SECTION OVERVIEW

Complete the following objectives:

- Understand when interrupts can be used, and how they are used.
- Demonstrate how a previous lab’s implementation can be improved by making use of external interrupts.
- Learn about some of the interrupt facilities that are available on the AT-mega128 microcontroller.
- Explore the ATmega128 datasheet to learn how to configure and enable specific interrupts on your mega128 microcontroller board.

PRELAB

To complete this prelab, you may find it useful to look at the full ATmega128 datasheet. If you consult any online sources to help answer the prelab questions, you must list them as references in your prelab.

1. In computing, there are traditionally two ways for a microprocessor to listen to other devices and communicate: polling and interrupts. Give a concise overview/description of each method, and give a few examples of situations where you would want to choose one method over the other.

2. Describe the function of each bit in the following ATmega128 I/O registers: EICRA, EICRB, and EIMSK. Do not just give a brief summary of these registers; give specific details for each bit of each register, such as its possible values and what function or setting results from each of those values. Also, do not just directly paste your answer from the datasheet, but instead try to describe these details in your own words.

3. The ATmega128 microcontroller uses interrupt vectors to execute particular instructions when an interrupt occurs. What is an interrupt vector? List the interrupt vector (address) for each of the following ATmega128 interrupts: Timer/Counter0 Overflow, External Interrupt 5, and Analog Comparator.

BACKGROUND

Most modern computing systems use interrupts to communicate with peripheral devices. Interrupts can be very beneficial because they allow a processor to continue executing useful instructions until a peripheral device indicates it needs attention.

Using interrupts can also be tricky, as they can sometimes result in lot of overhead (i.e., time that must be spent, but is not spent doing anything productive). When an interrupt request comes in, the processor has to stop what it is doing, save its current place in the program (including storing any in-use or otherwise special variables), and then it can service the interrupt. Once the event that has caused an interrupt as been handled, the processor must take the time to reload any stored variables, and then it can finally resume what it was doing before the interrupt occurred. This process, which is the cost of servicing an interrupt in this manner, is referred to as a context switch.

Depending on how long it takes to service an interrupt, and depending on how frequently the event causing the interrupt occurs, the processor may not be able to spend much time on its original task before another interrupt occurs. For example, if a peripheral device wants the processor to store a single byte
of data every couple of clock cycles, it may try to interrupt the processor every
time another byte is ready. This would cause the processor to spend all of
its time storing variables, servicing the interrupt, and reloading variables, only
to immediately be interrupted again since the next byte is ready. Since this
scenario is very clearly undesirable, many modern computers use coprocessors
and peripheral controllers (like a DMA controller, for example) to handle frequent
requests.

Despite the potential downsides of using interrupts, there are of course still
situations where their use is preferred, such as handling infrequent events which
do not justify spending any time busy-waiting.

PROCEDURE

For this lab, you need to write a short assembly program that causes your TekBot
to move forward. Then, when either the right or left whisker is hit, it will need
to react by backing up for 1 second, turning away for 1 second, and then moving
forward again. The TekBot counts how many times each whisker is triggered
and the LCD displays two counters for left/right whiskers. As you can probably
tell, this is the same BumpBot behavior that you saw previously in both Lab 1
and Lab 2. Polling was used to detect whisker hits in these prior labs, but this
time you must use external interrupts to detect a falling edge on either
of the whisker inputs. You need to use INT0, INT1, INT2, and INT3 for a right
whisker input, a left whisker input, clearing a right whisker counter, and clearing
a left whisker counter, respectively.

You must write your code so that your TekBot can only be interrupted by
bumper hits when it is moving forward (i.e., not while currently in the middle of
any HitRight or HitLeft behavior). Additionally, you must not allow bumper hit
interrupts to queue up while you are in the middle of any HitRight or HitLeft
behavior. For example, if you hit the right bumper first and then hit the left
bumper while the TekBot is still performing its HitRight behavior, the TekBot
must not go directly into HitLeft once HitRight has finished.

A skeleton file has been provided to assist you; also, you can reuse some code
from BasicBumpBot.asm. To demonstrate you have completed the implementa-
tion portion of this lab, show your TA the BumpBot operation, and explain how
your code was written to meet the additional requirements mentioned above.

STUDY QUESTIONS / REPORT

A full lab write-up is required for this lab. When writing your report, be sure
to include a summary that details what you did and why, explains any
problems you may have encountered, and answers the study questions
given below. Your write-up and code must be submitted by the beginning of
next week’s lab. Remember, NO LATE WORK IS ACCEPTED.

Study Questions

1. As this lab, Lab 1, and Lab 2 have demonstrated, there are always multiple
ways to accomplish the same task when programming (this is especially
true for assembly programming). As an engineer, you will need to be able
to justify your design choices. You have now seen the BumpBot behavior
implemented using two different programming languages (AVR assembly
and C), and also using two different methods of receiving external input
(polling and interrupts).

   Explain the benefits and costs of each of these approaches. Some important
   areas of interest include, but are not limited to: efficiency, speed, cost of
   context switching, programming time, understandability, etc.

2. Instead of using the Wait function that was provided in BasicBumpBot.asm,
is it possible to use a timer/counter interrupt to perform the one-second
   delays that are a part of the BumpBot behavior, while still using external
   interrupts for the bumpers? Give a reasonable argument either way, and be
   sure to mention if interrupt priority had any effect on your answer.

CHALLENGE

With the basic BumpBot behavior, a TekBot can sometimes get stuck in a loop if
it encounters a corner: continually backing up, hitting the right whisker, backing
up again, hitting the left whisker, then the right whisker again, then the left
whisker again, and so on.

To complete the challenge for this lab, add a “memory” to your TekBot so that
it can detect this problem. When the TekBot has hit alternating whiskers five
times, it should reverse like normal, but then turn around 180 degrees (i.e., turn
for a few seconds), and then resume forward motion to get out of the corner.

In addition, correct the simple problem of the TekBot hitting the same object
multiple times in a row. In this scenario, the TekBot hits one of its whiskers,
backs up and turns away as usual, but does not turn far enough and hits the
same object with the same whisker. Add code to detect this scenario, so that the TekBot will back up and turn away twice as long as normal whenever the same whisker is hit twice in a row.

Add these changes to your code, and make sure your additions are well-documented. Demonstrate the improved behavior to your TA to receive challenge credit for this lab, and be sure to submit your challenge code as a separate file.