**ELEC3042 Minor Project Report**

Name: Tyler Johnson

MQID: 46978518

# Overview

In designing the stopwatch, I decided to handle the inputs in the ISR as it was important to the operation of the stopwatch that the inputs be precise. The main state machine handles switching between the three features of the stopwatch, paused, counting, and initial. The main loop is responsible for updating the digits[] array, which is sent to the 7 Segment Display by the sendData() function.

Figure 1 shows the system block diagram and the signals that flow between them. Diagram

Description automatically generated

Figure : System Block Diagram

# System Resources

To keep time in the system I used timer0 in CTC mode to keep time. Timer0 generates an interrupt every millisecond, which increments two variables, clock\_count and debounce timer. This saved me having to use two timers to keep track of time for debouncing and for the clock. Changes on the input pins for S1-A1 and S2-A2 (PC1, PC2) generate an interrupt that simultaneously handles debouncing and setting flags that are read by the state machine (button\_flags).



# PCINT1 Interrupt

PCINT1 is responsible for handling the inputs and interrupting the system to ensure they are read as quickly as possible. The buttons are debounced within the ISR to ensure inputs are only read once. A separate variable, debounce timer, is used to keep time as it makes it much easier to manipulate two separate variables instead of managing the same one.



# Main State Machine Diagram

Diagram

Description automatically generated

Figure : State Machine Diagram

## Truth Table

Graphical user interface, application, table, Excel

Description automatically generated

Table : State Machine truth table

## State Machine

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

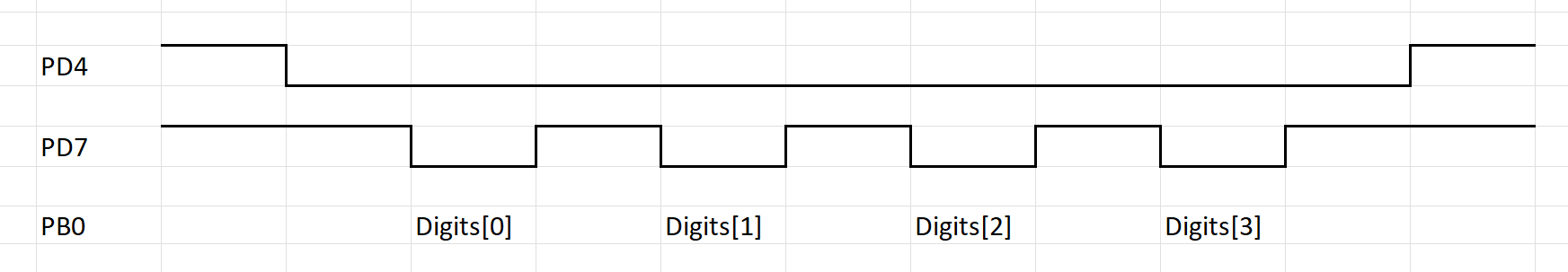
Description automatically generatedThe stopwatch is required to have three different functions. An initial state (stateInitial) where the count is zero and the timer isn’t incrementing. A counting state (stateCount) which counts upwards from the time initially displayed when S2-A2 is pressed. And a paused state (statePaused) which pauses the timer and turns on LED4. It is also required to resume the count from wherever it was paused. The code for the State Machine is implemented in the main loop.

# sendData()

sendData() sends the given the segments from segmentMap() and the digits from digits[], sends the requisite signals to the 74HC595 8-bit shift register and latches them to the display.



The code above follows the general signal graph detailed below.



New data is sent from PB0 to the shift register on the rising edge of PD7. Once all the data is sent PD4 is set to high which latches all the values of

# Full Code

/\*

 \* File: MinorProject.c

 \* Author: Tyler Johnson

 \*

 \* Created on 19th March 2022, 4:55 PM

 \*/

//==============Header files==============

#include <xc.h>

#include <avr/io.h>

#include <avr/interrupt.h>

#define A1 0b00000010                                   //Define variable for A1 pin

#define A2 0b00000100                                   //Define variable for A2 pin

//==== Global Variables and Functions ====

char digits[4];                                         //char array of length 4 - in initial state

volatile uint32\_t clock\_count = 0;                      //millisecond timer, this is updated every millisecond using a CTC timer interrupt

volatile uint32\_t display\_time = 0;                     //Time being displayed

volatile uint32\_t saved\_time = 0;                       //Time saved (during pause state)

volatile uint32\_t A1\_time = 0;                          //Time variable for storing time since A1 is pressed first

volatile uint32\_t A2\_time = 0;                          //Time variable for storing time since A1 is pressed first

volatile uint8\_t old\_button = 0b00001110;               //Old Button State

volatile uint8\_t new\_button = 0b00001110;               //New Button State

volatile uint32\_t debounce\_timer = 0;                   //another millisecond timer running off the same timer interrupt to keep track of time for debouncing

volatile uint8\_t button\_flags = 0;                      //Variable for setting button flags

enum STATE {stateInitial, stateCount, statePaused};     //Define each state for the state machine

enum STATE cur\_state;                                   //Define a variable to keep track of the current state

enum STATE next\_state;                                  //Define a variable to keep track of the state being transitioned into

//Segment byte maps for numbers 0 to 9

const uint8\_t SEGMENT\_MAP[] = {

    0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0X80, 0X90,

// Continuing on for A (10) to F (15)

    0x88, 0x83, 0xC6, 0xA1, 0x86, 0x8E,

// Then blank (16), dash (17)

    0xFF, 0x40

};

//Returns the corresponding segmap value

uint8\_t segmentMap(uint8\_t value) {

    return SEGMENT\_MAP[value];

}

//Interrupt for S1-A1 and S2-A2

ISR(PCINT1\_vect) {

    new\_button = PINC;                                  //Current state of PINC

    uint8\_t changed\_bits = old\_button^new\_button;       //Calculate which bits have changed

    if (changed\_bits & A1){                             //Check if button change has occurred on A1

        if ((debounce\_timer - A1\_time)>3){              //If 30 ms has passed

            A1\_time = debounce\_timer;                   //Set A1\_time to equal the value of de-bounce (zero it)

            if ((old\_button & ~new\_button) & A1){       //If there is still a change in A1

                button\_flags |= A1;                     //Set flag to high so it can be processed in loop

            }

            old\_button = new\_button;                    //Change the old button state to the new button state

        }

    }

    if (changed\_bits & A2){                             //Check if button change has occurred on A1

        if ((debounce\_timer - A2\_time)>3){              //If 30 ms has passed

            A2\_time = debounce\_timer;                   //Set A2\_time to equal the value of de-bounce (zero it)

            if ((old\_button & ~new\_button) & A2){       //If there is still a change in A2

                button\_flags |= A2;                     //Set flag to high so it can be processed in loop

            }

            old\_button = new\_button;                    //Change the old button state to the new button state

        }

    }

}

//Interrupt for timer

ISR(TIMER1\_COMPA\_vect) {

    clock\_count++;                                      //Increment clock count (every millisecond))

    debounce\_timer++;                                   //Increment debounce tracking variable

}

void timer1Setup (){

    TCCR1A = 0b01000000;                                //CTC Mode

    TCCR1B = 0b00001011;                                // /64 Prescaler

    OCR1A = 2500;                                       //Number we are "counting" up to.

    TCNT1 = 0;                                          //Number we are "counting" up from.

    ICR1 = 0;                                           //Input Capture Register

    TIFR1 = 0b00000000;                                 //Timer interrupt flag register

    TIMSK1 = 0b00000010;                                //Timer interrupt mask

}

void setup() {

    timer1Setup();   //Runs timerSetup

    //DDR - 0 for input - 1 for output

    //PORT - 0 for no pullup - 1 for pullup

    DDRB = 0b00111101;          //Direction Register B - LEDs

    PORTB = 0b00111100;         //PORTB Internal Pullups

    DDRC = 0b00000000;          //Direction Register C - Attached to buttons and the turnpot

    PORTC = 0b00001110;         //PORTC Internal Pullups

    DDRD = 0b10010000;          //Direction Register D - For clocks

    PORTD = 0b00000000;         //PORTD Internal Pullups

    PCMSK1 = 0b00001110;        //Pin Change Mask Register 0

    PCICR = 0b00000010;         //Pin Change Interrupt Control Register

    sei();                      //Enable global interrupts

}

//Sending data to the 7 segment display using segments and digit array as input

void sendData(uint8\_t segments, uint8\_t digits) {

        PORTD |= \_BV(7);                            //Set PD7 High

        for (int i = 0; i < 8; i++){                //Loop 8 times

            PORTD &= ~\_BV(7);                       //Set PD7 Low

                if(segments & (0b10000000>>i))      //Compare segments to a mask that shifts based on the value of i

                {

                    PORTB |= \_BV(0);                //Set PB0 High

                }

                else

                {

                    PORTB &= ~\_BV(0);               //Set PB0 Low

                }

            PORTD |= \_BV(7);                        //Set PD7 High

        }

        for(int j = 0; j < 8; j++){                 //Loop 8 times

            PORTD &= ~\_BV(7);                       //Set PD7 Low

                if(digits & (0b10000000>>j))

                {

                    PORTB |= \_BV(0);                //Set PB0 High

                }

                else

                {

                   PORTB &= ~\_BV(0);                //Set PB0 Low

                }

            PORTD = 0b10000000;

        }

         PORTD = 0b10010000;

}

//Sends data from digits to the sendData function - which in turn

//displays on the 7 segment

void showDigits() {

    sendData(segmentMap(digits[0]) & 0b01111111, (1<<0));   //Sends data and enables dp

    sendData(segmentMap(digits[1]), (1<<1));

    sendData(segmentMap(digits[2]) & 0b01111111, (1<<2));   //Sends data and enables dp

    sendData(segmentMap(digits[3]), (1<<3));

    sendData(segmentMap(16), 0);                            // blank the display

}

//===Main Loop==//

int main(void) {

    setup();

    cur\_state = stateInitial;                       //Set state to Initial off the bat

    while(1) {                                      //Main Loop

        showDigits();                               //Displays digits (0.00.0)

        switch(cur\_state){                          //Switch statement

            case stateInitial:                      //Initial State - timer is at 0 and count doesn't increment

                PORTB |= 0b11111111;                //Turn off all LEDs

                clock\_count = 0;                    //Reset the time

                display\_time = 0;                   //Change the saved time to the current display time (0)

                if(button\_flags & A1){                      //If A1 is pressed

                    button\_flags = button\_flags & ~A1;      //Reset the flag

                    next\_state = stateInitial;              //Set the next state to initial

                }

                else{

                    if(button\_flags & A2){                  //If A2 is pressed

                        next\_state = stateCount;            //Set the next state transition to stateCount

                        button\_flags = button\_flags & ~A2;  //Reset the flag

                    }

                }

                break;

            case stateCount:                                //Count State – Timer counting up from paused or initial

                PORTB |= 0b11111111;                        //Turn off all LEDs

                display\_time = clock\_count;                 //Assign clock count to display\_time

                if(button\_flags & A1){                      //If A1 is pressed

                    button\_flags = button\_flags & ~A1;      //Reset the flag

                    next\_state = stateInitial;              //Set the next state transition to stateInitial

                }

                else{

                    if(button\_flags & A2){                  //If A2 is pressed

                        button\_flags = button\_flags & ~A2;  //Reset the flag

                        saved\_time = display\_time;          //Save the value of display\_time

                        next\_state = statePaused;           //Set the next state transition to statePaused

                    }

                    else{

                        next\_state = stateCount;            //Return to stateCount

                    }

                }

                break;

            case statePaused:                               //Pauses count and displays the same value, when start/stop is pressed it resumes

                PORTB &= 0b11111011;                        //Turn on D4 LED

                if (button\_flags & A2){                     //If A2 is pressed

                    button\_flags = button\_flags & ~A2;      //Reset the flag

                    clock\_count = saved\_time;               //Assign saved\_time to clock\_count

                    next\_state = stateCount;                //Set the next state transition to stateCount

                }

                else{

                    next\_state = statePaused;               //Otherwise stay in statePaused

                    if (button\_flags & A1){                 //If A1 is pressed

                        button\_flags = button\_flags & ~A1;  //Reset the flag

                        next\_state = stateInitial;          //Set the next state transition to stateCount

                        clock\_count = 0;                    //Reset clock\_count to 0

                    }

                    else{

                        if ((button\_flags & A2)==1){            //If A2 = button\_flags

                            button\_flags = button\_flags & ~A2;  //Reset the flag

                            next\_state = statePaused;           //Set the next state transition to statePaused

                        }

                    }

                }

                break;

                }

        cur\_state = next\_state;                             //set cur\_state to next\_state

        //update display\_time to change the time displayed on the 7 segment display

        digits[0] = ((display\_time/10)/600)%10;

        digits[1] = ((display\_time/100)%60)/10;

        digits[2] = ((display\_time/100)%60)%10;

        digits[3] = (display\_time/10)%10;

        showDigits();

    }

}