OBJECTIVES

- Learn how to configure the *ATxmega128A1U* system clock via AVR® assembly.
- Understand how to call an assembly routine via the "C" programming language.

INTRODUCTION

For most synchronous digital systems, it is preferred that a clock signal have a specific 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 configure the system clock signal of the *ATxmega128A1U*.

Important Note: It appears that Group Configurations do **NOT** work in .s files (but bit positions and bit masks do work)! Weird!

REQUIRED MATERIALS

- Atmel *ATxmega128A1U* AU Manual (doc8331)
- Atmel *ATxmega128A1U* Manual (doc8385)
- *OOTB µPAD* kit, with accompanying schematic(s)
- Digital Analog Discovery (DAD) kit, with *WaveForms*
- `clock.s.txt` ("C"-callable assembly file)

SUPPLEMENTAL MATERIALS

- AVR Instruction Set (doc0856)

HOMEWORK PROCEDURE

1. SYSTEM CLOCK CONFIGURATION

For this homework, you must understand how to design and utilize a "C"-callable assembly subroutine that adjusts the frequency of the *ATxmega128A1U* system clock.

NOTE: The main difference between a "C"-callable assembly subroutine and a normal assembly subroutine is the use of the `.global` assembler directive, which is used to help the "linker" software tool within *Microchip (Atmel) Studio* properly link all the relevant files together. If you wish to know more about how files are linked together with a "linker" tool, see the following article: https://www.lurklurk.org/linkers/linkers.html.

1.1. Download the provided `clock.s.txt` file. Once downloaded, rename this file to `clock.s`.

1.2. Create a "C" executable project within *Atmel Studio*. Add the relevant `clock.s` file to your project by right clicking on the project name within the *Solution Explorer* window and then selecting *Add* | *Existing Item*.

Now, you will walk through the relevant documentation to understand the implementation of the `clock_init` routine, given within the `clock.s` file.

1.3. Carefully read § 7 (*System Clock and Clock Options*) of the 8331 manual.

The *OSC_CTRL* register (§ 7.10.1) is used to enable available clock oscillators. An oscillator must first be enabled before it can be selected as a source for the system clock signal. After an oscillator is enabled, it must first be allowed to stabilize before it can be selected as a system clock source. The *OSC_STATUS* register (§ 7.10.2) contains useful flags that are set only when an oscillator is stable and ready for use.

After an oscillator has stabilized, this oscillator can be selected as a source for the system clock. The *CLK_CTRL* register (§ 7.9.1) is used to select such a source.

After the system clock source has been selected, a clock prescaler should be configured. The *CLK_PSCTRL* register (§ 7.9.2) is used to select such a prescaler.

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

Sometimes, as in this homework, it will be appropriate to measure the system clock signal, e.g., to make sure that it is configured as intended. To be able to measure the clock signal of our system, this signal needs to be output to a particular I/O port 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 will be utilized for this homework.

1.4. Study the provided `clock.s` file. Ensure that you understand all portions of the given code.

For the remainder of this homework, you will create a "C" program to [1] configure the system clock of the microcontroller and [2] output the clock signal to an appropriate I/O port pin. Then, after executing this program on your hardware, you will utilize the *Scope* feature of *Waveforms* to measure the relevant clock signal.

1.5. In the relevant project of *Atmel Studio*, create a "C" file, `clock_test.c`.

1.5.1. To allow the "C" compiler to be aware of the `clock_init` function located within the `clock.s` assembly file, the `clock_init` routine needs to be

"declared" within the scope of the `clock_test.c` file. To do so, write the following statement somewhere near the top of the `clock_test.c` file, outside all routines (if any exist at the moment):

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

1.5.2. Within the `clock_test.c` file, write a main routine that [1] calls the `clock_init` routine as if it were a "C" function (i.e., `clock_init();`), [2] configures the *CLKEVOUT* register to output the system clock signal to pin 7 of PORTC, [3] sets pin 7 of PORTC to be an output, and then [4] enters an empty infinite loop.

1.6. Slightly alter the `clock_init` routine within the `clock.s` file such that the system clock frequency would be 4 MHz after the routine is completely executed. (To do so, only one or two lines of code should be updated.) If necessary, refer to information regarding the *CLK_PSCTRL* register (§ 7.9.2).

1.7. Ensure that no backpack is connected to the *µPAD*.

1.8. Build and execute the relevant "C" program on your hardware. If everything specified above was done correctly, the clock signal should now be output to pin 7 of PORTC, which can be readily probed via the *J2* header of the *µPAD*.

1.9. Use the *Scope* feature of *WaveForms* to measure the clock signal and display a precise measurement of the waveform frequency. (To yield an accurate frequency measurement, set your time base to an appropriate value, like "20 ns/div", as shown in *Figure 1*.) Submit a screenshot of the *Scope* window, including both the relevant waveform and frequency measurement.

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

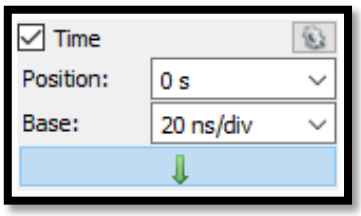

Figure 1. Appropriate time base for 4 MHz square waveform.

1.10. Repeat steps 1.6 through 1.9, but have the system clock frequency configured to 32 MHz.

NOTE: Due to the maximum sampling frequency of 100 MHz for the analog-to-digital converter (ADC) within the DAD, there will only be a few samples taken per period of the required clock waveforms. Thus, this waveform may not look as "square" as it would if a higher frequency oscilloscope were used.

HOMEWORK EXERCISES

i. What is the purpose of the Configuration Change Protection (CCP) register?

ii. What would need to be done for the *ATxmega128A1U* to have a system clock frequency of 8 MHz?

iii. An internal 32 MHz clock is available in *ATxmega128A1U* it can also run much faster. Describe how the *ATxmega128A1U* may be configured to run clock frequencies of 64 MHz or even 128 MHz, without using any external clock sources? **See section 37.1.14 of doc8385.**

HOMEWORK PROCEDURE SUMMARY

1) Answer homework exercises, when appropriate.
2) Read the relevant sections of the 8331 manual to learn how to configure the system clock.
3) Fully understand the provided `clock.s` file.
4) Create a "C" program to utilize the provided `clock.s` file and output the system clock signal to a relevant I/O port pin.
5) Use the DAD and *Waveforms* software to measure the microcontroller system clock signal twice: once when the system clock signal has a frequency of 4 MHz, and again when the system clock signal has a frequency of 32 MHz. Take screenshots for each case, as described above.