**Article title**

*A General Purpose Alarm Device (GPAD)*

**Authors**

*Robert L. Read\*, Lawrence Kincheloe, Forrest Lee Erickson*

**Affiliations**

Robert L. Read Lawrence R. Kincheloe III Forrest Lee Erickson

Public Invention SPEC & Public Invention Public Invention

1709 Norris Dr. 3217 Brush Creek Road 1709 Norris Dr.

Austin, TX, 78704 Oklahoma City, OK 73120 Austin, TX, 78704

**Corresponding author’s email address and Twitter handle**

[read.robert@pubinv.org](mailto:Iread.robert@pubinv.org)

twitter: @robertleeread

**Abstract**

The General Purpose Alarm Device (GPAD) shines lights and makes loud noises to draw the attention of a human being to a problem. It has a programmable 80 character display to provide textual information. A mute button toggles the sound on and off. Fundamentally it is designed to act as a peripheral to a controlling computer or microcontroller. The controller may communicate over a USB (COM) connection or through a 5V SPI connection via an RJ12 cable. The GPAD is intended to be as general purpose as possible, so that it can be used to provide alarm functionality for many engineering, scientific, hobby machines, instruments, and situations. The original driving use case is to provide medical alarm capability to the PolyVent [5] open-source mechanical medical ventilator. The GPAD supports 5 alarm levels above “silent” of increasing urgency in terms of light, rhythm, and frequency. It is based on the Arduino Uno Atmega328 design and potentially extensible the same way an Uno is, through headers and shields. It includes the printed wiring assembly, firmware for the GPAD peripheral, a simple documented API and a 3D printable enclosure. The repo includes instructions for using a second GPAD as a controller.

**Keywords**

alarms, arduino, UNO, SPI, medical alarms, monitoring

**Specifications table**

|  |  |
| --- | --- |
| Hardware name | General Purpose Alarm Device (GPAD) v2.0 |
| Subject area | General |
| Hardware type | Other: general alarms for digitally detectable alert conditions |
| Closest commercial analog | No known commercial analog is available. |
| Open source license | *CERN Open Hardware Licence Version 2 - Strongly Reciprocal* |
| Cost of hardware | $100.00 |
| Source file repository | *https://zenodo.org/records/10065096* |
| OSHWA certification UID *(OPTIONAL)* | US002352 |

1. **Hardware in context**

We are not aware of any alarm device that has the programmable features of the GPAD. Electronic component firms such as Mouser and Digikey sell buzzers, sirens, and loudspeakers which can make noise; they sell lights that can represent an alarm condition; they sell proprietary alarm devices that are tied to special purposes (such as a smoke alarm), but we know of know digitally controllable system that combines these features. The Adafruit Towerlight [6], an example of a “stacklight”, but it does not seem to have a digital interface, and has no text output capability, and no abstract API. Mucco makes a complete line [4] of line of integrated warning horns and stack lights, but they appear to have no digital interface, no text, and no alarm levels; they are simply on or off. Although we believe a GPAD is a widely usable device, its use is of course confined to manufacturers, makers, or engineers who are capable of programming it, which may preclude the market volume associated with consumer devices.

The digital controllers for 3D printers can often make a beep, and have a digital display, which is sometimes backlit. This has some of the features of the GPAD, but is not as bright or as loud, nor programmed for this purpose as conveniently. A cheap example is the HiLetGo 3-01-0889 [7].

1. **Hardware description**

We have designed a printed wiring assembly and an enclosure. The enclosure is derived from the parametric OpenSCAD project “The Ultimate box maker” [8]. The enclosure is approximately 160mm x 110mm x 50mm (6”x4.25”x1.75”) The assembly weighs 232 grams (exclusive of any power supply).

The device has a 20x4 character display (English characters), five bright white programmable LEDs, a programmable buzzer, a “Mute” button which silences the buzzer and a non programmable power indicator LED. The enclosure hangs by (for example) tie wraps from a rail for vertical display or can rest on a horizontal surface. Typical signal connection is by an RJ12 data cable. Power is by an external wall supply with a 2.1mm barrel jack power connector at 9-12V at <1000 mA.

The GPAD integrates several features hobbyists find useful in one device. Although you can purchase 20x4 displays with parallel or I2C interfaces it is the integration in one PCB and enclosure of all of the features which our device adds to make this tethered device integrate into a system easily.

The GPAD hardware features the following:

* Five bright LEDs to visually indicate alert levels.
* A buzzer to indicate urgent attention is required.
* A button for minimum user feedback and to mute alerts.
* A back lit 20x4 character display to provide detailed information to the user.
* An enclosure that can be vertically mounted by hanging, for example with wire ties.

Example Use Case

Scientific experiments and laboratory equipment often monitor something over time whose condition can reach a state that requires human attention. The GPAD can thus be useful in the field or laboratory for:

* Alerting to conditions which may ruin the experiment if not corrected,
* Alerting to the very conditions sought by the experimenter and therefore demanding attention.
* Indicating potential failure of the equipment or the need for maintenance, such as low battery charge, over-temperature, over-pressure, etc.

The GPAD is designed to be loosely integrated with new devices. At the cost of modifying the enclosure, a researcher may integrate the GPAD into their own equipment.

1. **Design files summary**

Design files are found at the Public Invention Github repository at: <https://github.com/PubInv/general-purpose-alarm-device>.

This is a guide to files found there.

| **Design file name** | **File type** | **Open source license** | **Location of the file** |
| --- | --- | --- | --- |
| Firmware | A folder with source files. A README.md there describes the various source files | Firmware: Affero GPL 3.0 | <https://github.com/PubInv/general-purpose-alarm-device/tree/main/Firmware> |
| Simulation | Wokwi, simulations for Factory Test and other development efforts. | Firmware: Affero GPL 3.0 | <https://github.com/PubInv/general-purpose-alarm-device/tree/main/simulation> |
| Hardware | Folder holding a README describing the subfolders to follow. | CERN Open Hardware Licence Version 2 - Strongly Reciprocal | <https://github.com/PubInv/general-purpose-alarm-device/tree/main/Hardware> |
| GeneralPurposeAlarmDevicePCB | KiCad schematic and PCB files | CERN Open Hardware Licence Version 2 - Strongly Reciprocal | <https://github.com/PubInv/general-purpose-alarm-device/tree/main/Hardware/GeneralPurposeAlarmDevicePCB> |
| Enclosure | OpenSCAD and FreeCAD source files, STL and STEP files for the enclosure | CERN Open Hardware Licence Version 2 - Strongly Reciprocal | <https://github.com/PubInv/general-purpose-alarm-device/tree/main/Hardware/Enclosure> |
| Manufacturing | Bill of materials, Gerber files and description records of parts ordered for the builds. | CERN Open Hardware Licence Version 2 - Strongly Reciprocal | <https://github.com/PubInv/general-purpose-alarm-device/tree/main/Hardware/Manufacturing> |
| Manufacturing and Unit Test Documentation | Assembly instructions and notes | CERN Open Hardware Licence Version 2 - Strongly Reciprocal | <https://github.com/PubInv/general-purpose-alarm-device/blob/main/Hardware/Documentation/ManufacturingUnitTestTroubleshootingRev2.md> |

1. ***Bill of materials summary***

The spread sheet used to receive the KiCad generated BOM/PARTS LIST and to transform it into the files necessary for ordering PCB assemblies from JLCPCB is found at:

<https://github.com/PubInv/general-purpose-alarm-device/blob/main/Hardware/GeneralPurposeAlarmDevicePCB/GPAD_V2_BOM.xls>

Instructions are built into the spreadsheet for its use. JLCPCB built the raw PCB and did SMT assembly and some through hole assembly. The balance of the through hole assembly was done by Public Invention volunteers.

**Break out of major components**

| **Designator** | **Component** | **Number** | **Cost per unit -currency** | **Total cost -**  **currency** | **Source of materials** | **Material type** |
| --- | --- | --- | --- | --- | --- | --- |
| Enclosure LED Standoff | U\_Box\_V105\_GPAD\_LED\_Standoff\_single.stl | 6 | $1.01 | $1.01 | Fabricated by JLCPCB | 321PA-F NYLON |
| Enclosure Frontpanel | U\_Box\_V105\_General\_Alarm\_Device\_FrontPanel.stl | 1 | $3.05 | $3.05 | Fabricated by JLCPCB | 321PA-F NYLON |
| Enclosure Top | U\_Box\_V105\_General\_Alarm\_Device\_Top.stl | 1 | $12.4840 | $12.48 | Fabricated by JLCPCB | 321PA-F NYLON |
| Enclosure Bottom | U\_Box\_V105\_General\_Alarm\_Device\_bottom.stl | 1 | $13.00 | $13.00 | Fabricated by JLCPCB | 321PA-F NYLON |
| Enclosure Button | U\_Box\_V104\_General\_Alarm\_Device\_button.stl | 1 | $1.00 | $1.00 | Fabricated by JLCPCB | 321PA-F NYLON |
| Enclosure Backpanel | U\_Box\_V105\_General\_Alarm\_Device\_Backpanel.stl | 1 NOTE use of this is exclusive of the BackPanel Hanging Compound | $3.34 | $3.34 | Fabricated by JLCPCB | 321PA-F NYLON |
| Enclosure BackPanel\_Hanging | BackPanel\_Hanging-CompoundV2.stl | 1 NOTE use of this is exclusive of the BackPanel | $5.90 | $5.90 | Fabricated by JLCPCB | 321PA-F NYLON |
| Enclosure Screws PCB mounting | Choose to fit the enclosure as fabricated | 5 | $0.10 | $0.50 | Used #6 sheet metal screws 3/8” long | Steel |
| Enclosure Screws, | Choose to fit the enclosure as fabricated | 4 | $0.10 | $0.40 |  | Steel |
| PCB Assembly exclusive of through hole components | BOM\_JLCPCB\_20230228Modified.xls | 1 | $13.86 | $13.86 | Fabricated by JLCPCB | GeneralPurposeAlarmDevicePCB-Placement20230228Modified.xls |
| U302 | LCD Display | 1 | $4.99 | $4.99 | https://www.aliexpress.com/item/3256803213374992.html | 3256 Display Assembly |
| Nylon spacers | Spacer\_0.0182x0.125 inch | 4 | $0.13 | $0.52 | McMasterCarr | 94639a702 |
| Screw | 4-40x 3/8” | 4 | $0.12 | $0.48 | Digikey | 36-9901-ND |
| Nut | Nut\_4-40\_3/16 | 4 | $0.10 | $0.40 | Digikey | 36-4694-ND |
| S101 | SWITCH\_TACTILE\_SPST-NO\_0.05A\_24V | 1 | $0.13 | $0.13 | Digikey | 450-1804-ND |
| S601 | SWITCH\_TACTILE\_12mmx12mm\_SPST-NO\_0.05A\_24V | 1 | $0.55 | $0.55 | Digikey | SW414-ND |
| LED White | LED\_T1.75\_CLEAR\_WHITE | 6 | $0.65 | $3.90 | Digikey | 160-1772-ND |
| LED Red | LED\_T1.75\_CLEAR\_RED | 1 | $0.36 | $0.36 | Digikey | 160-1682-ND |
| RV301 | POT 0.375 10K | 1 | $1.61 | $1.61 | Digikey | 3386P-103LF-ND |
| 9-12V wall power supply | Example Jameco.com 9V 200mA | 1 | $6.95 | $6.95 | Jameco.com | 100845 |

These parts sum to $74.43.

1. **Build instructions**

Partial assembly of the GPAD consists of PCB fabrication and most components at JLCPCB. Components not available at JLCPCB were later added by Public Invention.

The printed wiring assemblies were ordered from JLCPCB using a BOM file and a placement file:

* <https://github.com/PubInv/general-purpose-alarm-device/blob/main/Hardware/Manufacturing/BOM_JLCPCB_20230228Modified.xls>
* <https://github.com/PubInv/general-purpose-alarm-device/blob/main/Hardware/Manufacturing/GeneralPurposeAlarmDevicePCB-Placement20230228Modified.xls>

LED stand-offs were 3D printed from JLCPCB

Order six for each assembly with file:

* <https://github.com/PubInv/general-purpose-alarm-device/blob/main/Hardware/Enclosure/U_Box_V105_GPAD_LED_Standoff_single.stl>

Enclosures were ordered 3D printed from JLCPCB.

To build an enclosure print one each of these six files:

* <https://github.com/PubInv/general-purpose-alarm-device/blob/main/Hardware/Enclosure/U_Box_V105_General_Alarm_Device_Backpanel.stl>
* <https://github.com/PubInv/general-purpose-alarm-device/blob/main/Hardware/Enclosure/U_Box_V105_General_Alarm_Device_FrontPanel.stl>
* <https://github.com/PubInv/general-purpose-alarm-device/blob/main/Hardware/Enclosure/U_Box_V105_General_Alarm_Device_Top.stl>
* <https://github.com/PubInv/general-purpose-alarm-device/blob/main/Hardware/Enclosure/U_Box_V105_General_Alarm_Device_bottom.stl>
* <https://github.com/PubInv/general-purpose-alarm-device/blob/main/Hardware/Enclosure/U_Box_V104_General_Alarm_Device_button.stl>
* <https://github.com/PubInv/general-purpose-alarm-device/blob/main/Hardware/Enclosure/BackPanel_Hanging-CompoundV2.stl>

The time to build assemblies (40 minutes) is documented in [Github Issue #229](https://github.com/PubInv/general-purpose-alarm-device/issues/229).

Detailed instructions for the balance of manufacturing was developed by journaling work on the assemblies received from JLCPCB. The journal is captured in a markdown document, “Manufacturing and Unit Test Documentation, PCB Version 2.0, 20230228” found at: <https://github.com/PubInv/general-purpose-alarm-device/blob/main/Hardware/Documentation/ManufacturingUnitTestTroubleshootingRev2.md>

This section is pulled from that markdown document and the numbering is independent of the numbering in this document.

**Manufacturing and Unit Test Documentation, PCB Version 2.0 February 28, 2023**

### Tools Required

* Solder Station with appropriate ventilation
* 3/16 Nut Driver
* Number 1 Phillips Screw Driver
* Diagonal or other Flush Cutting Hand Tool for lead trimming.
* Assembly fixture, detailed below.

### Reference Material

The most recent schematic is for Rev 2 PCB Assemblies is:

[https://github.com/PubInv/general-purpose-alarm-device/Hardware/GeneralPurposeAlarmDevicePCB/PDF/Schematic-GeneralPurposeAlarmDevicePCB-V2.2.pdf](https://github.com/PubInv/general-purpose-alarm-device/blob/main/Hardware/Documentation/ManufacturingUnitTestTroubleshootingRev2.md)

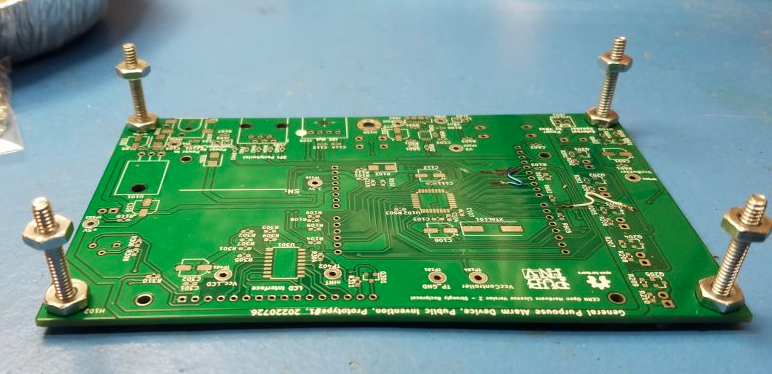
### Assembly Notes and Tips

* The board will be received from JLCPCB with SMT components placed and some but not all through-hole components.
* The serial number should be written on the PCB assembly at the location provided.

***Further management of serial numbers is beyond the scope of this document.***

### An Assembly Fixture

* An assembly assistant/fixture can be made by using a raw PCB with some long #6 screws and nuts to hold at the PCB mounting points.



**Required rework for GPAD Rev 1.x PCB**

***Please note:*** The following changes have been made to the GPAD Rev 2.0 PCB per issue #213:

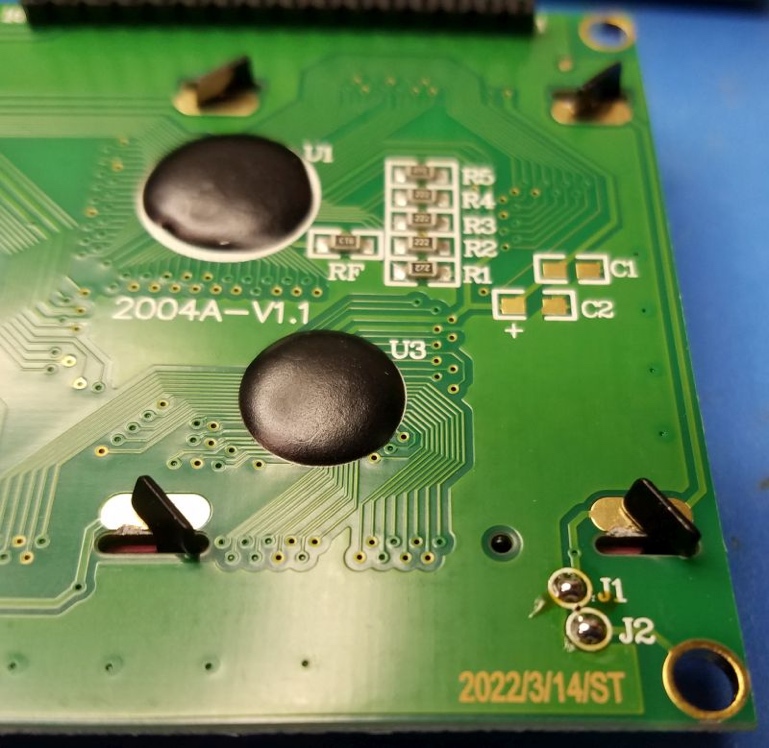
* Resistor R103 has been removed.
* Factory test for D601 has been updated.

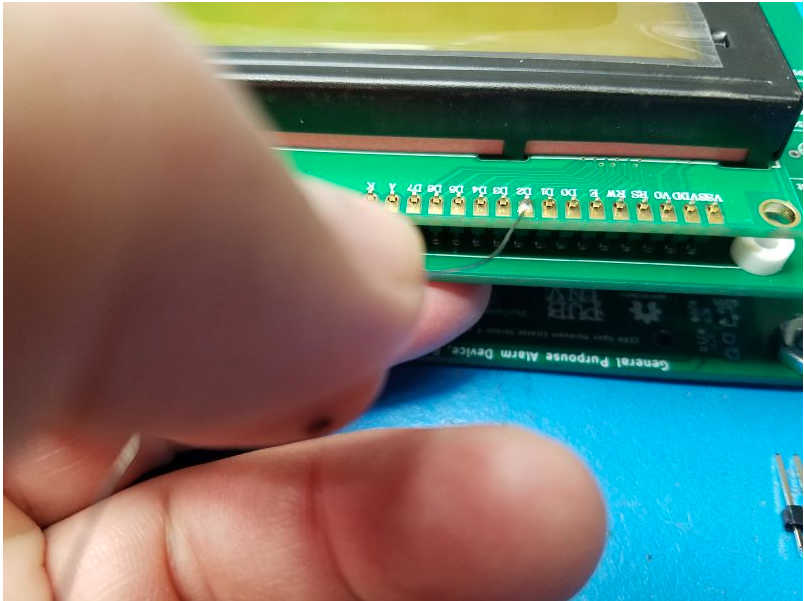
The problem that was identified is that the assemblies built per the BOM\_JLCPCB\_20230228Modified.xls have R103 fitted with a 1K resistor. This resistor, together with LED D102, loads the SPI\_CLK signal and is incompatible with proper operation of the GPAD as an SPI Peripheral from a 3.3V SPI Controller using the level shifting method using the common gate MOSFET.

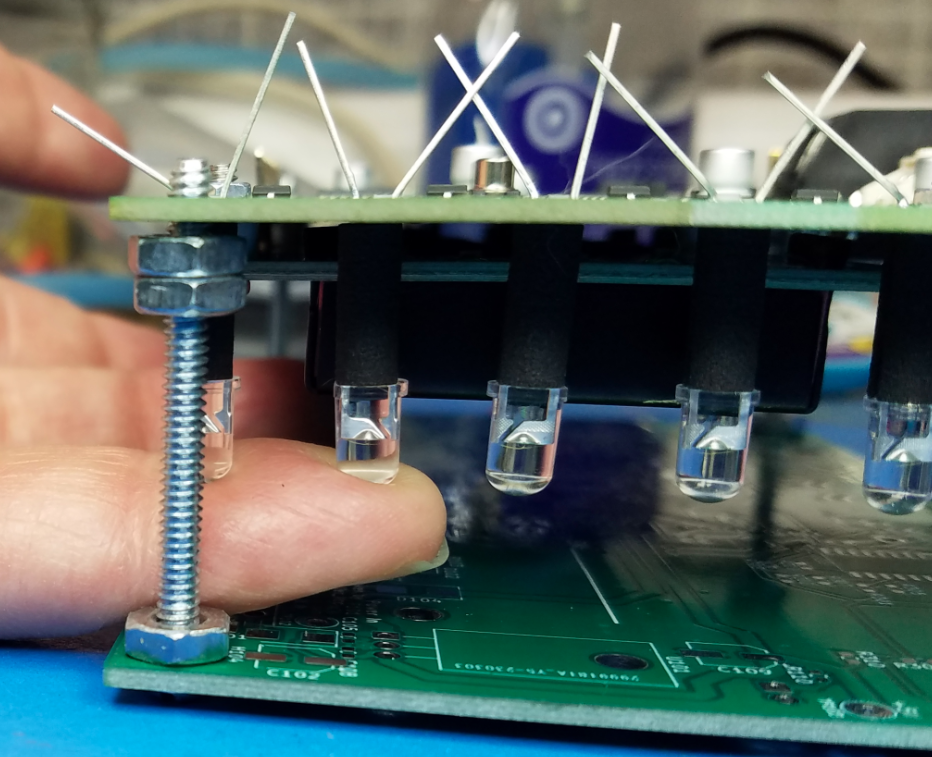
**Solution:**The recommended solution is to remove R103.

### Assembly Steps

1. **LCD Bezel Grounding**
   * Locate the J1 and J2 solder pads on the LCD module.
   * Solder them.



1. **16-pin Header**
   * Fit the 16-pin header onto the LCD sub module.
   * Solder the header pins into the GPAD PCB.
   * Place four nylon 1/8" spacers at the four corners of the LCD sub module.
   * Place four 4-40 x 3/8" screws with 4-40 x 3/16" nuts through both boards and torque to 3.4 - 4.8 Inch-Pounds.
2. **Buttons and Pot**
   * Place the Reset button, S101, into the PCB from the display side.
   * Place the Mute button, S401, into the PCB from the display side.
   * Place the Buzzer, BZ601, into the PCB from the display side. Bending leads may help retain.
   * Place the Contrast pot, RV301, into the PCB from the display side. Bending leads may help retain.
   * Solder to PCB.
3. **LEDs**
   * Thread the LED leads through the stand-offs.
   * The longer LED lead is the anode. The cathode has a flat side on the plastic case.
   * Place the LEDs into the PCB so that the flat cathode side corresponds to the silk screen marking. Bend the leads to retain the LED into the PCB.
   * Placing the assembly on the fixture lets you have access to the top and the bottom of the assembly.
   * Life LED and reflow solder for a flush fit on the PCB.
   * For each LED, after soldering one lead by holding the LED from below supported by the spacer and held against the PCB; reheat / reflow the solder for a flush fit.
   * Solder the second lead on the LED.
   * Trim the excess leads on RV301 and the LEDs.



**Assembly Tip:** Sharpie Oil-Based Paint Markers can be used to mark polarity on LED standoffs and mark PCB's version, serial number and programming status of the microprocessor.

**Congratulations! Electrical assembly is complete.**

### Electrical Tests

Electrical tests are divided into two parts:

1. Unprogrammed measurements made before flash programming the boot loader and other firmware.
2. Programmed measurements made after a boot loader and firmware have been placed into the microcontroller.

#### Electrical Measurements before programming bootloader and firmware

Measure and record the following electrical parameters by serial number.

Investigate and correct any abnormal measurements before applying power. Remove J102 and J103 and retain them if present. Note where they should be replaced. Start with no connections to the DUT (Device Under Test).

**Power Jack** Measure resistance to ground at J101 center pin as open or greater than 1Meg ohms.

**SPI Interface** Measure resistance to ground at J401 pin 5 as open or greater than 1Meg ohms..

**Vin net** Measure resistance to ground at TP102 as greater than 75K ohms. This net is capacitive and the resistance measured will climb as the multimeter charges the net.

**+5V net** Measure resistance to ground at TP103 +5 at 1K +/- 5% (950-1050 Ohms).

With a current limited supply set for 12V and maximum of 200 mA, apply power at J101 and note and record the unprogrammed current.

(FYI, when unprogrammed, the first time power up current is normally about 75-80 mA.)

Check that the power LED D105 is lit and is RED.

**+5V net** Measure the voltage of the +5V Net at TP103.

Please be advised that the current drawn by a programmed DUT that has been powered up and with display backlight on is approximately 61 mA when the reset switch is held down.

#### Vo Initial Set / LCD Contrast.

With a voltmeter, measure the voltage of the Vo pin of the LCD header to ground. Adjust RV103 for 1.3 V.

See records of measurements of some of the Rev 2 assemblies at this issue: https://github.com/PubInv/general-alarm-device/issues/217 #217

#### Electrical Test Results Table

* All resistance measured on 20Meg scale except R@5V net measured on 2K scale. Multimeter EMCO Model DMR-3800 unless noted.
* All current measured 200mA scale. Multimeter EMCO Model DMR-3800 unless noted.
* Add rows to this table for each unit under test.

Capture:

DUT Serial Number, R@PowerJack, R@SPI Interface, R@Vin net, R@5V net, UnProgramCurrent, Volt@+5 TP103, FullCurrent mA, Vo Volts, Notes

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| DUT Serial Number | R@PowerJack | R@SPI Interface | R@Vin net | R@5V net | UnProgramCurrent | Volt@+5 TP103 | FullCurrent mA | Vo Volts | Notes |
|  |  |  |  |  |  |  |  |  |  |

#### Measurements of current with in subcircuits (Issue #230)

(See actual data from Rev2 build here: <https://github.com/PubInv/general-alarm-device/issues/230>)

Test condition. Measure before firmware is loaded into device

**Capture current on circuit blocks.**

These measurements were made before the DUT was programmed with any firmware. NO BOOTLOADER even. Measure the current by measuring the voltage across the 1 Ohm decoupling resistor from the raw power to the test points indicated. Raw power is at TP103 +5 test point. Volts in mV will be a measurement of mA.

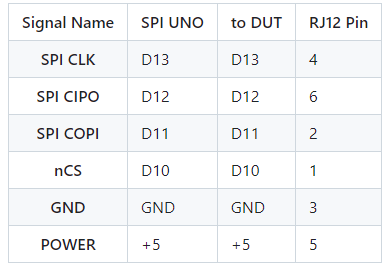
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DUT S# | VR101 (Current U102) mA | VR310 (Current LCD) mA | VR601 (Current Buzzer) mA | Notes |
| 29 | 2.1 | 49.1 |  |  |
| 30 | 2.5 | 48.8 |  |  |

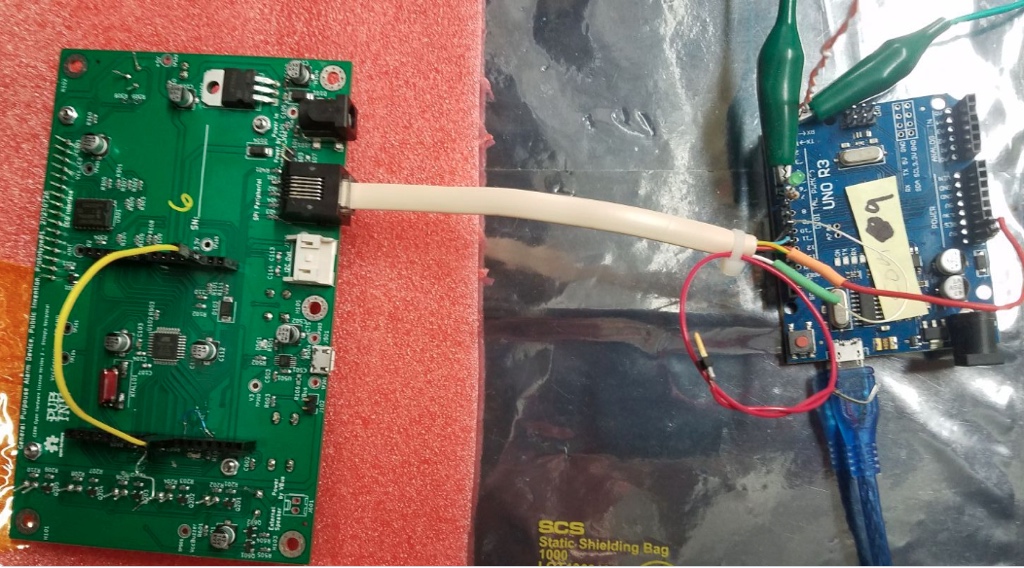
### Load Firmware

\*\* Note: Loading firmware through the SPI interface is a manufacturing test of the SPI hardware components. \*\*

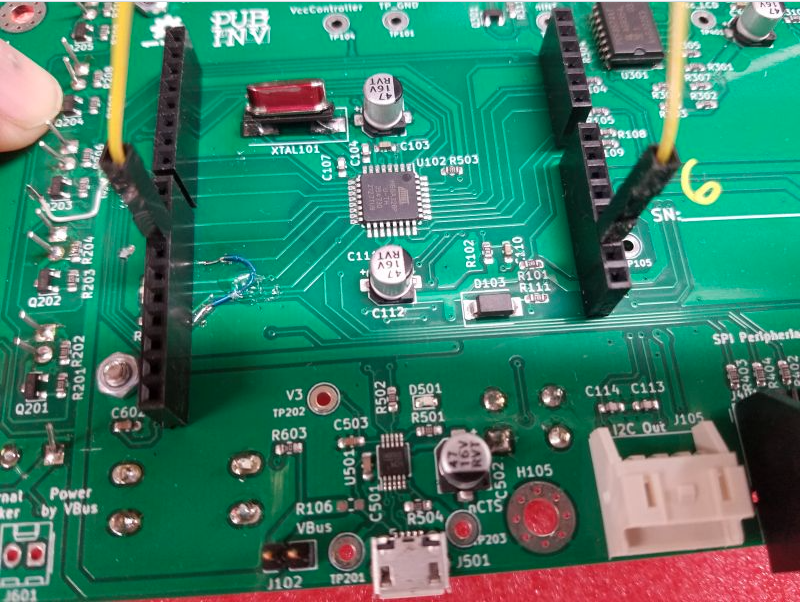
#### Load Bootloader

Use an Arduino UNO as an ISP (In Circuit Serial Programmer) which will load the boot loader into the DUT. Cable Connect the ISP UNO to the DUT as follows:

Wiring of the UNO to the DUT.



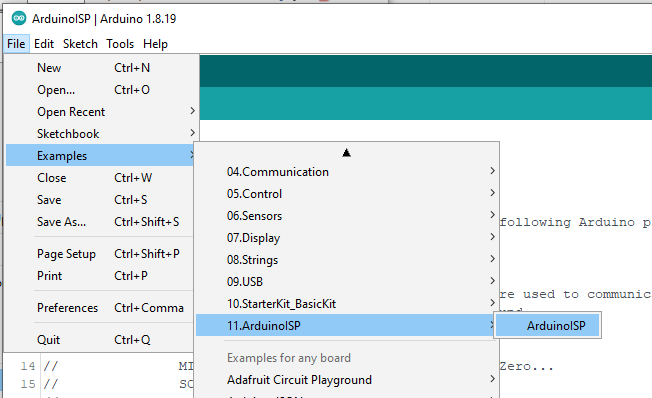
Place a Jumper on the DUT from D10 to Reset Jumper for D10 to Reset.



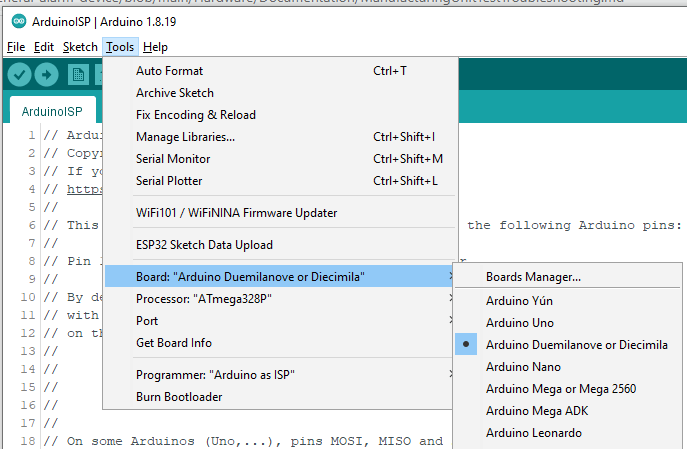
##### Missing Jumper to D10 When Load Bootloader

If you forget the above jumper the IDE will give an error something like this:  
>Arduino: 1.8.19 (Windows 10), Board: “Arduino Duemilanove or Diecimila, ATmega328P” >avrdude: Yikes! Invalid device signature.  
> Double check connections and try again, or use -F to override  
> this check.  
>Error while attempting burning of bootloader without the RJ12 connection.  
>This report would have more information with the “Show verbose output during compilation” option enabled in File -> Preferences.

#### Load into the IDE the sketch “ArduinoISP”.

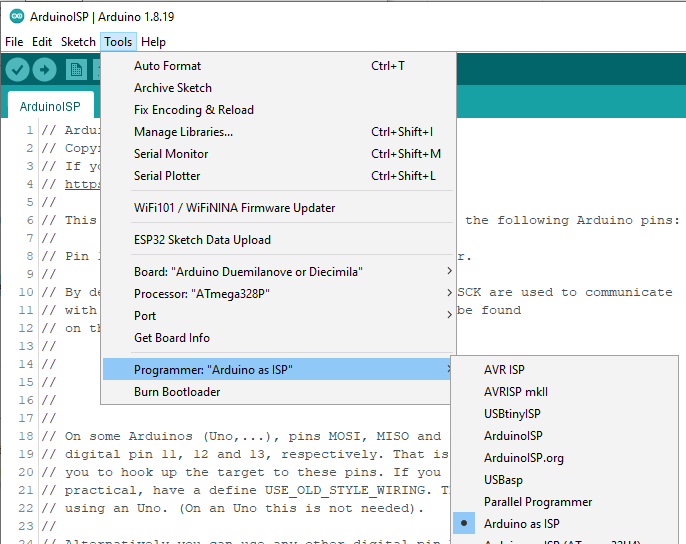
  
ExampleArduinoISP.gif

To load into the UNO, Select the serial port for the UNO and compile and upload with the “ArduinoISP” by pressing **U**.

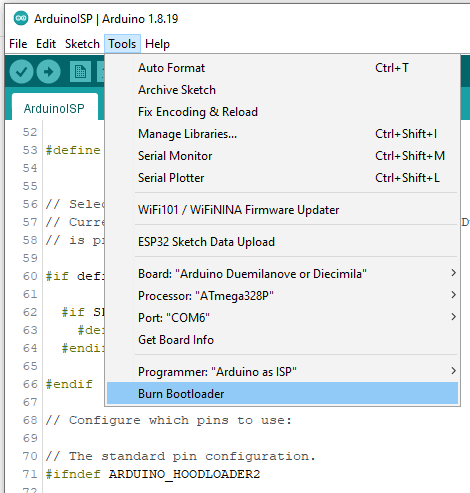
Setup the UNO to burn the boot loader into the GPAD target. Select the board type (Boot loader type) to “Arduino Duemilanove…” .  


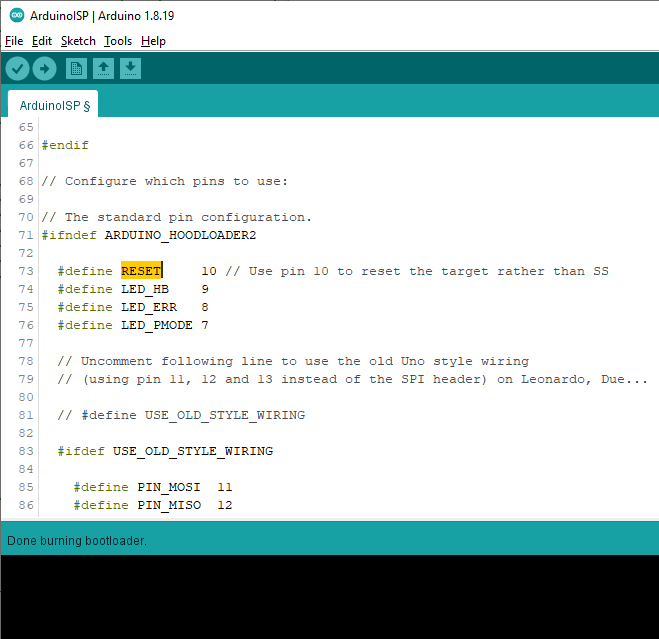
Select the Processor type to “ATmega328P” .

Select the programmer type.



In the Arduino IDE, select TOOLS > Burn Bootloader .

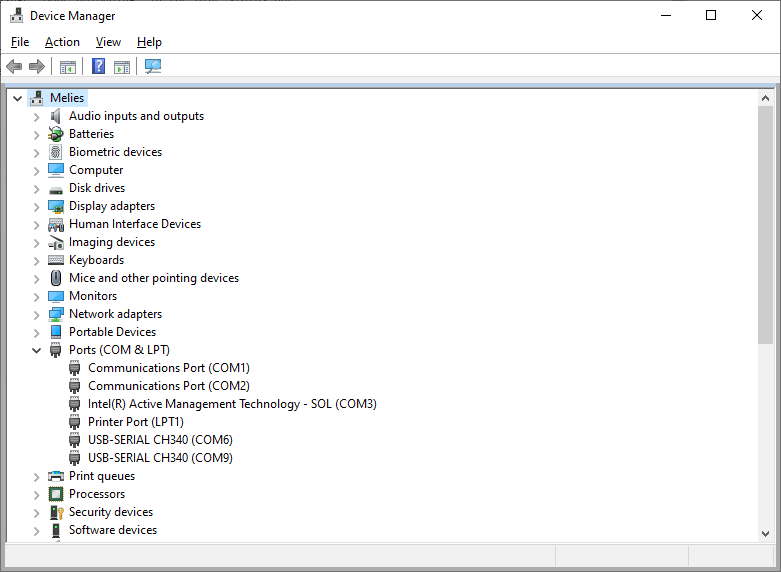
  
Watch the progress bar in the IDE and look for success with the message “Done burning bootloader.” in the blue status bar.



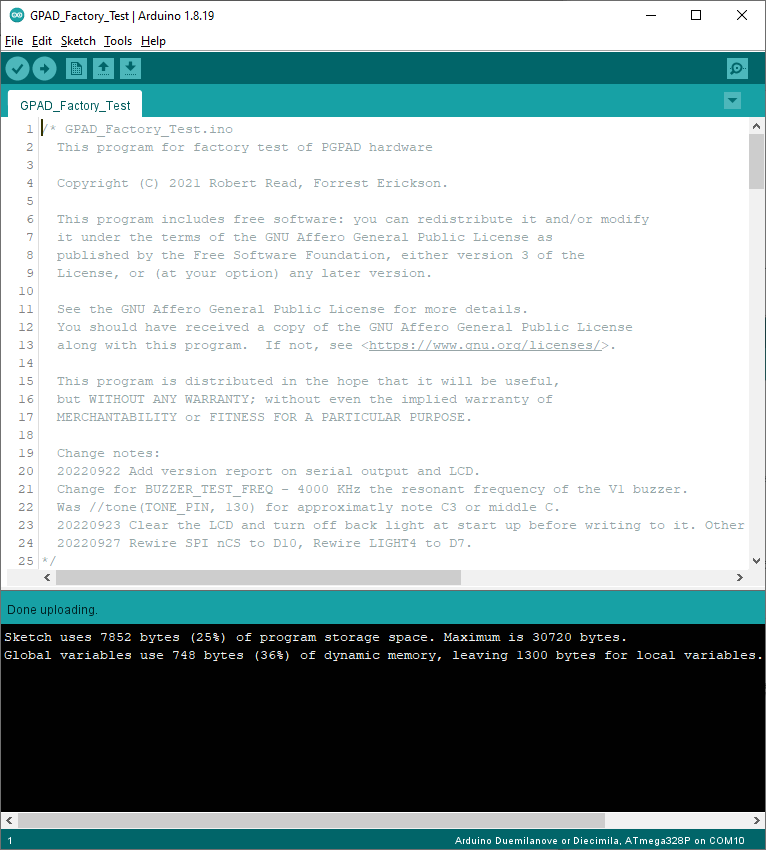
If R103 has not been removed, check that LED D102 is winking with a short “on time” and longer “off time” indicating that the boot loader has been loaded. Until any other sketch is loaded, this is the expected behaviour of the unit under test.

#### Load Factory Test Firmware.

Connect a USB cable to the DUT. Note the COM port enumerated in Device Manager Ports(COM&LPT) drop down



In the Arduino IDE, open the new file “GPAD\_Factory\_Test.ino”.  
Set the IDE for the COM port of the DUT. Using the Arduino IDE, compile and upload to the DUT the “GPAD\_Factory\_Test.ino”



Watch the progress bar in the IDE and look for success with the message “Done uploading” in the blue status bar.

Open a terminal to the COM port of the DUT and set for appropriate BAUD rate. Press the reset switch on the DUT and the LCD should display a message. The terminal should display a boot message too. This example is of a RealTerminal connected to the DUT. 

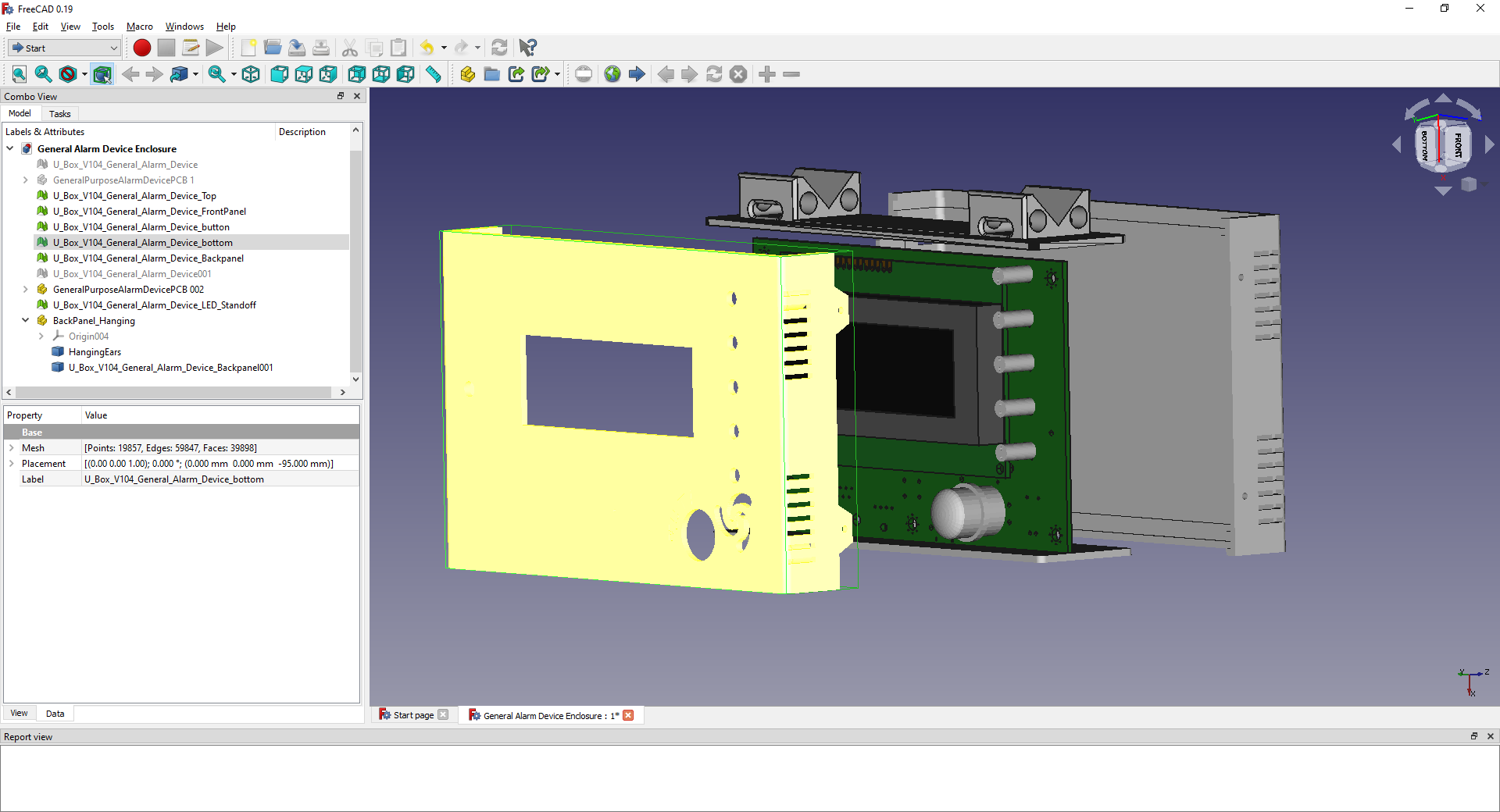
**Electrical Measurements After programming bootloader and firmware**

Measure and record by serial number the following electrical parameters.

Observe the current on the DUT. Press the Mute Switch S601 and the white LEDs D201-D205 should light. The Buzzer will make a sound. Record this full current in the table above.

**End of Rev2 Tests as of March 2023**

**Enclosure Assembly**

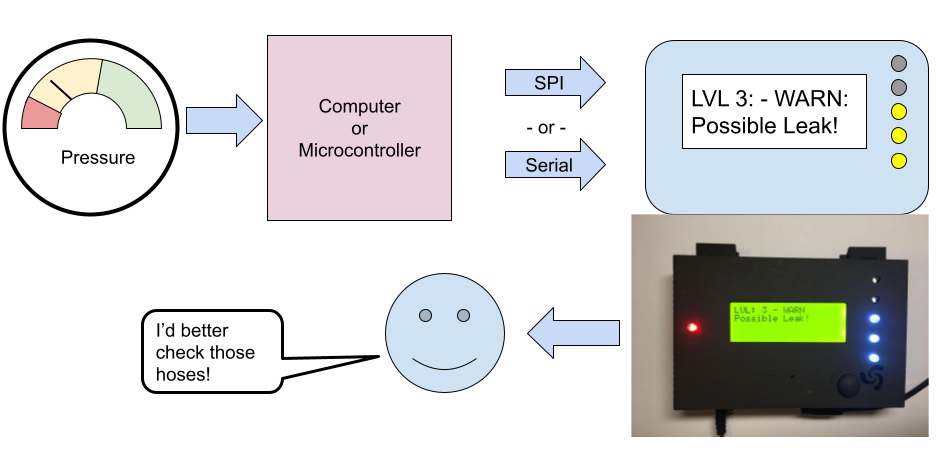


Use five screws to hold the printed wiring assembly (Green) to the yellow part.

The screws must be selected to fit to the enclosure as 3D printed. The enclosures printed of nylon worked with a sheet metal screw of thread diameter 0.14”   
  
Use four screws to fix the grey part to the yellow part, two on each side.

The screws must be selected to fit to the enclosure as 3D printed. The enclosures printed of nylon worked with a sheet metal screw of thread diameter 0.087”

1. **Operation instructions**



A [video demonstrating](https://youtu.be/Ie10kegTEIk) [3] of one use of the GPAD shows the usage and shows the flashing lights and allows you to listen to the audio of escalating urgency.

To develop and test the functioning of the GPAD, it can be driven easily by a serial interface from a laptop computer by simply using a serial monitor. We have designed a trivial interface of commands that the GPAD listens for on the serial interface. Each command begins with one of 4 characters and is terminated by a newline. These are:

1. “a” is the most important command and is used to set the alarm level. The “a” command is of the “aN”, where N is one of {0,1,2,3,4,5}, specifying increasing alarm levels respectively. These two characters are followed by up to 80 characters defining a message, terminated by an end-of-line (eoln) character.
2. “h” prints a helpful message.
3. “s” means “silence” – mute the GPAD.
4. “u” means “unsilence” or “unmute” – the GPAD is allowed to make noise.

The easiest way to send these serial commands is with a terminal program, or the “serial monitor” feature of the Arduino IDE, which is useful for testing. However, to actually program the GPAD to alert to a recondition, a library such as PySerial that allows a message to be transmitted over the serial port from general purpose programming language.

The simplest way to demonstrate the GPAD is by simply typing commands into the serial port on a computer that acts as a USB host. These commands are simply of the form:

aNmessage

where “a” is just the character “a” for alarm, “N” is a digit 0, 1, 2, 3, 4 or 5 representing the alarm level, and “message” is a message up to 80 characters long to be displayed.

However, a more typical expected use case is to drive the GPAD with a microcontroller that cannot act as a USB host, and so must use the I2C or SPI interface.

Conceptually, the sensing device treats the GPAD as an API—an Application Programmers Interface. The six alarm levels are conceptually named:

1. Silent,
2. Informational,
3. Problem,
4. Warning,
5. Critical,
6. Panic.

The exact meaning of the six levels can be defined by the user for the application. Of course, not all six must be used. However, the effect of the device should be clear: increasing severity leads to increasing sound and light levels produced by the device to obtain the attention of a human being. The GPAD has 5 LEDs, which are normally unlit in the Silent mode. Each increase in alarm level increases the number of LEDs lit. Our intention is that even from across a room a user may comprehend the alarm level at a glance.

Similarly, at increasing urgency levels, the GPAD plays a different rhythmic pattern of high-pitched sirens, which could be described as a “squeal” or a “scream”. This sound currently gets more urgent-sounding in terms of speed, rhythm, and frequency alternation as alarm level increases. Of course, the user is free to change or improve the coding of the GPAD itself, and a programmer will easily see where we implement the “songs’”.

To actually send the message, the programmer of the sensing device will likely use our software library. A C/C++ programmer will likely immediately understand the API from its actual declaration in the file alarm\_api.h:

#ifndef ALARM\_API

#define ALARM\_API

#include <Stream.h>

enum AlarmLevel { silent, informational, problem, warning, critical, panic };

// const char \*AlarmNames[] = { "OK ","INFO.","PROB.","WARN ","CRIT.","PANIC" };

const int NUM\_LEVELS = 6;

const int MAX\_MSG\_LEN = 80;

const int MAX\_BUFFER\_SIZE = MAX\_MSG\_LEN + 1;

typedef struct {

uint8\_t lvl;

// we will use a null-terminated string!

char msg[MAX\_MSG\_LEN+1];

} AlarmEvent;

int alarm\_event(AlarmEvent& event,Stream &serialport);

int alarm(AlarmLevel level,char \*str,Stream &serialport);

#endif

The API has two entry points: alarm and alarm\_event. The function alarm is simply a convenience function for calling AlarmEvent which constructs an AlarmEvent for the programmer. An AlarmEvent, as stated above, consists of an AlarmLevel and an 80-character (null-terminated) character array holding the message to be displayed on the LCD.

### Hardware Connection

The GPAD can be controlled through a USB connection, an I2C connection, or and SPI connection. The SPI connection is made with a RJ12 jack, which was chosen for its ubiquitous availability. The RJ12 jack can provide 12V power to the GPAD, although more commonly it will be powered through a standard 12V barrel jack with a “wall wart” transformer.

A close up of a device

Description automatically generated

Schematics in PDF form can be found in the folder at:

<https://github.com/PubInv/general-purpose-alarm-device/tree/main/Hardware/GeneralPurposeAlarmDevicePCB/PDF>

You can View Schematics on Line with the links at: <https://github.com/PubInv/general-purpose-alarm-device/tree/main/Hardware/GeneralPurposeAlarmDevicePCB>

The GPAD requires a 12V power supply via a standard 5.5mm OD 2.1mm pin. It draws less that 1000 mA, even at the highest alarm level.

#### SPI Peripheral Pins

The connector is JR12 6P6C and pin 1 is on the left then the connector is held with the contact pins up and facing the observer.

Note the GPAD is a 5V logic level peripheral device. Level shifters must be used when controlled by a 3.3V device.

| **Pin#** | **Signal Name** | **Note** |
| --- | --- | --- |
| 1 | nCS | Active low |
| 2 | COPI | Input to GPAD |
| 3 | GND |  |
| 4 | CLK/SCK | Input to GPAD |
| 5 | ControllerVcc | 5 Volts, an optional way to put power into the GPAD |
| 6 | CIPO | Output from GPAD, 5V. |

The typical RJ12 jack uses an offset six-pin connection to a PCB, which looks like this:

A diagram of a circuit diagram

Description automatically generated

Unfortunately, this format is not compatible with standard breadboards. In experimentation, you may wish to use male-to-female jumper wires.

**Muting**

The GPAD has a mute button. The button silences the sound but does not change the LED lighting. The GPAD stores an internal state as muted or not.

1. **Validation and characterization**

The most important use-case for the GPAD to be programmed by another microcontroller. The SPI interface is particularly useful for controlling a GPAD from another Arduino-class microcontroller. In fact, to make testing as easy as possible, we have software that allows a GPAD peripheral to be programmed by another GPAD acting as a controller. This can be found in the directory GPAD\_API\_SPI\_CONTROLLER (<https://github.com/PubInv/general-purpose-alarm-device/tree/main/Firmware/GPAD_API_SPI_CONTROLLER>). This code is a useful starting point for anyone programming a GPAD for any given use case because it exercises the SPI interface, and in particular, the GPAD Application Programmers Interface.

However, since it uses a second GPAD as a controller, most users may prefer to use the example described in the video above, which shows how to use an Arduino R4. Please watch our 4:12 minute [video demonstrating](https://youtu.be/Ie10kegTEIk) [3] of the use GPAD with an Arduino R4 before reading this section. This video demonstrates a circuit and code available at the GitHub Repository:

<https://github.com/PubInv/general-purpose-alarm-device/tree/main/Firmware/ExampleR4VoltageRead>

This circuit and code are meant to validate the full range of features of the GPAD. Although it uses a simple voltage read of a rotary potentiometer which can be conveniently turned by hand, it is meant to demonstrate the generality of the GPAD, which can be activated by any condition which can be detected or computed by a microcontroller.

A setup for use testing the GPAD with an Arduino R4 is depicted below:

A black electronic device with wires and wires

Description automatically generated

A Fritzing diagram of building this circuit is depicted below.

*A circuit board with wires and wires

Description automatically generated*

Note that in this diagram we represent the Six-position, six-contract (6P6C) plug as a straight-line header. This plug accepts the RJ12 connector described in the Operating Instructions section (see Figure # above), and should be wired that way, whether you use an RJ12 connector or a director wiring to an RJ12 cable. (Note: RJ12 connectors are very similar to RJ11 wires, but we need the to use the RJ12 wires.)

The rotary potentiometer (pot) is provided with 3.3V power and GND. The wiper of the pot is tied to the A0 analog input pin. After measuring the maximum and minimum voltage on the wiper, the following simple code, shown in the Arduino IDE, breaks the voltage range into intervals associated with the 6 alarm levels:

A screenshot of a computer program

Description automatically generated

The GPAD is activated by sending digital commands over a cable from a different microcontroller, such as the Arduino R4 in this example. The basic use case is that a condition which needs human attention is detected in whatever manner possible by a sensing microcontroller or computer, called the *controller*. For example, the sensing device might detect that a temperature is too high, or that the total flow through an airway in one minute is too low. It then calculates which of five alert levels (not counting the “no alert” condition, which could be considered a sixth) represents the severity of the condition, and a message of up to 80 characters that will inform a person how to remedy the problem. This message is displayed on the back-lit LCD on the GPAD itself. The LCD is dark when the alarm level is silent and back-lit in the other alarm levels.

The following is the code in the main loop that runs on the Arduino R4:

A screenshot of a computer program

Description automatically generated

It shows the use of the GPAD\_API which in the call to the entry point “alarm\_event”. It further demonstrates the use of an text message (up to 80 characters), by reporting the voltage (whether it changes the alarm level or not) in every message.

### Conclusion

The GPAD is designed to be a part of the [Freepsireco ecosystem](https://github.com/PubInv/freespireco) [2] and serve as the medical alarm in an ICU ward for the PolyVent open-source ventilator. Specifically, it is designed to alert clinicians to conditions which can be electronically computed by the PolyVent microcontroller. These include (but have not yet been coded):

1. A hose disconnect to the patient
2. A loss of pressure
3. Patient-ventilator asynchrony
4. In ability to achieve programmed Peak Inspiratory Pressure (PIP), etc.

The GPAD has been demoed in public quite a bit.

After some experimentation to obtain loudness, we built Revision 2 with two different buzzers (Digi-key part numbers 433-WST-1210T-ND and 102-CMI-1295-0585T-ND) which are both rated at 85 dB at 10cm. The LEDs are fairly bright. The basic electrical performance of this simple device seems clear.

However, the true value of an alarm device can only be measured by its ability to draw human attention, and then to inform a human being how to take action, and then not to distract them further. There are both simple and complex human/machine interaction performance metrics which we have not begun to measure for the GPAD. A good introduction to the complexity of this field can found in *Konkani, Oakley and Bauld (2012)* [1].

In the language of the field, the GPAD is not yet a “smart alarm” or an “alarm management system”. It is rather, an annunciation device. (If we called it that, however, non-experts would not understand its intended use.) We hope researchers in this area will find the GPAD a useful programmable component for such such systems and tool for such investigations.

**CRediT author statement**

**Robert L. Read:** Conceptualization, Software, Funding Acquisition, Writing – Original Draft.

**Lawrence Kincheloe:** Supervision, Methodology, Hardware.

(**Forrest) Lee Erickson:** Software, Validation, Hardware.

**Oluseyi 'Seyi' Adeniyi:** Software

Lawrence Kincheloe designed the enclosure and mentored the Oklahoma University capstone projects.

Lee Erickson performed schematic capture, designed the PCBs, wrote the assembly instructions and captured measurements on units during assembly and during some troubleshooting. Lee wrote initial factory test firmware and the factory test procedure which applies the firmware.

Oluseyi 'Seyi' Adeniyi expanded the factory test firmware to include “walking one” like tests for more thorough tests against shorts and bad LCD displays. Seyi also designed the alarm sounds.

**Acknowledgments**

We would like to thank two senior capstone engineering classes at the University of Oklahoma who made potential extensions to the GPAD. Although they have not yet been used in this project, they have clarified our thinking for future enhancements.

Funding: This work was supported by Public Invention, Austin, TX, a US 501c3 public charity.

**References**

[1] A. Konkani, B. Oakley, T. J. Bauld, Reducing hospital noise: a review of medical device alarm management. Biomedical Instrumentation & Technology, 46(6) (2012), 478-487. <https://array.aami.org/doi/full/10.2345/0899-8205-46.6.478>

[2] R. Read, The Freespireco Project: A complete, free-libre open-source, easily repairable, widely usable, safe and effective respiratory support medical device ecosystem. (2023) <https://github.com/PubInv/freespireco>.

[3] R. Read, GPAD Demonstration Video. <https://youtu.be/Ie10kegTEIk>. Accessed May 3rd, 2024.

[4] Mucco Test Horn literature. <https://muccosignal.com/product/warning-horns-with-16-tons/> (accessed 12.08.23)

[5] S.Z. Abidi, V. Suturin, R.L. Read, N. Bechard, 2023, June. A democratized open-source platform for medical device troubleshooting. In 2023 ASEE Annual Conference & Exposition. <https://peer.asee.org/a-democratized-open-source-platform-for-medical-device-troubleshooting>.

[6] Adafruit Tower Light - Red Yellow Green Alert Light with Buzzer – 12VDC. <https://www.adafruit.com/product/2993> (accessed 12.08.2023)

[7] HiLetgo 3D Printer Reprap Smart Controller (Model # 3-01-0889). <https://www.amazon.com/HiLetgo-Printer-Controller-Display-Arduino/dp/B07X378P8X>. (accessed 12.08.2023).

[8] The Ultimate box maker. <https://www.thingiverse.com/thing:1264391>. (accessed 24.10.2023).