|  |
| --- |
| AGSTU |
| IRQ |
| NIOS IRQ hantering |
|  |
|  |
| **Ashraf Tumah** |
| **21-09-21** |

Sammanfattning: Rapporten beskriver grundläggande hur NIOS bearbetar ett avbrott. Vikten läggs på att beskriva hur ett hårdvaruavbrott bearbetas genom interrupt service routine(ISR) med hjälp av hardware abstraction layer(HAL).

INNEHÅLLSFÖRTECKNING

[1 INLEDNING 3](#_Toc22468)

[2 NIOS IRQ 4](#_Toc23947)

[3 NIOS Interrupt Service Routine 6](#_Toc17550)

[3.1 HAL ISR 6](#_Toc7663)

[4 PIO och IRQ 7](#_Toc16445)

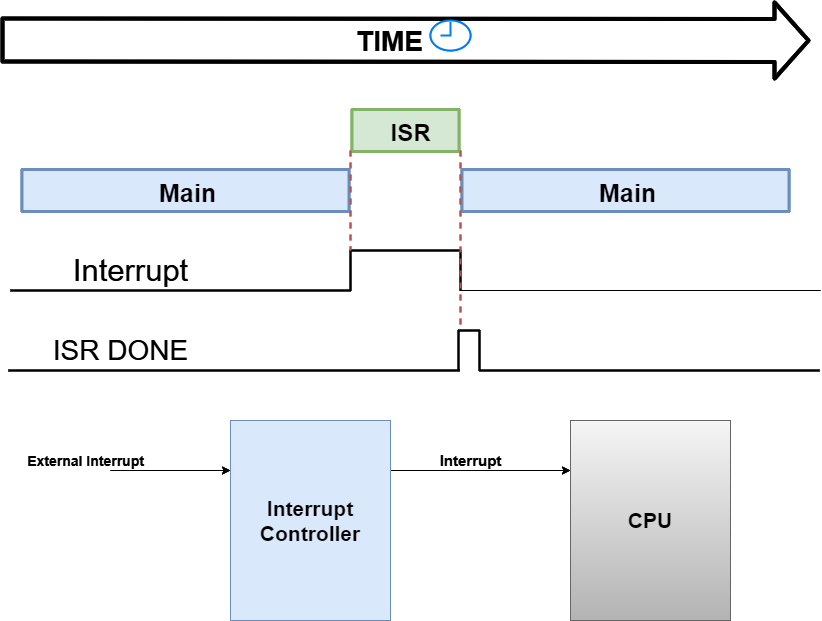
[5 Referenser 9](#_Toc242)

# INLEDNING

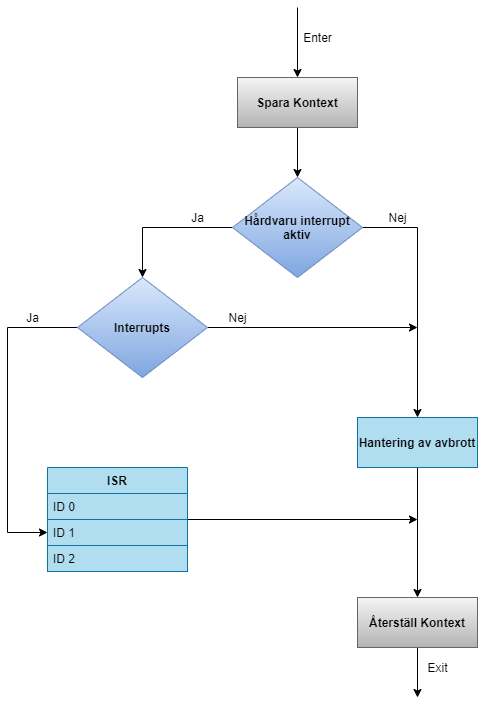
IRQs används för att program ska kunna svara på oväntade händelser vilka kan dyka upp närhelst med hjälp av exception handlers. En IRQ signal fångas upp av CPUn vilket genererar ett avbrott i main programmet. Eftersom det är ett uppehåll i main programmet bör en ISR vara kort och simpel. I en ISR görs en acknowledge vilket sänker interrupt flaggan och gör det möjligt för CPUn att återgå till main programmet.

# NIOS IRQ

Interrupts vilka kan vara externa eller hårdvarudrivna PIO signaler ändrar det normala exekveringsflödet.   
När en interrupt flaggas flyttar NIOS CPU den nuvarande context som programräknare, registers etc till stack för att hantera interrupt rutinen. CPUn exekverar ISR och rensar flaggor för att återställa den kontext som lagrats i stack. Figuren nedan beskriver interrupt signaler och figur 2 visar exekveringsflödet för en interrupthantering.



Figur 1. Beskrivning av en avbrottssignal.



Figur 2. Flödesdiagram - interrupthantering

# NIOS Interrupt Service Routine

När ett BSP(bard support package) projekt skapas, inkluderar verktyget alla de nödvändiga drivrutiner för HAL ISR som behövs.   
HAL tillhandahåller *legacy* samt *enhanced* API hantering för avbrott:

* **Enhanced**- Kompatibel med både externa och interna avbrottskontroller samt har ett förbättrad gränssnitt för att skriva, registrera och hantera ISR.
* **Legacy**- Stödjer endast Interna avbrotts kontrollers.

## HAL ISR

Tabell 1 beskriver API för avbrottshanteraren.

Tabell 1. IRQ Funktionsanrop

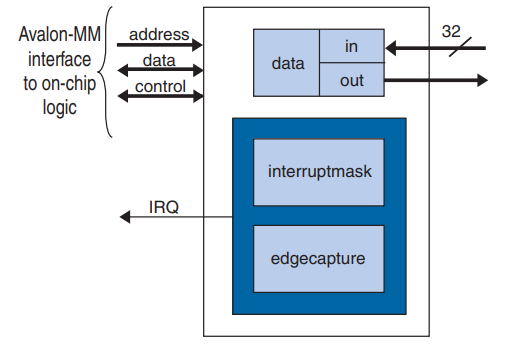
|  |  |
| --- | --- |
| **Funktion** | **Beskrivning** |
| **Support för Enhanced API** | |
| **alt\_ic\_isr\_register()** | Användas för att registrera ISR. Om funktionen registreras korrekt aktivers processorns IRQ |
| **alt\_ic\_irq\_enable()** | Anropas för att aktivera processorns globala IRQ. |
| **alt\_ic\_irq\_disable()** | Anropas för att avaktivera processorns globala IRQ. |
| **alt\_ic\_irq\_enabled()** | Aktiverar en specifik interrupt från processorns IRQ portar. |
| **alt\_ic\_irq\_disabled()** | Avaktiverar en specifik interrupt från processorns IRQ portar. |
| **Support för både Legacy och Enhanced API** | |
| **alt\_irq\_enabled()** | Aktiverar en IRQ |
| **alt\_irq\_enable\_all()** | Aktiverar all IRQ |
| **alt\_irq\_disable\_all()** | Avaktiverar all IRQ |

# PIO och IRQ

Platform Designer gör det möjligt att integrera med PIO vilka inställda som input signal kan generera en IRQ. PIO IRQ kan ha olika avbrotts- och flank uppfattningar. I Platform Designer kan en PIO konfigureras till att genera en IRQ signal efter ett specifikt villkor. Villkoren för IRQ signalen är följande:

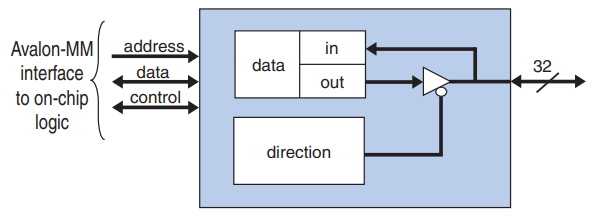
* **Level:** PIO-kärnan känner av en aktiv ”hög” signal för att generera en IRQ.
* **Edge:** PIO-kärnan upptäcker flanker med avseende på inställningar i Platform Designer genererar en IRQ om det är en stigande eller fallande flank.  
  Typer av flank avkänning för PIO *edge capture* konfigurering: *Rising Edge, Falling Edge* eller *Either Edge*.

Figuren 3 visar ett blockdiagram av en PIO inställd för in och ut signaler med support för IRQ.



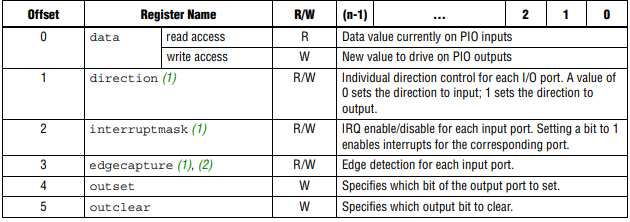
Figur 3. PIO Blockdiagram med in,ut signaler och support för IRQ

Figur 4 visar ett blockdiagram för en PIO med dubbelriktad signal utan support för IRQ.



Figur 4. PIO Blockdiagram med dubbelriktad signal utan support för IRQ

Generellt innehåller PIO som genererar IRQs olika styrregister vilka beskrivs i figuren nedan.



Figur 5. PIO register map

Varje register är 32 bitar långt. Dock kan vissa register sakna tillgång till en del bitar. De fyra huvudregistren har följande syfte:

* **Data:** lagrar bit data samt håller kommunikation mellan PIO och NIOS CPU.
* **Direction:** Definierar in/ut riktning för databitar när en signal generas.
* **Interruptmask:** Aktiverar interrupts kommande från input signaler ansluten till PIO.
* **Edgecapture:** Indikerar när en ändring av logik uppfångas på input signalen ansluten till PIO

# Referenser

1. 21-09-21  
   <https://www.intel.co.jp/content/dam/altera-www/global/ja_JP/pdfs/literature/hb/nios2/n2cpu_nii51007.pdf>
2. 21-09-21  
   <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/n2sw_nii52006.pdf>
3. 21-09-15  
   <https://www.youtube.com/watch?v=HF7Low_sUig>