1. DUST SENSOR (USING ADC): -

/\*Author: Sahana Sadagopan and Sanika Dongre\*/

**#include** "em\_device.h"

**#include** "em\_chip.h"

**#include** "hal-config.h"

**#include** "em\_letimer.h"

**#include** "em\_timer.h"

**#include** "em\_cmu.h"

**#include** "em\_gpio.h"

**#include** "em\_emu.h"

**#include** "em\_core.h"

**#include** "em\_acmp.h"

**#include** "em\_adc.h"

**#include** <stdint.h>

**#include** <stdbool.h>

**void** **SleepMode**(**void**){

EMU\_DCDCInit\_TypeDef dcdcInit = EMU\_DCDCINIT\_DEFAULT;

**EMU\_DCDCInit**(&dcdcInit);

**EMU\_EnterEM2**(true);

}

**void** **sleep**(**void**)

{

**if** (sleep\_block\_counter [0] > 0) {

**return**; // Blocked everything below EM0, so just return

}

**else** **if** (sleep\_block\_counter [1] > 0) {

EMU\_EnterEM1(); // Blocked everything below EM1, enter EM1

}

**else** **if** (sleep\_block\_counter[2] > 0) {

**EMU\_EnterEM2**(true); // Blocked everything below EM2, enter EM2

}

**else** **if** (sleep\_block\_counter[3] > 0) {

**EMU\_EnterEM3**(true); // Blocked everything below EM3, enter EM3

}

**return**;

}

**void** **blockSleepMode**(uint32\_t minimumMode)

{

//instead of INT\_Disable;

CORE\_DECLARE\_IRQ\_STATE;

CORE\_ENTER\_ATOMIC();

sleep\_block\_counter[minimumMode]++;

CORE\_EXIT\_ATOMIC();

//instead of INT\_Enable;

}

**void** **unblockSleepMode**(uint32\_t minimumMode)

{

//instead of INT\_Disable();

CORE\_DECLARE\_IRQ\_STATE;

CORE\_ENTER\_ATOMIC();

**if**(sleep\_block\_counter[minimumMode]>0){

sleep\_block\_counter[minimumMode]--;

}

CORE\_EXIT\_ATOMIC();

//instead of INT\_Enable();

}

**void** **ADC0\_IRQHandler**(**void**)

{

uint32\_t Flag = ADC\_IntGet(ADC0);

CORE\_ATOMIC\_IRQ\_DISABLE();

/\*Clear the interrupting flags\*/

ADC\_IntClear(ADC0, Flag);

CORE\_ATOMIC\_IRQ\_ENABLE();

}

**void** **adc\_Setup**(**void**){

//SleepMode();

NVIC\_DisableIRQ(*ADC0\_IRQn*);

uint8\_t timeBaseValue = **ADC\_TimebaseCalc**(**CMU\_ClockFreqGet**(*cmuClock\_HFPER*));

ADC\_Init\_TypeDef initadc={

.ovsRateSel = *adcOvsRateSel2*, //ADC\_OvsRateSel\_TypeDef

.warmUpMode = *adcWarmupKeepADCWarm*, //ADC\_Warmup\_TypeDef

.em2ClockConfig = 0,

.timebase = timeBaseValue, //uint8\_t

.prescale = 54, //calculated for

.tailgate = 0

};

ADC\_InitScan\_TypeDef scaninit = ADC\_INITSCAN\_DEFAULT;

**ADC\_Init**(ADC0,&initadc);

**ADC\_InitScan**(ADC0,&scaninit);

//ADC0->CMD= ADC\_CMD\_SCANSTART;

ADC0->SCANCTRL |= ADC\_SCANCTRL\_CMPEN || ADC\_SCANCTRL\_AT\_DEFAULT || ADC\_SCANCTRL\_REF\_DEFAULT;

ADC0->SCANCTRLX |= \_ADC\_SCANCTRLX\_VREFSEL\_VDDXWATT || ADC\_SCANCTRLX\_VREFATTFIX;

ADC0->SCANMASK |= \_ADC\_SCANMASK\_SCANINPUTEN\_INPUT0|\_ADC\_SCANMASK\_SCANINPUTEN\_INPUT2;

ADC0->SCANINPUTSEL = \_ADC\_SCANINPUTSEL\_INPUT0TO7SEL\_APORT1CH0TO7|\_ADC\_SCANINPUTSEL\_INPUT0TO7SEL\_APORT2CH0TO7;

ADC\_IntEnable(ADC0, ADC\_IEN\_SCANCMP );

NVIC\_ClearPendingIRQ(*ADC0\_IRQn*);

NVIC\_EnableIRQ(*ADC0\_IRQn*);

ADC\_Start(ADC0, *adcStartScan*);

}

**void** **adc\_read**(){

uint32\_t ADCdata;

ADC0->SCANFIFOCLEAR |=ADC\_SCANFIFOCLEAR\_SCANFIFOCLEAR;

ADCdata=ADC\_DataScanGet(ADC0);

}

**int** **main**(**void**)

{

/\* Chip errata \*/

CHIP\_Init();

/\*To set in EM2\*/

//Low Noise initialization

/\*Start ADC setup\*/

adc\_Setup();

adc\_read();

}