ADC

Tim Ackermans & Dominic Voets

Om te voldoen aan de opdracht hebben we een aantal aanpassingen moeten doen. Om erachter te komen welke dit zijn hebben we het een en ander onderzocht. Over elk onderwerp hebben we een klein stukje verklaring

IRQ nummer

In de opdracht werd gehint dat ADC_INT de naam is van de interrupt. Als we deze opzoeken in de datasheet gaf dit de naam TS_IRQ.

Table 56. Interrupt Enable Register for Sub Interrupt Controller 1 (SIC1_ER - 0x4000 C000)

Bits	Name	Description	Reset value
31	USB_i2c_int	Interrupt from the USB I^2C interface.	0
30	USB_dev_hp_int	USB high priority interrupt.	0
8	TS_AUX	Touch screen aux interrupt	0
7	TS_IRQ (ADC_INT)	Touch screen irq interrupt	0
6	TS_P	Touch screen pen down interrupt.	0

Als we deze in irqs.h zoeken vinden we onderstaande define:

```
#define IRQ_LPC32XX_TS_IRQ        LPC32XX_SIC1_IRQ(7)
```

Hetzelfde geldt voor de EINT0 knop.

Table 57. Interrupt Enable Register for Sub Interrupt Controller 2 (SIC2_ER - 0x4001 0000)

Bits	Name	Description	Reset value
31	SYSCLK mux	Status of the SYSCLK Mux (SYSCLK_CTRL[0]). May be used to begin operations that require a change to the alternate clock source.	0
30:29	Reserved	Reserved, do not modify.	0
28	GPI_6	Interrupt from the GPI_6 (HSTIM_CAP) pin.	0
27	GPI_5	Interrupt from the GPI_5 pin.	0
26	GPI_4	Interrupt from the GPI_4 (SPI1_BUSY) pin.	0
25	GPI_3	Interrupt from the GPI_3 pin.	0
24	GPI_2	Interrupt from the GPI_2 pin.	0
23	GPI_1	Interrupt from the GPI_1 (SERVICE_N) pin.	0
22	GPI_0	Interrupt from the GPI_0 pin.	0

In irqs.h gevonden

```
#define IRQ_LPC32XX_GPI_01        LPC32XX_SIC2_IRQ(23)
```

ADC start en reset

Een van de TODOs is het aanzetten en resetten van de ADC, daarvoor hebben we het onderstaande register gevonden. Dit register kan de ADC aanzetten (bit 2), daarnaast wordt het gebruikt om de conversie te starten (Bit 1)

Datasheet: Register 0x4004 8008

		1 = Auto mode including position detect.	
2	TS_ADC_PDN_CTRL	This bit has no effect if AUTO_EN = 1 0 = the ADC is in power down. (Default) 1 = the ADC is powered up and reset.	0
1	TS_ADC_STROBE	This bit has no effect if AUTO_EN = 1 Setting this bit to logic 1 will start an AD conversion. This bit is write only	0

Bit 2 start de AD converter en voert een reset uit.
Bit 1 start de AD conversie.

Daarnaast hebben we het register gevonden om de waarden van de ADC uit te lezen na de conversie. Hiervoor gebruiken we een 10 bit mask.

12.3.5 ADC Value register (ADC_VALUE - 0x4004 8048)

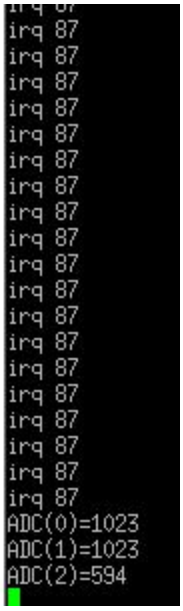
The ADC_VALUE register contains the result of the last completed A/D conversion. The result field in ADC_VALUE is shown in Table 261.

Table 261. A/D Data Register (ADC_VALUE - 0x4004 8048)

Bits	Function	Description	Reset value
31:10	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	-
9:0	ADC_VALUE	The ADC value of the last conversion.	

GPIO Interrupt

Bij het testen kwamen we erachter dat de GPIO interrupt vaak getriggerd wordt:



Om dit op te lossen schrijven we een 1 naar onderstaande bit in adres:

Table 65. Activation Type Register (SIC2_ATR - 0x4001 0010)

Bits	Name	Description	Operational value	Reset value
31	SYSCLK mux	Interrupt Activation Type, see bit 0 description		0
30:29	Reserved	Reserved, do not modify.	-	0
28	GPI_6	Interrupt Activation Type, see bit 0 description	user defined	0
27	GPI_5	Interrupt Activation Type, see bit 0 description	user defined	0
26	GPI_4	Interrupt Activation Type, see bit 0 description	user defined	0
25	GPI_3	Interrupt Activation Type, see bit 0 description	user defined	0
24	GPI_2	Interrupt Activation Type, see bit 0 description	user defined	0
23	GPI_1	Interrupt Activation Type, see bit 0 description	user defined	0
0	GPIO_0	Interrupt Activation Type, determines whether each interrupt is level sensitive or edge sensitive. 0 = Interrupt is level sensitive. (Default) 1 = Interrupt is edge sensitive.	user defined	0

Hierna triggered de interrupt maar 1x:

```
irq 87
ADC(0)=1023
ADC(1)=1023
ADC(2)=610
irq 87
ADC(0)=1023
ADC(1)=1023
ADC(2)=571
```

Naamgeving interrupt

Ook hebben we onze interrupts namen gegeven (adc, eint0), dit is terug te vinden in /proc/interrupts

```
# cat /proc/interrupts
           CPU0
  9:       1176          -  serial
 16:     116030          -  LPC32XX Timer Tick
 20:        240          -  pl022
 28:          0          -  DMA
 29:       1357          -  eth0
 39:        390          -  adc
 50:          0          -  pnx-i2c
 51:          0          -  pnx-i2c
 52:          0          -  rtc-lpc32xx
 59:       3132          -  ohci_hcd:usb1
 63:         46          -  pnx-i2c
 64:          0          -  ads7846
 87:     279834          -  eint0
Err:          0
#
```

Major nr

Dynamisch major nr gaf nr 253 terug, deze is nog vrij dus hebben we deze vastgest voor adc

```
# ./startADC
adc: major number=253
```

Daarnaast hebben we weer een opstart en afsluit script gemaakt

```
insmod ESadc.ko

mknod /dev/adc0 c 253 0
mknod /dev/adc1 c 253 1
mknod /dev/adc2 c 253 2
```

```
File  Edit  Search  C
rmmod ESadc.ko

rm /dev/adc0
rm /dev/adc1
rm /dev/adc2
```

Waiting for interrupt

We hebben de adc interrupt aangepast zodat deze niet automatisch alle channels langsgaat. Daarnaast hebben we de device read zo aangepast dat hij wacht op een event trigger voordat deze functie verder gaat.

```
static irqreturn_t adc_interrupt (int irq, void * dev_id)
{
    adc_values[adc_channel] = (READ_REG(ADC_VALUE) & ADC_VALUE_MASK);
    printk(KERN_WARNING "ADC(%d)=%d\n", adc_channel, adc_values[adc_channel]);

    conversionDone = 1;
    wake_up(&event);

    return (IRQ_HANDLED);
}
```

```
adc_start (channel);

// Wait for interrupt to be done handling
wait_event_interruptible(event, conversionDone == 1);
conversionDone = 0;
```

Timing testen

Om de timing te testen gebruiken we deels de kernel module die we voor de GPIO opdracht geschreven hebben:

```
# ./startGpio
/sys/kernel/gpio/config created
example to configure J2_24 as output: "echo "J2 24 0" > /sys/kernel/gpio/config"
# echo "J2 11 0" > /sys/kernel/gpio/config
sysfile_write (/sys/kernel/gpio/config) called
buffer: J2 11 0

Configured J2_11 as output
```

Vervolgens gebruiken we onderstaande code om de pin hoog en laag te maken voor en na de conversie, hoe lang deze pin hoog is geweest is te meten met een scope:

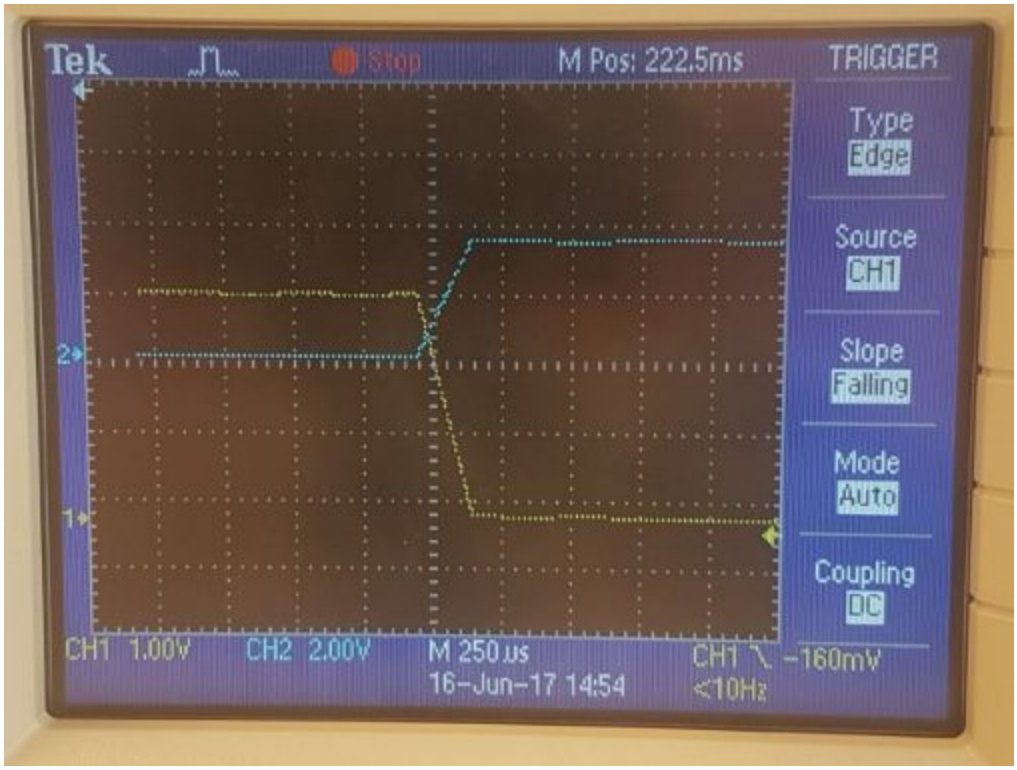
```
    // Pin hoog voor tijdmeting
    setNewRegValue(P0_OUT_SET, HIGH_MASK, BIT(2));

    // start conversie
    setNewRegValue(ADC_CTRL, HIGH_MASK, ADC_CONVERSIE_MASK);
}

static irqreturn_t adc_interrupt (int irq, void * dev_id)
{
    // Pin laag na tijdmeting
    setNewRegValue(P0_OUT_CLR, HIGH_MASK, BIT(2));
```

Voor het meten van de tijd tussen het indrukken van de knop en de interrupt meten we de spanning van de knop en een piek op dezelfde pin:

```
static irqreturn_t gp_interrupt(int irq, void * dev_id)
{
    // Pin hoog voor tijdmeting
    setNewRegValue(P0_OUT_SET, HIGH_MASK, BIT(2));
```

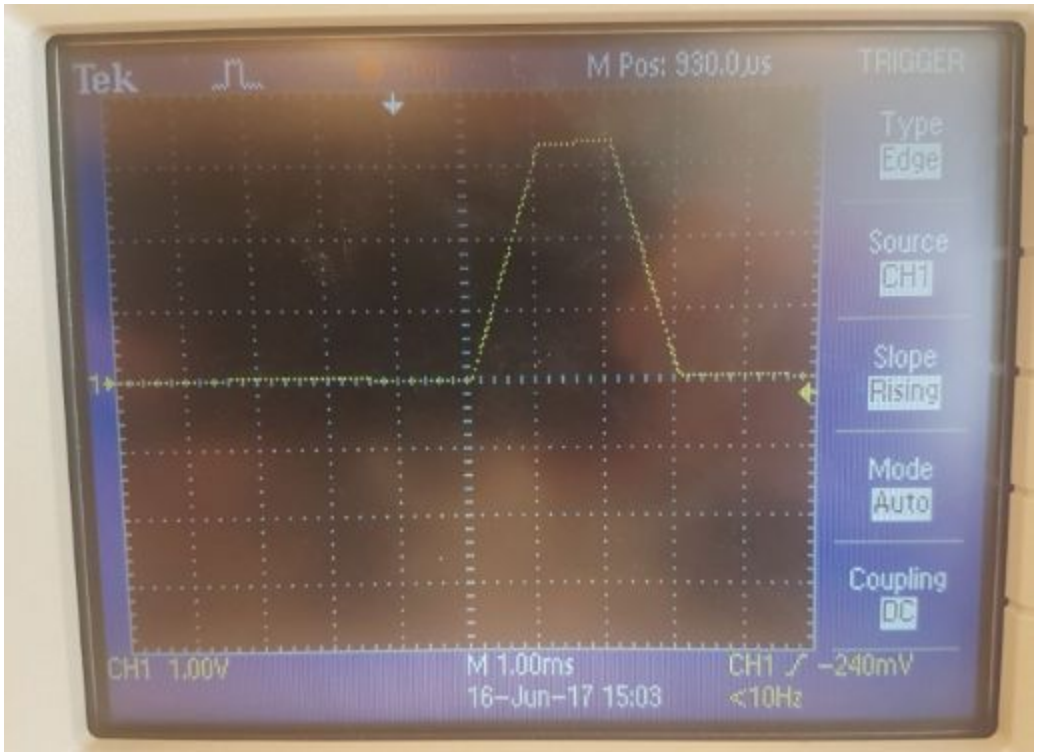
Het doen van de meting hebben wij samen met het groepje van Sam en Jesse gedaan. We gebruikte hiervoor onze code.

Knop timing:



Vrijwel gelijk

ADC conversion timing:



~3ms

DEMO

We kunnen alle channels uitlezen aan de hand van een simpel cat commando.

```
# cat /dev/adc2
adc:device_read(2)
adc: device_release()
ADC(2)=657
# cat /dev/adc1
adc:device_read(1)
adc: device_release()
ADC(1)=1023
# cat /dev/adc0
adc:device_read(0)
adc: device_release()
ADC(0)=1023
```

Een demo dat het uitlezen van de ADC werkt is te zien in onderstaande filmpje:
https://youtu.be/8Ci401azYcc