(4) Question 1. You are working for an engineering firm and have been given the task to design a new DAC. After completing the design and construction of an initial prototype, you now have to test it. List four experimental parameters you will measure to evaluate the quality of your new DAC. These parameters will be measurable with test equipment, like you used in lab or like the professor/TAs demonstrated to you in this class. For each parameter, give both the name of the parameter and the definition of the parameter. You do not need to give the experimental procedure, rather simply list four names and four definitions of the parameters. I do not mean physical parameters like size, weight, or color. Furthermore, I do not mean economic factors like production costs, profit, mean time between failures, or maintenance costs.

Part a) Give a parameter and its definition

Part b) Give a parameter and its definition

Part c) Give a parameter and its definition

Part d) Give a parameter and its definition
(6) Question 2. What is the voltage $V$?

\[ \begin{array}{c}
+5 \\
\downarrow \\
20 \text{k}\Omega \\
\downarrow \\
10 \text{k}\Omega \\
\downarrow \\
10 \text{k}\Omega \\
\downarrow \\
- \\
\end{array} \]

(10) Question 3. Write a C function to sample ADC channels 3 and 4 of the 9S12DP512. Assume the ADC initialized for a 10-bit sample, sequence length is 2, and the ADC clock is 1 MHz. Implement right-justified conversions, and return by value the larger of the two ADC samples.
(15) **Question 4.** Write three C functions that operate the SCI0 module.

(5) **Part a)** Write an initialization function that turns on both the transmitter and receiver for the SCI0. Set the baud rate equal to 38400 bits/sec. You may assume the E clock is 8 MHz.

(5) **Part b)** Write a C function that outputs one ASCII character to SCI0 using busy-wait synchronization and call by value parameter passing.

(5) **Part c)** Write a C function that outputs an ASCII string to the SCI0. You may assume the variable length string is null-terminated. Use call by reference parameter passing.
(5) **Question 5.** Write assembly code that implements the unsigned operation \( \text{RegD} = 0.3456 \times \text{RegX} \), eliminating overflow and minimizing dropout errors.

(5) **Question 6.** Consider the result of executing the following two 9S12 assembly instructions.

\[
\text{ldaa } #156 \\
\text{suba } #-50
\]

What will be the value of the carry (C) bit?


What will be the value of the overflow (V) bit?


(5) **Question 7.** Give the simplified memory cycles produced when the following one instruction is executed. Assume the PC contains $4200$, and the SP equals $3FF2$. Just show R/W=Read or Write, Address, and Data for each cycle. You may not need all 5 entries in the solution box.

\[
\text{bstr } #4210
\]

<table>
<thead>
<tr>
<th>R/W</th>
<th>Addr</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(5) Question 8. Consider a serial port operating with a baud rate of 10,000 bits per second. Draw the waveform occurring at the PS1 output (voltage levels are +5 and 0) when the ASCII ‘a’ ($61$) is transmitted on SCI0. The protocol is 1 start, 8 data and 1 stop bit. SCI0 is initially idle, and the software writes the $61$ to SCI0DRL at time=0. Show the PS1 line before and after the frame, assuming the channel is idle before and after the frame.

![PS1 waveform diagram]

(5) Question 9. For each application choose the data structure that best matches. Choose answers from the word bank.

**Word bank:** first in first out queue, buffer, last in first out stack, linked list, table

<table>
<thead>
<tr>
<th>Application</th>
<th>Data structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>A data structure used to implement buffered I/O.</td>
<td>first in first out queue, buffer, last in first out stack</td>
</tr>
<tr>
<td>A data structure used to store calibration data</td>
<td>buffer</td>
</tr>
<tr>
<td>A data structure used to implement a local variables</td>
<td>table</td>
</tr>
<tr>
<td>A data structure used to implement a debugging dump</td>
<td>linked list</td>
</tr>
<tr>
<td>A data structure used to implement a Mealy machine</td>
<td>buffer</td>
</tr>
</tbody>
</table>

(20) Question 10. Write assembly code to implement the following FSM using output compare interrupts. After initialization, the entire FSM will run within the output compare 0 ISR. The command sequence will be output, wait 1ms, input, then branch to next state. There can be NO backward jumps or conditional branches within the ISR. The 2-bit input is on Port H (PH1 and PH0) and the 3-bit output is on Port T (PT2, PT1, PT0). Assume the E clock is 8 MHz. You may use a 16-bit pointer named $\text{Pt}$, located in RAM.

![FSM diagram]
(5) Part a) Show the assembly code defining the FSM graph in ROM.

(7) Part b) Show the assembly subroutine to initialize the FSM controller, arm OC0, and enable interrupts. Please initialize the state pointer Pt and perform the first output.

(8) Part c) Show the assembly language for the output compare 0 ISR that runs the FSM including interrupt vector.
(10) **Question 11.** In this question, you will translate the C code line by line into 9S12 assembly. The assembly code for the global variables and main are given. The two parameters are passed into `Add` using call by reference, and both MUST be pushed on the stack. You must use SP-relative binding for the `p1` and `p2` parameters within the `Add` function.

<table>
<thead>
<tr>
<th>C Code</th>
<th>Assembly Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>short D1;</td>
<td>org $0800</td>
</tr>
<tr>
<td>short D2;</td>
<td>D1 rmb 2</td>
</tr>
<tr>
<td>void main(void){</td>
<td>D2 rmb 2</td>
</tr>
<tr>
<td>D1 = -1000;</td>
<td>org $4000</td>
</tr>
<tr>
<td>D2 = +500;</td>
<td>main lds #$4000</td>
</tr>
<tr>
<td>for(;;){</td>
<td>movw #-1000,D1</td>
</tr>
<tr>
<td>Run();</td>
<td>movw #500,D2</td>
</tr>
<tr>
<td>}</td>
<td>loop bsr Run</td>
</tr>
<tr>
<td>}</td>
<td>bra loop</td>
</tr>
</tbody>
</table>

```c
void Run(void){
    Add(&D1,&D2);
}
```

```c
void Add(short *p1,
short *p2){
    p1 set ___________ ;SP relative binding
    p2 set ___________ ;SP relative binding
    Add
    *p2 = *p2 + *p1;
}
```
(10) Question 12. For each definition, fill in the term that best matches. Choose answers from the word bank. Some words will not be used.

**Word bank:** accuracy, arm, baud rate, acknowledge, bandwidth, big endian, ceiling, disarm, drop out, enable, friendly, intrusive, little endian, nonintrusive, noninvasive, nonvolatile, open collector, overflow, real-time, resolution, stabilization, thread, tristate, volatile

<table>
<thead>
<tr>
<th>Definition</th>
<th>Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate an individual trigger so that interrupts are requested when that trigger flag is set.</td>
<td></td>
</tr>
<tr>
<td>The information transfer rate, the amount of real data transferred per second.</td>
<td></td>
</tr>
<tr>
<td>A measure of how close our instrument measures the desired parameter referred to the NIST.</td>
<td></td>
</tr>
<tr>
<td>A debugging process that fixes all the inputs to a system</td>
<td></td>
</tr>
<tr>
<td>Mechanism for storing multiple byte numbers such that the most significant byte exists first</td>
<td></td>
</tr>
<tr>
<td>Establishing an upper bound on the result of an operation.</td>
<td></td>
</tr>
<tr>
<td>Clearing the interrupt trigger flag that requested the interrupt.</td>
<td></td>
</tr>
<tr>
<td>An error that occurs after a right shift or a divide, losing its ability to represent all of the values.</td>
<td></td>
</tr>
<tr>
<td>A characteristic when the presence of the collection of information itself does not affect the parameters being measured.</td>
<td></td>
</tr>
<tr>
<td>A digital logic output that has two states low and HiZ.</td>
<td></td>
</tr>
</tbody>
</table>
movw 16-bit move memory to memory
mul unsigned RegD=RegA*RegB
neg 8-bit 2's complement negate memory
negb 8-bit 2's complement negate RegB
ora 8-bit logical or to RegA
orab 8-bit logical or to RegB
orcc 8-bit logical or to RegCC
psha push 8-bit RegA onto stack
psb 8-bit push RegB onto stack
psch push 8-bit RegCC onto stack
psdh push 16-bit RegD onto stack
pshx push 16-bit RegX onto stack
pshy push 16-bit RegY onto stack
pula pop 8 bits off stack into RegA
pulb pop 8 bits off stack into RegB
pulc pop 8 bits off stack into RegCC
puld pop 16 bits off stack into RegD
pulx pop 16 bits off stack into RegX
puly pop 16 bits off stack into RegY
rev Fuzzy logic rule evaluation
revv weighted Fuzzy rule evaluation
rol 8-bit roll shift left Memory
rola 8-bit roll shift left RegA
rolb 8-bit roll shift left RegB
ror 8-bit roll shift right Memory
rorc 8-bit roll shift right RegA
rorb 8-bit roll shift right RegB
rtc return in expanded memory
rti return from interrupt
rts return from subroutine
sba 8-bit subtract RegA=RegA-RegB
sbca 8-bit subtract with carry from RegA
sec set carry bit, C=1
sei set I=1, disable interrupts
sev set overflow bit, V=1
sex sign extend 8-bit to 16-bit reg
staa 8-bit store memory from RegA
stab 8-bit store memory from RegB
std 16-bit store memory from RegD
sts 16-bit store memory from SP
stx 16-bit store memory from RegX
styx 16-bit store memory from RegY
suba 8-bit sub from RegA
subb 8-bit sub from RegB
subd 16-bit sub from RegD
swi software interrupt, trap
tab transfer A to B
tap transfer A to CC
tba transfer B to A
tbeq test and branch if result=0
thl 8-bit look up and interpolation
tbne test and branch if result≠0
trf transfer register to register
tpa transfer CC to A
trap illegal instruction interrupt
trap illegal op code, or software trap
tst 8-bit compare memory with zero
tsta 8-bit compare RegA with zero
tstb 8-bit compare RegB with zero
tsx transfer S to X
tsy transfer S to Y
txs transfer X to S
tys transfer Y to S
wait for interrupt
weighted Fuzzy logic average

Example | Mode | Effective Address
--- | --- | ---
idaa #u | immediate | No EA
idaa u | direct | EA is 8-bit address
idaa U | extended | EA is a 16-bit address
idaa m,r | 5-bit index | EA=r+m (-16 to 15)
idaa v,+r | pre-inc | r=r+v, EA=r (1 to 8)
idaa v,-r | pre-dec | r=r-v, EA=r (1 to 8)
idaa v,r+ | post-inc | EA=r, r=r+v (1 to 8)
idaa v,r- | post-dec | EA=r, r=r-v (1 to 8)
idaa A,r | Reg A offset | EA=r+A, zero padded
idaa B,r | Reg B offset | EA=r+B, zero padded
idaa D,r | Reg D offset | EA=r+D
idaa q,r | 9-bit index | EA=r+q
idaa W,r | 16-bit index | EA=r+W
idaa [D,r] | D indirect | EA={r+D}
idaa [W,r] | indirect | EA={r+W}

Freescale 6812 addressing modes: r is X, Y, SF, or PC

Pseudo op Meaning
set Where to put subsequent code
equ Define a constant symbol
byte Allocate byte(s) with values
long Allocate 32-bit with values
word Allocate word(s) without init
file Allocate word(s) without init

n is Metrowerks number

Vector | n | Interrupt Source | Arm
--- | --- | --- | ---
FFFF | reset | None
FFFF8 | Trap | None
FFFF6 | SWI | None
FFFF0 | Real time interrupt | CRGINT.RTIE
FFFFE | Timer channel 0 | TIE.C0I
FFFFC | Timer channel 1 | TIE.C1I
FFFFA | Timer channel 2 | TIE.C2I
FFFF8 | Timer channel 3 | TIE.C3I
FFFF6 | Timer channel 4 | TIE.C4I
FFFF4 | Timer channel 5 | TIE.C5I
FFFF2 | Timer channel 6 | TIE.C6I
FFFF0 | Timer channel 7 | TIE.C7I
FFFFD | Timer overflow | TSCR2.TO1
FFFD6 | SCI0 TDRE, RDRF | SCI0CR2.TIE.RIE
FFFD4 | SCI1 TDRE, RDRF | SCI1CR2.TIE.RIE
FFFCCE | Key Wakeup J | PIEJ.[7,6,1,0]
FFFC | Key Wakeup H | PIEH.[7:0]
FFFE | Key Wakeup P | PIEP.[7:0]

Interrupt Vectors and interrupt number.
<table>
<thead>
<tr>
<th>Address</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0040$</td>
<td>IOS7</td>
<td>IOS6</td>
<td>IOS5</td>
<td>IOS4</td>
<td>IOS3</td>
<td>IOS2</td>
<td>IOS1</td>
<td>IOS0</td>
<td>TIOS</td>
</tr>
<tr>
<td>$0044$</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>TCNT</td>
</tr>
<tr>
<td>$0046$</td>
<td>TEN</td>
<td>TSWAI</td>
<td>TSFRZ</td>
<td>TFFCA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>TSCR1</td>
</tr>
<tr>
<td>$004C$</td>
<td>C7I</td>
<td>C5I</td>
<td>C4I</td>
<td>C3I</td>
<td>C2I</td>
<td>C1I</td>
<td>C0I</td>
<td>0</td>
<td>C1F</td>
</tr>
<tr>
<td>$004D$</td>
<td>TOI</td>
<td>PUPT</td>
<td>RDPT</td>
<td>TCIE</td>
<td>PR2</td>
<td>PR1</td>
<td>PR0</td>
<td>0</td>
<td>TSCR2</td>
</tr>
<tr>
<td>$004E$</td>
<td>C7F</td>
<td>C5F</td>
<td>C4F</td>
<td>C3F</td>
<td>C2F</td>
<td>C1F</td>
<td>C0F</td>
<td>0</td>
<td>TFLG1</td>
</tr>
<tr>
<td>$004F$</td>
<td>TOF</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>TFLG2</td>
</tr>
<tr>
<td>$0050$</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>TC0</td>
</tr>
<tr>
<td>$0052$</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>TC1</td>
</tr>
<tr>
<td>$0054$</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>TC2</td>
</tr>
<tr>
<td>$0056$</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>TC3</td>
</tr>
<tr>
<td>$0058$</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>TC4</td>
</tr>
<tr>
<td>$005A$</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>TC5</td>
</tr>
<tr>
<td>$005C$</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>TC6</td>
</tr>
<tr>
<td>$005E$</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>TC7</td>
</tr>
</tbody>
</table>

**TSCR1** is the first 8-bit timer control register  
bit 7 **TEN**, 1 allows the timer to function normally, 0 means disable timer including **TCNT**  
**TIOS** is the 8-bit output compare select register, one bit for each channel (1 = output compare, 0 = input capture)  
**TIE** is the 8-bit output compare arm register, one bit for each channel (1 = armed, 0 = disarmed)
**TSCR2** is the second 8-bit timer control register

bits 2, 1, 0 are **PR2, PR1, PR0**, which select the rate, let \( n \) be the 3-bit number formed by **PR2, PR1, PR0**

- without PLL **TCNT** is \( 8 \text{MHz}/2^n \), with PLL **TCNT** is \( 24 \text{MHz}/2^n \), \( n \) ranges from 0 to 7

<table>
<thead>
<tr>
<th>PR2</th>
<th>PR1</th>
<th>PR0</th>
<th>Divide by</th>
<th>( E = 8 \text{MHz} )</th>
<th>( E = 24 \text{MHz} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TCNT period</td>
<td>TCNT frequency</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>125 ns</td>
<td>8 MHz</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>250 ns</td>
<td>4 MHz</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>500 ns</td>
<td>2 MHz</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1 ( \mu )s</td>
<td>1 MHz</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>2 ( \mu )s</td>
<td>500 kHz</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>32</td>
<td>4 ( \mu )s</td>
<td>250 kHz</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>64</td>
<td>8 ( \mu )s</td>
<td>125 kHz</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>128</td>
<td>16 ( \mu )s</td>
<td>62.5 kHz</td>
</tr>
</tbody>
</table>

**SCI0DRL** 8-bit SCI0 data register

**SCI0BD** is 16-bit SCI0 baud rate register, let \( n \) be the 13-bit number Baud rate is EClk/\( n/16 \)

**SCI0CR1** is 8-bit SCI0 control register

- bit 4 M, Mode, 0 = One start, eight data, one stop bit, 1 = One start, eight data, ninth data, one stop bit

**SCI0CR2** is 8-bit SCI0 control register

- bit 7 TIE, Transmit Interrupt Enable, 0 = TDRE interrupts disabled, 1 = interrupt whenever TDRE set
- bit 5 RIE, Receiver Interrupt Enable, 0 = RDRF interrupts disabled, 1 = interrupt whenever RDRF set
- bit 3 TE, Transmitter Enable, 0 = Transmitter disabled, 1 = SCI transmit logic is enabled
- bit 2 RE, Receiver Enable, 0 = Receiver disabled, 1 = Enables the SCI receive circuitry.

**SCI0SR1** is 8-bit SCI0 status register

- bit 7 TDRE, Transmit Data Register Empty Flag
  - Set if transmit data can be written to **SCI0DRL**
  - Cleared by **SCI0SR1** read with TDRE set followed by **SCI0DRL** write
- bit 5 RDRF, Receive Data Register Full
  - Set if a received character is ready to be read from **SCI0DRL**
  - Clear the RDRF flag by reading **SCI0SR1** with RDRF set and then reading **SCI0DRL**

---

**EMUL**

Extended Multiply 16-Bit by 16-Bit (Unsigned)

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMUL</td>
<td>INH</td>
<td>13</td>
</tr>
</tbody>
</table>

**EDIV**

Extended Divide 32-Bit by 16-Bit (Unsigned)

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDIV</td>
<td>INH</td>
<td>11</td>
</tr>
</tbody>
</table>

**BSR**

Branch to Subroutine

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSR rel0</td>
<td>REL</td>
<td>07 xx</td>
</tr>
</tbody>
</table>

**RTS**

Return from Subroutine

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Address Mode</th>
<th>Object Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTS</td>
<td>INH</td>
<td>3D</td>
</tr>
</tbody>
</table>