Lab 9

Using Interrupt-Driven Input/Output

Due: Week of April 18, Before the start of your lab section

This is an individual-effort project. You may discuss concepts and syntax with other students, but you may discuss solutions only with the professor and the TAs. Sharing code with or copying code from another student or the internet is prohibited.

In this assignment, you will write code for your Arduino Nano-based class hardware kit that will use interrupts from external devices and from a timer to implement the ability to convert numbers between radices.

Figure 1: Interrupts. Image by 20th Century Fox Television

The instructions are written assuming you will edit the code in the Arduino IDE and run it on your Arduino Nano-based class hardware kit, constructed according to the pre-lab instructions. If you wish, you may edit the code in a different environment; however, our ability to provide support for problems with other IDEs is limited.

Please familiarize yourself with the entire assignment before beginning. Section 2 has the functional specification of the system system you will develop. Section 3 describes implementation constraints. Section 4 relates interrupt-driven I/O to event-driven programming (which you should have learned about in CSCE 156, RAIK 184H, or SOFT 161). Section 5 identifies code that you can reuse from PollingLab. Section 6 will guide you through configuring a hardware timer and responding to interrupts from that timer. Section 7 will guide you through registering and writing interrupt service routines for external interrupts. Finally,

Sections 8 and 9 offers suggestions for using your interrupt service routines to implement the system's specifications.

Learning Objectives

After successful completion of this assignment, students will be able to:

- Use tables from a datasheet to determine the bit vectors needed to configure I/O devices
- Configure a hardware timer to generate interrupts
- Register an interrupt service routine for an interrupt vector
- Register an interrupt service routine using a higher abstraction
- Use interrupt-driven I/O to realize simple requirements

Continuing Forward

After completing PollingLab and IntegerLab, you will be ready for the Group Project, in which you will implement a simple embedded system.

During Lab Time

During your lab period, the TAs will demonstrate how to use tables from a datasheet to construct bit vectors. During the remaining time, the TAs will be available to answer questions.

No Spaghetti Code Allowed

In the interest of keeping your code readable, you may *not* use any **`goto`** statements, nor may you use any **`continue`** statements, nor may you use any **`break`** statements to exit from a loop, nor may you have any functions **`return`** from within a loop.

1 Scenario

Smoke wafts from Herb's soldering iron as he looks up when you approach. Cleaning the iron's tip, he quotes: "Somebody once said, 'The three most dangerous things in the world are a programmer with a soldering iron, a hardware engineer with a software patch, and'" –

he glances nervously in Archie's direction – "'a user with an idea.'[1,2]" Herb gets straight to the point. "We promised Archie that we'd be able to start using the Cow Pi to build systems in a week. So far we've tested its input/output functionality, but we still need to test its timer and also whether we can take inputs without constantly polling the input devices. As before, we don't need to do anything too fancy; let's try a number base conversion tool."

2 Number-Base Conversion Tool Specification

1. The user shall be able to build a number.
   (a) The number being built shall initially hold the value 0.
   (b) When the **right switch** is toggled to the left, the conversion tool will use the decimal number base. When the user presses a button on the **matrix keypad** with a decimal numeral (*0-9*) then the conversion tool shall take the appropriate action as specified below. Any other buttons on the keypad shall be ignored.
   (c) When the **right switch** is toggled to the right, the conversion tool will use the hexadecimal number base. When the user presses a button on the **matrix keypad** then the conversion tool shall take the appropriate action as specified below. Buttons with decimal numerals (*0-9*) or alphabetic letters (*A-D*) shall be interpreted as having the corresponding hexadecimal numeral; the button with the octothorp (#) shall be interpreted as having the hexadecimal numeral *E*; and the button with the asterisk (*) shall be interpreted as having the hexadecimal numeral *F*.
   (d) When the number being built holds the value 0, the **7-segment display module** shall display 0 in the least-significant digit; no other digits shall be initially illuminated.
   (e) Whenever the user presses a button on the **matrix keypad**, the corresponding numeral shall be displayed in the least-significant position of the **7-segment display module**, and any digits already displayed shall increase in significance by one order of magnitude. For example, if 234 is displayed and the user presses 5 then 2345 shall be displayed. The numeral displayed shall follow the interpretations specified in requirements 1b and 1c.
      i. There shall be no noticeable lag in updating the display.
      ii. The new value shall be printed to the Serial Monitor in decimal or hexadecimal, depending on the system's current number base.
   (f) "0x" shall not be displayed as part of a hexadecimal value.

[1] Rick Cook, *The Wizardry Consulted*, 1995.

[2] The notion of being wary of programmers wielding screwdrivers or soldering irons long pre-dated this quote, as there are apocryphal tales of people who found it easier to modify the hardware to suit the software rather than the other way around.

(g) A positive sign shall not be displayed as part of a positive value.

(h) If a negative decimal value is displayed, the negative sign shall be displayed immediately to the left of the most-significant digit being displayed. For example, `-456` is correctly displayed, but `-  456` is not correctly displayed.

(i) Except when the number being built holds the value 0 (see requirement 1d), leading 0s shall not be displayed. For example, `782` is allowed, but `00000782` is not allowed.

(j) Whenever the user presses and then releases the **right pushbutton** at least 150ms later, the number being built shall reset to 0, and the **7-segment display module** shall be cleared, *except* that the least-significant digit shall display `0` (this being the new number being built). *The change shall take effect only when the user releases the **right pushbutton**.*

(k) Whenever the user presses and then releases the **left pushbutton** at least 150ms later, the value being displayed shall be negated. *The change shall take effect only when the user releases the **left pushbutton**.*

i. In the decimal number base, the presence or absence of negative sign shall indicate whether or not a value is negative

ii. In the hexadecimal number base, 32-bit two's complement shall be used.

(l) In no case shall the tool allow the user to input a value too great to be displayed on the **7-segment display module**. If the user attempts to enter a value greater than $0x7FFF,FFFF$ in hexadecimal, less than $0x8000,0000$ in hexadecimal, greater than $99,999,999$ in decimal, or less than $-9,999,999$ in decimal, then the **7-segment display module** shall display `too  bIG`.

- *n.b.*, in a hexadecimal 32-bit negative integer, an `F` in the most-significant hex-digit might only be a sign extension; in this scenario, if the user inputs another hex-digit then it is still a valid number because it would not be less than $0x8000,0000$.

  For example, adding another digit to the 32-bit integer $0xF865,4321$ would *not* produce a too-great value because the 36-bit integer $0xF,8654,3210 = -2,041,302,512_{10}$, and the 32-bit integer $0x8654,3210 = -2,041,302,512_{10}$. Because $0x8000,0000 = -2,147,483,648_{10} \leq -2,041,302,512_{10}$, adding the `0` digit did not produce a "too big" value.

  On the other hand, adding another digit to the 32-bit integer $0xF765,4321$ *would* produce a too-great value because the 36-bit integer $0xF,7654,3210 = -2,309,737,968_{10}$, but the 32-bit integer $0x8654,3210 = 1,985,229,328_{10}$. Because $0x8000,0000 = -2,147,483,648_{10} > -2,309,737,968_{10}$, adding the `0` digit did produce a "too big" value.

(m) Nothing shall be printed on the Serial Monitor when the system is in building mode.

2. The user shall be able to see the number converted to a different number base.

   (a) Whenever the user presses double-clicks the **left pushbutton**, the **7-segment display module** shall display the number in the opposite number base. **A *double-click* is defined as pressing the button and releasing it no more than 150ms later, and pressing it again no later than 500ms after that.**

      i. If the **right switch** is toggled to the left, then the decimal number shall be displayed as the equivalent hexadecimal number.

      ii. If the **right switch** is toggled to the right, then the hexadecimal number shall be displayed as the equivalent decimal number.

      iii. If the value is too great to be displayed in the new number base, then Error shall be displayed.

   (b) The conversion shall take place immediately after the **left pushbutton** is double-clicked.

   (c) If the **left switch** is toggled to the left, then exactly 20 seconds after the **left pushbutton** is double-clicked, the number shall be displayed in its original number base.

   (d) If the **left switch** is toggled to the right, then exactly 7.5 seconds after the **left pushbutton** is double-clicked, the number shall be displayed in its original number base.

   (e) If Error is displayed due to the value being too great to be displayed in the new number base, then after 7.5 seconds or 20 seconds (as appropriate) has elapsed, Error shall no longer be displayed, and the number shall be displayed in its original number base.

   (f) While the converted number is being displayed, the **external LED** shall be illuminated. When the number is being displayed in its original number base, the **external LED** shall not be illuminated.

   (g) If the **left pushbutton** is double-clicked, then releasing the **left pushbtton** shall not cause the number to be negated.

3. If the user changes the position of the **right switch** while the number is a value other than 0, the system's behavior is unspecified.

4. If the user changes the position of the **left switch** while the coverted number is being displayed, the system's behavior is unspecified.

5. If the user presses a key on the keypad or one of the pushbuttons while the converted number is being displayed, the system's behavior is unspecified.

   - After the number in its original number base is again displayed, the system's behavior is as specified above.

3 Constraints

You may continue to use the memory-mapped I/O registers, or you can use functions provided by the Arduino core read from and write to pins.

You may re-use code from the previous lab, or you can rewrite the necessary code using functions, macros, types, or constants that are part of the Arduino core;[3] functions, macros, types, or constants of avr-libc;[4] or one of Arduino's standard libraries.[5] You may *not* use a third-party library, not even one that the Arduino Libraries reference page links to (if it isn't among those available in the Arudino IDE's "Sketch" → "Include Library" menu under "Arduino libraries" then you may not use it).

You may *not* poll the matrix keypad nor the pushbuttons to determine if they have been pressed. You must use interrupts to determine if a key or button has been pressed. Once a press has been detected, you may scan the matrix keypad or read the pushbuttons to determine which key or button has been pressed.

You may poll the switches to determine if either's position has changed; however, the specification has been written such that your code should only need to occasionally check the switches' positions rather than polling them for changes.

While it is possible to configure the SPI hardware to generate an interrupt after the content of the SPI Data Register is transmitted, you may use your **`displayData()`** function that polls the `SPIF` bit.

While you may use **`millis()`** for debouncing, you may *not* use **`millis()`** nor **`micros()`** to determine when a converted number should return to being displayed in its original number base. You must use an interrupt from the ATmega328P's Timer1 or Timer2 to determine when a converted number should reutrn to being displayed in its original number base.

While it is typically okay to have code in **`loop()`** while also using interrupts, you may *not* have any code in the **`loop()`** function for this assignment. You also may not introduce an infinite loop elsewhere in your program. While you typically want to keep your interrupt service routines (ISRs) as short as possible, you may (indeed, you must) have all of your logic in your interrupt service routines (or functions called by your ISRs) for this assignment.

Do *not* edit *cowpi.h*; all of your code must go in *InterruptLab.ino*.

4 A Different but Familiar Programming Paradigm

Most of the code you wrote in the earlier lab assignments for this courses used imperative programming: you, the programmer, had full control over changes to the program state. In PollingLab, you stretched this model to something resembling shared-memory concurrent programming: you still had full control over changes to the program state of your program's flow of control, but your program periodically read variables (memory-mapped input

[3] https://www.arduino.cc/reference/en/

[4] https://www.nongnu.org/avr-libc/user-manual/index.html

[5] https://www.arduino.cc/reference/en/libraries/ The standard libraries are those under the heading, "Standard Libraries."

registers) that coul changed by another flow of control (the physical world).

In this lab assignment, you will write code that reacts to things happening in the physical world; none of your code will run except in reaction to interrupts. Conceptually, this isn't very different from event-driven programming that you learned in CSCE 156, RAIK 184H, or SOFT 161. While configuring the hardware timer is a little more complex than configuring a GUI framework's timer, handling a timer interrupt is very much like writing an **`onTimeout()`** event handler. Similarly, handling a change in a pushbutton's position is very much like writing an **`onMouseClick`** event handler. Your code is not focused on *when* the button is pressed or released, nor even on *detecting* that a button has been pressed or released; it is focused solely on *what should happen* when the button is pressed or released.

5 Getting Started

Copy code from your *PollingLab.ino* into *InterruptLab.ino* that you think may be useful, such as **`displayData()`**, the address assignments to `ioPorts` and `spi`, and assorted global arrays. You will want to keep your code for scanning the keypad handy, as you will be able to reuse most of it with only a few modifications. You will also want to keep your code for building numbers handy, though it will require several changes.

If you did not successfully implement **`displayData()`** during PollingLab, you may have **`displayData()`** simply call **`cowpi_sendDataToMax7219(address, value)`**. If you did not successfully implement **`getKeypressed()`** during PollingLab, you may have **`handleKeypress()`** call **`cowpi_getKeypress()`** (note that **`cowpi_getKeypress()`**) does not take care of switch debouncing, and it returns the character corresponding to the symbol on the key that was pressed, not the numeric value specified by this assignment.)

If you need to rewrite any other code from the previous lab, you may do so without explicitly using the memory-mapped I/O registers. You may instead use **`digitalRead()`**[6] and **`digitalWrite()`**[7] to perform input and output on specific pins.

6 Detecting Time

You must use timer interrupts from either Timer1 or Timer2 as part of your implementation of the conversion timeout. Without using an external clock source, you won't be able to configure an interrupt to occur every 20 seconds, or even every 7.5 seconds. Since we will not use an external clock source, you will need to use timer interrupts along with other logic.

6.1 Preparation

Design your logic and determine how often you need a timer interrupt to make it work. For example, the Arduino Nano's pseudorealtime clock relies on an interrupt from Timer0

[6] https://www.arduino.cc/reference/en/language/functions/digital-io/digitalread/
[7] https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/

every 1.024µs and advances the millisecond counter after 1,000 of these interrupts have occurred. The pseudorealtime clock uses an overflow-based timer interrupt to arrive at an approximation of milliseconds. The conversion tool's specification calls for the converted number to display for exactly 7.5 seconds or exactly 20 seconds (depending on the switch position) before the original number is displayed again. To achieve exactness, you will use a comparison-based timer.

You can then determine the parameters for the timer using this equation:

$$16,000,000\frac{\text{cycles}}{\text{second}} = comparison\_value\frac{\text{beats}}{\text{interrupt}} \times prescaler\frac{\text{cycles}}{\text{beat}} \times interrupt\_frequency\frac{\text{interrupts}}{\text{second}}$$

or, equivalently:

$$comparison\_value\frac{\text{beats}}{\text{interrupt}} \times prescaler\frac{\text{cycles}}{\text{beat}} = 16,000,000\frac{\text{cycles}}{\text{second}} \times interrupt\_period\frac{\text{seconds}}{\text{interrupt}}$$

where:

16,000,000 Hz is the clock frequency (the inverse of the clock period).

comparison_value is a number you will place in one of the timer's `compare` registers for a comparison-based timer interrupt. This can be any possible value of an unsigned 16-bit integer for Timer1, or any possible value of an unsigned 8-bit integer for Timer2.

prescaler is a multiplier applied to the clock period to adjust the time between counter increments ("beats"). Possible values are 1, 8, 64, 256, and 1024 for Timer1, or 1, 8, 32, 64, 128, 256, and 1024 for Timer2.

interrupt_frequency is how often you want a timer interrupt (the inverse of the interrupt period).

interrupt_period is the time between timer interrupts (the inverse of the interrupt frequency).

You may have to iterate on your design until you arrive at one that works with the constraints of that equation's terms for whichever timer you choose to use.

The Waveform Generation Mode you will use is *CTC* (Clear Timer on Compare) with the "TOP" value set by the `OCRnA` (*compareA*) register, where n is the Timer number. Use Figure 2 (for Timer1) or Figure 4 (for Timer2) to select the appropriate `WGM13` `WGM12` `WGM11` `WGM10` or `WGM22` `WGM21` `WGM20` bits.

Based on the prescaler you chose for the above equation, select the appropriate `CS12` `CS11` `CS10` or `CS22` `CS20` `CS20` bits from Figure 3 (for Timer1) or Figure 5 (for Timer2).

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, phase correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, phase correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, phase correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, phase and frequency correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, phase and frequency correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, phase correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, phase correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Note: 1. The CTC1 and PWM11:0 bit definition names are obsolete. Use the WGM12:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

Figure 2: Waveform Generation Mode Bit Description for Timer1. Copied from ATmega382P Data Sheet, Table 15-5

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Figure 3: Clock Select Bit Description for Timer1. Copied from ATmega382P Data Sheet, Table 15-6

Mode	WGM2	WGM1	WGM0	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, phase correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, phase correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Notes: 1. MAX = 0xFF
2. BOTTOM = 0x00

Figure 4: Waveform Generation Mode Bit Description for Timer2. `WGM22`, `WGM21`, and `WGM20` are incorrectly shown here as `WGM2`, `WGM1`, and `WGM0`. `OCR2A` is incorrectly shown here as `OCRA`. Copied from ATmega382P Data Sheet, Table 17-8

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk_{T2S}/(no prescaling)
0	1	0	clk_{T2S}/8 (from prescaler)
0	1	1	clk_{T2S}/32 (from prescaler)
1	0	0	clk_{T2S}/64 (from prescaler)
1	0	1	clk_{T2S}/128 (from prescaler)
1	1	0	clk_{T2S}/256 (from prescaler)
1	1	1	clk_{T2S}/1024 (from prescaler)

Figure 5: Clock Select Bit Description for Timer2. Copied from ATmega382P Data Sheet, Table 17-9

6.2 Setup

You may configure the timer using either memory-mapped I/O or using macros provided by AVR-libc.[8]

In the **`setupTimer()`** function:

6.2.1 If using memory-mapped I/O

Configure the timer:

- If you will use Timer1, remove the comment marks and elipses for the `timer1` global variable. Assign to `timer1` the address `(cowpi_timerRegisters16bit *)(cowpi_IObase + 0x60)`. You can delete the commented-out assignment to `timer2`.
- If you will use Timer2, remove the comment marks and elipses for the `timer2` global variable. Assign to `timer2` the address `(cowpi_timerRegisters8bit *)(cowpi_IObase + 0x90)`. You can delete the commented-out assignment to `timer1`.
- Use `timer1`'s or `timer2`'s `control` field to set the `WGM` and `CS` bits in the timer's control registers.
  - Use Tables 1 and 2 (for Timer1) or Tables 3 and 4 (for Timer2) to determine where the `WGM` and `CS` bits are located in the control registers.
- Subtract one from your computed *comparison_value*, and assign the resulting value to `timer1`'s or `timer2`'s `compareA` field.[9]

Enable the comparison-based timer interrupt:

- Create a pointer to a **`volatile`** `uint8_t` and assign to it the address `cowpi_IObase + 0x4E`.
- Treat the pointer as an array, and use the timer number (1 or 2) for the index when making an assignment.

You want to place a 1 in the `OCIEnA` bit (where n is the timer number); use Figure 6 to determine the appropriate bit to set.

6.2.2 If using AVR-libc macros

Configure the timer:

- If you will use Timer1, use Table 2 to determine where the `WGM` and `CS` bits are located in the control registers. Make the relevant assignments to `TCCR1A` and/or `TCCR1B`.

[8] AVR-libc provides macros named after the I/O registers that allow you to read from and write to these registers as though they were ordinary variables.

[9] You will subtract one because counting the integer values in the range $0 \ldots (comparison\_value - 1)$ will count the *comparison_value* beats between timer interrupts. Realistically, on the human-scale, you probably won't notice the difference.

- If you will use Timer2, use Table 4 to determine where the `WGM` and `CS` bits are located in the control registers. Make the relevant assignments to `TCCR2A` and/or `TCCR2B`.
- Subtract one from your computed *comparison_value*, and assign the resulting value to to OCR1A (Timer1) or OCR2A (Timer2).[9]

Enable the comparison-based timer interrupt:

- Make an assignment to `TIMSK1` for Timer1, or to `TIMSK2` for Timer2.

You want to place a 1 in the `OCIEnA` bit (where n is the timer number); use Figure 6 to determine the appropriate bit to set.

6.3 Interrupt Service Routine

Because the Arduino core does not provide a more convenient way to create an interrupt service routine (ISR) for timer interrupts, you will use AVR-libc's **`ISR`**[10] macro.

In *CalculatorLab.ino*, outside of any other function, write this code that looks like a function:

```
ISR(vector) {
    ...
}
```

where *vector* is `TIMER1_COMPA_vect` for Timer1 or `TIMER2_COMPA_vect` for Timer2. Replace "..." with the code that should execute whenever the timer interrupt occurs.

For now, simply place a **`println`** statement in your ISR code to verify that you have the timing correct. Later you will replace the **`println`** statement with the code you need for your logic.

NOTE Any global variables used by your ISR should be declared as **`volatile`**.

NOTE If you ever need to "reset" a timer's count back to 0, you can simply write 0 to `timer1`/`timer2`'s' `counter` field or to `TCNT1`/`TCNT2`.

7 Detecting Key and Button Presses

For external interrupts, the Arduino core has abstracted-away all of the configuration details.[11] Placing a call to

```
attachInterrupt(digitalPinToInterrupt(pinNumber), isrName, mode);
```

will configure all of the necessary registers to call the function **`isrName()`** whenever the input value on the pin *pinNumber* satisfies the *mode*.

[10] https://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html

[11] https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

Bits	**31..24**	**23..16**	**15..8**	**7..0**
control	*reserved*	TCCR1C	TCCR1B	TCCR1A
counter			TCNT1	
capture			ICR1	
compareA			OCR1A	
compareB			OCR1B	

Table 1: Relationship of `cowpi_timerRegisters16bit` fields to Timer1's registers. Adapted from ATmega382P Data Sheet, §15.11.

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TCCR1C	FOC1A	FOC1B	-	-	-	-	-	-
TCCR1B	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10
TCCR1A	COM1A1	COM1A0	COM1BA1	COM1B0	-	-	WGM11	WGM10

Table 2: Timer1's control registers. Adapted from ATmega382P Data Sheet, §15.11.

Bits	**15..8**	**7..0**
control	TCCR2B	TCCR2A
counter		TCNT2
compareA		OCR2A
compareB		OCR2B

Table 3: Relationship of `cowpi_timerRegisters8bit` fields to Timer2's registers. Adapted from ATmega382P Data Sheet, §17.11.

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TCCR2B	FOC2A	FOC2B	-	-	WGM22	CS22	CS21	CS20
TCCR2A	COM2A1	COM2A0	COM2BA1	COM2B0	-	-	WGM21	WGM20

Table 4: Timer2's control registers. Adapted from ATmega382P Data Sheet, §17.11.

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TIMSK2	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2
TIMSK1	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1
TIMSK0	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0

Figure 6: Timer interrupt registers. Cropped from ATmega382P Data Sheet, §30

7.1 Setup

The two functions that we recommend be called in response to a keypress on the matrix keypad and in response to pressing or releasing a pushbutton, **`handleKeypress()`** and **`handleButtonAction()`**, are already stubbed in *InterruptLab.ino*.

Recall that the NAND output for the matrix keypad columns provides input to `D3` and that the NAND output for the pushbuttons provides input to `D2`. Decide on the *mode* for the external interrupts:

LOW to trigger the interrupt whenever the pin is low

RISING to trigger the interrupt whenever the pin goes from low to high

FALLING to trigger the interrupt whenever the pin goes from high to low

CHANGE to trigger the interrupt whenever the pin rises or falls

Because we did not provide a hardware solution to switch bounce, you can expect the pin input to both rise and fall a few times when a button or key is pressed and again when it is released – but there is no guarantee that bouncing will occur. For this reason we recommend:

- Use the `CHANGE` mode, combined with software debouncing, and use a variable to keep track of whether a key or button has been pressed (toggle this variable whenever an interrupt occurs, and take action based on whether a key or button has been pressed or released).
- The software debouncing will look similar to what you used in PollingLab except that it only needs to be a few milliseconds instead of 500. Since we are not polling the buttons and keypad to detect presses, we do not need to worry about the button or key being held for several dozen milliseconds being interpretted as multiple presses.
  - I very strongly advise against using **`delay()`** for software debouncing:
  - As described in the PollingLab assignment sheet, **`delay()`** will leave your system unresponsive to anything except interrupts.
  - Including **`delay()`** calls in an interrupt handler is particularly ill-advised since you may find yourself in a situation in which you need to disable interrupts in the interrupt handler and then re-enable interrupts before exiting the interrupt handler. (I do not expect this to happen in this assignment, but I've been wrong before.) The **`delay()`** function will never exit, blocking forever, if interrupts are disabled.

If you arrive at a different solution that works (including using more than the two stubbed ISRs), you may use your solution.

In the **`setup()`** function, register your ISR functions for the external interrupts:

```
attachInterrupt(digitalPinToInterrupt(2), handleButtonAction, CHANGE);
attachInterrupt(digitalPinToInterrupt(3), handleKeypress, CHANGE);
```

(Here I assumed you would use the CHANGE mode. If you use a different mode, replace CHANGE with the mode you chose. Similarly, if you use more than the two stubbed ISRs, register those ISRs, too.)

7.2 Handling Button Actions

Your pushbutton handler (**`handleButtonAction()`**) needs to determine which button was pressed or released, and whether it was pressed or released – this can be as simple as keeping track of each button's position and calling `digitalRead(8)` and `digitalRead(9)`.[12] Don't forget to include software debouncing. If insufficient time (a few milliseconds) has passed since the last time the button handler was invoked, you can simply exit the handler function under the assumption that switch bounce is generating erroneous interrupts.

NOTE Any global variables used by your ISR should be declared as **`volatile`**.

For now, place a **`println`** statement in your ISR code indicating the pushbuttons' positions to confirm that you are correctly detecting which button has been pressed or released and whether it was pressed or released.

7.3 Handling Key Presses

Your keypad handler (**`handleKeypress`**) needs to determine which key was pressed – you can reuse your code from PollingLab that scans the keypad with minimal changes. One such change is that you don't need to **`return`** a value. If insufficient time (a few milliseconds) has passed since the last time the keypad handler was invoked, you can simply exit the handler function under the assumption that switch bounce is generating erroneous interrupts.

NOTE Any global variables used by your ISR should be declared as **`volatile`**.

For now, place a **`println`** statement in your ISR code indicating which key was pressed to confirm that you are correctly detecting keypresses.

8 Implementing Conversion Timeout

Because you weren't be able to configure an interrupt to occur every 20 seconds, or even every 7.5 seconds, you need to use timer interrupts along with other logic to determine when the correct amount of time for displaying the converted number has lapsed. This will probably take place in your timer ISR, in **`handleButtonAction()`**, and/or helper functions.

NOTE Any global variables used by your ISRs or their helper functions should be declared as **`volatile`**.

Implement code to determine when the left button has been released more than or less than 150ms after it was pressed.

[12] Another way to determine if a button was pressed or released is to use separate ISRs for `RISING` and `FALLING`; however, you will still need to determine *which* button rose or fell, and whether it was due to the button being manipulated or due to switch bounce.

Implement code to determine when the left button has been pressed, released no more than 150ms later, and pressed again no more than 500ms after that; *i.e.*, determine when the left button has been double-clicked.

Implement code that, based on the position of the left switch, will determine when 7.5 seconds or 20 seconds have elapsed since the left button was double-clicked.

Modify the **`println`** statements in your timer ISR and in **`handleButtonAction()`** to:

- Print when the left button has been double-clicked.
- Print when 7.5 seconds or 20 seconds (as appropriate) have elapsed since the double-click.
- Print when a button is released, and which button, but only if it was more than 150ms after it was pressed.

Add code that will illuminate the external LED when the left button has been double-clicked and that will deluminate the external LED 7.5 seconds or 20 seconds (as appropriate) later.

9 Building and Converting Numbers

The specification for building numbers is generally the same as was in PollingLab, except that leading zeroes are prohibited. Modify your "Building Mode" code from PollingLab to work within the constraints of this assignment. Your code will likely be spread across **`handleKeypress()`**, **`handleButtonAction()`**, and possibly helper functions. Further modify your to prevent leading zeroes if necessary.

NOTE Any global variables used by your ISRs or their helper functions should be declared as **`volatile`**.

Modify your code from Section 8 so that, instead of printing that the left button has been double-clicked, the converted number will be displayed (*i.e.*, if the number is being built in decimal then the equivalent number in hexadecimal will be displayed, and if the number is being build in hexadecimal then the equivalent number in decimal will be displayed). Further modify your code so that, instead of printing that 7.5 seconds or 20 seconds (as appropriate) have elapsed, the number in its original number base will be displayed. *Don't forget to display* `Error` *if the value is too great to be displayed in the conversion number base.*

Remove any remaining `println` statements.

Turn-in and Grading

When you have completed this assignment, upload *InterruptLab.ino* to Canvas.

This assignment is worth 40 points.

Rubric:

Pushbutton Interrupts

_____ **+4** Pushbutton presses and releases are detected with an external interrupt.

_____ **+2** The pushbuttons' interrupt handler determines which button was released.

_____ **+2** The pushbuttons' interrupt handler determines whether the left button has been double-clicked.

Matrix Keypad Interrupts

_____ **+4** Matrix keypad presses are detected with an external interrupt.

_____ **+2** The keypad's interrupt handler determines which key was pressed.

Timer Interrupts

_____ **+2** Timer interrupts for Timer1 or Timer2 are enabled and handled.

_____ **+4** The timer interrupts configured such that, when combined with other logic, the software is able to determine when exactly 7.5 seconds or exactly 20 seconds have passed since the left button was double-clicked.

Conversion Timeout

_____ **+2** The system is able to determine when exactly 7.5 seconds have passed since the left button was double-clicked.

_____ **+2** The system is able to determine when exactly 20 seconds have passed since the left button was double-clicked.

Building Numbers

_____ **+3** Builds a value consistent with requirements 1a–1i. ($\frac{3}{4}$ point for each of: displaying correct positive decimal values, displaying correct negative decimal values, displaying correct positive hexadecimal values, displaying correct negative hexadecimal values)

_____ **+1** Negates value when left pushbutton is released. ($\frac{1}{4}$ point for each of: displaying correct positive decimal values, displaying correct negative decimal values, displaying correct positive hexadecimal values, displaying correct negative hexadecimal values)

_____ **+1** Releasing the right pushbutton clears all digits on the Display Module except the least-significant digit, which displays 0.

_____ **+1** Detects and displays correct message when the number being built is too big. ($\frac{1}{2}$ point for detecting too-big numbers; $\frac{1}{4}$ point for no false detections; $\frac{1}{4}$ point for displaying the correct message)

Converting Numbers

_____ **+2** Converts from decimal to hexadecimal when the left button is double-clicked. (1 point for correctly-displayed positive numbers; 1 point for correctly-displayed negative numbers)

_____ **+2** Converts from hexadecimal to decimal when the left button is double-clicked. (1 point for correctly-displayed positive numbers; 1 point for correctly-displayed negative numbers)

_____ **+1** Displays the number in its original number base 7.5 seconds or 20 seconds (as appropriate) after displaying the converted number.

_____ **+1** Detects and displays correct message when the number being converted to the other number base is too great for the new number base.

_____ **+1** Displays the number in its original number base 7.5 seconds or 20 seconds (as appropriate) after displaying the error message.

_____ **+1** The external LED illuminates while and only while the converted number (or the convered error message) is displayed.

Other Requirements

_____ **+1** The number being built does not change when a button is pressed.

_____ **+1** If the left button is double-clicked, then the number being built does not change when the button is released.

_____ **Bonus +2** Get assignment checked-off by TA or professor during office hours before it is due. (You cannot get both bonuses.)

_____ **Bonus +1** Get assignment checked-off by TA at *start* of your scheduled lab immediately after it is due. (Your code must be uploaded to Canvas *before* it is due. You cannot get both bonuses.)

Penalties

_____ **-0** No penalty needed for "Pushbutton Interrupts," "Matrix Keypad Interrupts," or "Timer Interrupts" portions of rubric since credit for these parts of the rubric is not possible without complying with the constraints in Section 3.

_____ **-4** Code associated with conversion timeout (other than that covered in the first penalty item) relies on code that violates the constraints in Section 3.

_____ **-6** Code associated with building numbers (other than that covered in the first penalty item) relies on code that violates the constraints in Section 3.

_____ **-8** Code associated with converting numbers (other than that covered in the first two penalty items) relies on code that violates the constraints in Section 3.

_____ **-2** Code associated with certain button actions having no effect (other than that covered in the first penalty item) relies on code that violates the constraints in Section 3.

_____ **-1** for each **`goto`** statement, **`continue`** statement, **`break`** statement used to exit from a loop, or **`return`** statement that occurs within a loop.

Epilogue

You and Herb look for Archie in the Pleistocene Petting Zoo's labs to give him the good news, and you find a blond woman wearing cargo shorts, butchering a Gilbert and Sullivan song...

♪ I am the very model of a modern vice admiral ♪
♪ I've information about all things paleobotanical ♪
♪ And I've been up to my armpits in problems scatological ♪
♪ During the regency I had experience matriarchical ♪
♪ I plot space travel, normal and superluminal ♪
♪ (Even if I challenge the Pauli exclusion principle) ♪

"I don't know how these people keep getting into our labs. *Please* tell me that you have good news," pleads Archie.

"Yes, the Cow Pi is ready for whatever you need: calculators, security systems, parking meters – you name it," Herb cheerfully responds.

"Excellent." Archie turns to you. "I'd like you and Newm... no, *not* Newman. I'd like you and someone else on the staff to get started right away. Here's what I'd like to have built first."

To be continued...

Appendix: Lab Checkoff

You are not required to have your assignment checked-off by a TA or the professor. If you do not do so, then we will perform a functional check ourselves. In the interest of making grading go faster, we are offering a small bonus to get your assignment checked-off at the start of your scheduled lab time immediately after it is due. Because checking off all students during lab would take up most of the lab time, we are offering a slightly larger bonus if you complete your assignment early and get it checked-off by a TA or the professor during office hours.

( ) Establish that the code you are demonstrating is the code you submitted to to Canvas.

- If you are getting checked-off during lab time, show the TA that the file was submitted before it was due.

- Download the file into your PollingLab directory. If necessary, rename it to *InterruptLab.ino*.

( ) Upload *InterruptLab.ino* to your Arduino Nano and open the Serial Monitor.

1. ( ) Show in your code that your **`loop()`** function has no executable code, and that there is not an infinite loop in the **`setup()`** fuction, nor in any functions called by **`setup()`**.
   establishing this now allows us to conclude that the observed functionality is in the ISRs or their helper functions

2. ( ) Show in your code that you registered interrupt handlers for the pushbuttons and for the matrix keypad.
   make note of the interrupt condition

3. ( ) Show in your code your interrupt handlers for the pushbuttons and for the matrix keypad.
   +4 pushbutton presses and releases are detected with an external interrupt
   (if a single handler is registered for pushbutton CHANGEs, or
   if handlers are registered for both FALLING and RISING pushbuttons)
   +4 matrix keypad presses are detected with an external interrupt

4. ( ) Show and explain how you determined the comparison value and prescaler for your timer interrupts.

5. ( ) Show in your code that you had configured the timer to use those comparison and prescaler values and that you had configured the correct timer mode.
   +4 timer interrupts are configured correctly

6. ( ) Show in your code that you enabled the correct timer.

7. ( ) Show in your code that you created an ISR for the correct timer.
   +2 timer interrupts are enabled and handled

8. ( ) Place the right switch in the left (decimal) position.

9. ( ) Press 1. One of two things will happen:

   - 1 appears on the Serial Monitor.
   - ¹ appears on the display module.

   +2 keypad ISR determines which key was pressed

10. ( ) Press the left pushbutton. The Serial Monitor may indicate that the left pushbutton is DOWN and the right pushbutton is UP, but there is no other observable behavior.

11. ( ) Release the left pushbutton. One or both of two things will happen:
    - The Serial Monitor indicates that both pushbuttons are UP.
    - -1 appears on the display module.
12. ( ) Press the right pushbutton. The Serial Monitor may indicate that the right pushbutton is DOWN and the left pushbutton is UP, but there is no other observable behavior. *+1 The number being build does not change when a button is pressed.*
13. ( ) Release right left pushbutton. One or both of two things will happen:
    - The Serial Monitor indicates that both pushbuttons are UP.
    - 0 appears on the display module.

    +2 pushbutton ISR determines which button was released
14. ( ) Place the left switch in the right (hexadecimal) position. Place the right switch in the right (7.5 seconds) position.
15. ( ) Press A. A may appear on the display module.
16. ( ) Double-click the left pushbutton. One or more of three things will happen:
    - The Serial Monitor indicates that a double-click occurred.
    - The display will update to 10 or an attempt to do so, such as 0 or 1.
    - The external LED illuminates.

    +2 pushbutton ISR determines whether the left button is double-clicked
17. ( ) Wait 7.5 seconds. One or more of three things will happen:
    - The Serial Monitor indicates that 7.5 seconds have elapsed.
    - The display will revert to A or an attempt to do so.
    - The external LED deluminates.

    +2 system determines when 7.5 seconds have passed
    +1 external LED illuminates while and only while converted number is displayed +1 the number being built does not change when the left button is released during a double-click
18. ( ) Place the right switch in the left (20 seconds) position.
19. ( ) Double-click the left pushbutton. One or more of the previously-mentioned indications of simultaneous button presses happen.

20. ( ) Wait 20 seconds. One or more of the previously-mentioned indications of conversion timeout happen.
+2 system determines when 20 seconds have passed

21. ( ) Place the right switch in the right (7.5 seconds) position.

22. ( ) Press and release the right pushbutton. The output is 0 on the display module.

23. ( ) Enter the value 0x123A. The output is 123A on the display module.
+¾ displays correct positive hexadecimal values

24. ( ) Press the and release the left pushbutton. The output is FFFFEDC6 on the display module.
+¼ correctly negates positive hexadecimal values

25. ( ) Press and release the right pushbutton. The output is 0 on the display module.
+1 releasing the right pushbutton clears all digits and then displays 0

26. ( ) Enter the value 0xB654789C. The output is B654789C on the display module.
+¾ displays correct negative hexadecimal values

27. ( ) Press the and release the left pushbutton. The output is 49A87B64 on the display module.
+¼ correctly negates negative hexadecimal values

28. ( ) Place the left switch in the left (decimal) position.

29. ( ) Press and release the right pushbutton. The output is 0 on the display module.

30. ( ) Enter the value 1478. The output is 1478 on the display module.
+¾ displays correct positive decimal values

31. ( ) Press the and release the left pushbutton. The output is -1478 on the display module.
+¼ correctly negates positive decimal values

32. ( ) Press and release the right pushbutton. The output is 0 on the display module.

33. ( ) Enter the value -5236. The output is -5236 on the display module.
+¾ displays correct negative decimal values

34. ( ) Press the and release the left pushbutton. The output is 5236 on the display module.
+¼ correctly negates negative decimal values

35. ( ) Press and release the right pushbutton. The output is 0 on the display module.

36. ( ) Enter the value 12369874. The output is 12369874 on the display module.
+¼ no false "too big" detections

37. ( ) Press 0. The output is 8too bIG on the display module.
+½ detects that a number is "too big" +¼ displays correct error message

38. ( ) Press and release the right pushbutton. The output is 0 on the display module.

39. ( ) Enter the value 1234. The output is 1234 on the display module.

40. ( ) Double-click the left pushbutton. The output is 4d2 on the display module.
+1 converts positive decimal to hexadecimal

41. ( ) Wait 7.5 seconds for the conversion to timeout, and press and release the left pushbutton. The output is -1234 on the display module.

42. ( ) Double-click the left pushbutton. The output is FFFFF62E on the display module.
+1 converts negative decimal to hexadecimal

43. ( ) Press and release the right pushbutton. The output is 0 on the display module.

44. ( ) Place the left switch in the right (hexadecimal) position.

45. ( ) Enter the value 0x56B7. The output is 5667 on the display module.

46. ( ) Double-click the left pushbutton. The output is 22199 on the display module.
+1 converts positive hexadecimal to decimal

47. ( ) Wait 7.5 seconds for the conversion to timeout, and press and release the left pushbutton. The output is FFFFA949 on the display module.

48. ( ) Double-click the left pushbutton. The output is -22199 on the display module.
+1 converts negative hexadecimal to decimal

49. ( ) Press and release the right pushbutton. The output is 0 on the display module.

50. ( ) Enter the value 0x6A0,0000. The output is 6A00000 on the display module.

51. ( ) Double-click the left pushbutton. The output is Error on the display module.
+½ detects that converted number is too great to display
+¼ displays correct error message

52. ( ) Press and release the right pushbutton. The output is 0 on the display module.

53. ( ) Enter the value 0x5F5,E0FF. The output is 5F5E0FF on the display module.

54. ( ) Double-click the left pushbutton. The output is 99999999 on the display module.
+¼ no false "error" detections

This concludes the demonstration of your system's functionality. The TAs will later examine your code for violations of the assignment's constraints. If your code looks like it is tailored for this checklist, the TAs may re-grade using a different checklist.