Patrick Austin

CPE 301 - 1104, Fall 2016

Homework 4

10/2/2016

6.1 a.

ATMega328P Table:

|  |  |
| --- | --- |
| **Pin** | **Alternative Function(s)** |
| PB0 | Pin change interrupt 0, timer 1 input capture input, divided system clock output |
| PB1 | Pin change interrupt 1, Output compare pin for timer 1 unit A |
| PB2 | Pin change interrupt 2, output compare pin for timer 1 unit B, SPI slave select |
| PB3 | Pin change interrupt 3, Output compare pin for timer 2 unit A, SPI master-out/slave-in |
| PB4 | Pin change interrupt 4, SPI master-in/slave-out |
| PB5 | Pin change interrupt 5, SPI clock input, LED |
| PB6 | Pin change interrupt 6, chip clock oscillator pin 1, timer oscillator pin 1 |
| PB7 | Pin change interrupt 7, chip clock oscillator pin 2, timer oscillator pin 2 |
| PC0 | Pin change interrupt 8, ADC input channel 0 |
| PC1 | Pin change interrupt 9, ADC input channel 1 |
| PC2 | Pin change interrupt 10, ADC input channel 2 |
| PC3 | Pin change interrupt 11, ADC input channel 3 |
| PC4 | Pin change interrupt 12, ADC input channel 4, TWI serial data |
| PC5 | Pin change interrupt 13, ADC input channel 5, TWI serial clock |
| PC6 | Pin change interrupt 14, processor reset signal |
| PC7 | No pinout |
| PD0 | Pin change interrupt 16, USART receive |
| PD1 | Pin change interrupt 17, USART transmit |
| PD2 | Pin change interrupt 18, external interrupt request 0 |
| PD3 | Pin change interrupt 19, external interrupt request 1, output compare pin for timer 2 unit B |
| PD4 | Pin change interrupt 20, timer0 external counter input, USART external clock I/O |
| PD5 | Pin change interrupt 21, timer1 external counter input, output compare pin for timer 0 unit B |
| PD6 | Pin change interrupt 22, analog comparator positive input, output compare pin for timer 0 unit A |
| PD7 | Pin change interrupt 23, analog comparator negative input |

6.1. b.

ATmega2560R3 table:

|  |  |
| --- | --- |
| **Pin** | **Alternative Function(s)** |
| PB0 | Pin change interrupt 0, Slave select |
| PB1 | Pin change interrupt 1, Serial clock |
| PB2 | Pin change interrupt 2, Master out/slave in |
| PB3 | Pin change interrupt 3, Master in/slave out |
| PB4 | Pin change interrupt 4, Output compare pin for timer 2 unit A, Pulse width modulation output |
| PB5 | Pin change interrupt 5, Output compare pin for timer 1 unit A, Pulse width modulation output |
| PB6 | Pin change interrupt 6, Output compare pin for timer 1 unit B, Pulse width modulation output |
| PB7 | Pin change interrupt 7, Output compare pin for timer 0 unit A, Pulse width modulation output, Output compare for timer 1 unit C, LED |
| PC0 | Analog pin 8 |
| PC1 | Analog pin 9 |
| PC2 | Analog pin 10 |
| PC3 | Analog pin 11 |
| PC4 | Analog pin 12 |
| PC5 | Analog pin 13 |
| PC6 | Analog pin 14 |
| PC7 | Analog pin 15 |
| PD0 | External interrupt 0, Two Wire Interface serial clock |
| PD1 | External interrupt 1, Two Wire Interface serial data |
| PD2 | External interrupt 2, Receive Serial Data Pin 1 |
| PD3 | External interrupt 3, Transmit Serial Data Pin 1 |
| PD4 | No pinout |
| PD5 | No pinout |
| PD6 | No pinout |
| PD7 | Timer 0 |

6.2

unsigned char\* portDDRB;

unsigned char\* portBData;

char testChar;

void setup()

{

/\* Use port B bits 0-6 for output to the 7 segment LED. So enable bits 0-6 in DDRB and

prepare the address of the port B data register for output.

Convention: port B bit 0 = segment A

bit 1 = segment B

…

bit 6 = segment G

as per the layout of LEDs in the question. Assumes cathode display, starts displaying zero. \*/

portDDRB = (unsigned char\* ) 0x24;

\*portDDRB = \*portDDRB | 0x7F;

portBData = (unsigned char\* ) 0x25;

testChar = ‘0’;

}

void loop()

{

//show a char on the LED, delay, then show the next. Cycles 0-9 then A-F then back to 0

displayChar(testChar);

testChar++;

if ( testChar == ‘:’ ) //passed char ‘9’, go to ‘A’

testChar = ‘A’;

if ( testChar == ‘G’ ) //passed char ‘F’, go back to ‘0’

testChar = ‘0’;

myDelay(500);

}

void displayChar( char arg )

{

/\* For a given input, mask portBData such that bit 7 is unchanged and bits 0-6 equal the

required segments. \*/

if ( arg == ‘A’ )

{

\*portBData = \*portBData | 0x77; //enforce high on bits 0-2 and 4-6 with mask 0111 0111

\*portBData = \*portBData & 0xF7; //enforce low on bit 3 with mask 1111 0111

}

else if ( arg == ‘B’ )

{

\*portBData = \*portBData | 0x7F; //enforce high on bits 0-6 with mask 0111 1111

}

else if ( arg == ‘C’ )

{

\*portBData = \*portBData | 0x39; //enforce high on bits 0, 3, 4, 5 with mask 0011 1001

\*portBData = \*portBData & 0xB9; //enforce low on bits 1, 2, 6 with mask 1011 1001

}

else if ( arg == ‘D’ )

{

\*portBData = \*portBData | 0x7F; //enforce high on bits 0-5 with mask 0011 1111

\*portBData = \*portBData & 0xBF; //enforce low on 6 with mask 1011 1111

}

else if ( arg == ‘E’ )

{

\*portBData = \*portBData | 0x79; //enforce high on bits 0, 3-6 with mask 0111 1001

\*portBData = \*portBData & 0xF9; //enforce low on 1-2 with mask 1111 1001

}

else if (arg == ‘F’ )

{

\*portBData = \*portBData | 0x71; //enforce high on bits 0, 4-6 with mask 0111 0001

\*portBData = \*portBData & 0xF1; //enforce low on 1-3 with mask 1111 0001

}

else if ( arg == ‘1’ )

{

\*portBData = \*portBData | 0x06; //enforce high on bits 2 and 3 with mask | 0000 0110

\*portBData = \*portBData & 0x86; //low on bits 0-1 and 4-6 with mask & 1000 0110

}

else if ( arg == ‘2’ )

{

\*portBData = \*portBData | 0x5B; //enforce high on bits 0-1, 3-4, 6 with mask 0101 1011

\*portBData = \*portBData & 0xDB; //enforce low on 2 and 5 with mask 1101 1011

}

else if ( arg == ‘3’ )

{

\*portBData = \*portBData | 0x4F; //enforce high on bits 0-3, 6 with mask 0100 1111

\*portBData = \*portBData & 0xCF; //enforce low on 4-5 with mask 1100 1111

}

else if ( arg == ‘4’ )

{

\*portBData = \*portBData | 0x66; //enforce high on bits 1, 2, 5, 6 with mask 0110 0110

\*portBData = \*portBData | 0xE6 ; //enforce low on bits 0, 3, 4 with mask 1110 0110

}

else if ( arg == ‘5’ )

{

\*portBData = \*portBData | 0x7F; //enforce high on bits 0, 2, 3, 5, 6 with mask 0110 1101

\*portBData = \*portBData & 0xED; //enforce low on bits 1, 4 with mask 1110 1101

}

else if ( arg == ‘6’ )

{

\*portBData = \*portBData | 0x7D; //enforce high on bits 0 and 1-6 with mask 0111 1101

\*portBData = \*portBData & 0xFD; //enforce low on bit 1 with mask 1111 1101

}

else if ( arg == ‘7’ )

{

\*portBData = \*portBData | 0x07; //enforce high on bits 0-2 with mask 0000 0111

\*portBData = \*portBData & 0x87; //enforce low on bits 3-6 with mask 1000 0111

}

else if ( arg == ‘8’ )

{

\*portBData = \*portBData | 0x7F; //enforce high on bits 0-6 with mask 0111 1111

}

else if ( arg == ‘9’ )

{

\*portBData = \*portBData | 0x6F; //enforce high on bits 0-3 and 5-6 with mask 0110 1111

\*portBData = \*portBData & 0xEF; //enforce low on bit 4 with mask 1110 1111

}

else if ( arg == ‘0’ )

{

\*portBData = \*portBData | 0x7F; //enforce high on bits 0-5 with mask 0011 1111

\*portBData = \*portBData & 0xBF; //enforce low on 6 with mask 1011 1111

}

else

{

\*portBData = \*portBData & 0x80; //for an unimplemented char, no light. 06 low with mask 1000 0000

}

}

void myDelay (unsigned long mSecondsDesired)

{

//delay method, adapted from textbook

volatile unsigned long i;

unsigned long time = 1000 \* mSecondsDesired;

for ( i = 0; i < time; i++ );

}

6.3

/\* Algorithm: this method should be called after a measurement has been taken on some input pin where testing for bounce is desired. After calling it, retest the input. If the original value and the value post-call match, the reading was valid. If not, the reading was invalid.\*/

void debounce()

{

myDelay(20);

}

void myDelay (unsigned long mSecondsDesired)

{

//delay method, adapted from textbook

volatile unsigned long i;

unsigned long time = 1000 \* mSecondsDesired;

for ( i = 0; i < time; i++ );

}

6.4

unsigned char\* portDDRB;

unsigned char\* portPinB;

unsigned char\* portDataB;

unsigned char currentValue;

unsigned char testValue;

unsigned char previousValue;

void setup()

{

previousValue = 0x00;

portDDRB = (unsigned char\*) 0x24;

portPinB = (unsigned char\*) 0x23;

portDataB = (unsigned char\*) 0x25;

\*portDDRB = \*portDDRB & 0xFD //enforce 0 on bit 0, to make it an input pin. mask 1111 1110

\*portDDRB = \*portDDRB | 0x20; //enforce 1 on bit 5 to enable LED output. mask 0010 0000

}

void loop ()

{

//take an input reading, but enforce 0 on all bits but the one of interest, bit 0. mask 0000 0001. Then wait

currentValue = \*portPinB & 0x01;

debounce(); //wait a little while, using debounce implemented in the previous problem

//take another reading of the input to see if the values still match after the delay

testValue = \*portPinB & 0x01;

if ( testValue == currentValue ) //if the values match the input was good, proceed normally.

//If they didn’t match we will do nothing for now and wait for the reading on the next loop

{

if ( currentValue != previousValue )

\*portDataB = \*portDataB ^ 0x20; //flip the value of bit 6 and leave the other bits unchanged

previousValue = currentValue; //update previousValue for the next loop

}

}

void debounce()

{

myDelay(20);

}

void myDelay (unsigned long mSecondsDesired)

{

//delay method, adapted from textbook

volatile unsigned long i;

unsigned long time = 1000 \* mSecondsDesired;

for ( i = 0; i < time; i++ );

}