Introduction

Author: Christopher Best, Microchip Technology Inc.

A thermographic camera, also known as an infrared camera or thermal imaging camera, uses infrared radiation to create an image that we can see in the visible light spectrum. This application note guides the reader through the steps to build a simple, inexpensive thermal camera.
# Table of Contents

Introduction......................................................................................................................1

1. Electromagnetic Radiation.........................................................................................3
   1.1. Infrared Radiation (IR)..........................................................................................3

2. Thermal Camera Components.................................................................................. 4
   2.1. Infrared Sensor.......................................................................................................4
   2.2. 128x128 Pixel RBG LCD......................................................................................5
   2.3. PIC18F27K42 Microcontroller..............................................................................7

3. Thermal Camera Application................................................................................... 18
   3.1. Initializing the Application......................................................................................18
   3.2. LCD Initialization..................................................................................................18
   3.3. Initializing the Grid-EYE® Sensor.........................................................................19
   3.4. Enabling the Splash Screen..................................................................................19
   3.5. Image Processing...................................................................................................19

4. Conclusion...............................................................................................................28

The Microchip Web Site................................................................................................ 29
Customer Change Notification Service..........................................................................29
Customer Support......................................................................................................... 29
Microchip Devices Code Protection Feature.................................................................29
Legal Notice...................................................................................................................30
Trademarks.....................................................................................................................30
Quality Management System Certified by DNV.............................................................31
Worldwide Sales and Service.........................................................................................32
1. **Electromagnetic Radiation**

All normal matter emits electromagnetic radiation when its temperature is above absolute zero (-273.15°C). This radiation, also known as thermal radiation, represents the conversion of matter’s thermal energy into electromagnetic energy, and may include both visible and infrared radiation.

Visible radiation, or visible light, is the electromagnetic radiation that is visible to the human eye, and is typically defined as having wavelengths in the range of 400 to 700 nm. Infrared radiation is invisible to the human eye, and is defined as having wavelengths ranging from 700 nm to 1 mm. Thermal radiation emitted by ordinary objects that are in thermodynamic equilibrium with their surrounding environments can be considered black-body radiation. Objects that are near room temperature (25°C) emit thermal radiation in the infrared spectrum.

Black-body objects are idealized physical objects that absorb all incident electromagnetic radiation, meaning that all radiation that interacts with an object is absorbed. Of course, in nature there are no ideal black-body objects – black holes are near-perfect black bodies since they absorb all radiation that falls into them, but may not be in perfect thermodynamic equilibrium with its surrounding environment.

When a black body is in thermal equilibrium (constant temperature), the body emits black-body radiation according to Planck’s Law, which describes the distribution of the electromagnetic radiation’s power in terms of frequency components at a given temperature. In other words, a black-body object that is held at a constant temperature will emit radiation of a specific magnitude and frequency that is dependent only on the object’s temperature, not its shape or composition.

Real-world objects, since true black-body objects don’t physically exist, emit energy at a fraction of black-body objects. This fraction is known as an object’s emissivity, and is used to determine the object’s actual effectiveness in emitting thermal radiation. An ideal black-body surface has an emissivity of ‘1’, meaning that all radiation that interacts with the surface is absorbed by the object. Polished silver, on the other hand, has an emissivity of ‘0.02’, which means that almost all the radiation is scattered or reflected from the surface and very little is absorbed.

1.1 **Infrared Radiation (IR)**

Infrared radiation is a type of electromagnetic radiation that radiates in wavelengths between 700 nanometers and 1 millimeter. These wavelengths are invisible to the human eye, but can be felt as heat. For example, the sun emits roughly half of its energy as infrared radiation, and although we can’t see the radiation with the naked eyes, the heat can be felt simply by standing in sunlight.
2. Thermal Camera Components
The thermal camera built for this application note consists of the following three main hardware components:

- Infrared sensor
- 128x128 pixel RBG LCD
- PIC18F27K42 8-bit microcontroller

2.1 Infrared Sensor
Infrared detection is performed using the Panasonic Grid-EYE® sensor. The Grid-EYE is an 8x8 (64) pixel infrared array sensor designed using MEMS (micro-electro-mechanical system) thermopile technology. The sensor communicates via the I²C bus operating at a maximum 400 kHz. The sensor also features an on-board gain amplifier, Analog-to-Digital Converter (ADC), and a thermistor (see figure below).

Figure 2-1. Grid-EYE Block Diagram

2.1.1 Thermopile Array
The Grid-EYE sensor uses a MEMS-based pixel structure to create a series of free-standing thermocouples. Each thermocouple consists of two thin wires of different thermal materials. The two wires are joined together at one end, known as the hot junction, with the other ends connected to a heat sink. The hot junction is connected to a very thin common IR absorption membrane, which is shared by all 64 thermocouples. If there is a difference in temperature between the two junctions, a tiny Electromotive Force (EMF) voltage is created, which can be measured and converted into temperature. This phenomenon is referred to as the Seebeck effect.

2.1.2 Grid-EYE Operation
The Grid-EYE sensor begins its operation by absorbing infrared thermal energy across its 60° field of view. The IR energy passes through an integrated silicon lens that acts as an optical filter, allowing absorption of IR energy for wavelengths between 5 and 13 µm (far infrared region). Once the IR energy passes through the lens, it is absorbed by each of the thermopile array’s 64 sensing elements. Each of the sensing elements converts the IR energy it absorbed into an analog output signal. As previously
mentioned, when there is a temperature difference between the junctions of a thermocouple, a small voltage is created.

The created voltage is typically in the low millivolt range, which may be too small to accurately detect small changes in energy. To correct this, each sensing element’s analog output is passed through a gain amplifier, effectively increasing the resolution of each element. Once each signal is amplified, it is passed through the ADC, where it is referenced against the on-board thermistor’s temperature value, and converted into a 12-bit (11 bits + 1 Sign bit) digital equivalent. Each of the 64 pixels has its own unique Temperature register, which holds the converted digital temperature equivalent. These Temperature registers can be read by a microcontroller over the I2C bus.

After initial power-up and sensor configuration, the Grid-EYE sensor requires 15 seconds to stabilize before it can begin capturing temperature data. Once stabilized, the sensor captures 10 frames per second. The sensor can be configured to either load the Temperature registers after each frame has been captured (frame frequency = 10 Hz), or load the Temperature registers with the average of the 10 frames (frame frequency = 1 Hz). Additionally, a moving average calculation can be performed on either frame rate.

When the sensor is configured for a frame frequency of 10 Hz, the host microcontroller may read the Temperature registers once every 100 ms. New frame data are available every 100 ms, which means that the microcontroller can update the output image quickly. When using this approach, the image will appear to update in ‘real time’, which means that if an object is moving, the image will show that movement quickly. The downside to this approach is that any sudden temperature fluctuations may cause the image to distort.

For example, if a single pixel in frame ‘A’ changes from a temperature of 21°C to 28°C, the color for that single pixel will change from one frame to the next. To combat this phenomenon, the moving average feature can be enabled. When the moving average feature is enabled, the temperature data from the previous frame is averaged with the temperature data from the new frame. Using the previous example, if a single pixel changes from 21°C to 28°C, the Temperature register for that pixel will be loaded with 24°C, and the color change from pixel to pixel will not be as dramatic.

When the sensor is configured for a frame frequency of 1 Hz, the sensor loads the Temperature registers with the average of all ten frames. When using this approach, temperature fluctuations from frame to frame will appear much less dramatic, but the image will only update every one second. This means that if an object is moving, the image will not have smooth ‘real-time’ motion.

The Grid-EYE sensor also has configurable interrupt capabilities. Each of the 64 pixels can be configured to generate an interrupt if an individual pixel value violates a programmable upper or lower threshold. The upper and lower threshold values apply to the entire frame, and hysteresis can be added to prevent interrupt toggling when a pixel value is very close to a threshold limit.

### 2.2 128x128 Pixel RBG LCD

This application uses the 1.44-inch Varitronix COG-C144MVGI-08 graphic display LCD module. The module features Color Super-Twist Nematic (CSTN) LCD technology, which uses passive-matrix addressing. In a CSTN LCD, row and column signals are used to directly address a pixel, and the pixel must maintain its ON/OFF state without the use of a switch or capacitor. Each visual pixel is divided into three physical sub-pixels, and each sub-pixel uses either a red, blue, or green filter to display color. The display uses a white LED backlight whose light passes through each sub-pixel. The intensity of each sub-pixel’s output is controlled by the display’s LCD driver, creating up to 65 thousand unique colors.
The COG-C144MVGI display is driven by a Samsung® S6B3306 LCD driver, which is integrated into the display module. The driver simplifies the interface between a microcontroller and the display, which means that fewer connections are necessary. The driver allows the user to control the backlight, turn the display on/off, reset the display, and send commands and color data via a serial communications bus. The driver also allows the user to rotate the image, which will prove to be useful for this application.

The LCD driver can be configured to output either 4,096 or 65,536 colors. In either color mode, the driver accepts a 16-bit color code. In the 4,096 color mode, the driver only needs 12 of the 16 bits to generate a color, and the upper four bits are ignored (see Figure 2-2). In the 65k color mode, all 16 bits are used (see Figure 2-3). In either mode, the color code is broken down into three color segments, one for red, one for green, and one for blue.

In 4,096 color mode, each segment is equally divided into four bits, or sixteen intensity levels. Since there are three 16-level color segments, there are a total of 4,096 possible color codes.

**Figure 2-2. 4,096-Color Mode 12-Bit Color Data**

```
X X X X R3 R2 R1 R0 G3 G2 G1 G0 B3 B2 B1 B0
R3-R0: Red intensity level
G3-G0: Green intensity level
B3-B0: Blue intensity level
```

In 65k color mode, the 16-bit word is divided into the standard RGB565 color format. The RGB565 format is a 16-bit color scheme in which bits<15:11>(5 bits) define the red intensity, bits<10:5>(6 bits) define the green intensity, and bits<4:0>(5 bits) define the blue intensity. The RGB565 format gives an extra bit to the green color due to the fact that the human vision is more sensitive to the green wavelengths of the visible light spectrum.

**Figure 2-3. 65K Color Mode 16-Bit Color Data**

```
R4 R3 R2 R1 R0 G5 G4 G3 G2 G1 G0 B4 B3 B2 B1 B0
R4-R0: Red intensity level
G5-G0: Green intensity level
B4-B0: Blue intensity level
```

Each color segment, regardless of the color mode, controls the output level of the corresponding pixel. For example, if a bright red color is desired in 65k color mode, one could write the value of 0xF800 to the LCD’s display data RAM location. This means that the red segment of the word is at the maximum output level, while the green and blue segments are at the lowest levels. When the 16-bit word is written into the pixel’s display data RAM location, the LCD driver fully energizes the line controlling the pixel’s red segment, but does not activate the blue or green segments (see Example 2-1).
The RGB565 color codes are transmitted to the LCD driver over the Serial Peripheral Interface (SPI) bus. The Samsung S6B3306 LCD driver only accepts data, which removes the need for the Master In Slave Out (MISO) line. The SPI master selects the LCD as a SPI slave by pulling the Slave Select (SS), also referred to as Chip Select (CS) to ground.

2.3 PIC18F27K42 Microcontroller

The PIC18F27K42 microcontroller is used to read the temperature data from the sensor, perform the image processing, and transmit the color data to the LCD. The PIC18F27K42 is an advanced 8-bit microcontroller that features 128 kB of Program Memory and 8 kB of SRAM. The PIC18F27K42 has a rich peripheral set; however, this application note will only describe the peripherals used in this application, which include:

- Timer1 module
- Direct Memory Access (DMA) module
- I²C module
- SPI module

For more information on the PIC18F27K42, visit the Microchip website at www.microchip.com.

2.3.1 Timer1

Timer1 is a 16-bit incrementing counter that is implemented in the thermal camera application to generate a 15-second delay. When the camera is first powered on and the Grid-EYE sensor has been configured for use, it requires a 15-second delay to stabilize. Rather than using a ‘delay’ function, which suspends program execution during the delay cycle, Timer1 can be used to do the same task. Since Timer1 operates in the background, code execution continues, allowing the core to focus on other tasks rather than suspending code execution for a 15-second ‘delay’ function.
Timer1 can be used with either an internal or external clocking source, but for this application the internal oscillator is used. Timer1 also features a clock prescaler, which allows the incoming Clock signal to be divided, or slowed down.

Timer1 increments its count on every instruction cycle when used with the internal oscillator. In normal operation, the Timer begins its count starting at 0, and increases it every instruction until it reaches the count of 0xFFFF. At that point, the next instruction causes the Timer to rollover and start counting again from 0 and sets the Timer1 Interrupt Flag (TMR1IF) bit. Since Timer1 will rollover very quickly (in milliseconds), the Timer must repeatedly count and rollover several times to create a delay.

Example 2-2 shows how the Timer is used to generate the 15-second delay. In this example, the $F_{OSC}/4$ option is selected as the Timer clock source, and a 1:8 prescaler is used to slow the Clock signal. The calculations used in this example result in the number of times Timer1 is required to rollover to generate a 15-second delay.

Alternatively, the MPLAB® X Code Configurator (MCC) plug-in tool found in the MPLAB X IDE can be used to generate the code to enable and configure Timer1 (see Figure 2-4). The MCC tool also allows the configuration of the 15-second delay, which saves setup time.

**Example 2-2. Creating a 15-Second Delay Using Timer1**

- **Conditions:**
  - $F_{OSC} = 64$ MHz
  - $F_{OSC}/4 = 16$ MHz
  - Prescaler = 1:8

- **Timer1 clock**
  \[
  \text{Timer1 clock} = \frac{F_{OSC}/4}{\text{Prescaler}} = \frac{16 \text{ MHz}}{8} = 2 \text{ MHz}
  \]

- **Instruction cycle time**
  \[
  \text{Instruction cycle time} = \frac{1}{F_{OSC}} = \frac{1}{2 \text{ MHz}} = 500 \text{ ns}
  \]

- **Timer1 rollover time**
  \[
  \text{Timer1 rollover time} = 2^{16} \times \text{Instruction cycle time} = 65,536 \times 500 \text{ ns} = 32,768 \text{ ms}
  \]

- **Rollover count needed for 15-second delay**
  \[
  \text{Rollover count needed} = \frac{15 \text{ s}}{32.768 \text{ ms}} = 457.76 \approx 458 \text{ times (round up)}
  \]
2.3.2 Direct Memory Access (DMA)

The Direct Memory Access (DMA) module allows data transfer between the memory regions of the PIC® microcontroller without any CPU intervention. The DMA eliminates the need for CPU handling of interrupts intended for tracking data transfers, allowing the CPU to carry out other tasks while transfers are taking place.

The DMA supports data transfer between the following memory regions:

- GPR and SFR memory (both read and write capabilities)
- Program Flash Memory (read-only)
- Data EEPROM memory (read-only)

The DMA transfers data from a memory source to a memory destination one byte at a time. This transfer is referred to as a DMA data transaction. A DMA message contains one or more DMA transactions.

Each DMA transaction consists of two independent actions:

1. The DMA reads the data located at the source address and loads it into the DMA Buffer register.
2. The DMA writes the value stored in the DMA Buffer into the destination address location.

It is important to note that the DMA can only read from the Program Flash Memory and the data EEPROM memory; writes to these regions are prohibited. Table 2-1 shows the possible read sources and write destination regions.
Table 2-1. DMA Memory Access

<table>
<thead>
<tr>
<th>Read Source</th>
<th>Write Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Flash Memory</td>
<td>GPR</td>
</tr>
<tr>
<td>Program Flash Memory</td>
<td>SFR</td>
</tr>
<tr>
<td>Data EEPROM</td>
<td>GPR</td>
</tr>
<tr>
<td>Data EEPROM</td>
<td>SFR</td>
</tr>
<tr>
<td>GPR</td>
<td>GPR</td>
</tr>
<tr>
<td>GPR</td>
<td>SFR</td>
</tr>
<tr>
<td>SFR</td>
<td>GPR</td>
</tr>
<tr>
<td>SFR</td>
<td>SFR</td>
</tr>
</tbody>
</table>

2.3.2.1 DMA Addressing

DMA source and destination addressing are handled through the DMA Source Start Address (DMAxSSA) register and the DMA Destination Start Address (DMAxDSA) register, respectively.

The DMAxSSA register is a 22 bit wide register that is used to define the read source’s beginning address. When the DMA is enabled and ready for use, the address stored in the DMAxSSA register is copied into the DMA Source Pointer (DMAxSPTR) register. After each DMA transfer, the DMAxSPTR register is instructed to do one of three things as defined by the Source Address Mode Selection (SMODE<1:0>) bits of DMA Control Register 1 (DMAxCON1):

- The source pointer register is incremented
- The source pointer is decremented
- The source pointer remains the same

When the source register is an SFR, such as the UART receive buffer, the SMODE bits may be configured such that the address pointer remains the same. In this case, the UART receive buffer’s address does not change, so the pointer may always point to the same location. When the source register points to an EEPROM location, the SMODE bits may be configured such that the pointer increments after each transfer so that it points to the next EEPROM location.

The DMAxDSA register operates similarly to the DMAxSSA register, with the main exception being the register size. The DMAxDSA register is a 16 bit wide register that stores the write destination’s beginning address. When the DMA is enabled and ready for use, the address stored in the DMAxDSA register is copied into the DMA Destination Pointer (DMAxDPTR) register. After each DMA transfer, the DMAxDPTR register is instructed to do one of three things as defined by the Source Address Mode Selection (DMODE<1:0>) bits of DMAxCON1:

- The destination pointer register is incremented
- The destination pointer is decremented
- The destination pointer remains the same

When the destination register is an SFR, such as the SPI Transmit Buffer, the DMODE bits may be configured such that the destination pointer remains unchanged. In this case, the SPI Transmit Buffer is a fixed address, so the pointer may always point to the same SFR.
2.3.2.2 DMA Message Size and Counter Registers
The DMA Source Size (DMAxSSZ) and DMA Destination Size (DMAxDSZ) registers are used to determine the total number of transactions in a DMA message. Both size registers are 12 bits wide, allowing for a message size of up to 4,096 bytes.

When the DMA is enabled and ready for use, the values loaded into the DMAxSSZ and DMAxDSZ registers are copied into the DMA Source Count (DMAxSCNT) and DMA Destination Count (DMAxDCNT) registers, respectively. Each time a transaction completes, each counter register is automatically decremented by one. When either counter reaches the count of one and decrements, that counter register is automatically reloaded by hardware with the value stored in its respective size register, which will also set either the DMA Source Count Interrupt (DMAxSCNTIF) or DMA Destination Count Interrupt (DMAxDCNTIF) flag, depending on which count register reloaded.

It is important to note that the size registers do not have to be loaded with the same value. For example, in this application, the DMAxSSZ register is loaded with the maximum value of ‘0xFFF’ (4,095 decimal), while the DMAxDSZ register is loaded with a value of ‘1’. In this case, the source size relates to the size of the message and how many address locations are to be read, while the destination size refers to the number of memory locations that will be written. Since the destination is only one memory location, DMAxDSZ is loaded with ‘1’.

As mentioned before, DMA size registers have a maximum value of 0xFFF. If the message is larger than what the size register can hold, the size register can be reloaded, in software, with the size of the remaining message. Of course, the source or destination address must also be updated to point to the correct memory location.

2.3.2.3 Starting/Stopping DMA Message Transfers
DMA transfers can be initiated through either user software control, or by using a hardware trigger.

User software can set the DMA transaction bit (DGO) to start DMA transactions, while clearing the DGO bit will stop DMA transactions. The DGO bit can also be used to indicate whether a transaction is in progress.

A hardware trigger is an interrupt request from another module that is sent to the DMA to initiate DMA transactions. Interrupt sources that can be used as a trigger are defined by the DMA Start Interrupt Request Source Selection (DMAxSIRQ) register. The Start of Transfer Interrupt Request Enable (SIRQEN) bit of DMAxCON0 is used to determine when the hardware trigger source begins sampling the trigger source’s Interrupt flag. Once the trigger source's Interrupt flag becomes set, DMA transactions begin. It is important to note that once the SIRQEN bit is set (by user software), DMA transaction may begin immediately. It is important to have configured and enabled the trigger source module prior to enabling SIRQEN to avoid untimely DMA transactions.

Hardware triggers can be used to stop DMA transactions. The DMA Abort Interrupt Request Source Selection (DMAxAIRQ) register holds the abort trigger interrupt source as selected by the user. The Abort of Transfer Interrupt Request Enable (AIRQEN) bit of DMAxCON0 is used to enable sampling of the trigger source’s Interrupt flag used to terminate transactions. Once an Abort Interrupt signal has been received, the hardware clears the DGO, AIRQEN and SIRQEN bits to prevent further transactions. In this case, the hardware performs a ‘soft-stop’, meaning that the DMA holds its current information, and if further transactions are desired, user software must set the SIRQEN bit.

2.3.2.4 8-Bit Thermal Camera DMA Module Implementation
The thermal camera application uses the DMA to transfer an image file, stored in Program Flash Memory, to the LCD during the Grid-EYE sensor’s required 15-second stabilization delay. In this case, the DMA read source is the Program Memory, while the write destination is the SPI Transmit Buffer (SPI1TXB).
As previously mentioned, the DMA can transfer a message up to 0xFFF (4,096) bytes in length. The Microchip logo image file is 34,868 bytes, therefore, the DMA must be reloaded several times in order to write the entire file to the LCD.

Since the application uses the SPI1TXB to write each byte, the DMA transfer trigger source was configured to use the SPI Transmit Interrupt Flag (SPI1TXIF). Each time the SPI Tx buffer has completed its transmission, the SPI module hardware sets the SPI1TXIF, which indicates that the buffer is empty and ready for new data.

When the DMA source counter value reaches zero, the module hardware sets the DMA Source Count Interrupt Flag (DMA1SCNTIF), which is used as the abort source. When the DMA1SCNTIF is set, an interrupt is generated, indicating the end of the source count. During normal DMA operation, the DMA source and destination registers, and source size and destination size registers, are reloaded with their initial values each time the counter registers reach zero. For the application, this would mean that the entire image would not be written; instead, the DMA would repeatedly write the first 4,095 values to the LCD. To prevent this, the source address is updated to point to the next consecutive address in the image file.

The abort interrupt occurs a total of eight times, and on the final interrupt cycle the source size register is updated (in addition to the source address register) with the total bytes remaining in the image file, which are less than the maximum 0xFFF. It is important to note that, when updating the address and size registers, each time the abort interrupt service routine is entered, the DMA Module Enable (EN) bit of DMAxCON0 register must be cleared before writing to the source (and destination) address and size registers, and then set before exiting the ISR. Example 2-3 shows the abort interrupt service routine code used in the application. For more information on the DMA module in its entirety, please refer to technical brief TB3164.

Example 2-3. DMA Abort Interrupt Service Routine Code Snippet

```c
void DMA1_DMAA_ISR(void)
{
    PIR2bits.DMA1AIF = 0;
    if (ISRcount == 0)
    {
        ISRcount++;
        DMA1CON0bits.EN = 0;
        DMAISSA = 0x87DE;
        PIR2bits.DMA1SCNTIF = 0;
        DMA1CON0bits.AIRQEN = 1;
        DMA1CON0bits.EN = 1;
        DMA1CON0bits.SIRQEN = 1;
    }
    else if(ISRcount == 1)
    {
        ISRcount++;
        DMA1CON0bits.EN = 0;
        DMAISSA = 0x97DD;
        PIR2bits.DMA1SCNTIF = 0;
        DMA1CON0bits.AIRQEN = 1;
        DMA1CON0bits.EN = 1;
        DMA1CON0bits.SIRQEN = 1;
    }
    else if(ISRcount == 2)
    {
        ISRcount++;
        DMA1CON0bits.EN = 0;
        DMAISSA = 0xA7DC;
        PIR2bits.DMA1SCNTIF = 0;
        DMA1CON0bits.AIRQEN = 1;
        DMA1CON0bits.EN = 1;
        DMA1CON0bits.SIRQEN = 1;
    }
```
else if(ISRcount == 3)
{
ISRcount++;
DMA1CON0bits.EN = 0;
DMAISSA = 0xB7DB;
PIR2bits.DMAISCNTIF = 0;
DMAICON0bits.AIRQEN = 1;
DMAICON0bits.EN = 1;
DMAICON0bits.SIRQEN = 1;
}
else if(ISRcount == 4)
{
ISRcount++;
DMAICON0bits.EN = 0;
DMAISSA = 0xC7DA;
DMAICON0bits.AIRQEN = 1;
DMAICON0bits.EN = 1;
DMAICON0bits.SIRQEN = 1;
}
else if(ISRcount == 5)
{
ISRcount++;
DMAICON0bits.EN = 0;
DMAISSA = 0xD7D9;
DMAICON0bits.AIRQEN = 1;
DMAICON0bits.EN = 1;
DMAICON0bits.SIRQEN = 1;
}
else if(ISRcount == 6)
{
ISRcount++;
DMAICON0bits.EN = 0;
DMAISSA = 0xE7D8;
DMAICON0bits.AIRQEN = 1;
DMAICON0bits.EN = 1;
DMAICON0bits.SIRQEN = 1;
}
else if(ISRcount == 7)
{
ISRcount++;
DMAICON0bits.EN = 0;
DMAISSA = 0xF7D7;
DMAISSZ = 0x083C;
PIR2bits.DMAISCNTIF = 0;
DMAICON0bits.AIRQEN = 1;
DMAICON0bits.EN = 1;
DMAICON0bits.SIRQEN = 1;
}
else
{
DMAICON0bits.EN = 0;
}

2.3.3 \textit{I}^2\text{C} Module

The PIC18F27K42 offers two independent \textit{I}^2\text{C} modules. The \textit{I}^2\text{C} module provides a synchronous serial interface between the microcontroller and other \textit{I}^2\text{C} compatible devices.

The \textit{I}^2\text{C} module allows devices to communicate on a two-wire bus that operates in a multiple-master, multiple-slave environment. The \textit{I}^2\text{C} bus utilizes two signal connections:

- Serial Clock (SCL)
- Serial Data (SDA)
The I^2C bus consists of bidirectional, open-drain lines that require pull-up resistors connected to the supply voltage. Pulling the line to ground constitutes a logic ‘0’, while allowing the line to float constitutes a logic ‘1’.

The master device always generates the SCL signal, meaning that all transactions on the bus must be initiated by a master. The I^2C module can be configured as either a master or slave, but for this application, the module is configured in 7-bit Master mode.

### 2.3.3.1 I^2C Master Mode

Communication in I^2C Master mode begins when the master issues a Start condition, followed by the 7-bit or 10-bit address of the slave it wishes to communicate with. In 7-bit Address mode, the Least Significant bit (LSb) of the address byte is considered the Read/Write (R/W) bit, while in 10-bit mode, the LSb of the high address byte is considered the R/W bit. If the slave device is present on the bus, it must send an active-low Acknowledge (ACK) bit back to the master during the ninth clock pulse of the SCL signal. When the ACK bit is received by the master, it can continue to transmit or receive data. When the master device has completed the desired transaction, it must send a Stop condition, indicating that it has completed the data transfer. Alternatively, the master may issue a Restart condition if it intends to keep control of the bus.

The I^2C module contains two address buffer registers, I2CxADB0 and I2CxADB1. The buffers are used as either transmit or receive address buffers, depending on the operation mode. In 7-bit Master mode, I2CxADB1 is used to store the slave’s address byte, including the R/W bit, while I2CxADB0 is unused. The Address Buffer Disable (ABD) bit of the I^2C Control Register 2 (I2CxCON2) enables/disables the address buffer. When ABD is clear, the address buffers are enabled, meaning that the slave’s address will be loaded into the I^2C Transmit Buffer (I2CxTXB) by hardware; when ABD is set, user software must load the slave address into I2CxTXB.

The I^2C module has independent transmit and receive buffers, I2CxTXB and I2CxRXB, respectively. When new data are received by the I2CxRXB register, the Receive Buffer Full (RXBF) Status bit of the I^2C Status Register 1 (I2CxSTAT1) is set, as is the I^2C Receive Interrupt Flag (I2CxRXIF). Reading I2CxRXB clears both the RXBF and I2CxRXIF bits.

When new data are ready to be transmitted, the Transmit Buffer Empty (TXBE) Status bit of I2CxSTAT1 must be set before data are loaded in order to avoid write collision errors. Writing to I2CxTXB clears the TXBE bit, and once the data has transmitted and the buffer is empty, the TXBE bit is set, as is the I^2C Transmit Interrupt Flag (I2CxTXIF).

The I^2C Byte Count Register (I2CxCNT) keeps track of the number of bytes contained in a full I^2C message. The I2CxCNT register is an 8-bit wide register that allows packet sizes up to 256 bytes. Each time a data byte (not address bytes) is transmitted or received, the I2CxCNT register is decremented by one by module hardware. When the module receives a data byte, I2CxCNT is decremented on the eighth falling edge of SCL. When the module completes a byte transmission, I2CxCNT decrements on the ninth falling edge of SCL. When I2CxCNT reaches zero, the Byte Count Interrupt Flag (CNTIF) of the I^2C Interrupt Flag Register (I2CxPIR) is set.

I^2C pin configuration is handled by the RxyI2C registers. The ‘x’ in RxyI2C refers to the I/O port, and the ‘y’ refers to the port pin. For example, this application uses pins RC3 (SCL) and RC4 (SDA) for I^2C communication. These registers determine the slew rate, pin logic levels, and internal pull-up configuration for each pin. In this case, the I^2C-specific slew rate limiting is used, the logic levels are set to match the I^2C specification levels, and no internal pull-ups are used.

For a more detailed explanation of the I^2C module in Master mode, please refer to technical brief TB3191.
2.3.3.2 8-Bit Thermal Camera I²C Module Implementation

The I²C module is used to configure and read temperature data from the Grid-EYE sensor. The module is configured to operate at a bus speed of 100 kHz, and uses address buffer I2C1ADB1 to store the sensor’s I²C address.

When the camera is powered on, the I²C module writes configuration data to the sensor’s internal registers. In this case, we are only interested in the sensor’s frame rate and moving average configuration. The sensor’s frame rate was configured to use the 1 Hz frame rate and enabled the moving average feature. The 1 Hz frame rate is the average of ten consecutive frames, so instead of reading the pixel data ten times and performing an average calculation in software, we allow the sensor to perform the calculation, reducing software overhead. The moving average feature simply means that the sensor’s previous frame data are added to the new frame data and divided by two. This helps to keep the image from flickering due to noise.

Once the Grid-EYE sensor has been configured and the 15-second delay has passed, the I²C module is used to collect the sensor’s thermistor and pixel temperature data. The thermistor requires a 2-byte read to recover its 12-bit temperature data, which is broken down into two bytes. Example 2-4 shows the I²C code used to obtain the thermistor data.

```
Example 2-4. I²C 16-Bit Read Operation Code Snippet

int16_t i2c1_read2ByteRegisterInt_16(uint8_t address, uint8_t reg)
{
    int16_t result;
    uint8_t dataLowByte;
    uint16_t dataHighByte;
    
    I2C1ADB1 = (uint8_t)(address << 1);   // Load slave address and shift
    I2C1TXB = reg;                        // Load slave register address
    I2C1CNT = 1;                          // Load count
    I2C1CON0bits.RSEN = 1;                // Set RSEN for master read
    I2C1CON0bits.S = 1;                   // Set Start to get things going
    while(!I2C1CON0bits.MDR);            // Wait until master is ready to receive
    address = (uint8_t)(address << 1);    // Load slave address and shift
    I2C1ADB1 = (uint8_t)(address | 0x01); // Load address with Read enabled
    I2C1CNT = 2;                          // Load with expected data size
    I2C1CON0bits.S = 1;                   // Set Start to begin Restart condition
    I2C1CON0bits.RSEN = 0;                // Clear RSEN
    while(!I2C1STAT1bits.RXBF);           // Wait until buffer receives data
    dataLowByte = I2C1RXB;               // Read low byte
    while(!I2C1STAT1bits.RXBF);           // Wait until buffer receives data
    dataHighByte = I2C1RXB;              // Read high byte
    if(dataHighByte & 0x08)               // Is data negative?
    {
        dataHighByte = dataHighByte | 0xF8;
    }
    result = dataHighByte << 8;
    result = result | dataLowByte;        // Form 16-bit word
    wait4Stop();                          // Wait for Stop Condition
    return result;
}
```

Reading the sensor’s pixel data requires a block read of the pixel registers. Each pixel contains a 12-bit temperature value broken into two individual bytes, and since there are a total of 64 pixels, the I²C performs a block read of 128 bytes. Luckily, the pixel data region is configured sequentially, meaning that the I²C can transmit a single slave address, followed by a single register address, but will receive all 128 bytes in a single transaction. After each pixel register is read, the sensor automatically points to the next register, so there is no need to start a new communication packet each time a pixel register is read. Example 2-5 shows the method for reading the pixel data.
**Example 2-5. I^2C Block Read Code Snippet**

```c
void i2c1_readDataBlock(uint8_t address, uint8_t reg, char *data, uint8_t len)
{
    I2C1ADB1 = (uint8_t)(address << 1);   // Load slave address and shift
    I2C1TXB = reg;                 // Load slave starting register address
    I2C1CNT = 1;                   // Load count
    I2C1CON0bits.RSEN = 1;         // Set RSEN for master read
    I2C1CON0bits.S = 1;            // Set Start to get things going
    while(!I2C1CON0bits.MDR);      // Wait until master is ready to receive data
    address = (uint8_t)(address << 1);    // Load slave address and shift
    I2C1ADB1 = (uint8_t)(address | 0x01); // Load address with Read enabled
    I2C1CNT = len;                 // Load length of expected packet in bytes
    I2C1CON0bits.S = 1;            // Set Start to begin Restart condition
    I2C1CON0bits.RSEN = 0;         // Clear RSEN
    while(I2C1CNT)                 // While there is a count
    {
        while(!I2C1STAT1bits.RXBF);  // Wait until buffer receives data
        *data++ = I2C1RXB;           // Write into array, hardware decrements count
    }
    wait4Stop();                   // Wait for hardware to issue a Stop
}
```

### 2.3.4 SPI Module

The PIC18F27K42 offers a single Serial Peripheral Interface (SPI) module. The SPI is a synchronous serial communication bus. Typically, SPI operates in Full-Duplex mode, meaning that the master and slave transmit data from one to the other simultaneously. The PIC18F27K42's SPI module can operate in Transmit-Only or Receive-Only modes in addition to Full-Duplex mode. SPI devices communicate in a single master, multiple slave environment in which the master always initiates communication. The SPI can operate in either Master mode or Slave mode.

The SPI bus is composed of four signal connections:

- **Serial Clock (SCK)**
- **Serial Data Out (SDO)** – sometimes referred to as Master Out Slave In (MOSI)
- **Serial Data In (SDI)** – sometimes referred to as Master In Slave Out (MISO)
- **Slave Select (SS)** – sometimes referred to as Chip Select (CS)

The SCK signal is always generated by the master. SDO data are transmitted on the clock edge as determined by the Clock Edge Select (CKE) bit of SPI Configuration Register 1 (SPI1CON1). SDI data are received into the receive buffer on the opposite clock edge. Clock polarity is determined by the Clock Polarity Select (CKP) bit of SPI1CON1. It is important to note that the master and slave devices may be configured such that the clock polarity is the same for both devices.

The SPI module uses a transfer counter to help track data transfers. This is handled by the SPI Transfer Counter Register (SPITCNT), the SPI Transfer Width Register (SPI1TWIDTH), and the Bit-Length Mode Select (BMODE) bit of SPI1CON0. SPITCNT is an 11-bit wide register that determines the number of bits to transfer. The SPI1TWIDTH register determines the size, in bits, of each transfer counted by SPITCNT. The BMODE bit determines how the SPI1TWIDTH settings are applied.

When BMODE is set, the SPI1TWIDTH settings apply to every byte in the packet. In this case, the total number of bits to transfer is the product of the SPI1TCNT value multiplied by the SPI1TWIDTH setting. For example, if a packet size is 100 bytes in length, the value loaded into SPI1TCNT would be 800 (100 bytes × 8 bits/byte), and since each byte is composed of eight bits, SPI1TWIDTH is set to ‘000’, which is the setting for eight bits.
Note: It is important to note that when BMODE is set in Transfer-Only mode, the SPI1TCNT register is ignored. In other words, Transfer-Only mode allows the user to transmit as much data as the application allows without having to load the SPI1TCNT with a non-zero value.

2.3.4.1 8-Bit Thermal Camera SPI Module Implementation

The SPI module is used to configure and write color information to the LCD driver found in the Varitronix COG-C144MVGI-08 graphic display. The module is configured in Transmit-Only mode at a SCK speed of 8 MHz.

Transmit-only configuration allows one-way transfers from the master to the slave device without the need for the master to read its SDI input. The Transmit Data-Required Control (TXR) and Receive FIFO Space-Required Control (RXR) bits of the SPI Configuration Register 2 (SPI1CON2) determine the Transfer mode. In this case, the TXR bit is set, and the RXR bit is clear, which puts the module into Transmit-Only mode. In Transmit-Only mode, module hardware automatically clears the receive buffer, which allows the SPI master to continuously transmit without the need for software intervention to read and clear the receive buffer. This results in faster overall cycle times since the extra instructions typically required to clear the receive buffer are eliminated.

Each image frame is composed of 17,434 16-bit words, which means each frame will require the SPI to transmit 34,868 8-bit bytes for each frame. As one can see, even saving one instruction cycle each time the SPI writes a byte of data would amount to 34,868 saved instructions, which means the SPI can write its data that much quicker. This helps prevent image lag from frame to frame.

To further simplify data transfers, the BMODE bit is set, so that no transfer counter intervention is required. Instead, software tracks the number of bytes contained in each packet. Each of the frames transferred to the LCD contains thousands of bits, which would require reloading the transfer counter several times for each frame. This would require more instructions to complete, slowing down the transfer process.

Example 2-6 shows the SPI transfer routine for an 8-bit (single byte) transfer. This routine is used primarily to write command information to the LCD driver, as well as row and column locations.

Example 2-6. SPI Transmit-Only (Single Byte Transfer)

```c
void SPI1TransmitOnly8bit(uint8_t data)
{
    SPI1TXB = data;                        // Load data
    while(SPI1STATUSbits.TXBE == 0);       // Wait until buffer is clear
}
```

Example 2-7 shows the SPI transfer routine for a 16-bit (two-byte) transfer. This routine is used to write color and coordinate data to the LCD.

Example 2-7. SPI Transmit-Only (Two-Byte Transfer)

```c
void SPI1TransmitOnly16bit(uint8_t MSB, uint8_t LSB)
{
    SPI1TXB = MSB;                          // Load MSB
    while(SPI1STATUSbits.TXBE == 0);        // Wait until buffer is clear
    SPI1TXB = LSB;                          // Load LSB
    while(SPI1STATUSbits.TXBE == 0);        // Wait until buffer is clear
}
```
3. **Thermal Camera Application**

The thermal camera is a low-resolution infrared detection system. The camera uses the Grid-EYE sensor to read an object’s temperature as well as the temperatures around the object. The data are read by the PIC18F27K42 over the I²C bus. The PIC then converts the raw temperature data into a series of color codes, which are transferred over the SPI bus to the LCD display.

3.1 **Initializing the Application**

When the application board is first powered on, the PIC must go through its initialization process. This process configures the necessary operation features, such as selecting the system clock source and speed, and configuring and enabling input and output pins. The initialization process also configures the peripherals required for the application, in this case, the Timer1, SPI, I²C, and DMA modules. Once the PIC’s peripherals are ready for use, the PIC can configure and enable the Grid-EYE sensor and LCD.

3.2 **LCD Initialization**

Example 3-1 shows the Initialization sequence for the LCD. This particular sequence initializes the LCD for the splash screen (Microchip logo). The sequence begins by pulling the LCD reset line low, which instructs the LCD to perform a Reset on its internal registers. Next, the LCD ‘A0’ line is pulled low, which puts the LCD into Command mode. Command mode allows the LCD Control registers to be configured. Features such as contrast control, frame frequency, display pattern settings, and row and column beginning and end points, are configured while in Command mode. Once these registers have been configured, the ‘A0’ line is pulled high, which enters Data mode, allowing color information to be written.

![Example 3-1. LCD Initialization](image-url)
3.3 Initializing the Grid-EYE® Sensor

Example 3-2 shows the Initialization sequence for the Grid-EYE sensor. Configuration data are transmitted over the I²C bus and written into the sensor’s frame rate and average registers, setting the frame rate to 1 Hz, and enabling the moving average feature. Once these two registers have been configured, software enables Timer1. Timer1 is used to create the 15-second delay required by the Grid-EYE sensor.

Example 3-2. Grid-EYE® Sensor Initialization

```c
void Grid_Eye_Init(void)
{
    i2c1_write1ByteRegister(GRIDEYE_ADR_GND, FPSC_REG, ONEHZFRAMERATE);
    i2c1_write1ByteRegister(GRIDEYE_ADR_GND, AVE_REG, MOVINGAVERAGE_ON);
    TMR1_StartTimer();
}
```

3.4 Enabling the Splash Screen

Once the Grid-EYE sensor has been enabled and Timer1 begins to count, software enables the DMA, which had already been configured during initialization. The DMA1CON0 register contains the Enable (EN), Source Interrupt Request Enable (SIRQEN), and Abort Interrupt Request Enable (AIRQEN) bits, which are all set to enable the DMA and monitor transactions.

Each time a data byte has been transmitted out of the SPI1TXB register, the SPI1TXIF is set, indicating an empty buffer. The SIRQEN hardware polls for this Interrupt flag, and when set, the DMA will load the next data byte into the SPI1TXB. This process continues until the AIRQEN source, in this case the DMA1SCNTIF, becomes set.

When the DMA1SCNTIF bit is set, the DMA will abort further transactions by clearing the SIRQEN and AIRQEN bits to avoid unwanted interrupts or transactions from occurring. Since the AIRQEN interrupt is enabled, software will jump into the DMA Abort ISR. In the Abort ISR, the DMA source address is updated, Interrupt flags are cleared, and the SIRQEN and AIRQEN bits are set in software, which allow the DMA to continue (see Example 2-3 for the Abort ISR source code).

After the complete image has been transferred to the LCD, the DMA is disabled, and the image will persist on screen until Timer1 finishes its delay. At this point, the camera begins to collect temperature data.

3.5 Image Processing

Image processing is handled by the PIC microcontroller in software. Image processing takes the temperature data acquired from the Grid-EYE sensor’s pixel array and converts the data into an image that can be observed on the LCD. Example 3-3 shows the main operation routine. The routine is composed of each of the steps, from reading temperature data to writing an image on the LCD.

Example 3-3. Image Processing Steps

```c
void Operate_IR_Camera(void)
{
    Grid_Eye_Read_Therm();            // Go read thermistor
    Grid_Eye_Read_Pixels();           // Get pixel temperatures
    Grid_Eye_Convert_Temps();         // Convert from 8-bit to 16-bit values
    Grid_Eye_Temp_Diff();             // Find temp differences (object detection)
```
3.5.1 Grid_Eye_Read_Therm()

The Grid_Eye_Read_Therm() routine commands the PIC microcontroller to read the Grid-EYE sensor’s on-board thermistor. The Grid-EYE sensor’s Thermistor register holds the 12-bit thermistor value, and is read via the I²C bus. The thermistor has a 1 LSb (Least Significant bit) resolution of 0.0625°C, therefore the value read by the PIC microcontroller must be properly scaled (see Example 3-4).

Example 3-4. Grid_Eye_Read_Therm() Routine

```c
void Grid_Eye_Read_Therm(void)         // Read thermistor (built-in) {
  int16_t rawResult;
  rawResult = i2c1_read2ByteRegisterInt_16(GRIDEYE_ADR_GND, THERM_REG);
  rawResult = rawResult * 0.0625;      // Scale value based on 1 LSB = 0.0625°C
  aveTemp = rawResult;                 // Scaled thermistor value
}
```

3.5.2 Grid_Eye_Read_Pixels()

The Grid_Eye_Read_Pixels() routine reads the Grid-EYE sensor’s 128 Pixel Temperature registers. The data are loaded into a 2 x 128 byte array. When the routine is entered, the PIC microcontroller reads the Grid-EYE sensor’s pixel array, stores the data, then reads the Pixel registers a second time and stores the data. Reading the Pixel registers twice will allow for an average of the two readings, which help with image stability (see Example 3-5).

It is important to note that the sensor’s pixel array field of view is upside down and backwards (see Figure 3-1). Pixel number 64 is located on the top left corner in its field of view, while pixel number 1 is located at the bottom right corner. The Pixel Data registers are ordered beginning with pixel number 1 and ending with pixel 64. The sensor has a pointer that increments each time a Pixel register is read, so reading sequentially is the fastest way to acquire the data, but will require the image to be ‘flipped’ around both the ‘X’ and ‘Y’ axes, otherwise the image will appear upside down and backwards to the viewer.

Example 3-5. Grid_Eye_Read_Pixels() Routine

```c
void Grid_Eye_Read_Pixels(void)           // Get pixel temperature data {
  i2c1_readDataBlock(GRIDEYE_ADR_GND, T01L_REG, &rawPixelData[0][0], RAWPIXELS);
  __delay_ms(100);                         // Wait while pixel temps are updated
  i2c1_readDataBlock(GRIDEYE_ADR_GND, T01L_REG, &rawPixelData[1][0], RAWPIXELS);
}
```
3.5.3 **Grid_Eye_Convert_Temps()**

The `Grid_Eye_Convert_Temps()` routine takes the 128 8-bit temperature values and converts them into 16-bit values. Each of the Grid-EYE sensor's pixels generates 12-bit (11 temperature bits plus 1 Sign bit) temperature values which are stored in two 8-bit registers. The `Grid_Eye_Convert_Temps()` routine combines each Pixel's high and low Temperature registers to form the 16-bit integer equivalent. The routine does this for the entire dynamic array, and then takes an average of the two pixel readings. Once the average for each pixel has been calculated, the averaged pixel data are stored in a new array, which is used in later routines. Additionally, the average of all pixels is taken, and then combined with the thermistor reading to create a background temperature reference (see Example 3-6). The background temperature reference is used to determine if an object within the sensor's field of view is hotter than the average temperature surrounding it.

### Example 3-6. Grid_Eye_Convert_Temps() Routine

```c
void Grid_Eye_Convert_Temps(void)
{
    uint8_t highPixelCount = 1;  // Keep track of upper and lower bytes
    uint8_t lowPixelCount = 0;
    int16_t temp = 0;
    uint8_t index = 0;

    for(uint8_t j = 0; j < FRAME_NUMBER; j++) // Start conversions
    {
        for(uint8_t i = 0; i < CONVERTEDPIXELS; i++) // Start conversions
        {
            if(rawPixelData[j][highPixelCount] & 0x08) // Check for negative values
            {
                rawPixelData[j][highPixelCount] | 0xF8; // Convert to 16-bit signed
            }

            temp = rawPixelData[j][highPixelCount]; // Convert 8-bits into one 16-bit
            temp = temp << 8;
            temp = temp + rawPixelData[j][lowPixelCount];
            temp = temp * 0.25;                        // Scale (1 LSB = 0.25C)

            pixelData[j][index++] = temp;
            highPixelCount += 2;                      // Increase counter to get correct byte
            lowPixelCount += 2;
        }
    }
}
```

3.5.4 Grid_Eye_Temp_Diff()

The Grid_Eye_Temp_Diff() routine compares the averaged pixel values to the background average to determine if an object whose temperature is higher than the background temperature average is present within the sensor’s field of view. The difference between a pixel’s temperature and the background temperature is compared to a threshold value. When the temperature difference is greater than the threshold limit, a value of ‘1’ is written into the pixel’s placeholder location within a new array. Conversely, if the temperature difference is less than the threshold limit, a ‘0’ is written into the array (see Example 3-7). After all pixels have been compared, the array locations containing a ‘1’ value will be used to form an image, while the locations containing a ‘0’ value will be considered the image’s background (see Figure 3-2).

Example 3-7. Grid_Eye_Temp_Diff() Routine

```c
void Grid_Eye_Temp_Diff(void) {
    int8_t tempDelta = 0;
    for(uint8_t i = 0; i < CONVERTEDPIXELS; i++) // Find diff for each pixel
    {
        tempDelta = avePixelValue[i] - aveTemp; // Compare pixel temp to thermistor
        if(tempDelta > DIFF_THRESHOLD) // Compare difference to threshold
        {
            temp_diff[i] = 1; // If above threshold, object detected
        }
        else
        {
            temp_diff[i] = 0; // No object, background data
        }
    }
}
```
3.5.5 Grid_Eye_Build_Img()

The Grid_Eye_Build_Img() routine uses the 'temp_diff[]' array (see Example 3-7) to begin creating an image. In this routine, any 'temp_diff[]' array location containing a '1' will allow its corresponding pixel location in the 'avePixelValue[]' array to keep its current temperature value. Any 'temp_diff[]' array location containing a '0' will have the corresponding pixel location in the 'avePixelValue[]' array to be loaded with a zero value (see Example 3-8). Figure 3-3 builds upon the image of Figure 3-2, replacing the '1' values with actual temperature data.

### Example 3-8. Grid_Eye_Build_Img() Routine

```c
void Grid_Eye_Build_Img(void)
{
  for(uint8_t i = 0; i < CONVERTEDPIXELS; i++)
  {
    if(temp_diff[i] == 0)           // If zero in array, clear pixel temp
    {
      avePixelValue[i] = 0;        // 0 = background will be dark
    }
    else                            // If one in array, keep pixel temp
    {
      avePixelValue[i] = avePixelValue[i];
    }
  }
}
```

---

Figure 3-2. Object Within the Field of View
3.5.6 Grid_Eye_LinearInterpolation()

The Grid_Eye_LinearInterpolation() routine converts the 8x8 image array to a size of 32x32 (see Example 3-9). The Grid-EYE sensor’s array is 8 pixels in the ‘X’ direction and 8 pixels in the ‘Y’ direction. If we were to observe this 64-pixel array on an LCD, the image would be too small to see. In order to properly view the image, the image must be expanded.

Linear interpolation is the process of finding an unknown value between two known values on a line. In other words, linear interpolation uses the information we already have to fill in the missing information needed to expand the image.

The equation below shows the interpolation formula used to find an unknown value between two known values.

**Equation 3-1. Linear Interpolation**

\[
y - y_1 = \frac{y_2 - y_1}{x_2 - x_1} \cdot (x - x_1)
\]

where:

- \(y\) = unknown value
- \(x\) = desired ‘x’ coordinate
- \(y_2, y_1\) = known values
- \(x_2, x_1\) = known ‘x’ coordinates

**Example 3-9. Grid_Eye_LinearInterpolation() Routine**

```c
void Grid_Eye_LinearInterpolation(void)
{
    const uint8_t res = 16;                            // 16-bit resolution
    uint8_t image_width = IMAGE_SIZE_X;                // Expanded image width
    uint8_t image_height = IMAGE_SIZE_Y;               // Expanded image height

    for(uint16_t imageIndex = 0; imageIndex < (image_width * image_height); imageIndex++)
```

© 2018 Microchip Technology Inc.
For the application, the bilinear interpolation method is used. In this case, software takes the values of four neighboring pixels, applies a scaling factor to each of the four pixels, and takes the average of the four scaled pixels and applies that value to the newly created pixel. The scaling factor depends on the distance the newly created pixel is from the original pixel; the further away the new pixel is, the smaller the scale factor (see Figure 3-4). After the interpolation calculations have been completed, an array consisting of 1,024 16-bit temperature values is updated.
3.5.7 LCD_Write_Data()

The final step of image processing involves writing color codes to the LCD. As mentioned in Section 2.2 “128x128 Pixel RBG LCD”, the LCD accepts 16-bit color codes in the standard RGB565 format. The LCD_Write_Data() routine converts the 16-bit temperature values into 16-bit color codes. The color codes are stored in an array, which creates a custom color palette. The array consists of 80 color codes representing temperatures from 0 degrees to 80 degrees C. For example, if a temperature is 75°C, the 75th element in the color array represents the color code to match the 75°C temperature value. This color code is then written to the LCD.

Additionally, the LCD_Write_Data() routine applies another interpolation technique to further expand the image (see Example 3-10). After the linear interpolation step above, the original 8x8 pixel array was expanded to a 32x32 pixel array. Unfortunately, the PIC18F27K42 does not have enough RAM to allow for an array to hold an image that will directly fit into the 128x128 pixel LCD. Instead, the image is expanded by writing a single pixel to the LCD into a 4x4 block (see Figure 3-5). This technique allows the entire 128x128 LCD screen to be filled without having to worry about the lack of memory needed to further expand the image array.

Example 3-10. LCD_Write_Data() Routines

```c
void LCD_Write_Data(void)
{
    LCD_A0_SetLow();
    NOP();
    SPI1TransmitOnly8bit(ROW_ADDRESS_AREA_SET); // Set row coordinates
    SPI1TransmitOnly8bit(LCDROWSTARTADDRESS); // Set row start address
    SPI1TransmitOnly8bit(LCDROWENDADDRESS); // Set row end address
    SPI1TransmitOnly8bit(COLUMN_ADDRESS_AREA_SET); // Set column coordinates
    SPI1TransmitOnly8bit(LCDCOLUMNSTARTADDRESS); // Set column start address
    SPI1TransmitOnly8bit(LCDCOLUMNENDADDRESS); // Set column end address
    NOP();
    LCD_A0_SetHigh();
    NOP();
}
```
for(uint8_t i = 0; i < 32; i++)                  // Write to each row
{
    LCD_Write_Column();                      // Write to each column in a row
    oldColumnStartAddress = LCDCOLUMNSTARTADDRESS; // return to origin
    oldColumnEndAddress = LCDCOLUMNENDADDRESS;
    oldRowStartAddress += ADDER;                   // Increment row
    oldRowEndAddress += ADDER;
    LCD_A0_SetLow();
    NOP();
    SPI1TransmitOnly8bit(ROW_ADDRESS_AREA_SET);    // New row addresses
    SPI1TransmitOnly16bit(oldRowStartAddress, oldRowEndAddress);
    SPI1TransmitOnly8bit(COLUMN_ADDRESS_AREA_SET); // New column addresses
    SPI1TransmitOnly16bit(oldColumnStartAddress, oldColumnEndAddress);
    NOP();
    LCD_A0_SetHigh();
    NOP();
}

index1 = 0;                            // Clear index counter for next frame
}

void LCD_Write_Column(void)              // Write to each column
{
    for(uint8_t j = 0; j < 32; j++)        // There will be 32 row/column blocks
    {
        tempData = output_image[index1++];   // Get temperature for a pixel
        colorCode = colorArray[tempData];    // Find the matching color code
        colorCodeLowByte = colorCode & 0xFF; // Convert 16-bit color code
        colorCodeHighByte = colorCode >> 8;
        for(uint8_t k = 0; k < 16; k++)      // Each pixel must be enlarged, so
            write to a 4x4 block
            SPI1TransmitOnly16bit(colorCodeHighByte, colorCodeLowByte);
        oldColumnStartAddress += ADDER;       // After each block, move to next block
        oldColumnEndAddress += ADDER;
        LCD_A0_SetLow();
        NOP();
        SPI1TransmitOnly8bit(COLUMN_ADDRESS_AREA_SET);
        SPI1TransmitOnly16bit(oldColumnStartAddress, oldColumnEndAddress);
        NOP();
        LCD_A0_SetHigh();
    }
}

Figure 3-5. LCD_Write_Data() Routine
4. Conclusion
This application note shows how to build an inexpensive, low-resolution thermal camera. For more information on the PIC18F27K42, or any other Microchip products, please visit www.microchip.com.
The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user’s guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Customer Change Notification Service

Microchip’s customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.


Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip’s Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip’s code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, AnyRate, AVR, AVR logo, AVR Freaks, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, Heldo, JukeBox, KeeLoq, Kleer, LANCheck, LINK MD, maXSylus, maXTouch, MediaLB, megaAVR, MOST, MOST logo, MPLAB, OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, Prochip Designer, QTos, SAM-BA, SpyNIC, SST, SST Logo, SuperFlash, tinyAVR, UNI/O, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

ClockWorks, The Embedded Control Solutions Company, EtherSynch, Hyper Speed Control, HyperLight Load, IntelliMOS, mTouch, Precision Edge, and Quiet-Wire are registered trademarks of Microchip Technology Incorporated in the U.S.A.


SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

Silicon Storage Technology is a registered trademark of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.
Quality Management System Certified by DNV

ISO/TS 16949
Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company’s quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001:2000 certified.
<table>
<thead>
<tr>
<th>AMERICAS</th>
<th>ASIA/PACIFIC</th>
<th>ASIA/PACIFIC</th>
<th>EUROPE</th>
</tr>
</thead>
</table>
| Corporate Office  
2355 West Chandler Blvd.  
Chandler, AZ 85224-6199  
Tel: 480-792-7200  
Fax: 480-792-7277  
Technical Support:  
http://www.microchip.com/support  
Web Address:  
www.microchip.com | Australia - Sydney  
Tel: 61-2-9868-6733 | India - Bangalore  
Tel: 91-80-3090-4444 | Austria - Wels  
Tel: 43-7242-2244-39 |
| | China - Beijing  
Tel: 86-10-8569-7000 | India - New Delhi  
Tel: 91-11-4160-8631 | Denmark - Copenhagen  
Tel: 45-4450-2828 |
| | China - Chengdu  
Tel: 86-28-8665-5511 | India - Pune  
Tel: 91-20-4121-0141 | Finland - Espoo  
Tel: 358-9-4520-820 |
| | China - Chongqing  
Tel: 86-23-8980-9588 | Japan - Osaka  
Tel: 81-6-6152-7160 | France - Paris  
Tel: 33-1-69-53-63-20 |
| | China - Dongguan  
Tel: 86-769-8702-9880 | Japan - Tokyo  
Tel: 81-3-6880-3770 | Germany - Garching  
Tel: 49-8931-9700 |
| | China - Guangzhou  
Tel: 86-20-8755-8029 | Korea - Daegu  
Tel: 82-53-744-4301 | Germany - Haan  
Tel: 49-2129-3766400 |
| | China - Hangzhou  
Tel: 86-571-8792-8115 | Korea - Daegu  
Tel: 82-53-744-4301 | Germany - Heilbronn  
Tel: 49-7131-67-3636 |
| | China - Hong Kong SAR  
Tel: 852-2943-5100 | Malaysia - Kuala Lumpur  
Tel: 60-3-7651-7906 | Germany - Karlsruhe  
Tel: 49-721-625370 |
| | China - Nanjing  
Tel: 86-25-8473-2460 | Malaysia - Penang  
Tel: 60-4-227-8870 | Germany - Munich  
Tel: 49-89-627-144-0 |
| | China - Qingdao  
Tel: 86-632-8502-7355 | Philippines - Manila  
Tel: 63-2-634-9065 | Germany - Rosenheim  
Tel: 49-8031-354-560 |
| | China - Shanghai  
Tel: 86-21-3326-8000 | Singapore  
Tel: 65-6334-8870 | Israel - Ra’anana  
Tel: 972-9-744-7705 |
| | China - Shenyang  
Tel: 86-24-2334-2829 | Taiwan - Hsin Chu  
Tel: 886-3-577-8366 | Italy - Milan  
Tel: 39-0331-742611 |
| | China - Shenzhen  
Tel: 86-755-8864-2200 | Taiwan - Kaohsiung  
Tel: 886-7-213-7830 | Fax: 39-0331-468781 |
| | China - Suzhou  
Tel: 86-186-6233-1526 | Taiwan - Taipei  
Tel: 886-2-2506-8600 | Italy - Padova  
Tel: 39-049-7625286 |
| | China - Wuhan  
Tel: 86-27-5980-5300 | Thailand - Bangkok  
Tel: 66-2-694-1351 | Netherlands - Drunen  
Tel: 31-416-690399 |
| | China - Xian  
Tel: 86-29-8833-7252 | Vietnam - Ho Chi Minh  
Tel: 84-28-5448-2100 | Fax: 31-416-690340 |
| | China - Xiamen  
Tel: 86-992-2388138 | | Norway - Trondheim  
Tel: 47-72864388 |
| | China - Zhuhai  
Tel: 86-755-3210040 | | Poland - Warsaw  
Tel: 48-22-3325737 |
| | | | Romania - Bucharest  
Tel: 40-21-407-87-60 |
| | | | Spain - Madrid  
Tel: 34-91-708-08-90 |
| | | | Sweden - Gothenberg  
Tel: 46-31-704-60-40 |
| | | | Sweden - Stockholm  
Tel: 46-8-5090-4654 |
| | | | UK - Wokingham  
Tel: 44-118-921-5800 |

© 2018 Microchip Technology Inc.