Fontys University of Applied Sciences

Software & Technology - 6th Semester

Assignment: **ADC**

Elviro Pereira Junior, **2719584**
Rafal Grasman, **289290**

24/05/2018

Research Results

Device Driver Type

To solve this assignment we decided that we would write a char device driver. Each read request will present a simple buffer, with a maximum of 3 device files (3 channel ADC).

ADC on LPC3250 board

The board has (from documentation) a 10-bit ADC which has three channels, with adjustable resolution (minimum 3 bits). Reducing the resolution reduces the conversion time. For the full 10 bits it takes 11 clock ticks (by default connected to 32kHz RTC clock, which makes it 0.34 ms maximum), with a maximum conversion rate of 400,000 samples / s.
(DataSheet, ch1.14.2/ch12.1.1)

The ADC works as follow, by setting the correct values in the corresponding registers (DataSheet, ch12.4):

1) Setup the ADC (channel to read):
    a) enable the clock configuration (ADCLK_CTRL, mask 0b1)

Table 46. ADC Clock Control register (ADCLK_CTRL - 0x4000 40B4)

Bit	Function	Reset value
31:1	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	-
0	0 = Disable 32 kHz clock to ADC block. 1 = Enable clock.	0

    b) Select the needed clock (RTC or PERIPH [faster] clock) (ADCLK_CTRL1, for example PERIPH/256 mask 0b111111111 [0x1FF])

Table 47. ADC Clock Control register (ADCLK_CTRL1 - 0x4000 4060)

Bit	Function	Reset value
31:9	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	-
8	ADCCLK_SEL; ADC clock select 0 = Clock ADC and touch screen from RTC clock. 1 = Clock ADC and touch screen from PERIPH_CLK clock.	0
7:0	ADC_FREQ. Controls the clock divider for ADC when Peripheral clock (bit 8) is enabled. Value in register is one less than divide value. reg value = (divider -1) 00000000 = 1 00000001 = 2 ..... 11111110 = 255 11111111 = 256	0

c) Set the accelerometer to measure both X and Y (ADC_SELECT, mask 0b1010000000 [0x280], ADC_CTRL is controlled in another part)

(DataSheet, p261)

- Configure the Y-axis for measurement by configuring ADC_SELECT to MEASURE_Y and start the ADC by setting the TS_ADC_STROBE bit in ADC_CTRL.
- On the TSC_IRQ interrupt, read the Y-axis data from ADC_VALUE register.
- Configure the Touch screen controller to drain the X-plate.
- Configure the AUX input for measurement by configuring ADC_SELECT to MEASURE_AUX and start the ADC by setting the TS_ADC_STROBE bit in ADC_CTRL.

(DataSheet, p268)

Table 238. A/D Select Register (ADC_SELECT - 0x4004 8004)

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.	-
9:8	TS_Ref- (AD_Ref–)	Selects the A/D negative reference voltage, Vref- in TS ADC 00 = TS_XM 01 = TS_YM 10 = Vss 11 = Not used	0
7:6	TS_Ref+ (AD_Ref+)	Selects the A/D positive reference voltage, Vref+ in TS ADC 00 = TS_XP 01 = TS_YP 10 = VDDTS 11 = Not used	0
5:4	TS_IN (AD_IN)	Selects the input TS ADC as follows: 00 = TS_YM 01 = TS_XM 10 = TS_AUX 11 = Not used	0
3	TS_YMC	YM Control 0 = the TS_YM signal is disconnected from GND. 1 = the TS_YM signal is connected to GND.	0
2	TS_YPC	YP Control 0 = the TS_YP signal is connected to VddTS. 1 = the TS_YP signal is disconnected from VddTS.	1
1	TS_XMC	XM Control 0 = the TS_XM signal is disconnected from GND. 1 = the TS_XM signal is connected to GND.	0
0	TS_XPC	XP Control 0 = the TS_XP signal is disconnected from VddTS. (Externally pulled down by 100k resistor.) (Typical) 1 = the TS_XP signal is connected to VddTS (only if TS_XMC bit is also 1).	0

d) Power on the ADC (ADC_CTRL, mask 0b100 [0x4])

(DataSheet, p269)

Table 239. A/D Control Register (ADC_CTRL - 0x4004 8008)

Bits	Function	Description	Reset value
31:13	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	-
12:11	TS_FIFO_CTRL	FIFO Control These bits set the level in the FIFO to trig the TSC_IRQ 00 = Send TSC_IRQ on FIFO level 1 01 = Send TSC_IRQ on FIFO level 4 10 = Send TSC_IRQ on FIFO level 8 11 = Send TSC_IRQ on FIFO level 16	0
10	TS_AUX_EN	This bit enables the AUX ANALOG measure in auto mode 0 = The AUX measured controller is disabled. 1 = The AUX measured controller is enabled.	0
9:7	TS_X_ACC	These bits sets the number of bits delivered by the ADC for all modes doing X direction measurement. (AUTO mode only). Fewer ADC bits used means fewer clocks to the ADC and faster acquire time. Note that the MSB bits used will stay in the same bit position in all registers. (They are not shifted down) 000 = ADC delivers 10 bits 001 = ADC delivers 9 bits ... 111 = ADC delivers 3 bits	0
6:4	TS_Y_ACC	These bits set the number of bits delivered by the ADC for all modes doing Y direction measurement. See description for TS_X_ACC.	0
3	TS_POS_DET	This bit has no effect if AUTO_EN = 0 0 = Normal auto mode. 1 = Auto mode including position detect.	0
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
0	TS_AUTO_EN	0 = The touch screen controller is disabled. The touch screen must be operated in manual mode. 1 = The touch screen controller is enabled. Bits 2:1 in this register are don't care when AUTO_EN is 1.	0

e) For this specific assignment: Make EINT0 edge level interrupt (SIC2_ATR, mask 0b10000000000000000000000 [0x800000]), see chapter ‘EINT0 Button’ for more info

2) Register the IRQs:

This can be done with the function:

```
int request_irq (unsigned int   irq,
        irq_handler_t  handler,
        unsigned long  irqflags,
        const char *     devname,
        void *   dev_id);
```

Don’t forget to:

```
void free_irq (  unsigned int irq,
        void * dev_id);
```

For example:

request_irq (IRQ_LPC32XX_TS_IRQ, adc_interrupt, IRQF_DISABLED, DEVICE_NAME "_CONVERT", NULL);

https://www.fsl.cs.sunysb.edu/kernel-api/re667.html
(the ‘devname’ can be checked for presence in ‘/proc/interrupts’ on linux)

a) IRQ_LPC32XX_TS_IRQ with name ES6_ADC_CONVERT for the conversion interrupt
b) IRQ_LPC32XX_GPI_01 with name ES6_ADC_EINT0 for the EINT0 button interrupt (see EINT0 button chapter, this is not needed for the ADC to work but needed for the Button interrupt to work)

3) Then, start the AD conversion when needed (ADC_CTRL, enable mask 0b10 [0x2], see previous table for ADC_CTRL), but before this:
   a) Select the channel in: ADC_SELECT (set AD_IN to the channel number, see previous table)
4) Wait for an A/D interrupt from AD_IRQ (the function chosen in 2) will be called)
5) Read from ADC_VALUE register (mask 0b1111111111 [0x3FF])

(DataSheet, p274)

Table 254. Touchscreen controller ADC Value Register (TSC_ADC_VALUE - 0x4004 8048)

Bits	Function	Description	Reset value
31:11	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	-
10	TSC_P_LEVEL	0 = the touch screen is pressed. 1 = the touch screen is not pressed.	-
9:0	ADC_VALUE	The ADC value of the last conversion.	0

6) Go to 3)

EINT0 Button

Listening for EINT0 is also required. The button on the base board schematic (v1.1) is connected to J3.Pin7 [P2.10], which leads to X1-35, which leads to GPI_01/SERVICE_N on the OEM board schematic (v1.3).

The button needs to be edge-interrupted, this can be done by setting GPI_01 to ‘edge’ (1) in the SIC2_ATR (mask 0b10000000000000000000000 [0x400000]) register (page 96, DataSheet):

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
22	GPI_0	Interrupt Activation Type, see bit 0 description	user defined	0
21	Reserved	Reserved, do not modify.	-	0
20	SPI1_DATIN	Interrupt Activation Type, see bit 0 description		0
19	U5_RX	Interrupt Activation Type, see bit 0 description		0

18	SDIO_INT_N	Interrupt Activation Type, see bit 0 description	Level (0)	0
17:16	Reserved	Reserved, do not modify.	-	0
15	GPI_7	Interrupt Activation Type, see bit 0 description	user defined	0
14:13	Reserved	Reserved, do not modify.	-	0
12	U7_HCTS	Interrupt Activation Type, see bit 0 description		0
11	GPI_19	Interrupt Activation Type, see bit 0 description	user defined	0
10	GPI_9	Interrupt Activation Type, see bit 0 description	user defined	0
9	GPI_8	Interrupt Activation Type, see bit 0 description	user defined	0
8	Pn_GPIO	Interrupt Activation Type, see bit 0 description		0
7	U2_HCTS	Interrupt Activation Type, see bit 0 description		0
6	SPI2_DATIN	Interrupt Activation Type, see bit 0 description		0
5	GPIO_5	Interrupt Activation Type, see bit 0 description	user defined	0
4	GPIO_4	Interrupt Activation Type, see bit 0 description	user defined	0
3	GPIO_3	Interrupt Activation Type, see bit 0 description	user defined	0
2	GPIO_2	Interrupt Activation Type, see bit 0 description	user defined	0
1	GPIO_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

Channels

As per ADC_SELECT register table, the following can be concluded:
Channel 0: Accelerometer Y value
Channel 1: Accelerometer X value
Channel 2: Red knob value

Design Decisions

To make reading of the data easy, the following devices have been made available:

/dev/adc0 /dev/adc1 /dev/adc2

Reading these files will return the ADC channel value. Writing is not possible (there is no need to in this assignment).

Concurrency

Only one conversion is possible at a time. Due to this we opted to make a semaphore in the critical section in the read function:

```
down(&channel_conversion); // decrease semaphore

current_task = current;
set_current_state(TASK_INTERRUPTIBLE);
adc_start (data->channel);
schedule();
value = adc_values[data->channel];

up(&channel_conversion); // increase semaphore
```

It will block until the interrupt is fired. Because the assignment required to start a conversion on interrupt, we decided to only start a conversion when ‘down_trylock’ semaphore succeeds in interrupt (we cannot sleep in interrupt so ‘down’ is not allowed). When the conversion finishes the semaphore is increased in value by using ‘up’ again in the adc interrupt. This way the conversion can only be started one time.

Testing

The module has been loaded and tested with the following:

Cat /dev/adc0 (accelerometer, tilting board forward/backward, values should stay same when stationary and change when moved)
Cat /dev/adc1 (accelerometer, tilting board sideways, values should stay same when stationary and change when moved)
Cat /dev/adc2 (the red knob turning and going from 0 to 512 linearly matching the knob turn)
Ls /proc/interrupts | grep ADC (to check whenever the interrupts are registered)
And the button EINT0 has been pressed, a message should appear then from the module that the interrupt has been executed.

For the time of conversation, based on the description we made above (section: ADC on LPC3250 board) we expected the ADC converversion to take approximately 0.34ms and after testing we measure ~0.45ms in average. The difference between our estimation and the result got from testing is due to the overhead of the code to calculate the timespan and executing the linux kernel.

To test the time between the Interrupt and button pressed we tried to figure out in which pin is the interrupt fired and we couldn't find it, and we assumed that it must be a wire on the circuit board. Then to make sure we verify the time, we decided to set a pin (J3.47 - see *GPIO assignment how to enable the pin*) HIGH when the button EINTO is pressed, and we used a oscilloscope to measure the time it takes from button press till the interrupt is handled and we reached the conclusion that it takes ~35 microseconds.

For testing the concurrency we ran the following script:

```
while :
do
        (echo "0: " $(cat /dev/adc0)) &
        (echo "1: " $(cat /dev/adc1)) &
        (echo "2: " $(cat /dev/adc2)) &
done
```

This script reads all three ADC channels concurrently (because the processes are spawned in the background). For "adc2" we turned the knob to the maximum so always expecting value 1023. Tests have shown the value keeps indeed at 1023 so our semaphore locking mechanism works and doesn't disrupt the results of other reads.

During our tests we saw no problems when pushing EINT0 while stressing.

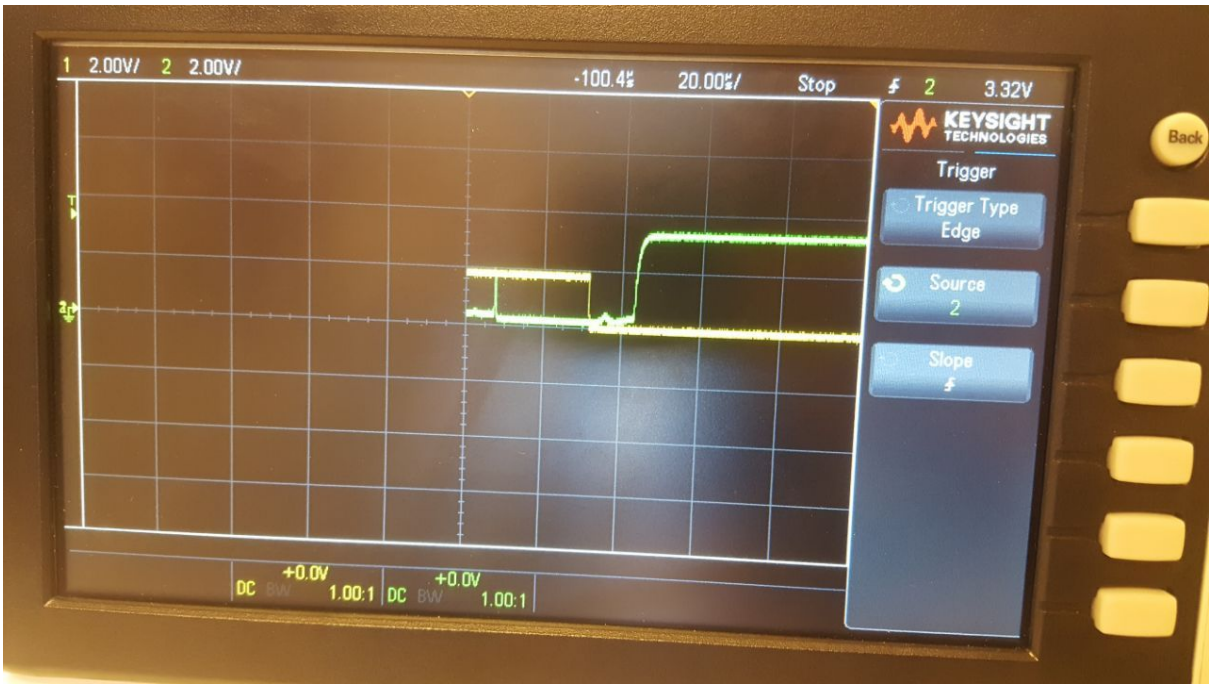
```
# cat /dev/adc0
ES6_ADC: device_read(0)
ES6_ADC: adc_start
ES6_ADC: adc_interrupt(0)=586
586ES6_ADC: device_release()
```

ScreenShots:

```
# cat /dev/adc1
ES6_ADC: device_read(1)
ES6_ADC: adc_start
ES6_ADC: adc_interrupt(1)=647
647ES6_ADC: device_release()
# cat /dev/adc1
ES6_ADC: device_read(1)
ES6_ADC: adc_start
ES6_ADC: adc_interrupt(1)=637
637ES6_ADC: device_release()
```

```
# ES6_ADC: gp_interrupt
ES6_ADC: adc_start
ES6_ADC: adc_interrupt measured 0 seconds and 448662 nanoseconds
ES6_ADC: adc_interrupt(0)=557
ES6_ADC: gp_interrupt
ES6_ADC: adc_start
ES6_ADC: adc_interrupt measured 0 seconds and 463115 nanoseconds
ES6_ADC: adc_interrupt(0)=599
```

```
# cat /dev/adc2
ES6_ADC: device_read(2)
ES6_ADC: adc_start
ES6_ADC: adc_interrupt(2)=691
691ES6_ADC: device_release()
# cat /dev/adc2
ES6_ADC: device_read(2)
ES6_ADC: adc_start
ES6_ADC: adc_interrupt(2)=674
674ES6_ADC: device_release()
# cat /dev/adc2
ES6_ADC: device_read(2)
ES6_ADC: adc_start
ES6_ADC: adc_interrupt(2)=681
681ES6_ADC: device_release()
```


1 2.00V/ 2 2.00V/
-100.4ns 20.00ns/ Stop
2 3.32V
KEYSIGHT TECHNOLOGIES
Back
Trigger
Trigger Type
Edge
Source
2
Slope
+0.0V
DC BW 1.00:1
+0.0V
DC BW 1.00:1

```
# insmod /home/adc.ko
ES6_ADC: mknod /dev/adc0 c 253 0
ES6_ADC: mknod /dev/adc1 c 253 1
ES6_ADC: mknod /dev/adc2 c 253 2
# cd home/
# ./stress.sh
1:  613
0:  596
0:  590
2:  1023
1:  646
0:  591
1:  642
2:  1023
2:  1023
0:  580
2:  1023
0:  585
1:  657
0:  585
1:  645
1:  643
1:  655
0:  585
0:  591
2:  1023
2:  1023
1:  618
2:  1023
1:  653
1:  646
2:  1023
0:  574
2:  1023
2:  1023
2:  1023
0:  607
2:  1023
0:  577
1:  664
0:  579
1:  633
0:  583
1:  631
1:  620
1:  635
2:  1023
0:  591
0:  580
0:  595
```

```
0:  551
2:  1023
2:  1023
ES6_ADC: gp_interrupt
0:  552
ES6_ADC: gp_interrupt
ES6_ADC: gp_interrupt
1:  598
2:  1023
ES6_ADC: gp_interrupt
1:  598
ES6_ADC: gp_interrupt
0:  622
1:  660
0:  565
ES6_ADC: gp_interrupt
0:  603
ES6_ADC: gp_interrupt
0:  577
0:  578
0:  595
2:  1023
ES6_ADC: gp_interrupt
2:  1023
ES6_ADC: gp_interrupt
ES6_ADC: gp_interrupt
0:  606
2:  1023
1:  653
2:  1023
1:  633
1:  618
1:  611
2:  1023
1:  622
ES6_ADC: gp_interrupt
0:  614
ES6_ADC: gp_interrupt
1:  659
2:  1023
1:  659
0:  584
2:  1023
0:  606
2:  1023
1:  648
0:  586
0:  589
1:  638
2:  1023
```