



Computer Hardware Engineering (IS1200)

Computer Organization and Components (IS1500)

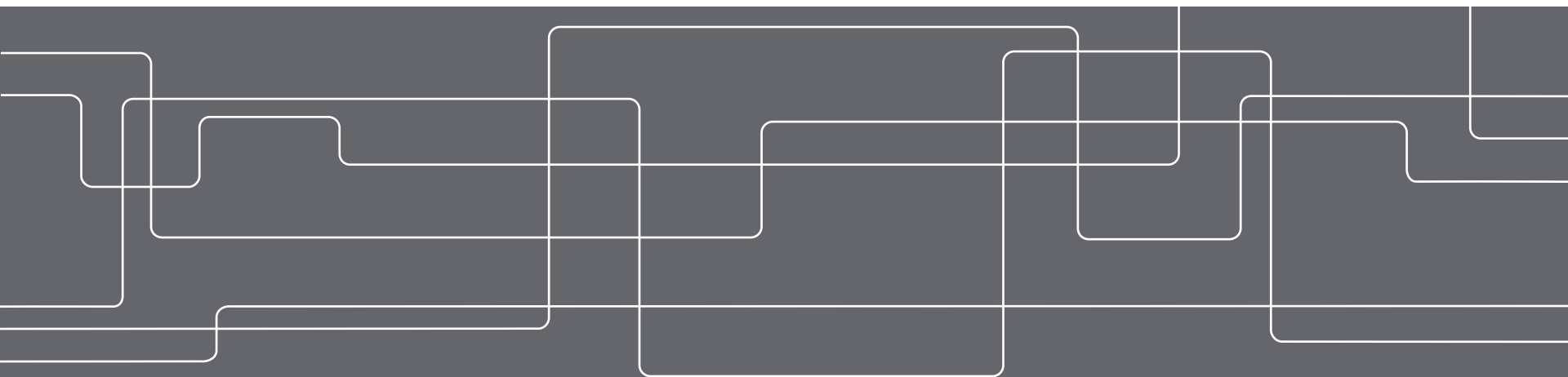
Spring 2021

Lecture 6: I/O Systems, part II

Daniel Lundén

PhD Student, KTH Royal Institute of Technology

Original Slides by David Broman (extensions by Artur Podobas), KTH





Announcement

Student representatives:

CINTE

- Frej Larssen, flarssen@kth.se

TCOMK

- Alicia van Zijl, aevz@kth.se

You can contact them if you have any feedback for us (you can of course also contact us directly).

Part I
Timers

Part II
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

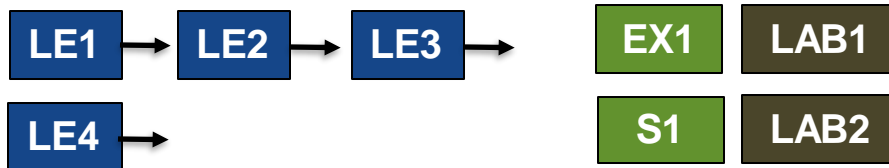
Part IV
Mini
Project



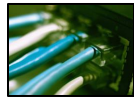
Course Structure



Module 1: C and Assembly Programming



Module 4: Processor Design



Module 2: I/O Systems



Module 5: Memory Hierarchy



Module 3: Logic Design (IS1500 only)

**PROJ
START**



Module 6: Parallel Processors and Programs



Proj. Expo

LE14

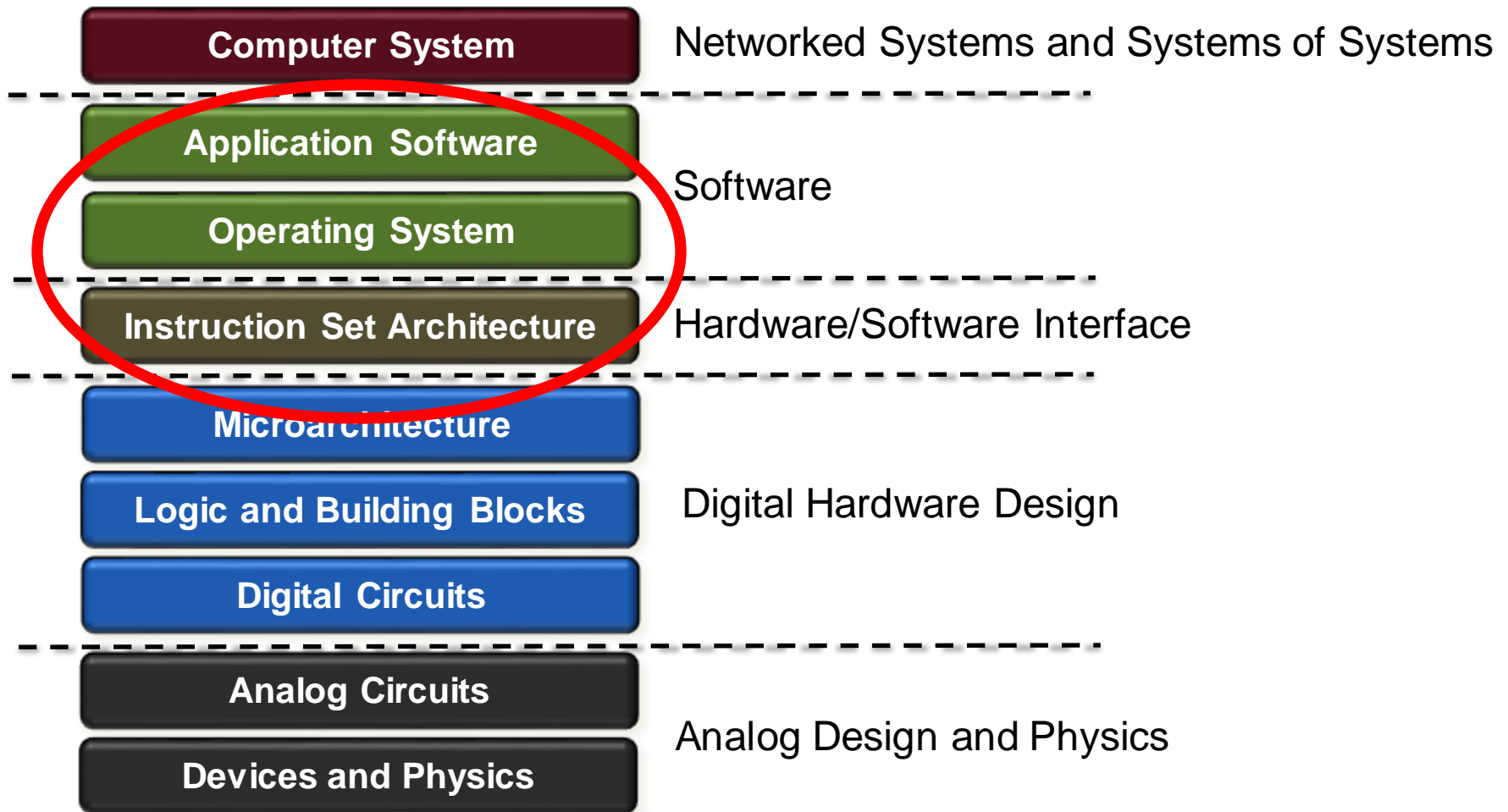
Part I
Timers

Part II
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

Part IV
Mini
Project

Abstractions in Computer Systems



Part I

Timers



Part II

Parallel and Serial Communication



Part III

Exceptions and Interrupts



Part IV

Mini Project



Part I

Timers



Acknowledgment: The structure and several of the good examples are derived from the book “Digital Design and Computer Architecture” (2013) by D. M. Harris and S. L. Harris.



Part I
Timers

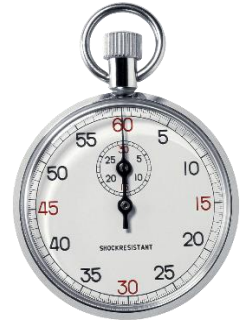
Part II
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

Part IV
Mini
Project

Timers

Timers are I/O devices that are used to measure elapsed time. Timers can be configured in different ways, but have basically the following components.

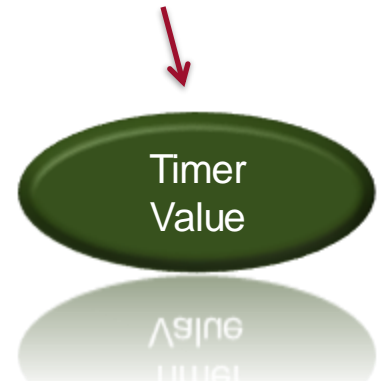
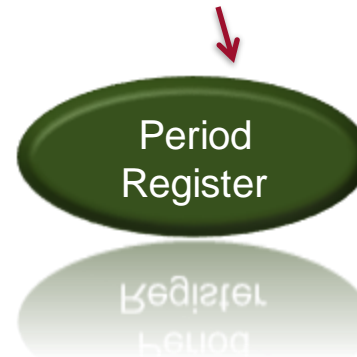
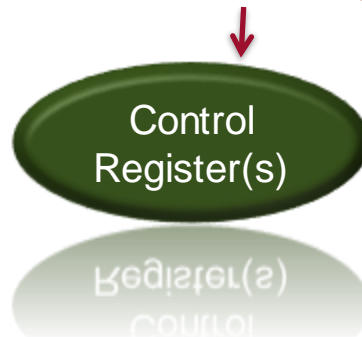
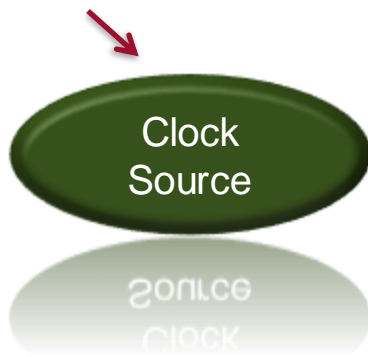


A timer is driven by a **clock source**. Example, 50Mhz external clock.

- Tells if time-out is reached.
- Stop/start counting
- Check if running.

Write registers, states the period

Read/write register. A “snap-shot” of the timer value.



PIC32 Timers – Searching for information

On the ChipKIT Uno32 board, we have a **PIC32MX320F128H** processor.



How can we find this information?

See the *chipKIT Uno32 Board reference manual*. Here called [UNO32 ref].



This processor has five 16-bit timers.

How can we find this information?

See the **PIC32MX3XX/4XX Family Data Sheet**. Here called [PIC32MX ref].

The PIC32 processor has two types of timers:

- **Type A timers (Timer 1):**
Can operate on an external clock
- **Type B timers (Timers #2-5):**
Can be combined to form 32-bit timers.
Runs with the peripheral clock.

How can we find this information?

See the **PIC32 Family Reference Manual, Section 14. Timers**.
Here called [PIC32Family ref, Sec 14].

See the course web, page “Literature & Resources” for links.



Part I
Timers

Part II
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

Part IV
Mini
Project

PIC32 Timers



Clock
Source

Timer 2 (that we will use) operates at 80 MHz.

Control
Register(s)

TxCON (x is the timer number)

- 16-bit control register.
- Configures the prescaling
- Starts/stops the timer

Period
Register

PRx (x is the timer number)

- The period register.
- TMRx counts upwards until reaching this value.

Timer
Value

TMRx (x is the timer number)

- Holds the current 16-bit timer value
- May be written to.

IFS0

- Interrupt register indicating when the counter has reached the period value.
- Note that the define in pic32mx.h is **IFS(0)** and not **IFS0**.



Part I
Timers

Part II
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

Part IV
Mini
Project

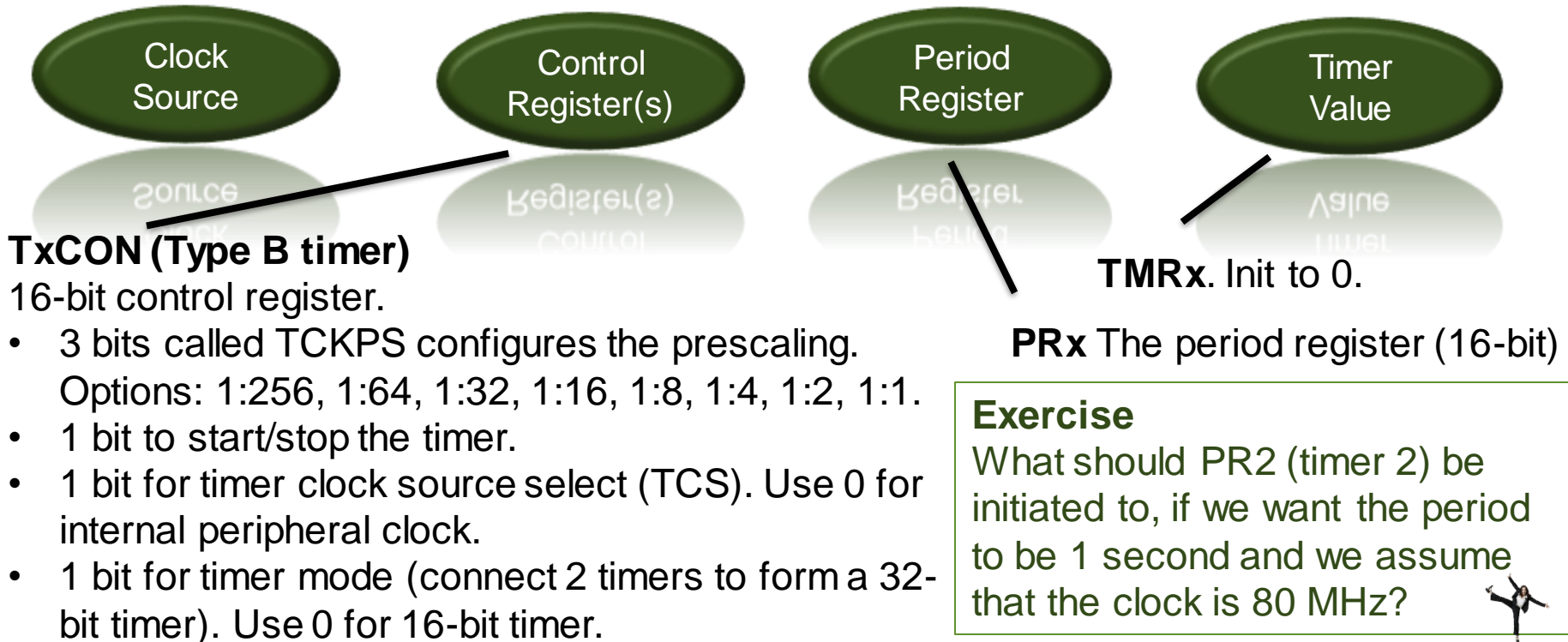
PIC32 Timers – Initialization (1/2)



10

E

E



For details on the bit layout, see
[PIC32Family ref, Sec 14, Page 14-9]

Solution: Not possible for a 16-bit timer.
 $80\,000\,000/256 = 312\,500$.
 Cannot fit in a 16-bit timer.



Part I
Timers

Part II
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

Part IV
Mini
Project

PIC32 Timers – Initialization (2/2)



Clock
Source

Control
Register(s)

Period
Register

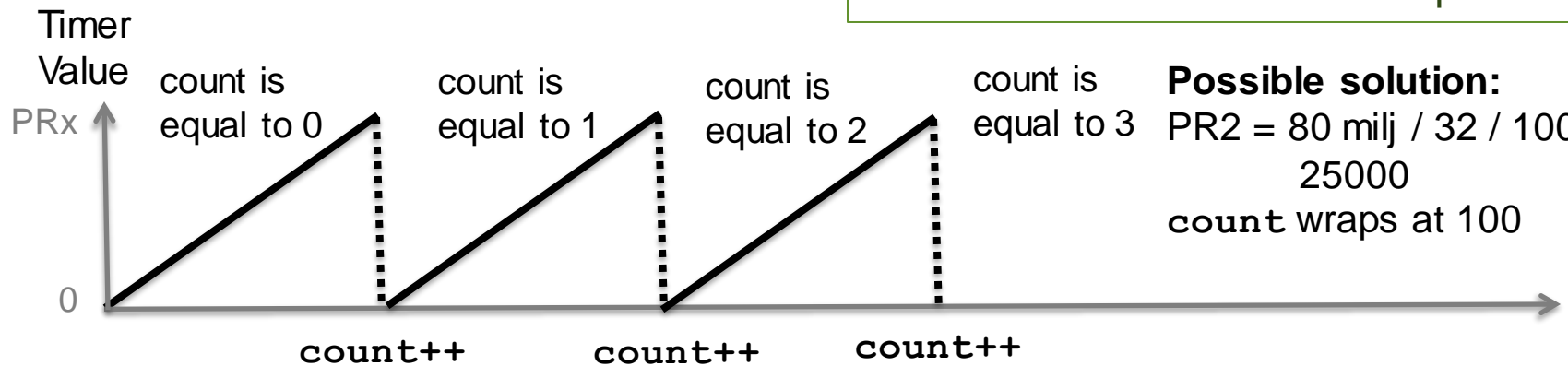
Timer
Value

Alternatives (previous exercise)

1. Use a 32-bit timer
(combine two 16-bit timers)
2. or, add another counter in the program.
`int count = 0;`

Exercise

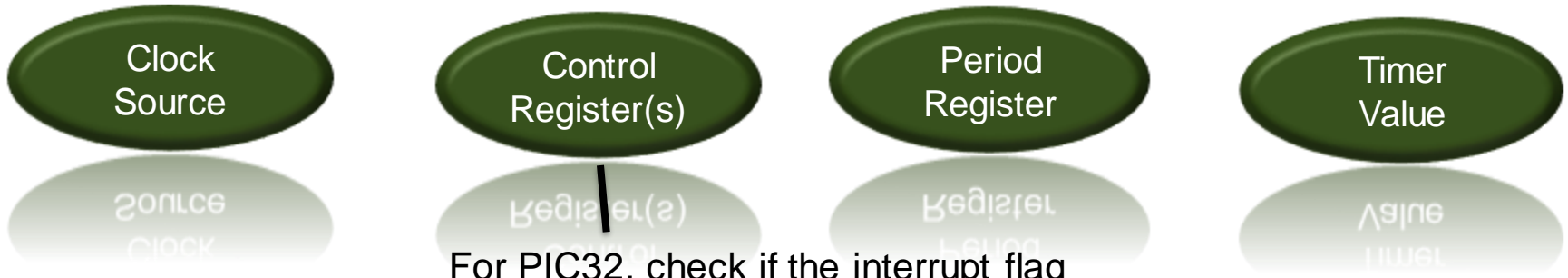
If prescaling is 1:32, and we use alternative 2, which number should then PR2 have if we want the software counter to wrap after 1 second? When should the `count` variable wrap?



Possible solution:

$$PR2 = 80 \text{ milj} / 32 / 100 = 25000$$
 count wraps at 100

PIC32 Timers – Polling

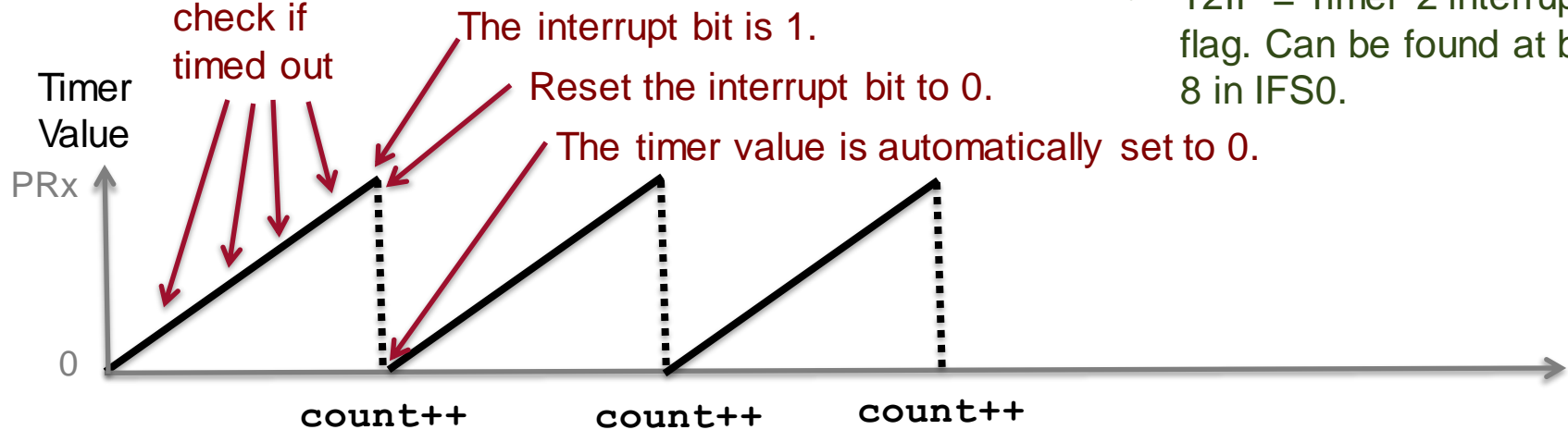


For PIC32, check if the interrupt flag IFS0 is set for the relevant timer.

Polling:
Repeatedly
check if
timed out

Look at TABLE 4-4 on page 53 in [PIC32MX ref].

- T2IF = Timer 2 interrupt flag. Can be found at bit 8 in IFS0.




Part II

Parallel and Serial Communication



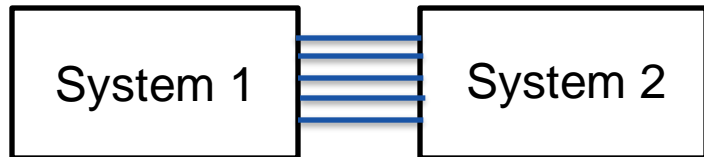
Part I
Timers

 **Part II**
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

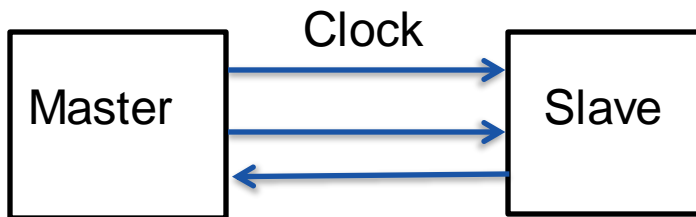
Part IV
Mini
Project

Parallel vs. Serial Communication



Parallel Communication

Several wires that transmit data in parallel.
Problem: Requires many cables



Synchronous Serial Communication

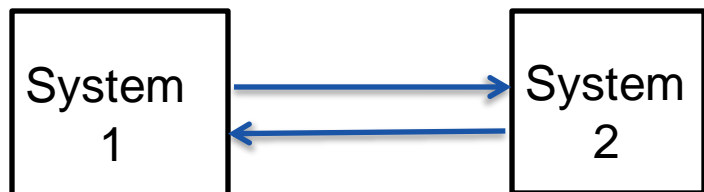
Examples:

Serial Peripheral Interface (SPI)

- Full duplex
- 4 wires, fast transfers, (10-100 MHz)

I²C (Inter-integrated circuit)

- Two lines: Serial Data Line (SDL) and Serial Clock Line (SCL)
- 7 or 10 bits addresses. (Fast mode 400kHz)



Asynchronous Serial Communication

Example: **UART** (pronounced “you-art”)



PC I/O Systems

- Demand for very high **performance**.
- Must be easy to **add devices**. Consequence: complex I/O protocols.

Peripheral Component Interconnect (PCI)

- Used for adding expansion cards (from 1994).
- For example, Ethernet directly on the motherboard.
- Fast (e.g., 64 GB/s for PCIe 5.0).




Universal Serial Bus (USB)

- Taking over. High speed serial communication.
- Versions 1.0 to 3.0. Up to 5Gbit/s.
- Simple for users, complex hardware and software.

Other Devices

- Ethernet (Wired networking).
- Wi-Fi (Wireless communication).
- SATA (Serial interface to hard disks).

Part I
Timers

 **Part II**
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

Part IV
Mini
Project

Part III

Exceptions and Interrupts



Acknowledgment: The structure and several of the good examples are derived from the book “Digital Design and Computer Architecture” (2013) by D. M. Harris and S. L. Harris.

Part I
Timers

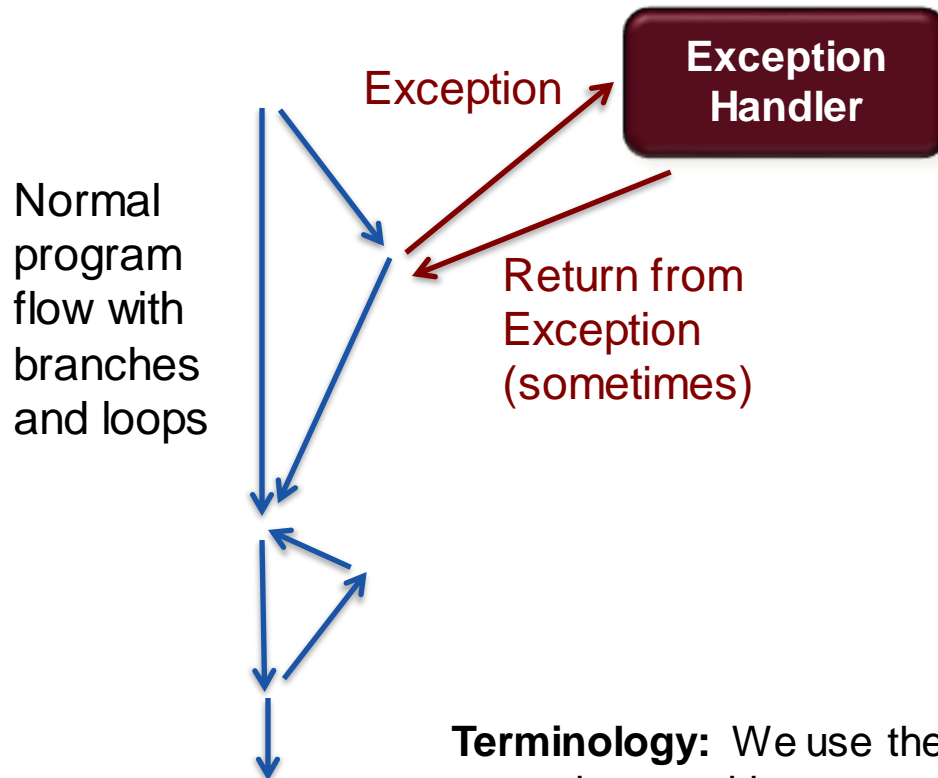
Part II
Parallel and
Serial Communication



Part III
Exceptions
and Interrupts

Part IV
Mini
Project

Exceptions and Interrupts



Causes of Exceptions

- **Error conditions.** For instance division by zero, unknown instruction etc.
- **Interrupts.** External interrupt, e.g., keyboard event.
- **Software Interrupts (also called traps).** For instance system calls (from user to supervisory mode).

Terminology: We use the MIPS terminology to differentiate between exceptions and interrupts. Some authors use the term interrupts for everything (e.g. used in x86 terminology).

Exception Handling in MIPS (general description)

In MIPS, the exception handler is located at address 0x80000180.

Exception handling step-by-step procedure

1. The processor stores the current PC in a register called **EPC**.
2. The processor stores the cause of the exception in register called **Cause**.
3. The processor jumps to address 0x80000180 where the exception handler is stored.
4. The exception handler saves registers on the stack.
5. The exception handler inspects the cause of the exception and handles the exception. EPC and Cause are not normal register. They are accessed using instruction **mfc0** (move from coprocessor 0).
6. The exception handler restores saved registers, copies the return program pointer EPC to \$k0, and jumps back using **jr \$k0**.
Alternatively, for MIPS32, use the **eret** instruction.

Exception Handling in MIPS (general description)

| Name | Number | Use |
|------------------|--------------|--------------------------------|
| \$0 | 0 | constant value of 0 |
| \$at | 1 | assembler temporary |
| \$v0-\$v1 | 2-3 | function return value |
| \$a0-\$a3 | 4-7 | function arguments |
| \$t0-\$t7 | 8-15 | temporary (caller-saved) |
| \$s0-\$s7 | 16-23 | saved variables (callee-saved) |
| \$t8-\$t9 | 24-25 | temporary (caller-saved) |
| \$k0-\$k1 | 26-27 | reserved for OS kernel |
| \$gp | 28 | global pointer |
| \$sp | 29 | stack pointer |
| \$fp | 30 | frame pointer |
| \$ra | 31 | function return address |



Exception Vectors and the ARM ISA

In **ARM**, the cause of the exception is shown by jumping to different exception handlers, depending on the cause. This is called an **exception vector**.

The vector will normally contain jumps to where the actual handler is stored.



| Exception | Vector address |
|--------------------------|----------------|
| Reset | 0x0000 0000 |
| Undefined instruction | 0x0000 0004 |
| Software Interrupt (SWI) | 0x0000 0008 |
| Prefetch abort | 0x0000 000C |
| Data abort | 0x0000 0010 |
| IRQ (normal interrupt) | 0x0000 0018 |
| FIQ (fast interrupt) | 0x0000 001C |

PIC32 can be configured to use either single vector (with cause register) or multi-vector mode.





Maskable Interrupts and Priorities

External interrupts may be enabled (allowed) or disabled. This is typically configured by using an interrupt flag status register.

Interrupts can be given different **priorities**. An interrupt with higher priority will **preempt** a lower priority interrupt.



For instance, a timer might have higher priority than the UART and will preempt the UART handling routine.



Non-maskable interrupts are interrupts that cannot be masked (disabled). Typically involves non-recoverable hardware errors.

Part I
Timers

Part II
Parallel and
Serial Communication



Part III
Exceptions
and Interrupts

Part IV
Mini
Project

PIC32 Interrupts - Initialization

IECx (Interrupt Enable Control)

- The initialization procedure should set the bit for the interrupt that should be enabled.

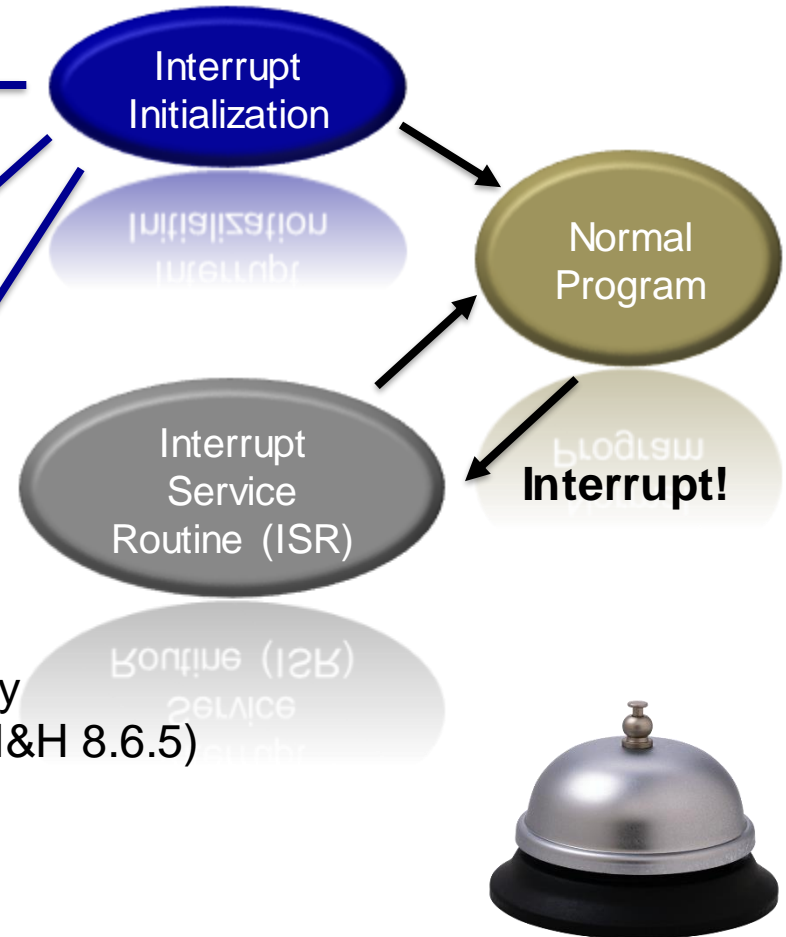
IPCx (Interrupt Priority Control)

- Configure the priority (0-7, where 7 is highest) and the sub-priority (0-3 where 3 is highest).

Enable Interrupts

You need to enable interrupts by executing instruction `ei` (See H&H 8.6.5)

The x means that there exist several registers, e.g. IPC0, IPC1, etc. Note that for pic32mx.h, the syntax is e.g., IPC(1).



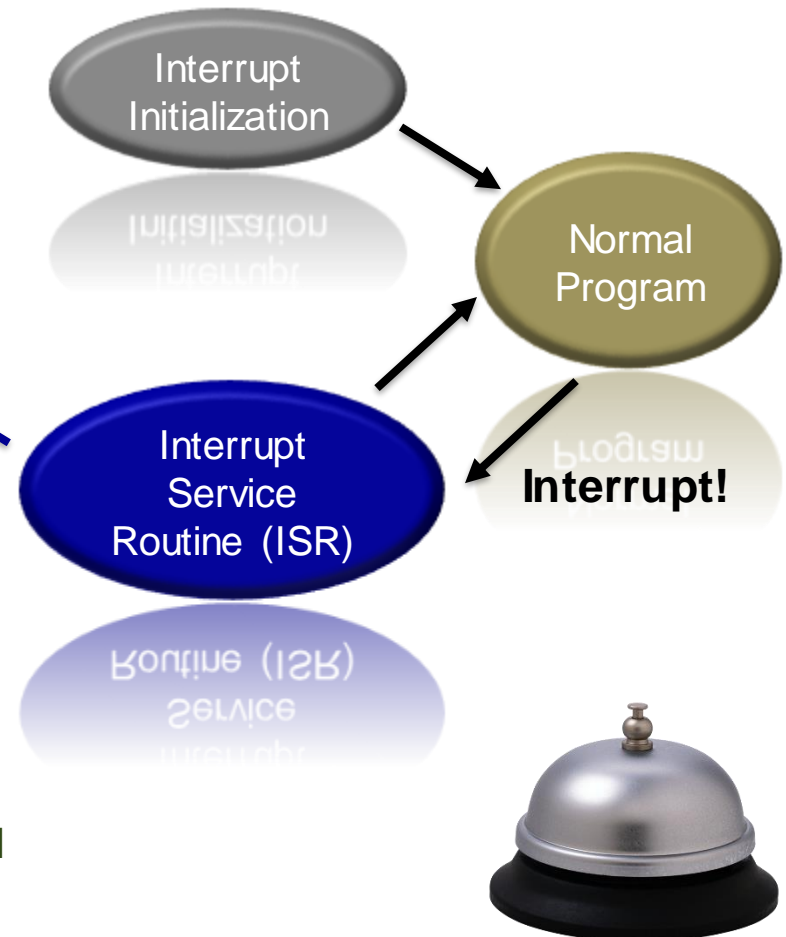
PIC32 Interrupts – Interrupt Execution

IFSx (Interrupt Flag Status Register)

- The interrupt request sets the bit (e.g. for a timer at timeout).
- The ISR must clear the flag before returning from the ISR.

The relevant bits for IECx, IPCx, and IFSx can be found in TABLE 4-4 on page 53 in [PIC32MX ref] or on page 90 (table).

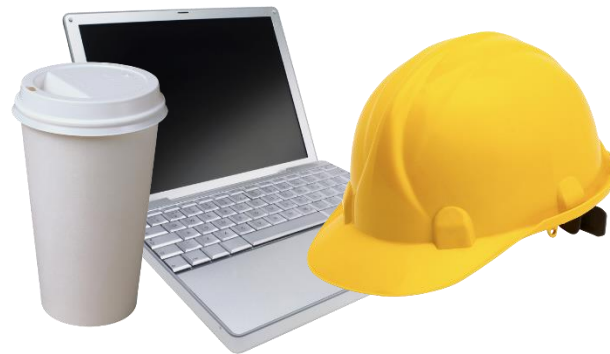
Advice for lab 3. Use the above and find:
i) T2IF (for timer 2 interrupt flag), ii) T2IE (for timer 2 interrupt enable), iii) T2IP for priority, and iv) T2IS for sub priority. Set highest priorities.





Part IV

Mini Project



Part I
Timers

Part II
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

 **Part IV**
Mini
Project

Mini Project

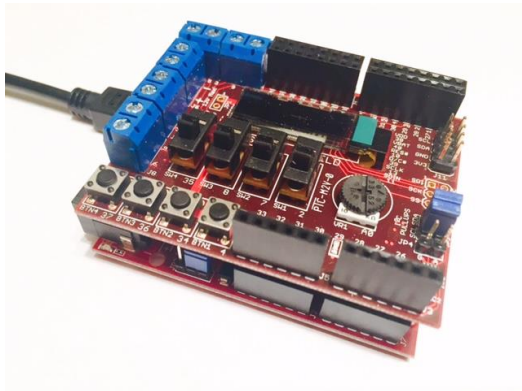


Groups

- Preferably, you work in the same group as you do with the labs (1-2 students)..

Equipment

- You must use the PIC32 platform supplied by the course.
- You may extend this platform in any way you want (Arduino shields, custom electronics, communication with computer...)
- You may use the MCB32 tool chain **or** the MPLAB X IDE, but **not** the MPIDE (a ported Arduino IDE)
- Use a version handling system, such as Git or SVN (recommended).



Part I
Timers

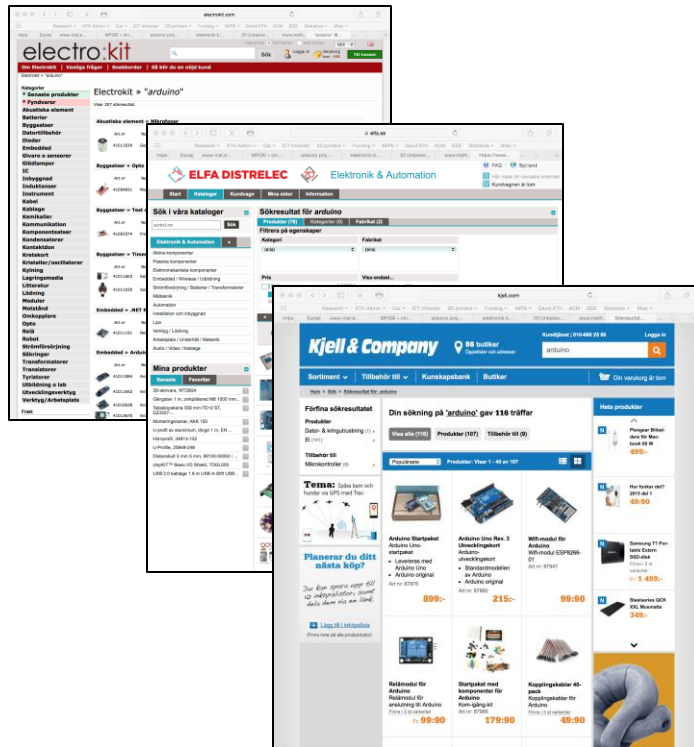
Part II
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

 Part IV
Mini
Project



Mini Project



Shields and Electronics

You may (optionally) extend your project with electronics. Check out the following for inspiration:
<http://www.instructables.com/id/Arduino-Projects/>

KTH cannot supply more hardware than the boards. If you want to buy e.g. shields, check out:

- Kjell & Company: www.kjell.se
 - ELFA: www.elfa.se
 - Electrokit: www.electrokit.se
 - Or international delivery, e.g. www.aliexpress.com or www.ebay.co.uk
- Search for e.g., “arduino sensors” or “pickKIT”

Part I Timers

Part II Parallel and Serial Communication

Part III Exceptions and Interrupts

Part IV Mini Project



Mini Project



What kind of project?

It is very flexible and it is up to you what you want to do. Examples:

- Make a game (snake, Tetris, ping/pong, Flappy bird, etc.). This is by far the most common project choice among students (historically), since it does not require any external hardware.
- Use serial communication or WIFI. Solve a problem.
- Use sensors such as temperature sensor (available on the I/O board), accelerometer etc.
- Do something with sound (add microphone or speaker). Play with A/D (built in) and alternative ways for analog output (see H&H section 8.6.6)
- ...

Part I
Timers

Part II
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

 **Part IV**
Mini
Project

Mini Project – Requirements

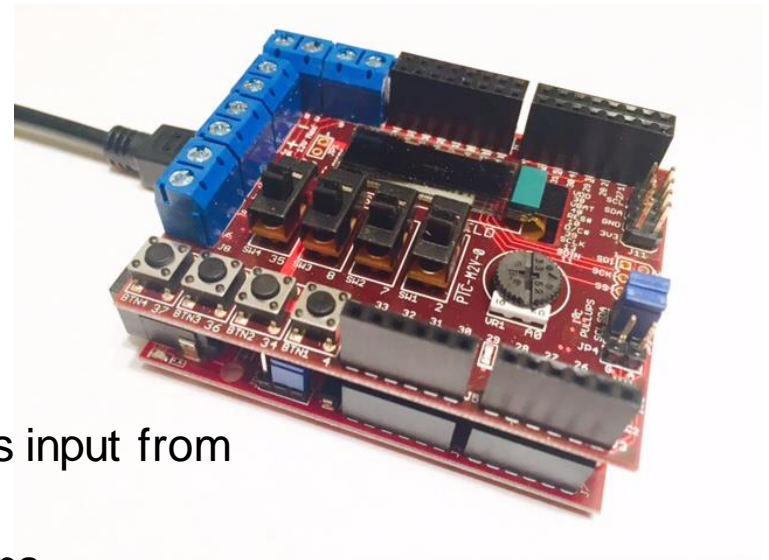
Basic Project

- Should use some other device/component besides the LEDs, switches, and push buttons on the board (e.g. display, temperature sensor etc.)
- Should be non-trivial and actually perform something. Should make use of timers and/or interrupts.

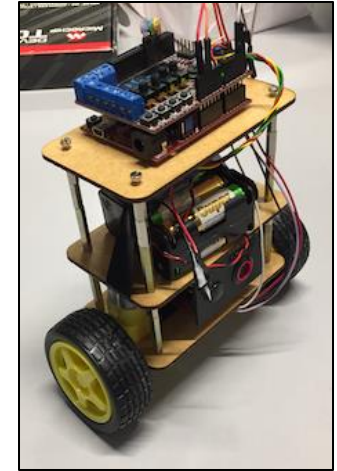
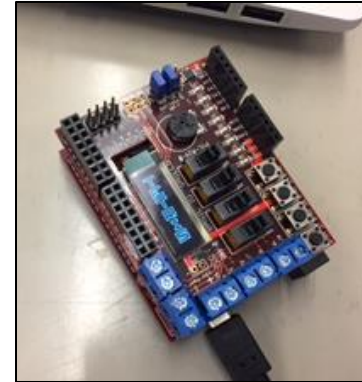
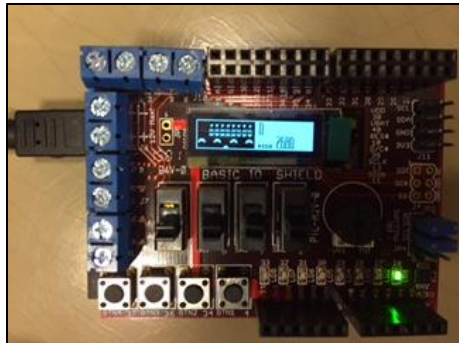
Not qualified as a project

- A program with very simple logic that just reads input from the buttons and displays some text.
- An implementation with a lot of code duplications.

In general, if you start to copy and duplicate your own code, then you are doing something that is not very good.



Mini Project – Requirements



Advanced project

Requirements for getting a final grade A or B. The main categories:

1. **[Graphics]** Sophisticated graphics on the screen (see examples on the next pages).
2. **[Protocol]** Use external components in a complex manner by using specific protocols: SPI, I²C, or or another non-trivial protocol (please check with Artur). UART is not counted as advanced.
3. **[Sound]** Advanced interaction with sound (see examples on the next pages).

Besides the above, the program logic must be rather complex. It must clearly show that the project required some substantial development effort.

Part I
Timers

Part II
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

 Part IV
Mini
Project



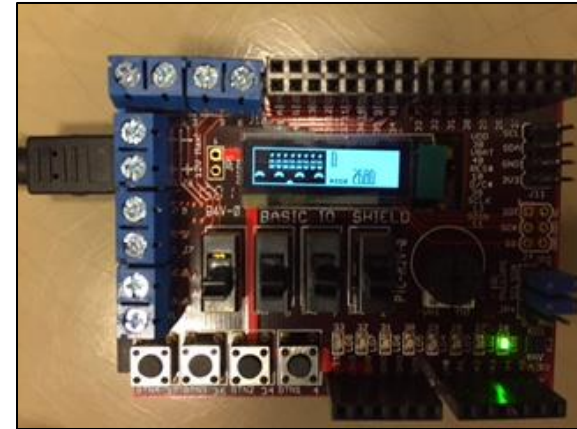
Examples – Advanced projects

[Graphics] Advanced graphics on the basic I/O shield's OLED display

Requirement: Pixel-by-pixel movements of objects that are larger than 2x2 pixels. The objects move smoothly 1 pixel at each frame update, both in X and Y directions.

Example 1: A pong game where the ball is 3x3 pixels, and moves according to the pixel-by-pixel requirement. The game supports several advanced features, such as 2 player mode, 1 player mode, simple AI for the computer player, different difficulty levels, bouncing angles, high score, and more.

Note that simple games like Pong, Snake, Tetris etc. need to fulfil the pixel-by-pixel requirement above **and** support advanced features such as different player modes (1 and 2 player) and difficulty levels.



Part I
Timers

Part II
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

 **Part IV**
Mini
Project



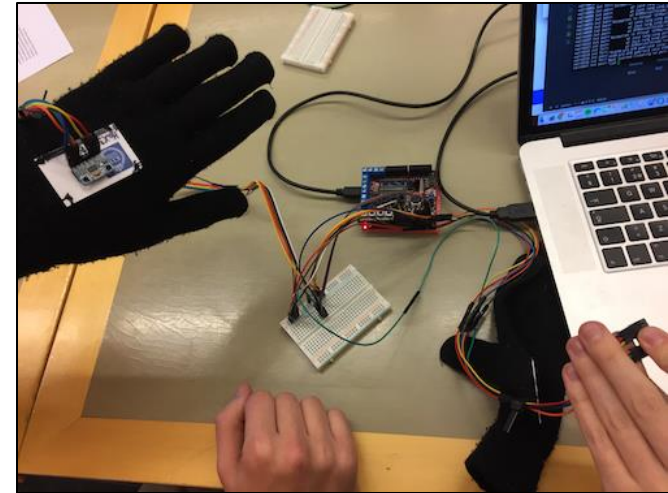
Examples – Advanced projects

[Protocol] Advanced protocols

Requirement: Make use of SPI, I²C or other advanced protocols.

Example 1: A program that is using external accelerometers, which communicate using I²C. The application shows a non trivial C program (e.g., a simple snake game).

Example 2: A program that is using an external display, which is communicating over SPI. The program itself is non-trivial, with different modes, menus, user interactions etc.





Examples – Advanced projects

[Sound] Advanced interaction with sound

Example 1: A music player that reads sampled music files (e.g. wav files) from the flash and outputs the sound to the speaker. The user can select songs, move forward and backwards within the song, etc.

Example 2: Reading MIDI signals directly on the ChipKIT, and then generate sounds, including polyphony (e.g., at least 8 notes can be played simultaneously). The user can select different sounds, and instruments, adjust volume etc. The music is output to a real speaker.



Part I
Timers

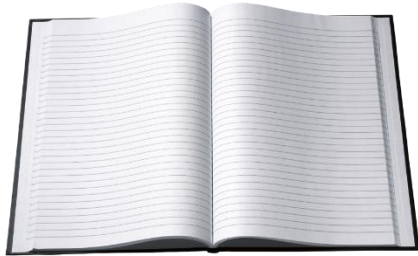
Part II
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

 **Part IV**
Mini
Project



Extended Abstract (1 page) - Contents



- **Title of the project**
The title must clearly state what the project is about.
- **Objective and Requirements**
These paragraphs should answer the question: What should your project do and which are the main requirements?
- **Solution**
The solution paragraph should state: How did you design and implement your solution?
- **Verification**
The verification paragraph should answer: How did you test or verify that your solution is correct?
- **Contributions**
This section should explain who did what of the two project members.
- **Reflections**
The last paragraph should include a few lines of reflections about the project.

Part I
Timers

Part II
Parallel and
Serial Communication

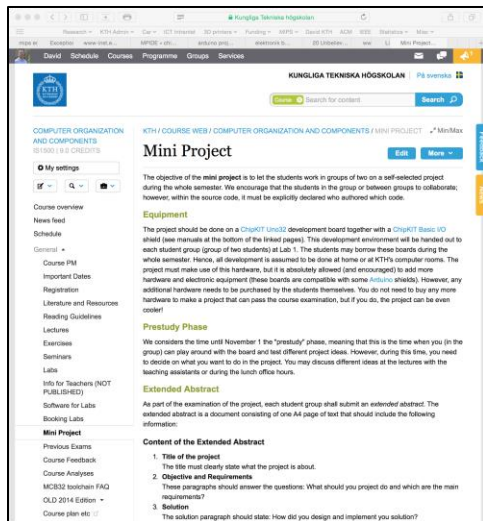
Part III
Exceptions
and Interrupts

 Part IV
Mini
Project



Extended Abstract - Submission

- **A DRAFT** of the extended abstract is submitted Canvas.
- The draft should include 1. the title, 2. the objective and requirements, 3. the solution (preliminary idea of how it will be done), and 4. verification (preliminary idea of how the verification is done).
- In the draft: **State if you aim for an advanced project!**
- **DRAFT Extended abstract for the project - Submission**
Deadline: Monday February 8, 17.00 CET.



Part I
Timers

Part II
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

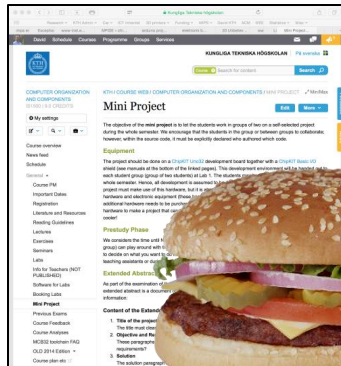
Part IV
Mini
Project



Project Expo



- **Everyone shows their projects!**
- **Nominations and Awards** for best projects.



Getting Help

- Post questions on Canvas discussion forum!
- Come to the lunch office hours.

Part I
Timers

Part II
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

 **Part IV**
Mini
Project

Yeah, you may leave soon



Part I
Timers

Part II
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

Part IV
Mini
Project



Reading Guidelines



Module 2 (I/O Systems)

H&H Chapters 8.5-8.7

For the labs, focus on 8.6.2-8.6.5
(GPIO, Timers, and Interrupts)

Also, look carefully at the solutions of exercise 2.

Reading Guidelines

See the course webpage
for more information.

Part I
Timers

Part II
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

Part IV
Mini
Project



Summary

Some key take away points:

- **A Timer** is used for keeping track of time and timing.
- **Exceptions** cause the processor to switch context and jump to an exception handling routine.
- **Interrupts** are special kinds of exceptions that are typically triggered by an external hardware event.
- **Good luck with the project!**



Thanks for listening!

Part I
Timers

Part II
Parallel and
Serial Communication

Part III
Exceptions
and Interrupts

Part IV
Mini
Project