Please read and affirm our honor code:
“The core values of The University of Texas at Austin are learning, discovery, freedom, leadership, individual opportunity, and responsibility. Each member of the university is expected to uphold these values through integrity, honesty, trust, fairness, and respect toward peers and community.”

**Question 1.** What is the voltage at V? Show your work.

![Circuit Diagram]

**Question 2.** The PLL is not active, and the E clock frequency is 8 MHz. The TCNT timer is active with TSCR2 equal to 2. The ADC is active with ATD0CTR4=03, so that the ADC clock is 1 MHz. The SCI1 baud rate is 9600 bits/sec. Output compare 7 is used to sample the 10-bit ADC once every interrupt. The data is stored in a buffer. The following code occurs in the output compare 7 ISR:

\[
TC7 = TC7 + 100;
\]

(3) **Part a)** What is the time period between output compare 7 interrupts?

(2) **Part b)** What is the largest frequency component faithfully represented in the data in the buffer?
(10) **Question 3.** Initialize PTT so that PT1 is an input and PT0 is an output. Write a C program that issues a short pulse on the PT0 output after every other each rising edge of the PT1 input.

Include *all the software* for this system. You may use the standard port names, such as PTT. Interrupts and the timer are not needed. It does not matter how long the pulse width is on PT0.
(20) **Question 4.** You can solve this problem in assembly or in C, your choice. This is a simplex communication channel; there is SCI1 output, but no SCI1 input. In C, the user will call your function by reference. In assembly, the user calls your function by reference using Reg D

```c
void user(void)
{
    SCI1_Output("Hello");
}
```

Your `SCI1_Output` includes SCI1 initialization and output. The data is null-terminated. You may assume the E clock is 8 MHz. The desired baud rate is 5000 Hz. You are allowed to add additional RAM-based variables. Write the `SCI1_Output` function, the SCI1 interrupt service routine, and the code to set the SCI1 interrupt vector. Disarm SCI1 interrupts after the last character is sent. You do not have to transmit the null character. For full credit you must implement the SCI1 output using interrupt synchronization. For a possible 5 points out of 20, you solve this problem with busy-wait.
(6) **Question 5.** Your embedded system uses a 12-bit ADC to sample the fluid flow through an oil pipe. The system will display flow rate on an LCD. The flow can vary from -20 to +20 L/min. The appropriate transducer and analog circuit maps in a linear fashion the full scale flow rate into the 0 to +5V full scale range of the ADC. I.e., -20 L/min maps to 0V and +20 L/min maps to +5V. The ADC uses straight binary: 0V maps to 0 and +5V maps to the largest digital output.

(2) **Part a)** What ADC value do you get if the flow rate is -10 L/min.

(2) **Part b)** What resolution will you use for fixed-point number system to store the oil flow data?

(2) **Part c)** What precision will you use for fixed-point number system to store the oil flow data?

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

```
ldaa #1
adda #-1
```

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


What will be the value of the zero (Z) bit?


(5) **Question 7.** Give the simplified memory cycles produced when the following one instruction is executed. Initially, PC is $4065$, \textbf{DDRT} is $FF$, and \textbf{PTT} is $08$. Just show R/W=Read or Write, Address, and Data for each cycle. You may need more or less entries than the 5 boxes given.

$4065 \text{ 1C024001} \text{ bset PTT, #01}$

<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.** For each activity choose the debugging term that \textit{best} matches. Choose answers from the word bank. Not all words in the bank will be used. No word is used twice.

**Word bank:** heartbeat, breakpoint, drop out, scanpoint, friendly, profile, minimally intrusive, highly intrusive, bandwidth, monitor, real-time, stabilization, volatile

<table>
<thead>
<tr>
<th>Activity</th>
<th>Debugging term</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measuring where and when software executes</td>
<td></td>
</tr>
<tr>
<td>Recording data during execution without pausing</td>
<td></td>
</tr>
<tr>
<td>Debugging with a small but inconsequential effect on the system itself</td>
<td></td>
</tr>
<tr>
<td>Adding a LCD to display important variables during execution; the LCD is not part of the necessary components of the system.</td>
<td></td>
</tr>
<tr>
<td>Flashing an LED letting the user know the software is running</td>
<td></td>
</tr>
</tbody>
</table>
(15) **Question 9.** Design and implement a FIFO that can hold up to 4 elements. Each element is 3 bytes. There will be three subroutines: **Initialization, Put** one element into FIFO and **Get** one element from the FIFO.

(4) **Part a)** Show the RAM-based variables are available, and NO additional storage may be allocated.

(4) **Part b)** Write an assembly function that initializes the FIFO.

(4) **Part c)** Write an assembly function that puts one 3-byte element into the FIFO. The input parameter is call by reference using Register Y. The output parameter is returned in Register D, 0 for failure because the FIFO was full, and 1 for success because the data was properly stored.

(3) **Part d)** Write an assembly function that gets one 3-byte element from the FIFO. When your Get function is called, Register Y points to an empty 3-byte RAM buffer. The output parameters are return by value in Register D and return by reference through Register Y. If data can be removed, three bytes are copied into the buffer pointed to by Register Y and Register D is returned as 1. If no data can be removed, because the FIFO was empty when the function was called, return Register D equal to 0.
(5) **Question 10.** Assume the initial state is Stop and the input is a constant value of 3. What will be the sequence of outputs?

(10) **Question 11.** Consider output compare 7 interrupts. Assume the name of the interrupt service routine is **TC7Handler**.

(3) **Part a)** What three events in general need to be true for any interrupt to occur? Furthermore, give those three events specifically for output compare 7.

(4) **Part b)** List the events that occur as the computer switches from running in the foreground to running an output compare 7 interrupt in the background? E.g., the 9S12 is running the main program, *stuff happens*, the 9S12 is running **TC7Handler**. List the events in *stuff happens*.

(3) **Part c)** Write assembly code to acknowledge an output compare 7 interrupt.
(10) Question 12. In this question, you will translate the C code into 9S12 assembly. You must use stack frame binding using Register Y for the c and d parameters within the main program. I.e., both local variables must be allocated on the stack. You may assume PTT is an input port. Use org statements to place the program at an appropriate place within the 9S12, and include the reset vector.

```c
void main(void){
    unsigned short c;
    unsigned char d;
    c = 0;
    for(;;){
        d = PTT;
        if(d&0x01){
            c = c+d;
        }
    }
}
```
aba 8-bit add RegA=RegA+RegB
abx unsigned add RegX=RegX+RegB
aby unsigned add RegY=RegY+RegB
adca 8-bit add with carry to RegA
adcb 8-bit add with carry to RegB
adda 8-bit add to RegA
addb 8-bit add to RegB
addd 16-bit add to RegD
anda 8-bit logical and to RegA
andb 8-bit logical and to RegB
andc 8-bit logical and to RegC
asl/lsl 8-bit left shift Memory
aslb/lslb 8-bit arith left shift RegB
asld/lslbd 16-bit left shift RegD
asr 8-bit arith right shift Memory
asra 8-bit arith right shift to RegA
asrb 8-bit arith right shift to RegB
bcc branch if carry clear
bcldr bit clear in memory
bcldr PTT,$01
bcs branch if carry set
beq branch if result is zero (Z=1)
beq PTT,$01
beq Y,loop
bge branch if signed ≥
bgt branch if signed >
big branch if unsigned >
big PTT,$01,loop
bita 8-bit and with RegA, sets CCR
bitb 8-bit and with RegB, sets CCR
ble branch if signed ≤
blt branch if signed <
bls branch if unsigned ≤
bls PTT,$01
bmi branch if result is negative (N=1)
bnl branch if result is nonzero (Z=0)
bnr branch if result is positive (N=0)
bra branch always
brcc branch if bits are clear
brcc PTT,$01,loop
brn branch never
brset branch if bits are set
bset bit set in memory
bsr branch to subroutine
bvc branch if overflow clear
bvs branch if overflow set
call subroutine in expanded memory
cba 8-bit compare RegA with RegB, RegA-RegB
cic clear carry bit, C=0
cil clear I=0, enable interrupts
cldr 8-bit memory
clr RegA clear
crlb RegB clear
crv clear overflow bit, V=0
cmpa 8-bit compare RegA with memory
cmpb 8-bit compare RegB with memory
com 8-bit logical complement to memory
compa 8-bit logical complement to RegA
compb 8-bit logical complement to RegB
cpd 16-bit compare RegD with memory
cpx 16-bit compare RegX with memory
cpy 16-bit compare RegY with memory
da 8-bit decrement memory
da Y,loop
db 8-bit decrement RegD
dba 8-bit decrement RegA
dec 8-bit decrement RegB
decq decrement and branch if result=0
dbeq Y,loop
dbeq A,loop
decn decrement and branch if result≠0
dbne A,loop
decr 8-bit decrement RegB
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
psa push 8-bit RegA onto stack
psab push 8-bit RegB onto stack
psac 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
revw 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
rorb 8-bit roll shift right RegB
rtc return sub in expanded memory
rti return from interrupt
rts return from subroutine
sba 8-bit subtract RegA=RegA-RegB
sbca 8-bit sub with carry from RegA
sbcb 8-bit sub with carry from RegB
sec set carry bit, C=1
sei set I=1, disable interrupts
sev set overflow bit, O=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
tbl 8-bit look up and interpolation
tbne test and branch if result≠0
tfr 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
wai wait for interrupt
wav weighted Fuzzy logic average

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

Freescale 6812 addressing modes r is X, Y, SP, or FC

Pseudo op | Meaning
--- | ---
equ set | Define a constant symbol
dc.b db fcb .byte | Allocate byte(s) with values
fcc | Create an ASCII string
dc.w dw fdb .word | Allocate word(s) with values
dc.l dl .long | Allocate 32-bit with values
ds ds.b rmb .blkb | Allocate bytes without init
ds .blkw | Allocate words without init

n is Metrowerks number

| Vector | n | Interrupt Source | Arm |
--- | --- | --- | ---
ffffe | Reset | None |
ffff8 | 3 | Trap | None |
ffff6 | 4 | SWI | None |
ffff0 | 7 | Real time interrupt | CRGINT.RTIE |
ffffe | 8 | Timer channel 0 | TIE.C0I |
fffec | 9 | Timer channel 1 | TIE.C1I |
fffea | 10 | Timer channel 2 | TIE.C2I |
ffff8 | 11 | Timer channel 3 | TIE.C3I |
ffffe | 12 | Timer channel 4 | TIE.C4I |
fffe4 | 13 | Timer channel 5 | TIE.C5I |
fffe2 | 14 | Timer channel 6 | TIE.C6I |
ffff0 | 15 | Timer channel 7 | TIE.C7I |
fffdde | 16 | Timer overflow | TSCR2.TOI |
fffd6 | 20 | SCI0 TDRE, RDRF | SCI0CR2.TIE,RIE |
fffd4 | 21 | SCI1 TDRE, RDRF | SCI1CR2.TIE,RIE |
ffffce | 24 | Key Wakeup J | PIEJ,[7,6,1,0] |
fffcc | 25 | Key Wakeup H | PIEH.[7:0] |
ffff8e | 56 | Key Wakeup P | PIEP.[7:0] |

Interrupt Vectors and interrupt number.
### Address

<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>$0044-5</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td>TCNT</td>
</tr>
<tr>
<td>$0046</td>
<td>TEN</td>
<td>TSWAI</td>
<td>TSFRZ</td>
<td>TFFCA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TSCR1</td>
</tr>
<tr>
<td>$004C</td>
<td>C7I</td>
<td>C6I</td>
<td>C5I</td>
<td>C4I</td>
<td>C3I</td>
<td>C2I</td>
<td>C1I</td>
<td>C0I</td>
<td>TIE</td>
</tr>
<tr>
<td>$004D</td>
<td>TOI</td>
<td>PUPT</td>
<td>RDPT</td>
<td>TCRE</td>
<td>PR2</td>
<td>PR1</td>
<td>PR0</td>
<td></td>
<td>TSCR2</td>
</tr>
<tr>
<td>$004E</td>
<td>C7F</td>
<td>C6F</td>
<td>C5F</td>
<td>C4F</td>
<td>C3F</td>
<td>C2F</td>
<td>C1F</td>
<td>C0F</td>
<td>TFLG1</td>
</tr>
<tr>
<td>$004F</td>
<td>TOF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TFLG2</td>
</tr>
<tr>
<td>$0050-1</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td>TC0</td>
</tr>
<tr>
<td>$0052-3</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td>TC1</td>
</tr>
<tr>
<td>$0054-5</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td>TC2</td>
</tr>
<tr>
<td>$0056-7</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td>TC3</td>
</tr>
<tr>
<td>$0058-9</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td>TC4</td>
</tr>
<tr>
<td>$005A-B</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td>TC5</td>
</tr>
<tr>
<td>$005C-D</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td>TC6</td>
</tr>
<tr>
<td>$005E-F</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td>TC7</td>
</tr>
<tr>
<td>$0062</td>
<td>ADPU</td>
<td>AFFC</td>
<td>ASWAI</td>
<td>ETRIGLE</td>
<td>ETRIGP</td>
<td>ETRIG</td>
<td>ASCIE</td>
<td>ASCIF</td>
<td>ATD0CTL2</td>
</tr>
<tr>
<td>$0063</td>
<td>S8C</td>
<td>S4C</td>
<td>S2C</td>
<td>S1C</td>
<td>FIF0</td>
<td>FRZ1</td>
<td>FRZ0</td>
<td></td>
<td>ATD0CTL3</td>
</tr>
<tr>
<td>$0065</td>
<td>SRES8</td>
<td>SMP1</td>
<td>SMP0</td>
<td>PRS4</td>
<td>PRS3</td>
<td>PRS2</td>
<td>PRS1</td>
<td>PRS0</td>
<td>ATD0CTL4</td>
</tr>
<tr>
<td>$0065</td>
<td>DJM</td>
<td>DSGN</td>
<td>SCAN</td>
<td>MULT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ATD0CTL5</td>
</tr>
<tr>
<td>$0066</td>
<td>SCF</td>
<td>0</td>
<td>ETORF</td>
<td>FIFOR</td>
<td>0</td>
<td>CC</td>
<td>CB</td>
<td>CA</td>
<td>ATD0STAT0</td>
</tr>
<tr>
<td>$0068</td>
<td>CCF7</td>
<td>CCF6</td>
<td>CCF5</td>
<td>CCF4</td>
<td>CCF3</td>
<td>CCF2</td>
<td>CCF1</td>
<td>CCF0</td>
<td>ATD0STAT1</td>
</tr>
<tr>
<td>$006D</td>
<td>Bit 7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
<td>ATD0DIE</td>
</tr>
<tr>
<td>$006F</td>
<td>PAD07</td>
<td>PAD06</td>
<td>PAD05</td>
<td>PAD04</td>
<td>PAD03</td>
<td>PAD02</td>
<td>PAD01</td>
<td>PAD00</td>
<td>PORTAD0</td>
</tr>
<tr>
<td>$0090-1</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td>ATD0R0</td>
</tr>
<tr>
<td>$0092-3</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td>ATD0R1</td>
</tr>
<tr>
<td>$0094-5</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td>ATD0R2</td>
</tr>
<tr>
<td>$0096-7</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td>ATD0R3</td>
</tr>
<tr>
<td>$0098-9</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td>ATD0R4</td>
</tr>
<tr>
<td>$009A-B</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td>ATD0R5</td>
</tr>
<tr>
<td>$009C-D</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td>ATD0R6</td>
</tr>
<tr>
<td>$009E-F</td>
<td>Bit 15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td>ATD0R7</td>
</tr>
<tr>
<td>$00C9</td>
<td>0</td>
<td>0</td>
<td>SBR12</td>
<td>SBR11</td>
<td>SBR10</td>
<td>SBR0</td>
<td>SC10BD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$00CA</td>
<td>LOOPS</td>
<td>SCISWAI</td>
<td>RSRC</td>
<td>M</td>
<td>WAKE</td>
<td>ILT</td>
<td>PE</td>
<td>PT</td>
<td>SC10CR1</td>
</tr>
<tr>
<td>$00CB</td>
<td>TIE</td>
<td>TCIE</td>
<td>ILIE</td>
<td>TE</td>
<td>RE</td>
<td>RWU</td>
<td>SBK</td>
<td>SC10CR2</td>
<td></td>
</tr>
<tr>
<td>$00CC</td>
<td>TDRE</td>
<td>TC</td>
<td>RDRF</td>
<td>IDLE</td>
<td>OR</td>
<td>NF</td>
<td>FE</td>
<td>PF</td>
<td>SC10SR1</td>
</tr>
<tr>
<td>$00CD</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BRK13</td>
<td>TXDIR</td>
<td>RAFF</td>
</tr>
<tr>
<td>$00CF</td>
<td>R7/T7</td>
<td>R6/T6</td>
<td>R5/T5</td>
<td>R4/T4</td>
<td>R3/T3</td>
<td>R2/T2</td>
<td>R1/T1</td>
<td>R0/T0</td>
<td>SC10DRL</td>
</tr>
<tr>
<td>$00D0-1</td>
<td>0</td>
<td>0</td>
<td>SBR12</td>
<td>SBR11</td>
<td>SBR10</td>
<td>SBR0</td>
<td>SC11BD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$00D2</td>
<td>LOOPS</td>
<td>SCISWAI</td>
<td>RSRC</td>
<td>M</td>
<td>WAKE</td>
<td>ILT</td>
<td>PE</td>
<td>PT</td>
<td>SC11CR1</td>
</tr>
<tr>
<td>$00D3</td>
<td>TIE</td>
<td>TCIE</td>
<td>ILIE</td>
<td>TE</td>
<td>RE</td>
<td>RWU</td>
<td>SBK</td>
<td>SC11CR2</td>
<td></td>
</tr>
<tr>
<td>$00D4</td>
<td>TDRE</td>
<td>TC</td>
<td>RDRF</td>
<td>IDLE</td>
<td>OR</td>
<td>NF</td>
<td>FE</td>
<td>PF</td>
<td>SC11SR1</td>
</tr>
<tr>
<td>$00D5</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BRK13</td>
<td>TXDIR</td>
<td>RAFF</td>
</tr>
<tr>
<td>$00D7</td>
<td>R7/T7</td>
<td>R6/T6</td>
<td>R5/T5</td>
<td>R4/T4</td>
<td>R3/T3</td>
<td>R2/T2</td>
<td>R1/T1</td>
<td>R0/T0</td>
<td>SC11DRL</td>
</tr>
<tr>
<td>$00D8</td>
<td>PJ7</td>
<td>PT6</td>
<td>PT5</td>
<td>PT4</td>
<td>PT3</td>
<td>PT2</td>
<td>PT1</td>
<td>PT0</td>
<td>PTT</td>
</tr>
<tr>
<td>$0242</td>
<td>DDR7</td>
<td>DDR6</td>
<td>DDR5</td>
<td>DDR4</td>
<td>DDR3</td>
<td>DDR2</td>
<td>DDR1</td>
<td>DDR0</td>
<td>DDR7</td>
</tr>
<tr>
<td>$0248</td>
<td>PS7</td>
<td>PS6</td>
<td>PS5</td>
<td>PS4</td>
<td>PS3</td>
<td>PS2</td>
<td>PS1</td>
<td>PS0</td>
<td>PPS</td>
</tr>
<tr>
<td>$024A</td>
<td>DDRS7</td>
<td>DDRS6</td>
<td>DDRS5</td>
<td>DDRS4</td>
<td>DDRS3</td>
<td>DDRS2</td>
<td>DDRS1</td>
<td>DDRS0</td>
<td>DDRS</td>
</tr>
<tr>
<td>$0250</td>
<td>PM7</td>
<td>PM6</td>
<td>PM5</td>
<td>PM4</td>
<td>PM3</td>
<td>PM2</td>
<td>PM1</td>
<td>PM0</td>
<td>PTFM</td>
</tr>
<tr>
<td>$0252</td>
<td>DDRM7</td>
<td>DDRM6</td>
<td>DDRM5</td>
<td>DDRM4</td>
<td>DDRM3</td>
<td>DDRM2</td>
<td>DDRM1</td>
<td>DDRM0</td>
<td>DDRM</td>
</tr>
<tr>
<td>$0258</td>
<td>PP7</td>
<td>PP6</td>
<td>PP5</td>
<td>PP4</td>
<td>PP3</td>
<td>PP2</td>
<td>PP1</td>
<td>PP0</td>
<td>PTP</td>
</tr>
<tr>
<td>$025A</td>
<td>DDRP7</td>
<td>DDRP6</td>
<td>DDRP5</td>
<td>DDRP4</td>
<td>DDRP3</td>
<td>DDRP2</td>
<td>DDRP1</td>
<td>DDRP0</td>
<td>DDRP</td>
</tr>
<tr>
<td>$0260</td>
<td>PH7</td>
<td>PH6</td>
<td>PH5</td>
<td>PH4</td>
<td>PH3</td>
<td>PH2</td>
<td>PH1</td>
<td>PH0</td>
<td>PTH</td>
</tr>
<tr>
<td>$0262</td>
<td>DHRH7</td>
<td>DHRH6</td>
<td>DHRH5</td>
<td>DHRH4</td>
<td>DHRH3</td>
<td>DHRH2</td>
<td>DHRH1</td>
<td>DHRH0</td>
<td>DHRH</td>
</tr>
<tr>
<td>$0268</td>
<td>PJ7</td>
<td>PJ6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>PJ1</td>
<td>PJ0</td>
<td>PTJ</td>
<td>PTT</td>
</tr>
<tr>
<td>$026A</td>
<td>DDRJ7</td>
<td>DDRJ6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>DDRJ1</td>
<td>DDRJ0</td>
<td>DDRJ</td>
<td>DDRJ</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)

Jonathan Valvano
May 17, 2010
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 8MHz/2^n, with PLL TCNT is 24MHz/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 MHz</th>
<th>E = 24 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 µs</td>
<td>1 MHz</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>2 µs</td>
<td>500 kHz</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>32</td>
<td>4 µs</td>
<td>250 kHz</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>64</td>
<td>8 µs</td>
<td>125 kHz</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>128</td>
<td>16 µs</td>
<td>62.5 kHz</td>
</tr>
</tbody>
</table>

SCI1DRL 8-bit SCI1 data register
SCI1BD is 16-bit SCI1 baud rate register, let n be the 13-bit number  Baud rate is EClk/n/16
SCI1CR1 is 8-bit SCI1 control register
bit 4 M, Mode, 0 = One start, eight data, one stop bit, 1 = One start, eight data, ninth data, one stop bit
SCI1CR2 is 8-bit SCI1 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.
SCI1SR1 is 8-bit SCI1 status register
bit 7 TDRE, Transmit Data Register Empty Flag
Set if transmit data can be written to SCI1DRL
Cleared by SCI1SR1 read with TDRE set followed by SCI1DRL write
bit 5 RDRF, Receive Data Register Full
set if a received character is ready to be read from SCI1DRL
Clear the RDRF flag by reading SCI1SR1 with RDRF set and then reading SCI1DRL

BSET
Set Bit(s) in Memory
Operation: (M) + (Mask) ⇒ M

ADDA
Add without Carry to A
Operation: (A) + (M) ⇒ A

N: Set if MSB of result is set; cleared otherwise
Z: Set if result is 000; cleared otherwise
V: A7 • M7 • R7 + A7 • M7 • R7
Set if two’s complement overflow resulted from the operation; cleared otherwise
C: A7 • M7 + M7 • R7 + R7 • A7
Set if there was a carry from the MSB of the result; cleared otherwise