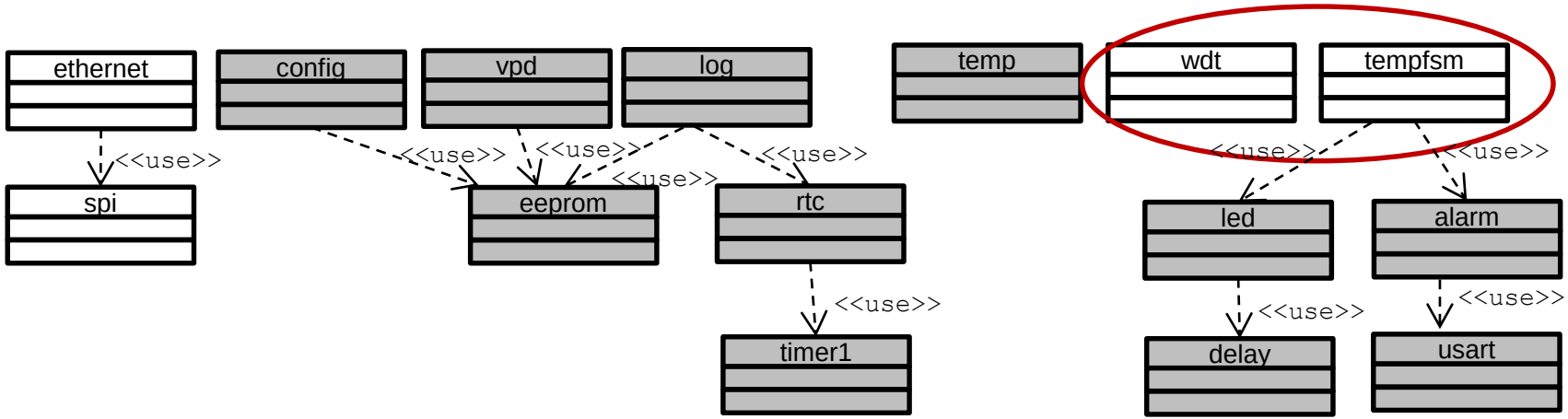
SER486: Embedded C Programming

Design Specification

Programming Project 3

Integration, Watchdog Timer, Hysteresis

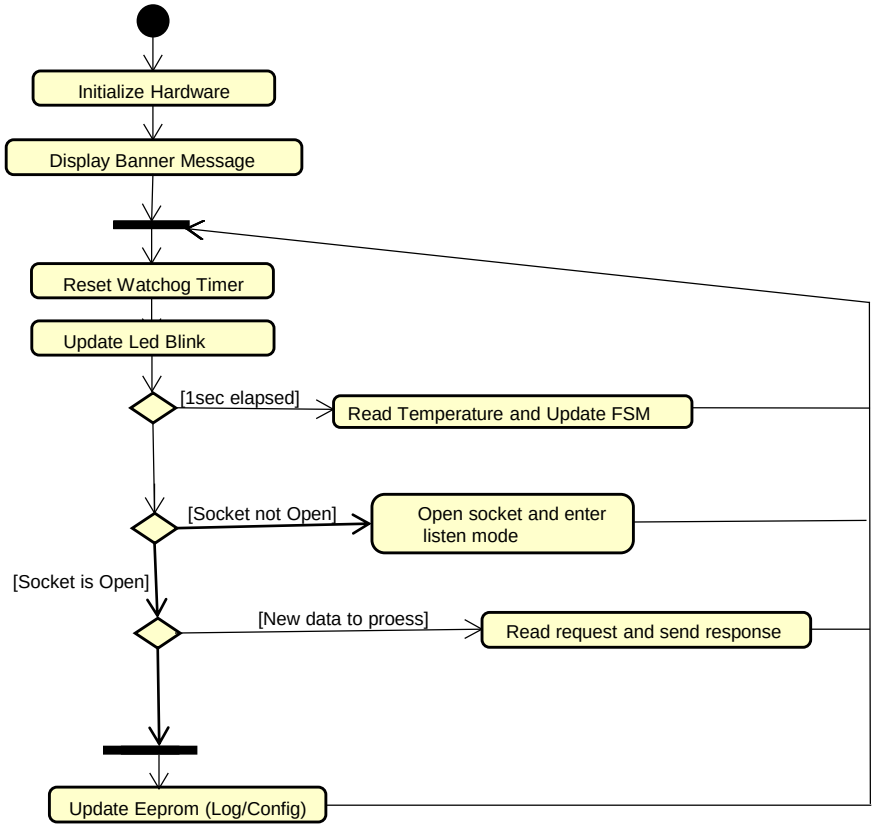
Device Class Diagram



This project focuses on the circled classes.

Classes shown in grey have been completed already and are part of the project 3 library code

Pulling it all together: Main code



WDT class

wdt
+init() : void +reset () : void +force_restart() : void -__vector_6()

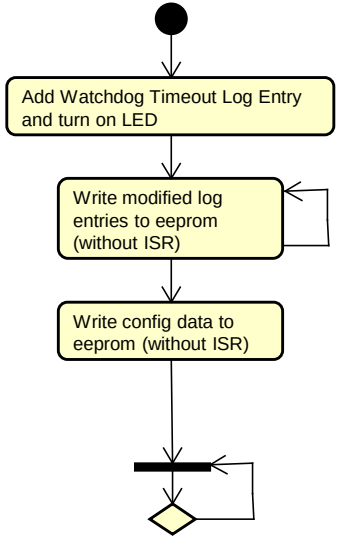
The wdt class implements a hardware device design pattern to encapsulate the ATMEGA 328P watchdog timer. Upon initialization, the timer is set for a 2 second timeout period that first generates an interrupt. A second timeout will generate a system reset. Other member functions of this class support resetting the timer, and utilizing the timer to force a system reset. More information on this watchdog timer device can be found in the ATMEGA 328P datasheet.

Internal State

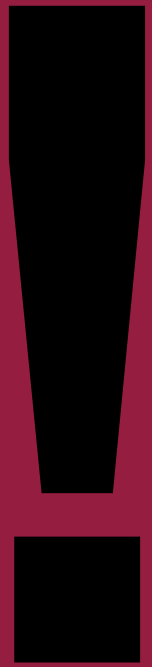
Member Functions

- **init()** – This function initializes the watchdog timer for a timeout period of 2 seconds. Upon timeout, the watchdog will first generate an interrupt, followed by a system reset. Once init is called, wdt.reset() must be called at least once every two seconds in order to avoid a watchdog timeout.
- **reset()** – resets the watchdog timer to prevent a timeout.
- **force_restart()**– When invoked, this function disables the watchdog interrupt and waits for the watchdog to timeout (generating a system reset).
- **void __vector_6()** – This function is enabled when the watchdog is initialized. When invoked, the function turns on the LED, adds a EVENT_WDT to the system event log and attempts to write back any modified log and configuration information before the second stage of the watchdog timer forces a system reset.

Conceptual WDT ISR behavior

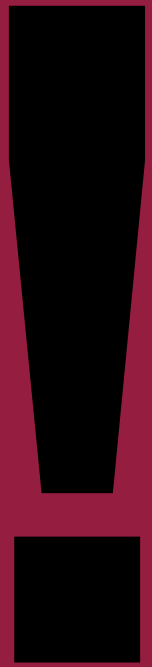


Use log.update_noisr(), and config.update_noisr() to write back caches in the watchdog ISR. Log writeback may require up to 16 calls to flush the cache



Watchdog timer initialization has a time-dependent sequence.

Use pragmas to force optimization to -Os around this sequence



Use __builtin_avr_wdr() to reset the watchdog timer

TempFSM Class

```
tempfsm
+int()
+reset()
+update(int tcurrent,int tcrit_h,int
twarn_h,int tcrit_l,int twarn_l)
```

This class implements a finite state machine that provides hysteresis for the device temperature readings.

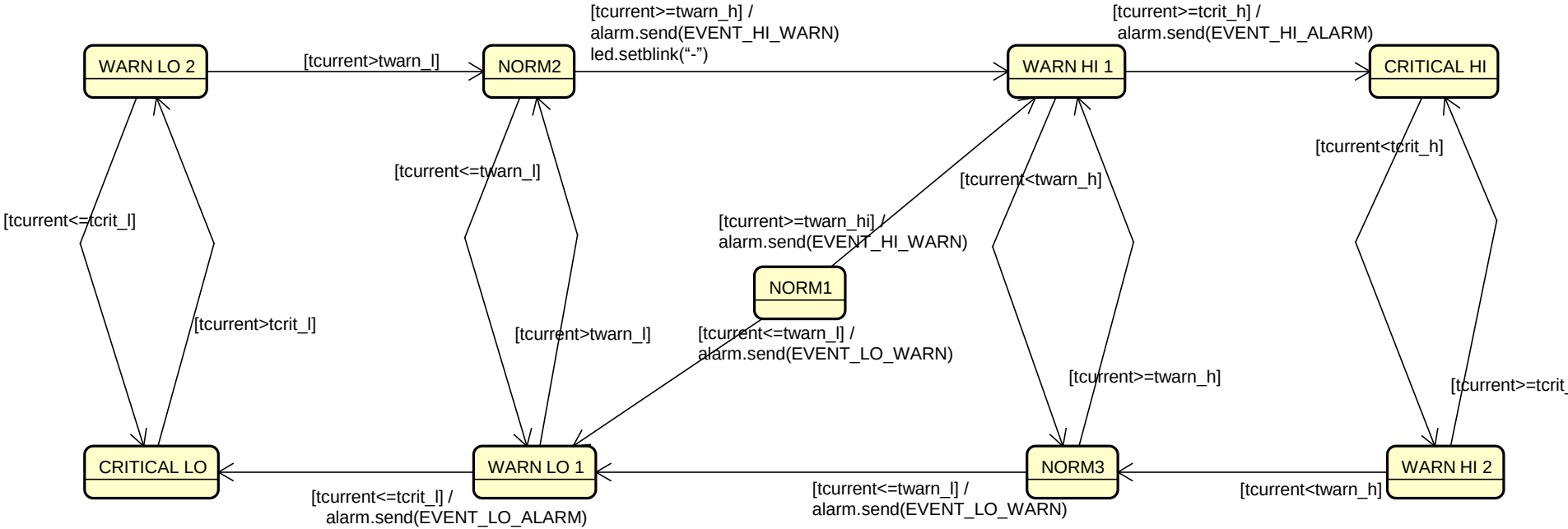
Internal State

- **Implementation specific – up to the student to define**

Member Functions

- **init() –** perform any class-specific initialization, including setting the state's initial value;
- **reset() –** reset the finite state machine's state to NORM1.
- **update()** – updates the state of the fsm based on the provided current temperature and temperature limit arguments. Upon state transitions, LED blink behavior will be updated (fast blink for critical, slow blink for warning). Alarms must also be sent according to the state diagram using the instructor-provided alarm class functions.
  - tcurrent = the device's current temperature
  - tcrit_h = the critical high temperature limit for the device
  - twarn_h = the warning high temperature limit for the device
  - tcrit_l = the critical low temperature limit for the device
  - twarn_l = the warning low temperature limit for the device

TEMPFSM State Diagram



- All state transitions are triggered by 1 second timeout
- On exit, LED should be set to "." if the next state is a critical state, "-" if the next state is a warning state, and " " for others
- Use the alarm class to send the appropriate alarm event code. Event codes are found in log.h

Utility functions (util.c / process_packet.h)

int update_tcrit_hi(int value):int

Update the configuration tcrit_hi limit with the specified value. This function is called by the packet command parser. It should perform range checking on the value. Valid inputs for tcrit_hi must be greater than twarn_hi but less than or equal to 0x3FF. ***Returns 0 if there is no error***, otherwise, returns 1;

int update_twarn_hi(int value):int

Update the configuration twarn_hi limit with the specified value. This function is called by the packet command parser. It should perform range checking on the value. Valid inputs for twarn_hi must be greater than twarn_lo and less than tcrit_hi. ***Returns 0 if there is no error,*** otherwise, returns 1;

update_twarn_lo(int value):int

Update the configuration twarn_lo limit with the specified value. This function is called by the packet command parser. It should perform range checking on the value. Valid inputs for twarn_lo must be greater than tcrit_lo and less than twarn_hi. ***Returns 0 if there is no error***, otherwise, returns 1;

update_tcirt_lo(int value):int

Update the configuration tcrit_lo limit with the specified value. This function is called by the packet command parser. ***It should perform range checking on the value.*** Valid inputs for tcrit_lo must be less than twarn_lo. Returns 0 if there is no error, otherwise, returns 1;

Project 3 Command Syntax

GET /device

- Displays a JSON representation of the device state on the console

PUT /device/config?tcrit_hi=<value>

PUT /device/config?tcrit_lo=<value>

PUT /device/config?twarn_hi=<value>

PUT /device/config?twarn_lo=<value>

- Writes a new value of the associated configuration parameter

DELETE /device/log

- Deletes all log entries

PUT /device?reset="true"

- Force a device reset

Use control-C in the terminal window to "connect" to the device.
During normal operation, alarm outputs will be sent to the terminal window

Project 3 Response format - GET


```
{
  "vpd":{
    "model":"Sandy",
    "manufacturer":"Douglas",
    "serial_number":"_UNASSIGNED",
    "manufacture_date":"01/01/2000",
    "mac_address":"44:4F:55:53:41:4E",
    "country_code":"USA"
  }
  "tcrit_hi":1023,
  "twarn_hi":1022,
  "tcrit_lo":0,
  "twarn_lo":1,
  "temperature":88,
  "state":CRIT_HI,
  "log":[
    {
      "timestamp":"01/01/2000 00:00:00","eventnum":0
    }
   ]
}
```


/* BrainFuck.c
* BrainFuck Interpreter */
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
int main(int argc, char **argv)
{
int memory[32768], space[32768];
int fd, len, buf, spointer, mpointer;
int kl = 0;
if (argc <= 1)
{
printf("Usage: %s <brainfuck file>\n", argv[0]);
exit(1);
}
/* Open brainfuck file */
if ((fd = open(argv[1], O_RDONLY)) < 0) exit(1);
/* Read the file byte by byte */
len = spointer = 1;
while (len > 0)
{
len = read(fd, &buf, 1);
space[spointer] = buf;
spointer++;
}
close(fd);
for (mpointer = 0; mpointer < 32768; mpointer++) memory[mpointer] = 0;
len = spointer;
spointer = mpointer = 0;
for (spointer = 0; spointer < len; spointer++)
{
switch(space[spointer])
{
// Increment pointer value
case '+':
memory[mpointer]++;
break;
// Decrement pointer value
case '-':
memory[mpointer]--;
break;
// Increment pointer
case '>':
mpointer++;
break;
// Decrement pointer
case '<':
mpointer--;
break;
// Print current pointer value
case '.':
putchar(memory[mpointer]);
break;
// Read value and store in current pointer
case ',':
memory[mpointer] = getchar();
break;
// Start loop
case '[':
if (memory[mpointer] == 0)
{
/* Find matching] */
spointer++;
while (kl > 0 || space[spointer] != ']')
{
if (space[spointer] == '[') kl++;
if (space[spointer] == ']') kl--;
// Go in right direction
spointer++;
}
}
break;
// End Loop
case ']':
if (memory[mpointer] != 0)
{
/* Find matching [*/
spointer--;
while (kl > 0 || space[spointer] != '[')
{
if (space[spointer] == ']') kl++;
if (space[spointer] == '[') kl--;
// Go left
spointer--;
}
spointer--;
}
break;
}
}
putchar('\n');
return (0);
}

[END]