OBJECTIVES

- Learn how to configure the ATXMEGA128A1U system clock frequency in AVR® assembly

INTRODUCTION

Timing is critical in most embedded systems, no matter how simple or complex they may be. The clock is the basis of all synchronous operations in the ATXMEGA128A1U.

In most computer systems, it is preferred that a microcontroller operate at a specific clock frequency. For example, if power consumption is a main concern, a low clock frequency is usually chosen; if execution time is a major factor, a higher frequency might instead be chosen. In this homework, you will learn how to program your microcontroller to execute at any of its allowable clock frequencies.

REQUIRED MATERIALS

- Atmel ATXMEGA128A1U AU Manual (doc8331)
- Atmel ATXMEGA128A1U Manual (doc8385)
- *OOTB µPAD v2.0* with USB A/B cable
- **D**igilent **A**nalog **D**iscovery (**DAD**)
- *WaveForms* software
- **`clock.s`** (C-callable assembly file)
  - Rename the file to **`clock.s`** (from **`clock.s.txt`**)

SUPPLEMENTAL MATERIALS

- Atmel ATXMEGA128A1U Manual (doc8385)
- AVR Instruction Set (doc0856)
- DMA videos on our website

1. SYSTEM CLOCK CONFIGURATION

(**NOTE**: The solution to the below procedure is available on the lab page of our course website, clock.s.txt. After downloading the file, rename the file to clock.s.)

The below procedure will step your through the design of a C-callable assembly subroutine (**`clock_init`**) that adjusts the frequency of the ATXMEGA128A1U system clock. Use the template assembly file, **`clock.s`**, as an outline for your subroutine. (After downloading this template file, clock.s.txt, rename the file to clock.s.) Make sure you add this file to your Atmel Studio C project by right clicking on your project within the *Solution Explorer*, then choosing *Add* | *Existing Item*. Then, navigate to the directory at which you saved the **`clock.s`** file and add it to the project.

NOTE: The main difference between a C-callable assembly subroutine and a normal subroutine is the use of new assembler directives that are used to help link all the files together. The ".global" directive is used to tell the compiler that this subroutine should be able to be accessed by other files. If you want to know more about how files are linked, see the following article: https://www.lurklurk.org/linkers/linkers.html

In your main source file, where you will call the clock_init subroutine, you will need to add an "extern" prototype to declare the subroutine. Somewhere above your main function, add the following line:

```
extern void clock_init(void);
```

1.1. Read § 7 (System Clock and Clock Options) of the 8331 manual.

The OSC_CTRL register (§ 7.10.1) enables any of the available clock oscillators. You must first enable an oscillator before you can select it as the new clock source. After the desired oscillator is enabled, you must give it time to stabilize. The OSC_STATUS register (§ 7.10.2) contains useful flags that are **set** only when the oscillator is **stable** and ready for use.

After the oscillator has stabilized, you must select it as the new clock source. See the CLK_CTRL register (§ 7.9.1).

After the new system clock source has been selected, you have the option to utilize a clock prescaler. See the CLK_PSCTRL (§ 7.9.2) register, as well as § 7.5 (*System Clock Selection and Prescalers*).

NOTE: Some of the aforementioned registers are protected. See § 3.12 (*Configuration Change Protection*) for more details.

To analyze the ATXMEGA128A1U's clock, you need to make the clock signal output to an I/O pin. There are several ways to do this, but the most straightforward technique is to use the CLKEVOUT register (§ 13.14.4). This is the method that you must use for this homework.

1.2. Remove any backpacks from the µPAD. (The memory base can remain attached.)

1.3. Create a C program, **`clock_test.c`**. In the same project, add the **`clock.s`** file as instructed above. Within the assembly file, fill in the provided subroutine (e.g., *clock_init*) so that it configures the system clock for 32 MHz, initializing everything described above. The subroutine should also allow the programmer to easily (manually) scale the system clock with any of the available prescalers, if desired.

1.4. Within the main routine, use the subroutine created above to configure the clock speed for **4 MHz.** Then, configure the CLKEVOUT register to output the system clock (**4 MHz)** to PORTC pin 7; remember to configure this pin as an output. You will submit **`clock_test.c`** and **`clock.s`** with the configuration for a clock speed of **4 MHz**.

The clock should now be outputting from PORTC pin 7. You will now measure the clock frequency with the *Scope* feature in

WaveForms. With no backpacks connected to the µPAD, you can access PORTC pin 7 via a female header on the top of the board.

1.5. Use the *Scope* feature in *WaveForms* to measure the clock signal and display the waveform frequency. Within the *Scope* window, display the frequency by selecting *View* | *Measurements*, and then within the *Measurements* window, navigate to *Add* | *Defined Measurement* | *Horizontal* | *Frequency*. To yield an accurate frequency measurement, set your time base to an appropriate value, like 20ns/div as shown in *Figure 1* below.

NOTE: In general, an appropriate time base for a periodic function is one that displays two periods of the given waveform.

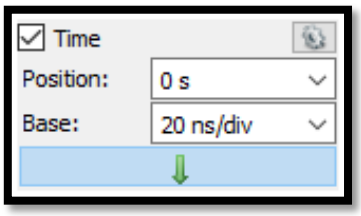

Figure 1: Appropriate time base for 4 MHz periodic function

Repeat this process for a system clock frequency of 32 MHz.

NOTE: The solution to this exercise is available on the lab page of our course website, clock.s.txt. After downloading the file, rename the file to clock.s.

NOTE: Due to the maximum sampling frequency of 100 MHz for the **a**nalog-to-**d**igital **c**onverter (**ADC**) within the DAD, there will only be a few samples taken per period of the required clock waveforms.