CMPT 433: Embedded Systems

A Course Overview

Jeffrey Leung
Simon Fraser University

Fall 2019

Contents

1 Introduction

- **Embedded system:** Computer system with one dedicated function
  - 98% of CPUs in 2008 manufactured by embedded systems
  - **Host machine:** Machine which is connected to and compiles code for the embedded system
  - **Target machine:** The embedded system
  - **Cross-compilation:** Development on a host device which produces code downloaded to a target hardware

2 Software

- Modular design:
  - Separate functionality into components (e.g. interface in header files and implementation on source files)
  - **External linkage:** Declaration of a member (function or variable) which is accessible by code in any other source file
    - Accessed from another source file using the keyword *extern*
    - Avoid externally linked variables by providing access through functions
  - **Internal linkage:** Declaration of a member (function or variable) which is only accessible by code in the same source file
    - In C, use the keyword `static`
  - Public files must have good comments or be easy to understand
- C:
  - Creates loose assumptions if a function does not have a prototype
  - Include guard: Structure which ensures a header file is only included once per program
  - Declare a pointer as NULL after freeing the memory, to avoid a dangling pointer
  - String: Null-terminated (\0) array of characters
  - Volatile: Keyword which specifies that a variable may change without the knowledge of the code, and therefore should not be optimized out
  - Differences between C90/C99:
    - Inline functions
    - Variable declarations throughout blocks
    - Single-line comments (//)
- HTML:
  - `div` is a section/paragraph, `span` is a block of text within a paragraph
- JavaScript and jQuery:
  - `\$("\#id").html("<p>text</p>");` to modify the HTML content
  - `{ \$("\#id").css(" {'color':'red'} "); }` to modify the CSS styling
  - `var input = \$("\#id").value();` to retrieve input field content
  - `\$("\#id").hide();` to hide content

3 Synchronization

- **Process:** Program which has a dedicated address space
  - **Virtual access space:** Component of a process which is an abstracted memory location
  - **fork():** Command which duplicates the address space of the current process
  - **exec():** Set of commands which replaces the current process with an executable file
- **Thread:** Program which shares an address space with other instances of the same program
  - Threads of the same process share a virtual access space, including global variables
- **Mutex:** Data structure which prevents simultaneous access to volatile data which may be changed
  - **Critical section:** Section of code which can only be running once at a time
    - Keep critical sections as minimal as possible
- **Pipe:** Unidirectional data stream for inter-thread/process communication

4 Debugging

- `readelf`: Linux utility which displays details of an executable in an Executable and Linkable Format
- gdb:
  - Generates debug symbols which can stripped later for a smaller, quicker executable
  - Informational commands:
    - `list`: Show code
    - `bt`: Show backtrace of current code
    - `info x`: Display information of a breakpoint, frame, etc.
    - `print x`: Prints the value of variable x
  - Control commands:
    - `target remote`: Connect a machine to debug an executable on
    - `break n` or `b n`: Sets a breakpoint at line or function n
    - `next`: Run until the next breakpoint occurs
    - up/down: Moves to the previous/next stack frame
  - **Core file:** File dumped by gdb upon a crash which is an image of the memory
    - `gdb ./executable core`: Run executable with a core file
- **Instrumentation:** Addition of a tool to measure information/performance
  - **Profiling:** Runtime analysis of an executable through instrumentation
    - **gprof:** Linux profiling utility which describes how much time was spent in each part of the program
      - `gprof ./executable gmon.out`: Generate a log showing time usage in each part of the program
  - **Valgrind:** Memory debugging and profiling tool
  - **mtrace:** Linux memory tracking utility which traces allocation and frees dynamic memory
    - `mtrace ./executable log.txt`: Generate a log showing memory usage
    - Less disruptive than Valgrind
- Use a GPIO pin to track software performance, and an oscilloscope to view changes in pin value
- **List Dynamic Dependencies (LDD):** Utility which locates required missing libraries

5 Communication and Networking

- **Serial port:** Direct (wire-to-wire) connection between two systems
  - Can transmit (Tx) and/or receive (Rx)
- **Serial protocol:** Ruleset for bitwise communication over a serial port
  - **Bitrate:** Frequency of each bit communicated
  - **Start bit:** Bit which represents the beginning of information transfer
  - **End bit:** Bit which represents the termination of information transfer
  - Examples of protocols:
    - **RS232:** Serial protocol which operates from -12V (representing 1) to 12V (representing 0)
    - **TTL:** Serial protocol which operates from 0V to 3V/5V
    - **Dynamic Host Configuration Protocol (DHCP):** UDP/IP protocol where an IP address is dynamically assigned to a machine so it can be connected to on a network
- **Pipe:** Unidirectional communication endpoint between two threads or processes
- **Socket:** Abstracted bidirectional communication endpoint for sending data between two processes on the same computer or across a network
  - Server listens for data, client sends data
  - **TCP:** Communication protocol where all data is sent in order with automatic retransmission
  - **Datagram (UDP):** Communication protocol where each packet is standalone
    - `htons()`: Function which sends a short data packet from host to network
    - `htonl()`: Function which sends a long data packet from host to network
    - `ntohs()`: Function which receives a short data packet from network to host
    - `ntohl()`: Function which receives a long data packet from network to host
- `netcat`: Linux utility which reads/writes from TCP or UDP network connections
- **Network byte order:** Protocol of whether the least or most significant bit is sent first
  - **Big endian:** Protocol where the most significant bit (MSB) is sent first

6 Signals and Noise

- **Signal:** Value to be recorded
- **Ground:** 0V
- **Voltage regulator:** Component which converts unstable input voltage to a stable output voltage
- **Quantization:** Resolution of a single bit which involves a detectable change
- **Sample rate:** How quickly an A2D samples the input
  - Audio requires high sample rates
  - Potentiometers require low sample rates
- **Piecewise Linear (PWL):** Approximation of a function with a series of lines
- Strategies to tolerate noise:
  - Change state only when a number of samples exceed the threshold
  - Require additional value beyond the threshold in each direction to change the value
  - Use a simple moving average of a past set of samples
  - **Simple Moving Average:** Method of smoothing equally weights all values
  - **Exponential Smoothing:** Method of smoothing which provides greater weights to recent values
- **Hysteresis:** Property of a state machine where the changes lag behind the actual values
- **Clamping:** Capping a signal at its maximum or minimum possible value to prevent overflow

7 Electronics

- **RS232:** Recommended Standard which is a specific bitrate with voltages -12V meaning 1 and 12V meaning 0
- Circuits are often damaged through water, short-circuiting, and ESD
  - **Electrostatic Discharge (ESD):** Static shock given to a system
- **Voltage:** Pressure which causes electricity flow; the difference in pressure between two points in a circuit
  - Symbol: V
  - SI Unit: Volt (V)
- **Current:** Flow of electrons caused by a difference in pressure between two points in a circuit
  - Symbol: I
  - SI Unit: Amp (A)
    - **Microamp:** $1/1,000,000$ of an amp
    - **Kiloamp:** $1,000$ amps
    - **Megaamp:** $1,000,000$ amps
  - Flows from higher voltage $(+)$ to lower voltage (-)
  - Electrons flow in the opposite direction
- **Resistance:** Resistance experienced by the current when flowing between two points
  - Symbol: R
  - SI Unit: Ohm (Ω)
  - Ohm's Law: $R = \frac{V}{I}$

7.1 Circuits

- **Open circuit:** Uncompleted circuit which has infinite resistance and no current
- **Closed circuit:** Completed circuit
- **Short circuit:** Circuit which has no resistance and high current
- Components:
  - Current is constant across all components
  - Voltage is lost across each component to equal the total voltage
  - **Sourcing:** Components which outputs voltage
  - **Sinking:** Component which inputs voltage
  - **Shorted:** Component which is directly connected to a power source or ground
  - **Floating:** Component which is neither connected to a power source nor a ground
- Serial circuits: Calculate the total resistance, calculate current (constant across all components), and use it to find voltage drop across each component
- **Diode:** Component which only allows current in one direction
  - **Light Emitting Diode:** Diode which creates illumination

- **Resistor:** Component which generates resistance and creates voltage drop
  - **Current-limiting resistor:** Resistor with the purpose of reducing current
  - **Pull-down resistor:** Large resistance between a component and ground to create a weak pull-down effect when a circuit is open
  - **Pull-up resistor:** Large resistance between a component and the power source to create a weak pull-up effect when a circuit is open
  - **Resistor divider:** Equal pull-up and pull-down resistors which divides the voltage in two
- **Pulse Code Modulation (PCM):** Format of an audio file which stores the amplitude at each sample point
  - **Quantization:** Accuracy of a measurement
  - **Bit depth:** Number of bits per quantization
  - WAV file has a bit depth of 16
  - **Superposition:** Combining multiple signals to create a single signal

7.2 Software Interactions

- **Debouncing:** Process of removing fluctuation when switching states to read a stable value
- **General Purpose Input-Output (GPIO):** Pins which can be set to read a value, or output a a value (0 or 1)
- **Inter-Integrated Circuit (I^2C):** Direct communication value setting/reading protocol for hardware chips
- **Analog-to-Digital (A2D):** Method to read analog voltages to a digital value
- **Pulse-Width Modulation (PWM):** Method to generate analog voltage by toggling a digital value over time

8 Bitwise

- Bitwise operators:
  - OR (|): Set selected bits
  - AND (&): Clear unselected bits; only show selected
  - NOT ($\tilde{}$): Invert all bits
  - XOR ($\hat{}$): Invert selected bits
  - Bitshifting (<< for left, >> for right): Moving all bits one way or another and removing the displaced digit
  - Mask: Selects a field in a bit-flag
- Bitshifting 1 by the number of a certain position creates a mask for that position, which can be ORed to toggle it (i.e. if LED0_BIT == 4, then 1 << LED0_BIT creates a mask at the 4th bit from the right)
- Test to check whether data fields are listed in the correct order in the data value
- Applying a mask with & and checking whether the remaining value is equal to 0 returns whether the bit is on or off
- Given the following predefined values where LEDs are active high and buttons are active low:

VALUE

LED0_BIT
LED1_BIT
LED2_BIT
LED_MASK = (1 << LED0_BIT) | (1 << LED3_BIT) | (1 << LED2_BIT)

BTN0_BIT
BTN1_BIT
BTN_MASK = (1 << BTN0_BIT) | (1 << BTN1_BIT)

SPD_BIT_BEGIN
SPD_MASK

The calculations are as follows:

```
_Bool isLed0On = (VALUE & (1 << LED0_BIT)) != 0;

_Bool isAnyLEDOn = ((VALUE & LED_MASK) != 0);

_Bool areAllLEDsOn = (VALUE & LED_MASK) == LED_MASK;

// If (VALUE & BTN_MASK) == BTN_MASK,
// then because buttons are active low, no buttons are pressed.
_Bool isAnyButtonPressed = (VALUE & BTN_MASK) != BTN_MASK;

_Bool areAllButtonsPressed = (VALUE & BTN_MASK) == 0;
```

```
void turnOnLed0() {
        VALUE |= (1 << LED0_BIT);
}

void turnOnAllLeds() {
        VALUE |= LED_MASK;
}

void turnOffLed() {
        // ~(1 << LED_BIT) sets the LED0 bit to 0, and all
        // other bits to 1.
        // ANDing it changes the LED0 bit to 0, and does not
        // change other bits.
        VALUE &= ~(1 << LED_BIT);
}

void turnOffLeds1And2() {
        VALUE &= ~(1 << LED1_BIT | 1 << LED2_BIT);
}

void turnOffAllLeds() {
        VALUE &= ~LED_MASK;
}

void turnOffAllLedsExcept2() {
        // Remove the LED2 bit from the inverted LED mask
        VALUE &= (~LED_MASK | (1 << LED2_BIT));
}

void toggleLed0() {
        VALUE ^= (1 << LED0_BIT);
}

void toggleAllLeds() {
        VALUE ^= LED_MASK;
}

// Assume ints are in the correct format and do not need
// to be converted to/from binary.
int getSpeed() {
        // Mask the correct bits
        // Bitshift the value so that the speed is at the least
        // significant (rightmost) bit
        return (VALUE & SPD_MASK) >> SPD_BIT;
}

int setSpeed(int speed) {
        // Shift speed to the correct location
        // Remove excess bits outside the speed bits
        int newSpeedBits = (speed << SPD_BIT) & SPD_MASK;
        // Create the value with cleared speed bits
```

```
        // Add the new speed bit
        VALUE = (VALUE & ~SPD_MASK) | newSpeedBits;
}
```

9 Operating System Components

- **Application Binary Interface (ABI):** Standard of how a program uses datatypes, function calling conventions, and system calls
- Bootup components:
  - **Embedded Multi-Media Controller (eMMC):** Flash storage on a chip on the board which contains the vital operating system files
  - **uSD Card:** SD card which can act as a replacement booter or flasher for the eMMC in case of corruption
  - **U-Boot:** Bootloader which initializes hardware and loads the kernel into memory
    - **Trivial Transfer File Protocol (TFTP):** Protocol used to boot a machine using specific files
- **Kernel:** Core operating system component which handles process control, memory, IO, etc.
- **Root filesystem:** Filesystem which contains an operating system
- **Network File System (NFS):** Direct method to mount a directory from another device on the network, mirroring changes immediately
- Linux directories:
  - `/dev`: Devices
  - `/etc`: Local system configuration
  - `/proc`: Kernel and process information
  - `/lib`: Shared libraries and kernel modules

10 Linux Kernel Programming

- Kernel is privileged; user space access to hardware is restricted
  - **Syscall:** System call to the kernel to access privileged instructions
  - `strace`: Linux utility to show syscalls
    - `ioctl()`: Function shown in strace which controls a device through its virtual file
  - Delays using system calls block the process and allow the CPU to work on other tasks
- **Monolithic kernel:** Kernel which has a single process image and address space
- **Device tree:** Structure used by the kernel to manage connected hardware
  - **Device Tree Source (.dts):** File which
  - **Device Tree Blob (.dtb):** Binary file which manages the device and is passed by U-Boot to the kernel
  - **Device Tree Overlay (.dtbo):** Binary file which can modify the device tree at runtime
- uboot downloads the kernel and device tree, runs Linux, and loads the root file system

11 Drivers

- File extension: `.ko`
- Process of creating a driver:
  - Allocate major/minor node numbers
  - Create nodes in `/dev` and `/sys` for interfacing
  - Register as a character driver
- **Character driver:** Driver which handles bytes and buffers
- **Miscellaneous driver:** Driver for which the system manages allocation of node numbers, `/dev` and `/sys` nodes, and character driver registering for ease of creation
  - Programming drivers:
  - `module_init()`, `module_exit()`: Functions which run during load and unload
    - Set functions to be static to prevent conflicts with other modules
  - `__init`, `__exit`: GCC extensions which reduce memory use by freeing or skipping the code when unnecessary
  - `file_operations` (fops): Struct which links functions to your kernel file read/write operations
    - When the file in `/dev/driver_name` is modified, the associated loaded file operations struct is registered with the function
- Linux commands to manage drivers:
  - `lsmod`: List all loaded modules
  - `insmod module.ko`: Load a module
  - `modinfo module.ko`: List the info of a module
  - `rmmod module.ko`: Unload a module

- Kernel must validate user-level pointers which could be NULL or invalid
  - Use functions `copy_from_user` and `copy_to_user`

12 systemd

- **.profile:** Program which runs at login
- **systemd (system daemon):** First user-space application which is executed at boot by the kernel
  - **Daemon:** Background procress
- Always use absolute paths
- **Watchdog (WD) timer:** Program which prevents system lock-up
  - **Hit/pet:** To contact the WD
    - Opening file `/dev/watchdog` restarts the timer
    - If the program exits, WD continues running
  - Applications use a WD to recover from faults:
    - Periodically hits the WD
    - WD restarts the board if the timer expires
    - Critical when a reboot cannot be manually triggered

13 Bare Metal Programming

- Disadvantages:
  - No OS services such as threads, processes, file system, NFS, memory protection, swapping
  - No drivers or applications available
  - No terminal available
- Advantages:
  - Full hardware control
  - Minimal space usage
  - Tightly controllable timing (no context switches, pre-emption, page faults)
  - Programs can be designed to never exit
- Controlling pins:
  - Can be input, output, and/or generate interrupts (multiple functions available simultaneously)
  - Can be written by register access, or by writing to the SET/CLEAR registers
- GPIO process:
  - Enable clocks on GPIO modules
  - Enable GPIO modules
  - Set the Output Enable Register (GPIO_OE)

13.1 Interrupt Service Routines

- **Interrupt Request (IRQ):** Hardware signal set to the processor to inform of a specific event
  - **Interrupt Service Routines (ISR):** Routine which is registered to handle a certain interrupt
- ISRs are effectively multi-threaded
  - Variables modified in an ISR should be labeled `volatile` in order to prevent compiler optimization when a value changes during the ISR
  - ISR adds data to all stacks in a multithreaded environment

13.2 Timers

- Timers can be set to trigger an IRQ
  - Loads the value in TLDR (Time LoaD Register)
  - Counts up at $25mHz$ or $32.768kHz$
  - Immediately resets to the value in TLDR when the value reaches $0xFFFF\ FFFF + 1$
- **Prescale/divider:** Reduction of amount added to the clock every cycle (value of N equals a 2^N slowdown)
  - Division of the clock frequency equals exponentiation of the period
- Calculating period of time and timer clock frequency:

$$\text{Period} = \frac{0xFFFF\ FFFF - TLDR + 1}{\text{Timer Clock Frequency}} * 2^N \tag{1}$$

- E.g. Given an 8-bit timer with a 32Hz clock and no divider, find the maximum timer duration.
  - An 8-bit timer has maximum value $0xFF$.
  - To achieve the maximum timer duration, set $TLDR$ to 0.
  - The frequency is 32.
  - The equation is:

$$\text{Period} = \frac{0xFF - TLDR + 1}{\text{Timer Clock Frequency}} * 2^N \tag{2}$$
$$= \frac{0xFF - 0 + 1}{32} * 2^0 \tag{3}$$
$$= \frac{2^8}{2^5} \tag{4}$$
$$= 2^3 \tag{5}$$
$$= 8s \tag{6}$$

  - Answer: Maximum timer duration is 8s

- E.g. Given an 8-bit timer with a 32Hz clock, calculate the necessary divider and TLDR for a 200s period.
  - To change the period to 1s, divide the 32Hz clock by $N = 5$ to multiply the period by $2^5 = 32$.
  - The equation is:

$$\text{Period} = \frac{0xFF - TLDR + 1}{\text{Timer Clock Frequency}} * 2^N \tag{7}$$
$$= \frac{2^8 - TLDR}{32} * 2^5 \tag{8}$$
$$= 2^8 - TLDR \tag{9}$$
$$= 256 - TLDR \tag{10}$$
$$TLDR = 256 - 200 \tag{11}$$
$$= 56 \tag{12}$$

  - Answer: $N = 0$, $TLDR = 56$

- E.g. Given a 32-bit timer with a 25MHz clock and no divider, find the maximum timer period.
  - A 32-bit timer has maximum value $0xFFFF\ FFFF$.
  - Let $TLDR = 0$ for the maximum amount of time.
  - The equation is:

$$\text{Period} = \frac{0xFFFF\ FFFF - TLDR + 1}{\text{Timer Clock Frequency}} * 2^N \tag{13}$$
$$= \frac{0xFFFF\ FFFF - 0 + 1}{25MHz} * 2^0 \tag{14}$$
$$= \frac{2^{32}}{25,000,000} \tag{15}$$
$$\approx 170s \tag{16}$$
$$\approx 2.8min \tag{17}$$

* Answer: Maximum timer duration is about 2.8 minutes

– E.g. Given a 32-bit timer with a 25MHz clock, calculate the necessary divider and TLDR for a 2s period.

* Since the maximum timer period is greater than 2s (see previous question), no divider is necessary. Let $N = 0$.

* The equation is:

$$\begin{aligned}
\text{Period } = 2s &= \frac{0xFFFF\ FFFF - TLDR + 1}{\text{Timer Clock Frequency}} * 2^N && (18)\\
&= \frac{2^{32} - TLDR}{25,000,000} * 2^0 && (19)\\
50,000,000 &= 2^{32} - TLDR && (20)\\
TLDR &= 2^{32} - 50,000,000 && (21)\\
&= 0xFD05\ 0F80 && (22)\\
& && (23)
\end{aligned}$$

* Answer: $N = 0$, $TLDR = 0xFD05\ 0F80$

– E.g. Given the same parameters and TLDR from the previous question but with $N = 2$, how long will the period be?

* If $N = 2$, the period is multiplied by $2^N = 2^2 = 4$.

* Period $= 2s \times 4 = 8s$

14 FMEA

- **Failure Mode and Effect Analysis (FMEA):** Overview of how system components could fail and details about the failure detection/likelihood/effects/risk
- For each failure possibility:
  - Rate the severity (1-10)
  - Rate the likelihood (1-10) based on causes
  - Rate the effectiveness of detection (1-10)
- List possible actions to reduce risk
- **Risk Priority Number (RPN):** Failure rating which is represents the product of the severity, likelihood, and detection effectiveness (1-1000)
- **Danger zone:** A complex system which is tightly coupled and, therefore, difficult to test