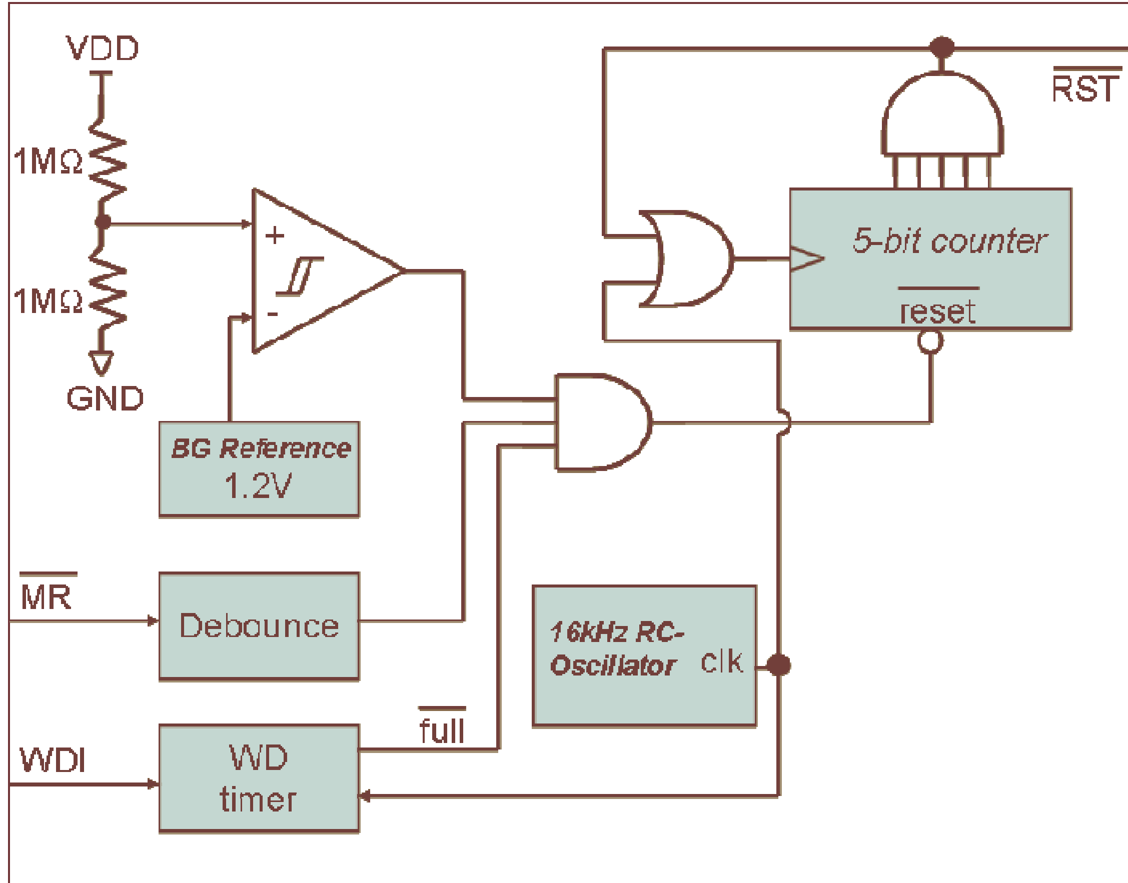
HOMEWORK ASSIGNMENT #4

Due Friday, March 27th, 2009

1. (14 points) What is it?



a) A block diagram of a chip is given above… What is it used for?
Microprocessor supervisor chip. Used to provide a robust POR derived from monitoring power level, and from a MR push-button, and from a WDT.

b) Under what 3 conditions does its active low output become active?
If power falls below 2.4V, If the Master Reset pushbutton is pushed, or if the WDT is allowed to expire.

c) What is the minimum amount of time that $\overline{\text{RST}}$ can be asserted?
The 5-bit counter has to count to a full count before reset is de-asserted. So that would be 32*1/16kHz = 2ms

d) At what approximate threshold level VDD has to cross? 2.4V

e) What is the purpose of the WDI input? The microprocessor needs to pulse it periodically to prevent the WDT from expiring and causing a reset.

f) What is the purpose of the MR input A momentary push-button switch would be connected here to ground (with a pullup to VDD). This allows for a manual push-button reset. The chip provides a Debounce circuit.

g) Would this be an appropriate chip to use with our processor?

The reset level of 2.4V is in the proper ballpark for the ADuc chip, however the minimum duration of the reset is uncomfortably short for this chip. We are going to be operating 2ms after the supply crosses 2.4V. Within this time the crystal oscillator and PLL are still coming up and stabilizing, plus VDD is still ramping up the minimum of 2.7V. This 2ms is just too short for this chip.

2. (20 points) To Sleep or Not to Sleep?

Assume that you have an ADuC7026 that is operating with a perfect 32.768 kHz crystal and the 1275x PLL. The processing requirements are such that the processor will be notified every 2ms by an external circuit (using IRQ0) that it needs to do its computations. When the processor completes the computations, it can go back to a low-power mode (if applicable) until the next time it is notified. It has been determined that the computations take 6,000 HCLK cycles. How could you use the various power control operating modes and the HCLK divider (CD) to control the power consumption while still performing the required computations? Determine the best use of the clock divider for each of the five (5) power control operating modes (including staying active continuously), and then select the best option to minimize overall power consumption. **HINT:** To get full credit you need to figure your strategy (and CD) for each mode, and estimate the power consumption for each mode.

With a 32.768kHz crystal and 1275x PLL, the undivided clock frequency is 32.678kHz * 1275 = 41.7792MHz.

ACTIVE: If the processor is kept continuously active, there is no wake-up time and it can use the entire 2ms for processing. In 2ms, there will be 41.7792MHz * 2ms or 83558.4 undivided clocks. Since we only need to accomplish 6000 clocks of processing, we select CD=3 (divide by 8) to have ~10,000 clock cycles per 2ms interval. This results in an average current consumption of 10mA.

PAUSE: If pause mode is used, the wake-up time is significantly less that 1% of the total time. If we ran with CD = 0, the wake-up would take 41ns (a single clock period), we would complete our 6000 HCLK processing in 6000/41.7792 = ~0.144ms, then we could return to pause mode for the remaining ~1.86ms. This would result in an average current of ((0.14ms*33.1mA)+(1.86ms*22.7mA))/2ms = ~23.4mA. If we reduced the core clock frequency by setting CD=3, the wake-up would take 328ns (a single core clock period), we would complete our 6000 HCLK processing in 6000/(41.7792/8) = ~1.15ms, then we could return to pause mode for the remaining ~0.85ms. This would result in an average current of ((1.15ms*10.0mA)+(0.85ms*6.1mA))/2ms = ~8.3mA.

NAP: The analysis for NAP mode is identical to pause, except that since the peripherals are not clocked the current is significantly lower. Repeating the calculations for CD=0, the average current would be 5.85mA. If CD=3, the average current would be 7.36mA.

SLEEP: In sleep mode, the wake up time is a significant fraction of 2ms. With a 1.58ms wake-up time, there are 0.42ms available for processing. The maximum core clocks during this period is 41.7792MHz * 0.42ms = 17,547 clocks. This means that we must operate with CD=0 or CD=1.

With CD=0, 6000 clocks equate to 6000/41.7792MHz= ~0.144ms. With CD=1, 6000 clocks equates to 0.287ms. In both cases, as soon as the processing is complete, we can reenter sleep mode. With CD=0, the average current will be (0.144ms*33.1mA)+(1.856ms*0.4mA))/2ms = ~2.75mA. With CD=1, the average current will be (0.287ms*21.2mA)+(1.713ms*0.4mA))/2ms = ~3.38mA.

A question with this analysis arises (what is the current during that 1.58ms wakup time?). During this period the analog portion of the PLL is powered, and the PLL is acquiring lock. Is the clock released to the rest of the chip during this period? Or do they have a lock detector on the PLL and not release the clock until the PLL is locked. Taking the current to be 0.4mA during this 1.58ms period (as done above) is not really accurate. Do we know the exact number? No. If you took the current during the wakeup period to be the full active current or some number inbetween then that is fine. Really looking for thought process.

STOP: The stop mode analysis is similar to the sleep mode. The only options are to operate at CD=0 or CD=1 in the 0.3ms available. The results will be similar.

The best option to reduce current (and hence power) will depend on your assumptions about the power consumption during the 1.58ms wakeup period. If you take the current during this period to be 0.4mA then Sleep/Stop will be the best modes. Otherwise you may discover NAP to be the best mode. Again looking for thought process and data analysis.

3. (10 points) Timer1 as a counter, and Port1 as an output

Create a program that will setup Timer1 as an up counter, counting Port0.6 transitions. The pre-scale should be 1:1. Port1 should be initialized as an output, and will display the value of the lower 8-bits of the Timer1 count.

The main loop should simply read Timer1 value and write it to Port1 output.

Under ***Peripherals →General Purpose Input/Output*** you can get GUI windows that you can use to control Port0.6, and see the contents of Port1. When you run the program you should be able to click on the Port0.6 input, and see the count in the Port1 output increment every time Port0.6 is transitioned high.

Important: Submit your program source code file ***teamX_cnt.s*** in the homework #4 dropbox in Learn@UW. Also, submit a **paper copy** of ***teamX_cnt.s*** with the rest of the assignment.

```
    INCLUDE aduc7026.inc

    AREA    FLASH, CODE, READONLY
__main
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;; Setup Timer1 for operation as counter from P0.6 ;;
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    LDR     R7, = (TIMER_MMR_BASE)
    LDR     R1, =(0x0780)           ;Setup for P0.6 as count with 1:1 pre-scale, count up
    STR     R1, [R7, #T1CON]
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;; Setup Port0 direction and data ;;
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    LDR     R8, =(GPIO_MMR_BASE)    ;Base pointer for GPIO registers
    MOV     R0, #0
    STR     R0, [R8,#GP0CON]        ;Setup P0 for all GPIC
    MOV     R0, #0                  ;Setup P0 for all input
    STR     R0, [R8,#GP0DAT]        ;Actually write to GP0DAT
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;; Setup Port1 direction and data ;;
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    MOV     R0, #0
    STR     R0, [R8,#GP1CON]        ;Setup P1 for all GPIC
    LDR     R0, =(0xFF000000)       ;Setup P1 for all output with a 0x00 as initial count
    STR     R0, [R8,#GP1DAT]        ;Actually write to GP1DAT

main_poll
    LDR     R0, [R7,#T1VAL]         ; read counter value
    LSL     R0, #16                 ; shift value over to output position
    ORR     R0, R0, #0xFF000000     ; ensure direction bits remain outputs
    STR     R0, [R8,#GP1DAT]        ; write counter value to port
    B       main_poll               ; loop forever
```

4. (20 points) Storm Stopper (game at Rocky Rocco's Pizza)

You will use the Timer0 and the GPIO to implement a simple game.

First you need to setup Timer0 to give you approximately a 0.1 second time base. (HINT: at a CD=3 (default config) with a 32kHz clock into the 1275 multiplying PLL (also default config) the MSB of Timer0 will toggle about once every 0.1 seconds if the Timer0 pre-scale is 16). **You are not to use interrupts for this program. All actions will occur as the result of polling.**

The 8-bits of Port0 will be configured as outputs. Think of them as connected to LED's. Start with a 1 seeded in the LSB of Port0. Every 0.1 seconds it should be shifted left one position. When it is at Port0.7 (MSB of Port0) then it should roll over to Port0.0 (LSB of Port0) next.

Port1 bits should be configured as output to display the score. Obviously score starts at zero.

Port2.0 (LSB of Port2) will be the stop_n/start input bit. By default this port pin will start with a logic 1. When Port2.0 is a 1 the loop is running, and the lit LED of Port0 is circulating around with a 0.1 second update rate.

When you click on Port2.0 peripheral input control to set it to zero you are stopping the lit LED from circulating.

The objective of the game is to stop it when it is on Port0.6. When your polling loop detects Port2.0 falls it should check to see where the LED was. If it is on Port0.5 or Port0.7 your score is increased by 1 point. If it is on Port0.6 your score is increased by 10 points. For all other locations your score remains unchanged for that try. At this point the score should be updated on Port1. The program should now be in a spin loop waiting for Port2.0 to be raised. When it is raised the LED should resume movement.

You should have another counter, counting the number of tries. When there have been 5 tries to stop the LED the program should vector to a section to handle game_over. In the game over the program should spin until Port2.1 is lowered and raised. Once it is the code should reinitialize itself and start over (i.e. score cleared, try counter set to zero, and if Port2.0 is high the LED moving again).

Base your code on the Kiel uVision3 sample project on the course web page, renaming *main.s* to ***teamX_prob4.s***. All references to the GPIO MMRs should use the definitions in ***aduc7026.inc***. Note that the definitions are set up to include a base address for a group of registers, and smaller offsets within the groups. A typical usage of these definitions would be as shown below.

```
LDR    R7, =(GPIO_MMR_BASE)   ;load base address
MOV    R0, #0
STR    R0, [R7, # GP0CON]     ;set all Port0 pins to GPIO function
```

To test your program, you should open windows to view the pin states of P0, P1 and P3 – you can do so by selecting ***Peripherals →General Purpose Input/Output*** from the menu when running the debugger. By checking/unchecking the bits in the I/O Pins block for Ports 2, you can create the game inputs, even while your program is running.

Solution posted a separate file. Note this solution is something I banged out to demonstrate the feasibility of this problem it does not follow good modular programming practices. Do as I say, not as I do.

Important: Submit ONLY your program source code file ***teamX_prob4.s*** in the homework #4 dropbox in Learn@UW. Also, submit a **paper copy** of ***teamX_prob4.s*** with the rest of the assignment.

5. (16 points) Using the ADuC7026 Watchdog Timer

Using the sample project provided as a starting point, write code to configure the watchdog timer to give a ~25ms timeout while operating with the **Secure Clear Bit set**. Base your code on the Kiel uVision3 sample project on the course web page, renaming *main.s* to ***teamX_prob6.s***.

When your code starts up, check if a watchdog reset has occurred (RSTSTA bit 1). If it has, branch immediately to the spin loop at the end of the program. Otherwise, set GPIO Port2 as all inputs, then configure and enable the watchdog timer. Use an initial seed value of 0x47 for the secure clear mode. After the watchdog timer is configured and operating, your code should enter an indefinite loop where it does the following;

- If pin P2.0 is read as a 1, continue to loop checking P2.0. Note that this will eventually cause a watchdog reset since we will not reset the watchdog timer.

- If pin P2.0 is read as a 0, check if the watchdog timer has counted down below one half (1/2) of its original load value. If it has, then reset the watchdog timer using the appropriate secure clear value (you will need to calculate it).

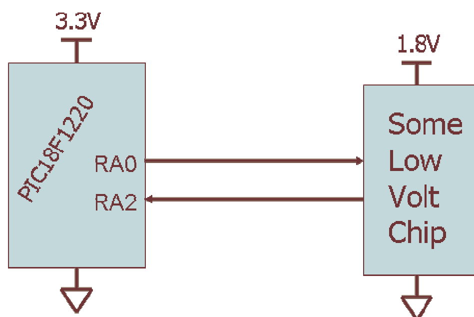
As a suggestion for organizing your work, you may want to get your code working without putting the watchdog timer into secure clear mode first (clearing the watchdog timer is easy when not in secure clear mode). Only after that is done, add the secure clear mode functionality. All references to the GPIO MMRs should use the definitions in ***aduc7026.inc***.

Important: Submit ONLY your program source code file ***teamX_prob6.s*** using the homework #4 dropbox in Learn@UW. Also, submit a **paper copy** of ***teamX_prob6.s*** with the rest of the assignment.

Krishna's Solution posted on webpage

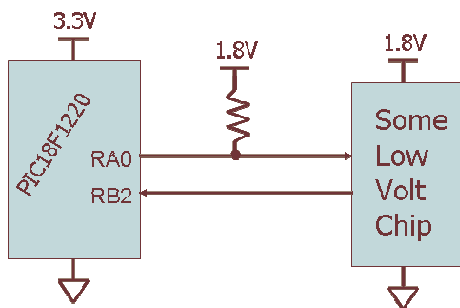
6. (10 points) Interfacing Different Voltages

Here we consider a very common situation where multiple power supply voltages are in use in a digital system.



A PIC chip is being interfaced with a 1.8V chip. There is one input and one output. The output from the 1.8V chip contains CMOS full push/pull driver (i.e. Voh = VDD, and Vol = VSS).

It was originally intended to hook it up as shown here. But luckily it was caught in a design review that this would have problems.



The proposed solution was to make the changes shown. Now the input from the low voltage chip comes into RB2 instead of RA2. The output to the low voltage chip is still from RA0, but it has this pullup resistor, and the guy in the design review said to: "run it open drain".

a) What were the problems with the original design? (what would have gone wrong)

For the signal that was output from the PIC to the Low voltage chip. The driven level would have been 3.3V. This would have forward biased the ESD protection structures in the low voltage chip. It could also have possibly damaged the gate oxide of that input device.

For the output of the low voltage chip going to the PIC. The 1.8V level is not high enough to exceed the V_{IH} of port input RA2.

b) Why does changing to RB2 help?

RB2 input uses a TTL compatible input buffer. Looking into the electrical characteristics in the PIC datasheet one can see the V_{IH} of the TTL compatible inputs is: 0.25*VDD+0.8V. That would be 1.625V. Getting marginal, but will still work.

c) What did the the guy in the design review mean by "Run it open drain". How does that solve this problem?

Open drain means only the pulldown (NMOS) in the GPIO driver of the PIC is used. This works because we don't want the VOH level to be the 3.3V of the PIC, but rather the 1.8V level. Thus when the PIC drives low it drives zero, but when it "drives" high it does not really drive at all, but rather just releases the line, and lets the pullup take it to 1.8V.

7. (10 points) Quiz Question

Design one original quiz question operating at Bloom's Taxonomy level 3 for any material covered in Module 4. This must test one of the educational objectives (see http://eceserv0.ece.wisc.edu/~morrow/ECE353/objectives.pdf) in a specific problem. Explicitly state which particular educational objective you are attempting to test. Provide a complete, detailed solution to your question.

I am not looking for really complex problems here. I am looking for salient questions that test your knowledge of a topic.