

ECP3 Final Project

Kevin Wu

June 11, 2008

Contents

- 1 Overview 1
- 2 Hardware 1
 - 2.1 The Control System 1
 - 2.2 Simplified Model of the Control System 4
- 3 Software Program for Control System Model 5
- 4 Additional Project Info 8

1 Overview

For the final project, we model a control system that dispenses tablets into bottles on a conveyor belt. The AtTiny44, via a sensor that detects a laser beam passing through the falling tablets, counts the number of tablets that are dispensed into a bottle, and triggers an interrupt when the desired number of tablets has been dispensed. At that point, the dispensing stops while the conveyor belt moves the next bottle into place. The process then repeats. See [Figure 1](#).

2 Hardware

2.1 The Control System

The following diagram illustrates the hardware layout of the tablet-dispensing control system, upon which we base our model (see [subsection 2.2](#)).

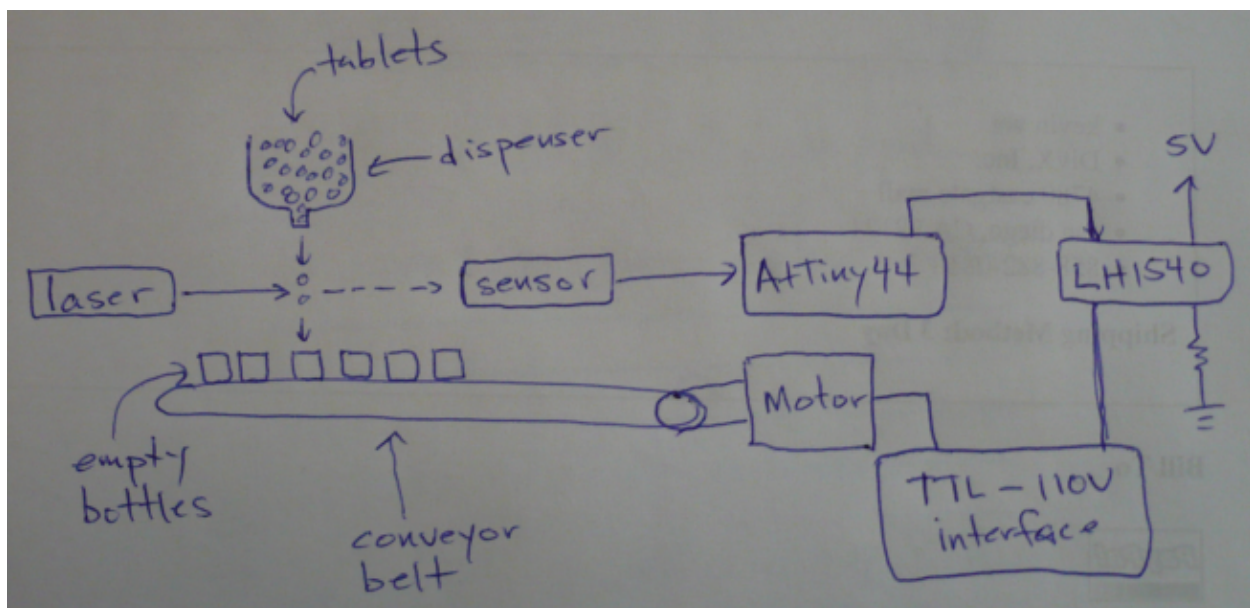


Figure 1: Tablet Dispensing Control System Diagram

The circuit diagram for the “TTL-to-110V interface” circuit is shown in [Figure 2](#) on page 2.

2.1.1 How It Works

A laser shines a beam through the path of the falling tablets, and strikes a sensor that is connected to the external clock pin of the AtTiny44. When no tablets are falling, the laser shines unobstructed onto the sensor, producing a logic-high level. When a tablet passes through the laser beam, momentarily obstructing the beam from the sensor, the sensor produces a logic-low level. In effect, as tablets fall through, a series of logic high and low levels is produced by the sensor. These pulses then clock a counter on the AtTiny44, so that as each tablet falls through the laser beam, the counter is incremented.

When the counter reaches a predefined number (in our case, 8), an interrupt is triggered. In servicing the interrupt, the AtTiny44 “turns on” an optocoupler that, in turn, turns on a conveyor belt motor via a TTL-to-110V interface circuit (see [Figure 2](#) on page 2). The AtTiny44 keeps the optocoupler on long enough so that the conveyor belt can move the next bottle into place, aligned under the dispenser.

Once the interrupt service routine is finished, program control is returned to the dispensing function, and the process repeats.

2.1.2 State Diagram

The basic operation of the control system can be described in the following state diagram.

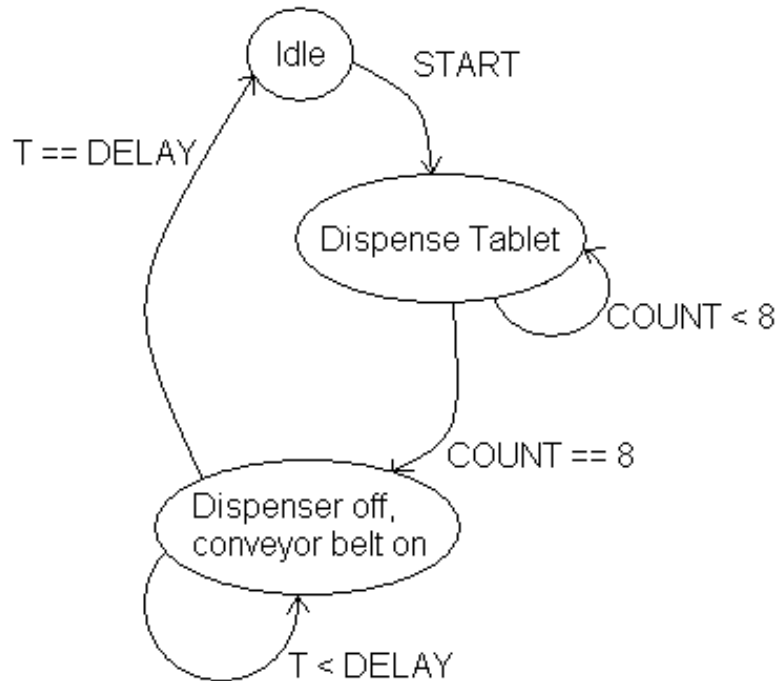


Figure 3: Control System State Diagram

2.2 Simplified Model of the Control System

Since we do not have access to the hardware required for the full control system, we create a model that represents the system, shown below. We also write a program to run the model system (see [section 3](#)).

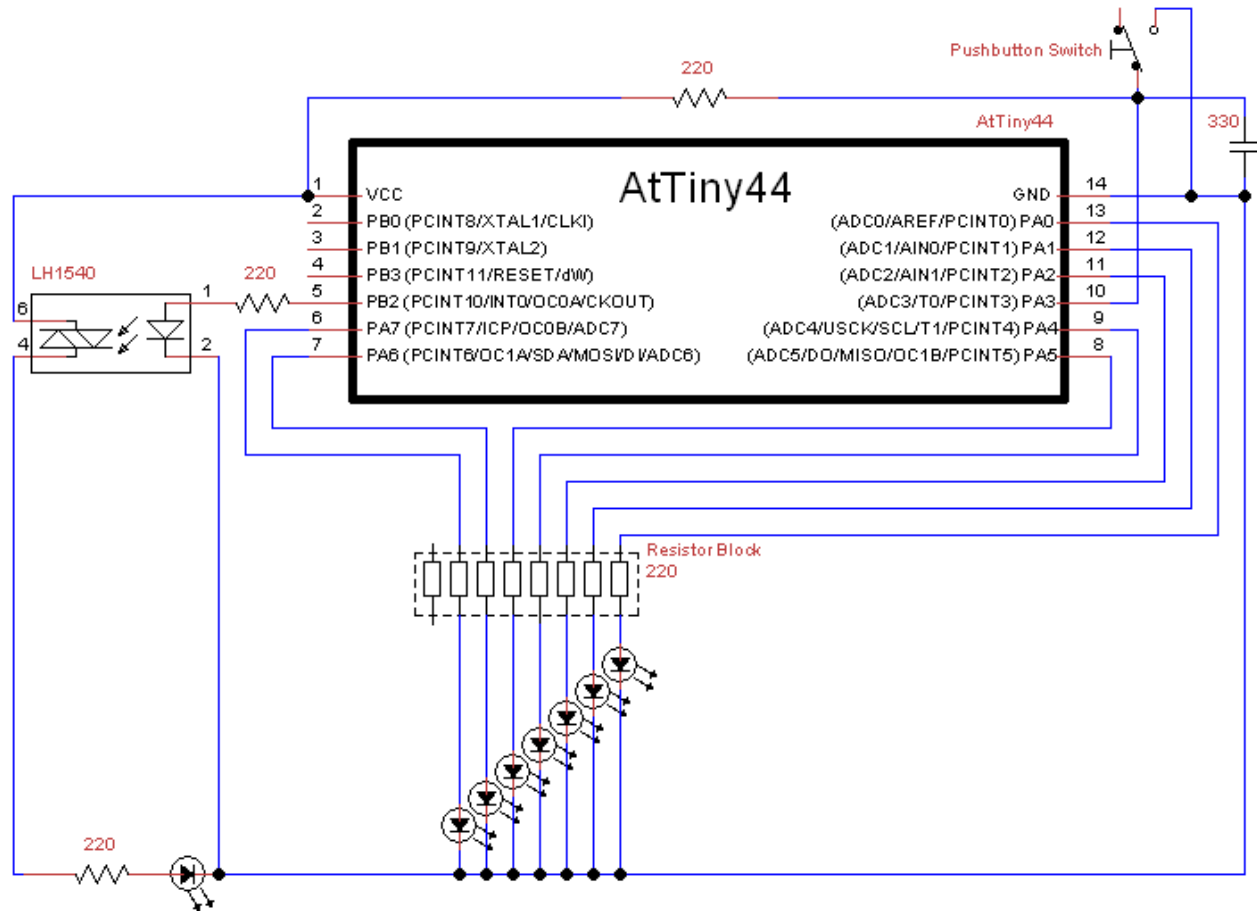


Figure 4: Circuit Diagram of Simplified Model

2.2.1 Differences Between Simplified Model and Control System

- Instead of the dispenser and laser, we use a push button switch. Pressing the push button signifies that a tablet has been dispensed and has passed through the laser beam. The push button switch is connected to the external clock input pin T0 (or PA3) of the AtTiny44. A pull-up resistor is also connected to T0. Closing the switch pulls pin T0 to ground. Therefore, pressing the push button causes a falling edge, and releasing the push button causes a rising edge.
- Instead of the TTL-to-110V interface and conveyor belt, we use an LED. The lighting of the LED signifies that the conveyor belt motor is running.

3 Software Program for Control System Model

The following is the program that controls the model for the tablet dispensing system. Explanation of code and operation is given in the comments.

```

#include <avr/io.h>                /* attiny register names */
#include <avr/interrupt.h>         /* for interrupts */
#include <util/delay.h>           /* for _delay_loop_2() */

#define DELAYCOUNT 8000         /* number of delay cycles */
#define DELAYMULT 150           /* multiplier for delay */
#define TABLETS 7              /* number of tablets/bottle */

void getdelay(unsigned char);     /* getdelay() prototype */

/*
 * main() accepts no arguments and returns int. See the comment blocks below
 * for an explanation of its purpose and function.
 */
int main(void)
{
    /*
     * We want to count a certain number of events, and then trigger something
     * to occur when that number has been reached. Therefore we need a
     * comparison between a counter and some value, and an interrupt to occur
     * when there is a match.
     *
     * To enable such an interrupt, we set the OCIEOA bit in the TIMSK0
     * register, and enable global interrupts by setting the I-bit in SREG.
     * To specify the comparison value, we assign it to OCROA.
     *
     * As the counter counts, its value is compared with the value in OCROA.
     * If a match occurs, the OCFOA bit is set in the TIFRO register. When
     * that bit is set, along with the I-bit in SREG and the OCIEOA bit in
     * TIMSK0, the Timer/Counter0 Compare Match Interrupt is executed, and is
     * serviced by TIM0_COMPA_vect() (see below).
     */
    SREG |= (1 << 7);             /* enable global interrupts */
    /* enable "compare match A" interrupt, if global interrupts are enabled */
    TIMSK0 |= (1 << OCIEOA);
    OCROA = (unsigned char)TABLETS; /* compare counter to this */

    /*
     * Now we set up the actual counter. In our case, we want to count the
     * number of times an external event occurs, namely, how many times a
     * sensor is tripped by falling tablets. Therefore, we use an external
     * clock source for the counter, where the external clock source is the
     * output of the sensor. When the sensor is tripped, a pulse is produced
     * at its output, and those pulses clock the counter.
     *
     * To setup the counter's clock source to be external, we set up the CS02:0
     * bits in the TCCR0B register to be 111. The clock will be read from pin
     * T0 (PA3), and will clock on rising edge.
     */
}

```

```

* Our circuit is set up such that pressing a push button signifies the
* tripping of the sensor. The push button is connected to PA3. PA3 is
* set high with a pull-up resistor, and is shorted to ground when the push
* button is pressed. Therefore, the timer will clock when the push button
* is released.
*/

/*
* select clock source as external on PA3, clock on rising edge; THIS ALSO
* TURNS THE COUNTER ON
*/
TCCROB |= (1 << CS02) | (1 << CS01) | (1 << CS00);
/* set PA3 to be an input pin, since it is being used as clock input */
DDRA = 0b11110111;

/*
* One last thing to set up is the counter mode. We want the counter to
* only count up to our desired value, and then start over from 0 again,
* instead of continuing to increment even after it has reached our
* desired value. In other words, when there is a compare match, clear the
* counter.
*
* To do this, we set the counter to mode 2, which is the CTC mode: "clear
* timer on compare match". This is done by setting the WGM bits in both
* the TCCROA and TCCROB registers. WGM02 is in TCCROB, and WGM01:0 are in
* TCCROA. Setting WGM02:0 to 010 enables CTC mode.
*/
TCCROB |= (0 << WGM02);
TCCROA |= (1 << WGM01) | (0 << WGM00);

/*
* We use PB2 to light an LED to signify conveyor belt movement. We
* therefore set PB2 as output.
*/
DDRB = 0b00000100;
SP = RAMEND;                                /* set SP to initial state */

/*
* In its normal "idling" state, the program performs the following
* infinite loop, which consists of simply turning on PORTA. This will
* light the bank of LEDs.
*
* When an interrupt occurs, the program jumps to the ISR. After servicing
* the interrupt, it returns to this while loop, and awaits another
* interrupt.
*/
while(1)                                     /* repeat forever */
{
    SREG |= (1 << 7);                         /* enable interrupts again */
    PORTA = 0xF7;                             /* turn on PORTA except PA3 */
    SP = RAMEND;                              /* set SP to initial state */
}

```

```

    return(0);                                /* exit */
}

/*
 * TIMO_COMPA_vect() is the ISR for the Timer/Counter0 Compare Match Interrupt.
 * It accepts no parameters, and returns void.
 */
void TIMO_COMPA_vect(void)
{
    /* clear all bits, including the CS2:0 bits, which stops timer/counter */
    TCCR0B = 0;

    /*
     * To show that we are inside the ISR, we turn off the LED bank by setting
     * PORTA to 0, and turning on PB2 for a short while to signify that the
     * conveyor belt is moving.
     */
    PORTA = 0;                                /* turn off PORTA */
    PORTB = (1 << 2);                          /* turn on PB2... */
    getdelay(DELAYMULT);                       /* ... for this amt of time */
    PORTB = 0;                                /* turn off PORTB */

    /*
     * Now that the conveyor belt has moved, putting the next bottle in place,
     * we repeat the tablet-dispensing and counting process. We turn on the
     * counter and enable interrupts.
     */
    /* turn on timer0 again by setting clock to external source, T0 */
    TCCR0B |= (0 << WGM02) | (1 << CS02) | (1 << CS01) | (1 << CS00);

    /* reenable interrupts */
    SREG |= (1 << 7);
    TIMSK0 |= (1 << OCIE0A);
}

/*
 * getdelay() essentially multiplies the delay, produced by _delay_loop_2(),
 * by a particular amount. It accepts an unsigned char argument, which is the
 * multiplier value, and returns void.
 */
void getdelay(unsigned char j)
{
    while (j > 0)                              /* while j greater than 0... */
    {
        _delay_loop_2(DELAYCOUNT);           /* do delay */
        j--;                                   /* decrement j */
    }
}

```

4 Additional Project Info

- Circuit diagrams drawn in TinyCad (tinycad.sf.net).
- State diagram drawn in MS Paint.
- Code written in AVR Studio 4 v4.13 SP2 (atmel.com) and compiled with WinAVR 2007-12-21 (winavr.sf.net).
- Document written in TeXnicCenter (toolscenter.org) using $\text{\LaTeX} 2_{\epsilon}$.
- For a copy of this document and a video of the model circuit in action, visit wuziq.com/ecp3.