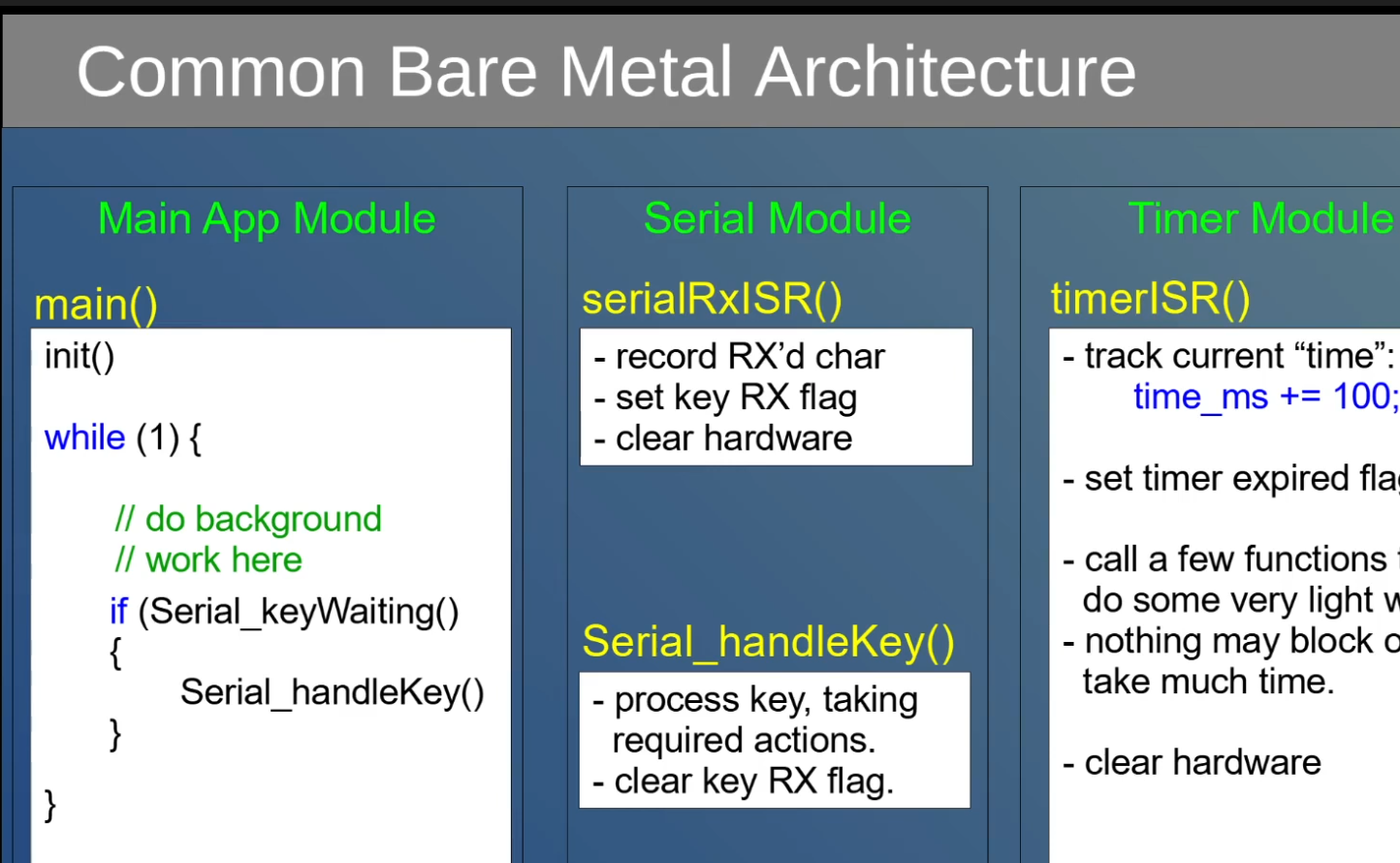
Bare Metal GPIO

* StarterWare user guide for library descriptions
* AM335x Technical Reference Manual for CPU subsystems and registers
* GPIO0 – GPIO3 modules
  + Each supports 32 i/o pins (32 bit CPU)
  + Pins are physical connections to the PCU
    - each pin can be an input, output, or generate an interrupt
    - most pins are multiplexed with many functions
  + Each GPIO module has memory-mapped registers
    - Changing the values in these control registers changes the voltages on the pins
  + 2 ways to drive pins
    - Standard register access: write 1 for on, 0 for off using bit-twiddling
    - Set and clear registers: writ 1 to SET to turn on, write 1 to clear register to turn off, setting 0 has no effect
  + GPIO process

1. Initialize GPIO module
   1. Enable clocks on GPIO modules, PIN muxing, enable GPIO modules, module rest
2. Set pin direction
   1. GPIO\_OE: output enable register
   2. Set bit for 1 for input, 0 for output
3. Read:
   1. GPIO\_DATAIN: read values from pins (unsigned int val = GPIO\_DATAIN)
4. Write:
   1. Drive pins with bit twiddling
   2. Write value to ports GPIO\_DATAOUT register
   * Each GPIO module (GPIO0, GPIO1, etc.) has its own set of registers
     + Each GPIO module has a base address
     + Each register has an offset
     + To access a specific register, it is GPIO Base address + register offset
   * HWREG(x) is a macro that treats x as a (\*volatile unsigned int \*) (x)
     + Value will continuously change so it needs to volatile
     + Dereference the pointer to the register
     + Used as HWREG(SOC\_GPIO\_1\_REGS + GPIO\_DATAOUT) |= (1 << LED1\_PIN);
   * C Guarantees that sizeof char <= short <= int <= long <= long long
     + Stdint types solve this issue of ambiguity
     + Uint32\_t, int8\_t, for example
     + These guarantee to be at least the size indicated
   * Debouncing a Read
     + Values of buttons often bounce between 0s and 1s before settling to the desired value
     + Debounce to read the stable value, not the spurious edges
     + Require the same value for X reads every N time before accepting it as a debounced value
   * Debouncing possible to be done by hardware
     + Set pin to input
     + Set debounce time in GPIO\_DEBOUNCINGTIME
       - Number of 32khz clock cycles to debounce for
       - Debouncing time = (DEBOUNCETIME + 1) \* 31 us
       - Set bit in GPIO\_DEBOUNCENABLE to turn on

Bare Metal Design

* ISR
  + IRQ: Interrupt Request
  + Hardware generates IRQs for certain events
    - Timers
    - Serial data arrived
    - Board losing power
  + Developers write ISR to handle a given interrupt
    - Hardware executes the correct ISR for an interrupt
    - Main thread suspended while in ISR
    - Foreground Task: ISR and functions it calls
    - Background Task: main() and its work
  + Minimize time in ISR
    - Do minimal work to service an IRQ
    - Postpone “real” work for background thread
      * Set flag in ISR; process flag in main()
      * Don’t to blocking IO (UART) in ISR, ex, printf
  + ISR often runts with interrupts disabled, so time in ISR adds latency to servicing other interrupts, like timers
    - Worst case latency can critically impede realtime performance
      * Sampling audio channel regularly
* ISR (Interrupt Service Routines) Programming Issues
  + Global variables written in ISR should be volatile
    - Prevents compiler optimization, value may change unexpectedly
    - Watch for race cases, its effectively multithreaded
  + Avoid using much stack in ISR
    - ISRs stack usage can be on top of normal usage
    - In multi-threaded RTOS (Real time operating system), ISR adds to all stacks, this can lead to stack overflow
    - ISR is added to each stack, need to have room (500 bytes) extra to account for worst case ISR
  + Complications
    - Can have nested interrupts
    - Can prioritize interrupts
* Common Interrupts
  + Timer
  + UART Rx
    - Demo prints the that was pressed by the keyboard
  + GPIO changes
  + Network packet Rx
  + Analog-to-digital data sample ready
* Modular Approach
  + .h files: expose an interface
  + .c files: implements functionality
    - Examples in makefiles automatically compile all .c files for you
  + Names
    - Control buzzer example:
    - Give function names such as Buzzer\_init()
      * Buzzer\_on(int durationInSec);
    - All global variables and functions not shared in .h are static (internal linkage)
  + 
* Common Pattern for ISR
  + ISR services interrupt and sets a flag and records data
  + Main() repeatedly checks if flag is set, then
    - Clears flag
    - Calls function to do some work, which is acceptable if it takes a long time to complete because it is on the background task
* Modularize
  + Encapsulate flag and functions to set/check flag into a module
  + Advanced: instead of serial module processing input commands: give serial module a function pointer to callback app’s processing method to do custom work for user inputs (decouples serial module)

Timers

* Hardware Resources
  + Processor has 8 configurable 32bit hardware timers
* Can generate an interrupt when timer overflows
  + Counts up to 0xFFFF FFFF +1
  + Generates interrupts
  + Loads TLDR on reset
    - Hardware resets counter to value in reload register TLDR
  + Change TLDRs value to control timer period
    - TLDR is an offset from 0 that determines the starting point of the timer
    - Larger number = less distance to overflow = shorter duration
    - ISR triggers at overflow
    - Timer counts up at 25mhz (fast) or 32.786khz (slow)
* Clock Divider
  + Clock has 2 possible sources: 32.768khz, 25Mhz
  + Can prescale (divide down) the clock
  + Prescale of N gives a 2^N slowdown
  + Effectively uses bit N of an extra counter to drive the clock into the timer
* Computing Timeout
  + 
  + Frequency(Hz) = 1 / period (in seconds)
  + N = 0 is no divider (2^0 = 1)
  + Period is in seconds
  + Watch for unit of clock frequency (hz vs khz)
* Watchdog Timer
  + 32 bit counter triggers, resets on overflow
  + Runs on 32.768khz clock
  + Hitting watchdog resets it to WDT\_WLDR’s value
  + Hit by writing a changing a value to WDT\_WTGR (WatchdogTimerTriggerSet)
    - Needs a new value every time the watchdog is hit
* Reset Source
  + Processor knows why it last rebooted
    - Cold
    - Watchdog
    - External reset (reset button)
    - Software reset (instruction)
  + Reset Register (PRM\_RSTST)
    - Indicates reset source
    - Must clear bits (write all 1s) to have next reboot show only the correct source
      * If not cleared, the register will add up and show multiple reset sources
* Systemd
  + Run a program at boot using system
  + Used by most Linux distros as first user-space application to be run by the kernel
    - System daemon is the process running the background
    - Use system to run programs at boot
  + Replaces old “init” system
    - Manages dependencies and allows concurrency when starting up applications
    - Does many things: login, networking, mounting, etc.
  + Create a systemd service
    - Need to setup a .service file
    - Working directory, exectstart and user are the parameters
    - Then enable it using systemctl