OVERVIEW

The Layad Circuits' Kimat TV module allows the user to drive any television/monitor with A/V (via RCA jacks) inputs using an Arduino board as a host device for displaying monochrome text. The module also has built-in push buttons that can be used as inputs.

The internal graphics controller handles all complexities associated with the implementation of using televisions or monitors with microcontrollers. While this may be done by software alone, such will entail extensive use of processor, memory and peripheral resources and is typically very challenging to implement with other user algorithms. With the Kimat TV module, no complicated programming or handling of several interrupts is required. This results in very minimum load on the host microcontroller (Arduino) and therefore allowing the user program to implement useful programs with ease.

Similar to the use of common alphanumeric LCD's, the Arduino host simply needs to feed the data that needs to be displayed to the module via the library provided.

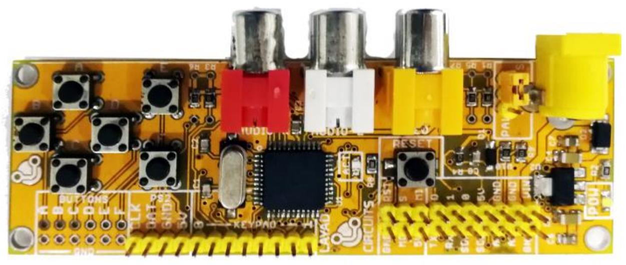

Figure 1 Kimat TV Module (Top View)

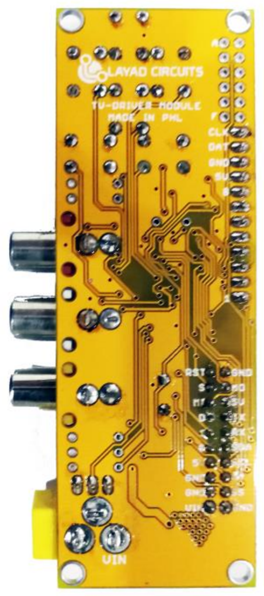

Figure 2 Kimat TV Module (Bottom View)

FEATURES

- Built-in graphics and tone controller
- 6 on-board user buttons
- I2C Interface
- Works on any screen with AV input ports
- Power and activity indicators
- < 50mA power consumption
- 6-9V Input via 5.5x2.1mm DC jack
- Low power
- Low memory load
- Low processor/peripheral load
- Designed primarily for Arduino users
- Arduino library available

HARDWARE OVERVIEW

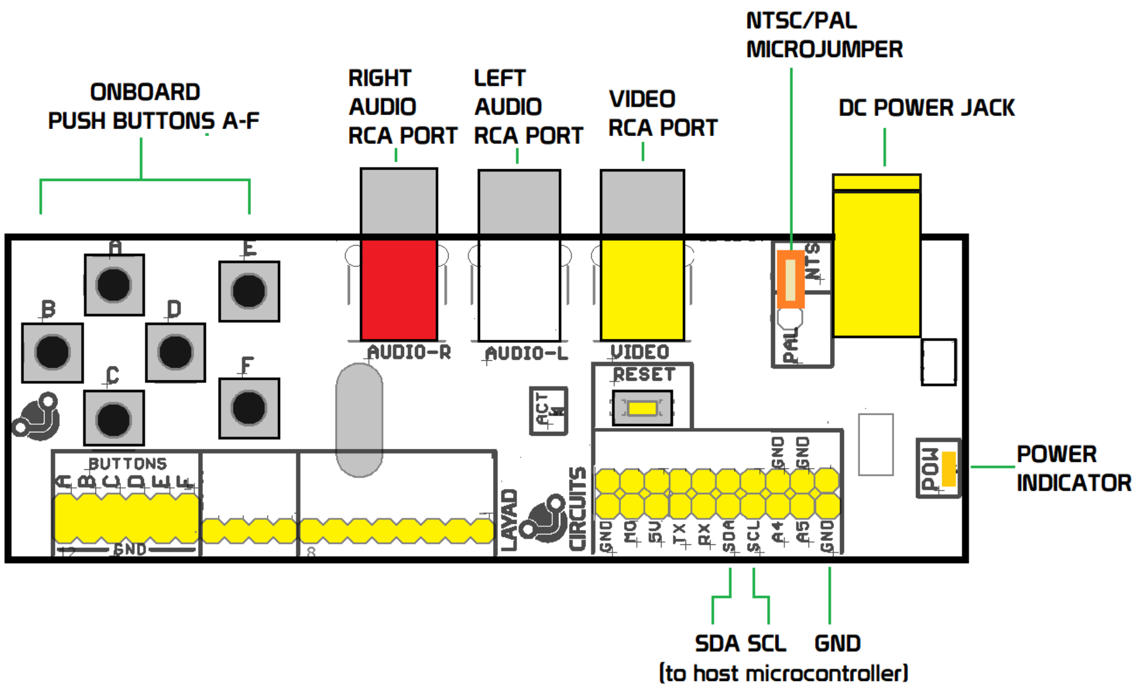

Figure 3 Kimat TV Module Major Components

APPLICATIONS

The Kimat TV module can be used in the following applications:

- Queuing Systems
- HMI
- Ticketing
- Point of Sales
- Kiosks
- Score Boards
- Datetime Display and Alarm

POWER REQUIREMENTS

The module may be powered with a regulated 5-volt source via the 5V pin header. This allows it to be conveniently powered from the 5V output pin of the Saleng Uno or any Arduino board. The module may also be powered from an external DC power supply rated with at least 50 mA and a voltage of 6 to 9V.

JUMPER OPTIONS

The 3-pin header with the label "NTSC" and "PAL" allows the user to choose whether the output format of the Kimat TV module is NTSC or PAL.

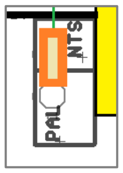

Figure 4 Kimat TV Module NTSC and PAL Jumper

Set the jumper accordingly. After setting the jumper, power cycle the Kimat TV module or press its reset button for the new output format to take effect.

PIN FUNCTIONS

The module has a 10-position header near the power LED indicator. The user only need to connect 4 pins: 5V,GND, SDA and SCL, the rest are reserved for factory or future use. Below are descriptions of the pins that are of importance to the user.

Pin Label	Function/Operation/Remarks
5V	+5Vdc. This is the Kimat TV module power. The LED indicator on board will light up when power is applied. Since the module alone draws less than 50ma at 5V, this pin may then be connected to the Saleng Uno/Arduino's 5V pin. There is no need to connect this if you are powering the module from an external power source using the DC jack.
GND	Ground pin.
SDA	I2C data pin. Connects to SDA pin of the host microcontroller.
SCL	I2C clock pin. Connects to SCL pin of the host microcontroller.

PRECAUTION

Use bidirectional level shifters when using 3.3V microcontrollers to safely connect the Kimat TV module to your host.

APPLICATION NOTES

I2C Interface with Arduino Uno

This section shows how to use the Kimat TV module module thru its I2C interface. An Arduino Uno is used for demonstration purposes, but any other I2C-capable MCU can be used. However, when using other MCU's, make sure to check that the I2C pins can tolerate 5V.

The Kimat TV module I2C interface has the following specifications:

- Up to 400kHz data transfer speed
- Device address is **0x38**

Procedure

1. Hookup the RCA cables to the corresponding sockets on the television/monitor. The RCA connectors and sockets are color-coded to prevent incorrect connections.
2. Plug the RCA cables to the corresponding sockets on the Kimat TV module. Again, observe the color-coding.
3. Connect the I2C SDA pin of the Arduino Uno to the I2C SDA pin of the Kimat TV module (see the image below)..
4. Connect the I2C SCL pin of the Arduino Uno to the I2C SCL pin of the Kimat TV module (see the image below).
5. Connect the GND pin of the Arduino Uno to the GND pin of the Kimat TV module (see the image below).

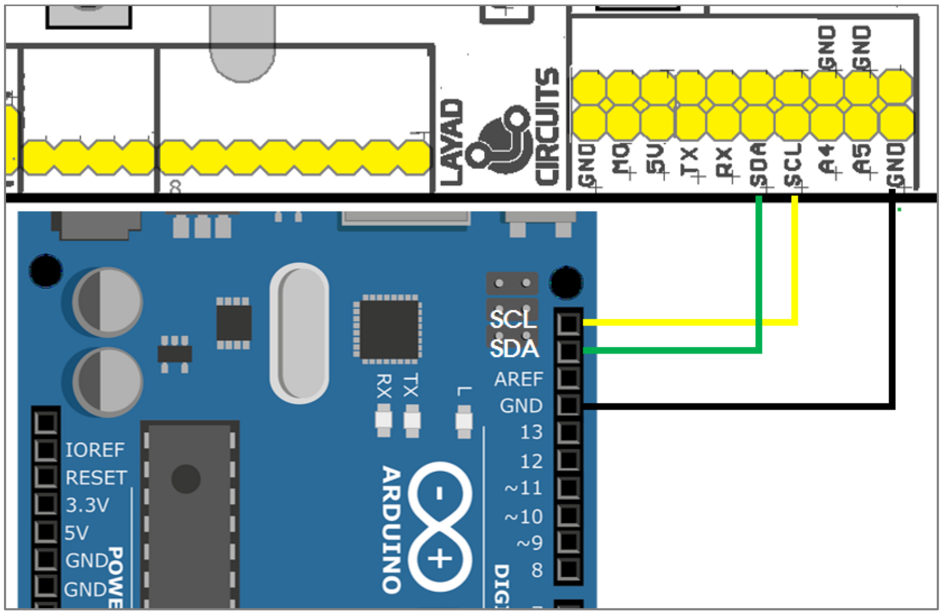

Figure 5 I2C Connection

6. Supply power to the Kimat TV module. There are two power source options for the Kimat TV module:
   a) External power supply thru the DC jack or VIN pin.
   b) 5V pin (connect to the 5V pin of the host Arduino Uno).

Choose any of the above options. Below is an image of a setup wherein the Kimat TV module is powered thru the DC jack (option 1).

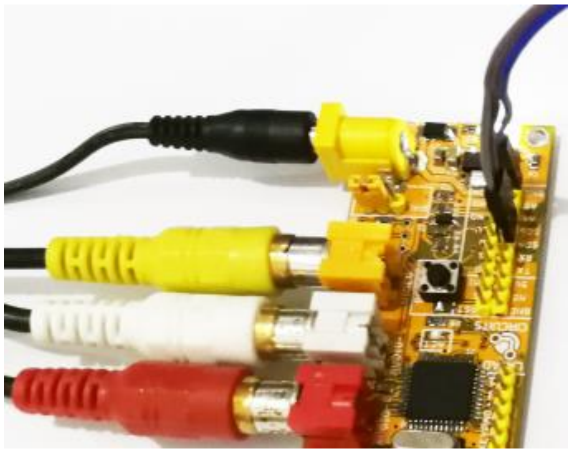

Figure 6 Power Supply Option: DC Jack

Below is a diagram showing the connection between the 5V pin of the host Arduino Uno to that of the Kimat TV module (option 2).

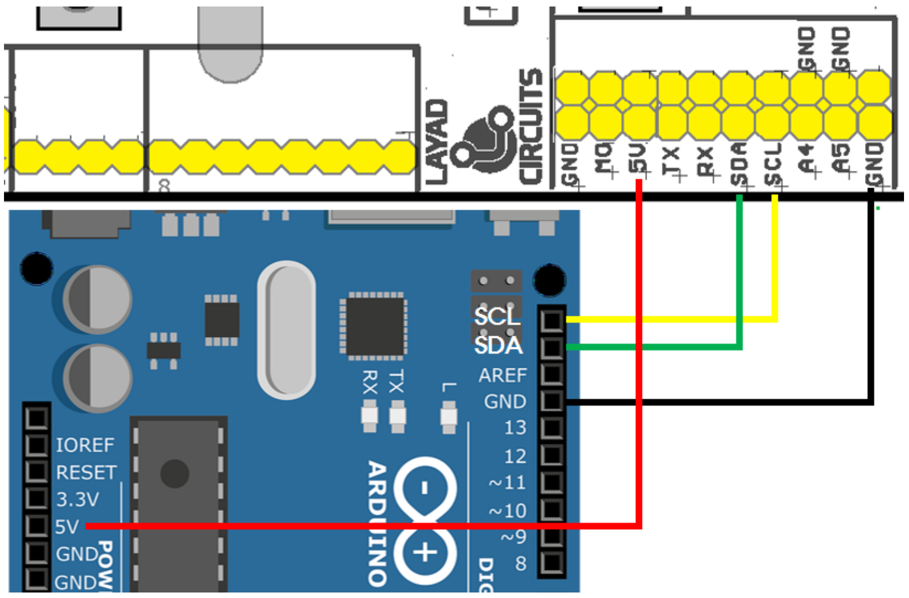

Figure 7 Power Supply Option: 5V from the Host Arduino

7. Download the Layad Circuits Kimat TV library from Github.
8. Once the library zip file is downloaded, open the Arduino IDE and go to Sketch>Include Library>Add .ZIP library and browse over the .zip file you downloaded. Click Open. Close the Arduino IDE and relaunch it. Now you are ready to use the library.

Included in the library are basic examples of how to use the Kimat TV modue. Open the example sketch "HelloWorld.ino" as shown below.

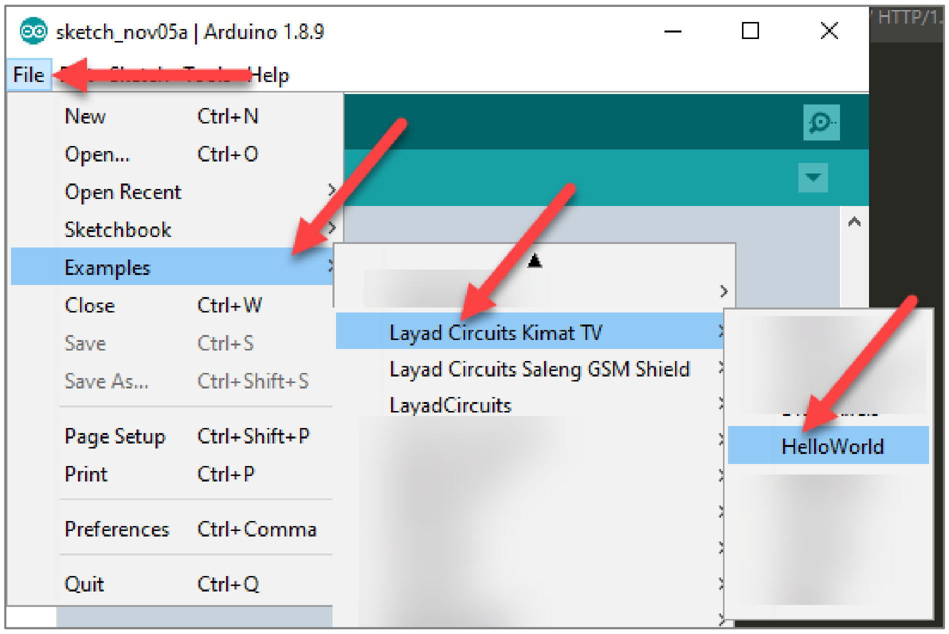

Figure 8 The Arduino Layad Circuits Kimat TV Library

Below is a copy of "HelloWorld.ino".

```
#include <KimatTv.h>
KimatTv ktv(0x38); // I2C device address of Kimat TV is 0x38.
// The library will call this function when a communication error occurs.
void errorHandler(KTvErrorCodes error) {
  Serial.print(F("Error state!!! Error code: "));
  // Print out the error code (see the datasheet for more info).
  Serial.println((uint8_t)error);
  while (1); // Hangup (or do something else here).
  // Resetting both the Kimat TV module and the host is recommended.
}
void setup() {
  Serial.begin(9600);
  Serial.println(F("Kimat TV Module"));
  // Attach the error handler function to the library.
  ktv.attachErrorHandler(errorHandler);
  ktv.init(); // Initialize communication.
  ktv.waitUntilReady(); // Wait for the device to become ready.
  ktv.fill(KTvColor::inverse); // Invert the current pixel colors.

  ktv.delay(3000); // Wait for 3 sec to see the effect.
  ktv.waitUntilReady(); // Wait for the device to become ready.
  ktv.fill(KTvColor::black); // Fill the screen with black color.

  ktv.delay(3000); // Wait for 3 sec to see the effect.
  ktv.waitUntilReady(); // Wait for the device to become ready.
  ktv.fill(KTvColor::white); // Fill the screen with white color.
  ktv.waitUntilReady(); // Wait for the device to become ready.

  ktv.delay(3000); // Wait for 3 sec to see the effect.
  ktv.waitUntilReady(); // Wait for the device to become ready.
  ktv.fill(KTvColor::black); // Fill the screen with black color.

  //////////01234567890123456789
  ktv.print("ABCDEFGHIJKLMNOPQRST"); // Print a string.
  ktv.waitUntilReady(); // Wait for the device to become ready.
}
void loop() {
  // Run other tasks here. Avoid blocking delays.
  ktv.run(); // Let the library run (very important).
}
```

For more information on the library and its methods, refer to the section “APPENDIX A: Arduino Kimat TV Library REFERENCE” at the end of this document.

9. Compile the code and upload the resulting program to the Arduino Uno.
10. Apply power to the television if it's not powered yet.
11. Simultaneously press and release the reset button of the Kimat TV module and the reset button of the Arduino host to make sure that both devices are synchronized at the start. Do this every time a new program is uploaded to the Arduino host. The HelloWorld program should now run.

Below is an image of the whole setup.

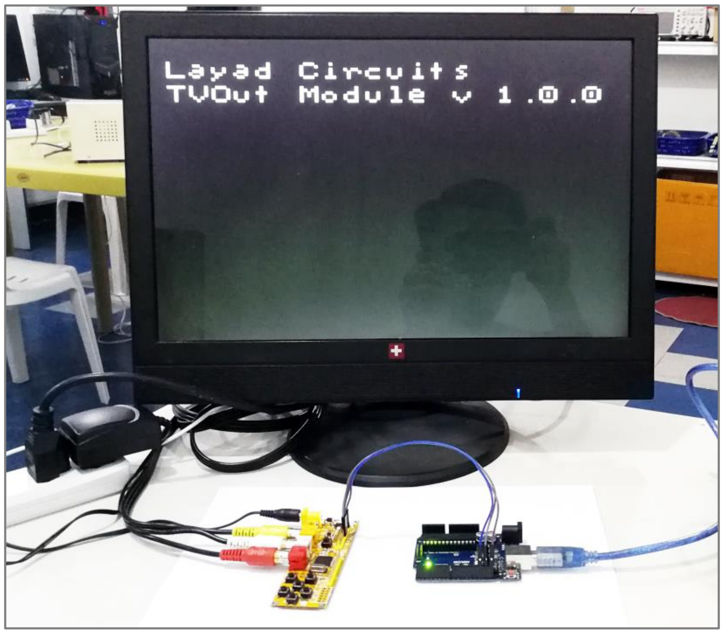

Figure 9 Kimat TV Module and Host Arduino Setup

DOCUMENT REVISION HISTORY

Revision:

v1.0 / 11 November 2019 /D.D.DEPONIO/C.D.MALECDAN

APPENDIX A: ARDUINO KIMAT TV LIBRARY REFERENCE

This section describes the essential user functions available in the Arduino Kimat TV library.

Name	KimatTv(uint8_t i2cAddress)
Description	This is the constructor of the KimatTv class. Use this to instantiate a KimatTv object.
Parameter(s)	
uint8_t i2cAddress	This is the I2C device address of the Kimat TV module. As of writing, the I2C device address of the Kimat TV module is 0x38.
Return	
none	This function does not return any value.
Example	
Refer to the example in section "getStatus()".	

Name	init()
Description	This function initializes the I2C communication of the host device with the Kimat TV module.
Parameter(s)	
none	This function has no parameters.
Return	
none	This function does not return any value.
Example	
Refer to the example in section "getStatus()".	

Name	attachErrorHandler(ErrorHandler handler)
Description	This function attaches a user-defined error handling function to the library. In the event of any communication error, the library will call the attached error handling function.
Parameter(s)	
ErrorHandler handler	This function accepts an argument with the type "ErrorHanlder", which is a pointer to a function that accepts an KTvErrorCodes type and returns nothing.
Return	
none	This function does not return any value.
Example	
Refer to the example in section "getStatus()".	

Name	run()
Description	This function maintains the state machine of the library. This function is also responsible for polling the Kimat TV module for the states of the buttons. It is very important to let this function run as frequently as possible in the main loop. Also, blocking delays must be avoided in the user code to ensure that the library routines are "synchronized" with the Kimat TV module.
Parameter(s)	
none	This function has no parameters.
Return	
none	This function does not return any value.
Example	
Refer to the example in section "getStatus()".	

Name	getStatus()
Description	This function returns the current status of the Kimat TV module.
Parameter(s)	
none	This function has no parameters.
Return	
KTvStatus status	This function returns a variable of the type KTvStatus. Possible values for this type are as follows: KTvStatus::ready: the device is ready to accept a new command. KTvStatus::busy : the device is processing a command. KTvStatus::error: a communication error occurred.

Example

```
#include <KimatTv.h>
KimatTv ktv(0x38); // I2C device address of Kimat TV is 0x38.

// The library will call this function when a communication error occurs.
void errorHandler(KTvErrorCodes error) {
  Serial.print(F("Error state!!! Error code: "));
  // Print out the error code (see the datasheet for more info).
  Serial.println((uint8_t)error);
  while (1); // Hangup (or do something else here).
  // Resetting both the Kimat TV module and the host is recommended.
}

void setup() {
  Serial.begin(9600);
  // Attach the error handler function to the library.
  ktv.attachErrorHandler(errorHandler);
  ktv.init(); // Initialize communication.
}

void loop() {
  // Get the current status of the Kimat TV module.
  KTvStatus deviceStatus = ktv.getStatus();
  if(deviceStatus == KTvStatus::ready)
    Serial.println(F("Kimat TV status: ready!"));
  else if(deviceStatus == KTvStatus::busy)
    Serial.println(F("Kimat TV status: busy!"));
  else if(deviceStatus == KTvStatus::error)
    Serial.println(F("Kimat TV status: error!"));
  else
    Serial.println(F("Kimat TV status: unknown!"));

  // Run other tasks here. Avoid blocking delays.
  ktv.run(); // Let the library run (very important).
}
```

Name	isReady()
Description	This function checks if the Kimat TV module is ready or not.. Use this function to make sure that the Kimat TV module is ready before sending a new command. If a new command is sent and the device is not ready yet, then the command will be ignored.
Parameter(s)	
none	This function has no parameters.
Return	
boolean isReady	This function returns true if the device is ready and false if otherwise.
Example	
Refer to the example in section "setCursor(uint8_t x, uint8_t y)".	

Name	delay(uint32_t period)
Description	This function implements a delay which ensures that the library and the Kimat TV module are synchronized (it internally executes the run() method until the provided period expires). Use this method instead of the Arduino delay() function, as the Arduino delay() function completely prevents the library from communicating with the Kimat TV module.
Parameter(s)	
uint32_t period	This is the delay period in milliseconds.
Return	
none	This function does not return any value.
Example	
Refer to the example in section "waitUntilReady()".	

Name	waitUntilReady()
Description	This function waits for the device to be in a ready state before proceeding. This is a blocking function, which means that other parts of the user code will be executed only after this function is done executing. Use this function to wait for the Kimat TV module to become ready before sending a new command. If a new command is sent and the device is not ready yet, then the command will be ignored.
Parameter(s)	
none	This function has no parameters.
Return	
none	This function returns nothing.
Example	

```
#include <KimatTv.h>
KimatTv ktv(0x38); // I2C device address of Kimat TV is 0x38.
// The library will call this function when a communication error occurs.
void errorHandler(KTvErrorCodes error) {
  Serial.print(F("Error state!!! Error code: "));
  // Print out the error code (see the datasheet for more info).
  Serial.println((uint8_t)error);
  while (1); // Hangup (or do something else here).
  // Resetting both the Kimat TV module and the host is recommended.
}
void setup() {
  Serial.begin(9600);
  Serial.println(F("Kimat TV Module"));
  // Attach the error handler function to the library.
  ktv.attachErrorHandler(errorHandler);
  ktv.init(); // Initialize communication.
  ktv.waitUntilReady(); // Wait for the device to become ready.
  ktv.fill(KTvColor::inverse); // Invert the current pixel colors.

  ktv.delay(3000); // Wait for 3 sec to see the effect.
  ktv.waitUntilReady(); // Wait for the device to become ready.
  ktv.fill(KTvColor::black); // Fill the screen with black color.

  ktv.delay(3000); // Wait for 3 sec to see the effect.
  ktv.waitUntilReady(); // Wait for the device to become ready.
  ktv.fill(KTvColor::white); // Fill the screen with white color.
  ktv.waitUntilReady(); // Wait for the device to become ready.
}
void loop() {
  // Run other tasks here. Avoid blocking delays.
  ktv.run(); // Let the library run (very important).
}
```

Name	getPinStatus(KTvBtn btn)
Description	This function returns the current status of a button on the Kimat TV module.
Parameter(s)	
KTvBtn btn	This is the button to be polled. This must be of the type KTvBtn whose possible values are as follows: KimatTvBtn::btnA, KimatTvBtn::btnB KimatTvBtn::btnC, KimatTvBtn::btnD KimatTvBtn::btnE, KimatTvBtn::btnF
Return	
KTvBtnStatus status	This function returns a KTvBtnStatus type, the possible values of which are as follows: KTvBtnStatus::released KTvBtnStatus::pressed
Example	

```
#include <KimatTv.h>
KimatTv ktv(0x38); // I2C device address of Kimat TV is 0x38.

// The library will call this function when a communication error occurs.
void errorHandler(KTvErrorCodes error){
  Serial.print(F("Error state!!! Error code: "));
  // Print out the error code (see the datasheet for more info).
  Serial.println((uint8_t)error);
  while(1); // Hangup (or do something else here).
  // Resetting both the Kimat TV module and the host is recommended.
}

void setup(){
  Serial.begin(9600);
  // Attach the error handler function to the library.
  ktv.attachErrorHandler(errorHandler);
  ktv.init(); // Initialize communication.
}

void loop(){
  static KTvBtnStatus btnAStPrev;
  static KTvBtnStatus btnAStCur;

  // Get the status of button A.
  btnAStCur = ktv.getPinStatus(KTvBtn::btnA);
  // Print only if there is a change in status.
  if(btnAStPrev != btnAStCur){
    Serial.print(F("user: Btn A: "));
    if(btnAStCur == KTvBtnStatus::pressed)
      Serial.println(F("pressed"));
    else if(btnAStCur == KTvBtnStatus::released)
      Serial.println(F("released"));
    btnAStPrev = btnAStCur;
  }

  // Run other tasks here. Avoid blocking delays.

  ktv.run(); // Let the library run (very important).
}
```

Name	setCursor(uint8_t x, uint8_t y)
Description	This function moves the cursor to a specific point on the screen. The origin (0,0) is the top-left corner of the screen.
Parameter(s)	
uint8_t x	The x coordinate of the point. The output resolution of the Kimat TV module is 120 x 96 pixels. Hence, the range of correct values for x is 0 to 119.
uint8_t y	The y coordinate of the point. The output resolution of the Kimat TV module is 120 x 96 pixels. Hence, the range of correct values for y is 0 to 95.
Return	
none	This function does not return any value.
Example	

```
#include <KimatTv.h>
KimatTv ktv(0x38); // I2C device address of Kimat TV is 0x38.
// The library will call this function when a communication error occurs.
void errorHandler(KTvErrorCodes error) {
  Serial.print(F("Error state!!! Error code: "));
  // Print out the error code (see the datasheet for more info).
  Serial.println((uint8_t)error);
  while (1); // Hangup (or do something else here).
  // Resetting both the Kimat TV module and the host is recommended.
}
void setup() {
  Serial.begin(9600);
  // Attach the error handler function to the library.
  ktv.attachErrorHandler(errorHandler);
  ktv.init(); // Initialize communication.
  ktv.waitUntilReady();// Wait for the device to become ready.
  ktv.fill(KTvColor::black); // Fill the screen with black color.
  ktv.waitUntilReady(); // Wait for the device to become ready.
}
void loop() {
  static uint8_t userState = 0;
  switch (userState) {
    case 0:
      if (ktv.isReady()) { // Wait until the device is ready.
        ktv.setCursor(0,16); // Move the cursor to (0,16) on the screen.
        userState = 1; // Go to the next state.
      }
      break;
    case 1:
      if (ktv.isReady()) { // Wait until the device is ready.
        //////////01234567890123456789
        ktv.print("ABCDEFGHIJKLMNOPQRST"); // Print a string.
        userState = 2; // We are done.
      }
      break;
    case 2:
      // do nothing
      break;
    default:
      break;
  }
  // Run other tasks here. Avoid blocking delays.
  ktv.run(); // Let the library run (very important).
}
```

Name	setFont(KTvFont font)
Description	This function sets the font of the device.
Parameter(s)	
KTvFont font	The font to be used. Possible values for KTvFont type are as follows: KTvFont::font4x6 KTvFont::font6x8 (this is the default font) KTvFont::font8x8
Return	
none	This function does not return any value.
Example	

```
#include <KimatTv.h>
KimatTv ktv(0x38); // I2C device address of Kimat TV is 0x38.
// The library will call this function when a communication error occurs.
void errorHandler(KTvErrorCodes error) {
  Serial.print(F("Error state!!! Error code: "));
  // Print out the error code (see the datasheet for more info).
  Serial.println((uint8_t)error);
  while (1); // Hangup (or do something else here).
  // Resetting both the Kimat TV module and the host is recommended.
}
void setup() {
  Serial.begin(9600);
  // Attach the error handler function to the library.
  ktv.attachErrorHandler(errorHandler);
  ktv.init(); // Initialize communication.
  ktv.waitUntilReady();// Wait for the device to become ready.
  ktv.fill(KTvColor::black); // Fill the screen with black color.
  ktv.waitUntilReady(); // Wait for the device to become ready.
}
void loop() {
  static uint8_t userState = 0;
  switch (userState) {
    case 0:
      if (ktv.isReady()) { // Wait until the device is ready.
        ktv.setFont(KTvFont::font4x6); // Use the 4x6 font.
        userState = 1; // Go to the next state.
      }
      break;
    case 1:
      if (ktv.isReady()) { // Wait until the device is ready.
        //////////01234567890123456789
        ktv.print("ABCDEFGHIJKLMNOPQRST"); // Print a string.
        userState = 2; // We are done.
      }
      break;
    case 2:
      // do nothing
      break;
    default:
      break;
  }

  // Run other tasks here. Avoid blocking delays.

  ktv.run(); // Let the library run (very important).
}
```

Name	print(char *str)
Description	This function prints a string on the screen.
Parameter(s)	
char *str	The string to be printed. The maximum allowed length is 20 characters. For strings with a length of more than 20 characters, use multiple print() functions.
Return	
none	This function does not return any value.
Example	
Refer to the examples in the sections “setCursor(uint8_t x, uint8_t y)” and/or “setFont(KTvFont font)”.	

Name	fill(KTvColor color)
Description	This function fills the screen with the specified color.
Parameter(s)	
KTvColor color	The color to be used. Possible values for KTvColor type are as follows: KTvColor ::black, KTvColor ::white, and KTvColor ::inverse (this will invert the current color of a pixel on the screen).
Return	
none	This function does not return any value.
Example	

```
#include <KimatTv.h>
KimatTv ktv(0x38); // I2C device address of Kimat TV is 0x38.
// The library will call this function when a communication error occurs.
void errorHandler(KTvErrorCodes error) {
  Serial.print(F("Error state!!! Error code: "));
  // Print out the error code (see the datasheet for more info).
  Serial.println((uint8_t)error);
  while (1); // Hangup (or do something else here).
  // Resetting both the Kimat TV module and the host is recommended.
}
void setup() {
  Serial.begin(9600);
  // Attach the error handler function to the library.
  ktv.attachErrorHandler(errorHandler);
  ktv.init(); // Initialize communication.
  ktv.waitUntilReady();// Wait for the device to become ready.
  ktv.fill(KTvColor::inverse); // Fill the screen with black color.
  ktv.delay(3000); // Wait for 3 sec to see the effect.
  ktv.waitUntilReady(); // Wait for the device to become ready.
}
void loop() {
  static uint32_t tRef; // Reference timer variable.
  static uint8_t userState = 0;
  switch (userState) {
    case 0:
      if(millis() - tRef > 3000){ // Wait for 3 sec to see the effect.
        if (ktv.isReady()) { // Wait until the device is ready.
          ktv.fill(KTvColor::white); // Fill the screen with white color.
          tRef = millis(); // Start the timer.
          userState = 1; // Go to the next state.
        }
      }
      break;
    case 1:
      if(millis() - tRef > 3000){ // Wait for 3 sec to see the effect.
        if (ktv.isReady()) { // Wait until the device is ready.
          ktv.fill(KTvColor::black); // Fill the screen with black color.
          tRef = millis(); // Start the timer.
          userState = 0; // Go back to the previous state.
        }
      }
      break;
    default:
      break;
  }
  // Run other tasks here. Avoid blocking delays.
  ktv.run(); // Let the library run (very important).
}
```

Name	setPixel(uint8_t x, uint8_t y, KTvColor color)
Description	This function draws a pixel on the screen at the specified point using the specified color. The origin (0,0) is the top-left corner of the screen.
Parameter(s)	
uint8_t x	The x coordinate of the point. The output resolution of the Kimat TV module is 120 x 96 pixels. Hence, the range of correct values for x is 0 to 119.
uint8_t y	The y coordinate of the point. The output resolution of the Kimat TV module is 120 x 96 pixels. Hence, the range of correct values for y is 0 to 95.
KTvColor color	The color to be used. Possible values for KTvColor type are as follows: KTvColor ::black KTvColor ::white KTvColor ::inverse (this will invert the current color of a pixel on the screen).
Return	
none	This function does not return any value.
Example	

```
#include <KimatTv.h>
KimatTv ktv(0x38); // I2C device address of Kimat TV is 0x38.
// The library will call this function when a communication error occurs.
void errorHandler(KTvErrorCodes error) {
  Serial.print(F("Error state!!! Error code: "));
  // Print out the error code (see the datasheet for more info).
  Serial.println((uint8_t)error);
  while (1); // Hangup (or do something else here).
  // Resetting both the Kimat TV module and the host is recommended.
}
void setup() {
  Serial.begin(9600);
  // Attach the error handler function to the library.
  ktv.attachErrorHandler(errorHandler);
  ktv.init(); // Initialize communication.
}
void loop() {
  static uint8_t userState = 0;
  switch (userState) {
    case 0:
      if (ktv.isReady()) { // Wait until the device is ready.
        ktv.fill(KTvColor::black); // Fill the screen with black color.
        userState = 1; // Go to the next state.
      }
      break;
    case 1:{ // This routines fills the screen with white pixels
      // from left to right, and top to bottom.
      static uint8_t x{}, y{};
      if(ktv.getStatus() == KTvStatus::ready){ // Wait until the device is ready.
        ktv.setPixel(x, y, KTvColor::white);
        x++;
        if (x >= 120){
          x = 0;
          y++;
          if (y >= 96){
            userState = 2; // We are done.
          }
        }
      }
    }
      break;
    case 2:
      // do nothing
      break;
    default:
      break;
  }
  // Run other tasks here. Avoid blocking delays.
  ktv.run(); // Let the library run (very important).
}
```

Name	drawPixelLine(KTvOrientation orientation, uint8_t offset, uint16_t *ptr)
Description	This function draws a bitmap line on the screen at the specified offset using the specified orientation and bitmap pattern source. This function can be used to draw bitmap images on the screen.
Parameter(s)	
KTvOrientation orientation	The orientation of the bitmap line to be drawn. Possible values for KTvOrientation are as follows: KTvOrientation::horizontal KTvOrientation::vertical Rendering a horizontal bitmap line takes approximately 0.53 seconds. On the other hand, rendering a vertical bitmap line takes approximately 0.42 seconds.
uint8_t offset	The offset of the bitmap line to be drawn. For a horizontal bitmap line, this is the offset along the y axis at which the horizontal bitmap line will be rendered. For a vertical bitmap line, this is the offset along the x axis at which the vertical bitmap line will be rendered. The output resolution of the Kimat TV module is 120 x 96 pixels. Hence, the range of correct values for the offset of a vertical bitmap line is 0 to 119. On the other hand, the range of correct values for the offset of a horizontal bitmap line is 0 to 95. The origin (0,0) is the top-left corner of the screen.

uint16_t *ptr	The source of the bitmap pattern to be rendered. This is a pointer to an array containing unsigned 16-bit integers. Horizontal bitmap lines will require 8 unsigned 16-bit integer elements as a bitmap source. On the other hand, vertical bitmap lines will require 6 unsigned 16-bit integer elements as a bitmap source.
	
	Zero bits in the bitmap source will be rendered as black pixels, while one bits will be rendered as white pixels on the bitmap line. To illustrate:
	
	Bitmap source:
	`uint16_t bitmap[8] = { 0xAAAA, 0xBBBB, 0xCCCC, 0xDDDD, 0xEEEE, 0xFFFF, 0xAABB, 0xCCDD };`
	
	In binary, the above is:
	
	`uint16_t bitmap[8] = {`
	`0b1010101010101010,0b1011101110111011,0b1100110011001100,0b1101110111011101,0b11`
	`10111011101110,0b1111111111111111,0b1010101010111011,0b1100110011011101};`

Return	
none	This function does not return any value.

Example (drawPixelLine())

```
#include <KimatTv.h>
KimatTv ktv(0x38); // I2C device address of Kimat TV is 0x38.

// The library will call this function when a communication error occurs.
void errorHandler(KTvErrorCodes error) {
  Serial.print(F("Error state!!! Error code: "));
  // Print out the error code (see the datasheet for more info).
  Serial.println((uint8_t)error);
  while (1); // Hangup (or do something else here).
  // Resetting both the Kimat TV module and the host is recommended.
}

// bitmap source
uint16_t bitmap[8] = {
  0xAAAA, 0xBBBB, 0xCCCC, 0xDDDD, 0xEEEE, 0xFFFF, 0xAABB, 0xCCDD
};

void setup() {
  Serial.begin(9600);
  // Attach the error handler function to the library.
  ktv.attachErrorHandler(errorHandler);
  ktv.init(); // Initialize communication.
  ktv.waitUntilReady(); // Wait for the device to become ready.
  ktv.fill(KTvColor::black); // Fill the screen with black color.

  // Draw bitmap lines using a blocking technique.
  ktv.waitUntilReady(); // Wait for the device to become ready.
  // Render the bitmap pattern as a horizontal bitmap line.
  for (uint8_t y = 0; y < 96; y++) {
    ktv.drawPixelLine(KTvOrientation::horizontal, y, bitmap);
  }

  ktv.waitUntilReady(); // Wait for the device to become ready.
  ktv.fill(KTvColor::black); // Fill the screen with black color.
}

void loop() {
  // Draw bitmap lines using a non-blocking technique.
  static uint8_t userState = 0;
  switch (userState) {
    case 0: {
        static uint8_t x = 0;
        // Render the bitmap pattern as a vertical bitmap line.
        if (ktv.isReady()) { // Wait until the device is ready.
          ktv.drawPixelLine(KTvOrientation::vertical, x, bitmap);
          x++;
          if (x >= 120) {
            userState = 1; // Done.
          }
        }
      }
      break;
    case 1:
      // do nothing
      break;
    default:
      break;
  }
  // Run other tasks here. Avoid blocking delays.
  ktv.run(); // Let the library run (very important).
}
```

Name	startTone(uint16_t freq, uint32_t duration)
Description	This function plays a tone at and for the specified frequency and duration.
Parameter(s)	
uint16_t freq	The frequency of the tone in Hertz.
uint32_t duration	The duration of the tone in milliseconds.
Return	
none	This function does not return any value.
Example	

```
#include <KimatTv.h>
KimatTv ktv(0x38); // I2C device address of Kimat TV is 0x38.
// The library will call this function when a communication error occurs.
void errorHandler(KTvErrorCodes error) {
  Serial.print(F("Error state!!! Error code: "));
  // Print out the error code (see the datasheet for more info).
  Serial.println((uint8_t)error);
  while (1); // Hangup (or do something else here).
  // Resetting both the Kimat TV module and the host is recommended.
}
void setup() {
  Serial.begin(9600);
  // Attach the error handler function to the library.
  ktv.attachErrorHandler(errorHandler);
  ktv.init(); // Initialize communication.
  ktv.waitUntilReady(); // Wait for the device to become ready.
  ktv.startTone(3000, 3000); // Play a 3-kHz tone for 3 seconds.
  ktv.delay(3000); // Wait for 3 sec before sending a new command.
  ktv.waitUntilReady(); // Wait for the device to become ready.
}

void loop() {
  static uint32_t tRef; // Reference timer variable.
  static uint8_t userState = 0;
  switch (userState) {
    case 0:
      if (ktv.isReady()) { // Wait until the device is ready.
        ktv.startTone(500, 1000); // Play a 500-Hz tone for 1 sec.
        tRef = millis(); // Start the timer.
        userState = 1; // Go to the next state.
      }
      break;
    case 1:
      if (millis() - tRef > 1000) { // Wait for the current tone to finish playing.
        if (ktv.isReady()) { // Wait until the device is ready.
          ktv.startTone(5000, 5000); // Play a 5-kHz tone for 5 seconds.
          tRef = millis(); // Start the timer.
          userState = 2; // Go to the next state.
        }
      }
      break;
    case 2:
      if (millis() - tRef > 5000) { // Wait for the current tone to finish playing.
        if (ktv.isReady()) { // Wait until the device is ready.
          ktv.startTone(10000, 10000); // Play a 10-kHz tone for 10 seconds.
          tRef = millis(); // Start the timer.
          userState = 3; // Go to the next state.
        }
      }
      break;
    case 3:
      if (millis() - tRef > 7000) { // Wait for 3 seconds.
        if (ktv.isReady()) { // Wait until the device is ready.
          ktv.stopTone(); // Stop the current tone.
          userState = 4; // Done.
        }
      }
      break;
    case 4:
      // do nothing
      break;
    default:
      break;
  }
  // Run other tasks here. Avoid blocking delays.
  ktv.run(); // Let the library run (very important).
}
```

Name	stopTone()
Description	This function stops the currently playing tone, if any.
Parameter(s)	
none	This function has no parameters.
Return	
none	This function does not return any value.
Example	
Refer to the example in the section “startTone(uint16_t freq, uint32_t duration)”.	

Handling and Recovering from Errors

The Arduino Kimat TV library will automatically attempt to recover from any error that may occur while the host device is communicating with the Kimat TV module. However, if the library fails to recover from an error, it will call a user-defined function so that the user can act on the error accordingly. This user-defined error-handling function must be attached to the library (see section "attachErrorHandler(ErrorHandler handler)" for more information.

The Arduino Kimat TV library will pass a KTvErrorCodes type variable to the user-defined error-handling function. The possible values and meanings for this type are described below. In the event that the library fails to recover from an error and it calls the user-defined error-handling function, a reset or power-cycle of both the host device and the Kimat TV module is highly recommended.

Error Codes

Value	Name	Description
0	ok	Device status as perceived by the library is OK.
1	btnQueryAkd	Last error event was an acknowledged button query command.
2	btnQueryNkd	Last error event was a NAKd button query command (the device did not understand the last button query command).
3	btnQueryNoReplyReady	Last error event was a no reply to a button query command (device is always ready).
4	btnQueryNoReplyBusy	Last error event was a no reply to a button query command (device is always busy).
5	btnQueryUnkRep	Last error event was an unknown reply to a button query command.
6	btnQueryTimeout	Last error event was a no reply to a button query command (timeout). Possible causes are faulty wirings, no connection, and wrong I2C device address.
7	userCmdNkd	Last error event was a nak'd user command (the device did not understand the last user command).
8	userCmdNoReplyReady	Last error event was a no reply to a user command (device is always ready).
9	userCmdNoReplyBusy	Last error event was a no reply to a user command (device is always busy).
10	userCmdUnkRep	Last error event was an unknown reply to a user command.
11	userCmdTimeout	Last error event was a no reply to a user command (timeout). Possible causes are faulty wirings, no connection, and wrong I2C device address.

Revision History:

v.1.0.0 – Initial Creation / DDDeponio / CDMalecdan / 11 November 2019

IMPORTANT NOTICE

Layad Circuits Electronics Engineering Supplies & Services (Layad Circuits) reserves the right to make corrections, enhancements, improvements and other changes to its products, services and documentations, and to discontinue any product or service. Buyers or clients should obtain the latest relevant information before placing orders and should verify that such information is current and complete. Additional terms may apply to the use or sale of Layad Circuits products and services.

Reproduction of significant portions of Layad Circuits information in Layad Circuits datasheets or user guides is permissible only if reproduction is without alteration, displays the Layad Circuits logo and is accompanied by all associated warranties, conditions, limitations, and notices. Layad Circuits is not responsible or liable for such reproduced documentation. Information of third parties may be subject to additional restrictions. Resale of Layad Circuits products or services with statements different from or beyond the parameters stated by Layad Circuits for that product or service voids all express and any implied warranties for the associated Layad Circuits product or service. Layad Circuits is not responsible or liable for any such statements.

Buyers and others who are developing systems that incorporate Layad Circuits products (collectively, “Designers”) understand and agree that Designers remain responsible for using their independent analysis, evaluation and judgment in designing their applications and that Designers have full and exclusive responsibility to assure the safety of Designers' applications and compliance of their applications (and of all Layad Circuits products used in or for Designers’ applications) with all applicable regulations, laws and other applicable requirements. Designer represents that, with respect to their applications, Designer has all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. Designer agrees that prior to using or distributing any applications that include Layad Circuits products, Designer will thoroughly test such applications and the functionality of such Layad Circuits products as used in such applications. Layad Circuits’ provision of technical, application or other design advice, quality characterization, reliability data or other services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, “Layad Circuits Resources”) are intended to assist designers who are developing applications that incorporate Layad Circuits products; by downloading, accessing or using Layad Circuits Resources in any way, Designer (individually or, if Designer is acting on behalf of a company, Designer’s company) agrees to use any particular Layad Circuits Resource solely for this purpose and subject to the terms of this Notice.

Layad Circuits' provision of Layad Circuits Resources does not expand or otherwise alter Layad Circuits’ applicable published warranties or warranty disclaimers for Layad Circuits products, and no additional obligations or liabilities arise from Layad Circuits providing such Layad Circuits Resources.

Layad Circuits reserves the right to make corrections, enhancements, improvements and other changes to its Layad Circuits Resources. Layad Circuits has not conducted any testing other than that specifically described in the published documentation for a particular Layad Circuits Resource.

NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER LAYAD CIRCUITS INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF LAYAD CIRCUITS OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which Layad Circuits products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of Layad Circuits Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from Layad Circuits under the patents or other intellectual property of Layad Circuits. Layad Circuits RESOURCES ARE PROVIDED “AS IS” AND WITH ALL FAULTS. LAYAD CIRCUITS DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS. LAYAD CIRCUITS SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY DESIGNER AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN LAYAD CIRCUITS RESOURCES OR OTHERWISE. IN NO EVENT SHALL LAYAD CIRCUITS BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF LAYAD CIRCUITS RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER LAYAD CIRCUITS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Unless Layad Circuits has explicitly designated an individual product as meeting the requirements of a particular industry standard, Layad Circuits is not responsible for any failure to meet such industry standard requirements. Where Layad Circuits specifically promotes products as facilitating functional safety or as compliant with industry functional safety standards, such products are intended to help enable customers to design and create their own applications that meet applicable functional safety standards and requirements. Using products in an application does not by itself establish any safety features in the application. Designers must ensure compliance with safety-related requirements and standards applicable to their applications. Designer may NOT use any Layad Circuits products in life-critical applications. Life-critical medical equipment is medical equipment where failure of such equipment would cause serious bodily injury or death (e.g., life support, pacemakers, defibrillators, heart pumps, neurostimulators, and implantables). Designers agree that it has the necessary expertise to select the product with the appropriate qualification designation for their applications and that proper product selection is at Designers’ own risk. Designers are solely responsible for compliance with all legal and regulatory requirements in connection with such selection. Designer will fully indemnify Layad Circuits and its representatives against any damages, costs, losses, and/or liabilities arising out of Designer’s noncompliance with the terms and provisions of this Notice.