```c
/*
 * adc_driver.h
 *
 *  Created on: Dec 10, 2018
 *      Author: Dominic Doty
 */

#ifndef INCLUDE_ADC_DRIVER_H_
#define INCLUDE_ADC_DRIVER_H_

/* INCLUDES */
#include "MKL25Z4.h"
#include "stddef.h"
#include "fsl_common.h"
#include "fsl_clock.h"
#include "fsl_port.h"

/* DEFINES & TYPEDEFS */
// Return errors
typedef enum
{
    ADC_ERROR_SUCCESS,
    ADC_ERROR_NULL_PTR,
    ADC_ERROR_CHANNEL_MODE_INCOMPATIBLE,
    ADC_ERROR_UNKNOWN_ADC,
    ADC_ERROR_FAILED_CAL
} adc_error;

// Channel definitions
typedef enum
{
    ADC_CHAN_DADP0,
    ADC_CHAN_DADP1,
    ADC_CHAN_DADP2,
    ADC_CHAN_DADP3,
    ADC_CHAN_AD4,
    ADC_CHAN_AD5,
    ADC_CHAN_AD6,
    ADC_CHAN_AD7,
    ADC_CHAN_AD8,
    ADC_CHAN_AD9,
    ADC_CHAN_AD10,
    ADC_CHAN_AD11,
    ADC_CHAN_AD12,
    ADC_CHAN_AD13,
    ADC_CHAN_AD14,
    ADC_CHAN_AD15,
    ADC_CHAN_AD16,
    ADC_CHAN_AD17,
    ADC_CHAN_AD18,
    ADC_CHAN_AD19,
    ADC_CHAN_AD20,
    ADC_CHAN_AD21,
    ADC_CHAN_AD22,
    ADC_CHAN_AD23,
    ADC_CHAN_TEMP = 0x1AU,
    ADC_CHAN_BANDGAP,
    ADC_CHAN_VREFSH = 0x1DU,
    ADC_CHAN_VREFSL,
    ADC_CHAN_DISABLED,
    ADC_CHAN_DAD0 = 0x20U,
    ADC_CHAN_DAD1,
    ADC_CHAN_DAD2,
    ADC_CHAN_DAD3,
    ADC_CHAN_TEMP_DIFF = 0x3AU,
    ADC_CHAN_BANDGAP_DIFF,
    ADC_CHAN_VREFSH_DIFF = 0x3DU
} adc_channel;

// Modified define to accommodate channel/diff enum setup
#define ADC_SC1_ADCH_DIFF(x)    (((uint32_t)(((uint32_t)(x)) << ADC_SC1_ADCH_SHIFT)) & (ADC_SC1_ADCH_MASK|ADC_SC1_DIFF_MASK))

// Interrupt on Complete
typedef enum
{
    ADC_NO_INT,
    ADC_INT_ON_COMPLETE
} adc_int_enable;
```

```c

// Low Power Mode
typedef enum
{
    ADC_POWER_NORMAL_MODE,
    ADC_POWER_LOW_MODE
} adc_power_mode;

// Clock Dividers
typedef enum
{
    ADC_CLOCK_DIV_1,
    ADC_CLOCK_DIV_2,
    ADC_CLOCK_DIV_4,
    ADC_CLOCK_DIV_8
} adc_clock_div;

// Clock Selections
typedef enum
{
    ADC_CLOCK_SEL_BUS,
    ADC_CLOCK_SEL_BUSDIV2,
    ADC_CLOCK_SEL_ALTCLK,
    ADC_CLOCK_SEL_ADACK
} adc_clock_sel;

// ADACK (Async Internal Clock) Frequency Look Up Table (I don't love this implementation)
#define ADACK_FREQUENCY_LUT    {5200000, 6200000, 2400000, 4000000}
#define ADACK_FREQUENCY_LUT_INDEX(ADC_CFG1_ADLPC, ADC_CFG1_ADLSMP) (((ADC_CFG1_ADLPC) << 1) | (ADC_CFG1_ADLSMP))

// Async Clock Mode
typedef enum
{
    ADC_ASYNC_CLOCK_ONLY_ADC,
    ADC_ASYNC_CLOCK_ALWAYS_ENABLED
} adc_async_clock_mode;

// Conversion Bit Modes
typedef enum
{
    ADC_BITS_8BIT,
    ADC_BITS_12BIT,
    ADC_BITS_10BIT,
    ADC_BITS_16BIT,
    ADC_BITS_9BIT_DIFF,
    ADC_BITS_13BIT_DIFF,
    ADC_BITS_11BIT_DIFF,
    ADC_BITS_16BIT_DIFF,
} adc_bits;

// ADC Bits Result Mask Look Up Table (convert mode to a &mask for the result)
#define ADC_BITS_RESULT_MASK_LUT {0xFFFFU,0xFFFFU,0xFFFFU,0xFFFFU,0x80FFU,0x8FFFU,0x83FF,0xFFFFU}

// ADC Bits Base # of Cyc Look Up Table (convert mode to # of cyc)
#define ADC_BITS_BASE_CYCLE_LUT {17,20,20,25,27,30,30,34}

// ADC Mode Mask
#define ADC_BITS(convert_mode)    ((convert_mode) & 0x3U)

// Sample Time Cycle Adder
typedef enum
{
    ADC_SMP_CYCLE_ADD_0,
    ADC_SMP_CYCLE_ADD_20 = 0x4U,
    ADC_SMP_CYCLE_ADD_12,
    ADC_SMP_CYCLE_ADD_6,
    ADC_SMP_CYCLE_ADD_2,
    ADC_SMP_CYCLE_ADD_HS_2 = 0x8U,
    ADC_SMP_CYCLE_ADD_HS_22 = 0xCU,
    ADC_SMP_CYCLE_ADD_HS_14,
    ADC_SMP_CYCLE_ADD_HS_8,
    ADC_SMP_CYCLE_ADD_HS_4
} adc_samp_cycle_adder;

// Sample Time Cycle Adder Look Up Table (convert Cyc add mode to # of cyc)
#define ADC_SAMP_CYCLE_ADDER_LUT    {0,0,0,0,20,12,6,2,2,0,0,0,22,14,8,4}

// Sample Time Cycle Adder Masks
#define ADC_SAMP_CYCLE_ADDER_ADLSTS(sample_cycle_add)    ((sample_cycle_add) & 0x3U)
```

```c
#define ADC_SAMP_CYCLE_ADDER_ADLSMP(sample_cycle_add)    (1 && ((sample_cycle_add) & 0x4U))
#define ADC_SAMP_CYCLE_ADDER_ADHSC(sample_cycle_add)    ((sample_cycle_add >> 3))

// Hardware Sample Averaging
typedef enum
{
    ADC_SAMP_AVG_1,
    ADC_SAMP_AVG_4 = 0x4U,
    ADC_SAMP_AVG_8,
    ADC_SAMP_AVG_16,
    ADC_SAMP_AVG_32,
} adc_samp_average;

// Hardware Sample Averaging Mask
#define ADC_SC3_AVG(mode)    ((((uint32_t)(mode)) << ADC_SC3_AVGS_SHIFT) & (ADC_SC3_AVGS_MASK | ADC_SC3_AVGE_MASK))

// Hardware Sample Averaging Look Up Table (convert avg mode to # of averages)
#define ADC_SAMP_AVERAGE_LUT    {1, 0, 0, 0, 4, 8, 16, 32}

// ADC Mux Selection
typedef enum
{
    ADC_MUX_A,
    ADC_MUX_B
} adc_mux_select;

// ADC Comparison Modes
typedef enum
{
    ADC_COMPARE_DISABLED,
    ADC_COMPARE_LESS = 0x4U,
    ADC_COMPARE_RANGE_EXCLUSIVE_INSIDE,
    ADC_COMPARE_GREATER,
    ADC_COMPARE_RANGE_INCLUSIVE_INSIDE,
    ADC_COMPARE_RANGE_EXCLUSIVE_OUTSIDE,
    ADC_COMPARE_RANGE_INCLUSIVE_OUTSIDE
}adc_compare_mode;

// ADC Comparison Modes Mask
#define ADC_SC2_COMP(mode)    ( ( ((uint32_t)(mode)) << ADC_SC2_ACREN_SHIFT) &                  \
                             (ADC_SC2_ACREN_MASK | ADC_SC2_ACFGT_MASK | ADC_SC2_ACFE_MASK) )

// ADC Trigger Modes
typedef enum
{
    ADC_TRIGGER_SOFTWARE,
    ADC_TRIGGER_HARDWRE
} adc_trigger_mode;

// ADC DMA Modes
typedef enum
{
    ADC_DMA_DISABLED,
    ADC_DMA_ENABLED
} adc_dma_mode;

// ADC Reference Voltages
typedef enum
{
    ADC_REFERENCE_VOLT_DEFAULT,
    ADC_REFERENCE_VOLT_ALT
} adc_ref_v;

// ADC Continuous Mode
typedef enum
{
    ADC_CONTINUOUS_ONESHOT,
    ADC_CONTINUOUS_CONTINUOUS
} adc_convert_mode;

// Initialization configuration structure
typedef struct
{
    ADC_Type * adc;
    adc_int_enable interrupt;
    adc_channel channel;
    adc_power_mode low_power;
    adc_clock_sel clock;
    adc_clock_div clock_div;
    adc_samp_cycle_adder sample_cycle_add;
    adc_bits bits;
    adc_samp_average avg_samps;
    adc_mux_select mux;
    adc_async_clock_mode async_state;
    adc_compare_mode compare_mode;
    uint16_t compare_1;
    uint16_t compare_2;
    adc_trigger_mode trigger;
    adc_dma_mode dma_mode;
    adc_ref_v    ref_volt;
    adc_convert_mode continuous;
    PORT_Type* port;
    uint32_t pin_1;
    uint32_t pin_2;
} adc_init_config;

// Default initialization
#define ADC_INIT_CONFIG_DEFAULT                        \
{                                                    \
        .adc = ADC0,                                \
        .interrupt = ADC_NO_INT,                    \
        .channel = ADC_CHAN_DISABLED,                \
        .low_power = ADC_POWER_NORMAL_MODE,            \
        .clock = ADC_CLOCK_SEL_ADACK,                \
        .clock_div = ADC_CLOCK_DIV_1,                \
        .sample_cycle_add = ADC_SMP_CYCLE_ADD_0,    \
        .bits = ADC_BITS_16BIT,                        \
        .avg_samps = ADC_SAMP_AVG_1,                \
        .mux = ADC_MUX_A,                            \
        .async_state = ADC_ASYNC_CLOCK_ONLY_ADC,    \
        .compare_mode = ADC_COMPARE_DISABLED,        \
        .compare_1 = 0,                                \
        .compare_2 = 0,                                \
        .trigger = ADC_TRIGGER_SOFTWARE,            \
        .dma_mode = ADC_DMA_DISABLED,                \
        .ref_volt = ADC_REFERENCE_VOLT_DEFAULT,        \
        .continuous = ADC_CONTINUOUS_ONESHOT,        \
        .port = NULL,                                \
        .pin_1 = 0,                                    \
        .pin_2 = 0                                    \
}


/* FUNCTION DECLARATIONS */

// Initialize the ADC with set parameters
adc_error adc_init(adc_init_config* config);

// Start an ADC Conversion - Only needed in single shot mode, init automatically starts continuous mode
void adc_start_conversion(ADC_Type* adc, adc_mux_select mux, adc_channel channel);

// Get result blocking
uint16_t adc_blocking_result(ADC_Type* adc, adc_mux_select mux, adc_bits bits);

// Calculate the actual sample rate from given configuration
uint32_t adc_sample_rate_calc(adc_init_config* config);

#endif /* INCLUDE_ADC_DRIVER_H_ */
```