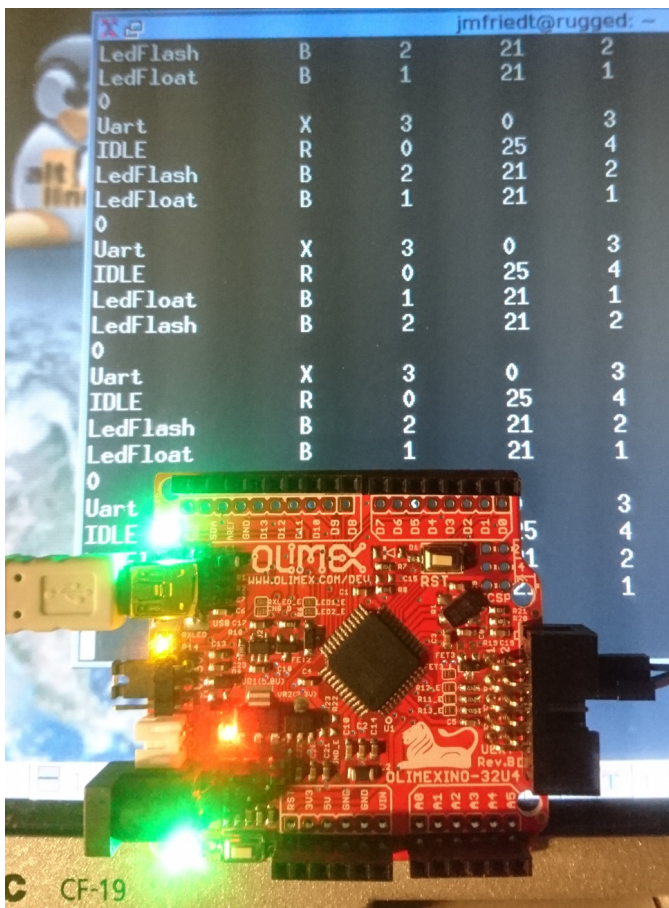


Embedded electronics exam: porting FreeRTOS to the Atmega32U4

J.-M Friedt, March 23, 2023

The Atmel Atmega32U4 microcontroller is a tiny 8-bit system fitted with a minute amount of RAM. Nevertheless, we wish to check if FreeRTOS can run on such a tiny device.

1. What is the address range of the RAM of the Atmega32U4? Where is the stack pointer allocated when compiling a C program to run on the Atmega32U4? Where/how did you find the information?
2. FreeRTOS handles its own memory space for allocating the stack of each task: what variable defines the memory space allocated to FreeRTOS and in which file is this variable defined?
3. We provide at http://github.com/jmfriedt/tp_freertos an Atmega32U4 directory as well as a Makefile.atmega32u4 in the 1basic subdirectory for attempting to run FreeRTOS on the tiny platform. Download the current (202212.01) version of FreeRTOS and compile the example provided in 1basic, pointing the various directory dependencies to match your installation setup. One might consider moving Atmega32U4 and its content to FreeRTOS/Source/portable/GCC since that would be its natural location to match the file organization of FreeRTOS, but doing so is not mandatory as long as the Makefile options match your file organization.
4. Which (port/pin number) are the LEDs toggled by Led_Hi1() and Led_Hi2? Where did you find the information?
5. Execute the compiled example on the Atmega32U4 emulator using the appropriate Makefile rule. How are the LED toggling represented in the emulator? Justify the output from the emulator when the ports change state (what is the first number displayed?)
6. Execute the compiled example on a genuine Atmega32U4 using the appropriate Makefile rule. What is the output on the communication port? How did you get this information?
7. Update the example to trap stack overflow and modify one of the tasks stack allocation to trigger the overflow. Demonstrate that the stack overflow trapping mechanism is functional, either on genuine hardware or on the emulator.
8. Add the display of the list of tasks, their execution status and the remaining memory on each task stack: what FreeRTOS instruction allows for achieving such a result?
9. Executing the output of the compilation will fail when linking with heap_3.c memory allocation implementation but should succeed with heap_2.c: analyze the memory management of the appropriate function implementation of FreeRTOS and identify the cause of the crash when calling memory allocation with heap_3.c and why heap_2.c solves the issue. Comment on the consequence of dynamic memory allocation on small footprint microcontrollers.
10. Analyzing the output of the previous task list and its memory allocation, how many tasks can you consider executing on the Atmega32U4 before it runs out of memory?



```
jmfriedt@rugged: ~/make -f Makefile.avr32u4 simu
simavr --freq 16000000 --mcu atmega32u4 output/main.elf
Loaded 14682 bytes at 0
Loaded 62 bytes at 800100
** DDRB(24) = 20
** DDRE(2d) = 40
** PORTB(25) = 20
Uart          .X.3.33.3..
LedFlash      .R.2.35.2..
LedFloat      .R.1.35.1..
IDLE          .R.0.35.4..
0..
** PORTE(2e) = 40
** PORTB(25) = 00
** PORTB(25) = 20
** PORTE(2e) = 00
** PORTB(25) = 00
Uart          .X.3.0.3..
IDLE          .R.0.25.4..
LedFlash      .B.2.21.2..
LedFloat      .B.1.21.1..
0..
** PORTB(25) = 20
** PORTE(2e) = 40
** PORTB(25) = 00
** PORTB(25) = 20
** PORTB(25) = 00
** PORTE(2e) = 00
Uart          .X.3.0.3..
IDLE          .R.0.25.4..
LedFlash      .B.2.21.2..
LedFloat      .B.1.21.1..
```