CS213A: Embedded Systems

Professor Srivastava

Thilan Tran

Fall 2021

Contents

CS213A: Embedded Systems

- an **embedded system** is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints:
  - vs. general-purpose computers that support a wider range of needs and typically do not have physical world interaction
  - eg. home thermometers, smart speakers, smart watches, Roombas, self driving technologies:
    - embedded systems make rich inferences about contexts, life patterns, behaviors, intents
    - diverse domains include health, transportation, security, entertainment, etc.
  - embedded trends include networked, miniaturization, human-coupling, AI-enabled, security and hacking concerns
- these embedded systems are typically connected to the Internet, ie. **Internet of Things (IoT)** devices:
  - IoT devices are these embedded computing devices that are interconnected in the Internet
    - allows for advanced connectivity of devices, systems, services
  - these "things" now dominate the edge of the internet rather than traditional computers
  - unique challenges include security, privacy, and configuration
  - *passive* IoT devices unilaterally upload sensor data
    - while *active* IoT devices upload data while also reading and analyzing results
  - common three-tier IoT systems:
    - the actual **embedded system** or device that senses, actuates, buffers, processes, provides UI
    - a **gateway** that relays, senses, buffers, processes, provides UI
    - the **cloud services** that archive, process, disseminate
- other related concepts:
  - **ubicomp** or ubiquitous computing, where computing can be made to appear anytime and everywhere in everyday life
  - **Cyber-Physical Systems (CPS)** have perception-cognition-action loops:
    - integrates computation, networking, sensing, and physical processes
    - build upon embedded systems with modeling of physical processes and their dynamics

- **edge computing** and **fog computing** aim to extend the cloud closer to the devices producing and consuming the data:
  * edge computing is a paradigm where much processing occurs on the end device, rather than in the centralized cloud
  * fog computing is a similar paradigm where processing occurs on network nodes near the end device

Embedded Hardware Platforms

- a generic embedded system platform has the following components:
  - human interface to interact with users
  - sensors and actuators to interact with the physical world
  - network and I/O interface to interact with servers and other devices
  - computation and storage system
  - power subsystem
- computation options:
  - custom SoC with ASIC or FPGA
  - custom board with COTS components
  - custom system using standard chassis like PCI, peripheral cards, etc.
  - use standard platform eg. ruggedized PC
  - specific ICs typically take least power and are fastest, while microprocessors and more flexible computation options take more power and are less fast
    - in terms of cost, mass volume production of specific ICs or SoCs will bring down their prices significantly
  - microcontroller units (MCUs) are the most common option:
    - microcontrollers package a microprocessor (CPU) with RAM, ROM, I/O, etc.
- power management options:
  - lower-power reduced-functionality modes (dynamic power management):
    - eg. idle, stop clock, power off, etc.
    - save power by changing modes
  - frequency scaling (dynamic frequency scaling) requires an appropriate clocking subsystem
    - power is proportional to frequency
  - voltage and frequency scaling (dynamic voltage-frequency scaling)
    - power is proportional to frequency and voltage squared
  - ideally, we want high performance for intensive workloads:
    - while conserving energy as workloads mitigate
    - can switch scaling type depending on workload type
    - one approach is big-little, where we have two cores that can be used depending on workload eg. by tracking a weighted average of CPU load
- I/O options:
  - inputs can include keyboard, mouse, pointing devices, speech

        * outputs can include monitor, projector, displays, audio
    - challenges include performance requirements eg. data rate, latency, synchronization
        * pin constraints, power supply requirements
    - typical supported API are get & set parameters, transmit & receive data, enabled & disable functions
    - I/O compared to data and program memory:
        * I/O device registers can't cache
        * timing matters
        * reading and writing can trigger hardware to perform some function
        * sharing across applications can be challenging
    - one technique is to have specialized I/O instructions just for I.O
        * restrict to kernel mode
    - another technique is to have memory-mapped I/O:
        * access devices just like they are memory
        * while using memory protection mechanisms
- pin types:
    - **general purpose I/O (GPIO)** pins are digital pins that can be read and written by software
        * eg. control external peripherals, LEDs, software serial protocols, etc.
    - analog I/O pins can be read and written by software:
        * AI pins perform analog to digital conversion at a certain bit resolution
        * AO pins perform digital to analog to digital conversion
        * **pulse width modulation (PWM)** is another way to convey analog values on a single digital pin using varying duty cycles
- serial interface protocols:
    - *pros*: serial interfaces reduce the pin counts, avoiding timing issues
    - *cons*: may need clock recovery and are difficult to memory map
    - eg. UART / RS232, I2C, SPI
    - synchronous vs. asynchronous is whether data is sent with or without a clock
    - full duplex vs. half duplex is whether data can be sent and received simultaneously or not
    - in a master / slave approach, one device is the master and others are slaves
        * usually synchronous, as the master supplies the timing clock for data in both directions
    - in a multi-master bus, the master / slave bus may have more than one master
        * needs an arbitration scheme
    - in a point-to-point or peer interface approach, there are no masters or

slaves
    * usually asynchronous
  - in a multi-drop approach, there is one transmitter and several receivers
  - in a multi-point approach, there are more than two peer transceivers allowing for bidirectional data exchange
- UART and RS-232 are a common serial interface found on MCUs and peripherals:
  - products that support RS-232 are called **universal asynchronous receiver transmitters (UARTs)**
  - unbalanced bus capable of full-duplex communication between two receiver / transmitter pairs
  - speeds up to 115.2 Kbps, with distance up to over 200 feet
  - typically consists of a start bit, data bits, and parity or stop bits (optional)
  - logic high typically is a negative voltage, while logic low is a positive voltage
  - communication protocol:
    1. start bit indicates a character is coming, and receiver resets its timers
    2. then, the receiver samples in the middle of the bits using a timer
        * transmission rate ie. bit width must be known!
    3. stop bit should be 1
- the **inter-integrated circuit (I2C)** bus is another popular, patented interface:
  - half-duplex, synchronous, multi-master bus requiring only two signal wires, data (SDA) and clock (SCL)
  - these lines are pulled high via pull-up resistors and controlled by the hardware via open-drain drivers
  - I2C is an addressable communications protocol:
    * allows master to communicate with individual slaves
    * 7 or 10-bit address
  - master initiates communication and generates all clock signals for data transfer in either direction
    * through clock stretching, the slave can pull SCL low if it needs more time to process data bits
  - three speeds include slow or 100 Kbps, fast or 400 Kbps, and high-speed or 3.4 Mbps
    * each speed is downward compatible
  - communication protocol:
    1. master generates a start condition
        * start condition is a high-to-low transition of SDA while SCL is high
    2. master sends an 8-bit data word
        * data bit transitions take place while SCL is low

3. the receiver generates the acknowledge bit:
    * pulls SDA low while master releases the line and allows it to float high
    * if the acknowledge bit is read by the master as high, it should consider the last communication word as not received
4. master generates a stop condition or a repeated start
    * stop condition is a low-to-high transition of SDA while SCL is high

- the **serial peripheral interface (SPI)** is a synchronous serial bus with multi-master / slave protocol:
    - present on many microcontrollers
    - four signals are master out slave in (MOSI), master in slave out (MISO), serial clock (SCK), and active-low slave select (/SS)
    - allows for data rates over 1 Mbps in full duplex mode
    - generally limited to on-board communications and short traces
    - data is clocked simultaneously into slave and master based on SCK pulses:
        * master supplies the SCK
        * allows for four different clocking types based on polarity and phase of SCK
    - typically requires many pins when using multiple slaves
- other I/O interfaces:
    - Microwire similar to SPI
    - 1-wire
    - USB, which allows devices to switch back and forth between host and device