Lab 0

Johannes Lemonde et Lucas Streit

September 24, 2018

3 Questions to chapter 3

3.1.1 Flow diagram for the `delay` subroutine

See at the end.

3.1.2 Calculating `delaycount` in order to achieve `1 ms` in `delay`'s inner-loop

We have $\frac{50\ [MHz\ op]}{4\ [\frac{op}{loop}]\ 1000\ [\frac{ms}{s}]} = 12'500[\frac{loops}{ms}]$, so we would set `delaycount` to 12'500.

Experimentally, we figured out that it should be set to 860 approx (why?)

3.2.1 What is BSP and what does it contain?

BSP means Board Support Package. It is the layer of software containing hardware-specific drivers and other routines that allow a particular operating system to function. It initialises the processor, bus, interrupts, clock, RAM, amongst others, and runs the boot loader.

The BSP project in Eclipse contains amongst others all the files necessary to access the LEDs, switches, buttons, etc.

3.2.2 What is a soft-core processor? What type of processor is present? Features?

We talk about soft-core processors when we configure a reprogrammable logic, such as a FPGA, to function as a processor. For these laboratories, we use the NIOS II soft processor, which is a soft microprocessor core for Intel FPGAs.
Clock frequency: 50 MHz; neither data nor instruction caches for NIOS II/e; No pipeline nor branch prediction for NIOS II/e.

3.2.3 How does the processor access peripherals? What is the main mechanism and what main piece of hardware stands between the processor and a peripheral?

Peripherals are accessed through the Avalon Switch Fabric, which lets several masters from the core to operate at the same time. In other words, in handles the requests from the masters to the slaves (peripherals).

3.2.4 What peripherals/IP cores are present in the provided architecture? Identify their symbolic names and base addresses.

Buttons, switches, green and red LEDs, seven segment displays, ... For the symbolic names and base addresses, see `system.h` (within the BSP project in Eclipse).

3.2.5 What types of memory are present in the provided architecture? Identify their symbolic names, base addresses and sizes. What are the typical access time for each of these memories?

On chip memory : `ONCHIP_MEMORY_BASE`, address: `0x0`, size: `25600 words`, typical access time: `1 cycle at 50MHz`.

SDRAM: `SDRAM_BASE`, base-address: `0x1000000`, size: `8388608 words`, typical access time (CAS latency): `3 cycles at 50 MHz`.

3.3.1 What macro is used to write to a parallel I/O port? How about to read from one?

We can use the macros `IOWR_ALTERA_AVALON_PIO_DATA(base, data)` and `IORD_ALTERA_AVALON_PIO_DATA(base)`, defined in `$BSP_PROJECT/drivers/inc/altera_avalon_pio_regs.h`, where `base` is the address of the port and `data` is the data to write.

3.3.2 What command will you use to turn on all red LEDs?

We have to run the previous command with `DE2_PIO_REDLED18_BASE` as the `base` and $(2^{18} - 1)$ as the `data`. It would look like that: `IOWR_ALTERA_AVALON_PIO_DATA(DE2_PIO_REDLED18_BASE, 0x3FFFF);`

3.3.3 What command would you use to read the current status of all push-buttons?

`IORD_ALTERA_AVALON_PIO_DATA(D2_PIO_KEYS4_BASE);`, where the function returns a 4 bit number, each bit corresponding to the status of one push-button, `0` meaning currently pushed and `1` meaning released.

3.3.4 Which header file do you need to include in your program to access the symbolic names? What about the read/write macros?

Basically, `"system.h"` defines the base-addresses of the peripherals (symbolic names), such as for instance `D2_PIO_KEYS4_BASE`, and `"altera_avalon_pio_regs.h"` defines the macros to read/write.

3.6.1 What is the IRQ level for buttons?

`#define D2_PIO_KEYS4_IRQ 8` in the `system.h` file.

3.6.2 Which other peripherals support interrupts?

The toggles (IRQ 6), the JTAG UART (IRQ 5), Timer 0 (IRQ 7), Timer 1 (IRQ 9). (found in `system.h`).

5.1.1 How are the timestamp timer and the performance counter functioning? Describe the main steps.

The timestamp is basically a counter which gets incremented – in this case – at every tick of the clock.

The performance counter is a peripheral containing several counters allowing to count the number of ticks within a section of code chosen for analysis, without

Results :

	Time (ms)	Error (ms)
Timestamp timer	54.222	0.273
Performance counter	TBD	TBD

Flag	Time (ms)	Size (KB) (ELF ; .o)
-O0	54.259	899.726 ; 10.236
-O2	54.223	899.726 ; 10.236
-Os	54.170	899.726 ; 10.236

5.3.1 Why SDRAM rather than SRAM ?

SDRAM is cheaper than SRAM, even if it is slower. In many cases, this is a deciding point. Moreover, it is possible to have much bigger blocs of memory, so it is possible to optimise locally more data, which is used together, in order to reduce the access times using a cache.

5.3.2 Typical access times:

SDRAM	(10ns)
External SRAM	0.5 - 5 ns
On-chip RAM	
Cache	

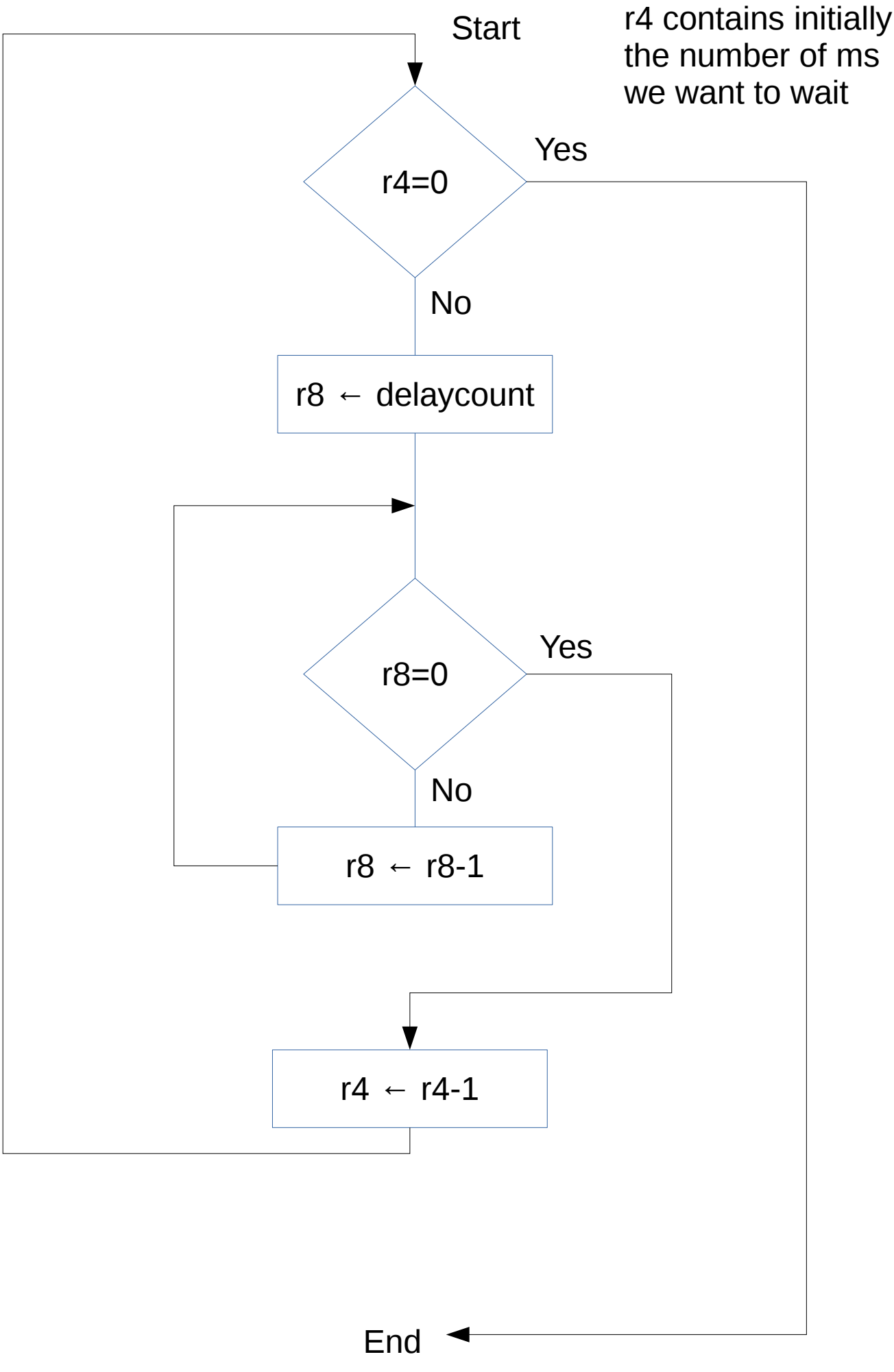
6 Questions to ask

Optimisations : -O0, -O2, -Os

Sur quelle mémoire on charge le programme ? (comment choisir)

Qu'entendent-ils par plot (à la toute fin) ?

Performance counter : comment faire. Note à moi-même : vérifier que c'est pas un overflow.


Start
r4 contains initially the number of ms we want to wait
r4=0
Yes
No
r8 ← delaycount
r8=0
Yes
No
r8 ← r8-1
r4 ← r4-1
End