Willis T. Allstead
Professor Egbert
CPE 301
16 November 2016

HW 8

1) An interrupt serves as a sort of function which can be called at any time within a program execution. This is different than a normal function though because normal functions require pre-defined calling positions throughout a program execution.

2) When an interrupt occurs the following happens: the ISR begins which in turn saves the context of the program, the ISR function's instructions are run, then the context is restored after the function is completed.

3) Interrupt features for the Atmega328P include ISR_BLOCK which is just an ISR with no attributes specified, ISR_NOBLOCK which enables global interrupts when it is initiated, ISR_NAKED which makes the compiler not generate context-management code by itself, and ISR_ALIASOF(vect) which is linked to the ISR specified by 'vect' which allows one ISR function to handle multiple interrupt signals.

4) Interrupt priority is the decides the order in which interrupts are dealt with if called at the same time. It is defined by the fact that lower addresses have higher priority to higher addresses.

5) To properly configure an interrupt a system designer has to know how and where to properly disable and enable interrupts. This is important because if interrupts change global variables that are in use by other actions in the program, serious problems can occur in the execution of the rest of the program one the interrupt is finished.

6) To turn off the interrupt system just & the status register with the not of 0x80. to turn it on, or it with 0x80. (This applies specifically to the Atmega328P).

7) (compiled program below)

ANSI-C_Blink_timers_5_interrupts

```
volatile unsigned char *myTCCR1A = (unsigned char *) 0x80; // Timer/Counter Control Register A
volatile unsigned char *myTCCR1B = (unsigned char *) 0x81; // Timer/Counter Control Register B
volatile unsigned char *myTCCR1C = (unsigned char *) 0x82; // Timer/Counter Control Register C
volatile unsigned char *myTIMSK1 = (unsigned char *) 0x6F; // Timer/Counter Interrupt Mask Register
volatile unsigned int  *myTCNT1  = (unsigned  int *) 0x84; // Timer/Counter Register (low & high)
volatile unsigned char *myTIFR1 =  (unsigned char *) 0x36; // Timer/Counter Interrupt Flag Register
volatile unsigned char *portDDRB = (unsigned char *) 0x24;
volatile unsigned char *portB =    (unsigned char *) 0x25;
volatile unsigned char *TIMSK1_ADDR =    (unsigned char *) 0x6F;
#define TIMSK1_ICIE_MASK = 0x20;
#define TIMSK1_OCIEB_MASK = 0x04;
#define TIMSK1_OCIEA_MASK = 0x02;
#define TIMSK1_TOIE_MASK = 0x01;

unsigned int Tcount = 5000;
unsigned char Tmode = 0x05;


void genFreq();

void setup() {
  /* Initialize Timer1 for NORMAL mode */
  *myTCCR1A = 0;
  *myTCCR1B = 0;
  *myTCCR1C = 0;
  *myTIMSK1 = 0; // Timer 1 should have no interrupts

  *portDDRB |= 0x80;  // Initialize GPIO PortB
}

void loop() {
  genFreq();
}

void genFreq() {
  /* TODO: 0.125 of a second, 12.5% interrupted <---> 87.5% not interrupted */

  /*  period = 0.125 seconds = 125 milliseconds
   *  125 milliseconds = 125,000 microseconds
   *  125,000 microseconds/64 microseconds = 1,953.125
   *  we need to subtract 1953 from the top of the timer and use pre-scalar of 1024
   */

  *portB |= 0x80; // turn on arduino LED

  *myTCCR1B &= 0xF8; // turn timer to OFF
  *myTCNT1 = (unsigned int) (65536 - 1953);
  *myTCCR1B |= 0x05; // turn timer ON with pre-scalar of 1024 (0b101 => 0x05)
  while((*myTIFR1 & 0x01)!=1); // once overflow flag is 1, we will stop
  *myTCCR1B &= 0xF8; // turn timer to OFF (prof. Egbert says this is optional)
  *myTIFR1 |= 0x01;// clear overflow flag bit (by setting to 1 for some reason)

  *portB &= 0x7F; // turn off LED
}

ISR(TIMER1_OVF_vect) {
  *myTCCR1B = 0; // turn off the timer
  *portB ^= 0x40; // toggle PortB bit 6
  *myTCNT1 = Tcount; //Reset timer count
  *myTCCR1B = Tmode; // Pre-scalar value
}
```

```
Done compiling.

Sketch uses 962 bytes (0%) of program storage space. Maximum is 253,952 bytes.
Global variables use 29 bytes (0%) of dynamic memory, leaving 8,163 bytes for local variables. Maximum is 8,192 bytes

60                                                    Arduino Mega ADK on /dev/cu.usbm
```