>99 USD personal edition / 2800 USD full support</td>
<td>2700 USD</td>
<td>MDK-Basic (256 KB) = €2000 (2895 USD)</td>
<td>445 USD</td>
</tr>
<tr>
<td><strong>JTAG Debugger</strong></td>
<td>J-Link, 299 USD</td>
<td>U-Link, 199 USD</td>
<td>XDS100, 79 USD</td>
<td></td>
</tr>
</tbody>
</table>

What is CCS?...
What is Code Composer Studio?

Integrated development environment for TI embedded processors
- Includes debugger, compiler, editor, simulator, OS…
- The IDE is built on the Eclipse open source software framework
- Extended by TI to support device capabilities

CCSv5 is based on “off the shelf” Eclipse (version 3.7 in CCS 5.2)
- Uses unmodified version of Eclipse
  - TI contributes changes directly to the open source community
- Drop in Eclipse plug-ins from other vendors or take TI tools and drop them into an existing Eclipse environment
- Users can take advantage of all the latest improvements in Eclipse

Integrate additional tools
- OS application development tools (Linux, Android, Sys/BIOS…)  
- Code analysis, source control…

Runs under Windows and Linux
$445 single seat. $99/year subscription fee
User Interface Modes

Simple Mode
- By default CCS will open in simple/basic mode
- Simplified user interface with far fewer menu items, toolbar buttons
- TI supplied CCS Edit and CCS Debug Perspectives

Advanced Mode
- Uses default Eclipse perspectives (similar to what existed in CCSv4)
- Recommended for users integrating other Eclipse based tools into CCS

Switching Modes
- On the CCS menu bar, select Window → Open Perspective → Other… Check the “Show all” checkbox
  “C/C++” and “Debug” are the advanced perspectives
Common Tasks

Creating New Projects
- Very simple to create a new project for a device using a template

Build options
- Simplified build options dialog from earlier CCS versions
- Updates to options are delivered via compiler releases and not dependent on CCS updates

Sharing projects
- Easy for users to share projects, including working with version control (portable projects)
- Setting up linked resources has been simplified from earlier CCS versions
A workspace contains your settings and preferences, as well as links to your projects. Deleting projects from the workspace deletes the links, not the files*

A project contains your build and tool settings, as well as links to your input files. Deleting files from the workspace deletes the links, not the files*

* Unless you have located or copied files into the workspace
Project Wizard

Single page wizard for majority of users
- Next button will show up if a template requires additional settings

Debugger setup included
- User chooses location, device and connection
- A modifiable ccxml file is created

Simple by default
- Compiler version, endianness… are under advanced settings
Adding Files to Projects

- Add Files to Project allows users to control how the file is added to the project
- Linking Files using built-in macros allows easy creation of portable projects
Lab 2: Code Composer Studio

- Create a new project
- Experiment with some CCS features
- Use the LM Flash Programmer
Agenda

Introduction to ARM® Cortex™-M4F and Peripherals
  Code Composer Studio

Introduction to StellarisWare, Initialization and GPIO

Interrupts and the Timers
  ADC12
  Hibernation Module
  USB
  Memory
  Floating-Point
  BoosterPacks and grLib

Synchronous Serial Interface
  UART
  µDMA
License-free and Royalty-free source code for TI Cortex-M devices:

- Peripheral Driver Library
- Graphics Library
- USB Library
- Ethernet stacks
- In-System Programming
StellarisWare Features

Peripheral Driver Library
- High-level API interface to complete peripheral set
- License & royalty free use for TI Cortex-M parts
- Available as object library and as source code
- Programmed in the on-chip ROM

Graphics Library
- Graphics primitive and widgets
- 153 fonts plus Asian and Cyrillic
- Graphics utility tools

USB Stacks and Examples
- USB Device and Embedded Host compliant
- Device, Host, OTG and Windows-side examples
- Free VID/PID sharing program

Ethernet
- lwip and uip stacks with 1588 PTP modifications
- Extensive examples

Extras
- SimpliciTI wireless protocol
- IQ math examples
- Bootloaders
- Windows side applications
In System Programming Options

Stellaris Serial Flash Loader
- Small piece of code that allows programming of the flash without the need for a debugger interface.
- All Stellaris MCUs ship with this pre-loaded in flash
- UART or SSI interface option
- The LM Flash Programmer interfaces with the serial flash loader
- See application note SPMA029

Stellaris Boot Loader
- Preloaded in ROM or can be programmed at the beginning of flash to act as an application loader
- Can also be used as an update mechanism for an application running on a Stellaris microcontroller.
- Interface via UART (default), I²C, SSI, Ethernet, USB (DFU H/D)
- Included in the Stellaris Peripheral Driver Library with full applications examples
Fundamental Clock Sources

**Precision Internal Oscillator (PIOSC)**
- 16 MHz ± 3%

**Main Oscillator (MOSC) using...**
- An external single-ended clock source
- An external crystal

**Internal 30 kHz Oscillator**
- 30 kHz ± 50%
- Intended for use during Deep-Sleep power-saving modes

**Hibernation Module Clock Source**
- 32,768Hz crystal
- Intended to provide the system with a real-time clock source
**System (CPU) Clock Sources**

The CPU can be driven by any of the fundamental clocks …

- Internal 16 MHz
- Main
- Internal 30 kHz
- External Real-Time

**- Plus -**

- The internal PLL (400 MHz)
- The internal 16MHz oscillator divided by four (4MHz ± 3%)

<table>
<thead>
<tr>
<th>Clock Source</th>
<th>Drive PLL?</th>
<th>Used as SysClk?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal 16MHz</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Internal 16Mhz/4</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Main Oscillator</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Internal 30 kHz</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Hibernation Module</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>PLL</td>
<td>-</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Stellaris Clock Tree

driverLib API SysCtlClockSet() selects:
- SYSDIV divider setting
- OSC or PLL
- Main or Internal oscillator
- Crystal frequency

- USB Clock
- UART Baud Clock
- System Clock
- SSI Baud Clock
- ADC Clock
General Purpose IO

- Any GPIO can be an interrupt:
  - Edge-triggered on rising, falling or both
  - Level-sensitive on high or low values
- Can directly initiate an ADC sample sequence or μDMA transfer
- Toggle rate up to the CPU clock speed on the Advanced High-Performance Bus. ½ CPU clock speed on the Standard.
- 5V tolerant in input configuration
- Programmable Drive Strength (2, 4, 8mA or 8mA with slew rate control)
- Programmable weak pull-up, pull-down, and open drain
- Pin state can be retained during Hibernation mode

New Pin Mux GUI Tool: www.ti.com/StellarisPinMuxUtility
GPIO Address Masking

Each GPIO port has a base address. You can write an 8-bit value directly to this base address and all eight pins are modified. If you want to modify specific bits, you can use a bit-mask to indicate which bits are to be modified. This is done in hardware by mapping each GPIO port to 256 addresses. Bits 9:2 of the address bus are used as the bit mask.

The register we want to change is GPIO Port D (0x4005.8000). Current contents of the register is:

```
00011101
```

The value we will write is 0xEB:

```
11101011
```

Instead of writing to GPIO Port D directly, write to 0x4005.8098. Bits 9:2 (shown here) become a bit-mask for the value you write.

Only the bits marked as “1” in the bit-mask are changed.

```
00111011
```

GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_5|GPIO_PIN_2|GPIO_PIN_1, 0xEB);

Note: you specify base address, bit mask, and value to write. The GPIOPinWrite() function determines the correct address for the mask.
Lab 3: Initialization and GPIO

- Configure the system clock
- Enable and configure GPIO
- Use a software delay to toggle an LED on the evaluation board
Agenda

Introduction to ARM® Cortex™-M4F and Peripherals
  Code Composer Studio
  Introduction to StellarisWare, Initialization and GPIO

Interrupts and the Timers
  ADC12
  Hibernation Module
  USB
  Memory
  Floating-Point
  BoosterPacks and grLib
  Synchronous Serial Interface
  UART
  μDMA
Nested Vectored Interrupt Controller (NVIC)

- Handles exceptions and interrupts
- 8 programmable priority levels, priority grouping
- 7 exceptions and 65 Interrupts
- Automatic state saving and restoring
- Automatic reading of the vector table entry
- Pre-emptive/Nested Interrupts
- Tail-chaining
- Deterministic: always 12 cycles or 6 with tail-chaining

Motor control ISRs (e.g. PWM, ADC)
Communication ISRs (e.g. CAN)
Main application (foreground)

Tail Chaining...
Interrupt Latency - Tail Chaining

Typical processor

Cortex-M4 Interrupt handling in HW

Highest Priority

IRQ1

IRQ2

Pre-emption ...
Interrupt Latency – Late Arrival

Typical processor:
- PUSH
- PUSH
- ISR 1
- POP
- PUSH
- ISR 2
- POP

Cortex-M4:
- PUSH
- ISR 1
- ISR 2
- POP

Highest Priority

IRQ1

IRQ2

Interrupt handling...
Interrupt handling is automatic. No instruction overhead.

**Entry**
- Automatically pushes registers R0–R3, R12, LR, PSR, and PC onto the stack
- In parallel, ISR is pre-fetched on the instruction bus. ISR ready to start executing as soon as stack PUSH complete

**Exit**
- Processor state is automatically restored from the stack
- In parallel, interrupted instruction is pre-fetched ready for execution upon completion of stack POP
# Cortex-M4® Exception Types

<table>
<thead>
<tr>
<th>Vector Number</th>
<th>Exception Type</th>
<th>Priority</th>
<th>Vector address</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Reset</td>
<td>-3</td>
<td>0x04</td>
<td>Reset</td>
</tr>
<tr>
<td>2</td>
<td>NMI</td>
<td>-2</td>
<td>0x08</td>
<td>Non-Maskable Interrupt</td>
</tr>
<tr>
<td>3</td>
<td>Hard Fault</td>
<td>-1</td>
<td>0x0C</td>
<td>Error during exception processing</td>
</tr>
<tr>
<td>4</td>
<td>Memory Management Fault</td>
<td>Programmable</td>
<td>0x10</td>
<td>MPU violation</td>
</tr>
<tr>
<td>5</td>
<td>Bus Fault</td>
<td>Programmable</td>
<td>0x14</td>
<td>Bus error (Prefetch or data abort)</td>
</tr>
<tr>
<td>6</td>
<td>Usage Fault</td>
<td>Programmable</td>
<td>0x18</td>
<td>Exceptions due to program errors</td>
</tr>
<tr>
<td>7-10</td>
<td>Reserved</td>
<td>-</td>
<td>0x1C - 0x28</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>SVCall</td>
<td>Programmable</td>
<td>0x2C</td>
<td>SVC instruction</td>
</tr>
<tr>
<td>12</td>
<td>Debug Monitor</td>
<td>Programmable</td>
<td>0x30</td>
<td>Exception for debug</td>
</tr>
<tr>
<td>13</td>
<td>Reserved</td>
<td>-</td>
<td>0x34</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>PendSV</td>
<td>Programmable</td>
<td>0x38</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>SysTick</td>
<td>Programmable</td>
<td>0x3C</td>
<td>System Tick Timer</td>
</tr>
<tr>
<td>16 and above</td>
<td>Interrupts</td>
<td>Programmable</td>
<td>0x40</td>
<td>External interrupts (Peripherals)</td>
</tr>
</tbody>
</table>
Cortex-M4® Vector Table

- After reset, vector table is located at address 0
- Each entry contains the address of the function to be executed
- The value in address 0x00 is used as starting address of the Main Stack Pointer (MSP)
- Vector table can be relocated by writing to the VTABLE register (must be aligned on a 1KB boundary)
- Open startup_ccs.c to see vector table coding
General Purpose Timer Module

- Six 16/32-bit and Six 32/64-bit general purpose timers
- Twelve 16/32-bit and Twelve 32/64-bit capture/compare/PWM pins

Timer modes:
- One-shot
- Periodic
- Input edge count or time capture with 16-bit prescaler
- PWM generation (separated only)
- Real-Time Clock (concatenated only)

- Count up or down
- Simple PWM (no deadband generation)

Support for timer synchronization, daisy-chains, and stalling during debugging

- May trigger ADC samples or DMA transfers
Lab 4: Interrupts and the GP Timer

- Enable and configure the Timer
- Enable and configure Interrupts
- Write the ISR code and test
- Generate an exception
Agenda

Introduction to ARM® Cortex™-M4F and Peripherals
- Code Composer Studio
- Introduction to StellarisWare, Initialization and GPIO
- Interrupts and the Timers

ADC12
- Hibernation Module
- USB
- Memory
- Floating-Point
- Booster Packs and grLib

Synchronous Serial Interface
- UART
- µDMA
Analog-to-Digital Converter

- Stellaris LM4F MCUs feature two ADC modules (ADC0 and ADC1) that can be used to convert continuous analog voltages to discrete digital values.
- Each ADC module has 12-bit resolution.
- Each ADC module operates independently and can:
  - Execute different sample sequences.
  - Sample any of the shared analog input channels.
  - Generate interrupts & triggers.

![Diagram of ADC](image-url)
LM4F120H5QR ADC Features

- Two 12-bit 1MSPS ADCs
- 12 shared analog input channels
- Single ended & differential input configurations
- On-chip temperature sensor
- Maximum sample rate of one million samples/second (1MSPS).
- Fixed references (VDDA/GNDA) due to pin-count limitations
- 4 programmable sample conversion sequencers per ADC
- Separate analog power & ground pins

- Flexible trigger control
  - Controller/ software
  - Timers
  - Analog comparators
  - GPIO
- 2x to 64x hardware averaging
- 8 Digital comparators / per ADC
- 2 Analog comparators
- Optional phase shift in sample time, between ADC modules ... programmable from 22.5° to 337.5°
ADC Sample Sequencers

- Stellaris LM4F ADC’s collect and sample data using programmable sequencers.
- Each sample sequence is a fully programmable series of consecutive (back-to-back) samples that allows the ADC module to collect data from multiple input sources without having to be re-configured.
- Each ADC module has 4 sample sequencers that control sampling and data capture.
- All sample sequencers are identical except for the number of samples they can capture and the depth of their FIFO.
- To configure a sample sequencer, the following information is required:
  - Input source for each sample
  - Mode (single-ended, or differential) for each sample
  - Interrupt generation on sample completion for each sample
  - Indicator for the last sample in the sequence
- Each sample sequencer can transfer data independently through a dedicated μDMA channel.

<table>
<thead>
<tr>
<th>Sequencer</th>
<th>Number of Samples</th>
<th>Depth of FIFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS 3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SS 2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>SS 1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>SS 0</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>
Lab 5: ADC12

- Enable and configure ADC and sequencer
- Measure and display values from internal temperature sensor
- Add hardware averaging
- Use ROM peripheral driver library calls and note size difference

USB Emulation Connection

Agenda...
Agenda

Introduction to ARM® Cortex™-M4F and Peripherals
Code Composer Studio
Introduction to StellarisWare, Initialization and GPIO
Interrupts and the Timers
ADC12
Hibernation Module
USB
Memory
Floating-Point
BoosterPacks and grLib
Synchronous Serial Interface
UART
µDMA
Key Features

- Real Time Clock is a 32-bit seconds counter with a 15-bit sub seconds counter & add-in trim capability
- Dedicated pin for waking using an external signal
- RTC operational and hibernation memory valid as long as $V_{\text{BAT}}$ is valid
- GPIO pins state retention provided during VDD3ON mode
- Two mechanisms for power control
  - System Power Control for CPU and other on-board hardware
  - On-chip Power Control for CPU only
- Low-battery detection, signaling, and interrupt generation, with optional wake on low battery
- 32,768 Hz external crystal or an external oscillator clock source
- 16 32-bit words of battery-backed memory are provided for you to save the processor state to during hibernation
- Programmable interrupts for RTC match, external wake, and low battery events.
Run mode

- Sleep mode stops the processor clock
  - 2 SysClk wakeup time

Deep Sleep mode stops the system clock and switches off the PLL and Flash
- 1.25 – 350 µS wakeup time

Hibernate mode with only hibernate module powered (VDD3ON, RTC and no RTC)
- ~500µS wakeup time
## Power Mode Comparison

<table>
<thead>
<tr>
<th>Mode</th>
<th>Run Mode</th>
<th>Sleep Mode</th>
<th>Deep Sleep Mode</th>
<th>Hibernation (VDD3ON)</th>
<th>Hibernation (RTC)</th>
<th>Hibernation (no RTC)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter ↓</strong></td>
<td><strong>I\text{DD}</strong></td>
<td>32 mA</td>
<td>10 mA</td>
<td>1.05 mA</td>
<td>5 (\mu\text{A})</td>
<td>1.7 (\mu\text{A})</td>
</tr>
<tr>
<td></td>
<td><strong>V\text{DD}</strong></td>
<td>3.3 V</td>
<td>3.3 V</td>
<td>3.3 V</td>
<td>3.3 V</td>
<td>0 V</td>
</tr>
<tr>
<td></td>
<td><strong>V\text{BAT}</strong></td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
<td>3 V</td>
<td>3 V</td>
</tr>
<tr>
<td></td>
<td><strong>System Clock</strong></td>
<td>40 MHz with PLL</td>
<td>40 MHz with PLL</td>
<td>30 kHz</td>
<td>Off</td>
<td>Off</td>
</tr>
<tr>
<td></td>
<td><strong>Core</strong></td>
<td>Powered On</td>
<td>Powered On</td>
<td>Powered On</td>
<td>Off</td>
<td>Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clocked</td>
<td>Not Clocked</td>
<td>Not Clocked</td>
<td>Not Clocked</td>
<td>Not Clocked</td>
</tr>
<tr>
<td></td>
<td><strong>Peripherals</strong></td>
<td>All On</td>
<td>All Off</td>
<td>All Off</td>
<td>All Off</td>
<td>All Off</td>
</tr>
<tr>
<td></td>
<td><strong>Code</strong></td>
<td>while{1}</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

Box denotes power modes available on LaunchPad board
LaunchPad Considerations

- The low-cost LaunchPad board does not have a battery holder
- VDD and VBAT are wired together on the board (this disables battery-only powered low-power modes)
- Device current is measured between test points H24 and H25
Lab 6: Low Power Modes

- Place device in low power modes
- Wake from pin
- Wake from RTC
- Measure current
- No battery holder on board
Agenda

Introduction to ARM® Cortex™-M4F and Peripherals
  Code Composer Studio
  Introduction to StellarisWare, Initialization and GPIO
  Interrupts and the Timers
    ADC12
  Hibernation Module

USB
  Memory
  Floating-Point
  BoosterPacks and grLib
  Synchronous Serial Interface
    UART
    μDMA
Multiple connector sizes

4 pins – power, ground and 2 data lines
(5th pin ID for USB 2.0 connectors)

Configuration connects power 1st, then data

Standards:

- **USB 1.1**
  - Defines **Host** (master) and **Device** (slave)
  - Speeds to 12Mbits/sec
  - Devices can consume 500mA (100mA for startup)

- **USB 2.0**
  - Speeds to 480Mbits/sec
  - OTG addendum

- **USB 3.0**
  - Speeds to 4.8Gbits/sec
  - New connector(s)
  - Separate transmit/receive data lines
USB Basics

USB Device ... most USB products are slaves
USB Host ... usually a PC, but can be embedded
USB OTG ... On-The-Go

- Dynamic switching between host and device roles
- Two connected OTG ports undergo host negotiation

Host polls each Device at power up. Information from Device includes:

- Device Descriptor (Manufacturer & Product ID so Host can find driver)
- Configuration Descriptor (Power consumption and Interface descriptors)
- Endpoint Descriptors (Transfer type, speed, etc)
- Process is called Enumeration ... allows Plug-and-Play
LM4F120H5QR USB

- USB 2.0 Device mode full speed (12 Mbps) and low speed (1.5 Mbps) operation
- Integrated PHY
- Transfer types: Control, Interrupt, Bulk and Isochronous
- Device Firmware Update (DFU) device in ROM

Stellaris collaterals
- Texas Instruments is a member of the USB Implementers Forum.
- Stellaris is approved to use the USB logo
- Vendor/Product ID sharing
Integrated USB Controller and PHY with up to 16 Endpoints

- 1 dedicated control IN endpoint and 1 dedicated control OUT endpoint
- Up to 7 configurable IN endpoints and 7 configurable OUT endpoints
- 4 KB dedicated endpoint memory (not part of device SRAM)
- Separate DMA channels (up to three IN Endpoints and three OUT Endpoints)
- 1 endpoint may be defined for double-buffered 1023-bytes isochronous packet size
StellarisWare USBLib

- License-free & royalty-free drivers, stack and example applications for Stellaris MCUs
- USBLib supports Host/Device and OTG, but the LM4F120H5QR USB port is Device only
- Builds on DriverLib API
  - Adds framework for generic Host and Device functionality
  - Includes implementations of common USB classes
- Layered structure
- Drivers and .inf files included where appropriate
- Stellaris MCUs have passed USB Device and Embedded Host compliance testing

Device Examples
- HID Keyboard
- HID Mouse
- CDC Serial
- Mass Storage
- Generic Bulk
- Audio
- Device Firmware Upgrade
- Oscilloscope

Windows INF for supported devices
- Points to base Windows drivers
- Sets config string
- Sets PID/VID
- Precompiled DLL saves development time

Device framework integrated into USBLib

Abstraction Levels...
USB API Abstraction Levels

- **Application 1**: Passes simplified data to a higher level API. (Custom HID mouse)
- **Application 2**: Passes key info to the Driver API. Driver API handles all lower level functions for the chosen class. (Custom HID device)
- **Application 3**: Uses existing API for generic host/device operation. Uses DriverLib for features not covered by these APIs. (Custom Classes)
- **Application 4**: Implements its own USB protocol using Driverlib. (Third party USB stack)
Lab 7: USB

- Run usb_bulk_example code and windows side app
- Inspect stack setup
- Observe data on device
Agenda

Introduction to ARM® Cortex™-M4F and Peripherals
  Code Composer Studio
  Introduction to StellarisWare, Initialization and GPIO
  Interrupts and the Timers
  ADC12
  Hibernation Module
  USB
  Memory
    Floating-Point
    BoosterPacks and grLib
  Synchronous Serial Interface
    UART
    µDMA
Flash, SRAM and ROM Control

Memory Blocks and Control Logic for:
- SRAM
- ROM
- Flash
EEPROM Control

- EEPROM Block and Control Logic
- EEPROM block is connected to the AHB (Advanced High Performance Bus)

Flash Features...
**Flash**

- 256KB / 40MHz starting at 0x00000000
- Organized in 1KB independently erasable blocks
- Code fetches and data access occur over separate buses
- Below 40MHz, Flash access is single cycle
- Above 40MHz, the prefetch buffer fetches two 32-bit words/cycle. No wait states for sequential code.
- Branch speculation avoids wait state on some branches
- Programmable write and execution protection available
- Simple programming interface

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>Flash</td>
</tr>
<tr>
<td>0x01000000</td>
<td>ROM</td>
</tr>
<tr>
<td>0x20000000</td>
<td>SRAM</td>
</tr>
<tr>
<td>0x22000000</td>
<td>Bit-banded SRAM</td>
</tr>
<tr>
<td>0x40000000</td>
<td>Peripherals &amp; EEPROM</td>
</tr>
<tr>
<td>0x42000000</td>
<td>Bit-banded Peripherals</td>
</tr>
<tr>
<td>0x45000000</td>
<td>Instrumentation, ETM, etc.</td>
</tr>
</tbody>
</table>

EEPROM...
EEPROM

- 2KB of memory starting at 0x400AF000 in Peripheral space
- Accessible as 512 32-bit words
- 32 blocks of 16 words (64 bytes) with access protection per block
- Built-in wear leveling with endurance of 500K writes
- Lock protection option for the whole peripheral as well as per block using 32-bit to 96-bit codes
- Interrupt support for write completion to avoid polling
- Random and sequential read/write access (4 cycles max/word)
SRAM

- 32KB / 80MHz starting at 0x20000000
- Bit banded to 0x22000000
- Can hold code or data

---

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>Flash</td>
</tr>
<tr>
<td>0x01000000</td>
<td>ROM</td>
</tr>
<tr>
<td>0x20000000</td>
<td>SRAM</td>
</tr>
<tr>
<td>0x22000000</td>
<td>Bit-banded SRAM</td>
</tr>
<tr>
<td>0x40000000</td>
<td>Peripherals &amp; EEPROM</td>
</tr>
<tr>
<td>0x42000000</td>
<td>Bit-banded Peripherals</td>
</tr>
<tr>
<td>0xE0000000</td>
<td>Instrumentation, ETM, etc.</td>
</tr>
</tbody>
</table>
Bit-Banding

- Reduces the number of read-modify-write operations
- SRAM and Peripheral space use address aliases to access individual bits in a single, atomic operation
- SRAM starts at base address 0x20000000
  Bit-banded SRAM starts at base address 0x22000000
- Peripheral space starts at base address 0x40000000
  Bit-banded peripheral space starts at base address 0x42000000

The bit-band alias is calculated by using the formula:

$$\text{bit-band alias} = \text{bit-band base} + (\text{byte offset} \times 0x20) + (\text{bit number} \times 4)$$

For example, bit-7 at address 0x20002000 is:

$$0x20002000 + (0x2000 \times 0x20) + (7 \times 4) = 0x2204001C$$
Memory Protection Unit (MPU)

- Defines 8 separate memory regions plus a background region accessible only from privileged mode
- Regions of 256 bytes or more are divided into 8 equal-sized sub-regions
- MPU definitions for all regions include:
  - Location
  - Size
  - Access permissions
  - Memory attributes
- Accessing a prohibited region causes a memory management fault
Cortex M4 Privilege Levels

- Privilege levels offer additional protection for software, particularly operating systems

- **Unprivileged**: software has ...
  - Limited access to the Priority Mask register
  - No access to the system timer, NVIC, or system control block
  - Possibly restricted access to memory or peripherals (FPU, MPU, etc)

- **Privileged**: software has ...
  - use of all the instructions and has access to all resources

- ISRs operate in privileged mode
- Thread code operates in unprivileged mode unless the level is changed via the Thread Mode Privilege Level (TMPL) bit in the CONTROL register
Lab 8: Memory and the MPU

- Create code to write to Flash
- Create code to read/write EEPROM
- Experiment with MPU and bit-banding

USB Emulation Connection
Agenda

Introduction to ARM® Cortex™-M4F and Peripherals
  Code Composer Studio
  Introduction to StellarisWare, Initialization and GPIO
  Interrupts and the Timers
    ADC12
    Hibernation Module
  USB
  Memory
  Floating-Point
    BoosterPacks and grLib
  Synchronous Serial Interface
    UART
    µDMA

What is Floating-Point?...
What is Floating-Point?

- Floating-point is a way to represent real numbers on computers

- IEEE floating-point formats:
  - Half (16-bit) →
  - Single (32-bit) →
  - Double (64-bit) →
  - Quadruple (128-bit) →

What is IEEE-754?...
What is IEEE-754?

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Sign (s)</th>
<th>Exponent (e)</th>
<th>Fraction (f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>0 1 0 0 0 1 1 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Decimal Value = \((-1)^s \times (1+f) \times 2^{e-bias}\)

where: \(f = \sum [(b_i)2^{-i}] \forall i \in (1,23)\)

bias = 127 for single precision floating-point

\[\text{sign} = (-1)^0, \text{exponent} = [10000110]_2 = [134]_{10}, \text{fraction} = [0.11010000100000000000000000000000]_2 = [0.814453]_{10}\]

**Decimal Value**

\[\begin{align*}
&= [1]_{10} \times ([1]_{10} + [0.814453]_{10}) \times [2^{134-127}]_{10} \\
&= [1.814453]_{10} \times 128 \\
&= [232.249]_{10}
\end{align*}\]
Floating-Point Unit (FPU)

- The FPU provides floating-point computation functionality that is compliant with the IEEE 754 standard.
- Enables conversions between fixed-point and floating-point data formats, and floating-point constant instructions.
- The Cortex-M4F FPU fully supports single-precision:
  - Add
  - Subtract
  - Multiply
  - Divide
  - Single cycle multiply and accumulate (MAC)
  - Square root
There are three different modes of operation for the FPU:

- **Full-Compliance mode** – In Full-Compliance mode, the FPU processes all operations according to the IEEE 754 standard in hardware. **No support code is required.**

- **Flush-to-Zero mode** – A result that is very small, as described in the IEEE 754 standard, where the destination precision is smaller in magnitude than the minimum normal value before rounding, is replaced with a zero.

- **Default NaN (not a number) mode** – In this mode, the result of any arithmetic data processing operation that involves an input NaN, or that generates a NaN result, returns the default NaN. ( $0 / 0 = \text{NaN}$ )
FPU Registers

- Sixteen 64-bit double-word registers, D0-D15
- Thirty-two 32-bit single-word registers, S0-S31
FPU Usage

- **The FPU is disabled from reset.** You must **enable it** before you can use any floating-point instructions. The processor must be in privileged mode to read from and write to the Coprocessor Access Control (CPAC) register.

- **Exceptions:** The FPU sets the cumulative exception status flag in the FPSCR register as required for each instruction. The FPU does not support user-mode traps.

- The processor can reduce the exception latency by using **lazy stacking**. This means that the processor reserves space on the stack for the FPU state, but does not save that state information to the stack.

* with a StellarisWare API function call
CMSIS* DSP Library Performance

* - ARM® Cortex™ Microcontroller Software Interface Standard

- DSP Library Benchmark: Cortex M3 vs. Cortex M4 (SIMD + FPU)
  - Fixed-point ~ 2x faster
  - Floating-point ~ 10x faster

Source: ARM CMSIS Partner Meeting Embedded World, Reinhard Keil
Lab 9: FPU

- Experiment with the FPU
- Profile floating-point code
Agenda

Introduction to ARM® Cortex™-M4F and Peripherals
  Code Composer Studio
  Introduction to StellarisWare, Initialization and GPIO
  Interrupts and the Timers
  ADC12
  Hibernation Module
  USB
  Memory
  Floating-Point
  **Booster Packs and grLib**
  Synchronous Serial Interface
  UART
  μDMA
TI LaunchPad Boards

MSP430
$9.99US

Stellaris
$12.99US

C2000 Piccolo
$17.00US

BoosterPack Connectors...
BoosterPack Connectors

◆ **Original Format (MSP430)**
  - VCC and Ground
  - 14 GPIO
  - Emulator Reset and Test
  - Crystal inputs or 2 more GPIO

◆ **XL Format (Stellaris/C2000)** is a superset of the original, adding two rows of pins with:
  - USB $V_{BUS}$ and Ground
  - 18 additional GPIO
Some of the Available Booster Packs

- Solar Energy Harvesting
- RF Module w/ LCD
- Olimex 8x8 LED Matrix
- TMP006 IR Temperature Sensor
- Universal Energy Harvesting
- Inductive Charging
- Sub-1GHz RF Wireless
- C5000 Audio
- Capacitive Touch
- Proto Board
- TPL0401 SPI Digital Pot.
- TPL0501 SPI Digital Pot.

Available Booster Packs...
Some of the Available Booster Packs

- Proto board
- ZigBee Networking
- OLED Display
- LCD Controller Development Package
- MOD Board Adapter
- Click Board Adapter

Kentec LCD Display...
KenTec TouchScreen TFT LCD Display

- Part# EB-LM4F120-L35
- Designed for XL BoosterPack pinout
- 3.5” QVGA TFT 320x240x16 color LCD with LED backlight
- Driver circuit and connector are compatible with 4.3”, 5”, 7” & 9” displays
- Resistive Touch Overlay
The Stellaris Graphics Library provides graphics primitives and widgets sets for creating graphical user interfaces on Stellaris controlled displays.

Note that Stellaris devices do not have an LCD interface. The interface to smart displays is done through serial or EPI ports.

The graphics library consists of three layers to interface your application to the display:

- **Display Driver Layer**: User written or modified
- **Graphics Primitives Layer**
- **Widget Layer**

Your Application Code

* = user written or modified
The design of the graphics library is governed by the following goals:

- Components are written entirely in C except where absolutely not possible.
- Your application can call any of the layers.
- The graphics library is easy to understand.
- The components are reasonably efficient in terms of memory and processor usage.
- Components are as self-contained as possible.
- Where possible, computations that can be performed at compile time are done there instead of at run time.
Display Driver

Low level interface to the display hardware

Routines for display-dependent operations like:
- Initialization
- Backlight control
- Contrast
- Translation of 24-bit RGB values to screen dependent color map

Drawing routines for the graphics library like:
- Flush
- Line drawing
- Pixel drawing
- Rectangle drawing

User-modified Hardware Dependent Code
- Connectivity of the smart display to the LM4F
- Changes to the existing code to match your display (like color depth and size)
Graphics Primitives

Low level drawing support for:

- Lines, circles, text and bitmap images
- Support for off-screen buffering
- Foreground and background drawing contexts
- Color is represented as a 24-bit RGB value (8-bits per color)
  - ~150 pre-defined colors are provided
- 153 pre-defined fonts based on the Computer Modern typeface
- Support for Asian and Cyrillic languages
- Widgets are graphic elements that provide user control elements
- Widgets combine the graphical and touch screen elements on-screen with a parent/child hierarchy so that objects appear in front or behind each other correctly

Canvas – a simple drawing surface with no user interaction
Checkbox – select/unselect
Container – a visual element to group on-screen widgets
Push Button – an on-screen button that can be pressed to perform an action
Radio Button – selections that form a group; like low, medium and high
Slider – vertical or horizontal to select a value from a predefined range
ListBox – selection from a list of options
Special Utilities

Utilities to produce graphics library compatible data structures

ftrasterize
- Uses the FreeType font rendering package to convert your font into a graphic library format.
- Supported fonts include: TrueType®, OpenType®, PostScript® Type 1 and Windows® FNT.

Imi-button
- Creates custom shaped buttons using a script plug-in for GIMP. Produces images for use by the pushbutton widget.

pnmtoc
- Converts a NetPBM image file into a graphics library compatible file.
- NetPBM image formats can be produced by: GIMP, NetPBM, ImageMajik and others.

mkstringtable
- Converts a comma separated file (.csv) into a table of strings usable by graphics library for pull down menus.
Lab 10: Graphics Library

- Connect Kentec Display
- Experiment with demo project
- Write graphics library code
Agenda

- Introduction to ARM® Cortex™-M4F and Peripherals
- Code Composer Studio
- Introduction to StellarisWare, Initialization and GPIO
- Interrupts and the Timers
  - ADC12
  - Hibernation Module
  - USB
  - Memory
  - Floating-Point
  - BoosterPacks and grLib

**Synchronous Serial Interface**

- UART
- μDMA
LM4F120H5QR SSI Features

Four SSI modules. Each with:

- Freescale SPI, MICROWIRE or TI Synchronous Serial interfaces
- Master or Slave operation
- Programmable bit clock rate and pre-scaler
- Programmable data frame size from 4 to 16-bits
- Separate Tx and Rx FIFOs (8 x16-bits)
- Interrupts and µDMA support
SSI Block Diagram

Signal Pinout (n = 0 to 3) …

**SSInClk**: SSI Module n Clock
**SSInFss**: SSI Module n Frame Signal
**SSInRx**: SSI Module n Receive
**SSInTx**: SSI Module n Transmit

Note that the LM4F120H5QR pins are extensively muxed with other signals. The Pin Mux Utility can ease the programming. See: www.ti.com/stellarispinmuxutility
SSI Interrupts

Single interrupt per module, cleared automatically

Interrupt conditions:

- Transmit FIFO service (when the transmit FIFO is half full or less)
- Receive FIFO service (when the receive FIFO is half full or more)
- Receive FIFO time-out
- Receive FIFO overrun
- End of transmission
- Receive DMA transfer complete
- Transmit DMA transfer complete

Interrupts on these conditions can be enabled individually

Your handler code must check to determine the source of the SSI interrupt and clear the flag(s)
SSI μDMA Operation

- Separate channels for Tx and Rx
- When enabled, the SSI will assert a DMA request on either channel when the Rx or Tx FIFO can transfer data
- **For Rx channel:** A single transfer request is made when any data is in the Rx FIFO. A burst transfer request is made when 4 or more items is in the Rx FIFO.
- **For Tx channel:** A single transfer request is made when there is at least one empty location in the Tx FIFO. A burst transfer request is made when 4 or more slots are empty.
Freescale SPI Signal Formats

- Four wire interface. Full duplex.
- **SSIFss** acts as chip select
- Inactive state and clock phasing are programmable via the **SPO** and **SPH** bits (SSI_FRF_MOTO_MODE_0-3 parameter)
  - SPO = 0: SSIClk low when inactive. SPO = 1: high
  - SPH = 0: Data is captured on 1st SSIClk transition. SPH = 1: 2nd

SPO = 0
SPH = 0
Single Transfer

SPO = 0
SPH = 1
Single Transfer
TI Synchronous Serial Signal Formats

- Three wire interface
- Devices are always slaves
- **SSIClk** and **SSIFss** are forced low and **SSITx** is tri-stated when the SSI is idle

---

**Single Transfer**

![Single Transfer Diagram]

**Continuous Transfer**

![Continuous Transfer Diagram]
Microwire Signal Formats

- Four wire interface
- Similar to SPI, except transmission is half-duplex
- Master – Slave message passing technique

**Single Transfer**

**Continuous Transfer**

Lab...
Lab 11: SPI Bus and the Olimex LED Boosterpack

- Carefully install pin-modified Olimex BoosterPack
- Run faces program (SoftSSI)
- Carefully install proto-board modified Olimex BoosterPack
- Create program to utilize SSI SPI
Agenda

Introduction to ARM® Cortex™-M4F and Peripherals
  Code Composer Studio
  Introduction to StellarisWare, Initialization and GPIO
  Interrupts and the Timers
    ADC12
    Hibernation Module
    USB
    Memory
    Floating-Point
    BoosterPacks and grLib
  Synchronous Serial Interface

UART
μDMA
UART Features

- Separate 16x8 bit transmit and receive FIFOs
- Programmable baud rate generator
- Auto generation and stripping of start, stop, and parity bits
- Line break generation and detection
- Programmable serial interface
  - 5, 6, 7, or 8 data bits
  - even, odd, stick, or no parity bits
  - 1 or 2 stop bits
  - baud rate generation, from DC to processor clock/16
- Modem control/flow control
- IrDA and EIA-495 9-bit protocols
- μDMA support
Basic Operation

- **Initialize the UART**
  - Enable the UART peripheral, e.g.
    ```c
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    ```
  - Set the Rx/Tx pins as UART pins
    ```c
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    ```
  - Configure the UART baud rate, data configuration
    ```c
    ROM_UARTConfigSetExpClk(UART0_BASE, ROM_SysCtlClockGet(), 115200,
                             UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                             UART_CONFIG_PAR_NONE));
    ```
  - Configure other UART features (e.g. interrupts, FIFO)

- **Send/receive a character**
  - Single register used for transmit/receive
  - Blocking/non-blocking functions in driverlib:
    ```c
    UARTCharPut(UART0_BASE, ‘a’);
    newchar = UARTCharGet(UART0_BASE);
    UARTCharPutNonBlocking(UART0_BASE, ‘a’);
    newchar = UARTCharGetNonBlocking(UART0_BASE);
    ```

Interrupts...
UART Interrupts

Single interrupt per module, cleared automatically

Interrupt conditions:

- Overrun error
- Break error
- Parity error
- Framing error
- Receive timeout – when FIFO is not empty and no further data is received over a 32-bit period
- Transmit – generated when no data present (if FIFO enabled, see next slide)
- Receive – generated when character is received (if FIFO enabled, see next slide)

Interrupts on these conditions can be enabled individually

Your handler code must check to determine the source of the UART interrupt and clear the flag(s)
Using the UART FIFOs

- Both FIFOs are accessed via the UART Data register (UARTDR)
- After reset, the FIFOs are enabled*, you can disable by resetting the FEN bit in UARTLCRH, e.g.
  
  ```c
  UARTFIFODisable(UART0_BASE);
  ```
- Trigger points for FIFO interrupts can be set at 1/8, 1/4, 1/2, 3/4, 7/8 full, e.g.
  
  ```c
  UARTFIFOLevelSet(UART0_BASE, UART_FIFO_TX4_8, UART_FIFO_RX4_8);
  ```

* Note: the datasheet says FIFOs are disabled at reset
UART “stdio” Functions

- **StellarisWare “utils” folder contains functions for C stdio console functions:**
  
  ```
  c:\StellarisWare\utils\uartstdio.h
  c:\StellarisWare\utils\uartstdio.c
  ```

- **Usage example:**

  ```
  UARTStdioInit(0); //use UART0, 115200
  UARTprintf("Enter text: ");
  ```

- **See uartstdio.h for other functions**

- **Notes:**
  - Use the provided interrupt handler `UARTStdioIntHandler()` code in `uartstdio.c`
  - Buffering is provided if you define UART_BUFFERED symbol
    - Receive buffer is 128 bytes
    - Transmit buffer is 1024 bytes

Other UART Features...
Other UART Features

- Modem control/flow control
- IrDA serial IR (SIR) encoder/decoder
  - External infrared transceiver required
  - Supports half-duplex serial SIR interface
  - Minimum of 10-ms delay required between transmit/receive, provided by software
- ISA 7816 smartcard support
  - UnTX signal used as a bit clock
  - UnRx signal is half-duplex communication line
  - GPIO pin used for smartcard reset, other signals provided by your system design
- LIN (Local Interconnect Network) support: master or slave
- μDMA support
  - Single or burst transfers support
  - UART interrupt handler handles DMA completion interrupt
- EIA-495 9-bit operation
  - Multi-drop configuration: one master, multiple slaves
  - Provides “address” bit (in place of parity bit)
  - Slaves only respond to their address
Lab 12: UART

- Initialize UART and echo characters using polling
- Use interrupts
- Use stdio utility, e.g. UARTprintf()
 Agenda

Introduction to ARM® Cortex™-M4F and Peripherals
  Code Composer Studio
  Introduction to StellarisWare, Initialization and GPIO
  Interrupts and the Timers
    ADC12
    Hibernation Module
    USB
    Memory
    Floating-Point
    BoosterPacks and grLib
  Synchronous Serial Interface
    UART
    μDMA
µDMA Features

- 32 channels
- SRAM to SRAM, SRAM to peripheral and peripheral to SRAM transfers (no Flash or ROM transfers are possible)
- Basic, Auto (transfer completes even if request is removed), Ping-Pong and Scatter-gather (via a task list)
- Two priority levels
- 8, 16 and 32-bit data transfer sizes
- Transfer sizes of 1 to 1024 elements (in binary steps)
- CPU bus accesses outrank DMA controller
- Source and destination address increment sizes: size of element, half-word, word, no increment
- Interrupt on transfer completion (per channel)
- Hardware and software triggers
- Single and Burst requests
- Each channel can specify a minimum # of transfers before relinquishing to a higher priority transfer. Known as “Burst” or “Arbitration”
Transfer Types

Basic
- Single to Single
- Single to Array
- Array to Single
- Array to Array

Auto
- Same as Basic but the transfer completes even if the request is removed

Ping-Pong
- Single to Array (and vice-versa). Normally used to stream data from a peripheral to memory. When the PING array is full the μDMA switches to the PONG array, freeing the PING array for use by the program.

Scatter-Gather
- Many Singles to an Array (and vice-versa). May be used to read elements from a data stream or move objects in a graphics memory frame.
**µDMA Channels**

- Each channel has 5 possible assignments made in the DMACHMAPn register.

<table>
<thead>
<tr>
<th>Enc.</th>
<th>Ch #</th>
<th>Peripheral Type</th>
<th>Peripheral Type</th>
<th>Type</th>
<th>Peripheral Type</th>
<th>Peripheral Type</th>
<th>Type</th>
<th>Peripheral Type</th>
<th>Peripheral Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>USB0 EP1 RX</td>
<td>SB</td>
<td>UART2 RX</td>
<td>SB</td>
<td>Software</td>
<td>B</td>
<td>GPTimer 4A</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>1</td>
<td>USB0 EP1 TX</td>
<td>B</td>
<td>UART2 TX</td>
<td>SB</td>
<td>Software</td>
<td>B</td>
<td>GPTimer 4B</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>2</td>
<td>USB0 EP2 RX</td>
<td>B</td>
<td>GPTimer 3A</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>3</td>
<td>USB0 EP2 TX</td>
<td>B</td>
<td>GPTimer 3B</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>4</td>
<td>USB0 EP3 RX</td>
<td>B</td>
<td>GPTimer 2A</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>GPIO A</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>5</td>
<td>USB0 EP3 TX</td>
<td>B</td>
<td>GPTimer 2B</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>GPIO B</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>6</td>
<td>Software</td>
<td>B</td>
<td>GPTimer 2A</td>
<td>B</td>
<td>UART5 RX</td>
<td>SB</td>
<td>GPIO C</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>7</td>
<td>Software</td>
<td>B</td>
<td>GPTimer 2B</td>
<td>B</td>
<td>UART5 TX</td>
<td>SB</td>
<td>GPIO D</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>8</td>
<td>UART0 RX</td>
<td>SB</td>
<td>UART1 RX</td>
<td>SB</td>
<td>Software</td>
<td>B</td>
<td>GPTimer 5A</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>9</td>
<td>UART0 TX</td>
<td>SB</td>
<td>UART1 TX</td>
<td>SB</td>
<td>Software</td>
<td>B</td>
<td>GPTimer 5B</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>10</td>
<td>SS10 RX</td>
<td>SB</td>
<td>SS11 RX</td>
<td>SB</td>
<td>UART6 RX</td>
<td>SB</td>
<td>GPTimer 6A</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>11</td>
<td>SS10 TX</td>
<td>SB</td>
<td>SS11 TX</td>
<td>SB</td>
<td>UART6 TX</td>
<td>SB</td>
<td>GPTimer 6B</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>12</td>
<td>Software</td>
<td>B</td>
<td>UART2 RX</td>
<td>SB</td>
<td>SS12 RX</td>
<td>SB</td>
<td>GPTimer 7A</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>13</td>
<td>Software</td>
<td>B</td>
<td>UART2 TX</td>
<td>SB</td>
<td>SS12 TX</td>
<td>SB</td>
<td>GPTimer 7B</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>14</td>
<td>ADC0 SS0</td>
<td>B</td>
<td>GPTimer 2A</td>
<td>B</td>
<td>SS13 RX</td>
<td>SB</td>
<td>GPIO E</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>15</td>
<td>ADC0 SS1</td>
<td>B</td>
<td>GPTimer 2B</td>
<td>B</td>
<td>SS13 TX</td>
<td>SB</td>
<td>GPIO F</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>16</td>
<td>ADC0 SS2</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>UART3 RX</td>
<td>SB</td>
<td>GPTimer 8A</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>17</td>
<td>ADC0 SS3</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>UART3 TX</td>
<td>SB</td>
<td>GPTimer 8B</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>18</td>
<td>GPTimer 0A</td>
<td>B</td>
<td>GPTimer 1A</td>
<td>B</td>
<td>UART4 TX</td>
<td>SB</td>
<td>GPIO B</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>19</td>
<td>GPTimer 0B</td>
<td>B</td>
<td>GPTimer 1B</td>
<td>B</td>
<td>UART4 TX</td>
<td>SB</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>20</td>
<td>GPTimer 1A</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>UART7 RX</td>
<td>SB</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>21</td>
<td>GPTimer 1B</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>UART7 TX</td>
<td>SB</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>22</td>
<td>UART1 RX</td>
<td>SB</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>23</td>
<td>UART1 TX</td>
<td>SB</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>24</td>
<td>SS11 RX</td>
<td>SB</td>
<td>ADC1 SS0</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>GPTimer 9A</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>25</td>
<td>SS11 TX</td>
<td>SB</td>
<td>ADC1 SS1</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>GPTimer 9B</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>26</td>
<td>Software</td>
<td>B</td>
<td>ADC1 SS2</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>GPTimer 10A</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>27</td>
<td>Software</td>
<td>B</td>
<td>ADC1 SS3</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>GPTimer 10B</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>28</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>GPTimer 11A</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>29</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>GPTimer 11B</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>30</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
<td>B</td>
<td>Software</td>
</tr>
<tr>
<td>31</td>
<td>Reserved</td>
<td>B</td>
<td>Reserved</td>
<td>B</td>
<td>Reserved</td>
<td>B</td>
<td>Reserved</td>
<td>B</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

S = Single  
B = Burst  
SB = Both
Channel Configuration

- Channel control is done via a set of control structures in a table
- The table must be located on a 1024-byte boundary
- Each channel can have one or two control structures; a primary and an alternate
- The primary structure is for BASIC and AUTO transfers. Alternate is for Ping-Pong and Scatter-gather

Control Structure Memory Map

<table>
<thead>
<tr>
<th>Offset</th>
<th>Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>0, Primary</td>
</tr>
<tr>
<td>0x10</td>
<td>1, Primary</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x1F0</td>
<td>31, Primary</td>
</tr>
<tr>
<td>0x200</td>
<td>0, Alternate</td>
</tr>
<tr>
<td>0x210</td>
<td>1, Alternate</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x3F0</td>
<td>31, Alternate</td>
</tr>
</tbody>
</table>

Channel Control Structure

<table>
<thead>
<tr>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td>Source End Pointer</td>
</tr>
<tr>
<td>0x004</td>
<td>Destination End Pointer</td>
</tr>
<tr>
<td>0x008</td>
<td>Control Word</td>
</tr>
<tr>
<td>0x00C</td>
<td>Unused</td>
</tr>
</tbody>
</table>

Control word contains:
- Source and Dest data sizes
- Source and Dest addr increment size
- # of transfers before bus arbitration
- Total elements to transfer
- Useburst flag
- Transfer mode
Lab 13: Transferring Data with the µDMA

- Perform an array to array memory transfer
- Transfer data to and from the UART

USB Emulation Connection
Thanks for Attending!

- Make sure to take your kits and workbooks with you
- Please leave the TTO flash drives and meters here
- Please fill out the feedback form
- Have safe trip home!