Final Exam

Date: December 11, 2013

UT EID: ___________________________  Circle one: Gerstlauer or Valvano+Yerraballi

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:

Signature: __________________________________________________________

Instructions:
• Closed book and closed notes. No books, no papers, no data sheets (other than the last two 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.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
</tr>
</tbody>
</table>
(10) Question 1 (Equations/relations you should know).

a) Consider a UART with one start bit, 1 stop bit, \( n \) data bits and no parity bits. The bus frequency is 80 MHz. Give the relationship between baud rate (\( BR \)) and maximum possible bandwidth (\( BW \)), assuming both are in bits/sec.

\[
BR = \frac{f_{bus}}{2n+1}
\]

b) Consider a resistor used to build your DAC in Lab 6. Avogadro’s Number is about \( 6.022 \times 10^{23} \). Give the relationship between resistance (\( R \), in k\( \Omega \)), voltage (\( V \) in volts), and current (\( I \) in mA).

\[
R = \frac{V}{I}
\]

c) Consider the sampling rate chosen for the ADC in Lab 8. Give the relationship for the slowest possible sampling rate (\( f_s \), in Hz), given these parameters: ADC resolution (\( \Delta V \), in volts), number of ADC bits (\( n \), in bits, e.g., 12 bits) and rate at which one moves the slide pot (\( r \), in oscillations per sec).

\[
s_s = \frac{2^n}{r}
\]

d) Sketch the current versus voltage curve of an LED like the ones used in lab. Include the (2V,10mA) operating point, but roughing sketch the other points from 0 to 3 volts.

![LED Current-Voltage Curve]

\[
\text{current (mA)}
\]

\[
\text{voltage (V)}
\]

\[
0 \quad 1 \quad 2 \quad 3
\]

\[
0 \quad 5 \quad 10 \quad 15
\]

e) Consider a two-dimensional array of half words (16 bits each), with \( n \) rows and \( m \) columns. The bus frequency is 80 MHz. The base address of the array is \( b \), and the array is defined in row major order. What is the address of the element in row \( i \) and column \( j \)?

\[
\text{Address} = b + (i \times m + j) \times 16
\]
Question 2 (Local Variables). The assembly subroutine below uses **three** local variables. Demonstrate your understanding of local variables in assembly by answering the following questions. You may assume the initial stack pointer is 0x20001008, no registers other than R0-R3 are used, and all three local variables are allocated on the stack.

<table>
<thead>
<tr>
<th>Assembly</th>
<th>C Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxx equ aa</td>
<td>void Locals(void) {</td>
</tr>
<tr>
<td>yyy equ bb</td>
<td>long xxx;</td>
</tr>
<tr>
<td>zzz equ cc</td>
<td>long yyy;</td>
</tr>
<tr>
<td>Locals</td>
<td>long zzz;</td>
</tr>
<tr>
<td>; Body of subroutine</td>
<td>// Body of subroutine</td>
</tr>
<tr>
<td>BX LR</td>
<td>}</td>
</tr>
</tbody>
</table>

a) (2 points) Which of the following is **not** relevant to the use of local variables?
   - Binding using equ pseudo-ops
   - Allocation on stack
   - Parameter-passing to the subroutine
   - Indexed access of the stack with SP as the base.
   - Deallocation by restoring the SP

b) (2 points) What will the value of the SP be after allocating space for all three?

   0x20001011

c) (3 points) Assuming that the three variables are allocated space in the order in which they appear **xxx** at higher address, **yyy** in the middle and **zzz** at the lower address. Lower address means smaller value than higher address. The values of **aa**, **bb** and **cc** are:
   - **aa** is 8; **bb** is 4; **cc** is 0
   - **aa** is 0; **bb** is 1; **cc** is 2
   - **aa** is 2; **bb** is 1; **cc** is 0
   - **aa** is 0; **bb** is 4; **cc** is 8
   - **aa** is 1; **bb** is 2; **cc** is 4
   - None of the above

d) (3 points) Assuming that the correct values for **bb** is set. Which of the sequences of instructions will add 1 to the local variable **yyy**.
   - LDRSB R0,[SP,#yyy]; ADD R0,R0,#1; STRB R0,[SP,#yyy]
   - LDR R0,[SP,yyy]; ADD R0,R0,#1; STR R0,[SP,yyy]
   - ADD [SP,yyy];#1;
   - LDRSH R0,[SP,yyy]; ADD R0,R0,#1; STRH R0,[SP,yyy]
   - None of the above
(15) Question 3 (C Programming with struct). Given the following struct declaration for a student, complete the subroutine which (a) calculates each student’s grade as 'P' or 'F' depending on whether the score is higher than or equal to 75, and (b) returns the average class score;

```c
#define SIZE 64
struct Student {
    unsigned long id;
    unsigned long score;
    unsigned char grade; // you will enter 'P' or 'F'
};
typedef struct Student STyp;

unsigned long Grades(STyp class[SIZE]){
    long i, avg=0;
    for(i=0; i < SIZE; i++){
        if (class[i].score >= 70){
            class[i].grade = 'P';
        } else {
            class[i].grade = 'F';
        }
        avg += class[i].score;
    }
    return(avg>>6);
}
```
(10) Question 4 (Interrupts).

a) (3 points) An Interrupt Service Routine executes the last line of its code, a return statement (BX LR). Which of the following registers are not popped from the stack? Put all the letters in the box that apply. For example, if you think i, ii are not popped, but iii, iv, v are popped, enter i+ii.
   i.  R0-R3
   ii. R12
   iii. LR
   iv. PC, SP, PSR
   v.  R4-R11

b) (5 points) Assume the bus clock is operating at 80 MHz. The SysTick initialization executes these instructions. SysTick will be used to generate a periodic interrupt with an interrupt period of 100μs (which is 10 kHz.) What assembly instructions go in the ????(a)???? and ????(b)???? places?

SysTick_Init
   LDR R1,=NVIC_ST_RELOAD_R
   STR R0,[R1]
   LDR R1,=NVIC_ST_CTRL_R
   STR R2,[R1]
   BX LR

   ????(a)????
   ????(b)????

(c) (2 points) All Interrupt Service Routines with the exception of SysTick_Handler must do this:
   i. Explicitly pop the SP before returning from the interrupt
   ii. Not use the Stack
   iii. Write to a FIFO
   iv. Explicitly acknowledge the Interrupt
   v. Write to a mailbox
   vi. None of the above
(10) Question 5 (UART).

a) (2 points) A programmer set the UART0_IBRD_R to 50 and UART0_FBRD_R to 0. If the Bus clock frequency is 80MHz, what is the baud rate? (bps stands for bits per second)
   i.  80 kbps
   ii. 100 kbps
   iii. 1 Mbps
   iv. 120 kbps
   v.  16 kbps
   vi. None of the above

b) (5 points) Assuming now a serial port operating with a baud rate of 2000 bits per second. The protocol is 1 start, 8 data and 1 stop bit. Draw the waveform when the decimal value 204 is transmitted. You may assume the channel is idle before and after the frame. Time flows from left to right.

![Waveform Diagram](image)

(c) (3 points) Assume the serial port is setup as in part b) (2000 bps, 1 stop, 8 data, 1 start bit) and the serial receive interrupt is set to trigger when the UART receive FIFO is half full. Furthermore, the receive interrupt handler empties the UART FIFO every time it is invoked. How long can interrupts at most be disabled to guarantee that no UART overflow (OE bit set) will occur.
(15) Question 6. (Hardware)

a) (5 points) Design a 6-bit DAC using the binary-weighted configuration. The DAC is controlled by six output port pins, PE5-0, where PE0 is the least significant and PE5 is the most significant bit. Carefully label the signal which is the DAC output, and specify the values for any resistors used.

![Diagram of a 6-bit DAC with output ports PE5 to PE0 labeled and a DAC output label DACout.]

b) (5 points) The desired LED operating point is 1V, 1mA. 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. Interface this LED to PA2 using positive logic. Full credit for the solution uses the fewest components, and partial credit if it works. Specify values for any resistors needed. Show equations of your calculations used to select resistor values.

![Diagram of a microcontroller connected to PA2, 7406, and a LED with equations for resistor calculations.]

c) (5 points) Interface a switch to PA3 such that if the switch is pressed the software sees a logic 0 and if the switch is not pressed the software sees a logic one. Specify values for any resistors needed. The software will clear both the internal pullup and pulldown registers.

![Diagram of a microcontroller connected to PA3 with a switch and equations for resistor calculations.]

Gerstlauer, Valvano, Yerraballi      December 11, 2013       7:00pm-10:00pm
(10) Question 7 (ADC). Assume the ADC has already been initialized to use sequencer 3 with a software trigger and channel 1. Write a C function that starts the ADC, waits for it to complete, reads the 12-bit result, clears the flag and returns the measured voltage (in the ADC’s range of 0-3V) as a value with units of mV. For example if the input is 1.234V then the software will return 1234. The prototype for this function is

```c
unsigned long ADC0_InSeq3(void);
```
(10) Question 8 (FIFO).

a) (2 points) Why is the first in first out (FIFO) queue really important for interfacing I/O devices?
   i) They can store data permanently, which is important because embedded systems are used in safety
      critical situations, and we need to know what the data was during operation.
   ii) They are a way to store data in the cloud. FIFOs provide backup and sharing.
   iii) The software and hardware can operate at variable speeds and data are temporarily spooled into the
        FIFO as it passes between them.
   iv) It can store an arbitrarily large amount of data. This is important
       because the size and complexity of embedded systems is growing.
   v) None of the above.
   vi) All of the above.

b) (5 points) Circle all the bugs in this FIFO implementation, and show the corrections needed to make
    this FIFO functional.

```c
unsigned char static PutI;
unsigned char static GetI;
short static FIFO[16]; // 16 halfwords or 32 bytes of data
void Fifo_Init(void){
   PutI = GetI = 5;
}
int Fifo_Put(short data){
   if(((PutI+1)&0x1F) == GetI) return 0;
   FIFO[PutI] = data;
   PutI = (PutI+1)&0x1F;
   return 1;
}
int Fifo_Get(short *datapt){
   if(PutI == GetI) return 0;
   *datapt = FIFO[GetI];
   GetI = (GetI+1)&0x1F;
   return 1;
}
```

c) (3 points) Assuming a SysTick handler calls Fifo_Put() every 1 ms and the main() is able to
   process one item every 2 ms. If the main() is processing the first item right after the first SysTick puts
   it into the FIFO at time 0, does the FIFO ever overflow and if so, at what time?
(10) Question 9 (FSM).

a) (4 points) Assume we start in the happy state. The input starts and remains 3. What sequence of outputs will occur?
   i) start and remain at 10
   ii) 10, 12, 0 (and remain at 0)
   iii) 10, 0, 10, 0, 10, 0, 10, 0, 10, 0, 10, 0, 10, over and over
   iv) 10, 12, 10, 12, 10, 12, 10, 12, 10, over and over
   v) None of the above

b) (6 points) Consider this FSM, the 6-bit output is on Port B (PB5-0) and the 2-bit input on Port E (PE1-0). You may call the SysTick function $\texttt{SysTick\_Wait10ms}(n)$; to wait $n*10$msec.

```
struct State {
  unsigned long Out;
  unsigned long Time;
  const struct State *Next[4];
} typedef const struct State STyp;
#define goN   &FSM[0]
#define waitN &FSM[1]
#define goE   &FSM[2]
#define waitE &FSM[3]
STyp FSM[4] = {
  {0x21,3000,{goN,waitN,goN,waitN}},
  {0x22, 500,{goE,goE,goE,goE}},
  {0x0C,3000,{goE,goE,waitE,waitE}},
  {0x14, 500,{goN,goN,goN,goN}}};
STyp *Pt;
int main(void){
  SysTick_Init(); // this function is given which initializes SysTick
  Port_Init();    // this function is given which initializes B E
  Pt = goN;
```

Write C code that completes this main program such that the FSM runs in the foreground. Write friendly code. You can assume that $\texttt{GPIO\_PORTB\_DATA\_R}$ and $\texttt{GPIO\_PORTE\_DATA\_R}$ have been defined.
Memory access instructions

LDR    Rd, [Rn]       ; load 32-bit number at [Rn] to Rd
LDR    Rd, [Rn,#off]  ; load 32-bit number at [Rn+offset] 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+offset] to Rd
LDRB   Rd, [Rn]       ; load unsigned 8-bit at [Rn] to Rd
LDRB   Rd, [Rn,#off]  ; load unsigned 8-bit at [Rn+offset] to Rd
STR    Rt, [Rn]       ; store 32-bit Rt to [Rn]
STR    Rt, [Rn,#off]  ; store 32-bit Rt to [Rn+offset]
STRH   Rt, [Rn]       ; store least sig. 16-bit Rt to [Rn]
STRH   Rt, [Rn,#off]  ; store least sig. 16-bit Rt to [Rn+offset]
STRB   Rt, [Rn]       ; store least sig. 8-bit Rt to [Rn]
STRB   Rt, [Rn,#off]  ; store least sig. 8-bit Rt to [Rn+offset]

Branch instructions

B    label   ; branch to label    Always
BEQ  label   ; branch if Z == 1    Equal
BNE  label   ; branch if Z == 0    Not equal
BHS  label   ; branch if C == 1    Higher or same, unsigned ≥
BLO  label   ; branch if C == 0    Lower, unsigned <
BMI  label   ; branch if N == 1    Negative
BLE  label   ; branch if Z==1 or N!=V Less than or equal, signed ≤
BX   Rm      ; branch indirect to location 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)

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)
### Arithmetic Instructions

- **ADD** \( \{Rd,\} Rn, \langle op2 \rangle \); \(Rd = Rn + op2\)
- **ADD** \( \{Rd,\} Rn, \#im12 \); \(Rd = Rn + im12\), \(im12\) is 0 to 4095
- **SUB** \( \{Rd,\} Rn, \langle op2 \rangle \); \(Rd = Rn - op2\)
- **SUB** \( \{Rd,\} Rn, \#im12 \); \(Rd = Rn - im12\), \(im12\) is 0 to 4095
- **RSB** \( \{Rd,\} Rn, \langle op2 \rangle \); \(Rd = op2 - Rn\)
- **RSB** \( \{Rd,\} Rn, \#im12 \); \(Rd = im12 - Rn\)
- **CMP** \( Rn, \langle op2 \rangle \); \(Rn - op2\) sets the NZVC bits
- **CMN** \( Rn, \langle op2 \rangle \); \(Rn - (-op2)\) sets the NZVC bits
- **MUL** \( \{Rd,\} Rn, Rm \); \(Rd = Rn * Rm\) 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
- \langle op2 \rangle \ 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 \(0xXYXYXYXY\)

### General Purpose Registers

- R0
- R1
- R2
- R3
- R4
- R5
- R6
- R7
- R8
- R9
- R10
- R11
- R12
- R13 (MSP)
- R14 (LR)
- R15 (PC)

### Condition Code Bits

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

### Memory Map

- **256k Flash ROM**
  - From 0x0000.0000 to 0x0003.FFFF
  - From 0x2000.0000 to 0x2000.FFFF
  - From 0x4000.0000 to 0x41FF.FFFF
  - From 0xE000.0000 to 0xE004.0FFF
- **64k RAM**
  - From 0x0200.0000 to 0x0200.FFFF
  - From 0x0400.0000 to 0x41FF.FFFF
- **I/O ports**
  - From 0xE000.0000 to 0xE004.0FFF

### Memory Access

- **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
ADC conversion, and clear it when no flags are needed. We will set
we just need to make sure each sequencer has a unique priority. We set bits 15–12 (ADC_PSSI_R)
conversion is complete. We can enable and disable the sequencers using the
the
TM4C123/LM4F120. Which channel we sample is configured by writing to the
register specifies the mode of the ADC sample. Clear
Set MAXADCSPD to 00 for slow speed operation. The ADC has four sequencers, but we will use only sequencer 3. We set

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 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-bit \( \text{ADC_PSSI}_R \) 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.

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 \(\frac{\text{Bus clock frequency}}{\text{divider}}\). The baud rate is

\[
\text{Baud rate} = \frac{\text{Baud16}}{16} = \left(\frac{\text{Bus clock frequency}}{16\times\text{divider}}\right)
\]

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 \(\geq \frac{1}{2}\) full, or 7 to 8 characters). TXIFLSEL specifies the transmit FIFO level that causes an interrupt (010 means interrupt on \(\leq \frac{1}{2}\) full, or 9 to 8 characters).

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

<table>
<thead>
<tr>
<th>$4000.C000</th>
<th>31-12 11 10 9 8 3 7-0 Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>OE</td>
<td>BE</td>
</tr>
<tr>
<td>$4000.C004</td>
<td>31-3 3 2 1 0</td>
</tr>
<tr>
<td>OE</td>
<td>BE</td>
</tr>
<tr>
<td>$4000.C018</td>
<td>31-8 7 6 5 4 3 2-0</td>
</tr>
<tr>
<td>TXFE</td>
<td>RXFF</td>
</tr>
<tr>
<td>$4000.C024</td>
<td>31-16 15-0</td>
</tr>
<tr>
<td>DIVINT</td>
<td></td>
</tr>
<tr>
<td>$4000.C028</td>
<td>31-6 5-0</td>
</tr>
<tr>
<td>DIVFRAC</td>
<td></td>
</tr>
<tr>
<td>$4000.C02C</td>
<td>31-8 7 6-5 4 3 2 1 0</td>
</tr>
<tr>
<td>SPS</td>
<td>WPEN</td>
</tr>
<tr>
<td>$4000.C030</td>
<td>31-10 9 8 7 6-3 2 1 0</td>
</tr>
<tr>
<td>RXE</td>
<td>TXE</td>
</tr>
<tr>
<td>$4000.C034</td>
<td>31-6 5-3 2-0</td>
</tr>
<tr>
<td>RXIFLSEL</td>
<td>TXIFLSEL</td>
</tr>
<tr>
<td>$4000.C038</td>
<td>31-11 10 9 8 7 6 5 4</td>
</tr>
<tr>
<td>OEM</td>
<td>BEIM</td>
</tr>
<tr>
<td>$4000.C03C</td>
<td>OEMIS</td>
</tr>
<tr>
<td>$4000.C040</td>
<td>OEMIS</td>
</tr>
<tr>
<td>$4000.C044</td>
<td>OEMIC</td>
</tr>
</tbody>
</table>