Atmel AVR Design Contest 2006

ATir Keyboard Interface

Introduction

The ATir is an interface device to emulate the function of a PC "AT" keyboard and controlled with an infarred (IR) remote control using an "AT"tiny45 microcontroller. This ATir connects in series with an existing keyboard, or if the device is already programmed, the ATir can be used without a keyboard present as shown in figure 1.

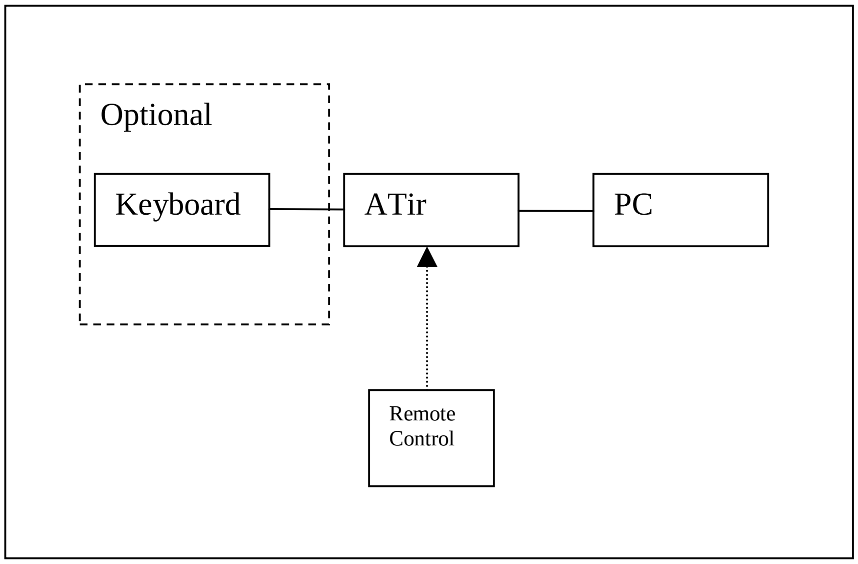

Figure 1 System Block Diagram

The actual device with the remote used for the project is shown in Figure 2 and an internal view is shown in Figure 3.

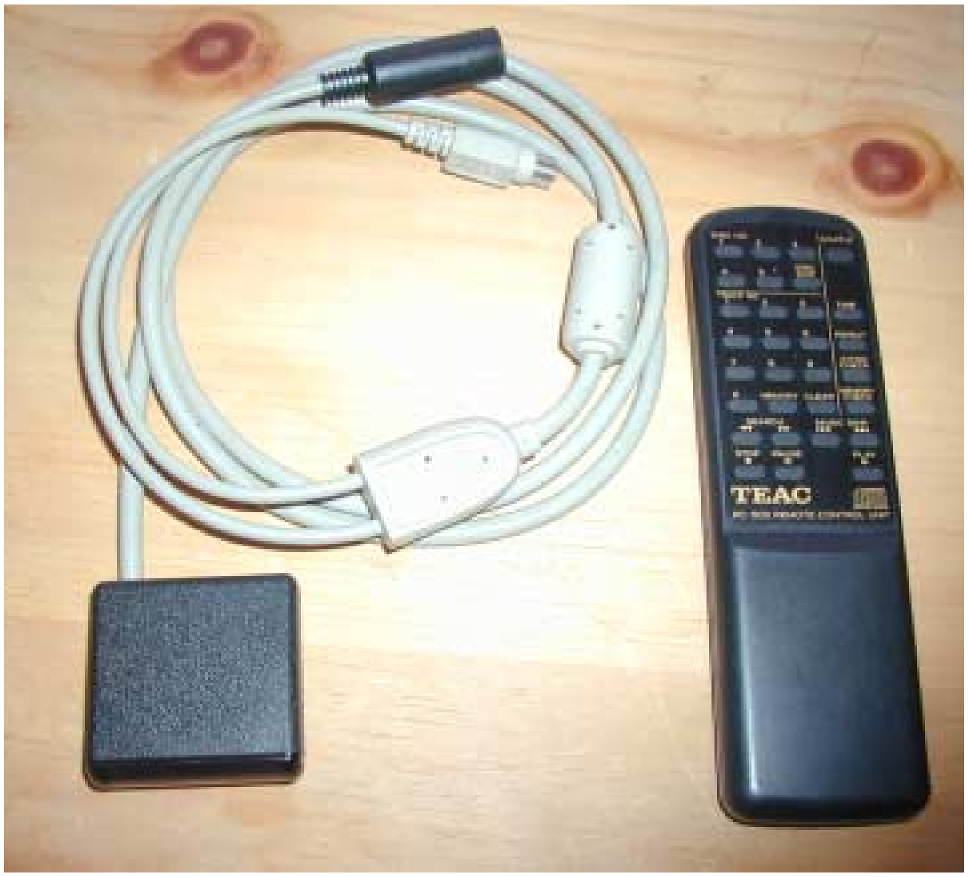

Figure 2 ATir assembled view with remote

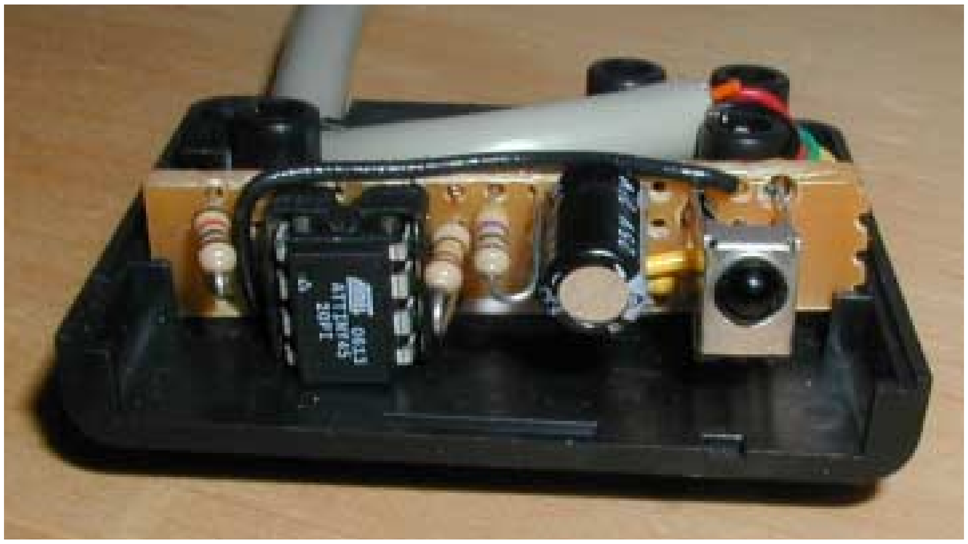

Figure 3 ATir internal view

Key sequences up to 64 keys can be stored in an individual macro, and these macros are store in the EEPROM section of the microcontroller, so the data is retained after power is removed. After macros are defined, the ATir can be connected to a PC without a keyboard, and the ATir will perform the necessary handshaking during boot up to allow operation. A keyboard can be connected to the ATir after the system boots, and the ATir will relay information between the keyboard and PC.

ATir Description

The device hardware is composed of an Atmel™, ATiny45 microcontroller, a Sharp™ infrared receiver / demodulator and a few discreet components consisting of low pass filter components and pull up resistors. Refer to figure 6 for the schematic diagram of the ATir. The ATtiny45 microcontroller IC1 is an 8 pin device that contains 4K bytes of flash program memory, 256 bytes of SRAM and 256 bytes of EEPROM. The microcontroller also contains 2, 8 bit timer/counters, Universal serial interface and a 10 bit successive approximation analog to digital converter.

The keyboard clock line is connected to the external interrupt pin, and the interrupt is configured for falling edge detection. The output of the IR detector IC2 is sensed using the pin change interrupt with a mask configured to sense only the IR detector output. To save space and components, the internal pull up feature is used instead of external pull up resistors for the required PC clock and data lines. A high resistance pull up R5 is used on the keyboard clock line to allow operation with a keyboard disconnected while being able to sense and operate a keyboard when connected. Resistor R4 and LED1 were used during development and gives a visual indication whenever the ATir is transmitting data to the PC. This occurs when a key is pressed on an attached keyboard or when an IR command that is linked to a valid keyboard macro is received. Indicator connection is possible because the ATir always generates the clock signal and the tiny45 has no trouble driving LED's directly from an IO pin.

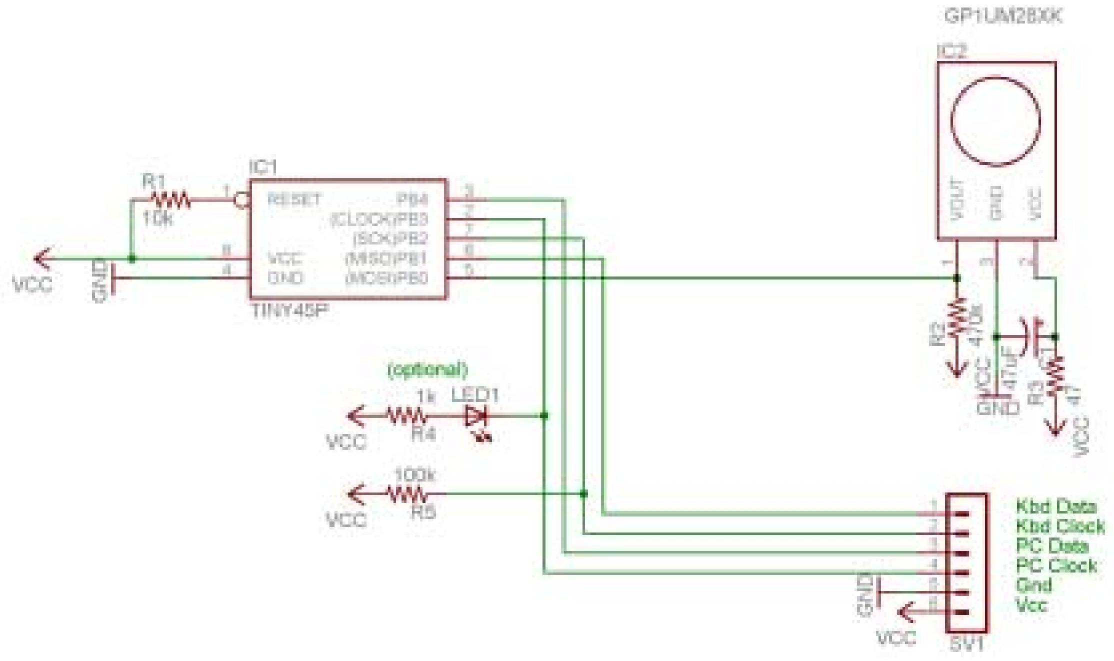

Figure 6 ATir Schematic Diagram

The internal, calibrated 8MhZ clock was used as a clock source for the ATir. This source proved to be surprising accurate, as the timing requirements for the AT keyboard data communication and IR pulse length timing require a consistent and accurate system clock.

Most of the digital resources of the ATtiny45 are used in this project. Timer 0 is used to time the pulses from the IR receiver. Timer 1 is used to implement a timeout condition during transmit and receive to ensure that synchronous communications remain synchronized. The EEPROM section is used to store IR commands and the associated macro data. Program memory utilization is a few bytes short of tiny45's 4K capacity. All of the IO pins are utilized. The reset pin has the capability of being reconfigured as general purpose IO, but this feature was not implemented because reprogramming of the chip would then require a high voltage programmer vs the inexpensive parallel port programmer that was used.

The microcontroller source code was developed in C using AVR GCC in AVR Studio 4 version 4.12 development environment. As a hobbyist with a budget, I thoroughly enjoyed working with the FREE AVR GCC compiler that integrates nicely with AVR Studio. Initially code was written and tested using the built in simulation tools in AVR Studio for a 40 pin ATmega32 as the target. Next the application was programmed into the ATmega for in circuit debugging. After the application was running on the ATmega, some minor architectural changes allowed the application to be ported on the ATtiny using the same complier and development environment, very nice.

The application places some real time signal processing demand on the microcontroller. Reception of data from the keyboard must be processed and retransmitted to the PC. Keyboard data is processed by a filtering and compression algorithm when recording macros to conserve EEPROM memory. A key press and release sequence causes the keyboard to transmit 3 or 5 scan code frames comprised of 11 bits/frame. This sequence is represented by storing 1 byte in EEPROM. Also as most everyone has experienced, if a key character key is held down in an application such as a word processor, the character repeats until released. This is because the keyboard resends the key down scancode at a configurable periodic rate, referred to as the typematic rate. This typematic data is filtered and therefore does not contribute to storage requirements. The compression algorithm is shown in listing 1.

```
void encode(uint8_t scancode){
static bool ext = false;            //extended key flag
static bool rel = false;            //key release flag
static bool lwc = false;            //last (input) was a code byte (not ext or rel)

static uint8_t lastcode = 0;


        switch (scancode){
        case 0xE0:
                if (lastcode && !ext){
                        addKeyDown(lastcode);
                        lastcode = 0;
                }
                ext = true;
                lwc = false;
```

```
		break;
	case 0xF0:
		rel = true;
		lwc = false;
		break;
	default:
		if (scancode==lastcode){
			if(rel){				//if release, its a keypress
				rel=false;
				if(ext){
					ext = false;
					addKeyPressExt(lastcode);
				}
				else{
					addKeyPress(lastcode);
				}
				lastcode = 0;
			}
			else{
				if(ext && lwc){
					addKeyDownExt(lastcode);
					ext = false;

				}
			}
		}
		else{//scancode != lastcode
			if(lastcode){	//a key was pressed pending
				if(rel){
					rel=false;
					if(ext){
						ext = false;
						addKeyDownExt(lastcode);
						addKeyReleaseExt(scancode);
					}
					else{
						addKeyDown(lastcode);
						addKeyRelease(scancode);
					}
					lastcode=0;
				}
				else{
					if(ext){
						addKeyDownExt(lastcode);

						if(lwc)
							ext = false;
					}
					else{
						addKeyDown(lastcode);
					}
				lastcode=scancode;
				}


			}
			else{	//key not pressed pending
				if(rel){
					rel = false;
					if(ext){
						ext = false;
						addKeyReleaseExt(scancode);
					}
					else{
						addKeyRelease(scancode);
					}
				lastcode=0;

				}
				else{
```

```
                                        lastcode=scancode;
                                        }
                                }
                        }
                lwc=true;
                }//end switch

        }
```

Listing 1 compression and storage algorithm

Conclusion

The Tiny AVR IR keyboard interface offers a convenient cross platform solution to interface an IR remote control to type keyboard macros to a PC. The interface plugs into a PS2 keyboard port on the PC and accepts commands from an infrared remote. The device can be configured to operate with common remote controls with little or no change in code. For those of us with laptops or new computers without PS2 ports, do not despair. USB to PS2 adapters are inexpensive and allow the ATir to also be used within USB environments. Other remote control for PC's are available, but the ones I have seen require the use of the remote control supplied, and drivers to be installed. With ATir, any operating system that recognizes the hardware will work. The ATir device has few components, and the system as constructed is about $10.