Computer architectures LAB 2

Assembler and basic I/O

L. Bruynseels ludo.bruynseels@kuleuven.be

# Goals of this lab session

* Configure Digital Inputs
* Learn about external interrupts.
* Use Timers on the microcontroller.

# Exercise 1 - Blink

* Clone (download) <https://github.com/ludobruynseels/GroepT-CA-LAB2.git>
* Open STVD (as administrator) and open the workspace that you just downloaded.
* Right click on ‘Blink’ and set as active project.
* What is the function of instruction **mov CLK\_CKDIVR, #%00011011** (See Reference Manual p 94
* Examine how register X is used as a 16 bit counter. Change the preset value to alter the blink frequency of the LED. The assembler directive DC.W is explained in UM0144-st-assembler-linker-user-manual on page 55. Very useful to initialize ROM locations to preset values!
* Watch PC increment as you step through the program.
* Examine the value of SP when a breakpoint is set in the main loop or in the delay subroutine. (Set breakpoint at line 58)

# Exercise 2 – SR flipflop

* Right click on on the ‘setreset’ project and set as active project.
* We will use the LED on port D6
* Notice #include "stm8s105k.inc" on line 4. Almost all the ‘EQU’ statements that we had to enter last week are in there!
* On the **schematic**, identify 2 pushbuttons.
* Their name = …………….
* The portnumbers are : ………
* Do we need to configure pull up resistors on the µC ? ( yes – no )
* Consult the STM8 reference manual (page 107) to figure out the configuration of the I/O ports. For each of these 3 ports you need to configure

Input or output,

floating or pullup (for inputs). Infer from schema.

**with** interrupt

Push-pull or Open drain (for outputs)

* Add initialization code to the project (just after label ‘init’ and before ‘mainLoop’ Use BSET and BRES to set/reset individual bits. Not MOV: only change bits that we need.
* In the Main loop, implement a SET/RESET flipflop: pressing one button switches the LED on, pushing the other button switches it off.
  + Use instructions BTJT (bit test jump if true) and BTJF (bit test jump if false)
  + To save time: draw a flowchart
  + Implementation = between 5 and 10 lines of code.

# Exercise 3 – SR FF with Interrupts

* Right click on the ‘sr-isr’ project and set as active project.
* Copy the config of the input from the previous project. But make sure interrupts are enabled! (See STM8 reference manual p 107 for the correct settings)
* Repeat: interrupts must be enabled in these digital inputs!
* Look up documentation about registers EXTI\_CR1 and EXTI\_CR2 in the reference manual. What is their purpose?
* Very small but very important: the ‘rim’ statement. Ask Toby what it means. Alternative: look it up in the reference manual.
* The implementation of the RS Flipflop is now in the ISR (Interrupt Service Routine)
* Find in the STM8S105K datasheet on page 44 which interrupt vector are associated with the digital inputs that we use. (or ask Toby)
* In the interrupt vector table replace ‘NonHandledInterrupt’ with ‘isr’
* Test your program.

# Exercise 4 – Timer

* Right click on the ‘timer’ project and set as active project.
* Timer init code is in file ‘timerinit.asm’. Expand the ‘Source Files’ node in the project to see it.
* A tuning fork is used by many musicians. The tone produced is 440 Hz (A4). Make your board to an electronic tuning fork.

It means: calculate TIM2\_ARR (automatic reload register) such that a tone of approx. 440 Hz is produced. Timer uses CPU clock = 16Mhz. TIM2\_CCR1 should half of this value to obtain a 50% duty cycle. You may experiment with this value as long as it lower than ARR. These edits must be done in file ‘timerinit.asm’.

* Design a system such that you can start and stop the sound with the buttons.
  + Possible solutions:
    - Start TIM2 when one of the buttons is pressed. Stop when the button is released.
    - Make reuse of the work in sr-isr: one button (the set button) starts the timer, the other button (reset) stops it.

# Exercise 5 – Flash

* Edit timerinit.asm and change TIM3\_AAR to flash the LED with a period of 1s. This means: interrupt every 500ms.
* This timer runs at 2 MHz and there is a prescaler of 1/32.