Using QTouch on The AT32UC3C-EK as HID Media Buttons

Introduction

In the project, I will be demonstrating The AVR Software Framework by creating a USB device that uses the six built in capacitive touch buttons for media control. By the end of the project you should be able to use the touch buttons to play/pause, next/previous track, volume up/down and mute you computer.

Atmel's AT32UC3C-EK board has six capacitive touch buttons. The buttons are arranged in a layout that, conveniently, lends itself to being used for media control. These buttons are interfaced using the AT42QT1060 QTouch controller. This device is connected to the micro controller via a two wire interface (TWI). To interface these buttons to a computer, the AT32UC3C0512C's USB controller will be used as a HID device.

To tie everything together, the AVR Software Framework (ASF) will be used and the project will be developed in AVR Studio 5.

Setting Up The Project

To start the project, the AVR32UC3C-EK template was used. Using this template, the board drivers were added and a blank header file was created for the ASF modules as they are imported into the project.
Next, drivers from the ASF for

- GPIO
- Power Manager
- Sleep Manager
- Power Manager
- Clock Control
- Touch for AT42QT1060
- USB HID keyboard (single device)

were added to the project from the *Project > Select Drivers from ASF* menu. When you add a component from the ASF, the dependencies are automatically added to the project.

To prevent the USB and QTouch examples' clocks from clashing, the USB clock was set to run of PLL2.

Configuring The QTouch Controller

Configuring the AT42QT1060 QTouch library is a little tricky. The configuration ended up being copied from the QTouch example project. The QTouch initialisation functions and callback were copied from the example into a new C file named TouchHelper.c/.h, to be called from the main program. Nothing was modified from the initialisation, however the touch callback was modified to send the key press to the HID controller.

Configuring the USB Stack

When the USB HID Keyboard module was added to the project, the conf_usb.h header contains a collection of defines where callback functions are intended to be placed. Finding how these are configured is difficult. Initially, a keyboard example project was used to figure out the USB stack, however this example project was made from an XMEGA board. Luckily the ASF functions are fairly portable, however the GPIO interrupts are handled differently, and was the only modification that needed to be made in order to get the example project to run on the AVR32.

The next step was to reconfigure the HID USB descriptor. The media control buttons exist on the *Consumer Control* page of the HID descriptor. This means the the descriptor needed to be rewritten to act as media buttons. To save time, the HID descriptor macros from Dean Camera's LUFA library were used to generate the descriptors. Finally a function was added to the udi_hid_kbd.c that would send the new bit-field values to the host.

Bringing it Together

All that remains is to pass the QTouch data into the USB stack and have the HID device send the report on the keep alive signal. A function was added to TouchHelper.c serve the touch data and map it to the report values. To send the report on the keep alive of USB2.0, another function was added to ui.c to collect the data from TouchHelper and call the media send function of udi_hid_kbd.c.

Now the device should be fully functional as some auxiliary media buttons for your HID compatible USB host.

ASF Discussion

I am very open to the idea of a framework to provide a nice software API to the micro contoller peripherals. Especially for devices like the AVR32s, where just to get the GPIO to function, you have to use the Bus Matrix and the clocking domains. Writing these modules from scratch would take some time and effort to do, especially if you are new to the AVR32 architecture. However, the implementation of this framework is inconsistent and in some cases poorly executed.

I expected the AT42QT1060 module to be fairly easy to use. However, based on the QTouch example I worked off, the TWI initialisation had to be done manually. If I was writing the module, I would have put the location of the TWI and clock configuration in the board.h file, for easy porting to a custom board. Also, abstract away the setup of the TWI module, again the clock speeds (and I guess, setup) should be handled from the board.h file and the initialisation should be handled from the module itself. This would also remove the need for boards specific defines to be removed from the module.

The USB keyboard HID module seems to make a bit of an assumption that you will always be driven using events. For key presses there are only udi_hid_kbd_up and udi_hid_kbd_down functions. The report is kept in static memory and now way to drive it directly form buttons in a polled manner. However it is fairly simple to add one, as I did.

The HID keyboard module seems to be convinced that it will be a keyboard, and that is final. The descriptor was stored as a fixed sized array, inside a structure (which had no other elements). Again this is easily overcome, but keyboards and mice arent the only HID devices developers are going to want to make. What if they need a Nine Iron?

I'm also going to take this opportunity to mention something in the GPIO module that bugged me. There is no function to write a value to a GPIO output. The are only set and clear functions. I might have missed it, bit i would like to see something like gpio_write_pin(uint32_t pin, uint8_value). While you could write a function that combines the set and clear with an if statement, to me, it does not feel quite right.

All considered, the ASF will be a great tool for developer to rapidly develop applications. The problem is that inconsistent and some untidy methods will lead end developers creating Frankencode. They will mash parts together and hope it ~~becomes alive~~ works.