Low-level C-programming and microprocessor architecture

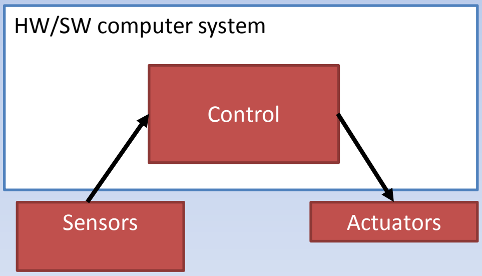
Welcome

- Polling system
- Interrupt system

AGSTU Arbete Genom STUdier Utbildning

Two different design implementations

- Polling system,
- Interrupt system.



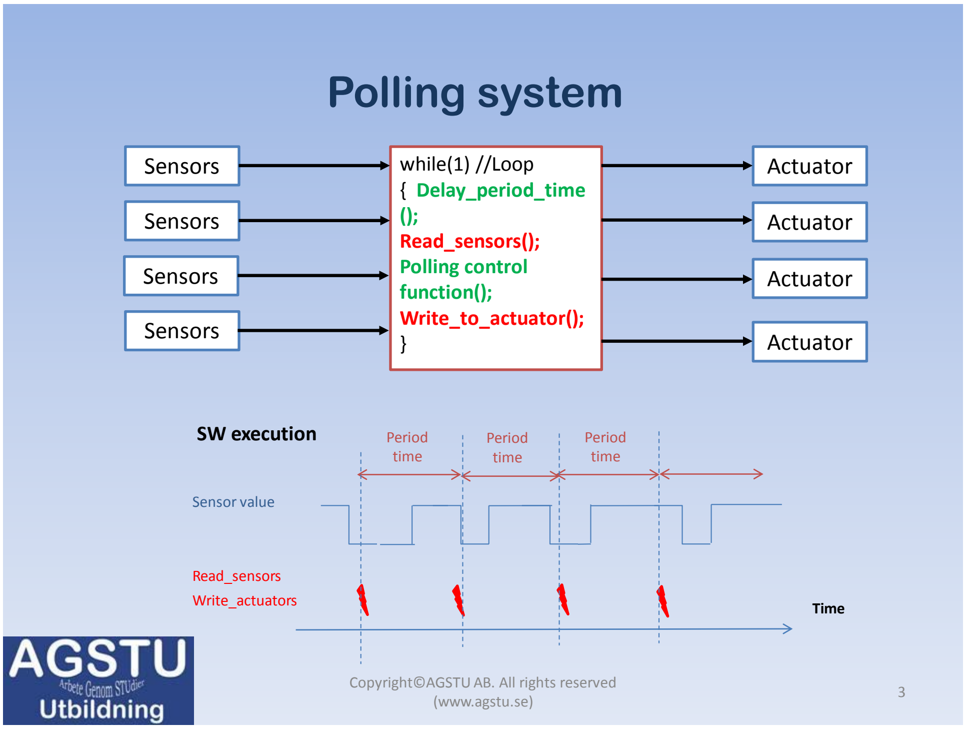
HW/SW computer system

Control

Sensors

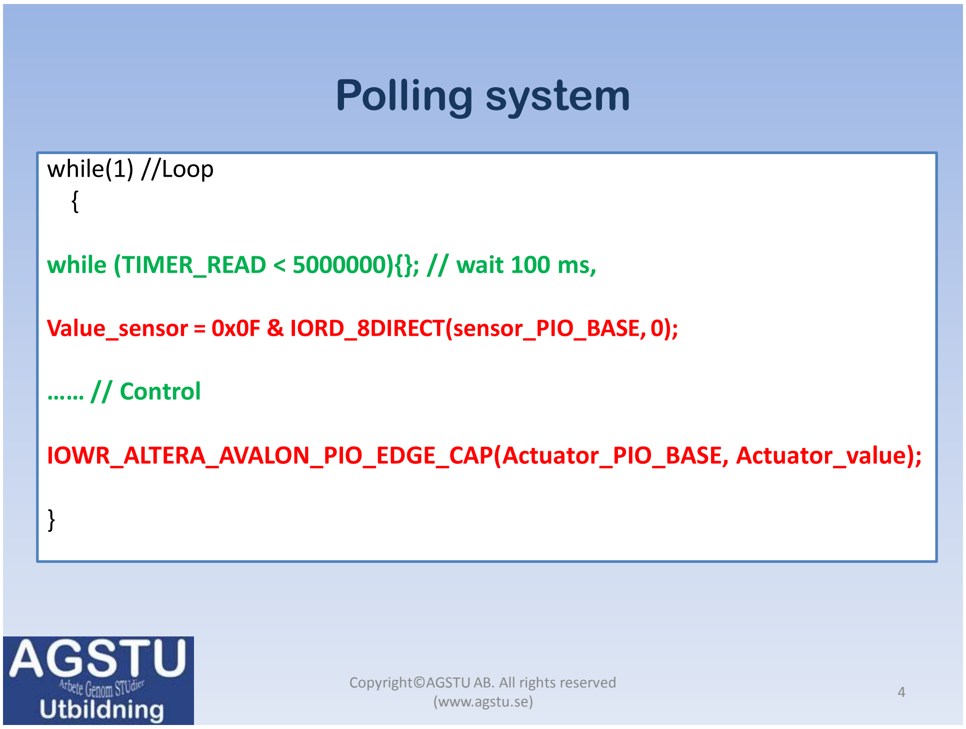
Actuators

AGSTU Arbete Genom STUdier Utbildning



Polling system

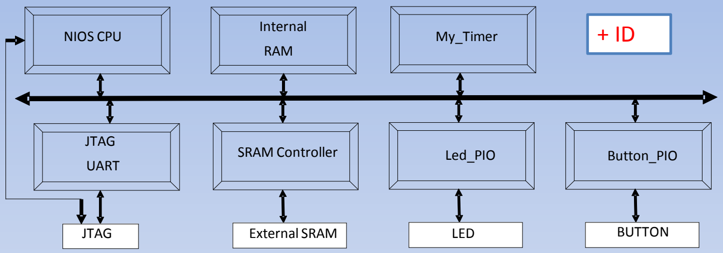
```
while(1) //Loop
{ Delay_period_time
();
Read_sensors();
Polling control
function();
Write_to_actuator();
}
```

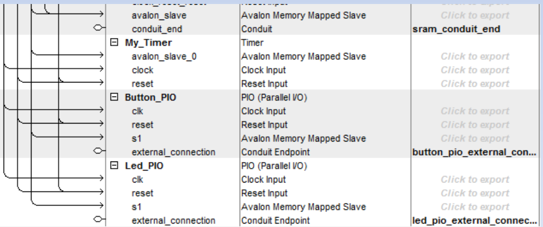


Polling system

```
while(1) //Loop
   {

while (TIMER_READ < 5000000){}; // wait 100 ms,

Value_sensor = 0x0F & IORD_8DIRECT(sensor_PIO_BASE, 0);

…… // Control

IOWR_ALTERA_AVALON_PIO_EDGE_CAP(Actuator_PIO_BASE, Actuator_value);

}
```

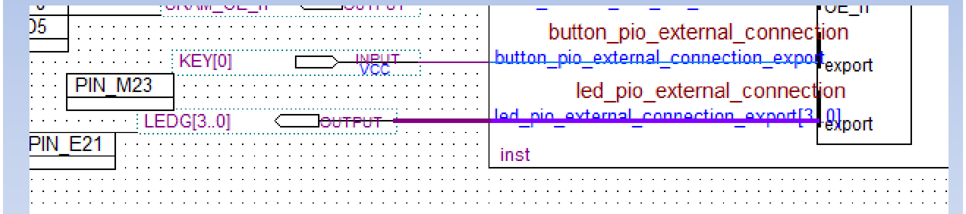
Hardware design (CASE 4)





avalon_slave	Avalon Memory Mapped Slave	Click to export
conduit_end	Conduit	**sram_conduit_end**
⊟ **My_Timer**	Timer	
avalon_slave_0	Avalon Memory Mapped Slave	Click to export
clock	Clock Input	Click to export
reset	Reset Input	Click to export
⊟ **Button_PIO**	PIO (Parallel I/O)	
clk	Clock Input	Click to export
reset	Reset Input	Click to export
s1	Avalon Memory Mapped Slave	Click to export
external_connection	Conduit Endpoint	**button_pio_external_con...**
⊟ **Led_PIO**	PIO (Parallel I/O)	
clk	Clock Input	Click to export
reset	Reset Input	Click to export
s1	Avalon Memory Mapped Slave	Click to export
external_connection	Conduit Endpoint	**led_pio_external_connec...**

I/O PIN connection

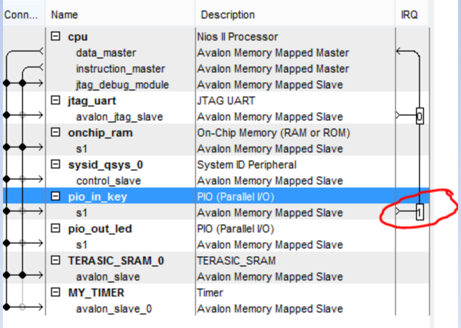

KEY[0]
INPUT
VCC
PIN_M23
LEDG[3..0]
OUTPUT
PIN_E21
button_pio_external_connection
button_pio_external_connection_export
export
led_pio_external_connection
led_pio_external_connection_export[3..0]
export
inst

Software design

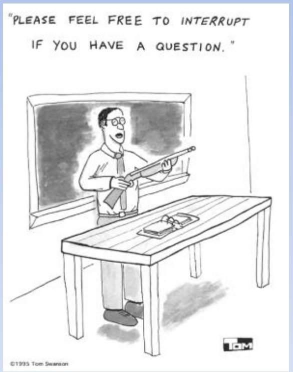
```
//*************** Controlling LEDs with a pushbutton *
#include <io.h>
#include <system.h>
#include <stdio.h>
#include <altera_avalon_timer_regs.h> // device driver for
my_timer


// The delay function is used to make a slow sample of the
button PIO
void delay_half_second(void)
{TIMER_RESET;
 TIMER_START;
while (TIMER_READ < 25000000){}; // wait half a second
}

int main(void)
{
int ledpio = 0;
int buttonpio = 0;
printf("We start");
while(1)
   {
    delay_half_second ();

    // Sample the button pio and mask the first bit
    buttonpio = 0x01 & IORD_8DIRECT(BUTTON_PIO_BASE, 0);
    // If the button is pressed increase the Led pio by one
    if(buttonpio == 0)
    {
      // Light the Leds
      IOWR_8DIRECT(LED_PIO_BASE , 0, ledpio);
      printf("ledpio = %d  ",ledpio);
      ledpio++;
      // Reset the ledpio when it reaches 16
      if(ledpio == 16)
      {
        ledpio = 0;
      }
// Wait until the button is released
      while(buttonpio == 0)
      {
        buttonpio = 0x01 & IORD_8DIRECT(BUTTON_PIO_BASE,
0);
      }
    }
   }
 return 0;
}
```

Two different design implementations

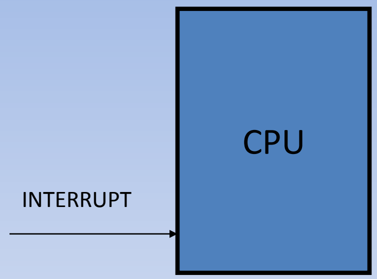
- Polling system,
- **Interrupt system**



Conn...	Name	Description	IRQ
	⊟ **cpu**	Nios II Processor	
	data_master	Avalon Memory Mapped Master	
	instruction_master	Avalon Memory Mapped Master	
	jtag_debug_module	Avalon Memory Mapped Slave	
	⊟ **jtag_uart**	JTAG UART	
	avalon_jtag_slave	Avalon Memory Mapped Slave	0
	⊟ **onchip_ram**	On-Chip Memory (RAM or ROM)	
	s1	Avalon Memory Mapped Slave	
	⊟ **sysid_qsys_0**	System ID Peripheral	
	control_slave	Avalon Memory Mapped Slave	
	⊟ **pio_in_key**	PIO (Parallel I/O)	
	s1	Avalon Memory Mapped Slave	1
	⊟ **pio_out_led**	PIO (Parallel I/O)	
	s1	Avalon Memory Mapped Slave	
	⊟ **TERASIC_SRAM_0**	TERASIC_SRAM	
	avalon_slave	Avalon Memory Mapped Slave	
	⊟ **MY_TIMER**	Timer	
	avalon_slave_0	Avalon Memory Mapped Slave	

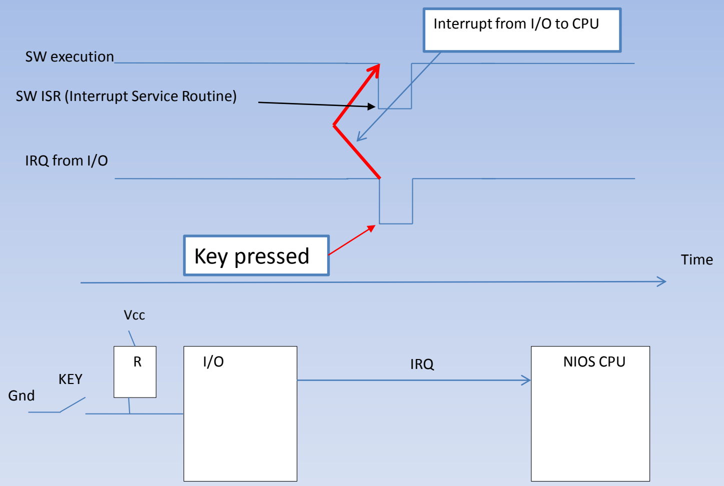


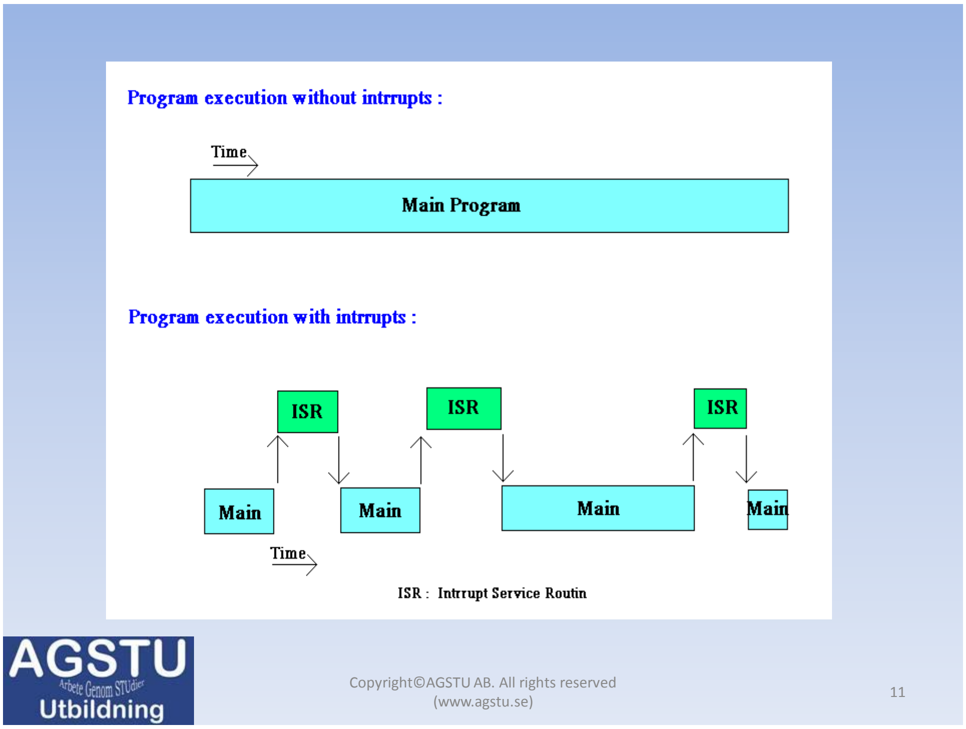
Interrupt



When the CPU has completed the execution of one instruction, it will sample the the signal on the INTERRUPT line. If it is active, the CPU will not execute the next instruction, and instead execute the **Interrupt Service Routine** (ISR).

Interrupt system





Program execution without intrrupts :

Time

Main Program

Program execution with intrrupts :

ISR

ISR

ISR

Main

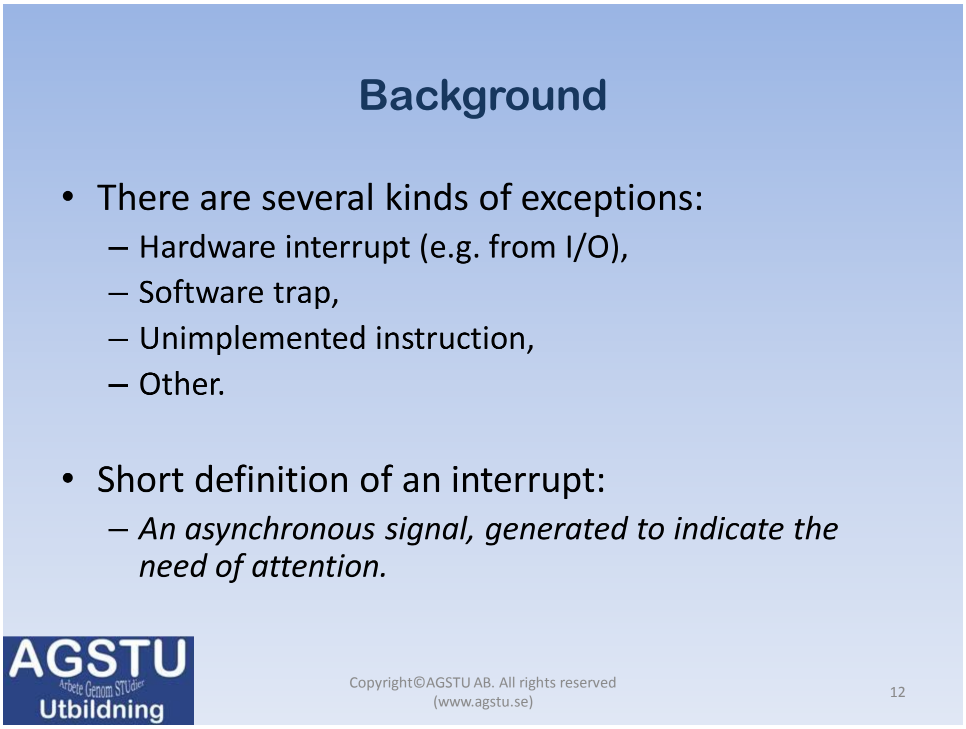
Main

Main

Main

Time

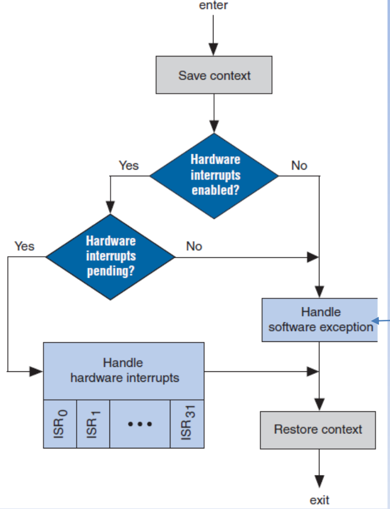
ISR : Intrrupt Service Routin



Background

- There are several kinds of exceptions:
  - Hardware interrupt (e.g. from I/O),
  - Software trap,
  - Unimplemented instruction,
  - Other.
- Short definition of an interrupt:
  - *An asynchronous signal, generated to indicate the need of attention.*

Inside CPU


enter
Save context
Yes
Hardware interrupts enabled?
No
Yes
Hardware interrupts pending?
No
Handle software exception
Handle hardware interrupts
ISR 0
ISR 1
• • •
ISR 31
Restore context
exit

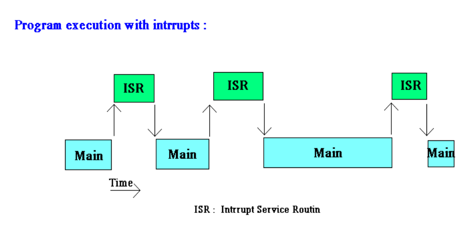
Software trap,
Unimplemented instruction,
…

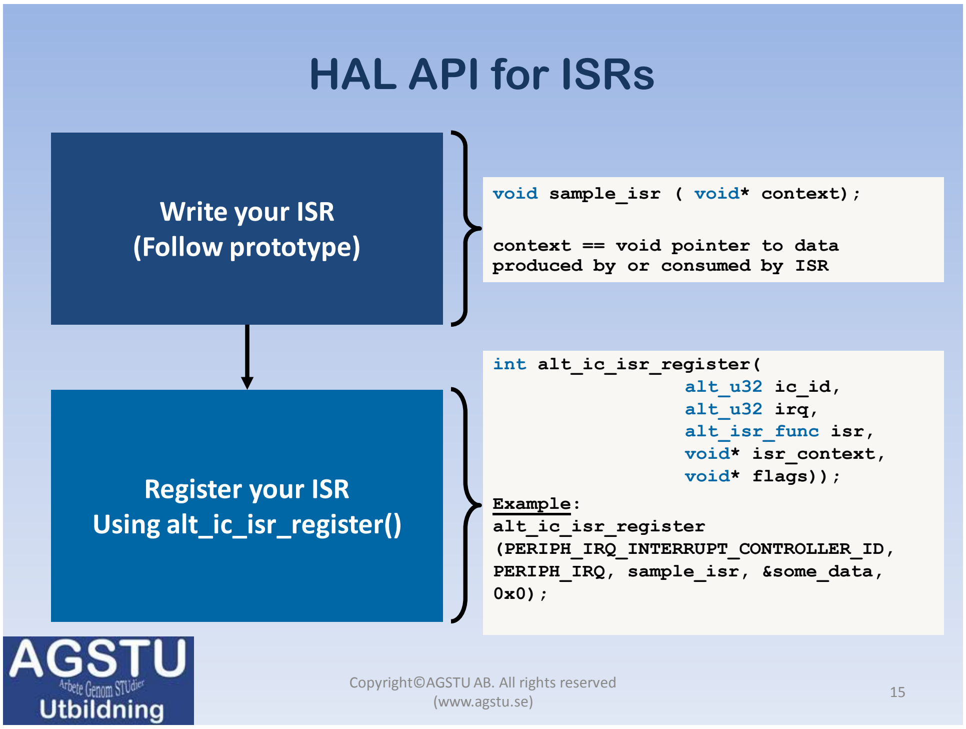
General SW structure

```
void MY_ISR(void *context, alt_u32 id)
{
  // Write ISR code here
  // Acknowledge interrupt
}

int main(void)
{
  if(alt_irq_register(COMPONENT_IRQ, NULL, my_isr))
    printf("Error registering irq handler\n");

while(1)
      ;
  }
  …
}
```

Some useful SW commands:

- alt_irq_disable_all()
- alt_irq_enable_all()


Program execution with intrrupts :
ISR
ISR
ISR
Main
Main
Main
Main
Time
ISR : Intrrupt Service Routin



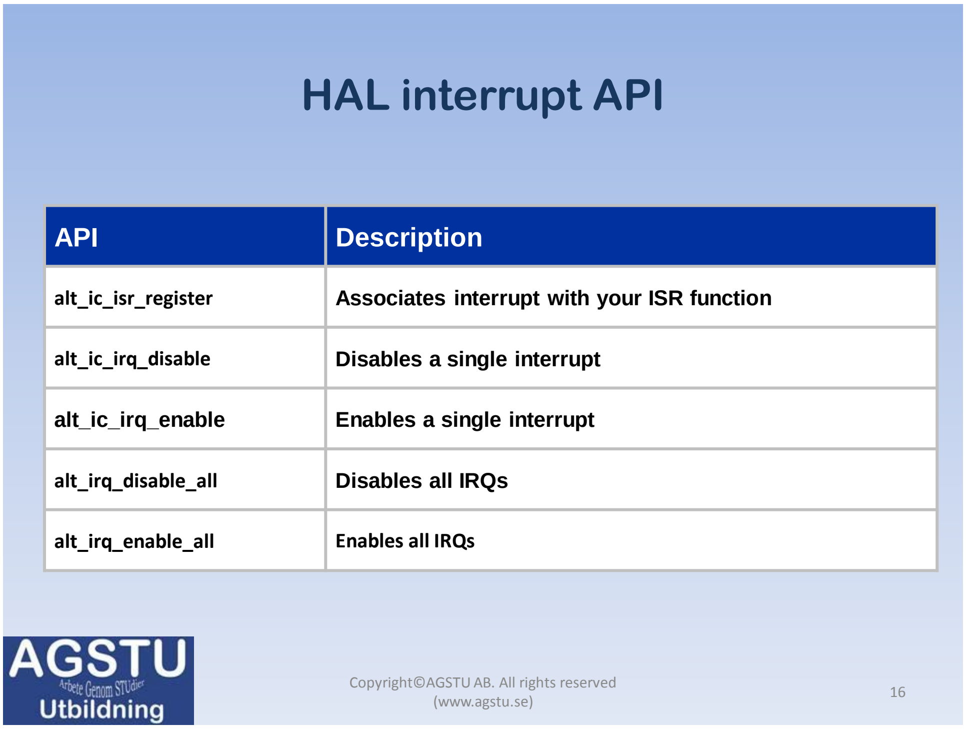
HAL API for ISRs

Write your ISR
(Follow prototype)

```
void sample_isr ( void* context);

context == void pointer to data
produced by or consumed by ISR
```

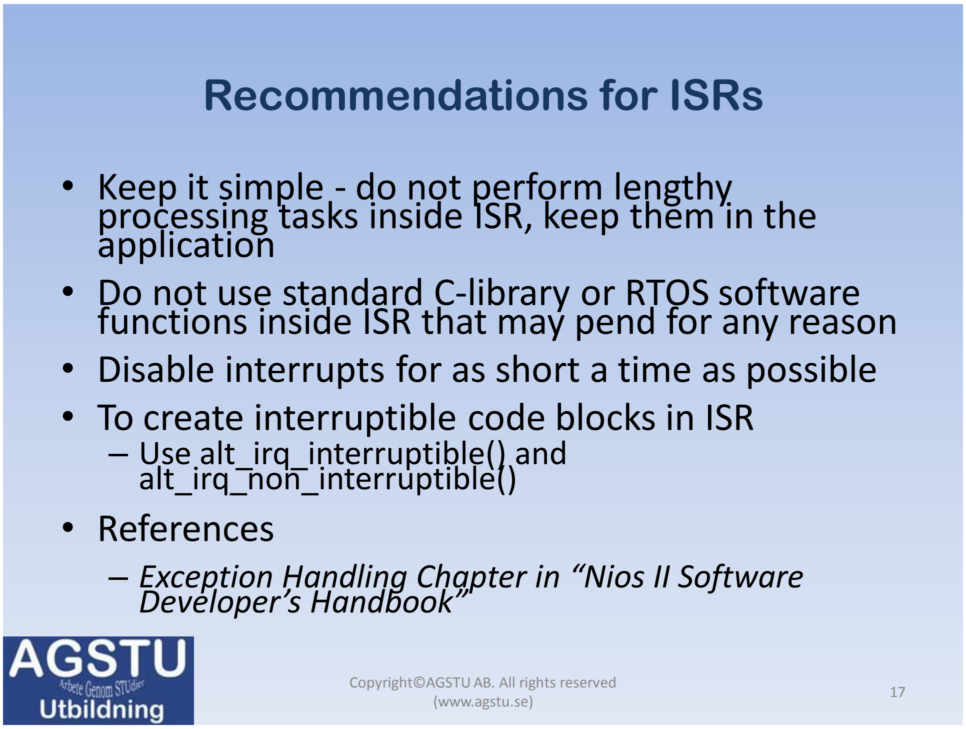
Register your ISR
Using alt_ic_isr_register()

```
int alt_ic_isr_register(
                 alt_u32 ic_id,
                 alt_u32 irq,
                 alt_isr_func isr,
                 void* isr_context,
                 void* flags));
Example:
alt_ic_isr_register
(PERIPH_IRQ_INTERRUPT_CONTROLLER_ID,
PERIPH_IRQ, sample_isr, &some_data,
0x0);
```



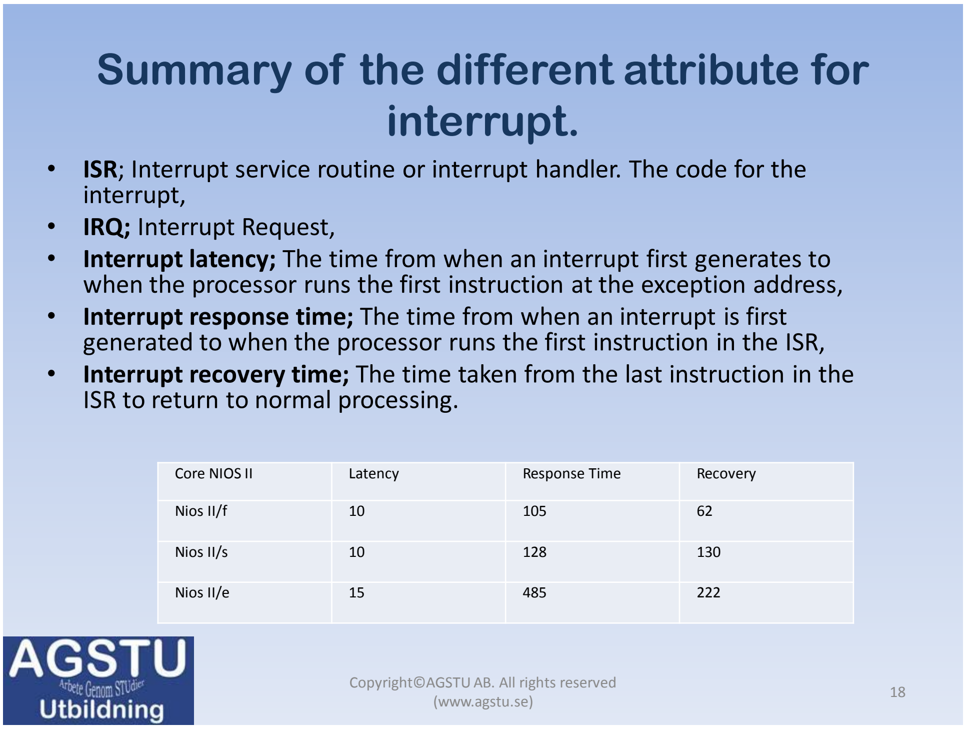
HAL interrupt API

API	Description
alt_ic_isr_register	Associates interrupt with your ISR function
alt_ic_irq_disable	Disables a single interrupt
alt_ic_irq_enable	Enables a single interrupt
alt_irq_disable_all	Disables all IRQs
alt_irq_enable_all	Enables all IRQs



Recommendations for ISRs

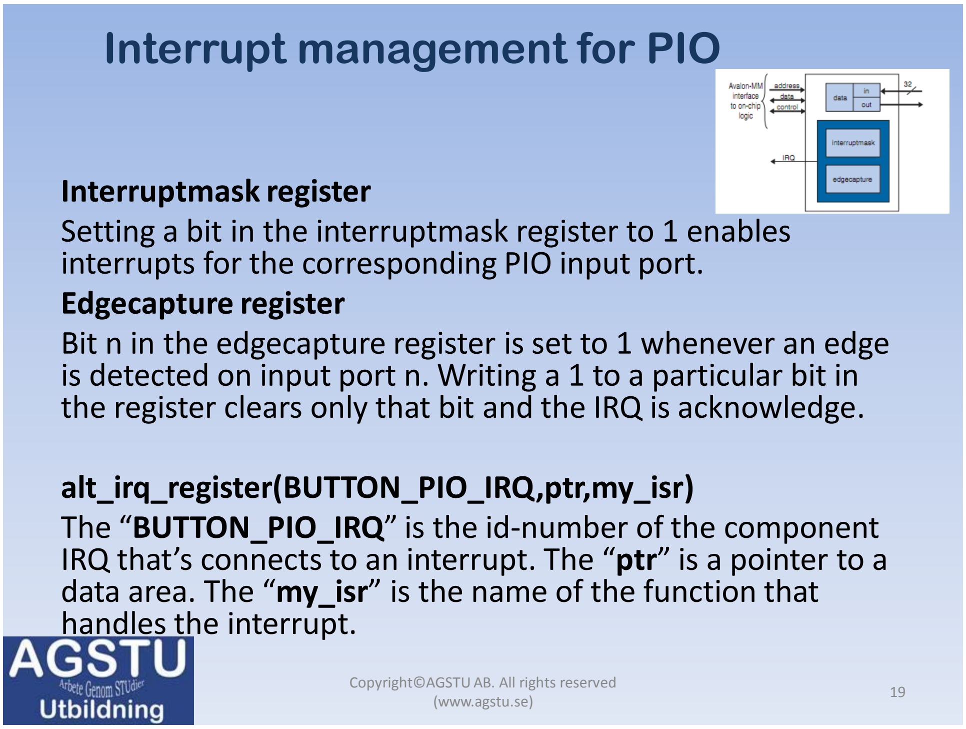
- Keep it simple - do not perform lengthy processing tasks inside ISR, keep them in the application
- Do not use standard C-library or RTOS software functions inside ISR that may pend for any reason
- Disable interrupts for as short a time as possible
- To create interruptible code blocks in ISR
  - Use alt_irq_interruptible() and alt_irq_non_interruptible()
- References
  - *Exception Handling Chapter in "Nios II Software Developer's Handbook"*



Summary of the different attribute for interrupt.

- **ISR**; Interrupt service routine or interrupt handler. The code for the interrupt,
- **IRQ;** Interrupt Request,
- **Interrupt latency;** The time from when an interrupt first generates to when the processor runs the first instruction at the exception address,
- **Interrupt response time;** The time from when an interrupt is first generated to when the processor runs the first instruction in the ISR,
- **Interrupt recovery time;** The time taken from the last instruction in the ISR to return to normal processing.

Core NIOS II	Latency	Response Time	Recovery
Nios II/f	10	105	62
Nios II/s	10	128	130
Nios II/e	15	485	222



Interrupt management for PIO

Avalon-MM interface to on-chip logic
address
data
control
data
in
out
32
interruptmask
edgecapture
IRQ

Interruptmask register

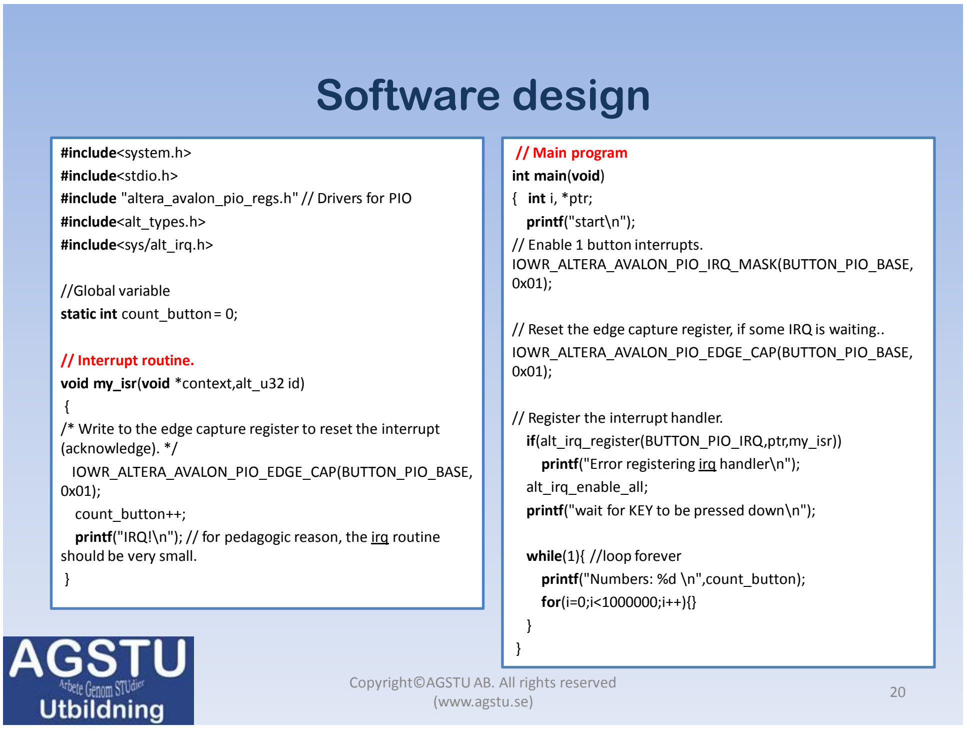
Setting a bit in the interruptmask register to 1 enables interrupts for the corresponding PIO input port.

Edgecapture register

Bit n in the edgecapture register is set to 1 whenever an edge is detected on input port n. Writing a 1 to a particular bit in the register clears only that bit and the IRQ is acknowledge.

alt_irq_register(BUTTON_PIO_IRQ,ptr,my_isr)

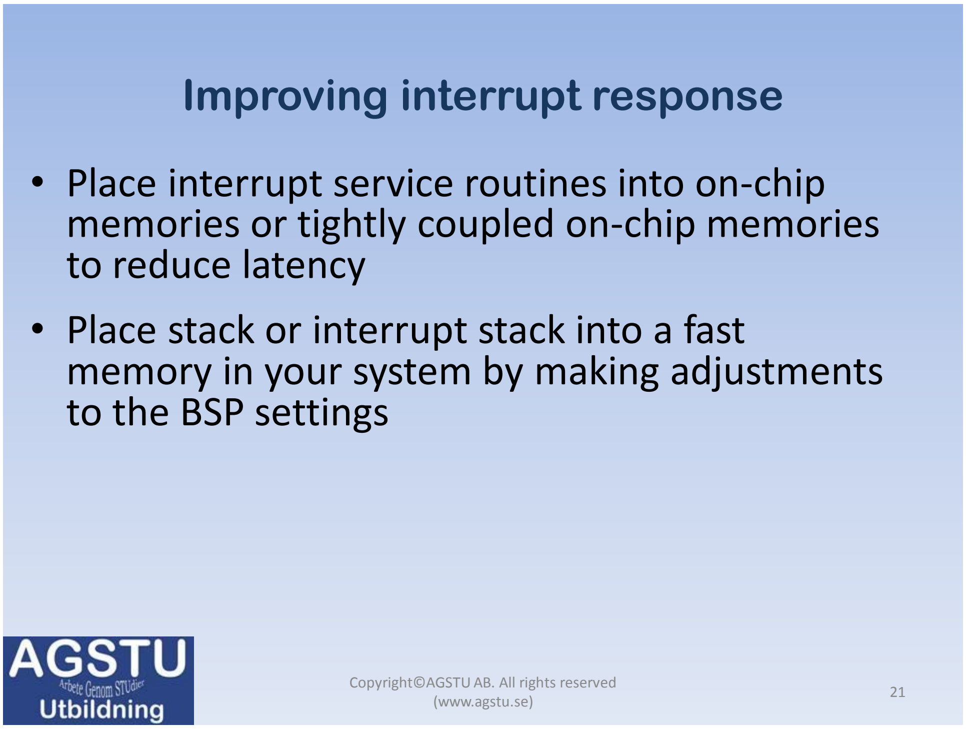
The “**BUTTON_PIO_IRQ**” is the id-number of the component IRQ that’s connects to an interrupt. The “**ptr**” is a pointer to a data area. The “**my_isr**” is the name of the function that handles the interrupt.



Software design

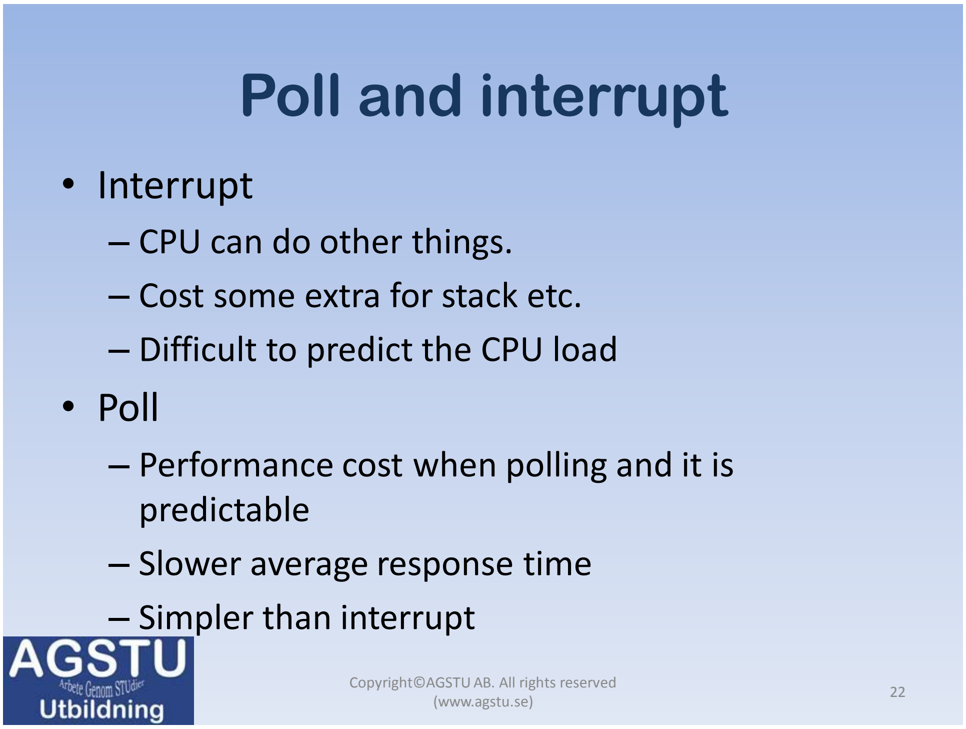
```
#include<system.h>
#include<stdio.h>
#include "altera_avalon_pio_regs.h" // Drivers for PIO
#include<alt_types.h>
#include<sys/alt_irq.h>

//Global variable
static int count_button = 0;

// Interrupt routine.
void my_isr(void *context,alt_u32 id)
 {
/* Write to the edge capture register to reset the interrupt
(acknowledge). */
  IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE,
0x01);
   count_button++;
   printf("IRQ!\n"); // for pedagogic reason, the irq routine
should be very small.
 }
```

```
 // Main program
int main(void)
{  int i, *ptr;
   printf("start\n");
// Enable 1 button interrupts.
IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTON_PIO_BASE,
0x01);

// Reset the edge capture register, if some IRQ is waiting..
IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE,
0x01);

// Register the interrupt handler.
   if(alt_irq_register(BUTTON_PIO_IRQ,ptr,my_isr))
      printf("Error registering irq handler\n");
   alt_irq_enable_all;
   printf("wait for KEY to be pressed down\n");

   while(1){ //loop forever
      printf("Numbers: %d \n",count_button);
      for(i=0;i<1000000;i++){}
   }
 }
```



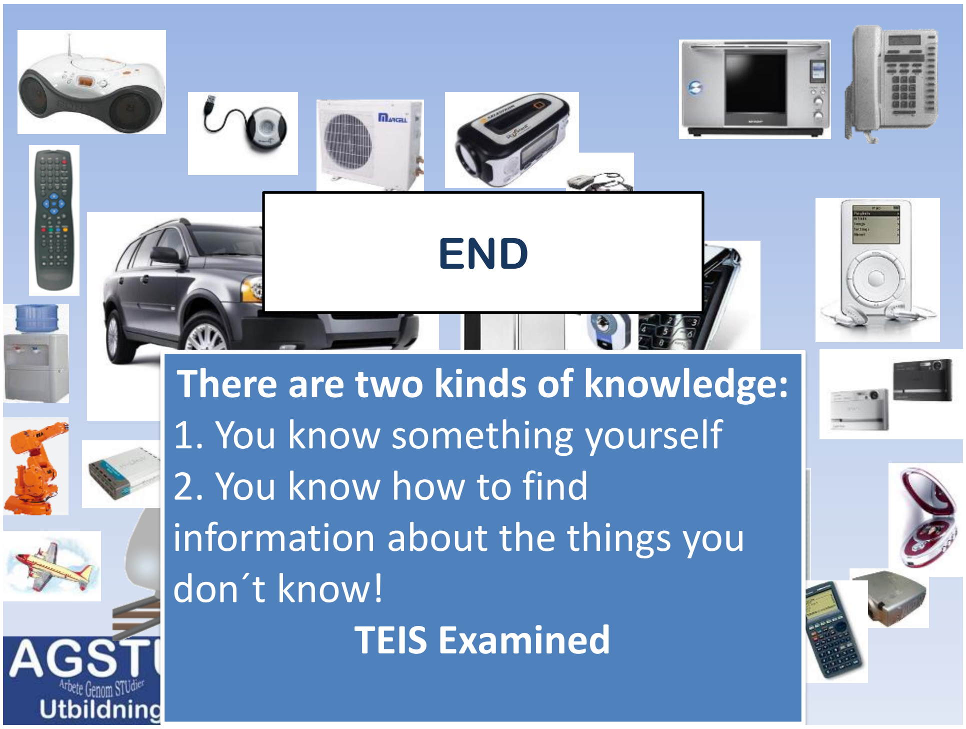
Improving interrupt response

- Place interrupt service routines into on-chip memories or tightly coupled on-chip memories to reduce latency
- Place stack or interrupt stack into a fast memory in your system by making adjustments to the BSP settings



Poll and interrupt

- Interrupt
  - CPU can do other things.
  - Cost some extra for stack etc.
  - Difficult to predict the CPU load
- Poll
  - Performance cost when polling and it is predictable
  - Slower average response time
  - Simpler than interrupt



END

There are two kinds of knowledge:
1. You know something yourself
2. You know how to find information about the things you don´t know!

TEIS Examined

All rights reserved and Disclaim

- **All rights reserved.** No part of this document (PPT, Doc, film etc.) may be reproduced, in any form or by any means, without permission in writing from the publisher. Unless otherwise specified, all information (including software, designs and files) provided are copyrighted by AGSTU AB (www.agstu.se)
- AGSTU AB, Dragverksgatan 138, SE-724 74 Vasteras, Sweden
- **Disclaim**
  All the information (including hardware, software, designs, text and files) are provided "as is" and without any warranties expressed or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose. In no event should the author be liable for any damages whatsoever (including without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising out of the use or inability to use information (including text, software, designs and files) provided in this document.

24