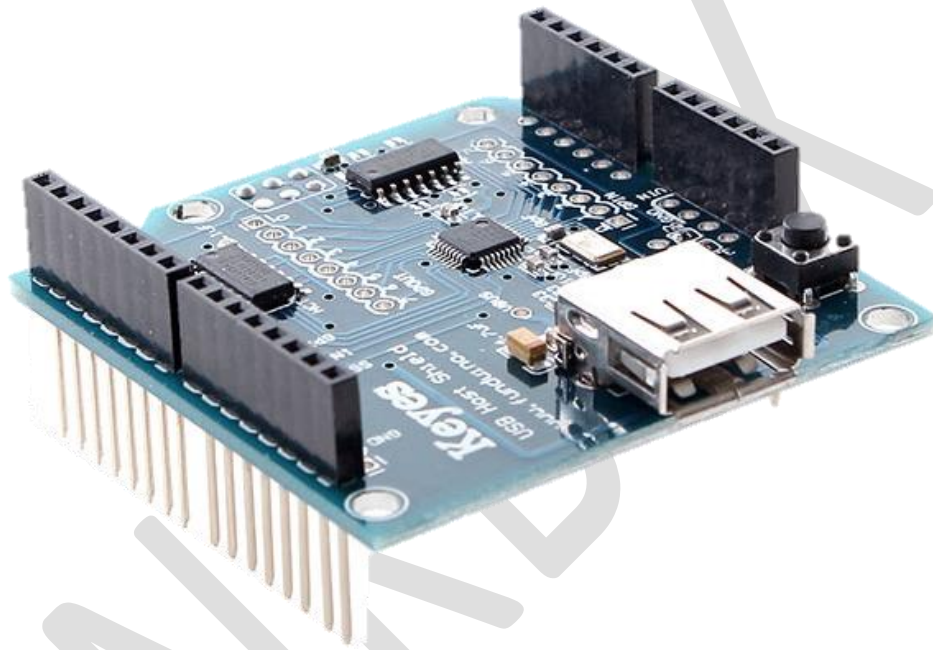


Keyes

USB Host Shield (Rev. 1)



General Description

The Keyes USB Host Shield allows you to connect a USB device to your Arduino board. It is based on the MAX3421E ([datasheet](#)), which is a USB peripheral/host controller containing the digital logic and analog circuitry necessary to implement a full-speed USB peripheral or a full-/low-speed host compliant to USB specification rev 2.0.

This is based on revision 2.0 of USB Host Shield. Thanks to new interface layout it is now compatible with more Arduinos - not only UNO and Duemilanove, but also Mega and Mega 2560 work with Standard variant of this shield out of the box. No more SPI re-wiring and code modifications - just solder included stackable connectors (2x3 ICSP connector's female side should be facing down), plug and play!

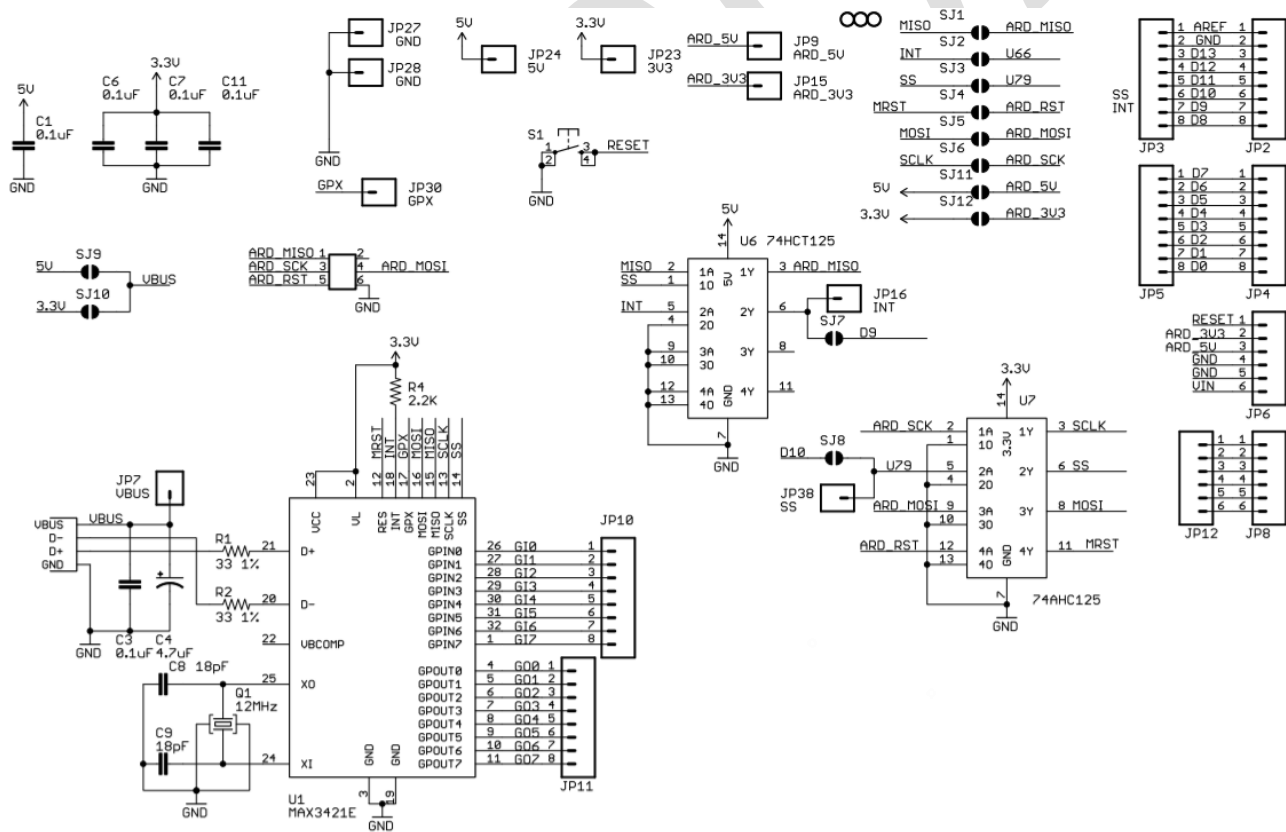
Specifications

- Works with standard (dual 5/3.3V) and 3.3V-only (for example, Arduino Pro) boards.
- Operates over the extended -40°C to +85°C temperature range
- Complies with USB Specification Revision 2.0 (Full-Speed 12Mbps Peripheral, Full-/Low-Speed 12Mbps/1.5Mbps Host)

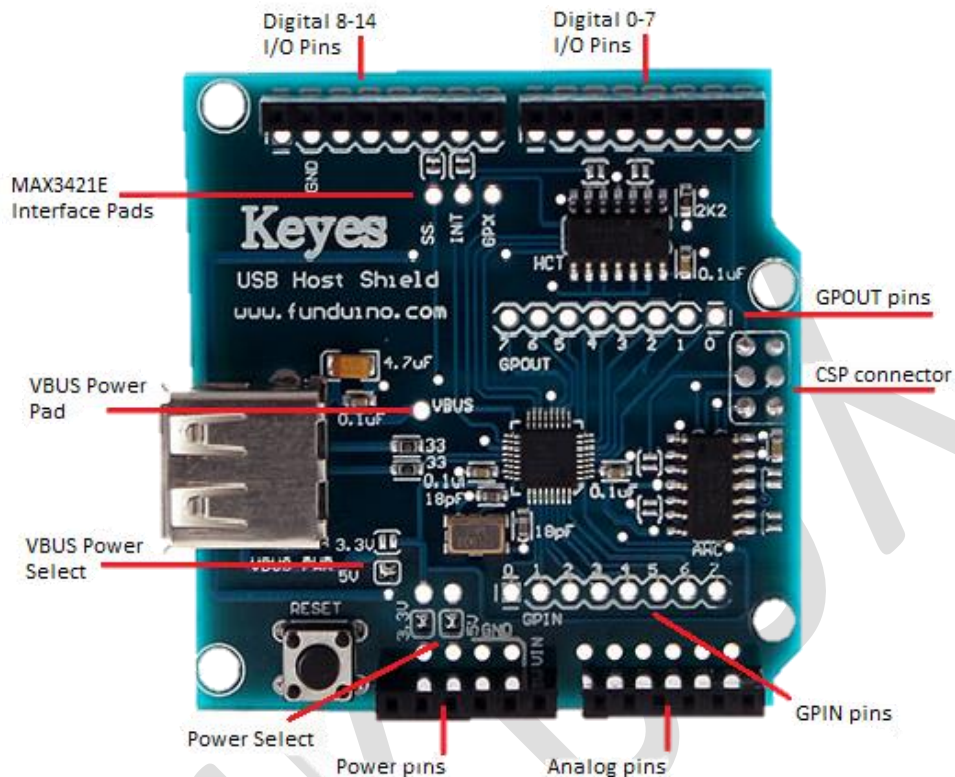
The following device classes are currently supported by the shield:

- HID devices, such as keyboards, mice, joysticks, etc.
- Mass storage devices, such as USB sticks, memory card readers, external hard drives (FAT32 Type File System - Arduino Mega only)

Schematic



Shield Overview



- **Power Select** 2 solder jumpers marked “5V” and “3.3V”. They are used for different power configurations. The configuration shown, when both jumpers are closed, is suitable for official Arduinos, such as UNO, Duemilanove, Mega and Mega 2560. See Power Options section for detailed explanation.
- **Power pins** are used to connect to power pins of Arduino board. RESET, 3.3V, 5V and GROUND signals from this connector are used.
- **Analog pins** are not used by the shield. They are provided to simplify mounting and provide pass-through for shields mounted atop of USB Host Shield in a stack.
- **GPIN pins**. Eight 3.3V general-purpose digital input pins of MAX3421E. They are used primarily to interface with buttons, rotary encoders and such. GPIN pins can also be programmed as a source of MAX3421E interrupt. An example of GPIN use can be seen in [digital camera controller](#) project.

- **ICSP connector** is used by the shield to send/receive data using SPI interface. SCK, MOSI, MISO and RESET signals from this connector are used.
- **GPOUT pins** are eight 3.3V general-purpose digital output pins of MAX3421E. They can be used for many purposes; I use it to drive HD44780-compatible character LCD, as can be seen in digital camera controller circuit, as well as this [keyboard example](#). Max_LCD library which is part of standard USB Host library software package uses some of GPOUT pins.
- **Digital I/O pins 0-7**, like already mentioned analog pins are not used by the shield and provided only for convenience.
- **Digital I/O pins 8-13**. In this group, the shield in its default configuration uses pins 9 and 10 for INT and SS interface signals. However, standard-sized Arduino boards, such as Duemilanove and UNO have SPI signals routed to pins 11-13 in addition to ICSP connector, therefore shields using pins 11-13 combined with standard-sized Arduinos will interfere with SPI. INT and SS signals can be re-assigned to other pins (see below); SPI signals cannot.
- **MAX3421E interface pads** are used to make shield modifications easier. Pads for SS and INT signals are routed to Arduino pins 10 and 9 via solder jumpers. In case pin is taken by other shield a re-routing is necessary, a trace is cut and corresponding pad is connected with another suitable Arduino I/O pin with a wire. To undo the operation, a wire is removed and jumper is closed. See interface modifications section for more information. GPX pin is not used and is available on a separate pad to facilitate further expansion. It can be used as a second interrupt pin of MAX3421E.
- **VBUS power pad**. This pad is used in advanced power configurations, described in Power Options section.

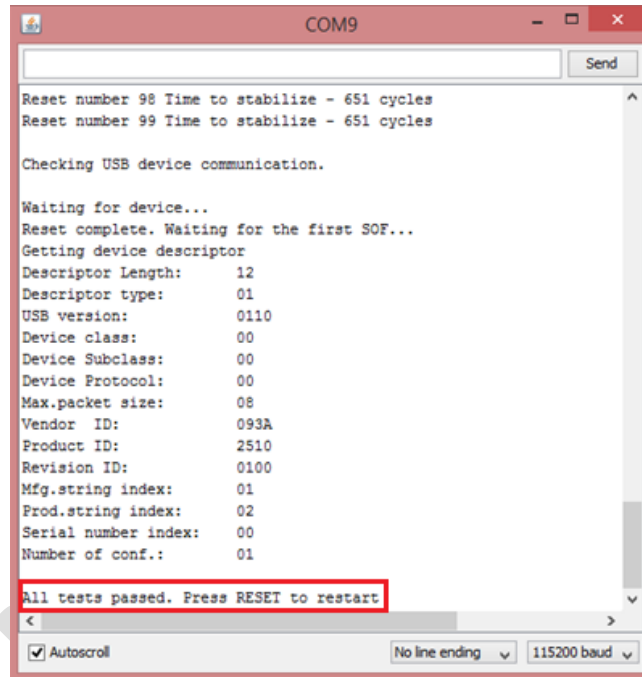
How to Test

1. Mount the Keyes USB Host Shield into your Arduino.
2. Download [USB Host Shield Library](#) and extract it to library folder in your Arduino directory.
3. Open board_qc then click upload. You can find this sketch at Arduino IDE File> Examples > USB_Host_Shield > board_qc.
4. Click Serial Monitor and set the baud rate to 115200.
5. When prompted, plug in a USB device to the USB Host Shield's USB port.

```
Checking USB device communication.

Waiting for device...
Reset complete. Waiting for the first SOF...
```

6. If the shield is fully functional, you should be seeing this Serial Monitor:



The screenshot shows the Serial Monitor window for COM9. The text displayed is as follows:

```
Reset number 98 Time to stabilize - 651 cycles
Reset number 99 Time to stabilize - 651 cycles

Checking USB device communication.

Waiting for device...
Reset complete. Waiting for the first SOF...
Getting device descriptor
Descriptor Length: 12
Descriptor type: 01
USB version: 0110
Device class: 00
Device Subclass: 00
Device Protocol: 00
Max.packet size: 08
Vendor ID: 093A
Product ID: 2510
Revision ID: 0100
Mfg.string index: 01
Prod.string index: 02
Serial number index: 00
Number of conf.: 01

All tests passed. Press RESET to restart
```

The last line, "All tests passed. Press RESET to restart", is highlighted with a red box. The window also shows a "Send" button, a scrollbar, and settings for "Autoscroll" (checked), "No line ending", and "115200 baud".

Sample Program

Track a mouse actions through Arduino. This program will display the status of the mouse connected to the Keys USB Host shield. This sketch is also included in the USB Host Library.

You need:

- Arduino
- Keys USB Host Shield

Sketch

```
#include <hidboot.h>
#include <usbhub.h>
// Satisfy IDE, which only needs to see the include statment in the in
o.
#ifdef dobogusinclude
#include <spi4teensy3.h>
#endif

class MouseRptParser : public MouseReportParser
```

```

{
protected:
    virtual void OnMouseMove      (MOUSEINFO *mi);
    virtual void OnLeftButtonUp   (MOUSEINFO *mi);
    virtual void OnLeftButtonDown (MOUSEINFO *mi);
    virtual void OnRightButtonUp  (MOUSEINFO *mi);
    virtual void OnRightButtonDown (MOUSEINFO *mi);
    virtual void OnMiddleButtonUp (MOUSEINFO *mi);
    virtual void OnMiddleButtonDown (MOUSEINFO *mi);
};
void MouseRptParser::OnMouseMove (MOUSEINFO *mi)
{
    Serial.print ("dx=");
    Serial.print (mi->dX, DEC);
    Serial.print (" dy=");
    Serial.println (mi->dY, DEC);
};
void MouseRptParser::OnLeftButtonUp (MOUSEINFO *mi)
{
    Serial.println ("L Butt Up");
};
void MouseRptParser::OnLeftButtonDown (MOUSEINFO *mi)
{
    Serial.println ("L Butt Dn");
};
void MouseRptParser::OnRightButtonUp (MOUSEINFO *mi)
{
    Serial.println ("R Butt Up");
};
void MouseRptParser::OnRightButtonDown (MOUSEINFO *mi)
{
    Serial.println ("R Butt Dn");
};
void MouseRptParser::OnMiddleButtonUp (MOUSEINFO *mi)
{
    Serial.println ("M Butt Up");
};
void MouseRptParser::OnMiddleButtonDown (MOUSEINFO *mi)
{
    Serial.println ("M Butt Dn");
};

USB      Usb;
USBHub   Hub (&Usb);
HIDBoot<HID_PROTOCOL_MOUSE> HidMouse (&Usb);

uint32_t next_time;

MouseRptParser Prs;

void setup ()
{
    Serial.begin ( 115200 );
}

```

```

    while (!Serial); // Wait for serial port to connect - used on
Leonardo, Teensy and other boards with built-in USB CDC serial
connection
    Serial.println("Start");

    if (Usb.Init() == -1)
        Serial.println("OSC did not start.");

    delay( 200 );

    next_time = millis() + 5000;
    HidMouse.SetReportParser(0, (HIDReportParser*)&Prs);
}

void loop()
{
    Usb.Task();
}

```

Results

```

COM9
Send
dx=-1 dy=0
R Butt Dn
R Butt Up
R Butt Dn
R Butt Up
L Butt Up
M Butt Dn
M Butt Up
M Butt Dn
M Butt Up
L Butt Dn
L Butt Up
dx=1 dy=-1
dx=1 dy=-1
dx=1 dy=-1
dx=2 dy=-2
dx=-2 dy=-2
dx=-1 dy=0
dx=-1 dy=0
dx=-2 dy=-1
dx=-1 dy=-1
dx=-2 dy=0
dx=-1 dy=-1
dx=-1 dy=-1

```

Autoscroll
 No line ending
115200 baud