Spot Pyrometer Analysis and System Design

A Major Qualifying Project

Submitted to the faculty

of

Worcester Polytechnic Institute
Worcester, Massachusetts, USA

in partial fulfilment of the requirements of the

Degree of Bachelor of Science

on this day of

October 14, 2005

by

Julia Cohn

Ryan Johnson

Joseph Papp

Acknowledgements

We would like to first thank our sponsors, Analog Devices, without whom this project would not have happened. We would like to thank John Reidy and John Wynn for creating this project and providing invaluable advice and direction. We would like to thank Eamon Hynes for his advice and knowledge pertaining to infrared sensors and their optics. We would also like to thank Edward Coyne for his help in the lab and his knowledge on infrared characteristics. We would like to extend our thanks to Aine O'Gorman and Michal Brychta for their help and guidance relating to LabVIEW and evaluation boards. Finally, we would like to thank Mary McCarthy for her help in providing evaluation board support and LabVIEW software, without which our project would have been immensely more difficult.

We would also like to extend our thanks to our advisor and faculty who provided advice and guidance along the way. We would like to acknowledge the help and support provided by our project advisor Professor Rick Vaz. Finally, we extend our thanks to Professor McNeill, who provided technical advice relating to our analogue design.

Abstract

The primary goal of this project was to design and construct a spot pyrometer for Analog Devices, Inc. We examined commercially available remote temperature sensors to determine their operating characteristics and we evaluated individual IR sensors. We designed, constructed, and calibrated our own spot pyrometer using a thermopile sensor, analogue to digital converter, and a USB interface, meeting our specifications.

Executive Summary

Non-contact thermometry is necessary in many applications where contact temperature sensing is hazardous or impractical because of environmental factors such as extremely high temperatures. Infrared thermopile sensors are one type of sensor used for remote temperature sensing. Thermopiles operate by having one of their two semiconductor junctions heated or cooled by infrared radiation emitted from an object, creating a temperature difference between the two junctions. This temperature difference induces a voltage from the Seeback effect, which is proportional to the difference in temperature between the IR sensor itself and the object whose temperature is being measured. Analog Devices, Inc. is interested in entering the business of producing IR temperature sensors. The company would like to manufacture an IR sensor containing an integrated analogue to digital converter in the near future.

Our project was divided into two primary goals. The first goal was to analyse existing infrared sensing technologies. This included characterising Fluke and Raytek spot pyrometers to determine system performance, disassembling and investigating the spot pyrometer system structures, and analysing and evaluating the sensor used in the Fluke 62 thermometer and a Melexis 90247 sensor. The second goal was to design and construct our own IR thermometer. This process involves using a thermopile sensor to remotely determine the temperature of an object, conditioning the signal and converting it to digital form, exporting it to a PC, and processing and displaying the data.

In the first part of our project, we evaluated two spot thermometers on the market, the Fluke 62 and the Raytek MT2. We noted an inverse relationship between the temperature displayed on the pyrometers and the distance of the devices away from a blackbody source. Additionally, our thermal shock tests showed that it takes over an hour for each pyrometer to reach thermal equilibrium with its environment.

Our team also investigated the temperature difference vs. voltage relationship for the thermopile sensor used in the Fluke 62 device and for a Melexis thermopile sensor using an evaluation board designed by ADI. We documented the quadratic relationship between temperature difference and sensor output voltage for each sensor. We determined the time constant of both sensors to be approximately 100ms.

The next stage of our project was to design our own spot pyrometer. We evaluated design options using an AD7787 or AD7794 analogue to digital converter. Both of these ADCs have 24-bit resolution; however the AD7794 contains a built in, programmable gain stage and an input biasing mechanism.

The AD7787 design option requires signal amplification prior to the ADC. We selected the AD8551, a zero-drift operational amplifier for this purpose instead of an instrumentation amplifier because of its better thermal drift characteristics. This design option uses the AD8551 in a non-inverting configuration with a gain of around 80.

The AD7794 design option requires almost no external circuitry. We were able to directly connect the thermopile sensor to the input of the ADC and we programmed it to have a gain of 32. We used the ADC in buffered mode with the negative inputs biased to half the analogue supply voltage through a programmable option. We decided to use the AD7794 ADC for our final spot pyrometer. This design option costs about the same as the AD7787, but it requires no external amplification or biasing circuitry.

We connected our ADC directly to a Cyprus USB evaluation board provided by Analog Devices. The USB interface enabled us to use a PC as the power supply, microprocessor, and display for our device. The PC interfaces with the USB board through National Instrument's LabView software. We calibrated our device over the ambient temperature range of 0-50 degrees Celsius and a source temperature of 30-450 degrees Celsius, using a quadratic least-squares regression equation.

Our device is housed in a case designed for ADI to use the board as a demonstration to potential customers of their upcoming line of IR sensors. The sensor is surrounded by a metal casting which acts as a thermal capacitor and ensures excellent thermal conductivity between the sensor and its surroundings. A laser attached to the device allows for easy spot targeting. The end user only has to plug the device into a computer with our software on it and begin recording temperatures.

Table of Contents

Table of Figures

Table of Equations

Table of Tables

1 Introduction

Temperature sensing is utilised in many applications to determine information about an object or its environment. Traditional methods of temperature measurement involve contact temperature sensing where a sensing element touches the object of interest and reaches thermal equilibrium with the object. This is impractical in many applications, particularly those that involve extremely hot temperatures or objects in unsafe or difficult to reach areas. In such situations, it is necessary to use an alternative non-contact method to determine the temperature of an object.

One method of non-contact temperature sensing is through the use of IR radiation. A spot pyrometer, better described as a IR thermometer, measures the amount of IR radiation emitted by an object. Based on the amount of IR radiation it absorbs, the pyrometer calculates the temperature of the object. Such non-contact temperature sensing is critical in applications such as industrial manufacturing, security systems, and thermal imaging.

One particular area where non-contact temperature sensing is especially useful is in the food industry. It is important to keep foods at certain temperatures to conform to health standards. Current methods of temperature measurement involve probing food, which create additional heath risks. Non-contact temperature measurement is quicker, more sanitary and more accurate.

Analog Devices, Inc. (ADI) is interested in manufacturing non-contact temperature sensors. To accomplish this, ADI requested a characterisation of current sensors on the market so that it can compare its sensors with those of its competitors. Additionally, it is important for ADI to understand how the IR sensor interfaces with surrounding circuitry in devices such as spot pyrometers in order to better design the sensors that reside in them.

This project was divided into two parts. The first part was to evaluate the current IR technology. This consisted of evaluating two commercially available spot pyrometers in the laboratory and determining how well their performance meets their specifications. This also included evaluating a thermopile sensor and comparing it to the sensor in a commercial available spot pyrometer. The second part of this project was to design and construct a spot pyrometer so that ADI can better understand critical design requirements for future generations of IR sensors.

2 Background Information

This section contains background information that pertains to our major qualifying project. It outlines the theory behind infrared temperature sensing, describes how several IR temperature sensors work, and compares two commercially available spot pyrometers. This report assumes a general electrical engineering background and familiarity with basic engineering concepts.

2.1 Thermal Radiation and Emissivity

Remote temperature sensing is based on the principles of electromagnetic radiation. This section introduces the electromagnetic spectrum with background information on thermal radiation and describes the concept of emissivity.

2.1.1 Electromagnetic Radiation and the Electromagnetic Spectrum

Temperature is a measure of the average kinetic energy of all the atoms in an object. The atoms in any material are always moving and colliding, transferring energy between atoms. Some atoms gain energy and others lose energy. The average energy level is referred to as absolute temperature and is measured in Kelvin (K). Any material whose temperature is above 0K contains kinetic energy. Since atoms are made up of electric charges, moving atoms create an electric field that produces a magnetic field, resulting in the creation of a propagating electromagnetic wave. The entire frequency range of these electromagnetic waves is referred to as the electromagnetic spectrum (Fraden, 1999).

The electromagnetic spectrum is broken up into sections, characterised by the wavelength and intensity of the electromagnetic waves. For example, as seen in Figure 1, the infrared portion of the spectrum is between 1 mm and 750 nm and the visible light portion of the spectrum is between 750nm and 400nm (Nave, 2005).

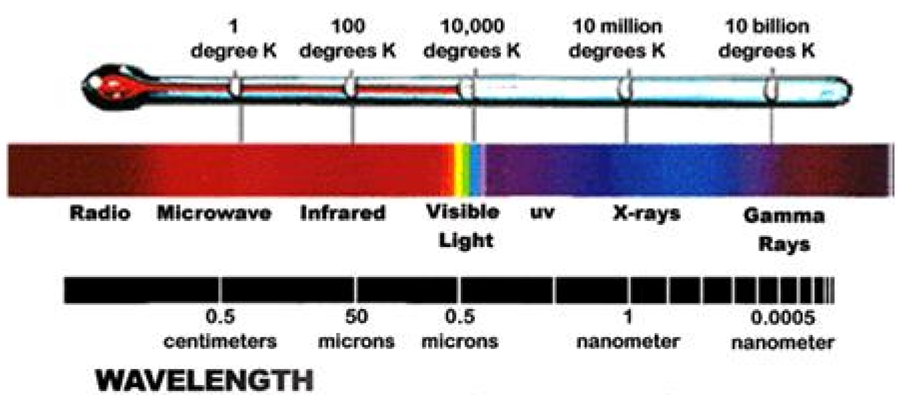


Figure 1: The Electromagnetic Spectrum (Porro and Flanagan, 2005)

The peak wavelengths of the emitted radiation decrease as the temperature of an object increases. All objects emit IR radiation, but the naked eye is unable to detect wavelengths longer than 750 nm. When the temperature increases beyond a certain point, the emitted radiation will enter the visible region of the electromagnetic spectrum where the eye can detect it. The wavelength at which the IR radiation is concentrated the most is given by Wien's Law as shown in Equation 1 (Fraden, 2004).

Equation 1: Wien's Law

$$\lambda = \frac{2898}{T}, \text{ where} \qquad (1)$$

T = temperature of the object [Kelvin]
λ = wavelength [μm]

The principles of the electromagnetic spectrum are the basis for non-contact temperature sensing. It is possible to determine the temperature of an object by measuring the wavelength and intensity of the radiation it emits. As shown in Figure 1, the hotter an object, the shorter the wavelength and the greater the intensity of the electromagnetic radiation it emits. The relationship between intensity and temperature can be determined through Planck's law, shown in Equation 2. Planck's law states that the magnitude of radiation emitted at a particular wavelength is dependent on the temperature and emissivity of the object.

Equation 2: Planck's Law (Fraden, 1999)

$$W_{\lambda} = \frac{\varepsilon(\lambda) C_1}{\pi \lambda^5 \left(e^{\frac{C_2}{\lambda T}} - 1 \right)}, \text{ where} \qquad (2)$$

$\varepsilon(\lambda)$ = Emissivity of an object
$C_1 = 3.74 \times 10^{-12} Wcm^2$
$C_2 = 1.44 cmK$

2.1.2 Infrared Absorption

Water vapour is the primary absorber of electromagnetic radiation in our atmosphere. While it absorbs most radiation, water reflects radiation at wavelengths associated with the colour blue, causing oceans and lakes appear blue to the human eye (Chaplin, 2005). The radiation that is absorbed, however, can lead to inconsistencies in non-contact temperature measurement. Water vapour absorbs large amounts of infrared radiation, with most being absorbed at wavelengths at about 3 micrometers. As a result, distance has a direct relationship to the output of an infrared sensor. The further away the sensor is from the source, the greater the amount of IR radiation absorbed by the atmosphere. This relationship is shown in Figure 2.

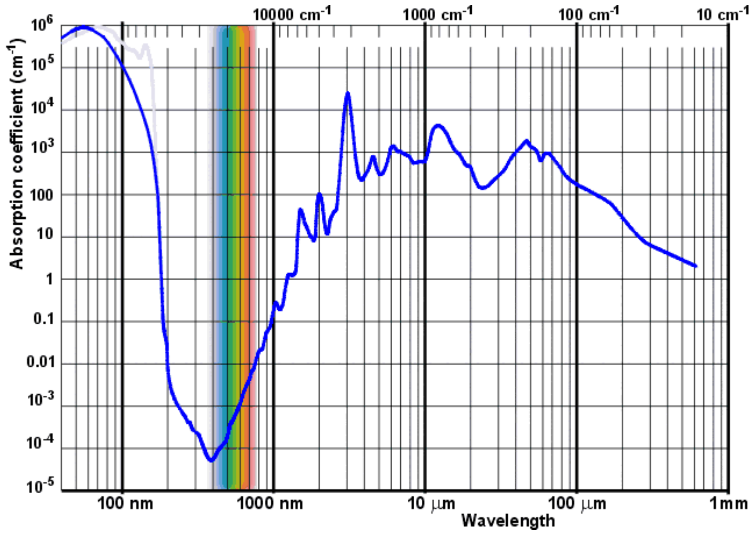


Figure 2: Water Absorption vs. Wavelength (Chaplin, 2005)

2.1.3 Emissivity

Emissivity is the ratio of electromagnetic radiation emitted by an object and that emitted by a blackbody at the same temperature (RReDC Glossary, 2005). A blackbody is an ideal surface that has an emissivity of 1 across all wavelengths, which means that it absorbs all radiation and reflects none. Real surfaces have an emissivity between 0 and 1 that varies depending on a number of factors including the particular material and the temperature it is at. While a true blackbody does not exist, there are blackbody cavities which behave close to ideal. These are often used for testing and calibrating temperature measurement equipment. Figure 3 shows the spectral radiances of a blackbody source at a range of temperatures.

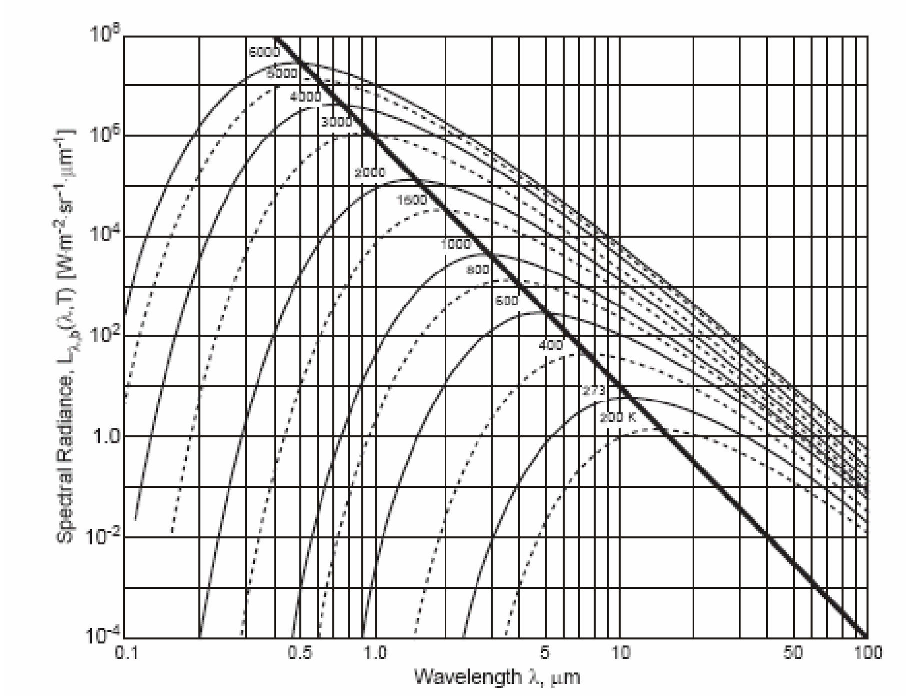


Figure 3: Spectral Densities for Blackbody Source (Fraden, 1999)

Since all objects emit electromagnetic radiation, it is possible to determine the temperature of an object based on the intensity and frequencies of its emitted wave. As seen in Figure 3, the spectral density for objects around room temperature (25 ˚C) peak at wavelengths in the IR region of the spectrum and therefore IR radiation is typically chosen for non-contact temperature measurement. There are a variety of different sensor types used in IR radiation detection which are discussed in the following sections.

2.2 Temperature Sensing Techniques

There are three main types of infrared temperature sensors: pyroelectric sensors, bolometers, and thermopiles. The following sections describe the characteristics, explain the operation, and make a comparison of each sensor type.

2.2.1 Pyroelectric Sensors

Pyroelectric sensors consist of two electrodes separated by a pyroelectric material. One of the electrodes is chopped with incoming IR radiation, causing its temperature to rise and fall, inducing a voltage from the pyroelectric effect. This effect occurs because the material is composed of tiny crystals, which become orientated along a preferred direction when a thermal flux exists within the material. When a thermal flux passes through the material, the crystals act as dipoles and generate an electrical charge. Since the material becomes polarised from the temperature gradient, pyroelectric sensors are considered heat flow detectors rather then heat detectors. In this way, pyroelectric sensors are different from thermopiles or bolometers since they detect a thermal gradient rather than a constant temperature. Pyroelectric sensors typically have a fast thermal response, since the detecting element does not need to reach thermal equilibrium with its environment. Until the past decade, pyroelectric sensors were not used in temperature measurement applications because upon exposure to high temperatures, the crystals would lose their polarisation. New materials are now used so the pyroelectric material maintains its polarity at higher temperatures and can be use in temperature measurement (Fraden, 1999).

2.2.2 RTDs and Bolometers

Another way of measuring temperature is to use a resistive temperature detector (RTD). These devices are often known as thermistors and are currently used in many applications for contact temperature measurement. The resistance of an RTD varies depending on its temperature. There are several types of metals used in thermistor construction, platinum being the most common. Although platinum is not the most sensitive metal for an RTD, it has the most consistent and linear resistance vs. temperature curve over the largest temperature range. Copper and nickel can also be used and are more sensitive than platinum, but they are unstable at higher temperatures (Burns, 1999).

Bolometers are small RTDs integrated with infrared absorption material that detect electromagnetic energy from the microwave to near-infrared range. The infrared absorption material heats up when exposed to IR radiation and raises the temperature of the bolometer. The temperature is calculated by measuring the change in electrical resistance of the thermistor. Recently, infrared bolometers have been constructed using a thin resistive film with a relatively large area. One issue still being addressed with bolometers is their slow response time since the thermistor needs time to heat up and cool down. Bolometers can also be used in thermal imaging in the form of resistive arrays (Fraden, 1999).

2.2.3 Thermocouples and Thermopiles

Thermopile sensors are a type of passive infrared detector. They consist of serially connected thermocouples whose hot junctions are thermally connected together and whose cold junctions are also thermally connected together. Each individual thermocouple produces an electric potential from phenomena known as the Seeback, Peltier, and Thompson effects; however, only the Seeback effect is significant in practical thermometry (Fraden, 2004).

The Seeback effect establishes an electric field inside a conductor containing a thermal gradient. The total voltage induced by this effect is proportional to the temperature difference between the hot and cold regions as well as a factor called the absolute Seeback coefficient. The Seeback coefficient depends on the type of material the conductor is made from. The fundamental equation for the Seeback effect is shown in Equation 3.

Equation 3: Seeback Effect

$$dE = \sigma \cdot dT \text{ , where} \qquad (3)$$

dE = differential induced Seeback emf
σ = absolute Seeback coefficient
dT = temperature differential

Since these infinitesimally small voltages are integrated over the length of the wire, the distribution of the temperature gradient along the wire has no effect on the total terminal voltage.

It is only the net temperature difference between the hot and cold junctions that affects the induced voltage (Reed, 1999).

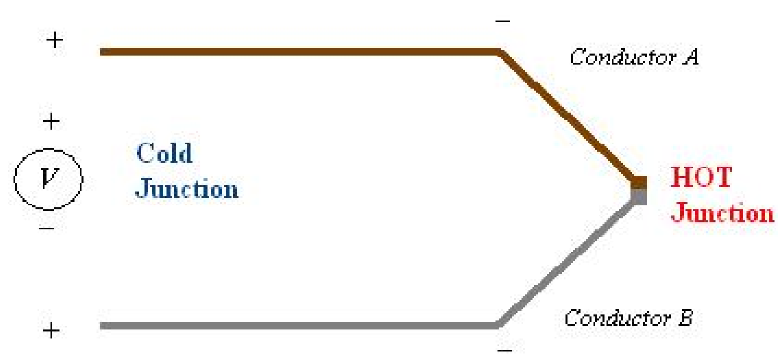


Figure 4: Thermocouple structure

The structure of a thermocouple is formed when two materials of differing Seeback coefficients are joined together as shown in Figure 4. Material A forms an electric potential in the opposite direction as material B, but since the Seeback coefficients are different, the two induced voltages are not the same and there is a net potential difference at the two cold ends of the materials. The voltages induced by each hot-cold and cold-hot segment would cancel each other out if the materials had the same Seeback coefficient (Reed, 1999).

A thermopile consists of many thermocouples connected together in series, with the hot junction containing IR absorbing material that heats up when exposed to IR radiation. Since each thermocouple produces a very small terminal voltage, connecting them in series increases the magnitude of the voltage produced by the overall sensor. However, thermopiles measure only the difference in temperature between the hot and cold terminals. To determine the absolute temperature of the hot junction, an ambient temperature sensor is also needed to measure the temperature of the cold terminal and add it to the temperature difference between the two terminals (Fraden, 2004).

2.3 Spot Pyrometers and the Distance to Spot Ratio

Fluke and Raytek are two of the major manufacturers of spot pyrometers. The following section explains the distance to spot ratio, an important parameter for characterising spot pyrometers as well as some basic optics involved in spot pyrometers. The next section describes the technical specifications of the Fluke and Raytek devices that we evaluated in the lab.

2.3.1 Distance to Spot Ratio and Optics

Spot pyrometers use infrared radiation to estimate the temperature of an object at a distance. In modern spot pyrometers, thermopiles are the preferred sensor, while in the past, devices often implemented pyroelectrics. These pyrometers use a single sensor, rather than an array, to take an average temperature measurement of a single "spot". Thermal imaging cameras use arrays of sensors to produce a real-time thermal image of an object.

One specification that is particularly important in spot pyrometers is the distance to spot ratio, often referred to as the D:S ratio. The distance to spot ratio is a direct consequence of the optics design in the spot thermometer. The lens system of the device focuses the infrared sensor on a given area a distance away. While the D:S ratio is theoretically limitless in the optics design, sensor sensitivity limits the D:S ratio from increasing beyond a certain point. As the spot size becomes smaller, the amount of thermal energy that the sensor receives is also reduced. For this reason, an extremely sensitive sensor is necessary in order to achieve high D:S ratios. Figure 5 shows how the spot size increases with increasing distance from the pyrometer.

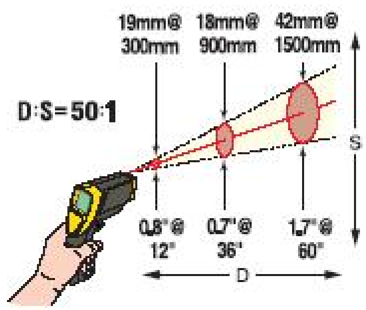


Figure 5: Distance to Spot Example (Fluke, 2005)

The spot size is defined through the use of an optical lens. Convex, concave, or a combination of both can be used in the design. As shown in Figure 6, convex lenses refract electromagnetic energy from a distant source to a specific point. A concave lens, on the other hand, will divert the path of light waves away from a specific point (Research Machines, 2005). In IR thermometers, the spot size is adjusted by changing different variables of the lens, including the amount of curvature, the aperture length, and the focal distance to the lens.

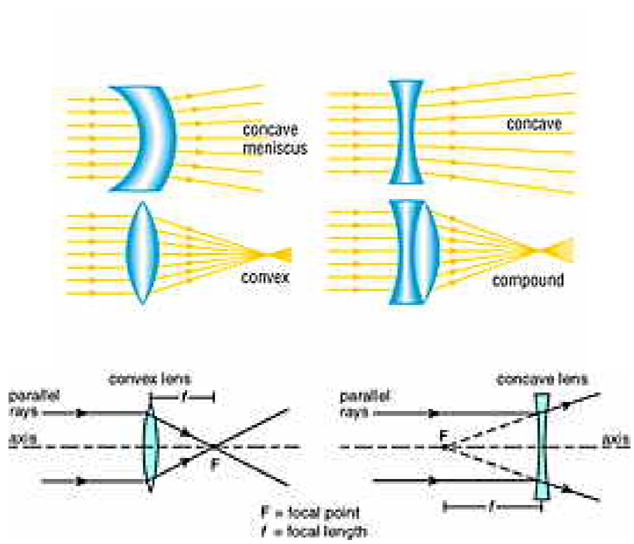


Figure 6: Lens Types (Research Machine, 2005)

The Fresnel lens shown in Figure 7 is typically used in spot pyrometers and other detector applications. It is a plano convex lens that has been collapsed down to a flat lens, but still maintains the same optical characteristics. It is a low cost solution and often eliminates the need for an additional window (Hartmann, 2003).

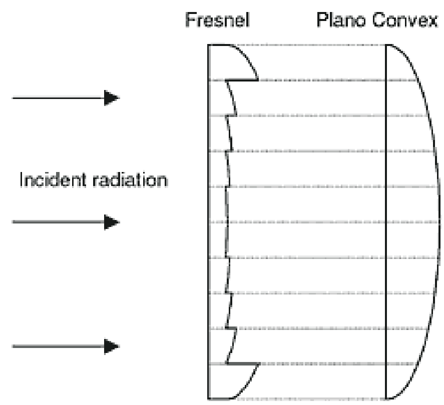


Figure 7: Fresnel Lens (Hartmann, 2003)

2.3.2 Fluke and Raytek Spot Pyrometers

In some environments, non-contact temperature sensing is much more practical than contact means. For instance, in industrial settings, non-contact temperature sensing is often essential for estimating the temperature of extremely hot objects or objects that are difficult to reach. One possible use for spot pyrometers is utility workers to determine the temperature of a transformer without making contact with it. The food industry can also benefit by testing the temperature of food without using a contact temperature probe.

For this project, we investigated and tested products from two companies: Fluke and Raytek. Both companies are part of the same parent company, Danaher Corporation. In 2002, Danaher acquired Raytek Corporation for approximately $75 million (ISA, 2002). The models that we investigated for this project were the Fluke 62 and the Raytek MT2. Both models are low to mid-range in the market and sell for an average of $100 USD.

The Fluke 62 sells for slightly more than the Raytek MT2. The Fluke 62 has a measurement range of -30 °C to 500 °C. Like most spot pyrometers, the Fluke 62 uses a laser to assist in targeting. The distance to spot ratio (D:S, described in section 2.7) is the ratio of the distance the pyrometer is from an object to the area on the surface of an object that the pyrometer senses. The D:S ratio of the Fluke pyrometer is specified to be 10:1 and it boasts an accuracy of ±1.5%. The pyrometer also has a resolution of 0.2° and has a response time of 0.5 seconds. It is powered by a single 9 Volt battery (Fluke, 2005).

Figure 8: Fluke 62 Handheld Thermometer (Fluke, 2005)

The Raytek MT2's specifications promise slightly inferior performance than the Fluke 62. It has a measurement range of -18 °C to 275 °C. The MT2 has a laser to assist targeting, much like similar spot pyrometers. The MT2 also has a slightly smaller D:S ratio than the Fluke 62, at 8:1. Raytek claims less accuracy than the Fluke, at ±2.0%. This spot pyrometer has a resolution of 0.2° and claims a response time of 0.5 seconds. It is powered by a 9 Volt battery (Raytek Corp., 2005).

Figure 9: Raytek MT2 Handheld Thermometer (Raytek Corp., 2005)

3 Goals and Specifications

Our first goal for this project was to evaluate IR sensing technology currently on the market for non-contact temperatures measurement. We used two commercially available spot pyrometers, the Fluke 62 and Raytek MT2, as samples of current technology. In order to accomplish this goal, we:

- Analysed the Fluke and Raytek thermometers to determine their system performance
- Disassembled and investigated the Fluke and Raytek system structures
- Characterised and evaluated the sensor used with the Fluke system and a Melexis sensor

Our next goal for this project was to construct a handheld infrared thermometer capable of measuring a temperature, conditioning the signal, and sending it to a PC for analysis. Considering this goal, we:

- Measured a controlled temperature using a Melexis IR thermopile sensor
- Conditioned the signal to a digitally convertible form
- Digitalised the signal and exported the data to a PC
- Processed the data to display a temperature on a computer screen in degrees Celsius

Based on this goal, the following are specifications for our spot pyrometer which are similar to the Fluke 62, as well as other similarly priced spot pyrometers on the market. This product was designed to serve as a demonstration tool for ADI engineers when demonstrating capabilities of their IR sensors to potential customers. Our prototype used a PC as a microcontroller and a computer screen in substitution for an LCD screen to avoid the initial cost and waiting time of a custom microcontroller and LCD screen. Our infrared thermometer must:

- Be reasonably small, eventually small enough to be hand-held
- Accurately read the temperature of an object from a distance, to the accuracy of ±1%
- Have a distance to spot ratio of at least 5:1
- Have a response time less than 500 ms
- Have a temperature range of at least -25 to 450 ˚C
- Have a resolution of 0.2 ˚C
- Have automatic ambient temperature adjustment
- Process the information in computer software and output the resulting temperature to the screen
- Have a final retail price of under $100.00
- Be powered by a 5V USB supply
- Have laser assisted targeting

4 System and Sensor Performance Evaluation

To design sensors that are competitive in the current market, we must first evaluate the performance of the sensors in a complete system. Once whole systems were characterised, an investigation of the sensors themselves provided additional information useful in designing future sensors. This first part of this section includes the testing procedures and results for the complete Fluke and Raytek systems. The next section briefly addresses the system layout of the Fluke 62 IR Thermometer. The last part of this section explains the testing procedure and results for the Melexis sensor and the sensor from the Fluke pyrometer. The complete results for the following tests are located in Appendix A.

4.1 System Performance Evaluation

A complete system evaluation makes it possible to understand the specifications IR sensors must adhere to. We conducted tests to determine how closely the Fluke and Raytek systems meet the specifications listed on their product data sheets. This section includes an explanation of how we conducted each of the six different tests along with their corresponding results.

4.1.1 Thermal Response Tests

To determine the accuracy of the Fluke and Raytek pyrometers over a broad temperature range, we conducted thermal response tests. We measured the temperature of the IR blackbody source with the Fluke and Raytek pyrometers a fixed distance away. In the initial test, we placed the lens of the pyrometer 25.5 cm away from blackbody cavity entrance. We chose this distance because the opening of the blackbody is 7.5 cm wide and at 25.5 cm the spot size is well contained within the blackbody. We recorded the temperature displayed on the pyrometers from 40°C to 300°C in 5° increments.

Next, because the Raytek pyrometer has a D:S ratio of 8:1 while the Fluke has a D:S ratio of 10:1, we completed a second thermal response test on the Raytek pyrometer with it 20% closer to the blackbody. This way, the Raytek device had the same spot size that the Fluke device had during the first set of tests. As shown in Figure 10, there was an error of up to 5% with each thermometer even though the specifications for the Fluke and Raytek devices are 1.5% and 2% respectively.

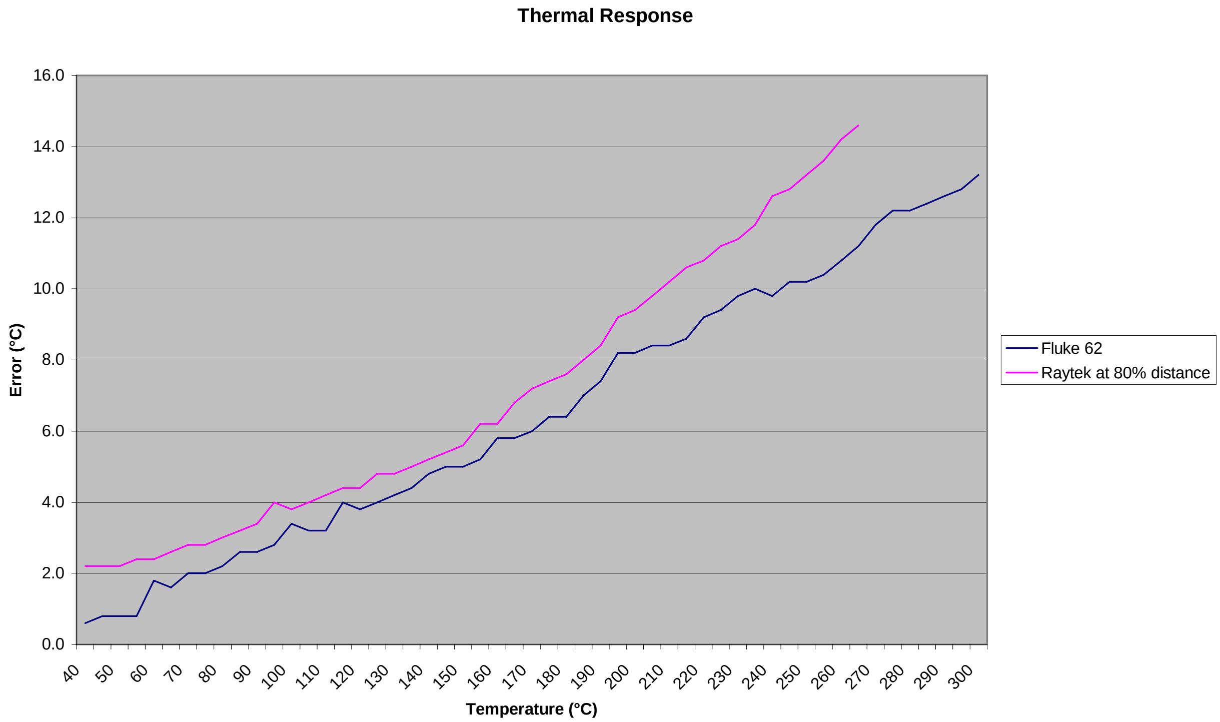


Figure 10: Thermal Response Test

4.1.2 Distance Tests

After completing the thermal response tests, we observed that the temperature displayed on the pyrometer changed depending on the distance of the system to the blackbody source. Based on this observation, we conducted a series of distance tests. In these tests, we varied the distance of the pyrometers from the IR source while keeping the blackbody temperature constant. We tested the responses at 50°C, 100°C, 150°C, and 200°C while moving the pyrometer away from the blackbody cavity in 2 cm increments from 2 cm to 80 cm.

The results show that there is an optimal distance at which the temperature measured by the pyrometer exactly matches the temperature of the blackbody source. As shown in Figure 11, the optimal distance for the Fluke and Raytek pyrometers is 42 cm and 45 cm respectively. Using this information, we performed an additional set of thermal response tests at this optimal distance from the source.

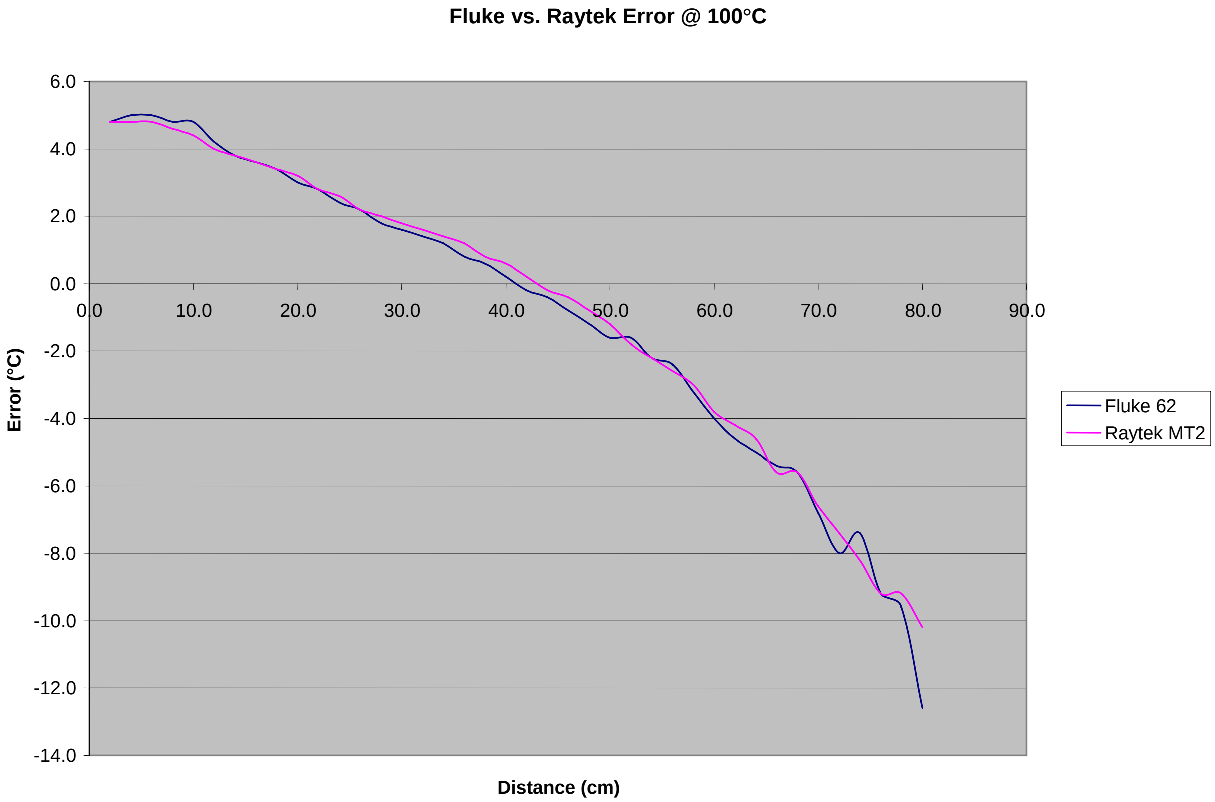


Figure 11: Distance Test

4.1.3 Optimal Thermal Response Tests

After noting the direct relationship between distance and measured temperature described in the previous section, we decided to again perform a thermal response test at the optimum distance for each thermometer. We positioned the Fluke pyrometer 42 cm from the blackbody and the Raytek pyrometer 45 cm from the blackbody. We varied the temperature from 40°C to 300°C for the Fluke and 40°C to 270°C for the Raytek, taking measurements every 10°C.

As shown in Figure 12, we found that at their optimal distances, the Fluke consistently had an error under 1% and the Raytek consistently had an error under 2%, both meeting specifications. The devices could have been calibrated at this distance since they become more inaccurate as the distance increases or decreases away from this point. One possible cause of this relationship between distance and measured temperature could be the absorption of the infrared radiation by water vapour present in the air.

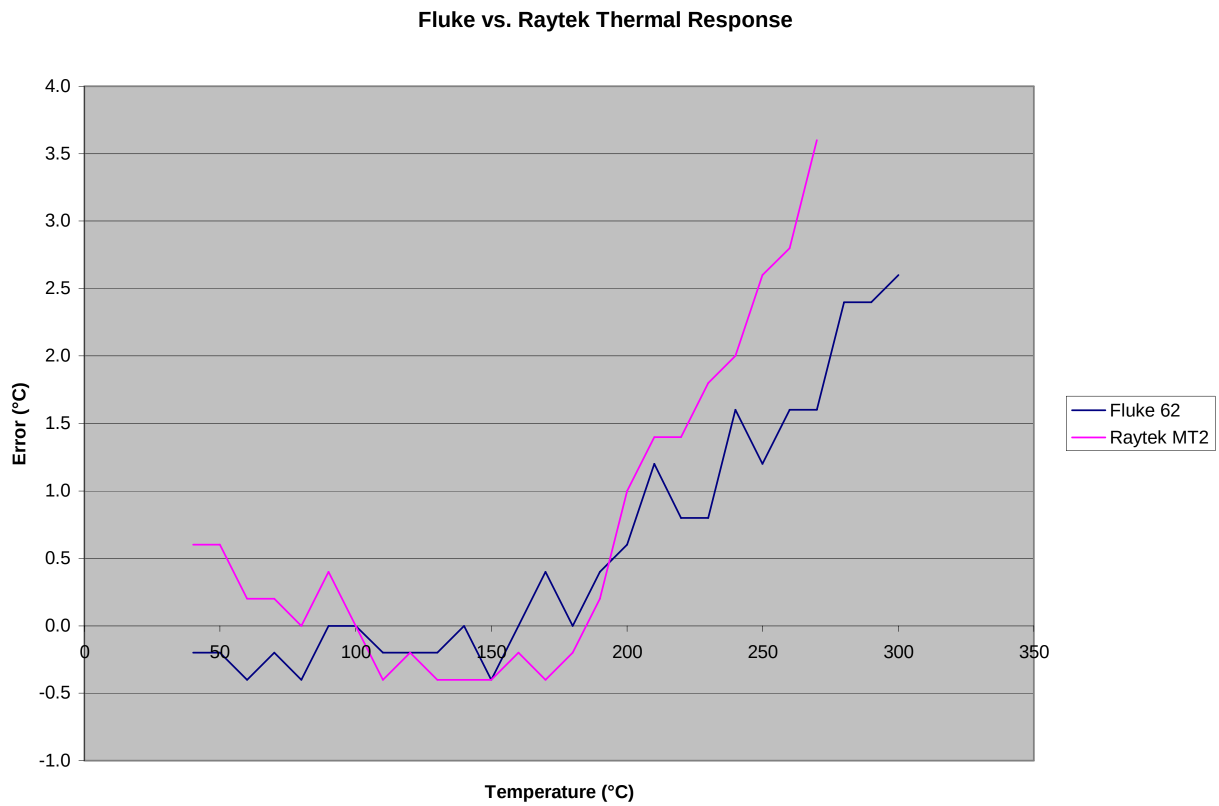


Figure 12: Optimal Thermal Response

4.1.4 Field of View Tests

The next test we conducted was a field of view test. The purpose of this test was to verify the distance to spot ratio specifications on each pyrometer. The field of view was recorded in two different ways: as an angle offset and as a position offset with respect to the pyrometer facing directly towards the blackbody. To determine the angle offset, we placed each pyrometer on a stand and placed a printout of a protractor directly below it. We varied the position of the stand while recording the temperature and orientation, starting at the edge of the blackbody box and proceeding until the pyrometer was facing directly into the cavity. At the same time, we taped a ruler below the casing of the blackbody and measured the position offset from the centre of the blackbody to centre of the laser sight. We were careful to remove the ruler from the source when taking temperature measurements to ensure it was not shielding any IR radiation. For this test, the pyrometer was placed 37.1 cm from the blackbody.

We estimated the distance to spot ratio first using the displacement data. We estimated the points of fringing on the Fluke pyrometer to be at approximately 5.25 cm and 1 cm from the centre of the blackbody cavity. The D:S ratio was then calculated:

$$D:S = \frac{Dist}{SpotDiameter} = \frac{37.05}{(5.25-1)} \cong 10.15$$

We estimated the points of fringing on the Raytek pyrometer to be at approximately 5.33 cm and 0.68 cm. The D:S ratio of the Raytek was then calculated:

$$D:S = \frac{Dist}{SpotDiameter} = \frac{37.05}{(5.33-0.68)} \cong 7.97$$

Next, we confirmed our estimations by using the recorded angle data. Our results for each thermometer are shown in Figure 13 as a function of angle. Pointing directly into the blackbody is defined as 0°. We calculated the displacement of each angle using laws of trigonometry and verified the displacement estimations. The complete set of data is found in Appendix A.

$$SpotDiameter = dist * \tan(\angle * \frac{\pi}{180})$$

$$D:S = \frac{Dist}{SpotDiameter} = \frac{37.05}{37.05 * \tan(\angle * \frac{\pi}{180})} = \cot(\angle * \frac{\pi}{180})$$

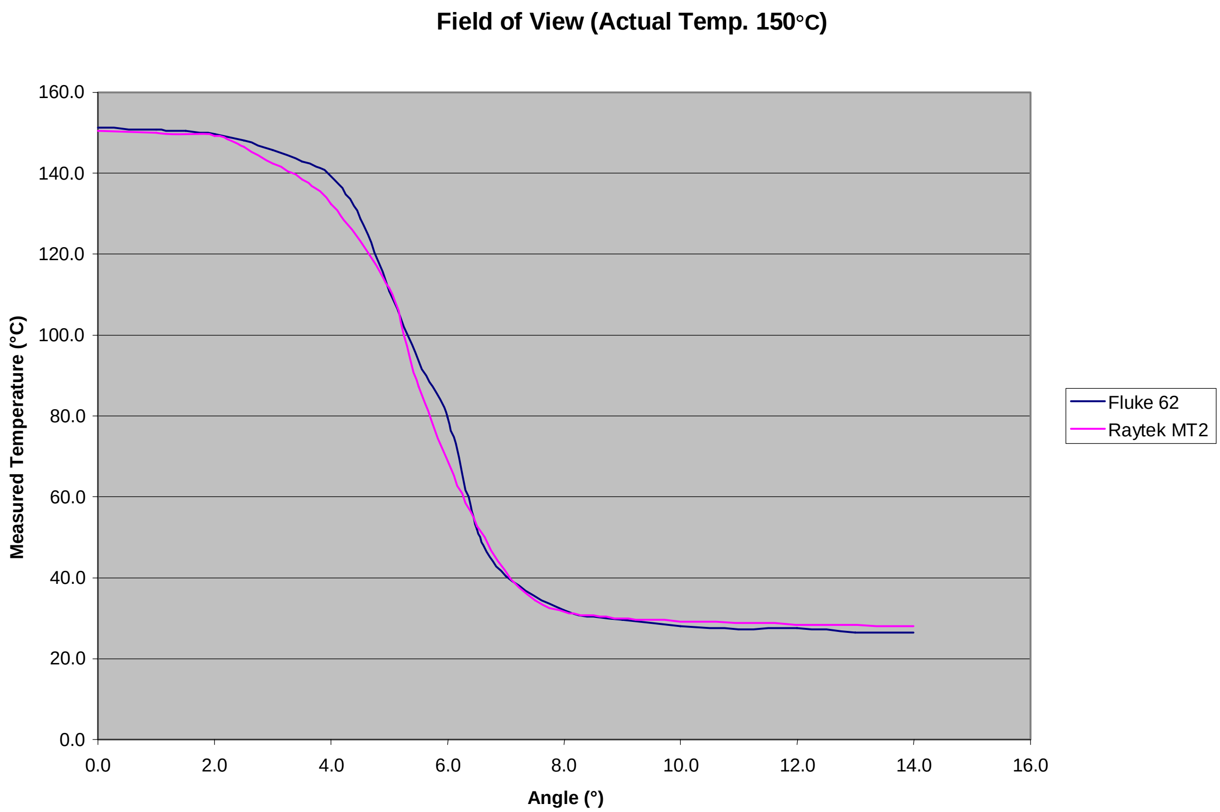


Figure 13: Field of View Test

4.1.5 Thermal Shock

Thermal shock is the effect that occurs when a device is exposed to one temperature and then used in an environment with an ambient temperature different than its own temperature. The temperature of the thermal casing needs to reach equilibrium with the IR sensor in order to record a correct measurement. To measure the effect of thermal shock, we placed the pyrometers in an environment colder then room temperature. We stored the Raytek pyrometer in a refrigerator (around 4°C) and the Fluke pyrometer in a freezer (around -20°C) overnight. The following day we took each pyrometer into an environment at room temperature (around 21°C) and recorded the temperature displayed on the device as a function of time. We began by recording the temperature of the room with the Raytek pyrometer. The pyrometer was placed in a stand with the trigger pressed in once at the beginning of the experiment and it remained consistently pressed in throughout the experiment. As seen in Figure 14, initially the pyrometer predicted the temperature was higher then the actual temperature of the room. As the device warmed up, it continued to compare its own temperature to the temperature of the wall. When it finally reached thermal equilibrium after 70 minutes, the pyrometer displayed that the temperature of the wall was at around 4°C, which was the ambient temperature value stored in the pyrometer when it was first triggered. This observation provides insight as to how the Raytek pyrometer performs its ambient temperature calculations. The device samples the ambient temperature once when the trigger is depressed and uses that value for ambient temperature calculations until the trigger is released and pressed in again.

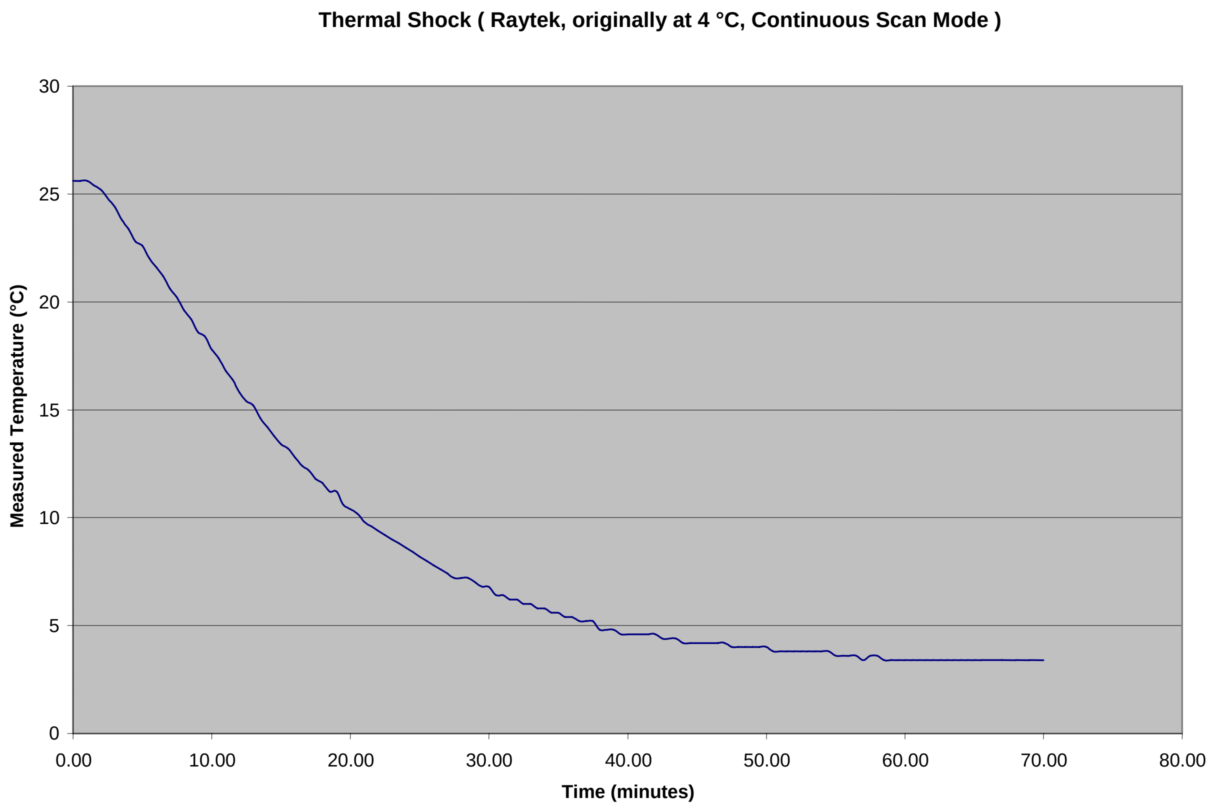


Figure 14: Raytek Thermal Shock

For the Fluke thermal shock test, we depressed the trigger each time we took a measurement. Similar to the Raytek pyrometer, the Fluke pyrometer initially overestimated the temperature of the room, as shown in Figure 15. Since we pressed the trigger each time we recorded the data, the pyrometer compared its temperature to the correct ambient temperature of the room with each reading. When trying to determine the temperature of the wall, the device overestimated and then underestimated the temperature of the room and took about 70 minutes to settle down.

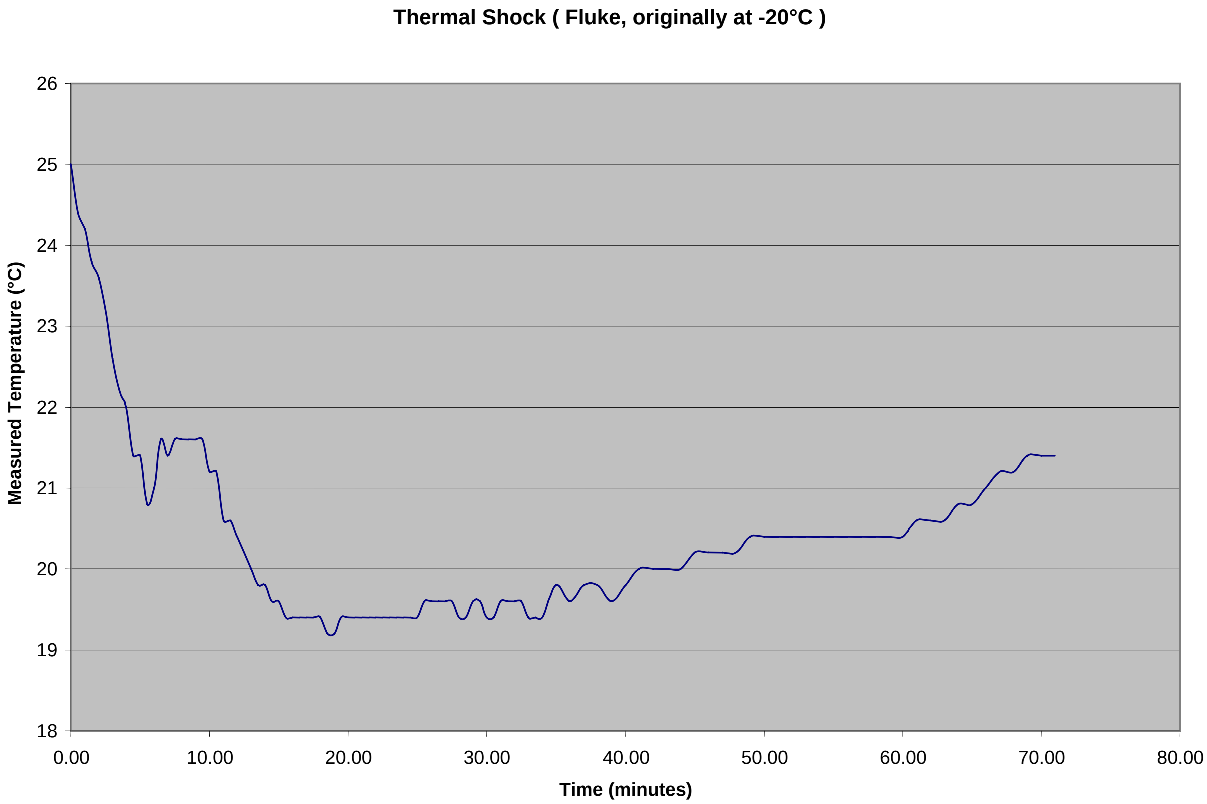


Figure 15: Fluke Thermal Shock

4.1.6 Thermal Non-Equilibrium

The final system test was to determine the effect of a thermal gradient on the pyrometers. Starting at room temperature, we placed the front of each device inside the blackbody and heated only the front for 10min at 200°C. Then we measured the temperature of the wall every 30sec until the pyrometers reached thermal equilibrium (around 21 °C). Figure 16 shows the results of the thermal equilibrium test for both the Fluke and Raytek pyrometers. The graph shows the time required by the thermal mass surrounding the sensor to reach equilibrium with the ambient temperature.

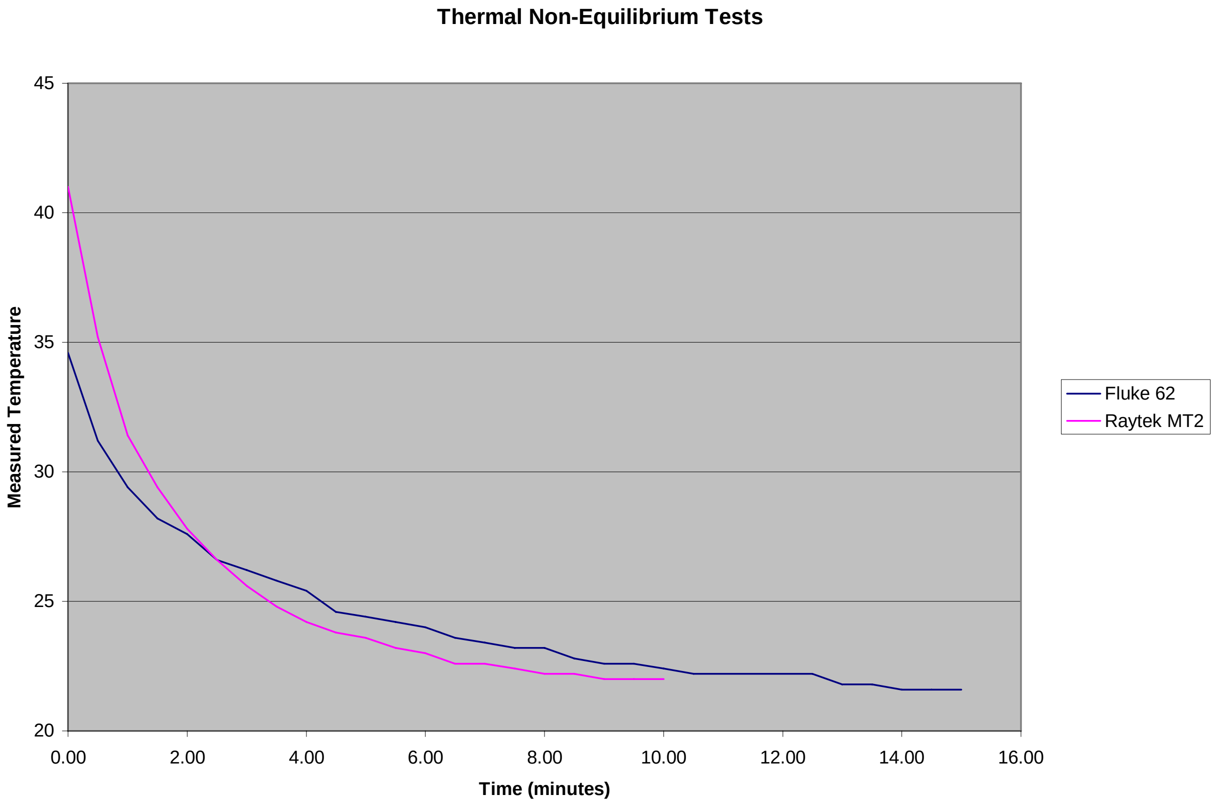


Figure 16: Thermal Non-Equilibrium Test

4.2 Fluke 62 System Layout

In order to understand how the Fluke 62 system works, we disassembled the device, documenting the process along the way. After initially disassembling the device, we noted that the circuitry was simple and controlled entirely by a single ASIC. This ASIC was labelled only as "R32260A," with no indication of the manufacturer. The sensor was connected directly to the ASIC with no signal conditioning or amplification beforehand. We noted that the ASIC internally drove the LCD screen, as there was no external LCD driver circuitry. A picture of the Fluke 62 circuit board is shown in Figure 17.

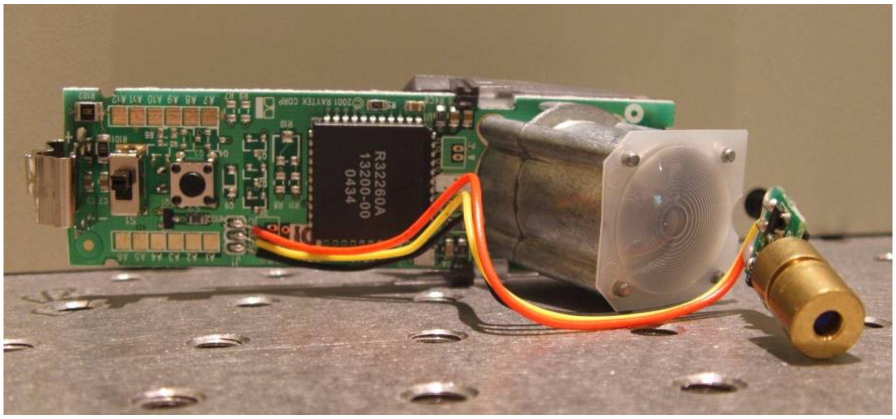


Figure 17: Fluke 62 System

The Fluke 62's circuit board contains a single button that is pushed when the trigger is squeezed. This button tells the ASIC when to begin sampling. There is also a switch that selects between units of Celsius and Fahrenheit. Finally, there is a laser that is triggered by the ASIC and driven by the main power source which is used to assist the user in targeting.

We next investigated the device's optics system and the sensor. The lens appeared to be a Fresnel lens manufactured from a type of polymer. We observed a die-cast metal base surrounding the sensor. This base prevents sudden changes in ambient temperature and shields the sensor from stray infrared radiation. The optics system is discussed in further detail in section 5.1.1.

Lastly, we removed the optics from the circuit board in order to inspect and test the sensor by itself. To perform these tests, we first removed the LCD screen that was attached to the PCB with glue. The metal casting was taken off by removing two screws. We noted that the sensor had four pins similar to the Melexis MLX90247. However, the aperture diameter of the Fluke sensor was roughly half the size of the Melexis'. This aperture, in conjunction with the optical system, determines the distance to spot ratio of the device. We noted an approximate ratio of 10:1 between the distance from the sensor to the lens and the diameter of the sensor aperture, agreeing with the device's D:S specification.

Because the ASIC controls almost the entire device, we were unable to determine what methods Fluke uses for amplification, filtering, analogue to digital conversion, or processing from disassembling it alone. We did learn valuable information on the optical system and the sensor itself, described in the following section.

4.3 Sensor Performance Evaluation

The next step in our evaluation process involved understanding the characteristics of the sensors themselves. Three characteristics that are of particular interest with IR sensors are thermal response, thermal time constant, and the wavelengths of IR radiation that the sensor is sensitive to. This section explains each of the three tests we conducted to evaluate these characteristics.

4.3.1 Sensor Thermal Response

In the first test, we determined the thermal response of two sensors: a Melexis 90247, and the sensor extracted from the Fluke 62 thermometer. Since each sensor is a thermopile, a direct voltage reading is not practical. A thermopile outputs a voltage of approximately 30 µV per °C. This voltage cannot be accurately detected using a conventional voltmeter, so we decided to use the sensor evaluation board that was developed by ADI and WPI students in 2004. This evaluation board amplified and conditioned the signal into a readable form and transmitted the signal through a USB interface to a PC. Using software we were able to output the data onto a real-time graph on the PC.

For this test, we positioned the sensor approximately 7 cm from the opening of the blackbody. We increased the temperature from 40°C to 400°C, taking a voltage measurement every 10°C. The results of this test are shown below in Figure 18. We noted that this relationship was not as linear as we expected it to be. We also found that the Melexis sensor was more sensitive than the Fluke sensor, emitting a larger voltage per °C increase. This could be due to the larger aperture of the Melexis sensor.

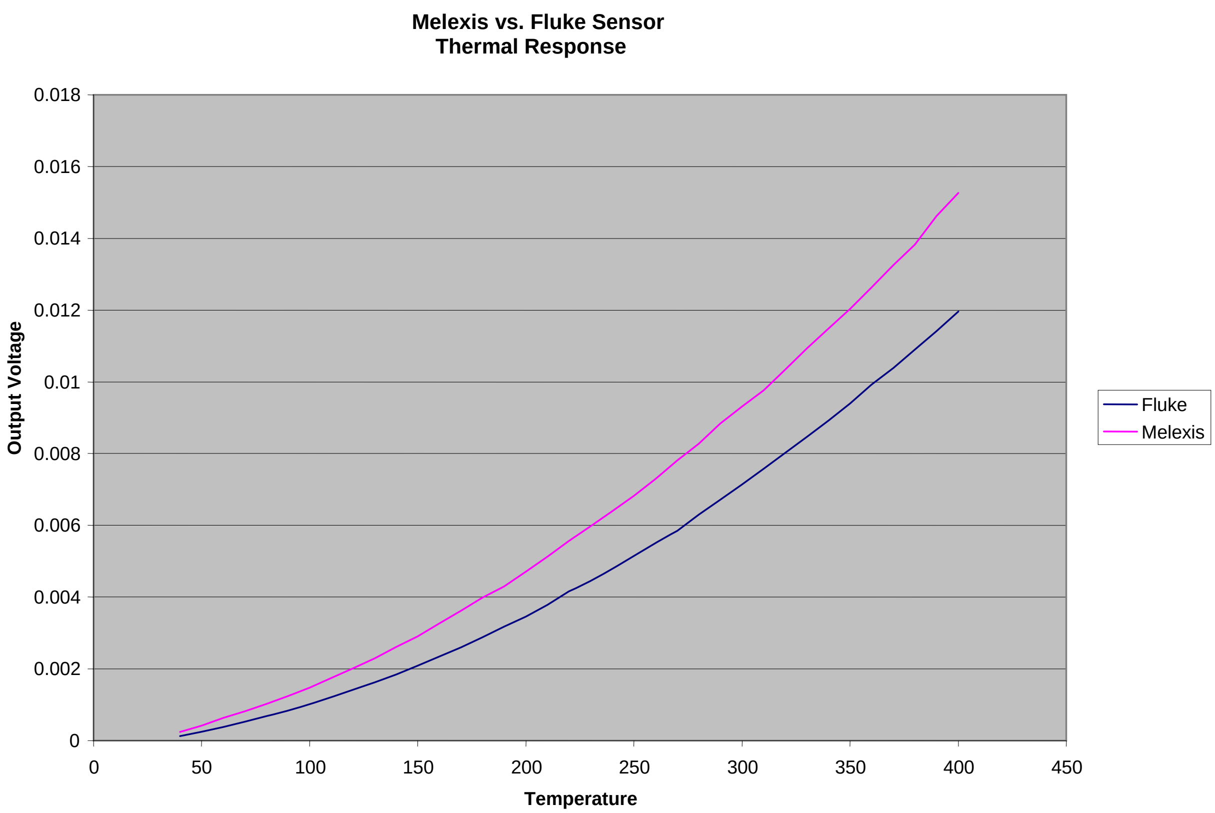


Figure 18: Sensor Thermal Response

4.3.2 Sensor Time Constant

Every thermopile has a time constant associated with it. This time constant indicates how long one must wait to obtain a stable output reading. In the next set of tests, we calculated an approximate time constant for the two sensors.

For our first test, we placed a chopper in front of the sensor and pointed it towards the blackbody. The sensor was approximately 20 cm from the blackbody and we started the chopper at a frequency of 15 Hz, the lowest possible setting allowed by the chopper. We then ramped up the frequency and took readings every 1 Hz using ADI's sensor evaluation board. After the data was taken, we calculated the actual exposure time to IR radiation. The sensor was exposed to IR radiation for half of each cycle and the other half of each cycle it was exposed to the chopper blade. While we plotted and analysed a part of the time response, unfortunately the chopper would not reach a low enough frequency to see the entire response needed to calculate the sensor time constant. A plot of this test is shown in Figure 19.

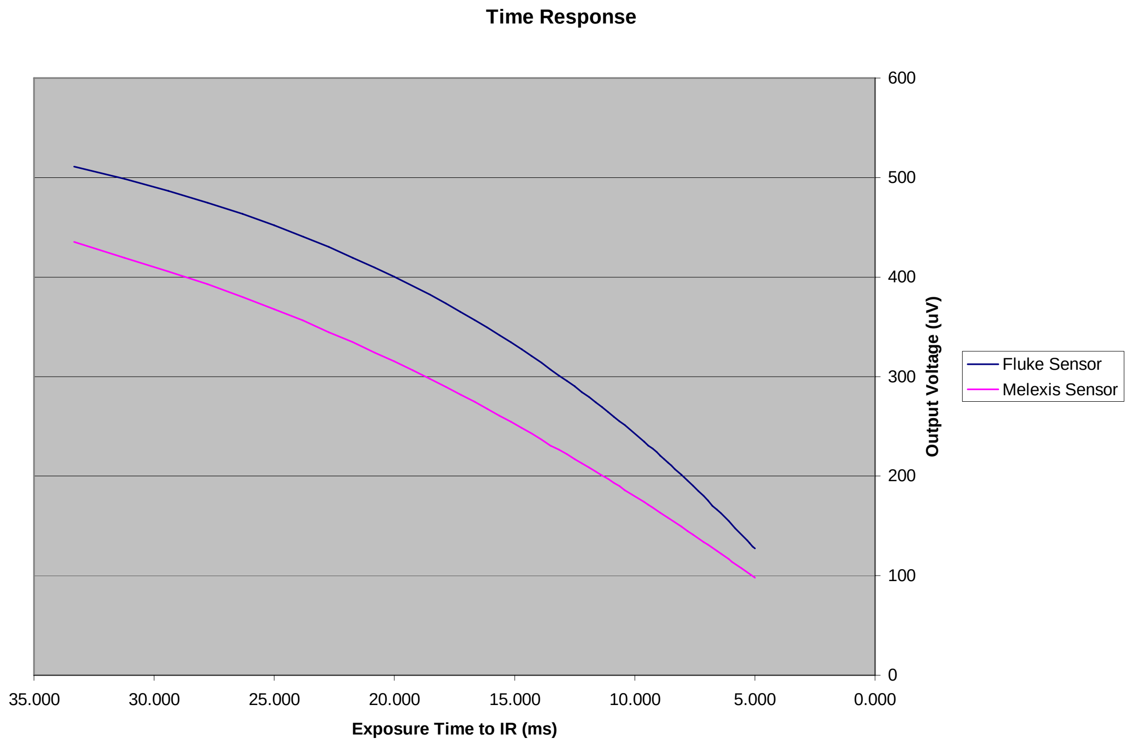


Figure 19: Chopper Time Response Test

Since the chopper cannot operate at low enough frequencies, we decided to use the evaluation board to determine the sensor time constant. The first step was to determine what each unit of time represented on the output graph of the evaluation software. Our group noted that although the graph plotted as a function of time, each horizontal unit did not correspond to one second. We used a stopwatch to time exactly 60 seconds of data acquisition. After the 60 seconds, we stopped the program and observed that 375 units of time had passed on the graph. Therefore, each marking on the horizontal scale represents 60/375=0.16 seconds.

The next step was to determine the time response of the system. The evaluation board consisted of multiple stages including an amplifier, a low-pass filter with a cut-off frequency of 0.5 Hz, an analogue to digital converter, a multiplexer, and a USB interface chip, all possessing time constants of their own. We pulsed the system with 15 mV directly from a power supply and measured both the rise and fall times of the system. The rise time of the system is shown below in Figure 20.

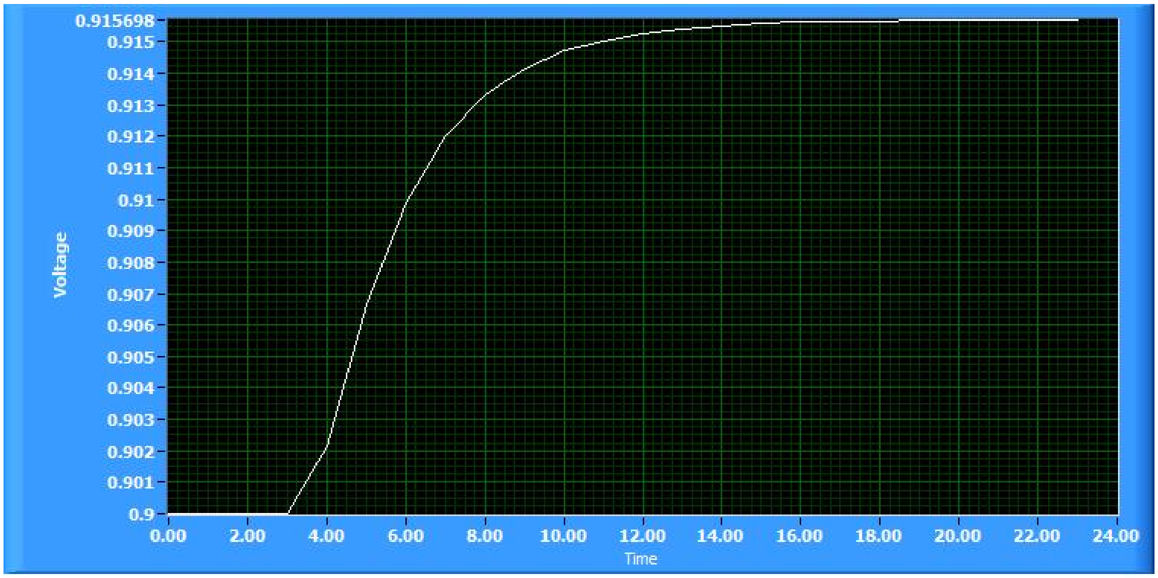


Figure 20: System Rise Time

Next, we measured the rise and fall times of the Fluke and Melexis sensors at 100°C, 200°C, and 300°C. For the rise time, we opened a shutter and measured the amount of time it took for the sensor voltage to rise from its offset and stabilise at its final value. For the fall times, we closed the shutter and measured the amount of time it took for the output to fall back to its offset voltage.

We calculated the time constant for each sensor from this data. In order to calculate the time constants, we applied the following equation:

Equation 4: First Order System Response

$$V(t) = V(0)e^{\frac{-t}{\tau}} \text{, where} \tag{4}$$

V(0) = Initial Voltage at t = 0
t = Time in seconds
τ = Time constant = RC

To calculate the sensor time constants, we took readings at 10% and 90% of the final output voltage. We derived the following equation for determining the sensor time constants:

Equation 5: Time Constant Derivation

$$0.1 = 1.0 - e^{\frac{-t_{10}}{\tau}} \Rightarrow t_{10} = -\tau \ln(0.9)$$

$$0.9 = 1.0 - e^{\frac{-t_{90}}{\tau}} \Rightarrow t_{90} = -\tau \ln(0.1)$$

$$t_{90} - t_{10} = \tau \ln(9)$$

$$\tau = \frac{t_{90} - t_{10}}{\ln(9)}, \text{ where} \qquad (5)$$

τ = Time constant;
t_{90} = Time at 90% to final voltage
t_{10} = Time at 10% to final voltage

We averaged the time constants calculated at each temperature to obtain an averaged value of the sensor time constant. We subtracted the time constant of the system alone from the time constant of the sensor and system to determine the time constant of the each sensor.

The time constant for the Melexis sensor was approximately 114 ms. The time constant for the sensor found in the Fluke thermometer was approximately 136 ms. Since the Melexis sensor has a larger sensor area and aperture, we expected the Melexis sensor to have a considerably longer response time, however this was not the case. Both sensors ended up having approximately the same response time, with the Melexis sensor being slightly quicker. The full set of results and calculations can be found in Appendix A.

5 System Design

Infrared thermometry involves acquiring data about an object's temperature and converting it to a readable format. This process involves receiving a signal, conditioning the signal, converting it into a digital format, and processing and displaying the data.

To design the system, we first broke the system into separate modules. We began by designing the modules conceptually. We constructed test circuits to examine design options and verify the functionality of individual areas of the circuit. The systems were integrated together and modified to optimise overall system performance. The complete block diagram for the system is shown in Figure 22. The following sections explain the technical details of our system design and describe each module in detail. A complete schematic of our system is shown in Figure 23.

5.1 Data Acquisition

In the data acquisition stage, the optical lens focuses incoming IR radiation on the sensor. The sensor converts the incoming radiation to a voltage across the cold thermopile junctions, which is conditioned and processed in later stages. In the first section, we discuss the optical method used to focus infrared energy onto a sensor and its implications on the distance to spot ratio and system sensitivity. The next two sections describe the IR thermopile sensor and ambient temperature sensor used in our design.

5.1.1 System Optics

The optical design for the spot pyrometer consists of a die cast metal cylinder housing, which surrounds the sensor, and a Fresnel lens. Because of limited resources, we reused the optical system from the Fluke 62 spot pyrometer, shown in Figure 21. For accurate readings, the die cast housing has a large thermal mass and prevents quick changes in the temperature of the device. Another feature of the casting is that the inside is covered with a thin coat of black, infrared-absorbing paint. When the lens focuses IR radiation towards the sensor, stray IR is absorbed by the paint and does not reflect in the direction of the sensor. The housing also provides an excellent thermal connection with the circuit board, minimising thermal gradients in the device.

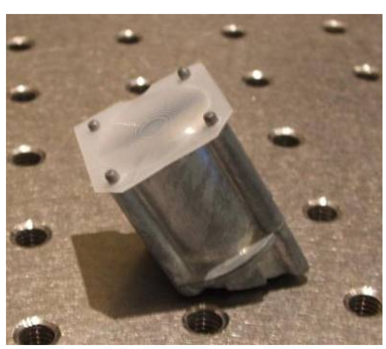


Figure 21: Fresnel Lens and Die Cast Sensor Housing

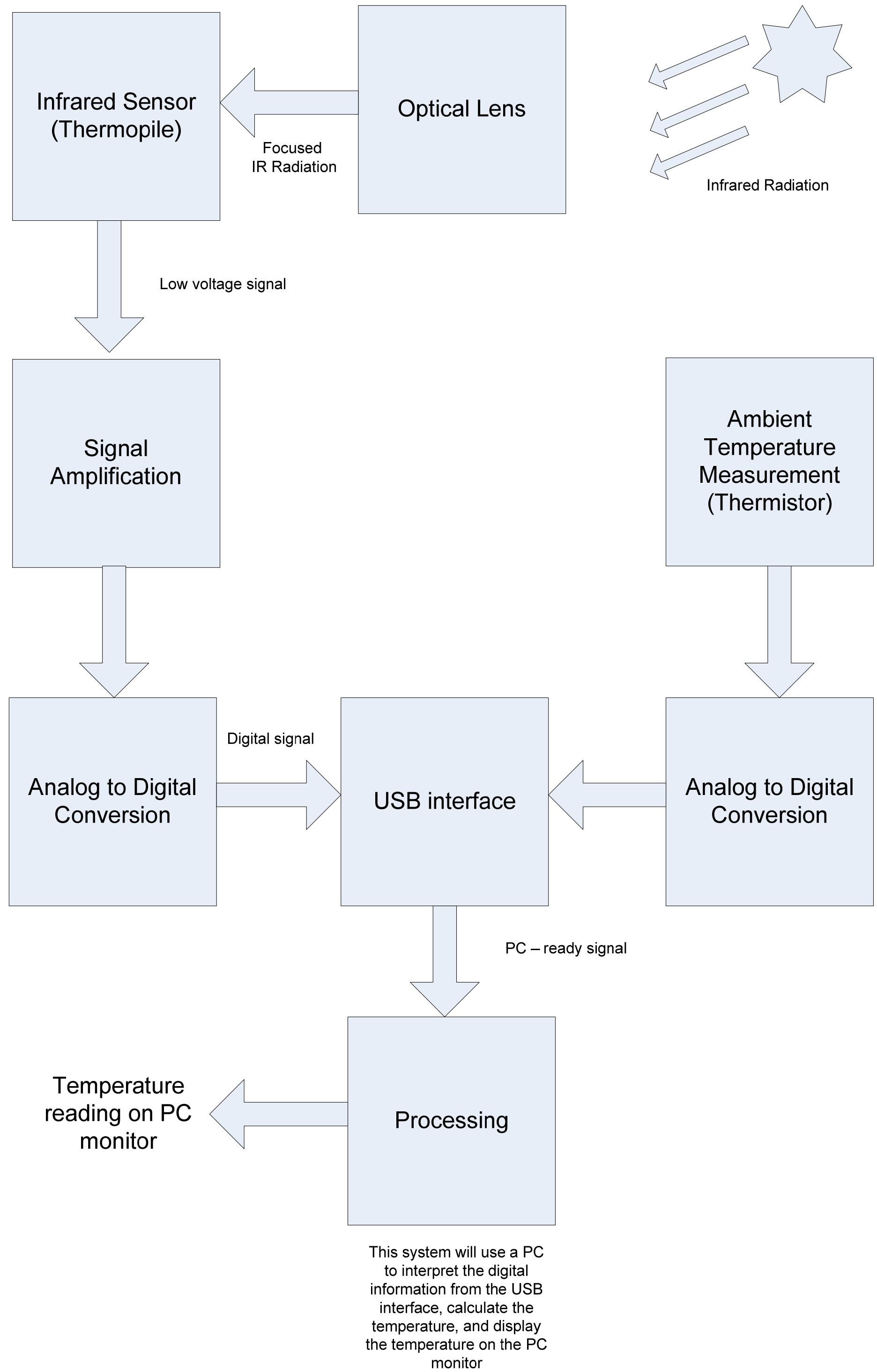


Figure 22: System Block Diagram

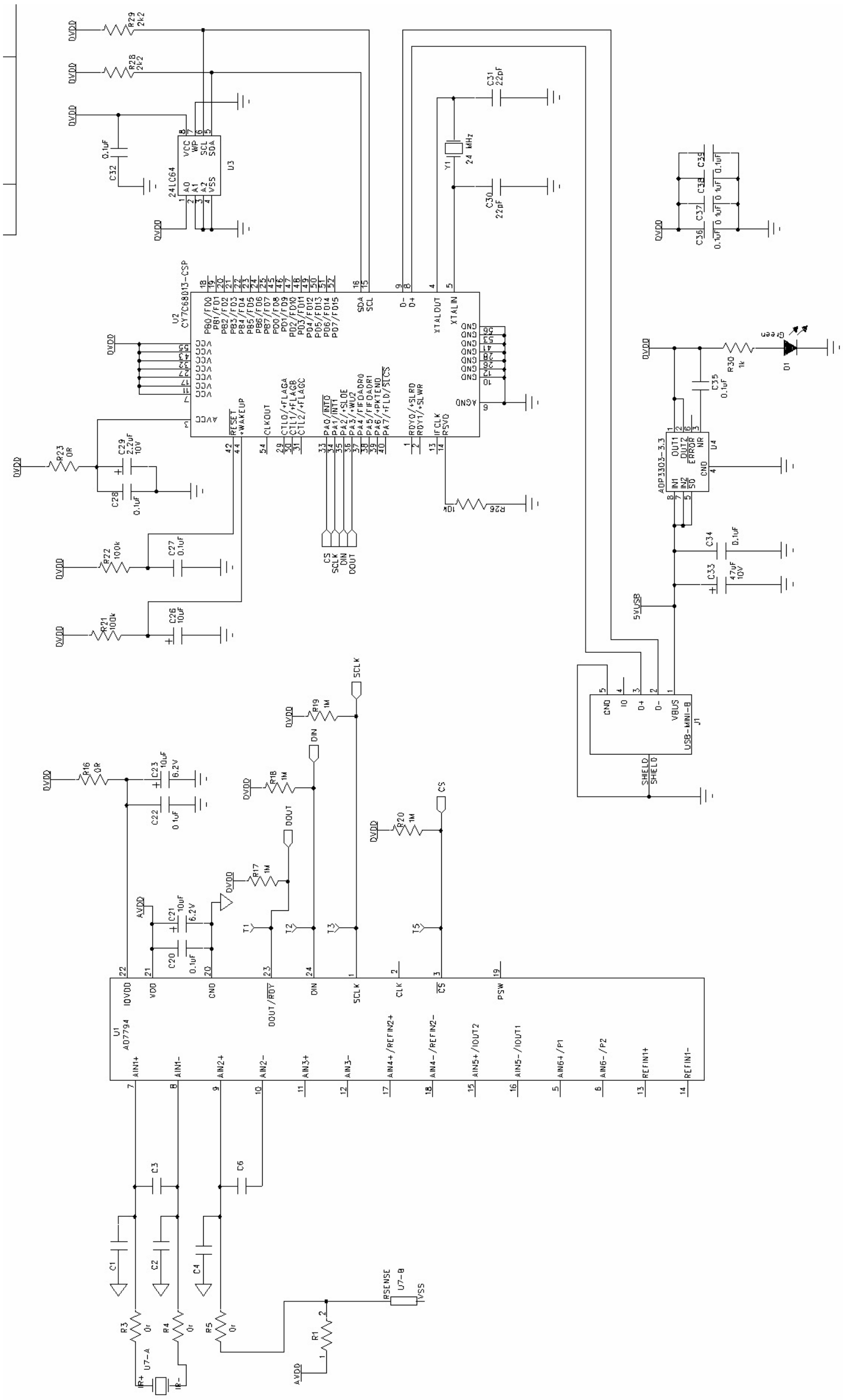


Figure 23: Complete System Schematic

The optics also determines the distance to spot ratio of the system. This ratio is controlled by the distance between the sensor and the lens (the length of the housing) and the window size of the sensor. Since sensor has an aperture twice as big as the one found in the Fluke 62, the distance to spot ratio of our system is approximately 5:1.

5.1.2 Infrared Thermopile Sensor

In our design, we used the Melexis MLX90247 thermopile sensor. This sensor was chosen by ADI engineers because it is comparable to sensors used in commercially available spot pyrometers. As seen in Figure 24, the MLX90247 is a four terminal device with a single-element thermopile and an on-chip thermistor. The close proximity of the thermistor to the sensor allows for the greatest possible precision for ambient temperature measurement. The window aperture size of the sensor is 2.5 mm with an output sensitivity of 34 μV/ °C ± 25% and an output resistance of 60 kΩ.

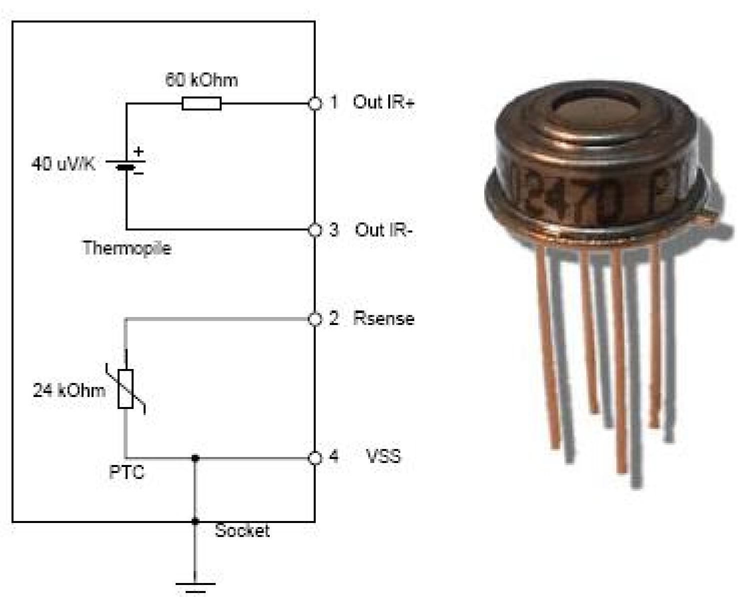


Figure 24: Melexis MLX90247 Thermopile Sensor (Melexis)

5.1.3 Ambient Temperature Thermistor

The built-in ambient temperature thermistor in the Melexis sensor has a resistance of around 24 kΩ at room temperature and varies at about 6500 ppm. The resistance of the thermistor is quadratic as a function of temperature for the ambient temperature range of our product (0-50 °C). We used the simple voltage divider circuit shown in Figure 25 to excite the thermistor. Filtering is not necessary since the thermistor produced very little noise and holds a nearly constant value due to the large thermal mass surrounding it.

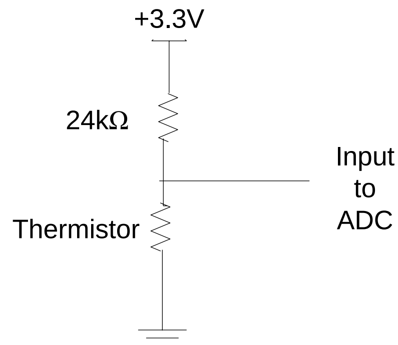


Figure 25: Ambient Temperature Schematic

5.2 Signal Conditioning and ADC Selection

After the raw data is acquired, it is processed into a readable form. The output from the infrared thermopile sensor is very small and produces a voltage of approximately 34 μV /°C. While the ambient temperature signal does not need to be filtered or amplified, the signal from the thermopile is too small for our specified temperature resolution and is susceptible to noise. As a result, the signal from the sensor must be amplified before input to the analogue to digital converter. Initially, we thought that filtering might be necessary, but after collecting data we found the noise was low enough not to warrant it.

When selecting the appropriate ADC, there were several important considerations. We first chose a converter with two channels in order to acquire the signal from both the thermopile and the ambient temperature sensor without the cost or additional complexity of using an external multiplexer. Another consideration was the desired degree of resolution. We chose 24 bits of resolution because it uses the latest technology and is compatible with the future plans of Analog Devices. Also, we found that the price of a 24-bit ADC was very similar to that of its 16-bit counterpart. The final consideration was that the ADC must use an SPI interface to communicate the digital representation of the signal to the USB chip. We chose to compare two ADCs with the desired characteristics, the AD7787 and the AD7794. The major difference between the two converters is the signal conditioning required for each part. The following sections describe the selection of an appropriate analogue to digital converter and the associated signal conditioning circuitry options.

5.2.1 AD7787 Design Option

One of the two ADCs we considered using was the AD7787. It is an appropriate choice because it is a low power, two channel 24-bit converter. In order to meet our resolution specification of ±0.2°C, we needed to amplify our thermopile signal. Since this converter has no built in gain stage, an external amplifier is needed. We compared two options for amplifying the signal for the AD7787. One option for amplification is to use an AD623 instrumentation amplifier. This amplifier uses the classic three op-amp approach with an overall gain of 80. The benefits of the instrumentation amplifier include its resistor-programmable, finite, gain and its independence of the common mode voltage of the thermopile sensor. The output voltage of the instrumentation amplifier with respect to its reference pin is proportional to the voltage difference between its two inputs.

Another option for amplification is the AD8551 zero-drift operational amplifier because it has an ultra low offset, low drift, and low bias currents with high gain. It provides the benefits of both auto-zeroing and chopper-stabilised amplifiers. As seen in Figure 26, internally the AD8551 contains two amplifiers which switch between an amplification phase and an auto-zero phase. The amplifier effectively brings the input offset voltage down to the sub-microvolt range, allowing for greater precision. The circuit configuration using the AD8551 is a simple non-inverting amplifier using two precision resistors to determine the gain.

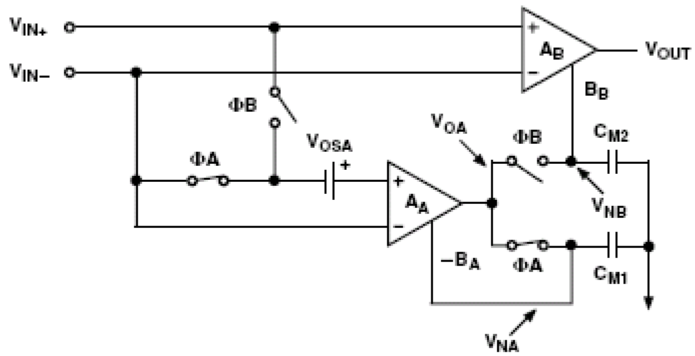


Figure 26: Auto-zero Phase of AD8551 (Analog Devices)

After comparing performance of the AD8551 and the AD623 in the gain circuitry, we chose to use the AD8551 in our design with the AD7787. The noise characteristics of both the amplifiers are comparable and low - on the order of 40nV - but the almost zero input offset drift of the AD8551 is well suited for low noise amplification. The input offset drift on the AD8551 is 0.005 µV/°C, while it is 2.0 µV/°C for the AD623. We chose to use the AD8551 because using the AD623 would result in an error of several degrees Celsius from temperature drift.

Once the signal conditioning circuitry was finalised using the AD8551, we input the signal into the AD7787. The AD7787 is well suited for use with a thermopile because it is designed for low power, low frequency, measurement applications. It contains one differential and one single-ended input channel. The device is driven by an internal clock. The analogue input signal is chosen by an onboard multiplexer and is digitised using a sigma-delta converter. The output is then transmitted over the SPI output, regulated by a serial clock provided by the USB microcontroller. The relatively low cost of $3.95 and the availability of an evaluation board also made this particular ADC favourable for prototyping.

There are four output pins on the AD7787. The pins are labelled SCLK, $\overline{CS}$, DIN, and DOUT. Since the Cypress 68013 prototype board is SPI compatible, we only needed to connect each of these pins to their respective pins on the USB board. $\overline{CS}$ is used to select the ADC during each conversion. SCLK is used to synchronise the data transfer between the USB driver and the ADC. DOUT sends the data to the USB driver one bit at a time. DIN is an input used to write to each of the registers of the ADC.

In the AD7787 design, we used the onboard multiplexer to switch between ambient temperature measurement and the thermopile sensor. The ADC is controlled by modifying its internal registers. For the purpose of our design, the only register we need to change is the communications register. The functions of the communication register are shown below in Table 1. The communication register is set at 38h for thermopile measurement and 39h for the ambient temperature measurement.

Table 1: AD7787 Communication Register (Analog Devices 2005)

CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0
$\overline{\text{WEN}}$ (0)	0 (0)	RS1 (0)	RS0 (0)	R/$\overline{\text{W}}$ (0)	CREAD (0)	CH1 (0)	CH0 (0)

Bit Location	Bit Name	Description
CR7	$\overline{\text{WEN}}$	Write Enable Bit. A 0 must be written to this bit so that the write to the communications register actually occurs. If a 1 is the first bit written, the part does not clock on to subsequent bits in the register. It stays at this bit location until a 0 is written to this bit. Once a 0 is written to the WEN bit, the next seven bits are loaded to the communications register.
CR6	0	This bit must be programmed to Logic 0 for correct operation.
CR5 to CR4	RS1 to RS0	Register Address Bits. These address bits are used to select which of the ADC's registers are being selected during this serial interface communication (see Table 6).
CR3	R/$\overline{\text{W}}$	A 0 in this bit location indicates that the next operation is a write to a specified register. A 1 in this position indicates that the next operation is a read from the designated register.
CR2	CREAD	Continuous Read of the Data Register. When this bit is set to 1 (and the data register is selected), the serial interface is configured so that the data register can be continuously read, i.e., the contents of the data register are placed on the DOUT pin automatically when the SCLK pulses are applied. The communications register does not have to be written to for data reads. To enable continuous read mode, the instruction 00111100 (Channel AIN1) or 00111101 (Channel AIN2) must be written to the communications register. To exit the continuous read mode, the instruction 001110XX must be written to the communications register while the $\overline{\text{RDY}}$ pin is low. While in continuous read mode, the ADC monitors activity on the DIN line so that it can receive the instruction to exit continuous read mode. Additionally, a reset occurs if 32 consecutive 1s are seen on DIN. Therefore, DIN should be held low in continuous read mode until an instruction is to be written to the device.
CR1 to CR0	CH1 to CH0	These bits are used to select the analog input channel. Channel AIN1 or AIN2 can be selected or an internal short (AIN1(−)/AIN1(−)) can be selected. Alternatively, the power supply can be selected, i.e., the ADC can measure the voltage on the power supply, which is useful for monitoring power supply variation. To perform this measurement, the power supply voltage is divided by 5 and then applied to the modulator for conversion. The ADC uses a 1.17 V ± 5% on-chip reference as the reference source when this channel is selected. Any change in channel resets the filter and a new conversion is started.

RS1	RS0	Register	Register Size
0	0	Communications Register during a Write Operation	8-Bit
0	0	Status Register during a Read Operation	8-Bit
0	1	Mode Register	8-Bit
1	0	Filter Register	8-Bit
1	1	Data Register	24-Bit

CH1	CH0	Channel
0	0	AIN1(+) − AIN1(−)
0	1	AIN2
1	0	AIN1(−) − AIN1(−)
1	1	V_{DD} Monitor

Since both of the input signals are converted with reference to ground, both channel references and the ADC ground reference were tied to the circuit ground. The REF+ pin was tied to a 2.5 V reference. A circuit diagram for the signal conditioning portion of this circuit is shown below in Figure 27.

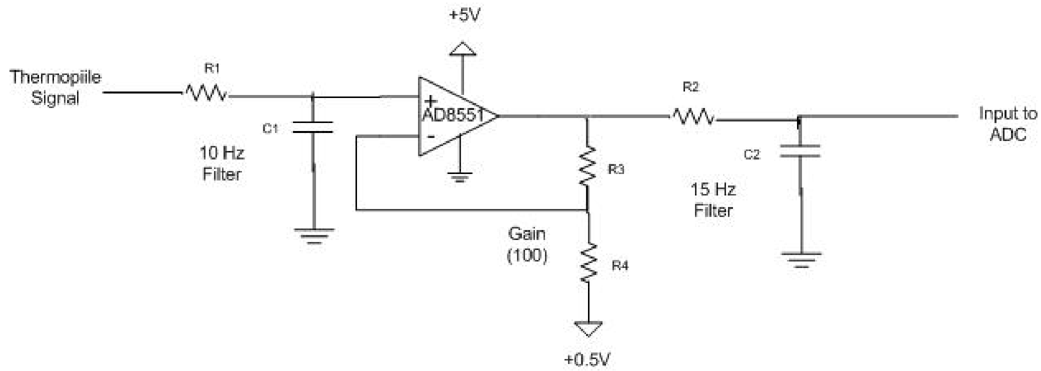


Figure 27: Signal Conditioning for the AD7787

5.2.2 AD7794 Design Option

The second option for analogue to digital conversion was to use the AD7794 because it is designed for low power, low noise, and high precision measurement. The AD7794 is part of the latest technology from Analog Devices and contains circuitry to act as the complete analogue side of the design. It contains a built in, programmable amplification stage so no pre-ADC amplification is necessary. Even though the output of the thermopile is very small, it can be input directly into the ADC.

The benefit of the AD7794 is the large number of available features without using external components. The AD7794 is a 24-bit sigma delta converter with six differential inputs. It includes an on-chip in-amp and reference with low drift characteristics. While the ADC can process the signal with little amplification, some amplification was used to reduce sensitivity to temperature drift. The amplification is easily modified in software. The device can be driven by an internal clock or external clock. Other features include options for low power setting, programmable excitation current sources, burnout currents, and a bias voltage generator. The output is then transmitted over the SPI output and regulated by a serial clock provided by the USB microcontroller. The low cost of $5.08 and the availability of an evaluation board also made this particular ADC favourable for prototyping.

The output of the AD7794 and AD7787 are very similar. Just like the AD7787, there are four output pins on the AD7794. The pins are labelled SCLK, $\overline{CS}$, DIN, and DOUT which can be directly connected to the respective pins on the USB board. DIN is used to control each of the functions of the ADC.

The different features for the AD7794 are set by writing to the appropriate register. The internal amplification is set by writing to the configuration register. The configuration register is also used to internally bias the negative terminals of the inputs to half of the analogue supply voltage to ensure that the common mode of the input signal remains within the specifications of the ADC. The schematic for the AD7794 design option is shown below in Figure 28.

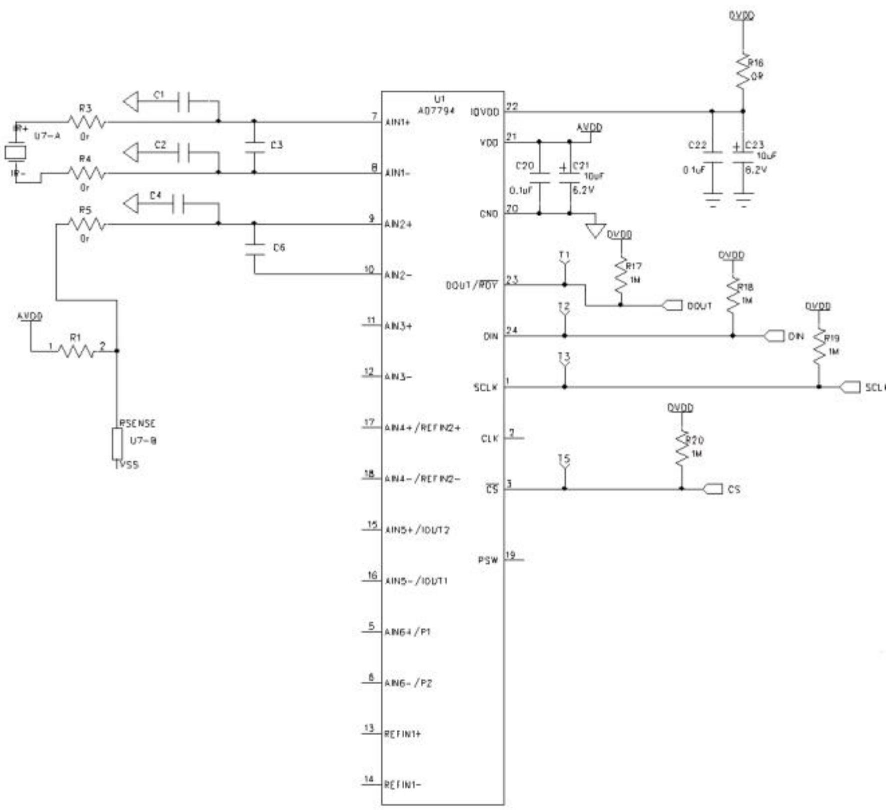


Figure 28: AD7794 Design Option

5.2.3 ADC Design Comparison

After building circuits test circuits using the AD7787 and AD7794, we compared the noise performance, cost, features and ease of use of each ADC along with the future direction of Analog Devices to select the most appropriate analogue to digital converter. The histograms in Figure 29 and Figure 30 show that both converters have comparable noise performance. While initially it appears the AD7787 is significantly noisier, the test circuit for the AD7787 has a gain of 80 compared to a gain of 32 with the AD7794 test circuit. The wires from the amplifier output to the ADC input were also relatively long on the AD7787 evaluation board we were using. The noise associated with either ADC is still low enough for our application. In our design, we decided to use the AD7794 because although the cost is higher, its ease of use and additional features make it attractive for future designs by Analog Devices.

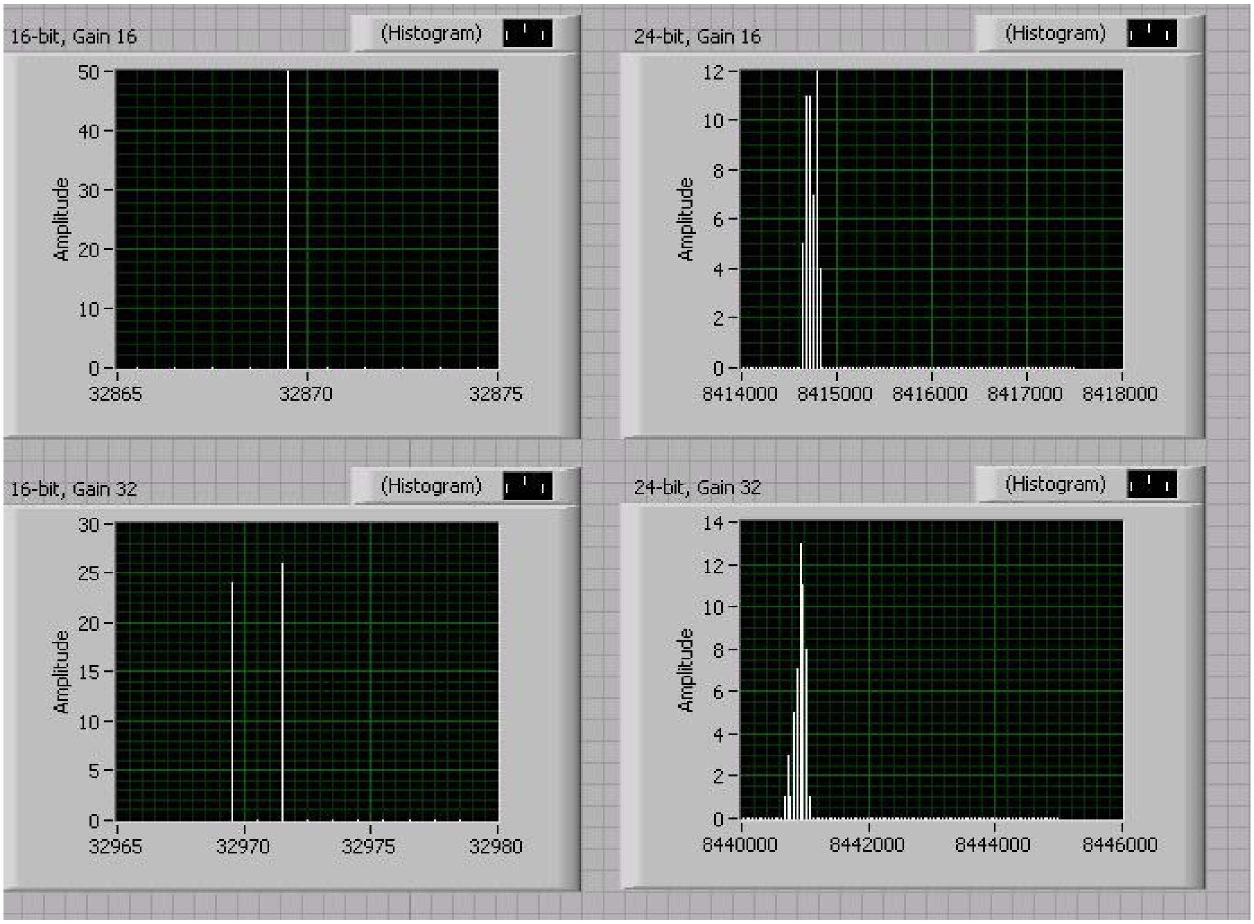


Figure 29: AD7794 Noise Distribution

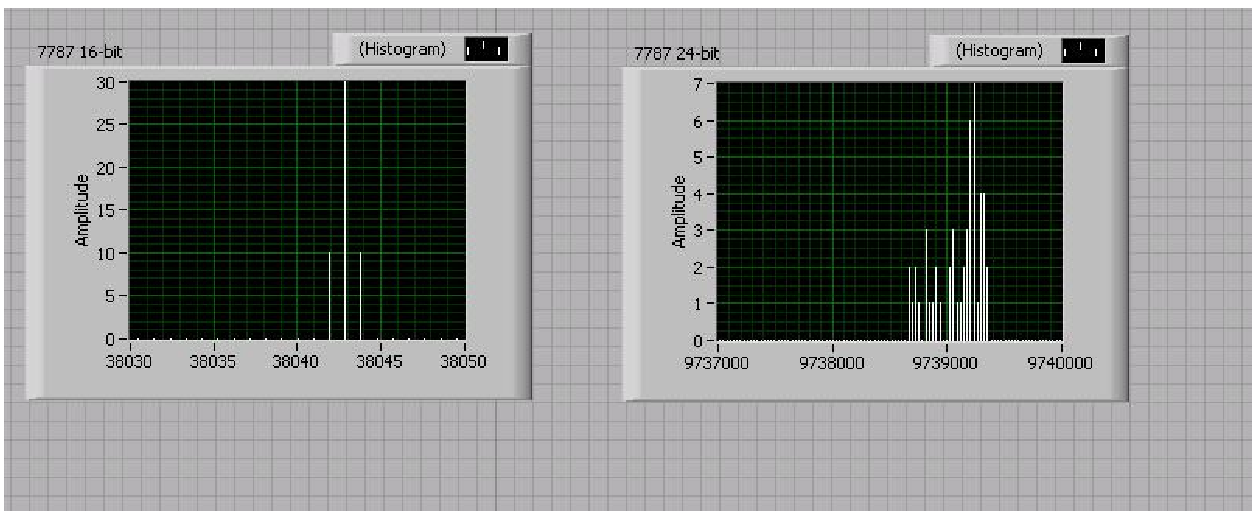


Figure 30: AD7787 Noise Distribution

5.3 USB Microcontroller

With the recommendation of Analog Devices engineers, we decided to use the Cyprus 68013 USB microcontroller for our design. The purpose of this microcontroller is to access our analogue to digital converter using an SPI interface. The USB microcontroller receives instructions from the PC to read and write from the data and communications registers, respectively. The USB controller transmits the signal data from the data register to the PC for processing.

The Cyprus 68013 is a complete integrated microcontroller. It combines a USB 2.0 transceiver, SIE, an enhanced 8051 microcontroller, and a programmable peripheral interface onto a single 56-pin chip. Since this chip is already supported and used by Analog Devices, it was particularly appealing in our design. Although the chip can be programmed to handle virtually any task in its firmware, we used this device simply to access and transmit data from our analogue to digital converter to the PC. The USB board is shown in Figure 31.

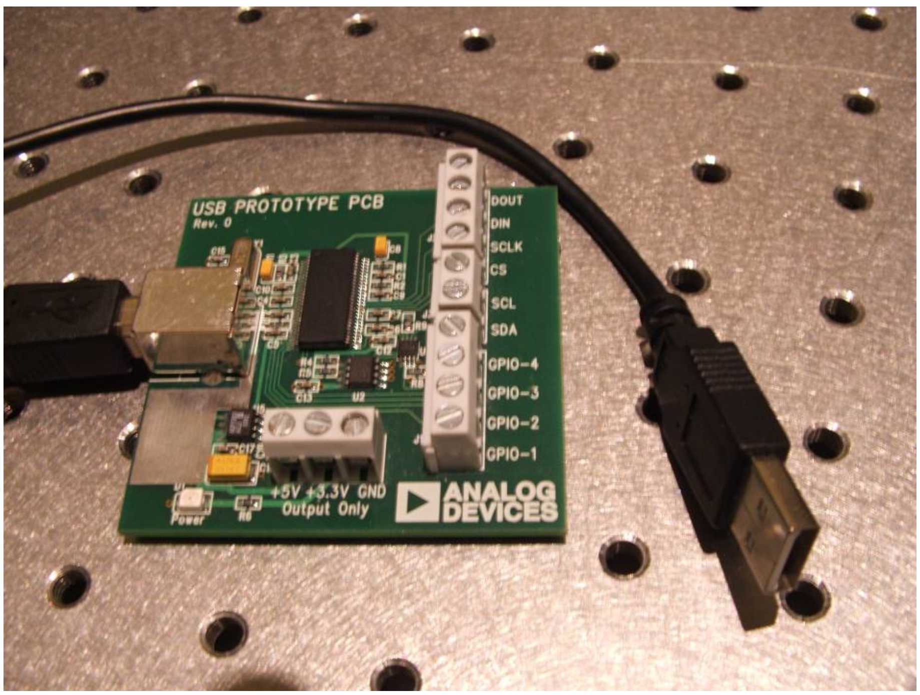


Figure 31: USB Board

For the purpose of prototyping, we used a USB board supplied by Analog Devices. The schematic of the USB board and interface are shown below in Figure 32. The USB board contains a Cyprus 68013, an EEPROM, a 24 MHz clock, and a 3.3 volt voltage reference. This board connects directly to a PC and has the following outputs: DOUT, DIN, SCLK, /CS, SCL, SDA, GPIO-4, GPIO-3, GPIO-2, GPIO-1, +5V, +3.3V, and GND. For the purpose of our design and prototyping, we only needed to use the SPI outputs DOUT, DIN, SCLK, /CS, and the voltage reference outputs. The schematic of the USB evaluation board is shown below in Figure 32.

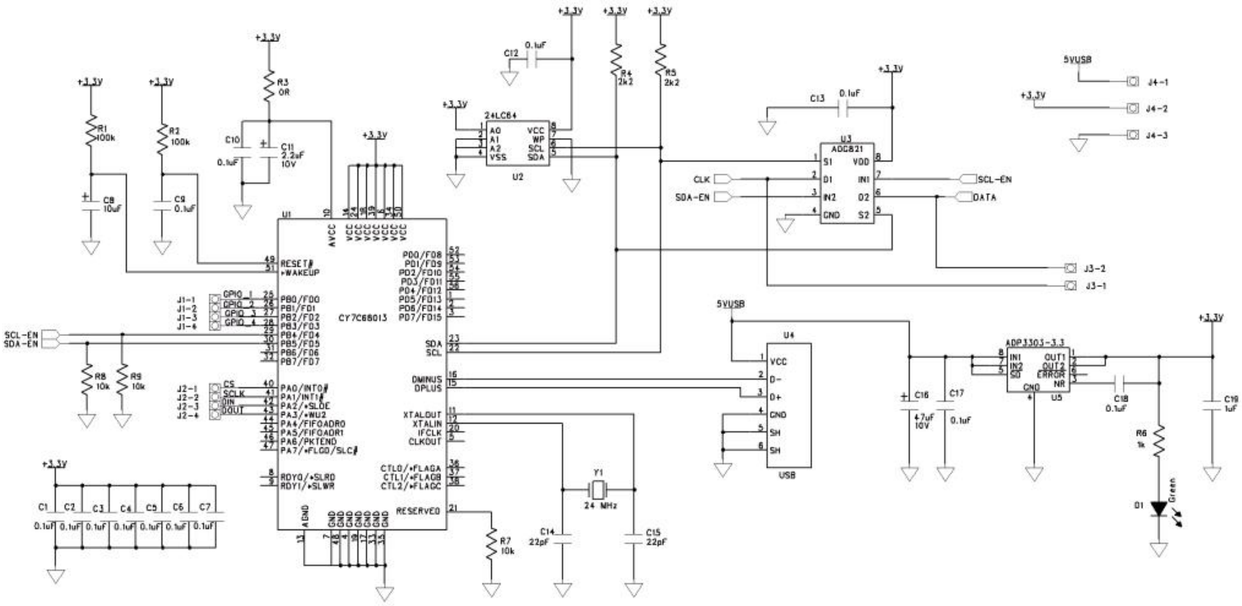


Figure 32: USB Evaluation Board Schematic

Since this device is a microcontroller as well as a transceiver, it requires firmware to function properly. The firmware for the USB chip handles the device specific setup packets that control the command of the device. For our design, we used the firmware already written and supplied to us by Analog Devices. The code is based upon sample code supplied by Cypress and modified by Michal Brychta of ADI to fit the desired functionality. Since this code includes SPI functionality, the firmware did not need to be modified by our group and could be used unaltered. The firmware source code can be found in Appendix B.

Every USB device uses a unique Product Identification number (PID) and Vendor Identification number (VID) to determine the necessary drivers. These PID and VID numbers are stored on an EEPROM and accessed when the device is first connected. For the purpose of our prototype, we used the same PID and VID numbers that default on the chip (VID 04b4, PID 8613). Although unique PID and VID numbers are required at Analog Devices for USB chips using the I^2C functionality, they are not required for an SPI calibrated device. To establish uniqueness of the USB device, we changed the setup file included on the software disc to read "WPI Spot Pyrometer" when the device is attached. The contents of the setup file can be found in Appendix B.

5.4 Signal Processing

The final stage of our design was signal processing. In this stage, we designed a user interface for the system as well as methods for processing a digital number and converting it into a temperature value. The software for processing the data was designed entirely using National Instrument's LabView software. LabView is a graphical programming language designed explicitly to interface instruments with a PC. It features options of tracking, logging, and displaying data in various graphs and windows. For our application, we designed a simple user interface that processes the raw data into a single temperature reading. A LabView program is made up of routines known as a Virtual Instrument, or a VI. Each routine can also have subroutines, called sub-VIs. The following section discusses each VI and sub-VI used to process data to display a final temperature reading. A block diagram of our LabView hierarchy is shown in Figure 33.

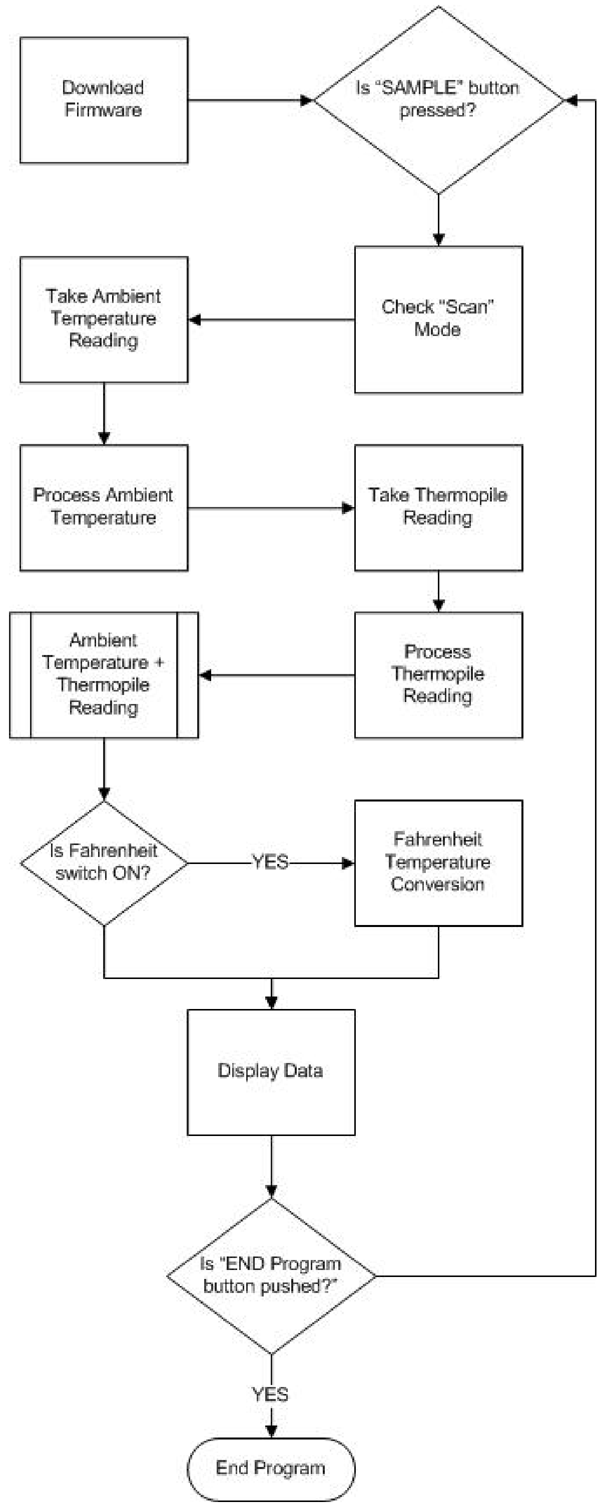


Figure 33: Software Flow Chart

5.4.1 Download Firmware VI

As described in the previous section, the USB microcontroller requires firmware to understand and process commands from the PC. This firmware is downloaded to the RAM of the USB chip at the start of program execution using a VI called Download Firmware. We received the VI from Analog Devices engineer Mary McCarthy. The front end of this VI is shown below in Figure 34. The front panel shows the location of firmware hex file and a status window indicates whether or not the firmware successfully downloaded to the USB chip. This function has no inputs and four outputs: Status, Path before, Path After, and a Downloaded LED.

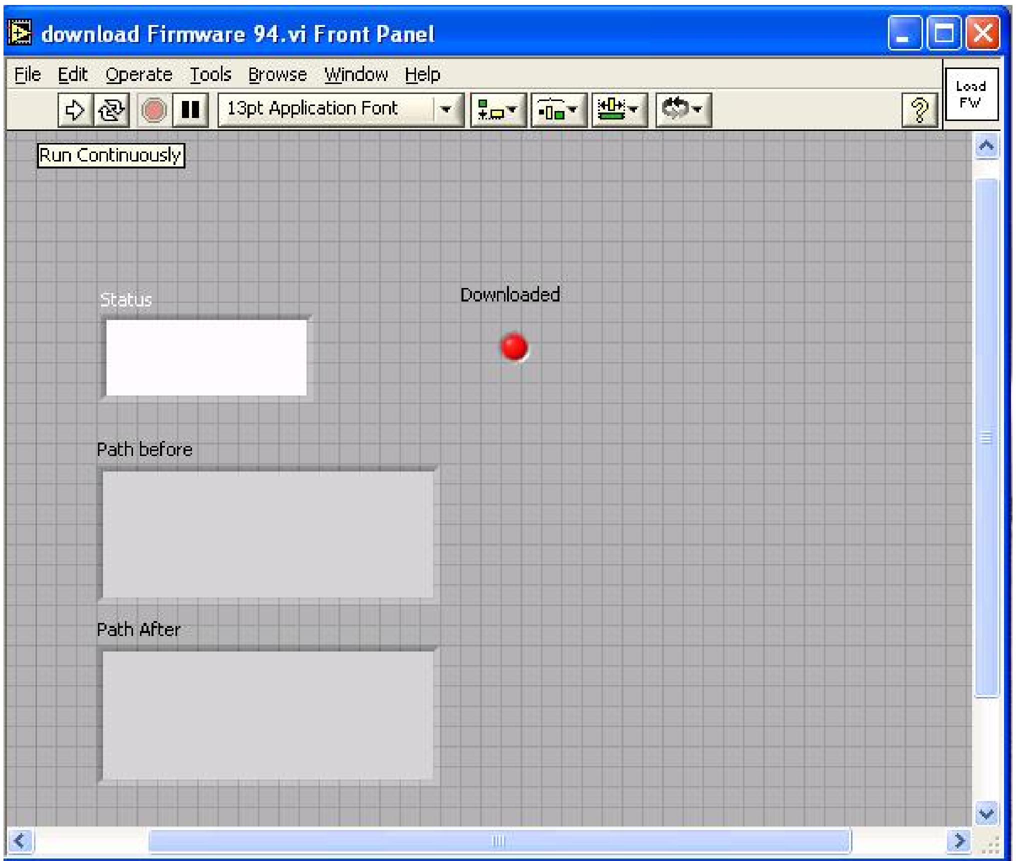


Figure 34: Download Firmware VI

5.4.2 SPI Communications

In order for the PC and USB microcontroller to communicate, the PC must send and receive setup packets from the USB device. The packets contain information which allow the PC to control the USB microcontroller. The function of each setup packet is determined by both device specific configuration and firmware. In our design, we were able to communicate with the USB microcontroller by using a sub-VI supplied by ADI. A screenshot of the front panel is shown in Figure 35. We never used the SPI Communications VI in its standalone form, but we applied it through parent VIs to read and control our ADC as well as to control I/O ports. The parent VIs for ADC and I/O control are explained in the following sections.

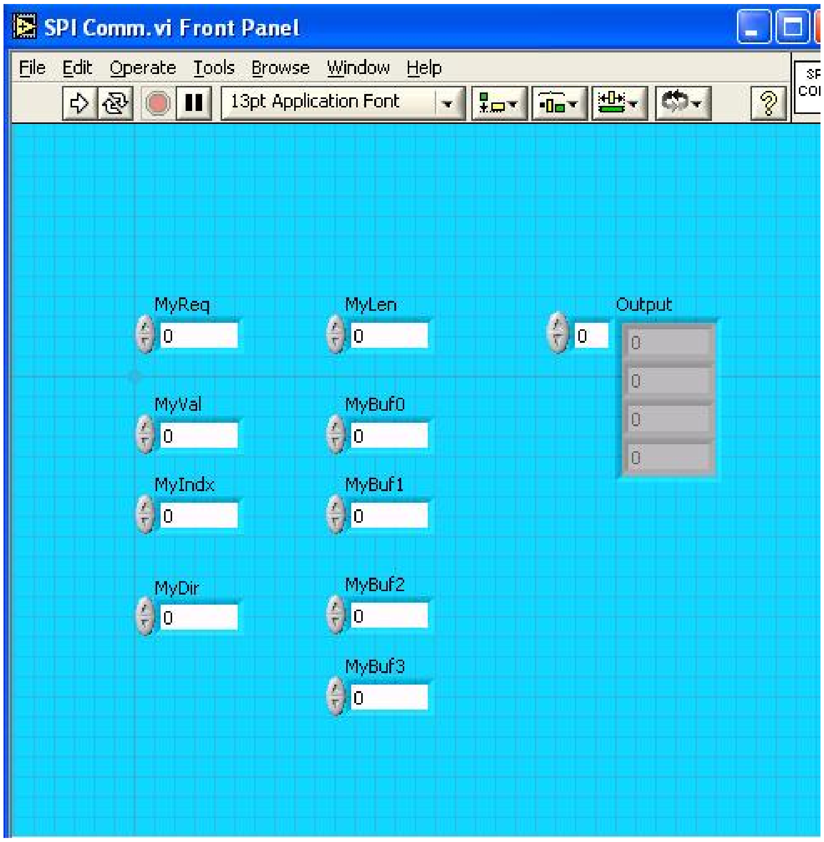


Figure 35: SPI Comm

5.4.3 Reading ADC

Another useful VI supplied to us by Analog Devices was Read ADC. This VI utilises SPI Communications to send specific setup packets to read data from the ADC through the SPI interface. The front panel of the read VI, shown in Figure 36, contains four inputs and one output. The inputs to this VI include Comms Reg, Length, 24_16_8_bit, and DRDY. The output is the hexadecimal result of the read.

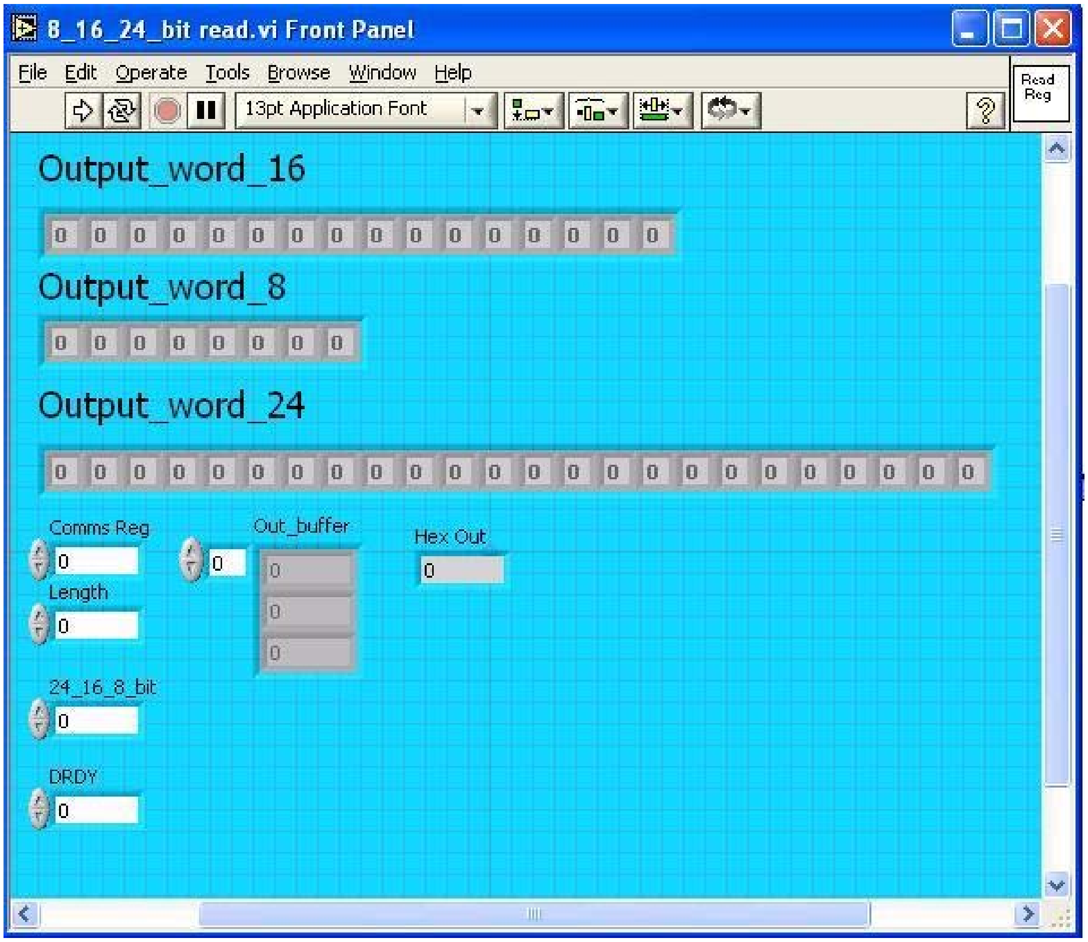


Figure 36: Read Reg

The Comms Reg field is a hexadecimal input that controls what is written to the Communication Register of the ADC. This register has to be written to before each read or write and instructs the ADC which register to use. For our device, the input to this register was set to 0101 1000, or 58h, before a read. This value tells the Communication Register to access and read from the Data Register and the continuous read mode should remain off. The details of the Communication Register are shown in Table 2.

Table 2: AD7794 Communications Register

CR7	CR6	CR5	CR4	CR3	CR2	CR1	CR0
$\overline{\text{WEN}}$(0)	R/$\overline{\text{W}}$(0)	RS2(0)	RS1(0)	RS0(0)	CREAD(0)	0(0)	0(0)

Bit Location	Bit Name	Description
CR7	$\overline{\text{WEN}}$	Write Enable Bit. A 0 must be written to this bit so that the write to the communications register actually occurs. If a 1 is the first bit written, the part will not clock on to subsequent bits in the register. It will stay at this bit location until a 0 is written to this bit. Once a 0 is written to the $\overline{\text{WEN}}$ bit, the next seven bits will be loaded to the communications register.
CR6	R/$\overline{\text{W}}$	A 0 in this bit location indicates that the next operation will be a write to a specified register. A 1 in this position indicates that the next operation will be a read from the designated register.
CR5 to CR3	RS2 to RS0	Register Address Bits. These address bits are used to select which registers of the ADC are being selected during this serial interface communication. See Table 12.
CR2	CREAD	Continuous Read of the Data Register. When this bit is set to 1 (and the data register is selected), the serial interface is configured so that the data register can be continuously read, that is, the contents of the data register are automatically placed on the DOUT pin when the SCLK pulses are applied after the $\overline{\text{RDY}}$ pin goes low to indicate that a conversion is complete. The communications register does not have to be written to for data reads. To enable continuous read mode, the instruction 01011100 must be written to the communications register. To exit the continuous read mode, the instruction 01011000 must be written to the communications register while the $\overline{\text{RDY}}$ pin is low. While in continuous read mode, the ADC monitors activity on the DIN line so that it can receive the instruction to exit continuous read mode. Additionally, a reset will occur if 32 consecutive 1s are seen on DIN. Therefore, DIN should be held low in continuous read mode until an instruction is to be written to the device.
CR1 to CR0	0	These bits must be programmed to Logic 0 for correct operation.

RS2	RS1	RS0	Register	Register Size
0	0	0	Communications Register During a Write Operation	8-bit
0	0	0	Status Register During a Read Operation	8-bit
0	0	1	Mode Register	16-bit
0	1	0	Configuration Register	16-bit
0	1	1	Data Register	24-bit
1	0	0	ID Register	8-bit
1	0	1	IO Register	8-bit
1	1	0	Offset Register	24-bit
1	1	1	Full-Scale Register	24-bit

Length is an input that controls the number of bytes involved in communication. Since the Communication Register is 8-bits (1 byte), and the Data Register is 24-bits (3 bytes), we set Length to 4. The 24_16_8_bit field is an input that sets the number of bits received in the read operation. An input of 0 corresponds to an 8-bit read, 1 to a 16-bit read, and 2 to a 24-bit read. For the purpose of our design, we decided to use only 16 bits of the ADC, therefore this input was assigned a value of 1. The final input was DRDY. This input is used to set timeout functionality of the VI. If this input is set to 0, it continuously tries to read the ADC. However, we set this value to 65535, enabling the routine to stop if the operation times out.

5.4.4 Writing ADC

Another useful VI supplied by ADI was Write Reg. Write Reg is a function used explicitly to write to 8-bit, 16-bit, or 24-bit registers. This function contains 5 inputs and no outputs. The five inputs are Length, Comms Reg, Value2, Value3, and Value4. The front end of this VI is shown below in Figure 37.

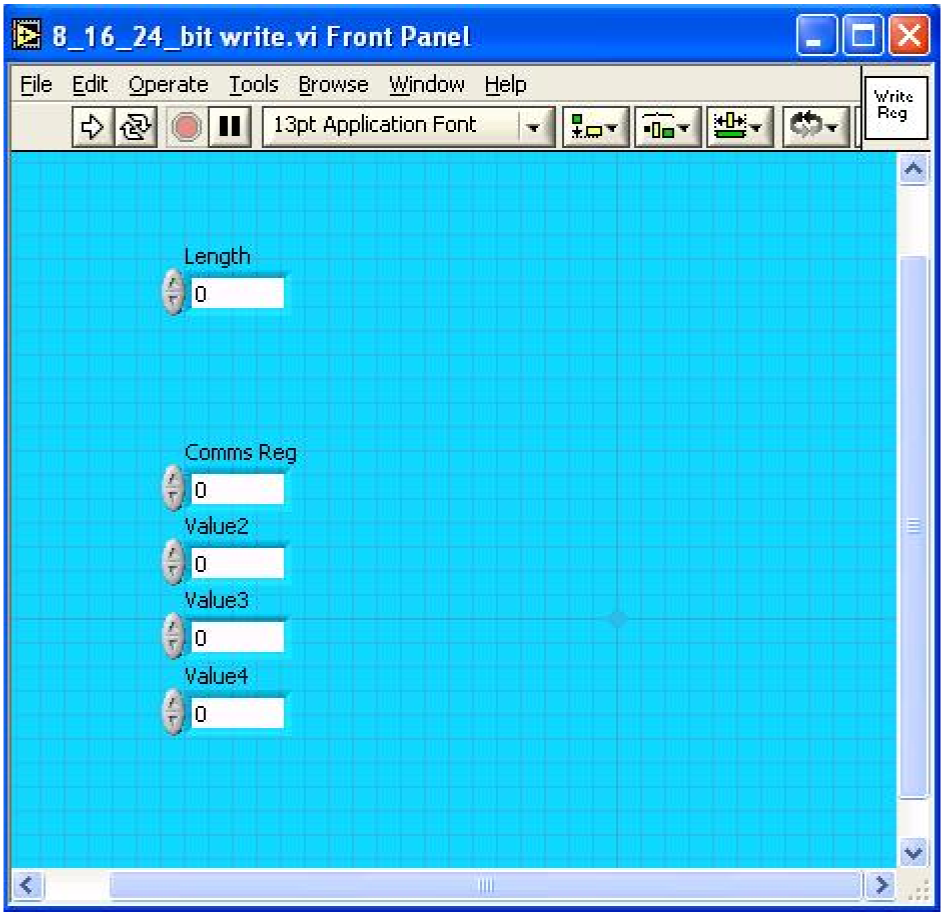


Figure 37: Write Reg

The first input, Length, sets the number of bytes that are involved in the transfer. One byte is necessary to write to the Communication Register before every write operation. The AD7794 also uses a two byte registers for both the Configuration and Mode registers. For our project, we needed to write to the Configuration Register. The Configuration Register of the AD7794 controls bias voltages, burnout current, unipolar/bipolar mode, programmable gain, references, buffers, and channel select. Since the Communication and Configuration registers are the only registers we needed to use in the write operation, we set the value of Length to 3. To select the Configuration Register, we set it to a value of "0001 0000", or 10h, during a write. A complete explanation of the Configuration register functions is shown in Table 3.

Table 3: AD7794 Configuration Register

CON15	CON14	CON13	CON12	CON11	CON10	CON9	CON8
VBIAS1(0)	VBIAS0(0)	BO(0)	U/$\overline{B}$(0)	BOOST(0)	G2(1)	G1(1)	G0(1)
CON7	CON6	CON5	CON4	CON3	CON2	CON1	CON0
REFSEL1(0)	REFSEL0(0)	REF_DET(0)	BUF(1)	CH3(0)	CH2(0)	CH1(0)	CH0(0)

Bit Location	Bit Name	Description
CON15 to CON14	VBIAS1 to VBIAS0	Bias Voltage Generator Enable. The negative terminal of the analog inputs can be biased up to $AV_{DD}/2$. These bits are used in conjunction with the BOOST bit.

VBIAS1	VBIAS0	Bias Voltage
0	0	Bias voltage generator disabled
0	1	Bias voltage generator connected to AIN1(−
1	0	Bias voltage generator connected to AIN2(−)
1	1	Bias voltage generator connected to AIN3(−)

Bit Location	Bit Name	Description
CON13	BO	Burnout Current Enable Bit. This bit must be programmed with a Logic 0 for correct operation. When this bit is set to 1 by the user, the 100 nA current sources in the signal path are enabled. When BO = 0, the burnout currents are disabled. The burnout currents can be enabled only when the buffer or in-amp is active.
CON12	U/$\overline{B}$	Unipolar/Bipolar Bit. *Set* by user to enable unipolar coding, that is, zero differential input will result in 0x000000 output and a full-scale differential input will result in 0xFFFFFF output. *Cleared* by the user to enable bipolar coding. Negative full-scale differential input will result in an output code of 0x000000, zero differential input will result in an output code of 0x800000, and a positive full-scale differential input will result in an output code of 0xFFFFFF.
CON11	BOOST	This bit is used in conjunction with the VBIAS1 and VBIAS0 bits. When *set*, the current consumed by the bias voltage generator is increased, which reduces its power-up time.
CON10 to CON8	G2 to G0	Gain Select Bits. Written by the user to select the ADC input range as follows:

G2	G1	G0	Gain	ADC Input Range (2.5 V Reference)
0	0	0	1 (in-amp not used)	2.5 V
0	0	1	2 (in-amp not used)	1.25 V
0	1	0	4	625 mV
0	1	1	8	312.5 mV
1	0	0	16	156.2 mV
1	0	1	32	78.125 mV
1	1	0	64	39.06 mV
1	1	1	128	19.53 mV

Bit Location	Bit Name	Description
CON7 to CON6	REFSEL1/REFSEL0	Reference Select Bits. The reference source for the ADC is selected using these bits.

REFSEL1	REFSEL0	Reference Source
0	0	External reference applied between REFIN1(+) and REFIN1(−)
0	1	External reference applied between REFIN2(+) and REFIN2(−)
1	0	Internal 1.17 V reference
1	1	Reserved

Bit Location	Bit Name	Description
CON5	REF_DET	Enables the Reference Detect Function. When *set*, the NOXREF bit in the status register indicates when the external reference being used by the ADC is open circuit or less than 0.5 V. When *cleared*, the reference detect function is disabled.
CON4	BUF	Configures the ADC for buffered or unbuffered mode of operation. If *cleared*, the ADC operates in unbuffered mode, lowering the power consumption of the device. If *set*, the ADC operates in buffered mode, allowing the user to place source impedances on the front end without contributing gain errors to the system. For gains of 1 and 2, the buffer can be enabled or disabled. For higher gains, the buffer is automatically enabled. With the buffer disabled, the voltage on the analog input pins can be from 30 mV below GND to 30 mV above AV_{DD}. When the buffer is enabled, it requires some headroom so the voltage on any input pin must be limited to 100 mV within the power supply rails.

Bit Location	Bit Name	Description
CON3 to CON0	CH3 to CH0	Channel Select Bits. Written by the user to select the active analog input channel to the ADC.

CH3	CH2	CH1	CH0	Channel	Calibration Pair
0	0	0	0	AIN1(+) – AIN1(−)	0
0	0	0	1	AIN2(+) – AIN2(−)	1
0	0	1	0	AIN3(+) – AIN3(−)	2
0	0	1	1	AIN4(+) – AIN4(−)	3
0	1	0	0	AIN5(+) – AIN5(−)	3
0	1	0	1	AIN6(+) – AIN6(−)	3
0	1	1	0	Temp Sensor	Automatically selects the internal reference and sets the gain to 1
0	1	1	1	AV_{DD} Monitor	Automatically selects the internal 1.17 V reference and sets the gain to 1/6
1	0	0	0	AIN1(−) – AIN1(−)	0
1	0	0	1	Reserved	
1	0	1	1	Reserved	
1	1	0	0	Reserved	
1	1	0	1	Reserved	
1	1	1	0	Reserved	
1	1	1	1	Reserved	

Since the Configuration Register controls the features of the ADC, it is used to set a voltage bias on pin AIN2(-). In order to read the ambient temperature from the thermistor, a voltage bias of half of the supply voltage is set to pin AIN2(-). To accomplish this, we set bits CON15 and CON14 values of 1 and 0 respectively. By setting this bias voltage, the common-mode voltage stays in the middle of the supply rails and keeps the ADC within specification. CON12 was set to 0, allowing the ADC to function in bipolar mode. Therefore, an input voltage of 0 V outputs 8000h, half of the ADC range. Since the ambient temperature reading did not need to be amplified, the gain was to be set to zero, or 000, for bits CON10 to CON8.

Our ADC operates on an internal 1.17 Volt reference, enabling us to work with a total bipolar range of 2.34 Volts. We set CON7 and CON6 to 1 and 0 respectively to use the internal reference and CON4 to 1 to enable buffered mode. Finally, when reading channel 2, we set CON3 through CON0 to 0001. All other values were kept at their default value. Therefore, Value2 was assigned 80h and Value3 was assigned 91h during the write before an ambient temperature read.

When preparing to read from the thermopile, the bias voltage was switched from AIN2(-) to AIN1(-) and bits CON15 and CON14 were assigned values of 0 and 1 respectively. Since we needed increased sensitivity from our sensor, we used a gain of 32 from the programmable gain amplifier. As a result, we assigned 101 to bits CON10 through CON8. Finally, CON3 through CON0 were set to 0000 to enable Channel 1. Therefore, before reading the thermopile Value2 and Value3 were assigned values of 45h and 90h during the write. All other register values remained the same as previously mentioned.

5.4.5 Sampling

When processing data through filters and putting the information into graphs, it was necessary to sample the incoming data and place it into an array. For this purpose, we used a VI called Get Samp, supplied by ADI. In order to use Get Samp to implement the Read Reg function in our design, we needed to modify it slightly. The default for Read Reg is to read 24-bits, but since we only read 16-bits in our program, we modified the VI to fit our application. The Get Samp function has two inputs and one output. Num Samples controls the number of samples that are taken and places the values into an array. Busy is a Boolean input that freezes the function until the Busy input is no longer true. The output is a 1-dimensional array containing each sample which can be used to output graphs, filter, or average. The front end of this VI is shown below in Figure 38.

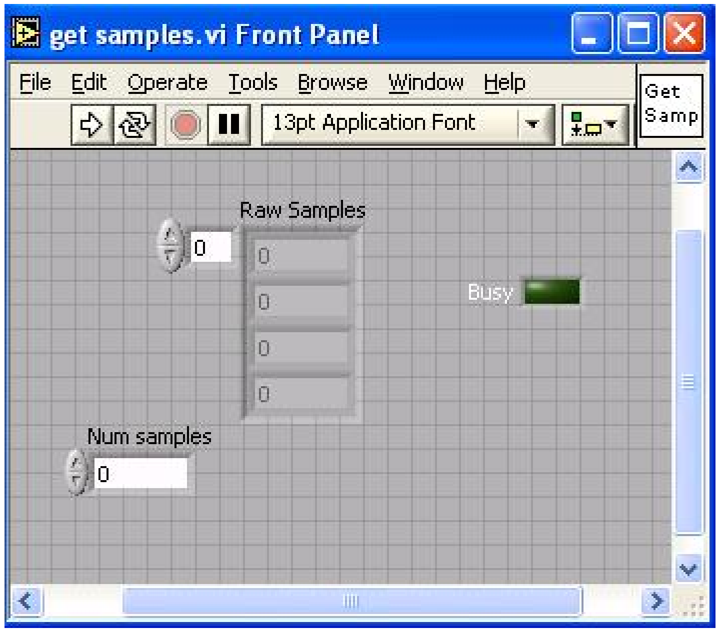


Figure 38: Get Samp

5.4.6 Main Application

Our main application was based on the input of Analog Devices as well as our own specifications. The main application follows the system block diagram shown in Figure 33. This program has several buttons for both numerical and graphical temperature displays of the measured temperature and ambient temperature. We designed the interface with several buttons which enable the user to control the type of data they wish to see. The user interface is shown in Figure 39. The SAMPLE button is used to begin taking data. The Samples field is a numerical input in the interface that enables the user to select the number of samples to be taken in Normal Mode. A separate button allows the user to enter Scan Mode. Both Scan Mode and Normal Mode are discussed in the following sections. The user also has the option of viewing the results in Celsius or Fahrenheit depending on the position of a toggle switch. Finally, a button labelled END PROGRAM allows the user to exit from the program.

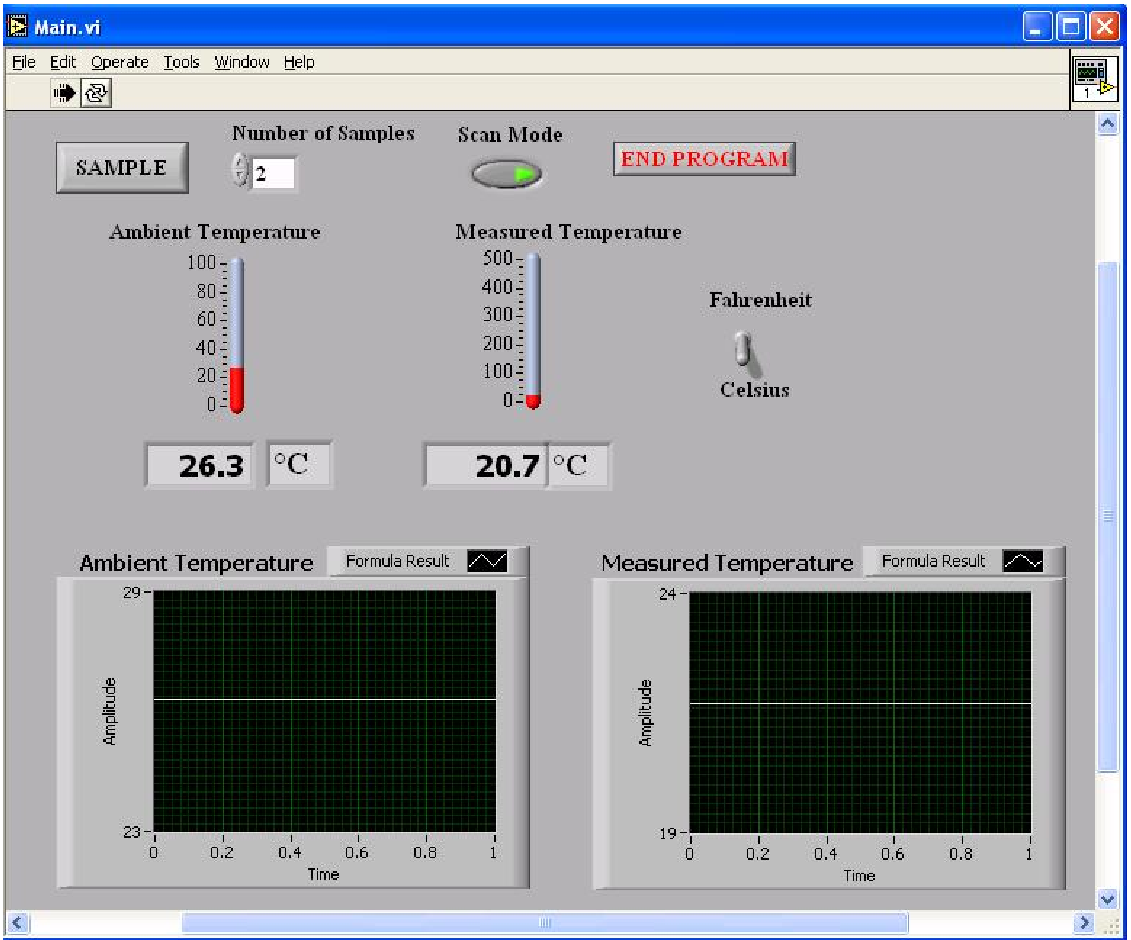


Figure 39: Main Application User Interface

The program begins by downloading the firmware to the USB microcontroller. This function is only performed once at the start of the program. The software then enters a loop and remains active until the END PROGRAM button is pushed. This button exits out of the loop and ends the program.

The next step of the program checks whether the SAMPLE button is pressed. The program does not respond until this occurs. When the button is pushed, the program first activates the laser, then checks to see if the Scan Mode button is pressed. If the Scan Mode button is pushed, the software goes into Scan Mode, which is covered in detail in the next section. Otherwise, the software takes a temperature reading in Normal Mode, which is discussed in Section 5.4.9.

5.4.7 Scan Mode

Scan mode is a mode of operation created to log continuous raw data. This mode outputs temperature readings in real-time, but does not filter the data or output graphs. This mode is favourable when the user wants to see quick results in real-time and does not require any graphs. A block diagram of this function is shown below in Figure 40.

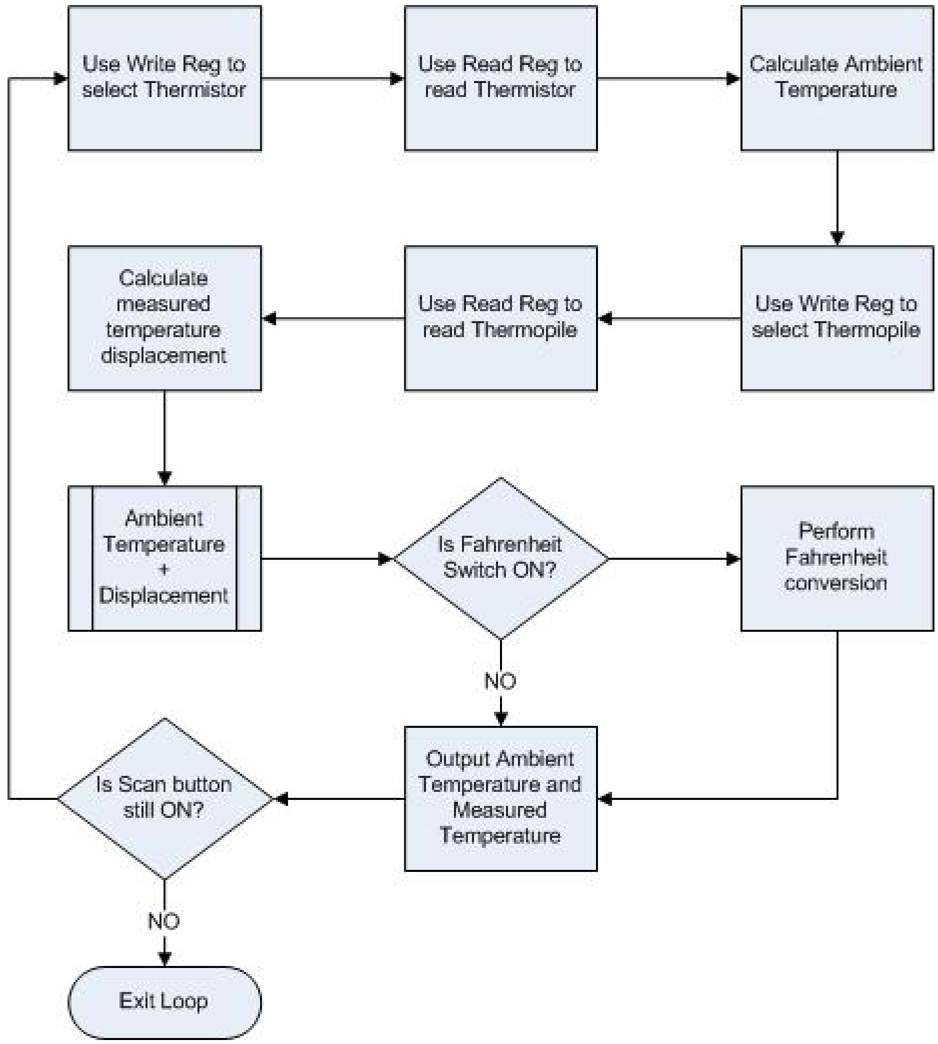


Figure 40: Scan Mode Block Diagram

This mode begins by using the Write Reg and Read Reg functions mentioned in previous sections to read data from the thermopile and the thermistor. These values are then passed through corresponding formulae to calculate the ambient temperature and the temperature difference between the spot and the sensor. These formulae are discussed further in section 6. The two values are then added together to calculate the measured temperature.

The final step of this program performs a Fahrenheit conversion if desired. If the Fahrenheit switch is on, the ambient temperature and spot temperature are both passed through a conversion formula and the data is displayed with the appropriate label affixed next to the numerical display. Otherwise, the data is displayed in degrees Celcius. The conversion formula used is shown in Equation 6. Our LabView algorithm for this portion of the program is found in Figure 41.

Equation 6: Fahrenheit-Celsius Conversion

$$T_f = (\frac{9}{5})T_C + 32 \text{, where} \quad (6)$$

T_f = Temperature in Fahrenheit
T_c = Temperature in Celsius

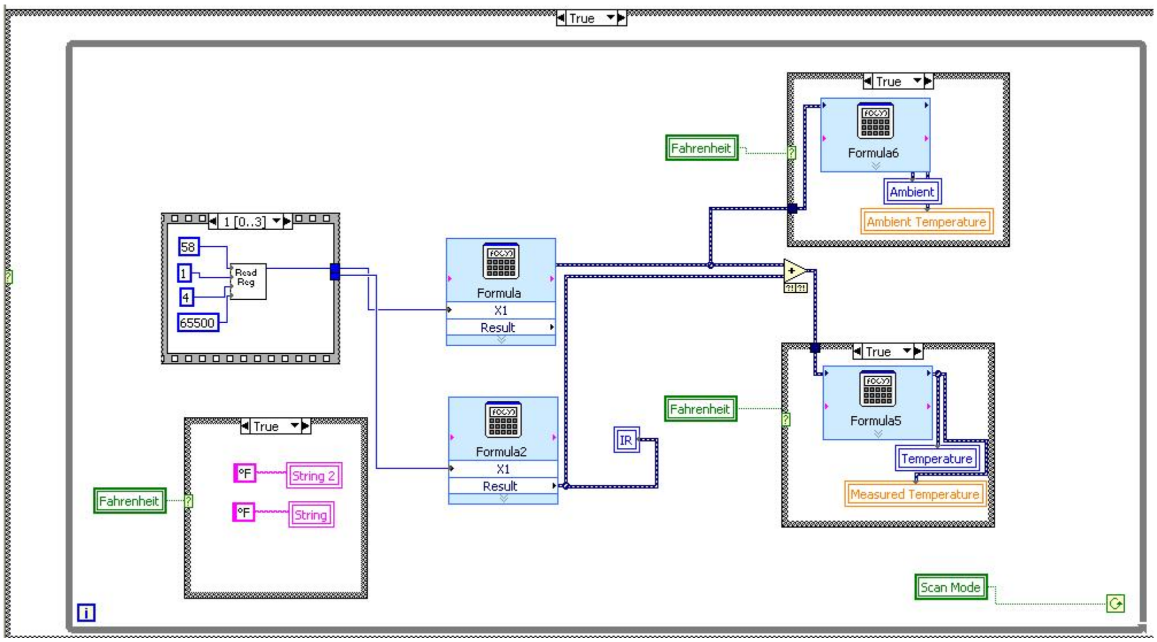


Figure 41: Scan Mode Software Algorithm

5.4.8 Normal Mode

Normal Mode is used when the user wants to take multiple data points, filter the points by averaging, and output both graphs and temperature readings. In Normal Mode, the user selects the desired number of samples. Normal Mode takes longer to update since sampling and processing takes time, but the output is more stable and yields more thorough results. This mode is used when the user does not need real-time functionality. A block diagram of Normal Mode is shown in Figure 42.

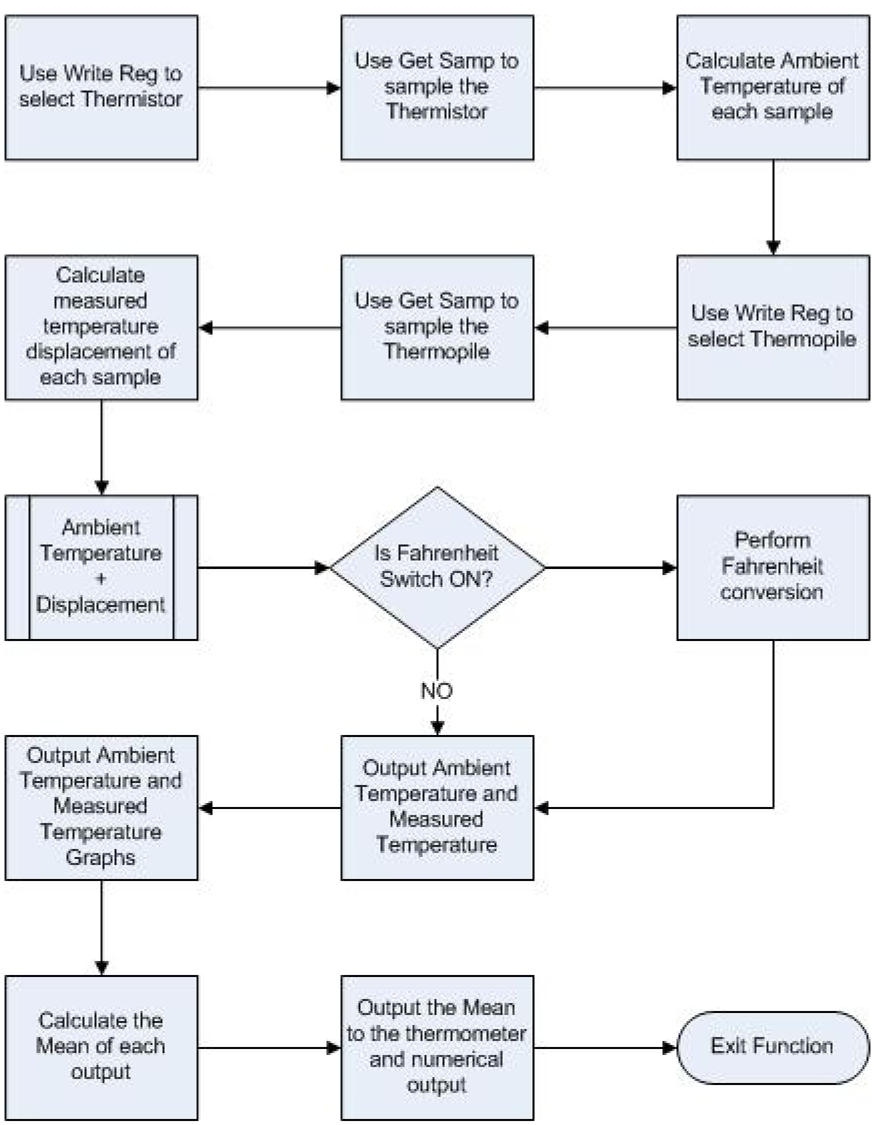


Figure 42: Normal Mode Block Diagram

This mode begins by using Write Reg and Get Samp to collect temperature data from both the thermopile and thermistor. Each array is then processed using the appropriate formulae. The software performs any needed Fahrenheit conversion, and outputs both the ambient temperature and measured temperature arrays to their respective graphs. The average temperature is then estimated by taking the mean of each array and outputting the values to the thermometer and numerical temperature displays. The LabView algorithm for this portion of the program is shown in Figure 43.

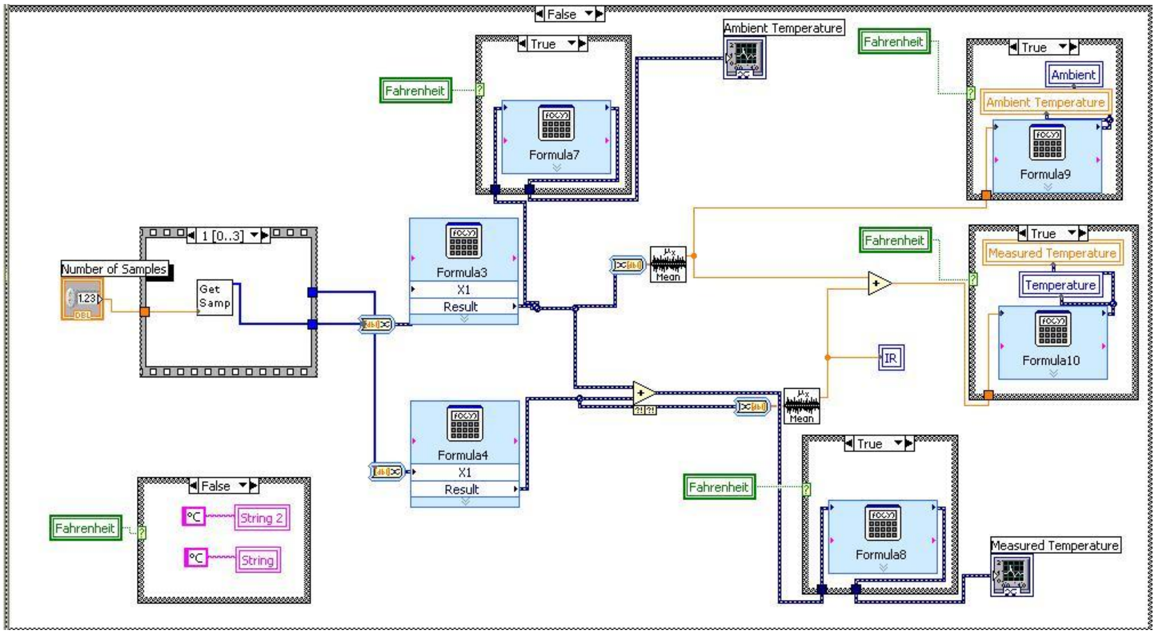


Figure 43: Normal Mode Software Algorithm

5.5 Final System Assembly

Figure 44 shows the final PCB layout for the sensor and ADC portion of our circuit. The Cyprus USB Evaluation Board connects with wires on the left side of the board. Our PowerLogic schematic was converted into a PCB layout by ADI applications engineer Brian O'Sullivan using PCB PADS software. The layout is arranged to minimise the length of the traces from the sensor to the ADC. The board is laid out in a thin rectangle so that it can easily be packaged in a box. The metal casting connects to a ground plane for optimal thermal conductivity.

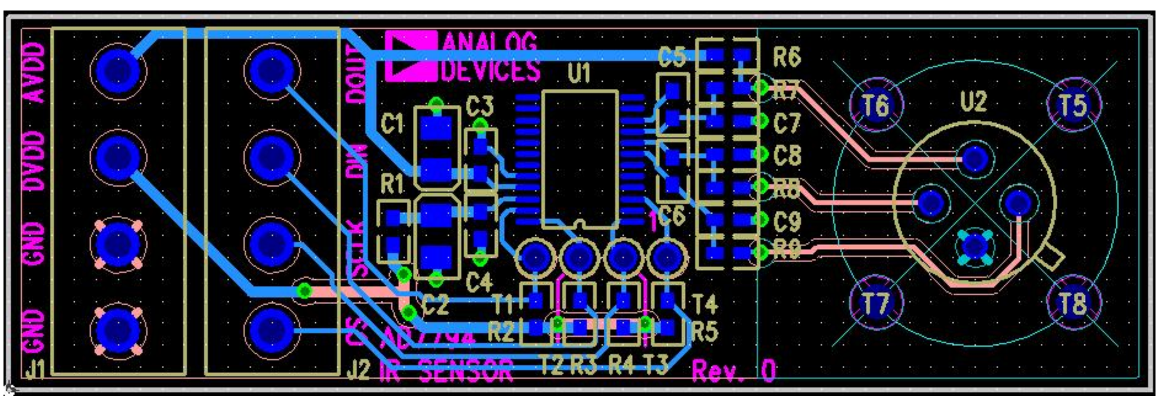


Figure 44: Final PCB Layout

Figure 45 shows a picture of our final circuit. The Cyprus USB Evaluation board interfaces with the PC and provides power to the device. We connected a laser for sighting to the USB board that is triggered using software.

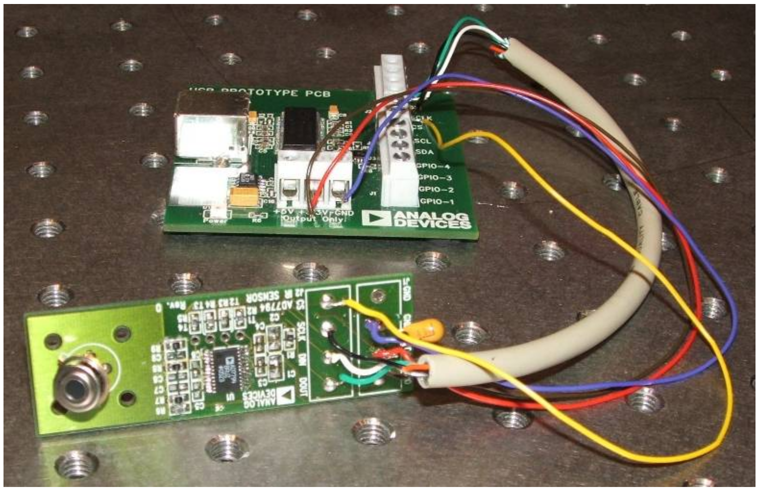


Figure 45: Final Circuit Board

Finally, we placed our PCB board back into the Fluke casing for the purpose of our prototype. We decided to implement a rotary switch for the laser system. The switch was placed on the package where the trigger previously existed. A serial cable was then used to connect our device to our USB board. A picture of our final spot thermometer is shown in Figure 46.

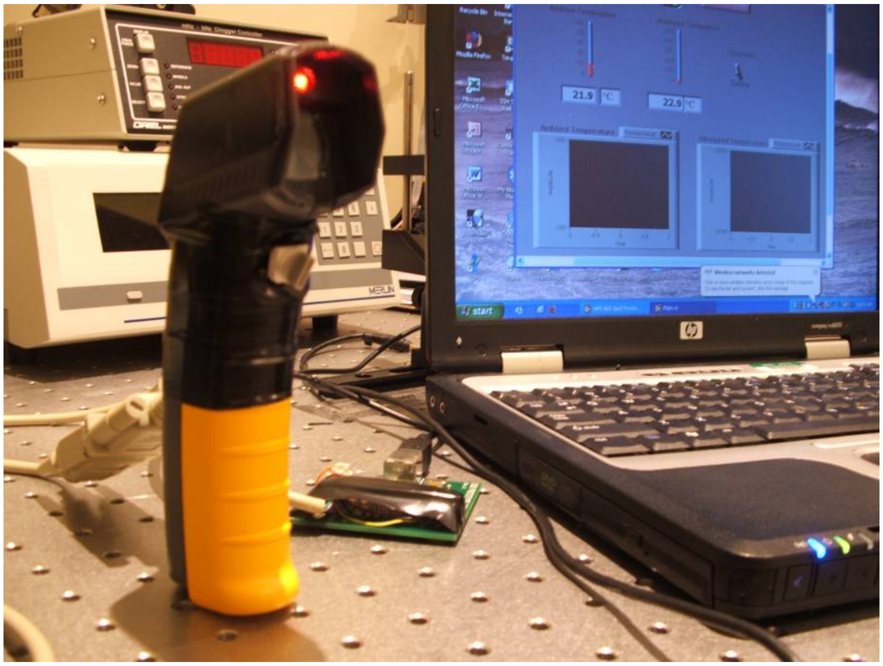


Figure 46: Final Product

6 Calibration and Testing

The following section provides information on the methods and results of the calibration and testing of our design. Each IR thermometer must be individually calibrated due to variances between sensors. For example, the Melexis thermopile used in our design is manufactured and specified to a tolerance of ±25%. Values in the thermistor coefficients also have variances of ±25%. As a result, we needed to first calibrate our thermometer to a blackbody source at a given distance. We then tested the final product to determine whether it met our original specifications.

In mass production, a testing strategy should be developed to limit the amount of time spent calibrating each device. Since each calibration point has significant costs from wages, equipment, and power, it is beneficial to reduce the time spent for each device calibration. For the purpose of our project, however, we decided to run thorough calibration tests for both the ambient temperature calibration and infrared calibration. As explained in the follow sections, this calibration was hard-coded into the software of our design.

6.1 Ambient Temperature Calibration

The Melexis MLX90247 sensor contains a built in ambient temperature sensor. The change in resistance of the thermistor (and therefore the change in voltage at the input of the ADC) is a quadratic function of ambient temperature and varies by up to 20% depending on the individual IR sensor used. For this reason, it was necessary to calibrate our spot pyrometer to the specific thermistor inside our sensor.

To perform the calibration, we took five voltage readings with the device at ambient temperatures ranging from 3 to 42˚C. However, the most accurate thermometer we were able to obtain was only accurate to ±2%. Therefore, the ambient temperature reading of our final prototype is only accurate by this amount instead of 1%. This can also add error up to 2% for remote temperature measurements that are conducted at ambient temperatures differing from the 22˚C that our device was calibrated at. To determine the thermistor voltage vs. temperature relationship, we calculated a linear least-squares regression equation based on the recorded data. We then solved the regression equation for temperature as a function of voltage. We used this result to calculate the ambient temperature as a function of the ADC's input voltage. A graph showing our ambient temperature calibration line is shown in Figure 47.

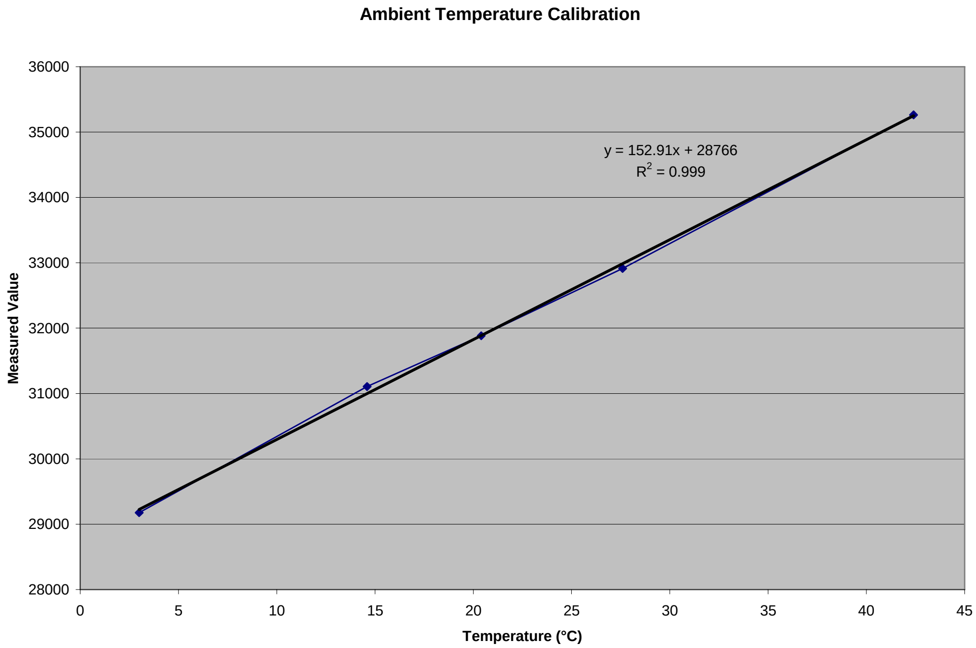


Figure 47: Ambient Temperature Calibration

6.2 Thermopile Calibration

The thermopile IR sensor produces an output voltage proportional to the difference in temperature of the heat source (or heat sink) and the temperature of its casing. The induced voltage is a quadratic function of the temperature difference between the hot and cold junction. Therefore, it is necessary to calibrate the sensor because the magnitude of the relationship depends on the specific sensor.

We calibrated the thermopile sensor every 10˚C, starting at 30 degrees and ending at 450 degrees Celsius. We calculated a quadratic least-squares regression equation based on these data points to determine the quadratic voltage vs. temperature difference equation for the thermopile. We then used the quadratic formula to solve the best fit equation for temperature as a function of voltage. The result is a function to calculate the temperature difference as a function of the square root of the ADC's input voltage.

Since we did not have cold temperature sources available for calibration, we used the same regression equation for temperature sources above and below ambient. We did not have any equipment available to test if this approximation is valid, so we recommend further study to determine if the operating characteristics of the thermopile are the same both above and below ambient temperature. A graph showing our thermopile calibration is shown in Figure 48.

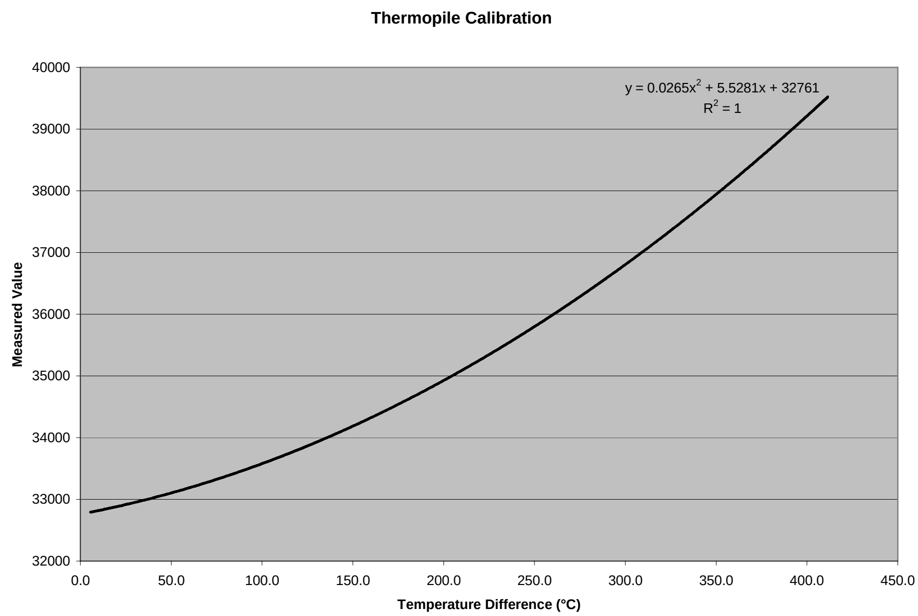


Figure 48: Thermopile Calibration

6.3 System Testing

We tested the thermal response of our device from 30 to 450˚C to verify its consistency with our calibration test. The results of this test show that the system is accurate within one percent of its original calibration. As previously mentioned, we did not have any cold temperature sources available to test our pyrometer on objects below ambient temperature. We theorise that the thermopile will behave the same at cold temperatures as it does at hot temperatures. Figure 49 shows the graph of percent error vs. temperature for our device when we tested its thermal response. Details on how well our device met each of its specifications are found in section 8.

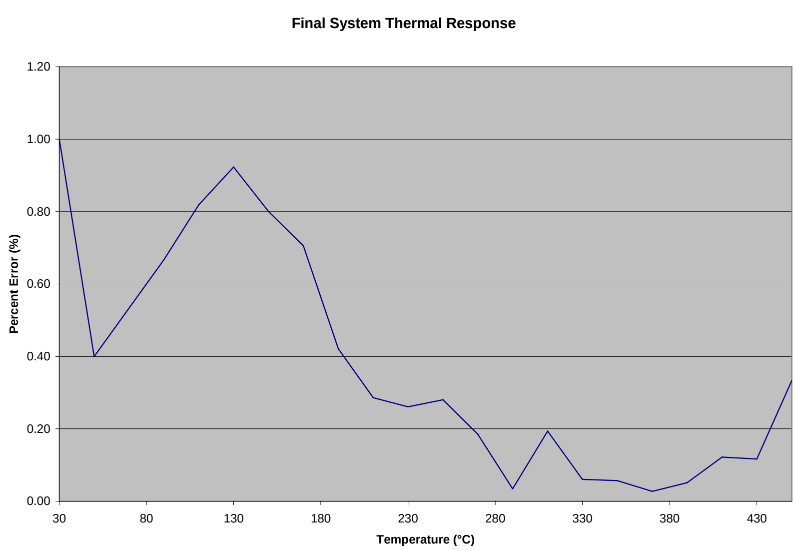


Figure 49: Final System Thermal Response

7 Recommendations

While we successfully accomplished our goals and were within specifications for the spot pyrometer, we have several recommendations for ADI in order to improve both the infrared thermometer prototype, as well as thermopile sensors produced in the future. Section 7.1 addresses recommendations to improve our IR thermometer prototype. Section 7.2 addresses our recommendations on IR thermopile sensors.

7.1 Prototype Recommendations

In order to complete our project in time, we implemented our spot pyrometer using the USB evaluation board provided by Analog Devices, as mentioned in section 5.2. This board was connected through wires to our sensor/ADC board. For this reason, the final circuit was slightly larger than necessary, and this restricted our packaging options. Also, longer wires have the tendency to produce noise, which interfere with the thermopile signal. **Our group recommends a new PCB layout with the USB microcontroller and supporting circuitry all on a single board.** While the USB microcontroller can reach high temperatures, our group feels this will not affect the sensor in an adverse way if the microcontroller and sensor are located on the opposite ends of the board.

Manufacturing a lens system and unique thermal mass will require a large amount of time. As a result, our group reused the optics system present in the Fluke 62 thermometer. Due to the specifications of the lens and the Fluke 62 sensor, the field of view in our prototype was roughly half that of the Fluke 62. Without a custom optics system, it was impossible to develop a better field of view. **We recommend that ADI develop a unique optics system for any future prototype.** With a unique lens, thermal mass, and sensor aperture, ADI will be capable of defining their own field of view. Also, by manufacturing their own optics system, ADI will learn more about the trade off between sensor sensitivity and field of view.

For similar reasons mentioned above, our group decided to reuse the laser system that was present in the Fluke 62. **For future prototypes, we recommend that ADI investigate and purchase a unique laser and supporting circuitry.** This will allow ADI to customise the targeting laser and manufacture a complete spot pyrometer for demonstration purposes.

Finally, our group decided to reuse the Fluke 62 packaging for our prototype. Our PCB fit into the Fluke 62 packaging and allowed us to implement this rather than design our own prototype packaging. **Our group recommends that ADI implement its own packaging for any future IR thermometer prototypes.**

7.2 Sensor Development Recommendations

Our group also has several recommendations for improving future thermopile sensors manufactured by ADI. **We first recommend that ADI focus on improving sensor sensitivity in future products.** While a quicker time response could be appealing in high-speed manufacturing settings, most of our specifications and design decisions were based on sensor sensitivity. With a higher sensitivity, it is possible to have an increased distance to spot ratio, a specification that is important in non-contact temperature sensing. Also, with a higher sensitivity, the output voltage for a given temperature difference will be higher. This could allow a designer to use little or no amplification in the system. Smaller amounts of amplification result in smaller amounts of noise in the temperature reading.

We recommend that ADI determine the spectrum sensitivity of commercially available thermopiles. This would allow them to determine if sensitivity to certain wavelengths can improve sensor performance. We recommend that ADI research spectrum sensitivity further and determine whether this characteristic can give them an advantage in the market.

We recommend that ADI repeat the time constant tests described in section 4.2.2. Although we are confident in our time constant estimation of approximately 100 ms, the test can be performed in a more controlled and accurate manner. We determined that when exposed to very high temperatures, the sensor can output a voltage measurable on an oscilloscope. However, since we did not have access to a fast shutter, we could not determine the time constant directly from the thermopile and we were forced to use an evaluation board. We recommend that ADI purchase a fast shutter and repeat this test to more accurately determine the time constants of the sensors.

Finally, we recommend that ADI pursue implementing a distance sensor with the IR sensor package. This distance sensor can used to correct the variation in readings due to a change in distance to the object being measured. This relationship between distance and output is documented in section 4.1.2. Also, a distance sensor would allow information such as distance from the object or spot size to be displayed for the end user.

8 Conclusion

We accomplished our goals of evaluating current IR sensing technology and constructing a spot pyrometer. We evaluated two commercially available spot pyrometers and two thermopile sensors. Additionally, we designed our own spot pyrometer. Based on the results of our final system tests, we meet the performance specifications for our system that are defined in section 3. This section describes our specific project accomplishments as they relate to our original project goals.

- Evaluate IR sensing technology currently on the market for non-contact temperatures measurement
    - Completed thermal response tests, distance tests, field of view tests, thermal shock tests and non-equilibrium tests to analyse the Fluke and Raytek thermometers to determine their system performance.
    - Disassembled and investigated the Fluke and Raytek system structures.
    - Characterised and evaluated the thermal response and time constant of the sensor used with the Fluke system and a Melexis sensor.
- Construct a handheld infrared thermometer
    - The spot pyrometer was calibrated with a blackbody source for a temperature range of -25 to 450 ˚C. A 24-bit ADC allows for resolution smaller than 0.2 ˚C.
    - Using an on-board thermistor in the sensor, the system has an automatic ambient temperature adjustment.
    - The thermopile signal is fed directly to the ADC and requires no filtering, enabling a response time of less than 500 ms.
    - The system has a distance to spot ratio of 5:1, which is created using the original Fluke optics.
    - The system is powered by the 5V USB interface.
    - The analogue board for the system fits in the original Fluke packaging with the USB board connected on the outside of the case. The laser from the Fluke system fits back into its original position, creating a portable handheld system.
    - The information is processed in a standalone LabView program which outputs the temperature to the screen.
    - The complete system will have final retail price of around $60, well under the $100 specification.

Through our evaluation of current IR sensing technology, Analog Devices has a base of characteristics to help them understand the market they will be competing in. Our spot pyrometer evaluation and construction provides ADI with a system level understanding of an application using a thermopile sensor. The results of our project will help Analog Devices in the future as they enter the IR sensing market.

References

Analog Devices (2004). *AD7787 Datasheet.* Analog Devices Inc.

Analog Devices (2005). *AD7794 Datasheet.* Analog Devices Inc.

Analog Devices (2002). *AD8551 Datasheet.* Analog Devices Inc.

Burns, Jim. (1999). “Resistive Thermometers.” *Temperature Measurement*. EngNetBase, CRC Press LLC

Chaplin, Martin. (2005). “Molecular Vibration and Absorption of Water Molecules”. Retrieved on the World Wide Web on August 25, 2005: http://www.lsbu.ac.uk/water/vibrat.html

Fluke. (2005). *Fluke 62 Mini Infrared Thermometer Manual*. Fluke Corporation.

Fraden, Jacob. (1999). “Infrared Thermometers.” *Temperature Measurement*. EngNetBase, CRC Press LLC

Fraden, Jacob. (2004). *Handbook of Modern Sensors: Physics, Designs, and Applications*. 3rd ed. Springer Science and Business Media Inc., New York.

Hartmann, Andreas. (2003, April). PIR: Detectors for Security. *Sensors Magazine.* Retrieved from the World Wide Web on 9th of September: http://www.sensorsmag.com/articles/0403/35/main.shtml

ISA: InTech with Industrial Computing. (2002). “Danaher Corporation Agrees to Buy Raytek Corporation.” Retrieved on the World Wide Web on September 9, 2005: http://www.isa.org/InTechTemplate.cfm?Section=InTech&template=/ContentManagement/ContentDisplay.cfm&ContentID=18072

Melexis. (2004). *MLX90247 Family Datasheet*, Melexis Corporation.

Nave, C.R. “Electromagnetic Spectrum.” Retrieved from the World Wide Web on 18th of August: http://hyperphysics.phy-astr.gsu.edu/hbase/ems3.html

Porro, Irene and Flanagan, Kathryn. (2001). “The Electromagnetic Spectrum.” Retrieved from the World Wide Web on 18th of August: http://space.mit.edu/CSR/outreach/MULTILING/CHANDRA_ENG_ITA/Slide5.html

Raytek. “Glossary of Terms.” Retrieved from the World Wide Web on 18th of August: http://www.raytek-northamerica.com/tools/glossary

Raytek. (2002). *Minitemp MT2 and MT4 manual*. Raytek Corporation.

Reed, R.P. (1999). "Thermocouple Thermometers." *Temperature Measurement*. EngNetBase, CRC Press, LLC

"Research Machines". (2005). Retrieved from the World Wide Web on 18th of August: http://www.tiscali.co.uk/reference/encylopaedia/hutchinson/m0031042.html

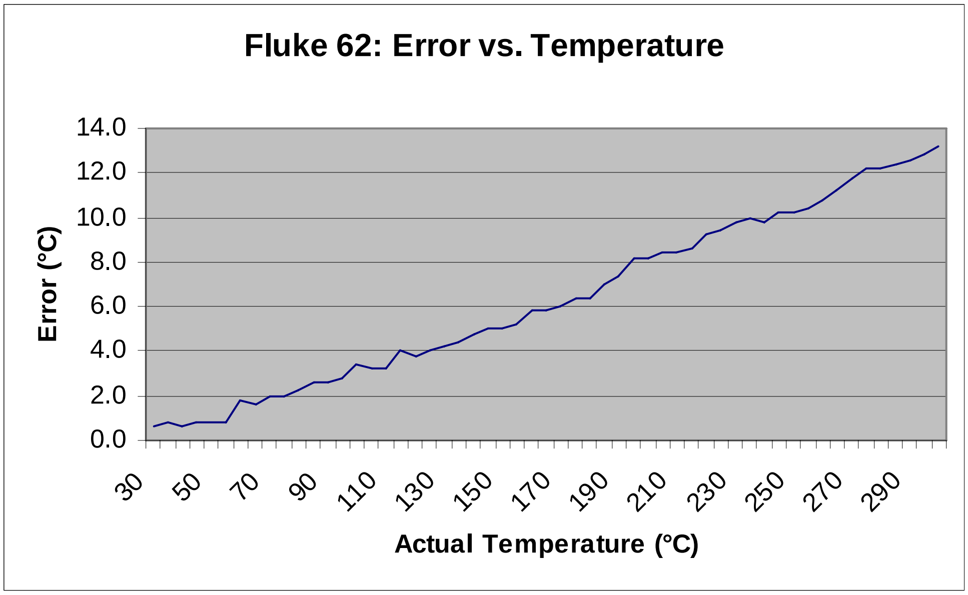
"RReDC Glossary of Solar Radiation Resource Terms". Retrieved from the World Wide Web on 18th of August 2005: http://rredc.nrel.gov/solar/glossary/gloss_e.html

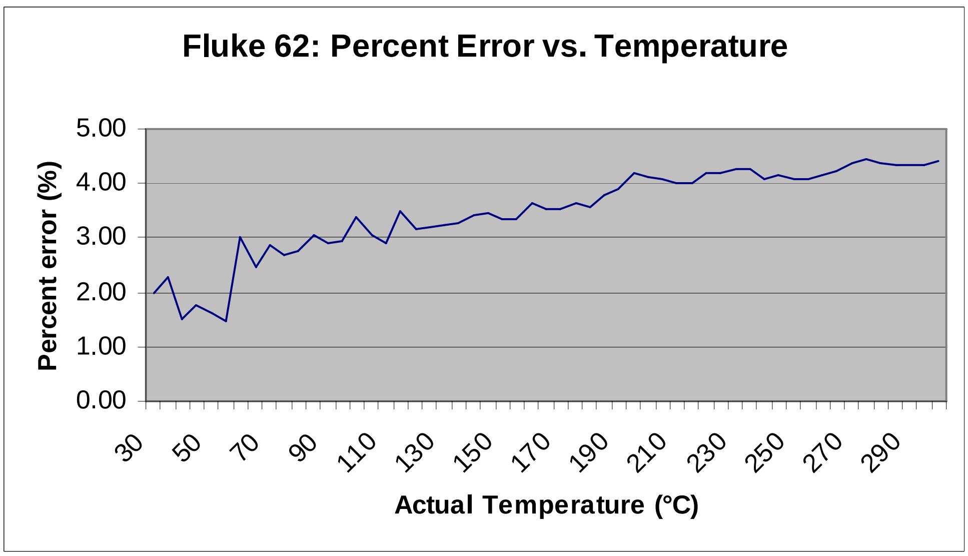
Appendix A: Test Data and Results

The following appendix contains the raw data and test results relating to section 4: System and Sensor Performance Evaluation. This section includes all tests we performed and graphs of the resulting data.

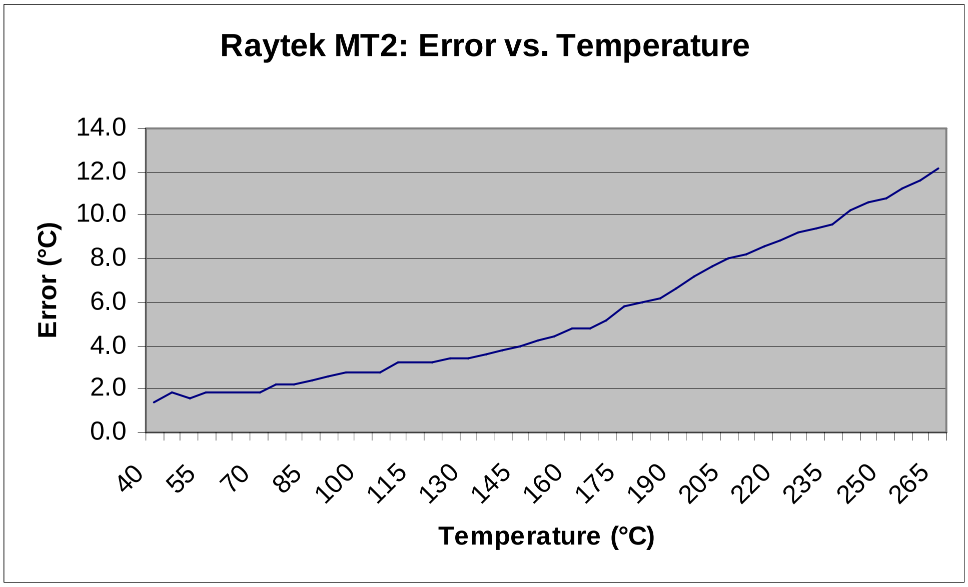
A.1 Thermal Response

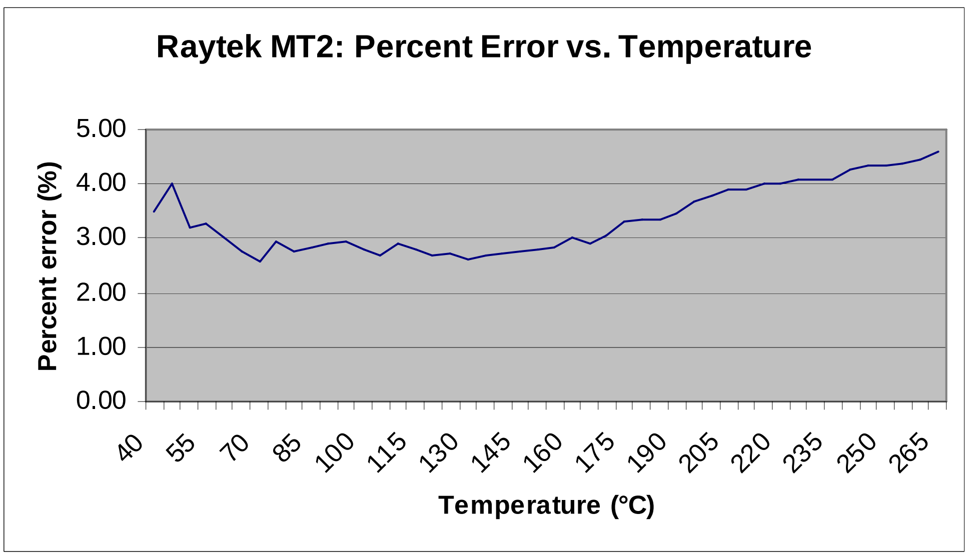
11/8/2005 13:00 - 14:00			
Fluke 62			
Thermal Response			
Distance from lens to blackbody: 25.51 cm			
Temp set (°C)	Measured Temp (°C)	Error (°C)	Percent Error (%)
30	30.6	0.6	2.00
35	35.8	0.8	2.29
40	40.6	0.6	1.50
45	45.8	0.8	1.78
50	50.8	0.8	1.60
55	55.8	0.8	1.45
60	61.8	1.8	3.00
65	66.6	1.6	2.46
70	72.0	2.0	2.86
75	77.0	2.0	2.67
80	82.2	2.2	2.75
85	87.6	2.6	3.06
90	92.6	2.6	2.89
95	97.8	2.8	2.95
100	103.4	3.4	3.40
105	108.2	3.2	3.05
110	113.2	3.2	2.91
115	119.0	4.0	3.48
120	123.8	3.8	3.17
125	129.0	4.0	3.20
130	134.2	4.2	3.23
135	139.4	4.4	3.26
140	144.8	4.8	3.43
145	150.0	5.0	3.45
150	155.0	5.0	3.33
155	160.2	5.2	3.35
160	165.8	5.8	3.63
165	170.8	5.8	3.52
170	176.0	6.0	3.53
175	181.4	6.4	3.66
180	186.4	6.4	3.56
185	192.0	7.0	3.78
190	197.4	7.4	3.89
195	203.2	8.2	4.21
200	208.2	8.2	4.10
205	213.4	8.4	4.10
210	218.4	8.4	4.00
215	223.6	8.6	4.00
220	229.2	9.2	4.18
225	234.4	9.4	4.18
230	239.8	9.8	4.26
235	245.0	10.0	4.26
240	249.8	9.8	4.08
245	255.2	10.2	4.16
250	260.2	10.2	4.08
255	265.4	10.4	4.08
260	270.8	10.8	4.15
265	276.2	11.2	4.23
270	281.8	11.8	4.37
275	287.2	12.2	4.44
280	292.2	12.2	4.36
285	297.4	12.4	4.35
290	302.6	12.6	4.34
295	307.8	12.8	4.34
300	313.2	13.2	4.40

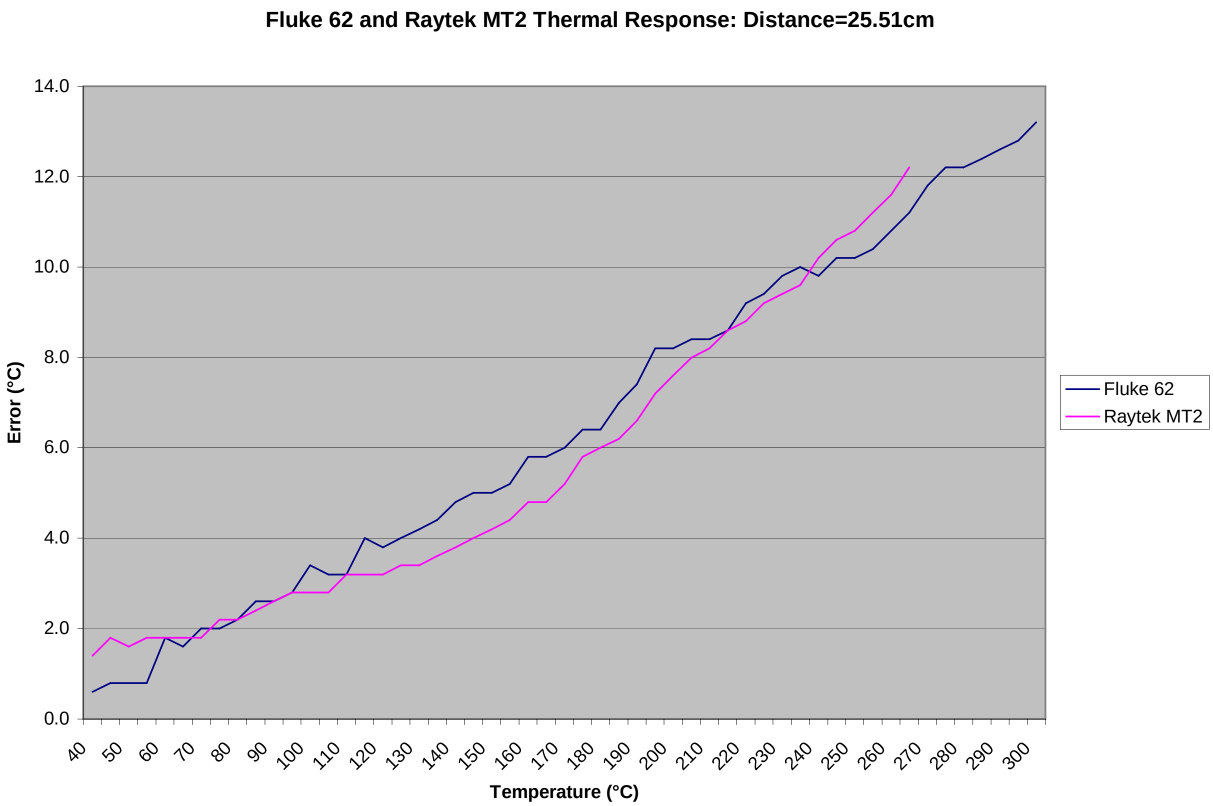

Fluke 62: Error vs. Temperature
Error (°C)
14.0
12.0
10.0
8.0
6.0
4.0
2.0
0.0
30
50
70
90
110
130
150
170
190
210
230
250
270
290
Actual Temperature (°C)


Fluke 62: Percent Error vs. Temperature
Percent error (%)
5.00
4.00
3.00
2.00
1.00
0.00
30
50
70
90
110
130
150
170
190
210
230
250
270
290
Actual Temperature (°C)

11/8/2005 15:45			
Raytek MT2			
Thermal Response			
Distance from lens to blackbody: 25.5 cm			
Set Temp (°C)	Measured Temp (°C)	Error (°C)	Percent Error (%)
40	41.4	1.4	3.50
45	46.8	1.8	4.00
50	51.6	1.6	3.20
55	56.8	1.8	3.27
60	61.8	1.8	3.00
65	66.8	1.8	2.77
70	71.8	1.8	2.57
75	77.2	2.2	2.93
80	82.2	2.2	2.75
85	87.4	2.4	2.82
90	92.6	2.6	2.89
95	97.8	2.8	2.95
100	102.8	2.8	2.80
105	107.8	2.8	2.67
110	113.2	3.2	2.91
115	118.2	3.2	2.78
120	123.2	3.2	2.67
125	128.4	3.4	2.72
130	133.4	3.4	2.62
135	138.6	3.6	2.67
140	143.8	3.8	2.71
145	149.0	4.0	2.76
150	154.2	4.2	2.80
155	159.4	4.4	2.84
160	164.8	4.8	3.00
165	169.8	4.8	2.91
170	175.2	5.2	3.06
175	180.8	5.8	3.31
180	186.0	6.0	3.33
185	191.2	6.2	3.35
190	196.6	6.6	3.47
195	202.2	7.2	3.69
200	207.6	7.6	3.80
205	213.0	8.0	3.90
210	218.2	8.2	3.90
215	223.6	8.6	4.00
220	228.8	8.8	4.00
225	234.2	9.2	4.09
230	239.4	9.4	4.09
235	244.6	9.6	4.09
240	250.2	10.2	4.25
245	255.6	10.6	4.33
250	260.8	10.8	4.32
255	266.2	11.2	4.39
260	271.6	11.6	4.46
265	277.2	12.2	4.60
270	---	---	---
275	---	---	---
280	---	---	---
285	---	---	---
290	---	---	---
295	---	---	---
300	---	---	---

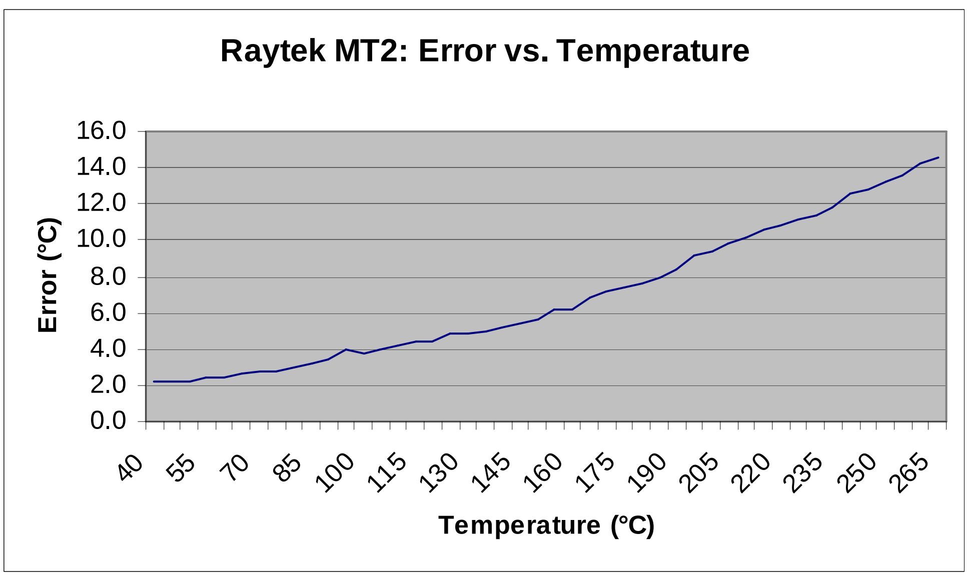

Raytek MT2: Error vs. Temperature
Error (°C)
14.0
12.0
10.0
8.0
6.0
4.0
2.0
0.0
40
55
70
85
100
115
130
145
160
175
190
205
220
235
250
265
Temperature (°C)

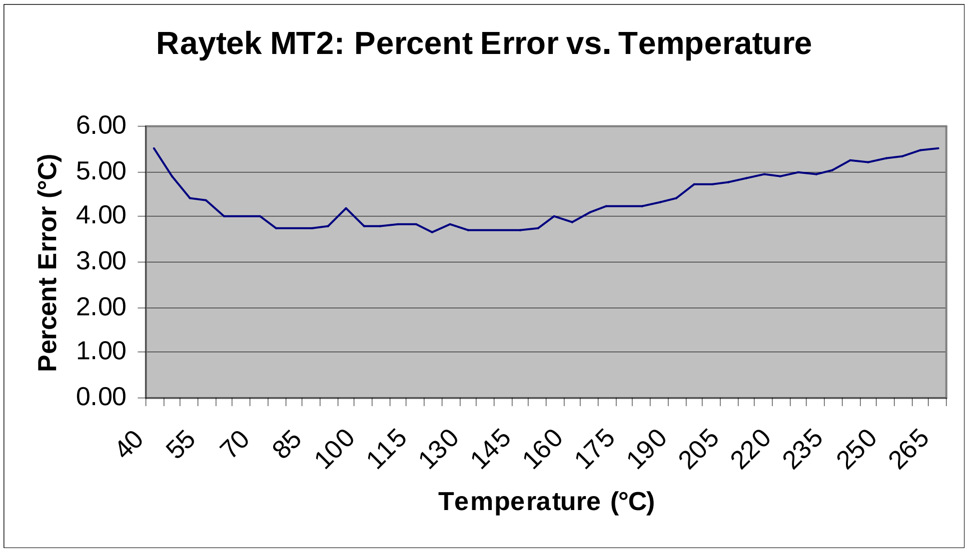

Raytek MT2: Percent Error vs. Temperature
Percent error (%)
5.00
4.00
3.00
2.00
1.00
0.00
40
55
70
85
100
115
130
145
160
175
190
205
220
235
250
265
Temperature (°C)

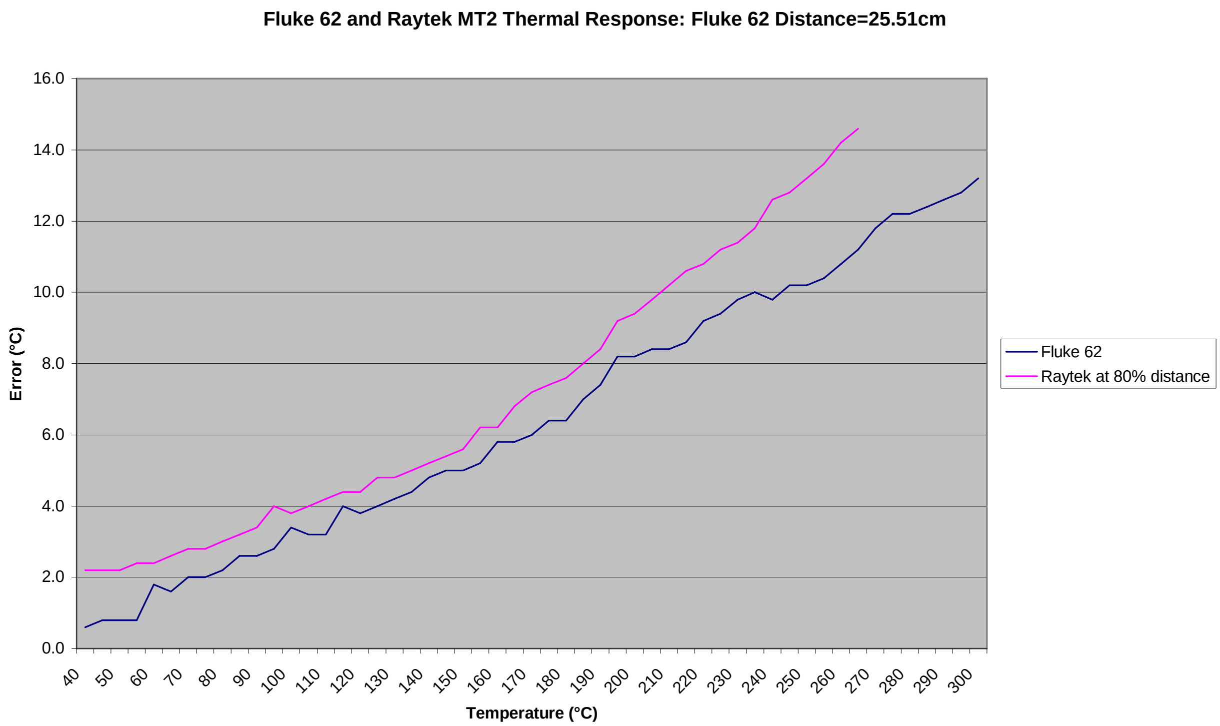


Fluke 62 and Raytek MT2 Thermal Response: Distance=25.51cm
Error (°C)
14.0
12.0
10.0
8.0
6.0
4.0
2.0
0.0
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260
270
280
290
300
Temperature (°C)
Fluke 62
Raytek MT2

16/8/05 10:30 - 11:00			
Raytek MT2			
Thermal Response			
Distance from lens to blackbody: 20.4 cm			
Temp set (°C)	Measured Temp (°C)	Error (°C)	Percent Error (%)
40	42.2	2.2	5.50
45	47.2	2.2	4.89
50	52.2	2.2	4.40
55	57.4	2.4	4.36
60	62.4	2.4	4.00
65	67.6	2.6	4.00
70	72.8	2.8	4.00
75	77.8	2.8	3.73
80	83.0	3.0	3.75
85	88.2	3.2	3.76
90	93.4	3.4	3.78
95	99.0	4.0	4.21
100	103.8	3.8	3.80
105	109.0	4.0	3.81
110	114.2	4.2	3.82
115	119.4	4.4	3.83
120	124.4	4.4	3.67
125	129.8	4.8	3.84
130	134.8	4.8	3.69
135	140.0	5.0	3.70
140	145.2	5.2	3.71
145	150.4	5.4	3.72
150	155.6	5.6	3.73
155	161.2	6.2	4.00
160	166.2	6.2	3.87
165	171.8	6.8	4.12
170	177.2	7.2	4.24
175	182.4	7.4	4.23
180	187.6	7.6	4.22
185	193.0	8.0	4.32
190	198.4	8.4	4.42
195	204.2	9.2	4.72
200	209.4	9.4	4.70
205	214.8	9.8	4.78
210	220.2	10.2	4.86
215	225.6	10.6	4.93
220	230.8	10.8	4.91
225	236.2	11.2	4.98
230	241.4	11.4	4.96
235	246.8	11.8	5.02
240	252.6	12.6	5.25
245	257.8	12.8	5.22
250	263.2	13.2	5.28
255	268.6	13.6	5.33
260	274.2	14.2	5.46
265	279.6	14.6	5.51
270	---	---	---
275	---	---	---
280	---	---	---
285	---	---	---
290	---	---	---
295	---	---	---
300	---	---	---




Raytek MT2: Percent Error vs. Temperature
Percent Error (°C)
6.00
5.00
4.00
3.00
2.00
1.00
0.00
40
55
70
85
100
115
130
145
160
175
190
205
220
235
250
265
Temperature (°C)

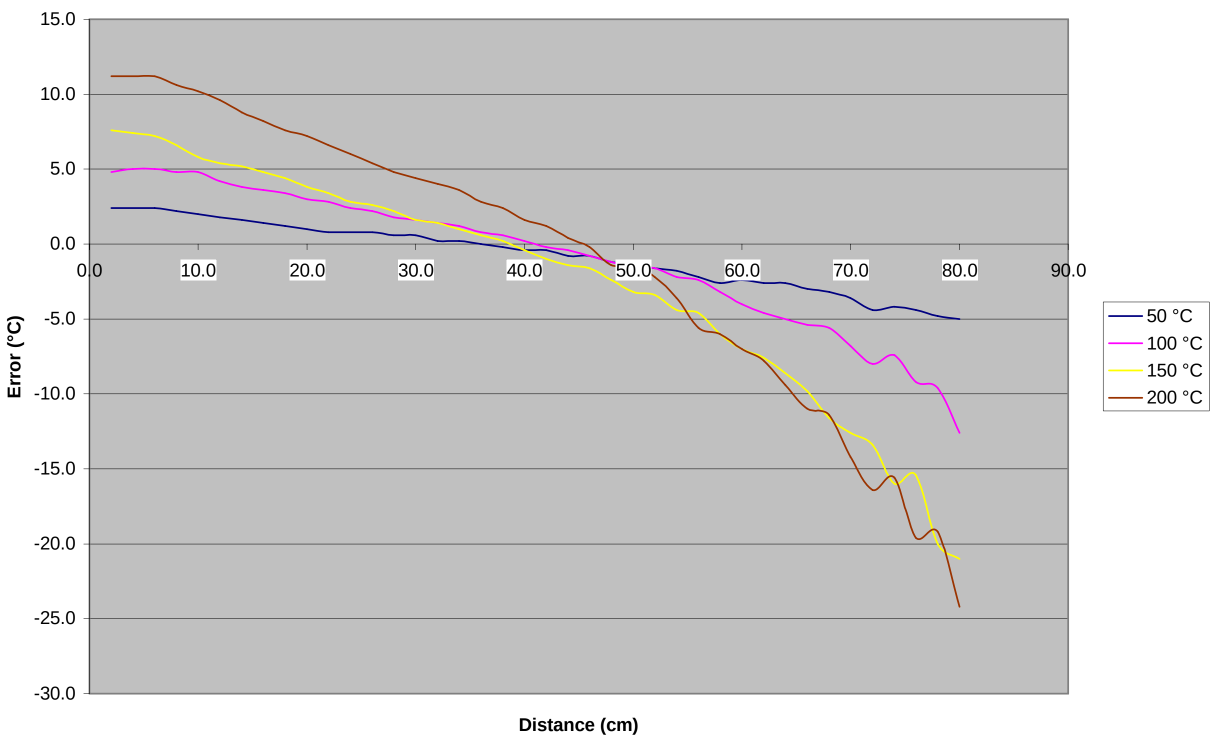

Fluke 62 and Raytek MT2 Thermal Response: Fluke 62 Distance=25.51cm
Error (°C)
16.0
14.0
12.0
10.0
8.0
6.0
4.0
2.0
0.0
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260
270
280
290
300
Temperature (°C)
Fluke 62
Raytek at 80% distance

A.2 Distance Test

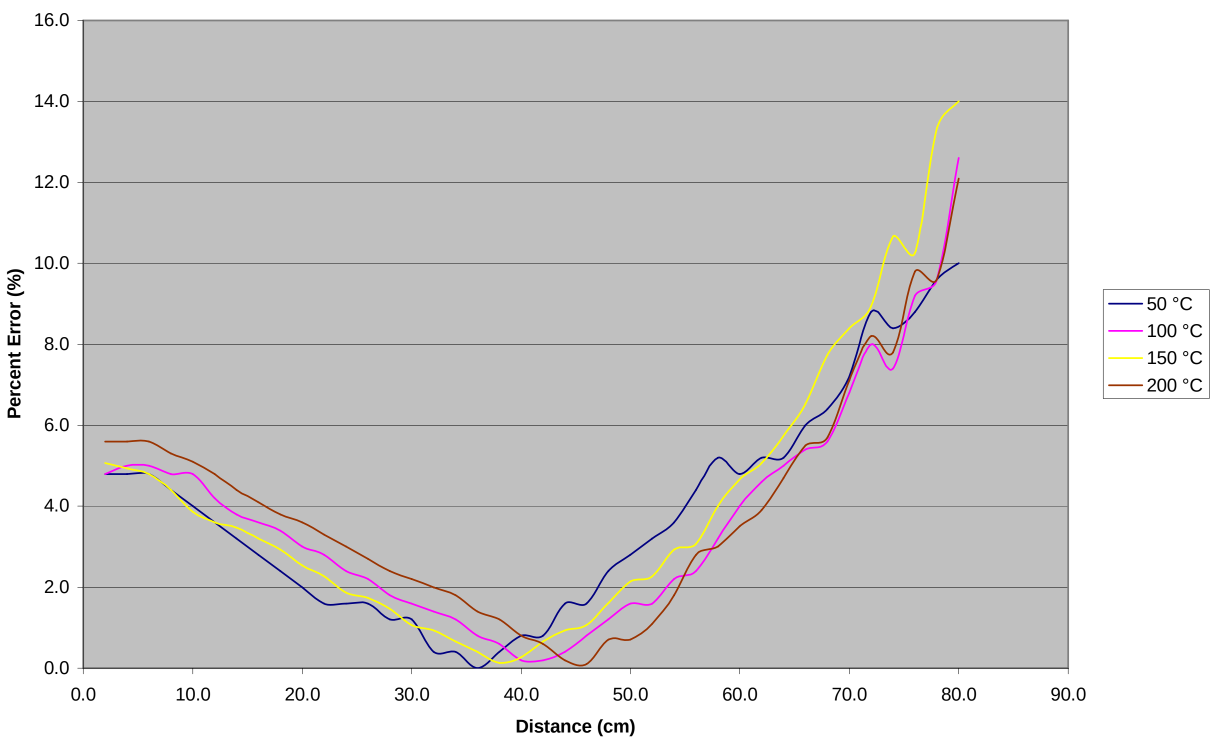
17/8/05 11:00 - 12:00						
Fluke 62						
Distance test 2						
50 °C				100 °C		
Position (cm)	Temp (°C)	Error (°C)	% Error	Temp (°C)	Error (°C)	% Error
2.0	52.4	2.4	4.8	104.8	4.8	4.8
4.0	52.4	2.4	4.8	105.0	5.0	5.0
6.0	52.4	2.4	4.8	105.0	5.0	5.0
8.0	52.2	2.2	4.4	104.8	4.8	4.8
10.0	52.0	2.0	4.0	104.8	4.8	4.8
12.0	51.8	1.8	3.6	104.2	4.2	4.2
14.0	51.6	1.6	3.2	103.8	3.8	3.8
16.0	51.4	1.4	2.8	103.6	3.6	3.6
18.0	51.2	1.2	2.4	103.4	3.4	3.4
20.0	51.0	1.0	2.0	103.0	3.0	3.0
22.0	50.8	0.8	1.6	102.8	2.8	2.8
24.0	50.8	0.8	1.6	102.4	2.4	2.4
26.0	50.8	0.8	1.6	102.2	2.2	2.2
28.0	50.6	0.6	1.2	101.8	1.8	1.8
30.0	50.6	0.6	1.2	101.6	1.6	1.6
32.0	50.2	0.2	0.4	101.4	1.4	1.4
34.0	50.2	0.2	0.4	101.2	1.2	1.2
36.0	50.0	0.0	0.0	100.8	0.8	0.8
38.0	49.8	-0.2	0.4	100.6	0.6	0.6
40.0	49.6	-0.4	0.8	100.2	0.2	0.2
42.0	49.6	-0.4	0.8	99.8	-0.2	0.2
44.0	49.2	-0.8	1.6	99.6	-0.4	0.4
46.0	49.2	-0.8	1.6	99.2	-0.8	0.8
48.0	48.8	-1.2	2.4	98.8	-1.2	1.2
50.0	48.6	-1.4	2.8	98.4	-1.6	1.6
52.0	48.4	-1.6	3.2	98.4	-1.6	1.6
54.0	48.2	-1.8	3.6	97.8	-2.2	2.2
56.0	47.8	-2.2	4.4	97.6	-2.4	2.4
58.0	47.4	-2.6	5.2	96.8	-3.2	3.2
60.0	47.6	-2.4	4.8	96.0	-4.0	4.0
62.0	47.4	-2.6	5.2	95.4	-4.6	4.6
64.0	47.4	-2.6	5.2	95.0	-5.0	5.0
66.0	47.0	-3.0	6.0	94.6	-5.4	5.4
68.0	46.8	-3.2	6.4	94.4	-5.6	5.6
70.0	46.4	-3.6	7.2	93.2	-6.8	6.8
72.0	45.6	-4.4	8.8	92.0	-8.0	8.0
74.0	45.8	-4.2	8.4	92.6	-7.4	7.4
76.0	45.6	-4.4	8.8	90.8	-9.2	9.2
78.0	45.2	-4.8	9.6	90.4	-9.6	9.6
80.0	45.0	-5.0	10.0	87.4	-12.6	12.6

17/8/05 11:00 - 12:00						
Fluke 62						
Distance test 2						
	150 °C			200 °C		
Position (cm)	Temp (°C)	Error (°C)	% Error	Temp (°C)	Error (°C)	% Error
2.0	157.6	7.6	5.1	211.2	11.2	5.6
4.0	157.4	7.4	4.9	211.2	11.2	5.6
6.0	157.2	7.2	4.8	211.2	11.2	5.6
8.0	156.6	6.6	4.4	210.6	10.6	5.3
10.0	155.8	5.8	3.9	210.2	10.2	5.1
12.0	155.4	5.4	3.6	209.6	9.6	4.8
14.0	155.2	5.2	3.5	208.8	8.8	4.4
16.0	154.8	4.8	3.2	208.2	8.2	4.1
18.0	154.4	4.4	2.9	207.6	7.6	3.8
20.0	153.8	3.8	2.5	207.2	7.2	3.6
22.0	153.4	3.4	2.3	206.6	6.6	3.3
24.0	152.8	2.8	1.9	206.0	6.0	3.0
26.0	152.6	2.6	1.7	205.4	5.4	2.7
28.0	152.2	2.2	1.5	204.8	4.8	2.4
30.0	151.6	1.6	1.1	204.4	4.4	2.2
32.0	151.4	1.4	0.9	204.0	4.0	2.0
34.0	151.0	1.0	0.7	203.6	3.6	1.8
36.0	150.6	0.6	0.4	202.8	2.8	1.4
38.0	150.2	0.2	0.1	202.4	2.4	1.2
40.0	149.6	-0.4	0.3	201.6	1.6	0.8
42.0	149.0	-1.0	0.7	201.2	1.2	0.6
44.0	148.6	-1.4	0.9	200.4	0.4	0.2
46.0	148.4	-1.6	1.1	199.8	-0.2	0.1
48.0	147.6	-2.4	1.6	198.6	-1.4	0.7
50.0	146.8	-3.2	2.1	198.6	-1.4	0.7
52.0	146.6	-3.4	2.3	197.8	-2.2	1.1
54.0	145.6	-4.4	2.9	196.4	-3.6	1.8
56.0	145.4	-4.6	3.1	194.4	-5.6	2.8
58.0	144.0	-6.0	4.0	194.0	-6.0	3.0
60.0	143.0	-7.0	4.7	193.0	-7.0	3.5
62.0	142.4	-7.6	5.1	192.2	-7.8	3.9
64.0	141.4	-8.6	5.7	190.6	-9.4	4.7
66.0	140.2	-9.8	6.5	189.0	-11.0	5.5
68.0	138.4	-11.6	7.7	188.6	-11.4	5.7
70.0	137.4	-12.6	8.4	185.8	-14.2	7.1
72.0	136.6	-13.4	8.9	183.6	-16.4	8.2
74.0	134.0	-16.0	10.7	184.4	-15.6	7.8
76.0	134.6	-15.4	10.3	180.4	-19.6	9.8
78.0	130.0	-20.0	13.3	180.8	-19.2	9.6
80.0	129.0	-21.0	14.0	175.8	-24.2	12.1

Fluke 62: Error vs. Distance from Source

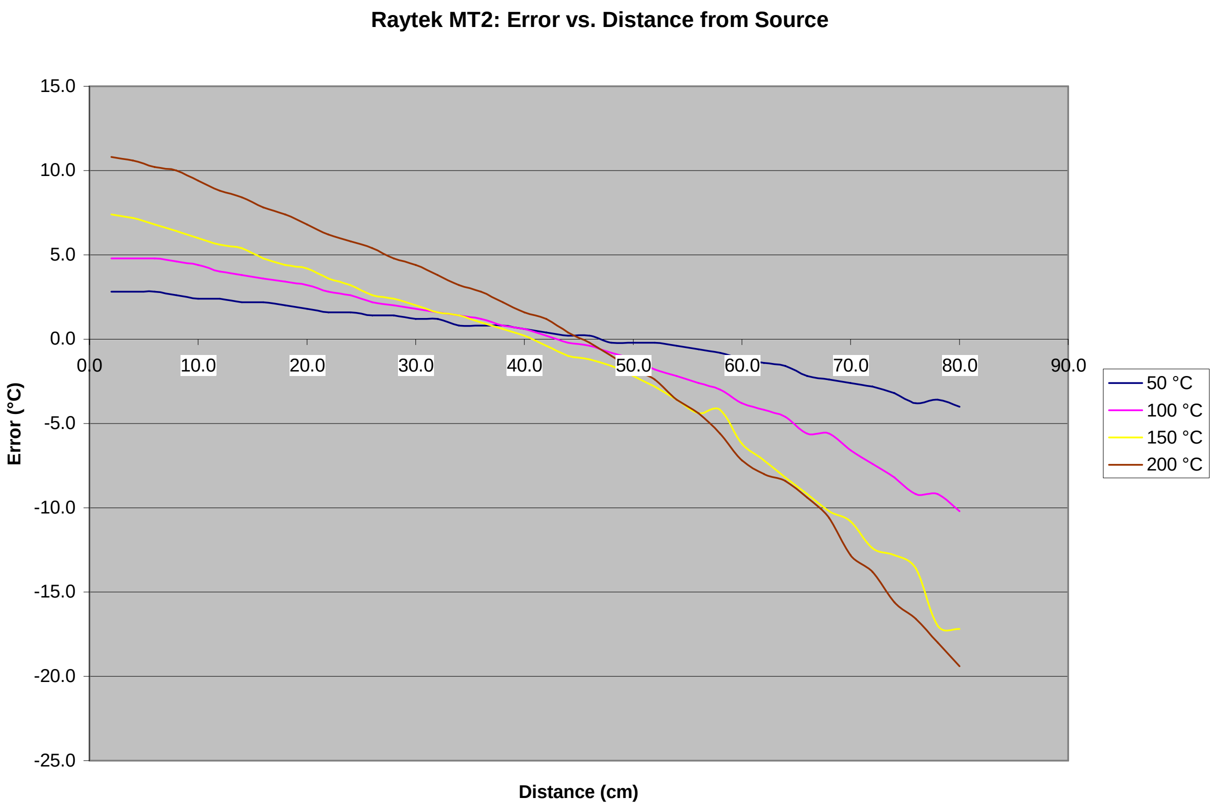


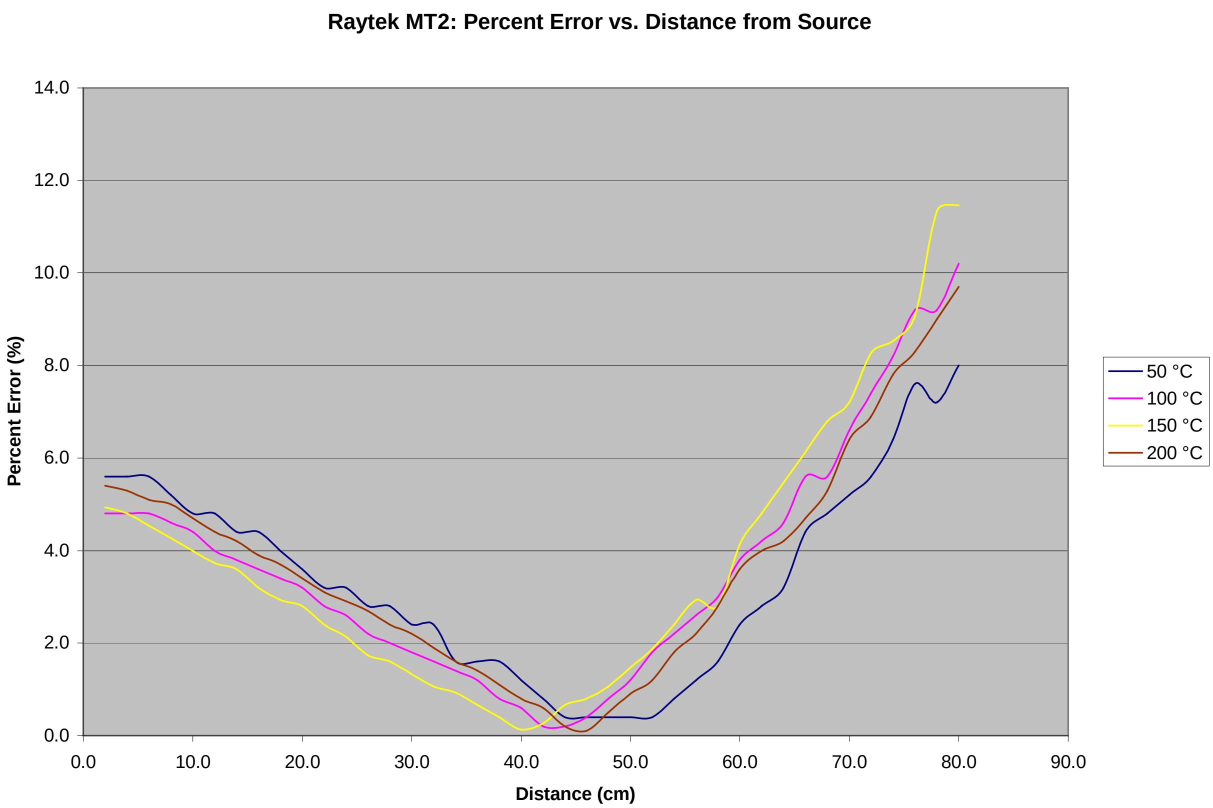
Fluke 62: Percent Error vs. Distance from Source

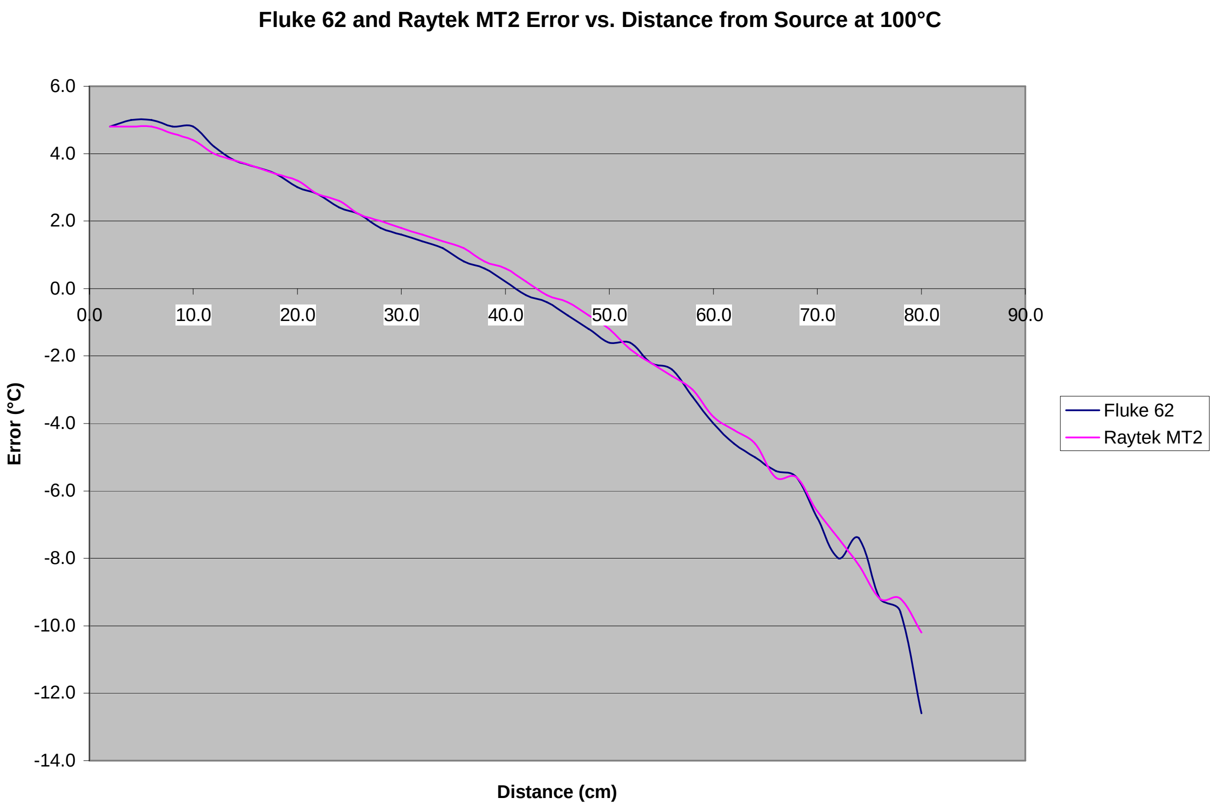


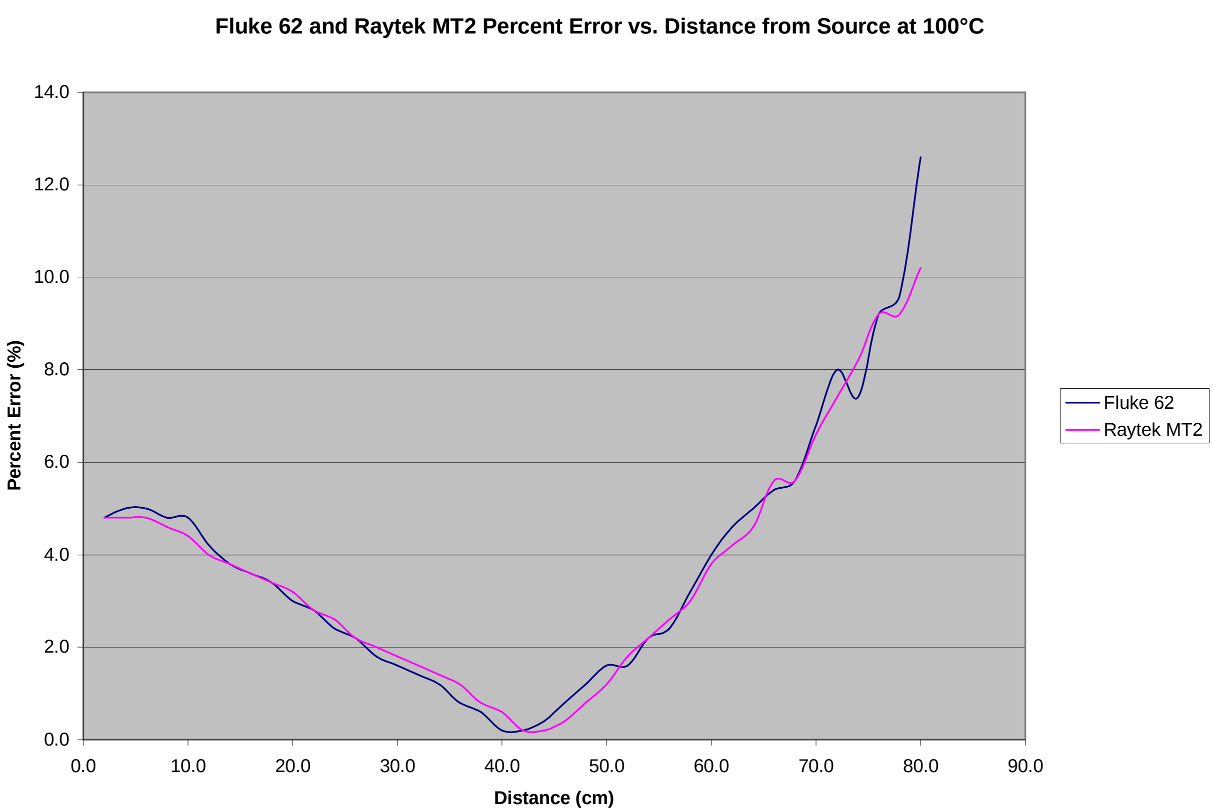
17/8/05 11:00 - 12:00						
Raytek MT2						
Distance test 2						
	50 °C			100 °C		
Position (cm)	Temp (°C)	Error (°C)	% Error	Temp (°C)	Error (°C)	% Error
2.0	52.8	2.8	5.6	104.8	4.8	4.8
4.0	52.8	2.8	5.6	104.8	4.8	4.8
6.0	52.8	2.8	5.6	104.8	4.8	4.8
8.0	52.6	2.6	5.2	104.6	4.6	4.6
10.0	52.4	2.4	4.8	104.4	4.4	4.4
12.0	52.4	2.4	4.8	104.0	4.0	4.0
14.0	52.2	2.2	4.4	103.8	3.8	3.8
16.0	52.2	2.2	4.4	103.6	3.6	3.6
18.0	52.0	2.0	4.0	103.4	3.4	3.4
20.0	51.8	1.8	3.6	103.2	3.2	3.2
22.0	51.6	1.6	3.2	102.8	2.8	2.8
24.0	51.6	1.6	3.2	102.6	2.6	2.6
26.0	51.4	1.4	2.8	102.2	2.2	2.2
28.0	51.4	1.4	2.8	102.0	2.0	2.0
30.0	51.2	1.2	2.4	101.8	1.8	1.8
32.0	51.2	1.2	2.4	101.6	1.6	1.6
34.0	50.8	0.8	1.6	101.4	1.4	1.4
36.0	50.8	0.8	1.6	101.2	1.2	1.2
38.0	50.8	0.8	1.6	100.8	0.8	0.8
40.0	50.6	0.6	1.2	100.6	0.6	0.6
42.0	50.4	0.4	0.8	100.2	0.2	0.2
44.0	50.2	0.2	0.4	99.8	-0.2	0.2
46.0	50.2	0.2	0.4	99.6	-0.4	0.4
48.0	49.8	-0.2	0.4	99.2	-0.8	0.8
50.0	49.8	-0.2	0.4	98.8	-1.2	1.2
52.0	49.8	-0.2	0.4	98.2	-1.8	1.8
54.0	49.6	-0.4	0.8	97.8	-2.2	2.2
56.0	49.4	-0.6	1.2	97.4	-2.6	2.6
58.0	49.2	-0.8	1.6	97.0	-3.0	3.0
60.0	48.8	-1.2	2.4	96.2	-3.8	3.8
62.0	48.6	-1.4	2.8	95.8	-4.2	4.2
64.0	48.4	-1.6	3.2	95.4	-4.6	4.6
66.0	47.8	-2.2	4.4	94.4	-5.6	5.6
68.0	47.6	-2.4	4.8	94.4	-5.6	5.6
70.0	47.4	-2.6	5.2	93.4	-6.6	6.6
72.0	47.2	-2.8	5.6	92.6	-7.4	7.4
74.0	46.8	-3.2	6.4	91.8	-8.2	8.2
76.0	46.2	-3.8	7.6	90.8	-9.2	9.2
78.0	46.4	-3.6	7.2	90.8	-9.2	9.2
80.0	46.0	-4.0	8.0	89.8	-10.2	10.2

17/8/05 11:00 - 12:00						
Raytek MT2						
Distance test 2						
	150 °C			200 °C		
Position (cm)	Temp (°C)	Error (°C)	% Error	Temp (°C)	Error (°C)	% Error
2.0	157.4	7.4	4.9	210.8	10.8	5.4
4.0	157.2	7.2	4.8	210.6	10.6	5.3
6.0	156.8	6.8	4.5	210.2	10.2	5.1
8.0	156.4	6.4	4.3	210.0	10.0	5.0
10.0	156.0	6.0	4.0	209.4	9.4	4.7
12.0	155.6	5.6	3.7	208.8	8.8	4.4
14.0	155.4	5.4	3.6	208.4	8.4	4.2
16.0	154.8	4.8	3.2	207.8	7.8	3.9
18.0	154.4	4.4	2.9	207.4	7.4	3.7
20.0	154.2	4.2	2.8	206.8	6.8	3.4
22.0	153.6	3.6	2.4	206.2	6.2	3.1
24.0	153.2	3.2	2.1	205.8	5.8	2.9
26.0	152.6	2.6	1.7	205.4	5.4	2.7
28.0	152.4	2.4	1.6	204.8	4.8	2.4
30.0	152.0	2.0	1.3	204.4	4.4	2.2
32.0	151.6	1.6	1.1	203.8	3.8	1.9
34.0	151.4	1.4	0.9	203.2	3.2	1.6
36.0	151.0	1.0	0.7	202.8	2.8	1.4
38.0	150.6	0.6	0.4	202.2	2.2	1.1
40.0	150.2	0.2	0.1	201.6	1.6	0.8
42.0	149.6	-0.4	0.3	201.2	1.2	0.6
44.0	149.0	-1.0	0.7	200.4	0.4	0.2
46.0	148.8	-1.2	0.8	199.8	-0.2	0.1
48.0	148.4	-1.6	1.1	199.0	-1.0	0.5
50.0	147.8	-2.2	1.5	198.2	-1.8	0.9
52.0	147.2	-2.8	1.9	197.6	-2.4	1.2
54.0	146.4	-3.6	2.4	196.4	-3.6	1.8
56.0	145.6	-4.4	2.9	195.6	-4.4	2.2
58.0	145.8	-4.2	2.8	194.4	-5.6	2.8
60.0	143.8	-6.2	4.1	192.8	-7.2	3.6
62.0	142.8	-7.2	4.8	192.0	-8.0	4.0
64.0	141.8	-8.2	5.5	191.6	-8.4	4.2
66.0	140.8	-9.2	6.1	190.6	-9.4	4.7
68.0	139.8	-10.2	6.8	189.4	-10.6	5.3
70.0	139.2	-10.8	7.2	187.2	-12.8	6.4
72.0	137.6	-12.4	8.3	186.2	-13.8	6.9
74.0	137.2	-12.8	8.5	184.4	-15.6	7.8
76.0	136.4	-13.6	9.1	183.4	-16.6	8.3
78.0	133.0	-17.0	11.3	182.0	-18.0	9.0
80.0	132.8	-17.2	11.5	180.6	-19.4	9.7


Raytek MT2: Error vs. Distance from Source
15.0
10.0
5.0
0.0
-5.0
-10.0
-15.0
-20.0
-25.0
Error (°C)
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
Distance (cm)
50 °C
100 °C
150 °C
200 °C

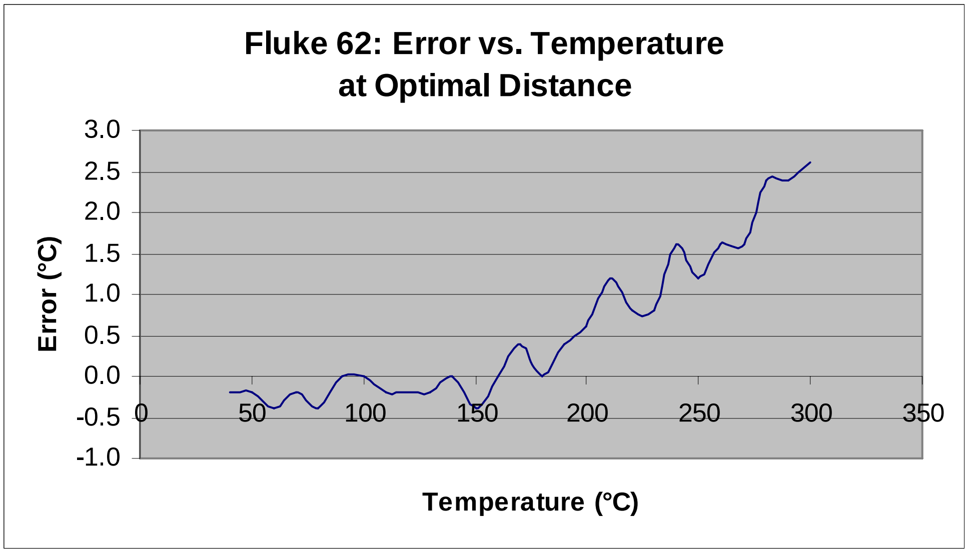

Raytek MT2: Percent Error vs. Distance from Source
14.0
12.0
10.0
8.0
6.0
4.0
2.0
0.0
Percent Error (%)
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
Distance (cm)
50 °C
100 °C
150 °C
200 °C

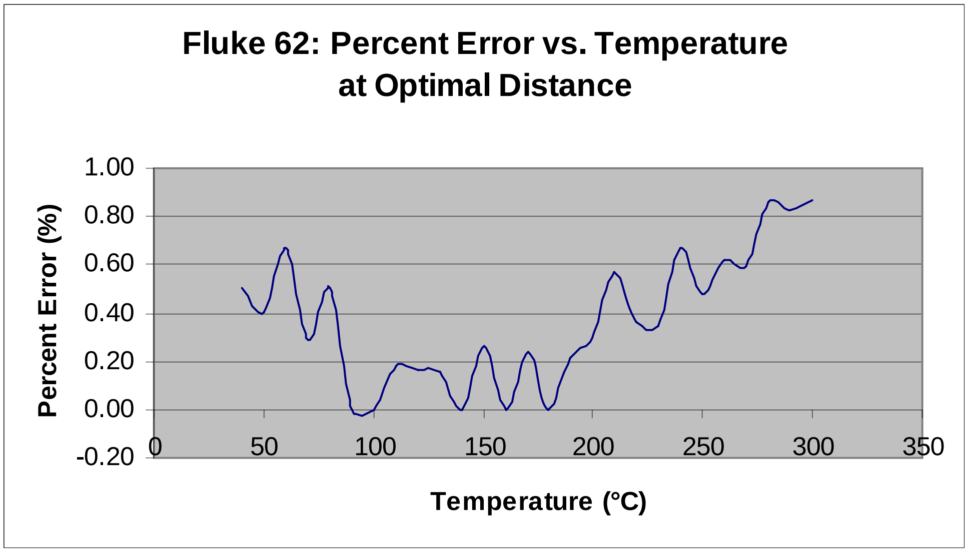

Fluke 62 and Raytek MT2 Error vs. Distance from Source at 100°C
6.0
4.0
2.0
0.0
-2.0
-4.0
-6.0
-8.0
-10.0
-12.0
-14.0
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
Error (°C)
Distance (cm)
Fluke 62
Raytek MT2


Fluke 62 and Raytek MT2 Percent Error vs. Distance from Source at 100°C
14.0
12.0
10.0
8.0
6.0
4.0
2.0
0.0
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
Percent Error (%)
Distance (cm)
Fluke 62
Raytek MT2

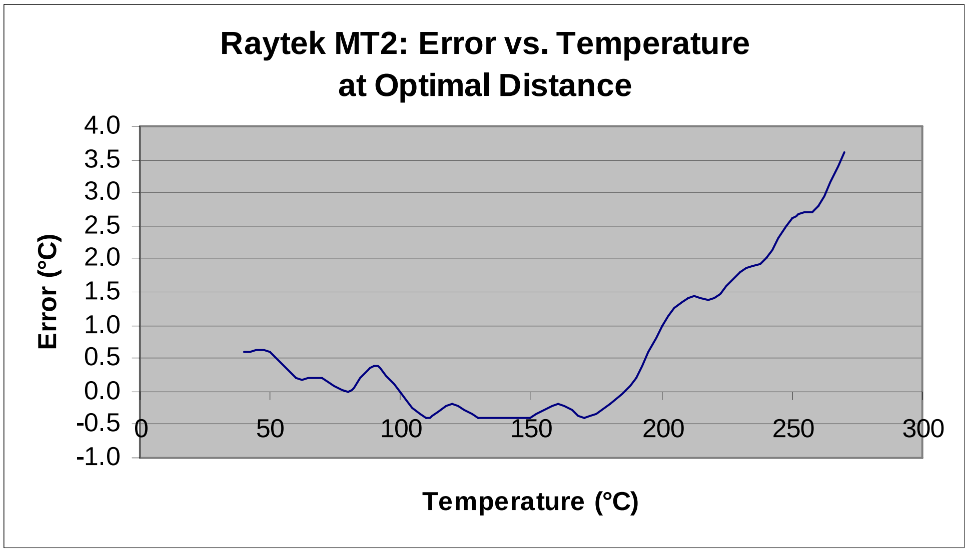
A.3 Optimal Thermal Response

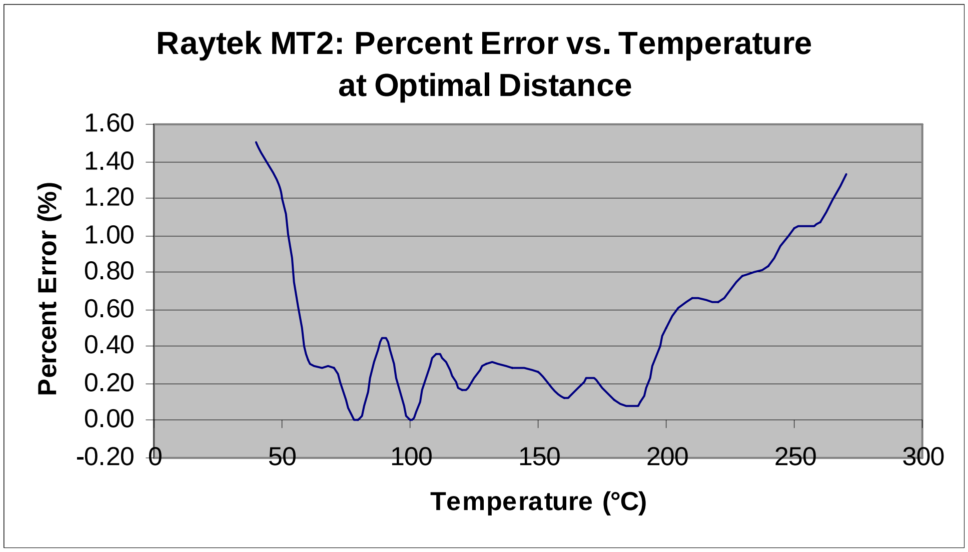
Fluke 62			
Thermal Response - 42 cm			
Distance from lens to blackbody: 42 cm			
Temp set (°C)	Measured Temp (°C)	Error (°C)	Percent Error (%)
40	39.8	-0.2	0.50
50	49.8	-0.2	0.40
60	59.6	-0.4	0.67
70	69.8	-0.2	0.29
80	79.6	-0.4	0.50
90	90.0	0.0	0.00
100	100.0	0.0	0.00
110	109.8	-0.2	0.18
120	119.8	-0.2	0.17
130	129.8	-0.2	0.15
140	140.0	0.0	0.00
150	149.6	-0.4	0.27
160	160.0	0.0	0.00
170	170.4	0.4	0.24
180	180.0	0.0	0.00
190	190.4	0.4	0.21
200	200.6	0.6	0.30
210	211.2	1.2	0.57
220	220.8	0.8	0.36
230	230.8	0.8	0.35
240	241.6	1.6	0.67
250	251.2	1.2	0.48
260	261.6	1.6	0.62
270	271.6	1.6	0.59
280	282.4	2.4	0.86
290	292.4	2.4	0.83
300	302.6	2.6	0.87

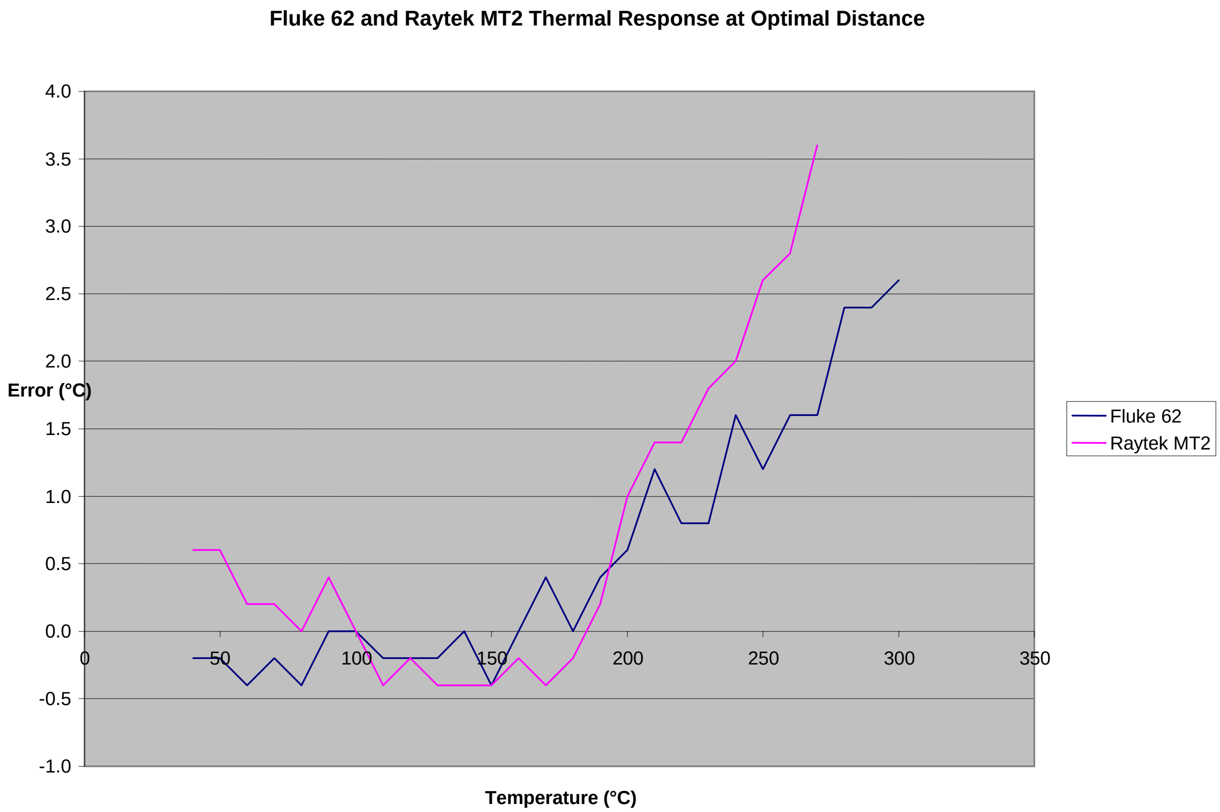




23/8/2005			
Raytek MT2			
Thermal Response - 45 cm			
Distance from lens to blackbody: 45 cm			
Temp set (°C)	Measured Temp (°C)	Error (°C)	Percent Error (%)
40	40.6	0.6	1.50
50	50.6	0.6	1.20
60	60.2	0.2	0.33
70	70.2	0.2	0.29
80	80.0	0.0	0.00
90	90.4	0.4	0.44
100	100.0	0.0	0.00
110	109.6	-0.4	0.36
120	119.8	-0.2	0.17
130	129.6	-0.4	0.31
140	139.6	-0.4	0.29
150	149.6	-0.4	0.27
160	159.8	-0.2	0.12
170	169.6	-0.4	0.24
180	179.8	-0.2	0.11
190	190.2	0.2	0.11
200	201.0	1.0	0.50
210	211.4	1.4	0.67
220	221.4	1.4	0.64
230	231.8	1.8	0.78
240	242.0	2.0	0.83
250	252.6	2.6	1.04
260	262.8	2.8	1.08
270	273.6	3.6	1.33


Raytek MT2: Error vs. Temperature
at Optimal Distance
Error (°C)
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
0
50
100
150
200
250
300
Temperature (°C)


Raytek MT2: Percent Error vs. Temperature
at Optimal Distance
Percent Error (%)
1.60
1.40
1.20
1.00
0.80
0.60
0.40
0.20
0.00
-0.20
0
50
100
150
200
250
300
Temperature (°C)



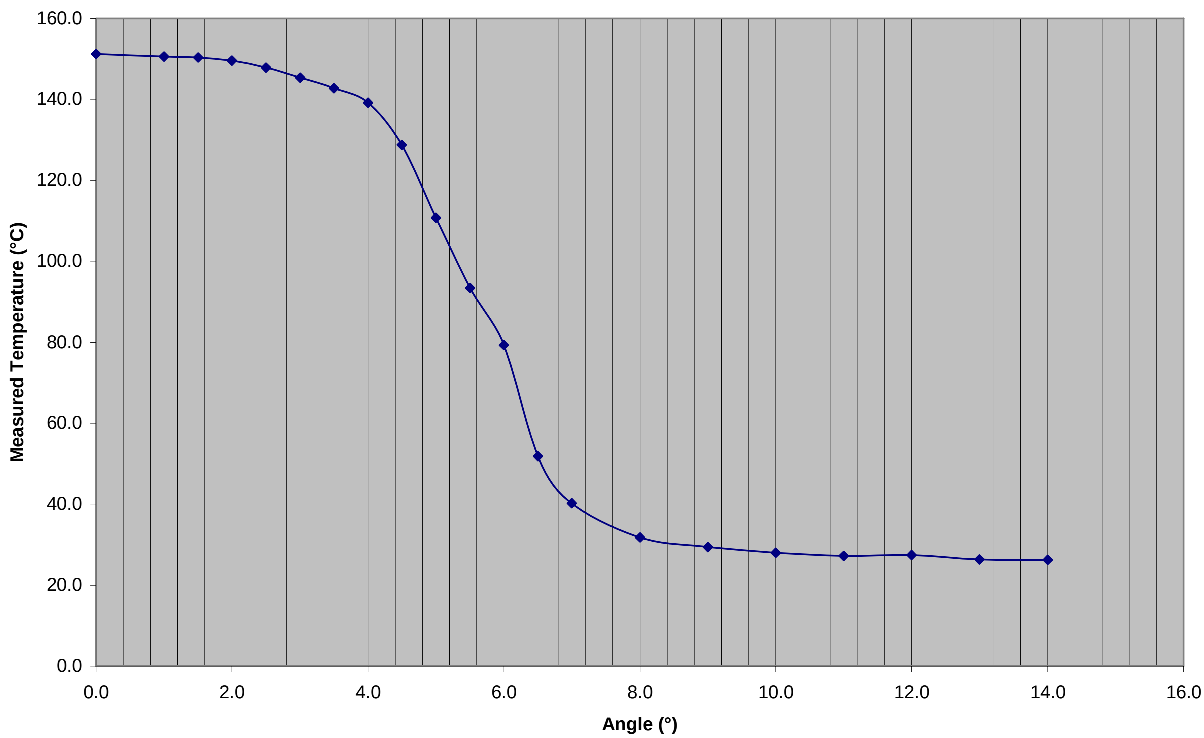
A.4 Field of View

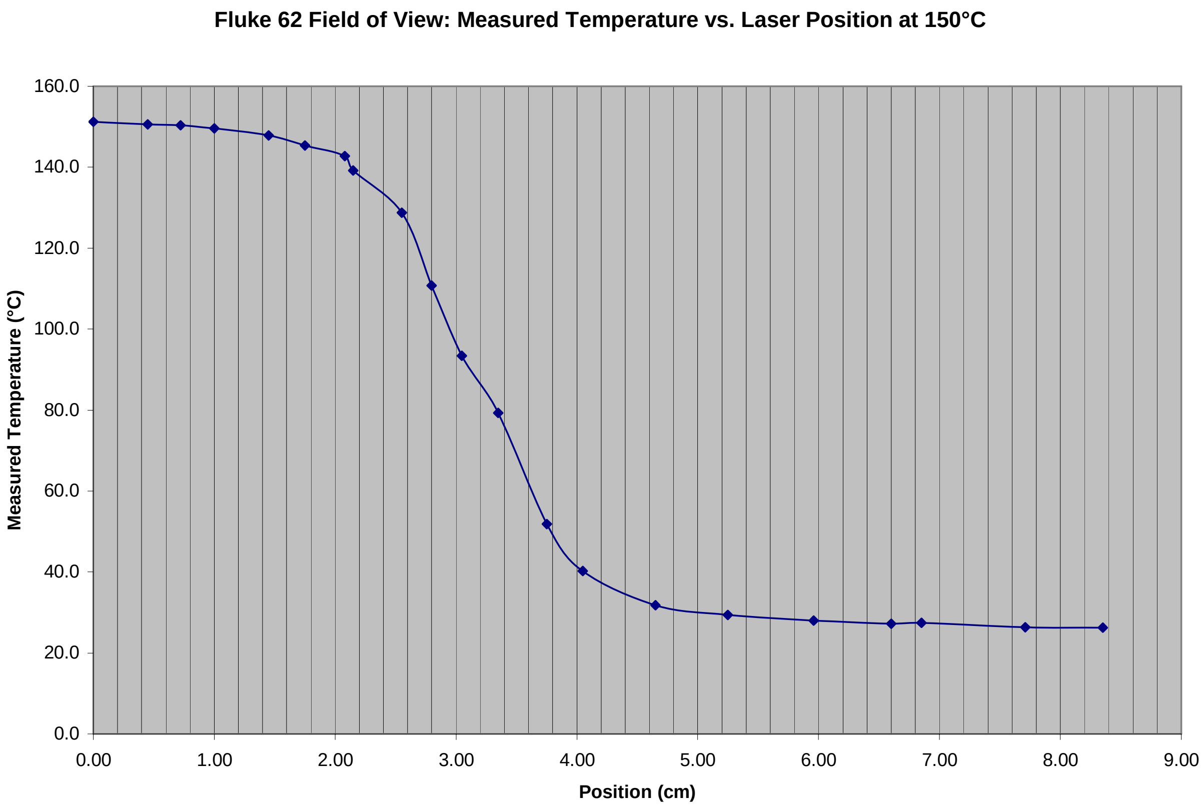
15/8/05 10:21			
Fluke 62			
Field of View (Distance to Spot)			
Distance from lens to blackbody: 37.05 cm			Blackbody Temp: 150
Position (cm)	Angle (°)	Temp (°C)	Disp by angle (cm)
8.35	14.0	26.2	9.237594383
7.71	13.0	26.4	8.553659002
6.85	12.0	27.4	7.875213759
6.60	11.0	27.2	7.201784218
5.96	10.0	28.0	6.532909003
5.25	9.0	29.4	5.868138475
4.65	8.0	31.8	5.20703347
4.05	7.0	40.2	4.5491641
3.75	6.5	51.8	4.221310691
3.35	6.0	79.3	3.894108603
3.05	5.5	93.4	3.567506204
2.80	5.0	110.8	3.241452232
2.55	4.5	128.8	2.915895765

2.15	4.0	139.2	2.590786187
2.08	3.5	142.8	2.266073158
1.75	3.0	145.4	1.941706579
1.45	2.5	147.8	1.617636567
1.00	2.0	149.6	1.293813416
0.72	1.5	150.3	0.970187574
0.45	1.0	150.6	0.646709609
0.00	0.0	151.2	0

	Position	Angle
D:S estimate	10.15068	9.475703325

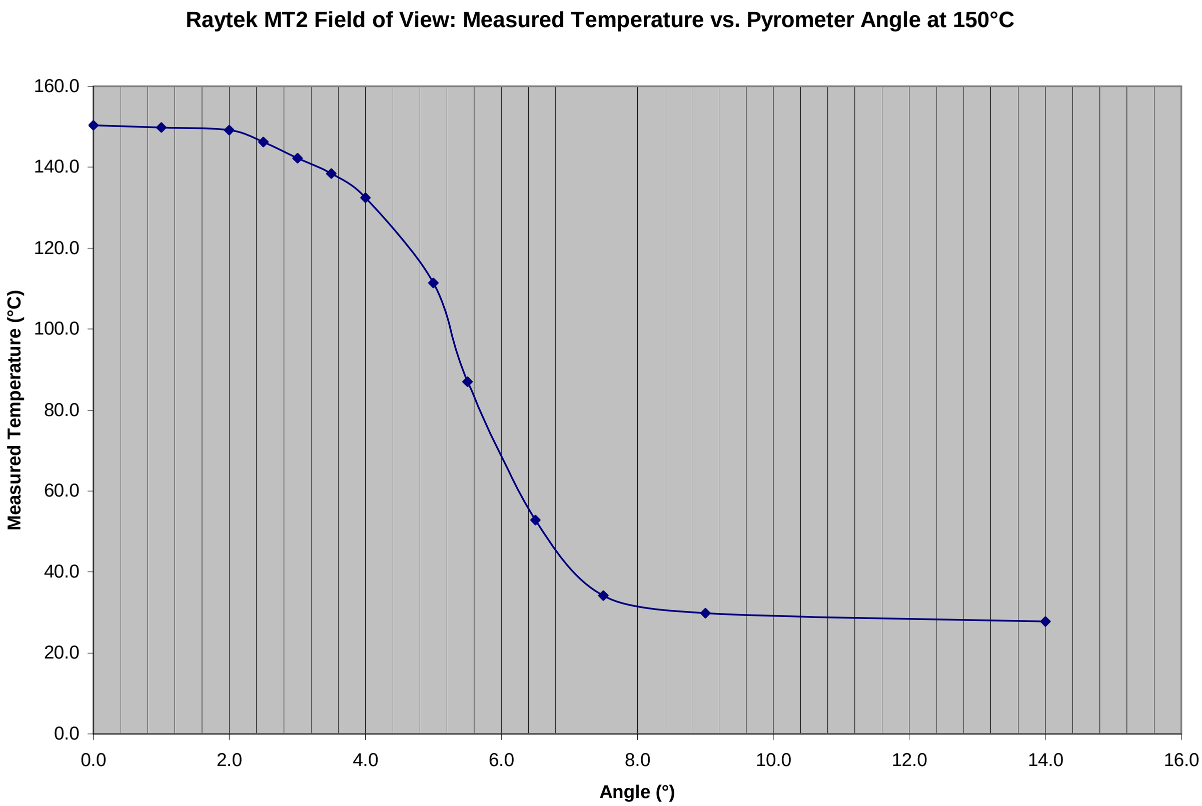
Fluke 62 Field of View: Measured Temperature vs. Pyrometer Angle at 150°C

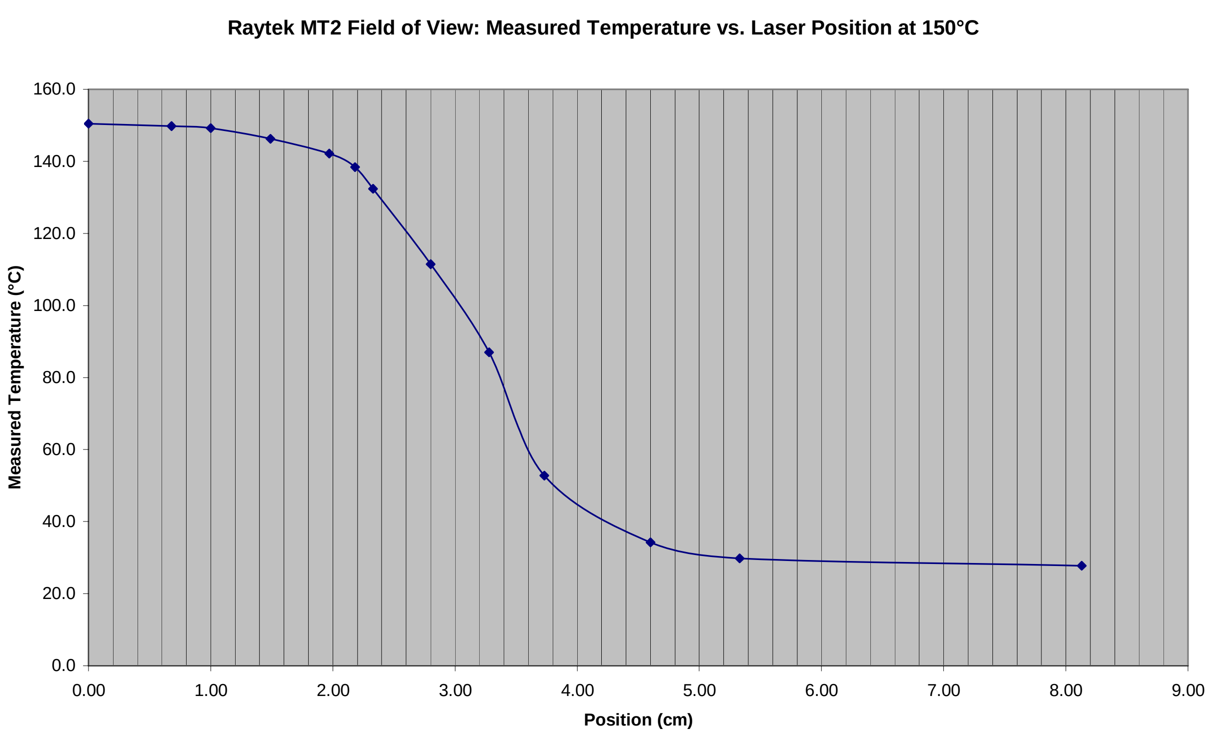


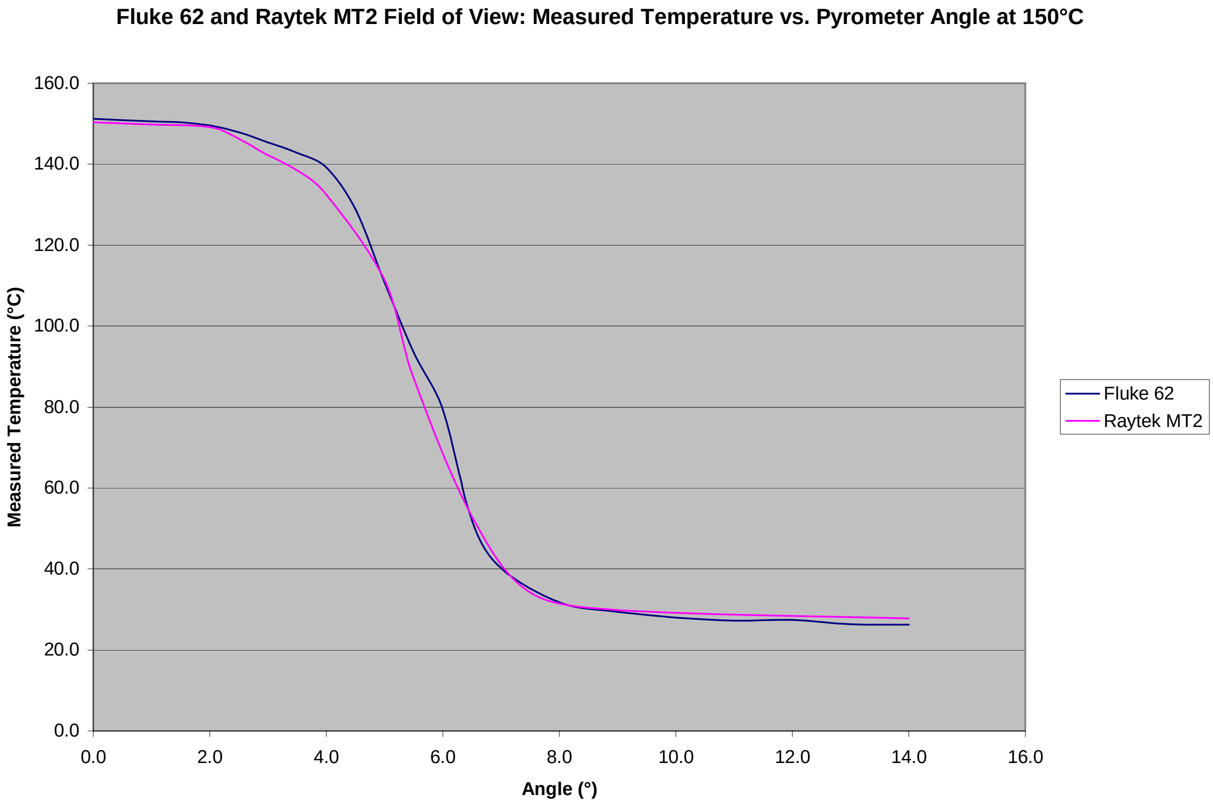


Raytek MT2			
Field of View (Distance to Spot)			
Distance from lens to blackbody: 37.05 cm			Blackbody Temp: 150
Position (cm)	Angle (°)	Temp (°C)	Disp by angle (cm)
8.13	14.0	27.8	9.24
5.33	9.0	29.8	5.87
4.60	7.5	34.2	4.88
3.73	6.5	52.8	4.22
3.28	5.5	87.0	3.57
2.80	5.0	111.4	3.24
2.33	4.0	132.4	2.59
2.18	3.5	138.4	2.27
1.97	3.0	142.2	1.94
1.49	2.5	146.2	1.62
1.00	2.0	149.2	1.29
0.68	1.0	149.8	0.65
0.00	0.0	150.4	0.00

	Position	Angle
D:S estimate	7.96774	8.089519651


Raytek MT2 Field of View: Measured Temperature vs. Pyrometer Angle at 150°C
Measured Temperature (°C)
160.0
140.0
120.0
100.0
80.0
60.0
40.0
20.0
0.0
0.0
2.0
4.0
6.0
8.0
10.0
12.0
14.0
16.0
Angle (°)


Raytek MT2 Field of View: Measured Temperature vs. Laser Position at 150°C
Measured Temperature (°C)
160.0
140.0
120.0
100.0
80.0
60.0
40.0
20.0
0.0
0.00
1.00
2.00
3.00
4.00
5.00
6.00
7.00
8.00
9.00
Position (cm)



A.5 Thermal Shock

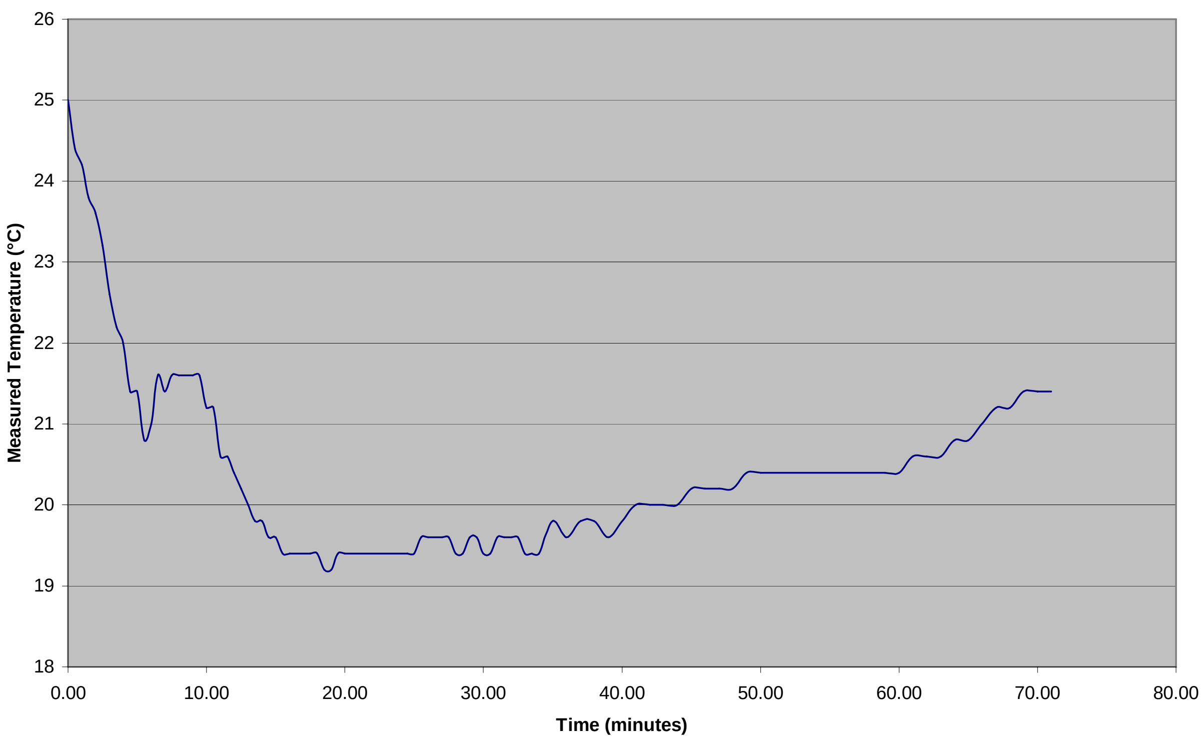
19/8/05	
Fluke 62	
Thermal Shock	
Time (sec)	Temp (°C)
0.00	25
0.50	24.4
1.00	24.2
1.50	23.8
2.00	23.6
2.50	23.2
3.00	22.6
3.50	22.2
4.00	22
4.50	21.4
5.00	21.4
5.50	20.8
6.00	21
6.50	21.6
7.00	21.4
7.50	21.6
8.00	21.6
8.50	21.6
9.00	21.6
9.50	21.6
10.00	21.2
10.50	21.2
11.00	20.6
11.50	20.6
12.00	20.4
12.50	20.2
13.00	20
13.50	19.8
14.00	19.8
14.50	19.6
15.00	19.6
15.50	19.4
16.00	19.4
16.50	19.4
17.00	19.4
17.50	19.4
18.00	19.4
18.50	19.2
19.00	19.2
19.50	19.4
20.00	19.4
20.50	19.4
21.00	19.4
21.50	19.4
22.00	19.4
22.50	19.4
23.00	19.4
23.50	19.4
24.00	19.4
24.50	19.4
25.00	19.4
25.50	19.6
26.00	19.6
26.50	19.6
27.00	19.6
27.50	19.6
28.00	19.4
28.50	19.4

29.00	19.6
29.50	19.6
30.00	19.4
30.50	19.4
31.00	19.6
31.50	19.6
32.00	19.6
32.50	19.6
33.00	19.4
33.50	19.4
34	19.4
35	19.8
36	19.6
37	19.8
38	19.8
39	19.6
40	19.8

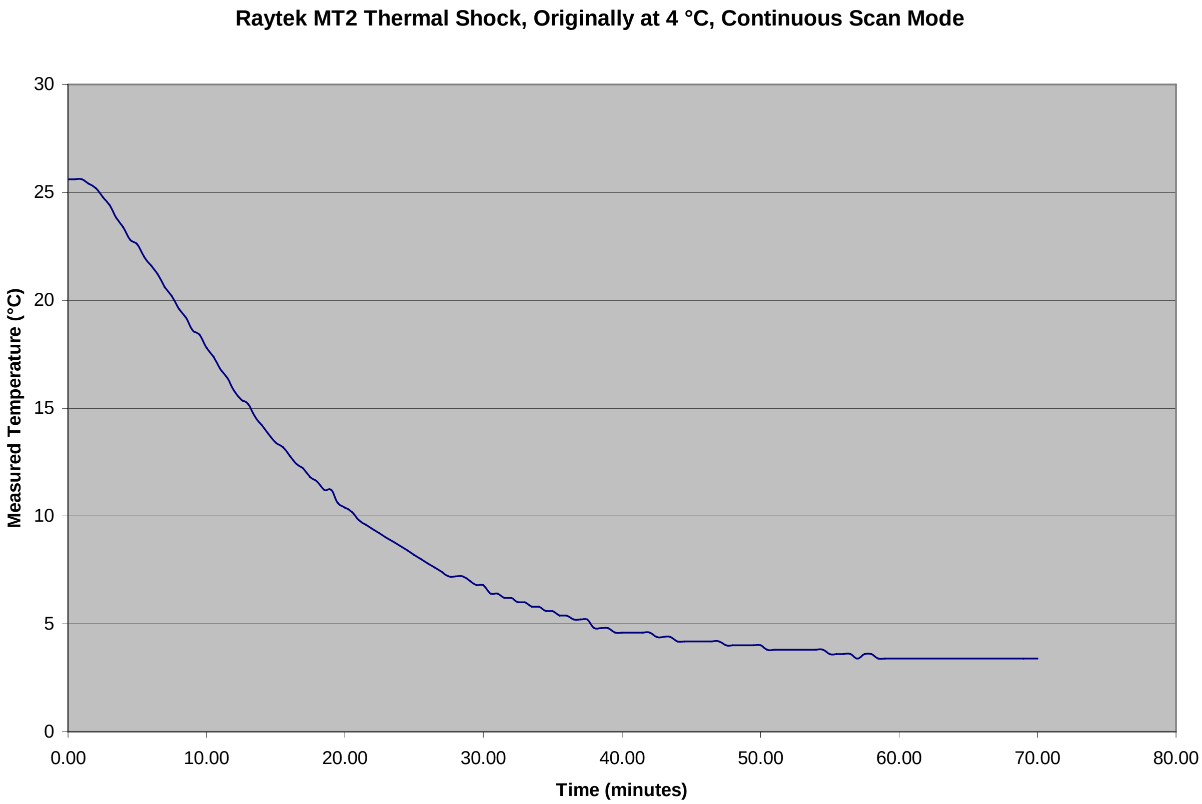
41	20
42	20
43	20
44	20
45	20.2
46	20.2
47	20.2
48	20.2
49	20.4
50	20.4
51	20.4
52	20.4
53	20.4
54	20.4
55	20.4
56	20.4
57	20.4

58	20.4
59	20.4
60	20.4
61	20.6
62	20.6
63	20.6
64	20.8
65	20.8
66	21
67	21.2
68	21.2
69	21.4
70	21.4
71	21.4

Fluke 62 Thermal Shock, Originally at -20°C

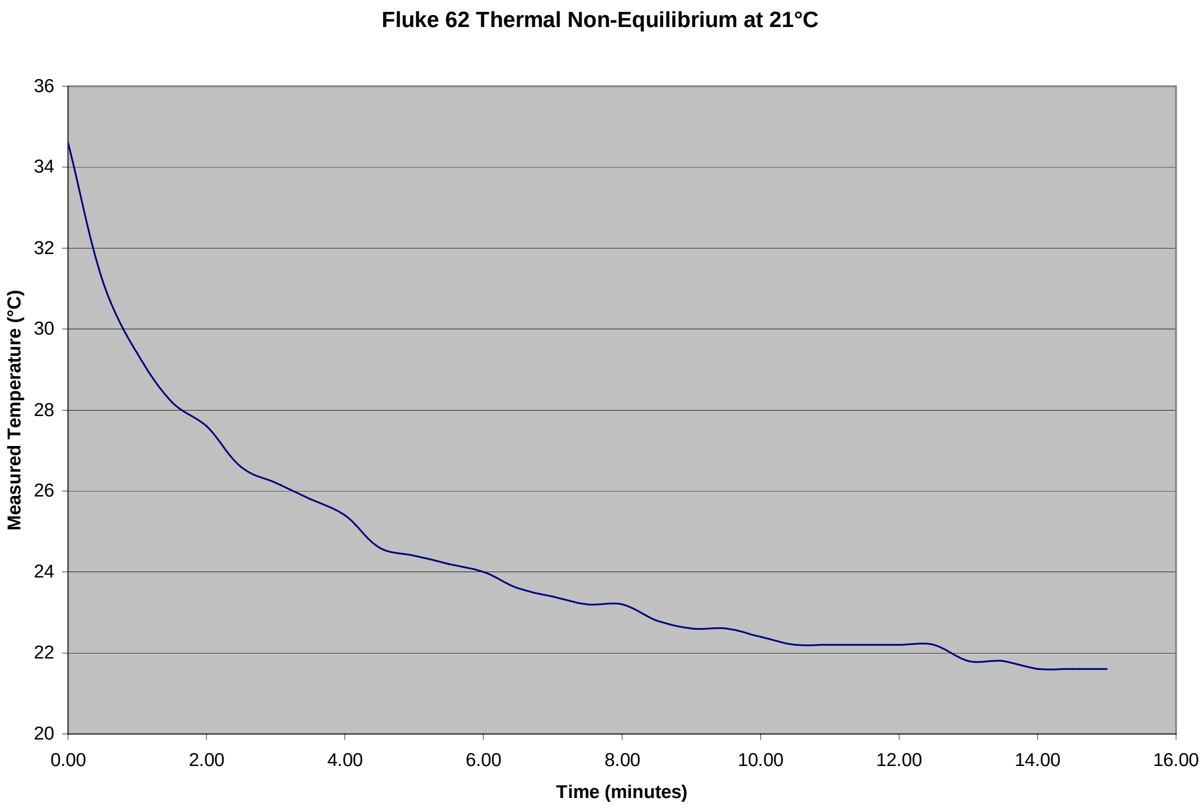


19/8/05	
Raytek MT2	
Thermal Shock	
Time (sec)	Temp (°C)
0.00	25.6
0.50	25.6
1.00	25.6
1.50	25.4
2.00	25.2
2.50	24.8
3.00	24.4
3.50	23.8
4.00	23.4
4.50	22.8
5.00	22.6
5.50	22
6.00	21.6
6.50	21.2
7.00	20.6
7.50	20.2
8.00	19.6
8.50	19.2
9.00	18.6
9.50	18.4
10.00	17.8
10.50	17.4
11.00	16.8
11.50	16.4
12.00	15.8
12.50	15.4
13.00	15.2
13.50	14.6
14.00	14.2
14.50	13.8
15.00	13.4
15.50	13.2
16.00	12.8
16.50	12.4
17.00	12.2
17.50	11.8
18.00	11.6
18.50	11.2
19.00	11.2
19.50	10.6
20.00	10.4
20.50	10.2
21.00	9.8
21.50	9.6
22.00	9.4
22.50	9.2
23.00	9
23.50	8.8
24.00	8.6
24.50	8.4
25.00	8.2
25.50	8
26.00	7.8
26.50	7.6
27.00	7.4
27.50	7.2
28.00	7.2
28.50	7.2
29.00	7
29.50	6.8
30.00	6.8
30.50	6.4
31.00	6.4
31.50	6.2
32.00	6.2
32.50	6
33.00	6
33.50	5.8
34.00	5.8
34.50	5.6
35.00	5.6
35.50	5.4
36.00	5.4
36.50	5.2
37.00	5.2
37.50	5.2
38.00	4.8
38.50	4.8
39.00	4.8
39.50	4.6
40.00	4.6
40.50	4.6
41.00	4.6
41.50	4.6
42.00	4.6
42.50	4.4
43.00	4.4
43.50	4.4
44.00	4.2
44.50	4.2
45.00	4.2
45.50	4.2
46.00	4.2
46.50	4.2
47.00	4.2
47.50	4
48.00	4
48.50	4
49.00	4
49.50	4
50.00	4
50.50	3.8
51.00	3.8
51.50	3.8
52.00	3.8
52.50	3.8
53.00	3.8
53.50	3.8
54.00	3.8
54.50	3.8
55.00	3.6
55.50	3.6
56.00	3.6
56.50	3.6
57.00	3.4
57.50	3.6
58.00	3.6
58.50	3.4
59.00	3.4
59.50	3.4
60.00	3.4
60.50	3.4
61.00	3.4
61.50	3.4
62.00	3.4
62.50	3.4
63.00	3.4
63.50	3.4
64.00	3.4
64.50	3.4
65.00	3.4
65.50	3.4
66.00	3.4
67.00	3.4
68.00	3.4
69.00	3.4
70.00	3.4

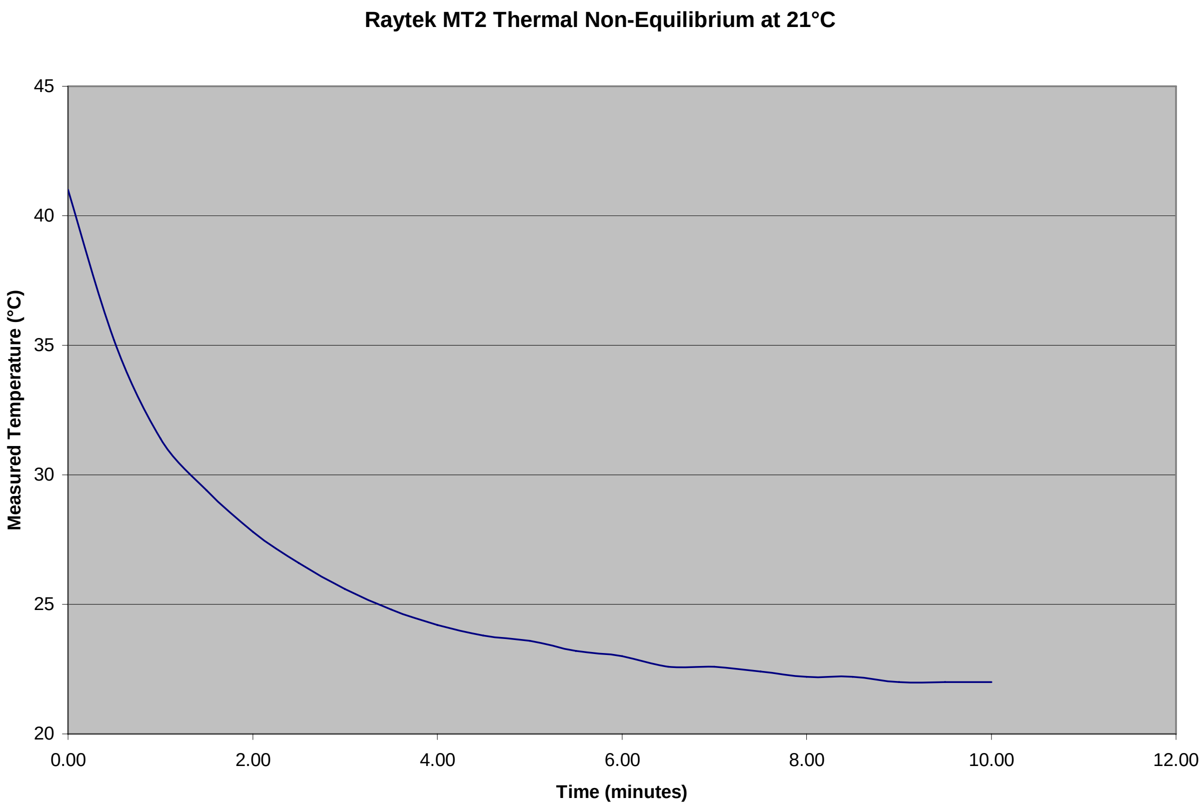


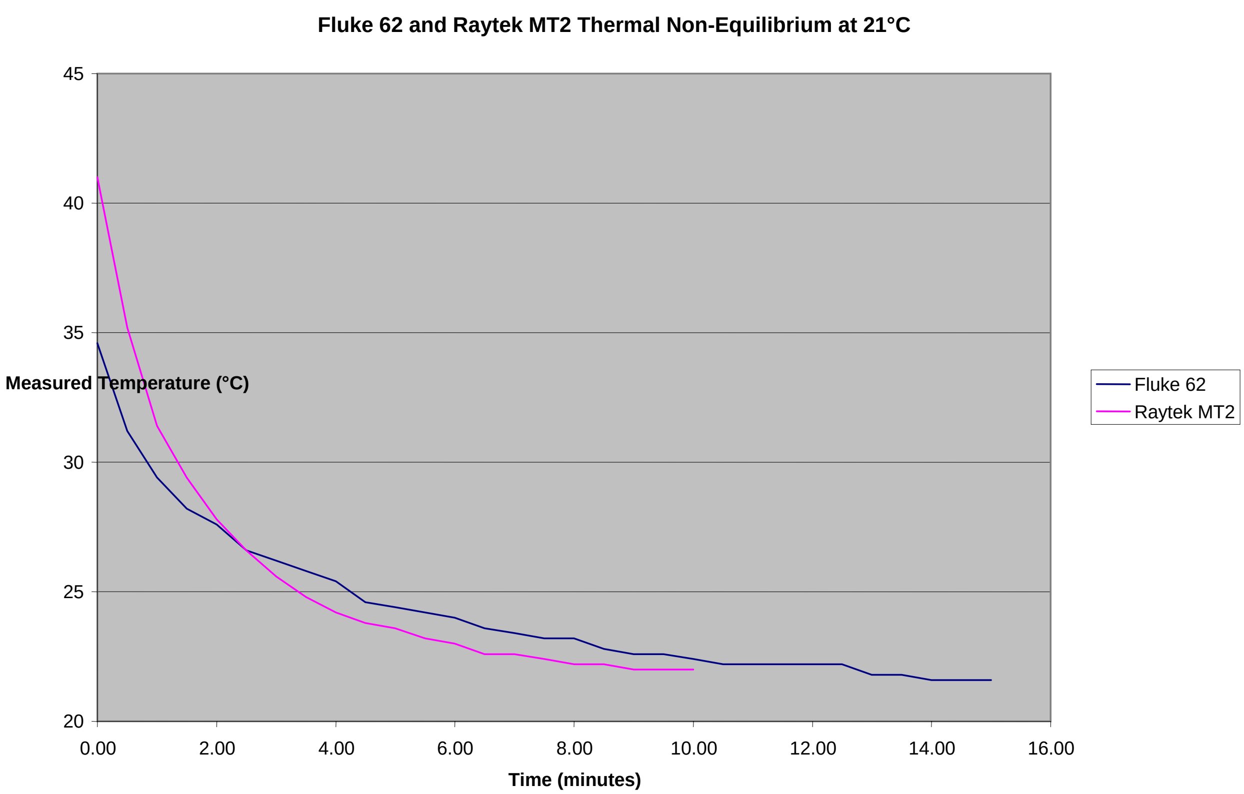
A.6 Thermal Non-Equilibrium

23/8/05	
Fluke 62	
Thermal Equilibrium	
Time (sec)	Temp (°C)
0.00	34.6
0.50	31.2
1.00	29.4
1.50	28.2
2.00	27.6
2.50	26.6
3.00	26.2
3.50	25.8
4.00	25.4
4.50	24.6
5.00	24.4
5.50	24.2
6.00	24
6.50	23.6
7.00	23.4
7.50	23.2
8.00	23.2
8.50	22.8
9.00	22.6
9.50	22.6
10.00	22.4
10.50	22.2
11.00	22.2
11.50	22.2
12.00	22.2
12.50	22.2
13.00	21.8
13.50	21.8
14.00	21.6
14.50	21.6
15.00	21.6



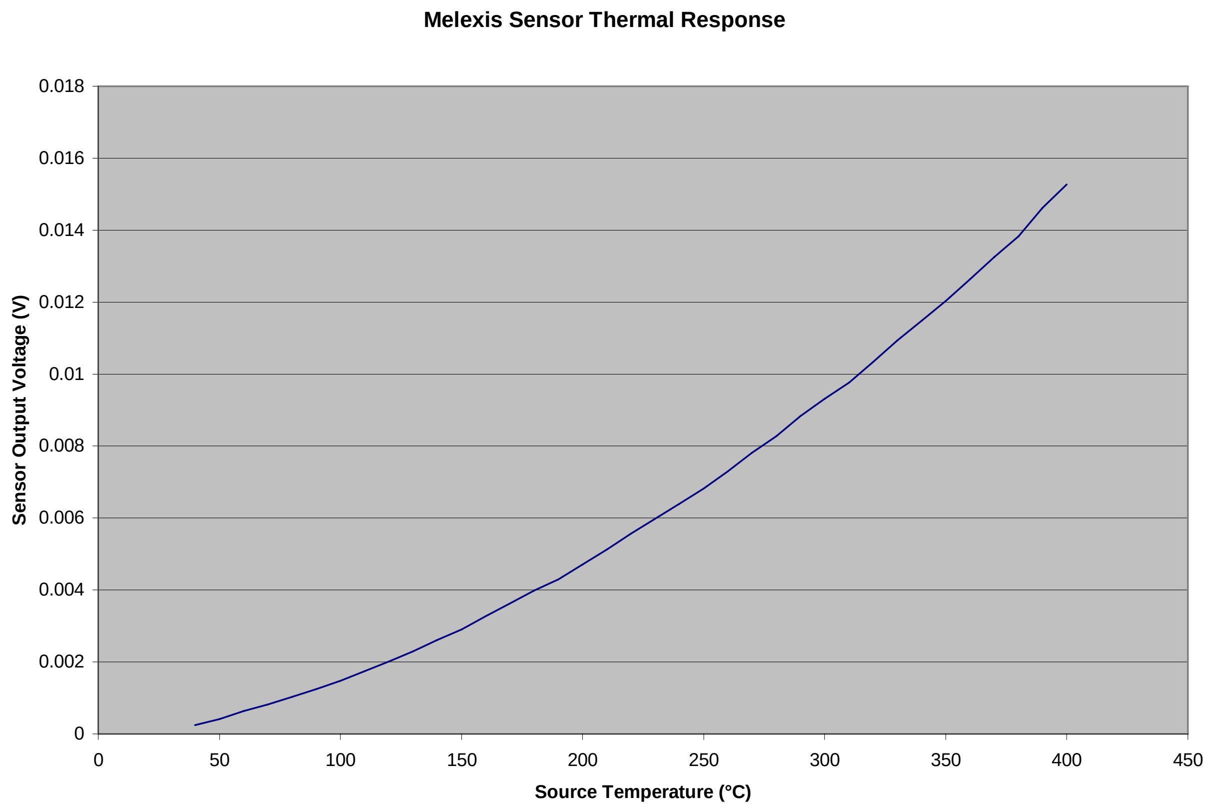
23/8/05	
Raytek MT2	
Thermal Equilibrium	
Time (sec)	Temp (°C)
0.00	41
0.50	35.2
1.00	31.4
1.50	29.4
2.00	27.8
2.50	26.6
3.00	25.6
3.50	24.8
4.00	24.2
4.50	23.8
5.00	23.6
5.50	23.2
6.00	23
6.50	22.6
7.00	22.6
7.50	22.4
8.00	22.2
8.50	22.2
9.00	22
9.50	22
10.00	22

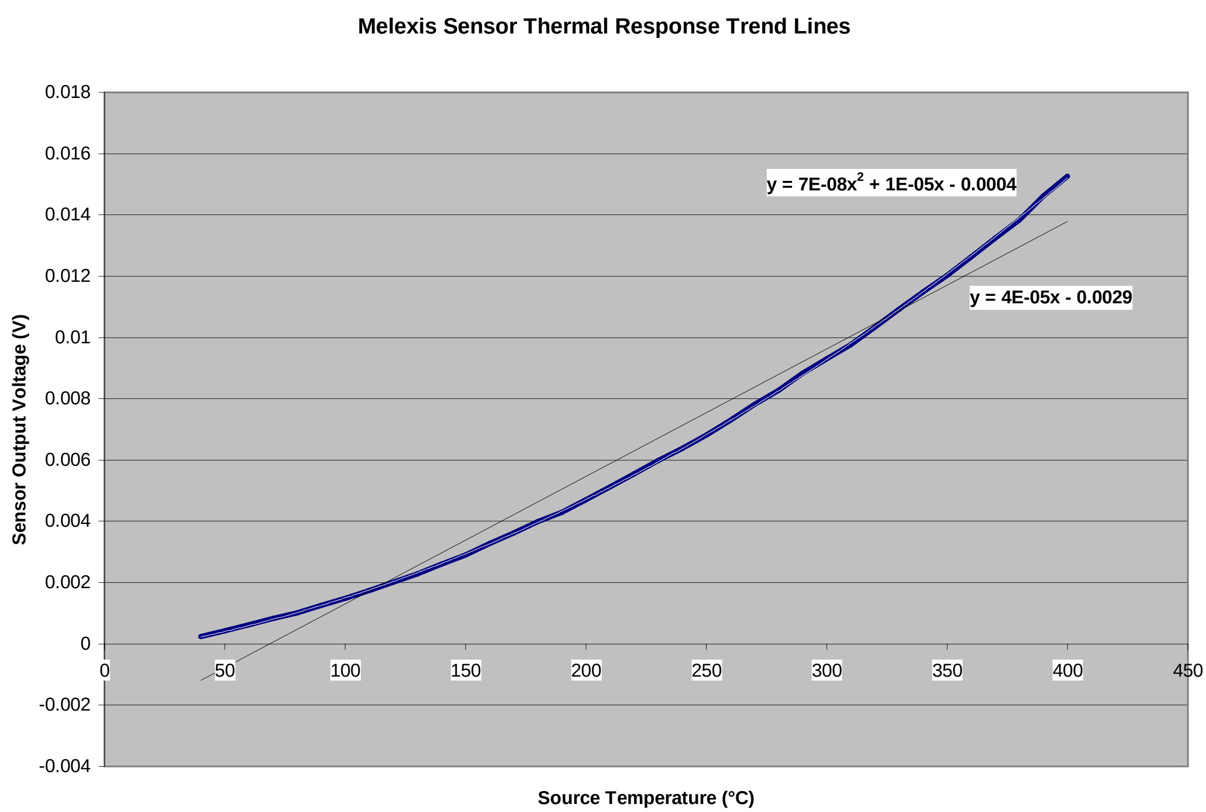

Raytek MT2 Thermal Non-Equilibrium at 21°C
Measured Temperature (°C)
45
40
35
30
25
20
0.00
2.00
4.00
6.00
8.00
10.00
12.00
Time (minutes)


Fluke 62 and Raytek MT2 Thermal Non-Equilibrium at 21°C
Measured Temperature (°C)
45
40
35
30
25
20
0.00
2.00
4.00
6.00
8.00
10.00
12.00
14.00
16.00
Time (minutes)
Fluke 62
Raytek MT2

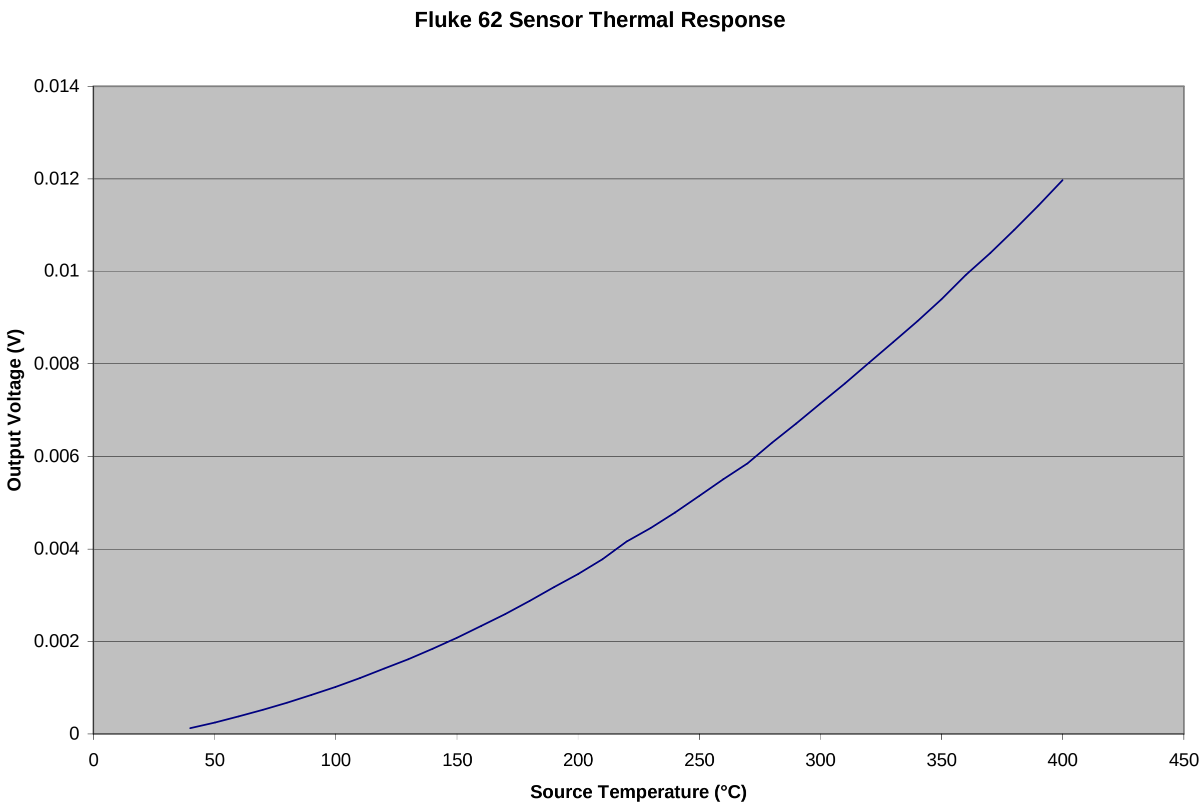
A.7 Sensor Thermal Response

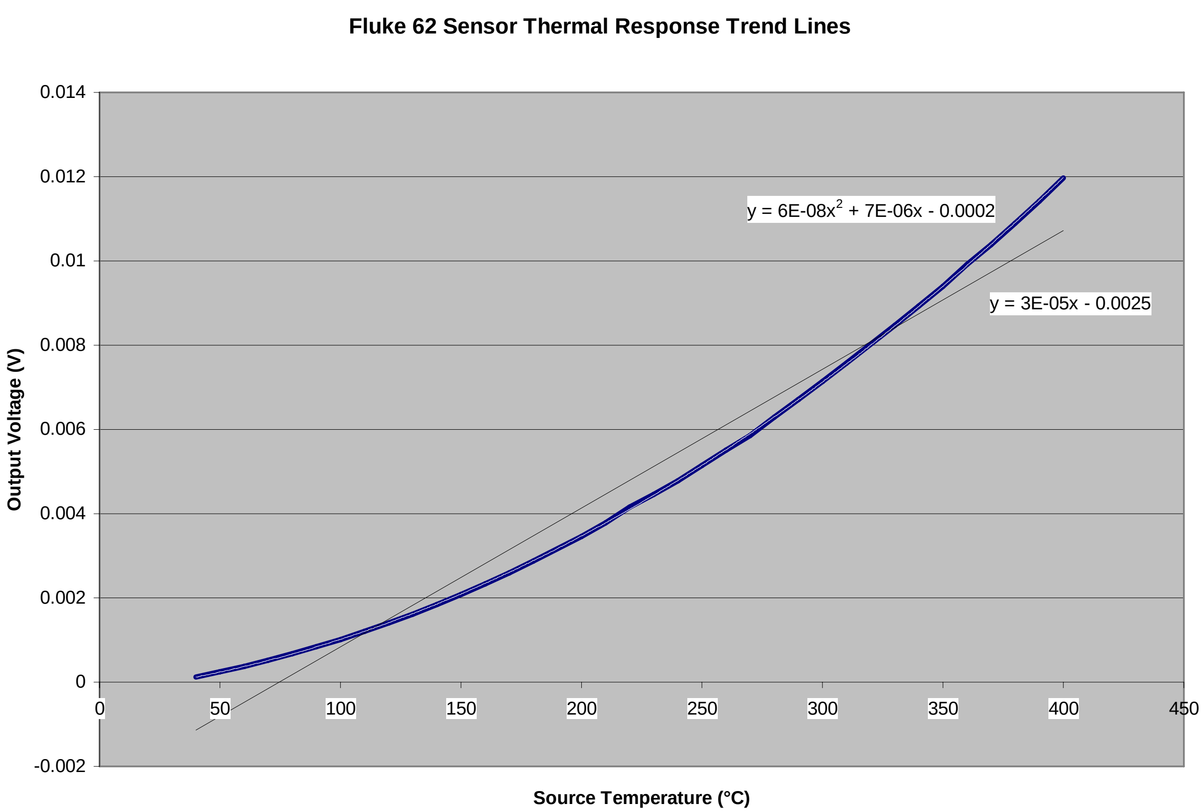
31/8/05		
Melexis 90247		Offset = 0.9 Volts
Thermal Response		D= 6.2 cm to case
Temp (°C)	Recorded Voltage (V)	Actual Voltage (V)
40	0.90024	0.00024
50	0.90042	0.00042
60	0.90063	0.00063
70	0.90082	0.00082
80	0.90102	0.00102
90	0.90125	0.00125
100	0.90148	0.00148
110	0.90174	0.00174
120	0.90201	0.00201
130	0.90229	0.00229
140	0.90261	0.00261
150	0.9029	0.0029
160	0.90327	0.00327
170	0.90362	0.00362
180	0.90399	0.00399
190	0.9043	0.0043
200	0.90471	0.00471
210	0.90512	0.00512
220	0.90556	0.00556
230	0.90598	0.00598
240	0.90639	0.00639
250	0.90682	0.00682
260	0.9073	0.0073
270	0.90781	0.00781
280	0.90827	0.00827
290	0.90884	0.00884
300	0.90931	0.00931
310	0.90976	0.00976
320	0.91034	0.01034
330	0.91093	0.01093
340	0.91148	0.01148
350	0.91203	0.01203
360	0.91263	0.01263
370	0.91325	0.01325
380	0.91383	0.01383
390	0.91462	0.01462
400	0.91527	0.01527

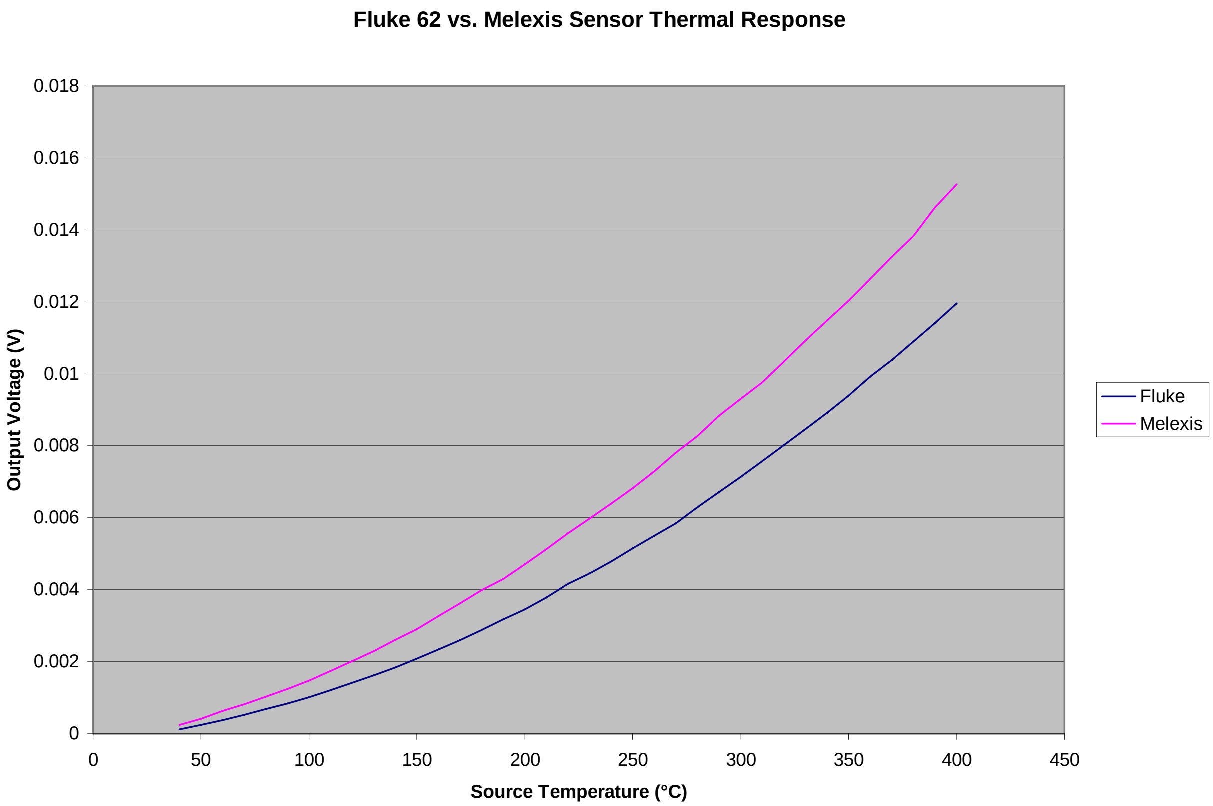

Melexis Sensor Thermal Response
Sensor Output Voltage (V)
0.018
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
0
50
100
150
200
250
300
350
400
450
Source Temperature (°C)


Melexis Sensor Thermal Response Trend Lines
Sensor Output Voltage (V)
y = 7E-08x2 + 1E-05x - 0.0004
y = 4E-05x - 0.0029
0.018
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
-0.002
-0.004
0
50
100
150
200
250
300
350
400
450
Source Temperature (°C)

31/8/05		
Fluke Sensor		Offset = 0.9 Volts
Thermal Response		D= 6.2 cm to case
Temp (°C)	Recorded Voltage (V)	Actual Voltage (V)
40	0.900124	0.000124
50	0.900247	0.000247
60	0.90038	0.00038
70	0.900524	0.000524
80	0.900678	0.000678
90	0.900842	0.000842
100	0.901015	0.001015
110	0.901208	0.001208
120	0.90141	0.00141
130	0.901618	0.001618
140	0.901842	0.001842
150	0.902083	0.002083
160	0.902338	0.002338
170	0.902595	0.002595
180	0.902875	0.002875
190	0.90317	0.00317
200	0.903458	0.003458
210	0.903778	0.003778
220	0.90416	0.00416
230	0.904455	0.004455
240	0.904785	0.004785
250	0.905145	0.005145
260	0.905507	0.005507
270	0.90585	0.00585
280	0.906293	0.006293
290	0.906713	0.006713
300	0.90714	0.00714
310	0.907573	0.007573
320	0.90802	0.00802
330	0.90847	0.00847
340	0.908925	0.008925
350	0.9094	0.0094
360	0.90992	0.00992
370	0.91039	0.01039
380	0.910895	0.010895
390	0.911415	0.011415
400	0.911965	0.011965

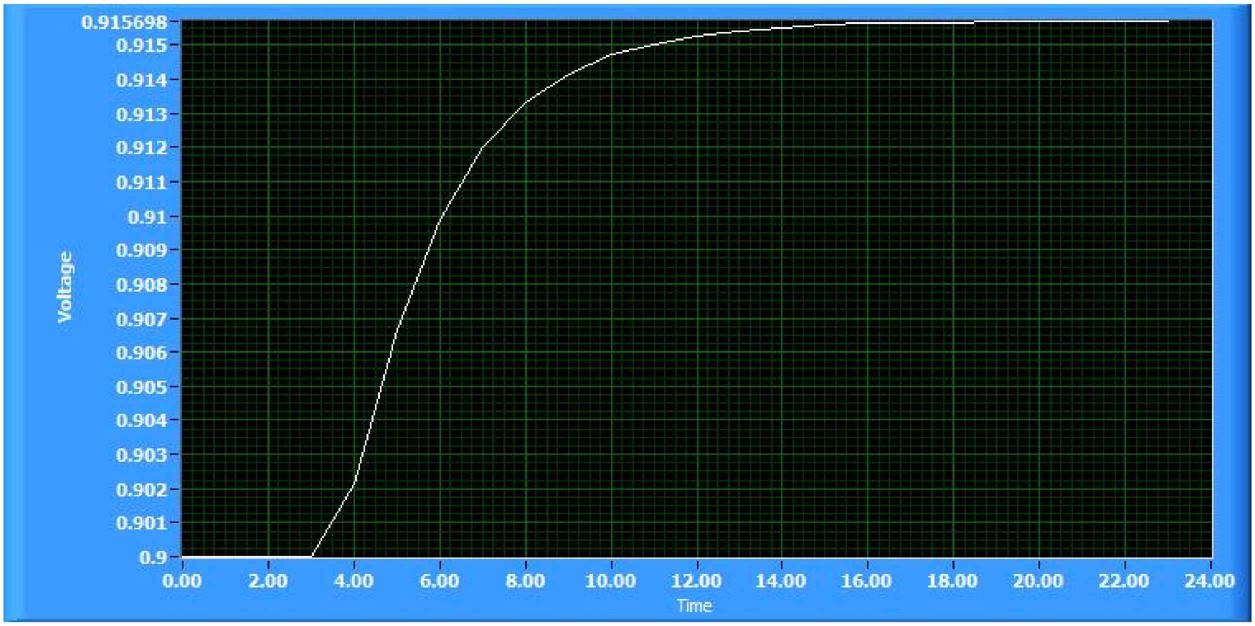

Fluke 62 Sensor Thermal Response
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
Output Voltage (V)
0
50
100
150
200
250
300
350
400
450
Source Temperature (°C)

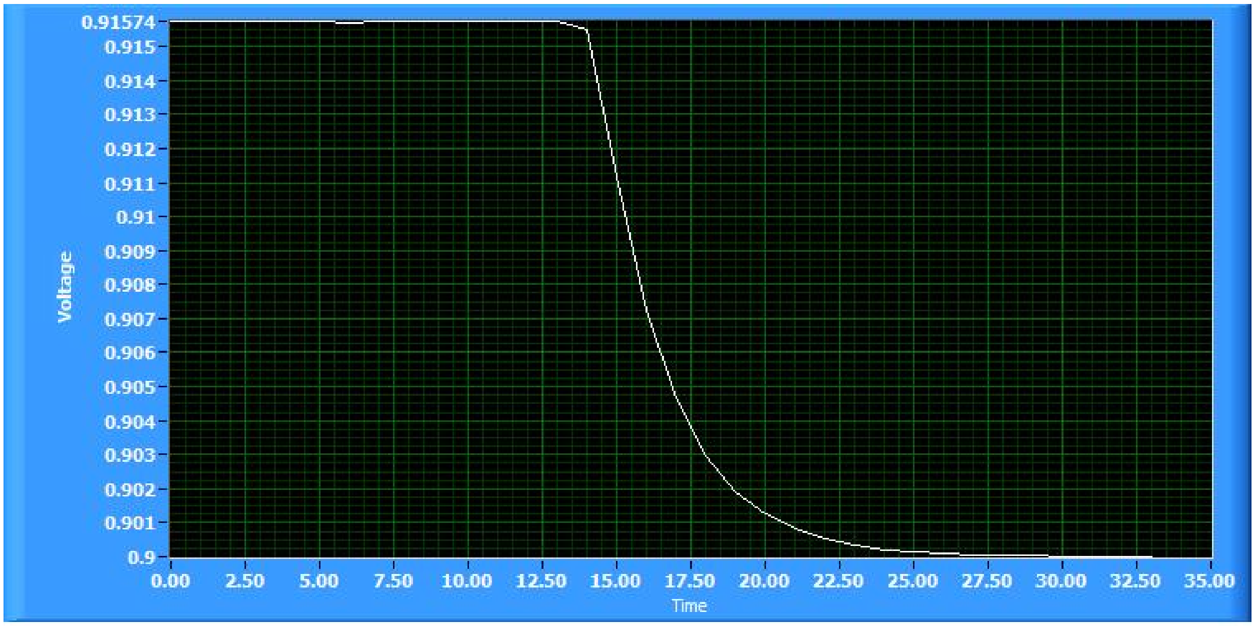

Fluke 62 Sensor Thermal Response Trend Lines
y = 6E-08x2 + 7E-06x - 0.0002
y = 3E-05x - 0.0025
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
-0.002
Output Voltage (V)
0
50
100
150
200
250
300
350
400
450
Source Temperature (°C)


Fluke 62 vs. Melexis Sensor Thermal Response
Output Voltage (V)
0.018
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
0
50
100
150
200
250
300
350
400
450
Source Temperature (°C)
Fluke
Melexis

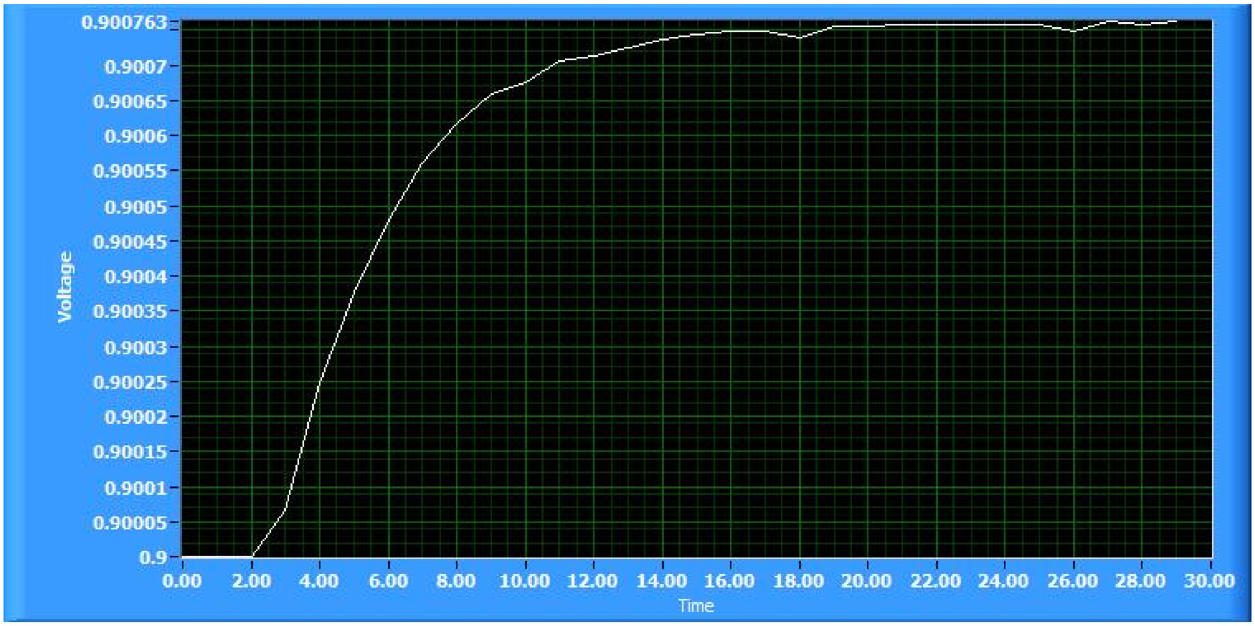
A.8 Time Constant Tests

No Sensor

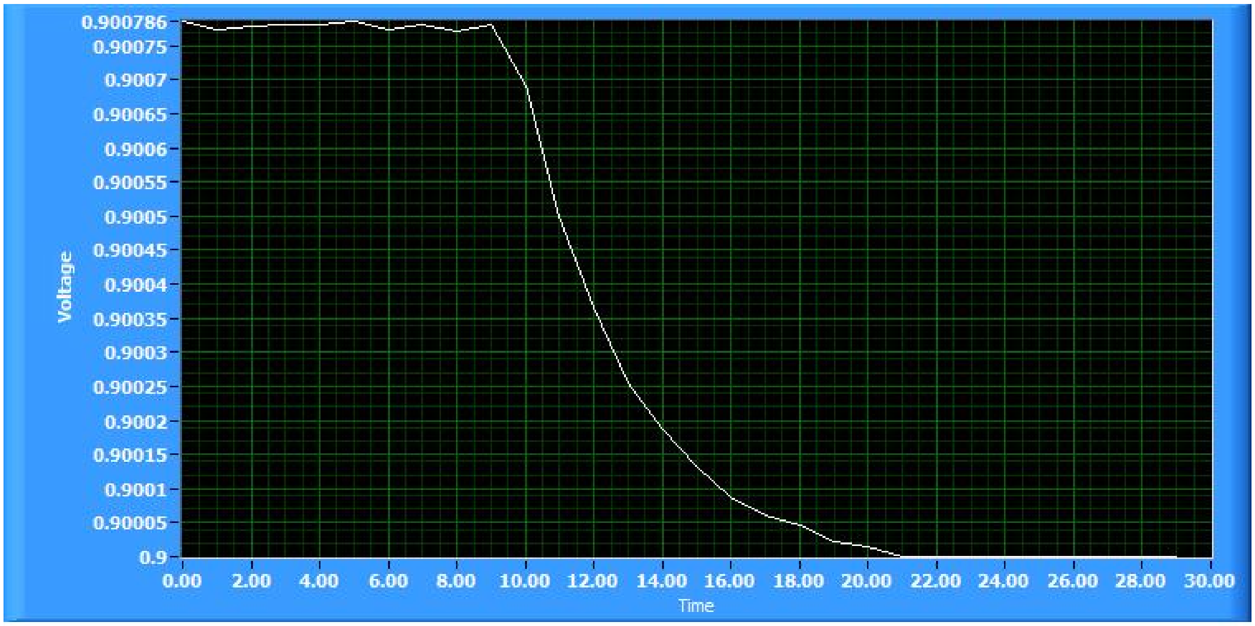




Fluke 100°C

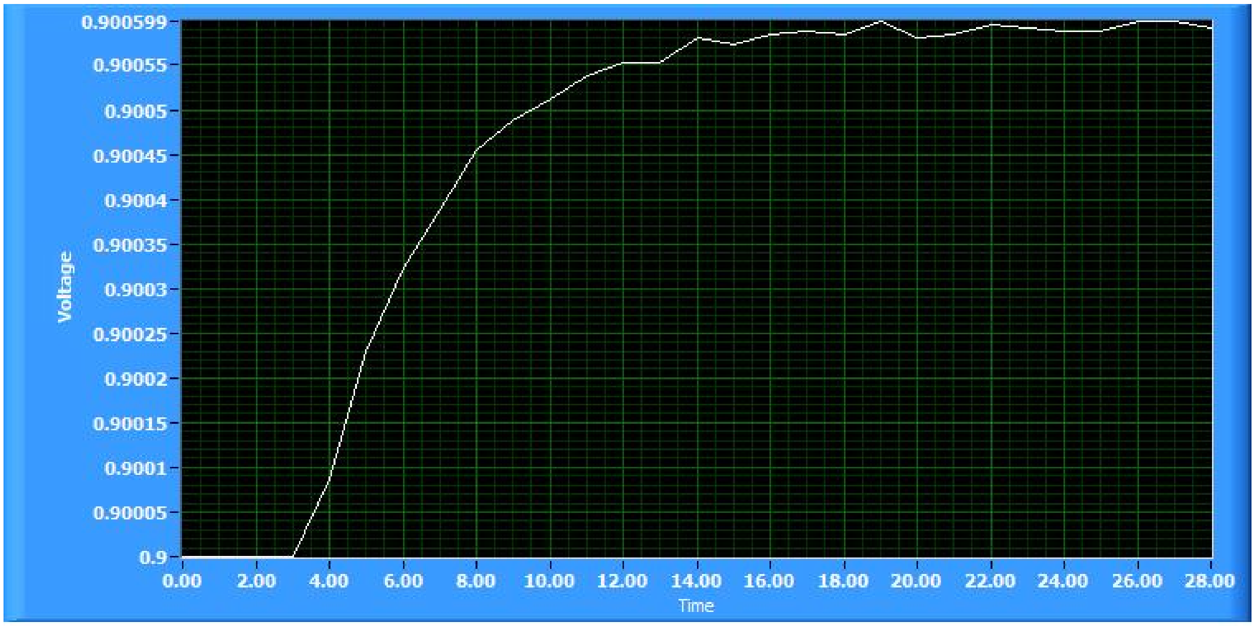


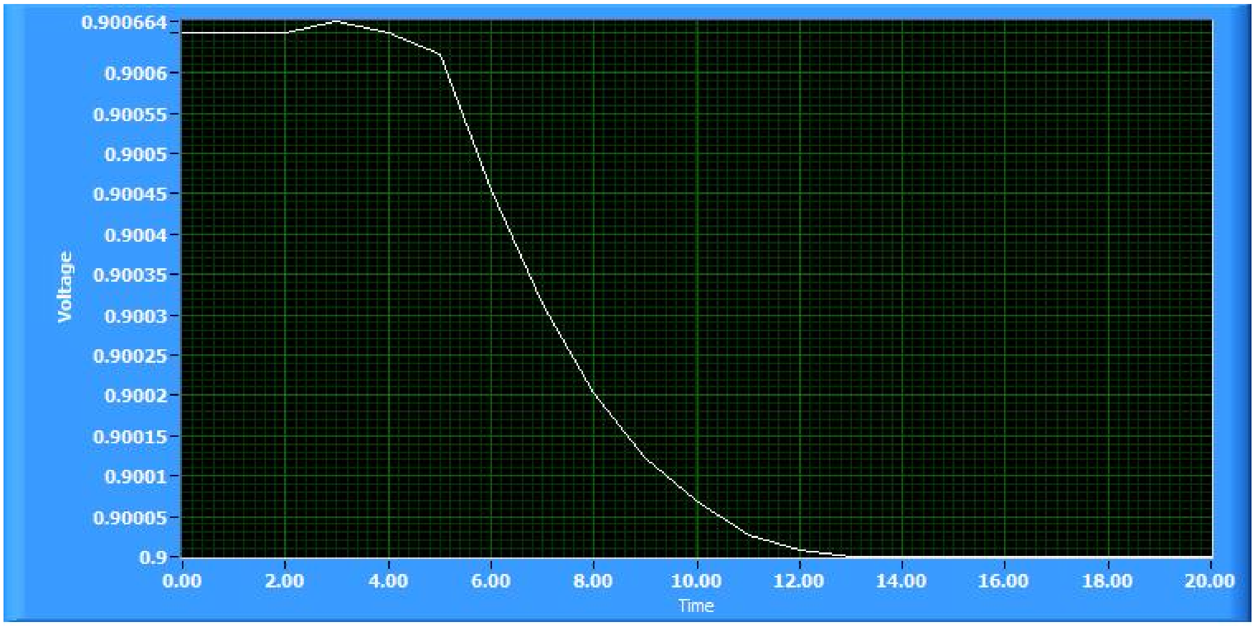
0.900763
0.9007
0.90065
0.9006
0.90055
0.9005
0.90045
0.9004
0.90035
0.9003
0.90025
0.9002
0.90015
0.9001
0.90005
0.9
Voltage
0.00
2.00
4.00
6.00
8.00
10.00
12.00
14.00
16.00
18.00
20.00
22.00
24.00
26.00
28.00
30.00
Time



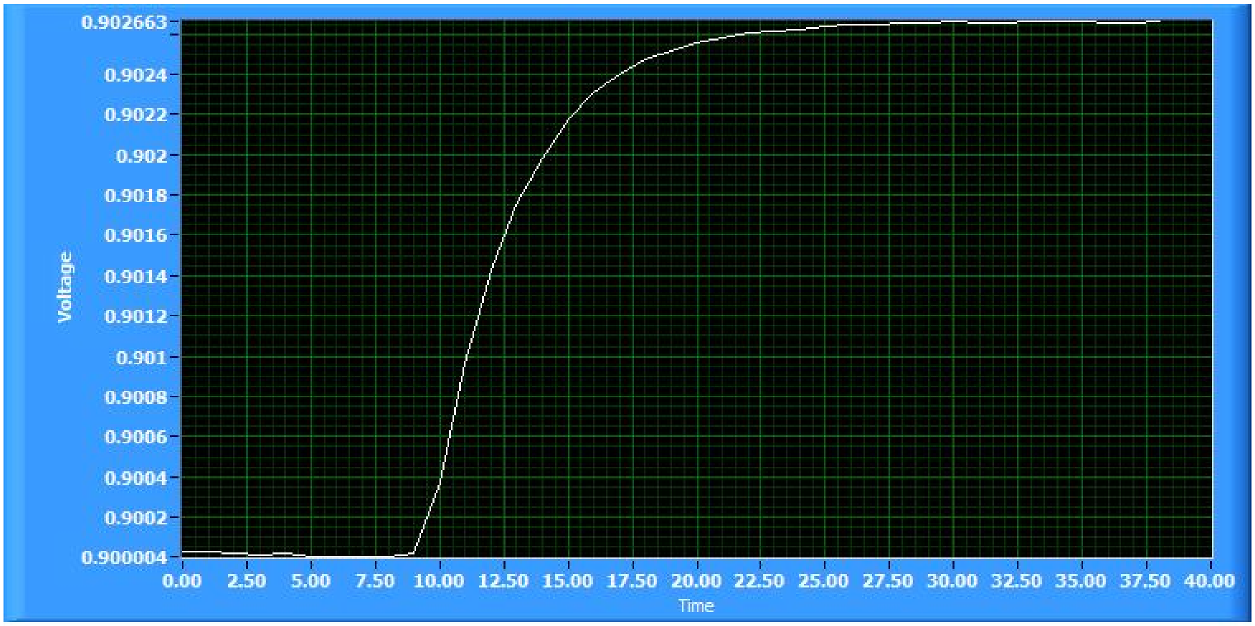
0.900786
0.90075
0.9007
0.90065
0.9006
0.90055
0.9005
0.90045
0.9004
0.90035
0.9003
0.90025
0.9002
0.90015
0.9001
0.90005
0.9
Voltage
0.00
2.00
4.00
6.00
8.00
10.00
12.00
14.00
16.00
18.00
20.00
22.00
24.00
26.00
28.00
30.00
Time

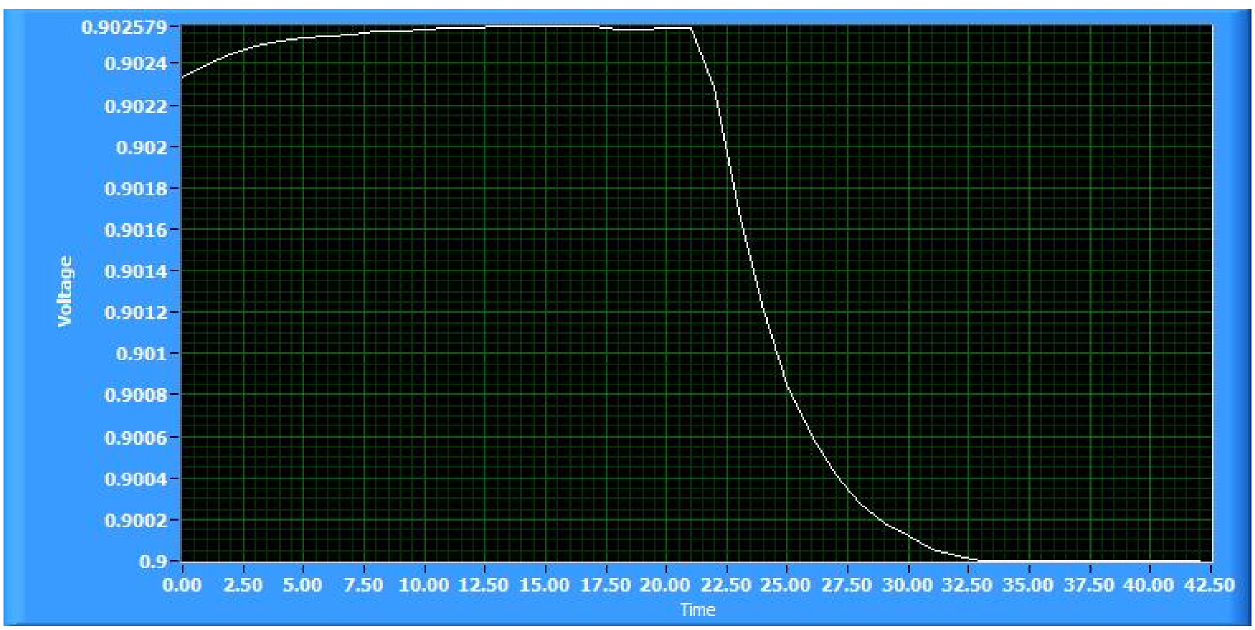
Melexis 100°C



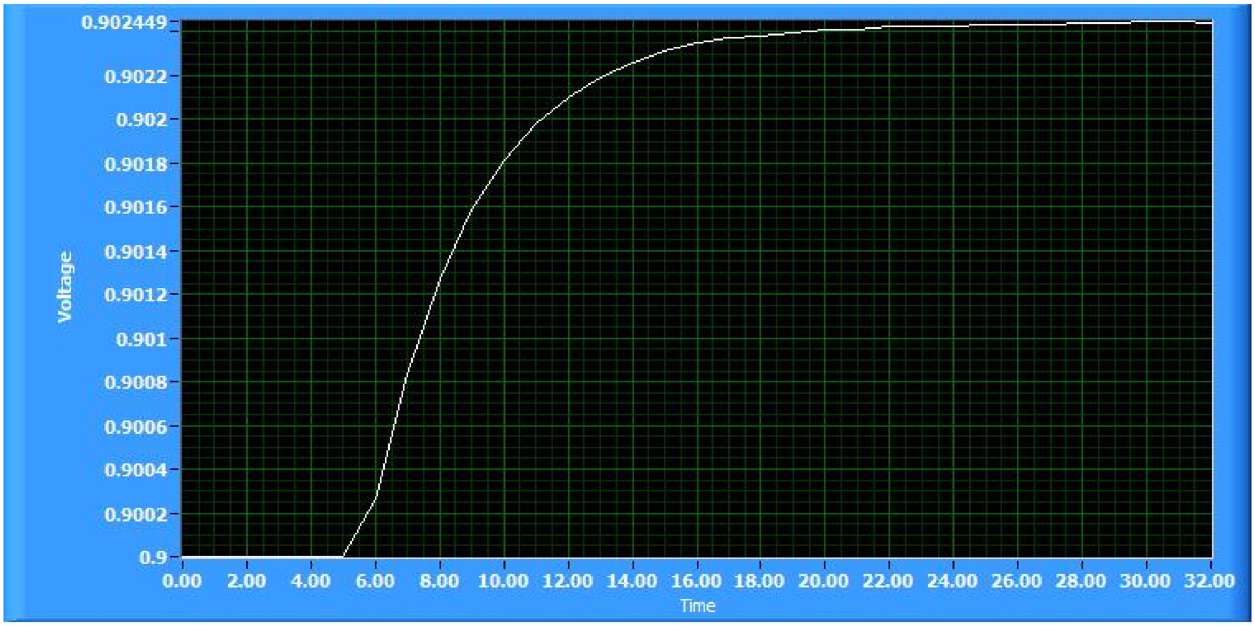


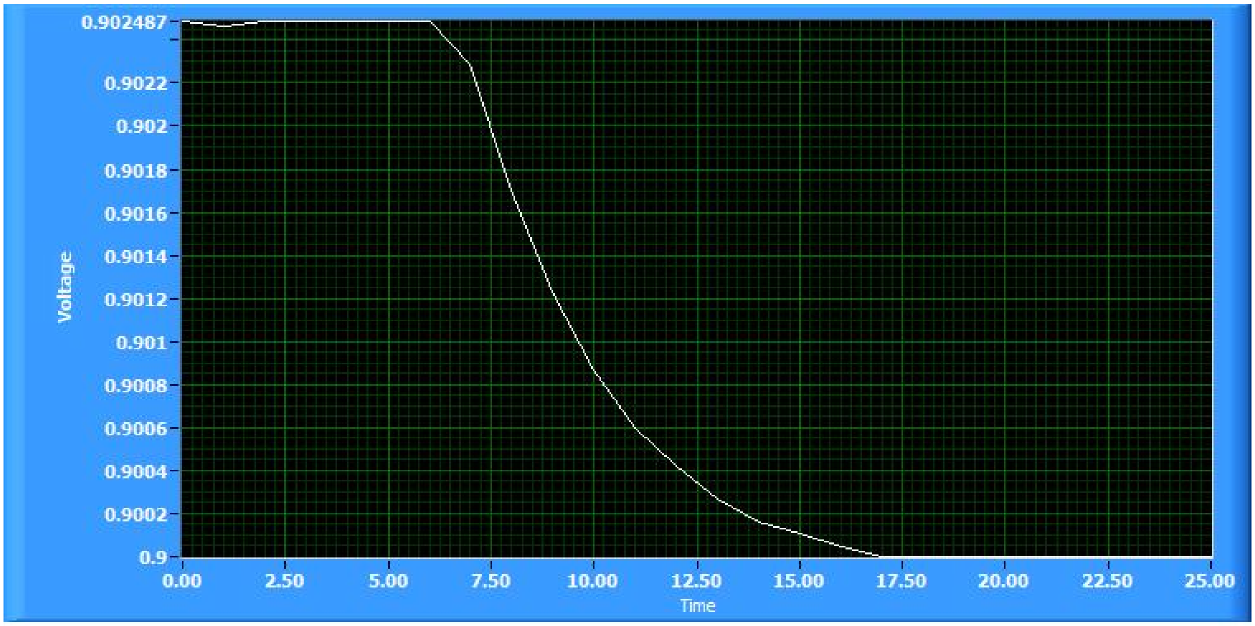
Fluke 200°C


0.902663
0.9024
0.9022
0.902
0.9018
0.9016
0.9014
0.9012
0.901
0.9008
0.9006
0.9004
0.9002
0.900004
Voltage
0.00 2.50 5.00 7.50 10.00 12.50 15.00 17.50 20.00 22.50 25.00 27.50 30.00 32.50 35.00 37.50 40.00
Time

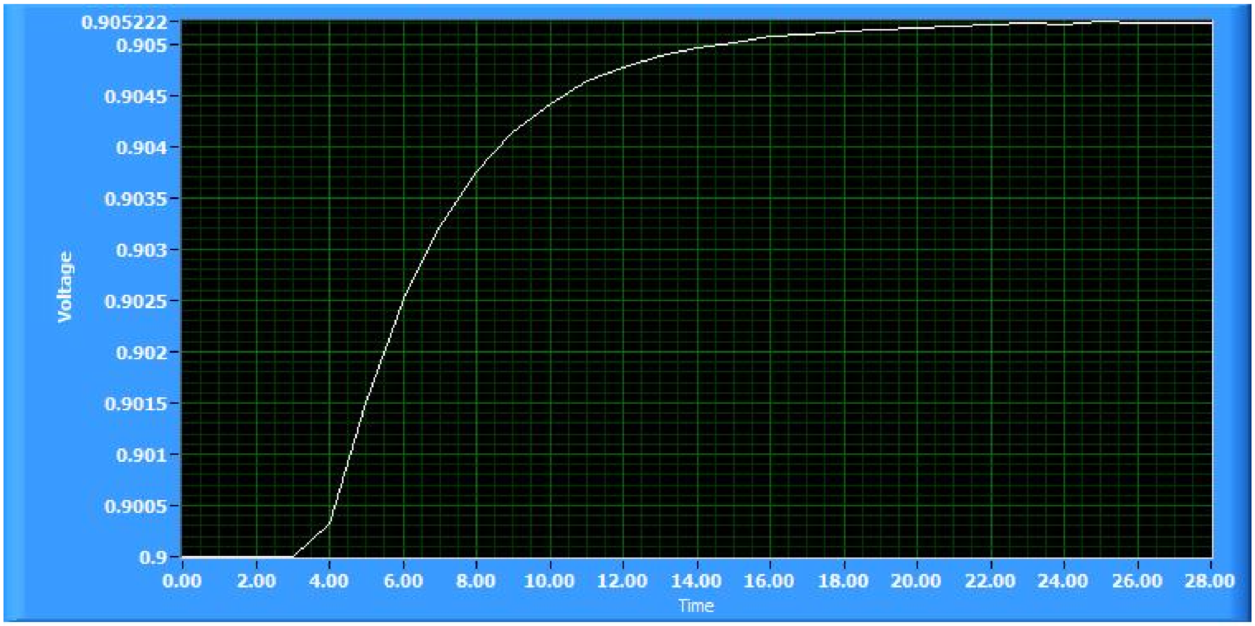

0.902579
0.9024
0.9022
0.902
0.9018
0.9016
0.9014
0.9012
0.901
0.9008
0.9006
0.9004
0.9002
0.9
Voltage
0.00 2.50 5.00 7.50 10.00 12.50 15.00 17.50 20.00 22.50 25.00 27.50 30.00 32.50 35.00 37.50 40.00 42.50
Time

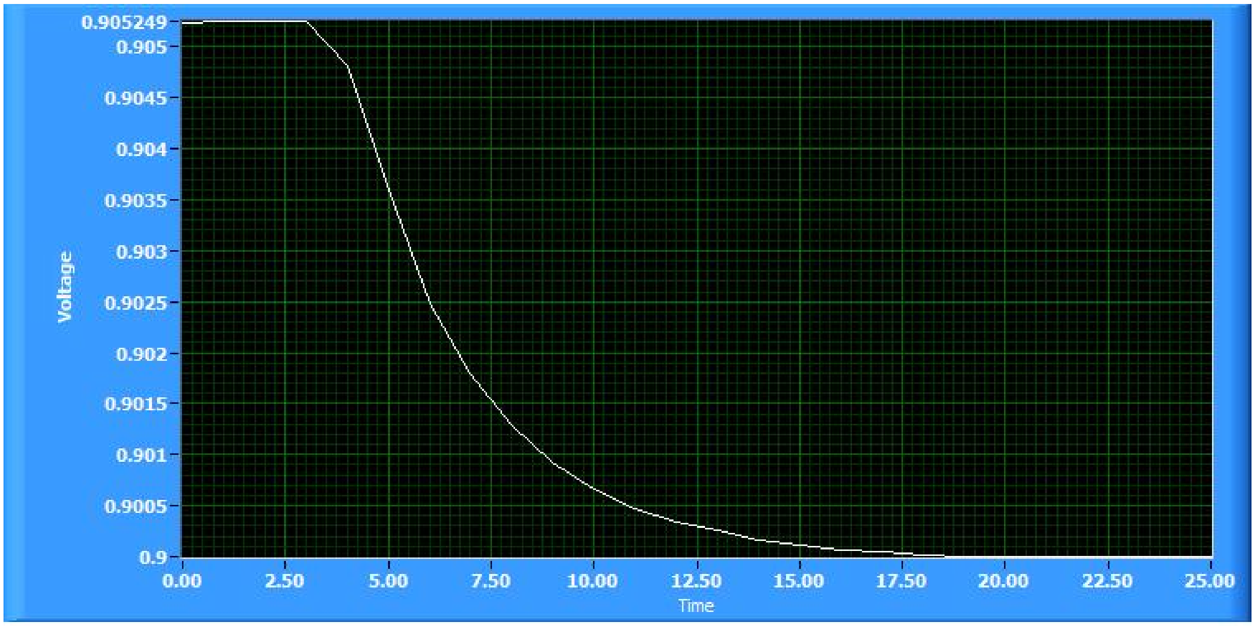
Melexis 200°C



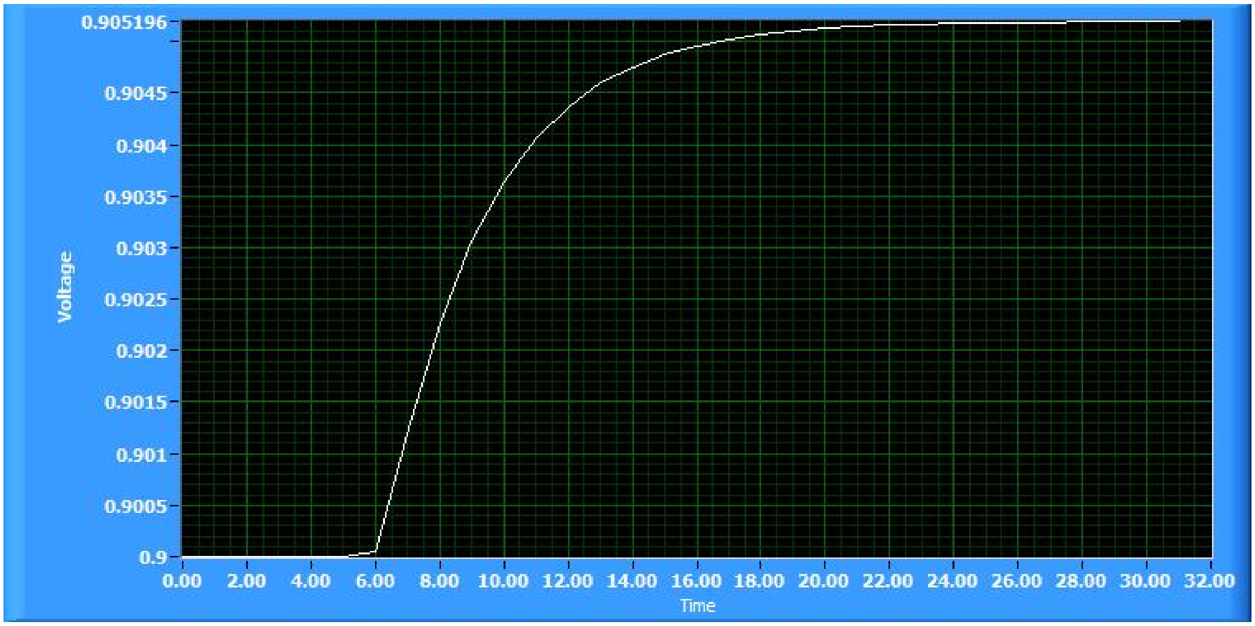


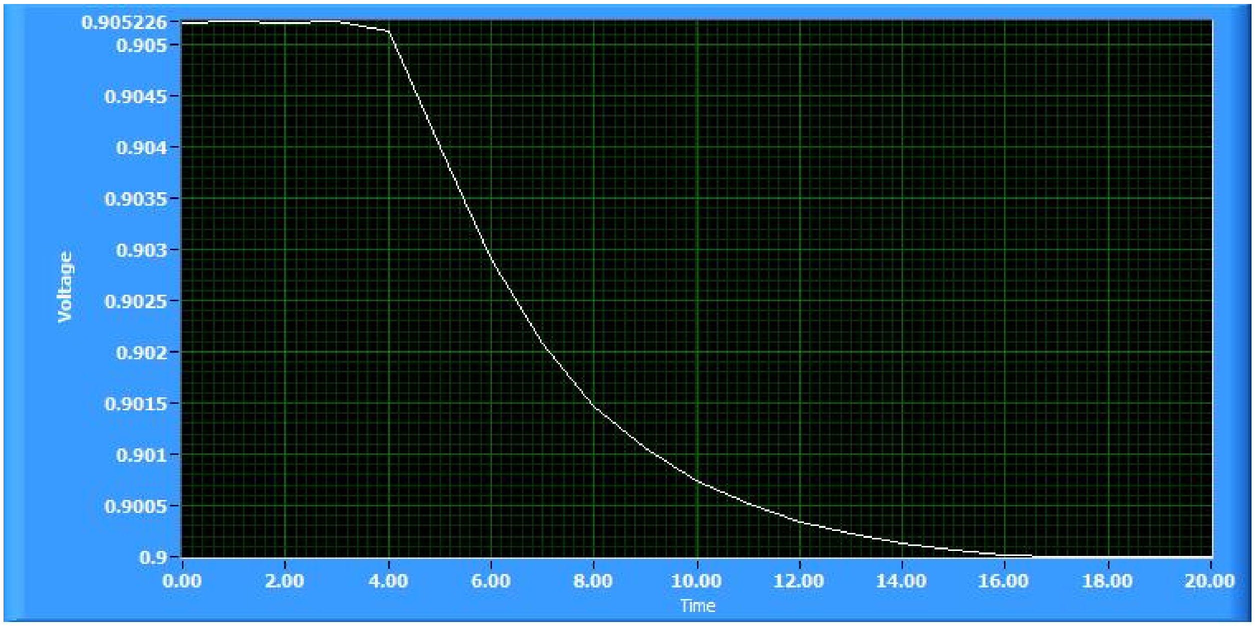
Fluke 300°C


0.905222
0.905
0.9045
0.904
0.9035
0.903
0.9025
0.902
0.9015
0.901
0.9005
0.9
Voltage
0.00
2.00
4.00
6.00
8.00
10.00
12.00
14.00
16.00
18.00
20.00
22.00
24.00
26.00
28.00
Time


0.905249
0.905
0.9045
0.904
0.9035
0.903
0.9025
0.902
0.9015
0.901
0.9005
0.9
Voltage
0.00
2.50
5.00
7.50
10.00
12.50
15.00
17.50
20.00
22.50
25.00
Time

Melexis 300°C





Time Constants

No Sensor

		Time @ 10% (seconds)	Time @ 90% (seconds)	Time constant (ms)
	Rise	0.6	1.52	419
	Fall	3.136	2.28	390

Time Constant (ms)	419	(system, rising)
Time Constant (ms):	390	(system, falling)

Melexis Sensor

100°C	Rise	0.592	1.792	546
	Fall	1.632	0.816	371
200°C	Rise	0.92	2.08	528
	Fall	2.096	1.152	430
300°C	Rise	1.024	2.176	524
	Fall	1.744	0.704	473

Avg Time Constant (ms):	533	(sensor plus system, rising)
Avg Time Constant (ms):	114	(sensor only, rising)
Avg Time Constant (ms):	425	(sensor plus system, falling)
Avg Time Constant (ms):	35	(sensor only, falling)

Fluke Sensor

100°C	Rise	0.48	1.72	564
	Fall	2.64	1.56	492
200°C	Rise	1.52	2.752	561
	Fall	4.52	3.48	473
300°C	Rise	0.656	1.84	539
	Fall	1.72	0.656	484

Avg Time Constant (ms):	555	(sensor plus system, rising)
Avg Time Constant (ms):	136	(sensor only, rising)
Avg Time Constant (ms):	483	(sensor plus system, falling)
Avg Time Constant (ms):	93	(sensor only, falling)

Appendix B: System Calibration Data

This section contains the data from our ambient and thermopile calibrations as well as our final system tests.

Ambient Temperature Calibration		
4/10/2005		
Actual Temp (°C)	Hex Measurement	Dec Value
42.4	89C0	35264
27.6	8090	32912
20.4	7C8B	31883
14.6	7982	31106
3	71F9	29177

IR Temperature Calibration - 15 cm						
4/10/2005						
Actual Temp (°C)	IR Hex	IR Dec	Ambient Temp (°C)	Ambient Hex	Ambient Dec	Temp Difference (°C)
30	801E	32798	24.5	7F06	32518	5.5
40	805A	32858	24.9	7F43	32579	15.1
50	8097	32919	25.2	7F72	32626	24.8
60	80DA	32986	25.5	7F9C	32668	34.5
70	8124	33060	25.7	7FBC	32700	44.3
80	8172	33138	25.9	7FDB	32731	54.1
90	81C7	33223	26.1	7FF9	32761	63.9
100	8220	33312	26.3	8016	32790	73.7
110	827D	33405	26.5	8032	32818	83.5
120	82DF	33503	26.7	8050	32848	93.3
130	8348	33608	26.8	8067	32871	103.2
140	83B3	33715	27.0	8083	32899	113.0
150	8422	33826	27.3	80A9	32937	122.7
160	849E	33950	27.8	80F3	33011	132.2
170	8519	34073	27.9	8108	33032	142.1
180	859B	34203	28.1	8123	33059	151.9
190	862C	34348	28.2	813A	33082	161.8
200	86C3	34499	28.3	814B	33099	171.7
210	8752	34642	28.5	8166	33126	181.5
220	87EF	34799	28.6	817A	33146	191.4
230	8889	34953	28.8	8191	33169	201.2
240	8919	35097	29.0	81B3	33203	211.0
250	89C0	35264	29.2	81D1	33233	220.8
260	8A72	35442	29.4	81EF	33263	230.6
270	8B26	35622	29.6	820A	33290	240.4
280	8BD8	35800	29.8	822F	33327	250.2
290	8C92	35986	30.1	8253	33363	259.9

300	8D53	36179	30.4	8280	33408	269.6
310	8E1C	36380	30.7	82BA	33466	279.3
320	8EE8	36584	31.1	82F5	33525	288.9
330	8FAA	36778	31.5	8336	33590	298.5
340	9079	36985	32.0	837A	33658	308.0
350	914C	37196	32.5	83C7	33735	317.5
360	9220	37408	32.9	840C	33804	327.1
370	92FD	37629	33.4	8457	33879	336.6
380	93D7	37847	34.0	84AB	33963	346.0
390	94BD	38077	34.6	8508	34056	355.4
400	95A0	38304	35.3	8574	34164	364.7
410	968C	38540	35.9	85D2	34258	374.1
420	977E	38782	36.5	8631	34353	383.5
430	9870	39024	37.1	868D	34445	392.9
440	995E	39262	37.8	86F6	34550	402.2
450	9A4D	39501	38.6	8765	34661	411.4

Final System Test					
5/10/2005					
Actual Temp (°C)	Calc Temp (°C)	Percent Error (%)	Hex Temp	Dec Temp	Ambient Temp (°C)
30	30.3	1.00	801D	32797	23.7
50	50.2	0.40	809D	32925	23.7
90	89.4	0.67	81D7	33239	23.6
110	109.1	0.82	8292	33426	23.7
130	128.8	0.92	8361	33633	23.8
150	148.8	0.80	8448	33864	23.9
170	168.8	0.71	8544	34116	24.1
190	189.2	0.42	8658	34392	24.3
210	209.4	0.29	8782	34690	24.5
230	229.4	0.26	88BA	35002	24.8
250	249.3	0.28	8A07	35335	25.1
270	269.5	0.19	8B6A	35690	25.4
290	289.9	0.03	8CE2	36066	26.1
310	310.6	0.19	8E74	36468	26.7
330	330.2	0.06	900A	36874	27.2
350	350.2	0.06	91B5	37301	27.8
370	370.1	0.03	936E	37742	28.6
390	389.8	0.05	9535	38197	29.4
410	409.5	0.12	9715	38677	30.1
430	429.5	0.12	9900	39168	31.3
450	448.5	0.33	9AEC	39660	32.1

Appendix C: Firmware Code

The following code was used to program the USB chip to send and receive information using LabView software. Most of the code was provided by Cypress and some of it was modified by Michal Brychta of Analog Devices. The firmware code was already compiled into a .hex format and provided by Analog Devices.

C.1 FX2.h

```
//-----------------------------------------------------------------------------
//   File:      FX2.h
//   Contents:   EZ-USB FX2 constants, macros, datatypes, globals, and library
//           function prototypes.
//
//   Copyright (c) 2000 Cypress Semiconductor, All rights reserved
//-----------------------------------------------------------------------------
#ifndef FX2_H     //Header sentry
#define FX2_H

#define INTERNAL_DSCR_ADDR 0x0080   // Relocate Descriptors to 0x80
#define bmSTRETCH 0x07
#define FW_STRETCH_VALUE 0x0      // Set stretch to 0 in frameworks
                        // Note: a RevE eratta states that stretch must=0 to set OUTxBC

//-----------------------------------------------------------------------------
// Constants
//-----------------------------------------------------------------------------
#define   TRUE    1
#define FALSE   0

#define bmBIT0   0x01
#define bmBIT1   0x02
#define bmBIT2   0x04
#define bmBIT3   0x08
#define bmBIT4   0x10
#define bmBIT5   0x20
#define bmBIT6   0x40
#define bmBIT7   0x80

#define DEVICE_DSCR      0x01      // Descriptor type: Device
#define CONFIG_DSCR      0x02      // Descriptor type: Configuration
#define STRING_DSCR      0x03      // Descriptor type: String
#define INTRFC_DSCR      0x04      // Descriptor type: Interface
#define ENDPNT_DSCR      0x05      // Descriptor type: End Point
#define DEVQUAL_DSCR     0x06      // Descriptor type: Device Qualifier
#define OTHERSPEED_DSCR  0x07      // Descriptor type: Other Speed Configuration

#define bmBUSPWR  bmBIT7         // Config. attribute: Bus powered
#define bmSELFPWR bmBIT6         // Config. attribute: Self powered
#define bmRWU     bmBIT5         // Config. attribute: Remote Wakeup

#define bmEPOUT   bmBIT7
#define bmEPIN    0x00

#define EP_CONTROL   0x00       // End Point type: Control
#define EP_ISO       0x01       // End Point type: Isochronous
#define EP_BULK      0x02       // End Point type: Bulk
#define EP_INT       0x03       // End Point type: Interrupt

#define SUD_SIZE           8      // Setup data packet size

//////////////////////////////////////////////////////////////////////////////
//Added for HID
```

```
#define SETUP_MASK                  0x60      //Used to mask off request type
#define SETUP_STANDARD_REQUEST   0            //Standard Request
#define SETUP_CLASS_REQUEST         0x20      //Class Request
#define SETUP_VENDOR_REQUEST     0x40         //Vendor Request
#define SETUP_RESERVED_REQUEST   0x60         //Reserved or illegal request

////////////////////////////////////////////////////////////////////////////


#define SC_GET_STATUS        0x00   // Setup command: Get Status
#define SC_CLEAR_FEATURE     0x01   // Setup command: Clear Feature
#define SC_RESERVED          0x02   // Setup command: Reserved
#define SC_SET_FEATURE       0x03   // Setup command: Set Feature
#define SC_SET_ADDRESS       0x05   // Setup command: Set Address
#define SC_GET_DESCRIPTOR    0x06   // Setup command: Get Descriptor
#define SC_SET_DESCRIPTOR    0x07   // Setup command: Set Descriptor
#define SC_GET_CONFIGURATION   0x08   // Setup command: Get Configuration
#define SC_SET_CONFIGURATION   0x09   // Setup command: Set Configuration
#define SC_GET_INTERFACE     0x0a   // Setup command: Get Interface
#define SC_SET_INTERFACE     0x0b   // Setup command: Set Interface
#define SC_SYNC_FRAME        0x0c   // Setup command: Sync Frame
#define SC_ANCHOR_LOAD       0xa0   // Setup command: Anchor load

#define GD_DEVICE         0x01  // Get descriptor: Device
#define GD_CONFIGURATION   0x02  // Get descriptor: Configuration
#define GD_STRING         0x03  // Get descriptor: String
#define GD_INTERFACE      0x04  // Get descriptor: Interface
#define GD_ENDPOINT       0x05  // Get descriptor: Endpoint
#define GD_DEVICE_QUALIFIER 0x06  // Get descriptor: Device Qualifier
#define GD_OTHER_SPEED_CONFIGURATION 0x07  // Get descriptor: Other Configuration
#define GD_INTERFACE_POWER 0x08  // Get descriptor: Interface Power
#define GD_HID            0x21        // Get descriptor: HID
#define GD_REPORT                0x22 // Get descriptor: Report

#define GS_DEVICE         0x80  // Get Status: Device
#define GS_INTERFACE      0x81  // Get Status: Interface
#define GS_ENDPOINT       0x82  // Get Status: End Point

#define FT_DEVICE         0x00  // Feature: Device
#define FT_ENDPOINT       0x02  // Feature: End Point

#define I2C_IDLE          0    // I2C Status: Idle mode
#define I2C_SENDING         1    // I2C Status: I2C is sending data
#define I2C_RECEIVING         2    // I2C Status: I2C is receiving data
#define I2C_PRIME           3    // I2C Status: I2C is receiving the first byte of a string
#define I2C_STOP          5    // I2C Status: I2C waiting for stop completion
#define I2C_BERROR          6    // I2C Status: I2C error; Bit Error
#define I2C_NACK            7    // I2C Status: I2C error; No Acknowledge
#define I2C_OK            8    // I2C positive return code
#define I2C_WAITSTOP          9    // I2C Status: Wait for STOP complete

/*-----------------------------------------------------------------------------
  Macros
-----------------------------------------------------------------------------*/

#define MSB(word)     (BYTE)(((WORD)(word) >> 8) & 0xff)
#define LSB(word)     (BYTE)((WORD)(word) & 0xff)

#define SWAP_ENDIAN(word)   ((BYTE*)&word)[0] ^= ((BYTE*)&word)[1];\
              ((BYTE*)&word)[1] ^= ((BYTE*)&word)[0];\
              ((BYTE*)&word)[0] ^= ((BYTE*)&word)[1]

#define EZUSB_IRQ_ENABLE()   EUSB = 1
#define EZUSB_IRQ_DISABLE()   EUSB = 0
#define EZUSB_IRQ_CLEAR()   EXIF &= ~0x10      // IE2_

#define EZUSB_STALL_EP0()          EP0CS |= bmEPSTALL
#define EZUSB_STALL_EP(ep_id)          // fx2bug
#define EZUSB_UNSTALL_EP(ep_id)          // fx2bug
#define   EZUSB_GET_EP_STATUS(ep_id)      // fx2bug
```

```
#define EZUSB_SET_EP_BYTES(ep_id,count)   // fx2bug


// WRITEDELAY() has been replaced by SYNCDELAY; macro in fx2sdly.h
// ...it is here for backwards compatibility...

// the WRITEDELAY macro compiles to the time equivalent of 3 NOPs.
// It is used in the frameworks to allow for write recovery time
// requirements of certain registers.  This is only necessary for
// EZ-USB FX parts.  See the EZ-USB FX TRM for
// more information on write recovery time issues.
#define WRITEDELAY() {char writedelaydummy = 0;}
// if this firmware will never run on an EZ-USB FX part replace
// with:
// #define WRITEDELAY()

// macro to reset and endpoint data toggle
#define EZUSB_RESET_DATA_TOGGLE(ep)     TOGCTL = (((ep & 0x80) >> 3) + (ep & 0x0F));\
                                TOGCTL |= bmRESETTOGGLE


#define EZUSB_ENABLE_RSMIRQ()      (EICON |= 0x20)      // Enable Resume Interrupt (EPFI_)
#define EZUSB_DISABLE_RSMIRQ()      (EICON &= ~0x20)   // Disable Resume Interrupt (EPFI_)
#define EZUSB_CLEAR_RSMIRQ()      (EICON &= ~0x10)   // Clear Resume Interrupt Flag (PFI_)

#define EZUSB_GETI2CSTATUS()      (I2CPckt.status)
#define EZUSB_CLEARI2CSTATUS()      if((I2CPckt.status == I2C_BERROR) || (I2CPckt.status == I2C_NACK))\
                        I2CPckt.status = I2C_IDLE;

#define EZUSB_ENABLEBP()         (BREAKPT |= bmBPEN)    // TGE fx2bug
#define EZUSB_DISABLEBP()         (BREAKPT &= ~bmBPEN)  // TGE fx2bug
#define EZUSB_CLEARBP()           (BREAKPT |= bmBREAK) // TGE fx2bug
#define EZUSB_BP(addr)           BPADDRH = (BYTE)(((WORD)addr >> 8) & 0xff);\
                          BPADDRL = (BYTE)addr                          // TGE fx2bug

#define EZUSB_EXTWAKEUP()      (((WAKEUPCS & bmWU2) && (WAKEUPCS & bmWU2EN)) ||\
                         ((WAKEUPCS & bmWU) &&  (WAKEUPCS & bmWUEN)))

#define EZUSB_HIGHSPEED()      (USBCS & bmHSM)

//-----------------------------------------------------------------------------
// Datatypes
//-----------------------------------------------------------------------------
typedef unsigned char   BYTE;
typedef unsigned short   WORD;
typedef unsigned long   DWORD;
typedef bit            BOOL;

#define  INT0_VECT   0
#define  TMR0_VECT   1
#define  INT1_VECT   2
#define  TMR1_VECT   3
#define  COM0_VECT   4
#define  TMR2_VECT   5
#define  WKUP_VECT   6
#define  COM1_VECT   7
#define  USB_VECT    8
#define  I2C_VECT    9
#define  INT4_VECT   10
#define  INT5_VECT   11
#define  INT6_VECT   12


// TGE fx2bug
#define   SUDAV_USBVECT      (0 << 2)
#define   SOF_USBVECT        (1 << 2)
#define   SUTOK_USBVECT      (2 << 2)
#define   SUSP_USBVECT       (3 << 2)
#define   URES_USBVECT       (4 << 2)
#define   HS_USBVECT         (5 << 2)
```

```
#define   EP0ACK_USBVECT      (6 << 2)
#define   SPARE0_USBVECT      (7 << 2)
#define   IN0BUF_USBVECT      (8 << 2)
#define   OUT0BUF_USBVECT     (9 << 2)
#define   IN1BUF_USBVECT      (10 << 2)
#define   OUT1BUF_USBVECT     (11 << 2)
#define   INOUT2BUF_USBVECT   (12 << 2)
#define   INOUT4BUF_USBVECT   (13 << 2)
#define   INOUT6BUF_USBVECT   (14 << 2)
#define   INOUT8BUF_USBVECT   (15 << 2)
#define   IBN_USBVECT         (16 << 2)
#define   SPARE1_USBVECT      (17 << 2)
#define   EP0PINGNAK_USBVECT  (18 << 2)
#define   EP1PINGNAK_USBVECT  (19 << 2)
#define   EP2PINGNAK_USBVECT  (20 << 2)
#define   EP4PINGNAK_USBVECT  (21 << 2)
#define   EP6PINGNAK_USBVECT  (22 << 2)
#define   EP8PINGNAK_USBVECT  (23 << 2)
#define   ERRLIM_USBVECT      (24 << 2)
#define   SPARE2_USBVECT      (25 << 2)
#define   SPARE3_USBVECT      (26 << 2)
#define   SPARE4_USBVECT      (27 << 2)
#define   EP2PIDERR_USBVECT   (28 << 2)
#define   EP4PIDERR_USBVECT   (29 << 2)
#define   EP6PIDERR_USBVECT   (30 << 2)
#define   EP8PIDERR_USBVECT   (31 << 2)

typedef struct
{
  BYTE   length;
  BYTE   type;
}DSCR;

typedef struct          // Device Descriptor
{
  BYTE   length;         // Descriptor length ( = sizeof(DEVICEDSCR) )
  BYTE   type;         // Decriptor type (Device = 1)
  BYTE   spec_ver_minor;   // Specification Version (BCD) minor
  BYTE   spec_ver_major;   // Specification Version (BCD) major
  BYTE   dev_class;      // Device class
  BYTE   sub_class;      // Device sub-class
  BYTE   protocol;      // Device sub-sub-class
  BYTE   max_packet;      // Maximum packet size
  WORD   vendor_id;      // Vendor ID
  WORD   product_id;      // Product ID
  WORD   version_id;      // Product version ID
  BYTE   mfg_str;      // Manufacturer string index
  BYTE   prod_str;      // Product string index
  BYTE   serialnum_str;   // Serial number string index
  BYTE   configs;      // Number of configurations
}DEVICEDSCR;

typedef struct          // Device Qualifier Descriptor
{
  BYTE   length;         // Descriptor length ( = sizeof(DEVICEQUALDSCR) )
  BYTE   type;         // Decriptor type (Device Qualifier = 6)
  BYTE   spec_ver_minor;   // Specification Version (BCD) minor
  BYTE   spec_ver_major;   // Specification Version (BCD) major
  BYTE   dev_class;      // Device class
  BYTE   sub_class;      // Device sub-class
  BYTE   protocol;      // Device sub-sub-class
  BYTE   max_packet;      // Maximum packet size
  BYTE   configs;      // Number of configurations
  BYTE  reserved0;
}DEVICEQUALDSCR;

typedef struct
{
  BYTE   length;         // Configuration length ( = sizeof(CONFIGDSCR) )
  BYTE   type;         // Descriptor type (Configuration = 2)
```

```
  WORD   config_len;     // Configuration + End Points length
  BYTE   interfaces;     // Number of interfaces
  BYTE   index;          // Configuration number
  BYTE   config_str;     // Configuration string
  BYTE   attrib;         // Attributes (b7 - buspwr, b6 - selfpwr, b5 - rwu
  BYTE   power;          // Power requirement (div 2 ma)
}CONFIGDSCR;

typedef struct
{
  BYTE   length;         // Interface descriptor length ( - sizeof(INTRFCDSCR) )
  BYTE   type;           // Descriptor type (Interface = 4)
  BYTE   index;          // Zero-based index of this interface
  BYTE   alt_setting;    // Alternate setting
  BYTE   ep_cnt;         // Number of end points
  BYTE   class;          // Interface class
  BYTE   sub_class;      // Interface sub class
  BYTE   protocol;       // Interface sub sub class
  BYTE   interface_str;  // Interface descriptor string index
}INTRFCDSCR;

typedef struct
{
  BYTE   length;         // End point descriptor length ( = sizeof(ENDPNTDSCR) )
  BYTE   type;           // Descriptor type (End point = 5)
  BYTE   addr;           // End point address
  BYTE   ep_type;        // End point type
  BYTE   mp_L;           // Maximum packet size
  BYTE   mp_H;
  BYTE   interval;       // Interrupt polling interval
}ENDPNTDSCR;

typedef struct
{
  BYTE   length;         // String descriptor length
  BYTE   type;           // Descriptor type
}STRINGDSCR;

typedef struct
{
  BYTE   cntrl;          // End point control register
  BYTE   bytes;          // End point buffer byte count
}EPIOC;

typedef struct
{
  BYTE   length;
  BYTE   *dat;
  BYTE   count;
  BYTE   status;
}I2CPCKT;

//-----------------------------------------------------------------------------
// Globals
//-----------------------------------------------------------------------------
extern code BYTE   USB_AutoVector;

extern WORD   pDeviceDscr;
extern WORD   pDeviceQualDscr;
extern WORD          pHighSpeedConfigDscr;
extern WORD          pFullSpeedConfigDscr;
extern WORD   pConfigDscr;
extern WORD   pOtherConfigDscr;
extern WORD   pStringDscr;

extern code DEVICEDSCR       DeviceDscr;
extern code DEVICEQUALDSCR   DeviceQualDscr;
extern code CONFIGDSCR       HighSpeedConfigDscr;
extern code CONFIGDSCR       FullSpeedConfigDscr;
extern code STRINGDSCR       StringDscr;
```

```
extern code DSCR            UserDscr;

extern I2CPCKT   I2CPckt;

//-----------------------------------------------------------------------------
// Function Prototypes
//-----------------------------------------------------------------------------
// fx2bug #ifdef CHIPREV_B
// fx2bug extern void EZUSB_IRQ_CLEAR(void);
// fx2bug #endif

extern void EZUSB_Renum(void);
extern void EZUSB_Discon(BOOL renum);

extern void EZUSB_Susp(void);
extern void EZUSB_Resume(void);

extern void EZUSB_Delay1ms(void);
extern void EZUSB_Delay(WORD ms);

extern CONFIGDSCR xdata*   EZUSB_GetConfigDscr(BYTE ConfigIdx);
extern INTRFCDSCR xdata*   EZUSB_GetIntrfcDscr(BYTE ConfigIdx, BYTE IntrfcIdx, BYTE AltSetting);
extern STRINGDSCR xdata*   EZUSB_GetStringDscr(BYTE StrIdx);
extern DSCR xdata*      EZUSB_GetDscr(BYTE index, DSCR* dscr, BYTE type);

extern void EZUSB_InitI2C(void);
extern BOOL EZUSB_WriteI2C_(BYTE addr, BYTE length, BYTE xdata *dat);
extern BOOL EZUSB_ReadI2C_(BYTE addr, BYTE length, BYTE xdata *dat);
extern BOOL EZUSB_WriteI2C(BYTE addr, BYTE length, BYTE xdata *dat);
extern BOOL EZUSB_ReadI2C(BYTE addr, BYTE length, BYTE xdata *dat);
extern void EZUSB_WaitForEEPROMWrite(BYTE addr);

extern void modify_endpoint_stall(BYTE epid, BYTE stall);

#endif   // FX2_H
```

C.2 FX2regs.h

```
//-----------------------------------------------------------------------------
//   File:      FX2regs.h
//   Contents:   EZ-USB FX2 register declarations and bit mask definitions.
//
// $Archive: /USB/Target/Inc/fx2regs.h $
// $Date: 4/19/02 9:53a $
// $Revision: 35 $
//
//
//   Copyright (c) 2000 Cypress Semiconductor, All rights reserved
//-----------------------------------------------------------------------------

#ifndef FX2REGS_H   /* Header Sentry */
#define FX2REGS_H

//-----------------------------------------------------------------------------
// FX2 Related Register Assignments
//-----------------------------------------------------------------------------

// The Ez-USB FX2 registers are defined here. We use FX2regs.h for register
// address allocation by using "#define ALLOCATE_EXTERN".
// When using "#define ALLOCATE_EXTERN", you get (for instance):
// xdata volatile BYTE OUT7BUF[64]   _at_   0x7B40;
// Such lines are created from FX2.h by using the preprocessor.
// Incidently, these lines will not generate any space in the resulting hex
// file; they just bind the symbols to the addresses for compilation.
// You just need to put "#define ALLOCATE_EXTERN" in your main program file;
// i.e. fw.c or a stand-alone C source file.
```

```
// Without "#define ALLOCATE_EXTERN", you just get the external reference:
// extern xdata volatile BYTE OUT7BUF[64]   ;//   0x7B40;
// This uses the concatenation operator "##" to insert a comment "//"
// to cut off the end of the line, "_at_   0x7B40;", which is not wanted.

#ifdef ALLOCATE_EXTERN
#define EXTERN
#define _AT_ _at_
#else
#define EXTERN extern
#define _AT_ ;/ ## /
#endif

EXTERN xdata volatile BYTE GPIF_WAVE_DATA     _AT_ 0xE400;
EXTERN xdata volatile BYTE RES_WAVEDATA_END   _AT_ 0xE480;

// General Configuration

EXTERN xdata volatile BYTE CPUCS          _AT_ 0xE600;  // Control & Status
EXTERN xdata volatile BYTE IFCONFIG       _AT_ 0xE601;  // Interface Configuration
EXTERN xdata volatile BYTE PINFLAGSAB     _AT_ 0xE602;  // FIFO FLAGA and FLAGB Assignments
EXTERN xdata volatile BYTE PINFLAGSCD     _AT_ 0xE603;  // FIFO FLAGC and FLAGD Assignments
EXTERN xdata volatile BYTE FIFORESET      _AT_ 0xE604;  // Restore FIFOS to default state
EXTERN xdata volatile BYTE BREAKPT        _AT_ 0xE605;  // Breakpoint
EXTERN xdata volatile BYTE BPADDRH        _AT_ 0xE606;  // Breakpoint Address H
EXTERN xdata volatile BYTE BPADDRL        _AT_ 0xE607;  // Breakpoint Address L
EXTERN xdata volatile BYTE UART230        _AT_ 0xE608;  // 230 Kbaud clock for T0,T1,T2
EXTERN xdata volatile BYTE FIFOPINPOLAR   _AT_ 0xE609;  // FIFO polarities
EXTERN xdata volatile BYTE REVID          _AT_ 0xE60A;  // Chip Revision
EXTERN xdata volatile BYTE REVCTL         _AT_ 0xE60B;  // Chip Revision Control

// Endpoint Configuration

EXTERN xdata volatile BYTE EP1OUTCFG      _AT_ 0xE610;  // Endpoint 1-OUT Configuration
EXTERN xdata volatile BYTE EP1INCFG       _AT_ 0xE611;  // Endpoint 1-IN Configuration
EXTERN xdata volatile BYTE EP2CFG         _AT_ 0xE612;  // Endpoint 2 Configuration
EXTERN xdata volatile BYTE EP4CFG         _AT_ 0xE613;  // Endpoint 4 Configuration
EXTERN xdata volatile BYTE EP6CFG         _AT_ 0xE614;  // Endpoint 6 Configuration
EXTERN xdata volatile BYTE EP8CFG         _AT_ 0xE615;  // Endpoint 8 Configuration
EXTERN xdata volatile BYTE EP2FIFOCFG     _AT_ 0xE618;  // Endpoint 2 FIFO configuration
EXTERN xdata volatile BYTE EP4FIFOCFG     _AT_ 0xE619;  // Endpoint 4 FIFO configuration
EXTERN xdata volatile BYTE EP6FIFOCFG     _AT_ 0xE61A;  // Endpoint 6 FIFO configuration
EXTERN xdata volatile BYTE EP8FIFOCFG     _AT_ 0xE61B;  // Endpoint 8 FIFO configuration
EXTERN xdata volatile BYTE EP2AUTOINLENH  _AT_ 0xE620;  // Endpoint 2 Packet Length H (IN only)
EXTERN xdata volatile BYTE EP2AUTOINLENL  _AT_ 0xE621;  // Endpoint 2 Packet Length L (IN only)
EXTERN xdata volatile BYTE EP4AUTOINLENH  _AT_ 0xE622;  // Endpoint 4 Packet Length H (IN only)
EXTERN xdata volatile BYTE EP4AUTOINLENL  _AT_ 0xE623;  // Endpoint 4 Packet Length L (IN only)
EXTERN xdata volatile BYTE EP6AUTOINLENH  _AT_ 0xE624;  // Endpoint 6 Packet Length H (IN only)
EXTERN xdata volatile BYTE EP6AUTOINLENL  _AT_ 0xE625;  // Endpoint 6 Packet Length L (IN only)
EXTERN xdata volatile BYTE EP8AUTOINLENH  _AT_ 0xE626;  // Endpoint 8 Packet Length H (IN only)
EXTERN xdata volatile BYTE EP8AUTOINLENL  _AT_ 0xE627;  // Endpoint 8 Packet Length L (IN only)
EXTERN xdata volatile BYTE EP2FIFOPFH     _AT_ 0xE630;  // EP2 Programmable Flag trigger H
EXTERN xdata volatile BYTE EP2FIFOPFL     _AT_ 0xE631;  // EP2 Programmable Flag trigger L
EXTERN xdata volatile BYTE EP4FIFOPFH     _AT_ 0xE632;  // EP4 Programmable Flag trigger H
EXTERN xdata volatile BYTE EP4FIFOPFL     _AT_ 0xE633;  // EP4 Programmable Flag trigger L
EXTERN xdata volatile BYTE EP6FIFOPFH     _AT_ 0xE634;  // EP6 Programmable Flag trigger H
EXTERN xdata volatile BYTE EP6FIFOPFL     _AT_ 0xE635;  // EP6 Programmable Flag trigger L
EXTERN xdata volatile BYTE EP8FIFOPFH     _AT_ 0xE636;  // EP8 Programmable Flag trigger H
EXTERN xdata volatile BYTE EP8FIFOPFL     _AT_ 0xE637;  // EP8 Programmable Flag trigger L
EXTERN xdata volatile BYTE EP2ISOINPKTS   _AT_ 0xE640;  // EP2 (if ISO) IN Packets per frame (1-3)
EXTERN xdata volatile BYTE EP4ISOINPKTS   _AT_ 0xE641;  // EP4 (if ISO) IN Packets per frame (1-3)
EXTERN xdata volatile BYTE EP6ISOINPKTS   _AT_ 0xE642;  // EP6 (if ISO) IN Packets per frame (1-3)
EXTERN xdata volatile BYTE EP8ISOINPKTS   _AT_ 0xE643;  // EP8 (if ISO) IN Packets per frame (1-3)
EXTERN xdata volatile BYTE INPKTEND       _AT_ 0xE648;  // Force IN Packet End
EXTERN xdata volatile BYTE OUTPKTEND      _AT_ 0xE649;  // Force OUT Packet End

// Interrupts

EXTERN xdata volatile BYTE EP2FIFOIE      _AT_ 0xE650;  // Endpoint 2 Flag Interrupt Enable
EXTERN xdata volatile BYTE EP2FIFOIRQ     _AT_ 0xE651;  // Endpoint 2 Flag Interrupt Request
```

```
EXTERN xdata volatile BYTE EP4FIFOIE        _AT_ 0xE652;  // Endpoint 4 Flag Interrupt Enable
EXTERN xdata volatile BYTE EP4FIFOIRQ       _AT_ 0xE653;  // Endpoint 4 Flag Interrupt Request
EXTERN xdata volatile BYTE EP6FIFOIE        _AT_ 0xE654;  // Endpoint 6 Flag Interrupt Enable
EXTERN xdata volatile BYTE EP6FIFOIRQ       _AT_ 0xE655;  // Endpoint 6 Flag Interrupt Request
EXTERN xdata volatile BYTE EP8FIFOIE        _AT_ 0xE656;  // Endpoint 8 Flag Interrupt Enable
EXTERN xdata volatile BYTE EP8FIFOIRQ       _AT_ 0xE657;  // Endpoint 8 Flag Interrupt Request
EXTERN xdata volatile BYTE IBNIE            _AT_ 0xE658;  // IN-BULK-NAK Interrupt Enable
EXTERN xdata volatile BYTE IBNIRQ           _AT_ 0xE659;  // IN-BULK-NAK interrupt Request
EXTERN xdata volatile BYTE NAKIE            _AT_ 0xE65A;  // Endpoint Ping NAK interrupt Enable
EXTERN xdata volatile BYTE NAKIRQ           _AT_ 0xE65B;  // Endpoint Ping NAK interrupt Request
EXTERN xdata volatile BYTE USBIE            _AT_ 0xE65C;  // USB Int Enables
EXTERN xdata volatile BYTE USBIRQ           _AT_ 0xE65D;  // USB Interrupt Requests
EXTERN xdata volatile BYTE EPIE             _AT_ 0xE65E;  // Endpoint Interrupt Enables
EXTERN xdata volatile BYTE EPIRQ            _AT_ 0xE65F;  // Endpoint Interrupt Requests
EXTERN xdata volatile BYTE GPIFIE           _AT_ 0xE660;  // GPIF Interrupt Enable
EXTERN xdata volatile BYTE GPIFIRQ          _AT_ 0xE661;  // GPIF Interrupt Request
EXTERN xdata volatile BYTE USBERRIE         _AT_ 0xE662;  // USB Error Interrupt Enables
EXTERN xdata volatile BYTE USBERRIRQ        _AT_ 0xE663;  // USB Error Interrupt Requests
EXTERN xdata volatile BYTE ERRCNTLIM        _AT_ 0xE664;  // USB Error counter and limit
EXTERN xdata volatile BYTE CLRERRCNT        _AT_ 0xE665;  // Clear Error Counter EC[3..0]
EXTERN xdata volatile BYTE INT2IVEC         _AT_ 0xE666;  // Interupt 2 (USB) Autovector
EXTERN xdata volatile BYTE INT4IVEC         _AT_ 0xE667;  // Interupt 4 (FIFOS & GPIF) Autovector
EXTERN xdata volatile BYTE INTSETUP         _AT_ 0xE668;  // Interrupt 2&4 Setup

// Input/Output

EXTERN xdata volatile BYTE PORTACFG         _AT_ 0xE670;  // I/O PORTA Alternate Configuration
EXTERN xdata volatile BYTE PORTCCFG         _AT_ 0xE671;  // I/O PORTC Alternate Configuration
EXTERN xdata volatile BYTE PORTECFG         _AT_ 0xE672;  // I/O PORTE Alternate Configuration
EXTERN xdata volatile BYTE I2CS             _AT_ 0xE678;  // Control & Status
EXTERN xdata volatile BYTE I2DAT            _AT_ 0xE679;  // Data
EXTERN xdata volatile BYTE I2CTL            _AT_ 0xE67A;  // I2C Control
EXTERN xdata volatile BYTE XAUTODAT1        _AT_ 0xE67B;  // Autoptr1 MOVX access
EXTERN xdata volatile BYTE XAUTODAT2        _AT_ 0xE67C;  // Autoptr2 MOVX access

#define EXTAUTODAT1 XAUTODAT1
#define EXTAUTODAT2 XAUTODAT2

// USB Control

EXTERN xdata volatile BYTE USBCS            _AT_ 0xE680;  // USB Control & Status
EXTERN xdata volatile BYTE SUSPEND          _AT_ 0xE681;  // Put chip into suspend
EXTERN xdata volatile BYTE WAKEUPCS         _AT_ 0xE682;  // Wakeup source and polarity
EXTERN xdata volatile BYTE TOGCTL           _AT_ 0xE683;  // Toggle Control
EXTERN xdata volatile BYTE USBFRAMEH        _AT_ 0xE684;  // USB Frame count H
EXTERN xdata volatile BYTE USBFRAMEL        _AT_ 0xE685;  // USB Frame count L
EXTERN xdata volatile BYTE MICROFRAME       _AT_ 0xE686;  // Microframe count, 0-7
EXTERN xdata volatile BYTE FNADDR           _AT_ 0xE687;  // USB Function address

// Endpoints

EXTERN xdata volatile BYTE EP0BCH           _AT_ 0xE68A;  // Endpoint 0 Byte Count H
EXTERN xdata volatile BYTE EP0BCL           _AT_ 0xE68B;  // Endpoint 0 Byte Count L
EXTERN xdata volatile BYTE EP1OUTBC         _AT_ 0xE68D;  // Endpoint 1 OUT Byte Count
EXTERN xdata volatile BYTE EP1INBC          _AT_ 0xE68F;  // Endpoint 1 IN Byte Count
EXTERN xdata volatile BYTE EP2BCH           _AT_ 0xE690;  // Endpoint 2 Byte Count H
EXTERN xdata volatile BYTE EP2BCL           _AT_ 0xE691;  // Endpoint 2 Byte Count L
EXTERN xdata volatile BYTE EP4BCH           _AT_ 0xE694;  // Endpoint 4 Byte Count H
EXTERN xdata volatile BYTE EP4BCL           _AT_ 0xE695;  // Endpoint 4 Byte Count L
EXTERN xdata volatile BYTE EP6BCH           _AT_ 0xE698;  // Endpoint 6 Byte Count H
EXTERN xdata volatile BYTE EP6BCL           _AT_ 0xE699;  // Endpoint 6 Byte Count L
EXTERN xdata volatile BYTE EP8BCH           _AT_ 0xE69C;  // Endpoint 8 Byte Count H
EXTERN xdata volatile BYTE EP8BCL           _AT_ 0xE69D;  // Endpoint 8 Byte Count L
EXTERN xdata volatile BYTE EP0CS            _AT_ 0xE6A0;  // Endpoint  Control and Status
EXTERN xdata volatile BYTE EP1OUTCS         _AT_ 0xE6A1;  // Endpoint 1 OUT Control and Status
EXTERN xdata volatile BYTE EP1INCS          _AT_ 0xE6A2;  // Endpoint 1 IN Control and Status
EXTERN xdata volatile BYTE EP2CS            _AT_ 0xE6A3;  // Endpoint 2 Control and Status
EXTERN xdata volatile BYTE EP4CS            _AT_ 0xE6A4;  // Endpoint 4 Control and Status
EXTERN xdata volatile BYTE EP6CS            _AT_ 0xE6A5;  // Endpoint 6 Control and Status
EXTERN xdata volatile BYTE EP8CS            _AT_ 0xE6A6;  // Endpoint 8 Control and Status
```

```
EXTERN xdata volatile BYTE EP2FIFOFLGS       _AT_ 0xE6A7;  // Endpoint 2 Flags
EXTERN xdata volatile BYTE EP4FIFOFLGS       _AT_ 0xE6A8;  // Endpoint 4 Flags
EXTERN xdata volatile BYTE EP6FIFOFLGS       _AT_ 0xE6A9;  // Endpoint 6 Flags
EXTERN xdata volatile BYTE EP8FIFOFLGS       _AT_ 0xE6AA;  // Endpoint 8 Flags
EXTERN xdata volatile BYTE EP2FIFOBCH        _AT_ 0xE6AB;  // EP2 FIFO total byte count H
EXTERN xdata volatile BYTE EP2FIFOBCL        _AT_ 0xE6AC;  // EP2 FIFO total byte count L
EXTERN xdata volatile BYTE EP4FIFOBCH        _AT_ 0xE6AD;  // EP4 FIFO total byte count H
EXTERN xdata volatile BYTE EP4FIFOBCL        _AT_ 0xE6AE;  // EP4 FIFO total byte count L
EXTERN xdata volatile BYTE EP6FIFOBCH        _AT_ 0xE6AF;  // EP6 FIFO total byte count H
EXTERN xdata volatile BYTE EP6FIFOBCL        _AT_ 0xE6B0;  // EP6 FIFO total byte count L
EXTERN xdata volatile BYTE EP8FIFOBCH        _AT_ 0xE6B1;  // EP8 FIFO total byte count H
EXTERN xdata volatile BYTE EP8FIFOBCL        _AT_ 0xE6B2;  // EP8 FIFO total byte count L
EXTERN xdata volatile BYTE SUDPTRH           _AT_ 0xE6B3;  // Setup Data Pointer high address byte
EXTERN xdata volatile BYTE SUDPTRL           _AT_ 0xE6B4;  // Setup Data Pointer low address byte
EXTERN xdata volatile BYTE SUDPTRCTL         _AT_ 0xE6B5;  // Setup Data Pointer Auto Mode
EXTERN xdata volatile BYTE SETUPDAT[8]       _AT_ 0xE6B8;  // 8 bytes of SETUP data

// GPIF

EXTERN xdata volatile BYTE GPIFWFSELECT      _AT_ 0xE6C0;  // Waveform Selector
EXTERN xdata volatile BYTE GPIFIDLECS        _AT_ 0xE6C1;  // GPIF Done, GPIF IDLE drive mode
EXTERN xdata volatile BYTE GPIFIDLECTL       _AT_ 0xE6C2;  // Inactive Bus, CTL states
EXTERN xdata volatile BYTE GPIFCTLCFG        _AT_ 0xE6C3;  // CTL OUT pin drive
EXTERN xdata volatile BYTE GPIFADRH          _AT_ 0xE6C4;  // GPIF Address H
EXTERN xdata volatile BYTE GPIFADRL          _AT_ 0xE6C5;  // GPIF Address L

EXTERN xdata volatile BYTE GPIFTCB3          _AT_ 0xE6CE;  // GPIF Transaction Count Byte 3
EXTERN xdata volatile BYTE GPIFTCB2          _AT_ 0xE6CF;  // GPIF Transaction Count Byte 2
EXTERN xdata volatile BYTE GPIFTCB1          _AT_ 0xE6D0;  // GPIF Transaction Count Byte 1
EXTERN xdata volatile BYTE GPIFTCB0          _AT_ 0xE6D1;  // GPIF Transaction Count Byte 0

#define EP2GPIFTCH GPIFTCB1   // these are here for backwards compatibility
#define EP2GPIFTCL GPIFTCB0   // before REVE silicon (ie. REVB and REVD)
#define EP4GPIFTCH GPIFTCB1   // these are here for backwards compatibility
#define EP4GPIFTCL GPIFTCB0   // before REVE silicon (ie. REVB and REVD)
#define EP6GPIFTCH GPIFTCB1   // these are here for backwards compatibility
#define EP6GPIFTCL GPIFTCB0   // before REVE silicon (ie. REVB and REVD)
#define EP8GPIFTCH GPIFTCB1   // these are here for backwards compatibility
#define EP8GPIFTCL GPIFTCB0   // before REVE silicon (ie. REVB and REVD)

// EXTERN xdata volatile BYTE EP2GPIFTCH     _AT_ 0xE6D0;  // EP2 GPIF Transaction Count High
// EXTERN xdata volatile BYTE EP2GPIFTCL     _AT_ 0xE6D1;  // EP2 GPIF Transaction Count Low
EXTERN xdata volatile BYTE EP2GPIFFLGSEL     _AT_ 0xE6D2;  // EP2 GPIF Flag select
EXTERN xdata volatile BYTE EP2GPIFPFSTOP     _AT_ 0xE6D3;  // Stop GPIF EP2 transaction on prog. flag
EXTERN xdata volatile BYTE EP2GPIFTRIG       _AT_ 0xE6D4;  // EP2 FIFO Trigger
// EXTERN xdata volatile BYTE EP4GPIFTCH     _AT_ 0xE6D8;  // EP4 GPIF Transaction Count High
// EXTERN xdata volatile BYTE EP4GPIFTCL     _AT_ 0xE6D9;  // EP4 GPIF Transactionr Count Low
EXTERN xdata volatile BYTE EP4GPIFFLGSEL     _AT_ 0xE6DA;  // EP4 GPIF Flag select
EXTERN xdata volatile BYTE EP4GPIFPFSTOP     _AT_ 0xE6DB;  // Stop GPIF EP4 transaction on prog. flag
EXTERN xdata volatile BYTE EP4GPIFTRIG       _AT_ 0xE6DC;  // EP4 FIFO Trigger
// EXTERN xdata volatile BYTE EP6GPIFTCH     _AT_ 0xE6E0;  // EP6 GPIF Transaction Count High
// EXTERN xdata volatile BYTE EP6GPIFTCL     _AT_ 0xE6E1;  // EP6 GPIF Transaction Count Low
EXTERN xdata volatile BYTE EP6GPIFFLGSEL     _AT_ 0xE6E2;  // EP6 GPIF Flag select
EXTERN xdata volatile BYTE EP6GPIFPFSTOP     _AT_ 0xE6E3;  // Stop GPIF EP6 transaction on prog. flag
EXTERN xdata volatile BYTE EP6GPIFTRIG       _AT_ 0xE6E4;  // EP6 FIFO Trigger
// EXTERN xdata volatile BYTE EP8GPIFTCH     _AT_ 0xE6E8;  // EP8 GPIF Transaction Count High
// EXTERN xdata volatile BYTE EP8GPIFTCL     _AT_ 0xE6E9;  // EP8GPIF Transaction Count Low
EXTERN xdata volatile BYTE EP8GPIFFLGSEL     _AT_ 0xE6EA;  // EP8 GPIF Flag select
EXTERN xdata volatile BYTE EP8GPIFPFSTOP     _AT_ 0xE6EB;  // Stop GPIF EP8 transaction on prog. flag
EXTERN xdata volatile BYTE EP8GPIFTRIG       _AT_ 0xE6EC;  // EP8 FIFO Trigger
EXTERN xdata volatile BYTE XGPIFSGLDATH      _AT_ 0xE6F0;  // GPIF Data H (16-bit mode only)
EXTERN xdata volatile BYTE XGPIFSGLDATLX     _AT_ 0xE6F1;  // Read/Write GPIF Data L & trigger transac
EXTERN xdata volatile BYTE XGPIFSGLDATLNOX   _AT_ 0xE6F2;  // Read GPIF Data L, no transac trigger
EXTERN xdata volatile BYTE GPIFREADYCFG      _AT_ 0xE6F3;  // Internal RDY,Sync/Async, RDY5CFG
EXTERN xdata volatile BYTE GPIFREADYSTAT     _AT_ 0xE6F4;  // RDY pin states
EXTERN xdata volatile BYTE GPIFABORT         _AT_ 0xE6F5;  // Abort GPIF cycles

// UDMA

EXTERN xdata volatile BYTE FLOWSTATE         _AT_  0xE6C6; //Defines GPIF flow state
```

```
EXTERN xdata volatile BYTE FLOWLOGIC         _AT_  0xE6C7; //Defines flow/hold decision criteria
EXTERN xdata volatile BYTE FLOWEQ0CTL        _AT_  0xE6C8; //CTL states during active flow state
EXTERN xdata volatile BYTE FLOWEQ1CTL        _AT_  0xE6C9; //CTL states during hold flow state
EXTERN xdata volatile BYTE FLOWHOLDOFF       _AT_  0xE6CA;
EXTERN xdata volatile BYTE FLOWSTB           _AT_  0xE6CB; //CTL/RDY Signal to use as master data strobe
EXTERN xdata volatile BYTE FLOWSTBEDGE       _AT_  0xE6CC; //Defines active master strobe edge
EXTERN xdata volatile BYTE FLOWSTBHPERIOD    _AT_  0xE6CD; //Half Period of output master strobe
EXTERN xdata volatile BYTE GPIFHOLDAMOUNT    _AT_  0xE60C; //Data delay shift
EXTERN xdata volatile BYTE UDMACRCH          _AT_  0xE67D; //CRC Upper byte
EXTERN xdata volatile BYTE UDMACRCL          _AT_  0xE67E; //CRC Lower byte
EXTERN xdata volatile BYTE UDMACRCQUAL       _AT_  0xE67F; //UDMA In only, host terminated use only


// Debug/Test

EXTERN xdata volatile BYTE DBUG           _AT_ 0xE6F8;  // Debug
EXTERN xdata volatile BYTE TESTCFG        _AT_ 0xE6F9;  // Test configuration
EXTERN xdata volatile BYTE USBTEST        _AT_ 0xE6FA;  // USB Test Modes
EXTERN xdata volatile BYTE CT1            _AT_ 0xE6FB;  // Chirp Test--Override
EXTERN xdata volatile BYTE CT2            _AT_ 0xE6FC;  // Chirp Test--FSM
EXTERN xdata volatile BYTE CT3            _AT_ 0xE6FD;  // Chirp Test--Control Signals
EXTERN xdata volatile BYTE CT4            _AT_ 0xE6FE;  // Chirp Test--Inputs

// Endpoint Buffers

EXTERN xdata volatile BYTE EP0BUF[64]        _AT_ 0xE740;  // EP0 IN-OUT buffer
EXTERN xdata volatile BYTE EP1OUTBUF[64]     _AT_ 0xE780;  // EP1-OUT buffer
EXTERN xdata volatile BYTE EP1INBUF[64]      _AT_ 0xE7C0;  // EP1-IN buffer
EXTERN xdata volatile BYTE EP2FIFOBUF[1024]  _AT_ 0xF000;  // 512/1024-byte EP2 buffer (IN or OUT)
EXTERN xdata volatile BYTE EP4FIFOBUF[1024]  _AT_ 0xF400;  // 512 byte EP4 buffer (IN or OUT)
EXTERN xdata volatile BYTE EP6FIFOBUF[1024]  _AT_ 0xF800;  // 512/1024-byte EP6 buffer (IN or OUT)
EXTERN xdata volatile BYTE EP8FIFOBUF[1024]  _AT_ 0xFC00;  // 512 byte EP8 buffer (IN or OUT)

#undef EXTERN
#undef _AT_

/*-----------------------------------------------------------------------------
  Special Function Registers (SFRs)
  The byte registers and bits defined in the following list are based
  on the Synopsis definition of the 8051 Special Function Registers for EZ-USB.
   If you modify the register definitions below, please regenerate the file
   "ezregs.inc" which includes the same basic information for assembly inclusion.
-----------------------------------------------------------------------------*/

sfr IOA     = 0x80;
sfr SP      = 0x81;
sfr DPL     = 0x82;
sfr DPH     = 0x83;
sfr DPL1    = 0x84;
sfr DPH1    = 0x85;
sfr DPS     = 0x86;
        /*  DPS  */
        sbit SEL   = 0x86+0;
sfr PCON    = 0x87;   /*  PCON  */
        //sbit IDLE   = 0x87+0;
        //sbit STOP   = 0x87+1;
        //sbit GF0    = 0x87+2;
        //sbit GF1    = 0x87+3;
        //sbit SMOD0  = 0x87+7;
sfr TCON    = 0x88;
        /*  TCON  */
        sbit IT0    = 0x88+0;
        sbit IE0    = 0x88+1;
        sbit IT1    = 0x88+2;
        sbit IE1    = 0x88+3;
        sbit TR0    = 0x88+4;
        sbit TF0    = 0x88+5;
        sbit TR1    = 0x88+6;
        sbit TF1    = 0x88+7;
sfr TMOD    = 0x89;
```

```
        /*  TMOD  */
        //sbit M00    = 0x89+0;
        //sbit M10    = 0x89+1;
        //sbit CT0    = 0x89+2;
        //sbit GATE0  = 0x89+3;
        //sbit M01    = 0x89+4;
        //sbit M11    = 0x89+5;
        //sbit CT1    = 0x89+6;
        //sbit GATE1  = 0x89+7;
sfr TL0     = 0x8A;
sfr TL1     = 0x8B;
sfr TH0     = 0x8C;
sfr TH1     = 0x8D;
sfr CKCON   = 0x8E;
        /*  CKCON  */
        //sbit MD0    = 0x89+0;
        //sbit MD1    = 0x89+1;
        //sbit MD2    = 0x89+2;
        //sbit T0M    = 0x89+3;
        //sbit T1M    = 0x89+4;
        //sbit T2M    = 0x89+5;
sfr SPC_FNC = 0x8F; // Was WRS in Reg320
        /*  CKCON  */
        //sbit WRS    = 0x8F+0;
sfr IOB     = 0x90;
sfr EXIF    = 0x91; // EXIF Bit Values differ from Reg320
        /*  EXIF  */
        //sbit USBINT = 0x91+4;
        //sbit I2CINT = 0x91+5;
        //sbit IE4    = 0x91+6;
        //sbit IE5    = 0x91+7;
sfr MPAGE  = 0x92;
sfr SCON0  = 0x98;
        /*  SCON0  */
        sbit RI    = 0x98+0;
        sbit TI    = 0x98+1;
        sbit RB8   = 0x98+2;
        sbit TB8   = 0x98+3;
        sbit REN   = 0x98+4;
        sbit SM2   = 0x98+5;
        sbit SM1   = 0x98+6;
        sbit SM0   = 0x98+7;
sfr SBUF0  = 0x99;

sfr APTR1H     = 0x9A; // old name
sfr APTR1L     = 0x9B; // old name
sfr AUTOPTR1H     = 0x9A;
sfr AUTOPTR1L     = 0x9B;
sfr AUTOPTRH2     = 0x9D;
sfr AUTOPTRL2     = 0x9E;
sfr IOC        = 0xA0;
sfr INT2CLR    = 0xA1;
sfr INT4CLR    = 0xA2;

sfr IE     = 0xA8;
        /*  IE  */
        sbit EX0   = 0xA8+0;
        sbit ET0   = 0xA8+1;
        sbit EX1   = 0xA8+2;
        sbit ET1   = 0xA8+3;
        sbit ES0   = 0xA8+4;
        sbit ET2   = 0xA8+5;
        sbit ES1   = 0xA8+6;
        sbit EA    = 0xA8+7;

sfr EP2468STAT     = 0xAA;
        /* EP2468STAT */
        //sbit EP2E   = 0xAA+0;
        //sbit EP2F   = 0xAA+1;
        //sbit EP4E   = 0xAA+2;
```

```
        //sbit EP4F   = 0xAA+3;
        //sbit EP6E   = 0xAA+4;
        //sbit EP6F   = 0xAA+5;
        //sbit EP8E   = 0xAA+6;
        //sbit EP8F   = 0xAA+7;

sfr EP24FIFOFLGS   = 0xAB;
sfr EP68FIFOFLGS   = 0xAC;
sfr AUTOPTRSETUP  = 0xAF;
          /* AUTOPTRSETUP */
          sbit EXTACC  = 0xAF+0;
          sbit APTR1FZ = 0xAF+1;
          sbit APTR2FZ = 0xAF+2;

sfr IOD     = 0xB0;
sfr IOE     = 0xB1;
sfr OEA     = 0xB2;
sfr OEB     = 0xB3;
sfr OEC     = 0xB4;
sfr OED     = 0xB5;
sfr OEE     = 0xB6;

sfr IP     = 0xB8;
        /*  IP  */
        sbit PX0   = 0xB8+0;
        sbit PT0   = 0xB8+1;
        sbit PX1   = 0xB8+2;
        sbit PT1   = 0xB8+3;
        sbit PS0   = 0xB8+4;
        sbit PT2   = 0xB8+5;
        sbit PS1   = 0xB8+6;

sfr EP01STAT    = 0xBA;
sfr GPIFTRIG    = 0xBB;

sfr GPIFSGLDATH     = 0xBD;
sfr GPIFSGLDATLX    = 0xBE;
sfr GPIFSGLDATLNOX  = 0xBF;

sfr SCON1  = 0xC0;
        /*  SCON1  */
        sbit RI1   = 0xC0+0;
        sbit TI1   = 0xC0+1;
        sbit RB81  = 0xC0+2;
        sbit TB81  = 0xC0+3;
        sbit REN1  = 0xC0+4;
        sbit SM21  = 0xC0+5;
        sbit SM11  = 0xC0+6;
        sbit SM01  = 0xC0+7;
sfr SBUF1  = 0xC1;
sfr T2CON  = 0xC8;
        /*  T2CON  */
        sbit CP_RL2 = 0xC8+0;
        sbit C_T2  = 0xC8+1;
        sbit TR2   = 0xC8+2;
        sbit EXEN2 = 0xC8+3;
        sbit TCLK  = 0xC8+4;
        sbit RCLK  = 0xC8+5;
        sbit EXF2  = 0xC8+6;
        sbit TF2   = 0xC8+7;
sfr RCAP2L = 0xCA;
sfr RCAP2H = 0xCB;
sfr TL2    = 0xCC;
sfr TH2    = 0xCD;
sfr PSW    = 0xD0;
        /*  PSW  */
        sbit P     = 0xD0+0;
        sbit FL    = 0xD0+1;
        sbit OV    = 0xD0+2;
        sbit RS0   = 0xD0+3;
```

```
        sbit RS1   = 0xD0+4;
        sbit F0    = 0xD0+5;
        sbit AC    = 0xD0+6;
        sbit CY    = 0xD0+7;
sfr EICON  = 0xD8; // Was WDCON in DS80C320; Bit Values differ from Reg320
        /*  EICON  */
        sbit INT6  = 0xD8+3;
        sbit RESI  = 0xD8+4;
        sbit ERESI = 0xD8+5;
        sbit SMOD1 = 0xD8+7;
sfr ACC    = 0xE0;
sfr EIE    = 0xE8; // EIE Bit Values differ from Reg320
                /*  EIE  */
        sbit EUSB    = 0xE8+0;
        sbit EI2C    = 0xE8+1;
        sbit EIEX4   = 0xE8+2;
        sbit EIEX5   = 0xE8+3;
        sbit EIEX6   = 0xE8+4;
sfr B      = 0xF0;
sfr EIP    = 0xF8; // EIP Bit Values differ from Reg320
                /*  EIP  */
        sbit PUSB    = 0xF8+0;
        sbit PI2C    = 0xF8+1;
        sbit EIPX4   = 0xF8+2;
        sbit EIPX5   = 0xF8+3;
        sbit EIPX6   = 0xF8+4;

/*-----------------------------------------------------------------------------
   Bit Masks
-----------------------------------------------------------------------------*/

/* CPU Control & Status Register (CPUCS) */
#define bmPRTCSTB    bmBIT5
#define bmCLKSPD     (bmBIT4 | bmBIT3)
#define bmCLKSPD1    bmBIT4
#define bmCLKSPD0    bmBIT3
#define bmCLKINV     bmBIT2
#define bmCLKOE      bmBIT1
#define bm8051RES    bmBIT0
/* Port Alternate Configuration Registers */
/* Port A (PORTACFG) */
#define bmFLAGD      bmBIT7
#define bmINT1       bmBIT1
#define bmINT0       bmBIT0
/* Port C (PORTCCFG) */
#define bmGPIFA7     bmBIT7
#define bmGPIFA6     bmBIT6
#define bmGPIFA5     bmBIT5
#define bmGPIFA4     bmBIT4
#define bmGPIFA3     bmBIT3
#define bmGPIFA2     bmBIT2
#define bmGPIFA1     bmBIT1
#define bmGPIFA0     bmBIT0
/* Port E (PORTECFG) */
#define bmGPIFA8     bmBIT7
#define bmT2EX       bmBIT6
#define bmINT6       bmBIT5
#define bmRXD1OUT    bmBIT4
#define bmRXD0OUT    bmBIT3
#define bmT2OUT      bmBIT2
#define bmT1OUT      bmBIT1
#define bmT0OUT      bmBIT0

/* I2C Control & Status Register (I2CS) */
#define bmSTART      bmBIT7
#define bmSTOP       bmBIT6
#define bmLASTRD     bmBIT5
#define bmID         (bmBIT4 | bmBIT3)
#define bmBERR       bmBIT2
#define bmACK        bmBIT1
```

```
#define bmDONE       bmBIT0
/* I2C Control Register (I2CTL) */
#define bmSTOPIE     bmBIT1
#define bm400KHZ     bmBIT0
/* Interrupt 2 (USB) Autovector Register (INT2IVEC) */
#define bmIV4        bmBIT6
#define bmIV3        bmBIT5
#define bmIV2        bmBIT4
#define bmIV1        bmBIT3
#define bmIV0        bmBIT2
/* USB Interrupt Request & Enable Registers (USBIE/USBIRQ) */
#define bmEP0ACK     bmBIT6
#define bmHSGRANT    bmBIT5
#define bmURES       bmBIT4
#define bmSUSP       bmBIT3
#define bmSUTOK      bmBIT2
#define bmSOF        bmBIT1
#define bmSUDAV      bmBIT0
/* Breakpoint register (BREAKPT) */
#define bmBREAK      bmBIT3
#define bmBPPULSE    bmBIT2
#define bmBPEN       bmBIT1
/* Interrupt 2 & 4 Setup (INTSETUP) */
#define bmAV2EN      bmBIT3
#define INT4IN       bmBIT1
#define bmAV4EN      bmBIT0
/* USB Control & Status Register (USBCS) */
#define bmHSM        bmBIT7
#define bmDISCON     bmBIT3
#define bmNOSYNSOF   bmBIT2
#define bmRENUM      bmBIT1
#define bmSIGRESUME  bmBIT0
/* Wakeup Control and Status Register (WAKEUPCS) */
#define bmWU2        bmBIT7
#define bmWU         bmBIT6
#define bmWU2POL     bmBIT5
#define bmWUPOL      bmBIT4
#define bmDPEN       bmBIT2
#define bmWU2EN      bmBIT1
#define bmWUEN       bmBIT0
/* End Point 0 Control & Status Register (EP0CS) */
#define bmHSNAK      bmBIT7
/* End Point 0-1 Control & Status Registers (EP0CS/EP1OUTCS/EP1INCS) */
#define bmEPBUSY     bmBIT1
#define bmEPSTALL    bmBIT0
/* End Point 2-8 Control & Status Registers (EP2CS/EP4CS/EP6CS/EP8CS) */
#define bmNPAK       (bmBIT6 | bmBIT5 | bmBIT4)
#define bmEPFULL     bmBIT3
#define bmEPEMPTY    bmBIT2
/* Endpoint Status (EP2468STAT) SFR bits */
#define bmEP8FULL    bmBIT7
#define bmEP8EMPTY   bmBIT6
#define bmEP6FULL    bmBIT5
#define bmEP6EMPTY   bmBIT4
#define bmEP4FULL    bmBIT3
#define bmEP4EMPTY   bmBIT2
#define bmEP2FULL    bmBIT1
#define bmEP2EMPTY   bmBIT0
/* SETUP Data Pointer Auto Mode (SUDPTRCTL) */
#define bmSDPAUTO    bmBIT0
/* Endpoint Data Toggle Control (TOGCTL) */
#define bmQUERYTOGGLE  bmBIT7
#define bmSETTOGGLE    bmBIT6
#define bmRESETTOGGLE  bmBIT5
#define bmTOGCTLEPMASK bmBIT3 | bmBIT2 | bmBIT1 | bmBIT0
/* IBN (In Bulk Nak) enable and request bits (IBNIE/IBNIRQ) */
#define bmEP8IBN     bmBIT5
#define bmEP6IBN     bmBIT4
#define bmEP4IBN     bmBIT3
#define bmEP2IBN     bmBIT2
```

```
#define bmEP1IBN     bmBIT1
#define bmEP0IBN     bmBIT0

/* PING-NAK enable and request bits (NAKIE/NAKIRQ) */
#define bmEP8PING     bmBIT7
#define bmEP6PING     bmBIT6
#define bmEP4PING     bmBIT5
#define bmEP2PING     bmBIT4
#define bmEP1PING     bmBIT3
#define bmEP0PING     bmBIT2
#define bmIBN         bmBIT0

/* Interface Configuration bits (IFCONFIG) */
#define bmIFCLKSRC    bmBIT7
#define bm3048MHZ     bmBIT6
#define bmIFCLKOE     bmBIT5
#define bmIFCLKPOL    bmBIT4
#define bmASYNC       bmBIT3
#define bmGSTATE      bmBIT2
#define bmIFCFG1      bmBIT1
#define bmIFCFG0      bmBIT0
#define bmIFCFGMASK   (bmIFCFG0 | bmIFCFG1)
#define bmIFGPIF      bmIFCFG1

/* EP 2468 FIFO Configuration bits (EP2FIFOCFG,EP4FIFOCFG,EP6FIFOCFG,EP8FIFOCFG) */
#define bmINFM       bmBIT6
#define bmOEP        bmBIT5
#define bmAUTOOUT    bmBIT4
#define bmAUTOIN     bmBIT3
#define bmZEROLENIN  bmBIT2
#define bmWORDWIDE   bmBIT0

/* Chip Revision Control Bits (REVCTL) - used to ebable/disable revision specidic
   features */
#define bmNOAUTOARM    bmBIT1
#define bmSKIPCOMMIT   bmBIT0

/* Fifo Reset bits (FIFORESET) */
#define bmNAKALL       bmBIT7

#endif   /* FX2REGS_H */
```

C.3 fw.c

```
//-----------------------------------------------------------------------------
//   File:      fw.c
//   Contents:   Firmware frameworks task dispatcher and device request parser
//            source.
//
// indent 3.  NO TABS!
//
// $Revision: 18 $
// $Date: 12/04/01 5:33p $
//
//   Copyright (c) 1997 AnchorChips, Inc. All rights reserved
//-----------------------------------------------------------------------------
#include "fx2.h"
#include "fx2regs.h"

//-----------------------------------------------------------------------------
// Constants
//-----------------------------------------------------------------------------
#define DELAY_COUNT   0x9248*8L  // Delay for 8 sec at 24Mhz, 4 sec at 48
#define _IFREQ  48000            // IFCLK constant for Synchronization Delay
#define _CFREQ  48000            // CLKOUT constant for Synchronization Delay
```

```
//-----------------------------------------------------------------------------
// Random Macros
//-----------------------------------------------------------------------------
#define   min(a,b) (((a)<(b))?(a):(b))
#define   max(a,b) (((a)>(b))?(a):(b))

  // Registers which require a synchronization delay, see section 15.14
  // FIFORESET        FIFOPINPOLAR
  // INPKTEND         OUTPKTEND
  // EPxBCH:L         REVCTL
  // GPIFTCB3         GPIFTCB2
  // GPIFTCB1         GPIFTCB0
  // EPxFIFOPFH:L     EPxAUTOINLENH:L
  // EPxFIFOCFG       EPxGPIFFLGSEL
  // PINFLAGSxx       EPxFIFOIRQ
  // EPxFIFOIE        GPIFIRQ
  // GPIFIE           GPIFADRH:L
  // UDMACRCH:L       EPxGPIFTRIG
  // GPIFTRIG

  // Note: The pre-REVE EPxGPIFTCH/L register are affected, as well...
  //      ...these have been replaced by GPIFTC[B3:B0] registers

#include "fx2sdly.h"             // Define _IFREQ and _CFREQ above this #include

//-----------------------------------------------------------------------------
// Global Variables
//-----------------------------------------------------------------------------
volatile BOOL   GotSUD;
BOOL      Rwuen;
BOOL      Selfpwr;
volatile BOOL   Sleep;                  // Sleep mode enable flag

WORD   pDeviceDscr;   // Pointer to Device Descriptor; Descriptors may be moved
WORD   pDeviceQualDscr;
WORD   pHighSpeedConfigDscr;
WORD   pFullSpeedConfigDscr;
WORD   pConfigDscr;
WORD   pOtherConfigDscr;
WORD   pStringDscr;

//-----------------------------------------------------------------------------
// Prototypes
//-----------------------------------------------------------------------------
void SetupCommand(void);
void TD_Init(void);
void TD_Poll(void);
BOOL TD_Suspend(void);
BOOL TD_Resume(void);

BOOL DR_GetDescriptor(void);
BOOL DR_SetConfiguration(void);
BOOL DR_GetConfiguration(void);
BOOL DR_SetInterface(void);
BOOL DR_GetInterface(void);
BOOL DR_GetStatus(void);
BOOL DR_ClearFeature(void);
BOOL DR_SetFeature(void);
BOOL DR_VendorCmnd(void);

// this table is used by the epcs macro
const char code  EPCS_Offset_Lookup_Table[] =
{
   0,    // EP1OUT
   1,    // EP1IN
   2,    // EP2OUT
   2,    // EP2IN
   3,    // EP4OUT
   3,    // EP4IN
   4,    // EP6OUT
```

```
   4,   // EP6IN
   5,   // EP8OUT
   5,   // EP8IN
};

// macro for generating the address of an endpoint's control and status register (EPnCS)
#define epcs(EP) (EPCS_Offset_Lookup_Table[(EP & 0x7E) | (EP > 128)] + 0xE6A1)

//-----------------------------------------------------------------------------
// Code
//-----------------------------------------------------------------------------

// Task dispatcher
void main(void)
{
   DWORD   i;
   WORD   offset;
   DWORD   DevDescrLen;
   DWORD   j=0;
   WORD   IntDescrAddr;
   WORD   ExtDescrAddr;

   // Initialize Global States
   Sleep = FALSE;               // Disable sleep mode
   Rwuen = FALSE;               // Disable remote wakeup
   Selfpwr = FALSE;             // Disable self powered
   GotSUD = FALSE;              // Clear "Got setup data" flag

   // Initialize user device
   TD_Init();

   // The following section of code is used to relocate the descriptor table.
   // Since the SUDPTRH and SUDPTRL are assigned the address of the descriptor
   // table, the descriptor table must be located in on-part memory.
   // The 4K demo tools locate all code sections in external memory.
   // The descriptor table is relocated by the frameworks ONLY if it is found
   // to be located in external memory.
   pDeviceDscr = (WORD)&DeviceDscr;
   pDeviceQualDscr = (WORD)&DeviceQualDscr;
   pHighSpeedConfigDscr = (WORD)&HighSpeedConfigDscr;
   pFullSpeedConfigDscr = (WORD)&FullSpeedConfigDscr;
   pStringDscr = (WORD)&StringDscr;

   if ((WORD)&DeviceDscr & 0xe000)
   {
      IntDescrAddr = INTERNAL_DSCR_ADDR;
      ExtDescrAddr = (WORD)&DeviceDscr;
      DevDescrLen = (WORD)&UserDscr - (WORD)&DeviceDscr + 2;
      for (i = 0; i < DevDescrLen; i++)
         *((BYTE xdata *)IntDescrAddr+i) = 0xCD;
      for (i = 0; i < DevDescrLen; i++)
         *((BYTE xdata *)IntDescrAddr+i) = *((BYTE xdata *)ExtDescrAddr+i);
      pDeviceDscr = IntDescrAddr;
      offset = (WORD)&DeviceDscr - INTERNAL_DSCR_ADDR;
      pDeviceQualDscr -= offset;
      pConfigDscr -= offset;
      pOtherConfigDscr -= offset;
      pHighSpeedConfigDscr -= offset;
      pFullSpeedConfigDscr -= offset;
      pStringDscr -= offset;
   }

   EZUSB_IRQ_ENABLE();            // Enable USB interrupt (INT2)
   EZUSB_ENABLE_RSMIRQ();            // Wake-up interrupt

   INTSETUP |= (bmAV2EN | bmAV4EN);     // Enable INT 2 & 4 autovectoring

   USBIE |= bmSUDAV | bmSUTOK | bmSUSP | bmURES | bmHSGRANT;   // Enable selected interrupts
   EA = 1;                  // Enable 8051 interrupts
```

```
#ifndef NO_RENUM
   // Renumerate if necessary.  Do this by checking the renum bit.  If it
   // is already set, there is no need to renumerate.  The renum bit will
   // already be set if this firmware was loaded from an eeprom.
   if(!(USBCS & bmRENUM))
   {
       EZUSB_Discon(TRUE);   // renumerate
   }
#endif

   // unconditionally re-connect.  If we loaded from eeprom we are
   // disconnected and need to connect.  If we just renumerated this
   // is not necessary but doesn't hurt anything
   USBCS &=~bmDISCON;

   CKCON = (CKCON&(~bmSTRETCH)) | FW_STRETCH_VALUE; // Set stretch to 0 (after renumeration)

   // clear the Sleep flag.
   Sleep = FALSE;

   // Task Dispatcher
   while(TRUE)               // Main Loop
   {
      if(GotSUD)            // Wait for SUDAV
      {
         SetupCommand();          // Implement setup command
           GotSUD = FALSE;            // Clear SUDAV flag
      }

      // Poll User Device
      // NOTE: Idle mode stops the processor clock.  There are only two
      // ways out of idle mode, the WAKEUP pin, and detection of the USB
      // resume state on the USB bus.  The timers will stop and the
      // processor will not wake up on any other interrupts.
      if (Sleep)
          {
          if(TD_Suspend())
              {
              Sleep = FALSE;            // Clear the "go to sleep" flag.  Do it here to prevent any race condition between wakeup and the next sleep.
              do
                  {
                    EZUSB_Susp();         // Place processor in idle mode.
                  }
                while(!Rwuen && EZUSB_EXTWAKEUP());
                // Must continue to go back into suspend if the host has disabled remote wakeup
                // *and* the wakeup was caused by the external wakeup pin.

             // 8051 activity will resume here due to USB bus or Wakeup# pin activity.
             EZUSB_Resume();   // If source is the Wakeup# pin, signal the host to Resume.
             TD_Resume();
              }
          }
      TD_Poll();
   }
}

// Device request parser
void SetupCommand(void)
{
   void   *dscr_ptr;

   switch(SETUPDAT[1])
   {
      case SC_GET_DESCRIPTOR:                  // *** Get Descriptor
         if(DR_GetDescriptor())
            switch(SETUPDAT[3])
            {
               case GD_DEVICE:            // Device
                  SUDPTRH = MSB(pDeviceDscr);
                  SUDPTRL = LSB(pDeviceDscr);
```

```
          break;
        case GD_DEVICE_QUALIFIER:           // Device Qualifier
          SUDPTRH = MSB(pDeviceQualDscr);
          SUDPTRL = LSB(pDeviceQualDscr);
          break;
        case GD_CONFIGURATION:         // Configuration
          SUDPTRH = MSB(pConfigDscr);
          SUDPTRL = LSB(pConfigDscr);
          break;
        case GD_OTHER_SPEED_CONFIGURATION:  // Other Speed Configuration
          SUDPTRH = MSB(pOtherConfigDscr);
          SUDPTRL = LSB(pOtherConfigDscr);
          break;
        case GD_STRING:            // String
          if(dscr_ptr = (void *)EZUSB_GetStringDscr(SETUPDAT[2]))
          {
            SUDPTRH = MSB(dscr_ptr);
            SUDPTRL = LSB(dscr_ptr);
          }
          else
            EZUSB_STALL_EP0();   // Stall End Point 0
          break;
        default:            // Invalid request
          EZUSB_STALL_EP0();      // Stall End Point 0
      }
    break;
  case SC_GET_INTERFACE:                  // *** Get Interface
    DR_GetInterface();
    break;
  case SC_SET_INTERFACE:                  // *** Set Interface
    DR_SetInterface();
    break;
  case SC_SET_CONFIGURATION:               // *** Set Configuration
    DR_SetConfiguration();
    break;
  case SC_GET_CONFIGURATION:               // *** Get Configuration
    DR_GetConfiguration();
    break;
  case SC_GET_STATUS:                  // *** Get Status
    if(DR_GetStatus())
      switch(SETUPDAT[0])
      {
        case GS_DEVICE:            // Device
          EP0BUF[0] = ((BYTE)Rwuen << 1) | (BYTE)Selfpwr;
          EP0BUF[1] = 0;
          EP0BCH = 0;
          EP0BCL = 2;
          break;
        case GS_INTERFACE:         // Interface
          EP0BUF[0] = 0;
          EP0BUF[1] = 0;
          EP0BCH = 0;
          EP0BCL = 2;
          break;
        case GS_ENDPOINT:         // End Point
          EP0BUF[0] = *(BYTE xdata *) epcs(SETUPDAT[4]) & bmEPSTALL;
          EP0BUF[1] = 0;
          EP0BCH = 0;
          EP0BCL = 2;
          break;
        default:            // Invalid Command
          EZUSB_STALL_EP0();      // Stall End Point 0
      }
    break;
  case SC_CLEAR_FEATURE:                  // *** Clear Feature
    if(DR_ClearFeature())
      switch(SETUPDAT[0])
      {
        case FT_DEVICE:            // Device
          if(SETUPDAT[2] == 1)
```

```
              Rwuen = FALSE;       // Disable Remote Wakeup
            else
              EZUSB_STALL_EP0();   // Stall End Point 0
            break;
          case FT_ENDPOINT:        // End Point
            if(SETUPDAT[2] == 0)
            {
              *(BYTE xdata *) epcs(SETUPDAT[4]) &= ~bmEPSTALL;
              EZUSB_RESET_DATA_TOGGLE( SETUPDAT[4] );
            }
            else
              EZUSB_STALL_EP0();   // Stall End Point 0
            break;
        }
      break;
    case SC_SET_FEATURE:                 // *** Set Feature
      if(DR_SetFeature())
        switch(SETUPDAT[0])
        {
          case FT_DEVICE:           // Device
            if(SETUPDAT[2] == 1)
              Rwuen = TRUE;      // Enable Remote Wakeup
            else if(SETUPDAT[2] == 2)
              // Set Feature Test Mode.  The core handles this request.  However, it is
              // necessary for the firmware to complete the handshake phase of the
              // control transfer before the chip will enter test mode.  It is also
              // necessary for FX2 to be physically disconnected (D+ and D-)
              // from the host before it will enter test mode.
              break;
            else
              EZUSB_STALL_EP0();   // Stall End Point 0
            break;
          case FT_ENDPOINT:        // End Point
            *(BYTE xdata *) epcs(SETUPDAT[4]) |= bmEPSTALL;
            break;
        }
      break;
    default:                    // *** Invalid Command
      if(DR_VendorCmnd())
        EZUSB_STALL_EP0();           // Stall End Point 0
  }

  // Acknowledge handshake phase of device request
  EP0CS |= bmHSNAK;
}

// Wake-up interrupt handler
void resume_isr(void) interrupt WKUP_VECT
{
  EZUSB_CLEAR_RSMIRQ();
}
```

C.4 DSCR.A51

```
;;-----------------------------------------------------------------------------
;;   File:      dscr.a51
;;   Contents:   This file contains descriptor data tables.
;;
;;   Copyright (c) 1997 AnchorChips, Inc. All rights reserved
;;-----------------------------------------------------------------------------

DSCR_DEVICE   equu   1   ;; Descriptor type: Device
DSCR_CONFIG   equ   2   ;; Descriptor type: Configuration
DSCR_STRING   equ   3   ;; Descriptor type: String
DSCR_INTRFC   equ   4   ;; Descriptor type: Interface
DSCR_ENDPNT   equ   5   ;; Descriptor type: Endpoint
```

```
DSCR_DEVQUAL  equ   6   ;; Descriptor type: Device Qualifier

DSCR_DEVICE_LEN   equ   18
DSCR_CONFIG_LEN   equ   9
DSCR_INTRFC_LEN   equ   9
DSCR_ENDPNT_LEN   equ   7
DSCR_DEVQUAL_LEN  equ   10

ET_CONTROL   equ   0   ;; Endpoint type: Control
ET_ISO       equ   1   ;; Endpoint type: Isochronous
ET_BULK      equ   2   ;; Endpoint type: Bulk
ET_INT       equ   3   ;; Endpoint type: Interrupt

public      DeviceDscr, DeviceQualDscr, HighSpeedConfigDscr, FullSpeedConfigDscr, StringDscr, UserDscr

DSCR   SEGMENT   CODE

;;-----------------------------------------------------------------------------
;; Global Variables
;;-----------------------------------------------------------------------------
    rseg DSCR      ;; locate the descriptor table in on-part memory.

DeviceDscr:
    db   DSCR_DEVICE_LEN      ;; Descriptor length
    db   DSCR_DEVICE   ;; Decriptor type
    dw   0002H      ;; Specification Version (BCD)
    db   00H       ;; Device class
    db   00H        ;; Device sub-class
    db   00H        ;; Device sub-sub-class
    db   64        ;; Maximum packet size

;;-----------------------------------------------------------------------------
;; THIS NEEDS TO BE CHANGED (Michal, 16.12.2003)
;;-----------------------------------------------------------------------------
    dw   5604H      ;; Vendor ID
    dw   06B0H      ;; Product ID (Sample Device)
;;-----------------------------------------------------------------------------

    dw   0000H      ;; Product version ID
    db   1        ;; Manufacturer string index
    db   2        ;; Product string index
    db   0        ;; Serial number string index
    db   1        ;; Number of configurations

DeviceQualDscr:
    db   DSCR_DEVQUAL_LEN   ;; Descriptor length
    db   DSCR_DEVQUAL   ;; Decriptor type
    dw   0002H      ;; Specification Version (BCD)
    db   00H       ;; Device class
    db   00H        ;; Device sub-class
    db   00H        ;; Device sub-sub-class
    db   64        ;; Maximum packet size
    db   1        ;; Number of configurations
    db   0        ;; Reserved

HighSpeedConfigDscr:
    db   DSCR_CONFIG_LEN              ;; Descriptor length
    db   DSCR_CONFIG                ;; Descriptor type
    db   (HighSpeedConfigDscrEnd-HighSpeedConfigDscr) mod 256 ;; Total Length (LSB)
    db   (HighSpeedConfigDscrEnd-HighSpeedConfigDscr)  /  256 ;; Total Length (MSB)
    db   1      ;; Number of interfaces
    db   1      ;; Configuration number
    db   0      ;; Configuration string
    db   10100000b   ;; Attributes (b7 - buspwr, b6 - selfpwr, b5 - rwu)
    db   50      ;; Power requirement (div 2 ma)

;; Interface Descriptor
    db   DSCR_INTRFC_LEN      ;; Descriptor length
    db   DSCR_INTRFC        ;; Descriptor type
    db   0             ;; Zero-based index of this interface
```

```
    db  0         ;; Alternate setting
    db  2         ;; Number of end points
    db  0ffH      ;; Interface class
    db  00H       ;; Interface sub class
    db  00H       ;; Interface sub sub class
    db  0         ;; Interface descriptor string index

;; Endpoint Descriptor
    db  DSCR_ENDPNT_LEN    ;; Descriptor length
    db  DSCR_ENDPNT        ;; Descriptor type
    db  02H       ;; Endpoint number, and direction
    db  ET_BULK        ;; Endpoint type
    db  00H       ;; Maximun packet size (LSB)
    db  02H       ;; Max packect size (MSB)
    db  00H       ;; Polling interval

;; Endpoint Descriptor
    db  DSCR_ENDPNT_LEN    ;; Descriptor length
    db  DSCR_ENDPNT        ;; Descriptor type
    db  86H       ;; Endpoint number, and direction
    db  ET_BULK        ;; Endpoint type
    db  00H       ;; Maximun packet size (LSB)
    db  02H       ;; Max packect size (MSB)
    db  00H       ;; Polling interval

HighSpeedConfigDscrEnd:

FullSpeedConfigDscr:
    db  DSCR_CONFIG_LEN        ;; Descriptor length
    db  DSCR_CONFIG            ;; Descriptor type
    db  (FullSpeedConfigDscrEnd-FullSpeedConfigDscr) mod 256 ;; Total Length (LSB)
    db  (FullSpeedConfigDscrEnd-FullSpeedConfigDscr)  /  256 ;; Total Length (MSB)
    db  1    ;; Number of interfaces
    db  1    ;; Configuration number
    db  0    ;; Configuration string
    db  10100000b   ;; Attributes (b7 - buspwr, b6 - selfpwr, b5 - rwu)
    db  50    ;; Power requirement (div 2 ma)

;; Interface Descriptor
    db  DSCR_INTRFC_LEN    ;; Descriptor length
    db  DSCR_INTRFC        ;; Descriptor type
    db  0         ;; Zero-based index of this interface
    db  0         ;; Alternate setting
    db  2         ;; Number of end points
    db  0ffH      ;; Interface class
    db  00H       ;; Interface sub class
    db  00H       ;; Interface sub sub class
    db  0         ;; Interface descriptor string index

;; Endpoint Descriptor
    db  DSCR_ENDPNT_LEN    ;; Descriptor length
    db  DSCR_ENDPNT        ;; Descriptor type
    db  01H       ;; Endpoint number, and direction
    db  ET_BULK        ;; Endpoint type
    db  40H       ;; Maximun packet size (LSB)
    db  00H       ;; Max packect size (MSB)
    db  00H       ;; Polling interval

;; Endpoint Descriptor
    db  DSCR_ENDPNT_LEN    ;; Descriptor length
    db  DSCR_ENDPNT        ;; Descriptor type
    db  81H       ;; Endpoint number, and direction
    db  ET_BULK        ;; Endpoint type
    db  40H       ;; Maximun packet size (LSB)
    db  00H       ;; Max packect size (MSB)
    db  00H       ;; Polling interval

FullSpeedConfigDscrEnd:

StringDscr:
```

```
StringDscr0:
    db  StringDscr0End-StringDscr0     ;; String descriptor length
    db  DSCR_STRING
    db  09H,04H
StringDscr0End:

StringDscr1:
    db  StringDscr1End-StringDscr1     ;; String descriptor length
    db  DSCR_STRING
    db  'C',00
    db  'y',00
    db  'p',00
    db  'r',00
    db  'e',00
    db  's',00
    db  's',00
StringDscr1End:

StringDscr2:
    db  StringDscr2End-StringDscr2     ;; Descriptor length
    db  DSCR_STRING
    db  'E',00
    db  'Z',00
    db  '-',00
    db  'U',00
    db  'S',00
    db  'B',00
    db  ' ',00
    db  'F',00
    db  'X',00
    db  '2',00
StringDscr2End:

UserDscr:
    dw   0000H
    end
```

C.5 Icv_usb.c

```
#pragma NOIV               // Do not generate interrupt vectors
//-----------------------------------------------------------------------------
//   Original file: C:\Cypres\Usb\Target\Fw\Fx2\periph.c
//   Modified file: ICV_USB.c
//
//   Contents:   Hooks required to implement USB peripheral function.
//               Experimental implementation of SPI interface
//
//   Copyright (c) 1997 AnchorChips, Inc. All rights reserved
//   Modified by Michal Brychta, ICV applications, Analog Devices
//   Modified by Ryan Johnson, WPI MQP Team, Analog Devices, Sep 2005
//   Last change in the code 18. October 2005
//
//-----------------------------------------------------------------------------
#include "fx2.h"
#include "fx2regs.h"

extern BOOL   GotSUD;         // Received setup data flag
extern BOOL   Sleep;
extern BOOL   Rwuen;
extern BOOL   Selfpwr;

BYTE   Configuration;      // Current configuration
BYTE   AlternateSetting;   // Alternate settings

// Added by Michal ***********************************************************
```

```
// SPI interface definitions
sbit    SPI_CS   = IOA^0; // CS
sbit    SPI_SCLK = IOA^1; // SCLK
sbit    SPI_MOSI = IOA^2; // DIN
sbit    SPI_MISO = IOA^3; // DOUT
sbit    SPI_RDY  = IOA^3; // RDY

// Added by Michal ************************************************************
void SPI_init(void)
// Software implemented SPI interface using the general purpose I/O port
// Sets direction and initial levels on the I/O port
{
  OEA = (OEA & 0xF7) | 0x07; // Direction bits for SPI, 0=input, 1=output
  SPI_CS = 1;
  SPI_SCLK = 1;
  SPI_MOSI = 1;
}

// Added by Michal ************************************************************
unsigned char SPI(unsigned char SPI_DATA)
// Software implemented SPI interface using the general purpose I/O port
// Sends / receives one SPI byte
{
  register unsigned char i;
  for(i=0; i<8; i++)
  {
    SPI_MOSI = SPI_DATA >> 7;   // send MOSI
    SPI_SCLK = 0;     // SCLK low
    SPI_DATA <<= 1;
    SPI_DATA |= SPI_MISO;   // sample MISO
    SPI_SCLK = 1;     // SCLK high
  }
  return(SPI_DATA);
}

// Added by Michal ************************************************************
void WaitForRdy(unsigned short timeout)
// Wait for "/RDY" high to low transition. limited by timeout
// for T_DELAY = 40, EZUSB core clock = 48MHz
// max timeout = 65535 ~ wait for 5.4 seconds
// min timeout = 0 ~ don't wait at all, ignore "/RDY"
{
  #define T_DELAY      0x40
  unsigned short i;
  while ((SPI_RDY == 0) && (timeout-- > 0)) // wait for RDY high
  {
    for(i=0; i<T_DELAY; i++)
    {
      if (SPI_RDY != 0) break;
    }
  }
  while ((SPI_RDY != 0) && (timeout-- > 0)) // wait for RDY Low
  {
    for(i=0; i<T_DELAY; i++)
    {
      if (SPI_RDY == 0) break;
    }
  }
}

// Added by Michal ************************************************************
// I2C read
// ****************************************************************************
/* Commented out by Ryan: We won't be using I2C

BOOL My_I2C_Read(BYTE I2C_Addr, I2C_Type, I2C_PtrL, I2C_PtrH, I2C_Len)
{

  #define TimeOut 255            // "time out" for each byte transfer
  BYTE i;                        // data index
```

```
BYTE t;                    // time index
BYTE dummy;                // dummy byte

if (I2C_Len == 0) return(FALSE);// Code doesn't work for Len = 0

for(i=0; i < I2C_Len; i++)     // Cycle through #of bytes
  *(EP0BUF + i) = 0xFF;        // Fill buffer - for case of error

t = TimeOut;
while ((I2CS & bmSTOP)&&(t-->0)); // Wait for any previous stop
if (t==0)                  // Check errors
{
  return(FALSE);           // Don't execute the rest
}

// ----------------------------------------------------------------
// This code is executed only if the I2C pointer is used
// ----------------------------------------------------------------

if (I2C_Type > 0)          // Is I2C pointer used?
{
  I2CS |= bmSTART;            // Start I2C transaction
  I2DAT = (I2C_Addr) & 0xFE;   // Send device address + write

  t = TimeOut;
  while(((I2CS & bmDONE)==0) && (t-- > 0)) // Wait for transfer
  dummy = I2CS;               // dummy read - for little delay
  if ((t==0) || ((I2CS & bmACK) == 0)) // Check errors
  {
     I2CS |= bmSTOP;          // Cancel I2C transaction
     return(FALSE);           // Don't execute the rest
  }

  if (I2C_Type > 1)          // Is 16-bit pointer used?
  {
    I2DAT = I2C_PtrH;         // Send device pointer

    t = TimeOut;
    while(((I2CS & bmDONE)==0) && (t-- > 0)) // Wait for transfer
    dummy = I2CS;             // dummy read - for little delay
    if ((t==0) || ((I2CS & bmACK) == 0)) // Check errors
    {
     I2CS |= bmSTOP;          // Cancel I2C transaction
     return(FALSE);           // Don't execute the rest
    }
  }

  I2DAT = I2C_PtrL;           // Send device pointer

  t = TimeOut;
  while(((I2CS & bmDONE)==0) && (t-- > 0)) // Wait for transfer
  dummy = I2CS;               // dummy read - for little delay
  if ((t==0) || ((I2CS & bmACK) == 0)) // Check errors
  {
     I2CS |= bmSTOP;          // Cancel I2C transaction
     return(FALSE);           // Don't execute the rest
  }
}

// ----------------------------------------------------------------
// From here the code is executed regardless on the I2C pointer
// ----------------------------------------------------------------

I2CS |= bmSTART;              // Start I2C transaction
I2DAT = (I2C_Addr) | 0x01;     // Send device address + read

t = TimeOut;
while(((I2CS & bmDONE)==0) && (t-- > 0)) // Wait for transfer
dummy = I2CS;                 // dummy read - for little delay
if ((t==0) || ((I2CS & bmACK) == 0)) // Check errors
```

```
  {
     I2CS |= bmSTOP;          // Cancel I2C transaction
     return(FALSE);          // Don't execute the rest
  }

  for(i=0; i < I2C_Len; i++)      // Cycle through #of bytes
  {
   if ((i+1) == I2C_Len) I2CS |= bmLASTRD;  // Set last-read
   if (i == 0) dummy = I2DAT;    // Dummy read to start transfer

   t = TimeOut;
   while(((I2CS & bmDONE)==0) && (t-- > 0)) // Wait for transfer
   if (t==0)                    // Check errors
   {
     I2CS |= bmSTOP;          // Cancel I2C transaction
     return(FALSE);          // Don't execute the rest
   }
   // Set stop condition before reading the last byte from register
   if ((i+1) == I2C_Len) I2CS |= bmSTOP;
   *(EP0BUF + i) = I2DAT;      // Read data from I2C to buffer
  }
  return(TRUE);
}


// Added by Michal ************************************************************
// I2C write
// ****************************************************************************
BOOL My_I2C_Write(BYTE I2C_Addr, I2C_Type, I2C_PtrL, I2C_PtrH, I2C_Len)
{
  #define TimeOut 255            // "time out" for each byte transfer
  BYTE i;                      // data index
  BYTE t;                      // time index
  BYTE dummy;                   // dummy byte

  if (I2C_Len == 0) return(FALSE);// Code doesn't work for Len = 0

  t = TimeOut;
  while ((I2CS & bmSTOP)&&(t-->0)); // Wait for any previous stop
  if (t==0)                     // Check errors
  {
   return(FALSE);                // Don't execute the rest
  }

  I2CS |= bmSTART;              // Start I2C transaction
  I2DAT = (I2C_Addr) & 0xFE;      // Send device address + write

  t = TimeOut;
  while(((I2CS & bmDONE)==0) && (t-- > 0)) // Wait for transfer
  dummy = I2CS;                 // dummy read - for little delay
  if ((t==0) || ((I2CS & bmACK) == 0)) // Check errors
  {
     I2CS |= bmSTOP;          // Cancel I2C transaction
     return(FALSE);          // Don't execute the rest
  }

  // ---------------------------------------------------------------
  // This code is executed only if the 16-bit I2C pointer is used
  // ---------------------------------------------------------------

  if (I2C_Type > 1)             // Is 16-bit pointer used?
  {
   I2DAT = I2C_PtrH;            // Send device pointer

   t = TimeOut;
   while(((I2CS & bmDONE)==0) && (t-- > 0)) // Wait for transfer
   dummy = I2CS;               // dummy read - for little delay
   if ((t==0) || ((I2CS & bmACK) == 0)) // Check errors
   {
     I2CS |= bmSTOP;          // Cancel I2C transaction
```

```
      return(FALSE);          // Don't execute the rest
    }
  }

  // ----------------------------------------------------------------
  // This code is executed only if the I2C pointer is used
  // ----------------------------------------------------------------

  if (I2C_Type > 0)             // Is I2C pointer used?
  {
   I2DAT = I2C_PtrL;             // Send device pointer
   t = TimeOut;
   while(((I2CS & bmDONE)==0) && (t-- > 0)) // Wait for transfer
   dummy = I2CS;               // dummy read - for little delay
   if ((t==0) || ((I2CS & bmACK) == 0)) // Check errors
     {
      I2CS |= bmSTOP;          // Cancel I2C transaction
      return(FALSE);          // Don't execute the rest
     }
   }

  // ----------------------------------------------------------------
  // From here the code is executed regardless on the I2C pointer
  // ----------------------------------------------------------------

  for(i=0; i < I2C_Len; i++)      // Cycle through #of bytes
  {
   I2DAT = *(EP0BUF + i);        // Send data from buffer to I2C
   t = TimeOut;
   while(((I2CS & bmDONE)==0) && (t-- > 0)) // Wait for transfer
   dummy = I2CS;               // dummy read - for little delay
   if ((t==0) || ((I2CS & bmACK) == 0)) // Check errors
   {
      I2CS |= bmSTOP;          // Cancel I2C transaction
      return(FALSE);          // Don't execute the rest
   }
  }
  I2CS |= bmSTOP;               // Stop I2C transaction
  return(TRUE);
}


*/
//-----------------------------------------------------------------------------
// Task Dispatcher hooks
//   The following hooks are called by the task dispatcher.
//-----------------------------------------------------------------------------

void TD_Init(void)           // Called once at startup
{
  BREAKPT &= ~bmBPEN;        // to see BKPT LED go out TGE
  Rwuen = TRUE;             // Enable remote-wakeup

// Added by Michal ************************************************************
  CPUCS = (CPUCS & 0xe7) | 0x10;           // 48MHz
  SPI_init();

// ****************************************************************************
}

void TD_Poll(void)            // Called repeatedly while the device is idle
{
}

BOOL TD_Suspend(void)         // Called before the device goes into suspend mode
{
  return(TRUE);
}

BOOL TD_Resume(void)         // Called after the device resumes
```

```
{
  return(TRUE);
}

//-----------------------------------------------------------------------------
// Device Request hooks
//   The following hooks are called by the end point 0 device request parser.
//-----------------------------------------------------------------------------

BOOL DR_GetDescriptor(void)
{
  return(TRUE);
}

BOOL DR_SetConfiguration(void)   // Called when a Set Configuration command is received
{
  Configuration = SETUPDAT[2];
  return(TRUE);            // Handled by user code
}

BOOL DR_GetConfiguration(void)   // Called when a Get Configuration command is received
{
  EP0BUF[0] = Configuration;
  EP0BCH = 0;
  EP0BCL = 1;
  return(TRUE);            // Handled by user code
}

BOOL DR_SetInterface(void)       // Called when a Set Interface command is received
{
  AlternateSetting = SETUPDAT[2];
  return(TRUE);            // Handled by user code
}

BOOL DR_GetInterface(void)       // Called when a Set Interface command is received
{
  EP0BUF[0] = AlternateSetting;
  EP0BCH = 0;
  EP0BCL = 1;
  return(TRUE);            // Handled by user code
}

BOOL DR_GetStatus(void)
{
  return(TRUE);
}

BOOL DR_ClearFeature(void)
{
  return(TRUE);
}

BOOL DR_SetFeature(void)
{
  return(TRUE);
}

BOOL DR_VendorCmnd(void)
{
// ********************************************************************************
// EZUSB Vendor Command - Modified by Michal

// This is the "heart" of communication with PC software
// Using the "UsbFunctions.dll"
// VendorIORequest
//   (short bMyReq,long wMyVal,long wMyIndx,short iMyDir,short MyLen,char* MyBuf)

// short iMyDir  ~~ byte SETUPDAT[0] .. direction of transaction
// iMyDir = 0    -> byte SETUPDAT[0] = 0x40 .. PC -> EZUSB
// iMyDir = 1    -> byte SETUPDAT[0] = 0xC0 .. EZUSB -> PC
```

```
// short bMyReq  -> byte SETUPDAT[1] .. request identification
// long  wMyVal  -> byte SETUPDAT[2](LSB),SETUPDAT[3](MSB) .. can be user defined
// long  wMyIndx -> byte SETUPDAT[4](LSB),SETUPDAT[5](MSB) .. can be user defined
// short MyLen   -> byte SETUPDAT[6] .. length of buffer, max 64
// char* MyBuf  <-> *EP0BUF

// *****************************************************************************

  #define VR_SPI   0xDA
  #define VR_IO    0xDB
/* Commented out by Ryan: Not using I2C
  #define VR_I2C0  0xDC
  #define VR_I2C1  0xDD
  #define VR_I2C2  0xDE
*/
  #define VR_Err   0xDF

  #define VR_UPLOAD    0xC0
  #define VR_DOWNLOAD  0x40

  int i;
//   BYTE I2C_Type;  Commented out by Ryan
  static BYTE Err_Count = 0;


  switch(SETUPDAT[1])    // Vendor Request
  {

    // *********************************************************************
    // EZUSB SPI interface
    // *********************************************************************

    // bMyReq  .. VR_SPI .. 0xDA

    // wMyVal .. low byte is passed to the SPI when reading
      // This is meant as value for ICV parts "communication register"
      // It allows using only one USB transaction when reading
      // a register "addressed" by the "communication register"

    // wMyIndx .. 16 bit value is passed as a "timeout"
      // This is meant for waiting for ICV parts "/RDY" signal
      // The SPI operation waits for "/RDY" high to low transition
      // max timeout = 65535 ~ wait for 5.4 seconds
      // min timeout = 0 ~ don't wait at all, ignore "/RDY"
      // Implemented as a simple loop in WaitForRdy

    // iMyDir = 0 .. PC -> EZUSB
    // iMyDir = 1 .. EZUSB -> PC
    // MyLen   .. Number of bytes to send / receive, max 64
    // MyBuf   .. Buffer, array of bytes

    // *********************************************************************

    case VR_SPI:        // Decoding bMyReq

      // *******************************************************************
      // SPI read
      // *******************************************************************

      if (SETUPDAT[0] == VR_UPLOAD)    // Decoding iMyDir
      {
        while(EP0CS & bmEPBUSY);      // Waiting for end point rdy?

        SPI_CS = 0;          // CS low

        // Waiting for RDY high to low, limited by timeout
        WaitForRdy(SETUPDAT[4] | (SETUPDAT[5] << 8));

        for(i=0; i<SETUPDAT[6]; i++)  // MyLen - cycle through #of bytes
```

```
        {
          // Read SPI and store data in buffer
          // Send lo(wMyVal) as the first byte to the SPI
          if (i==0) *EP0BUF = SPI(SETUPDAT[2]);
          // Send 0 to the SPI for the rest of the transaction
          else *(EP0BUF + i) = SPI(0);
        }

        SPI_CS = 1;           // CS high

        EP0BCH = 0;
        EP0BCL = SETUPDAT[6];  // Arm endpoint with # bytes to transfer back
      }

      // ********************************************************************
      // SPI write
      // ********************************************************************

      if (SETUPDAT[0] == VR_DOWNLOAD)  // Decoding iMyDir
      {
        EP0BCH = 0;   // Clear bytecount to allow new data in
        EP0BCL = 0;   // Also stops NAKing
        while(EP0CS & bmEPBUSY); // Waiting for end point rdy?

        SPI_CS = 0;         // CS low

        WaitForRdy(SETUPDAT[4] | (SETUPDAT[5] << 8));

        for(i=0; i<SETUPDAT[6]; i++)  // MyLen - cycle through #of bytes
        {
          // Write SPI data from buffer
          SPI(*(EP0BUF + i));
        }

        SPI_CS = 1;         // CS high

      }
    break;

    // **********************************************************************
    // EZ_USB I/O ports
    // **********************************************************************

    // Read/write the EZ USB I/O ports configuration

    // bMyReq  .. VR_IO .. 0xDB
    // wMyVal  .. not used
    // wMyIndx .. not used
    // iMyDir = 0 .. PC -> EZUSB
    // iMyDir = 1 .. EZUSB -> PC
    // MyLen   .. Number of bytes to send / receive, should be 1..6
    // MyBuf   .. Buffer, array of bytes

    // The values in buffer represent bytes in following order:
    // 0 .. port A value
    // 1 .. port A direction
    // 2 .. port B value
    // 3 .. port B direction
    // 4 .. port D value
    // 5 .. port D direction
    // The I/O port direction is per bit, 0=input, 1=output
    // It is possible to use only part of configuration,
    // For example only read or write the port A value

    // **********************************************************************

    case VR_IO:       // Decoding bMyReq

      // ********************************************************************
      // EZ_USB I/O ports read
```

```
    // *******************************************************************

    if (SETUPDAT[0] == VR_UPLOAD)    // Decoding iMyDir
    {
      while(EP0CS & bmEPBUSY);      // Waiting for end point rdy?
      for(i=0; i<SETUPDAT[6]; i++)  // MyLen - cycle through #of bytes
      {
        switch(i)
        {
          // read the I/O port(s) configuration(s)
          case 0: *(EP0BUF+i) = IOA; break;
          case 1: *(EP0BUF+i) = OEA; break;
          case 2: *(EP0BUF+i) = IOB; break;
          case 3: *(EP0BUF+i) = OEB; break;
          case 4: *(EP0BUF+i) = IOD; break;
          case 5: *(EP0BUF+i) = OED; break;
        }
      }
      EP0BCH = 0;
      EP0BCL = SETUPDAT[6];  // Arm endpoint with # bytes to transfer back
    }

    // *******************************************************************
    // EZ_USB I/O ports write
    // *******************************************************************

    if (SETUPDAT[0] == VR_DOWNLOAD)  // Decoding iMyDir
    {
      EP0BCH = 0;   // Clear bytecount to allow new data in
      EP0BCL = 0;   // Also stops NAKing
      while(EP0CS & bmEPBUSY);      // Waiting for end point rdy?
      for(i=0; i<SETUPDAT[6]; i++)  // MyLen - cycle through #of bytes
      {
        switch(i)
        {
          // write the I/O port(s) configuration(s)
          case 0: IOA = *(EP0BUF+i); break;
          case 1: OEA = *(EP0BUF+i); break;
          case 2: IOB = *(EP0BUF+i); break;
          case 3: OEB = *(EP0BUF+i); break;
          case 4: IOD = *(EP0BUF+i); break;
          case 5: OED = *(EP0BUF+i); break;
        }
      }
    }
  break;

  // *******************************************************************
  // EZUSB I2C interface
  // *******************************************************************

  // Functional, but locks when I2C Bus Error
  // I.e., locks when the EZUSB cannot control the I2C bus as a master

  // When NAK, the I2C bus is released and my error counter is incremented

  // bMyReq  .. VR_I2C0 .. 0xDC .. Simple I2C, without pointer
  //         .. VR_I2C1 .. 0xDD .. Extended I2C, 8-bit pointer
  //         .. VR_I2C2 .. 0xDE .. Extended I2C,16-bit pointer
  // wMyVal  .. I2C Device (Slave) Address in low byte
  // wMyIndx .. I2C Pointer (Register Address)
  // iMyDir = 0 .. PC -> EZUSB
  // iMyDir = 1 .. EZUSB -> PC
  // MyLen   .. Number of bytes to send / receive, max 64
  // MyBuf   .. Buffer, array of bytes

  // *******************************************************************
/* Commented out by Ryan: I2C not needed
  case VR_I2C0:       // Decoding bMyReq
  case VR_I2C1:
```

```
    case VR_I2C2:

      switch (SETUPDAT[1])                  // Decoding bMyReq again
      {
        case VR_I2C0: I2C_Type = 0; break;  // Simple I2C, without pointer
        case VR_I2C1: I2C_Type = 1; break;  // Extended I2C, 8-bit pointer
        case VR_I2C2: I2C_Type = 2; break;  // Extended I2C,16-bit pointer
      }

      // ********************************************************************
      // I2C read
      // ********************************************************************

      if (SETUPDAT[0] == VR_UPLOAD)      // Decoding iMyDir
      {
        while(EP0CS & bmEPBUSY);       // Waiting for end point rdy?

        //              I2C_Addr   ,I2C_Type,I2C_PtrL   , I2C_Ptr   , I2C_Len    )
        if (!My_I2C_Read(SETUPDAT[2],I2C_Type,SETUPDAT[4],SETUPDAT[5],SETUPDAT[6]))
          Err_Count++;                 // If error, Increment error counter

        EP0BCH = 0;
        EP0BCL = SETUPDAT[6];  // Arm endpoint with # bytes to transfer back
      }

      // ********************************************************************
      // I2C write
      // ********************************************************************

      if (SETUPDAT[0] == VR_DOWNLOAD)  // Decoding iMyDir
      {
        EP0BCH = 0;   // Clear bytecount to allow new data in
        EP0BCL = 0;   // Also stops NAKing

        while(EP0CS & bmEPBUSY);      // Waiting for USB?

        //              I2C_Addr   ,I2C_Type,I2C_PtrL   , I2C_Ptr   , I2C_Len    )
        if (!My_I2C_Write(SETUPDAT[2],I2C_Type,SETUPDAT[4],SETUPDAT[5],SETUPDAT[6]))
          Err_Count++;                 // If error, Increment error counter
      }
    break;

*/
    // ************************************************************************
    // EZUSB Error counter
    // ************************************************************************

    // I have implemented this counter to make software more robust
    // The error counter is incremented with any I2C communication error
    // The error counter is cleared when read via USB

    // bMyReq  .. VR_Err  .. 0xDF .. Read error counter
    // wMyVal  .. not used
    // wMyIndx .. not used
    // iMyDir = 1 .. EZUSB -> PC
    // MyLen  = 1 .. Number of bytes to receive
    // MyBuf(0)   .. Error counter value - # of errors

    // ************************************************************************

    case VR_Err:         // Decoding bMyReq
      if (SETUPDAT[0] == VR_UPLOAD)    // Decoding iMyDir
      {
        while(EP0CS & bmEPBUSY);      // Waiting for end point rdy?

        *EP0BUF = Err_Count;          // Copy error counter for sending
        Err_Count = 0;                // Clear error counter

        EP0BCH = 0;
        EP0BCL = SETUPDAT[6];  // Arm endpoint with # bytes to transfer back
```

```
        }
        if (SETUPDAT[0] == VR_DOWNLOAD)  // Decoding iMyDir
        {
          EP0BCH = 0;   // Clear bytecount to allow new data in
          EP0BCL = 0;   // Also stops NAKing
          while(EP0CS & bmEPBUSY);      // Waiting for end point rdy?
          // Don't do anything
        }
      break;

      // ********************************************************************
      // End of user Vendor Commands
      // ********************************************************************

   }
   return(FALSE);
}

//-----------------------------------------------------------------------------
// USB Interrupt Handlers
//   The following functions are called by the USB interrupt jump table.
//-----------------------------------------------------------------------------

// Setup Data Available Interrupt Handler
void ISR_Sudav(void) interrupt 0
{
   GotSUD = TRUE;            // Set flag
   EZUSB_IRQ_CLEAR();
   USBIRQ = bmSUDAV;         // Clear SUDAV IRQ
}

// Setup Token Interrupt Handler
void ISR_Sutok(void) interrupt 0
{
   EZUSB_IRQ_CLEAR();
   USBIRQ = bmSUTOK;         // Clear SUTOK IRQ
}

void ISR_Sof(void) interrupt 0
{
   EZUSB_IRQ_CLEAR();
   USBIRQ = bmSOF;            // Clear SOF IRQ
}

void ISR_Ures(void) interrupt 0
{
   // whenever we get a USB reset, we should revert to full speed mode
   pConfigDscr = pFullSpeedConfigDscr;
   ((CONFIGDSCR xdata *) pConfigDscr)->type = CONFIG_DSCR;
   pOtherConfigDscr = pHighSpeedConfigDscr;
   ((CONFIGDSCR xdata *) pOtherConfigDscr)->type = OTHERSPEED_DSCR;

   EZUSB_IRQ_CLEAR();
   USBIRQ = bmURES;         // Clear URES IRQ
}

void ISR_Susp(void) interrupt 0
{
   Sleep = TRUE;
   EZUSB_IRQ_CLEAR();
   USBIRQ = bmSUSP;
}

void ISR_Highspeed(void) interrupt 0
{
   if (EZUSB_HIGHSPEED())
   {
      pConfigDscr = pHighSpeedConfigDscr;
      ((CONFIGDSCR xdata *) pConfigDscr)->type = CONFIG_DSCR;
      pOtherConfigDscr = pFullSpeedConfigDscr;
```

```
    ((CONFIGDSCR xdata *) pOtherConfigDscr)->type = OTHERSPEED_DSCR;
  }

  EZUSB_IRQ_CLEAR();
  USBIRQ = bmHSGRANT;
}
void ISR_Ep0ack(void) interrupt 0
{
}
void ISR_Stub(void) interrupt 0
{
}
void ISR_Ep0in(void) interrupt 0
{
}
void ISR_Ep0out(void) interrupt 0
{
}
void ISR_Ep1in(void) interrupt 0
{
}
void ISR_Ep1out(void) interrupt 0
{
}
void ISR_Ep2inout(void) interrupt 0
{
}
void ISR_Ep4inout(void) interrupt 0
{
}
void ISR_Ep6inout(void) interrupt 0
{
}
void ISR_Ep8inout(void) interrupt 0
{
}
void ISR_Ibn(void) interrupt 0
{
}
void ISR_Ep0pingnak(void) interrupt 0
{
}
void ISR_Ep1pingnak(void) interrupt 0
{
}
void ISR_Ep2pingnak(void) interrupt 0
{
}
void ISR_Ep4pingnak(void) interrupt 0
{
}
void ISR_Ep6pingnak(void) interrupt 0
{
}
void ISR_Ep8pingnak(void) interrupt 0
{
}
void ISR_Errorlimit(void) interrupt 0
{
}
void ISR_Ep2piderror(void) interrupt 0
{
}
void ISR_Ep4piderror(void) interrupt 0
{
}
void ISR_Ep6piderror(void) interrupt 0
{
}
void ISR_Ep8piderror(void) interrupt 0
```

```
{
}
void ISR_Ep2pflag(void) interrupt 0
{
}
void ISR_Ep4pflag(void) interrupt 0
{
}
void ISR_Ep6pflag(void) interrupt 0
{
}
void ISR_Ep8pflag(void) interrupt 0
{
}
void ISR_Ep2eflag(void) interrupt 0
{
}
void ISR_Ep4eflag(void) interrupt 0
{
}
void ISR_Ep6eflag(void) interrupt 0
{
}
void ISR_Ep8eflag(void) interrupt 0
{
}
void ISR_Ep2fflag(void) interrupt 0
{
}
void ISR_Ep4fflag(void) interrupt 0
{
}
void ISR_Ep6fflag(void) interrupt 0
{
}
void ISR_Ep8fflag(void) interrupt 0
{
}
void ISR_GpifComplete(void) interrupt 0
{
}
void ISR_GpifWaveform(void) interrupt 0
{
}
```