# 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.***

| | | |
|---|---|---|
| **Problem 1** | 10 | |
| **Problem 2** | 10 | |
| **Problem 3** | 15 | |
| **Problem 4** | 10 | |
| **Problem 5** | 10 | |
| **Problem 6** | 10 | |
| **Problem 7** | 15 | |
| **Problem 8** | 10 | |
| **Problem 9** | 10 | |
| **Total** | 100 | |

**(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.

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

**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.

**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?

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

**(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`".

**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.

```
___________ LCDOut(_____________________________) {









}
```

**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.

```
void main() {
    MyString outStr; // Assume outStr already initialized









}
```

**(10) Question 4 (Interrupts).**
Using SysTick Interrupt only to generate the following signal on Port E pin 2.


**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?

| |
|---|
| NVIC_ST_CTRL_R |
| NVIC_ST_RELOAD_R |
| NVIC_ST_CURRENT_R |

**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.

```
void SysTick_Handler() {




}
```

**(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?

**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){
```

**(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

(ii) ADC range

(iii) ADC resolution

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

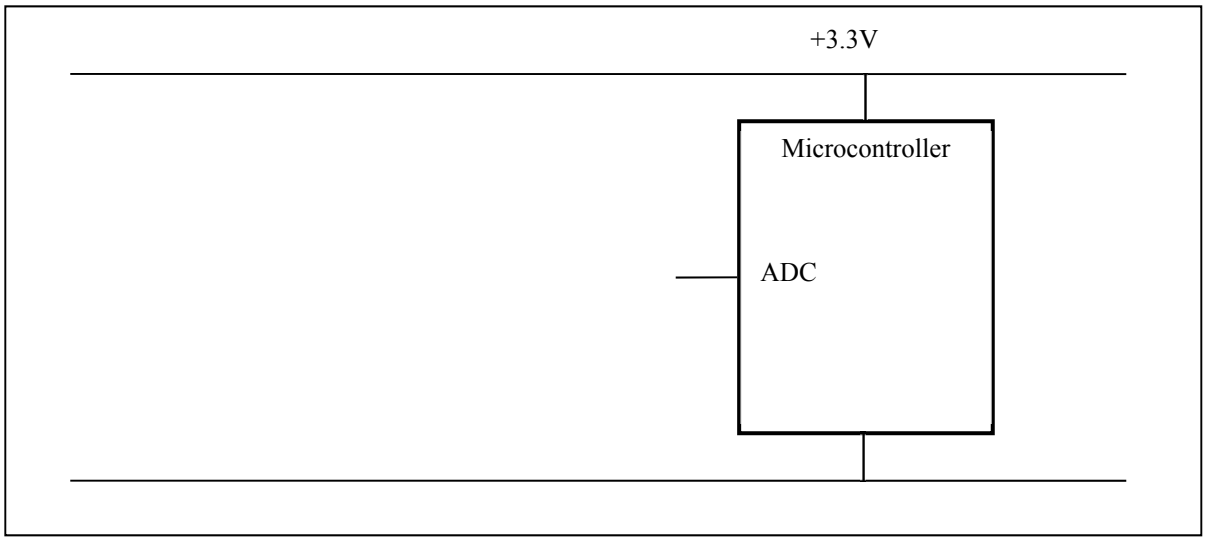
**c) (5 points)** Write an ***ADC0_In*** function (in C) that uses busy-wait synchronization to sample the ADC. The function reads the ADC output, and returns the 12-bit binary number. Assume the ADC has already been initialized to use sequencer 3 with a software trigger and channel 1. See supplement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

```
uint32_t ADC0_In(void) {
```

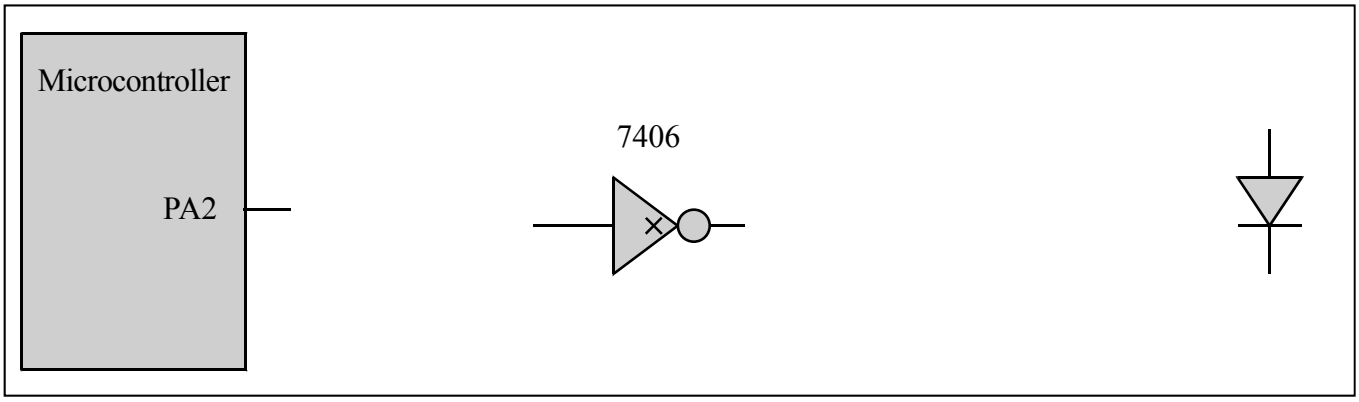
**(15) Question 7 (Hardware)**

**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 Rs. The resistance Rs (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?




**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.



**(10) Question 8 (FIFO).**

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

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

**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)?

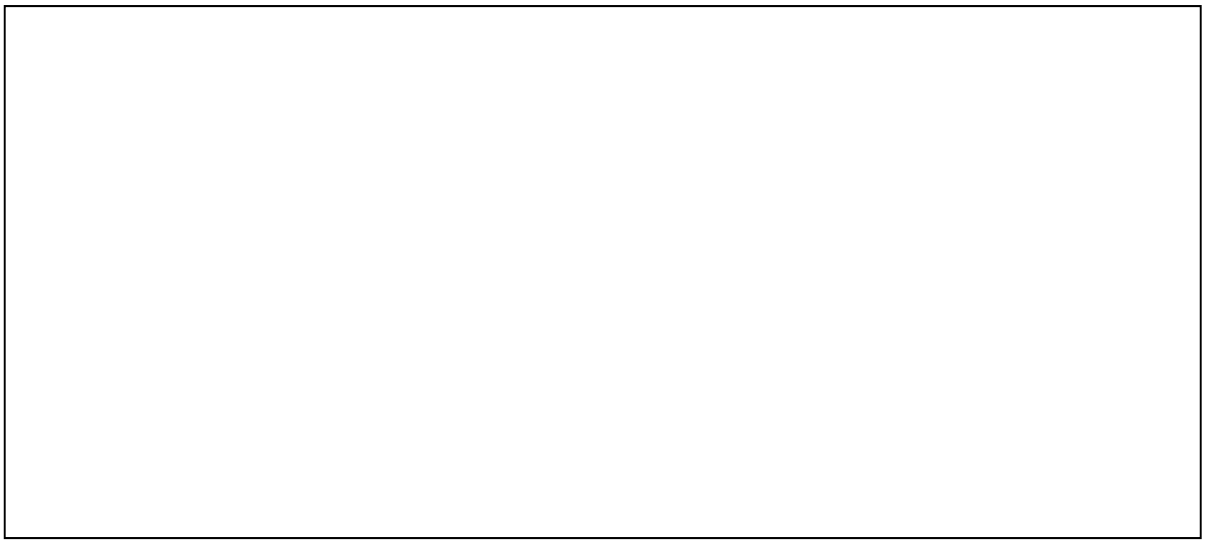
**(10) Question 9 (FSM).** Given the following Moore FSM implementation:

```
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.



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`

**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 or 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
```

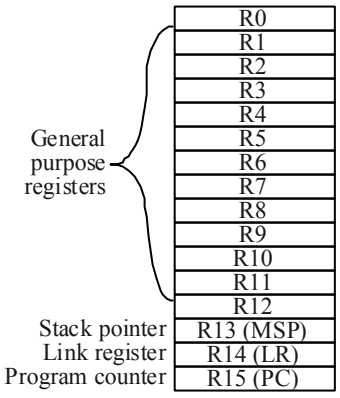
**Notes** `Ra Rd Rm Rn Rt represent 32-bit registers`

| | |
|---|---|
| `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> |

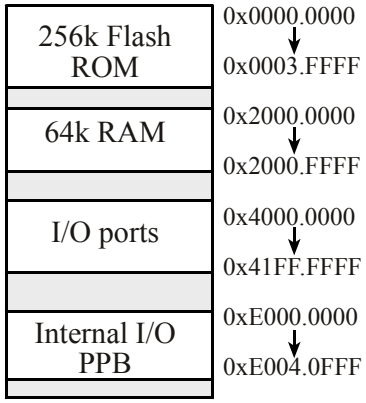
Examples of flexible operand **<op2>** creating the 32-bit number. E.g., **`Rd = Rn+op2`**

```
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




```
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
```

| Address | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Name |
|---|---|---|---|---|---|---|---|---|---|
| $400F.E108 | | | GPIOF | GPIOE | GPIOD | GPIOC | GPIOB | GPIOA | SYSCTL_RCGCGPIO_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 TM4C123/LM4F120 parallel ports. Each register is 32 bits wide. Bits 31 – 8 are zero.**

| Address | 31 | 30 | 29-7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Name |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0xE000E100 | | F | … | UART1 | UART0 | E | D | C | B | A | NVIC_EN0_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 | SYSTICK | 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 |

| | 31-12 | 11-0 | |
|---|---|---|---|
| $4003.80A8 | | 12-bit DATA | ADC_SSFIFO3 |

**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 (**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_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. 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

**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.**