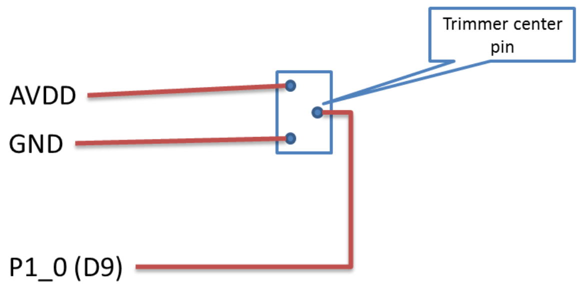
Introduction

ADC conversion testing

Wire I^2C temperature sensor and a trimmer to your board (in addition to the LCD). The trimmer is used to adjust the voltage into ADC channel 8 (P1_0).

Ask the instructor for the trimmer and the temperature sensor if you don't have them yet.



Now the trimmer acts an adjustable voltage divider where the voltage on the ADC input increases/decreases when you turn the trimmer.

Create a C++ project. Copy the example from the end of this document. Change the code to use ITM printing and test that you get proper readings from ADC. You should get values between 0 and 4095 by adjusting the trimmer.

Exercise 1 - Thermostat

Write a program that allows user to adjust target temperature with trimmer. Scale ADC values so that 0 corresponds to lowest operating temperature of the I2C temperature sensor and 4095 corresponds to the highest operating temperature. You can find the minimum and maximum operating temperature values in the temperature sensor data sheet. How close the trimmer setting is to the measured temperature is indicated by blinking a red or blue led. When the trimmer reading is lower than the temperature reading the blue led is blinking. The blinking frequency depends on the difference between the trimmer (scaled) value and temperature: the bigger difference the higher blinking rate. The red led blinks when the trimmer reading is higher than the temperature reading and the blinking frequency depends on difference. When the trimmer setting matches the temperature then the green led is constantly on (blue and red are off).

Summary:

- Green led on when the trimmer value matches temperature
- Red led blinks when trimmer reading is higher than temperature
- Blue led blinks when trimmer reading is lower than temperature
- Blinking frequency depends on difference:
  - big difference → high frequency
  - small difference → low frequency

Other requirements:

- ADC sample rate must be ~10 Hz (~100 ms intervals)
    - Use Sleep() for timing. It is not absolutely accurate but it is good enough for this purpose
    - Print ACD values and blink rate to ITM console
- Led blinking must be implemented in the Systick interrupt
- Blinking frequency must change smoothly
- Do not print anything in Systick_Handler (or any ISR in general)

Instructions:

- Read ADC values and do calculations in the main program
    - Set the variable(s) that determine blinking frequency in the main
    - Remember to use critical sections when modifying variables, that are shared with an ISR, in the main program
- Implement the led blinking in the ISR

Exercise 2 –Bar graph thermostat

In the course workspace you'll find a class for drawing a bar graph on the character lcd.

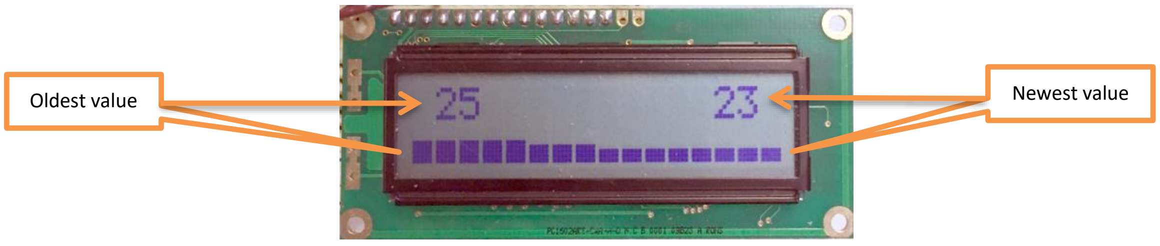
Update exercise 1 (Thermostat) to display a bar graph on the lcd in addition to blinking the tuning leds.

The upper row of the display must display a bar graph that shows the trimmer value. The maximum size of the bar must be 50 pixels long. The ADC value must be displayed on the left-hand side of the graph with four digits and leading spaces if less digits are needed. The bar graph must be scaled so that full trimmer range can be displayed. (0 = no pixels visible, 4095 = full bar).

The lower row displays the temperature sensor value. The bar must be 50 pixels long (max size of graph) and the temperature must be displayed on the left-hand side of the graph with four digits with leading spaces for shorter values. Temperature bar graph must be scaled so that the bar covers range from 0 to 30 C.

Exercise 3 – Trend display

Write a program that measures temperature at 5 second intervals and displays the values with 16 vertical bar graphs. Draw the bar graphs on the lower line. The oldest value is displayed on the left and newest on the right. The upper line displays oldest and newest temperature values in numerical form. Oldest value on the left and newest on the right (only the two numerical values are displayed). Scale the bar graphs so that 20 C (or lower) corresponds to lowest bar (no lines visible) and 28 C to full bar (8 lines visible). Display must be updated after each measurement.



```
/* Start ADC calibration */
void ADC_StartCalibration(LPC_ADC_T *pADC)
{
		// clock divider is the lowest 8 bits of the control register
		/* Setup ADC for about 500KHz (per UM) */
		uint32_t ctl = (Chip_Clock_GetSystemClockRate() / 500000) - 1;

		/* Set calibration mode */
		ctl |= ADC_CR_CALMODEBIT;

		pADC->CTRL = ctl;

		/* Calibration is only complete when ADC_CR_CALMODEBIT bit has cleared */
		while(pADC->CTRL & ADC_CR_CALMODEBIT) { };
}

int main(void) {

#if defined (__USE_LPCOPEN)
    // Read clock settings and update SystemCoreClock variable
    SystemCoreClockUpdate();
#if !defined(NO_BOARD_LIB)
    // Set up and initialize all required blocks and
    // functions related to the board hardware
    Board_Init();
    // Set the LED to the state of "On"
    Board_LED_Set(0, true);
#endif
#endif

    // TODO: insert code here
		/* Setup ADC for 12-bit mode and normal power */
		Chip_ADC_Init(LPC_ADC0, 0);

		/* Setup for ADC clock rate */
		Chip_ADC_SetClockRate(LPC_ADC0, 500000);

		/* For ADC0, sequencer A will be used without threshold events.
		   It will be triggered manually, convert CH8 and CH10 in the sequence  */
		Chip_ADC_SetupSequencer(LPC_ADC0, ADC_SEQA_IDX, (ADC_SEQ_CTRL_CHANSEL(8) |
ADC_SEQ_CTRL_CHANSEL(10) | ADC_SEQ_CTRL_MODE_EOS));

		// fix this and check if this is needed
		Chip_IOCON_PinMuxSet(LPC_IOCON, 0, 0, (IOCON_MODE_INACT | IOCON_ADMODE_EN));
		Chip_IOCON_PinMuxSet(LPC_IOCON, 1, 0, (IOCON_MODE_INACT | IOCON_ADMODE_EN));
		/* For ADC0, select analog input pin for channel 0 on ADC0 */
		Chip_ADC_SetADC0Input(LPC_ADC0, 0);

		/* Use higher voltage trim for both ADC */
		Chip_ADC_SetTrim(LPC_ADC0, ADC_TRIM_VRANGE_HIGHV);

		/* Assign ADC0_8 to PIO1_0 via SWM (fixed pin) and ADC0_10 to PIO0_0 */
		Chip_SWM_EnableFixedPin(SWM_FIXED_ADC0_8);
		Chip_SWM_EnableFixedPin(SWM_FIXED_ADC0_10);

		/* Need to do a calibration after initialization and trim */
		//while (!(Chip_ADC_IsCalibrationDone(LPC_ADC0))); // The NXP library function violates their
own access rules given in data sheet so we can't use it
		ADC_StartCalibration(LPC_ADC0);

		/* Set maximum clock rate for ADC */
		/* Our CPU clock rate is 72 MHz and ADC clock needs to be 50 MHz or less
		 * so the divider must be at least two. The real divider used is the value below + 1
		 */
		Chip_ADC_SetDivider(LPC_ADC0, 1);

		/* Chip_ADC_SetClockRate set the divider but due to rounding error it sets the divider too low
		 * which results in a clock rate that is out of allowed range
		 */
		//Chip_ADC_SetClockRate(LPC_ADC0, 500000); // does not work with 72 MHz clock when we want 
maximum frequency

		/* Clear all pending interrupts and status flags */
		Chip_ADC_ClearFlags(LPC_ADC0, Chip_ADC_GetFlags(LPC_ADC0));

		/* Enable sequence A completion interrupts for ADC0 */
		Chip_ADC_EnableInt(LPC_ADC0, ADC_INTEN_SEQA_ENABLE);
		/* We don't enable the corresponding interrupt in NVIC so the flag is set but no interrupt is 
triggered */

		/* Enable sequencer */
		Chip_ADC_EnableSequencer(LPC_ADC0, ADC_SEQA_IDX);

		/* Configure systick timer */
		SysTick_Config(Chip_Clock_GetSysTickClockRate() / TICKRATE_HZ);

		uint32_t a0;
		uint32_t d0;
		uint32_t a10;
		uint32_t d10;
		char str[80];

		while(1) {
			Chip_ADC_StartSequencer(LPC_ADC0, ADC_SEQA_IDX);

			// poll sequence complete flag
			while(!(Chip_ADC_GetFlags(LPC_ADC0) & ADC_FLAGS_SEQA_INT_MASK));
			// clear the flags
			Chip_ADC_ClearFlags(LPC_ADC0, Chip_ADC_GetFlags(LPC_ADC0));

			// get data from ADC channels
			a0 = Chip_ADC_GetDataReg(LPC_ADC0, 8); // raw value
			d0 = ADC_DR_RESULT(a0); // ADC result with status bits masked to zero and 
shifted to start from zero

			a10 = Chip_ADC_GetDataReg(LPC_ADC0, 10);
			d10 = ADC_DR_RESULT(a10);
			int diff = (int)d0 - (int)d10;
			sprintf(str, "%6d a0 = %08lX, d0 = %lu a10 = %08lX, d10 = %lu\n", diff, a0, d0, 
a10, d10);

			Board_UARTPutSTR(str);

			Sleep(100);

		}

    return 0 ;
}
```