1. (5 pts each) Complete the following table by writing in values (or updating values) for variables changed by the expression in the “Operation” column. Assume all variables have datatype int8. Perform each operation independent from the others; that is do not carry the results of one operation into the next row of the table. If a value is updated, then strikethrough the value and write-in a new value. If an empty cell does not change value, leave the cell empty.

<table>
<thead>
<tr>
<th>Operation</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>(z = x + y;)</td>
<td>100</td>
<td>120</td>
<td>220</td>
</tr>
<tr>
<td>(z += x;)</td>
<td>strikethrough 100</td>
<td>2</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>200-300-256 = 44</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Values and answers in binary (space put between every four bits for readability)

<table>
<thead>
<tr>
<th>Operation</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>(z = x &gt;&gt; 3;)</td>
<td>0b1001 1010</td>
<td>0b0001 0011</td>
<td>0b0001 0011</td>
</tr>
<tr>
<td>(z = \sim x;)</td>
<td>0b1001 1010</td>
<td>0b1100 1001</td>
<td>0b0110 0101</td>
</tr>
</tbody>
</table>

2. (5 pts) Write a single line of C-code that toggles bit 3 of an 8-bit variable \(x\). Store the result in an 8-bit variable \(z\). Index bits starting at 0.

\[z = x ^ {0b0000 1000};\]

3. (5 pts) Write a single line of C-code that sets bit 3 and clears bit 5 of an 8-bit variable \(x\). Store the result in an 8-bit variable \(z\). Index bits starting at 0.

\[z = (x | 0b0000 1000) & 0b1101 1111; \quad \text{// Order is important, clear last!}\]

4. (10 pts each) Given the following C-code, find the value of \(a\) after executing the code with the value of \(x\), \(y\), \(z\) given in each row of the table.

```c
if (((x > 10) && (y <= 20) && (z < 40)) {
    a = 0;
} else if (((x <= 10) && (y != 30) && (z == 20)) {
    a = 1;
} else if (((x >= 15) || (y != 15)) {
    a = 2;
} else {
    a = 3;
} // end if
```

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>10</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>20</td>
<td>30</td>
<td>0</td>
</tr>
</tbody>
</table>

5. (10 pts) Write a single line of C-code that waits (does nothing) while an active-high push button on PTT_PTT0 is not pressed.

```c
while(PTT_PTT0 == 0);
```
6. (10 pts) Write the function prototype for \texttt{minimum}, a function that takes as input a 1-D array of int8’s called \texttt{array} and the length of the array \texttt{length}. The array has less than 100 elements. The function returns the smallest element in the array.

\begin{verbatim}
int8 minimum(int8 array[], int8 length);
\end{verbatim}

7. (10 pts) Write a valid line of C-code that illuminates the LED using the register and bit-field convention in the mc9s12c32.h file.

\texttt{PTT_PTT0 = 0;}

8. (10 pts) Write a valid line of C-code that configures the LED pin using the register and bit-field naming convention in the mc9s12c32.h file.

\texttt{DDR_DDRT0 = 1;}

---

Use the following C-code snippet to answer questions 9-11.

1. \texttt{TSCR2\_PR = 0bxxx;}
2. \texttt{TIOS\_IOS0 = 1;}
3. \texttt{TC0 = TCNT + delayValue;}
4. \texttt{TFLG1 = TFLG1\_C0F\_MASK;}
5. \texttt{while (TFLG1\_C0F == 0);}

9. (10 pts) What prescaler will create the most accurate 25 ms delay in line 4?
A 16:1 prescaler is the smallest prescaler with a maximum period over 25 ms with a maximum period of 43.7 ms.

10. (10 pts) With the prescaler from question #10, what \texttt{delayValue} generates a 25 ms delay in line 5? State your answer using dimensional analysis.

\begin{verbatim}
1 second  1000 ms  16 clk
------------ * -------- * ------- * x counts = 25 ms  x = 37,500
24*10^6 clks  1 second  1 count
\end{verbatim}

11. (10 pts) What delay, in milliseconds, is generated in line 5 with \texttt{TSCR2\_PR = 0b101} and \texttt{delayValue=30,000}? State your answer using dimensional analysis.

\begin{verbatim}
1 second  1000 ms  32 clk
------------ * -------- * ------- * 30,000 counts = 40 ms
24*10^6 clks  1 second  1 count
\end{verbatim}
12. You are building a device that measures the speed of individual alpha particles using a detector loop attached to PTT_PTT4. The detector loop outputs logic-1 while the alpha particle is inside the detector loop. The detector loop is 5 cm wide. Thus, the duration of the logic-1 pulse on PTT_PTT4 is equal to the amount of time that it takes the alpha particle to travel 5 cm. Alpha particles move fast, so the timer prescaler is initialized to 1:1.

a) (10 pts) Use dimensional analysis to convert the pulse duration (measured in timer counts) on PTT_PTT4 into the velocity of the alpha particle in meters per second. You can assume that the pulse duration is stored in a variable called x.

\[
\frac{24 \times 10^6 \text{ clks}}{1 \text{ second}} \times \frac{1 \text{ count}}{1 \text{ clk}} \times \frac{1 \text{ flight}}{x \text{ counts}} \times \frac{5 \text{ cm}}{1 \text{ flight}} \times \frac{1 \text{ m}}{100 \text{ cm}} = \frac{1,200,000}{x} \text{ m/s}
\]

b) (40 pts) Write a C-code to measure the pulse width (in timer counts) using the capture subsystem inside the infinite loop and store the pulse width in the variable x. Assume that the timer subsystem is initialized for you – just write the initialization code for the capture subsystem. Declare any additional variables at the top of your program.

```c
int16 x;
int16 start, end;

for (;;) {
    TCTL3_EDG4x = 0b01;  // Capture on rising edge of PTT_PTT4 - optional
    TFLG1 = TFLG1_C4F_MASK; // Clear flag
    while(TFLG1_C4F == 0); // Wait for rising edge on PTT_PTT4
    start = TC4; //

    TCTL3_EDG4x = 0b10;  // Capture on falling edge of PTT_PTT4 - optional
    TFLG1 = TFLG1_C4F_MASK; // Clear flag
    while(TFLG1_C4F == 0); // Wait for falling edge of PTT_PTT4
    end = TC4;

    x = end - start; // compute pulse width
} // end infinite loop
```