Final Exam

Date: May 8, 2014

UT EID: ___________________________ Circle one: VJR, NT, RY

Printed Name: ___________________________ Last, ___________________________ First

Your signature is your promise that you have not cheated and will not cheat on this exam, nor will you help others to cheat on this exam. You will not reveal the contents of this exam to others who are taking the makeup thereby giving them an undue advantage:

Signature: ________________________________________________________________

Instructions:
• Closed book and closed notes. No books, no papers, no data sheets (other than the last four pages of this Exam)
• No devices other than pencil, pen, eraser (no calculators, no electronic devices), please turn cell phones off.
• Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space (boxes) provided. Anything outside the boxes will be ignored in grading.
• You have 180 minutes, so allocate your time accordingly.
• For all questions, unless otherwise stated, find the most efficient (time, resources) solution.
• Unless otherwise stated, make all I/O accesses friendly.
• Please read the entire exam before starting. See supplement pages for Device I/O registers.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>10</td>
</tr>
<tr>
<td>Problem 2</td>
<td>10</td>
</tr>
<tr>
<td>Problem 3</td>
<td>15</td>
</tr>
<tr>
<td>Problem 4</td>
<td>10</td>
</tr>
<tr>
<td>Problem 5</td>
<td>10</td>
</tr>
<tr>
<td>Problem 6</td>
<td>10</td>
</tr>
<tr>
<td>Problem 7</td>
<td>15</td>
</tr>
<tr>
<td>Problem 8</td>
<td>10</td>
</tr>
<tr>
<td>Problem 9</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
</tr>
</tbody>
</table>
(10) Question 1:

(i) What is the name given to 1024 bytes?

(ii) The ____________ thread is the execution of the main program, while the ____________ thread is the execution of the ISR.

(iii) Name the type of FSM where the output value depends on both the current state and input.

(iv) Name the C programming language term that describes the storage of a data structure where the elements of each row are stored in succession.

(v) The smallest complete unit of serial transmission is called a ____________.

(vi) The term given to the collection of software functions that allow the higher level software to utilize an I/O device.

(vii) The name given to a local variable with permanent allocation.

(viii) Name the step in an interrupt service routine where the trigger flag is cleared?

(ix) What two actions are implicitly performed after the SysTick counter reaches a zero.

(x) The assembler directive that places a 32 bit word into memory.
(10) Question 2 (Local Variables).
Given the following C code and its equivalent Assembly code, answer each of the sub-questions.

<table>
<thead>
<tr>
<th>Line#</th>
<th>Assembly Code</th>
<th>C Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sum EQU 0 number</td>
<td>uint32_t comp(void)</td>
</tr>
<tr>
<td>2</td>
<td>n EQU 4 number</td>
<td>{</td>
</tr>
<tr>
<td>3</td>
<td>comp PUSH (R4,R5,R11,LR)</td>
<td>uint32_t sum,n;</td>
</tr>
<tr>
<td>4</td>
<td>MOV R11,SP</td>
<td>sum = 0;</td>
</tr>
<tr>
<td>5</td>
<td>SUB R11,#8</td>
<td>for(n=1000; n&gt;0 ; n--)</td>
</tr>
<tr>
<td>6</td>
<td>MOV R0,#0</td>
<td>{</td>
</tr>
<tr>
<td>7</td>
<td>STR R0,[R11,#sum]</td>
<td>sum=sum+n;</td>
</tr>
<tr>
<td>8</td>
<td>MOV R1,#1000</td>
<td>}</td>
</tr>
<tr>
<td>9</td>
<td>STR R1,[R11,#n]</td>
<td>return sum;</td>
</tr>
<tr>
<td>10</td>
<td>loop LDR R1,[R11,#n]</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>LDR R0,[R11,#sum]</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>ADD R0,R1</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>STR R0,[R11,sum]</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>LDR R1,[R11,#n]</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>SUBS R1,#1</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>STR R1,[R11,#n]</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>BNE loop</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>ADD R11,#8</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>POP (R4,R5,R11,PC)</td>
<td></td>
</tr>
</tbody>
</table>

**a) (4 points)** There are four key stages in the implementation of local variables. Identify each of those stages in the assembly routine above. Write the instruction number that marks the beginning of a stage and provide a brief one-line statement explaining the purpose of the stage.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binding</td>
<td>Line 1</td>
</tr>
<tr>
<td>Allocation</td>
<td>Line 5</td>
</tr>
<tr>
<td>Access</td>
<td>Line 7</td>
</tr>
<tr>
<td>DeAllocation</td>
<td>Line 18</td>
</tr>
</tbody>
</table>

**b) (2 points)** Identify the base pointer in the assembly code and explain its usefulness. In other words, can we always use the stack pointer for accessing local variables?

R11 is the frame pointer (base)

**c) (4 points)** Assuming $n$ were changed from `uint32_t` to `uint16_t` data type, identify all lines of assembly code that require changing. List below the corrected versions of these lines.

8 n FOR 2 STR→STRH (Lines 7,9,13,16)
5 SUB R11,#4 LDR→LDRH (Lines 10,11,14)
18 ADD R11,#4
(15) Question 3 (C Programming with struct).

a) (4 points) Define a generic C struct called MyString that contains two attributes, an array of chars and an index variable. The character array must be large enough to hold the string “ABCDEFGHIJ”.

```c
struct MyString {
    char Arr[11]; // 11 + null
    int idx;
};
```

b) (5 points) Write a function called LCDOut which accepts a pointer to a struct of type MyString as a parameter and prints the character at the current index-th location to the LCD using LCDOutChar(char c). It should then increment the index by 1.

```c
void LCDOut(struct MyString *s) {
    LCDOutChar(s->Arr[s->idx]);
    s->idx ++;
    // s can be replaced by (*s)
}
```

c) (6 points) Call the LCDOut function in a loop from your main program until all characters of the variable, outStr, are output to the LCD.

```c
void main() {
    MyString outStr; // Assume outStr already initialized
    int i = 0; outStr.idx = 0;
    while (outStr.Arr[i]) {
        LCDOut(&outStr);
        i++;
    }
    // can also use just a for loop
    // and iterate 10 time(s)
    // (Not perfect)
}
```
(10) Question 4 (Interrupts).
Using SysTick Interrupt only to generate the following signal on Port E pin 2.

| PE2 | 4ms | 6ms | 4ms | 6ms | 4ms | 6ms |

a) (3 points) Assuming the following initialization steps have been done for you:
- Clock is setup at 50MHz
- GPIO Port E pin 2 has been configured and an initial value of 0 written to it.
What values should these three registers be initialized to?

<table>
<thead>
<tr>
<th>NVIC_ST_CTRL_R</th>
<th>0x07</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIC_ST_RELOAD_R</td>
<td>200000</td>
</tr>
<tr>
<td>NVIC_ST_CURRENT_R</td>
<td>0</td>
</tr>
</tbody>
</table>

b) (7 points) Complete the `SysTick_Handler` ISR that generates the desired signal. You may assume a global variable called `hilo`, is initialized to zero and use it in your ISR.

```c
void SysTick_Handler() {
    GPIO_PORTE_DATA_R ^= 0x04;
    if (NVIC_ST_RELOAD_R == 200000) {
        NVIC_ST_RELOAD_R = 300000;
    } else {
        NVIC_ST_RELOAD_R = 200000;
    }
    NVIC_ST_RELOAD_R ^= 100000;
}
```
(10) Question 5 (UART).

a) (3 points) A serial port (UART1) is configured with default settings to run with a bandwidth of 50K bytes/sec. What is the baud-rate of this port in bits/sec?

\[
\text{BW} = \frac{8 \times \text{BR}}{10} = \frac{10 \times 50 \times 10^3 \times 8}{8} = 500,000 \text{ bps}
\]

a) (7 points) Complete the subroutine UART_InString that reads a CR-terminated string from the UART0. The subroutine uses call-by-reference parameter passing. For each character, it waits for new input using busy-wait synchronization. Read the input character and place it in the string passed as input. When a CR is read, insert a Zero (Null) in the string and return. You don’t need to write the UART initialization. The ASCII code for Carriage Return (CR) is 13. You may write the routine in C or Assembly.

**Assembly Code**

; Input; R0 has the address of the location where the read string of characters are to be placed
UART_InString

**C Code**

; Input: str is a pointer to the location where the string of characters read are to be placed
void UART_InString(char *str){
    int i = 0; char ch;
    while((UART0_FR & 0x10) != 0){
        ch = UART0_DR - R
        if (ch == 13) {
            str[i] = 0; break;
        }
        str[i] = ch;
        i ++
    }
}
(10) Question 6 (ADC).
a) (3 points) For a 12-bit ADC with an analog input voltage of 0-3V, what are the following:

(i) ADC precision

\[ 12 \text{ bits} \quad 2^{12} = 4096 \text{ (levels)} \]

(ii) ADC range

input range is 0 to 3V, output range is 0 to 4095

(iii) ADC resolution

\[ \frac{3 \text{ V}}{4095} \]

b) (2 points) What will the above 12-bit ADC return if the input voltage is 1.0V?

\[ \frac{1}{3} \times 4095 \]

c) (5 points) Write an \texttt{ADC0_In} function (in C) that uses busy-wait synchronization to sample the ADC. The function reads the ADC output, and returns the 12-bit binary number. Assume the ADC has already been initialized to use sequencer 3 with a software trigger and channel 1. See supplement pages for ADC registers.

```c
uint32_t ADC0_In(void) {
    while ((ADC_RS-R & 0x08) == 0) {} // \frac{1}{3}
    return (ADC_RS[03]);
}
```
**(15) Question 7 (Hardware)**

a) **(5 points)** For the ADC in the previous question, the input analog voltage is provided by the voltage drop across a resistance consisting of a variable resistor $R$ in series with a resistance $R_s$. The resistance $R_s$ (in series with $R$), is due to the connecting wires, the source resistance and any extraneous effects, and is roughly 10% of $R$.

Draw this external circuit in the box below with the series resistances shown clearly. Mark the source voltage connected across the series resistance connection clearly. Pick any suitable value of $R$. What is the voltage that needs to be connected across the series resistance such that the maximum voltage at the ADC input is 3V?

```
+3.3V

Microcontroller

ADC

R

0.1R

V = \frac{R \times 3.3}{R + 0.1R} = 3V

No need to connect any voltage across the series resistance
```

b) **(10 points)** The desired LED operating point is 1V, 10mA. Interface this LED to PA2 using negative logic. You can use any number of 7406 inverters, and any number of resistors. Assume the $V_{OL}$ of the 7406 is 0.5V. Assume the microcontroller output voltages are $V_{OH} = 3.1V$ and $V_{OL} = 0.2V$. Specify values for any resistors needed. Show equations of your calculations used to select resistor values.

```
Microcontroller

PA2

7406

Microcontroller

7406

R

3.3V

GND

0.5V

10mA

R1 = \frac{3.3 - 0.5}{10mA} = 0.18 k\Omega
```
(10) Question 8 (FIFO).

a) (2 points) What is the most important feature that first-in-first-out (FIFO) offers for I/O devices?

Buffering

b) (3 points) In the FIFO implementation using a dummy slot, what are the checks for Full and Empty FIFO.

\[ \text{Empty} \ : \ (\text{Put} = \text{Get}) \quad \text{Full} \ : \ ((\text{Put} + 1) \mod N = \text{Get}) \]

c) (5 points) You are designing a low-budget embedded systems microcontroller and are told to reuse hardware structures aggressively to keep the costs low. Assume you have multiple stacks in your micro-controller. Explain how you can implement FIFO using only stack(s) that are last-in-first-out (LIFO).

Two stacks A, B

FIFO Add \rightarrow Push to Stack A

FIFO Get \rightarrow Pop all elements from A, push to B until empty
return \text{top}

Pop from B and push to A
(10) Question 9 (FSM). Given the following Moore FSM implementation:

```c
const struct State{
    uint8_t out;  // Output to PT0
    uint8_t wait; // Wait time in 500ns units
    const struct State next[4];  // Next states
};
typedef const struct State StateType;
#define S0 &fsm[0]
#define S1 &fsm[1]
#define S2 &fsm[2]
#define S3 &fsm[3]

StateType fsm[4] = {
    {0x00, 80, {S0, S1, S0, S2}},
    {0x01, 200, {S1, S2, S1, S3}},
    {0x10, 80, {S2, S3, S2, S0}},
    {0x00, 200, {S3, S0, S3, S1}}
};

StateType *cState;  // Current State
```

a. (7 points) Draw a FSM diagram for the implementation provided. The diagram must capture all the information included in the implementation.

![FSM Diagram]

b. (3 points) Assuming, S0 is the initial state, and the 2-bit input is from Port E pins 1 and 0 what output sequence is produced upon this sequence of inputs on PE1-0:
   01, 11, 11, 10, 00, 11

   ![Output Sequence]
Memory access instructions

- **LDR** \( \text{Rd, [Rn]} \): load 32-bit number at \([Rn]\) to \(\text{Rd}\)
- **LDR** \( \text{Rd, [Rn,#off]} \): load 32-bit number at \([Rn]+\text{off}\) to \(\text{Rd}\)
- **LDR** \( \text{Rd, =value} \): load unsigned 16-bit at \([Rn]\) to \(\text{Rd}\)
- **LDRH** \( \text{Rd, [Rn]} \): load unsigned 16-bit at \([Rn]\) to \(\text{Rd}\)
- **LDRH** \( \text{Rd, [Rn,#off]} \): load signed 16-bit at \([Rn]+\text{off}\) to \(\text{Rd}\)
- **LDRB** \( \text{Rd, [Rn]} \): load unsigned 8-bit at \([Rn]\) to \(\text{Rd}\)
- **LDRB** \( \text{Rd, [Rn,#off]} \): load signed 8-bit at \([Rn]+\text{off}\) to \(\text{Rd}\)
- **STR** \( \text{Rt, [Rn]} \): store 32-bit \(\text{Rt}\) to \([Rn]\)
- **STR** \( \text{Rt, [Rn,#off]} \): store 32-bit \(\text{Rt}\) to \([Rn]+\text{off}\)
- **STRH** \( \text{Rt, [Rn]} \): store least sig. 16-bit \(\text{Rt}\) to \([Rn]\)
- **STRH** \( \text{Rt, [Rn,#off]} \): store least sig. 16-bit \(\text{Rt}\) to \([Rn]+\text{off}\)
- **STRB** \( \text{Rt, [Rn]} \): store least sig. 8-bit \(\text{Rt}\) to \([Rn]\)
- **STRB** \( \text{Rt, [Rn,#off]} \): store least sig. 8-bit \(\text{Rt}\) to \([Rn]+\text{off}\)
- **PUSH** \( \{\text{Rt}\} \): push 32-bit \(\text{Rt}\) onto stack
- **POP** \( \{\text{Rd}\} \): pop 32-bit number from stack into \(\text{Rd}\)
- **ADR** \( \text{Rd, label} \): set \(\text{Rd}\) equal to the address at \(\text{label}\)
- **MOV(S)** \( \text{Rd, <op2>} \): set \(\text{Rd}\) equal to \(<\text{op2}>\)
- **MVN(S)** \( \text{Rd, <op2>} \): set \(\text{Rd}\) equal to \(-<\text{op2}>\)

Branch instructions

- **B** \( \text{label} \): branch to \(\text{label}\)  
- **BEQ** \( \text{label} \): branch if \(Z == 1\)  
- **BNE** \( \text{label} \): branch if \(Z == 0\)  
- **BCS** \( \text{label} \): branch if \(C == 1\)  
- **BHS** \( \text{label} \): branch if \(C == 1\)  
- **BCC** \( \text{label} \): branch if \(C == 0\)  
- **BLO** \( \text{label} \): branch if \(C == 0\)  
- **BMI** \( \text{label} \): branch if \(N == 1\)  
- **BPL** \( \text{label} \): branch if \(N == 0\)  
- **BVS** \( \text{label} \): branch if \(V == 1\)  
- **BVC** \( \text{label} \): branch if \(V == 0\)  
- **BHI** \( \text{label} \): branch if \(C==1\) and \(Z==0\)  
- **BLS** \( \text{label} \): branch if \(C==0\) or \(Z==1\)  
- **BGE** \( \text{label} \): branch if \(N == V\)  
- **BLT** \( \text{label} \): branch if \(N != V\)  
- **BGT** \( \text{label} \): branch if \(Z==1\) or \(N!=V\)  
- **BX** \( \text{Rm} \): branch indirect to location specified by \(\text{Rm}\)
- **BL** \( \text{label} \): branch to subroutine at \(\text{label}\)
- **BLX** \( \text{Rm} \): branch to subroutine indirect specified by \(\text{Rm}\)

Interrupt instructions

- **CPSIE** \( \text{I} \): enable interrupts (\(I=0\))
- **CPSID** \( \text{I} \): disable interrupts (\(I=1\))

Logical instructions

- **AND(S)** \( \{\text{Rd}, \text{Rn}, <op2>\} \): \(\text{Rd}==\text{Rn}&<\text{op2}>\) (\(<\text{op2}>\) is 32 bits)
- **ORR(S)** \( \{\text{Rd}, \text{Rn}, <op2>\} \): \(\text{Rd}==\text{Rn}|<\text{op2}>\) (\(<\text{op2}>\) is 32 bits)
- **EOR(S)** \( \{\text{Rd}, \text{Rn}, <op2>\} \): \(\text{Rd}==\text{Rn}^<\text{op2}>\) (\(<\text{op2}>\) is 32 bits)
- **BIC(S)** \( \{\text{Rd}, \text{Rn}, <op2>\} \): \(\text{Rd}==\text{Rn}&(~<\text{op2}>\)) (\(<\text{op2}>\) is 32 bits)
- **ORN(S)** \( \{\text{Rd}, \text{Rn}, <op2>\} \): \(\text{Rd}==\text{Rn}|(~<\text{op2}>\)) (\(<\text{op2}>\) is 32 bits)
- **LSR(S)** \( \text{Rd}, \text{Rm}, \text{Rs} \): logical shift right \(\text{Rd}==\text{Rm}>>\text{Rs}\) (unsigned)
- **LSR(S)** \( \text{Rd}, \text{Rm}, \#n \): logical shift right \(\text{Rd}==\text{Rm}>>\text{n}\) (unsigned)
- **ASR(S)** \( \text{Rd}, \text{Rm}, \text{Rs} \): arithmetic shift right \(\text{Rd}==\text{Rm}>>\text{Rs}\) (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
\(\{S\}\) 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

256k Flash ROM
64k RAM
I/O ports
Internal I/O
PPB

DCB 1,2,3 ; allocates three 8-bit byte(s)
DCW 1,2,3 ; allocates three 16-bit halfwords
DCD 1,2,3 ; allocates three 32-bit words
SPACE 4 ; reserves 4 bytes
Table 4.5. Some TM4C123/LM4F120 parallel ports. Each register is 32 bits wide. Bits 31 – 8 are zero.

<table>
<thead>
<tr>
<th>Address</th>
<th>31-24</th>
<th>23-17</th>
<th>16</th>
<th>15-3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4000.E100</td>
<td></td>
<td>F</td>
<td>...</td>
<td>UART1</td>
<td>UART0</td>
<td>E</td>
<td>D</td>
<td>C</td>
</tr>
</tbody>
</table>

Address 7 6 5 4 3 2 1 0 Name  
$4000.E43FC  DATA  DATA  DATA  DATA  DATA  DATA  DATA  GPIOF GPIOE GPIOD GPIOC GPIOB GPIOA SY SCTL_RCGC_GPIO_R

Address 7 6 5 4 3 2 1 0 Name  
$4000.E4400  DIR  DIR  DIR  DIR  DIR  DIR  DIR  GPIOD DIR GPIO_PORTA_DIR_R

Address 7 6 5 4 3 2 1 0 Name  
$4000.E4420  SEL  SEL  SEL  SEL  SEL  SEL  SEL  GPIOC SEL GPIO_PORTA_AFSEL_R

Address 7 6 5 4 3 2 1 0 Name  
$4000.E451C  DEN  DEN  DEN  DEN  DEN  DEN  DEN  GPIOB DEN GPIO_PORTA_DEN_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.

Table 10.3. The TM4C123/LM4F120ADC registers. Each register is 32 bits wide.

Set MAXADCSPEED 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 11 on the TM4C123/LM4F120. Which channel we sample is configured by writing to the ADC_SSMUX3_R register. The ADC_SSMUX3_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. Write one to ADC_ISC_R to clear the corresponding bit in the ADC_RIS_R register.

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

$$\text{Baud rate} = \text{Baud16/16} = (\text{Bus clock frequency})/(16 \times \text{divider})$$

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 placed 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).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>31–12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7–0</td>
<td>Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OE</td>
<td>BE</td>
<td>PE</td>
<td>FE</td>
<td>DATA</td>
<td>UART0_DR_R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>31–3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>UART0_RSR_R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>31–8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TXFE</td>
<td>RXFF</td>
<td>TXFF</td>
<td>RXFE</td>
<td>BUSY</td>
<td>UART0_FR_R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>31–16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15–0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DIVINT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>31–6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DIVFRAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>31–8</td>
<td>7</td>
<td>6–5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SPS</td>
<td>WPEN</td>
<td>FEN</td>
<td>STP2</td>
<td>EPS</td>
<td>PEN</td>
<td>BRK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>31–10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6–3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RXE</td>
<td>TXE</td>
<td>LBE</td>
<td>SIRLP</td>
<td>SIREN</td>
<td>UARTEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>31–6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5–3</td>
<td>2–0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RXIFLSEL</td>
<td>TXIFLSEL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>31–11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OEIM</td>
<td>BEIM</td>
<td>PEIM</td>
<td>FEM</td>
<td>RTIM</td>
<td>TXIM</td>
<td>RXIM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OERIS</td>
<td>BERIS</td>
<td>PERIS</td>
<td>PERIS</td>
<td>RTRIS</td>
<td>TXRIS</td>
<td>RXRIS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OEMIS</td>
<td>BEMIS</td>
<td>PERIS</td>
<td>PERIS</td>
<td>RTMIS</td>
<td>TXMIS</td>
<td>RXMIS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OEIC</td>
<td>BEIC</td>
<td>PEIC</td>
<td>FEM</td>
<td>RTIC</td>
<td>TXIC</td>
<td>RXIC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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