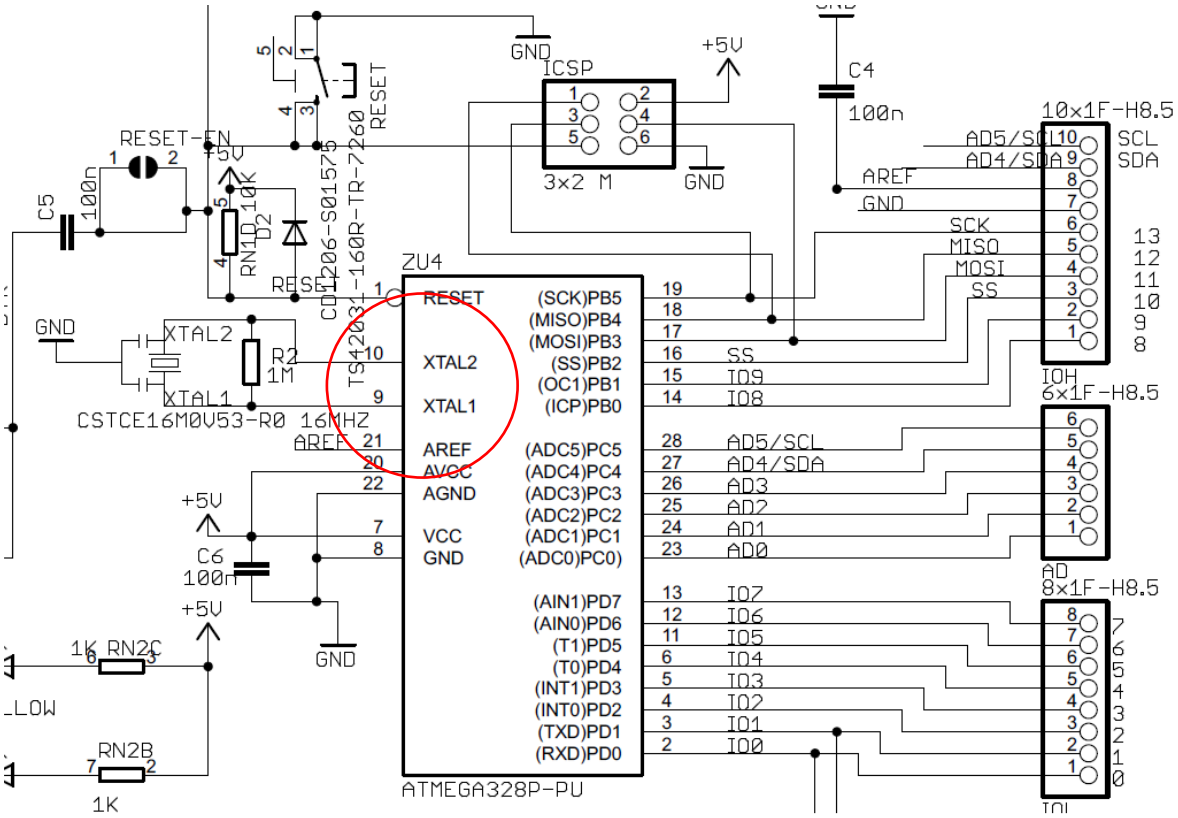
CG1112 Engineering Principles and Practices II for CEG
Tutorial 2
GPIO, Interrupts and PWM

Part I – GPIO

Question 1

Most Microcontroller Pins have multiple functionalities. It is up to the designer to plan ahead and be aware of the peripherals that are needed and select the appropriate pins to be used in the project. In the studio we have used PORT B pins for the LED's and Switches.

PB7 and PB6 are currently mapped to the external crystal oscillator (XTAL1 and XTAL2 for the Uno Board).



You have decided to create your own board using the Atmega328p and want to make use of all 8-bits of PORTB. Is it possible? What are the factors to consider?

(Hint: Refer to "Section 13: System Clock and Clock Options" in the Atmega datasheet)

Answer:

Referring to Table 13-1, it can be seen that there are different clocking options available for the 328p. Two of those options refer to the ability to use an internal clock. By selecting those options we can free up PB7 and PB6 to be used as GPIO.

Table 13-1. Device Clocking Options Select

Device Clocking Option	CKSEL[3:0]
Low Power Crystal Oscillator	1111 - 1000
Full Swing Crystal Oscillator	0111 - 0110
Low Frequency Crystal Oscillator	0101 - 0100
Internal 128kHz RC Oscillator	0011
Calibrated Internal RC Oscillator	0010
External Clock	0000
Reserved	0001

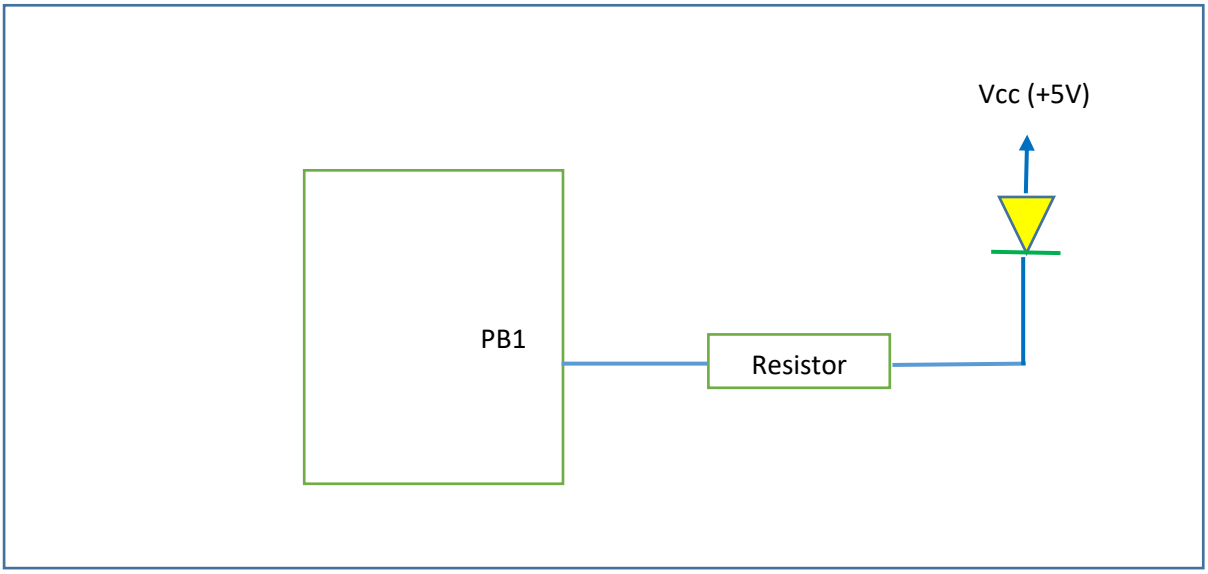
The most important factor to consider is that when we use an external clock, we have a wide range of frequencies to choose from. Once we switch to an internal clock, we are limited by what the device provides. We need to check and see if what is provided is sufficient for our needs.
Internal 128KHz RC Oscillator: 128khz
Calibrated Internal RC Oscillator: 8MHz

DISCUSSION POINTS:
What are the factors to consider when choosing the Frequency? Speed, Power Consumption, Stability.
Implication to the code: Calculated values for timing related features like ADC Sampling, Timers, etc.

Question 2.
In our studio we configured LED's in "Active-High" logic, i.e. we applied a Logic '1' to Turn it 'ON' and a Logic '0' to turn it 'OFF'.

a. Draw a circuit connection to connect an LED in "Active-Low" logic to PORTB Pin 1, i.e. you need to apply a Logic '0' to turn it 'ON' and a Logic '1' to turn it 'OFF'.

Answer:



b. Write the code to set the DDRB register value according to your schematic.

Answer: DDRB |= 0b00000010; // Set Bit 1 to output by OR with '1'.

c. In a design such as this, we say that the microcontroller is "sinking" current. That means, that current is now flowing into the pin of the device. The manufacturer specifies a limit to the amount of sinking current. Anything more than this, could lead to a "smoke-effect" + "water-sprinkler" effect in the lab.

Refer to Table 32-2 in the Atmega328p datasheet and state the maximum sinking current when the device is operating at 5V.

Answer: 20mA

d. The voltage drop across the LED is 0.7V. Choose an appropriate R value to prevent a "smoke-effect".

Answer:

R >= (5 – 0.7) / 20 mA = 215 ohms

The minimum value of R is 215 ohms.

DISCUSSION POINTS:

How much is the source current when you set the output to '1'?

Answer: 20mA. The absolute maximum is 40mA.

Is that enough?

Answer: Depends on the application. If you are controlling an LED, then it should be. If you are trying to interface to a Motor, then it may not be enough.

Question 3

Power Consumption is a critical factor in Embedded Systems and it is important to minimize is to as to extend its usage before recharging the batteries. You read from Section 14 of the datasheet that the AT328P has some energy-saving features that you want to implement for "Alex" your robot. Mainly, you wish to put the robot in "Standby Mode" when the robot is idle and not doing anything useful.

a. Which is the main register that controls these features and what value should be written to it?

Answer:
The Sleep Mode Control Register (SMCR) contains the control bits for power management. To put the device in Standby Mode, a value of 110 must be written to it bits SM2:SM0 (Bits 3:1).

b. In Standby Mode, which events can trigger the device to “wake-up” and resume full-functionality?

Answer:

Only one of these events can wake up the MCU:
- External Reset
- Watchdog System Reset
- Watchdog Interrupt
- Brown-out Reset
- 2-wire Serial Interface address match
- External level interrupt on INT
- Pin change interrupt

c. How many clock cycles does it take for the device to come out of Standby mode to full operation.

It takes 6 clock cycles.

d. Based on your initial assessment, you feel that you may not need to use the ADC module for this project. As such, you want to disable it so that it doesn’t consume any additional power. How can this be achieved?

Answer:

The Power Reduction Register (PRR) provides a method to stop the clock to individual peripherals to reduce power consumption.

Bit	7	6	5	4	3	2	1	0
	PRTWI0	PRTIM2	PRTIM0		PRTIM1	PRSPI0	PRUSART0	PRADC
Access	R/W	R/W	R/W		R/W	R/W	R/W	R/W
Reset	0	0	0		0	0	0	0

Bit 0 – PRADC: Power Reduction ADC
Writing a logic one to this bit shuts down the ADC. The ADC must be disabled before shut down. The analog comparator cannot use the ADC input MUX when the ADC is shut down.

DISCUSSION POINTS:

Implication of Power Consumption.

- Battery Life / Electricity Bill

- Challenges to meet Certification Standard
https://www.energystar.gov/

If you don't meet their standard, you can't put their sticker on your product. Most Consumer Electronic Products have to meet Energy Star requirements. In some countries, it is mandatory for some classes of products to meet E* requirements. In local context, we can think of the "ticks" assigned to products like Washing Machines, Air-Cons, etc.

It is important that Energy Consumption must be thought through while you are designing the project and not after it is complete. The microcontroller is only one component in the system. Your device may have a lot of other subsystems that consume power. Putting those components into a low-energy state may be more complicated and require HW design like "electronic-switches".

Part II – Interrupts

Question 4.

What are hardware interrupts? Why are they needed? Give some examples of hardware interrupts in your laptop or PC.

A hardware interrupt consists of a series of request lines built into the CPU or MCU. Its purpose is to let hardware signal to the CPU to get its attention.

The CPU can be busy running code, but when an interrupt request line is triggered (either by a rising edge, falling edge or change in signal level), the CPU stops what it is doing, and runs a special piece of code called an interrupt service routine (ISR) to handle the interrupt. The CPU then resumes what it was doing, where it left off.

Examples on a PC or laptop include interrupts from the keyboard when someone has pressed a key, or from the network interface when a frame of data has come in over the network, etc.

Similarly, what are software interrupts? Give some examples of how software interrupts can be used.

A software interrupt is an interrupt that is triggered using a special machine instruction, rather than a hardware line. Intel machines, for example, use the INT command to trigger software interrupts:

INT <interrupt number>

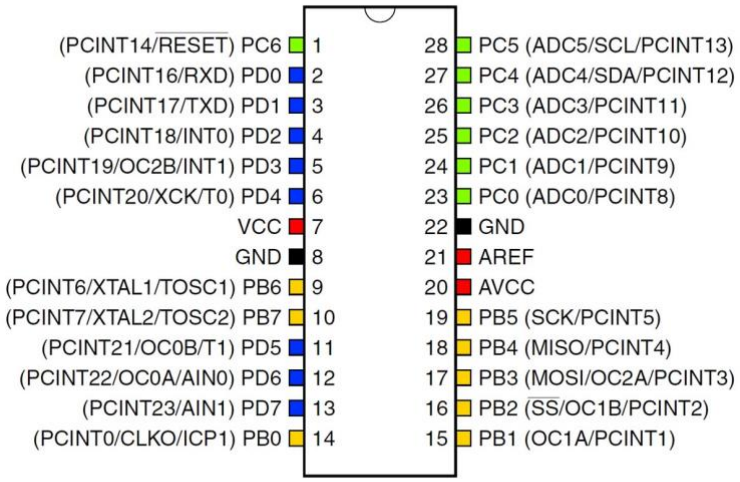
Here <interrupt number> is an interrupt identification number (analogous to the interrupt request lines in hardware interrupts), ranging from 0 to 255. Like hardware interrupts, software interrupts cause the CPU to jump to an ISR, with a different ISR for each of the 256 possible software interrupts. The address of each ISR is flexible and the CPU can be configured to jump to the correct ISR.

Software interrupts are very useful for providing flexible entry points for software APIs (application programming interfaces).

For example, on the LINUX operating system, if a program wishes to make a request to the OS (e.g. to write to the screen), it can do this by executing INT 128 (INT 80h, if you understand hexadecimal). The flexibility in setting the ISR's address for INT 128 means that it is possible for LINUX programmers to change the entry point of the OS, without affecting programs written for LINUX.

Question 5.

What interrupt request lines are available on the Atmega328P? Describe these lines and how they are.



There are two main types of interrupt request lines:

- Two external interrupt lines INT0 and INT1 (PD2 and PD3)
- 23 “pin change” interrupt request lines PCINT0 to PCINT23 (note: No PCINT15).

INT0 and INT1 are much more flexible. They can be triggered by:

- A low signal level
- A change in signal level
- Rising edge
- Falling edge

In addition INT0 and INT1 each have their own ISR.

The pin change interrupt requests (PCINT0 to PCINT23) only respond to changes in voltage levels. In addition, the 23 PCINT lines are grouped into 2 groups of 8 lines each and one group of 7 lines. All lines in the same group will trigger the execution of the same ISR. I.e. there are only 3 unique ISRs even though there are 23 lines.

Question 6.

Discuss used how hardware interrupts are implemented. In particular, talk about how hardware interrupts are detected, how the CPU/MCU decides with ISR to run, how control is handed over to the ISR, and how execution resumes normally after the ISR has completed execution.

Detection:

- Every interrupt request line is assigned an index number.
- The CPU tests the state of the interrupt request lines many times a second. (On the MIPS range of microprocessors, for example, the CPU tests the lines at the end of each instruction execution).
- When one line is detected to have been triggered, the CPU takes note of its index number.
- The CPU consults a table called the Interrupt Vector Table using the index number of the triggered line. This table tells the CPU where the ISR for this particular line is. (This table is set up at power-up with the correct ISR addresses. Like-wise the ISRs are actually loaded into the addresses indicated. This is usually done by the OS)
- The CPU saves the contents of the Program Counter (PC) onto the "process stack" – a data structure similar to what you have learnt in CS2040C, which tells the CPU where to get the next program instruction for execution.
- The CPU loads the address of the ISR into PC, causing it to execute the ISR code.
- The ISR ends with a Return from Interrupt (RETI) instruction, that causes the CPU pop the stack containing the previous PC value into PC. This causes execution to resume at the point of interruption.

Nested interrupts:

- What if an interrupt occurs while processing another interrupt?
    - Interrupts have priority levels.
    - If the new interrupt has a lower priority than the current interrupt, it is ignored until processing for the current interrupt completes.
    - If the new interrupt has a higher priority than the current one, PC is again saved on the process stack, and the vector for the new interrupt is loaded into PC, causing it to execute the new ISR.
    - When the new interrupt exits, PC is popped off the stack, causing the previous ISR to resume.

Part III – PWM

Question 7.
We saw that the AT328p has 3 Timers capable of generating 6 individual PWM signals. Consider the following scenarios and describe how you will be able to resolve the challenges.

a. Describe a SW approach to generate a PWM signal that is not dependent on the HW PWM peripheral block within the microcontroller. What is the drawback of generating the PWM using this approach?

   Answer:

   Pseudo-Code:

   1. Configure Pin as output
   2. Set as '1'
   3. Delay for 'T-on'
   4. Set as '0'
   5. Delay for 'T-off'
   6. Loop to Step 2.

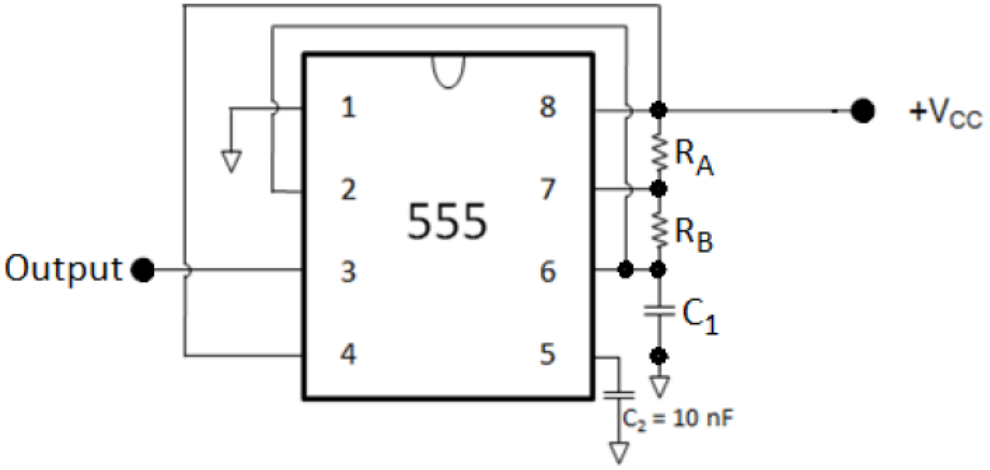
   With this approach, the microcontroller is continuously executing the loop to generate the required PWM. As such, it cant do anything else more productive. Furthermore, if there are interrupts in the system, the timing characteristics for the PWM will also be affected.

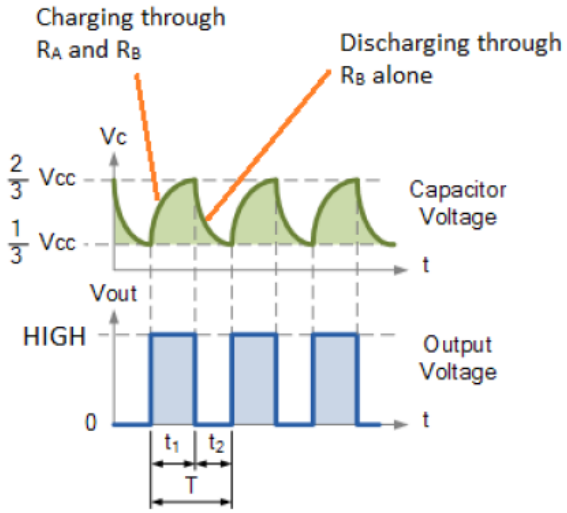
   When the peripheral block is used to generate the PWM, only the initial setup code needs to be executed. Subsequently, it is driven by the HW with minimal processing time in the Interrupts.

b. The microcontroller that you have to use for a particular project (not the AT328p) doesn't have any PWM module. How can you still generate a PWM signal WITHOUT relying on the software-based approach that you did Part A?
   (Recall HW-based solutions from EPP1)

   Answer:

   One possible approach is to use the 555 Timer to generate the required PWM signal. The PWM and the Duty Cycle can be varied by adjusting the Ra and Rb value of the circuit.





* You have learnt this in EPP1. You can refer to
Week6 Studio1 from EPP1 to refresh your memory.

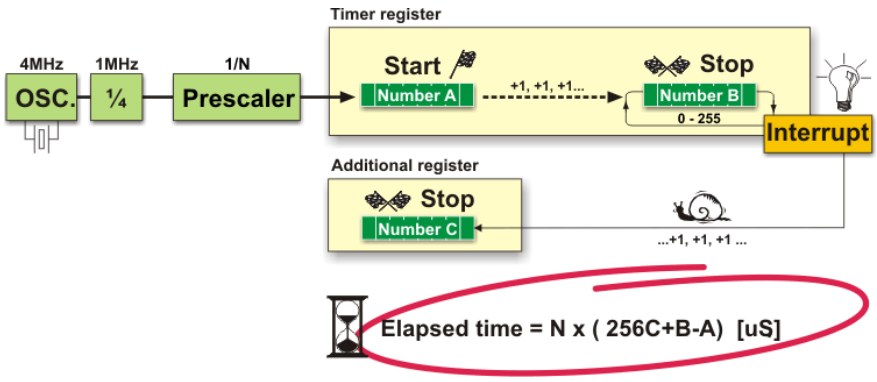
c. You feel that the approach in the earlier part is going to add additional cost to your project and you decide that stick with the AT328P. You require a very low-frequency PWM signal and even with the largest pre-scaler setting, the period is still too high. How will you be able to generate a PWM signal with the required period?

Propose at least **TWO** different approaches that can involve both HW and SW.

Answer:

Option A:

A software approach would be to use the ISR to keep track of the number of Interrupts and then toggle the output pin accordingly. In this approach, we cannot allow the PWM block to control the pin directly. The ISR needs to handle it.



Option B:

Consider using a higher-bit timer. In the AT328P, we have Timer 1, which is a 16-bit timer/counter.

Question 8.

In the studio, you were driving a single motor. If we were to drive both Motors concurrently, one possibility is to use both Timer 0 and 2. Assume that both Timer blocks are configured to use Interrupts and that the microcontroller was executing code in your loop() routine prior to the interrupts.

a. If both these interrupts were to be triggered at the same time, what would be the sequence of execution?

Answer:

b. If Timer 0 triggered the interrupt just before Timer 2, what would be the sequence of execution?

Answer:

By default, Nested Interrupts are disabled. When Timer 0 triggers the interrupt first, the controller will jump to Timer 0 ISR and start executing. When Timer 2 triggers, the event is captured by the flag, but the current ISR will complete execution and perform a context switch to the loop(). At this time, the flag will cause the jump to the Timer 2 ISR

DISCUSSION POINT:

What if nested interrupts were enabled?