NTNU – Trondheim
Norwegian University of
Science and Technology

Recap
TDT4258 Microcontroller System Design Lab

Stefano Nichele
Department of Computer and Information Science
2011, April 5th

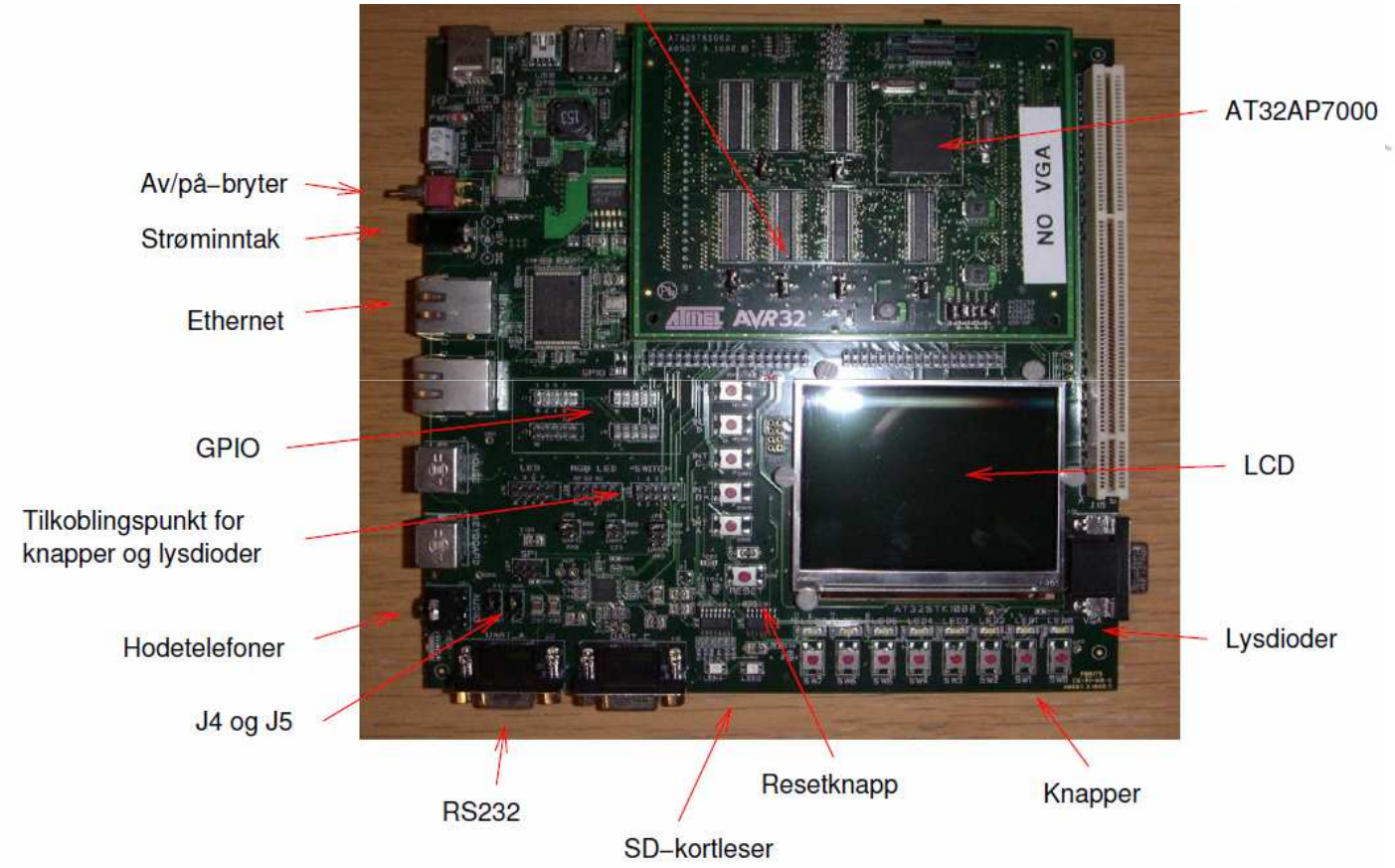
Exercise 3 – pong game

- Deadline: April, 7th kl. 20.00 – It's Learning
- 1 report + code each group
- (Brief) presentation to vit.ass the week after submission (only selected groups). The presentations will be held in the lab.

Exercises objectives

- Microcontroller programming (C and assembler)
- I/O programming
- Interrupts
- Programming on Linux Kernel
  - Create your own hardware driver
- Development using GNU tools

STK1000 dev. board: why?



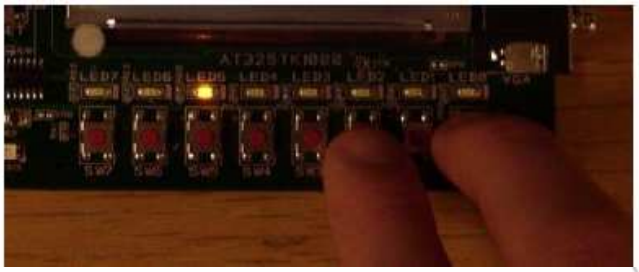
NTNU – Trondheim
Norwegian University of
Science and Technology

AVR32 vs. AT32AP7000

- AT32AP7000 microcontroller:
  - AVR32-based microcontroller (32-bit RISC processor from Atmel)
  - Many built-in I/O devices:
    - General I/O pins (buttons, LEDs)
    - DAC (audio)
- AVR32 microprocessor:
  - Registry: 16 registers
    - 13 general: r0 – r12
    - Link Register (lr), Stack Pointer (sp), Program Counter (pc)
  - Many system registers, including:
    - Status register
    - EVBA

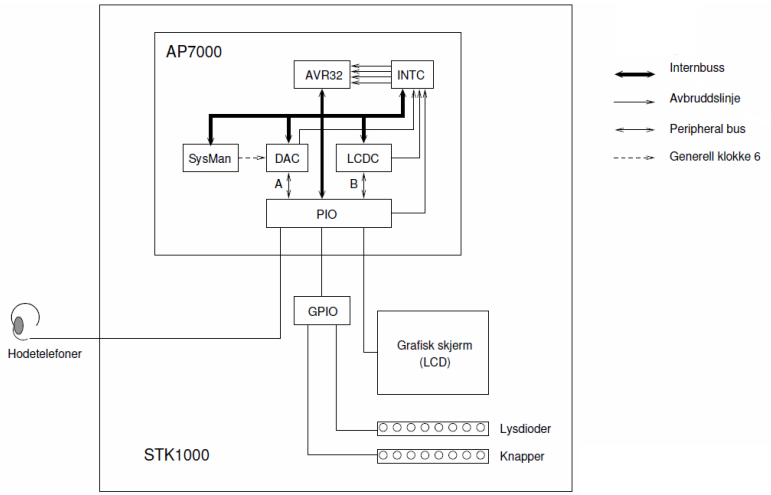
1st Exercise

- Assembly (based on load/store)
- I/O (buttons & leds)
- Memory mapped
- Interrupt (routine)



Parallel I/O: PIO

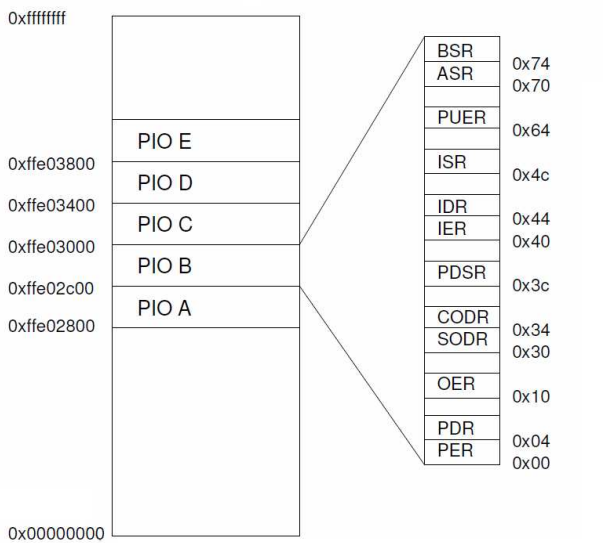
- I/O-controller: internally on the microcontroller
  - Controls the I/O pins of the microcontroller
  - General I/O pin:
    - Either input or output
- The microcontroller has memory mapped I/O:
  - Each I/O controller has a set of registers, each register is mapped on a specific address in the processor's address space
  - I/O controllers are controlled / programmed by writing to these registers



PIO

- 5 PIO ports, port A-E (5 sets of memory mapped registers)
- Each PIO port has 32 bits
  - 32 I/O pins per PIO port
  - Each register has 32 bits, each bit corresponds to a given I/O pin on the microcontroller

Memory map



NTNU – Trondheim
Norwegian University of Science and Technology

Interrupt

- Instead of polling I/O devices
- I/O units provide information when they want attention
- CPU saves the state of its parts and jumps to an interrupt routine
- Jumps back when the interrupt routine is completed

Exercise 2

- Produce sounds with built-in DAC
  - Use a clock to generate interrupts regularly
  - An interrupt routine is feeding the DAC with audio samples
- C language
  - High-level language with good low-level opportunities
  - Similar to Java – not object oriented
  - Pointer (variable that holds memory addresses)

```
int a = 5;          // variable of type int
int *p;             // pointer to int
p = &a;             // set p to point to a
*p = 42;            // modify the value pointed by p  (dereference)
```

Exercise 3

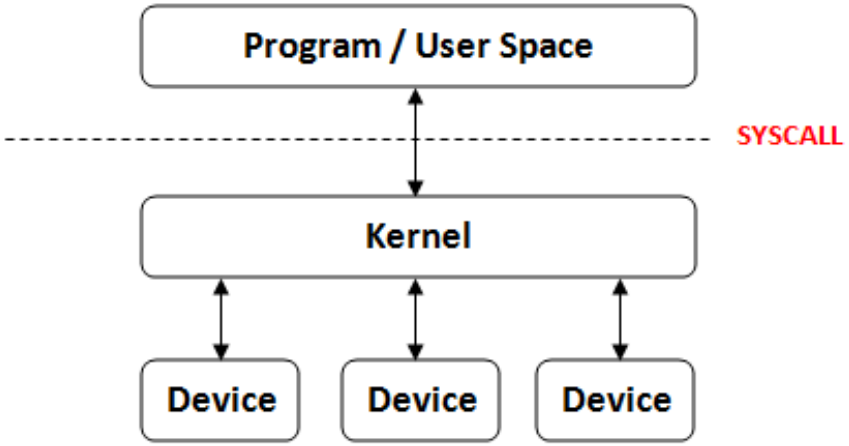
1. Write a Linux driver (for the use of buttons & LEDs)
   - Device driver: software layer between the applications and the actual device
     - they hide the details of how the device works
     - they make a particular piece of hardware respond to a well defined programming interface
     - can be built separately from the rest of the kernel and ”plugged in” when needed
2. Pong game that runs under Linux on STK1000

IO devices in Linux

- IO devices are represented by special files in */dev* directory
- To make the I/O
  - Open (with the system call *open*) the file that represents the device to use
  - Execute *ioctl* call, if necessary
  - Read/write with *read* / *write* using *lseek* to switch position
  - Close the file (*close*)

Drivers

- The drivers should be created as kernel modules
- The driver should be the only part of the system that has direct access to the relevant PIO registers



Examination plan

Exercise 1	20 %	
Exercise 2	20 %	
Exercise 3	20 %	
Final Test	40 %	(Apr, 12th)
Total	100 %	

LAST EFFORT!!