Milestone 1 Report - Sudoku on AVR

ACE411-Embedded Microprocessor Systems
Winter semester of academic year 2021-2022

Kallinteris Andreas:	2017030066	Team number on eclass: 5
Lioudakis Emmanouil:	2018030020	

Introduction

Description of the program

Block diagram of the program (pseudocode)

The main function constists of a main loop that call the sudoku solving function, if needed:

```
function MAIN
init()
enable global interrupts
    while true do
        if solve_barrier is true then
            if solved then
                solve_barrier ← false
            end if
        end if
    end while
end function
```

The interrupt handler to receive data from the serial port:

```
function USART-ISR
    switch received_character do
        case A or T or B or S or B or O or K
          set the appropriate flag
        case C
          set the appropriate flag
          clear(); /* To clear the grid */
        case N or D
          set the appropriate flag
          prepare the arguments counter
        case <CR>
          do nothing
        case <LF>
            if command is AT or C or N then send OK
```

```
        else if command is P then set the solving barrier flag send OK
        else if command is S then set the transmit barrier flag send the first cell
        else if command is T then
            if transmit_barrier is true then send the next cell
            end if
        else if command is B then unset the solving barrier flag send OK
        else if command is D then send the specific cell
        end if
    case  number between 1 and 9
        if command is N then store X, Y or VALUE (based on the arguments counter)
        else if command is D then store X or Y (based on the arguments counter)
        end if
end function
```

The interrupt handler to update the LEDs:

```
function TIMER1-COMPARE_MATCH-ISR
        update PORTA
end function
```

The init function:

```
function INIT
        set all the counters to their initial values
        unset all the flags
        initialize PORTA
        initialize TIMER1
        initialize the sudoku board
        initialize the serial port
end function
```

The sudoku solving algorithm

*TO BE COMPLETED BY ANDREAS

Controlling the LED progress bar

To update the LEDs that show the progress of solving, there were two options. The first one was updating the PORTA register every time we change the value of a cell and the second one was refreshing the LEDs with a 30Hz frequency. The former option would consume many clock cycles (because on every PORTA update, its new value is read from flash, which needs 3 clock cycles many times each second), when the latter would consume those 3 cycles only 30 times per second, which is pleasant for the human eye. So, the latter was used by enabling Timer/Counter1 and using its Output Compare Match Interrupt. The interrupt is fired with a 30Hz frequency and it simply updates the value of PORTA.

USART interfacing

As the assignment states, the serial port is configured at 9600 baud, 8 data bits, 1 stop bit and no parity.

Receiving from the serial port

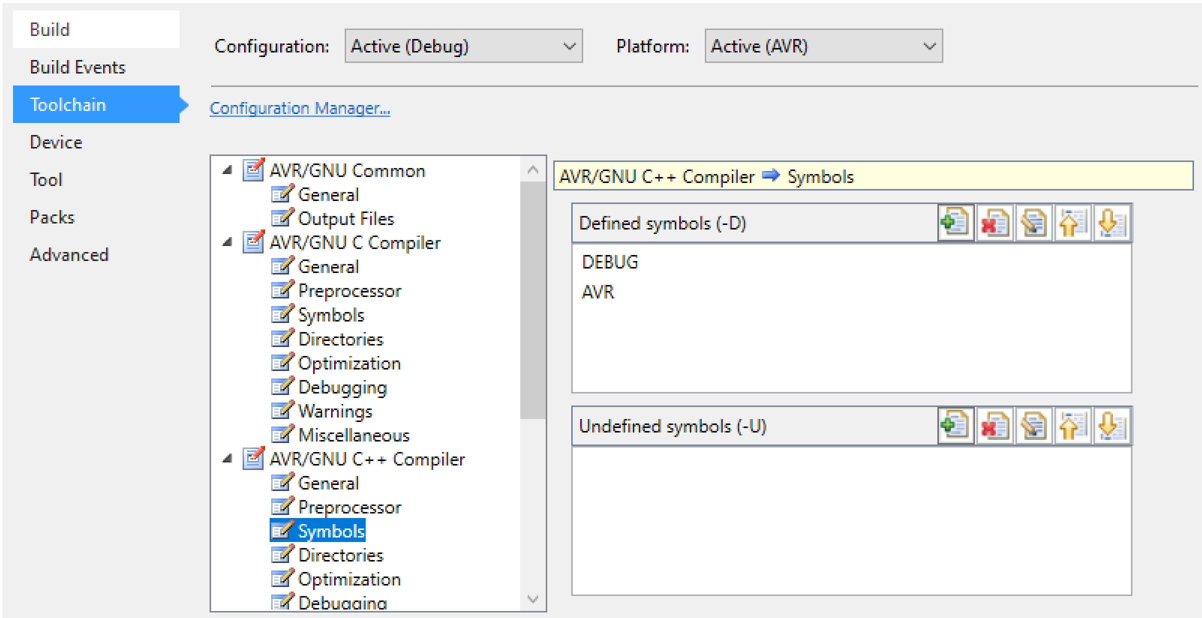
When a character is sent to the serial port, the USART_RXC interrupt is triggered. In the interrupt handler, based on the receiver character, if it is a command character, a flag is set, else if it is a number it is stored in the memory. After receiving the line-feed character, the command is executed.
It is important to note that in the case of "play" command, the interrupt service routine does not call the solving function, but simply sets the solving flag and returns. This is done to return back as fast as possible, and then start solving with the interrupts enabled (when the ISR is called, the interrupts are disabled until it returns). In that way, while solving a break or a debug command can be received and executed (by interrupting the solving function). If the ISR called the solve function, until completing solving, the interrupts would be disabled, and any new characters from the serial port would not be read.

Transmitting to the serial port

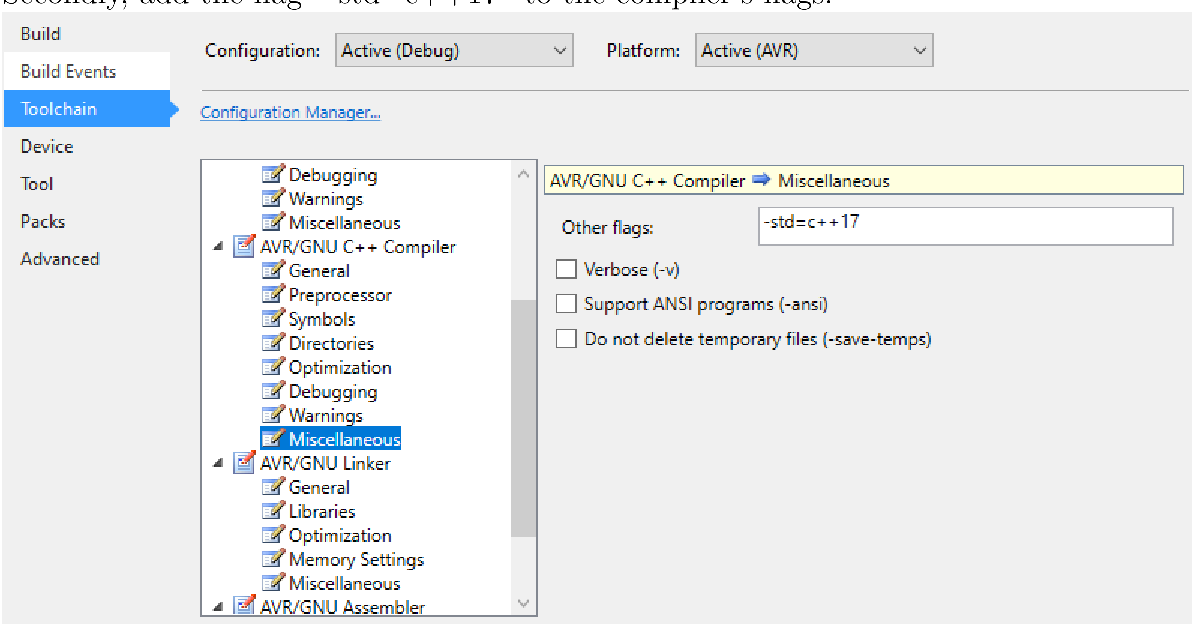
To transmit a character, a polling procedure is used. A loop is done to wait until the UDR register to get empty and then we write the characted to be transmitted. Practically, in this project polling is not actually done. The baud rate is extremely low, so the UDR register will be already empty, every time we want to transmit a character (every 10000 clock cycles approximately).

Microchip configuration - Compilation process

Since the code is written using the C++ language (using newer standards than the supported from Microchip Studio), some modifications should be done to the default settings of a new C++ project.
Firstly, add the symbol “AVR” to the compiler’s symbols:



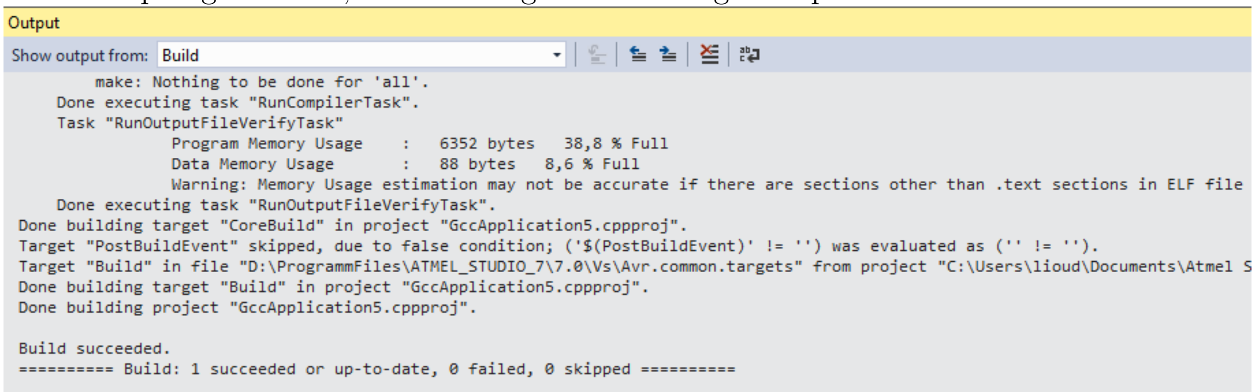
Secondly, add the flag "-std=c++17" to the compiler's flags:



The optimization level should be set to "-O2 (Optimize more)".
Finally, the SIMULATION_MODE should be defined (by uncommenting the line 25 of main.cpp). When it is defined, the program will read from TCNT2 instead of UDR and will redi-rect its output from UDR to TCNT0. By doing that, the program can be simulated with the provided stimuli files.

Resource usage

After compiling the code, the following resource usage is reported:




```
        make: Nothing to be done for 'all'.
    Done executing task "RunCompilerTask".
    Task "RunOutputFileVerifyTask"
                Program Memory Usage    :   6352 bytes   38,8 % Full
                Data Memory Usage       :   88 bytes   8,6 % Full
                Warning: Memory Usage estimation may not be accurate if there are sections other than .text sections in ELF file
    Done executing task "RunOutputFileVerifyTask".
Done building target "CoreBuild" in project "GccApplication5.cppproj".
Target "PostBuildEvent" skipped, due to false condition; ('$(PostBuildEvent)' != '') was evaluated as ('' != '').
Target "Build" in file "D:\ProgrammFiles\ATMEL_STUDIO_7\7.0\Vs\Avr.common.targets" from project "C:\Users\lioud\Documents\Atmel S
Done building target "Build" in project "GccApplication5.cppproj".
Done building project "GccApplication5.cppproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

Program memory (flash)

Except from the code, two arrays (look up tables) are stored in flash:

- The led_bar_LUT (defined in sudoku.h), which contains the decoding data for the LED progress bar (holds 82 bytes, in addresses 0x018c – 0x01dd).
- The div_9_LUT (defined in LUT.hpp), which contains pre-calculated value to make the solving proccess faster (holds 162 bytes, in adresses 0x00ea – 0x018b).

The rest bytes (6352-82-162 = 6108) are reserved by the code instructions.

Static RAM (SRAM)

The global variables used to control the serial port are the struct of flags for the received character(11 bits or 2 bytes – 0x0062-0x0063), the union for the X counters (1 byte - 0x0061), the union for the Y counters (1 byte - 0x0060) and the counter for the arguments (1 byte - 0x00b7). The rest of the 88 bytes are used for the base_board object (an instance of the sudoku class). Inside this object is stored the 9x9 sudoku grid as a two dimensioned array of unsigned integers.
During the solving process, the solver uses up to 790 bytes, leaving 146 bytes for context switching during interrupts.

Simulation in Microchip Studio, using stimuli files

Three stimuli files are submitted with the code (in the folder STIMULI_FILES):

- simple_solve.stim, which feeds the program with one sudoku board (the one shown in the assignment), waits until the sudoku is solved and sends the results back.
- break_debug_test.stim, which feeds the program with the same sudoku board as above, but while solving, a “break” command stops the solving process, and using the “debug” command, the contents of some cells are read.

- two_sudokus.stim, which feeds the program with two sudoku boards, one after another. After solving the first and sending back the results, a "clear" command is executed and then the grid is filled with the second sudoku, it is solved and the results are sent back to the serial port.

Using the Microchip Studio debugging tools (Run To Cursor and Step Into) and by watching the memory contents (e.g. the values that are completed each moment in the sudoju grid, or if the flag bits are set to true/false), the functionality of the program can be evaluated.

Testing the code on real hardware (STK500)

Using PuTTY

Three tests have been done on hardware using PuTTY. The content of the PuTTY terminal is provided in the following files:

- simple_solve_tty.log, which feeds the program with one sudoku board (the one shown in the assignment), waits until the sudoku is solved and sends the results back.
- break_debug_tty.log, which feeds the program with a difficult sudoku board from `https://www.websudoku.com/images/example-steps.html` (to slow the solving process and have the time to give manually a break command), but while solving, a "break" command stops the solving process, and using the "debug" command, the contents of some cells are read.
- two_sudokus_tty.log, which feeds the program with two sudoku boards, one after another. After solving the first and sending back the results, a "clear" command is executed and then the grid is filled with the second sudoku, it is solved and the results are sent back to the serial port.

On each file, an identifier is placed in the start of each line, to show who sent this command ([PC] or [STK]).
Important note: While configuring the serial connection in PuTTY the flow control can be set either to "None", or to "XON/XOFF" without any problem. Even when using XON/XOFF, the baud rate is so low, so the buffers of the two communicating devices (the PC and the STK) will never overflow.

Using the interface program developed by Odysseas Stavrou

Although the program has still some bugs (reported to Odysseas) and part of the functionality cannot be tested with it (such as the break command), it can show useful statistics for the sudoku solver. It was used to find the solving time of many sudokus. Five random sudokus of each difficulty level were solved, giving the following results:

	Game 1	Game 2	Game 3	Game 4	Game 5
Easy	57 µs	33.6 µs	62.8 µs	22.4 µs	84.3 µs
Medium	7.2512 ms	72.5 µs	2.4176 ms	74.2 µs	81.7 µs
Hard	8.6831 ms	51.1 µs	16.447 ms	63.6 µs	53.9 µs
Ultra	432.3558 ms	2.123 291 8 s	53.717 825 9 s	200.7779 ms	7.296 628 1 s

In the final report (which will be submitted on the 22th of December, 100 boards of each difficulty level will be solved, to provide an accurate statistic performance analysis of the program. This procedure is postponed for now, because of the interface's bugs.

STK500 configuration

The STK500 development board we have received has installed the ATmega16L microcontroller, which is fully compatible with the ATmega16 used in Microchip Studio. PORTA is fully functional, and therefore connected to the LEDs. An external crystal of 10MHz is used. Since ISP (In System Programmer) programming mode is used, the following jumpers are mounted:

VTARGET mounted, to use the on-boars supply voltage.

AREF mounted, to use the AVR's AREF as reference volatge for the A/D converter on AVR. Although the A/D converter is not used in this project, the jumper is mounted, because this is the default setting.

RESET mounted, to be able to use the on-board reset button.

XTAL1 mounted, to use the external crystal.

OSCSEL mounted to pins 2 and 3 to use the on-board crystal signal as clock signal.

Conclusion - Future work