First:________________ Last:____________________

This is a closed book exam. You must put your answers in the boxes provided. You have 3 hours, so allocate your time accordingly. ***Please read the entire exam before starting.***

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."

Signature ______________________________________

<table>
<tr><td colspan="3">5a)</td><td colspan="3">5b)</td></tr>
<tr><td colspan="3">5c)</td><td colspan="3">5d)</td></tr>
<tr><td colspan="3">5e)</td><td colspan="3">5f)</td></tr>
<tr><td colspan="3">5g)</td><td colspan="3">5h)</td></tr>
<tr><td colspan="3">5i)</td><td colspan="3">5j)</td></tr>
<tr><td colspan="3">6a)</td><td colspan="3">6b)</td></tr>
<tr><td colspan="3">6c)</td><td colspan="3">6d)</td></tr>
<tr><td colspan="3">7a)</td><td colspan="3">7b)</td></tr>
<tr><td>8a)</td><td>8b)</td><td>8c)</td><td>8d)</td><td>8e)</td><td>8f)</td></tr>
<tr><td colspan="2">9)</td><td>11a)</td><td colspan="3" rowspan="3">10)</td></tr>
<tr><td colspan="2">11b)</td><td>11c)</td></tr>
<tr><td colspan="2">11d)</td><td>11e)</td></tr>
<tr><td colspan="3">12)</td><td colspan="3">13)</td></tr>
</table>

14)	
15a)	15b)
16a)	16b)
17a)	17b)
17c)	
18a)	18c)
18b)	

(10) Question 1. Write two debugging functions in C. Your debugging instrument will record the values observed on Port A. You may assume someone else will initialize Port A. The main program will call your function **`Init`** once at the start of the experiment. At strategic times during the experiment, someone else will call your function **`Record`** to capture Port A data. Your system will save the last *N* values in RAM, where *N* is defined like this

```
#define N 10
```

To change *N* one only needs to edit the above line and recompile the code. After *N* data values are collected, your debugging instrument will record the next data by discarding the oldest data. After that, your system maintains a record of the last *N* measurements.

Part a) Show the C code you would place in the header file (Debug.h). Comments will be graded.

Part b) Show the C code you would place in the code file (Debug.c). Comments are not required for this part. There is no particular requirement about the order of how the data is saved; you just need to be able to save the last *N* measurements. Show the implementations of **`Init`** and **`Record`**.

(5) Question 2. Interface a switch to PA0. Implement the interface in negative logic. Assume the port pin is initialized as an input with internal pull-up. Minimize cost of the interface. Show hardware connections; no software is required.

(5) Question 3. Interface an LED to PA1. Implement the interface in positive logic. The desired LED operating point is 1V 1mA. The V_{OH} is 3.0V and V_{OL} = 0.1V. Minimize cost of the interface. Show hardware connections; no software is required.

(5) Problem 4. Assume the UART0 has been initialized. Use busy-wait synchronization to implement a C function with the following steps

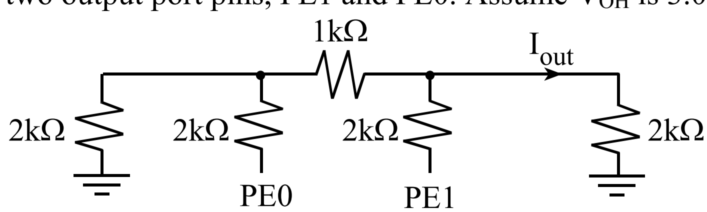
1) Wait for new serial port input
2) Read the new 8-bit data
3) Echo the data by transmitting the same 8-bit data just received
4) Return by value the one byte received.

Define a function in C that performs these four steps. Be careful to define the input and output parameters in an appropriate manner.

(10) Question 5. State the term that is best described by each definition.
Part a) You are given a DAC to test. You increment the input to the DAC stepping through all possible values. For each change in input you notice that the change in output voltage, ΔV, is always positive.
Part b) A property of RAM such that data is lost if power is removed and then restored.
Part c) A UART transmission communicates 8 bits of information, but each frame is 10 bits wide. What are the other two bits? Give there names as words rather than as numbers.
Part d) A subset of a number system from which all values in the set can be constructed.
Part e) A characteristic of a debugger when the presence of the collection of information itself has a small but unimportant effect on the parameters being measured.
Part f) A synchronization method used to link a background thread to a foreground thread. No data is being passed. The foreground thread spins waiting for a condition to occur. The background thread triggers this condition. After the trigger, the foreground is released so it will continue.
Part g) A type of software variable where the scope of access is restricted.
Part h) A debugging process that allows you to determine what software is being run and when it runs.
Part i) The name given to describe 1,024 bytes.
Part j) A type of digital logic where the voltage representing true is less than the voltage representing false.

(4) Question 6. List four limitations occurring when analog signals are converted into digital numbers using an ADC. Give your answer as one word or a short phrase.

(6) Question 7. This circuit is a 2-bit DAC using the R-2R configuration. The DAC is controlled by two output port pins, PE1 and PE0. Assume V_{OH} is 3.0V and V_{OL} = 0V.



Part a) What is the output current I_{out} when PE1 is high, and PE0 is low?
Part b) What is the output current I_{out} when PE1 is low, and PE0 is high?

(6) Question 8. Consider the following file with one function and 6 variables. Which type are **v1**–**v6**? Each selection A-F may be used zero, one, or more times.

A) A public permanently-allocated variable
B) A public temporarily-allocated variable
C) A temporary variable private to the function **Fun_Init**
D) A permanently-allocated variable private to the function **Fun_Init**
E) A permanently-allocated variable, private to the file **Fun.c**.
F) A syntax error causing this code to not compile
It is possible one letter code could be used multiple times, while other codes might not be used.

```
// This is the first line of the Fun.c code file
long v1;
volatile long v2;
static long v3;
void Fun_Init(int in){     // code
long v4;
static long v5;
  v4 = 0;
long v6;
  if(in==0) {
    v1 = 0;
  }
  v2 = 10;
}
// this is the last line of the Fun.c code file
```

Part a) What is the best classification for the variable v1? Specify a letter from A to F.
Part b) What is the best classification for the variable v2? Specify a letter from A to F.
Part c) What is the best classification for the variable v3? Specify a letter from A to F.
Part d) What is the best classification for the variable v4? Specify a letter from A to F.
Part e) What is the best classification for the variable v5? Specify a letter from A to F.
Part f) What is the best classification for the variable v6? Specify a letter from A to F.

(1) Question 9. If R0 and R1 both equal $2*10^9$ will the instruction **ADDS R2,R1,R0** set the V bit? Answer Yes or No.

(3) Question 10. Assume there is a buffer is defined in assembly with the equivalent size and type.

;assembly	// C
Buffer SPACE 400	long Buffer[100];

Show the assembly code that sets element number 50 to the value -1. This will be 2 to 4 assembly instructions. I.e., your assembly code is equivalent to the following C.

```
   Buffer[50] = -1;
```

(10) Question 11. In this question, the subroutine implements a call by value parameter passed on the stack. There are no return parameters. Call by value means the data itself is pushed on the stack. A typical calling sequence is

```
     AREA    |.text|, CODE, READONLY, ALIGN=2
Data DCD  100          ;32-bit information
Main LDR  R0,=Data
     LDR  R0,[R0]
     PUSH {R0}         ;the value of the Data is pushed
     MOV  R0,#0        ;no cheating, parameter not in R0, on stack
     BL   Subroutine
     ADD  SP,SP,#4     ;discard parameter
```

The subroutine allocates two 32-bit local variables, **L1 L2**, and uses SP stack pointer addressing to access the local variables and the parameter. The binding for these three are

```
In  EQU  ??(a)??  ;32-bit value that is the input parameter
L1  EQU  ??(b)??  ;32-bit local variable
L2  EQU  ??(c)??  ;32-bit local variable
Subroutine
    PUSH {R10,R11,LR}
    ***(d)****    ;allocate L1, L2
;---------start of body-------------------
    LDR  R11,[SP,#In]  ;Reg R11 is the input parameter data
    STR  R11,[SP,#L2]  ;save parameter into local L1
;---------end of body---------------------
    ???(e)???     ;deallocate L1,L2
    POP  {R10,R11,PC}
```

Part a) Show the binding for **In**. I.e., give the value that goes in the **???(a)???** spot.
Part b) Show the binding for **L1**. I.e., give the value that goes in the **???(b)???** spot.
Part c) Show the binding for **L2**. I.e., give the value that goes in the **???(c)???** spot.
Part d) Show the allocation instruction(s) for the **???(d)???** in the above program.
Part e) Show the deallocation instruction(s) for the **???(e)???** in the above program.

(5) Question 12. Consider a system similar to Lab 9 but with these specifications. The bus cycle is 50MHz. The baud rate is 50,000 bits/sec. The SysTick interrupt rate is 100 Hz. Each interrupt the 10-bit ADC is sampled and the information is transmitted as an 8 byte message. What is the actual bandwidth of the communication system? I am not asking the peak possible bandwidth.

(5) Question 13. Consider this FIFO put function. There are no bugs in the C implementation, but there is one bug in the assembly implementation. In other words, you can edit one of the assembly lines to make the assembly function operational. Specify the line number and the corrected code.

```
Fifo_Put LDR  R1,=PutPt       ;1
         LDR  R2,[R1]         ;2
         ADD  R3,R2,#1        ;3
         LDR  R12,=Fifo+20    ;4
         CMP  R3,R12          ;5
         BNE  NoWrap          ;6
         LDR  R3,=Fifo        ;7
NoWrap   LDR  R12,=GetPt      ;8
         LDR  R12,[R12]       ;9
         CMP  R3,R12          ;10
         BNE  NotFull         ;11
         MOV  R0,#0           ;12
         BX   LR              ;13
NotFull  STRH R0,[R2]         ;14
         STR  R3,[R1]         ;15
         MOV  R0,#1           ;16
         BX   LR              ;17
```

```
#define FIFO_SIZE 10
int Fifo_Put(short data){
short *tempPt;
  tempPt = PutPt+1;

if(tempPt==&Fifo[FIFO_SIZE]){
    tempPt = &Fifo[0];
  }
  if(tempPt == GetPt){
    return(0);
  }
  else{
    *PutPt = data;
    PutPt = tempPt;
    return(1);
  }
}
```

(4) Problem 14. The Stellaris LM4F120 has a 0 to 3V 12-bit ADC. What will be the digital output of the ADC if the input voltage is 1 V?

(5) Question 15. Assume the bus clock is operating at 50 MHz. The SysTick initialization executes these instructions. SysTick will be used with busy-wait synchronization to create time delays

```
SysTick_Init
    LDR R1,=NVIC_ST_RELOAD_R
    ????(a)????
    STR R0,[R1]
    LDR R1,=NVIC_ST_CTRL_R
    ????(b)????
    STR R2,[R1]
    BX  LR
```

What assembly instructions go in the `????(a)????` and `????(b)????` places?

(5) Question 16. Consider the following Mealy FSM

```
struct State {
  unsigned long Out[2];
  unsigned long Delay;
  const struct State *Next[2];};
typedef const struct State STyp;
#define Stop  &FSM[0]
#define Go    &FSM[1]
#define PA0   (*((volatile unsigned long *)0x40004004))
#define PA21  (*((volatile unsigned long *)0x40004018))
STyp FSM[2]={
 {{2,0},10,{Stop,Go}},
 {{0,1},10,{Stop,Go}}};
int main(void){  STyp *Pt;      // state pointer
  unsigned long Input;
  PLL_Init();                   // configure for 50 MHz clock
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOA; // activate port A
  SysTick_Init();               // initialize SysTick timer
  GPIO_PORTA_DIR_R &= ~0x01;    // make PA0 in
  GPIO_PORTA_DIR_R |= 0x06;     // make PA2-1 out
  GPIO_PORTA_AFSEL_R &= ~0x07;  // disable alt func on PA3-0
  GPIO_PORTA_DEN_R |= 0x07;     // enable digital I/O on PA3-0
  Pt = Stop;                    // initial state: stopped
  while(1){
    ????(a)?????                // get new input from Control
    ????(b)?????                // output to Brake and Gas
    SysTick_Wait10ms(Pt->Delay);// wait 10 ms * Delay value
    Pt = Pt->Next[Input];       // transition to next state
  }
}
```

Fill in the missing C code that first inputs from PA0, and second outputs the appropriate value to the PA2, PA1 pins. The input stage should set the variable **`Input`** to 0 or 1 depending on PA0. For example, if then input on PA0 is 1, then your software will make the **`Input`** variable 1. For example, if the motor controller FSM wished to output 2, then your software makes PA2=1 and PA1=0. Your code must be friendly.

(6) Question 17. Consider the following SysTick ISR. Assume SysTick is initialized to interrupt every 50μs. The SysTick is armed and enabled. Assume Port G bit 2 has been configured as an output. Assume also the main program was running when SysTick interrupts are triggered. 0x40026010 is the bit-specific address for the PG2 pin.

```
        AREA    DATA, ALIGN=2
Counts  SPACE   4     ; records number of SysTick interrupts
        AREA    |.text|, CODE, READONLY, ALIGN=2
GPIO_PORTG2  EQU 0x40026010
SysTick_Handler
    LDR R1,=GPIO_PORTG2     ; LED
    MOV R0,#0
    STR R0,[R1]
    EOR R0,R0,#0x04
    LDR R2,=Counts
    LDR R3,[R2]
    ADD R3,R3,#1                  ; Counts = Counts + 1
    STR R3,[R2]
    STR R0,[R1]
    BX  LR
```

Part a) What is in LR during the execution of the ISR?
Part b) What gets pushed on the stack during the invocation of the ISR?
Part c) Sketch the output voltage versus time on PG2

(10) Question 18. A distance is represented as a signed decimal fixed-point number with resolution of 0.001 cm. Assume the variable integer is 32 bits and signed. Assume the variable integer is passed by value into a subroutine using Register R0. Calculate the cost = (2.5 dollars/cm)*distance. The cost is represented as a signed decimal fixed-point number with resolution of $0.01. The function should return the variable integer representing cost in Register R0.
Part a) For example if the distance is 1.20 cm. The cost will be (2.5 dollars/cm)*1.20 cm = 3 dollars. Given this example what do you expect the input value to be in Register R0? Give your answer in decimal (not binary, not hexadecimal).

Part b) Given the example data from part a), what output value should the function return in Register R0? Give your answer in decimal (not binary, not hexadecimal).

Part c) Write the assembly subroutine that converts distance to cost. Verify that the input given in a) results in the output you gave for b). Optimize for speed, eliminate overflow, and minimize dropout.

Memory access instructions

```
LDR    Rd, [Rn]        ; load 32-bit number at [Rn] to Rd
LDR    Rd, [Rn,#off]   ; load 32-bit number at [Rn+off] to Rd
LDR    Rd, =value      ; set Rd equal to any 32-bit value (PC rel)
LDRH   Rd, [Rn]        ; load unsigned 16-bit at [Rn] to Rd
LDRH   Rd, [Rn,#off]   ; load unsigned 16-bit at [Rn+off] to Rd
LDRSH  Rd, [Rn]        ; load signed 16-bit at [Rn] to Rd
LDRSH  Rd, [Rn,#off]   ; load signed 16-bit at [Rn+off] to Rd
LDRB   Rd, [Rn]        ; load unsigned 8-bit at [Rn] to Rd
LDRB   Rd, [Rn,#off]   ; load unsigned 8-bit at [Rn+off] to Rd
LDRSB  Rd, [Rn]        ; load signed 8-bit at [Rn] to Rd
LDRSB  Rd, [Rn,#off]   ; load signed 8-bit at [Rn+off] to Rd
STR    Rt, [Rn]        ; store 32-bit Rt to [Rn]
STR    Rt, [Rn,#off]   ; store 32-bit Rt to [Rn+off]
STRH   Rt, [Rn]        ; store least sig. 16-bit Rt to [Rn]
STRH   Rt, [Rn,#off]   ; store least sig. 16-bit Rt to [Rn+off]
STRB   Rt, [Rn]        ; store least sig. 8-bit Rt to [Rn]
STRB   Rt, [Rn,#off]   ; store least sig. 8-bit Rt to [Rn+off]
PUSH   {Rt}            ; push 32-bit Rt onto stack
POP    {Rd}            ; pop 32-bit number from stack into Rd
ADR    Rd, label       ; set Rd equal to the address at label
MOV{S} Rd, <op2>       ; set Rd equal to op2
MOV    Rd, #im16       ; set Rd equal to im16, im16 is 0 to 65535
MVN{S} Rd, <op2>       ; set Rd equal to -op2
```

Branch instructions

```
B    label   ; branch to label    Always
BEQ  label   ; branch if Z == 1   Equal
BNE  label   ; branch if Z == 0   Not equal
BCS  label   ; branch if C == 1   Higher or same, unsigned ≥
BHS  label   ; branch if C == 1   Higher or same, unsigned ≥
BCC  label   ; branch if C == 0   Lower, unsigned <
BLO  label   ; branch if C == 0   Lower, unsigned <
BMI  label   ; branch if N == 1   Negative
BPL  label   ; branch if N == 0   Positive or zero
BVS  label   ; branch if V == 1   Overflow
BVC  label   ; branch if V == 0   No overflow
BHI  label   ; branch if C==1 and Z==0  Higher, unsigned >
BLS  label   ; branch if C==0 or  Z==1  Lower or same, unsigned ≤
BGE  label   ; branch if N == V   Greater than or equal, signed ≥
BLT  label   ; branch if N != V   Less than, signed <
BGT  label   ; branch if Z==0 and N==V  Greater than, signed >
BLE  label   ; branch if Z==1 and N!=V  Less than or equal, signed ≤
BX   Rm      ; branch indirect to location specified by Rm
BL   label   ; branch to subroutine at label
BLX  Rm      ; branch to subroutine indirect specified by Rm
```

Interrupt instructions

```
CPSIE  I               ; enable interrupts  (I=0)
CPSID  I               ; disable interrupts (I=1)
```

Logical instructions

```
AND{S} {Rd,} Rn, <op2> ; Rd=Rn&op2     (op2 is 32 bits)
ORR{S} {Rd,} Rn, <op2> ; Rd=Rn|op2     (op2 is 32 bits)
EOR{S} {Rd,} Rn, <op2> ; Rd=Rn^op2     (op2 is 32 bits)
BIC{S} {Rd,} Rn, <op2> ; Rd=Rn&(~op2)  (op2 is 32 bits)
ORN{S} {Rd,} Rn, <op2> ; Rd=Rn|(~op2)  (op2 is 32 bits)
```

```
LSR{S} Rd, Rm, Rs      ; logical shift right Rd=Rm>>Rs  (unsigned)
LSR{S} Rd, Rm, #n      ; logical shift right Rd=Rm>>n   (unsigned)
ASR{S} Rd, Rm, Rs      ; arithmetic shift right Rd=Rm>>Rs (signed)
ASR{S} Rd, Rm, #n      ; arithmetic shift right Rd=Rm>>n  (signed)
LSL{S} Rd, Rm, Rs      ; shift left Rd=Rm<<Rs (signed, unsigned)
LSL{S} Rd, Rm, #n      ; shift left Rd=Rm<<n  (signed, unsigned)
```

Arithmetic instructions

```
ADD{S} {Rd,} Rn, <op2> ; Rd = Rn + op2
ADD{S} {Rd,} Rn, #im12 ; Rd = Rn + im12, im12 is 0 to 4095
SUB{S} {Rd,} Rn, <op2> ; Rd = Rn - op2
SUB{S} {Rd,} Rn, #im12 ; Rd = Rn - im12, im12 is 0 to 4095
RSB{S} {Rd,} Rn, <op2> ; Rd = op2 - Rn
RSB{S} {Rd,} Rn, #im12 ; Rd = im12 – Rn
CMP    Rn, <op2>       ; Rn – op2      sets the NZVC bits
CMN    Rn, <op2>       ; Rn - (-op2)   sets the NZVC bits
MUL{S} {Rd,} Rn, Rm    ; Rd = Rn * Rm       signed or unsigned
MLA    Rd, Rn, Rm, Ra  ; Rd = Ra + Rn*Rm    signed or unsigned
MLS    Rd, Rn, Rm, Ra  ; Rd = Ra - Rn*Rm    signed or unsigned
UDIV   {Rd,} Rn, Rm    ; Rd = Rn/Rm         unsigned
SDIV   {Rd,} Rn, Rm    ; Rd = Rn/Rm         signed
```

Notes

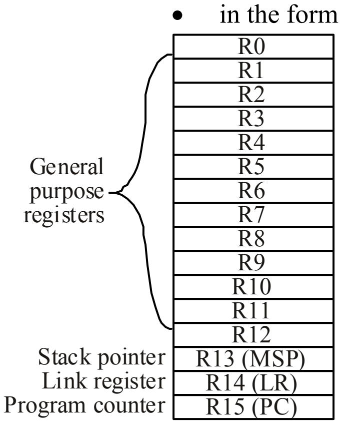
```
Ra Rd Rm Rn Rt represent 32-bit registers
 value   any 32-bit value: signed, unsigned, or address
 {S}     if S is present, instruction will set condition codes
 #im12   any value from 0 to 4095
 #im16   any value from 0 to 65535
 {Rd,}   if Rd is present Rd is destination, otherwise Rn
 #n      any value from 0 to 31
 #off    any value from -255 to 4095
 label   any address within the ROM of the microcontroller
 op2     the value generated by <op2>
```

Examples of flexible operand **<op2>** creating the 32-bit number. E.g., **Rd = Rn+op2**

```
ADD Rd, Rn, Rm         ; op2 = Rm
ADD Rd, Rn, Rm, LSL #n ; op2 = Rm<<n  Rm is signed, unsigned
ADD Rd, Rn, Rm, LSR #n ; op2 = Rm>>n  Rm is unsigned
ADD Rd, Rn, Rm, ASR #n ; op2 = Rm>>n  Rm is signed
ADD Rd, Rn, #constant  ; op2 = constant
```

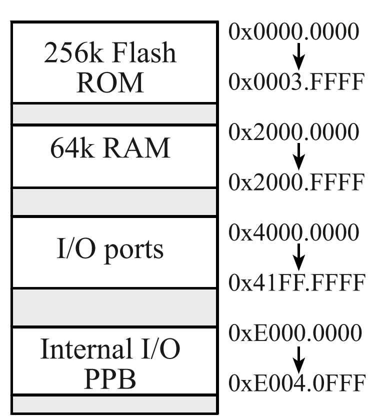
, where **X** and **Y** are hexadecimal digits:

- produced by shifting an 8-bit unsigned value left by any number of bits
- in the form **0x00XY00XY**
- in the form **0xXY00XY00**
- in the form **0xXYXYXYXY**



Condition code bits

N negative
Z zero
V signed overflow
C carry or unsigned overflow



Address	7	6	5	4	3	2	1	0	Name
$400F.E108	GPIOH	GPIOG	GPIOF	GPIOE	GPIOD	GPIOC	GPIOB	GPIOA	SYSCTL_RCGC2_R
$4000.43FC	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	GPIO_PORTA_DATA_R
$4000.4400	DIR	DIR	DIR	DIR	DIR	DIR	DIR	DIR	GPIO_PORTA_DIR_R
$4000.4420	SEL	SEL	SEL	SEL	SEL	SEL	SEL	SEL	GPIO_PORTA_AFSEL_R
$4000.451C	DEN	DEN	DEN	DEN	DEN	DEN	DEN	DEN	GPIO_PORTA_DEN_R

Table 4.5. Some LM3S1968 parallel ports. Each register is 32 bits wide. Bits 31 – 8 are zero.

We set the direction register (e.g., `GPIO_PORTA_DIR_R`) to specify which pins are input (0) and which are output (1). We will set bits in the alternative function register when we wish to activate the alternate functions (not GPIO). We use the data register (e.g., `GPIO_PORTA_DATA_R`) to perform input/output on the port. For each I/O pin we wish to use whether GPIO or alternate function we must enable the digital circuits by setting the bit in the enable register (e.g., `GPIO_PORTA_DEN_R`).

Address	31	30	29-7	6	5	4	3	2	1	0	Name
0xE000E100	G	F	…	UART1	UART0	E	D	C	B	A	NVIC_EN0_R
0xE000E104			…						UART2	H	NVIC_EN1_R

Address	31-24	23-17	16	15-3	2	1	0	Name
$E000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	NVIC_ST_CTRL_R
$E000E014	0	24-bit RELOAD value						NVIC_ST_RELOAD_R
$E000E018	0	24-bit CURRENT value of SysTick counter						NVIC_ST_CURRENT_R

Address	31-29	28-24	23-21	20-8	7-5	4-0	Name
$E000ED20	TICK	0	PENDSV	0	DEBUG	0	NVIC_SYS_PRI3_R

Table 9.6. SysTick registers.

Table 9.6 shows the SysTick registers used to create a periodic interrupt. SysTick has a 24-bit counter that decrements at the bus clock frequency. Let f_{BUS} be the frequency of the bus clock, and let n be the value of the **RELOAD** register. The frequency of the periodic interrupt will be $f_{BUS}/(n+1)$. First, we clear the **ENABLE** bit to turn off SysTick during initialization. Second, we set the **RELOAD** register. Third, we write to the `NVIC_ST_CURRENT_R` value to clear the counter. Lastly, we write the desired mode to the control register, `NVIC_ST_CTRL_R`. To turn on the SysTick, we set the **ENABLE** bit. We must set **CLK_SRC**=1, because **CLK_SRC**=0 external clock mode is not implemented on the LM3S/LM4F family. We set **INTEN** to enable interrupts. The standard name for the SysTick ISR is **SysTick_Handler**.

Address	31-17	16	15-10	9	8	7-0	Name
$400F.E000		ADC		MAXADCSPD			SYSCTL_RCGC0_R

	31-14	13-12	11-10	9-8	7-6	5-4	3-2	1-0	
$4003.8020		SS3		SS2		SS1		SS0	ADC_SSPRI_R

	31-16	15-12	11-8	7-4	3-0	
$4003.8014		EM3	EM2	EM1	EM0	ADC_EMUX_R

	31-4	3	2	1	0	
$4003.8000		ASEN3	ASEN2	ASEN1	ASEN0	ADC_ACTSS_R
$4003.80A0			MUX0			ADC_SSMUX3_R
$4003.80A4		TS0	IE0	END0	D0	ADC_SSCTL3_R
$4003.8028		SS3	SS2	SS1	SS0	ADC_PSSI_R
$4003.8004		INR3	INR2	INR1	INR0	ADC_RIS_R
$4003.8008		MASK3	MASK2	MASK1	MASK0	ADC_IM_R
$4003.800C		IN3	IN2	IN1	IN0	ADC_ISC_R

	31-10	9-0	
$4003.80A8		DATA	ADC_SSFIFO3

Table 10.3. The LM3S ADC registers. Each register is 32 bits wide.

Set MAXADCSPD to 00 for slow speed operation. The ADC has four sequencers, but we will use only sequencer 3. We set the **ADC_SSPRI_R** register to 0x3210 to make sequencer 3 the lowest priority. Because we are using just one sequencer, we just need to make sure each sequencer has a unique priority. We set bits 15–12 (**EM3**) in the **ADC_EMUX_R** register to specify how the ADC will be triggered. If we specify software start (**EM3**=0x0), then the software writes an 8 (**SS3**) to the **ADC_PSSI_R** to initiate a conversion on sequencer 3. Bit 3 (**INR3**) in the **ADC_RIS_R** register will be set when the conversion is complete. We can enable and disable the sequencers using the **ADC_ACTSS_R** register. There are eight on the LM3S1968. Which channel we sample is configured by writing to the **ADC_SSMUX3_R** register. The **ADC_SSCTL3_R** register specifies the mode of the ADC sample. Clear **TS0**. We set **IE0** so that the **INR3** bit is set on ADC conversion, and clear it when no flags are needed. We will set **IE0** for both interrupt and busy-wait synchronization. When using sequencer 3, there is only one sample, so **END0** will always be set, signifying this sample is the end of the sequence. Clear the **D0** bit. The **ADC_RIS_R** register has flags that are set when the conversion is complete, assuming the **IE0** bit is set. Do not set bits in the **ADC_IM_R** register because we do not want interrupts.

UART0 pins are on PA1 (transmit) and PA0 (receive). The `UART0_IBRD_R` and `UART0_FBRD_R` registers specify the baud rate. The baud rate **divider** is a 22-bit binary fixed-point value with a resolution of 2^{-6}. The **Baud16** clock is created from the system bus clock, with a frequency of (Bus clock frequency)/**divider**. The baud rate is

Baud rate = **Baud16**/**16** = (Bus clock frequency)/(16***divider**)

We set bit 4 of the `UART0_LCRH_R` to enable the hardware FIFOs. We set both bits 5 and 6 of the `UART0_LCRH_R` to establish an 8-bit data frame. The **RTRIS** is set on a receiver timeout, which is when the receiver FIFO is not empty and no incoming frames have occurred in a 32-bit time period. The arm bits are in the `UART0_IM_R` register. To acknowledge an interrupt (make the trigger flag become zero), software writes a 1 to the corresponding bit in the `UART0_IC_R` register. We set bit 0 of the `UART0_CTL_R` to enable the UART. Writing to `UART0_DR_R` register will output on the UART. This data is placed in a 16-deep transmit hardware FIFO. Data are transmitted first come first serve. Received data are place in a 16-deep receive hardware FIFO. Reading from `UART0_DR_R` register will get one data from the receive hardware FIFO. The status of the two FIFOs can be seen in the `UART0_FR_R` register (FF is FIFO full, FE is FIFO empty). The standard name for the UART0 ISR is **UART0_Handler**. RXIFLSEL specifies the receive FIFO level that causes an interrupt (010 means interrupt on ≥ ½ full, or 7 to 8 characters). TXIFLSEL specifies the transmit FIFO level that causes an interrupt (010 means interrupt on ≤ ½ full, or 9 to 8 characters).

	31–12	11	10	9	8	7–0	Name
$4000.C000		OE	BE	PE	FE	DATA	UART0_DR_R

	31–3	3	2	1	0	
$4000.C004		OE	BE	PE	FE	UART0_RSR_R

	31–8	7	6	5	4	3	2–0	
$4000.C018		TXFE	RXFF	TXFF	RXFE	BUSY		UART0_FR_R

	31–16	15–0	
$4000.C024		DIVINT	UART0_IBRD_R

	31–6	5–0	
$4000.C028		DIVFRAC	UART0_FBRD_R

	31–8	7	6 – 5	4	3	2	1	0	
$4000.C02C		SPS	WPEN	FEN	STP2	EPS	PEN	BRK	UART0_LCRH_R

	31–10	9	8	7	6–3	2	1	0	
$4000.C030		RXE	TXE	LBE		SIRLP	SIREN	UARTEN	UART0_CTL_R

	31–6	5-3	2-0	
$4000.C034		RXIFLSEL	TXIFLSEL	UART0_IFLS_R

	31-11	10	9	8	7	6	5	4		
$4000.C038		OEIM	BEIM	PEIM	FEIM	RTIM	TXIM	RXIM		UART0_IM_R
$4000.C03C		OERIS	BERIS	PERIS	FERIS	RTRIS	TXRIS	RXRIS		UART0_RIS_R
$4000.C040		OEMIS	BEMIS	PEMIS	FEMIS	RTMIS	TXMIS	RXMIS		UART0_MIS_R
$4000.C044		OEIC	BEIC	PEIC	FEIC	RTIC	TXIC	RXIC		UART0_IC_R

Table 11.2. UART0 registers. Each register is 32 bits wide. Shaded bits are zero.