|  |
| --- |
| AGSTU |
| PIC |
| Programmable Interrupt Controller |
|  |
|  |
| **Ashraf Tumah** |
| **21-09-17** |

Sammanfattning: Rapporten beskriver och ger en analys av en konstruktion. Konstruktionen som är en programmerbar avbrotts frekvens ska tillsammans med rapporten ge grundläggande förståelse över hur IRQs och ISR fungerar. Rapporten kan även användas som underlag för att konstruera och utveckla vidare en programmerbar avbrottsfrekvens.

INNEHÅLLSFÖRTECKNING

[1 INLEDNING 5](#_Toc32194)

[2 KRAVSPECIFIKATION 6](#_Toc20968)

[3 Testprotokoll 7](#_Toc23898)

[4 Konstruktionsbeskrivning 9](#_Toc30533)

[4.1 Interrupt controller IP: 10](#_Toc14570)

[4.1.1 Interrupt\_counter 10](#_Toc14649)

[4.1.2 Interrupt\_sender 11](#_Toc20036)

[4.2 ECS 11](#_Toc23598)

[4.3 Platform designer 11](#_Toc24189)

[4.3.1 Interrupt\_controller\_ip 12](#_Toc22996)

[5 Hardware 14](#_Toc14926)

[5.1 Delsystem - Interrupt Ip 15](#_Toc28156)

[5.1.1 Interrupt\_counter 15](#_Toc12295)

[5.1.2 Interrupt\_sender 15](#_Toc30294)

[6 Software 17](#_Toc21255)

[6.1 Exempel Kod 17](#_Toc28490)

[6.1.1 Kod beskrivning 17](#_Toc32755)

[6.2 Drivrutiner 18](#_Toc26701)

[6.2.1 Interrupt\_hw\_ack 18](#_Toc16426)

[6.2.2 Ir\_timer 19](#_Toc9732)

[6.2.3 Print\_ir\_data 19](#_Toc2963)

[7 Verifiering 20](#_Toc31259)

[7.1 Testfall 0-2 20](#_Toc3861)

[7.2 Testfall 3 21](#_Toc27130)

[7.3 Testfall 4 21](#_Toc13442)

[7.4 Testfall 5 22](#_Toc2352)

[7.5 Testfall 6 22](#_Toc23684)

[8 Validering 23](#_Toc25564)

[8.1 Testfall 0 23](#_Toc10632)

[8.2 Testfall 1 24](#_Toc26203)

[8.3 Testfall 2 24](#_Toc25489)

[8.4 Testfall 3 24](#_Toc31056)

[8.5 Testfall 4 25](#_Toc22173)

[9 Referenser 26](#_Toc20233)

# INLEDNING

IRQs används för att program ska kunna svara på oväntade händelser vilka kan dyka upp närhelst. Detta sker med hjälp av exception handlers. Ett exempel på exception handler kan vara en musnedtryckning på en standard PC vilket aktiverar en avbrottshantering för att registrera den nedtryckta signalen.  
En IRQ signal fångas upp av CPUn vilket genererar ett avbrott i main programmet. ISR exekveras för att sedan låta CPUn återgå till main programmet. Eftersom det är ett uppehåll i main programmet bör en ISR vara kort och simpel.   
Interrupts ger effektivare användning av CPU tiden. I stället för att kontrollera en PIO anslutnings status emellanåt(polling) kan PIO-anslutningen rapportera att något hänt genom att “flagga”. Detta berättar att PIO är redo för data. Huvudprogrammet svarar tillbaka och kör ISR alternativt läser/skriver data.

# KRAVSPECIFIKATION

Tabell 1 visar en kravspecifikation för projektet.

Tabell 1. Kravspecifikation

|  |  |  |
| --- | --- | --- |
| **Krav** | **Beskrivning** | **Utfört** |
|  | | **Ja/Nej** |
| **Leverans Ett** | | |
| **1.1** | Bygg en enkel pulsgivarkomponent som ger en programmerbar avbrotts frekvens inom valfritt intervall. Efter HW reset ska inga avbrott ges. En teknisk rapport ska levereras. | **Ja** |
| **1.2** | Testkör även att systemet ska fungera efter en HW-reset. | **Ja** |

# Testprotokoll

Tabell 2 och 3 visar testprotokoll för verifiering och validering av konstruktionen.

Tabell 2. Testprotokoll Verifiering

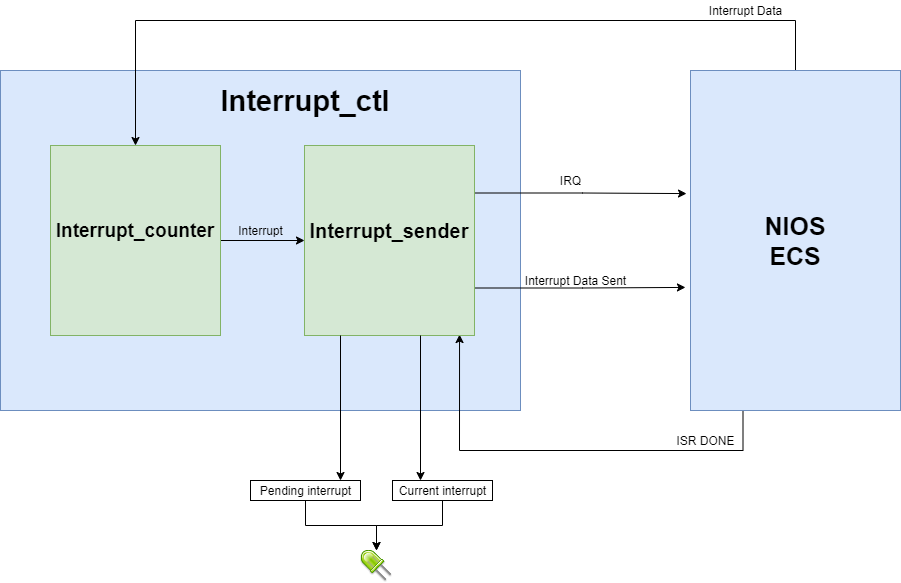
|  |  |  |  |
| --- | --- | --- | --- |
| **Testfall** | **Beskrivning** | **Acceptans** | **Godkänt (J/N)** |
| **0** | Reset Aktiv Låg | System reset |  |
| **1** | Reset Aktiv Hög | System start |  |
| **2** | Enable Aktiv | Räknare startar |  |
| **3** | Interrupt0 räknare= data0 | Interrupt =’1’ intr\_presenr(0)=’1’ intr\_queue(0)=’1’ |  |
| **4** | Interrupt1 räknare= data1 | Interrupt =’1’ intr\_presenr(1)=’1’ intr\_queue(1)=’1’ |  |
| **5** | Interrupt2 räknare= data2 | Interrupt =’1’ intr\_presenr(2)=’1’ intr\_queue(2)=’1’ |  |
| **6** | Interruptbitar aktiv simultan (Kontroll av prio) | Intr\_present=”001”  Intr\_queue “111” |  |

Tabell 3. Testprotokoll Validering

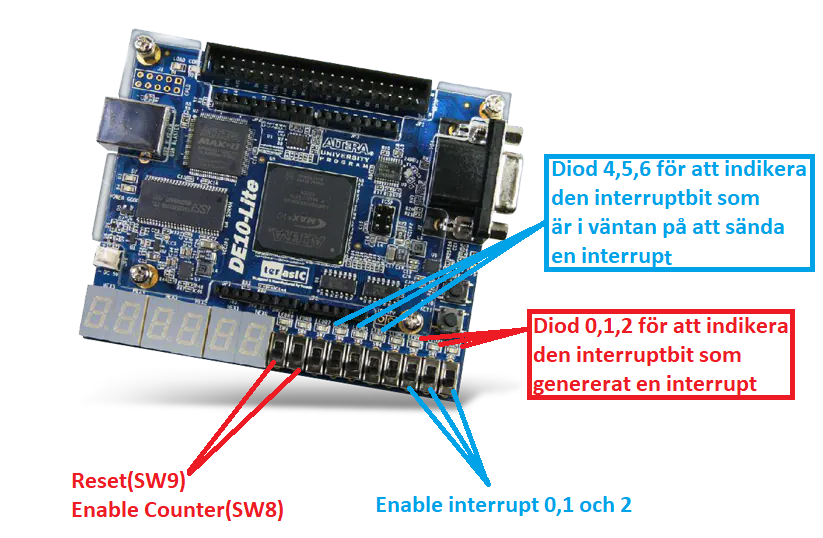
|  |  |  |  |
| --- | --- | --- | --- |
| **Testfall** | **Beskrivning** | **Acceptans** | **Godkänt (J/N)** |
| **0** | Interruptbit 0 flaggar för interrupt | LEDR(0)=’1’ LEDR(4)=’1’ NIOS ISR Exekverar |  |
| **1** | Interruptbit 1 flaggar för interrupt | LEDR(1)=’1’ LEDR(5)=’1’ NIOS ISR Exekverar |  |
| **2** | Interruptbit 2 flaggar för interrupt | LEDR(2)=’1’ LEDR(6)=’1’ NIOS ISR Exekverar |  |
| **3** | Interruptbitar flaggar simultan | LEDR(2 downto 0)=”001” LEDR(6 downto 4)=”111” NIOS ISR Exekverar |  |
| **4** | Switch(2 downto 0)=”000” | LEDR(6 downto 0)=’0’ Ingen ISR |  |

# Konstruktionsbeskrivning

Systemet består av två komponenter. Ett inbäddat datorsystem och en interrupt kontroller.  
Det inbäddade datorsystemet skriver data till hårdvarukomponenten. Interrupt kontrollern sänder en IRQ till det inbäddade datorsystemet. Kontrollern indikerar även vilken interrupt som står i kö för att generera en irq samt vilken interrupt som sänder en irq. Beskrivs senare i rapporten. Figuren nedan beskriver en del anslutningar inom systemet och figur 2 visar utvecklingskortet DE10-Lite med beskrivningar.



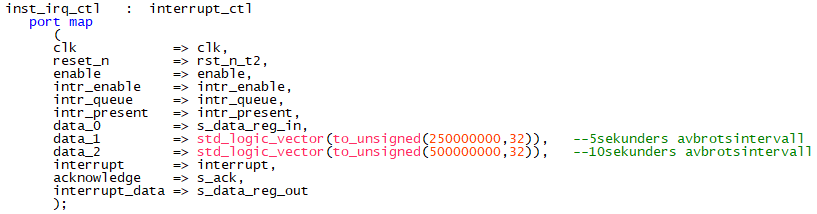
Figur 1. Konstruktionsbeskrivning HW



Figur 2. DE10-Lite utvecklingskort

## Interrupt controller IP:

Interrupt controller sänder avbrottsfrekvenser till det inbäddade datorsystemet med hjälp av en räknar komponent och en komponent som genererar avbrottsfrekvensen.  
NIOS CPU skriver till interrupt bit 0 för att generera en avbrottsintervall efter angiven data. Interrupt bit1 samt interrupt bit 2 har förinställda värden genom Quartus där bit 1 genererar en avbrottsfrekvens var 5te sekund och bit 2 genererar en avbrottsfrekvens var 10de sekund. Se figur nedan.



Figur 3. Quartus Instansiering

### Interrupt\_counter

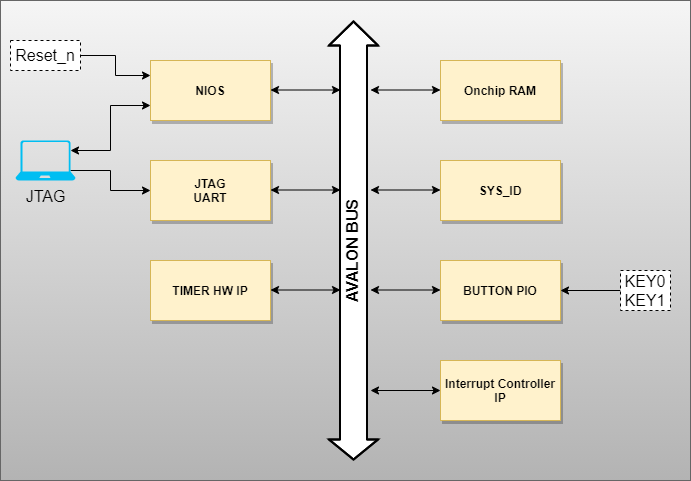
Räknare som sänder en interrupt signal till nästa komponent då räknarvärdet är den samma som angiven data. Komponenten skickar upp till 3 signaler för varje interrupt bit till nästa komponent.

### Interrupt\_sender

De mottagna signalerna sorteras och genererar en IRQ signal fram till att en acknowledge signal når komponenten.  
Den interrupt som är aktiv visas på LEDR(2 downto 0) som ett “id” för respektive interrupt bit.

## ECS

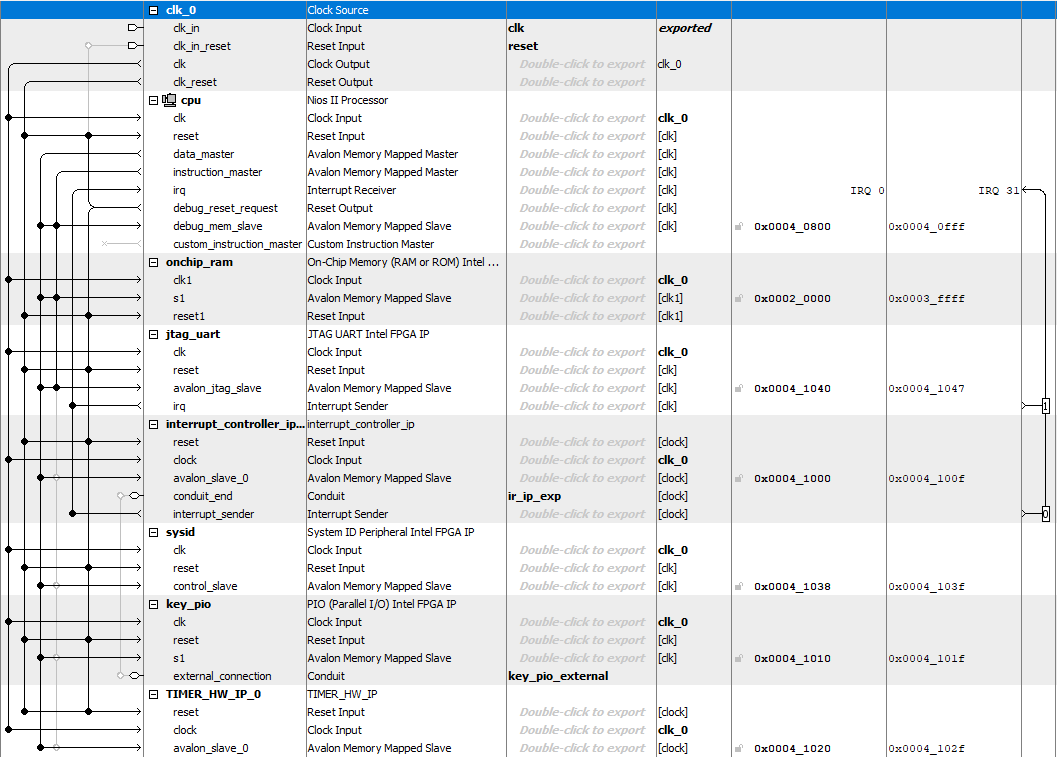
Det inbäddade datorsystemet styr verken mellan interrupt kontrollen och datorsystemet. Figur 4 visar komponenter anslutna mot Avalon busen.



Figur 4. Avalon Busen

## Platform designer

Figur 5 visar de interna anslutningar i Platform Designer.

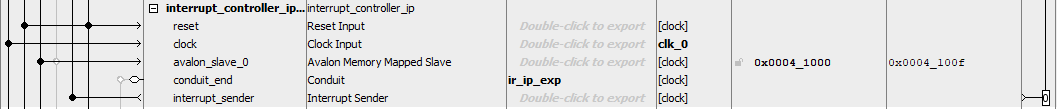


Figur 5. Platform Designer

Nedan beskrivs interrupt kontrollen i Platform Designer.

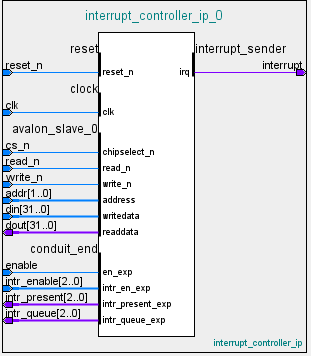
### Interrupt\_controller\_ip

Figur 6 visar IP-komponenten i Platform Designer



Figur 6. Interrupt ip i Platform Designer

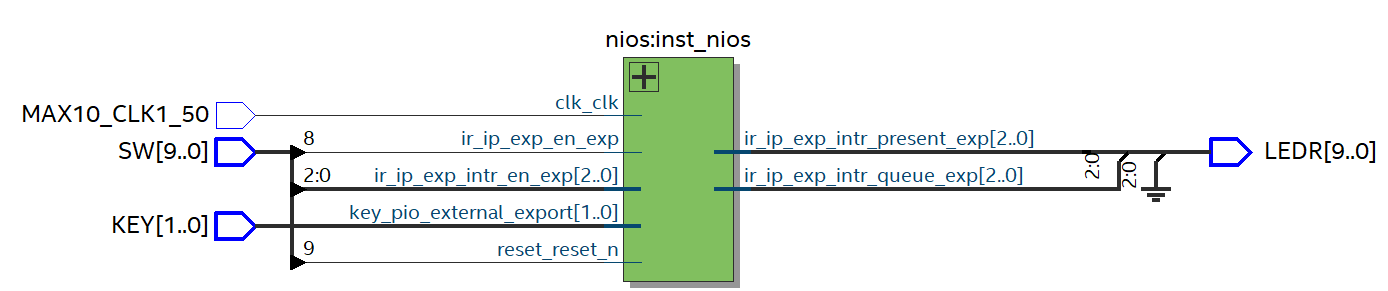
Interrupt\_controller\_ip genererar en interrupt signal som är ansluten direkt till NIOS CPU irq. Interrupt id på signalen är 0 vilket betyder att IRQs från interrupt IP har högst prioritering. För att medla IP-komponenten att ISR är utförd behövs en acknowledge vilket återställer interrupt flaggan.  
Figuren nedan visar komponentens blockdiagram i Platform Designer.



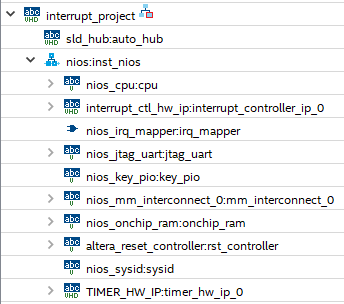
Figur 7. Blockdiagram för IP komponenten i Platform Designer

# Hardware

Figur 8 visar systemet i rtl-nivå och figur 9 visar komponenthierarkin.

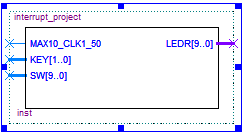


Figur 8. RTL-Topnivå



Figur 9. Komponenthierarki

Figuren nedan visar systemets in och ut signaler.



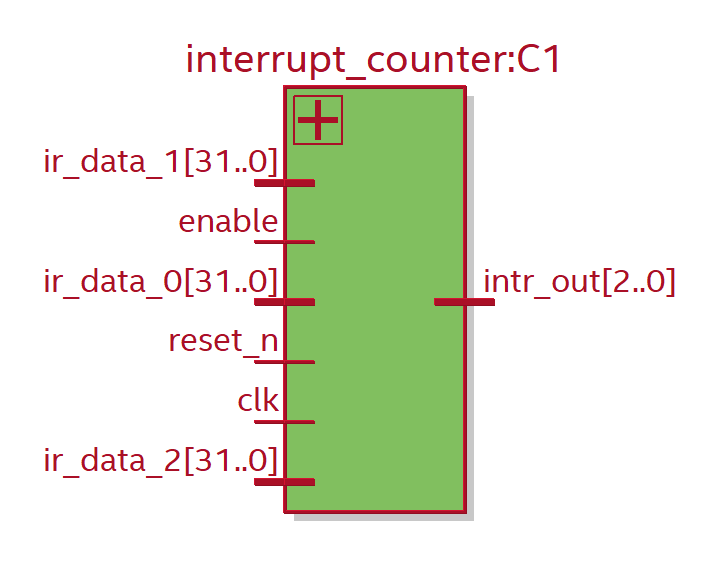
Figur 10. Symbolblock, interrupt\_project

## Delsystem - Interrupt Ip

### Interrupt\_counter

Komponentnamn: interrupt\_counter  
Instansnamn: C1

Komponenten är en räknare som sänder en 3bitars avbrottspuls då räknaren för respektive interrupt bit får samma värde som angiven in data. Figur 11 visar komponenten och tabell 4 ger en signalbeskrivning.



Figur 11. interrupt\_counter

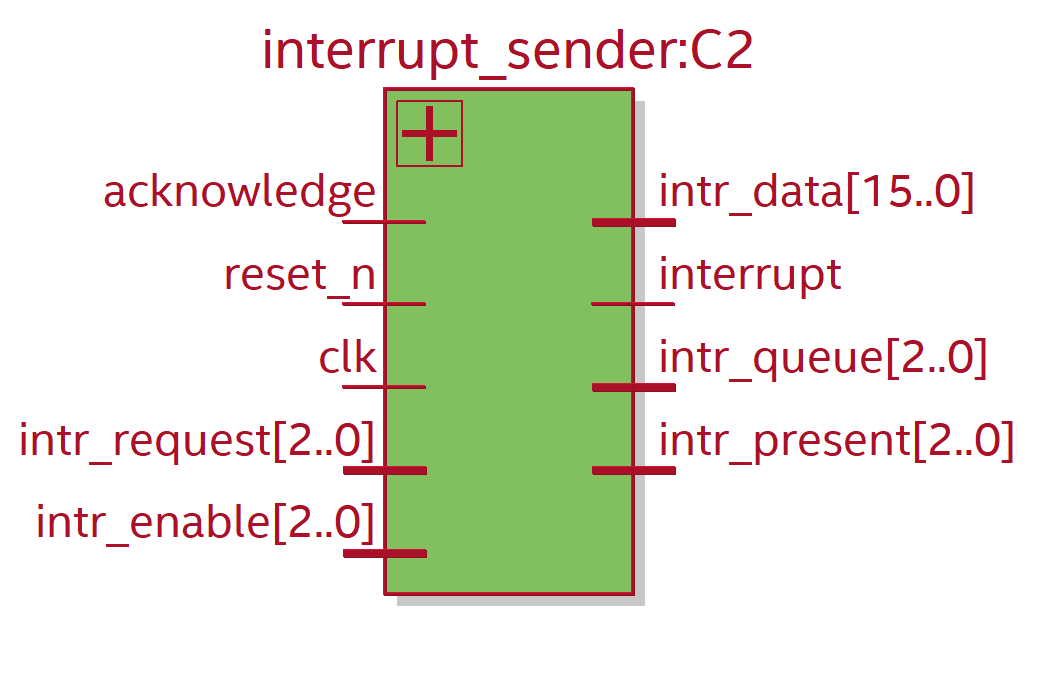
Tabell 4. Signalbeskrivning

|  |  |  |  |
| --- | --- | --- | --- |
| **Namn** | **Beskrivning** | **Riktning** | **Typ** |
| clk | Systemklocka 50Mhz | IN | STD\_LOGIC |
| reset\_n | Reset signal | IN | STD\_LOGIC |
| ir\_data\_0 | In data för interruotbit 0 | IN | STD\_LOGIC\_VECTOR |
| ir\_data\_1 | In data för interruotbit 1 | IN | STD\_LOGIC\_VECTOR |
| ir\_data\_2 | In data för interruotbit 2 | IN | STD\_LOGIC\_VECTOR |
|  |  |  |  |
| Intr\_out | Interruptbitar ut |  | STD\_LOGIC\_VECTOR |

### Interrupt\_sender

Komponentnamn: interrupt\_sender  
Instansnamn: C2

Läser status på de 3 interrupt bitar in till komponenten. Komponenten prioriterar den lägsta interrupt biten och sänder en interrupt signal då interrupt biten är “enabled”. Utgångssignalen intr\_present visar den interrupt bit som genererat den senaste interrupt signalen. Intr\_queue visar de interrupt bitar som är avvaktande för att generera nästa interrupt signal. Figur 12 visar komponenten och tabell 5 ger en signalbeskrivning.



Figur 12.interrupt\_sender

Tabell 5. Signalbeskrivning

|  |  |  |  |
| --- | --- | --- | --- |
| **Namn** | **Beskrivning** | **Riktning** | **Typ** |
| clk | Systemklocka 50Mhz | IN | STD\_LOGIC |
| reset\_n | Reset signal | IN | STD\_LOGIC |
| acknowledge | Ack signal från pio | IN | STD\_LOGIC |
| Intr\_request | Interruptbitar | IN | STD\_LOGIC\_VECTOR |
| Intr\_enable | Aktivering av interrupt[2..1] | IN | STD\_LOGIC\_VECTOR |
|  |  |  |  |
| Intr\_data | Data Antal IRQs | OUT | STD\_LOGIC\_VECTOR |
| Intr\_queue | Avvaktande interrupt | OUT | STD\_LOGIC\_VECTOR |
| Intr\_present | Indikera Interruptbit IRQ | OUT | STD\_LOGIC\_VECTOR |
| interrupt | Interruptsignal till ECS | OUT | STD\_LOGIC |

# Software

## Exempel Kod

|  |
| --- |
| **#include** <system.h>  **#include** <stdio.h>  **#include** <stdlib.h>  **#include** "altera\_avalon\_pio\_regs.h"  **#include** <alt\_types.h>  **#include** <io.h>  **#include** <HAL/inc/sys/alt\_irq.h>  //Declare ISR  **static** **void** **handle\_interrupt**(**void** \*pContext);  //Declare Global Variable  **static** ir\_freq;  **int** **main**()  {  //Säkerställa återställd ack-signal//  IOWR\_32DIRECT(INTERRUPT\_CONTROLLER\_IP\_0\_BASE,8,1);  IOWR\_32DIRECT(INTERRUPT\_CONTROLLER\_IP\_0\_BASE,8,0);  //--------------------------------//  /\*Skriva till Interrupt kontrollern  50000000 = 1 sec avbrottsintervall\*/  IOWR\_32DIRECT(INTERRUPT\_CONTROLLER\_IP\_0\_BASE,4,50000000);  //-------------------------------//  **int** \*pContext;  **printf**("Start\n");  //Registrera ISR  **if** (alt\_ic\_isr\_register(0,0, handle\_interrupt, \*pContext, 0x0)){ //IRQ och IRQ ID = 0  **printf**("Error Reg IRQ");  }  **while**(1)  {  // "." för att visa att CPUn arbetar//  **printf**(".");  **for**(size\_t i=0;i<1000000;i++);  }  }  **static** **void** **handle\_interrupt**(**void** \*pContext)  {  //Verifiera IRQ  **printf**("IRQ ");  /\* Antal IRQs registrerade i HW  Läs data från interrupt kontrollern\*/  ir\_freq= IORD\_32DIRECT(INTERRUPT\_CONTROLLER\_IP\_0\_BASE,0);  **printf**("Sent interrupt = %d\n",ir\_freq);  //HW-ACK//  // Isr = DOne, Återställ IRQ//  IOWR\_32DIRECT(INTERRUPT\_CONTROLLER\_IP\_0\_BASE,8,1);  IOWR\_32DIRECT(INTERRUPT\_CONTROLLER\_IP\_0\_BASE,8,0);  /\*\*\*\*\*\*\*\*\*\*\*\*\*/  } |

### Kod beskrivning

Tabell 6 beskriver delar i koden för att ge bättre förståelse av kodens funktion.

Tabell 6. Kod beskrivning

|  |  |
| --- | --- |
| **KOD** | **Beskrivning** |
| IOWR\_32DIRECT(INTERRUPT\_CONTROLLER\_IP\_0\_BASE,8,1);  IOWR\_32DIRECT(INTERRUPT\_CONTROLLER\_IP\_0\_BASE,8,0); | Aktiverar och återställer acknowledge signalen i ip komponenten för att medla slutförd ISR. HW: ack : std\_logic=’1’; ack : std\_logic=’0’; |
| alt\_ic\_isr\_register(0,0, handle\_interrupt, \*pContext, 0x0) | Registrera ISR. IP-komponentens interrupt signal i Platform Designer har prio 0. isr\_reg(irq,irq\_id,ISR,pointer,0x0); |
| IOWR\_32DIRECT(INTERRUPT\_CONTROLLER\_IP0\_BASE,4,50000000); | Skriver till ip-komponenten för ett avbrotsintervall. HW: data\_reg <=50000000; |
| **static** ir\_freq;  ir\_freq= IORD\_32DIRECT(INTERRUPT\_CONTROLLER\_IP0\_BASE,0);  **printf**("Sent interrupt = %d\n",ir\_freq); | Läser antal IRQs som registreras i Ip-komponenten för att skriva värdena i konsol fönstret. |

## Drivrutiner

Tabell 7 visar hur koden kan simplifieras med drivrutiner.

Tabell 7. Funktioner för interrupt IP komponenten

|  |  |
| --- | --- |
| **KOD** | **Med Drivrutiner** |
| IOWR\_32DIRECT(INTERRUPT\_CONTROLLER\_IP\_0\_BASE,8,1);  IOWR\_32DIRECT(INTERRUPT\_CONTROLLER\_IP\_0\_BASE,8,0); | **interrupt\_hw\_ack**(); |
| IOWR\_32DIRECT(INTERRUPT\_CONTROLLER\_IP0\_BASE,4,50000000); | **ir\_timer**(SEC\_1); |
| **static** ir\_freq;  ir\_freq= IORD\_32DIRECT(INTERRUPT\_CONTROLLER\_IP0\_BASE,0);  **printf**("Sent interrupt = %d\n",ir\_freq); | **print\_ir\_data**()**;** |

### Interrupt\_hw\_ack

Sänder en acknowledge signal till interrupt ip komponenten för att återställa interrupt flagga.

Funktionsanrop: interrupt\_hw\_ack();

Pseudokod:

* Aktivera acknowledge signalen, IOWR (interrupt\_IP,1);
* Avaktivera acknowledge signalen, IOWR (interrupt\_IP,0);

### Ir\_timer

Skriver timer värde till interrupt ip komponentenför interrupt bit 0.

Funktionsanrop: ir\_timer(*timer data*);

Pseudokod:

* IOWR (pio,0,data);

### Print\_ir\_data

Skriver ut antal genererade interrupt i konsol fönstret.

Funktionsanrop: print\_ir\_data();

Pseudokod:

* Int var
* Var= IORD32 interrupt ip
* Printf (“data %d”,var)

# Verifiering

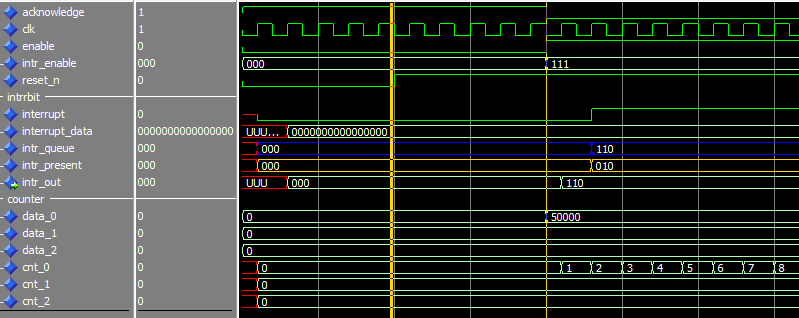
Tabell 8 visar testprotokollet för verifieringen.

|  |  |  |  |
| --- | --- | --- | --- |
| **Testfall** | **Beskrivning** | **Acceptans** | **Godkänt (J/N)** |
| **0** | Reset Aktiv Låg | System reset | Ja |
| **1** | Reset Aktiv Hög | System start | Ja |
| **2** | Enable Aktiv | Räknare startar | Ja |
| **3** | Interrupt0 räknare= data0 | Interrupt =’1’ intr\_presenr(0)=’1’ intr\_queue(0)=’1’ | Ja |
| **4** | Interrupt1 räknare= data1 | Interrupt =’1’ intr\_presenr(1)=’1’ intr\_queue(1)=’1’ | Ja |
| **5** | Interrupt2 räknare= data2 | Interrupt =’1’ intr\_presenr(2)=’1’ intr\_queue(2)=’1’ | Ja |
| **6** | Interruptbitar aktiv simultan (Verifiera prio) | Intr\_present=”001”  Intr\_queue “111” | Ja |

Tabell 8. Testprotokoll Verifiering

## Testfall 0-2

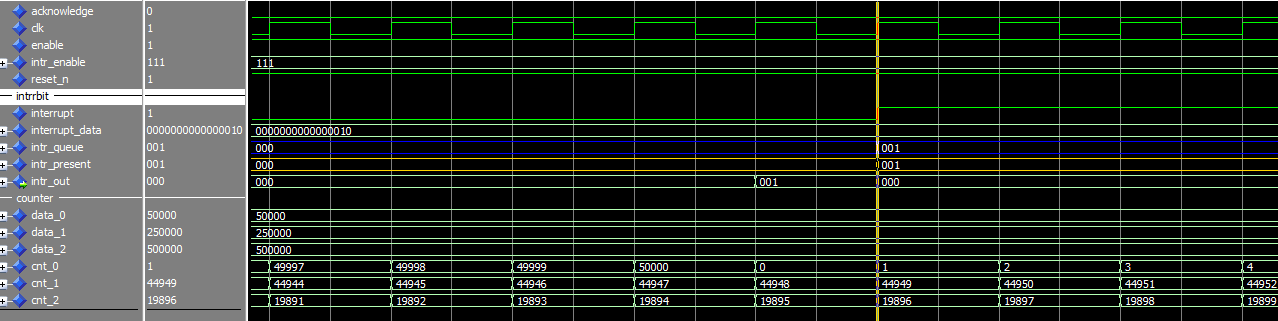
Figur 13 visar testfall 0-2 i Modelsim.



Figur 13. Testfall 0-2

## Testfall 3

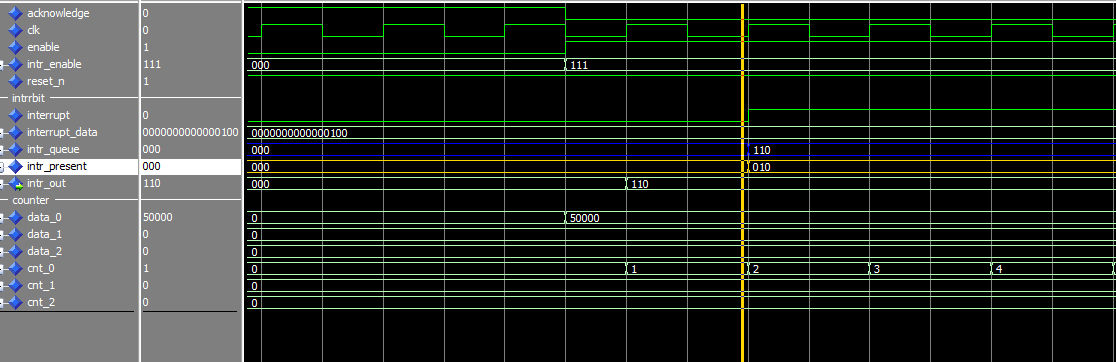
Figur 14 visar testfall 3 i Modelsim.



Figur 14. Testfall 3

## Testfall 4

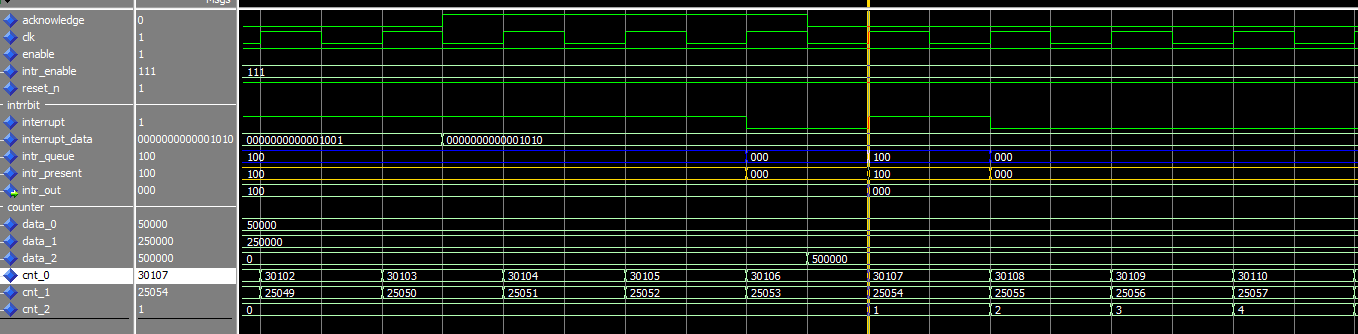
Figur 15 visar testfall 4 i Modelsim.



Figur 15. Testfall 4

## Testfall 5

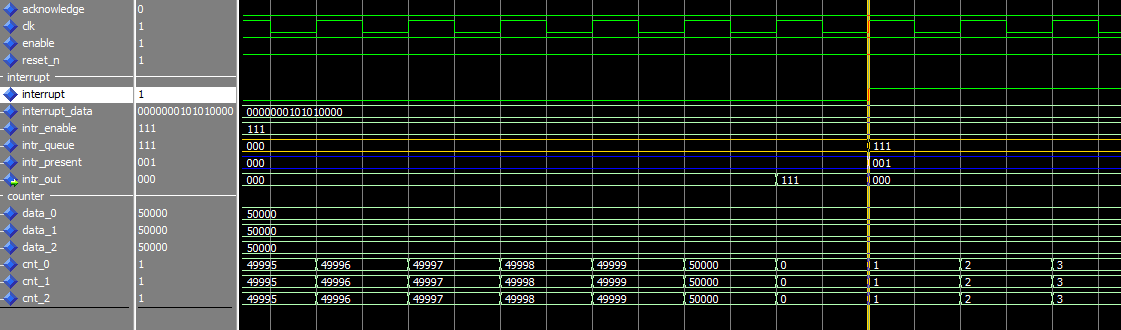
Figur 16 visar testfall 5 i Modelsim.



Figur 16. Testfall 5

## Testfall 6

Figur 17 visar testfall 6 i Modelsim.



Figur 17. Testfall 6

# Validering

Tabell 9 visar testprotokollet för valideringen.

Tabell 9. Testprotokoll Validering

|  |  |  |  |
| --- | --- | --- | --- |
| **Testfall** | **Beskrivning** | **Acceptans** | **Godkänt (J/N)** |
| **0** | Interruptbit 0 flaggar för interrupt | LEDR(0)=’1’ LEDR(4)=’1’ NIOS ISR Exekverar |  |
| **1** | Interruptbit 1 flaggar för interrupt | LEDR(1)=’1’ LEDR(5)=’1’ NIOS ISR Exekverar |  |
| **2** | Interruptbit 2 flaggar för interrupt | LEDR(2)=’1’ LEDR(6)=’1’ NIOS ISR Exekverar |  |
| **3** | Interruptbitar flaggar simultan (Validera prio) | LEDR(2 downto 0)=”001” LEDR(6 downto 4)=”111” NIOS ISR Exekverar |  |
| **4** | Switch(2 downto 0)=”000” | LEDR(6 downto 0)=’0’ Ingen ISR |  |

## Testfall 0

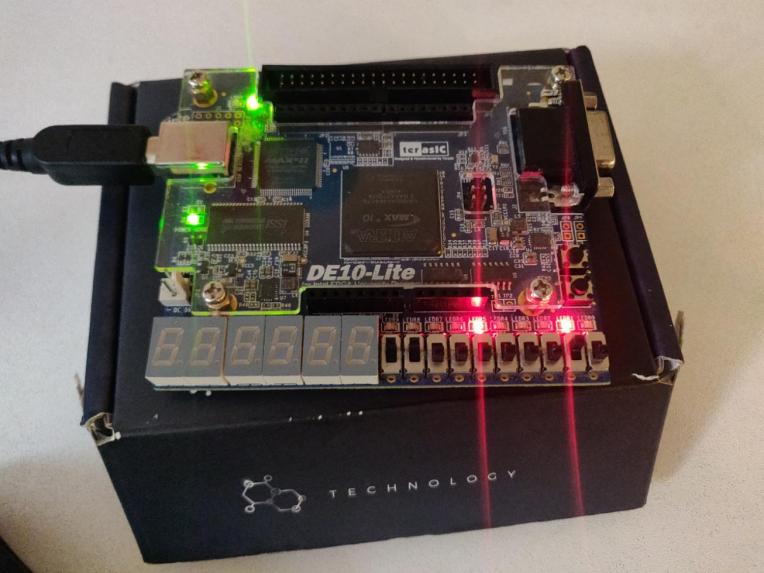
Figur 18 visar testfall 0 på utvecklingskortet.



Figur 18. Testfall 0

## Testfall 1

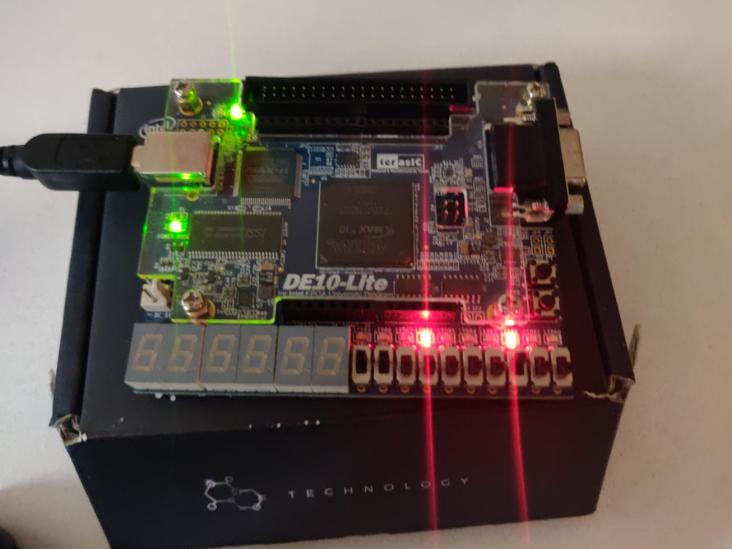
Figur 19 visar testfall 1 på utvecklingskortet.



Figur 19. Testfall 1

## Testfall 2

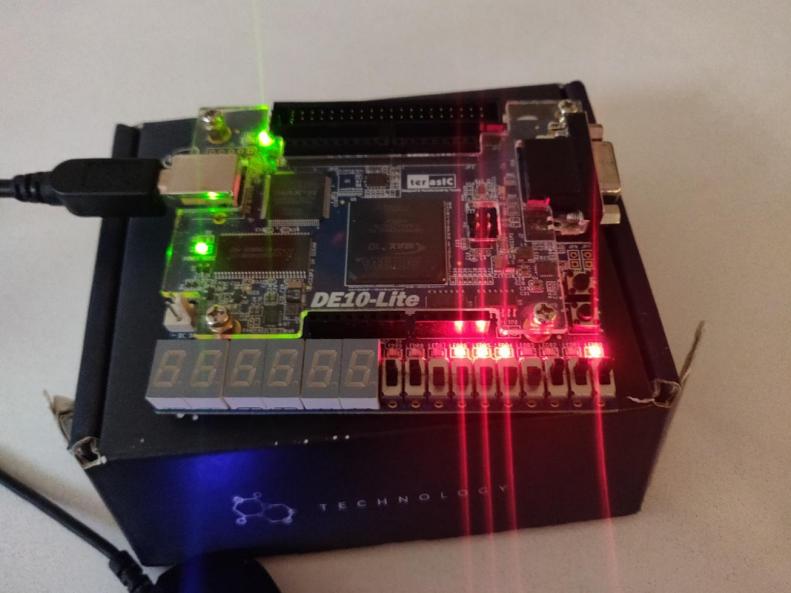
Figur 20 visar testfall 2 på utvecklingskortet.



Figur 20. Testfall 2

## Testfall 3

Figur 21 visar testfall 3 på utvecklingskortet.



Figur 21. Testfall 3

## Testfall 4

Figur 22 visar testfall 4 på utvecklingskortet.



Figur 22. Testfall

# Referenser

1. 21-09-01  
   <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/n2sw_nii52006.pdf>
2. 21-08-27  
   <https://opencores.org/projects/pic>
3. 21-08-27  
   <https://youtu.be/54BrU82ANww>
4. 21-08-27  
   <https://www.youtube.com/watch?v=yBYtfOJ4hcM>