//Problem 1

#include <avr/io.h>

//#include <asf.h>

int main (void)

{

/\* Insert system clock initialization code here (sysclk\_init()). \*/

DDRB = 0xFF;

DDRD = 0x00;

PORTD = 0xFF;

while(1)

{

//Because the switch is being connected to ground, we need to check if the pressed switch is connected to ground or not

if ((PIND & (1<<PIND0))==0)

PORTB = 0xFF;

else

PORTB = 0x00;

}

}

//Problem 2

#include <avr/io.h>

//#include <asf.h>

int main (void)

{

DDRB = 0xFF;

DDRD = 0x00;

PORTD = 0xFF;

while(1)

{

//Because the switch is being connected to ground, we need to check if the pressed switch is connected to ground or not

if ((PIND & (1<<PIND3))==0)

PORTB = 0xFF;

else

PORTB = 0x00;

}

}

//Problem 3

#include <avr/io.h>

//#include <asf.h>

#include <avr/interrupt.h>

int main (void)

{

DDRB = (1<<DDB5);

TCCR1B = (1<<CS11);

*uint16\_t* delta\_t= 25000\*1.6;

TCNT1 = 0x0000;

while(1)

{

PORTB = 0b00000000;

while(TCNT1 < delta\_t)

{

}

PORTB = (1<<PINB0);

}

}

//problem 4

#include <avr/io.h>

//#include <asf.h>

#include <avr/interrupt.h>

#define SS PINB2

#define MOSI PINB3

#define MISO PINB4

#define SCK PINB5

void datatransmission(char cData)

{

/\* Start transmission \*/

SPDR = cData;

/\* Wait for transmission complete \*/

while(!(SPSR & (1<<SPIF)));

}

void mainout\_init(void)

{

DDRB |= (1<<DDB3)|(1<<DDB5)|(1<<DDB2);

SPCR |= (1<<SPE)|(0<<DORD)|(1<<MSTR)|(1<<CPOL)|(1<<CPHA)|(0<<SPR0);

SPSR |= (1<<SPI2X);

}

int main (void)

{

mainout\_init();

char data = 0x00;

while(1)

{

datatransmission(data);

data++;

}

}

//Problem 5

#include <avr/io.h>

//#include <asf.h>

#include <avr/interrupt.h>

#define SS PINB2

#define MOSI PINB3

#define MISO PINB4

#define SCK PINB5

void datatransmission(char cData)

{

/\* Start transmission \*/

SPDR = cData;

/\* Wait for transmission complete \*/

while(!(SPSR & (1<<SPIF)));

}

void mainout\_init(void)

{

DDRB |= (1<<DDB3)|(1<<DDB5)|(1<<DDB2);

SPCR |= (1<<SPE)|(0<<DORD)|(1<<MSTR)|(1<<CPOL)|(1<<CPHA)|(0<<SPR0);

SPSR |= (1<<SPI2X);

}

int main (void)

{

char data = 0x00;

while(1)

{

mainout\_init();

datatransmission(data);

data++;

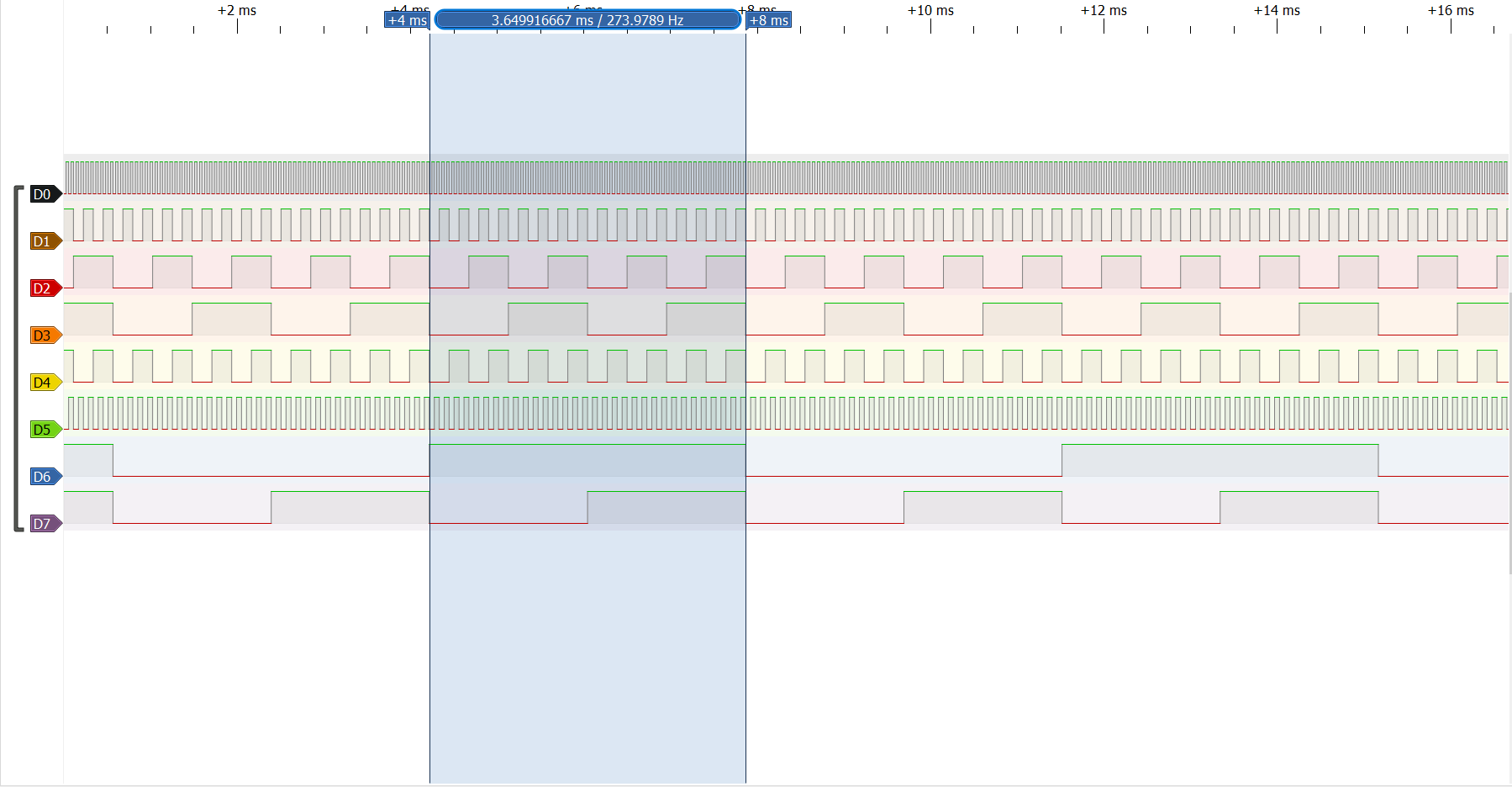
PORTB ^= (1<<PORTB2);

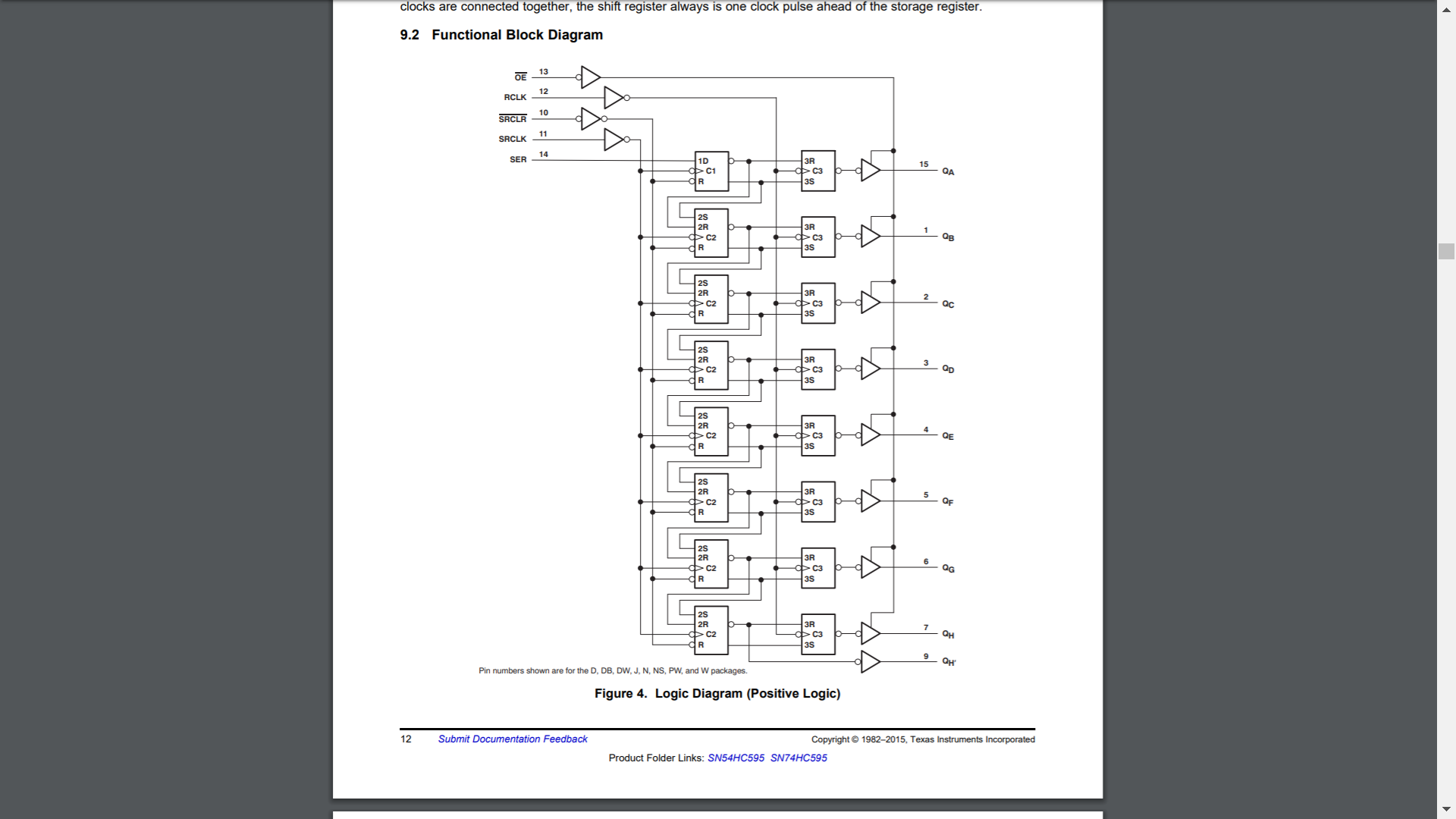
PORTB ^= (1<<PORTB2);

}

}

Note: The smallest rectangles is QA and biggest is QH, which agrees with the graph of shift register





//Problem 6

#include <avr/io.h>

//#include <asf.h>

#include <avr/interrupt.h>

void datatransmission(char cData)

{

/\* Start transmission \*/

SPDR = cData;

/\* Wait for transmission complete \*/

while(!(SPSR & (1<<SPIF)));

}

char init\_port(int a)

{

switch(a){

case 1:

PORTC = (1<<DDC5)|(0<<DDC4)|(0<<DDC3)|(0<<DDC2);

break;

case 2:

PORTC = (0<<DDC5)|(1<<DDC4)|(0<<DDC3)|(0<<DDC2);

break;

case 3:

PORTC = (0<<DDC5)|(0<<DDC4)|(1<<DDC3)|(0<<DDC2);

break;

case 4:

PORTC = (0<<DDC5)|(0<<DDC4)|(0<<DDC3)|(1<<DDC2);

break;

}

}

void SPI\_init(void)

{

DDRB |= (1<<DDB3)|(1<<DDB5)|(1<<DDB2);

SPCR |= (1<<SPE)|(0<<DORD)|(1<<MSTR)|(1<<CPOL)|(1<<CPHA)|(0<<SPR0);

SPSR |= (1<<SPI2X);

TCCR1B = (1<<CS11); //prescaler of 8

}

int main (void)

{ //initialize output port from micro processor to clock

DDRC = (1<<DDC5)|(1<<DDC4)|(1<<DDC3)|(1<<DDC2);

//Array of clock's digits in decimal

char x[] = {63,6,91,79,102,109,125,7,127,103};

int mult = 1;

int display = 0;

char data = 0x00;

int display1 = 0;

*uint16\_t* delta\_t= 0x61A8;

init\_port(mult);

while(1)

{

SPI\_init();

data = x[display];

datatransmission(data);

TCNT1 = 0x0000;

while(TCNT1 <= delta\_t)

{

}

//^= is X-OR => return the opposite of output

//2 X-OR => toggling the port on and off

PORTB ^= (1<<PORTB2);

PORTB ^= (1<<PORTB2);

if(display > 9)

{

display=0;

}

else

{

display++;

}

}

}

//Problem 7

#include <avr/io.h>

void setSPI(void) {

//init values

SPCR = (1<<SPE)|(1<<MSTR)|(0<<SPR0|(0<<SPR1)); //SPE enables SPI, MSTR makes it main, SPR0 and SPR1 select freq

SPSR = (1<<SPI2X); //with SPR0 and SPR1 selects freq

DDRB = (1<<DDB3)|(1<<DDB5)|(1<<DDB2); //makes SCK, MOSI, and SS' outputs

DDRC = (1<<DDC5)|(1<<DDC4)|(1<<DDC3)|(1<<DDC2);

TCCR1B = 0b00000011; //sets up the 64 prescalar value, initiates timer1

}

void digitDisplay\_dot(int a, char digitDisplay[])

{

switch(digitDisplay[a]) {

case(191):

digitDisplay[a]=134;

break;

case(134):

digitDisplay[a]=219;

break;

case(219):

digitDisplay[a]=207;

break;

case(207):

digitDisplay[a]=230;

break;

case(230):

digitDisplay[a]=237;

break;

case(237):

digitDisplay[a]=253;

break;

case(253):

digitDisplay[a]=135;

break;

case(135):

digitDisplay[a]=255;

break;

case(255):

digitDisplay[a]=231;

break;

case(231):

digitDisplay[a]=191;

break;

}

}

void digitDisplay\_noDot(int a, char digitDisplay[])

{

switch(digitDisplay[a]) {

case(63):

digitDisplay[a]=6;

break;

case(6):

digitDisplay[a]=91;

break;

case(91):

digitDisplay[a]=79;

break;

case(79):

digitDisplay[a]=102;

break;

case(102):

digitDisplay[a]=109;

break;

case(109):

digitDisplay[a]=125;

break;

case(125):

digitDisplay[a]=7;

break;

case(7):

digitDisplay[a]=127;

break;

case(127):

digitDisplay[a]=103;

break;

case(103):

digitDisplay[a]=63;

break;

}

}

void digitDisplay\_10sec(int a, char digitDisplay[])

{

switch(digitDisplay[a]) {

case(63):

digitDisplay[a]=6;

break;

case(6):

digitDisplay[a]=91;

break;

case(91):

digitDisplay[a]=79;

break;

case(79):

digitDisplay[a]=102;

break;

case(102):

digitDisplay[a]=109;

break;

case(109):

digitDisplay[a]=63;

break;

}

}

//data transmission method

void dataTransmission(*uint8\_t* data){

SPDR = data;

while(!(SPSR & (1<<SPIF))) //waits for the current transmission to finish

{

}

}

int main(void)

{

//Holder variable declaration:

int i = 0; //temporary value that will increment

int mSec = 0; //mili sec counter

int seconds = 0; //sec counter

int sec10th = 0; //10th sec counter

int minutes = 0; //minute counter

int digit = 0; //index of digitDisplay

*uint8\_t* data = 0b00000000;

//Hardware variables

char portON[4] = {(1<<PORTC5), (0<<PORTC5)|(1<<PORTC4)|(0<<PORTC3)|(0<<PORTC2), (0<<PORTC5)|(0<<PORTC4)|(1<<PORTC3)|(0<<PORTC2),

(0<<PORTC5)|(0<<PORTC4)|(0<<PORTC3)|(1<<PORTC2)}; //array for current output being selected (Current digit)

char x[10] = {63, 6, 91, 79, 102, 109, 125, 7, 127, 103}; //decimal values for 0-9 converted from binary

char digitDisplay[4] = {191, 63, 191, 63}; //Array for storing timer's digits that are being displayed

int temp = 0; //multiplexing variable

PORTD |= (1<< PORTD0); // PORTD OR (1<<PORTD0) 1 is being left shifted by PORTD0

{

while(1) {

//Initialize which port is being turned on and SetSPI ready for transmission

PORTC = portON[digit];

setSPI(); //sets up the SPI

dataTransmission(digitDisplay[temp]);

TCNT1 = 0;

//setting the digits to be on (by making them flashing really fast)

while(TCNT1 < 100){}

//needed for transmission and to start next transmission

PORTB ^= (1<<PORTB2);

PORTB ^= (1<<PORTB2);

// selects the different display digits

if(digit !=3){digit++;}

else {digit = 0;}

// selects the different digit value

if(temp != 3){temp++;}

else {temp = 0;}

//millisecond digit

if(mSec >= 1561){ //1561 is 0.1s (1000000/64/10)-1

mSec = 0; //resets the temp

digitDisplay\_noDot(3,digitDisplay);

}

//seconds digit

if(seconds >= 15619){

if(digitDisplay[3] == 63){ //checks if the milliseconds have hit 0

seconds = 0; //resets the temp

digitDisplay\_dot(0,digitDisplay);

}

//ten-seconds digit

if(1){

if(digitDisplay[0] == 191 && digitDisplay[3] == 63){ //checks if the milliseconds and seconds have hit 0

sec10th = 0; //resets the temp

minutes = 1; //sets minutes so that the the minutes can start next cycle

digitDisplay\_10sec(1,digitDisplay);

}

}

//minute digit

if(minutes == 1){ //minutes == 1 is used because minutes = 0 only occurs at the very beginning, which is when minutes should be 0

if(digitDisplay[1] == 63 && digitDisplay[0] == 191){ // checks if the seconds are at 0

minutes = 0; //resets the temp

digitDisplay\_dot(2,digitDisplay);

}

}

}

mSec = mSec + 53;

seconds = seconds + 53;

sec10th = sec10th + 52;

}

}

}

// Program 8

#include <avr/io.h>

//Initialize SPI function

void setSPI(void) {

//init values

SPCR = (1<<SPE)|(1<<MSTR)|(0<<SPR0|(0<<SPR1)); //SPE enables SPI, MSTR makes it main, SPR0 and SPR1 select freq

SPSR = (1<<SPI2X); //with SPR0 and SPR1 selects freq. 0<<SPR1 and 1<<SPI2X will give divide by 4 then multiply by two (since main out)

DDRB = (1<<DDB3)|(1<<DDB5)|(1<<DDB2); //makes SCK, MOSI, and SS' outputs

DDRC = (1<<DDC5)|(1<<DDC4)|(1<<DDC3)|(1<<DDC2);

TCCR1B = 0b00000011; //sets up the 64 prescalar value, initiates timer1

}

//function for display any digit from 0-9 with dot

void digitDisplay\_dot(int a, char digitDisplay[])

{

//Switch case to cycle through the digit that is being displayed

//E.g: case (current digit is 0) turns it into 1

switch(digitDisplay[a]) {

case(191):

digitDisplay[a]=134;

break;

case(134):

digitDisplay[a]=219;

break;

case(219):

digitDisplay[a]=207;

break;

case(207):

digitDisplay[a]=230;

break;

case(230):

digitDisplay[a]=237;

break;

case(237):

digitDisplay[a]=253;

break;

case(253):

digitDisplay[a]=135;

break;

case(135):

digitDisplay[a]=255;

break;

case(255):

digitDisplay[a]=231;

break;

case(231):

digitDisplay[a]=191;

break;

}

}

//function is used for displaying digits from 0-9 with no dot

void digitDisplay\_noDot(int a, char digitDisplay[])

{

switch(digitDisplay[a]) {

case(63):

digitDisplay[a]=6;

break;

case(6):

digitDisplay[a]=91;

break;

case(91):

digitDisplay[a]=79;

break;

case(79):

digitDisplay[a]=102;

break;

case(102):

digitDisplay[a]=109;

break;

case(109):

digitDisplay[a]=125;

break;

case(125):

digitDisplay[a]=7;

break;

case(7):

digitDisplay[a]=127;

break;

case(127):

digitDisplay[a]=103;

break;

case(103):

digitDisplay[a]=63;

break;

}

}

//function is used for displaying the 10th of second (from 0-5)

void digitDisplay\_10sec(int a, char digitDisplay[])

{

switch(digitDisplay[a]) {

case(63):

digitDisplay[a]=6;

break;

case(6):

digitDisplay[a]=91;

break;

case(91):

digitDisplay[a]=79;

break;

case(79):

digitDisplay[a]=102;

break;

case(102):

digitDisplay[a]=109;

break;

case(109):

digitDisplay[a]=63;

break;

}

}

//function for data transmission

void dataTransmission(*uint8\_t* data){

SPDR = data;

while(!(SPSR & (1<<SPIF))) //waits for the current transmission to finish

{

}

}

int main(void)

{

//Holder variable declaration:

int i = 0; //temporary value that will increment

int mSec = 0; //mili sec counter

int seconds = 0; //sec counter

int sec10th = 0; //10th sec counter

int minutes = 0; //minute counter

int digit = 0; //index of digitDisplay

*uint8\_t* data = 0b00000000; //initialize data variable for transmission

//Hardware variables

//left shift respective port to turn on each of timer’s display

char portON[4] = {(1<<PORTC5),(1<<PORTC4),(1<<PORTC3),(1<<PORTC2)}; //{seconds, 10th secs, minutes, mS}

char x[10] = {63, 6, 91, 79, 102, 109, 125, 7, 127, 103}; //decimal values for 0-9 converted from binary

char digitDisplay[4] = {191, 63, 191, 63}; //Array for storing timer's digits that are being displayed

int temp = 0; //multiplexing variable

//Switch initialization – ports and variables

int run = 0; //checks if allowed to run or not

int press = 0; //checks if button is pressed

//turning on portD so that we start accepting user input from the switches

PORTD |= (1<< PORTD0); //left shift portd0 by 1 to turn it on within portd

DDRD = 0x00; //portD data direction register – (r/w) (e.g: set the state is either in our out) (sets every pinD as input)

PORTD = 0xFF; //portD data register – (r/w) (sets every pinD to high state)

{

while(1) {

//Initialize which port is being turned on and SetSPI ready for transmission

PORTC = portON[digit];

setSPI(); //sets up the SPI

dataTransmission(digitDisplay[temp]);

//resets TCNT1

TCNT1 = 0;

while(TCNT1 < 100){}

//toggling of SS for transmission to start and finish

PORTB ^= (1<<PORTB2);

PORTB ^= (1<<PORTB2);

// selects the different display digits

if(digit !=3){digit++;}

else {digit = 0;}

// selects the different digit value

if(temp != 3){temp++;}

else {temp = 0;}

//If this pin is pressed, reset display

if(!(PIND & (1<<PIND0) == 1)){

TCNT1 = 0;

minutes = 0;

seconds = 0;

sec10th = 0;

mSec = 0;

digitDisplay[0] = 191;

digitDisplay[1] = 63;

digitDisplay[2] = 191;

digitDisplay[3] = 63;

}

//if this pin is pressed, either: start/stop the timer

if ((PIND & (1 << PIND3)) == 0) {

press = 1;

while (press == 1) {

//This step is to tell the timer to save the display of the clock to avoid flashing

PORTC = portON[digit];

setSPI(); //sets up the SPI

dataTransmission(digitDisplay[temp]);

TCNT1 = 0;

//setting the digits to be on (by making them flashing really fast)

while(TCNT1 < 100){}

//needed for transmission and to start next transmission

PORTB ^= (1<<PORTB2);

PORTB ^= (1<<PORTB2);

// select

if(digit !=3){digit++;}

else {digit = 0;}

// selects the different digit value

if(temp != 3){temp++;}

else {temp = 0;}

//ends of multiplexing step

if(!(PIND & (1<<PIND0) == 1)){

TCNT1 = 0;

minutes = 0;

seconds = 0;

sec10th = 0;

mSec = 0;

digitDisplay[0] = 191;

digitDisplay[1] = 63;

digitDisplay[2] = 191;

digitDisplay[3] = 63;

}

//if pin is pressed toggle the state of run/stop of timer

if (!((PIND & (1 << PIND3)) == 0)) {

if (run == 0) {

run = 1;

} else {

run = 0;

}

press = 0;

}

}

}

//if pressed and is run, the timer will continue to run

if (run == 1) {

//millisecond digit

if(mSec >= 1561){ //1561 is 0.1s (frequency/prescaler/10) - 1

mSec = 0; //resets mSec once it exceeded the tick for 100ms

digitDisplay\_noDot(3,digitDisplay);

}

//seconds digit

if(seconds >= 15619){

if(digitDisplay[3] == 63){ //checks if the milliseconds have hit 0

seconds = 0; //resets seconds once it exceed the tick for 1s (100ms tick \* 10)

digitDisplay\_dot(0,digitDisplay);

}

//ten-seconds digit

if(1){

if(digitDisplay[0] == 191 && digitDisplay[3] == 63){ //checks if the milliseconds and seconds have hit 0

sec10th = 0; //resets sec10th if exceed the tick for 10s

minutes = 1; //sets minutes so that the the minutes

digitDisplay\_10sec(1,digitDisplay);

}

}

//minute digit

if(minutes == 1){ //minutes == 1 is used because minutes = 0 only occurs at the very beginning, which is when minutes should be 0

if(digitDisplay[1] == 63 && digitDisplay[0] == 191){ // checks if the seconds are at 0

minutes = 0; //resets the temp

digitDisplay\_dot(2,digitDisplay);

}

}

}

//this is the interval of the respective display wait time to fine tune the timer running speed

mSec = mSec + 53;

seconds = seconds + 53;

sec10th = sec10th + 52;

}

}

}

}