Preview

Many people with disabilities (PwD) use specialised controllers, such as joysticks and accessible buttons, to interact with digital devices. The 3.5mm audio jack has become the standard interface for these assistive technology (AT) devices. We present a versatile AT interface that allows a microcontroller to act as a controller *or* to have a controller plugged into it, to assist prototyping new solutions for PwD.

A Versatile Assistive Technology Interface

Matthew Oppenheim
Lancaster University

Steve Hodges
Microsoft Research

Many people with disabilities (PwD) require specialised controllers, such as joysticks and accessible buttons, to enable them to interact with digital devices. This includes the applications they use on computers, tablets and smartphones—for instance to create speech. The 3.5mm audio socket has become established as the standard physical connector for these assistive technology (AT) devices. Accessible buttons with a 3.5mm plug are known as 'access switches' since they act as a simple on/off switches. Devices with 3.5mm sockets that are controlled by these access switches are known as 'switchable devices'. Many modern digital devices—such as Android and iOS smartphones—support interaction in this way, using a Bluetooth or USB switchable device interface along with an operating system interaction mode called 'switch access' or 'switch control' respectively.

This article details a versatile AT interface board that we have developed to allow a microcontroller to act as an access switch *or* as a switchable device. This makes it easier to prototype new solutions for PwD, whether the aim is to create a new input device—such as a flex sensor that acts as a light-touch 'button'—or to create a new switchable device which is to be controlled by existing access switches. Our board incorporates the same 3.5mm audio socket connectors that most AT devices and controllers use as an interface. We include two of these sockets, one of which is associated with each of two separate channels. The functionality of each channel can be independently configured through software, meaning that our design can operate as two access switches, two switchable devices or one of each. Existing access switches may be plugged into the board's sockets, or plug-to-plug 3.5mm cables may be used to connect the board to an existing switchable device.

We hope that others will see our solution as a resource to help create new assistive technology projects.

Assistive technology switch interface

The standard connector for assistive technology devices is the same 3.5mm connector that audio headphones use. An access switch has a 3.5mm plug. A switchable device has a 3.5mm socket. Table 1 shows the differences between the interfaces for access switches and switchable devices.

Figure 1 shows how a stereo 3.5mm audio plug is configured. When operated, an access switch connects the tip to the sleeve of the plug; when released the tip and sleeve are disconnected again. The plug is inserted into the 3.5mm socket on the switchable device to be controlled.

Analog signals are supported too, but are beyond the scope of this article. An explanation of analog access switches can be found on the Adafruit website7.

The switchable device must be capable of detecting when the tip and sleeve connections on its 3.5mm socket are shorted and when they are open circuit. In this way, the device can react when the switch is turned on or off.

To enable the 3.5mm sockets on our board to act as access switches, 3.5mm plug-to-plug adapter cables are used to physically convert the sockets to plugs. These cables only cost a couple of dollars and are easily sourced from the usual online retailers.

Table 1: Access switch and switchable device setup.

	Access switch	Switchable device
Physical interface	3.5mm plug	3.5mm socket
Switch closed action	Tip and sleeve connected	Detects low impedance between the tip and sleeve
Switch open action	Tip and sleeve unconnected	Detects high impedance between tip and sleeve

Leveraging Jacdac

Jacdac 7 is an open-source modular electronics prototyping system under active development by Microsoft Research and Lancaster University. Jacdac allows small sensors and devices to be connected together dynamically so that the assembly recognises what is being added or taken away; in this sense it's a bit like USB, but cheaper! Each Jacdac module is designed for a specific function, such as a button or a slider. Modules communicate using a custom protocol over a three-pin PCB edge connector; the three pins are power, common and signal. They are connected together using Jacdac cables.

Each Jacdac input module has a cheap microcontroller that identifies that board to a main 'brain' module The 'brain' module contains a more complex microcontroller that controls the assembly and can be connected to a PC for programming through USB. As microcontrollers suitable for Jacdac input modules cost as little as 5 cents each, it is inexpensive to add a microcontroller to each board. These module microcontrollers are flashed with firmware that implements both the functionality of the board and the Jacdac communications code that allows the module to act as a part of a Jacdac assembly. A reference Jacdac board designed in KiCAD can be found on GitHub7. Jacdac kits are now commercially available from Kittenbot7.

The Jacdac system works with the learn-to-program web-based MakeCode programming platform7. When Jacdac is used, the MakeCode programming interface automatically populates with the relevant code blocks for the specific Jacdac modules that are connected. This makes it pretty easy to develop code to run on a Jacdac brain, such as a BBC micro:bit V2.

Jacdac enables people who do not have a specialist electronics, engineering or coding skill-set to construct and program novel devices. This makes the Jacdac system an ideal platform for experimenting with new types of assistive technology devices and to interact with existing assistive technology devices. All that's needed is a way to interface Jacdac with the 3.5mm plug in the right way!

Hardware

We received suggestions from colleagues to make our board capable of handling two channels, because access switches are often used in pairs. Of course, if more than two channels are needed, it's easy to use more than one board. Each 3.5mm socket has three signal traces: tip, ring and barrel. Each of these signal traces has over-current and over-voltage protection. The over-current protection is provided by a 10k resistor in series with the signal path to the MCU. The over-voltage protection is provided by BAT54S Schottky clamping diodes used to protect the Jacdac data line. There are two BAT54S diodes in a single three-pin package. The diodes are connected to the power and ground rails as well as the signal trace, so that voltages more than a diode drop above the power rail or more than a diode drop less than the ground rail are protected against.

The Jacdac interface uses an NCP114ASN330T1G 3.3V low dropout voltage regulator, rated to 300mA, to provide isolation from the Jacdac power rail. The Jacdac power rail is specified to be 3.7V to 5.2V. According to the regulator's data sheet, the typical dropout voltage with an output of 3.3V and a current output of 300mA is 0.15V, so there is plenty of headroom even if the power rail is the minimum voltage in the specified range.

The data and power lines from the Jacdac connector have electrostatic discharge (ESD) protection using RCLAMP0521-P ultra-low capacitance ESD diodes. The two Schottky diodes in another BAT54S provide over-voltage and under-voltage protection on the data line.

The STM32 was the most suitable MCU available for this project from the range of MCUs that were part of the Jacdac system at the time the board was designed. The STM32 has sufficient input-output pins to enable all of the intended functionality without multiplexing. A push button switch was added onto one of the input pins. At the time of design, one suggested Jacdac requirement was to require a physical button to be depressed to allow a firmware update on the MCU. This is to increase security, as requiring a button to be pushed prior to a firmware update helps to prevent a malicious remote firmware update from occurring.

The final PCBA is shown in Figure 2 and the schematic for the design is shown in Figure 3. KiCad v6 was used to prepare the schematics and to lay out the PCB. The design files and bill of materials were sent to JLCPCB.com to manufacture and populate the boards. JLCPCB's website has instructions on how to prepare the files needed for manufacture using KiCad, including how to automate creating the bill of materials by adding their component part numbers to the symbols as an extra field. Using the components that JLCPCB has in stock enables JLCPCB to supply them and populate the boards. Receiving fully populated boards saves a lot of time over soldering them up yourself. The downside is that with a prototype, populating and testing one section at a time can be a useful way of locating design errors. In the unlikely event that you make any.

How the circuit works as a switchable device

In this mode of operation, each channel needs to behave like a switch. In a previous project, we made a prototype switch that was tested using an access-switch-to-USB adapter. This adapter is a standard way of connecting switches to a PC or Apple device to interact with switch-controlled software. A common use of switch-controlled software is to create speech. However, user feedback indicated that our prototype switch interface board did not work with all of the different devices that were tried with it.

Of course, we could not anticipate how the design would be abused in the Real World. The board may have been connected to a toy adapted for use with an access switch. In this configuration, the current needed to operate the toy often passes through the switch interface. To ensure that our new board, presented in this article, works with modified toys like this (unlike our earlier prototype), we use a TLP240A opto-isolated relay to connect and disconnect the tip and the sleeve of the 3.5mm connector. In this case, the pins on the STM32 microcontroller (MCU) that connect to the tip and sleeve in order to detect access switches are configured as inputs which puts them in a high impedance state. The relay is rated to 500mA of continual current. This setup allows the board to control switchable devices.

How the circuit responds to a switch

In this second mode of operation, the MCU pin connected to the sleeve of the connector is configured as an output and set low. Since the pin is configured as an output, it is in a low impedance state.

The MCU pin connected to the tip of the connector is configured to behave as a button through the Jacdac configuration script—see Listing 1. The ‘NO_PIN’ parameter means that the button is not configured to have a backlight. When the attached switch is open or closed, the voltage difference between the tip and ring varies. This difference is detected as button actuation by the Jacdac software.

```
void app_init_services() {

    pin_setup_output(SLEEVE_1);

    pin_set(SLEEVE_1, 0);

    pin_setup_output(SLEEVE_2);

    pin_set(SLEEVE_2, 0);

    button_init(TIP_1, 0, NO_PIN ); // button, high/low, backlight

    button_init(TIP_2, 0, NO_PIN);

}
```

Listing 1: Jacdac service script.

Figure 4 shows the board connected to both an external switch and to a switch-to-USB adapter. These allowed the board to be verified as responding to an external switch input and that the board appears as a switch. The rocker switch was connected using a couple of Wago 221 quick connects. These are great connectors to quickly connect together low frequency devices. As mentioned above, the switch-to-USB adapter is a standard device used to interface assistive technology switches to devices with USB.

The voltages at the STM32 pins connected to the tip and sleeve of the 3.5mm connector were measured, with the attached switch open and closed. The measured voltages were well in specification for appearing as a logic high or low. The 10k resistors on the signal rails were lowered in value to around 5.3k by stacking 11k resistors on top of the existing 10k resistors. The voltage values were measured again and found to be closer to the rail voltages than when using the 10k resistors on their own. Future designs will replace the 10k resistor with a 4.7k resistor. This value produces voltages with a better margin of noise immunity, while still giving adequate input current protection.

Doing some classic resistor bridge calculations showed that the switch sees an input impedance of about 85 Ohms when the MCU pin it is connected to is set as an output.

The board was verified as working to appear as a switch using the access-switch-to-USB adapter. One of the most widely used software packages used in assistive technology for creating speech responded to the channel when it was configured as an open or closed switch. The channel was verified as being able to pass the 500mA that it is rated to. Over current testing took place. The failure mode was that the current was throttled and then further reduced as the relay heated. This is how FETs behave—their resistance increases with temperature, throttling current. Unlike bipolar transistors where the resistance can decrease with temperature, leading to thermal runaway and catastrophic failure.

Firmware

The code was written in the C programming language using Visual Studio Code with Debian as the operating system. It was flashed onto the STM microcontroller on our board using a home-made Black Magic Probe 7 (BMP).

The BMP is a microcontroller debugger with a built in GNU debugger (GDB) server and UART. Our home-made BMP was made using a 128k Bluepill board, following the instructions provided online7. A generic ST-Link v2 bought from eBay was used to flash the necessary firmware onto the Bluepill board. Be sure to use the 128k version of the Bluepill board to make a BMP or you will not be able to flash the firmware. Some boards sold as a 64k Bluepill do have 128k of usable RAM, but finding one of these is a bit of a lottery. We bought a Bluepill that had been reworked to have a 128k processor from eBay.

The BMP is a useful tool that can run interactively from the command line or from scripts. We wrote a couple of short scripts to automate loading the hex file produced by compiling the C code for the board and another to display memory locations from the STM32 flash memory. GUI based programmers and debuggers offer some convenience, but interacting through the command line offers a lower level of abstraction from the embedded hardware. There is one less piece of software in-between you and the target processor that can potentially introduce errors.

JacConnect programming connector

The programming/debugging interface to the board uses the JacConnect7 connector proposed by Microsoft. The PCB side of this connector is a pattern of five 22mil holes with 50mil pin spacing and 62mil row spacing. These holes can be seen on the top left of the PCBA shown in Figure 2. The cable side of the connector is a 0.05” male header. The difference in spacing in-between the header pins and the spacing of the holes on the board creates enough friction to keep the plug and socket connected for programming and debugging. The system works well in practice but you need to make your own programming cable.

Jacdac firmware

One of the goals of Jacdac is to abstract away—to ‘hide’—the intricacies of module firmware. Instead, we write a simple description of the board’s functionality using pre-existing Jacdac ‘services’. Typically a module does one thing, presenting one of the these services. However, our assistive technology interface can act as two different devices—as an input for an access switch or as an output to control a switchable device. This means that each of the two channels on this board needs to appear as two different services to the Jacdac infrastructure.

To enable each channel to act as two devices, we added a ‘virtual relay’ to each of the channels. Switching the relay from on to off toggles the functionality of the associated channel.

When the channel is controlling a switch, it exposes a Jacdac relay service and appears as a relay on the Jacdac web interface. When it the channel is being controlled by a switch, the channel appears as a button on the Jacdac web interface. Two memory locations in non-volatile memory record the state that each channel is in. When the functionality of a channel is changed using this virtual relay, a new value is written to the appropriate memory location. Then the device is restarted through software. When the board starts up, the values representing each channel’s state are read from non-volatile memory and the channel is configured accordingly.

Non-volatile memory has a finite number of read-writes, but this number is so high that the frequency that the channel states are changed poses no risk of causing the memory location to fail through over-use.

We encountered some issues implementing this functionality. Initially the board was not responding correctly when the state of a channel was changed. The BMP proved its value with helping to debug this issue. The debugger allows memory locations to be dumped interactively. By looking at the memory locations where channel state was being saved, we quickly saw that these memory locations were already in use. Looking through the linker script, we found that these memory locations are used by the bootloader—see Listing 2. The memory locations used for saving state were changed to an unused area of flash memory. The board then worked as expected, with state saved when the virtual relay is switched and the board resetting with the expected functionality enabled.

```
MEMORY {
RAM (rwx)   : ORIGIN = 0x20000000, LENGTH = 8K
FLASH (rx)  : ORIGIN = 0x8000000, LENGTH = 32K - 4K - 0K - 12
}
INCLUDE jacdac-stm32x0/ld/gcc_arm.ld
```

Listing 2: Linker script, showing that the last 4k of flash is used for the bootloader.

Figure 5 shows a typical display on the Jacdac web interface for our board, showing one channel set up as a relay to control an external device (switchable device mode) and the second channel indicating the status of an attached switch (access switch mode).

Discussion

The board we made works well. It's not realistic to expect a device that is created in low volume to be as polished as a product that is mass produced. However, will people who use assistive technology and the technologists who support them accept a self-assembly system like Jacdac? The technologists who work with assistive technology have an excellent knowledge of assistive technology devices and how to install these in the best way to assist the people who use them. They may not have the time or inclination to also assemble and test equipment. On the other hand, the economics and time spent in manufacturing a new piece of assistive technology which may only be sold in low-volume make a rapid-prototyping system like Jacdac attractive for getting devices to the people who need them.

The experience gained in designing and testing an access switch is already being used in the design of a new assistive technology device.

References

[1] https://learn.adafruit.com/diy-adaptive-game-controllers/interfacing-buttons-and-switches
[2] https://microsoft.github.io/jacdac-docs/
[3] https://makecode.microbit.org
[4] https://circuitcellar.com/research-design-hub/projects/devices-aid-speech-for-people-with-disabilities/
[5] https://github.com/mattoppenheim/jacdac
[6] https://www.kittenbot.cc/products/kittenbot-jacdac-kit-for-micro-bit
[7] https://arcade.makecode.com/hardware/dbg
[8] https://black-magic.org/index.html
[9] https://github.com/mmoskal/blackmagic-bluepill/

Illustrations

Can we re-draw more like the following, to make it clear that we're showing how the switch is wired into the 3.5mm plug? As it currently stands, some might interpret it as how one should connect to the 3.5mm plug.

Figure 1: How an on/off access switch is wired to a 3.5mm audio socket, then connected with a 3.5mm audio plug. Note that stereo plugs are commonly used but it's also possible to use mono or 4-pole plugs.

Figure 2: Access switch interface PCBA.

Figure 3: Access switch interface schematic.

Figure 4: Testing the board using a switch as an input device and an access-switch-to-USB adapter as an output.

Figure 5: Jacdac web interface showing the board with one channel configured as a relay, the other channel configured as a switch input.