Development of an Android accessory interface to CPIT AVR training kits

Yao Li

Christchurch Polytechnic Institute of Technology
Christchurch, New Zealand

ABSTRACT: Christchurch Polytechnic Institute of Technology (CPIT) has a long history of teaching embedded systems and microcontrollers in their engineering programmes. The AVR microcontroller training kit was developed as a platform for teaching and for student projects. This article presents the development of an Android accessory interface to the training kit. The hardware design was based on the Max3421e USB controller. The accessory's software was designed using the object-oriented concept and the top-down approach. The general behaviour of the accessory was defined in the base class and the USB protocol support defined in its subclass. These are hardware independent and can easily be extended to a subclass with specific hardware support, such as the Max3421e or any microcontroller's built-in USB controllers. The Android accessory interface together with the AVR microcontroller training kit will be the platform for students to develop embedded systems, which can fully use the powerful features of Android devices.

INTRODUCTION

Cell-phones and tablets powered by Android are powerful mobile platforms with an Internet connection and a variety of built-in sensors, such as cameras, GPS, G-sensors, proximity sensors, accelerometers and touch screens. In 2011, Google released the Android Open Accessory protocol, which provides Android devices with the ability to communicate with external activities [1]. This has opened up opportunities for students to develop embedded systems that take advantage of powerful Android devices.

Christchurch Polytechnic Institute of Technology (CPIT) has a long history of teaching embedded systems and microcontrollers in their engineering programmes. *Real world* projects have always been used as assignments for teaching and assessment. Examples of these projects include a Boom Sprayer Controller, a Well Monitoring System, ZigBee wireless networks and so on [2][3]. Among these projects, the AVR microcontroller training kits, developed in-house, were used for the prototype development.

The design and implementation of an Android accessory interface for the training kit is discussed in this article, one which allows students to develop their embedded systems capable of interacting with Android devices if needed. The hardware design was based on the Max3421e USB controller. The accessory software was designed using the object-oriented concept and the top-down approach. The base class defines the general behaviour of the accessory and the subclasses are derived for the USB protocol support, the Max3421e controller and, then, the hardware abstract layer (HAL). A student project using the Android accessory interface is also discussed.

RELATED WORK

Android Open Accessory Protocol

Android, developed by the Open Handset Alliance (an alliance led by Google), is an open-source software stack for a wide range of mobile devices including cell-phones and tablets [4]. Before the Android Open Accessory protocol was released in 2011, Android devices were only capable of acting as USB devices and could not initiate connections with external USB devices.

Android Open Accessory support is included in Android 3.1 and higher, and supported through an Add-On Library in Android 2.3.4 and higher [1].

The initial version of Android Open Accessory Protocol defined how an accessory detects and sets up communication with an Android device. In general, an Android accessory is a USB host that should carry out the following steps [5]:

- Wait for and detect a connected device.
- Determine the device's accessory mode support.
- Attempt to start the device in accessory mode if needed.
- Establish communication with the device if it supports the Android Accessory protocol.

Version 2.0 has added two new features: audio output and support for the accessory acting as one or more Human Interface Device (HID) to the Android device [6]. Version 1.0 requires an Android app in the device to communicate with the accessory, while Version 2.0 allows the accessory to access the Android system directly.

AVR Microcontroller Training Kits

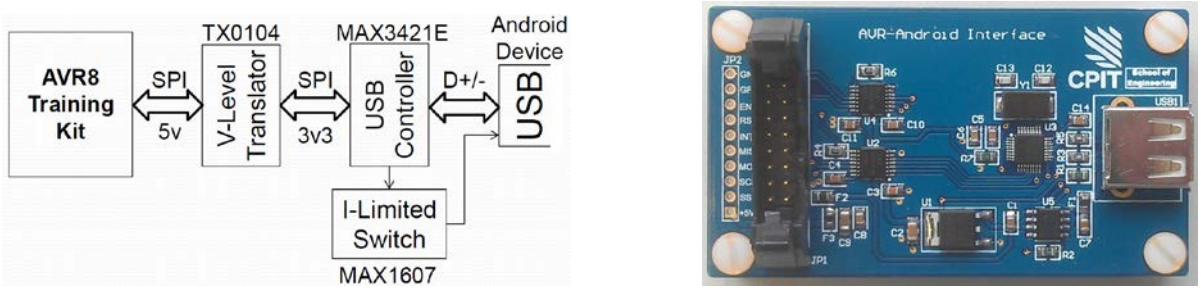
The AVR microcontroller training kit, developed in-house at CPIT, was designed specifically as a platform not only for teaching in class, but also for students' design projects in microcontrollers and embedded systems [7][8]. It has an integrated debugger/programmer, which allows the in-circuit debugging and programming of the ATMega128, as well as other supported ATMega microcontrollers via an external JTAG connector. The USB interface allows the debugger and the ATMega128 to communicate with a PC and also provides an optional power supply for the training kit. The *all-in-one* solution of the training kit provides a platform for easy hardware prototyping.

For the AVR microcontroller training kit to interact with an Android device, the interface module must adhere to the Android Open Accessory Protocol [1]. The microcontroller with the interface is the Android accessory and acts as the USB host. The Android device is in accessory mode, acting in the USB peripheral role.

As specified by USB 2.0 Specifications, the USB host powers the bus and initiates the connection [9]. Therefore, the Android USB accessories would detect Android devices that support accessory mode, initiate the connection and also provide 500mA at 5V for charging power. The Android devices in accessory mode have the ability to interact with the USB hardware.

HARDWARE DESIGN

The block diagram of the hardware design is shown in Figure 1a.



(a) (b)

Figure 1: The Android accessory interface: a) the block diagram; and b) the final product.

Central to the hardware design is the Max3421e USB Controller. The controller contains all circuitry necessary to implement a full-speed USB peripheral or a full-/low-speed host compliant to USB specification 2.0 [10]. The built-in transceiver, with features of ±15kV ESD protection, can be connected directly to D+/D- of the USB A connector via 33Ω±1% resistors when working as a USB host. Its internal serial interface engine (SIE) handles low-level USB protocol details, such as error checking and bus retries. The high level data transaction is controlled and initiated by a microcontroller via the SPI interface.

The Max3421e is powered by 3.3V, while the AVR microcontroller training kit is powered by 5V. Two voltage-level translators (i.e. TX0104) are used. Apart from the SPI connections (SCK, MOSI, MISO and SS#), the other three pins from the Max3421e (INT, GPX and RES#) are also connected to the microcontroller via the voltage-level translators. INT is the interrupt output that is asserted when a USB event occurs. GPX is a multiplexed output indicating various statuses. RES# is the chip reset so that the microcontroller can reset the controller.

With the SPI in its lower nibble, Port B of the ATMega128 microcontroller is used. Three pins of the higher nibble are connected to INT, GPX and RES# and the remaining pin is used to enable the voltage-level translators.

As a USB host, the circuitry supplies 5V power to the VBUS by the current-limited switch, i.e. the Max1067. The switch is enabled by one of the general purpose outputs from the Max3421e.

The final product of the Android accessory interface is shown in Figure 1b.

SOFTWARE DESIGN AND IMPLEMENTATION

The Android accessory software was designed using the object-oriented concept. The class diagram is shown in Figure 2.

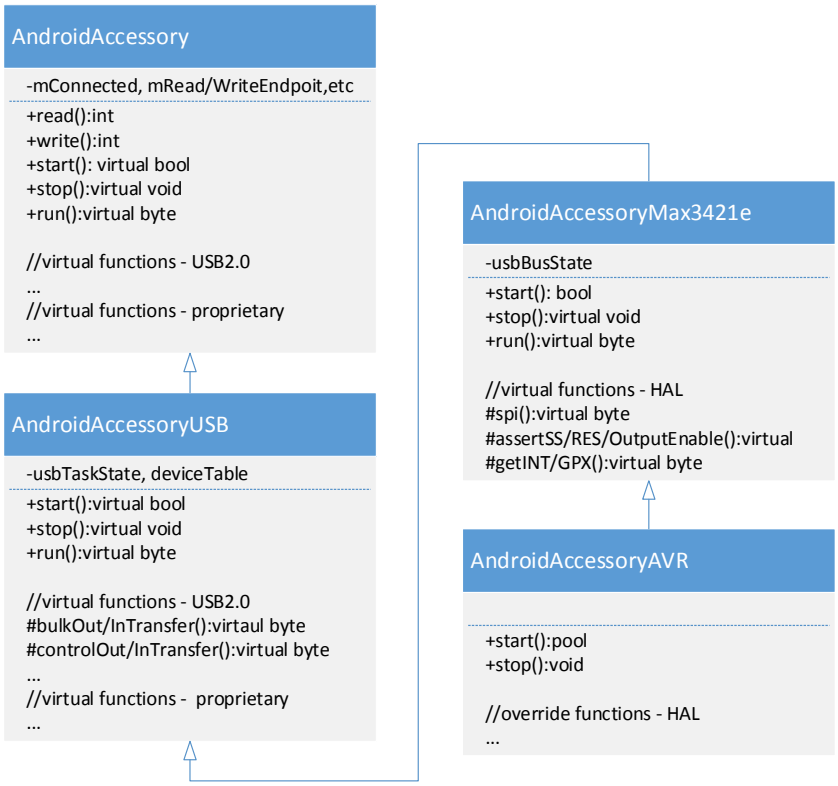


Figure 2: Class diagram of the Android accessory.

The AndroidAccessory Class

The base class, AndroidAccessory, defines the general behaviour of the accessory. It provides the users with five public functions (Figure 2). The function, run(), is the key function which implements the Android Open Accessory 1.0 [5].

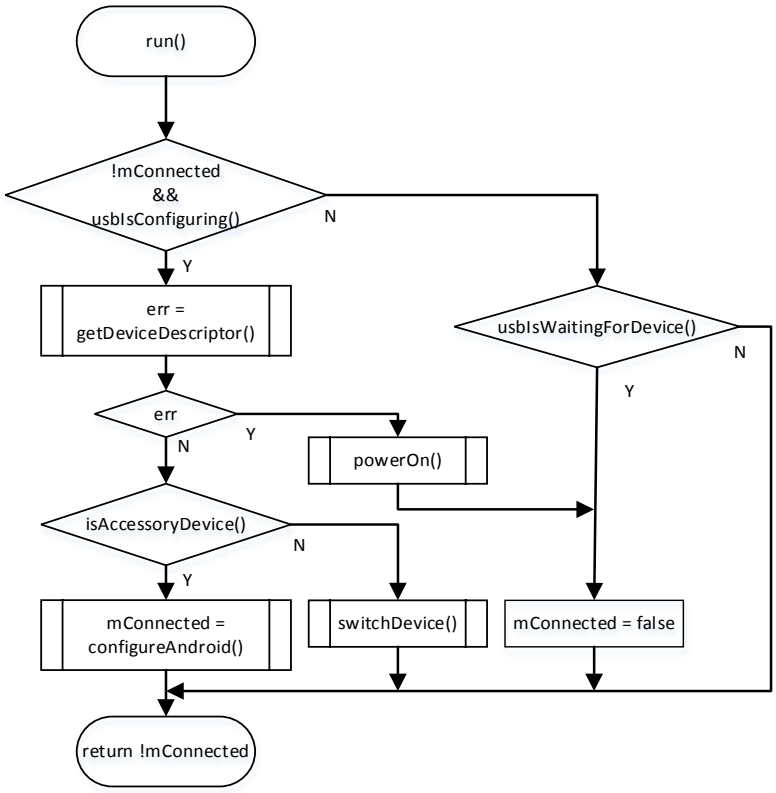


Figure 3: The flowchart of AndroidAccessory::run().

As illustrated by the flowchart in Figure 3, run() first calls a virtual function, usbIsConfiguring(), to detect the connected device. If the device is connected and the USB is in the configuring state, the accessory will check the vendor and product IDs of the device (isAndroidDevice()). If the vendor ID matches Google's ID (0x18D1) and the product ID is 0x2D00 or 0x2D01, the device is already in accessory mode. The accessory then calls configureAndroid() to establish communication with the device through the bulk transfer endpoints with its own communication protocol. If the vendor and product IDs do not correspond to an Android device in accessory mode (i.e. isAndroidDevice() returns false), the accessary will call switchDevice() to attempt to start in accessory mode. When the connected device is in accessory mode, the member variable mConnected is set to true and run() returns false, indicating that the accessory is not busy and ready for communications with the Android device. The mConnected will become false and run() returns true again if the device is disconnected, which is indicated by usbWaitForDevice().

The run() function must be called frequently, so that the connection of the device is monitored. When it returns false, the write()/read() function can be used to send/receive data to/from the Android device.

The run() function must be overridden by the subclass that provides the AndroidAccessory class with USB protocol supports. When overriding in the subclass, run() must call AndroidAccessory::run() when it returns.

The microcontroller that controls the accessory may go into sleep modes to save power. The stop() function is called before going to sleep and then start() is called after waking up. Both start() and stop() are defined as virtual functions that allow subclasses to take actions as needed.

The AndroidAccessoryUSB Class

The AndroidAccessory class sets up the platform for the accessory software. A subclass, AndroidAccessoryUSB, must be defined to provide USB protocol support.

From the implementer's point of view, the USB host consists of three logical layers: Client SW, USB System SW, and USB Host Controller [9]. The AndroidAccessoryUSB class is the combination of the Client SW and USB System SW layers. The USB 2.0 supports four basic types of data transfers: Control, Bulk Data, Interrupt Data and Isochronous Data Transfers. The Android Open Accessory protocol, however, only supports Control and Bulk Data transfers. This simplifies the design of the AndroidAccessoryUSB class.

The AndroidAccessoryUSB class overrides all virtual functions declared in AndroidAccessory. Some of these functions are USB 2.0 protocol functions, such as those that set up configuration using control transfers and reads/writes data using bulk data transfers; and others are proprietary for easy implementation such as usbIsConfiguring() that reports if the USB is in the CONFIGURING state.

The AndroidAccessoryUSB is also responsible for establishing communication with the device. This is illustrated by the state machine diagram in Figure 4.

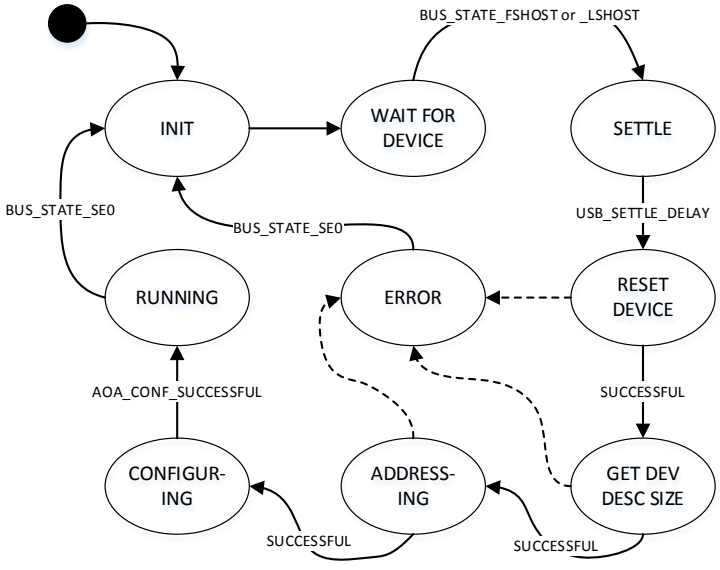


Figure 4: The state machine diagram of AndroidAccessoryUSB::run().

When started, the USB will go through the INIT state in which it initialises the internal data structures and then falls into the WAIT_FOR_DEVICE state. If a USB device is connected, the bus state will become BUS_STATE_FSHOST or BUS_STATE_LSHOST [9]. This triggers the transition into the SETTLE state. After a settle-delay, the USB will go

through the RESET_DEVICE, GET_DEV_DESC_SIZE, ADDRESSING and then CONFIGURING states if all are successful. In the CONFIGURING state, the AndroidAccessory class configures the device into accessory mode, triggering the USB into the RUNNING state. The USB remains in the RUNNING state until the device is disconnected, which changes the bus state into BUS_STATE_SE0 (single-end 0) that triggers the transition into the INIT state. In case there is an error, the USB will be transited to the ERROR state and only the disconnection of the device can recover the USB.

The AndroidAccessoryMax3421e Class

Both AndroidAccessory and AndroidAccessoryUSB classes are hardware independent. The latter can be extended into a subclass with any particular USB hardware support. The hardware can be a USB controller built in a microcontroller or a stand-alone chip such as Max3421e.

The AndroidAccessoryMax3421e class is specifically for the Max3421e. As discussed in the previous section, the internal SIE of the Max3421e automatically handles low-level USB protocol details. The SIE is accessed by writing or reading the register set of the Max3421e through the SPI interface. The typical operation to access a register is to write a command byte that specifies the register address and the direction bit, followed by subsequent bytes of data in the direction indicated by the direction bit [10]. The AndroidAccessoryMax3421e class will override those virtual functions declared in AndroidAccessoryUSB by using operations of accessing Max3421e's registers. These functions are: bulkIn/ OutTransfer() and controlIn/OutTransfer().

In the AndroidAccessoryMax3421e class, the run() function is overridden so that it monitors the USB activities reported by the Max3421e.

The AndroidAccessoryAVR Class

The AndroidAccessoryAVR class is the hardware abstract layer (HAL). It defines the hardware connections between the Max3421e and the microcontroller. It provides functions to assert chip reset to the Max3421e, initiate the SPI transactions, and retrieve interrupt's information.

This class also provides the entire class hierarchy with timing facilities, such as that to get system time in milliseconds.

A STUDENT PROJECT USING THE ANDROID ACCESSORY INTERFACE

MG7013 Embedded Systems is one of the final year courses in the CPIT Bachelor of Engineering Technology (BEngTech) programme. One of the assessments of the course is a project, in which students are expected to analyse, design and implement an embedded system, using various techniques, such as finite state machines in a multitasking system. The project brief is given to the student as follows:

> CoE Ltd has experience in developing embedded systems using AVR8 microcontrollers. Recently, they have developed an Android accessory interface between the AVR8 and an Android device.
>
> A client has approached CoE Ltd to develop a handheld calculator. Below is the requirement definition prepared by the client:
>
> *The calculator must perform addition, subtraction, multiplication and division of integers. It should allow users to key in a number sentence, such as "34+5*2=". The result should be displayed after the equal sign. The calculator should recognize that multiplication and division have higher precedence than addition and subtraction. The calculator must also display the current time and ambient temperature in real-time while it is awake. The calculator should go to sleep modes when it is in idle for a certain time and be woken up by any user input.*
>
> *The calculator must be able to connect to an Android phone or tablet through a USB cable. While it is connected, the ambient temperature is displayed in the Android device and the Android device can control the LEDs in the calculator.*
>
> You are a member of the development team at CoE Ltd. The development team has decided to choose an AVR8 as the microcontroller and use the AVR development kit and the Android accessory interface to develop the hardware prototype.

This project is more software oriented. With the Android accessory interface and other hardware modules available, students focus on modular design and implementation by applying theory and techniques introduced in the course, such as the cooperative scheduling of multitasking, the finite state machines for non-linear programming and the design structured diagram (DSD).

Involving emerging technologies, such as an Android Smartphone motivates students to put extra effort in learning, especially in developing the project. Their working prototype can work with their own Smartphone when they download an app from Google Play [11]. This application was developed in-house for the Android accessory interface to the CPIT AVR training kits. Figure 5 shows screenshots when the prototype is connected to the Android Smartphone.

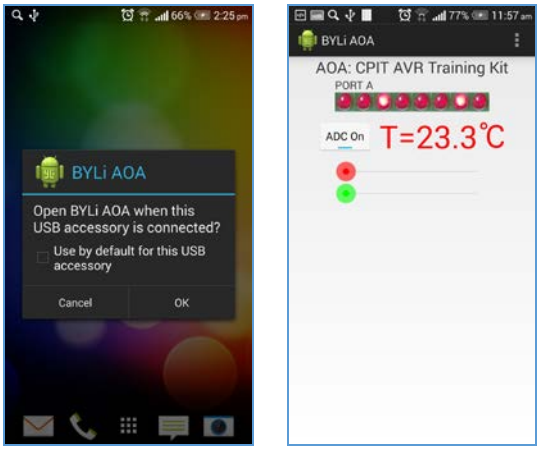


Figure 5: Screenshots of the demo application.

CONCLUSIONS

The design and implementation of the Android accessory interface has been presented in the article. The hardware design was based on the Max3421e USB controller. The software of the accessory was designed using the object-oriented concept and top-down approach. The general behaviour of the accessory is defined in the base class and the USB protocol support defined in its subclass. These are hardware independent and can easily be extended to a subclass with specific hardware support, such as the Max3421e or any microcontroller built-in USB controllers.

The Android accessory interface, together with the AVR microcontroller training kit, will be the platform for students to develop embedded systems, which can fully use powerful features of the Android device. The accessory interface has the feature of providing 500mA at 5V for charging power. Students can build their prototypes of embedded systems, capable of using an Android tablet as a permanent control panel when connected via the accessory interface.

REFERENCES

1. Android open accessory protocol, 22 August 2014, https://source.android.com/accessories/protocol.html
2. Li, Y., Implementing ZigBee protocol as assignments in teaching embedded systems. *Proc. Sixth IEEE Inter. Symposium on Electronic Design, Test and Applications*, Queenstown, New Zealand, 34-38 (2011).
3. Bright, M. and Li, Y., DDSWG: direct digital synthesis waveform generator. *Proc. Fifteenth Electronics New Zealand Conf. (ENZCon'08)*, Auckland, New Zealand, 7-12 (2008).
4. Android open source project, 22 August 2104, http://source.android.com
5. Android Open Source Project, Android open accessory protocol 1.0, 22 August 2014, https://source.android.com/accessories/aoa.html
6. Android Open Source Project, Android open accessory protocol 2.0, 22 August 2014, https://source.android.com/accessories/aoa2.html
7. Li, Y., Teaching embedded systems using a modular-approach micro-controller training kit. *World Transactions on Engng. and Technol. Educ.*, 6, **1**, 135-138 (2007).
8. Li, Y. and Benbow, C, An educational AVR microcontroller training kit. *Proc. Thirteenth Electronics New Zealand Conf. (ENZCon'06)*, Christchurch, New Zealand, 218-221 (2006).
9. Universal Serial Bus Specification, version 2.0 (2000), 22 August 2014, http://www.usb.org/developers/docs/usb20_docs
10. Maxim Integrated Products, Inc., MAX3421E USB peripheral/host controller with SPI interface (2013).
11. Google Play, 22 August 2014, https://play.google.com/store/apps/details?id=net.li_yao.android.aoa