Real Time Operating Systems

UNIVERSITY OF IDAHO – JOHN C. SHOVIC, PHD

COEUR D'ALENE

Assignment #16/17 Discussion

- I2C issues
- Stepper Motors
- Buttons

IOT Presentations

IOT Presentation Schedule

John Shovic

All Sections

Last Name	First Name	IOT Presentation Date
Fredrickson	Samuel	April 21st
Harris	Justin	April 21st
Hawkins-Start	Hunter	April 21st
McVickar	Christopher	April 21st
Nardi	Gage	April 21st
Olds	Kristie	April 21st (1:45pm)
Palmer	Nathaniel	April 26th
Preston	Zachariah	April 26th
Prestwitch	Ross	April 26th
Shoup	Scott	April 26th
Wells	Garret	April 26th
Williams	Taegan	April 26th (1:45pm)


University of Idaho
Coeur d'Alene

New Appliances – I lied.

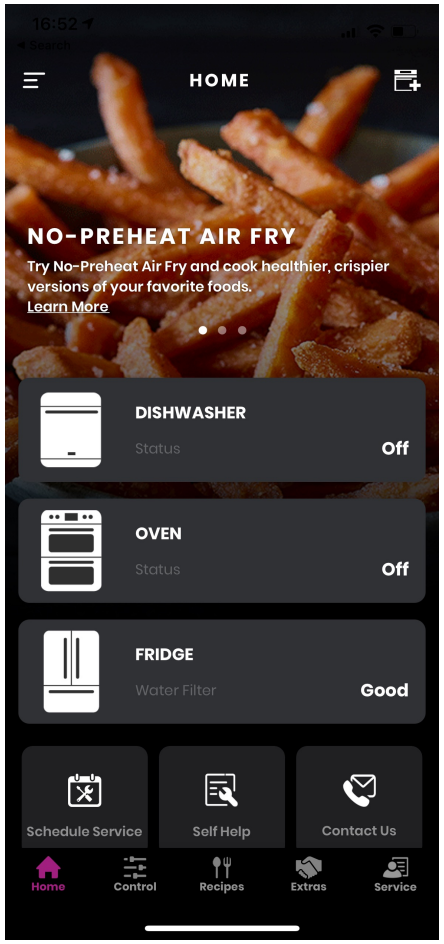
- We bought 4 new medium high end GE Café Applicances
- Dishwasher
- Gas Stove
- Microwave
- Refrigerator

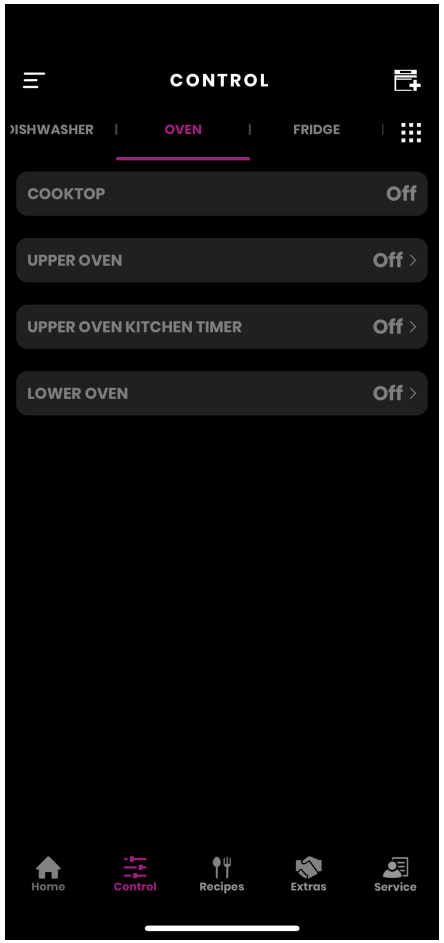
- Ack! They are all connected appliances!

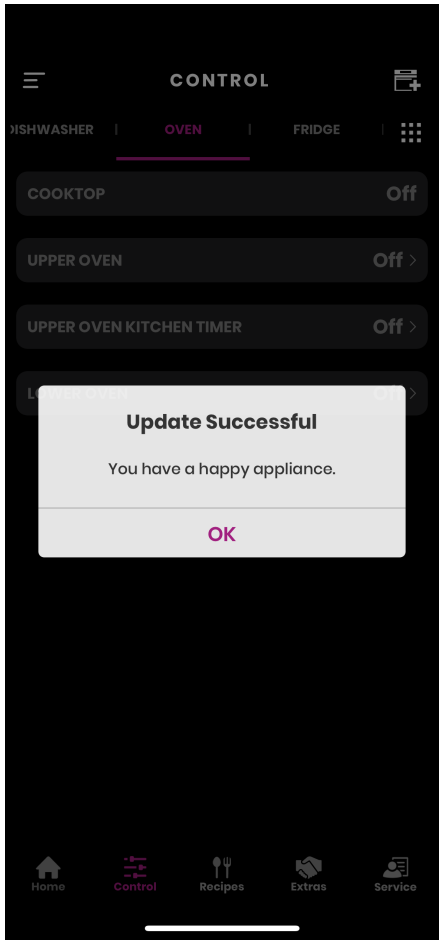
- The setup was a different screen and flow for each one! No commonality except for password length

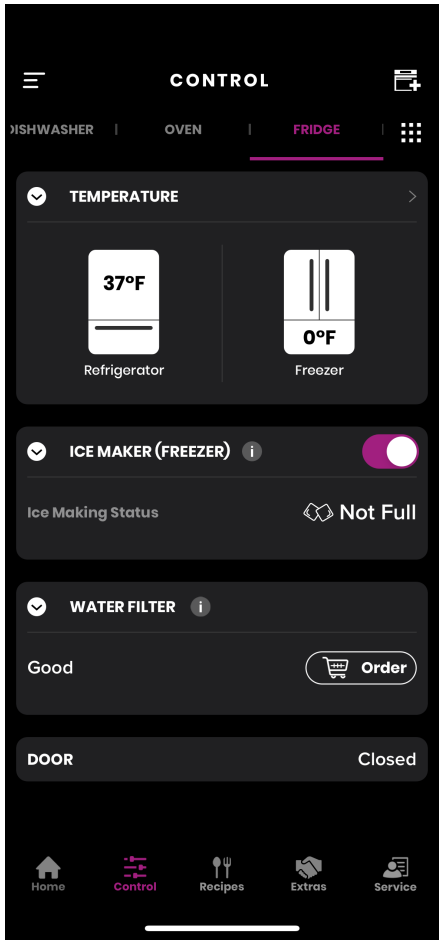

Refrigerator
Recommended: 37°F
Actual
37°F
Recommended: 0°F
Freezer
Hold 3 sec for °F/°C
Door Alarm
Ice Maker
Lock Controls
Reset Filter

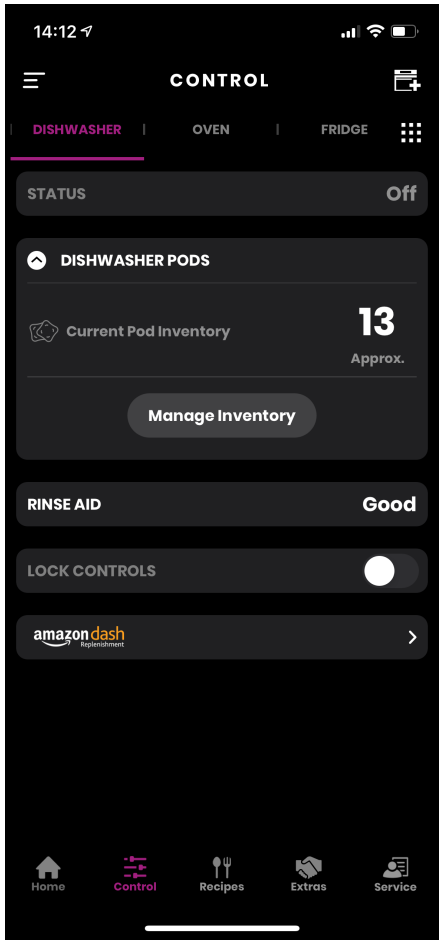

WiFi
Clean
Sanitized
Low Rinse Aid
Auto
Pots
Normal
China
Expres

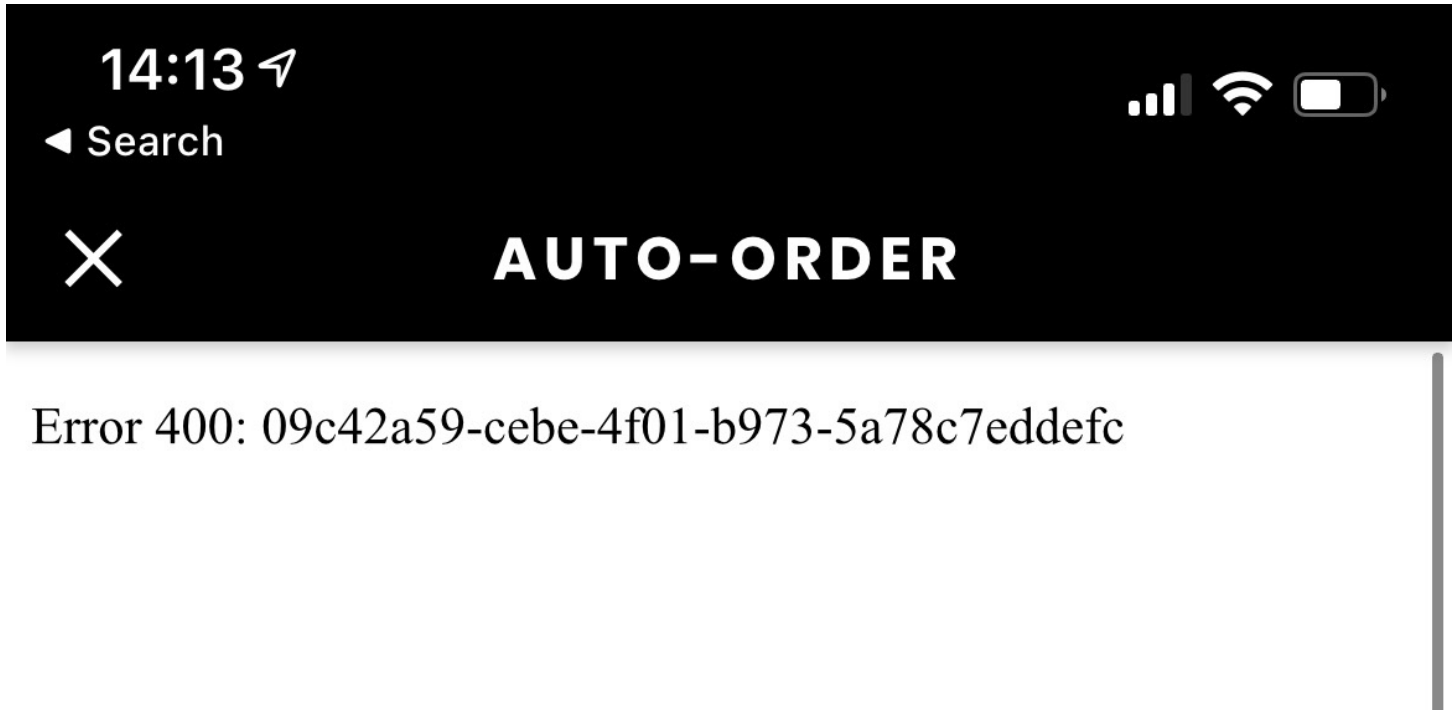

16:52
HOME
NO-PREHEAT AIR FRY
Try No-Preheat Air Fry and cook healthier, crispier versions of your favorite foods.
Learn More
DISHWASHER
Status
Off
OVEN
Status
Off
FRIDGE
Water Filter
Good
Schedule Service
Self Help
Contact Us
Home
Control
Recipes
Extras
Service

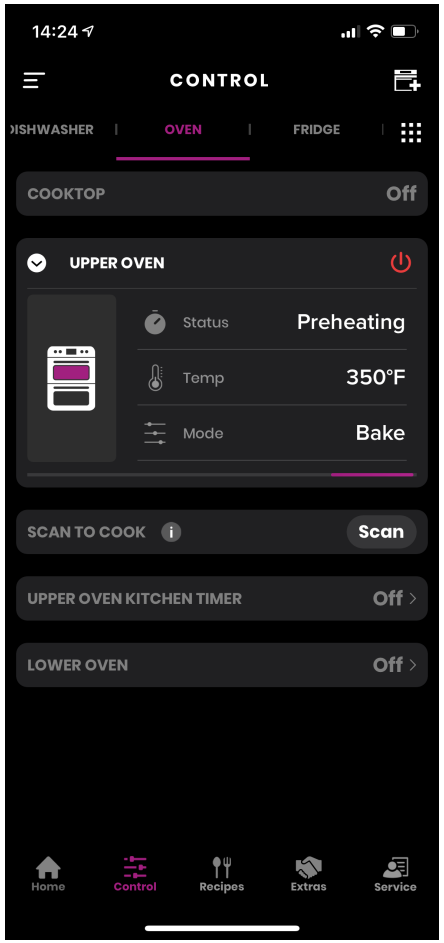

CONTROL
ISHWASHER
OVEN
FRIDGE
COOKTOP
Off
UPPER OVEN
Off
UPPER OVEN KITCHEN TIMER
Off
LOWER OVEN
Off
Home
Control
Recipes
Extras
Service


CONTROL
DISHWASHER
OVEN
FRIDGE
COOKTOP
Off
UPPER OVEN
Off
UPPER OVEN KITCHEN TIMER
Off
Update Successful
You have a happy appliance.
OK
Home
Control
Recipes
Extras
Service


CONTROL
ISHWASHER
OVEN
FRIDGE
TEMPERATURE
37°F
Refrigerator
0°F
Freezer
ICE MAKER (FREEZER)
Ice Making Status
Not Full
WATER FILTER
Good
Order
DOOR
Closed
Home
Control
Recipes
Extras
Service


14:12
CONTROL
DISHWASHER
OVEN
FRIDGE
STATUS
Off
DISHWASHER PODS
Current Pod Inventory
13
Approx.
Manage Inventory
RINSE AID
Good
LOCK CONTROLS
amazon dash
Replenishment
Home
Control
Recipes
Extras
Service


14:13
Search
AUTO-ORDER
Error 400: 09c42a59-cebe-4f01-b973-5a78c7eddefc


14:24
CONTROL
DISHWASHER
OVEN
FRIDGE
COOKTOP
Off
UPPER OVEN
Status
Preheating
Temp
350°F
Mode
Bake
SCAN TO COOK
Scan
UPPER OVEN KITCHEN TIMER
Off
LOWER OVEN
Off
Home
Control
Recipes
Extras
Service

Comparing freeRTOS to other mainstream RTOS products

- There are dozens of RTOS systems out in the wild
- Major differentiator?
  - Closed
  - Open Source

Some Major Open Source RTOS

- freeRTOS (stewardship by Amazon now)
- uCOS
- DSPnano RTOS
- Trampoline Operating System (just like the name)
- TI-RTOS Kernel
- Real-time Linux
- eCOS
- Micro C/OS-II

Some Major Closed Source RTOS Systems

- vXWorks
- Windows 10 IoT
- Transaction Processing Facility
- RTOS-32
- RedHawk Linux (RHEL, CentOS; Ubuntu compatible)
- QNX (Blackberry, Ford Cars) - went OS to CS
- Phoenix-RTOS (IOT)
- Flexible Safety RTOS

Advanced RTOS Topics

Synchronization Techniques

- Unilateral rendezvous
- Credit Tracking (token passing)
- Bilateral Rendezvous
- Client-Server
- Synchronizing multiple tasks
- Selecting the right synchronization method

RTOS Pitfalls and Issues

- Complexity
- Issues – Task Jitter
- Issues – Thread Starvation
- Issues – Priority Inversion
- Issues – Deadlock

Debugging RTOS Applications

- Printf
- ITM Instrumentation Trace Macrocell – printf through hardware
- ASSERT
- SWD (JTAG requires 4 signal lines, SWD only requires 2 signal lines
- OS-Aware Debugging
- Real-time Tracing (needs hardware support for good accuracy)
- Debugging tools (CDB, etc. Need breakpoints!)
- Error Trapping

RTOS Application Design Patterns

- UART command processing
- Sensor Sampling Strategies
- Command Processing
- User Interactions
- TCP/IP Integration
- System Testing

Optimizing RTOS Applications

- RTOS Configuration
- Memory
- Performance
- Energy
- Trade-offs

Developing Secure RTOS Applications

- Using the MPU with Tasks
- Defining trusted and untrusted code
- Using Arm Trustzone with an RTOS
- Thinking like a hacker

RTOS Pitfalls and Issues

- Complexity
- Issues – Task Jitter
- Issues – Thread Starvation
- Issues – Priority Inversion
- Issues – Deadlock

Debugging RTOS Applications

- Printf
- ITM Instrumentation Trace Macrocell – printf through hardware
- ASSERT
- SWD (JTAG requires 4 signal lines, SWD only requires 2 signal lines
- OS-Aware Debugging
- Real-time Tracing (needs hardware support for good accuracy)
- Debugging tools (CDB, etc. Need breakpoints!)
- Error Trapping

RTOS Application Design Patterns

- UART command processing
- Sensor Sampling Strategies
- Command Processing
- User Interactions
- TCP/IP Integration
- System Testing

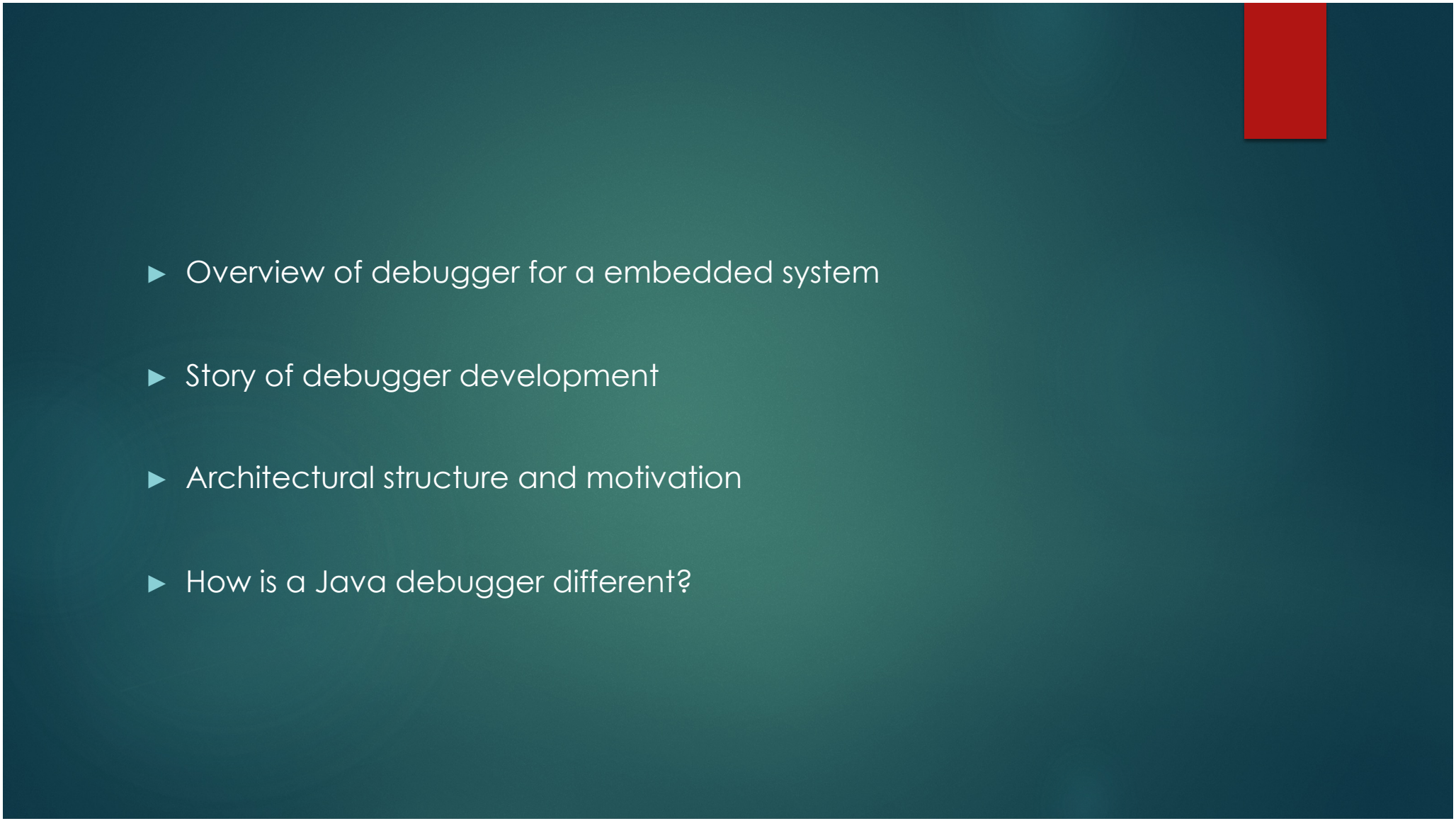
Optimizing RTOS Applications

- RTOS Configuration
- Memory
- Performance
- Energy
- Trade-offs

Developing Secure RTOS Applications

- Using the MPU with Tasks
- Defining trusted and untrusted code
- Using Arm Trustzone with an RTOS
- Thinking like a hacker

Debugging RTOS


Overview of debugger for a embedded system
Story of debugger development
Architectural structure and motivation
How is a Java debugger different?

What is a Debugger

- A tool to remove bugs from a program
- Used in program testing/inspection
- Used to test a module or algorithm

Current State of the Art

- GUI debugger that may be part of a larger development environment
- Many windows each of which supports different aspects of debugging
- Should have a non-GUI, script-driven mode for batch testing

Basic Principles

- Debugging a program affects the program
  - Debugger is present and running
    - On cross systems this often means a debug monitor is running on the target
    - On cross systems, the debugger may be "in control" of the target. It controls interrupts, I/O, etc.
    - On native systems, the program is sharing resources with the debugger
  - Stopping a program affects timing.

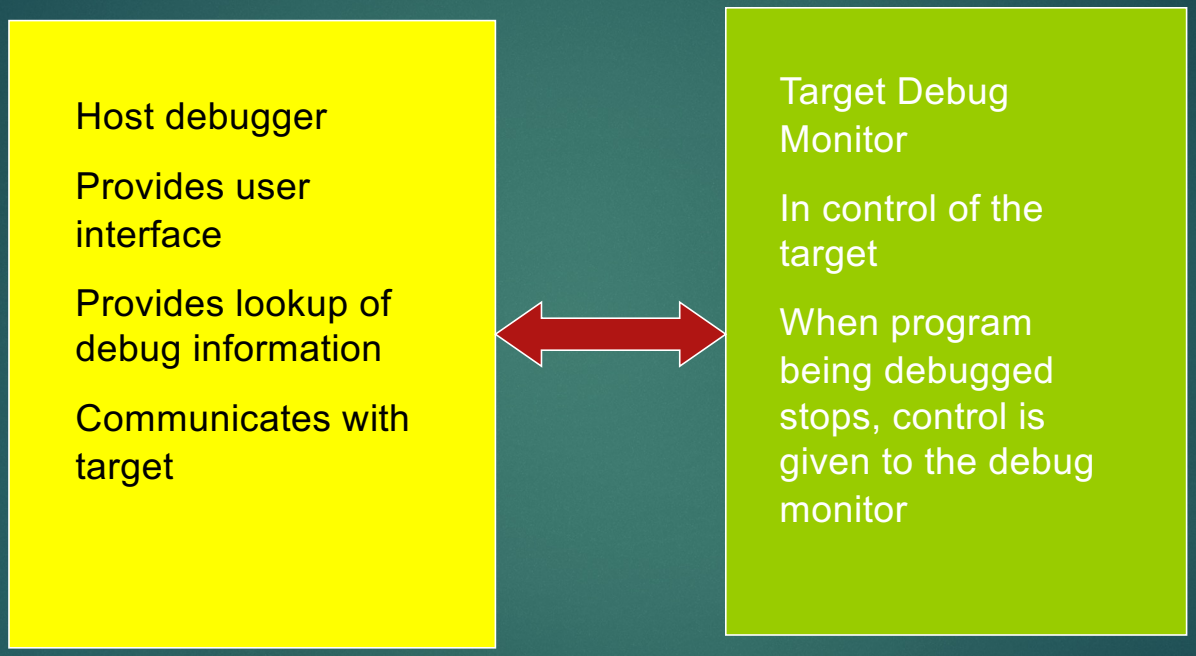
Basic Principles

- Debuggers should be truthful
  - Sounds simple, but can be very difficult, especially when compiler optimizations are involved.
  - Examples:
    - Code motion
    - Variable sometimes in register, sometimes in memory (sometimes in cache)

Basic Principles

- Context is important
  - When stopped, show associated source code
  - Provide ability to see call stack
  - Provide tracing mechanism (source and machine level)
  - When multiple threads are involved, provide information on the state of the threads.

Basic Debugger Division



Basic Requirements for Debug Monitor

- Set and delete breakpoints
  - Software breakpoints
  - Hardware breakpoints
- Single step one machine instruction
- Read and write memory
- Read and write registers

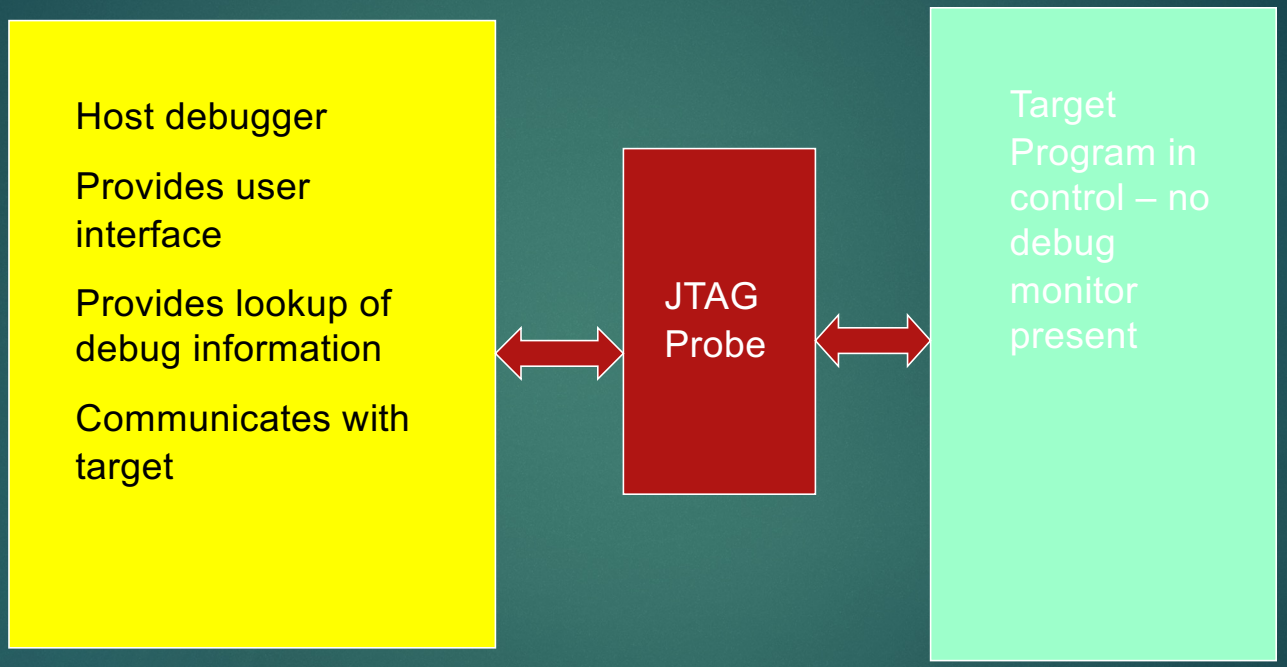
Debug Monitor Functionality

- Software breakpoint
  - Usually write some specific instruction in memory. Attempt to execute causes a trap. Trap handler gives control to D.M.
- Hardware breakpoint
  - Write an address in one of a number of special registers. When the instruction about to be executed is at an address in one of these registers, a Trap occurs.

Debug Monitor Functionality

- When trap occurs, D.M. must save context of program, so that its use of resources does not corrupt program. Probably needs to flush any data cache, invalidate instruction cache.
- Now can read/write any memory. Read/write registers by manipulating the saved context.

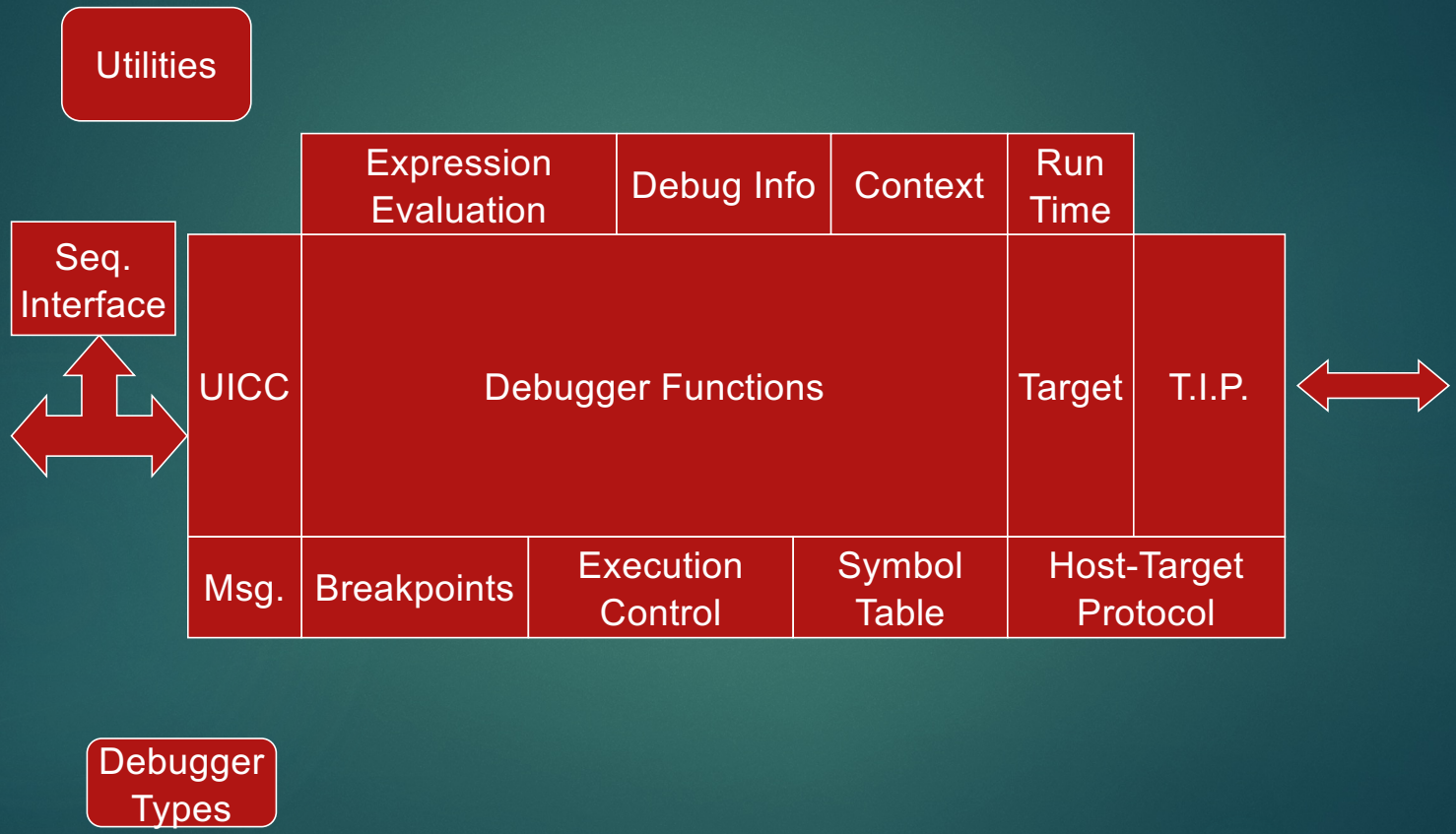
JTAG Debugging



JTAG Probe

- Provides direct access to the registers and memory of the target through the hardware. May be "smart enough" to know about cache.
- Using JTAG means program being debugged runs more normally – no debug monitor.

Host Debugger Components



Setting Breakpoints

- Find the address or addresses at which to set the breakpoint.
- Breakpoint Handler keeps track of breakpoints (inc. conditions, associated commands, etc.)
- Read memory at breakpoint address and save contents.
- Write breakpoint instruction to address

Evaluating an Expression

- Find appropriate debug information for current context.
- For each variable, look up.
  - Determine location, type info (for structs, this means finding offsets of each component)
- Read appropriate location (reg or mem) from target. Create appropriate "object"
- Perform appropriate operations on objects
- Format and display result

Modifying a Variable

- As with expression, get context, find variable, get its "shape" and location.
- If the location is not constant, create an appropriate "memory image" for the variable
- Write the memory (or register) on the target.

Showing a Call Stack

- Need to determine return address for a call
  - Can use debug information that tells where to find the return address
  - Can use architecture knowledge of how generated code is supposed to behave (if there is a standard).

Continuing from a Breakpoint

- If you continue from address, you will just trap again
- Get saved instruction for that address.
- Write that instruction to memory
- Do a machine singlestep away from the break location.
- Write the breakpoint instruction back.

Source Code SingleStep

- Find all locations where the execution can go next. (involves consulting debug information)
- Set a temporary breakpoint at address for each location.
- Go
- Delete the temporary breakpoints
- Report step completed

Other interesting actions

- Machine code tracing
- Source code tracing
- Breakpoints with conditions
- Parameters for a procedure or function
- Watching a set of registers
- Watching a number of expressions
- Watching memory

Other interesting actions

- Data access breakpoints
- Debugging code in ROM
- Supporting multiple languages

Coming Soon!

- IOT Device Presentations – schedule posted
- Oral Exams – Shedule Posted