Instituto Tecnológico y de Estudios Superiores de Occidente

Recognition of official validity of higher education studies as set forth in ministerial agreement 15018, published in the Official Journal of the Federation on November 29, 1976.

Department of Electronics, Systems and Informatics

Master's Degree in Electronic Design

ITESO, Jesuit University of Guadalajara

Pluggable Fan Controller

DEGREE-EARNING PROJECT that, in order to earn the **DEGREE** of **MASTER IN INDUSTRIAL ELECTRONIC**

Is submitted by: **CARLOS LUIS BERNAL QUINTERO**

Tutor: **Dr. Omar Longoria**

Tlaquepaque, Jalisco. October 2023.

Dedication

To my family for being my inspiration and the motor to keep me challenging myself to be a better person day-by-day.

To my wife for always being supportive and emphatic, giving me the time needed to learn new things and keep progressing in my professional life and conquer my goals no matter how far they seem to be.

To all the people that was involved and supported the with the implementation of this idea showing trust in me and my proposals.

To my parents for showing me that hard work leads to great achievements.

Summary

This document covers the implementation of a microcontroller based, low-cost, configurable, and reusable fan speed controller used in multiple segments: Client, Server, Networking, and Graphics, used to:

1) *Reduce the noise in validation laboratories and provide automated fan speed control during the early stages of the new silicon devices test,*
2) *Minimize the need for special equipment to cover temperature range tests and,*
3) *Enable the reuse of heatsinks among the different variants of a product.*

The solution has been used in different programs on multiple Intel sites across the globe, demonstrating its capability not only to reduce the audible noise levels in our labs, but also to cover temperature tests around 60ºC without the need for a thermal head.

Keywords: *CPU Thermal management, Fan Speed Control, Fan Noise Reduction, Embedded System, Heatsink reuse*

Content

Introduction

As part of new CPU validation and verification methodologies, testing under different thermal environments is critical to guarantee that the product meets the temperature ranges required by customers' applications.

Intel's Thermal management teams design heatsinks to cover the worst-case usage scenario, but Original Equipment Manufacturers (OEM) and Original Design Manufacturers (ODM) use solutions closer to their standard usage models to reduce the overall solution cost.

The delta between the heat dissipation on both solutions had led to bugs escapes that are detected by Intel's customers when working under different thermal conditions. Those escapes affect the product's reliability and brand's reputation.

The Intel Thermal design teams cannot afford the engineering investment to have different thermal solutions for a single product, so a solution is needed.

There are off-the-shelf solutions to those problems, but none of the available solutions cover all of Intel's needs in a single solution.

This document presents the implemented solution that is now being used across Intel Validation Labs around the world:

1) Reducing noise levels,

2) minimizing the need for special equipment to cover temperature range tests and,
3) Successfully enabling heatsink reuse across different variants of a product.

The document is divided into five chapters, providing details on the problem statement and motivation in Chapter 1, describing the solution in Chapter 2, and diving into a detailed walk-through of the hardware and firmware implementations. Chapter 3 provides evidence on the test and validation procedures that supported the system verification, then jumping to the discussion of the results in Chapter 4, before presenting our conclusions in Chapter 5.

1. Background

1.1.1 Products Temperature Range Validation

Intel System-on-Chips (SoCs) and CPUs are used in a wide block of applications and environments so, as part of their usage spec verification, it is necessary to validate and verify their proper functionality over a very wide operating temperature range.

Depending on the end segment, that operating temperature range is between 0ºC to 95ºC or -40ºC to 125ºC (IoT).

To help cover that wide temperature operation range, the Thermal solution uses heat sinks and fans to help move the heat from the Device-Under-Test (DUT).

Depending on the segment and the board implementation, the number of fans in a system can be around 8 different fans. Figure 1 shows a system with a fan for each CPU (dual socket systems) and 2 for each memory group (2 per CPU side) for a total of 6 fans in a single system.

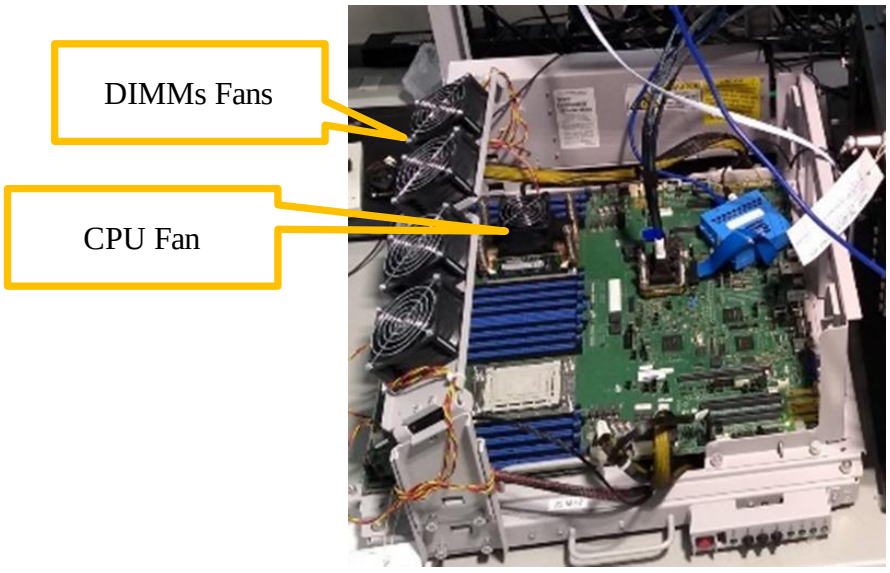

Figure 1 - Server Validation Platform.

To support the extreme temperature ranges, the validation teams use expensive and specialized equipment called thermal heads to heat-up or cool-down the DUT.

Since the Intel thermal design teams need to cover all the temperature ranges, the solution differs from the solution that OEMs implemented.

Thermal heads are used, in some cases, to mimic the behavior of the OEM thermal solution. That implementation is expensive and in some cases the use of that external heating element is not loyal to the behavior observed when the part heats up itself.

In some cases, that difference had led to silicon bugs escapes harming the product's reputation and brand's image.

1.1.2 Validation Labs Support

Validation labs around the world host several systems running multiple tests at a time. Due to the noise that the fans in the systems generate, having all those systems working at the same time and in the same place in a closed area, generates noise levels above the healthy allowed margins and noise levels are above the working environment recommendations permanent hearing damage can be generated.

A lab noise level map is show in Figure 2 and as we will see, there are several areas with noise levels above 80dBs.

Figure 2 – Intel's Guadalajara Design Center validation lab Noise Level Mapping

In Mexico, NOM-011-STPS-2001[1] specifies the allowed noise levels and the time of exposure to avoid permanent hearing damage.

Figure 3 and Figure 4 show some examples related to noise levels and the permissible exposure time to avoid permanent hearing damage to the people exposed to those environments.

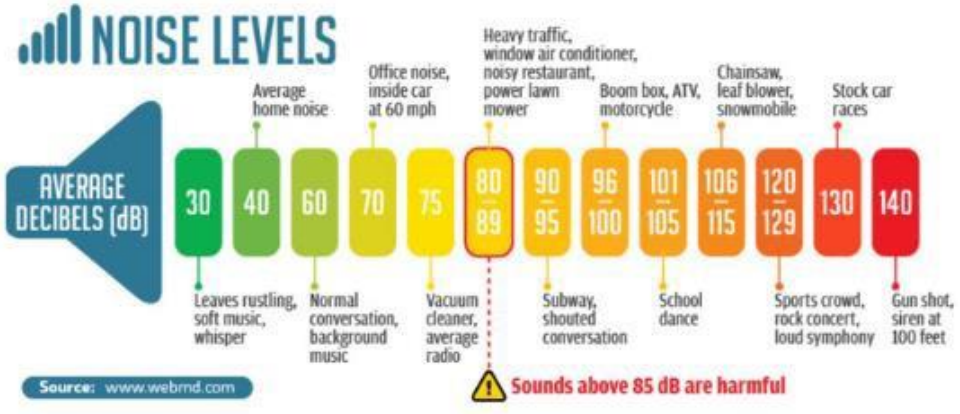

Figure 3 - Audible Noise Levels Examples

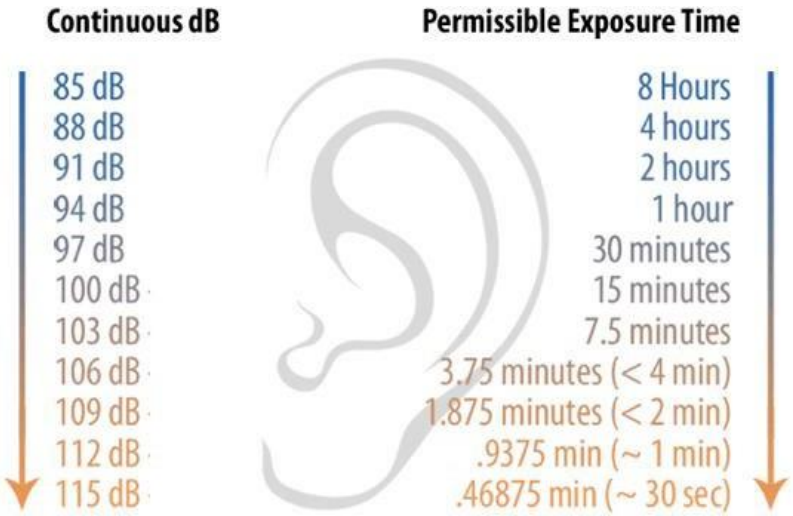

Figure 4 - Allowed Noise level exposure duration to prevent hearing damage.

1.1.3 Thermal Solution Design

Now, talking about the heat dissipation solution design cost, the Non-recurrent Engineering Cost (NRE) is very high. For instance, in the client segment, there are different Stock-Keeping

Units (SKUs). Among other factors, the difference between those SKUs is the power dissipation so we can have a very wide range of Power dissipation solutions – heatsinks.

Figure 5 shows an example of a heatsink design for a Server platform.

Figure 5 – Server CPU Heatsink solution example.

Using Skylake for client systems, Table 1 provided by [2] shows the thermal Design Power (TDP) for each SKU.

Table 1 - Client CPU SKU's Power

Model	Core	Clock Speed	Thermal Design Power (TDP)
Core i5-6200U	Skylake (14 nm)	2.3 GHz [Turbo 2.8 GHz]	15 W
Core i5-6260U	Skylake (14 nm)	1.8 GHz [Turbo 2.9 GHz]	15 W
Core i5-6267U	Skylake (14 nm)	2.9 GHz [Turbo 3.3 GHz]	28 W
Core i5-6287U	Skylake (14 nm)	3.1 GHz [Turbo 3.5 GHz]	28 W
Core i5-6300U	Skylake (14 nm)	2.4 GHz [Turbo 3 GHz]	15 W
Core i5-6360U	Skylake (14 nm)	2 GHz [Turbo 3.1 GHz]	15 W
Core i5-6400T	Skylake (14 nm)	2.2 GHz [Turbo 2.8 GHz]	35 W
Core i5-6500T	Skylake (14 nm)	2.5 GHz [Turbo 3.1 GHz]	35 W
Core i5-6300HQ	Skylake (14 nm)	2.3 GHz [Turbo 3.2 GHz]	45 W
Core i5-6300HQ	Skylake (14 nm)	2.3 GHz [Turbo 3.2 GHz]	45 W
Core i5-6350HQ	Skylake (14 nm)	2.3 GHz [Turbo 3.2 GHz]	45 W
Core i5-6440HQ	Skylake (14 nm)	2.6 GHz [Turbo 3.5 GHz]	45 W
Core i5-6600T	Skylake (14 nm)	2.7 GHz [Turbo 3.5 GHz]	35 W
Core i5-6600K	Skylake (14 nm)	3.5 GHz [Turbo 3.9 GHz]	91 W

In that case, the Thermal Design team provides a different heatsink for each SKU. Each design

Reducing the number of heatsinks designed to cover the different Thermal Design Power (TDP) solutions provides a great opportunity to save engineering resources, time, and money to Intel.

In summary, completing a Silicon product verification requires several tests, and a great part of those tests needs to be performed over different temperature ranges.

Intel Validation Laboratories work with several systems in parallel so they can execute different tests at the same time and reduce the validation time.

All those systems running different tests at the same time generate very high noise levels and require the acquisition of the corresponding heatsinks.

1.1.4 Existing Fan Control Solutions

There are several Fan control solutions in the market that cover the mentioned challenges but none of them cover all the options at the same time.

There are low-cost systems like Occkic's Fan speed controller shown in Figure 6, that allow the user to set the fan speed using pushbuttons, dipswitches like Great ZT in Figure 7, or potentiometers (Figure 8).

Not all of them provide close loop control, serial output for data logging or different operation modes.

Figure 6 - Occkic's Fan Speed controller

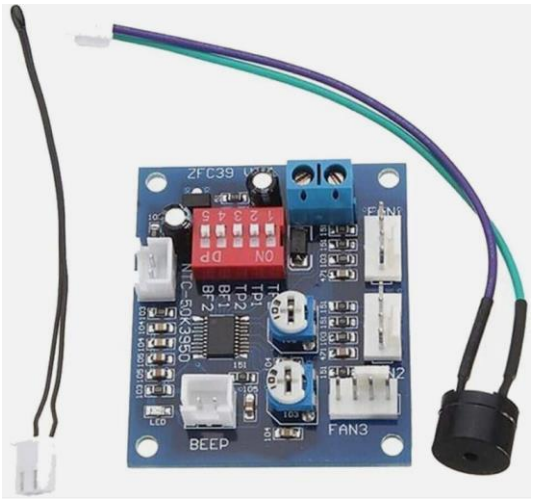

Figure 7 – Great ZT's Fan Controller

Figure 8 - Noctua NA-FC1Controller

The systems that provide closed-loop control work to keep the DUT cool and don't have a way to force a specific fan speed or to keep the DUT warm.

1.2. Problem Statement

Intel needs a solution that:

1) helps to reduce audible noise generated by fans running in the systems.
2) allow the reuse of the heatsink solution across SKUs of the same product to reduce cost and engineering time and,
3) supports mimicking OEM and customers' thermal solutions without the need for tools to heat-up or cool-down the DUT.

1.3. General Goal

Our general goal is to design an apparatus capable of:

1) Help to reduce noise at Intel validation laboratories.
2) Enables the reuse of the heatsinks across different SKUs.
3) Reuse the use of external thermal heads to test parts on mid temperature ranges (45ºC to 85ºC).

1.4. Specific Goals

The apparatus should:

- Cost less than us$25.00 per unit.

- Have a small form-factor that minimizes the required keep-out zone on the target systems (Validation boards)
- Provide the capability to log information regarding system behavior.

2. The Pluggable Fan Controller

To provide a solution to the established problem statement, we implemented an apparatus called a Pluggable Fan Controller (PFC) that has been demonstrated to be able to reduce the noise levels in our labs due to the Fans from >86dBs to less than 75dBs and to simplify, enable the reuse of heatsinks across SKUs in the same program and support mimicking the OEM heatsinks design by modifying the thermal response of the heatsink solution without external components like thermal heads.

The PFC is a small and low-cost microcontroller-based system flexible enough and capable of multiple configurations able to work in the existing platform with minimum risk of mechanical interference.

2.1. Solution Description

Figure 9 shows the proposed solution mounted in a system. The PFC uses the same Fan connector that is already included in the validation platforms.

The PFC also includes a low-cost temperature sensor to read the temperature of the heatsink and provide a close-loop temperature control.

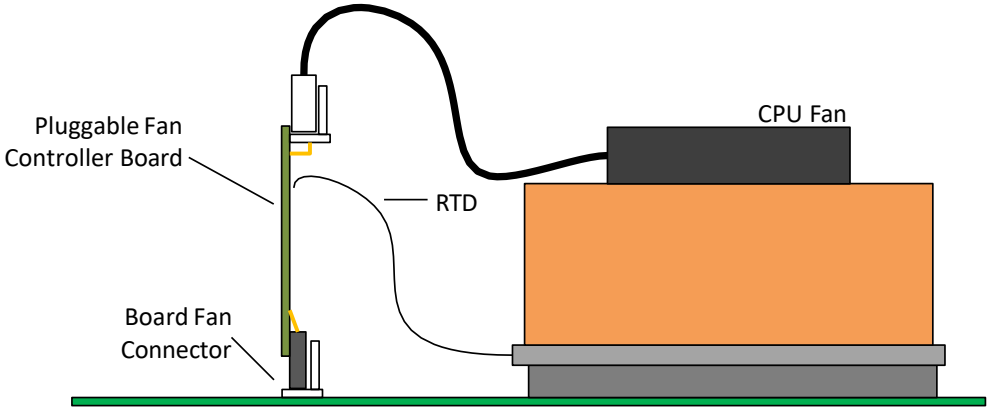

Figure 9 - PFC Concept

As shown in Figure 9, the PFC works as an interposer between the fan connector and the fan, so when the PFC is there, the fan is connected to the PFC and the PFC is connected to the board.

Figure 10 shows the block diagram of the PFC.

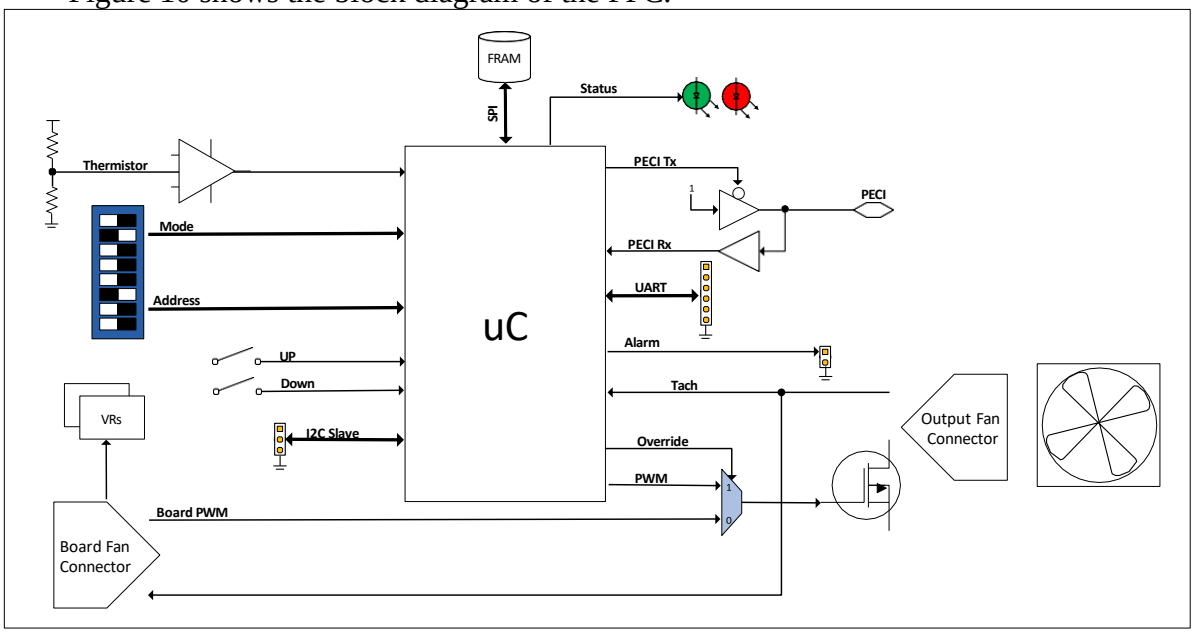

Figure 10 - PFC Block Diagram

Our solution includes the following features:

- Remote monitoring and configuration via I2C with configurable Device ID (up to 8 different addresses).

- Serial port output to report current system status.
- Non-volatile configuration – internal EEPROM (128 bytes) and data storing – 64KB Ferromagnetic Random Access Memory (FRAM).
- User buttons for manual operation setpoint setup.
- Fan Tachometer to report speed and fan stuck detection.
- System status LEDs (Green and Red).
- Open-drain output that can be used as an alarm or to control external heating elements.
- PECI protocol support.
- Configuration jumpers to set the Device I2C address and operation mode.
- On-board voltage regulators to support 12V or 5V power input.
- On-board low-cost thermistor for temperature reading and control.

Figure 11 shows the assembled board and the location of the different elements.

The board is 6x3cms – 2 layers FR4.

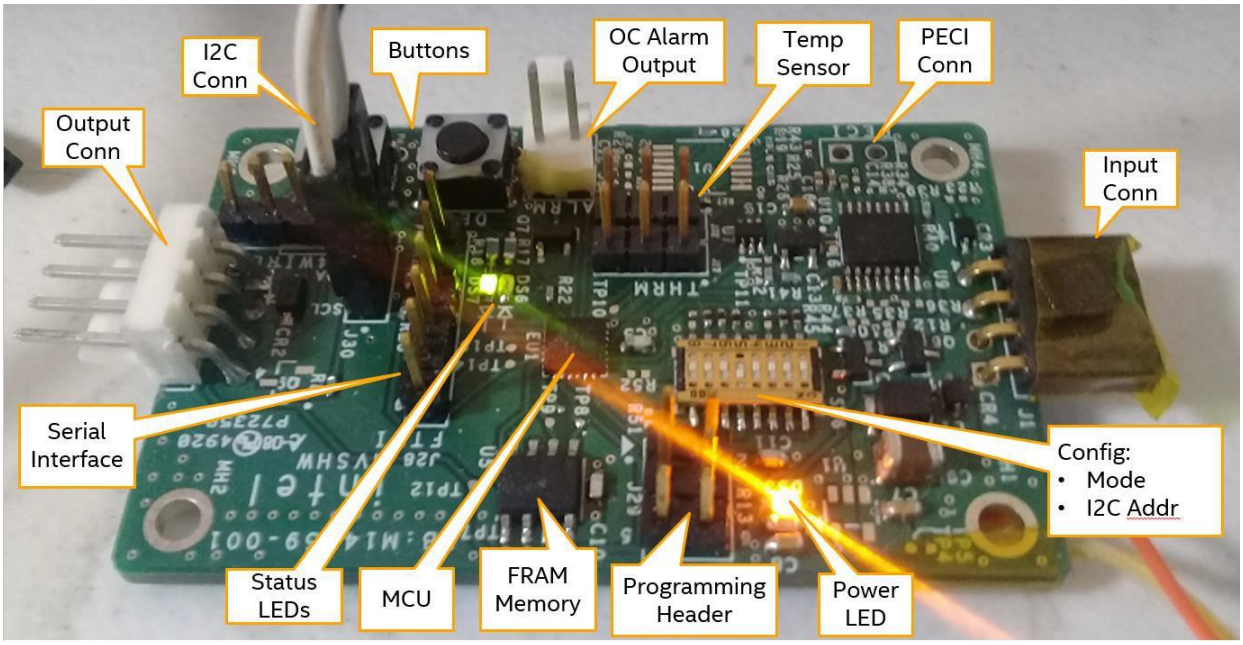

Figure 11 - PFC Rev A Assembled Board

2.1.1 Operating Modes

The solution provides different operation modes to cover the different needs. There are 3 operation modes:

Manual Mode

The system uses a fixed PWM established by the board buttons so users can increase or decrease the PWM value based on their particular needs and forcing a specific airflow in the system for some specific test.

The manual mode is the simplest mode available since there is no close loop. The temperature reading is ignored by the system, and it is only used to report the current system state.

When working in this mode an ‘M’ shown in the serial report and Display.

Control Mode

In this mode, the system’s control algorithm regulates the fan speed using PWM values to maintain the temperature below the established temperature setpoint.

The system push buttons are used to set the system’s operating temperature.

The temperature control algorithm will work to prevent the system temperature from going above the setpoint and cool down the system by increasing the airflow through the heat sink.

The mode is represented by a ‘C’ in the serial report and the display screen as well.

Hot Mode

The Hot mode is similar to the control mode but in this case, the temperature established by using the up and down buttons is used to indicate the minimum operating temperature of the system, so the control will turn on the fan only when the temperature is above the operating setpoint and will shut the fan off if the temperature is below, allowing the system to heat itself to work in the warmer area that the user needs to mimic the thermal solution of OEMs and ODMs.

An ‘H’ is shown in the serial report and the screen to indicate that the Hot mode is configured.

The following sections provide details on the implemented Hardware and Firmware to support the mentioned features and operating modes.

2.2. Hardware Implementation Details

The Pluggable Fan Controller hardware is described in this section, covering details and the rationale behind the selected ingredients.

2.2.1 The Microcontroller

We are using an ATTiny817 taking advantage of all the features and low cost – less than us$1. As can be seen in Table 1, the ATTiny817 includes several hardware features in a very small package – 4x4mm.

Table 2- ATTiny817 Features[3]

Feature	Value
Program Memory Size (KB)	8
RAM (bytes)	512
Data EEPROM (bytes)	128
Pin Count	24
Operation Voltage Max.(V)	5.5
Operation Voltage Min.(V)	1.8
Timers	1 x 8-bit and 2 x 16-bit
Max ADC Resolution (bits)	10
ADC Channels	12
Number of Comparators	1
SPI	1 -SPI
I2C	1 -I2C
Stand-alone PWM	6

In this proposal, Figure 12 shows the Microcontroller's connection, and Table 3 shows the pin assignment.

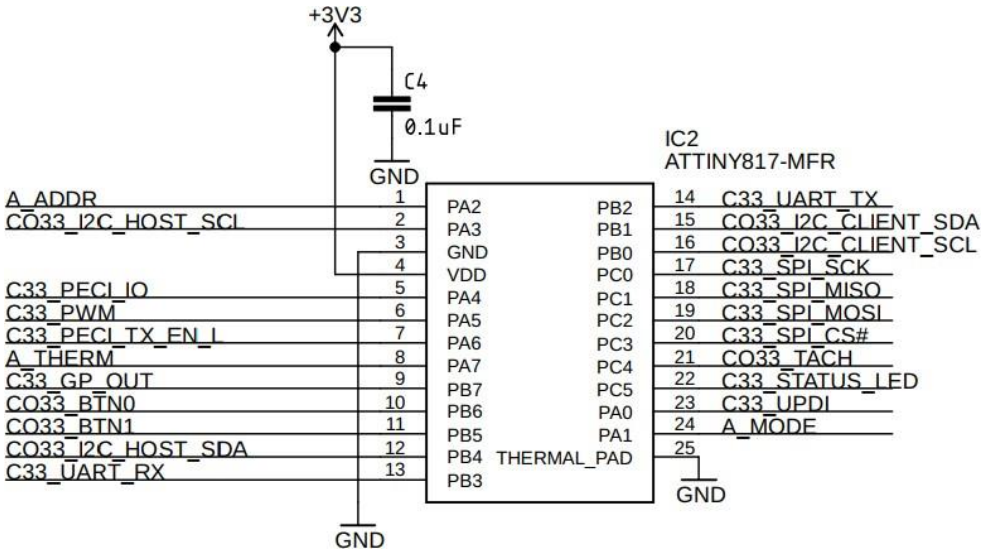

Figure 12- The Microcontroller - ATTiny817

Table 3 – Microcontroller's Pin assignment

Pin	Port	Net Name	Description
1	PA2	A_ADDR	Analog input port to read the I2C Device Address
2	PA3	CO33_I2C_HOST_SCL	Open drain SCL output in the Host I2C port
3	GND	GND	Ground reference connection
2	VDD	+3V3	3v3 Power
5	PA4	C33_PECI_IO	Translated PECI Data signal
6	PA5	C33_PWM	PWM Output
7	PA6	C33_PECI_TX_EN_L	PECI Translator direction control signal
8	PA7	A_THERM	Analog input to read the thermistor value
7	PB7	C33_GP_OUT	General purpose output, connected to an open drain MOSFET
10	PB6	CO33_BTN0	Increment button Input
11	PB5	CO33_BTN1	Decrement button Input
12	PB4	CO33_I2C_HOST_SDA	Open drain SDA bidirectional data for Host I2C Port
13	PB3	C33_UART_RX	UART receivers' data input
14	PB2	C33_UART_TX	UART transmitter's data input
15	PB1	CO33_I2C_CLIENT_SDA	Open drain bidirectional data line for Client I2C Port
16	PB0	CO33_I2C_CLIENT_SCL	Open drain clock input line for Client I2C Port
17	PC0	C33_SPI_SCK	FRAM SPI port Clock output
18	PC1	C33_SPI_MISO	FRAM SPI Master input data line
19	PC2	C33_SPI_MOSI	FRAM SPI Master output data line
20	PC3	C33_SPI_CS#	FRAM Chip selects active low output
21	PC4	CO33_TACH	Fan tachometer input
22	PC5	C33_STATUS_LED	System status LED

23	PC5	C33_UPDI	Programming interface data line
24	PC7	A_MODE	Analog input to read the operation mode

2.2.2 I2C Client

One of the key differences between the PCF and other options is the capability to read information and configure the module using I2C.

The PFC includes an I2C client that allows the user to read information like: Thermistor temperature, fan tachometer, current PWM percentage, as some examples.

The user can also use the channel to remotely configure different operation parameters in the system like operating temperature, PWM value, among others.

Figure 13 shows the hardware connection of the client port, including pull-up resistors.

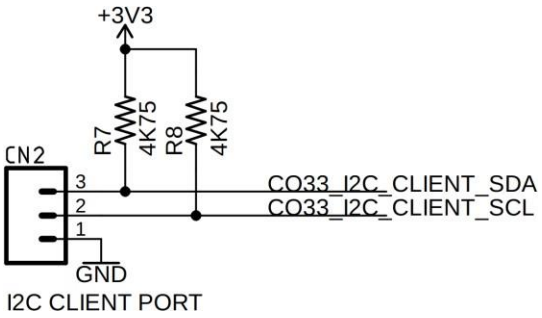

Figure 13- I2C Client Interface

2.2.3 I2C Host

The Rev B of the hardware, we included an OLED Display to show relevant system information to the user and to avoid the need to use the serial port to "see" the working parameters of the system.

That OLED display is connected to an I2C channel that is driven by the microcontroller code. We are calling that channel the Host I2C.

Figure 14 shows the hardware connection of the Host ports. One of those ports support the connection of the OLED LCD and the other one is available for future system expansion, i.e.: Adding more sensors or expanding IO ports.

The Host I2C port provides 3v3 power to the connectors so we can power the connected devices.

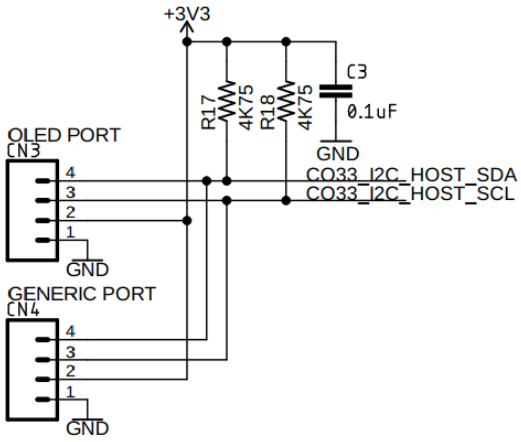

Figure 14- I2C Host Interface

2.2.4 UART

To support interaction with the user, the system implements a serial port to provide a report of the current system status. The information in the serial report is described in the Firmware implementation section.

Even though, the hardware provides capability to receive and transmit data, the system uses the UART to report data and not to receive commands at this moment. We are exploring the need to incorporate serial commands to control the PFC.

To reduce cost, the UART implementation uses pin headers to connect to USB to TTL cables so no on-board USB to UART Bridge is supported. There are many off-the-shelf solutions available from FTDI, Digi-Key, or SparkFun.

Since the Computer keeps its TX output high (C33_UART_RX), having the system connected to the cable provides leakage current. To avoid the PFC using that leakage current to power the Microcontroller, we added a series 10KΩ resistor to limit the current flow through that signal as Figure 15 shows.

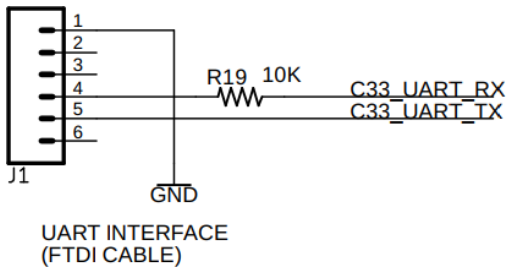

Figure 15- USB To UART 3v3 FTDI Cable Connection

2.2.5 PECI Support

The Platform Environment Control Interface Protocol (PECI)[4] is an asynchronous protocol that provides access to read and write multiple internal CPU environment status registers through a single pin connection, similar to Dallas Semiconductor's 1-wire protocol.

PECI is mainly used to read the Discrete Thermal Sensors (DTS) in the CPU and get the CPU temperature, but it also supports different commands to detect devices in the bus and read parameters.

Figure 16 shows the Ping command, used to detect if the CPU is present in the bus.

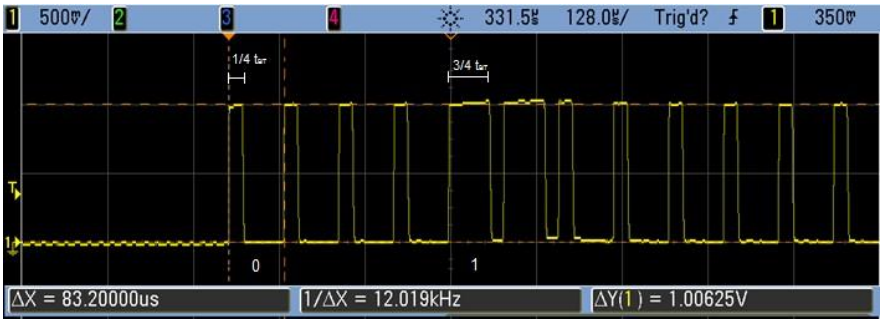

Figure 16- PECI Ping Frame

The protocol works at a CPU voltage level and in most of the cases it is 1v0.

Since our microcontroller works at 3v3, we used voltage translators to support the connection between the microcontroller and the CPU. As shown in Figure 17, the PFC uses a connector (CN9) to allow connection between the PFC board and the System's Motherboard to read CPU temperature through PECI. Figure 17 also include the translation interface, based on the 74AVC1T45.

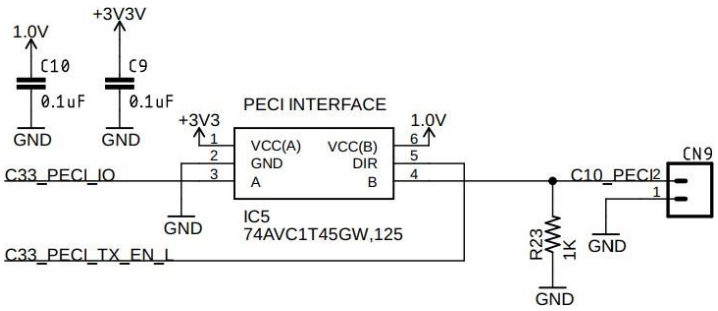

Figure 17- PECI Interface

The 74AVC1T45 provides direction control, so the microcontroller controls the signal direction using a DIR signal (C33_PECI_T_EN_L) and reads and writes data accordingly using A signal (C33_PECI_IO).

The translator is powered by 3v3 and 1v0, as Figure 17 shows. Side A provided the interface to the microcontroller and side B to the CPU.

2.2.6 Configuration Jumpers

Taking advantage of the available ADC channels and looking to save IO pins, a voltage divider circuitry approach was followed. This approach was used to enable the user to change the I2C Client device ID and to set the operation mode.

In the case of the I2C Client device ID, the voltage divider circuit uses 3 pins as shown in the top of Figure 18, while we use only 2 pins for the Operation Mode setting.

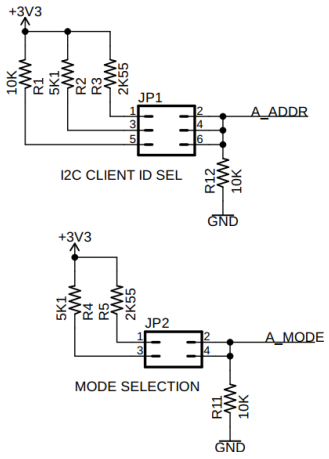

Figure 18- System Configuration Jumpers

Table 4 and Table 5 shows the different voltage level and the selected options by combining the different jumper positions.

Table 4- Device ID Jumpers Selection

JP2.3	JP2.2	JP2.1	Voltage	Device ID (7-bits)
Open	Open	Open	0v	0x10
Open	Open	Short	2.63v	0x11
Open	Short	Open	2.185v	0x12
Open	Short	Short	2.82v	0x13
Short	Open	Open	1.65v	0x14
Short	Open	Short	2.74v	0x15
Short	Short	Open	2.466v	0x16
Short	Short	Short	2.88v	0x17

Table 5 - Mode Selection Jumpers

JP1.2	JP1.1	Voltage	Mode
Open	Open	0v	Manual (M)
Open	Short	2.63v	Manual (M)
Short	Open	2.185v	Control Mode (C)
Short	Short	2.82v	Hot Mode (H)

2.2.7 Fan Interface

The main temperature control element of PCF is the CPU fan. By controlling the fan speed, we can increment or decrement the air flow through the heat sink and keep the CPU's temperature in the range specified by the user.

The PFC uses the fan connector that is already in the board as shown in Figure 19. The input connector provides the power that the fan uses and the power source that the PFC uses to power the microcontroller and the different elements in the board.

The PFC supports +12V and +5V fans. As mentioned before, the voltage source is provided by the input connector.

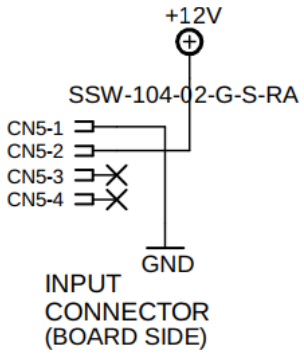

Figure 19- Fan Input Connector

The PFC works as an interposer between the board connector and the fan as Figure 9 shows. The board intercepts the PWM from the board and replaces that with the PWM generated by the Microcontroller, based on the selected operation mode.

The Microcontroller's PWM output drives a MOSFET (Q2) that drives the fan as shown in Figure 20.

The fan's tachometer output is isolated from the Microcontroller using another MOSFET, that works as a voltage translator – Q1 in Figure 20, and is connected an interrupt capable pin so the pulses are detected, enabling the fan speed measurement.

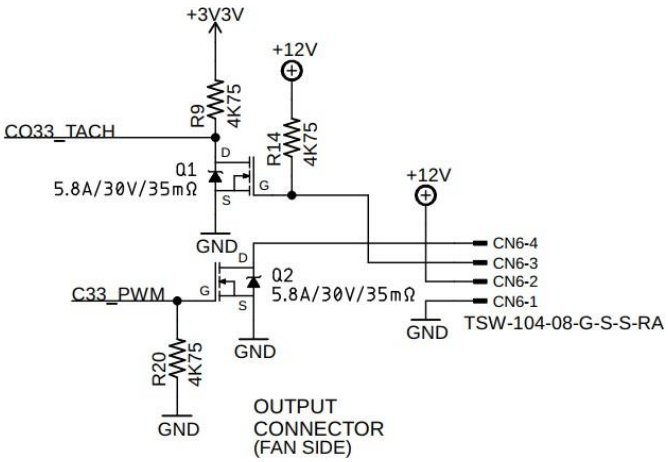

Figure 20- Fan Output Connector

The power coming from the board is passed to the output connector, so all the current is directly drained from the motherboard.

2.2.8 Open-drain Output

A general purpose open-drain output is included as shown in Figure 21, to provide the capability of generating an alarm in case of overheating or to control an external heating element to heat-up very low TDP parts.

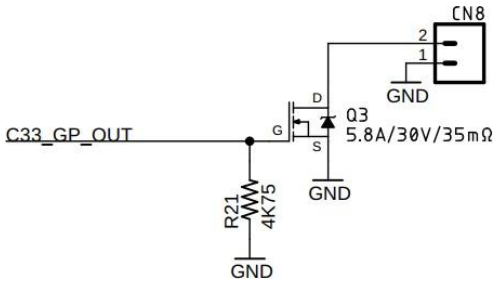

Figure 21- Open Drain Output

In the case of the alarm, the output can be connected to a special signal in the base board to force the system to shut down in case of overtemperature to avoid damage to the parts due to overheating.

For external heating elements control, the power should be provided externally. The selected MOSFET supports up to 5A so small heating elements can be used if necessary and help to heat-up the parts when necessary.

2.2.9 Thermistor Support

Due to the low-cost and the availability of many different form-factors, we included a 10KΩ 3380K Negative Temperature Coefficient (NTC) thermistor to read the temperature.

The thermistor connection is implemented as a voltage divider with a pull-down resistor (Figure 22), so the Microcontroller reads the thermistor voltage using an analog pin.

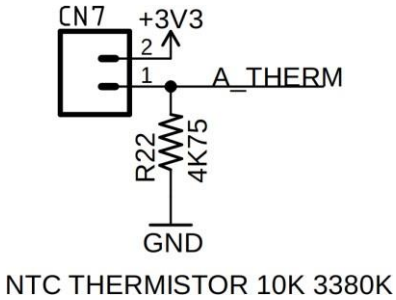

Figure 22- Thermistor Interface

Table 6 shows the resistor values depending on the temperature, the corresponding output voltage, and the Microcontroller ADC Value (10-bits).

Table 6 - Thermistor Values[5]

Temp (C)	Ohms	Voltage	ADC Val
-40	195600	0.077433849	63
-35	148100	0.101505236	83
-30	113300	0.131440678	108
-25	87550	0.168130081	138
-20	68230	0.212669683	174
-15	53650	0.265809769	218
-10	42500	0.328601695	269
-5	33890	0.401917595	329
0	27210	0.486054528	398
5	22020	0.580464072	476
10	17920	0.685676393	562
15	14670	0.800722767	656
20	12080	0.92431466	757
25	10000	1.055102041	864
30	8315	1.191701882	976
35	6948	1.331559066	1091
40	5834	1.472375166	1206
45	4917	1.612769055	1321
50	4161	1.750366776	1434
55	3535	1.883424408	1543
60	3014	2.010630023	1647
65	2586	2.128740049	1744
70	2228	2.238741339	1834
75	1925	2.341132075	1918
80	1669	2.435233161	1995
85	1452	2.521131339	2065
90	1268	2.59886059	2129
95	1110	2.669535284	2187
100	974	2.733521325	2239
105	858	2.790572148	2286
110	758	2.841700257	2328
115	672	2.887192852	2365
120	596	2.928625378	2399
125	531	2.965016249	2429

Taking advantage on the implementation, the system is able to detect if the thermistor is not connected or is pulled out, since the ADC value will provide a reading very close to 0.

2.2.10 Voltage Regulators

The PCF has two on-board LDO voltage regulators to generate the necessary voltage that its components require. The main one of those regulators is a +3v3 regulator that supports up to 16V as input. It provides power to the microcontroller and most of the support components around it (Figure 23), like the FRAM and other boards elements.

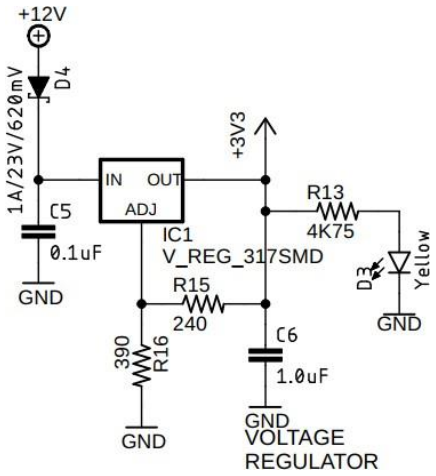

Figure 23- 3v3 Voltage Regulator

The +3v3 regulator also provides the input power of another regulator to generate +1v0 used as the reference for the PECI voltage translator as Figure 24 shows.

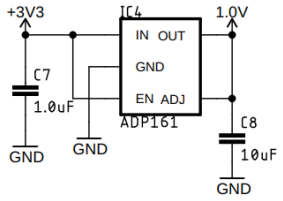

Figure 24- 1v0 Voltage Regulator

2.2.11 User Interface

To support interaction with the users, the PFC includes output and input blocks.

The first block is the status LEDs. We have two LEDs connected to the same IO pin to indicate the status of the system.

One of the LEDs is green and the other is LED (Figure 25). To turn on the red LED, the Microcontroller sets the pin high and to turn on the green LED, the pin is set to low.

Tri-stating the pin, will turn off both LEDs.

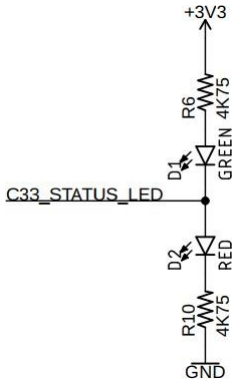

Figure 25- Status LEDs

To set the operation temperatures and PWM values, the system includes a couple of push buttons directly connected to the Microcontroller. The hardware implementation takes advantage of the internal pins pull-up capability to simplify the connection.

As Figure 25 shows, the buttons force the pin low when pressed.

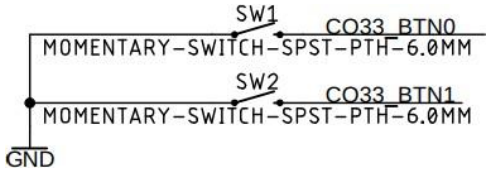

Figure 26- Buttons Connection

In hardware Rev B we included a Graphic OLED (Organic Light Emitting Diode) Display to show the system status and some parameters in an easy-to-use way.

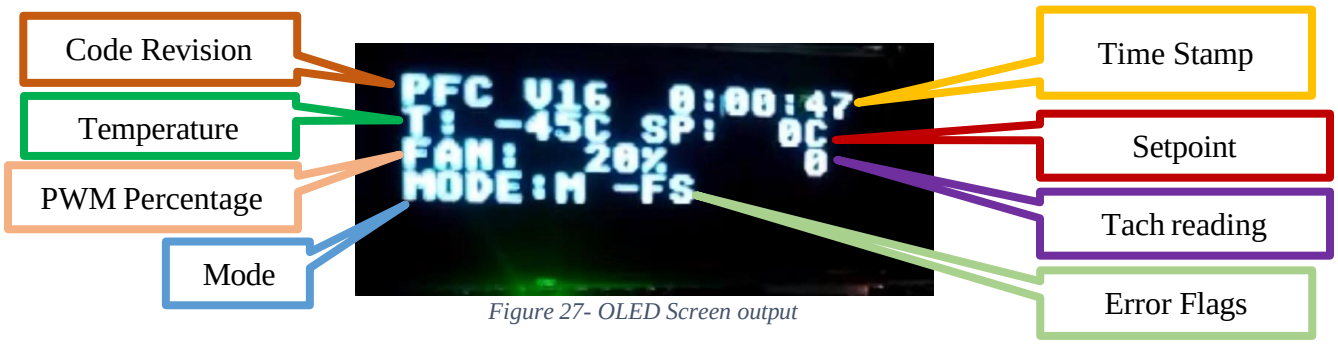

Figure 27- OLED Screen output

The OLED is 128x32 pixels and measures 0.96" diagonal, and as Figure 27 shows, it is used to expose key system information.

2.2.12 FRAM Support

Data logging is part of the PFC features, and the board includes an FRAM in the system. The selected FRAM uses SPI as the access interface, as the connections in Figure 28 show.

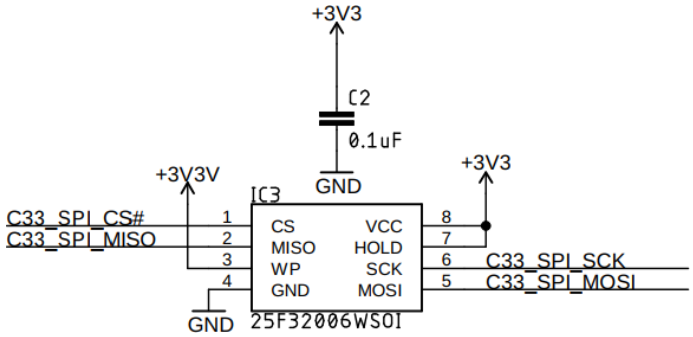

Figure 28- FRAM

The design takes advantage of the hardware SPI port in the Microcontroller to provide access as fast as 10Mhz.

2.3. Board Revisions

There are two version of the PFC boards. The 2nd revision, called Rev B, was designed considering boards enhancements to our initial proposal and to include learnings from the first prototype (Rev A), like the OLED, simplify the circuitry implementation and the ease of use to our end users by changing the Dipswitch to pin headers with jumpers.

Figure 29 shows different angles of the Rev B board.

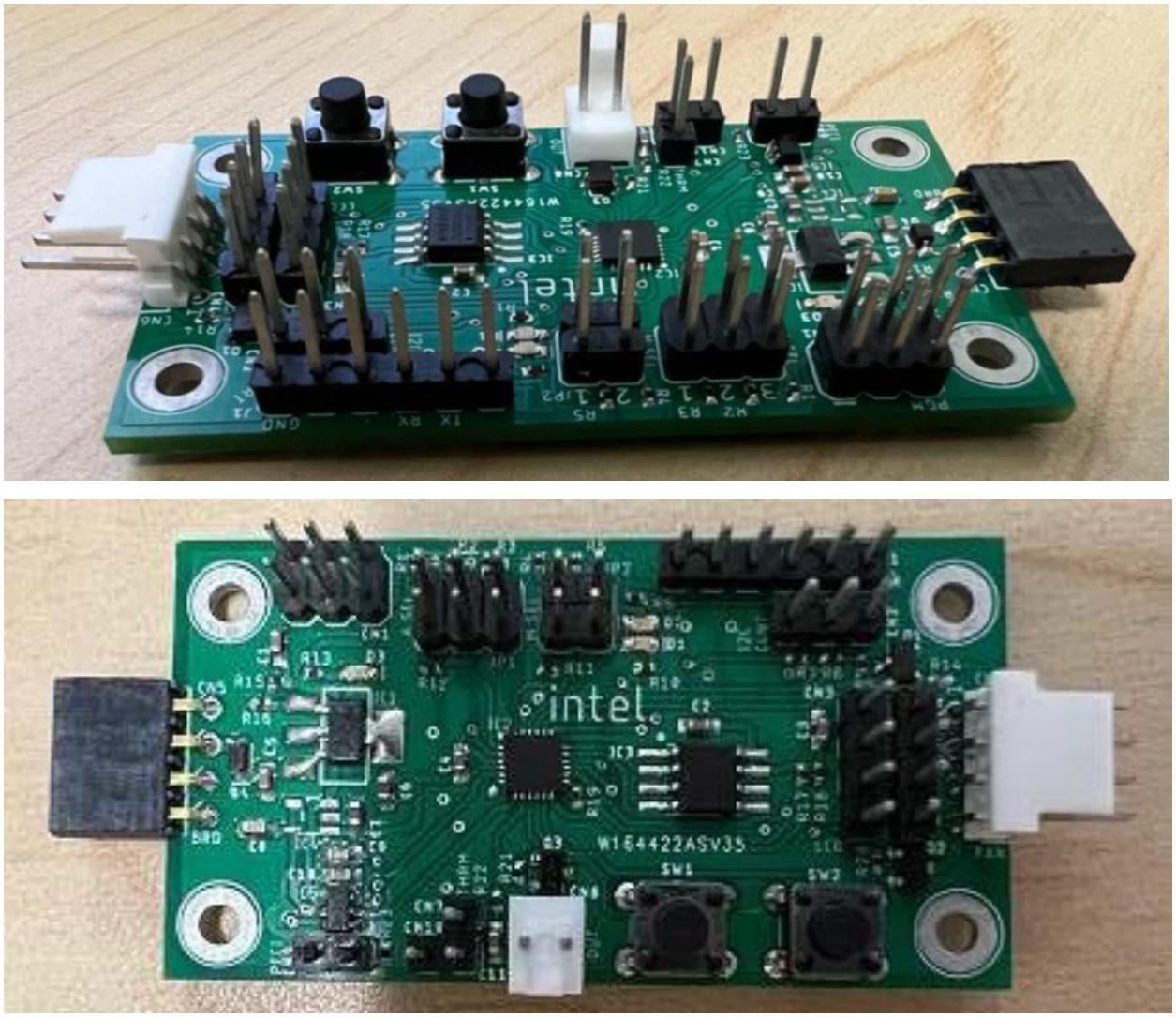

Figure 29 - PFC Rev B

2.4. Firmware Implementation Details

This section describes the PFC firmware implementation. The firmware is coded in C using Microchip Studio which is based on GCC (GNU Compiler Collection). It includes support for Attiny817. In Figure 30 we can see the version [Microchip Studio 7.0.2542].

Figure 30 - FW Development Tool Version

In Figure 31 we can see the Microcontroller's memories usage after compiling the final code version. Flash is ~80%, RAM is ~33% and EEPROM is ~4% so there is still room to add new features.

```
   text    data     bss     dec     hex filename
   6541      12     156    6709    1a35 PFC817.elf
Done executing task "RunCompilerTask".
Using "RunOutputFileVerifyTask" task from assembly "C:\Program Files (x86)\Atmel\Studio\7.0\Extensions\Application\AvrGCC.dll".
Task "RunOutputFileVerifyTask"
        Program Memory Usage    :   6544 bytes   79.9 % Full
        Data Memory Usage       :   168 bytes   32.8 % Full
        Warning: Memory Usage estimation may not be accurate if there are sections other than .text sections in ELF file
        EEPROM Memory Usage     :   9 bytes   3.5 % Full
```

Figure 31 - Microcontroller memory usage report

The code is modularized in multiple files to simplify implementation and reuse. Figure 32 shows some of the source files used in the project.

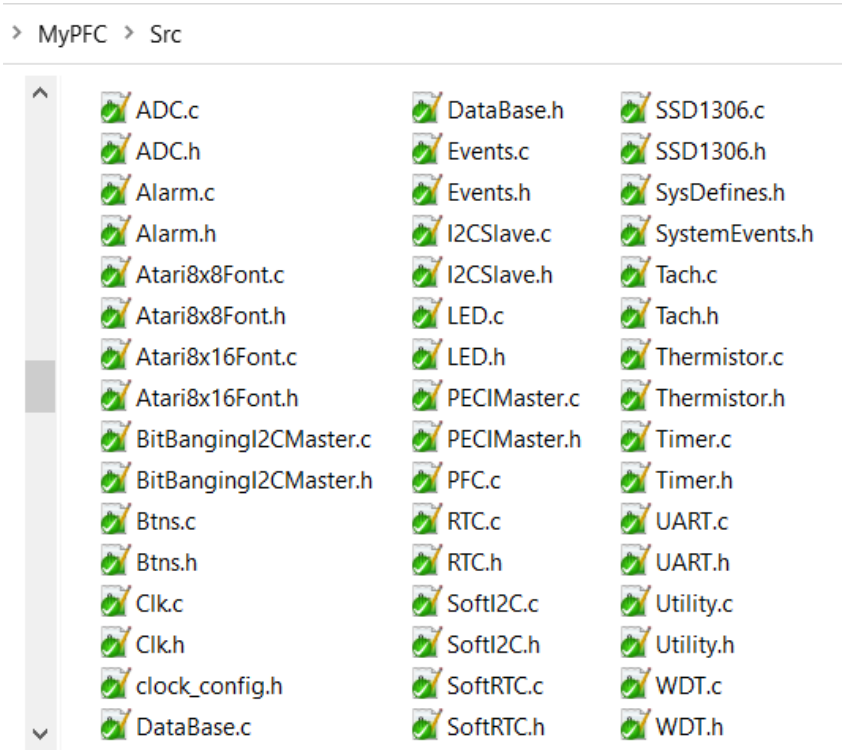

Figure 32 - Source Files

Figure 33 shows the Control Version tool: GitHub, that was selected to guarantee a prover code back-up and changes tracking.

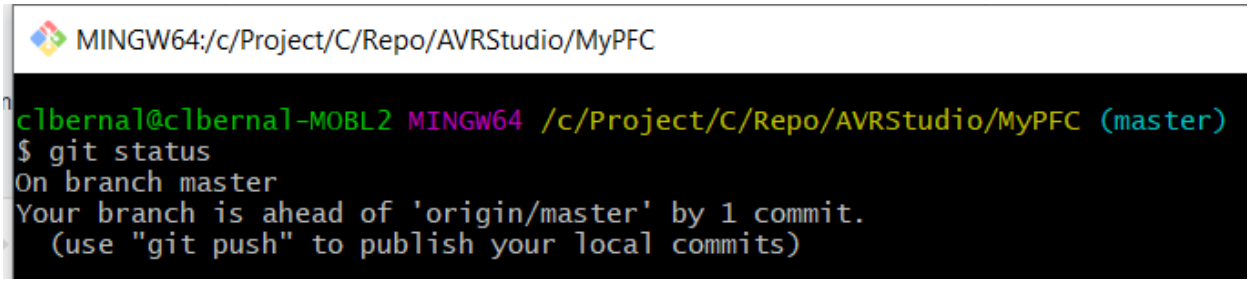

Figure 33 - GitHub Repository

We are not using any RTOS (Real-time Operating System) to prioritize the different tasks performed by the microcontroller. Instead, we are using an event approach combined with state machines implemented as function pointers arrays. With that approach, we provide the look and feel of a multitasking system without the resources consumed by the RTOS. The event-driven approach and the different task timing was properly distributed to be handled by the CPU capability.

2.4.1 System Events and the Events Engine

As previously mentioned, the code uses an event approach to support the different tasks. In Figure 34 we can see how the system is continuously checking the different event flags to determine which tasks need CPU time.

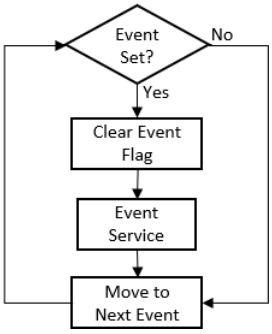

Figure 34-Event Engine Simplified Flow Chart

Following this approach and keeping the different tasks short, we provide a multi-tasking experience without the need for an RTOS to administrate the CPU usage and bandwidth.

The events are grouped into asynchronous events and time-based events.

The time-based events are sourced by a “tick” timer implemented using one of interrupt capable hardware timers provided by the Microcontroller and it’s used as a time base with a 10mS period.

From that “Tick” base, the code generates several other software timers based on counters to:

1) Feed the software the real-time clock used for the time stamps in the serial reports,
2) Read the board buttons every 20ms,
3) Calculate the Fan revolutions per minute every 250ms,
4) Trigger PECI frames every 500ms,
5) Sent serial frames every second,
6) Update the OLED every second,
7) Display the system status using the LEDs blinking every 200ms,
8) Software PWM on the alarm output to control an external heating element,
9) Store status in the FRAM every 30 seconds and
10) Determine when the new configuration is stable and needs to be stored in EEPROM.

For asynchronous events, we implemented the following events:

1) Analog-to-digital converter data ready,
2) Board buttons press,
3) Incoming I2C frame and
4) Base timer overflow event to feed the tick timer and the rest of the software timers.

2.4.2 Operation Modes

The PCF firmware continuously monitors the ADC to detect changes in the mode input coming from the Mode Selection Header.

When the mode changes, the firmware reads the selected operation mode parameters stored in the on-chip EEPROM database and starts working using those parameters.

Depending on the mode, the parameters may change.

Since the EEPROM has a limited write cycles per byte (100,000 cycles), when a parameter is changed, the system waits 1 minute before saving the new values to reduce the number of EEPROM write cycles and increase the memory endurance.

In the case of Manual mode (‘M’), the system stores in EEPROM the selected PWM value. For Hot Mode (‘H’) and Control Modes (‘C’) the stored value is the Setpoint.

In Manual Mode, the system keeps reading the temperature, but the temperature value is not used to change the PWM, while in Hot Mode and Control Modes, the system, based on a control algorithm changes the PWM to keep the system close to the selected Setpoint value.

The difference between Hot Mode and Control mode is when the PWM is activated. The hot mode purpose is to keep the DUT warm, while the Control Mode works to keep the processor cold. Depending on the difference between the Setpoint and current thermistor temperature read value, the code determines the PWM value to apply which changes in increments of 25%.

2.4.3 Thermistor Reading

As shown in Table 6 and Figure 35, the thermistor’s resistance values change with temperature is not linear. Resistance change follows the Steinhart–Hart equation[6] illustrated in Figure 36.

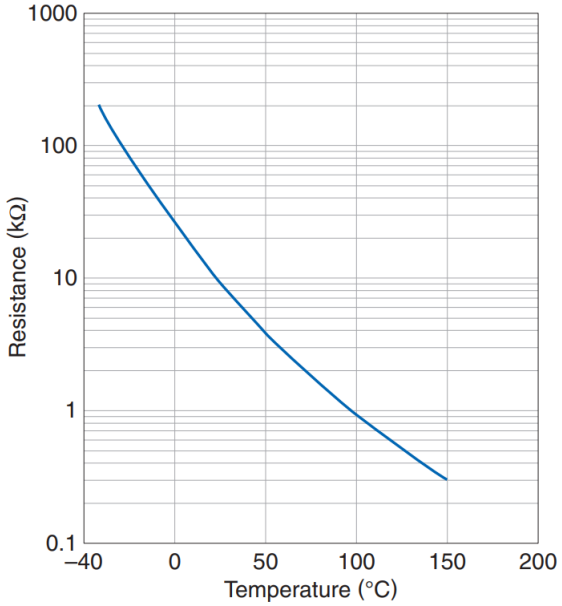

Figure 35 - NTC 10K B3380 Thermistor Curve[5]

$$\frac{1}{T} = a + b\ln R + c\,(\ln R)^3$$

Figure 36 - Steinhart–Hart equation

To simplify the calculation of the value, the code uses a look-up-table approach. The implemented table includes values every 5ºC per Table 6, covering from -40ºC to 125ºC (33 values in total). To calculate the intermediate value, we use interpolation. The code uses the thermistor ADC output and search for the immediate above and immediate below values and using the difference to both and the "distance" obtains the new measure value.

2.4.4 I2C Client Support

To provide remote access to the PFC, we included an I2C Client that allows users to read and modify internal registers.

Any USB to I2C Module, i.e.: Aardvark from Total Phase, Bus Pirate or FTDI boards, can be used to change the system configuration and read parameters remotely.

The complete register set is presented in

Table 7. It includes the size, range, and access capability of each register.

Table 7- I2C Register Map

Index (hex)	Name	Access Type	Value Type	Range / Options	Description
0x00	Code Revision	RO	Number	0x00 to 0xff	Current code revision
0x01	Status	RO	Number	0x00 to 0xff	System status flags and errors [0] - Sensor Error - Set if the system detects that the sensor is missing [1] - Fan Error - Sets if there are no Tach pulses while the PWM % is not 0 [2] - Overtemperature Error - Set if the current temperature is above the limit [7:3] – Reserved
0x02	Control	RW	Number	0/1	Control register [0] - Override Enable Flag - Enables Remote values overriding and blocks the system from automatically updating Mode ID, etc... [7:1] - Reserved
0x03	Temperature	RO	Number	-45 to 125	Sensor temperature
0x04	PWM Percentage	RO	Number	0 to 100	Current PWM Percentage
0x05	Mode ID	RW	Character	R', 'C', 'M', 'H'	Current system's Mode ID - can be changes
0x06	Hours	RW	Number	0 to 23	RTC Hours
0x07	Mins	RW	Number	0 to 59	RTC Minutes
0x08	Secs	RW	Number	0 to 59	RTC Seconds
0x09	Control Set Point	RW	Number	40 to 100	Control mode setpoint
0x0A	Hot Mode Set Point	RO	Number	40 to 100	Hot mode setpoint
0x0B	Override Set Point	RO	Number	40 to 100	Remote control setpoint
0x0C	PWM Value MSB	RO	Number	0 to 320	System PWM Value Most Significant Byte
0x0D	PWM Value LSB	RO	Number		
0x0E	PWM Override Value MSB	16 bits RW	Number	0 to 320	Remote PWM Value Most Significant Byte. To Change the value, first write the MSB and then the LSB Any change in that order will discard the update
0x0F	PWM Override Value LSB	16 bits RW	Number	0 to 320	

0x10	Tach RPMs MSB	RO	Number	0 to 65535	Current Tach RPMs Most Significant Byte
0x11	Tach RPMs LSB	RO	Number		
12	Thermistor ADC Value	RO	Number	0 to 255	Raw Thermistor ADC Value
13	Heater PWM Value	RW	Number	0 to 255	Heater PWM Value

The implemented client supports single-byte access, so bigger transactions will not be acknowledged.

To allow correct access to 16-bit registers, we implemented a special sequence in the code that prevents incomplete modifications to the registers. When writing to a 16-bit register, it is necessary to write the MSB first. When that happens, the code will set an internal flag and will store the value in a dedicated variable.

The system expects that the next access writes to the LSB of the 16-bit register. If that happens, then the complete 16-bit update will be completed, otherwise no register will be updated.

To simplify the usage, a Python library is provided to our end users to access the register set in a friendly way.

Figure 37 shows examples of some of the implemented commands – printPWMValue and printStatus.

Figure 37 - Python Commands Examples

2.4.5 Serial Report Output

The firmware implements an automatic and continuous status serial report like the one presented in Figure 38. A new status line is sent every second using the following serial port configuration: 1200 bauds 8N1.

The firmware uses the on-board hardware Universal Asynchronous Receive-Transmitter module (UART) to transmit the report.

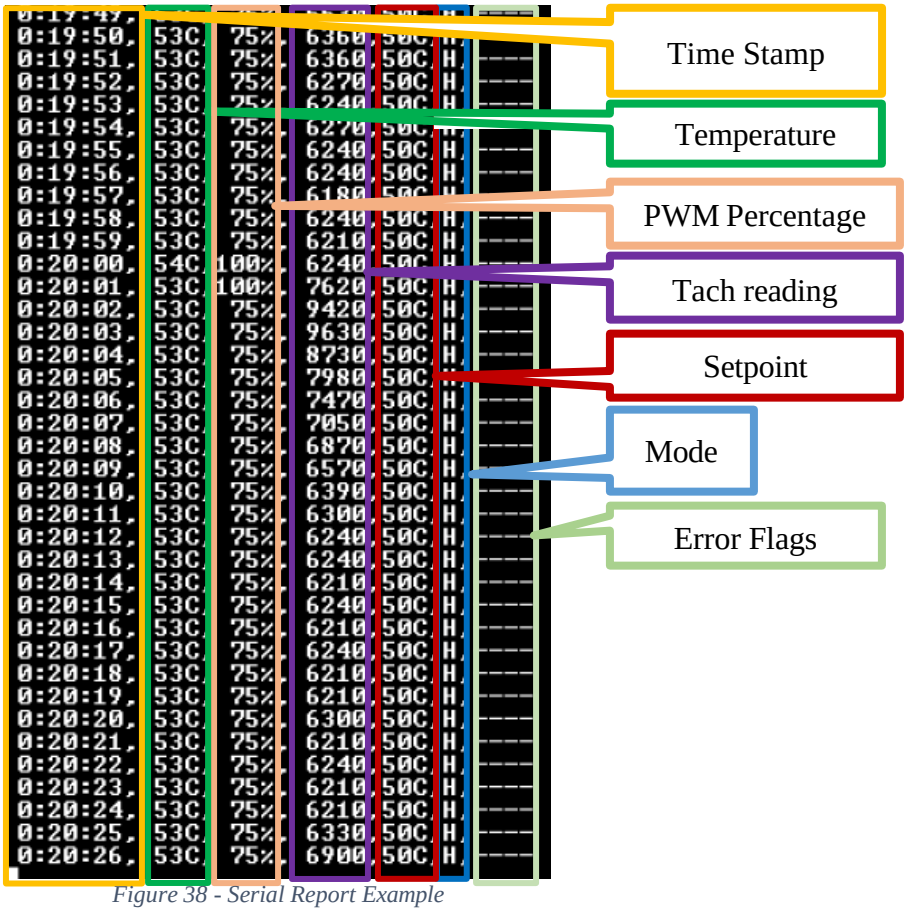

Figure 38 - Serial Report Example

Table 8 shows the fields included in the report and Figure 39 presents the different status flags and how they are displayed.

Table 8- Serial Report Fields

Field	Name	Comments
1	Timestamp	Shows for how long the system has been working. Includes hours, minutes, and seconds
2	Temperature	The current temperature value from the Thermistor
3	PWM	PWM percentage value

4	Tachometer	Fan tach pulses count
5	Set Point	System setpoint for multiple control modes
6	Mode	Current system operation mode
7	Status Flags	Override enable(+), Sensor Error (S), Fan error (F) and Over temperature (O)

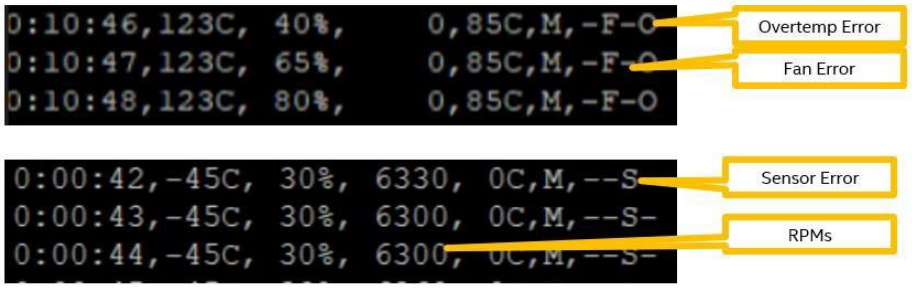

Figure 39- Serial Report Errors

The serial report can be used by the end-users to generate graphs and analyze data on how the system was performing as Figure 40 illustrates.

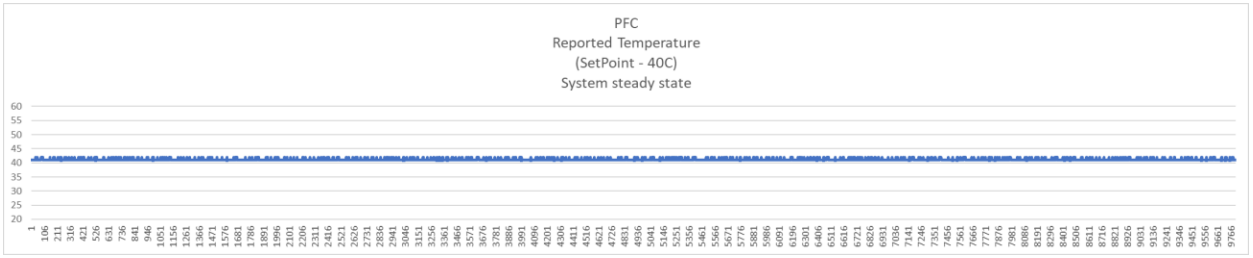

Figure 40 - System Performance at 40C

2.4.6 PECI Protocol Support

PECI[2] is a single-wire protocols used to read the temperature from CPUs. It supports bit rates from 2KHz to 2MHz.

In our case, the support was implemented using bit-banging using a low-seed data rate to simplify the solution.

The full protocol stack was not implemented, only the required commands:

- Ping – Check that the CPU is available in the bus,
- GetTemp – Reads the temperature from the CPU.

By implementing those commands, we read the temperature sensed by the thermal Diodes in the CPU.

2.4.7 Fan PWM and Tachometer

Depending on the operation mode, the PWM value is set by the control algorithm. The PWM is supported using a hardware 16-bit timer sourced by the 16MHz clock to generate a 25KHz PWM output signal.

The allowed PWM Counter values for that timer are between 0 and 320 in our case.
To avoid problems breaking the static resistance when the fan is stopped, the code turns on the fan using 30% of the PWM.

The PFC monitors the fan tachometer and reports the revolutions power minute (RPMs) through the serial port, OLED display and I2C Client. We use the interrupt-on-pin-change to accumulate the pulses on periods of 250ms. A 250ms event is set to support the Tachometer measurements. On every event activation, the system calculates the RPM by reading the counter and multiplying the value by 240. Then the counter is set to 0 to start accumulating the next set of pulses.

2.4.8 FRAM Driver

FRAM uses SPI its physical interface. The code implemented the SPI driver taking advantage of the hardware support to prepare the packages providing the memory commands.

The system creates a record every 30 seconds that contains the relevant information of the system's status: operation mode, error flags (Over temperature, fan stuck, Missing Sensor), Operation setpoint, PWM Percentage and current temperature, for a total of 4 bytes per write.

The Error flags (3 bits) and the operation mode are combined in a single byte. That same byte also indicates if the frame is valid, using 3 bits with a specific code (3'b101) the system indicates valid data.

Table 9 details how the status record is store in memory and each one of the fields.

Table 9 - FRAM Log Frame

31	30	29	28 : 27	26	25	24	23 : 16	15 : 8	7 : 0
1	0	1	Mode	O	F	S	Set Point	PWM	Temp

Based on the memory size that we are using we can store up a history up to 8 hours.

On every system power cycle, the code clears all the memory to invalidate any previous data. By doing that, the system is able to check if the next location is used by valid data or not and to calculate how many status records are included.

2.4.9 OLED Display Support

The OLED display provides an I2C interface to the internal driver. The PFC code implements a software I2C host to control the display.

Since this is a graphic display, the code provides support to the characters map. Our implementation uses a look-up table to implement that character map.

Different routines handle the translation from text to the specific bit map.

As we can see in Figure 27, we only implemented capital letters, numbers, and some special characters to save Flash memory space (69 symbols in total – 552 bytes).

To create the character's map, we implemented an Excel Spreadsheet to simplify the process of converting bitmaps to numbers in an array.

2.4.10 Errors Detection

The PFC has error flags that are set when a problem is detected. Those flags are available through I2C registers, Serial report, and OLED screen.

The system detects and reports the following errors:

1) Stuck Fan (‘F’): PWM output is active, and the PFC is not detecting any tachometer pulses,
2) Missing Sensor (‘S’): sensor reading is out of range and
3) Overtemperature (‘O’): temperature range is above the allowed range.

When errors are detected, the red status LED blinks instead of the green one.

3. Test and Validation

3.1. Noise Level Reduction Verification

One of the first problems we worked to solve with the PFC was the noise levels in our labs.

To demonstrate and verify the noise level reduction with the PFC implementation, we worked with the Intel Labs team to use their Anechoic chamber and measure the differences between the regular platform control and the implementation with the PFC.

Figure 41 shows the system installed in the Anechoic chamber to be able to isolate any external noise and determine exactly how much noise reduction we get by using the PFC.

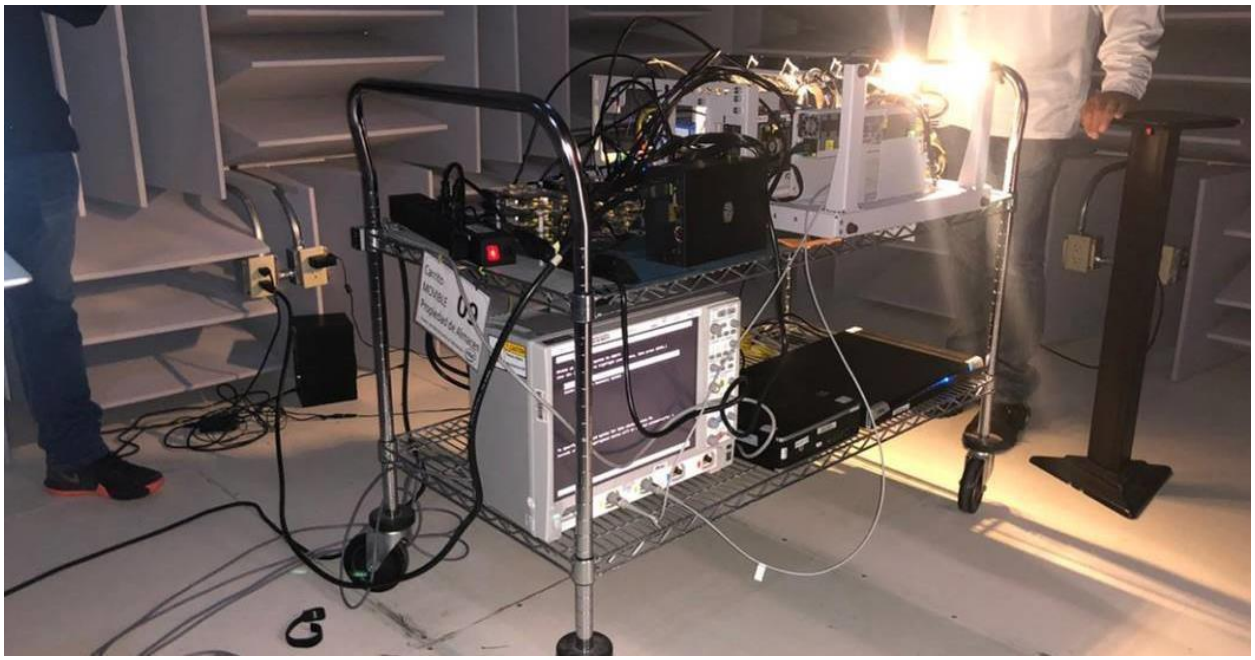

Figure 41 - Anechoic Chamber test for noise level reduction

The test indicated that the PFC reduces the noise level by up to 27%!

3.2. Temperature Measurement Difference

Since the PFC is measuring the temperature in the area between the heatsink and the CPU cover, we wanted to verify the difference observed from the internal temperature diodes in the CPU.

From Figure 42 we can determine that:

1) The difference between the thermal diodes (DTS) and the thermistor is constant and,
2) The temperature reported by the thermistor is lower than what the CPU is sensing.

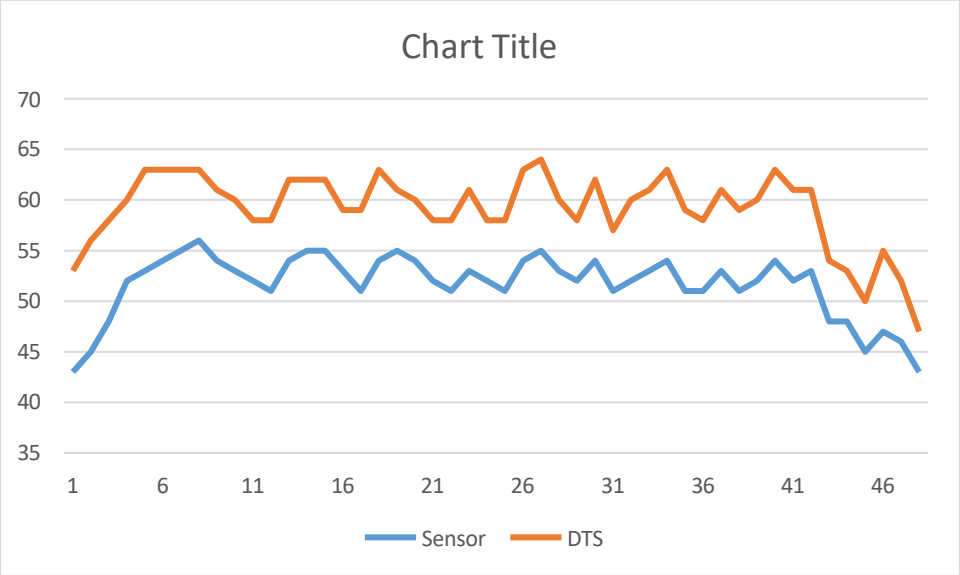

Figure 42 - Thermistor vs. Thermal Diodes values

Those two findings were considered in the code to improve the control algorithm.

3.3. Firmware Integration Test

To guarantee the correct functionality of the implemented hardware and code, we prepare a test plan that covers the different features of the PFC. The basic test plan is shown in Table 10. It includes validations that help to verify the hardware connectivity and the interaction with the code.

Table 10 - Verification Test

Item	Group	Test Name	Description	Tools	Result
1	Stand Alone	Sensor Error	Remove the Sensor and verify that the Status register flag is set. Check that the flag is activated in the serial report 'S'	Aardvark, Serial Cable	Pass
2	Stand Alone	Over-temperature error	Use the heat gun to increase the sensor temperature and verify that the status register flag is set. Monitor the temperature using the serial output and the Aardvark. Check that the flag is activated in the serial report 'O'	Aardvark, Serial Cable, Heat Gun	Pass
3	Stand Alone	Tach Error	With the fan unplugged, verify that the error flag is set. Check that the flag is activated in the serial report 'F'	Aardvark	Pass
4	Stand Alone	RPMs Check	In Manual Mode, use the push buttons to verify the tach values using the serial cable and the I2C port. Use the scope to verify the RPMs	Aardvark, Serial Cable, Scope	Pass
5	Stand Alone	Sensor Test	Verify the temperature reading using the serial cable and I2C Port	Aardvark, Serial Cable	Pass
6	Stand Alone	PWM Override	Use the I2C port to set the force mode by writing "F" to the Mode Register and verify that it is possible to change the PWM Value. Verify the PWM Percentage using the serial output and the I2C Reg. Serial output should show the Mode value too	Aardvark, Serial Cable	Pass
7	Stand Alone	F Mode - Buttons Ignored	Confirm that the board buttons do not work while in F mode. Confirm that returning to the M mode, the original values are still there	Aardvark, Serial Cable	Pass
8	Stand Alone	Setpoint Override verification	Use I2C Access to set the Override Flat in the Ctrl Reg and verify that the system uses the new setpoint to control the temperature	Aardvark, Serial Cable	Pass
9	Stand Alone	Mode Override verification	Use I2C Access to set the Override Flat in the Ctrl Reg and verify that the system responds to the new mode. Verify that the mode is updated in the serial message and that the '+' is added	Aardvark, Serial Cable	Pass
10	Stand Alone	Over-temp Output	Use the heat gun to increase the sensor temperature and verify that the output alarm is set when working in 'C' Mode. Monitor the temperature using the serial output and the Aardvark	Aardvark, Serial Cable, Heat Gun	Pass
11	Stand Alone	Manual Mode Verification	Verify that the PWM can be changed using the board buttons in Manual Mode. Verify the Mode using I2C and Serial Report	Aardvark, Serial Cable	Pass

12	Stand Alone	E2PROM Back-up	Verify that the PWM settings in Manual Mode are saved into E2PROM after 1 min of no change. Verify the values using I2C and the serial report	Aardvark, Serial Cable	Pass
13	Stand Alone	Mode Selection	Verify that we can set the operation mode using the board dipswitches. Verify the mode using I2C and Serial output	Aardvark, Serial Cable	Pass
14	Stand Alone	Code Revision	Verify the code revision using I2C	Aardvark	Pass

This test list is being shared with our external assembly houses as part of the test and validation that they apply to all the assembled parts before shipping to the different customers.

3.4. Hot Mode Testing – Self Heat Use for Mid-Temperature Range Testing

One of the greatest features of the PFC is the support for the 1xTDP and Hot mode support.

1xTDP enables the reuse of sockets across programs supporting different parts SKUs, while Hot Mode enables the mid-range temperatures test to be performed without the need of an external heating elements, since the system uses the heat generated by the CPU to keep it warm during the test.

Figure 43 shows different curves measured using different workloads with a fixed PWM percentage.

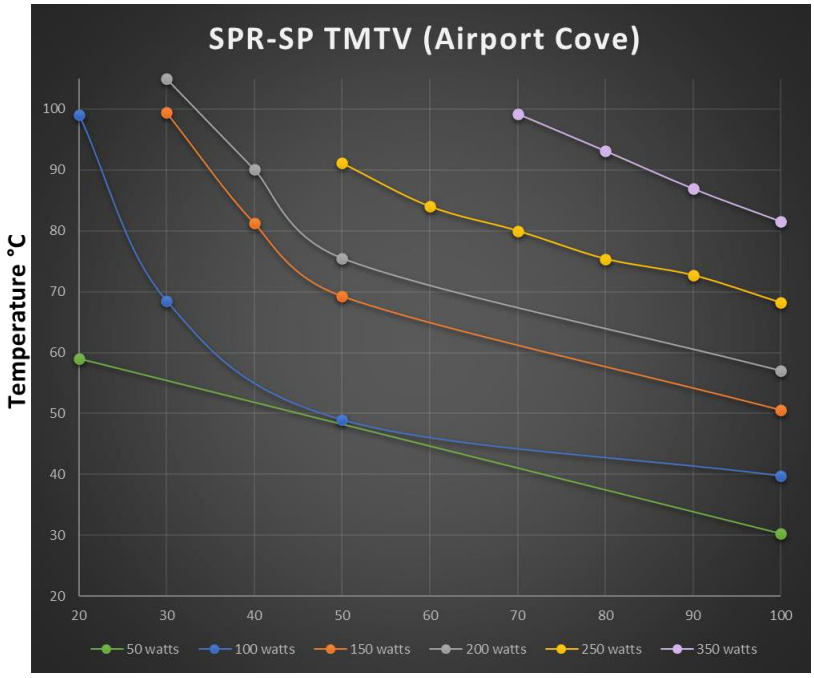

Figure 43 - CPU Self-heating based using different workloads.

Those results were the base for implementing the Hot mode in the PFC and Figure 44 and Figure 45 show our results running test on a CPU system at 328W and alternating between 328 and 220W by using a controlable load.

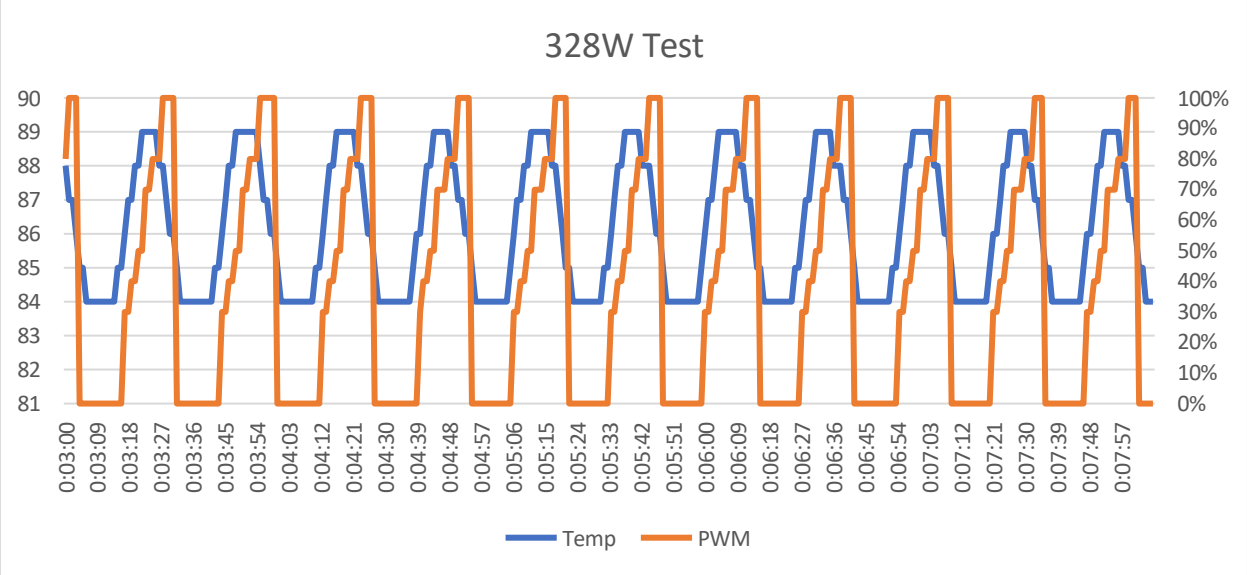

Figure 44- Hot Mode 328W test

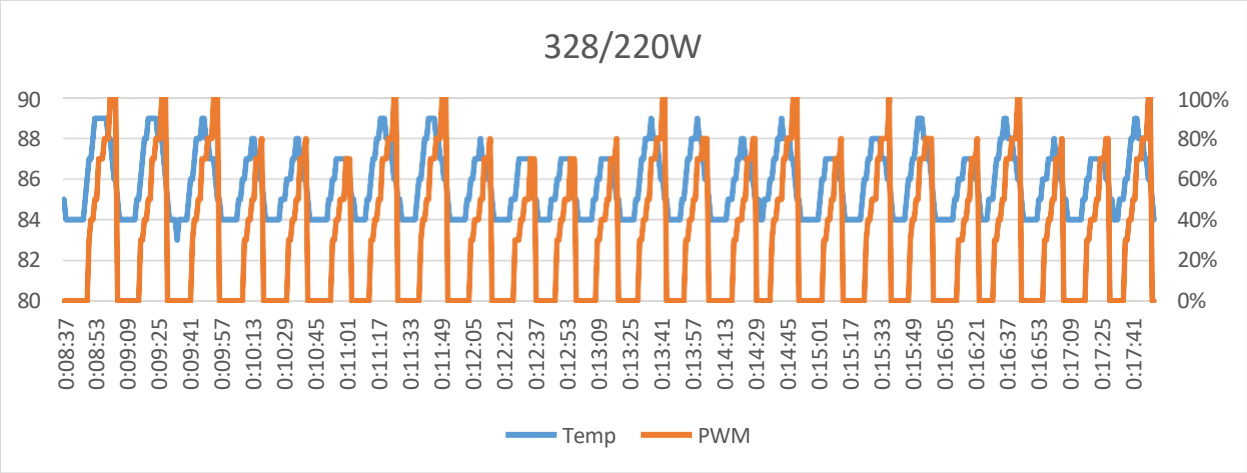

Figure 45 - Hot Mode test alternating between 328W and 220W

As we can see there, the PFC only turns the fan on when the temperature rises above the selected level.

When the system is below the test temperature the fan is off to allow the CPU to heat itself to the required test level.

4. Discussions

The PFC has been supporting projects in both segments: Client, Server and Testchips.

The initial tests were supported using Server platforms and that helped to improve the Control mode algorithm to solve the noise level issues on that segment.

After using the PFC for the first time in a Client Segment project, we got valuable feedback regarding the usage mode for Hot Mode and that triggered the design of a Rev B board. The Rev B board includes hardware enhancement that covers the received feedback. i.e.: Adding the OLED display to show data and minimize the need for the serial cable, changing from a Dipswitch to jumpers to provide easier access to the users, and removing unused features.

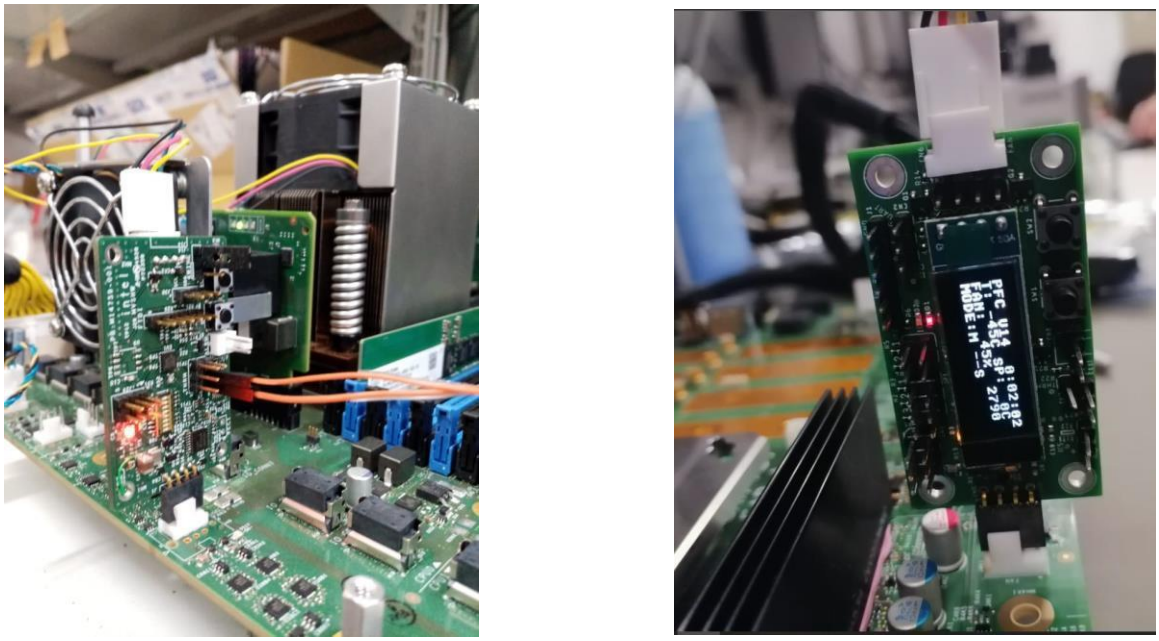

Figure 46- PFC Installed in a Server Validation Platform and an Electrical Validation Platform

4.1. What Is New In This Work?

The main differences between the PFC and other similar implementations are:

- The PFC uses the same fan connector that the board provides, so no requirements are added to the main board,
- PFC is an all-in-one solution low-cost solution that includes multiple operation modes to cover different gaps related to thermal testing and support in validation platforms.

A key aspect to the success of the PFC is that it can be implemented almost transparently in any platform, making it an easy-to-use and install option compared to other available solutions plus the implemented features that provide a flexible configuration covering different needs.

4.2. Project Impact

The PFC covers different gaps in multiple areas of CPU validation in a single and low-cost solution.

So far, over 2000 PFCs have been deployed across validation labs at Intel worldwide supporting multiple projects. It was an initiative started in Guadalajara and it is becoming popular across Intel's validation labs which had help to put Guadalajara in the spot regarding solving common problems. We have been able to:

1) Reduce the noise in our labs from 87dB to less than 65dB, which translate in 27% noise reduction, reducing the risk of hearing damage to the people in our labs,
2) Enable the reuse of heatsinks on different part SKUs in the Client segment saving engineering time and the total solution cost and,
3) Reduce the spending budget on thermal heads by enabling mid-range temperature tests by replacing the thermal heads with PFCs.

In the case of Client Segment, the new generation heatsinks are including the PFC as part the thermal kit offered.

We are starting to see the saving on time and cost, been able to design a single thermal solution to cover multiple SKUs in a product.

Our internal customers are proposing new usage modes and possible applications to the solution.

This work has been presented in different Intel internal forums and congreses as papers or presentations, i.e.:

LitePFC – A Reusable In-line Fan Speed Control To Reduce Noise In Our Labs - A Guadalajara Lab Success Story (DTTC2021),

Pluggable Fan Controller – Reusable And Configurable In-line Solution For Platforms Fan Speed Control (IMTES 2022),

A Versatile And Reusable Solution For Hot Temperature Cycling Without Thermal Head Using Fan Control iVE (Tech Week 2022).

5. Conclusions

Due to my role as a hardware design engineer, my work during the last years has been focused on schematic capture, board design, FPGA code development and validation platforms architecture definition.

In that area, there was not much room to work on embedded systems as much as I used to do in previous roles at other companies.

During the pandemic, I was invited to work in the design of a breathing machine, so I reactivated my work and involvement with Embedded Systems Design. That helped me to re-encounter my passion on designing embedded systems, so I started to get tools to test different things at home.

The embedded designed in me, wake-up and I started looking at problems at work in a different way and I started proposing different solutions using embedded systems to the problems we were facing on multiple areas, and one of those was the noise level in our labs.

By implementing this solution, I was able to apply knowledge acquired from previous companies, roles and my master's degree courses as mentioned before. I was able to interact with

different disciplines like the Thermomechanical team, understanding how they simulate the heatsink design and its implications.

During the first deployed batch I was able to use and polish my firmware design skills since during the component shortage, we were not able to get Attiny817 (8Kbytes parts) and we used Attiny417(4Kbytes). I had to optimize the code to make it fit the smaller part, keeping the main features. That helped me to learn more about how the compiler "thinks".

Since the project is covering different segments, I also learned about the different fans used and the workloads that the different validation teams use to stress and test the CPUs and the challenges they face to cover all the test plans.

My current role at Intel is validation platform architect and being able to work and cross domains to other knowledge areas helps me to be able to provide bold solutions to the validation teams while also reducing the overall cost.

As a senior technical leader, it is expected that I have a breadth of knowledge covering different disciplines besides my hardware design expertise.

Getting involved in projects like the Pluggable Fan Controller helped me to accomplish that goal while having fun playing with Embedded Systems.

As a final message to others that will like to enroll is a similar journey, I will like to hightlight the importance of deepely analyze and understand the capabilities that we can found in small and low cost microcontrollers. Nowdays, there are many different options in the market supported by software frameworks like Arduino. To me, those are good options to enable proof-of-concept, but when we talk about developing robust products, I recommend full control on the code and hardware that we are using so, in cases that we really need to provide a low-cost and reliable solution, you can optimize your implementation as much as possible.

Knowledge and deep understanding on areas like: code optmization, CPU performance and IO capabilities are crusial to extract until the last drop of efficiency on an Embedded System, so we should stay hungry and keep leaning.

Bibliography

[1] NOM-011-STPS-2001
https://www.dof.gob.mx/nota_detalle.php?codigo=734536&fecha=17/04/2002#gsc.tab=0

[2] List of CPU power dissipation figures
https://en.wikipedia.org/wiki/List_of_CPU_power_dissipation_figures.

[3] ATTINY817 Datasheets -
https://ww1.microchip.com/downloads/en/DeviceDoc/40001901B.pdf

[4] Platform Environment Control Interface (PECI) Specification, Nov. 2006 – Intel

[5] TDK Chip NTC Thermistor Datasheet -
https://product.tdk.com/system/files/dam/doc/product/sensor/ntc/chip-ntc-thermistor/catalog/tpd_commercial_ntc-thermistor_ntcg_en.pdf

[6] Steinhart-Hart Equation Coefficients -
https://web.archive.org/web/20110708192840/http://www.cornerstonesensors.com/reports/ABC%20Coefficients%20for%20Steinhart-Hart%20Equation.pdf