Hochschule Heilbronn
Heilbronn University
ENGINEERING BUSINESS INFORMATICS
Campus Künzelsau
Reinhold-Würth-Hochschule

Chapter 4: Peripheral programming in C

Stellaris® Cortex-M3 - Microcontroller Family

Texas Instruments

Texas Instruments - University Program

Heilbronn University, Campus Künzelsau

Prof. Dr.-Ing. Ralf Gessler

Rev. 1.0

Contents

- Chapter 4: Peripheral Programming in C

- **Topics:** ADC, internal temperature sensor, EKS-LM3S1968, OLED, StellarisWare®, API, C language

Learning Objectives

- This chapter gives the user practical experience of the Stellaris® Analog Digital Converter (ADC)
- StellarisWare® is used to configure the ADC
- The application "ADC" prints the measured values of the on-chip temperature sensor on the OLED display, to provide a small real-world project.
- Structure and questions:
  - What are the functions of the ADC module?
  - What are the features of the ADC?
  - How can I apply StellarisWare® to configuring and running the ADC?

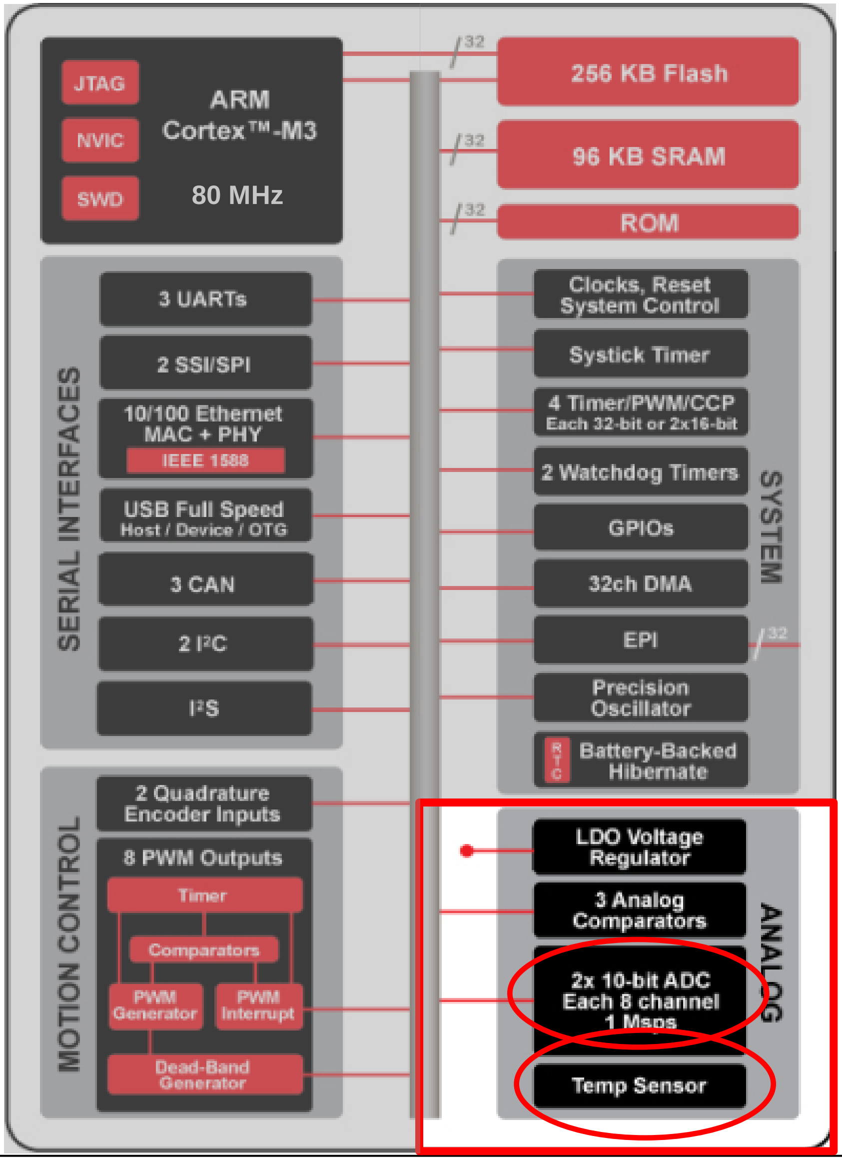
General view

HOCHSCHULE HEILBRONN
HEILBRONN UNIVERSITY
ENGINEERING BUSINESS INFORMATICS
Campus Künzelsau
Reinhold-Würth-Hochschule

- Analog-to-Digital Converter (ADC)
  - Principle
    - Converts analog signals to digital values:
      - From continuous time and magnitude
        to discrete time and continuous magnitude (Sample and Hold)
      - From discrete time and continuous magnitude
        to discrete time and magnitude (Quantisation)
    - Nyquist-Shannon Theorem:
      - $f_s > 2 \cdot f_{max}$
      - f_s: sampling frequency; f_{max}: maximum frequency (band limited)
  - Applications (examples)
    - Industrial applications, including:
      - Remote monitoring
      - Factory automation
      - Motion control,
      - Medical instrumentation
      - Fire and security.

Overview: Stellaris®

HOCHSCHULE HEILBRONN
HEILBRONN UNIVERSITY
ENGINEERING BUSINESS INFORMATICS
Campus Künzelsau
Reinhold-Würth-Hochschule



Explanations based on Stellaris® LM3S1968

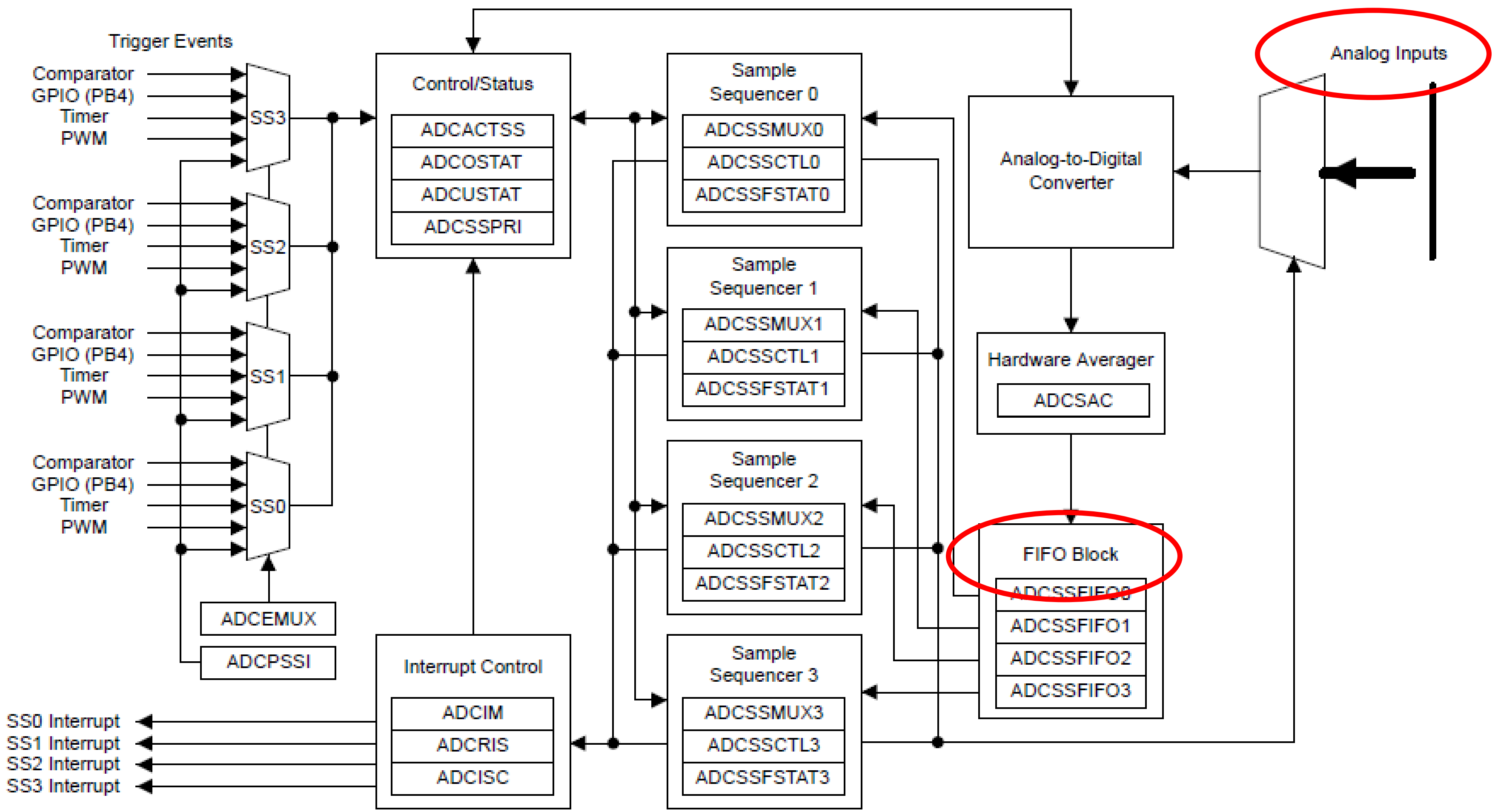
Main Features

- Eight multiplexed analog input channels
- Single-ended and differential-input configurations
- On-chip internal temperature sensor
- Up-to one million samples/second sample rate
- Flexible, configurable analog-to-digital conversion
- Four programmable sample conversion sequences from one to eight entries long, with corresponding conversion result First-In-First-Out registers (FIFOs)

Main Features

- Flexible trigger control
  - Controller (software)
  - Timers
  - Analog Comparators
  - Pulse Width Modulation (PWM)
  - General Purpose Input/Output (GPIO)
- Hardware averaging of up to 64 samples for improved accuracy
- Converter uses an internal 3V reference
- Power and ground for the analog circuitry is separate from the digital power and ground

ADC Block Diagram



Functional Description

- Overview
  - Apply the StellarisWare® API to the ADC
  - The analog to digital converter (ADC) API provides a set of functions to configure and run the ADC
  - Functions are provided to configure the sample sequencers, read the captured data, register a sample sequence interrupt handler, and handle interrupt masking/clearing
  - The StellarisWare® driver is contained in driverlib/adc.c, with driverlib/adc.h containing the API definitions for use by applications
- Link: see Chapter 3.4 “Stellarisware®” for further information.

ADC driver lib

- Analog-to-Digital Converter Application Programming Interface (API) consists of four function groups
  - Configuration
  - Enabling / Disabling
  - Obtaining data
  - Interrupt handling

- Configuration
  - The sample sequences are configured with:
    ADCSequenceConfigure() and ADCSequenceStepConfigure()

- Sample sequence FIFO overflow and underflow are managed by:
  ADCSequenceOverflow(), ADCSequenceOverflowClear(),
  ADCSequenceUnderflow() and ADCSequenceUnderflowClear()

- Hardware oversampling of the ADC is controlled by:
  ADCHardwareOversampleConfigure()

ADC driver lib II

- Configuration (cont.)
  - software oversampling of the ADC is controlled with: ADCSoftwareOversampleConfigure(), ADCSoftwareOversampleStepConfigure(), and ADCSoftwareOversampleDataGet()
  - enabling/Disabling with: ADCSequenceEnable() and ADCSequenceDisable()
  - the captured data is obtained with: ADCSequenceDataGet()
  - processor trigger is generated by: ADCProcessorTrigger()
  - interrupt handler for the ADC sample sequence interrupts are managed by: ADCIntRegister() and ADCIntUnregister().
  - the sample sequence interrupt sources are managed by: ADCIntDisable(), ADCIntEnable(), ADCIntStatus(), and ADCIntClear().

ADC driver lib III

HOCHSCHULE HEILBRONN
HEILBRONN UNIVERSITY
ENGINEERING BUSINESS INFORMATICS
Campus Künzelsau
Reinhold-Würth-Hochschule

- Interrupt-Handling
  - processor trigger is generated by:
    ADCProcessorTrigger()
  - the interrupt handler for the ADC sample sequence interrupts are managed by:
    ADCIntRegister() and ADCIntUnregister()
  - the sample sequence interrupt sources are managed by: ADCIntDisable(), ADCIntEnable(), ADCIntStatus(), and ADCIntClear()

Example “ADC”

- Description
  - This example application demonstrates the use of the ADC.
  - Reads the internal temperature sensor of the Cortex^{TM}-M3
  - Prints the measured digitalised values on the display
- Learning elements
  - Microcontroller
    - ADC
    - Internal temperature sensor
  - Evaluation Board
    - On-board display
- Functional test (debugging)
  - Measures room temperature
- Link: see Lab “lab44a.zip”

Example “ADC” Part I

HOCHSCHULE HEILBRONN
HEILBRONN UNIVERSITY
ENGINEERING BUSINESS INFORMATICS
Campus Künzelsau
Reinhold-Würth-Hochschule

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/adc.h"
#include "driverlib/sysctl.h"
#include "drivers/rit128x96x4.h"
#include "utils/ustdlib.h“
…

int
main(void)
{
    //
    // This array is used for storing the data read from the ADC FIFO. It
    // must be as large as the FIFO for the sequencer in use.  This example
    // uses sequence 3 which has a FIFO depth of 1.  If another sequence
    // was used with a deeper FIFO, then the array size must be changed.
    //
    unsigned long ulADC0_Value[1];

    //
    // These variables are used to store the temperature conversions for
    // Celsius and Fahrenheit.
    //
    unsigned long ulTemp_ValueC;
    //
    // HHN
    //
    // declare display output variable
    //
    char output[20];
…
```

- Load ADC-header
- Load display-header
- Load UART header

- Main routine

- Declare some variables

Example “ADC” Part II

HOCHSCHULE HEILBRONN
HEILBRONN UNIVERSITY
ENGINEERING BUSINESS INFORMATICS
Campus Künzelsau
Reinhold-Würth-Hochschule

```
…
    //
    // Set the clocking to run at 20 MHz (200 MHz / 10) using the PLL.  When
    // using the ADC, you must either use the PLL or supply a 16 MHz clock
    // source.
    //
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
                   SYSCTL_XTAL_8MHZ);


    //
    // Init OLED-Display
    //
    RIT128x96x4Init(1000000)
…
```

- Configure clocking
- Initialise the display

Example "ADC" Part III

Hochschule Heilbronn
Heilbronn University
ENGINEERING BUSINESS INFORMATICS
Campus Künzelsau
Reinhold-Würth-Hochschule

```
…
    //
    // The ADC0 peripheral must be enabled for use.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);

    //
    // Enable sample sequence 3 with a processor signal trigger.  Sequence 3
    // will do a single sample when the processor sends a signal to start the
    // conversion.  Each ADC module has 4 programmable sequences, sequence 0
    // to sequence 3.  This example is arbitrarily using sequence 3.
    //
    ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);

    //
    // Configure step 0 on sequence 3.  Sample the temperature sensor
    // (ADC_CTL_TS) and configure the interrupt flag (ADC_CTL_IE) to be set
    // when the sample is done.  Tell the ADC logic that this is the last
    // conversion on sequence 3 (ADC_CTL_END).  Sequence 3 has only one
    // programmable step.  Sequence 1 and 2 have 4 steps, and sequence 0 has
    // 8 programmable steps.  Since we are only doing a single conversion using
    // sequence 3 we will only configure step 0.  For more information on the
    // ADC sequences and steps, reference the datasheet.
    //
    ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_TS | ADC_CTL_IE |
                             ADC_CTL_END);

    //
    // Since sample sequence 3 is now configured, it must be enabled.
    //
    ADCSequenceEnable(ADC0_BASE, 3);
…
```

- Enable ADC Clock
- Configure ADC
- Use temperature sensor as source
- Enable sequencer

Example “ADC” Part IV


```
…
    //
    // Clear the interrupt status flag.  This is done to make sure the
    // interrupt flag is cleared before we sample.
    //
    ADCIntClear(ADC0_BASE, 3);

    //
    // Sample the temperature sensor forever.  Display the value on the
    // console.
    //
    while(1)
    {
        //
        // Trigger the ADC conversion.
        //
        ADCProcessorTrigger(ADC0_BASE, 3);

        //
        // Wait for conversion to be completed.
        //
        while(!ADCIntStatus(ADC0_BASE, 3, false))
        {
        }

        //
        // Read ADC Value.
        //
        ADCSequenceDataGet(ADC0_BASE, 3, ulADC0_Value);
…
```

- Prepare interrupt flag
- Loop forever
- Start conversion
- Wait for ADC to finish
- Get captured data

Example “ADC” Part V

```
…
    //
    // Use non-calibrated conversion provided in the data sheet.  Make
    // sure you divide last to avoid dropout.
    //
    ulTemp_ValueC = ((1475 * 1023) - (2250 * ulADC0_Value[0])) / 10230;

…
    //
    // generating the output string
    //
    usprintf(output, "Temperature = %d C", ulTemp_ValueC);
    //
    // display temperature on display
    //
    RIT128x96x4StringDraw(output, 2, 24, 15);

    //
    // This function provides a means of generating a constant length
    // delay.  The function delay (in cycles) = 3 * parameter.  Delay
    // 250ms arbitrarily.
    //
    SysCtlDelay(SysCtlClockGet() / 12);
  }
}
```

- Convert to temperature
- Generate output string
- Write temperature on display
- Wait 250ms

Questions and Exercises

1. Describe the principle of an analog to digital conversion.
2. Design a structured flowchart of the example “ADC”.
3. Which programming models apply to example “ADC”?
4. What is an API?

- Exercise
  1. Change the temperature in the room and compare the measured values with an analog thermometer.
  2. Connect a lineal potentiometer acting as voltage divider with an external analog input. Measure the voltage using the ADC.
     Link: see Chapter 3.1 “Evaluation boards”

Summary and Outlook

- Summary
  - From Analog-to-Digital Converter Module to StellarisWare® API
- Outlook/How to go on?
  - See Chapter 3.4 “StellarisWare®” for further information to the API.
  - The application “UART” introduced in Chapter 4.5 “Serial interface: UART” applies the ADC.

References

- [1] Texas Instruments: *Data Sheet - Stellaris® LM3S1968.* Chapter 12; spms037f.pdf, 2011.
- [2] Texas Instruments: *User's Guide - Stellaris ® Peripheral Driver Library*. Chapter 4; SW-DRL-UG-6075.pdf, 2010.
- [3] Avraam, N.: *Aufbau eines Mikroprozessortechnik-Labors mit der Stellaris® M3 Familie*. Labor-Versuch 4 "Blinky with SysTick"; Bachelor-Thesis; HS Heilbronn, Campus KÜN; 10.11.'11.
- [4] Texas Instruments: *StellarisWare®.* Path: examples/peripherals/adc/temperature_sensor, 2011
- [5] Brenner, S.; Mitsch, F.: *Project Teaching ROM*. Lab "temperature_sensor"; MEE-Project-Lab; HS Heilbronn, Campus KÜN, 20.06.2011.