EEL 4742C: Embedded Systems
Homework 5

QUESTION 1. **(10 points)**

The analog-to-digital converter (ADC) module uses a clock signal that varies in the range [20 -40] Hz due to the operating conditions and error. The sample-and-hold time of the signal we're converting is 3 seconds. The ADC allows selecting a number of cycles from the list below.

Number of cycles: 10, 40, 80, 110, 150, 200, 250, 300

Part a) Show the analysis for choosing the suitable number of cycles.

$$40 * 3 = 120 \rightarrow 150\ cycles$$

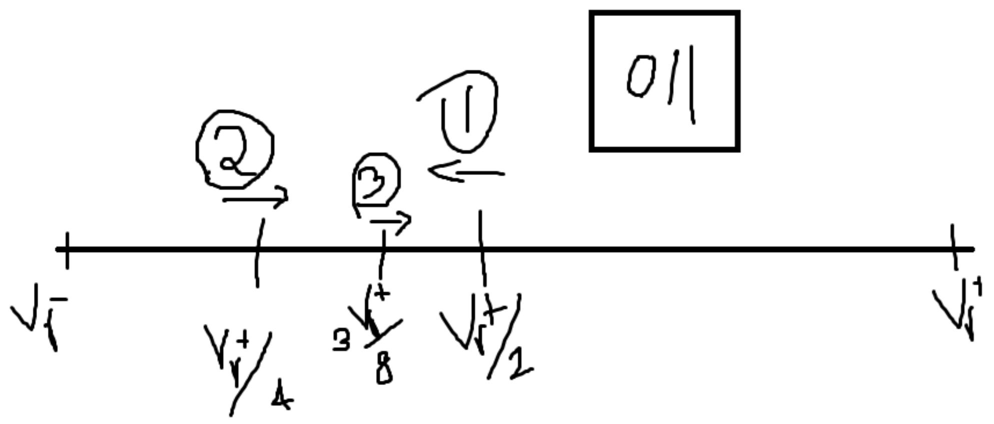
Part b) The conversion takes 10 cycles. What is the total operation time (that includes the sample-and hold time and the conversion)?

$$150 + 10 = 160\ cycles \rightarrow \frac{160}{40} = 4\ seconds$$

Note: the numeric values in this question are chosen to simplify the computations; an ADC usually uses a much higher clock frequency and the sample-and-hold time is usually in the micro-seconds range.

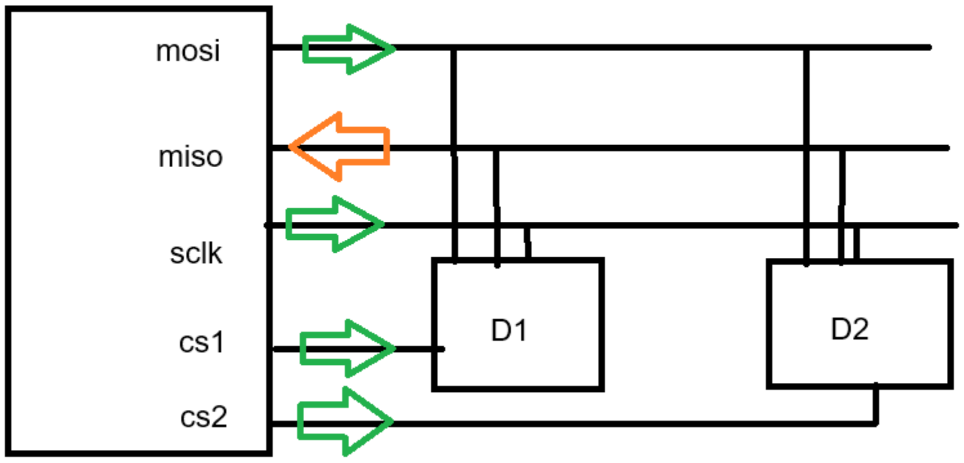
QUESTION 2. **(10 points)**

An SAR ADC has a 3-bit resolution (result). The reference voltages are Vr-=0 and Vr+=Vr. The signal we're converting is: Vin = 3.4/8 Vr+. Show all the voltage comparisons that are done by the ADC and show the conversion's binary result.

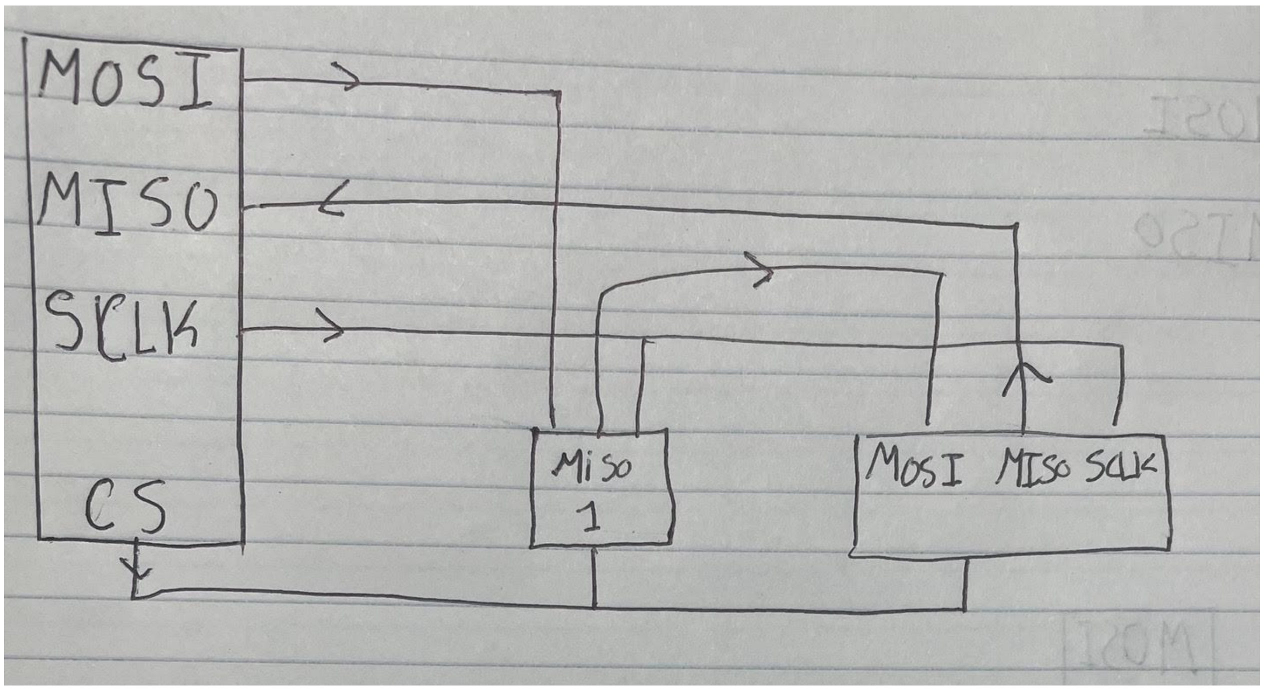


QUESTION 3. **(10 points)**

Part a) Draw the configuration where an SPI master is connected to two devices that are individually selected.



Part b) Draw the configuration where an SPI master is connected to two devices in a daisy chain.



Part c) What is the 3-pin SPI configuration? What's the difference between 3-pin SPI and 3-wire serial?
The 3-pin SPI configuration is used when the controller is only communicating with one device only, in that case, we can make the chip select to be always active and only use the 3-pin SPI which is SCLK, MOSI, and MISO pins. The 3-wire serial does not require a specific configuration and it may have different usages for the wires such as SDA, SCL, and ground.

QUESTION 4. **(10 points)**

Part a) Describe the graphics software stack in embedded systems.
At the lowest level, there is the Hardware abstraction layer (HAL). The HAL communicates with the LCD module using SPI and use's the hardware specific commands to draw pixels and configure this specific LCD module. The graphics library uses the HAL to draw pixels on the LCD module, however, the graphics library allows us to perform complex graphical operations such as drawing shapes, images, or text.

Part b) What services does the display driver provide? To what party does it provide these services? Comment on the compatibility of a driver with multiple displays.
The display driver is the same as the HAL, it configures the SPI communication and sends the unique commands required by LCD module.

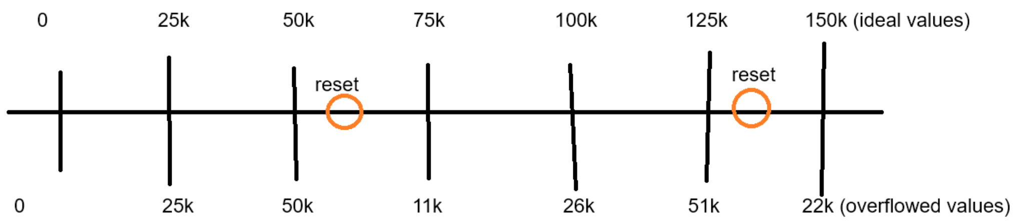
Part c) What services does the graphics library provide? To what party are these services provided? Comment on the compatibility of a graphics library with multiple displays.
The graphics library provides utility functions that allow drawing of text, images, and shapes (circle/rect). The graphics library also provides cross-compatibility with other drivers.

Part d) What is a graphics context?
The graphics context is an object that hold information about the current LCD module and graphics mode configuration and it also contains necessary state and data variables used internally by the graphics library.

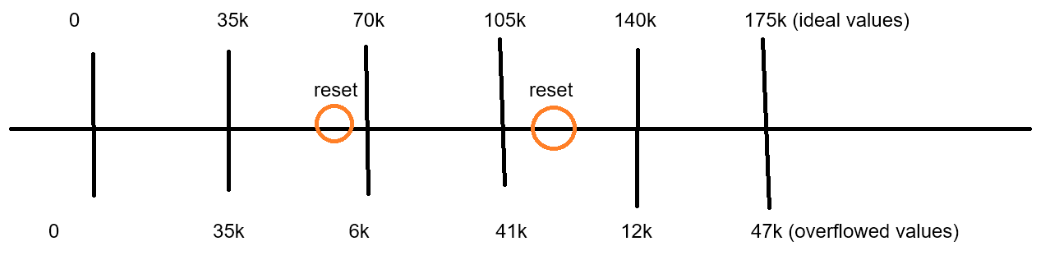
Part e) What is the clipping region of a graphics context?
The boundary region of the graphics context, everything outside this region cannot be drawn/dispalyed.

QUESTION 5. **(10 points)**

Part a) Channel 1 is scheduling periodic interrupts every 25K cycles (K=1024). Show how the interrupts are scheduled and demonstrate that the operation is still correct when a rollback to zero occurs on TACCR1. Show enough cases that span two rollbacks to zero.

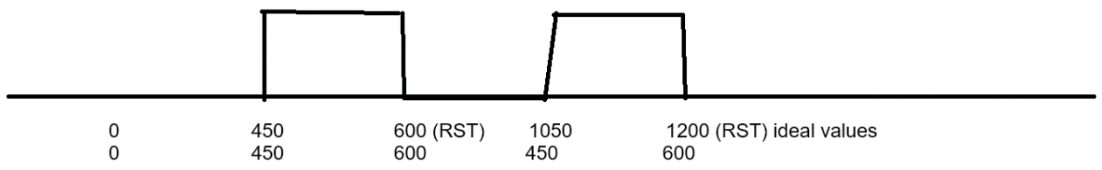


Part b) Channel 2 is scheduling periodic interrupts every 35K cycles (K=1024). Show how the interrupts are scheduled and demonstrate that the operation is still correct when a rollback to zero occurs on TACCR2. Show enough cases that span two rollbacks to zero.

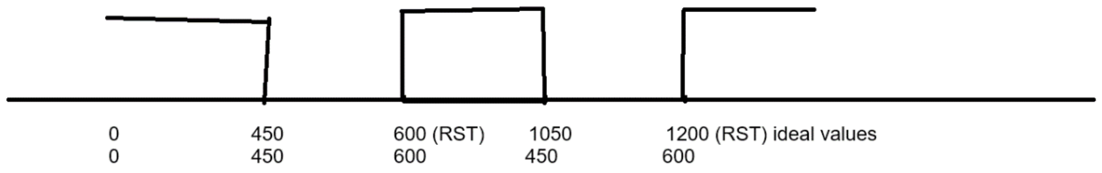


QUESTION 6. **(10 points)**

Part a) The timer is running in the up mode with a period of 600 cycles. Channel 1 engages the output mode (Set/Reset) and sets its register TACCR1 = 450. Explain what happens. Draw a timeline.



Part b) Repeat the previous part based on the (Reset/Set) mode.



QUESTION 7. **(10 points)**

Each time an event occurs (e.g. button push), the program needs to time a duration of 0.25 seconds (using ACLK=32 KHz) and raise an interrupt when this duration elapses. Note that this interval is timed only when the event occurs, not repeatedly back-to-back.

Part a) Can we set this up using the up mode? If yes, explain how.
 Yes, we need to reset the channel and set TAxCCR0 to $\frac{32}{4} = 8k$
Part b) Can we set this up using the continuous mode? If yes, explain how.
 Yes, we can add 8 kHz to TAxCCR0 above the value in the TAR register.

Part c) If both modes are possible, explain the preferable mode.

Up mode is preferred because we can have easier control when we use multiple channels.

Part d) If the timer is used intermittently, is it possible to turn the timer on/off throughout the program? How is this done?
Yes, the timer can be turned on/off. We can turn it off when the condition to do so is met and we can turn it on again in another ISR such as button interrupt.

Part e) Timer_A is running in the continuous mode and none of its interrupts is enabled. TAR has cycled the full range (0 to 64 K) a few times and the program didn't interact with the timer's bit fields or registers. At this point, what is the value of each of the flags: TAIFG, CCIFG flags of Channels 0, 1, 2?
TAIFG and CCIFG we all be 1. The hardware would normally clear channel 0 interrupt flag because it is not shared, however, since there are no interrupts enabled the ISR would not be called.

Part f) In our program, when an interrupt event occurs, it's disabled for a little while (e.g. 10 seconds) and then it's re-enabled. When is the best time to clear the interrupt's flag: when it disabled or when it's reenabled?
It is required to clear the flag before re-enabling the interrupt, otherwise, the interrupt may execute once the timer is enabled.

QUESTION 8. **(10 points)**

Part a) There are multiple levels of programming microcontrollers that are shown in the list below. Which level did we use in this course?

- Assembly level
- Register level
- Library level (Graphics Library)
- Operating system level (this was discussed but not used in Lab; RTOS)

Part b) The Arduino line of microcontrollers is very popular thanks to the programming library known as the Wiring API. This API provides simple functions that can be used by the programmer. These functions are implemented for all the Arduino microcontrollers. Therefore, the user can use the same (portable) code for any Arduino device chosen. The Wiring API has been implemented for MSP430 microcontrollers in the Energia IDE.

To give you an idea of how an API is implemented, let's write a function that's similar in style to the Wiring API. The function finds the first available channel of Timer_A. A channel is considered available if its CCIE bit is disabled. This is the header of the function. It returns the first available channel out of Channels 0, 1, 2, in this order. If all the channels are used, the function returns -1. Write the code.

```
int get_available_timer_channel();
```

```
int get_available_timer_channel() {
    if(!(CCIE & TA0CCTL0)) {
        return 0;
    }
    if(!(CCIE & TA0CCTL1)) {
        return 1;
    }
    if(!(CCIE & TA0CCTL2)) {
        return 2;
    }
    return -1;
}
```

QUESTION 9. **(10 points)**

ARM 32-bit microcontrollers have a 32-bit CPU and a 32-bit memory address. The 32-bit address can reference 2^32 = 4 GB of memory, which is far more memory than a microcontroller usually has. Therefore, there is an abundance of unused addresses, which has inspired the bit-banding technique.

Modifying a single bit in the memory takes three assembly instructions, e.g. (load-OR-store to set a bit). There is interest in making this type of operation faster since it's used frequently. The bit-banding technique takes unused addresses and maps them to bits of a variable. As shown below, the variable Data is at address 100. Accessing address 100 references the whole 8 bits in Data. The bit-banding technique takes the non-used addresses in the 2000 range and maps them to individual bits of Data. That is, address 2000 maps to bit #0 of Data, address 2001 maps to bit #1 of Data, etc. With this technique, the load-ORstore operation to modify a bit can be reduced to a store operation. Writing 1 to address 2000 results in writing 1 to bit #0 of Data, leaving the other bits unchanged.

```
         Data:  __  __  __  __  __  __  __  __
Address: 100    #7  #6  #5  #4  #3  #2  #1  #0

Address:       2007          ...          2000
```

Part a) Write a piece of code that reads bit #4 of Data (into the variable temp) using the bit-banding technique.
uint8_t* ptr = (uint8_t*)0x2004;
temp = *ptr;
Part b) Write a piece of code that sets bit #5 of Data using the bit-banding technique.
uint8_t* ptr = (uint8_t*)0x2005;
*ptr = 1;
Part c) Write a piece of code that counts how many bits in Data are equal to 1, using the bit-banding technique.
uint8_t* ptr = (uint8_t*)0x2000;
uint8_t counter = 0;
uint8_t i;
for(i = 0; i < 8; i++) {
if(ptr[i] != 0) {

```
counter = counter + 1;
}
}
```

Practice Questions

Do not submit; these questions were done in the class or lab.

PRACTICE 3.

Write a C code that flashes three LEDs using three channels of Timer_A based on interrupts. The timer should use ACLK which is based on a 32 KHz crystal.

- LED1 mapped to P1.0 is toggled every 0.25s using Channel 0
- LED2 mapped to P1.1 is toggled every 0.5s using Channel 1
- LED3 mapped to P1.2 is toggled every 0.75s using Channel 2

PRACTICE 1.

Write a code that turns on the LED (P1.0, active high) for 3 seconds when the button (P1.3 active low) is pushed. The interval is non-renewing, i.e., the button is ignored if it's pushed again during the three second interval. Configure the button via interrupt. Time the 3-second interval using Timer_A with interrupt (preferred mode: continuous mode with the timer running all the time). Use ACLK at 32 KHz.

PRACTICE 2.

Write a code that debounces the push button (P1.3, active low). When the first button edge is detected, wait for the maximum bounce duration (15 ms) and, if the button is still pushed, take the action, which is toggling the LED (P1.0, active high). Configure the button via interrupt. Time the duration using Timer_A with interrupt (preferred mode: continuous with the timer running all the time). Use ACLK at 32 KHz.