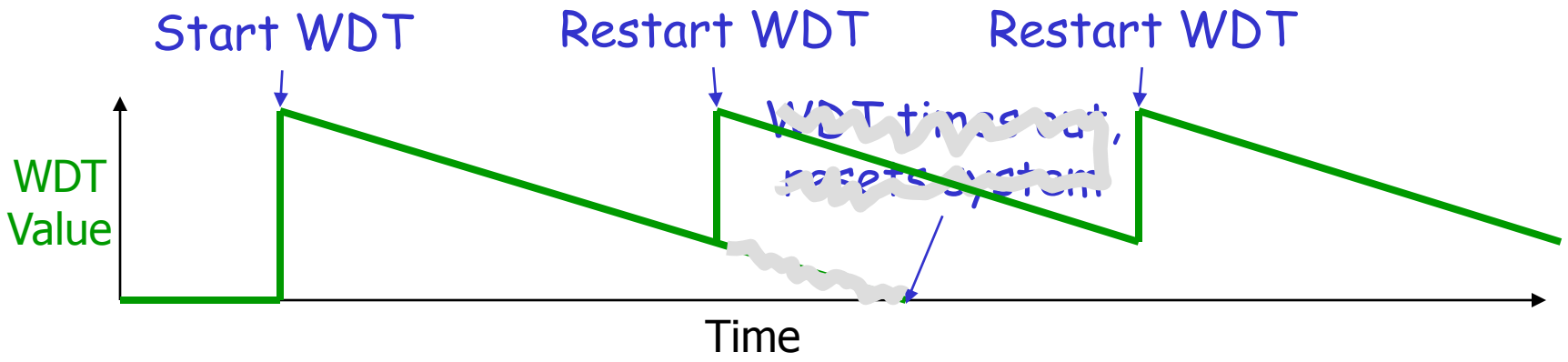
Run-Time Methods for Making Embedded Systems Robust

These lecture notes created by Mr. James B. (Jim) Carlson, NCSU

Today

- Need to make embedded systems robust
  - Implementation flaws: Code may have implementation bugs
  - Design flaws: Real world may not behave the way we expected and designed for
  - Component failures: Sometimes things break
- Run-time mechanisms for robust embedded systems
  - Watchdog timer
  - Stack-pointer monitor
  - Voltage brown-out detector

Watchdog Timer Concepts (WDT)



- Goal: detect if software is not operating correctly
- Assumption: healthy threads/tasks will periodically send a heartbeat ("I'm alive") signal
- Mechanism
  - Use heartbeat signals from tasks to restart a timer
  - If timer ever expires, the system is sick, so reset
- Typically used as a final, crude catastrophic mechanism for forcing system software back into known state

Time-Out Actions

- Simple solution: reset entire system
  - May need to explicitly toggle reset pin to ensure CPU is fully reset (rather than just jumping to reset ISR)
  - Reset should configure all I/O to safe state
- NMI Solution: generate non-maskable interrupt for debug
  - Use NMI ISR to save picture of CPU and thread state
  - Can then examine what happened with debugger or in-circuit emulator
- WDT Time-Out flag in memory
  - Set flag upon time-out before reset
  - Examine this bit in reset ISR to determine whether to boot system normally or with debug mode (without overwriting RAM)

Resetting the WDT in a Multithreaded Application

- Each periodic task updates a timestamp when it starts running
- Checker thread checks timestamp for each thread *i* to make sure it was run no more than T_i ago. If all threads are ok, restart the WDT.
- Does this detect *every possible problem?*
- Why not put it into the scheduler?

Design Suggestions for WDT

- Don’t scatter WDT reset commands throughout your code
  - There should just be one or a few such commands in the entire program
- WDT should be difficult to accidentally disable in software
- Should be able to disable WDT externally with a very obvious jumper (use to simplify debugging)
- Choose WDT period appropriately
  - Too long and system is out of control long enough to get into real trouble
  - Too short and you need to reset WDT frequently in your code (code writing and analysis overhead)

MSP430 Watchdog Timer (WDT)

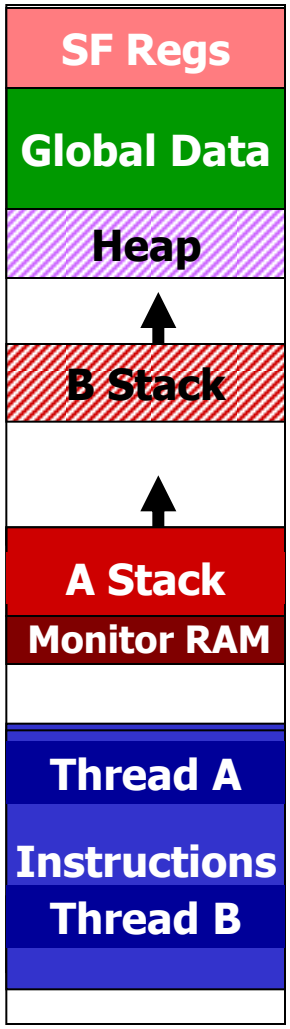
- The watchdog timer is a 32-bit timer that can be used as a watchdog or as an interval timer. The enhanced watchdog timer, WDT_A, is implemented in all devices.
- Features of the watchdog timer module include:
  - Eight software-selectable time intervals
  - Watchdog mode
  - Interval mode
  - Password-protected access to Watchdog Timer Control (WDTCTL) register
  - Selectable clock source
  - Can be stopped to conserve power
  - Clock fail-safe feature

Mechanisms for robust embedded systems

- Watchdog timer
- Stack-pointer monitor
- Brown-out detector

Stack Pointer Monitor

- What makes the stack grow?
  - Nested subroutine calls – each adds 5 bytes (3 bytes for return address, 2 bytes for dynamic link)
    - Local data in the subroutine call – automatic variables
    - Arguments passed to the subroutine
  - Nested interrupt handling – each adds 4 bytes (3 bytes for return address, 1 byte for flag register)
    - Local storage for the interrupt
- How large does the stack get?
  - Starts at 0x07FFF (top of RAM), grows to smaller addresses
  - Will overwrite heap or global data if gets too large
  - Need to allocate space for multiple stacks in system with a preemptive scheduler
  - Renesas Tool Manager provides some info in asm listing and Stack Viewer



Stack Pointer Monitoring Code

- Partial Solution
  - Examine SP periodically. If SP is below the allowable minimum (SP_LIMIT), reset the system or run a debug routine
  - Not guaranteed to detect all stack overflows, but lets us detect some.
  - *Note: some MCUs have hardware stack overflow detectors built in*

- Mechanism
  - Enhance the Timer B0 overflow interrupt to examine ISP
    - Use stc (Store Control register) instruction and asm macro to store ISP value to variable tmp_SP on stack frame (referenced from Frame Base register FB)
  - If SP is too small, do something
    - Reset system by jumping to system initialization code (at start)
    - Or start executing a debug routine. However, *there may not be enough space on the stack* to push the debug routine's activation record. May be able to use jump, inline code into the ISR, etc.
  - Setting SP_LIMIT
    - Start with beginning of RAM
    - Add in size of globals and possibly heap
    - Increase by some value for a greater margin of safety

Stack Pointer Sampling Code

- Useful during system development
  - How much space needs to be allocated for the stack?
  - Especially useful for multi-tasking systems (multiple stacks)
  - What's the cheapest MCU we can buy? (RAM costs money)
- Modified "Solution"
  - Sample SP periodically. If smaller than minimum value observed so far, save in global variable `min_obs_SP`
  - Not guaranteed to detect minimum stack size, but lets us detect common ones.
- Mechanism
  - Initialize `min_obs_SP` to value larger than expected, so first valid access will update it
  - Use ISR as before, but update `min_obs_SP` if needed rather than reset system

Issues to Consider

- Need all ISRs to re-enable interrupts to allow TimerB0 ISR to run
- This code is statistical, not absolute. It uses sampling to try to find the minimum, but is not guaranteed.
  - How long do we need to run the sampling code to have a good sense that we have captured a minimum close to the real minimum?
  - Want to make sure code is running in a wide variety of situations – including with many frequent interrupts
- What's the duration of the most-deeply-nested subroutine?
  - Might be missed if it's very short.

Mechanisms for robust embedded systems

- Watchdog timer
- Stack-pointer monitor
- Voltage brown-out detector

Voltge Brown-Out Detector

- Black-out == total loss of electricity
- Brown-out == partial loss of electricity
  - Voltage is low enough that the system is not guaranteed to work completely
  - We can't guarantee that it won't do anything at all. Parts may still work.
    - "CPU runs, except for when trying to do multiplies"
- Want to detect brown-out automatically
  - Possibly save critical processor information to allow warm boot
  - Then hold processor in reset state until brown-out ends