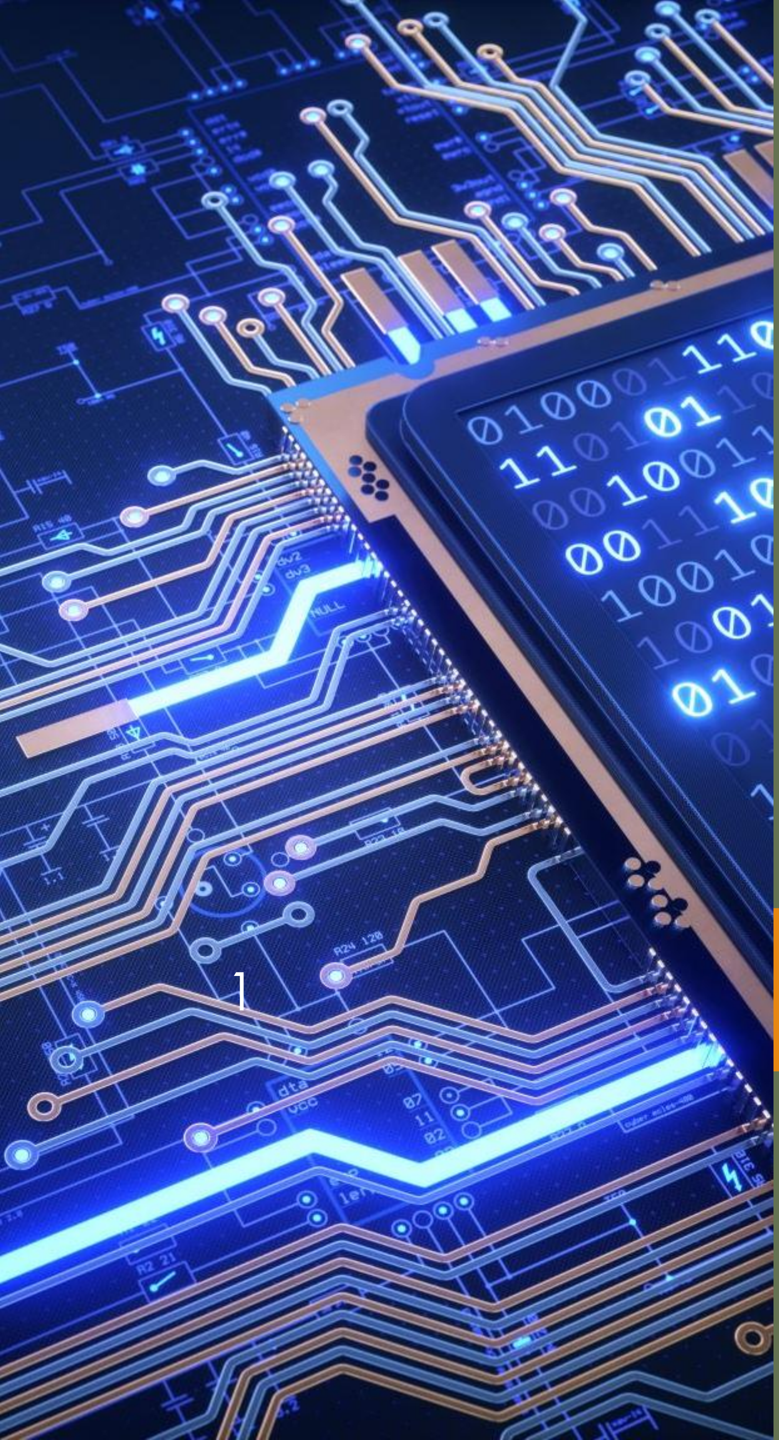


Embedded Controller programming for Real Time Systems: ECE-40097

Lesson 9

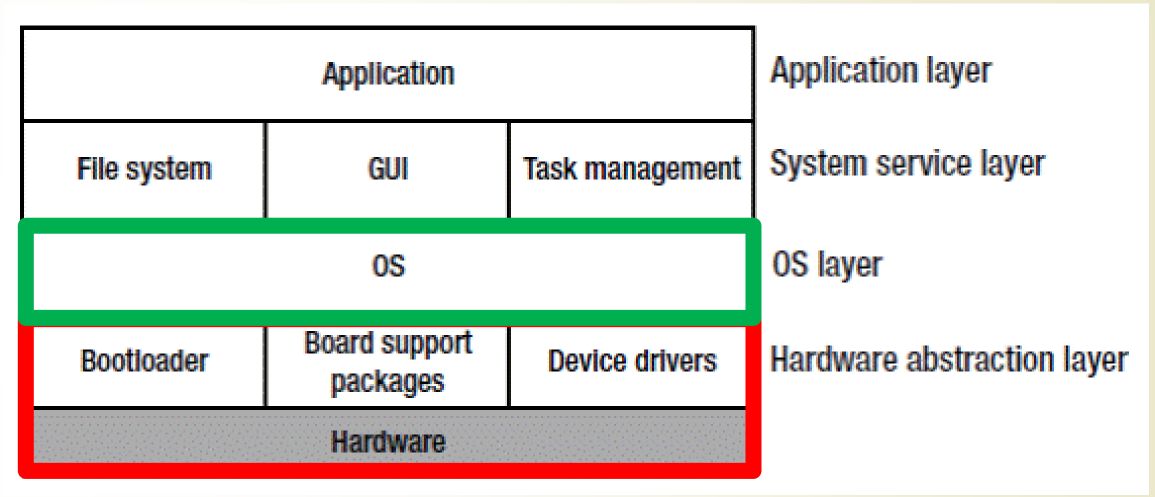
Vijay Kumar

Main Topics

- EMBEDDED SOFTWARE DESIGN
- DESIGN GUIDELINES
- Feedback from you
- FUTURE DEVELOPMENT ON BOARD
- COURSE OBJECTIVE MATCH-UP
- COURSE WRAP UP

Embedded SW design

- Very Generic design
- Goal should be to protect the SW as HW will change over time



Roles

- Very likely you will be involved in one or more layers of SW
- Depends on the system
    - And business needs
    - And your Role (developer, Lead, Manager, VP)
- Strongly recommend to master in one layer
    - Will be easier to comprehend other layers

HAL Layer

- HAL is abstraction layer between OS and HW
  - We have used HAL in our coding examples without OS
- HAL consists of bootloader, BSP, device drivers and other components
  - Bootloader is used to load OS
    - Usually provided by vendor

BSP

- Stands for Board Support Package
- It empowers OS to be independent of HW
- Same OS could run on different HW but will require different BSP
  - Vendors support some from of BSP – at least for one OS type
  - Otherwise, this requires separate development effort

Service Layer

- Service layer uses OS services and provides its own services
  - Task/thread management
  - Log services
  - OTA
  - Timers
- Manages transport
  - Local – BLE, WIFI, NFC
  - Long distance – Cellular, Satellite
- House keeping services
  - Status, statistics
  - Watchdog
- Power management
  - Sleep, Wakeup, resume, shutdown

Service Layer HAL

- Sometimes there is HAL layer at service layer too
  - It is done to run the same service layer on different OS
- Sincere effort should be made to demarcate layers
  - Clearly defined the role and responsibility
  - Design loosely coupled interface
  - To do parallel development
- If service layer is modified, application layer should be agnostics
  - This should be the design goal

Application Layer

- Application layer provides system functionality and business logic
  - This is what user sees and feels
- Should only change if we are changing Language or Framework
- Should be protected as much as possible
  - Lot of time is spent here to have everything working

Design principles

- Single Responsibility
  - Class, File, methods should have single responsibility
- Cohesive design
  - Designed to work together
- Loose coupling
  - Change in one module shouldn’t impact other modules
- Hooks for debugging
  - Especially required for embedded devices
  - Must be thought at design phase
- Power management
  - Many devices run on batteries
- And Security
  - Getting more attention with connected devices

Design principles

- Design big but implement small
  - Don’t spend too much time in finding every defect in SW and design
  - Get software out even with little confidence in competitive market
- Use of delay() is devil for embedded software
  - Use only when pushed to corner
- Be super careful with task and thread priority
- Polling is recipe for disaster
  - Unless there are no options but Polling
- Always design for housekeeping thread/task
  - To monitor the health of system and SW
- Have a way to enable/disable features remotely

Types of Devices

- With Display
  - Easy to manage and maintain
  - All combined and packaged in one unit
  - maintenance headache – as this requires complete device change
- Without display
  - Also known as black box/Gateway/Hub
  - Could be used as edge device but no storage or processing
  - Behaves like stand alone or a router to forward messages between entities
  - Primarily used for connecting to bring your own devices (BYOD)
  - Future upgradability

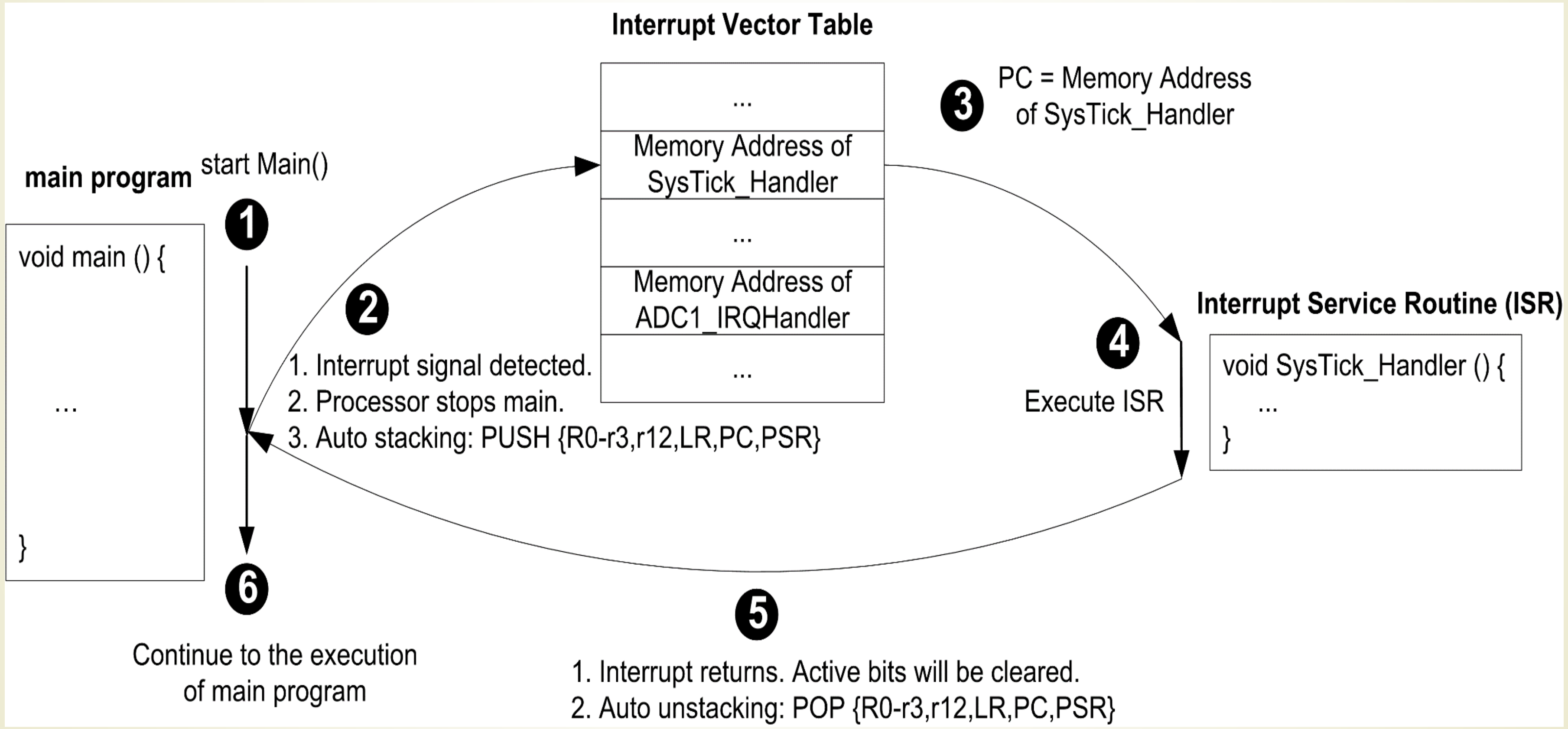
Embedded debugging

- Have the configurable trace levels
  - Error, Info, Debug and more
- Circular buffer to store the log message
  - Could be also stored in flat files
  - Retention period
- Make sure to have capability added to upload logs/debugging information over the air
  - Plan and design for alternate ways to get logs
  - OTA is preferred
  - Alternatively, USB could be used copy logs from device
- Visual indicator on the device
  - May be colored LED
  - Serial port to connect and find the status
  - **Any guess – why not USB ?**

Connectivity

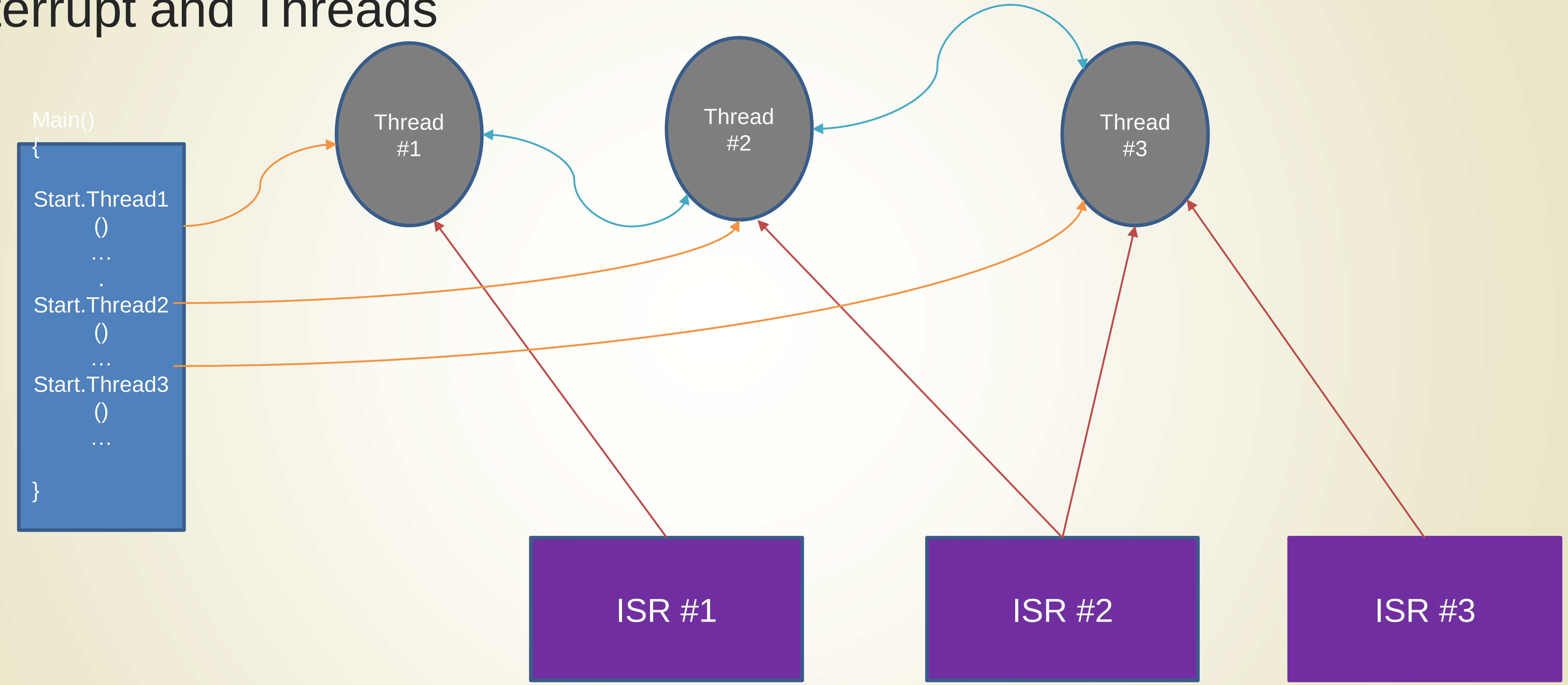
- Support for more than one communication link
  - For un-interrupted coverage
  - Managing multiple communication is daunting task, especially for custom board
  - Gets complicated when we implement near end communication (BLE, WIFI)
- Use of WI-FI as client and host mode
  - Used as client mode to connect to WIFI Node for back-end connectivity
  - Used as host when near by devices want to connect to
  - Especially useful with Hotspot to save cost of embedded device and BYOD
- Recommend to start with one communication and add more in later releases
  - Probably with Cellular or Satellite

Interrupt and Threads



Interrupt Vector Table
...
Memory Address of SysTick_Handler
...
Memory Address of ADC1_IRQHandler
...
3 PC = Memory Address of SysTick_Handler
main program
start Main()
1
void main () {
...
}
2
1. Interrupt signal detected.
2. Processor stops main.
3. Auto stacking: PUSH {R0-r3,r12,LR,PC,PSR}
4
Execute ISR
Interrupt Service Routine (ISR)
void SysTick_Handler () {
...
}
6
Continue to the execution of main program
5
1. Interrupt returns. Active bits will be cleared.
2. Auto unstacking: POP {R0-r3,r12,LR,PC,PSR}

Interrupt and Threads



Communication between Threads and ISR

- Between ISR and threads
  - Semaphore
  - **Global variable**
  - **Callback method**
  - Shared memory

- Between threads
  - Semaphore/MuTex
  - Queues
  - Sockets
  - Shared memory
  - Global variable

Watchdog - Software

- Implemented in reliable thread
- Action of failed watchdog is user intended
- Watchdog thread must be running to watch system and other threads
- Easy to implement and suitable for any type of real time system
- Action could be multitiered
- Designed for tolerance and filter out the occasional hiccups
- Collect stats for system improvement

Watchdog - Hardware

- Implemented in HW, if available
  - Good examples are: IWDG and WWDG
- Must have for hard real time systems
- Ideal for systems without user interactions
  - Space probe (Mars rover)
- Could be implemented as multistage
  - Each staged to perform specific action
- Combination of Hardware and Software
  - More fail-safe approach
  - Guaranteed to detect failure in timely fashion and take action
  - Adds complexity
  - Primarily suited for large system with more than one processor

Future work on this Board

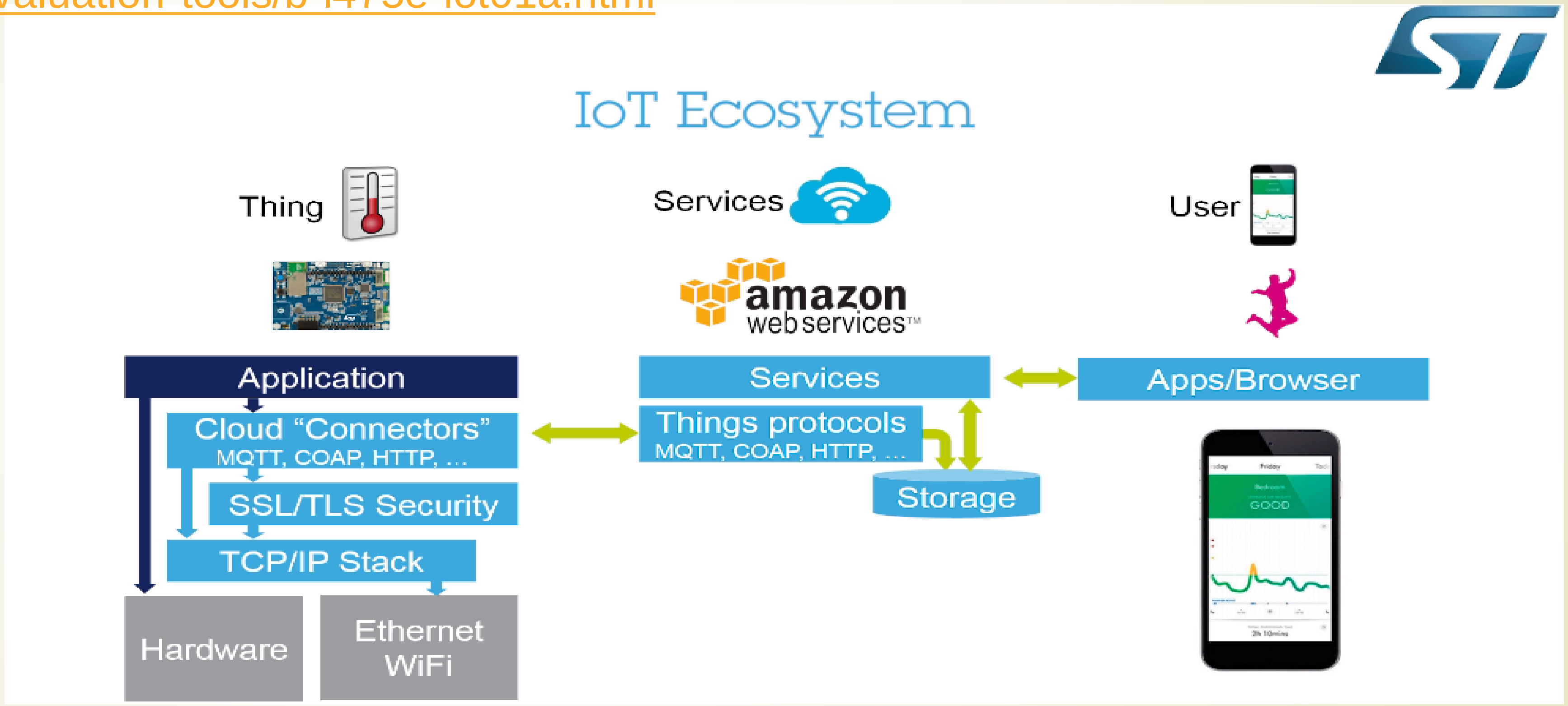
- Supports communication
  - WIFI
  - BLE
  - NFC ??
- Accelerometer
- Gyroscope
- Humidity and Temperature sensor

IOT Use

- Great board to be used as IOT node
- Connectivity to cloud
  - Azure or AWS
- Use the sensor to send data in the cloud
- Develop application on Android/IOS phone to read data from cloud.
- Extension has added course on IOT, if you are interested
  - https://extension.ucsd.edu/courses-and-programs/introduction-to-iot-and-embedded-systems

IOT Ecosystem

https://www.st.com/en/evaluation-tools/b-l475e-iot01a.html



Course Objective

The objective is to equip students with understanding of controller (ARM Coretx-M4), controller peripherals, programming the controllers, such as interrupt, timers, RTC etc. in C and Assembly language.	Completed
Overall goal of this course is to provide students with hands on experiences of writing interrupt service routines (ISR), timers interrupt, subroutines in Assembly language.	**Completed**
Other objectives include developing practical knowledge of designing, and development of software for real time embedded systems, which involves applying the best design practice with or without an operating system	Completed

Course Objective

- Learn assembly language for Cortex-M4 controller and write subroutines, in-line code with C language. **Completed**
- Understand vector table and write interrupt service routines for UART and reset button. **Completed**
- Apply timer interrupt and program timers (with HAL and without HAL) for applications such as delay, RTC, watchdog. **Completed**
- Develop software based on events and interrupts, e.g. character received on UART, pressing reset button on the development Kit. **Completed**

Course Wrap-up

- Thank you for giving me the opportunity to teach this course
  - Do let me know if you any questions for me anytime in future
    - vijay88@gmail.com
  - I will end this course with quote from the book
    - I hear and I forget. I see and I remember. I do and I understand
  - Wish you all the best for future courses and in your career
    - Happy embedded design and programming
    - Take care and Bye - Bye