Class 16: AppleFanning

Action Items

Problem Set 3 is due Monday, 28 October.

Everyone should have received an email with their midterm results (and other recorded grades so far). If you didn't get it, send me email (evans@cs.virginia.edu). If you already sent me email but didn't get a response yet, please let me know after class today.

Project

4 Nov: Due: Project Proposals (more details posted soon)

18 Nov: Due: Project Design Reviews

5 Dec: Due: Project Demos

Do something that is **fun** (for you to do, and others to see), **relevant** (to the class), **technically interesting** (to you and me), and **useful** (at least to you, hopefully to many). You probably can't maximize all of these! It is okay to sacrifice one or two of them to increase others. A good project should be strong on at least 2 of these, which is much better than being mediocre of all four.

AppleFan

From http://opensource.apple.com/source/AppleFan/AppleFan-110.3.1/AppleFan.cpp:

AppleFan.cpp [plain text]

```
/*
 * Copyright (c) 1998-2000 Apple Computer, Inc. All rights reserved.
 *
 * @APPLE_LICENSE_HEADER_START@
 *
 * The contents of this file constitute Original Code as defined in and
 * are subject to the Apple Public Source License Version 1.1 (the
 * "License").  You may not use this file except in compliance with the
 * License.  Please obtain a copy of the License at
 * http://www.apple.com/publicsource and read it before using this file.
 *
 * This Original Code and all software distributed under the License are
 * distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, EITHER
 * EXPRESS OR IMPLIED, AND APPLE HEREBY DISCLAIMS ALL SUCH WARRANTIES,
 * INCLUDING WITHOUT LIMITATION, ANY WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT.  Please see the
 * License for the specific language governing rights and limitations
 * under the License.
 *
 * @APPLE_LICENSE_HEADER_END@
```

```
 */
/*
 * Copyright (c) 2002 Apple Computer, Inc.  All rights reserved.
 *
 */

#include <IOKit/pwr_mgt/RootDomain.h>
#include <IOKit/IOMessage.h>
#include <mach/clock_types.h>
#include "AppleFan.h"

/*
 * Default Parameters
 *
 * First look for defaults in the personality, otherwise we fall back to these
 * hard coded ones.
 *
 * Speed Table: Linear Ramp, Minimum 57C, Maximum 62C
 *
 * Note: These temperatures are expressed in 8.8 fixed point values.
 */
static fan_speed_table_t gDefaultSpeedTable =
  { 0x3900, 0x3A4A, 0x3Ad3, 0x3B3C,
    0x3B94, 0x3BE3, 0x3C29, 0x3C6A,
    0x3CA6, 0x3CD7, 0x3D15, 0x3D48,
    0x3D78, 0x3DA7, 0x3DD4, 0x3E00 };

/*
 * Hysteresis Temperature 55 C
 */
static SInt16 gDefaultHysteresisTemp = 0x3700;

/*
 * Polling Period (seconds)
 */
static UInt64 gDefaultPollingPeriod = 8;

/*
 * Speedup Delay (seconds)
 */
static UInt64 gDefaultSpeedupDelay = 8;

...

void AppleFan::free(void)
{
  if (pollingPeriodKey) pollingPeriodKey->release();
  if (speedTableKey) speedTableKey->release();
  if (speedupDelayKey) speedupDelayKey->release();
  if (slowdownDelayKey) slowdownDelayKey->release();
  if (hysteresisTempKey) hysteresisTempKey->release();
  if (getTempSymbol) getTempSymbol->release();

#ifdef APPLEFAN_DEBUG
```

```
    if (currentSpeedKey) currentSpeedKey->release();
    if (currentCPUTempKey) currentCPUTempKey->release();
    if (forceUpdateKey) forceUpdateKey->release();
#endif

    super::free();
 }

 ...

 bool AppleFan::start(IOService *provider)
 {
    OSData*tmp_osdata;
    UInt32*tmp_uint32;
    IOService*tmp_svc;
    const OSSymbol*uninI2C;
    mach_timespec_t WaitTimeOut;

    // We have two power states - off and on
    static const IOPMPowerState powerStates[2] = {
      { 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
      { 1, IOPMDeviceUsable, IOPMPowerOn, IOPMPowerOn, 0, 0, 0, 0, 0, 0, 0, 0 }
    };

    // Use a 30 second timeout when calling waitForService()
    WaitTimeOut.tv_sec = 30;
    WaitTimeOut.tv_nsec = 0;

    if (!super::start(provider)) return(false);

    // get my I2C address from the device tree
    tmp_osdata = OSDynamicCast(OSData, provider->getProperty("reg"));
    if (!tmp_osdata)
      {
        return(false);
      }

    tmp_uint32 = (UInt32 *)tmp_osdata->getBytesNoCopy();
    fI2CBus = (UInt8)(*tmp_uint32 >> 8);
    fI2CAddr = (UInt8)*tmp_uint32;
    fI2CAddr >>= 1;// right shift by one to make a 7-bit address

    DLOG("@AppleFan::start fI2CBus=%02x fI2CAddr=%02x\n",
     fI2CBus, fI2CAddr);

    // find the UniN I2C driver
    uninI2C = OSSymbol::withCStringNoCopy("PPCI2CInterface.i2c-uni-n");
    tmp_svc = waitForService(resourceMatching(uninI2C), &WaitTimeOut);
    if (tmp_svc)
      {
        I2C_iface = (PPCI2CInterface *)tmp_svc->getProperty(uninI2C);
      }

    if (uninI2C) uninI2C->release();
```

```
  ...
  // I2C_iface is initialized to 0 so if it is not set here, we didn't find I2C
  PMinit();
  provider->joinPMtree(this);
  registerPowerDriver(this, (IOPMPowerState *)powerStates, 2);

  // Get restart and shutdown events too
  registerPrioritySleepWakeInterest(sPMNotify, this, NULL);

  // Register with I/O Kit matching engine
  registerService();

  // do the initial update.  this sets a timeout as its last step and
  // begins the fan control in motion.
  doUpdate(true);

  DLOG("-AppleFan::start\n");
  return(true);
}

... // ~100 lines removed
          IOLog("AppleFan::setProperties SOMETHING IS VERY WRONG!!!");
...

/*****************************************************************************
doUpdate() is where we read the CPU temp and look it up in the speed table to
choose a fan speed.  Also here is the timer callback routine which repeatedly
calls doUpdate().
*****************************************************************************/

void AppleFan::doUpdate(bool first)
{
  AbsoluteTime interval;
  UInt8 newSpeed;
  SInt16 cpu_temp;

  DLOG("+AppleFan::doUpdate\n");

  // Get temperature data
  if (!getCPUTemp(&cpu_temp))
    {
      IOLog("AppleFan::doUpdate ERROR FETCHING CPU TEMP!!!\n");
      restoreADM1030State(&fSavedRegs);
      terminate();
      return;
    }

  // look up the fan speed
  newSpeed = 0;
  while ((cpu_temp >= fSpeedTable[newSpeed]) && (newSpeed < (kNumFanSpeeds - 1)))
    newSpeed++;

  // set fan speed cfg register
```

```
  setFanSpeed(newSpeed, cpu_temp, first);

  // implement a periodic timer
  if (first) clock_get_uptime(&fWakeTime);

  nanoseconds_to_absolutetime(fPollingPeriod, &interval);
  ADD_ABSOLUTETIME(&fWakeTime, &interval);

  thread_call_enter_delayed( timerCallout, fWakeTime );

  DLOG("-AppleFan::doUpdate\n");
}

/*****************************************************************************
Routines which take a fan speed as input and program the ADM1030 to run at the
desired speed.  Some nasty tricks are in this code, but it is pretty well
encapsulated and explained in comments...

The routines are setFanSpeed and setADM1030SpeedMagically (lots of comments in
the latter for obvious reasons...)
*****************************************************************************/
void AppleFan::setFanSpeed(UInt8 speed, SInt16 cpu_temp, bool first)
{
  UInt8 desiredSpeed;
  SInt16 rmt_temp;
  AbsoluteTime ticksPassed;
  UInt64 nsecPassed;

  if (!getRemoteTemp(&rmt_temp))
    {
      IOLog("AppleFan::setFanSpeed FATAL ERROR FETCHING REMOTE CHANNEL TEMP!!!\n");
      restoreADM1030State(&fSavedRegs);
      terminate();
      return;
    }

  if (first)
    {
      // If this is the first run, don't apply any of the hysteresis mechanisms,
      // just program the chip with the speed that was produced from the table
      // lookup
      DLOG("@AppleFan::setFanSpeed initial speed is %u\n", speed);
      setADM1030SpeedMagically(speed, rmt_temp);
      clock_get_uptime(&fLastTransition);
    }
  else
    {
      if (speed == fLastFanSpeed)
    {
      if (rmt_temp != fLastRmtTemp)
        {
          // need to update the remote temp limit register
          DLOG("@AppleFan::setFanSpeed environmental update\n");
          setADM1030SpeedMagically(speed, rmt_temp);
```

```
        return;
      }

    DLOG("@AppleFan::setFanSpeed no update needed\n");
  }
    else
  {
    // calculate nanoseconds since last speed change
    clock_get_uptime(&ticksPassed);
    SUB_ABSOLUTETIME(&ticksPassed, &fLastTransition);
    absolutetime_to_nanoseconds(ticksPassed, &nsecPassed);

    if (speed < fLastFanSpeed)
      {
        // Hysteresis mechanism - don't turn off the fan unless we've reached
        // the hysteresis temp
        if (speed == kDutyCycleOff && fLastFanSpeed == kDutyCycle07)
      {
        DLOG("@AppleFan::setFanSpeed hysteresis check cpu_temp 0x%04x fHysteresisTemp %04x\n",
         cpu_temp, fHysteresisTemp);

        if (cpu_temp > fHysteresisTemp)
          {
            DLOG("@AppleFan::setFanSpeed hysteresis active\n");

            // do an environmental update if needed
            if (rmt_temp != fLastRmtTemp)
          setADM1030SpeedMagicall}
      }
        else if (speed > fLastFanSpeed)
      {
        DLOG("@AppleFan::setFanSpeed upward check nsecPassed 0x%llX fSpeedupDelay 0x%llX\n",
         nsecPassed, fSpeedupDelay);

        // apply upward hysteresis

        if (nsecPassed > fSpeedupDelay)
          {
            desiredSpeed = fLastFanSpeed + 1;
            DLOG("@AppleFan::setFanSpeed speedup to %u\n", desiredSpeed);
            setADM1030SpeedMagically(desiredSpeed, rmt_temp);
            clock_get_uptime(&fLastTransition);
          }
        else
          {
            DLOG("@AppleFan::setFanSpeed speedup delay active\n");

            // do an environmental update if needed
            if (rmt_temp != fLastRmtTemp)
          setADM1030SpeedMagically(fLastFanSpeed, rmt_temp);
          }
      }
        else { /* not reached */ }
      }
```

```
  }
  }


void AppleFan::setADM1030SpeedMagically(UInt8 desiredSpeed, SInt16 rmt_temp)
{
  UInt8 TminTrange, speed;

  // shift rmt_temp(14:10) into TminTrange(7:3)
  TminTrange = (UInt8)(rmt_temp >> 7);
  TminTrange &= ~kTrangeMask;// clear out the 3 LSBs
  TminTrange |= 0x7;// T_range = highest possible

  /*
   * setFanSpeed() calculates a speed between 0x0 and 0xF and passes it
   * into this routine (in the variable named "speed").  setFanSpeed
   * is responsible for setting the remote T_min/T_range and the speed
   * config register to make the PWM match the requested speed.
   *
   * If we want the PWM to be completely inactive, we have to set
   * Tmin ABOVE the current remote temp.  We set the speed config
   * register to zero in this case.
   *
   * For other PWM values, we program Tmin to a value just below the
   * current remote temp.  This will instruct the ADM1030 to operate
   * the PWM at a speed just above whatever speed is programmed into
   * the speed config register (which, in automatic control mode,
   * sets the minimum speed at which the fan runs when the current
   * remote temp exceeds Tmin).  Then, we program the speed config
   * register with speed - 1 ; that is, one less than the value that
   * was passed in by doUpdate().  It may seem less than intuitive
   * to program the speed config register with 0 when we want speed
   * 1, just remember that the difference is made by Tmin -- is the
   * fan operating below the linear range (PWM completely inactive),
   * or just inside the linear range (PWM active)?
   */

  // The first "if" clause handles two cases:
  //
  // 1.  The fan is already set below the linear range.  This is
  //     when speed=0 and fLastFanSpeed=0.  We program Tmin 8 degrees
  //     above rmt_temp, and preserve the speed config reg at 0.
  //
  // 2.  The fan is currently in the linear range, but we are about
  //     to shift below the linear range and shut off PWM entirely.
  //     This is denoted by speed=0 and fLastFanSpeed=1.
  if (desiredSpeed == 0)
    {
  // Transition from 1 to 0 takes fan out of linear range
  TminTrange += 0x10;// raise Tmin above current rmt_temp
  speed = desiredSpeed;
  fLastFanSpeed = desiredSpeed;
    }
  // Put the hw control loop into the linear range
```

```
      else
        {
      TminTrange -= 0x08;
      speed = desiredSpeed - 1;
      fLastFanSpeed = desiredSpeed;
        }

      // Recored the rmt_temp at the time of this update
      fLastRmtTemp = rmt_temp;

#ifdef APPLEFAN_DEBUG
      char debug[16];
      temp2str(rmt_temp, debug);
#endif

      DLOG("@AppleFan::setADM1030SpeedMagically speed=%u rmt_temp=%x (%sC) TminTrange=%x\n",
       speed, rmt_temp, debug, TminTrange);

      if (!doI2COpen())
        {
      IOLog("AppleFan failed to open bus for setting fan speed\n");
      return;
        }

      if (!doI2CWrite(kRmtTminTrange, &TminTrange, 1))
        {
      doI2CClose();
      IOLog("AppleFan failed to write to T_min/T_range register!\n");
      return;
        }

      if (!doI2CWrite(kSpeedCfgReg, &speed, 1))
        {
      doI2CClose();
      IOLog("AppleFan failed to write to fan speed register\n");
      return;
        }

      doI2CClose();
    }
```