**Part 1 – Library GPIO**

1. The LED on the Arduino blinks at the expected frequency of 1 Hz, which equates to about one second per second. This is correct as the delay is 1000 milliseconds in the code, otherwise 1 second.

A picture containing diagram

Description automatically generated

*Figure 1*

1. . It would approximatively be 0.5 Hz (Figure 1) rather than 1 Hz as from earlier thoughts.

A picture containing graphical user interface

Description automatically generated

*Figure 2*

1. . It would be 1 Hz (Figure 2) due to the 1000 milliseconds inputted into the delay function now being half that of 500 milliseconds.

A picture containing diagram

Description automatically generated’

*Figure 3*

1. . The frequency when the library function has been inputted with 1 millisecond results in a 1002 Hz as Figure 3 shows above.

A picture containing table, sitting, computer, light

Description automatically generated

*Figure 4*

1. (See Figure 4)
2. Within this configuration, the LED is active-high (Figure 4).

A wooden table

Description automatically generated

*Figure 5*

1. Within this configuration, the LED is active-low (Figure 5).

Graphical user interface, text, application

Description automatically generated

**Part 2: Register-Level Output**

**A picture containing diagram

Description automatically generated**

*Figure 6*

1. . (Figure 6)
2. The functions of the three lines of code in the first lines of the program initialize the program to use and manipulate the register at Port B by defining their memory addresses. One aspect of accessing these registers and having them ready for use is defining if that register will be used for inputting or outputting data. This defined via the Data Direction Register (DDR). In this case, the first line within setup function is bit masking the register to have data “flow” as output data.
3. Port Registers act as communication areas for the microcontroller. Mainly, they provide data on manipulating input/output through bit masking. As it can be seen, a method to bit mask and modify the bits within these port registers is to use operations such as “&” and “|.” In this case, the “&” is clearing the bits in the register making them zero while the “|” operator sets bits. In this “|” operator is setting the bits to designate a “on” sending an active on signal.

**Part 3: Preprocessor Definitions and Function Wrapping**

1. The difference between two mainly comes down two how they bit masks registers and in the methods in the structure of this bit masking. When structure is mentioned, mainly what is being referenced is the use of macros in “example\_2” with the “WRITE\_HIGH\_PB(pin\_num)” macro. As for the bit-masking aspect, example\_2 outright within its macro manipulates the bits by conducting a shift to the left on a specific pin. This is far different case for “example\_1” which does outright bit masking via the use of “and” and “or” operators for bit manipulation.
2. The different between the two mainly comes down to the use of native C functions to carry out bit manipulation while “example\_2” uses C macros to create defined text to bit manipulate registers. Specifically, from nearly start to finish “example\_3” deploys use of C functions to not only flip the states of the register from on and off (the bit shifts are nested within the C functions), it also deploys functions to shift the Data Direction Register also. This is a far cry from “example\_2” as it does DDR setting through just regular bit masking.
3. All programs use 858 Bytes of ROM and use 9 Bytes of RAM.
4. Readability between the three programs are different in their one way. Each being more readable as it goes with “example\_1” being the least readable while “example\_3” is the most readable program. The reason why due to the different is the more abstraction taking place within each iteration.

**Part 4: Register-Level Input**

**A picture containing table

Description automatically generated**

*Figure 7*

1. The reason for the pull resistor on PB4 is due to the possibility that input maybe not captured easily if the line is turned off by default. An active signal would be difficult to capture due to the user may let go of the button too fast but with the behavior being a pull-up instead if a pull-down this may not occur frequently. It may not occur frequently as an open-circuit signal among of always active signal tests when looping are easier to defernite. (Figure 7)

Graphical user interface, text, application

Description automatically generated

*Figure 8*

1. The input if in this configuration is not reliable. Within this pull-down configuration, user input either is not captured is not captured but for far too long. This results in the non-user intended input into the program. During this state, the pin is in a “buffer” state which results in user input being delayed.
2. . The program checks for user input every 250 milliseconds, this results in a 4 Hz Sampling Rate.

**Part 5: Putting Everything Together**

**Text

Description automatically generated**

*Figure 9*

*A circuit board on a table

Description automatically generated*

*Figure 10*

*A picture containing graphical user interface

Description automatically generated*

*Figure 11*

1. See Figure 11 (NOTE: Teal is PD0 and Yellow is PK2). This delay demonstrates the program very slight overhead in terms of the “if” loop having an effect on continuously checking the peripheral input register.

A picture containing chart

Description automatically generated

*Figure 12*

A picture containing chart

Description automatically generated

*Figure 13*

1. See Figure 12 and Figure 13 (NOTE: Teal is PD0 and Yellow is PE3).