ECE 477 Lab 6

Christian Knight, Nikko Noble

September 16, 2020

1 Introduction

The purpose of this lab is to use the AT90USB162 microcontroller to measure signals from an IR remote and send them over serial to a PC using the LUFA library. Specifically, the goal is to have the AVR send the lengths of received IR bursts and gaps over serial in order to have the PC write valid Pronto code, which is the hexadecimal representation of the transmitted IR signal, so that button presses can be interpreted.

For Part B, a C program was desired that configures the AVR as a human interface device (HID) rather than an serial device so that an IR remote control can be used to register mouse and keyboard presses on a PC with the AVR connected as an IR receiver.

2 Part A

This section describes the development of the C program that configures the AT90USB162 microcontroller to detect IR signals and send over serial the length of the measured IR bursts and gaps using the virtual serial functions in the LUFA library and timers on the AVR. It also describes the test of a C program for the PC that interprets the IR burst and gap lengths and writes Pronto code, which is how IR remote codes are learned and recognized.

2.1 Hardware

The hardware schematic for this lab is shown in Figure 1.

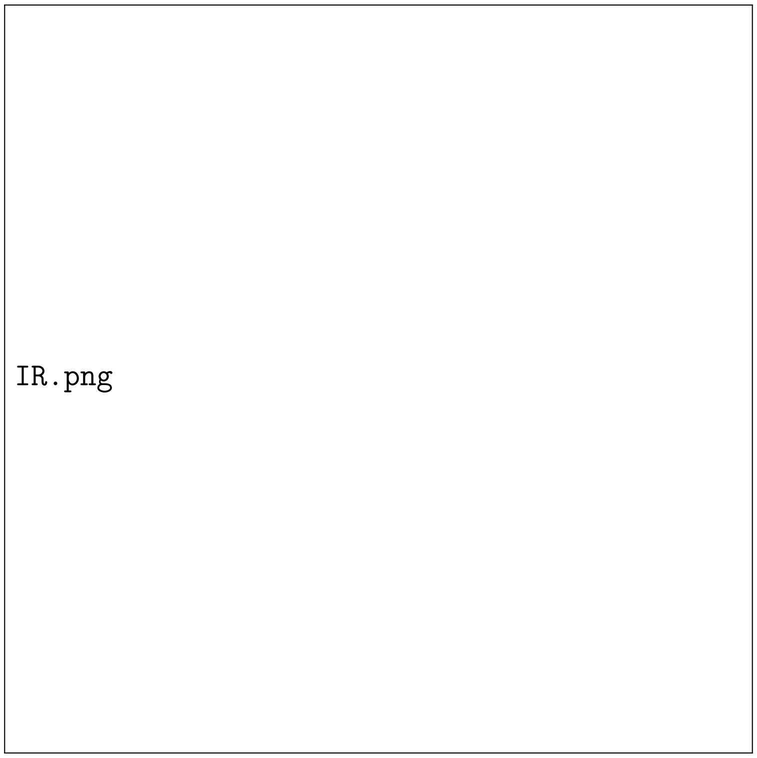

Figure 1: Hardware schematic for Lab 6

As shown in Figure 1, the main component needed to detect IR signals is a phototransistor, indicated by Q2. When no IR light is present, the emitter current of Q2 is zero, and it is nonzero but small when an IR burst is present (in the μA to low mA range). The emitter of Q2 feeds into the base of Q1 to amplify its emitter current when IR light is present, which increases the chance of an IR burst corresponding to a logic low input as desired (PC7 pulled to GND since Q2 and Q1 are on).

In addition to the hardware shown, the AVR board also has two LEDs connected to pins PD4 and PD6 which are used to echo the presence of IR bursts and gaps. Also, the AVR is connected to a PC over USB to enable serial communication.

2.2 Software

The program `partA.c` starts by performing the necessary initializations for serial over USB as shown in the 'VirtualSerial' demo [1]. Then, the timer and input/output pins of the AVR are configured using the `timer_init()` function. Within this function, timer 1 is enabled, pin PC7 is set as an input for the phototransistor receiving the IR signal, and pins PD4 and PD6 are set as outputs for the LEDs on the board. Pins are configured as inputs or outputs by setting up the data direction registers DDRC and DDRD, and timer 1 is set up in the timer/counter control register TCCR1B[2]. To set up the DDRC and DDRD registers, the bits corresponding to the desired outputs are set, and the bits corresponding to the inputs are cleared. Also, timer 1 is set up by setting the TCCR1B register equal to 1. This sets the prescaler to 1, enabling the clock source, and sets the capture mode of pin PC7 to the falling edge, which occurs in the event an IR burst is present. The following statements set up the timer and input/output pins:

```
TCCR1B = 1; // set prescaler to 1 and capture falling edge
DDRC &= ~(1<<PC7);  // set pin PC7 to input (for phototransistor)
DDRD |= (1<<PD4) | (1<<PD6); // set pins PD4 and PD6 to output (for board LEDs)
```

After the AVR has been initialized for serial and configured for measuring pulses on the PC7 pin, the program waits for a handshake between the board and the PC over serial, and then moves into the main loop, which runs as long as the device is powered. Within this loop, the `echo_LED()` function sets the LED on PD6 and clears the LED on PD4 if an IR pulse is detected, and does the opposite if no pulse is present. This helps to confirm visually that the AVR is receiving an IR signal when testing with an IR remote.

The program then measures the length of the first IR burst by waiting for the `measure_dark()` function to return true, indicating a dark gap of less than 4ms has occurred (LEDs echo the IR signal in the meantime), and then executing the `measure_burst()` function, which keeps track of the number of ticks that pass while an IR burst is present.

Both the `measure_dark()` and `measure_burst()` functions work by getting the timer value at the beginning from the TCNT1 register to save the current time, waiting for either the pulse or gap to end, and getting the timer value again at the end from TCNT1. The difference between the beginning timer value and end timer value give the number of timer ticks that elapsed during the event, and can be converted back in to time by dividing by the timer speed (8MHz).

2.3 Programming and testing

To compile the AVR program `partA.c`, a makefile is used which specifies the AVR processor and links to the necessary LUFA library files. Once the `make` command is run, a `.hex` file is generated which needs to be programmed to the microcontroller. The microcontroller is programmed using the following commands:

```
make clean  // gets rid of old program files
make    // compiles new program
dfu-programmer at90usb162 erase --force // erases AVR
dfu-programmer at90usb162 flash partA.hex   // flashes AVR
```

Now the AVR will appear as a USB serial device. To communicate with the AVR, the serial terminal program needs to be compiled and run on the PC. The program is compiled using `gcc` and is run with the name of the serial port to be used as an argument. In the case of this lab, the AVR shows up on serial port ACM0, so the serial terminal is compiled and run using the following commands:

```
gcc serial_term.c o serial_term
./serial_term /dev/ttyACM0
```

From here, the terminal should be able to print lines of pronto code from the IR burst and gap lengths detected by the AVR. To test the code, an IR remote control for a Panasonic TV was used, measuring the IR signal received when the power button is pressed and held versus the volume up button. As mentioned earlier, the output of the phototransistor circuit is +5V when no IR is present due to the transistors being off and the output is 0V when IR is present. Figure 2 shows both the IR signal received due to the power button being pressed and held.

pwr.jpg

Figure 2: IR signal received and echoed for PWR button

As shown in Figure 2, the initial button press occurs at about 100ms and repeats about every 120ms, and the AVR does not echo at the actual IR carrier frequency for the first three bursts. Figure 5 shows the length of a single burst and gap from the power button.

pwrburst.jpg

Figure 3: IR burst received for PWR button

As shown in Figure 3, the length of an IR burst due to the power button being pressed is about 62ms. The length of the gap is about 72ms. Another interesting note is the carrier frequency of the IR signal from the Panasonic remote seemed to be about 1kHz, which is much less than the expected 38kHz that was mentioned in class (regrettably, other remotes were not tested to contrast the results). Figure 4 shows the same plots, but with the volume up button being pressed and held instead of the power button on the same remote.

volup.jpg

Figure 4: IR signal received and echoed for VOL UP button

As shown in Figure 4, the initial button press occurs at about 100ms and repeats about every 120ms, much like the power button. Figure 5 shows the length of a single burst and gap from the volume up button.

Figure 5: IR burst received for VOL UP button

As shown in Figure 5, the length of an IR burst due to the volume up button being pressed is about 55ms (roughly 7ms less than the power button). The gap stays the same, at about 73ms. This difference in burst length distinguishes the two different buttons.

Unfortunately, there was not much success printing the lengths of bursts and gaps over serial. This was only observed one time; the AVR was able to read back the number of ticks passed for each burst and gap. Every other time, the incoming IR signal was still echoed on the board's LEDs, but the AVR prints nothing over serial. It wasn't clear what resulted in the successful communication over serial that one time, but there seems to be a limitation on the AVR's end (possibly not enough RAM).

3 Part B

Given that Part A was not completely successful, it was tough to build upon it for Part B. However, one really cool idea was to use the LUFA library to set up the AVR as a generic human interface device (HID) to allow for recieved IR signals from a remote control to register key presses on a computer. This would involve substituting the LUFA virtual serial code for the LUFA HID code and sending button presses based on the IR signal rather than ASCII strings.

4 Conclusion

The development of a C program that configures the AT90USB162 microcontroller to measure signals from an IR remote and send them over serial to a PC has been described. The goal was to have the AVR send the lengths of received IR bursts and gaps over serial in order to have the PC write valid Pronto code, which can then be interpreted as a command.

For Part B, a C program was desired that configures the AVR as a human interface device (HID) rather than an serial device so that an IR remote control can be used to register mouse and keyboard presses on a PC with the AVR connected as an IR receiver. Since Part A was unsuccessful as a whole, Part B couldn't be tackled, but would just require putting the LUFA HID code in place of the virtual serial code.

A Part A Source Code

Attached: `partA.c`, `serial_term.c`, `makefile`

Sources

[1] Dean Camera. *LUFA Library*. Mar. 31, 2017. URL: http://www.fourwalledcubicle.com/LUFA.php.

[2] Atmel Corporation. *AT90USB162 Datasheet*. Mar. 3, 2017. URL: http://www.atmel.com/Images/doc7707.pdf.

[3] Wikibooks.org. *Serial Programming/termios*. Apr. 13, 2016. URL: https://en.wikibooks.org/wiki/Serial_Programming/termios.

[4] Barry Gordon. *RC: The Pronto's IR Code Format*. Apr. 28, 2016. URL: http://www.remotecentral.com/features/irdisp1.htm.