Final Exam

Date: May 11, 2013

UT EID: ________________

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 calculators or any electronic devices (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.
- For all questions, unless otherwise stated, find the most efficient (time, resources) solution.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>15</td>
</tr>
<tr>
<td>Problem 2</td>
<td>25</td>
</tr>
<tr>
<td>Problem 3</td>
<td>20</td>
</tr>
<tr>
<td>Problem 4</td>
<td>15</td>
</tr>
<tr>
<td>Problem 5</td>
<td>25</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
</tr>
</tbody>
</table>
Problem 1 (15 points): Subroutines

Given the following C functions:

```c
long max1(long a[2]) {
    long m;
    m = a[0];
    if (a[1] > m) m = a[1];
    return m;
}

void max2(long a, long b) {
    long data[2];
    data[0] = a; data[1] = b;
    return max1(data);
}
```

(a) Assume a calling convention in which only the first parameter of a function is passed via register R0. All other parameters are passed via the stack and functions can freely use registers R0-R3. Complete the partial assembly code generated for these two functions:

```
AREA    | .text |, CODE, READONLY, ALIGN=2

max1
LDR    R1,[R0]
LDR    R2,[R0,________]
CMP    R1,R2
      n1
MOV    R1,R2
n1    MOV    R0,R1
      BX    LR

b    EQU  __________  ; input parameter ‘b’
dat0 EQU  __________  ; local variable ‘data[0]’
dat1 EQU  __________  ; local variable ‘data[1]’

max2
_______________________ ; save registers
_______________________ ; allocate ‘data’
MOV    R11,SP
STR    R0,[R11,#dat0] ; store first ‘data’ element
LDR    R0,[R11,#c]    ; load ‘b’
STR    R0,[R11,#dat1] ; store second ‘data’ element
ADD    R0,R11,#dat0   ; pass address to ‘data’
BL     max1
_______________________ ; de-allocate ‘data’
_______________________ ; restore registers
      BX    LR
```
(b) Assume that \textit{max2} is called with parameters \textit{max2(42,-14)}. Show the contents of the stack frame of \textit{max2} at the point right before the \texttt{BL max1} instruction gets executed. Mark any allocated but uninitialized stack items with a ‘?’.

Indicate the location of both stack and frame pointer. Each entry below corresponds to a 32-bit word.

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

Problem 2 (25 points): Serial Communication

(a) Assume you are observing the following waveform on an oscilloscope attached to a serial communication line. Assuming that the line was idle before, mark the frame boundaries and indicate the start, stop and data bits within each observed frame.

![Waveform Diagram](image-url)
(b) What is the baud rate and bandwidth of the observed communication channel?

<table>
<thead>
<tr>
<th>Baud rate</th>
<th>Bandwidth (bytes/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(c) Assuming that ASCII characters are transmitted over the line, what partial message have we observed so far?

(d) Assume that you want to implement a receiver that taps into the communication line to observe and decode transmitted messages. Given the code template for the UART initialization function below, fill in the blanks to complete the initialization code such that it matches communication requirements and enables the receive FIFO with interrupts on ¼ full and when idle. Assume that the systems is running at an 8MHz bus clock.

```c
void UART0_Init(void) {
    SYSCTL_RCGC1_R |= 0x0001;
    SYSCTL_RCGC2_R |= 0x0001;
    UART0_CTL_R &= ~0x0001;
    UART0_IBRD_R __________________
    UART0_FBRD_R __________________
    UART0_LCRH_R = 0x0070;
    UART0_IFLS_R &~ ~0x38;
    UART0_IIFLS_R | = 0x80;
    UART0_IM_R __________________
    UART0_CTL_R __________________
    GPIO_PORTA_AFSEL_R |= 0x03;
    GPIO_PORTA_DEN_R | = 0x03;
    NVIC_PRI1_R = (NVIC_PRI1_R & 0xFFFF00FF) | 0x00004000;
    NVIC_EN0_R | = NVIC_EN0_INT5;
}
```
(e) Write the code for the UART interrupt handler. The handler is supposed to read ASCII characters from the UART and put them into a global software FIFO until the UART receive FIFO is empty. You can ignore errors (full conditions) of the software FIFO.

```c
void UART0_Handler(void) {
    FIFO_Put(data);
}
```

---

**Problem 3 (20 points): Finite State Machine**

You are asked to implement a Mealy FSM that recognizes a certain pattern of ASCII characters received over the serial port (see Problem 2). The FSM processes characters by reading from the software FIFO at the beginning of each state. It outputs a ‘1’ for one character duration every time the pattern is received in the input stream. The output of the FSM should be written to an LED attached to PA0. Given the following state diagram:

(a) What input pattern does the FSM recognize?
(b) Define the C state structure to use in the FSM as well as the FSM array to encode the given machine. Hint: there is a naïve and a smart way to encode input dependencies.

```c
struct state {
    // Define the C state structure to use in the FSM
    typedef const struct state_stateType;

    #define S0 _____________________
    #define S1 _____________________
    #define S2 _____________________

    stateType FSM[3] = {
        // FSM array to encode the given machine
    };
}
```

(c) Fill in the blanks to write the main program that implements the recognizer.

```c
#define PA0 (*((volatile unsigned long *)0x40004004))

stateType *current = S0;

void main(void) {
    PLL_Init();
    UART0_Init();
    PortA_Init();

    while(1) {
        FIFO_Get(&input);
    }
}
```
Problem 4 (15 points): Digital to Analog Conversion

(a) Design a 3-bit binary-weighted DAC interfaced to PB2 through PB0 using 3kΩ, 6kΩ and 12kΩ resistors. What is the output voltage Vout when applying bit patterns 111 and 101?

(b) What is the voltage at Vout for the bit pattern 111 if a load of 12kΩ is applied between Vout and ground?
Problem 6 (25 points): Communication System

You are asked to implement the receiver side of a communication system that uses sound to wirelessly transmit a binary signal. The signal is transmitted by modulating a sound wave at 1000Hz such that the sound is on when the binary digit being transmitted is one and off otherwise (so-called Binary Amplitude Shift Keying, BASK modulation).

(a) Assume that the microphone can convert air pressure changes of sound waves into a voltage between 0 and 3V with a precision corresponding to only 4 bits. What is the resolution of the system and what 4-bit value will get sampled for a microphone voltage of 2.25V?

<table>
<thead>
<tr>
<th>Resolution (V)</th>
<th>Sampled value at 2.25V</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) What is the minimum rate at which the receiver needs to sample the sound signal in order to be able to properly decode it?
(c) In order to improve decoding and demodulation, we want to oversample the received sound signal at a rate of 32000Hz. Given the template below for a system running at a bus clock of 8MHz, fill in the blanks to complete the SysTick initialization routine to set the SysTick interrupt priority to 2 and trigger SysTick interrupts at the desired sampling rate. Make sure your code is friendly.

```
SysTick_Init
  LDR R1, =NVIC_ST_CTRL_R
  MOV R0, #0
  STR R0, [R1]
  LDR R1, =NVIC_ST_RELOAD_R
  ___________________________
  STR R0, [R1]
  LDR R1, =NVIC_SYS_PRI3_R
  LDR R0, [R1]
  ___________________________
  STR R0, [R1]
  LDR R1, =NVIC_ST_CTRL_R
  ___________________________
  STR R0, [R1]
  BX  LR
```

(d) Finally, write the `SysTick_Handler` that calls an `ADC_In` routine to acquire 4-bit samples and collect it into a global `Samples` array. Every time 32 new samples have been collected, the handler is supposed to call a `Demodulate` function and put a zero or one into a global mailbox depending on whether the `Demodulate` result was greater than the predefined THRESHOLD. `ADC_In` and `Demodulate` functions are declared externally as follows:

ADC.h:
```
// Acquire 4-bit sample
unsigned char ADC_In(void);
```

Demodulate.h:
```
// Demodulate signal of 32 samples into amplitude
unsigned short Demodulate(unsigned char signal[32]);
```

Make sure that your code defines and uses the following variables:
- All mailbox-related variables should be public and permanently allocated.
- `Samples` should be a permanently allocated array that is private to the `SysTick.c` file.
- `Cur` should be a permanently allocated variable that is private to the `SysTick_Handler`. It should be initialized to zero and used to index into the `Samples` array.
- Any other necessary variables should be temporary and private to the handler.
SysTick.h

```c
void SysTick_Handler(void);
```

SysTick.c

```c
#define THRESHOLD  6000
#include “ADC.h”
#include “Demodulate.h”
#include “SysTick.h”

void SysTick_Handler(void) {
}
```
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
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 == 0 No 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 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, #n ; 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

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

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., \(Rd = Rn+op2\)

\[
\begin{align*}
\text{ADD} & Rd, Rn, Rm ; op2 = Rm \\
\text{ADD} & Rd, Rn, Rm, LSL \#n ; op2 = Rm<<n Rm is signed, unsigned \\
\text{ADD} & Rd, Rn, Rm, LSR \#n ; op2 = Rm>>n Rm is unsigned \\
\text{ADD} & Rd, Rn, Rm, ASR \#n ; op2 = Rm>>n Rm is signed \\
\text{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 \(0xXYXYXYXY\)
\]

\[\begin{array}{c|c}
\text{Condition code bits} & \text{256k Flash} \\
N \text{ negative} & 0x0000.0000 \downarrow \\
Z \text{ zero} & 0x000.0000 \\
V \text{ signed overflow} & 0x0003.FFFF \\
C \text{ carry or} & \text{unsigned overflow} \\
\end{array}\]

\[\begin{array}{c|c}
\text{General purpose registers} & \text{64k RAM} \\
R0 & 0x0200.0000 \downarrow \\
R1 & 0x4000.0000 \\
R2 & 0x41FF.FFFF \\
R3 & \text{I/O ports} \\
R4 & \text{Internal I/O} \\
R5 & \text{PPB} \\
R6 & 0xE000.0000 \downarrow \\
R7 & 0xE004.0FFF \\
R8 & \text{Stack pointer} \\
R9 & \text{Link register} \\
R10 & \text{Program counter} \\
R11 & \text{R13 (MSP)} \\
R12 & \text{R14 (LR)} \\
R13 & \text{R15 (PC)} \\
\end{array}\]
We set the direction register (e.g., \texttt{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., \texttt{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., \texttt{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 \texttt{RELOAD} register. The frequency of the periodic interrupt will be \( f_{BUS} / (n + 1) \). First, we clear the \texttt{ENABLE} bit to turn off SysTick during initialization. Second, we set the \texttt{RELOAD} register. Third, we write to the \texttt{NVIC\_ST\_CURRENT\_R} value to clear the counter. Lastly, we write the desired mode to the control register, \texttt{NVIC\_ST\_CTRL\_R}. To turn on the SysTick, we set the \texttt{ENABLE} bit. We must set \texttt{CLK\_SRC}=1, because \texttt{CLK\_SRC}=0 external clock mode is not implemented on the LM3S/LM4F family. We set \texttt{INTEN} to enable interrupts. The standard name for the SysTick ISR is \texttt{SysTick\_Handler}.

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 \texttt{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 (\texttt{EM3}) in the \texttt{ADC\_EMUX\_R} register to specify how the ADC will be triggered. If we specify software start (\texttt{EM3}=0x0), then the software writes an 8 (\texttt{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_SSCCTL3_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 ENDO 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 $2^{-22}$. The Baud16 clock is created from the system bus clock, with a frequency of (Bus clock frequency)/divider. The baud rate is

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

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>Address</th>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4000.C000</td>
<td>31–12</td>
<td>OE</td>
<td>Receive enable</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>BE</td>
<td>Baud rate</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>PE</td>
<td>Framing error</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>FE</td>
<td>Data</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7–0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$4000.C004</td>
<td>31–3</td>
<td>OE</td>
<td>Receive enable</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>BE</td>
<td>Baud rate</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>PE</td>
<td>Framing error</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>FE</td>
<td>Data</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$4000.C018</td>
<td>31–8</td>
<td>TXFE</td>
<td>Transmission empty</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>RXFF</td>
<td>Receive full</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>TXFF</td>
<td>Transmission full</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>RXFE</td>
<td>Receive error</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>Busy</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2–0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$4000.C024</td>
<td>31–16</td>
<td>DIVINT</td>
<td>Divide</td>
</tr>
<tr>
<td></td>
<td>15–0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$4000.C028</td>
<td>31–6</td>
<td>D16FRAC</td>
<td>Divide fraction</td>
</tr>
<tr>
<td></td>
<td>5–0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$4000.C02C</td>
<td>31–8</td>
<td>SPS</td>
<td>Start bit</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>WPEN</td>
<td>Write enable</td>
</tr>
<tr>
<td></td>
<td>6–5</td>
<td>FEN</td>
<td>Frame error</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>STP2</td>
<td>Stop bit 2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>EPS</td>
<td>Error stop</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>PEN</td>
<td>Parity enable</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>BRK</td>
<td>Break</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$4000.C030</td>
<td>31–10</td>
<td>RXE</td>
<td>Receive enable</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>TXE</td>
<td>Transmit enable</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>LBE</td>
<td>Leading bit enable</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td></td>
<td>SIRLP</td>
</tr>
<tr>
<td></td>
<td>6–3</td>
<td></td>
<td>SIREN</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>UARTEN</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$4000.C034</td>
<td>31–6</td>
<td>RXIFLSEL</td>
<td>Receive FIFO level</td>
</tr>
<tr>
<td></td>
<td>5–3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2–0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$4000.C038</td>
<td>31–11</td>
<td>OEIM</td>
<td>Receive enable</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>BEIM</td>
<td>Baud rate</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>PEIM</td>
<td>Framing error</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>FEIM</td>
<td>Data</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6–5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

## BITS 4 to 6

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0</strong></td>
<td>NUL</td>
<td>DLE</td>
<td>SP</td>
<td>0</td>
<td>@</td>
<td>P</td>
<td>`</td>
</tr>
<tr>
<td><strong>1</strong></td>
<td>SOH</td>
<td>DC1</td>
<td>!</td>
<td>1</td>
<td>A</td>
<td>Q</td>
<td>a</td>
</tr>
<tr>
<td><strong>2</strong></td>
<td>STX</td>
<td>DC2</td>
<td>&quot;</td>
<td>2</td>
<td>B</td>
<td>R</td>
<td>b</td>
</tr>
<tr>
<td><strong>3</strong></td>
<td>ETX</td>
<td>DC3</td>
<td>#</td>
<td>3</td>
<td>C</td>
<td>S</td>
<td>c</td>
</tr>
<tr>
<td><strong>4</strong></td>
<td>EOT</td>
<td>DC4</td>
<td>$</td>
<td>4</td>
<td>D</td>
<td>T</td>
<td>d</td>
</tr>
<tr>
<td><strong>5</strong></td>
<td>ENQ</td>
<td>NAK</td>
<td>%</td>
<td>5</td>
<td>E</td>
<td>U</td>
<td>e</td>
</tr>
<tr>
<td><strong>6</strong></td>
<td>ACK</td>
<td>SYN</td>
<td>&amp;</td>
<td>6</td>
<td>F</td>
<td>V</td>
<td>f</td>
</tr>
<tr>
<td><strong>7</strong></td>
<td>BEL</td>
<td>ETB</td>
<td>'</td>
<td>7</td>
<td>G</td>
<td>W</td>
<td>g</td>
</tr>
<tr>
<td><strong>8</strong></td>
<td>BS</td>
<td>CAN</td>
<td>(</td>
<td>8</td>
<td>H</td>
<td>X</td>
<td>h</td>
</tr>
<tr>
<td><strong>9</strong></td>
<td>HT</td>
<td>EM</td>
<td>)</td>
<td>9</td>
<td>I</td>
<td>Y</td>
<td>i</td>
</tr>
<tr>
<td><strong>A</strong></td>
<td>LF</td>
<td>SUB</td>
<td>*</td>
<td>:</td>
<td>J</td>
<td>Z</td>
<td>j</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>VT</td>
<td>ESC</td>
<td>+</td>
<td>;</td>
<td>K</td>
<td>[</td>
<td>k</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>FF</td>
<td>FS</td>
<td>,</td>
<td>&lt;</td>
<td>L</td>
<td>\</td>
<td>l</td>
</tr>
<tr>
<td><strong>D</strong></td>
<td>CR</td>
<td>GS</td>
<td>-</td>
<td>=</td>
<td>M</td>
<td>]</td>
<td>m</td>
</tr>
<tr>
<td><strong>E</strong></td>
<td>SO</td>
<td>RS</td>
<td>.</td>
<td>&gt;</td>
<td>N</td>
<td>^</td>
<td>n</td>
</tr>
</tbody>
</table>