ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN-ĐIỆN TỬ
BỘ MÔN KỸ THUẬT ĐIỆN TỬ

# Embedded System Design

## Chapter 5: Using Peripherals and Interrupts

5.1 Timers
5.2 Interrupts
5.3 ADC
5.4 UART
5.5 USB

# 1. Timer

- **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
- PWM

# 1. Timer: Configuration

**Timer configuration**

- void **SysCtlPeripheralEnable**(unsigned long ulPeripheral)
  - ulPeripheral: SYSCTL_PERIPH_TIMER0, …, SYSCTL_PERIPH_TIMER5, SYSCTL_PERIPH_WTIMER0, …, SYSCTL_PERIPH_WTIMER5
- void **TimerConfigure**(unsigned long ulBase, unsigned long ulConfig)
  - ulConfig: TIMER_CFG_ONE_SHOT, TIMER_CFG_PERIODIC, TIMER_CFG_RTC, TIMER_CFG_SPLIT_PAIR, TIMER_CFG_32_BIT_PER, …
- Example:
  - SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
  - TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_PER);

# 1. Timer: Delay

- void **TimerLoadSet**(unsigned long ulBase, unsigned long ulTimer, unsigned long ulValue)
  - ulBase is the base address of the timer module.
  - ulTimer specifies the timer(s) to adjust; must be one of TIMER_A, TIMER_B, or TIMER_BOTH. Only TIMER_A should be used when the timer is configured for full-width operation.
  - ulValue is the load value.
- Example: to toggle a GPIO at 10Hz and a 50% duty cycle, we need to delay 0.05s
  - ulPeriod = (SysCtlClockGet() / 10) / 2;
  - TimerLoadSet(TIMER0_BASE, TIMER_A, ulPeriod -1);

# Sample Codes

```
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h“
void Timer0IntHandler(void) {
    // Clear the timer interrupt
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    // Read the current state of the GPIO pin and write back the opposite state
    if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2))  {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
    }
    else {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
    }
}
```

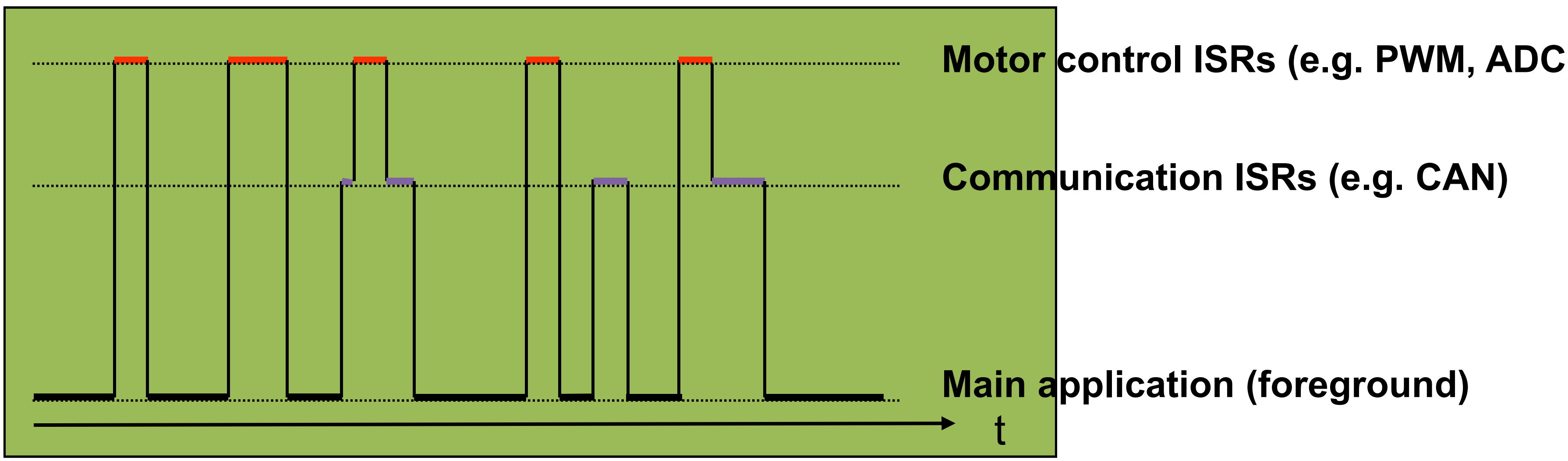
# Sample Codes

```
int main(void) {
unsigned long ulPeriod;
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_
    OSC_MAIN);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,
    GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_PER);
ulPeriod = (SysCtlClockGet() / 10) / 2;
TimerLoadSet(TIMER0_BASE, TIMER_A, ulPeriod -1);
IntEnable(INT_TIMER0A);
TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
IntMasterEnable();
TimerEnable(TIMER0_BASE, TIMER_A);
while(1) { }
}
```

# 2. Interrupt

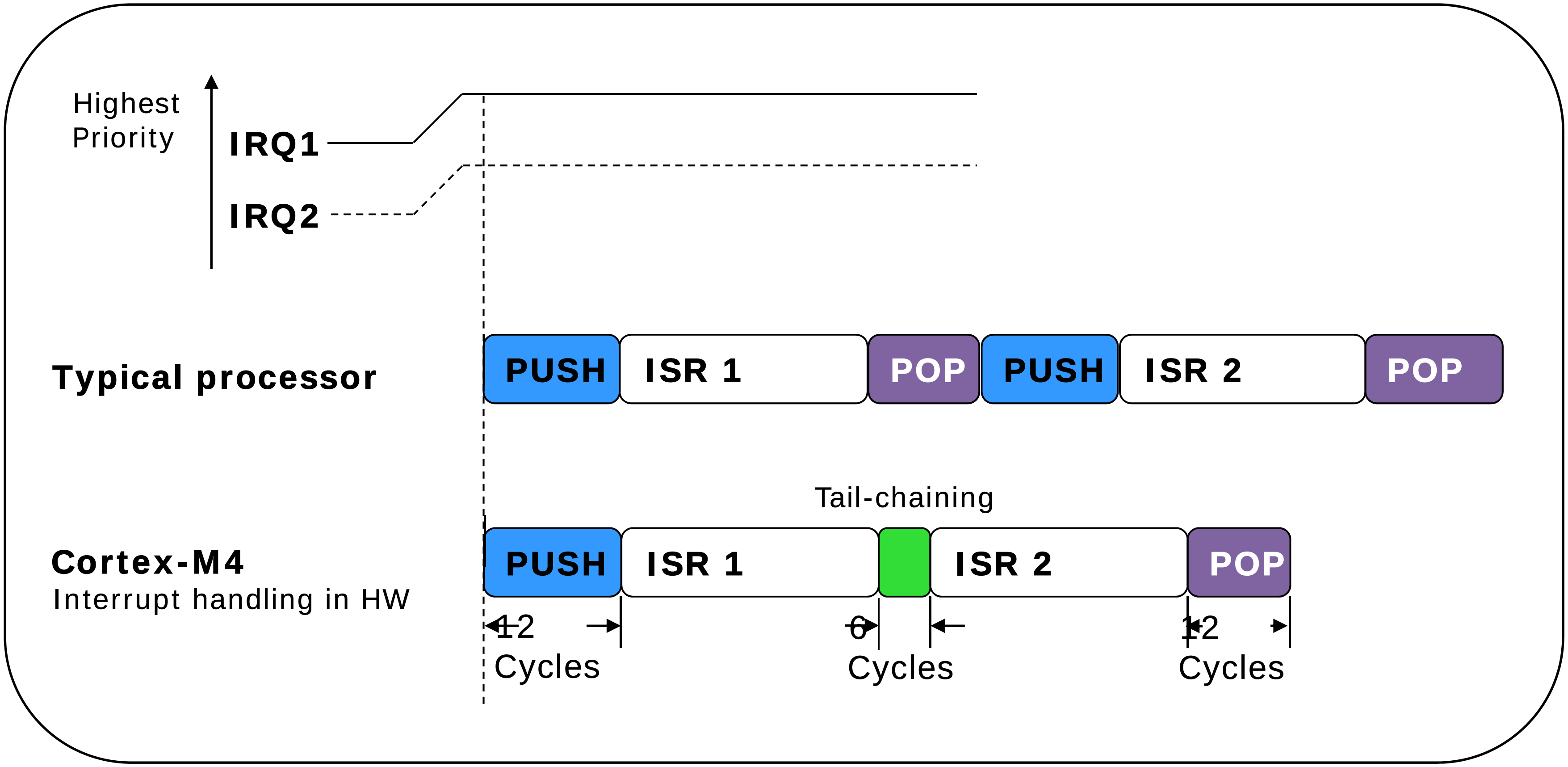
**Nested Vectored Interrupt Controller (NVIC)**

- 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




Tail Chaining...

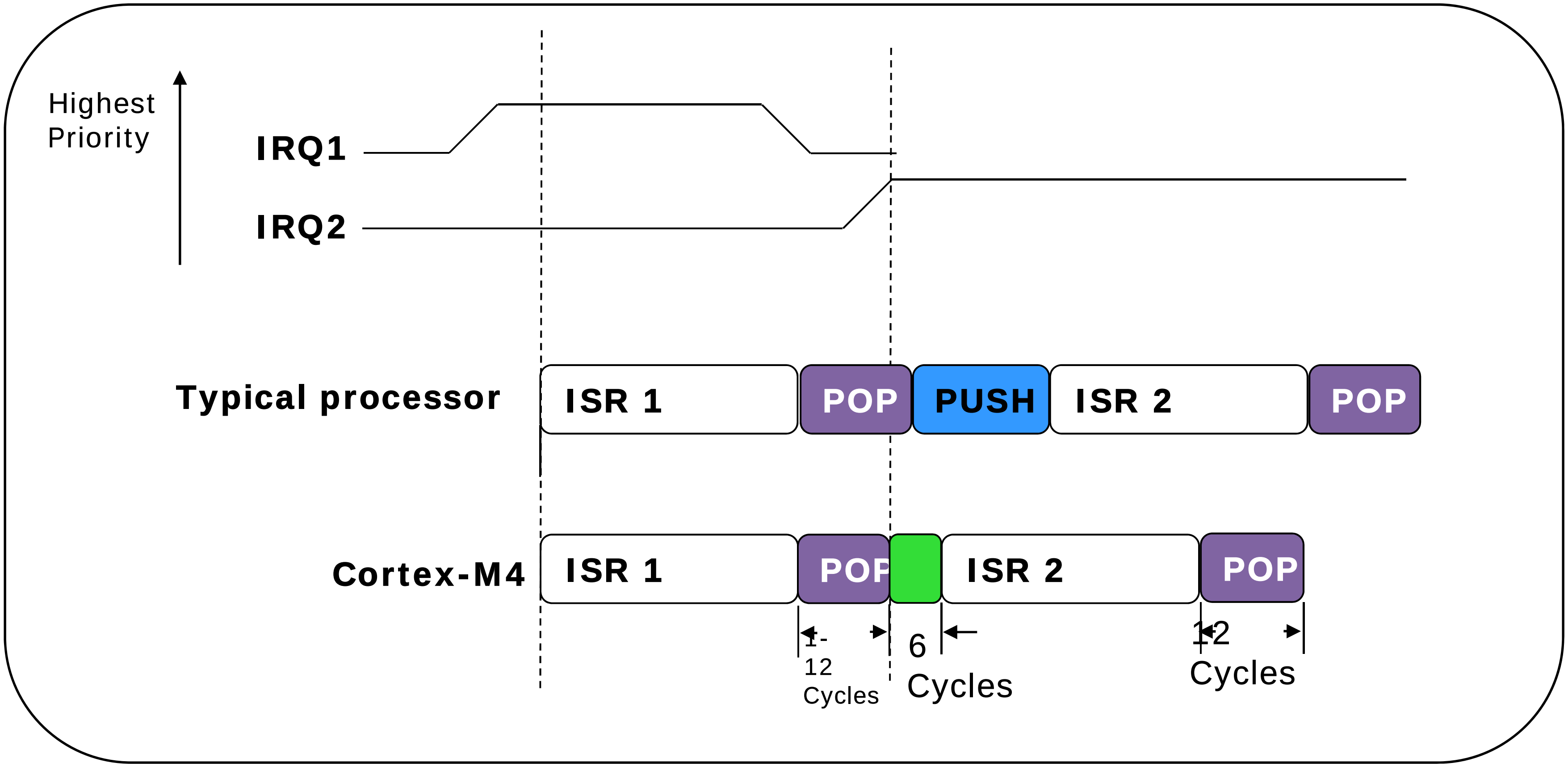
# Interrupt Latency - Tail Chaining





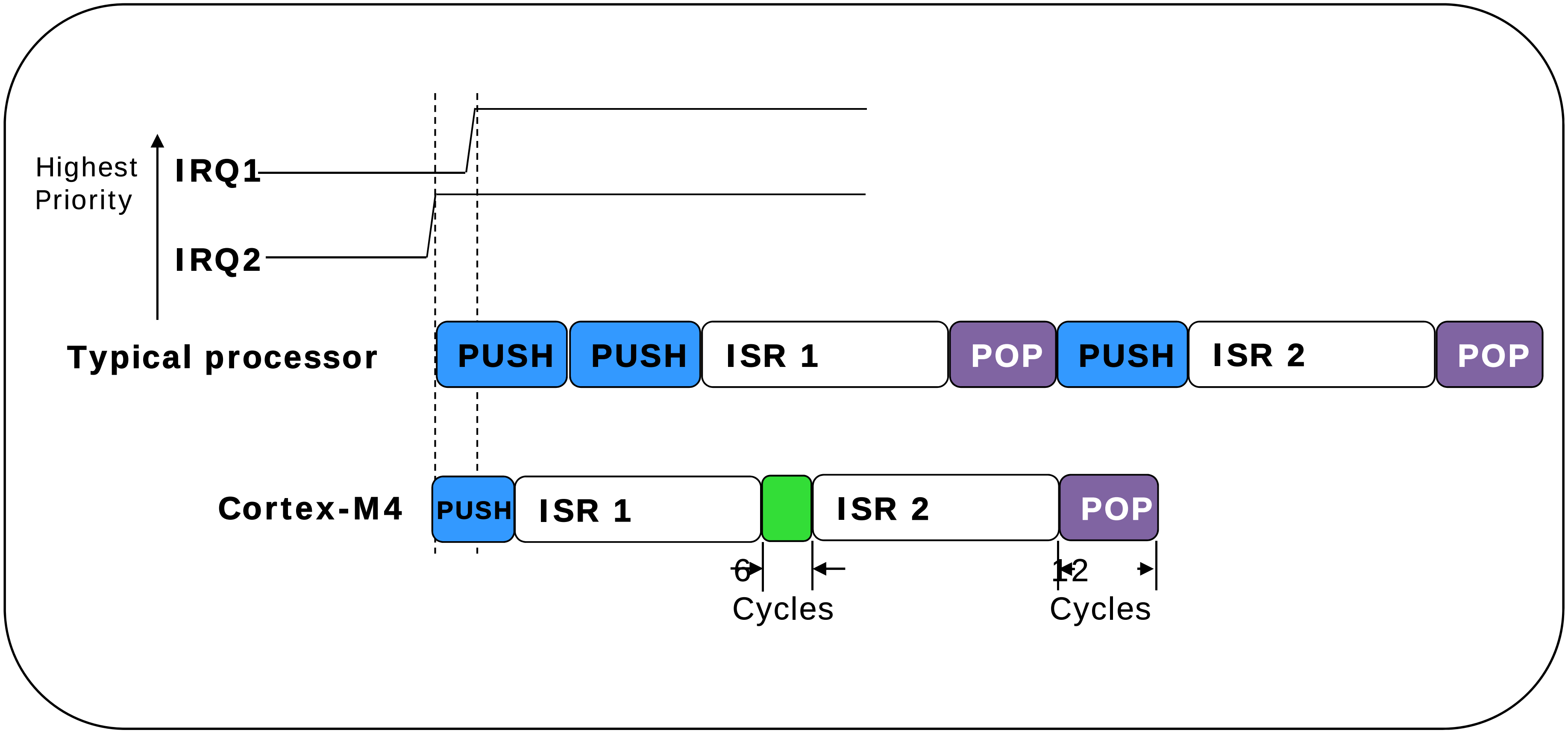

Pre-emption …

# Interrupt Latency – Pre-emption




Late arrival...

# Interrupt Latency – Late Arrival



# Cortex-M4® 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



Exception types...

# Cortex-M4® Exception Types

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

Vector 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

| Exception number | IRQ number | Offset | Vector |
|---|---|---|---|
| 154 | 138 | 0x0268 | IRQ131 |
| ... | | ... | ... |
| 18 | 2 | 0x0048 | IRQ2 |
| 17 | 1 | 0x0044 | IRQ1 |
| 16 | 0 | 0x0040 | IRQ0 |
| 15 | -1 | 0x003C | Systick |
| 14 | -2 | 0x0038 | PendSV |
| 13 | | | Reserved |
| 12 | | | Reserved for Debug |
| 11 | -5 | 0x002C | SVCall |
| 10 | | | Reserved |
| 9 | | | |
| 8 | | | |
| 7 | | | |
| 6 | -10 | 0x0018 | Usage fault |
| 5 | -11 | 0x0014 | Bus fault |
| 4 | -12 | 0x0010 | Memory management fault |
| 3 | -13 | 0x000C | Hard fault |
| 2 | -14 | 0x0008 | NMI |
| 1 | | 0x0004 | Reset |
| | | 0x0000 | Initial SP value |

# Interrupt Configuration

- Interrupt Enable
  - void IntDisable (unsigned long ulInterrupt)
  - void IntEnable(unsigned long ulInterrupt)
  - tBoolean IntMasterDisable (void)
  - tBoolean IntMasterEnable (void)
  - void TimerIntEnable (unsigned long ulBase, unsigned long ulIntFlags)
- Example
  - IntEnable(INT_TIMER0A);
  - TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
  - IntMasterEnable();

# Interrupt Handler

Interrupt handler sample: toggle a LED at GPIO_PIN2

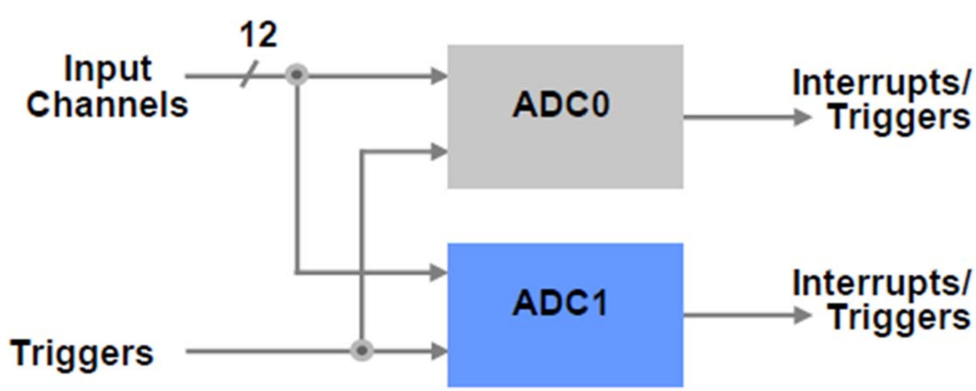
```
void Timer0IntHandler(void)
{
    // Clear the timer interrupt
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    // Read the current state of the GPIO pin and
    // write back the opposite state
    if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2))
    {
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
    }
    else
    {
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
    }
}
```

# Class Assignment

- Write a program to toggle the LED on the LM4F120 kit with the frequency 20Hz using Timer1 and interrupt

# 3. Analog to Digital Converter (ADC)

- Stellaris LM4F MCUs feature two ADC modules (ADC0 and ADC1)
  - 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



# 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.

| Sequencer | Number of Samples | Depth of FIFO |
|---|---|---|
| SS 3 | 1 | 1 |
| SS 2 | 4 | 4 |
| SS 1 | 4 | 4 |
| SS 0 | 8 | 8 |

# 3. ADC - Configuration

- Enable ADC
  - SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
- Set speed
  - void SysCtlADCSpeedSet(unsigned long ulSpeed)
  - ulSpeed:
    - SYSCTL_ADCSPEED_1MSPS
    - SYSCTL_ADCSPEED_500KSPS
    - SYSCTL_ADCSPEED_250KSPS
    - SYSCTL_ADCSPEED_125KSPS
- Example
  - SysCtlADCSpeedSet(SYSCTL_ADCSPEED_250KSPS);

# 3. ADC - Configuration

- Configure ADC sequencer
  - void **ADCSequenceConfigure**(unsigned long ulBase, unsigned long ulSequenceNum, unsigned long ulTrigger, unsigned long ulPriority)
    - ulBase is the base address of the ADC module.
    - ulSequenceNum is the sample sequence number.
    - ulTrigger is the trigger source that initiates the sample sequence; must be one of the ADC_TRIGGER_ values.
    - ulPriority is the relative priority of the sample sequence with respect to the other sample sequences.
- Example
  - ADCSequenceDisable(ADC0_BASE, 1); //disable before configuring
  - ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);

# 3. ADC - Configuration

- Configure a step of the sample sequencer
  - void **ADCSequenceStepConfigure**(unsigned long ulBase, unsigned long ulSequenceNum, unsigned long ulStep, unsigned long ulConfig)
  - ulBase is the base address of the ADC module.
  - ulSequenceNum is the sample sequence number.
  - ulStep is the step to be configured.
  - ulConfig is the configuration of this step; must be a logical OR of ADC_CTL_TS, ADC_CTL_IE, ADC_CTL_END, ADC_CTL_D, one of the input channel selects (ADC_CTL_CH0 through ADC_CTL_CH23), and one of the digital comparator selects (ADC_CTL_CMP0 through ADC_CTL_CMP7).
- Example: steps 0-2 sample the temperature sensor, step 3 configure the interrupt and end the conversion
  - ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS); //CTL_TS: Temperature Sensor
  - ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);//CTL_IE: interrupt enable
  - ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);//CTL_END: end conversion
  - ADCSequenceStepConfigure(ADC0_BASE, 1, 3, ADC_CTL_TS | ADC_CTL_IE | ADC_CTL_END);

# 3. ADC: Sample code

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h"
#ifdef DEBUG
void__error__(char *pcFilename, unsigned long ulLine)
{
}
#endif
int main(void)
{
    unsigned long ulADC0Value[4];
    volatile unsigned long ulTempAvg;
    volatile unsigned long ulTempValueC;
    volatile unsigned long ulTempValueF;
```

# 3. ADC: Sample code

```
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTA
L_16MHZ);
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
SysCtlADCSpeedSet(SYSCTL_ADCSPEED_250KSPS);
ADCSequenceDisable(ADC0_BASE, 1);
ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
ADCSequenceStepConfigure(ADC0_BASE, 1, 3, ADC_CTL_TS | ADC_CTL_IE
|ADC_CTL_END);
ADCSequenceEnable(ADC0_BASE, 1);
```

# 3. ADC: Sample code

```
while(1)
{
    ADCIntClear(ADC0_BASE, 1);
    ADCProcessorTrigger(ADC0_BASE, 1);
    while(!ADCIntStatus(ADC0_BASE, 1, false))
    {
    }
    ADCSequenceDataGet(ADC0_BASE, 1, ulADC0Value);
    ulTempAvg = (ulADC0Value[0] + ulADC0Value[1] + ulADC0Value[2] + ulADC0Value[3]
    + 2)/4;
    ulTempValueC = (1475 - ((2475 * ulTempAvg)) / 4096)/10;
    ulTempValueF = ((ulTempValueC * 9) + 160) / 5;
    }
}
```

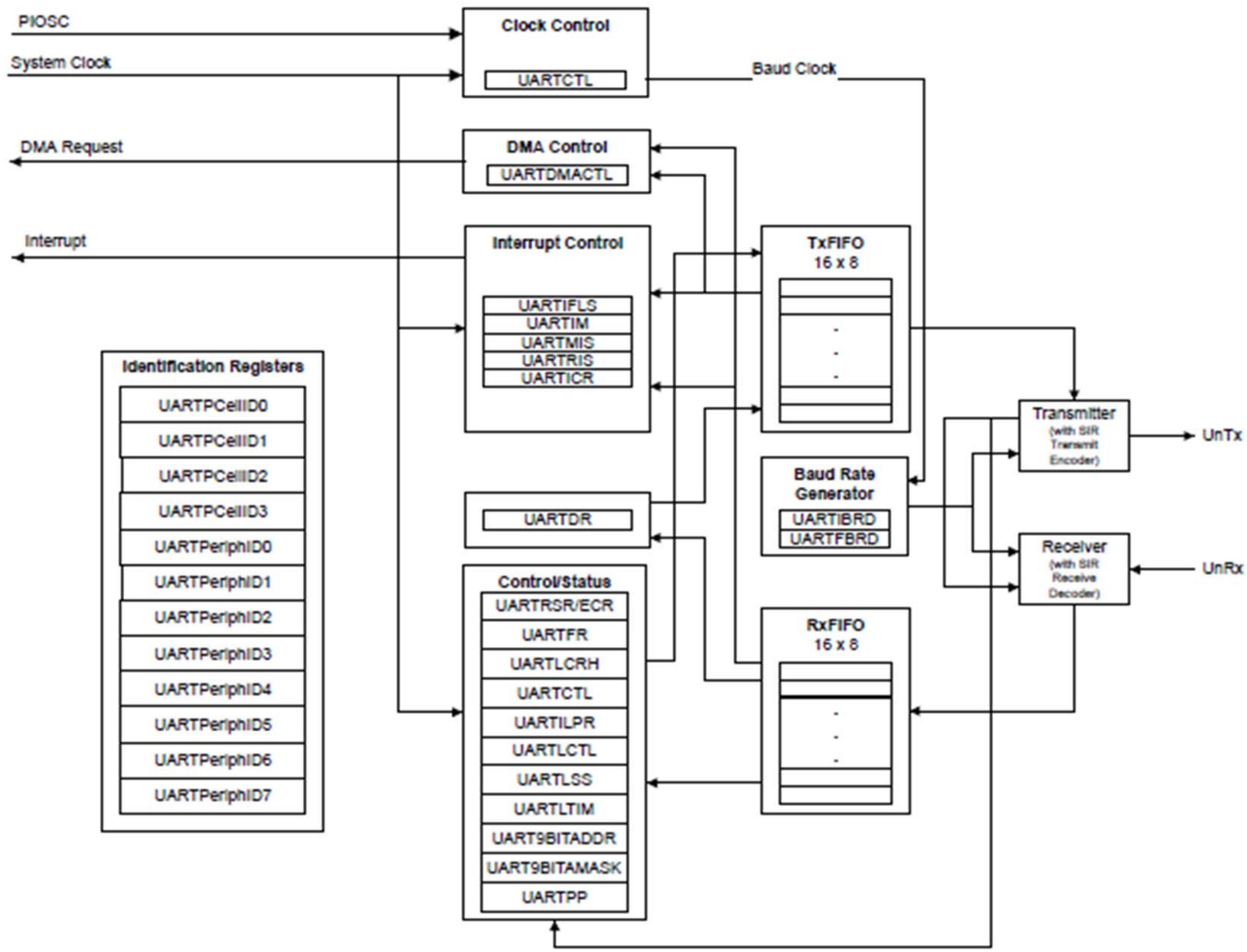
# 4. UART: Features

- 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



Block Diagram...

# 4. UART: Block Diagram



# 4. UART: Basic Operation

- Initialize the UART
  - Enable the UART peripheral, e.g.

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
```

  - Set the Rx/Tx pins as UART pins

```
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

```
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:

```
UARTCharPut(UART0_BASE, ‘a’);
newchar = UARTCharGet(UART0_BASE);
UARTCharPutNonBlocking(UART0_BASE, ‘a’);
newchar = UARTCharGetNonBlocking(UART0_BASE);
```

# 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)

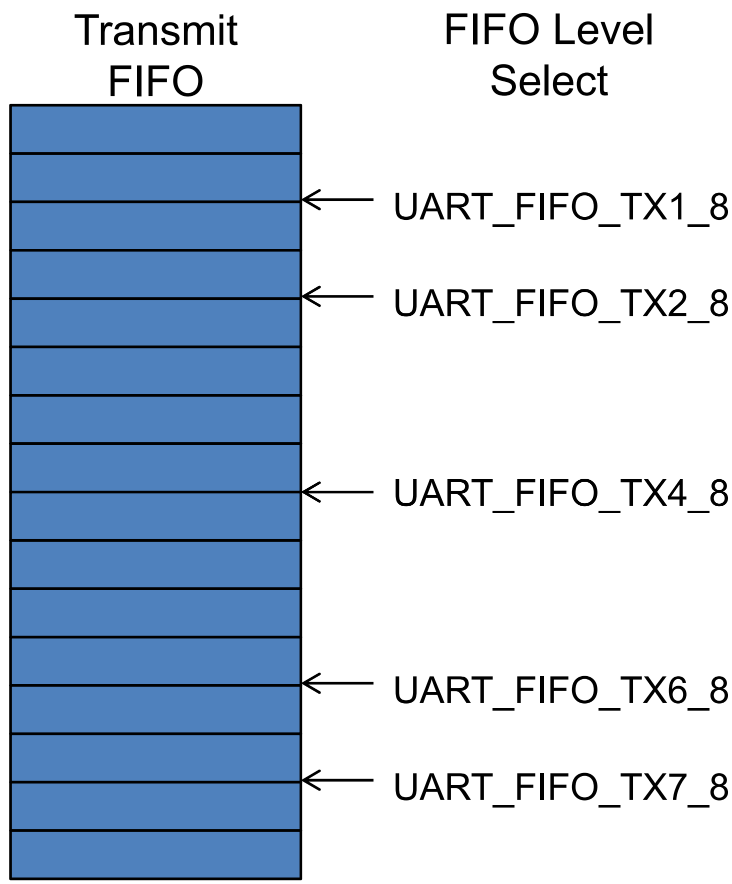
Interrupts on these conditions can be enabled individually

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



FIFOs...

# 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.

```
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.

```
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

- 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

# Example Code

```
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
int main(void) {
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
    SYSCTL_XTAL_16MHZ);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
    (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
    UART_CONFIG_PAR_NONE));
```

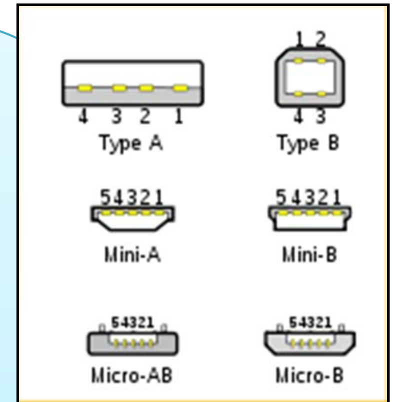
# Example code

```
    UARTCharPut(UART0_BASE, 'E');
    UARTCharPut(UART0_BASE, 'n');
    UARTCharPut(UART0_BASE, 't');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'r');
    UARTCharPut(UART0_BASE, ' ');
    UARTCharPut(UART0_BASE, 'T');
    UARTCharPut(UART0_BASE, 'e');
    UARTCharPut(UART0_BASE, 'x');
    UARTCharPut(UART0_BASE, 't');
    UARTCharPut(UART0_BASE, ':');
    UARTCharPut(UART0_BASE, ' ');
    while (1)
    {
    if (UARTCharsAvail(UART0_BASE)) UARTCharPut(UART0_BASE, UARTCharGet(UART0_BASE));
    }
}
```

# 5. USB Basics

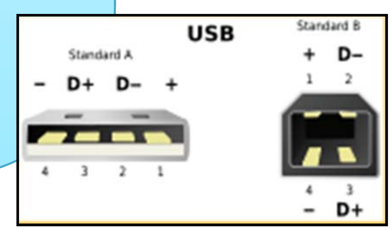
Multiple connector sizes
4 pins – power, ground and 2 data lines
(5th pin ID for USB 2.0 connectors)
Configuration connects power 1st, then data
Standards:

- 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





Different types of USB connectors from left to right
- 8-pin AGOX
- Mini-B plug
- Type B plug
- Type A receptacle
- Type A plug



# 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

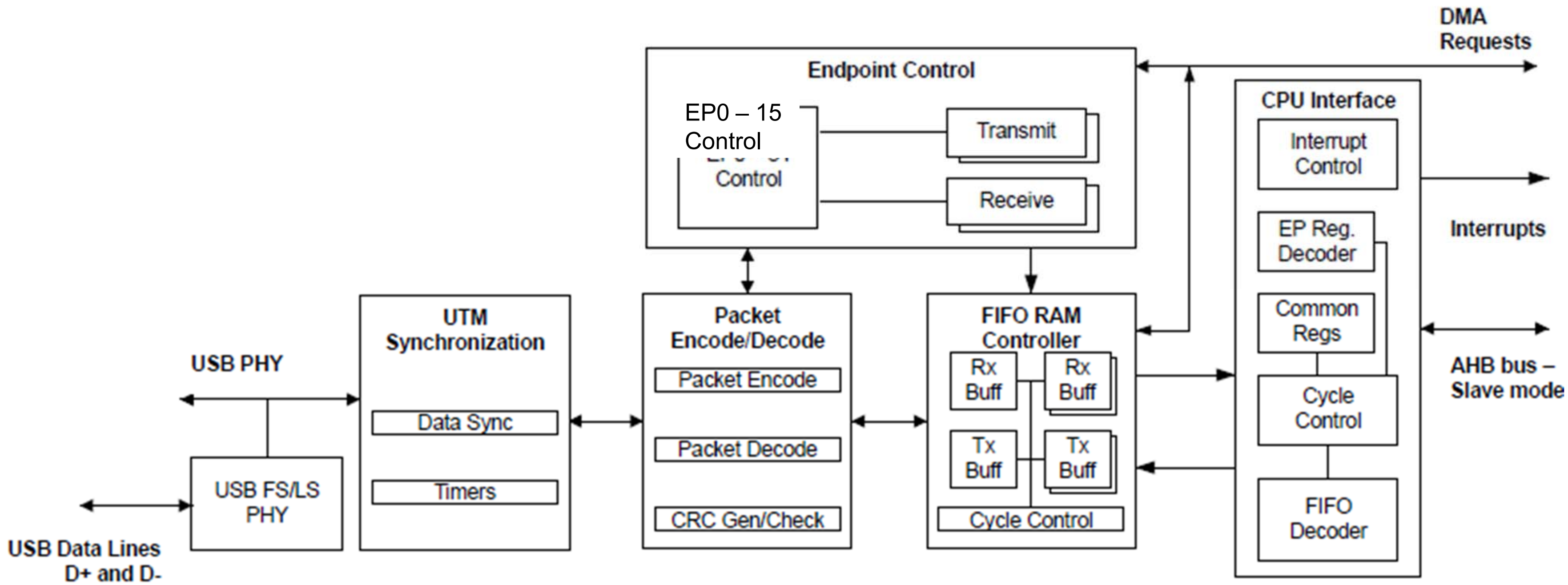
- Texas Instruments is a member of the USB Implementers Forum.
- Stellaris is approved to use the USB logo
- Vendor/Product ID sharing

  http://www.ti.com/lit/pdf/spml001

FREE
Vendor ID/ Product ID
sharing program

VID
Request
for embedded
USB products

# USB Peripheral Block Diagram




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: Example

- Lab 7:
  - c:\StellarisWare\boards\ek-lm4f120xl\usb_dev_bulk\
- Content:
  - This example provides a generic USB device offering simple bulk data transfer to and from the host
  - Data received from the host is assumed to be ASCII text and it is echoed back with the case of all alphabetic characters swapped.

# USB: Example

- Program flow

Declare libraries variables, constants

↓

void SysTickIntHandler(void)

↓

static unsigned long EchoNewDataToHost(...)

↓

unsigned long TxHandler(...)

↓

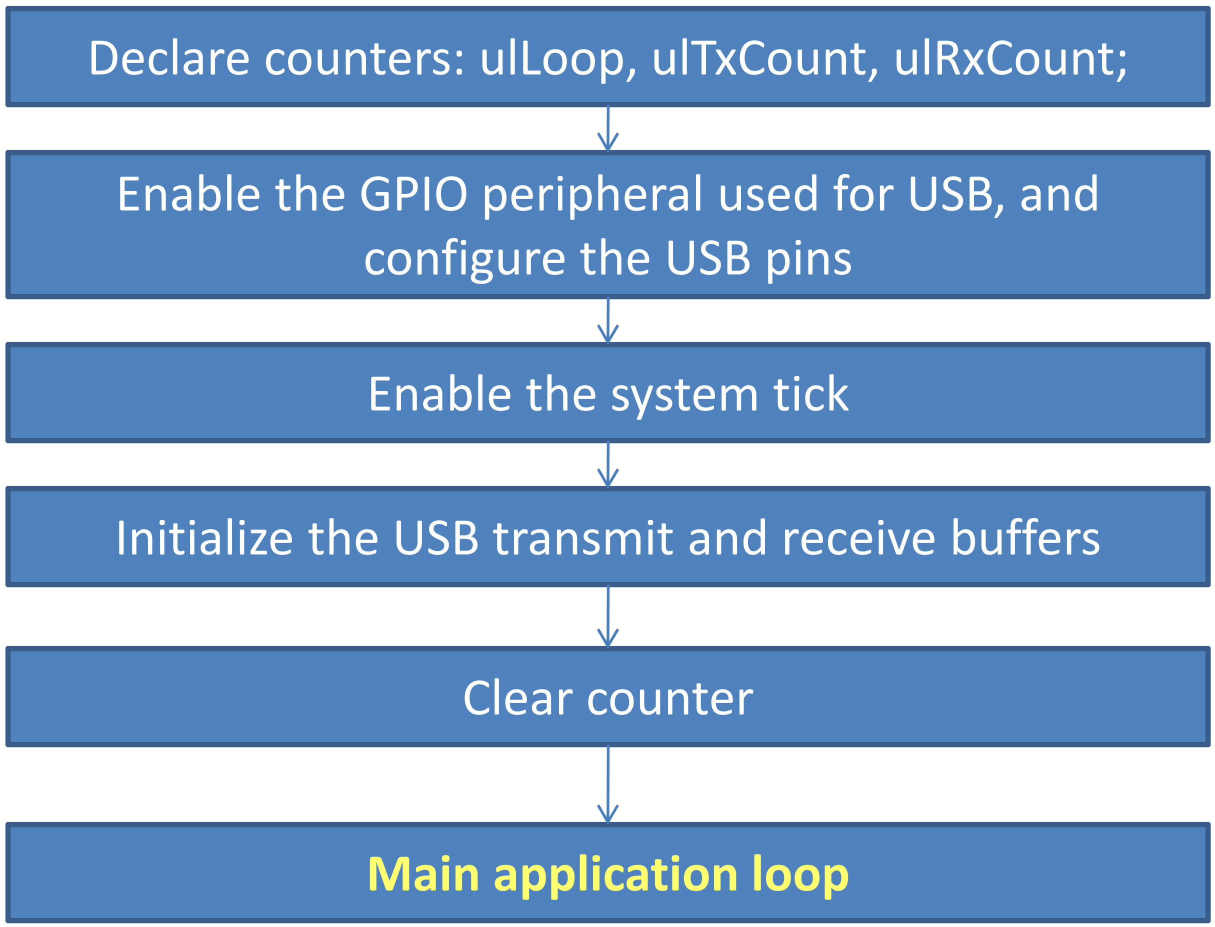
unsigned long TxHandler(...)

↓

**int main(void)**

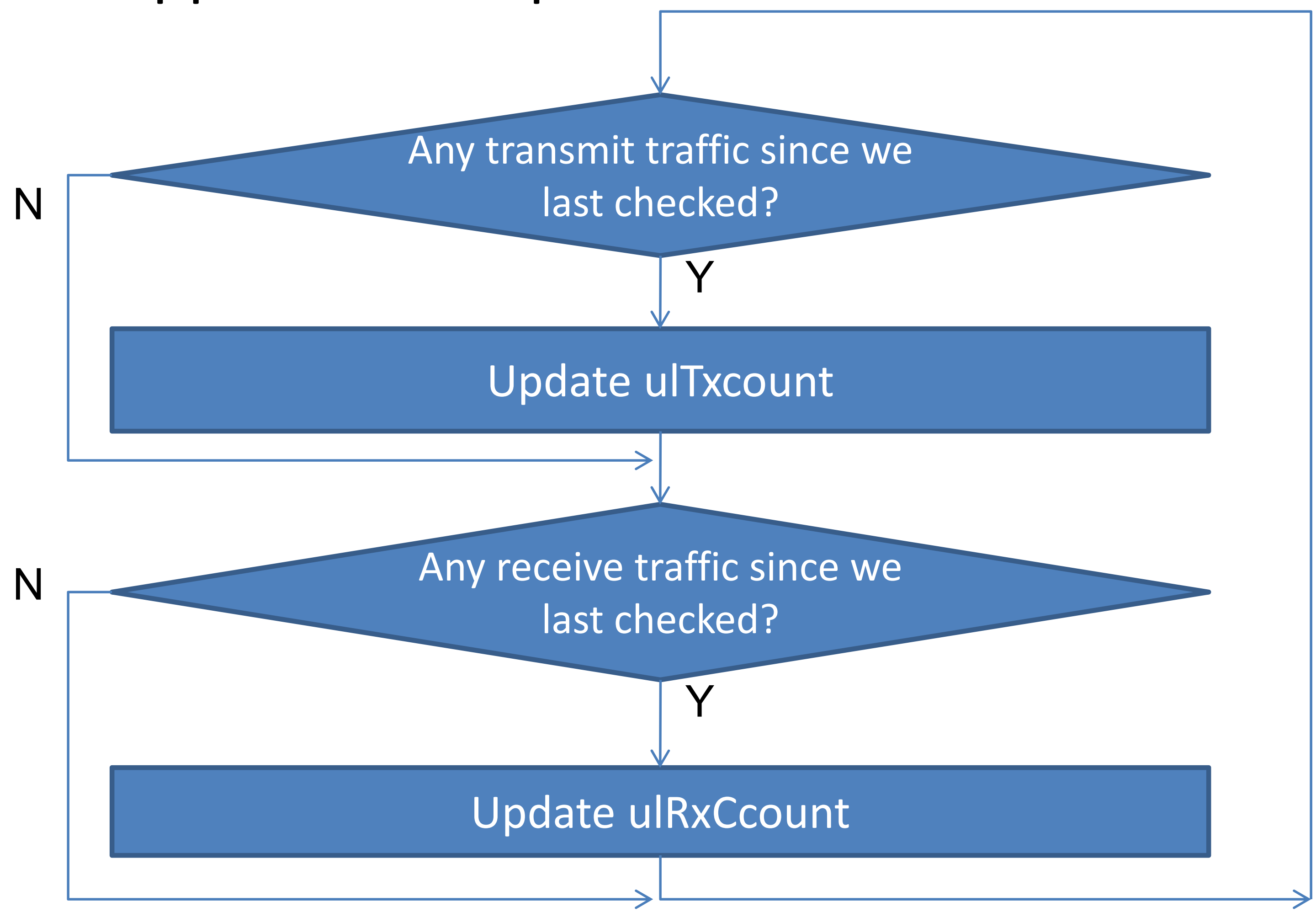
# USB: Example

- Main flow



# USB: Example

- Main application loop



# USB: Example – C code

```
int main(void)
{  volatile unsigned long ulLoop;
   unsigned long ulTxCount;
   unsigned long ulRxCount;
   // Enable lazy stacking for interrupt handlers.
   ROM_FPULazyStackingEnable();
   // Set the clocking to run from the PLL at 50MHz
   ROM_SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
                 SYSCTL_XTAL_16MHZ);
   // Enable the GPIO peripheral used for USB, and configure the USB pins.
   ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
   ROM_GPIOPinTypeUSBAnalog(GPIO_PORTD_BASE, GPIO_PIN_4 | GPIO_PIN_5);
   // Enable the system tick.
   ROM_SysTickPeriodSet(ROM_SysCtlClockGet() / SYSTICKS_PER_SECOND);
   ROM_SysTickIntEnable();
   ROM_SysTickEnable();
```

# USB: Example – C code

```
// Initialize the transmit and receive buffers.
USBBufferInit((tUSBBuffer *)&g_sTxBuffer);
USBBufferInit((tUSBBuffer *)&g_sRxBuffer);

// Set the USB stack mode to Device mode with VBUS monitoring.
 USBStackModeSet(0, USB_MODE_FORCE_DEVICE, 0);
 // Pass our device information to the USB library and place the device
// on the bus.
 USBDBulkInit(0, (tUSBDBulkDevice *)&g_sBulkDevice);

// Clear our local byte counters.
ulRxCount = 0;
ulTxCount = 0;
```

# USB: Example – C Code

```
// Main application loop.
while(1)
  {  // See if any data has been transferred.
     if((ulTxCount != g_ulTxCount) || (ulRxCount != g_ulRxCount))
     {
        // Has there been any transmit traffic since we last checked?
        if(ulTxCount != g_ulTxCount)
        {   // Turn on the Green LED.
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GPIO_PIN_3);
            // Delay for a bit.
            for(ulLoop = 0; ulLoop < 150000; ulLoop++) { }
           // Turn off the Green LED.
           GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);

            ulTxCount = g_ulTxCount;
        }
```

# USB: Example – C Code

```
        // Has there been any receive traffic since we last checked?
        if(ulRxCount != g_ulRxCount)
        {   // Turn on the Blue LED.
             GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2);
             // Delay for a bit.
            for(ulLoop = 0; ulLoop < 150000; ulLoop++) { }
            // Turn off the Blue LED.
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);
            // Take a snapshot of the latest receive count.
            ulRxCount = g_ulRxCount;
        }
      }
   }
}
```

# USB: Example

```
// Receive new data and echo it back to the host.
static unsigned long
EchoNewDataToHost(tUSBDBulkDevice *psDevice, unsigned char *pcData,
                unsigned long ulNumBytes)
{
   unsigned long ulLoop, ulSpace, ulCount;
   unsigned long ulReadIndex;
   unsigned long ulWriteIndex;
   tUSBRingBufObject sTxRing;
   // Get the current buffer information
   USBBufferInfoGet(&g_sTxBuffer, &sTxRing);
   // How much space is there in the transmit buffer?
   ulSpace = USBBufferSpaceAvailable(&g_sTxBuffer);
   // How many characters can we process this time round?
   ulLoop = (ulSpace < ulNumBytes) ? ulSpace : ulNumBytes;
   ulCount = ulLoop;
```

# USB: Example

```
// Update our receive counter.
  g_ulRxCount += ulNumBytes;
  // Dump a debug message.
  DEBUG_PRINT("Received %d bytes\n", ulNumBytes);
  // Set up to process the characters by directly accessing the USB buffers.
  ulReadIndex = (unsigned long)(pcData - g_pucUSBRxBuffer);
  ulWriteIndex = sTxRing.ulWriteIndex;
while(ulLoop)
  {
     // Copy from the receive buffer to the transmit buffer converting
     // character case on the way.
     if((g_pucUSBRxBuffer[ulReadIndex] >= 'a') &&
       (g_pucUSBRxBuffer[ulReadIndex] <= 'z'))
     {  // Convert to upper case and write to the transmit buffer.
        g_pucUSBTxBuffer[ulWriteIndex] =  (g_pucUSBRxBuffer[ulReadIndex] - 'a') + 'A';
     }
```

# USB: Example

```
else
    { if((g_pucUSBRxBuffer[ulReadIndex] >= 'A') && (g_pucUSBRxBuffer[ulReadIndex]
   <= 'Z'))
       {   // Convert to lower case and write to the transmit buffer.
           g_pucUSBTxBuffer[ulWriteIndex] = (g_pucUSBRxBuffer[ulReadIndex] - 'Z') + 'z';
       }
       else {     // Copy the received character to the transmit buffer.
            g_pucUSBTxBuffer[ulWriteIndex] = g_pucUSBRxBuffer[ulReadIndex];
       }
    }
    // Move to the next character taking care to adjust the pointer for the buffer wrap
    ulWriteIndex++;
    ulWriteIndex = (ulWriteIndex == BULK_BUFFER_SIZE) ? 0 : ulWriteIndex;
    ulReadIndex++;
    ulReadIndex = (ulReadIndex == BULK_BUFFER_SIZE) ? 0 : ulReadIndex;
    ulLoop--;
  }
```

# USB: Example

```
    USBBufferDataWritten(&g_sTxBuffer, ulCount);
    DEBUG_PRINT("Wrote %d bytes\n", ulCount);

   //
   // We processed as much data as we can directly from the receive buffer so
   // we need to return the number of bytes to allow the lower layer to
   // update its read pointer appropriately.
   //
   return(ulCount);
}
```