#define LOG\_TAG "app\_task"

#include "app\_task.h"

#include <stdlib.h>

#include <stdio.h>

#include <board.h>

#include <elog.h>

#include <elog\_flash.h>

#include <easyflash.h>

#include <delay\_conf.h>

#include <cpuusage.h>

#include <utils.h>

#include <shell.h>

#include <finsh.h>

#include <cm\_backtrace.h>

#include <dfs\_elm.h>

#include <dfs\_fs.h>

#include <spi\_flash.h>

#include <spi\_flash\_sfud.h>

#include <partition.h>

#define SOFTWARE\_VERSION "1.07.15"

static uint8\_t cpu\_usage\_major, cpu\_usage\_minor;

static rt\_uint32\_t total\_mem, used\_mem, max\_used\_mem;

#define thread\_sys\_monitor\_prio 30

ALIGN(RT\_ALIGN\_SIZE)

static rt\_uint8\_t thread\_sys\_monitor\_stack[512];

struct rt\_thread thread\_sys\_monitor;

rt\_spi\_flash\_device\_t nor\_flash;

static void rtt\_user\_assert\_hook(const char\* ex, const char\* func, rt\_size\_t line);

static void elog\_user\_assert\_hook(const char\* ex, const char\* func, size\_t line);

static rt\_err\_t exception\_hook(void \*context);

/\*\*

\* @param parameter parameter

\*/

void thread\_entry\_sys\_monitor(void\* parameter)

{

while (1)

{

rt\_memory\_info(&total\_mem, &used\_mem, &max\_used\_mem);

// rt\_kprintf("Total\_Mem:%ld Used\_Mem:%ld Max\_Used\_Mem:%ld\n",total\_mem,used\_mem,max\_used\_mem);

IWDG\_Feed(); //

}

}

/\*\*

\* @param parameter parameter

\*/

void sys\_init\_thread(void\* parameter){

set\_system\_status(SYSTEM\_STATUS\_INIT);

#ifdef RT\_USING\_COMPONENTS\_INIT

rt\_components\_init();

#endif

#if defined(RT\_USING\_FINSH) && !defined(RT\_USING\_POSIX)

finsh\_set\_device(RT\_CONSOLE\_DEVICE\_NAME);

#endif

if ((nor\_flash = rt\_sfud\_flash\_probe("nor\_flash", "spi20")) == NULL) {

set\_system\_status(SYSTEM\_STATUS\_FAULT);

return;

}

extern int rt\_partition\_init(const char\* flash\_device, const struct rt\_partition\* parts, rt\_size\_t num);

static const struct rt\_partition partition[] = RT\_PARTITION\_DEFAULT\_TABLE;

rt\_partition\_init("nor\_flash", partition, sizeof(partition) / sizeof(struct rt\_partition));

easyflash\_init();

elog\_init();

elog\_set\_fmt(ELOG\_LVL\_ASSERT, ELOG\_FMT\_ALL & ~ELOG\_FMT\_P\_INFO);

elog\_set\_fmt(ELOG\_LVL\_ERROR, ELOG\_FMT\_ALL & ~ELOG\_FMT\_P\_INFO);

elog\_set\_fmt(ELOG\_LVL\_WARN, ELOG\_FMT\_ALL & ~ELOG\_FMT\_P\_INFO);

elog\_set\_fmt(ELOG\_LVL\_INFO, ELOG\_FMT\_LVL | ELOG\_FMT\_TAG | ELOG\_FMT\_TIME);

elog\_set\_fmt(ELOG\_LVL\_DEBUG, ELOG\_FMT\_ALL & ~(ELOG\_FMT\_FUNC | ELOG\_FMT\_P\_INFO));

elog\_set\_fmt(ELOG\_LVL\_VERBOSE, ELOG\_FMT\_ALL & ~(ELOG\_FMT\_FUNC | ELOG\_FMT\_P\_INFO));

#ifdef DEBUG

elog\_set\_filter\_lvl(ELOG\_LVL\_VERBOSE);

#elif RELEASE

elog\_set\_filter\_lvl(ELOG\_LVL\_INFO);

#endif

#ifdef ELOG\_COLOR\_ENABLE

elog\_set\_text\_color\_enabled(true);

#endif

elog\_flash\_init();

elog\_start();

extern void flash\_read\_normalized(void);

flash\_read\_normalized();

elog\_assert\_set\_hook(elog\_user\_assert\_hook);

rt\_hw\_exception\_install(exception\_hook);

rt\_assert\_set\_hook(rtt\_user\_assert\_hook);

cm\_backtrace\_init("ART-6LoWPAN", HARDWARE\_VERSION, SOFTWARE\_VERSION);

if (dfs\_mount("fatfs", "/", "elm", 0, 0) == 0) {

log\_i("File System initialized!");

} else {

log\_e("File System initialization failed!");

}

extern void contiki\_init(void);

contiki\_init();

set\_system\_status(SYSTEM\_STATUS\_RUN);

log\_i("ART-6LoWPAN initialize success. Firmware version is %s", SOFTWARE\_VERSION);

}

static void rtt\_user\_assert\_hook(const char\* ex, const char\* func, rt\_size\_t line) {

rt\_enter\_critical();

#ifdef ELOG\_ASYNC\_OUTPUT\_ENABLE

elog\_async\_enabled(false);

#endif

elog\_output\_lock\_enabled(false);

elog\_flash\_lock\_enabled(false);

elog\_a("rtt", "(%s) has assert failed at %s:%ld.", ex, func, line);

cm\_backtrace\_assert(cmb\_get\_sp());

elog\_flash\_flush();

while(1);

}

static void elog\_user\_assert\_hook(const char\* ex, const char\* func, size\_t line) {

rt\_enter\_critical();

#ifdef ELOG\_ASYNC\_OUTPUT\_ENABLE

elog\_async\_enabled(false);

#endif

elog\_output\_lock\_enabled(false);

elog\_flash\_lock\_enabled(false);

elog\_a("elog", "(%s) has assert failed at %s:%ld.\n", ex, func, line);

cm\_backtrace\_assert(cmb\_get\_sp());

elog\_flash\_flush();

while(1);

}

static rt\_err\_t exception\_hook(void \*context) {

extern long list\_thread(void);

uint8\_t \_continue = 1;

rt\_enter\_critical();

#ifdef ELOG\_ASYNC\_OUTPUT\_ENABLE

elog\_async\_enabled(false);

#endif

elog\_output\_lock\_enabled(false);

elog\_flash\_lock\_enabled(false);

#ifdef RT\_USING\_FINSH

list\_thread();

#endif

cm\_backtrace\_fault(\*((uint32\_t \*)(cmb\_get\_sp() + sizeof(uint32\_t) \* 8)), cmb\_get\_sp() + sizeof(uint32\_t) \* 9);

elog\_flash\_flush();

while (\_continue == 1);

return RT\_EOK;

}

\*/

void rt\_hw\_console\_output(const char \*str) {

extern void output\_log\_to\_console\_or\_flash(bool console, bool flash, const char \*log, size\_t size);

output\_log\_to\_console\_or\_flash(true, true, str, rt\_strlen(str));

}

int rt\_application\_init(void)

{

rt\_thread\_t thread = NULL;

rt\_thread\_init(&thread\_sys\_monitor,

"sys\_monitor",

thread\_entry\_sys\_monitor,

RT\_NULL,

thread\_sys\_monitor\_stack,

sizeof(thread\_sys\_monitor\_stack),

thread\_sys\_monitor\_prio, 5);

rt\_thread\_startup(&thread\_sys\_monitor);

thread = rt\_thread\_create("sys init", sys\_init\_thread, NULL, 2048, 10, 10);

if (thread != NULL) {

rt\_thread\_startup(thread);

}

return 0;

}

/\*\*

\* This function will startup RT-Thread RTOS.

\*/

void rtthread\_startup(void)

{

/\* init board \*/

rt\_hw\_board\_init();

/\* show version \*/

rt\_show\_version();

/\* init tick \*/

rt\_system\_tick\_init();

/\* init kernel object \*/

rt\_system\_object\_init();

/\* init timer system \*/

rt\_system\_timer\_init();

#ifdef RT\_USING\_HEAP

/\* init memory system \*/

rt\_system\_heap\_init((void\*)STM32\_SRAM\_BEGIN, (void\*)STM32\_SRAM\_END);

#endif

/\* init scheduler system \*/

rt\_system\_scheduler\_init();

/\* initialize timer \*/

rt\_system\_timer\_init();

/\* init timer thread \*/

rt\_system\_timer\_thread\_init();

/\* Add CPU usage to system \*/

cpu\_usage\_init();

/\* init application \*/

rt\_application\_init();

/\* init idle thread \*/

rt\_thread\_idle\_init();

/\* start scheduler \*/

rt\_system\_scheduler\_start();

/\* never reach here \*/

return;

}

void sys\_reboot\_entry(void\* parameter)

{

static rt\_uint32\_t sys\_cnt = 0;

while(1)

{

rt\_thread\_delay(RT\_TICK\_PER\_SECOND); //1s

sys\_cnt++;

if(sys\_cnt >= 60\*60\*4) //6 1-5 2-10 3-20 4-30 5-1h

{

sys\_cnt = 0;

NVIC\_SystemReset();

}

}

}

int sys\_reboot\_thread\_init(void)

{

rt\_thread\_t reboot\_tid;

/\* \*/

reboot\_tid = rt\_thread\_create("reboot"

sys\_reboot\_entry,

RT\_NULL,

1024,

30,

10);

if (reboot\_tid != RT\_NULL){

rt\_thread\_startup(reboot\_tid);

}

return 0;

}

INIT\_APP\_EXPORT(sys\_reboot\_thread\_init);

int main(void) {

/\* disable interrupt first \*/

rt\_hw\_interrupt\_disable();

/\* startup RT-Thread RTOS \*/

rtthread\_startup();

return 0;

}/\*

\* File : board.h

\* This file is part of RT-Thread RTOS

\* COPYRIGHT (C) 2017, RT-Thread Development Team

\*

\* The license and distribution terms for this file may be

\* found in the file LICENSE in this distribution or at

\* http://www.rt-thread.org/license/LICENSE

\*

\* Change Logs:

\* Date Author Notes

\* 2017-12-19 xidongxu first version

\*/

#include "board.h"

#include <rtthread.h>

\*/

static uint16\_t get\_adc(u8 ch) {

ADC\_RegularChannelConfig(ADC1, ch, 1, ADC\_SampleTime\_112Cycles );

ADC\_SoftwareStartConv(ADC1);

while(!ADC\_GetFlagStatus(ADC1, ADC\_FLAG\_EOC ));

return ADC\_GetConversionValue(ADC1);

}

/\*\*

\*

\*/

uint32\_t cpu\_temp\_get(void) {

uint32\_t adc[10] = { 0 }, adc\_max = 0, adc\_min = 0;

uint32\_t adc\_value = 0, adc\_sum = 0;

double temperature = 0.0;

for (uint8\_t t = 0; t < 10; t++) {

adc[t] = get\_adc(ADC\_Channel\_16);

adc\_sum = adc\_sum + adc[t];

rt\_thread\_delay(rt\_tick\_from\_millisecond(5));

}

adc\_max = adc[0];

adc\_min = adc[0];

for (uint8\_t t = 1; t < 10; t++) {

if (adc\_max < adc[t]) {

adc\_max = adc[t];

}

if (adc\_min > adc[t]) {

adc\_min = adc[t];

}

}

adc\_value = ((adc\_sum - adc\_max - adc\_min) / 8);

temperature = (float) adc\_value \* (3.3 / 4096);

temperature = (temperature - 0.76) / 0.0025 + 25;

return (uint32\_t)(temperature);

}

\*/

int stm32\_cpu\_temperature\_init(void) {

ADC\_CommonInitTypeDef ADC\_CommonInitStructure;

ADC\_InitTypeDef ADC\_InitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_ADC1, ENABLE);

RCC\_APB2PeriphResetCmd(RCC\_APB2Periph\_ADC1, ENABLE);

RCC\_APB2PeriphResetCmd(RCC\_APB2Periph\_ADC1, DISABLE);

ADC\_TempSensorVrefintCmd(ENABLE);

ADC\_CommonInitStructure.ADC\_Mode = ADC\_Mode\_Independent;

ADC\_CommonInitStructure.ADC\_TwoSamplingDelay = ADC\_TwoSamplingDelay\_20Cycles;

ADC\_CommonInitStructure.ADC\_DMAAccessMode = ADC\_DMAAccessMode\_Disabled;

ADC\_CommonInitStructure.ADC\_Prescaler = ADC\_Prescaler\_Div4;

ADC\_CommonInit(&ADC\_CommonInitStructure);

ADC\_InitStructure.ADC\_Resolution = ADC\_Resolution\_12b;

ADC\_InitStructure.ADC\_ScanConvMode = DISABLE;

ADC\_InitStructure.ADC\_ContinuousConvMode = DISABLE;

ADC\_InitStructure.ADC\_ExternalTrigConvEdge = ADC\_ExternalTrigConvEdge\_None;

ADC\_InitStructure.ADC\_ExternalTrigConv = ADC\_ExternalTrigConv\_T1\_CC1;

ADC\_InitStructure.ADC\_DataAlign = ADC\_DataAlign\_Right;

ADC\_InitStructure.ADC\_NbrOfConversion = 1;

ADC\_Init(ADC1, &ADC\_InitStructure);

ADC\_RegularChannelConfig(ADC1, ADC\_Channel\_16, 1, ADC\_SampleTime\_112Cycles);

ADC\_Cmd(ADC1, ENABLE);

return 0;

}

INIT\_BOARD\_EXPORT(stm32\_cpu\_temperature\_init);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Copyright 2018-2023 zhuangwei@nblink-tech.com

\* FileName: drv\_ds18b20.c

\* Desc:

\*

\*

\* Author: zhuangwei

\* Date: 2018-12-15

\* Notes:

\*

\* -----------------------------------------------------------------

\* Histroy: v1.0 2018-12-15, zhuangwei create this file

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*------------------------------- Includes ----------------------------------\*/

#include <rtdevice.h>

#include "utils.h"

#include "drv\_ds18b20.h"

/\*------------------- Global Definitions and Declarations -------------------\*/

/\*---------------------- Constant / Macro Definitions -----------------------\*/

#define DS18B20\_PIN 54

/\*----------------------- Type Declarations ---------------------------------\*/

/\*----------------------- Variable Declarations -----------------------------\*/

static int temp\_value = 0;

/\*----------------------- Function Prototype --------------------------------\*/

/\*----------------------- Function Implement --------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Name: ds18b20\_rst

\*

\* Desc: DS18B20

\* Param:

\* Return:

\* Global:

\* Note:

\* Author: zhuangwei

\* -------------------------------------

\* Log: 2018/12/15, Create this function by zhuangwei

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static rt\_uint8\_t ds18b20\_rst(void)

{

rt\_pin\_mode(DS18B20\_PIN, PIN\_MODE\_OUTPUT);

rt\_pin\_write(DS18B20\_PIN, PIN\_LOW);

delay\_us(750);

rt\_pin\_write(DS18B20\_PIN, PIN\_HIGH);

delay\_us(20);

rt\_pin\_mode(DS18B20\_PIN, PIN\_MODE\_INPUT);

delay\_us(100);

return rt\_pin\_read(DS18B20\_PIN);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Name: ds18b20\_read\_byte

\*

\* Desc:

\* Param:

\* Return:

\* Global:

\* Note:

\* Author: zhuangwei

\* -------------------------------------

\* Log: 2018/12/15, Create this function by zhuangwei

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static rt\_uint8\_t ds18b20\_read\_byte(void) // read one byte

{

rt\_uint8\_t i;

rt\_uint8\_t data = 0;

for (i=0; i<8; i++)

{

rt\_pin\_mode(DS18B20\_PIN, PIN\_MODE\_OUTPUT);

rt\_pin\_write(DS18B20\_PIN, PIN\_LOW);

delay\_us(2);

rt\_pin\_write(DS18B20\_PIN, PIN\_HIGH);

rt\_pin\_mode(DS18B20\_PIN, PIN\_MODE\_INPUT);

delay\_us(12);

data >>= 1;

if(rt\_pin\_read(DS18B20\_PIN))

{

data |= 0x80;

}

delay\_us(50);

rt\_pin\_mode(DS18B20\_PIN, PIN\_MODE\_OUTPUT);

rt\_pin\_write(DS18B20\_PIN, PIN\_HIGH);

}

return data;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Name: ds18b20\_write\_byte

\*

\* Desc:

\* Param:

\* Return:

\* Global:

\* Note:

\* Author: zhuangwei

\* -------------------------------------

\* Log: 2018/12/15, Create this function by zhuangwei

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void ds18b20\_write\_byte(rt\_uint8\_t data)

{

rt\_uint8\_t i,tmp;

rt\_pin\_mode(DS18B20\_PIN, PIN\_MODE\_OUTPUT);

for (i=1; i<=8; i++)

{

tmp = data&0x01;

data = data>>1;

if(tmp)

{

rt\_pin\_write(DS18B20\_PIN, PIN\_LOW);

delay\_us(5);

rt\_pin\_write(DS18B20\_PIN, PIN\_HIGH);

delay\_us(60);

}

else

{

rt\_pin\_write(DS18B20\_PIN, PIN\_LOW);

delay\_us(60);

rt\_pin\_write(DS18B20\_PIN, PIN\_HIGH);

delay\_us(5);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Name: ds18b20\_get\_temp

\*

\* Desc:

\* Param:

\* Return:

\* Global:

\* Note:

\* Author: zhuangwei

\* -------------------------------------

\* Log: 2018/12/15, Create this function by zhuangwei

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static int ds18b20\_get\_value(void)

{

rt\_uint8\_t TL,TH;

rt\_uint16\_t temp;

float ret;

if(ds18b20\_rst()) return 0;

delay\_us(100);

ds18b20\_write\_byte(0xCC);//skip rom

ds18b20\_write\_byte(0x44);//convert

//rt\_thread\_delay(1);

if(ds18b20\_rst()) return 0;

delay\_us(100);

ds18b20\_write\_byte(0xCC);// skip rom

ds18b20\_write\_byte(0xBE);

delay\_us(10);

TL = ds18b20\_read\_byte();

TH = ds18b20\_read\_byte();

temp = (TH<<8)|TL;

if(temp&0xf000)

{

ret = (~temp+1)\*0.0625;

}

else

{

ret = temp\*0.0625;

}

return (int)ret;

}

int ds18b20\_get\_temp(void)

{

int ret;

rt\_enter\_critical();

ret = temp\_value;

rt\_exit\_critical();

return ret;

}

static void ds18b20(void)

{

rt\_uint8\_t count;

rt\_uint16\_t temp;

for(count=1; count<=3; count++)

{

temp =ds18b20\_get\_temp();

rt\_kprintf("temperature : %d\r\n", temp);

rt\_thread\_mdelay(500);

}

}

MSH\_CMD\_EXPORT(ds18b20, ds18b20 test);

void first\_get\_temperature(void)

{

char j = 0;

rt\_int32\_t value[5] = {0};

/\* 500ms \*/

rt\_thread\_delay(RT\_TICK\_PER\_SECOND/2);

rt\_enter\_critical();

for(j=0; j<5; j++)

{

rt\_thread\_delay(10);

value[j] = ds18b20\_get\_value();

}

temp\_value = Bubble(value, 5);

rt\_exit\_critical();

}

rt\_int32\_t value[5] = {0};

void ds18b20\_thread\_entry(void\* p)

{

char i = 0;

char ON\_OFF = 0; //

rt\_int32\_t last\_temp = 0;

while(1)

{

if(ON\_OFF == 0){

first\_get\_temperature();

ON\_OFF = 1;

}

else{

rt\_thread\_delay(RT\_TICK\_PER\_SECOND/5);// 0.2s 10s

value[i++] = ds18b20\_get\_value();

if(i >= 5){// 5

temp\_value = Bubble(value, 5);

i = 0;

if(temp\_value > -10 && temp\_value < 55){ //

last\_temp = temp\_value; //

}

else if(abs(temp\_value-last\_temp)>5 && last\_temp != 0 )// 5 0

{

temp\_value = last\_temp;//

}

rt\_kprintf("temp:$%d\r\n",temp\_value);

//rt\_kprintf("last\_temp:$%d\r\n",last\_temp);

}

}

}

}

//void ds18b20\_thread\_entry(void\* p)

//{

// int i = 0;

// int value[5] = {0};

// static rt\_int32\_t last\_temp = 0;

//

// while(1)

// {

// /\* 500ms \*/

// rt\_thread\_delay(RT\_TICK\_PER\_SECOND/2);

// rt\_enter\_critical();

// for(i=0; i<5; i++)

// {

// rt\_thread\_delay(1);

// value[i] = ds18b20\_get\_value();

// }

//

// temp\_value = Bubble((rt\_int32\_t \*)value, 5);

// if(temp\_value > -20 && temp\_value < 70){ //

// last\_temp = temp\_value; //

// }

// else {

// temp\_value = last\_temp;//

// }

// rt\_exit\_critical();

// }

//}

static int ds18b20\_init(void)

{

rt\_thread\_t thread = RT\_NULL;

/\* Create background ticks thread \*/

thread = rt\_thread\_create("ds18b20", ds18b20\_thread\_entry, RT\_NULL, 1024, 30, 10);

if(thread == RT\_NULL)

{

return RT\_ERROR;

}

rt\_thread\_startup(thread);

return 0;

}

INIT\_DEVICE\_EXPORT(ds18b20\_init);

/\*---------------------------------------------------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Copyright 2018-2023 zhuangwei@nblink-tech.com

\* FileName: drv\_hx711.c

\* Desc:

\*

\*

\* Author: zhuangwei

\* Date: 2018-12-19

\* Notes:

\*

\* -----------------------------------------------------------------

\* Histroy: v1.0 2018-12-19, zhuangwei create this file

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*------------------------------- Includes ----------------------------------\*/

#define LOG\_TAG "drv\_hx711"

#include <rtdevice.h>

#include "utils.h"

#include "drv\_hx711.h"

#include <elog.h>

#include "multi\_button.h"

#include "easyflash.h"

#include <stdio.h>

#include <stdlib.h>

/\*------------------- Global Definitions and Declarations -------------------\*/

/\*---------------------- Constant / Macro Definitions -----------------------\*/

#define FLASH\_INIT\_ADDRESS (4096) //

#define HX711\_DT1\_PIN 10

#define HX711\_SCK1\_PIN 11

#define HX711\_DT2\_PIN 14

#define HX711\_SCK2\_PIN 15

#define HX711\_LED\_PIN 55

#define HX711\_KEY\_PIN 28

#define HX711\_DT(pin) rt\_pin\_read(pin)

#define HX711\_SCK\_HIGH(pin) rt\_pin\_write(pin, PIN\_HIGH)

#define HX711\_SCK\_LOW(pin) rt\_pin\_write(pin, PIN\_LOW)

#define HX711\_LED\_ON rt\_pin\_write(HX711\_LED\_PIN, PIN\_HIGH)

#define HX711\_LED\_OFF rt\_pin\_write(HX711\_LED\_PIN, PIN\_LOW)

/\*----------------------- Type Declarations ---------------------------------\*/

/\*----------------------- Variable Declarations -----------------------------\*/

static struct button btn;

uint32\_t var\_min[2]={999999999,999999999}; // -2147483648 +2141483647

float weight = 0;

float side\_weight[2];

double Normalized[2]={0}; // float double

rt\_int32\_t Reverse\_Normalized[2] = {0}; // flash

static int key\_status = 0;

/\*----------------------- Function Prototype --------------------------------\*/

/\*----------------------- Function Implement --------------------------------\*/

int flash\_read\_normalized(void)

{

char string[100];

ef\_port\_read(FLASH\_INIT\_ADDRESS+0,(rt\_int32\_t \*)&Reverse\_Normalized[0],4);

ef\_port\_read(FLASH\_INIT\_ADDRESS+4,(rt\_int32\_t \*)&Reverse\_Normalized[1],4);

ef\_port\_read(FLASH\_INIT\_ADDRESS+8 ,&var\_min[0],4);

ef\_port\_read(FLASH\_INIT\_ADDRESS+12,&var\_min[1],4);

Normalized[0] = (double)1 / Reverse\_Normalized[0] ;

Normalized[1] = (double)1 / Reverse\_Normalized[1] ;

// string

sprintf(string,"Normalized\*100: %f , %f ",Normalized[0]\*100,Normalized[1]\*100);

log\_i(string);

sprintf(string,"var\_min:%ld , %ld ",var\_min[0],var\_min[1]);

log\_i(string);

return 0;

}

void flash\_writer\_normalized(void)

{

ef\_port\_erase(FLASH\_INIT\_ADDRESS,4); // 4096 Byte

ef\_port\_write(FLASH\_INIT\_ADDRESS+0,(uint32\_t \*)&Reverse\_Normalized[0],4);// FLASH\_INIT\_ADDRESS+0

ef\_port\_write(FLASH\_INIT\_ADDRESS+4,(uint32\_t \*)&Reverse\_Normalized[1],4);

ef\_port\_write(FLASH\_INIT\_ADDRESS+8 ,&var\_min[0],4);

ef\_port\_write(FLASH\_INIT\_ADDRESS+12,&var\_min[1],4);

}

rt\_uint32\_t hx711\_read(int pin\_dat, int pin\_sck)

{

unsigned char i;

unsigned int value;

HX711\_SCK\_LOW(pin\_sck);// AD

value=0;

while(HX711\_DT(pin\_dat));//AD ,

for(i=0; i<24; i++)

{

HX711\_SCK\_HIGH(pin\_sck);//SCK ( )

value = value<<1;// Count ,

HX711\_SCK\_LOW(pin\_sck);//SCK

if(HX711\_DT(pin\_dat)) value++;

}

HX711\_SCK\_HIGH(pin\_sck);

value = value^0x800000;// 25 ,

HX711\_SCK\_LOW(pin\_sck);

return value;

}

static void first\_hx711\_get\_ad(rt\_int32\_t \*ad)//

{

rt\_int32\_t var0[5];

rt\_int32\_t var1[5];

rt\_uint8\_t i = 0;

if(!ad) return;

for(i=0; i<5; i++)

{

rt\_thread\_delay(RT\_TICK\_PER\_SECOND/10);// 100ms

var0[i] = hx711\_read(HX711\_DT1\_PIN, HX711\_SCK1\_PIN);

var1[i] = hx711\_read(HX711\_DT2\_PIN, HX711\_SCK2\_PIN);

}

ad[0] = Bubble(var0, 5);

ad[1] = Bubble(var1, 5);

}

static void hx711\_get\_ad(rt\_int32\_t \*ad)

{

rt\_int32\_t var0[5];

rt\_int32\_t var1[5];

uint8\_t i = 0;

if(!ad) return;

for(i=0; i<5; i++)

{

var0[i] = hx711\_read(HX711\_DT1\_PIN, HX711\_SCK1\_PIN);

var1[i] = hx711\_read(HX711\_DT2\_PIN, HX711\_SCK2\_PIN);

//rt\_kprintf("var: %d , %d\n",var0[i],var1[i]);

}

ad[0] = Bubble(var0, 5);

ad[1] = Bubble(var1, 5);

//rt\_kprintf("ad: %d , %d\n",ad[0],ad[1]);

}

void hx711\_get\_min(void)

{

rt\_int32\_t temp\_res[2]={0};

uint8\_t j = 5;

while(j--)

{

hx711\_get\_ad(temp\_res);

rt\_kprintf("temp\_res: %ld , %ld \n",temp\_res[0],temp\_res[1]);

if(var\_min[0] >= temp\_res[0] || var\_min[0] < 800000 ) {// var\_min 800000 ,var\_min

var\_min[0] = temp\_res[0];

}

if(var\_min[1] >= temp\_res[1] || var\_min[1] < 800000 ) {

var\_min[1] = temp\_res[1];

}

rt\_kprintf("var\_min: %ld , %ld\n",var\_min[0],var\_min[1]);

}

log\_i("var\_min: %ld, %ld ",var\_min[0],var\_min[1]);

}

void hx711\_normalized(void)

{

rt\_int32\_t temp[2]={0};

hx711\_get\_ad(temp);

Reverse\_Normalized[0]= abs(temp[0]-var\_min[0]);

Reverse\_Normalized[1] = abs(temp[1]-var\_min[1]);

Normalized[0] =(double) 1/(temp[0]-var\_min[0]);

Normalized[1] =(double) 1/(temp[1]-var\_min[1]);

flash\_writer\_normalized();//

log\_i("FLASH Write Reverse\_Normalized: %d, %d ",Reverse\_Normalized[0],Reverse\_Normalized[1]);

}

float hx711\_weight\_get(void)

{

float ret;

rt\_enter\_critical();

ret = weight;

rt\_exit\_critical();

return ret;

}

float hx711\_weight\_update(void)

{

rt\_int32\_t AD\_Value[2] = {0};

static uint8\_t count = 0;

static float last\_weight = 0;

static char ON\_OFF = 0;

while(1)

{

if(0 == ON\_OFF){

first\_hx711\_get\_ad(AD\_Value);

ON\_OFF = 1;

}

else{

hx711\_get\_ad(AD\_Value);

}

side\_weight[0] = (float)abs(AD\_Value[0]-var\_min[0])/abs(Reverse\_Normalized[0]);

side\_weight[1] = (float)abs(AD\_Value[1]-var\_min[1])/abs(Reverse\_Normalized[1]);

//rt\_kprintf("--- %ld --- %ld ---\n",abs(AD\_Value[0]-var\_min[0]),abs(AD\_Value[1]-var\_min[1]));

if ( ((side\_weight[0] > 50 || side\_weight[1] > 50) || (side\_weight[0] < 0 || side\_weight[1] < 0)) && count <= 5) // 50kg

{

count ++;

//rt\_kprintf("count = %d \n",count);

continue;// 6

}

else if(count > 5){ //

count = 0;

rt\_kprintf("last\_weight = %d \n",last\_weight);

return last\_weight;//

}

last\_weight = (side\_weight[0]+side\_weight[1])/2;

return (side\_weight[0]+side\_weight[1])/2; //

}

}

void hx711\_key\_callback(void \*btn)

{

uint32\_t btn\_event\_val;

btn\_event\_val = get\_button\_event((struct button \*)btn);

if(btn\_event\_val == SINGLE\_CLICK)

{

switch(key\_status)

{

case 0://

hx711\_get\_min();

HX711\_LED\_ON;

key\_status = 1;

log\_i("please put 1KG");

break;

case 1://

hx711\_normalized();//

HX711\_LED\_OFF;

key\_status = 0;

log\_i("hx711 normalized");

break;

default:

break;

}

}

}

static uint8\_t hx711\_key\_read(void)

{

return rt\_pin\_read(HX711\_KEY\_PIN);

}

void hx711\_key\_thread\_entry(void\* p)

{

int cnt = 200\*55;

while(1)

{

/\* 5ms \*/

rt\_thread\_delay(RT\_TICK\_PER\_SECOND/200);

button\_ticks();

cnt++;

if(cnt == 200\*60)//60s

{

rt\_enter\_critical();

weight = hx711\_weight\_update();

rt\_exit\_critical();

cnt = 0;

}

}

}

static int hx711\_init(void)

{

rt\_pin\_mode(HX711\_DT1\_PIN, PIN\_MODE\_INPUT\_PULLUP);

rt\_pin\_mode(HX711\_SCK1\_PIN, PIN\_MODE\_OUTPUT);

rt\_pin\_write(HX711\_SCK1\_PIN, PIN\_HIGH);

rt\_pin\_mode(HX711\_DT2\_PIN, PIN\_MODE\_INPUT\_PULLUP);

rt\_pin\_mode(HX711\_SCK2\_PIN, PIN\_MODE\_OUTPUT);

rt\_pin\_write(HX711\_SCK2\_PIN, PIN\_HIGH);

rt\_pin\_mode(HX711\_LED\_PIN, PIN\_MODE\_OUTPUT);

HX711\_LED\_OFF;

rt\_pin\_mode(HX711\_KEY\_PIN, PIN\_MODE\_INPUT);

button\_init(&btn, hx711\_key\_read, PIN\_LOW);

button\_attach(&btn, SINGLE\_CLICK, hx711\_key\_callback);

button\_start (&btn);

rt\_thread\_t thread = RT\_NULL;

/\* Create background ticks thread \*/

thread = rt\_thread\_create("hx711", hx711\_key\_thread\_entry, RT\_NULL, 1024, 30, 10);

if(thread == RT\_NULL)

{

return RT\_ERROR;

}

rt\_thread\_startup(thread);

return 0;

}

INIT\_DEVICE\_EXPORT(hx711\_init);

/\*---------------------------------------------------------------------------\*/

/\* Get weight \*/

void get\_weight\_hx711(void)

{

// string

char str[100];

float temp\_weight = 0;

temp\_weight = hx711\_weight\_update();

sprintf(str,"Weight:%f \n",temp\_weight);

rt\_kprintf(str);

sprintf(str,"weight1:%f weight2:%f\n",side\_weight[0],side\_weight[1]);

rt\_kprintf(str);

}

MSH\_CMD\_EXPORT(get\_weight\_hx711,get\_weight\_hx711[a]);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Copyright 2018-2023 zhuangwei@nblink-tech.com

\* FileName: drv\_relay.c

\* Desc:

\*

\*

\* Author: zhuangwei

\* Date: 2018-12-15

\* Notes:

\*

\* -----------------------------------------------------------------

\* Histroy: v1.0 2018-12-15, zhuangwei create this file

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*------------------------------- Includes ----------------------------------\*/

#define LOG\_TAG "drv\_relay"

#include <rtdevice.h>

#include "drv\_relay.h"

#include <elog.h>

/\*------------------- Global Definitions and Declarations -------------------\*/

typedef struct

{

char \*name;

rt\_base\_t pin;

rt\_timer\_t timer;

int status;

}relay\_t;

/\*---------------------- Constant / Macro Definitions -----------------------\*/

static relay\_t relay\_table[] = {

{"water", 62, RT\_NULL, 0},

{"feed", 61, RT\_NULL, 0},

{"heat1", 4, RT\_NULL, 0},

{"heat2", 9, RT\_NULL, 0}

};

/\*----------------------- Type Declarations ---------------------------------\*/

/\*----------------------- Variable Declarations -----------------------------\*/

/\*----------------------- Function Prototype --------------------------------\*/

/\*----------------------- Function Implement --------------------------------\*/

static void timeout\_handler(void\* parameter)

{

relay\_t \*relay\_ptr = (relay\_t\*)parameter;

relay\_ptr->status = relay\_ptr->status?0:1;

rt\_pin\_write(relay\_ptr->pin, relay\_ptr->status);

relay\_ptr->timer = RT\_NULL;

}

int relay\_control(relay\_dev\_t dev, int expect\_status, int hold\_s)

{

if(dev >= RELAY\_MAX) return -1;

if(relay\_table[dev].status == expect\_status) return 0;

if(relay\_table[dev].timer)

{

if(relay\_table[dev].status == expect\_status)

return 0;

else

return -1;

}

else

{

rt\_pin\_write(relay\_table[dev].pin, expect\_status);

relay\_table[dev].status = expect\_status;

if(hold\_s > 0)

{

relay\_table[dev].timer = rt\_timer\_create(relay\_table[dev].name, \

timeout\_handler, &relay\_table[dev], rt\_tick\_from\_millisecond(hold\_s\*1000), \

RT\_TIMER\_FLAG\_ONE\_SHOT);

if (relay\_table[dev].timer != RT\_NULL) rt\_timer\_start(relay\_table[dev].timer);

}

}

return 0;

}

static int relay\_init(void)

{

int i;

for(i=0; i<RELAY\_MAX; i++)

{

rt\_pin\_mode(relay\_table[i].pin, PIN\_MODE\_OUTPUT);

rt\_pin\_write(relay\_table[i].pin, relay\_table[i].status);

}

return 0;

}

INIT\_DEVICE\_EXPORT(relay\_init);

/\*---------------------------------------------------------------------------\*/

/\*

\* Copyright (c) 2013, Institute for Pervasive Computing, ETH Zurich

\* All rights reserved.

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\* 1. Redistributions of source code must retain the above copyright

\* notice, this list of conditions and the following disclaimer.

\* 2. Redistributions in binary form must reproduce the above copyright

\* notice, this list of conditions and the following disclaimer in the

\* documentation and/or other materials provided with the distribution.

\* 3. Neither the name of the Institute nor the names of its contributors

\* may be used to endorse or promote products derived from this software

\* without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ``AS IS'' AND

\* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE

\* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE

\* ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE

\* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL

\* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS

\* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)

\* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT

\* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY

\* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF

\* SUCH DAMAGE.

\*

\* This file is part of the Contiki operating system.

\*/

/\*\*

\* \file

\* Erbium (Er) CoAP client example.

\* \author

\* Matthias Kovatsch <kovatsch@inf.ethz.ch>

\*/

#define LOG\_TAG "coap\_client"

#include <elog.h>

#include "rtthread.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "contiki.h"

#include "contiki-net.h"

#include "er-coap-engine.h"

#include "net/rpl/rpl.h"

#include "net/ip/uip.h"

#include "string.h"

#define DEBUG 0

#if DEBUG

#include <stdio.h>

#define PRINTF(...) log\_i(\_\_VA\_ARGS\_\_)

#define PRINT6ADDR(addr) PRINTF("[%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x]", ((uint8\_t \*)addr)[0], ((uint8\_t \*)addr)[1], ((uint8\_t \*)addr)[2], ((uint8\_t \*)addr)[3], ((uint8\_t \*)addr)[4], ((uint8\_t \*)addr)[5], ((uint8\_t \*)addr)[6], ((uint8\_t \*)addr)[7], ((uint8\_t \*)addr)[8], ((uint8\_t \*)addr)[9], ((uint8\_t \*)addr)[10], ((uint8\_t \*)addr)[11], ((uint8\_t \*)addr)[12], ((uint8\_t \*)addr)[13], ((uint8\_t \*)addr)[14], ((uint8\_t \*)addr)[15])

#define PRINTLLADDR(lladdr) PRINTF("[%02x:%02x:%02x:%02x:%02x:%02x]", (lladdr)->addr[0], (lladdr)->addr[1], (lladdr)->addr[2], (lladdr)->addr[3], (lladdr)->addr[4], (lladdr)->addr[5])

#else

#define PRINTF(...)

#define PRINT6ADDR(addr)

#define PRINTLLADDR(addr)

#endif

/\* FIXME: This server address is hard-coded for Cooja and link-local for unconnected border router. \*/

//#define SERVER\_NODE(ipaddr) uip\_ip6addr(ipaddr, 0xbbbb, 0, 0, 0, 0, 0, 0, 0x100) /\* cooja2 \*/

//#define SERVER\_NODE(ipaddr) uip\_ip6addr(ipaddr, 0xbbbb, 0, 0, 0, 0xe581, 0xc232, 0x64b3, 0x6d5c)

#define SERVER\_NODE(ipaddr) uip\_ip6addr(ipaddr, 0xbbbb, 0, 0, 0, 0x42d6, 0x3cff, 0xfe02, 0x2082)

#define LOCAL\_PORT UIP\_HTONS(COAP\_DEFAULT\_PORT + 1)

#define REMOTE\_PORT UIP\_HTONS(COAP\_DEFAULT\_PORT)

#define TOGGLE\_INTERVAL 10

PROCESS(er\_example\_client, "Erbium Example Client");

//AUTOSTART\_PROCESSES(&er\_example\_client);

uip\_ipaddr\_t server\_ipaddr;

static struct etimer et;

/\* Example URIs that can be queried. \*/

#define NUMBER\_OF\_URLS 4

/\* leading and ending slashes only for demo purposes, get cropped automatically when setting the Uri-Path \*/

char \*service\_urls[NUMBER\_OF\_URLS] =

{ ".well-known/core", "online", "battery/", "error/in//path" };

/\* This function is will be passed to COAP\_BLOCKING\_REQUEST() to handle responses. \*/

void client\_chunk\_handler(void \*response) {

const uint8\_t \*chunk;

int len = coap\_get\_payload(response, &chunk);

}

PROCESS\_THREAD(er\_example\_client, ev, data) {

PROCESS\_BEGIN();

static char msg[24];

static coap\_packet\_t request[1]; /\* This way the packet can be treated as pointer as usual. \*/

#ifdef auto\_find\_server

/\* \*/

static uip\_ip6addr\_t \*globaladdr = NULL;

static rpl\_dag\_t \*dag;

uip\_ds6\_addr\_t \*addr\_desc = uip\_ds6\_get\_global(ADDR\_PREFERRED);

if(addr\_desc != NULL) {

globaladdr = &addr\_desc->ipaddr;

dag = rpl\_get\_any\_dag();

if(dag) {

uip\_ipaddr\_copy(&server\_ipaddr, globaladdr);

memcpy(&server\_ipaddr.u8[8], &dag->dag\_id.u8[8], sizeof(uip\_ipaddr\_t) / 2);

} else {

SERVER\_NODE(&server\_ipaddr);

}

}

#else

SERVER\_NODE(&server\_ipaddr);

#endif

sprintf(msg, "%02x:%02x:%02x:%02x:%02x:%02x:%02x:%02x", linkaddr\_node\_addr.u8[0], linkaddr\_node\_addr.u8[1],

linkaddr\_node\_addr.u8[2], linkaddr\_node\_addr.u8[3], linkaddr\_node\_addr.u8[4],

linkaddr\_node\_addr.u8[5], linkaddr\_node\_addr.u8[6], linkaddr\_node\_addr.u8[7]);

/\* receives all CoAP messages \*/

coap\_init\_engine();

etimer\_set(&et, TOGGLE\_INTERVAL \* CLOCK\_SECOND);

while (1) {

PROCESS\_YIELD();

if (etimer\_expired(&et)) {

PRINTF("--Toggle timer--");

/\* prepare request, TID is set by COAP\_BLOCKING\_REQUEST() \*/

coap\_init\_message(request, COAP\_TYPE\_CON, COAP\_POST, 0);

coap\_set\_header\_uri\_path(request, service\_urls[1]);

coap\_set\_payload(request, (uint8\_t \*) msg, sizeof(msg) - 1);

PRINT6ADDR(&server\_ipaddr);

PRINTF(" : %u", UIP\_HTONS(REMOTE\_PORT));

COAP\_BLOCKING\_REQUEST(&server\_ipaddr, REMOTE\_PORT, request, client\_chunk\_handler);

PRINTF("--Done--");

etimer\_reset(&et);

}

}

PROCESS\_END();

}

/\*

\* Copyright (c) 2013, Institute for Pervasive Computing, ETH Zurich

\* All rights reserved.

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\* 1. Redistributions of source code must retain the above copyright

\* notice, this list of conditions and the following disclaimer.

\* 2. Redistributions in binary form must reproduce the above copyright

\* notice, this list of conditions and the following disclaimer in the

\* documentation and/or other materials provided with the distribution.

\* 3. Neither the name of the Institute nor the names of its contributors

\* may be used to endorse or promote products derived from this software

\* without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE INSTITUTE AND CONTRIBUTORS ``AS IS'' AND

\* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE

\* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE

\* ARE DISCLAIMED. IN NO EVENT SHALL THE INSTITUTE OR CONTRIBUTORS BE LIABLE

\* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL

\* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS

\* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)

\* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT

\* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY

\* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF

\* SUCH DAMAGE.

\*

\* This file is part of the Contiki operating system.

\*/

/\*\*

\* \file

\* Erbium (Er) REST Engine example.

\* \author

\* Matthias Kovatsch <kovatsch@inf.ethz.ch>

\*/

#define LOG\_TAG "coap\_example"

#define LOG\_LVL ELOG\_LVL\_INFO

#include <elog.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <contiki.h>

#include <contiki-net.h>

#include "rest-engine.h"

#if PLATFORM\_HAS\_BUTTON

#include "dev/button-sensor.h"

#endif

#undef DEBUG

#define DEBUG 0

#if DEBUG

#include <stdio.h>

#define PRINTF(...) printf(\_\_VA\_ARGS\_\_)

#define PRINT6ADDR(addr) PRINTF("[%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x]", ((uint8\_t \*)addr)[0], ((uint8\_t \*)addr)[1], ((uint8\_t \*)addr)[2], ((uint8\_t \*)addr)[3], ((uint8\_t \*)addr)[4], ((uint8\_t \*)addr)[5], ((uint8\_t \*)addr)[6], ((uint8\_t \*)addr)[7], ((uint8\_t \*)addr)[8], ((uint8\_t \*)addr)[9], ((uint8\_t \*)addr)[10], ((uint8\_t \*)addr)[11], ((uint8\_t \*)addr)[12], ((uint8\_t \*)addr)[13], ((uint8\_t \*)addr)[14], ((uint8\_t \*)addr)[15])

#define PRINTLLADDR(lladdr) PRINTF("[%02x:%02x:%02x:%02x:%02x:%02x]", (lladdr)->addr[0], (lladdr)->addr[1], (lladdr)->addr[2], (lladdr)->addr[3], (lladdr)->addr[4], (lladdr)->addr[5])

#else

#define PRINTF(...)

#define PRINT6ADDR(addr)

#define PRINTLLADDR(addr)

#endif

/\*

\* Resources to be activated need to be imported through the extern keyword.

\* The build system automatically compiles the resources in the corresponding sub-directory.

\*/

extern resource\_t

res\_hello,

res\_cpu\_usage,

res\_cpu\_temperature,

res\_temperature,

res\_water,

res\_feed,

res\_heat,

res\_weight,

res\_mirror,

res\_chunks,

res\_separate,

res\_push,

res\_event,

res\_sub,

res\_b1\_sep\_b2;

#if PLATFORM\_HAS\_LEDS

extern resource\_t res\_leds, res\_toggle;

#endif

#if PLATFORM\_HAS\_LIGHT

#include "dev/light-sensor.h"

extern resource\_t res\_light;

#endif

#if PLATFORM\_HAS\_BATTERY

#include "dev/battery-sensor.h"

extern resource\_t res\_battery;

#endif

/\*

extern resource\_t res\_battery;

#endif

#if PLATFORM\_HAS\_RADIO

#include "dev/radio-sensor.h"

extern resource\_t res\_radio;

#endif

#if PLATFORM\_HAS\_SHT11

#include "dev/sht11/sht11-sensor.h"

extern resource\_t res\_sht11;

#endif

\*/

PROCESS(er\_example\_server, "Erbium Example Server");

PROCESS\_THREAD(er\_example\_server, ev, data)

{

PROCESS\_BEGIN();

PROCESS\_PAUSE();

#ifdef RF\_CHANNEL

log\_d("RF channel: %u", RF\_CHANNEL);

#endif

#ifdef IEEE802154\_PANID

log\_d("PAN ID: 0x%04X", IEEE802154\_PANID);

#endif

log\_d("uIP buffer: %u", UIP\_BUFSIZE);

log\_d("LL header: %u", UIP\_LLH\_LEN);

log\_d("IP+UDP header: %u", UIP\_IPUDPH\_LEN);

log\_d("REST max chunk: %u", REST\_MAX\_CHUNK\_SIZE);

/\* Initialize the REST engine. \*/

rest\_init\_engine();

log\_i("Start Erbium CoAP Server success");

/\*

\* Bind the resources to their Uri-Path.

\* WARNING: Activating twice only means alternate path, not two instances!

\* All static variables are the same for each URI path.

\*/

rest\_activate\_resource(&res\_hello, "test/hello");

rest\_activate\_resource(&res\_cpu\_usage, "test/cpu\_usage");

rest\_activate\_resource(&res\_cpu\_temperature, "test/cpu\_temperature");

rest\_activate\_resource(&res\_temperature, "sensors/temperature");

rest\_activate\_resource(&res\_water, "control/water");

rest\_activate\_resource(&res\_feed, "control/feed");

rest\_activate\_resource(&res\_heat, "control/heat");

rest\_activate\_resource(&res\_weight, "sensors/weight");

/\* rest\_activate\_resource(&res\_mirror, "debug/mirror"); \*/

/\* rest\_activate\_resource(&res\_chunks, "test/chunks"); \*/

/\* rest\_activate\_resource(&res\_separate, "test/separate"); \*/

/\* rest\_activate\_resource(&res\_push, "test/push"); \*/

/\* rest\_activate\_resource(&res\_event, "sensors/button"); \*/

/\* rest\_activate\_resource(&res\_sub, "test/sub"); \*/

/\* rest\_activate\_resource(&res\_b1\_sep\_b2, "test/b1sepb2"); \*/

#if PLATFORM\_HAS\_LEDS

/\* rest\_activate\_resource(&res\_leds, "actuators/leds"); \*/

rest\_activate\_resource(&res\_toggle, "actuators/toggle");

#endif

#if PLATFORM\_HAS\_LIGHT

rest\_activate\_resource(&res\_light, "sensors/light");

SENSORS\_ACTIVATE(light\_sensor);

#endif

#if PLATFORM\_HAS\_BATTERY

rest\_activate\_resource(&res\_battery, "sensors/battery");

SENSORS\_ACTIVATE(battery\_sensor);

#endif

/\*

#if PLATFORM\_HAS\_RADIO

rest\_activate\_resource(&res\_radio, "sensors/radio");

SENSORS\_ACTIVATE(radio\_sensor);

#endif

#if PLATFORM\_HAS\_SHT11

rest\_activate\_resource(&res\_sht11, "sensors/sht11");

SENSORS\_ACTIVATE(sht11\_sensor);

#endif

\*/

/\* Define application-specific events here. \*/

while(1) {

PROCESS\_WAIT\_EVENT();

#if PLATFORM\_HAS\_BUTTON

if(ev == sensors\_event && data == &button\_sensor) {

log\_d("\*\*\*\*\*\*\*BUTTON\*\*\*\*\*\*\*\n");

/\* Call the event\_handler for this application-specific event. \*/

res\_event.trigger();

/\* Also call the separate response example handler. \*/

res\_separate.resume();

}

#endif /\* PLATFORM\_HAS\_BUTTON \*/

} /\* while (1) \*/

PROCESS\_END();

}

/\*

\* This file is part of the ART-6LoWPAN.

\*

\* Copyright (c) 2017, xidongxu, <xi\_dongxu@126.com>

\*

\* Permission is hereby granted, free of charge, to any person obtaining

\* a copy of this software and associated documentation files (the

\* 'Software'), to deal in the Software without restriction, including

\* without limitation the rights to use, copy, modify, merge, publish,

\* distribute, sublicense, and/or sell copies of the Software, and to

\* permit persons to whom the Software is furnished to do so, subject to

\* the following conditions:

\*

\* The above copyright notice and this permission notice shall be

\* included in all copies or substantial portions of the Software.

\*

\* THE SOFTWARE IS PROVIDED 'AS IS', WITHOUT WARRANTY OF ANY KIND,

\* EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF

\* MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.

\* IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY

\* CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,

\* TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE

\* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

\*

\* Function: Example of get the cpu inside temperature resource.

\* Created on: 2017-12-08

\*/

/\*\*

\* \file

\* Example of get the cpu inside temperature resource

\*/

#include "contiki.h"

#include <limits.h>

#include <stdlib.h>

#include <string.h>

#include "rest-engine.h"

#include <drv\_cpu\_temp.h>

static void res\_get\_handler(void \*request, void \*response, uint8\_t \*buffer, uint16\_t preferred\_size, int32\_t \*offset);

RESOURCE(res\_cpu\_temperature,

"title=\"CPU-temperature\";rt=\"CPU-temperature\"",

res\_get\_handler,

NULL,

NULL,

NULL);

static void res\_get\_handler(void \*request, void \*response, uint8\_t \*buffer, uint16\_t preferred\_size, int32\_t \*offset) {

uint8\_t length = 0;

char temperature[10] = { 0 }, temperature\_major[5] = { 0 };

snprintf(temperature\_major, sizeof(temperature\_major), "%d", cpu\_temp\_get());

strncat(temperature, temperature\_major, sizeof(temperature\_major));

memcpy(buffer, temperature, sizeof(temperature));

length = strlen((char \*) buffer);

REST.set\_header\_content\_type(response, REST.type.TEXT\_PLAIN);

REST.set\_header\_etag(response, (uint8\_t \*) &length, 1);

REST.set\_response\_payload(response, (uint8\_t \*) buffer, length);

}

\* Example of get the cpu usage resource

\*/

#include "contiki.h"

#include <limits.h>

#include <stdlib.h>

#include <string.h>

#include "rest-engine.h"

#include "rtthread.h"

#include <cpuusage.h>

static void res\_get\_handler(void \*request, void \*response, uint8\_t \*buffer, uint16\_t preferred\_size, int32\_t \*offset);

static uint8\_t cpu\_usage\_major, cpu\_usage\_minor;

RESOURCE(res\_cpu\_usage,

"title=\"CPU-usage\";rt=\"CPU-usage\";obs",

res\_get\_handler,

NULL,

NULL,

NULL);

static void res\_get\_handler(void \*request, void \*response, uint8\_t \*buffer, uint16\_t preferred\_size, int32\_t \*offset) {

uint8\_t length = 0;

char usage[10] = { 0 }, usage\_major[5] = { 0 }, usage\_minor[5] = { 0 };

cpu\_usage\_get(&cpu\_usage\_major, &cpu\_usage\_minor);

snprintf(usage\_major, sizeof(usage\_major), "%d", cpu\_usage\_major);

snprintf(usage\_minor, sizeof(usage\_minor), "%d", cpu\_usage\_minor);

strncat(usage, usage\_major, sizeof(usage\_major));

strncat(usage, ".", strlen("."));

strncat(usage, usage\_minor, sizeof(usage\_minor));

memcpy(buffer, usage, sizeof(usage));

length = strlen((char \*) buffer);

REST.set\_header\_content\_type(response, REST.type.TEXT\_PLAIN);

REST.set\_header\_etag(response, (uint8\_t \*) &length, 1);

REST.set\_response\_payload(response, (uint8\_t \*) buffer, length);

}

\*/

#define LOG\_TAG "feed"

#include "contiki.h"

#include <string.h>

#include "rest-engine.h"

#include <elog.h>

#include "cJSON.h"

#include "drv\_relay.h"

static void res\_post\_put\_handler(void \*request, void \*response, uint8\_t \*buffer, uint16\_t preferred\_size, int32\_t \*offset);

/\*A simple actuator example, depending on the color query parameter and post variable mode, corresponding led is activated or deactivated\*/

RESOURCE(res\_feed,

"title=\"feed\";rt=\"Control\"",

NULL,

res\_post\_put\_handler,

res\_post\_put\_handler,

NULL);

static void

res\_post\_put\_handler(void \*request, void \*response, uint8\_t \*buffer, uint16\_t preferred\_size, int32\_t \*offset)

{

size\_t len = 0;

uint8\_t \*payload = 0;

unsigned int accept = -1;

unsigned int content\_format = -1;

int ack = -1;

REST.get\_header\_accept(request, &accept);

REST.get\_header\_content\_type(request, &content\_format);

if(content\_format == REST.type.APPLICATION\_JSON)

{

if(accept == REST.type.APPLICATION\_JSON)

{

len = REST.get\_request\_payload(request,&payload);

if(len)

{

cJSON \*root = cJSON\_Parse(payload);

if(root)

{

cJSON \*mode = cJSON\_GetObjectItem(root, "mode");

if(mode)

{

log\_i("mode:%s",mode->valuestring);

if(strncmp(mode->valuestring, "on", 2) == 0)

{

cJSON \*delay = cJSON\_GetObjectItem(root, "delay");

if(delay)

{

log\_i("delay:%d",delay->valueint);

relay\_control(RELAY\_FEED, 1, delay->valueint);

ack = 0;

}

}

else if(strncmp(mode->valuestring, "off", 3) == 0)

{

relay\_control(RELAY\_FEED, 0, -1);

ack = 0;

}

}

cJSON\_Delete(root);

}

}

}

}

else

{

REST.set\_response\_status(response, REST.status.BAD\_REQUEST);

return;

}

REST.set\_header\_content\_type(response, REST.type.APPLICATION\_JSON);

snprintf((char \*)buffer, REST\_MAX\_CHUNK\_SIZE, "{\"ack\":%d}", ack);

REST.set\_response\_payload(response, buffer, strlen((char \*)buffer));

}

#define LOG\_TAG "heat"

#include "contiki.h"

#include <string.h>

#include "rest-engine.h"

#include <elog.h>

#include "cJSON.h"

#include "drv\_relay.h"

static void res\_post\_put\_handler(void \*request, void \*response, uint8\_t \*buffer, uint16\_t preferred\_size, int32\_t \*offset);

/\*A simple actuator example, depending on the color query parameter and post variable mode, corresponding led is activated or deactivated\*/

RESOURCE(res\_heat,

"title=\"heat\";rt=\"Control\"",

NULL,

res\_post\_put\_handler,

res\_post\_put\_handler,

NULL);

static void

res\_post\_put\_handler(void \*request, void \*response, uint8\_t \*buffer, uint16\_t preferred\_size, int32\_t \*offset)

{

size\_t len = 0;

uint8\_t \*payload = 0;

unsigned int accept = -1;

unsigned int content\_format = -1;

int ack = -1;

REST.get\_header\_accept(request, &accept);

REST.get\_header\_content\_type(request, &content\_format);

if(content\_format == REST.type.APPLICATION\_JSON)

{

if(accept == REST.type.APPLICATION\_JSON)

{

len = REST.get\_request\_payload(request,&payload);

if(len)

{

cJSON \*root = cJSON\_Parse(payload);

if(root)

{

cJSON \*mode = cJSON\_GetObjectItem(root, "mode");

if(mode)

{

log\_i("mode:%s",mode->valuestring);

if(strncmp(mode->valuestring, "on", 2) == 0)

{

cJSON \*delay = cJSON\_GetObjectItem(root, "delay");

if(delay)

{

log\_i("delay:%d",delay->valueint);

relay\_control(RELAY\_HEAT1, 1, delay->valueint);

relay\_control(RELAY\_HEAT2, 1, delay->valueint);

ack = 0;

}

}

else if(strncmp(mode->valuestring, "off", 3) == 0)

{

relay\_control(RELAY\_HEAT1, 0, -1);

relay\_control(RELAY\_HEAT2, 0, -1);

ack = 0;

}

}

cJSON\_Delete(root);

}

}

}

}

else

{

REST.set\_response\_status(response, REST.status.BAD\_REQUEST);

return;

}

REST.set\_header\_content\_type(response, REST.type.APPLICATION\_JSON);

snprintf((char \*)buffer, REST\_MAX\_CHUNK\_SIZE, "{\"ack\":%d}", ack);

REST.set\_response\_payload(response, buffer, strlen((char \*)buffer));

}

#include "contiki.h"

#include <limits.h>

#include <stdlib.h>

#include <string.h>

#include "rest-engine.h"

#include "drv\_ds18b20.h"

static void res\_get\_handler(void \*request, void \*response, uint8\_t \*buffer, uint16\_t preferred\_size, int32\_t \*offset);

static void res\_periodic\_handler(void);

#define MAX\_AGE 60 // 60s

#define INTERVAL\_MIN 5

#define INTERVAL\_MAX (MAX\_AGE - 1)

#define CHANGE 1 // 1

static int32\_t interval\_counter = INTERVAL\_MIN;

static int temperature\_old = INT\_MIN;

PERIODIC\_RESOURCE(res\_temperature,

"title=\"Temperature\";rt=\"Temperature\";obs",

res\_get\_handler,

NULL,

NULL,

NULL,

CLOCK\_SECOND,

res\_periodic\_handler);

static void

res\_get\_handler(void \*request, void \*response, uint8\_t \*buffer, uint16\_t preferred\_size, int32\_t \*offset)

{

/\*

\* For minimal complexity, request query and options should be ignored for GET on observable resources.

\* Otherwise the requests must be stored with the observer list and passed by REST.notify\_subscribers().

\* This would be a TODO in the corresponding files in contiki/apps/erbium/!

\*/

int temperature = ds18b20\_get\_temp();

//rt\_kprintf("temp $%d\r\n", temperature);

unsigned int accept = -1;

REST.get\_header\_accept(request, &accept);

if( accept == REST.type.TEXT\_PLAIN) {

REST.set\_header\_content\_type(response, REST.type.TEXT\_PLAIN);

snprintf((char \*)buffer, REST\_MAX\_CHUNK\_SIZE, "%d", temperature);

REST.set\_response\_payload(response, (uint8\_t \*)buffer, strlen((char \*)buffer));

} else if(accept == -1 || accept == REST.type.APPLICATION\_JSON) {

REST.set\_header\_content\_type(response, REST.type.APPLICATION\_JSON);

snprintf((char \*)buffer, REST\_MAX\_CHUNK\_SIZE, "{\"temperature\":%d}", temperature);

REST.set\_response\_payload(response, buffer, strlen((char \*)buffer));

} else {

REST.set\_response\_status(response, REST.status.NOT\_ACCEPTABLE);

const char \*msg = "Supporting content-types text/plain and application/json";

REST.set\_response\_payload(response, msg, strlen(msg));

}

REST.set\_header\_max\_age(response, MAX\_AGE);

/\* The REST.subscription\_handler() will be called for observable resources by the REST framework. \*/

}

/\*

\* Additionally, a handler function named [resource name]\_handler must be implemented for each PERIODIC\_RESOURCE.

\* It will be called by the REST manager process with the defined period.

\*/

static void

res\_periodic\_handler()

{

int temperature = ds18b20\_get\_temp();

//rt\_kprintf("temp $%d\r\n", temperature);

++interval\_counter;

if((abs(temperature - temperature\_old) >= CHANGE && interval\_counter >= INTERVAL\_MIN) ||

interval\_counter >= INTERVAL\_MAX) {

interval\_counter = 0;

temperature\_old = temperature;

/\* Notify the registered observers which will trigger the res\_get\_handler to create the response. \*/

REST.notify\_subscribers(&res\_temperature);

}

}

\* Matthias Kovatsch <kovatsch@inf.ethz.ch>

\*/

#define LOG\_TAG "water"

#include "contiki.h"

#include <string.h>

#include "rest-engine.h"

#include <elog.h>

#include "cJSON.h"

#include "drv\_relay.h"

static void res\_post\_put\_handler(void \*request, void \*response, uint8\_t \*buffer, uint16\_t preferred\_size, int32\_t \*offset);

/\*A simple actuator example, depending on the color query parameter and post variable mode, corresponding led is activated or deactivated\*/

RESOURCE(res\_water,

"title=\"water control\";rt=\"Control\"",

NULL,

res\_post\_put\_handler,

res\_post\_put\_handler,

NULL);

static void

res\_post\_put\_handler(void \*request, void \*response, uint8\_t \*buffer, uint16\_t preferred\_size, int32\_t \*offset)

{

size\_t len = 0;

uint8\_t \*payload = 0;

unsigned int accept = -1;

unsigned int content\_format = -1;

int ack = -1;

REST.get\_header\_accept(request, &accept);

REST.get\_header\_content\_type(request, &content\_format);

if(content\_format == REST.type.APPLICATION\_JSON)

{

if(accept == REST.type.APPLICATION\_JSON)

{

len = REST.get\_request\_payload(request,&payload);

if(len)

{

cJSON \*root = cJSON\_Parse(payload);

if(root)

{

cJSON \*mode = cJSON\_GetObjectItem(root, "mode");

if(mode)

{

log\_i("mode:%s",mode->valuestring);

if(strncmp(mode->valuestring, "on", 2) == 0)

{

cJSON \*delay = cJSON\_GetObjectItem(root, "delay");

if(delay)

{

log\_i("delay:%d",delay->valueint);

relay\_control(RELAY\_WATER, 1, delay->valueint);

ack = 0;

}

}

else if(strncmp(mode->valuestring, "off", 3) == 0)

{

relay\_control(RELAY\_WATER, 0, -1);

ack = 0;

}

}

cJSON\_Delete(root);

}

}

}

}

else

{

REST.set\_response\_status(response, REST.status.BAD\_REQUEST);

return;

}

REST.set\_header\_content\_type(response, REST.type.APPLICATION\_JSON);

snprintf((char \*)buffer, REST\_MAX\_CHUNK\_SIZE, "{\"ack\":%d}", ack);

REST.set\_response\_payload(response, buffer, strlen((char \*)buffer));

}

#include "contiki.h"

#include <limits.h>

#include <stdlib.h>

#include <string.h>

#include "rest-engine.h"

#include "drv\_hx711.h"

static void res\_get\_handler(void \*request, void \*response, uint8\_t \*buffer, uint16\_t preferred\_size, int32\_t \*offset);

static void res\_periodic\_handler(void);

#define MAX\_AGE 60 // 60s

#define INTERVAL\_MIN 1

#define INTERVAL\_MAX (MAX\_AGE - 1)

#define CHANGE 0.5 // 0.5kg

static int32\_t interval\_counter = INTERVAL\_MIN;

static float weight\_old = 0;

PERIODIC\_RESOURCE(res\_weight,

"title=\"Weight\";rt=\"Weight\";obs",

res\_get\_handler,

NULL,

NULL,

NULL,

CLOCK\_SECOND\*20,

res\_periodic\_handler);

static void

res\_get\_handler(void \*request, void \*response, uint8\_t \*buffer, uint16\_t preferred\_size, int32\_t \*offset)

{

/\*

\* For minimal complexity, request query and options should be ignored for GET on observable resources.

\* Otherwise the requests must be stored with the observer list and passed by REST.notify\_subscribers().

\* This would be a TODO in the corresponding files in contiki/apps/erbium/!

\*/

char str[50];

float weight = hx711\_weight\_get();

sprintf(str,"get\_weight:%f---------------------------",weight);

rt\_kprintf(str);

unsigned int accept = -1;

REST.get\_header\_accept(request, &accept);

if(accept == REST.type.TEXT\_PLAIN) {

REST.set\_header\_content\_type(response, REST.type.TEXT\_PLAIN);

snprintf((char \*)buffer, REST\_MAX\_CHUNK\_SIZE, "%.1f", weight);

REST.set\_response\_payload(response, (uint8\_t \*)buffer, strlen((char \*)buffer));

} else if(accept == -1 || accept == REST.type.APPLICATION\_JSON) {

REST.set\_header\_content\_type(response, REST.type.APPLICATION\_JSON);

snprintf((char \*)buffer, REST\_MAX\_CHUNK\_SIZE, "{\"weight\":%.1f}", weight);

REST.set\_response\_payload(response, buffer, strlen((char \*)buffer));

} else {

REST.set\_response\_status(response, REST.status.NOT\_ACCEPTABLE);

const char \*msg = "Supporting content-types text/plain and application/json";

REST.set\_response\_payload(response, msg, strlen(msg));

}

REST.set\_header\_max\_age(response, MAX\_AGE);

/\* The REST.subscription\_handler() will be called for observable resources by the REST framework. \*/

}

/\*

\* Additionally, a handler function named [resource name]\_handler must be implemented for each PERIODIC\_RESOURCE.

\* It will be called by the REST manager process with the defined period.

\*/

static void

res\_periodic\_handler()

{

char str[50] = {0};

float weight = hx711\_weight\_get();

float change = weight>weight\_old?(weight-weight\_old):(weight\_old-weight);

sprintf(str,"res\_weight:%f---------------------------",weight);

rt\_kprintf(str);

++interval\_counter;

if((change >= CHANGE && interval\_counter >= INTERVAL\_MIN) ||

interval\_counter >= INTERVAL\_MAX) {

interval\_counter = 0;

weight\_old = weight;

/\* Notify the registered observers which will trigger the res\_get\_handler to create the response. \*/

REST.notify\_subscribers(&res\_weight);

}

}

#include <string.h>

#include <stdio.h>

#include "contiki.h"

#include "rest-engine.h"

#define DEBUG 0

#if DEBUG

#include <stdio.h>

#define PRINTF(...) printf(\_\_VA\_ARGS\_\_)

#define PRINT6ADDR(addr) PRINTF("[%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x]", ((uint8\_t \*)addr)[0], ((uint8\_t \*)addr)[1], ((uint8\_t \*)addr)[2], ((uint8\_t \*)addr)[3], ((uint8\_t \*)addr)[4], ((uint8\_t \*)addr)[5], ((uint8\_t \*)addr)[6], ((uint8\_t \*)addr)[7], ((uint8\_t \*)addr)[8], ((uint8\_t \*)addr)[9], ((uint8\_t \*)addr)[10], ((uint8\_t \*)addr)[11], ((uint8\_t \*)addr)[12], ((uint8\_t \*)addr)[13], ((uint8\_t \*)addr)[14], ((uint8\_t \*)addr)[15])

#define PRINTLLADDR(lladdr) PRINTF("[%02x:%02x:%02x:%02x:%02x:%02x]", (lladdr)->addr[0], (lladdr)->addr[1], (lladdr)->addr[2], (lladdr)->addr[3], (lladdr)->addr[4], (lladdr)->addr[5])

#else

#define PRINTF(...)

#define PRINT6ADDR(addr)

#define PRINTLLADDR(addr)

#endif

PROCESS(rest\_engine\_process, "REST Engine");

/\*---------------------------------------------------------------------------\*/

LIST(restful\_services);

LIST(restful\_periodic\_services);

/\*---------------------------------------------------------------------------\*/

/\*- REST Engine API ---------------------------------------------------------\*/

/\*---------------------------------------------------------------------------\*/

/\*\*

\* \brief Initializes and starts the REST Engine process

\*

\* This function must be called by server processes before any resources are

\* registered through rest\_activate\_resource().

\*/

void

rest\_init\_engine(void)

{

list\_init(restful\_services);

REST.set\_service\_callback(rest\_invoke\_restful\_service);

/\* Start the RESTful server implementation. \*/

REST.init();

/\*Start REST engine process \*/

process\_start(&rest\_engine\_process, NULL);

}

/\*---------------------------------------------------------------------------\*/

/\*\*

\* \brief Makes a resource available under the given URI path

\* \param resource A pointer to a resource implementation

\* \param path The URI path string for this resource

\*

\* The resource implementation must be imported first using the

\* extern keyword. The build system takes care of compiling every

\* \*.c file in the ./resources/ sub-directory (see example Makefile).

\*/

void

rest\_activate\_resource(resource\_t \*resource, char \*path)

{

resource->url = path;

list\_add(restful\_services, resource);

PRINTF("Activating: %s\n", resource->url);

/\* Only add periodic resources with a periodic\_handler and a period > 0. \*/

if(resource->flags & IS\_PERIODIC && resource->periodic->periodic\_handler

&& resource->periodic->period) {

PRINTF("Periodic resource: %p (%s)\n", resource->periodic,

resource->periodic->resource->url);

list\_add(restful\_periodic\_services, resource->periodic);

}

}

/\*---------------------------------------------------------------------------\*/

/\*- Internal API ------------------------------------------------------------\*/

/\*---------------------------------------------------------------------------\*/

list\_t

rest\_get\_resources(void)

{

return restful\_services;

}

/\*---------------------------------------------------------------------------\*/

int

rest\_invoke\_restful\_service(void \*request, void \*response, uint8\_t \*buffer,

uint16\_t buffer\_size, int32\_t \*offset)

{

uint8\_t found = 0;

uint8\_t allowed = 1;

resource\_t \*resource = NULL;

const char \*url = NULL;

for(resource = (resource\_t \*)list\_head(restful\_services);

resource; resource = resource->next) {

/\* if the web service handles that kind of requests and urls matches \*/

if((REST.get\_url(request, &url) == strlen(resource->url)

|| (REST.get\_url(request, &url) > strlen(resource->url)

&& (resource->flags & HAS\_SUB\_RESOURCES)))

&& strncmp(resource->url, url, strlen(resource->url)) == 0) {

found = 1;

rest\_resource\_flags\_t method = REST.get\_method\_type(request);

PRINTF("/%s, method %u, resource->flags %u\n", resource->url,

(uint16\_t)method, resource->flags);

if((method & METHOD\_GET) && resource->get\_handler != NULL) {

/\* call handler function \*/

resource->get\_handler(request, response, buffer, buffer\_size, offset);

} else if((method & METHOD\_POST) && resource->post\_handler != NULL) {

/\* call handler function \*/

resource->post\_handler(request, response, buffer, buffer\_size,

offset);

} else if((method & METHOD\_PUT) && resource->put\_handler != NULL) {

/\* call handler function \*/

resource->put\_handler(request, response, buffer, buffer\_size, offset);

} else if((method & METHOD\_DELETE) && resource->delete\_handler != NULL) {

/\* call handler function \*/

resource->delete\_handler(request, response, buffer, buffer\_size,

offset);

} else {

allowed = 0;

REST.set\_response\_status(response, REST.status.METHOD\_NOT\_ALLOWED);

}

break;

}

}

if(!found) {

REST.set\_response\_status(response, REST.status.NOT\_FOUND);

} else if(allowed) {

/\* final handler for special flags \*/

if(resource->flags & IS\_OBSERVABLE) {

REST.subscription\_handler(resource, request, response);

}

}

return found & allowed;

}

/\*-----------------------------------------------------------------------------

PROCESS\_THREAD(rest\_engine\_process, ev, data)

{

PROCESS\_BEGIN();

/\* pause to let REST server finish adding resources. \*/

PROCESS\_PAUSE();

/\* initialize the PERIODIC\_RESOURCE timers, which will be handled by this process. \*/

periodic\_resource\_t \*periodic\_resource = NULL;

for(periodic\_resource =

(periodic\_resource\_t \*)list\_head(restful\_periodic\_services);

periodic\_resource; periodic\_resource = periodic\_resource->next) {

if(periodic\_resource->periodic\_handler && periodic\_resource->period) {

PRINTF("Periodic: Set timer for /%s to %lu\n",

periodic\_resource->resource->url, periodic\_resource->period);

etimer\_set(&periodic\_resource->periodic\_timer,

periodic\_resource->period);

}

}

while(1) {

PROCESS\_WAIT\_EVENT();

if(ev == PROCESS\_EVENT\_TIMER) {

for(periodic\_resource =

(periodic\_resource\_t \*)list\_head(restful\_periodic\_services);

periodic\_resource; periodic\_resource = periodic\_resource->next) {

if(periodic\_resource->period

&& etimer\_expired(&periodic\_resource->periodic\_timer)) {

PRINTF("Periodic: etimer expired for /%s (period: %lu)\n",

periodic\_resource->resource->url, periodic\_resource->period);

/\* Call the periodic\_handler function, which was checked during adding to list. \*/

(periodic\_resource->periodic\_handler)();

etimer\_reset(&periodic\_resource->periodic\_timer);

}

}

}

}

PROCESS\_END();

}

/\*---------------------------------------------------------------------------\*/

#include <stdlib.h>

#include <string.h>

#include "rest-engine.h"

static void res\_get\_handler(void \*request, void \*response, uint8\_t \*buffer, uint16\_t preferred\_size, int32\_t \*offset);

/\*

\* A handler function named [resource name]\_handler must be implemented for each RESOURCE.

\* A buffer for the response payload is provided through the buffer pointer. Simple resources can ignore

\* preferred\_size and offset, but must respect the REST\_MAX\_CHUNK\_SIZE limit for the buffer.

\* If a smaller block size is requested for CoAP, the REST framework automatically splits the data.

\*/

RESOURCE(res\_hello,

"title=\"Hello world: ?len=0..\";rt=\"Text\"",

res\_get\_handler,

NULL,

NULL,

NULL);

static void

res\_get\_handler(void \*request, void \*response, uint8\_t \*buffer, uint16\_t preferred\_size, int32\_t \*offset)

{

const char \*len = NULL;

/\* Some data that has the length up to REST\_MAX\_CHUNK\_SIZE. For more, see the chunk resource. \*/

char const \*const message = "Hello World! ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxy";

int length = 12; /\* |<-------->| \*/

/\* The query string can be retrieved by rest\_get\_query() or parsed for its key-value pairs. \*/

if(REST.get\_query\_variable(request, "len", &len)) {

length = atoi(len);

if(length < 0) {

length = 0;

}

if(length > REST\_MAX\_CHUNK\_SIZE) {

length = REST\_MAX\_CHUNK\_SIZE;

}

memcpy(buffer, message, length);

} else {

memcpy(buffer, message, length);

} REST.set\_header\_content\_type(response, REST.type.TEXT\_PLAIN); /\* text/plain is the default, hence this option could be omitted. \*/

REST.set\_header\_etag(response, (uint8\_t \*)&length, 1);

REST.set\_response\_payload(response, buffer, length);

}

#include "net/llsec/anti-replay.h"

#include "net/packetbuf.h"

#include "net/llsec/llsec802154.h"

#if LLSEC802154\_USES\_FRAME\_COUNTER

/\* This node's current frame counter value \*/

static uint32\_t counter;

/\*---------------------------------------------------------------------------\*/

void

anti\_replay\_set\_counter(void)

{

frame802154\_frame\_counter\_t reordered\_counter;

++counter;

reordered\_counter.u32 = LLSEC802154\_HTONL(counter);

packetbuf\_set\_attr(PACKETBUF\_ATTR\_FRAME\_COUNTER\_BYTES\_0\_1, reordered\_counter.u16[0]);

packetbuf\_set\_attr(PACKETBUF\_ATTR\_FRAME\_COUNTER\_BYTES\_2\_3, reordered\_counter.u16[1]);

}

/\*---------------------------------------------------------------------------\*/

uint32\_t

anti\_replay\_get\_counter(void)

{

frame802154\_frame\_counter\_t disordered\_counter;

disordered\_counter.u16[0] = packetbuf\_attr(PACKETBUF\_ATTR\_FRAME\_COUNTER\_BYTES\_0\_1);

disordered\_counter.u16[1] = packetbuf\_attr(PACKETBUF\_ATTR\_FRAME\_COUNTER\_BYTES\_2\_3);

return LLSEC802154\_HTONL(disordered\_counter.u32);

}

/\*---------------------------------------------------------------------------\*/

void

anti\_replay\_init\_info(struct anti\_replay\_info \*info)

{

info->last\_broadcast\_counter

= info->last\_unicast\_counter

= anti\_replay\_get\_counter();

}

/\*---------------------------------------------------------------------------\*/

int

anti\_replay\_was\_replayed(struct anti\_replay\_info \*info)

{

uint32\_t received\_counter;

received\_counter = anti\_replay\_get\_counter();

if(packetbuf\_holds\_broadcast()) {

/\* broadcast \*/

if(received\_counter <= info->last\_broadcast\_counter) {

return 1;

} else {

info->last\_broadcast\_counter = received\_counter;

return 0;

}

} else {

/\* unicast \*/

if(received\_counter <= info->last\_unicast\_counter) {

return 1;

} else {

info->last\_unicast\_counter = received\_counter;

return 0;

}

}

}

#include <rthw.h>

#include <rtdevice.h>

#include <board.h>

#ifdef RT\_USING\_PIN

#define STM32\_PIN\_NUMBERS 64 //[48, 64, 100, 144 ]

#define \_\_STM32\_PIN(index, rcc, gpio, gpio\_index) { 0, RCC\_##rcc##Periph\_GPIO##gpio, GPIO##gpio, GPIO\_Pin\_##gpio\_index}

#define \_\_STM32\_PIN\_DEFAULT {-1, 0, 0, 0}

#define ITEM\_NUM(items) sizeof(items)/sizeof(items[0])

/\* STM32 GPIO driver \*/

struct pin\_index

{

int index;

uint32\_t rcc;

GPIO\_TypeDef \*gpio;

uint32\_t pin;

};

/\* STM32 GPIO irq information \*/

struct pin\_irq

{

/\* EXTI port source gpiox, such as EXTI\_PortSourceGPIOA \*/

rt\_uint8\_t port\_source;

/\* EXTI pin sources, such as EXTI\_PinSource0 \*/

rt\_uint8\_t pin\_source;

/\* NVIC IRQ EXTI channel, such as EXTI0\_IRQn \*/

enum IRQn irq\_exti\_channel;

/\* EXTI line, such as EXTI\_Line0 \*/

rt\_uint32\_t exti\_line;

};

static const struct pin\_index pins[] =

{

#if (STM32\_PIN\_NUMBERS == 48)

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(2, AHB1, C, 13),

\_\_STM32\_PIN(3, AHB1, C, 14),

\_\_STM32\_PIN(4, AHB1, C, 15),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(10, AHB1, A, 0),

\_\_STM32\_PIN(11, AHB1, A, 1),

\_\_STM32\_PIN(12, AHB1, A, 2),

\_\_STM32\_PIN(13, AHB1, A, 3),

\_\_STM32\_PIN(14, AHB1, A, 4),

\_\_STM32\_PIN(15, AHB1, A, 5),

\_\_STM32\_PIN(16, AHB1, A, 6),

\_\_STM32\_PIN(17, AHB1, A, 7),

\_\_STM32\_PIN(18, AHB1, B, 0),

\_\_STM32\_PIN(19, AHB1, B, 1),

\_\_STM32\_PIN(20, AHB1, B, 2),

\_\_STM32\_PIN(21, AHB1, B, 10),

\_\_STM32\_PIN(22, AHB1, B, 11),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(25, AHB1, B, 12),

\_\_STM32\_PIN(26, AHB1, B, 13),

\_\_STM32\_PIN(27, AHB1, B, 14),

\_\_STM32\_PIN(28, AHB1, B, 15),

\_\_STM32\_PIN(29, AHB1, A, 8),

\_\_STM32\_PIN(30, AHB1, A, 9),

\_\_STM32\_PIN(31, AHB1, A, 10),

\_\_STM32\_PIN(32, AHB1, A, 11),

\_\_STM32\_PIN(33, AHB1, A, 12),

\_\_STM32\_PIN(34, AHB1, A, 13),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(37, AHB1, A, 14),

\_\_STM32\_PIN(38, AHB1, A, 15),

\_\_STM32\_PIN(39, AHB1, B, 3),

\_\_STM32\_PIN(40, AHB1, B, 4),

\_\_STM32\_PIN(41, AHB1, B, 5),

\_\_STM32\_PIN(42, AHB1, B, 6),

\_\_STM32\_PIN(43, AHB1, B, 7),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(45, AHB1, B, 8),

\_\_STM32\_PIN(46, AHB1, B, 9),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

#endif

#if (STM32\_PIN\_NUMBERS == 64)

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(2, AHB1, C, 13),

\_\_STM32\_PIN(3, AHB1, C, 14),

\_\_STM32\_PIN(4, AHB1, C, 15),

\_\_STM32\_PIN(5, AHB1, D, 0),

\_\_STM32\_PIN(6, AHB1, D, 1),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(8, AHB1, C, 0),

\_\_STM32\_PIN(9, AHB1, C, 1),

\_\_STM32\_PIN(10, AHB1, C, 2),

\_\_STM32\_PIN(11, AHB1, C, 3),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(14, AHB1, A, 0),

\_\_STM32\_PIN(15, AHB1, A, 1),

\_\_STM32\_PIN(16, AHB1, A, 2),

\_\_STM32\_PIN(17, AHB1, A, 3),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(20, AHB1, A, 4),

\_\_STM32\_PIN(21, AHB1, A, 5),

\_\_STM32\_PIN(22, AHB1, A, 6),

\_\_STM32\_PIN(23, AHB1, A, 7),

\_\_STM32\_PIN(24, AHB1, C, 4),

\_\_STM32\_PIN(25, AHB1, C, 5),

\_\_STM32\_PIN(26, AHB1, B, 0),

\_\_STM32\_PIN(27, AHB1, B, 1),

\_\_STM32\_PIN(28, AHB1, B, 2),

\_\_STM32\_PIN(29, AHB1, B, 10),

\_\_STM32\_PIN(30, AHB1, B, 11),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(33, AHB1, B, 12),

\_\_STM32\_PIN(34, AHB1, B, 13),

\_\_STM32\_PIN(35, AHB1, B, 14),

\_\_STM32\_PIN(36, AHB1, B, 15),

\_\_STM32\_PIN(37, AHB1, C, 6),

\_\_STM32\_PIN(38, AHB1, C, 7),

\_\_STM32\_PIN(39, AHB1, C, 8),

\_\_STM32\_PIN(40, AHB1, C, 9),

\_\_STM32\_PIN(41, AHB1, A, 8),

\_\_STM32\_PIN(42, AHB1, A, 9),

\_\_STM32\_PIN(43, AHB1, A, 10),

\_\_STM32\_PIN(44, AHB1, A, 11),

\_\_STM32\_PIN(45, AHB1, A, 12),

\_\_STM32\_PIN(46, AHB1, A, 13),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(49, AHB1, A, 14),

\_\_STM32\_PIN(50, AHB1, A, 15),

\_\_STM32\_PIN(51, AHB1, C, 10),

\_\_STM32\_PIN(52, AHB1, C, 11),

\_\_STM32\_PIN(53, AHB1, C, 12),

\_\_STM32\_PIN(54, AHB1, D, 2),

\_\_STM32\_PIN(55, AHB1, B, 3),

\_\_STM32\_PIN(56, AHB1, B, 4),

\_\_STM32\_PIN(57, AHB1, B, 5),

\_\_STM32\_PIN(58, AHB1, B, 6),

\_\_STM32\_PIN(59, AHB1, B, 7),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(61, AHB1, B, 8),

\_\_STM32\_PIN(62, AHB1, B, 9),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

#endif

#if (STM32\_PIN\_NUMBERS == 100)

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(1, AHB1, E, 2),

\_\_STM32\_PIN(2, AHB1, E, 3),

\_\_STM32\_PIN(3, AHB1, E, 4),

\_\_STM32\_PIN(4, AHB1, E, 5),

\_\_STM32\_PIN(5, AHB1, E, 6),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(7, AHB1, C, 13),

\_\_STM32\_PIN(8, AHB1, C, 14),

\_\_STM32\_PIN(9, AHB1, C, 15),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(15, AHB1, C, 0),

\_\_STM32\_PIN(16, AHB1, C, 1),

\_\_STM32\_PIN(17, AHB1, C, 2),

\_\_STM32\_PIN(18, AHB1, C, 3),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(23, AHB1, A, 0),

\_\_STM32\_PIN(24, AHB1, A, 1),

\_\_STM32\_PIN(25, AHB1, A, 2),

\_\_STM32\_PIN(26, AHB1, A, 3),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(29, AHB1, A, 4),

\_\_STM32\_PIN(30, AHB1, A, 5),

\_\_STM32\_PIN(31, AHB1, A, 6),

\_\_STM32\_PIN(32, AHB1, A, 7),

\_\_STM32\_PIN(33, AHB1, C, 4),

\_\_STM32\_PIN(34, AHB1, C, 5),

\_\_STM32\_PIN(35, AHB1, B, 0),

\_\_STM32\_PIN(36, AHB1, B, 1),

\_\_STM32\_PIN(37, AHB1, B, 2),

\_\_STM32\_PIN(38, AHB1, E, 7),

\_\_STM32\_PIN(39, AHB1, E, 8),

\_\_STM32\_PIN(40, AHB1, E, 9),

\_\_STM32\_PIN(41, AHB1, E, 10),

\_\_STM32\_PIN(42, AHB1, E, 11),

\_\_STM32\_PIN(43, AHB1, E, 12),

\_\_STM32\_PIN(44, AHB1, E, 13),

\_\_STM32\_PIN(45, AHB1, E, 14),

\_\_STM32\_PIN(46, AHB1, E, 15),

\_\_STM32\_PIN(47, AHB1, B, 10),

\_\_STM32\_PIN(48, AHB1, B, 11),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(51, AHB1, B, 12),

\_\_STM32\_PIN(52, AHB1, B, 13),

\_\_STM32\_PIN(53, AHB1, B, 14),

\_\_STM32\_PIN(54, AHB1, B, 15),

\_\_STM32\_PIN(55, AHB1, D, 8),

\_\_STM32\_PIN(56, AHB1, D, 9),

\_\_STM32\_PIN(57, AHB1, D, 10),

\_\_STM32\_PIN(58, AHB1, D, 11),

\_\_STM32\_PIN(59, AHB1, D, 12),

\_\_STM32\_PIN(60, AHB1, D, 13),

\_\_STM32\_PIN(61, AHB1, D, 14),

\_\_STM32\_PIN(62, AHB1, D, 15),

\_\_STM32\_PIN(63, AHB1, C, 6),

\_\_STM32\_PIN(64, AHB1, C, 7),

\_\_STM32\_PIN(65, AHB1, C, 8),

\_\_STM32\_PIN(66, AHB1, C, 9),

\_\_STM32\_PIN(67, AHB1, A, 8),

\_\_STM32\_PIN(68, AHB1, A, 9),

\_\_STM32\_PIN(69, AHB1, A, 10),

\_\_STM32\_PIN(70, AHB1, A, 11),

\_\_STM32\_PIN(71, AHB1, A, 12),

\_\_STM32\_PIN(72, AHB1, A, 13),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(76, AHB1, A, 14),

\_\_STM32\_PIN(77, AHB1, A, 15),

\_\_STM32\_PIN(78, AHB1, C, 10),

\_\_STM32\_PIN(79, AHB1, C, 11),

\_\_STM32\_PIN(80, AHB1, C, 12),

\_\_STM32\_PIN(81, AHB1, D, 0),

\_\_STM32\_PIN(82, AHB1, D, 1),

\_\_STM32\_PIN(83, AHB1, D, 2),

\_\_STM32\_PIN(84, AHB1, D, 3),

\_\_STM32\_PIN(85, AHB1, D, 4),

\_\_STM32\_PIN(86, AHB1, D, 5),

\_\_STM32\_PIN(87, AHB1, D, 6),

\_\_STM32\_PIN(88, AHB1, D, 7),

\_\_STM32\_PIN(89, AHB1, B, 3),

\_\_STM32\_PIN(90, AHB1, B, 4),

\_\_STM32\_PIN(91, AHB1, B, 5),

\_\_STM32\_PIN(92, AHB1, B, 6),

\_\_STM32\_PIN(93, AHB1, B, 7),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(95, AHB1, B, 8),

\_\_STM32\_PIN(96, AHB1, B, 9),

\_\_STM32\_PIN(97, AHB1, E, 0),

\_\_STM32\_PIN(98, AHB1, E, 1),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

#endif

#if (STM32\_PIN\_NUMBERS == 144)

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(1, AHB1, E, 2),

\_\_STM32\_PIN(2, AHB1, E, 3),

\_\_STM32\_PIN(3, AHB1, E, 4),

\_\_STM32\_PIN(4, AHB1, E, 5),

\_\_STM32\_PIN(5, AHB1, E, 6),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(7, AHB1, C, 13),

\_\_STM32\_PIN(8, AHB1, C, 14),

\_\_STM32\_PIN(9, AHB1, C, 15),

\_\_STM32\_PIN(10, AHB1, F, 0),

\_\_STM32\_PIN(11, AHB1, F, 1),

\_\_STM32\_PIN(12, AHB1, F, 2),

\_\_STM32\_PIN(13, AHB1, F, 3),

\_\_STM32\_PIN(14, AHB1, F, 4),

\_\_STM32\_PIN(15, AHB1, F, 5),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(18, AHB1, F, 6),

\_\_STM32\_PIN(19, AHB1, F, 7),

\_\_STM32\_PIN(20, AHB1, F, 8),

\_\_STM32\_PIN(21, AHB1, F, 9),

\_\_STM32\_PIN(22, AHB1, F, 10),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(26, AHB1, C, 0),

\_\_STM32\_PIN(27, AHB1, C, 1),

\_\_STM32\_PIN(28, AHB1, C, 2),

\_\_STM32\_PIN(29, AHB1, C, 3),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(34, AHB1, A, 0),

\_\_STM32\_PIN(35, AHB1, A, 1),

\_\_STM32\_PIN(36, AHB1, A, 2),

\_\_STM32\_PIN(37, AHB1, A, 3),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(40, AHB1, A, 4),

\_\_STM32\_PIN(41, AHB1, A, 5),

\_\_STM32\_PIN(42, AHB1, A, 6),

\_\_STM32\_PIN(43, AHB1, A, 7),

\_\_STM32\_PIN(44, AHB1, C, 4),

\_\_STM32\_PIN(45, AHB1, C, 5),

\_\_STM32\_PIN(46, AHB1, B, 0),

\_\_STM32\_PIN(47, AHB1, B, 1),

\_\_STM32\_PIN(48, AHB1, B, 2),

\_\_STM32\_PIN(49, AHB1, F, 11),

\_\_STM32\_PIN(50, AHB1, F, 12),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(53, AHB1, F, 13),

\_\_STM32\_PIN(54, AHB1, F, 14),

\_\_STM32\_PIN(55, AHB1, F, 15),

\_\_STM32\_PIN(56, AHB1, G, 0),

\_\_STM32\_PIN(57, AHB1, G, 1),

\_\_STM32\_PIN(58, AHB1, E, 7),

\_\_STM32\_PIN(59, AHB1, E, 8),

\_\_STM32\_PIN(60, AHB1, E, 9),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(63, AHB1, E, 10),

\_\_STM32\_PIN(64, AHB1, E, 11),

\_\_STM32\_PIN(65, AHB1, E, 12),

\_\_STM32\_PIN(66, AHB1, E, 13),

\_\_STM32\_PIN(67, AHB1, E, 14),

\_\_STM32\_PIN(68, AHB1, E, 15),

\_\_STM32\_PIN(69, AHB1, B, 10),

\_\_STM32\_PIN(70, AHB1, B, 11),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(73, AHB1, B, 12),

\_\_STM32\_PIN(74, AHB1, B, 13),

\_\_STM32\_PIN(75, AHB1, B, 14),

\_\_STM32\_PIN(76, AHB1, B, 15),

\_\_STM32\_PIN(77, AHB1, D, 8),

\_\_STM32\_PIN(78, AHB1, D, 9),

\_\_STM32\_PIN(79, AHB1, D, 10),

\_\_STM32\_PIN(80, AHB1, D, 11),

\_\_STM32\_PIN(81, AHB1, D, 12),

\_\_STM32\_PIN(82, AHB1, D, 13),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(85, AHB1, D, 14),

\_\_STM32\_PIN(86, AHB1, D, 15),

\_\_STM32\_PIN(87, AHB1, G, 2),

\_\_STM32\_PIN(88, AHB1, G, 3),

\_\_STM32\_PIN(89, AHB1, G, 4),

\_\_STM32\_PIN(90, AHB1, G, 5),

\_\_STM32\_PIN(91, AHB1, G, 6),

\_\_STM32\_PIN(92, AHB1, G, 7),

\_\_STM32\_PIN(93, AHB1, G, 8),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(96, AHB1, C, 6),

\_\_STM32\_PIN(97, AHB1, C, 7),

\_\_STM32\_PIN(98, AHB1, C, 8),

\_\_STM32\_PIN(99, AHB1, C, 9),

\_\_STM32\_PIN(100, AHB1, A, 8),

\_\_STM32\_PIN(101, AHB1, A, 9),

\_\_STM32\_PIN(102, AHB1, A, 10),

\_\_STM32\_PIN(103, AHB1, A, 11),

\_\_STM32\_PIN(104, AHB1, A, 12),

\_\_STM32\_PIN(105, AHB1, A, 13),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(109, AHB1, A, 14),

\_\_STM32\_PIN(110, AHB1, A, 15),

\_\_STM32\_PIN(111, AHB1, C, 10),

\_\_STM32\_PIN(112, AHB1, C, 11),

\_\_STM32\_PIN(113, AHB1, C, 12),

\_\_STM32\_PIN(114, AHB1, D, 0),

\_\_STM32\_PIN(115, AHB1, D, 1),

\_\_STM32\_PIN(116, AHB1, D, 2),

\_\_STM32\_PIN(117, AHB1, D, 3),

\_\_STM32\_PIN(118, AHB1, D, 4),

\_\_STM32\_PIN(119, AHB1, D, 5),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(122, AHB1, D, 6),

\_\_STM32\_PIN(123, AHB1, D, 7),

\_\_STM32\_PIN(124, AHB1, G, 9),

\_\_STM32\_PIN(125, AHB1, G, 10),

\_\_STM32\_PIN(126, AHB1, G, 11),

\_\_STM32\_PIN(127, AHB1, G, 12),

\_\_STM32\_PIN(128, AHB1, G, 13),

\_\_STM32\_PIN(129, AHB1, G, 14),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(132, AHB1, G, 15),

\_\_STM32\_PIN(133, AHB1, B, 3),

\_\_STM32\_PIN(134, AHB1, B, 4),

\_\_STM32\_PIN(135, AHB1, B, 5),

\_\_STM32\_PIN(136, AHB1, B, 6),

\_\_STM32\_PIN(137, AHB1, B, 7),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN(139, AHB1, B, 8),

\_\_STM32\_PIN(140, AHB1, B, 9),

\_\_STM32\_PIN(141, AHB1, E, 0),

\_\_STM32\_PIN(142, AHB1, E, 1),

\_\_STM32\_PIN\_DEFAULT,

\_\_STM32\_PIN\_DEFAULT,

#endif

};

struct rt\_pin\_irq\_hdr pin\_irq\_hdr\_tab[] =

{

{-1, 0, RT\_NULL, RT\_NULL},

{-1, 0, RT\_NULL, RT\_NULL},

{-1, 0, RT\_NULL, RT\_NULL},

{-1, 0, RT\_NULL, RT\_NULL},

{-1, 0, RT\_NULL, RT\_NULL},

{-1, 0, RT\_NULL, RT\_NULL},

{-1, 0, RT\_NULL, RT\_NULL},

{-1, 0, RT\_NULL, RT\_NULL},

{-1, 0, RT\_NULL, RT\_NULL},

{-1, 0, RT\_NULL, RT\_NULL},

{-1, 0, RT\_NULL, RT\_NULL},

{-1, 0, RT\_NULL, RT\_NULL},

{-1, 0, RT\_NULL, RT\_NULL},

{-1, 0, RT\_NULL, RT\_NULL},

{-1, 0, RT\_NULL, RT\_NULL},

{-1, 0, RT\_NULL, RT\_NULL},

};

const struct pin\_index \*get\_pin(uint8\_t pin)

{

const struct pin\_index \*index;

if (pin < ITEM\_NUM(pins))

{

index = &pins[pin];

if (index->index == -1)

index = RT\_NULL;

}

else

{

index = RT\_NULL;

}

return index;

};

void stm32\_pin\_write(rt\_device\_t dev, rt\_base\_t pin, rt\_base\_t value)

{

const struct pin\_index \*index;

index = get\_pin(pin);

if (index == RT\_NULL)

{

return;

}

if (value == PIN\_LOW)

{

GPIO\_ResetBits(index->gpio, index->pin);

}

else

{

GPIO\_SetBits(index->gpio, index->pin);

}

}

int stm32\_pin\_read(rt\_device\_t dev, rt\_base\_t pin)

{

int value;

const struct pin\_index \*index;

value = PIN\_LOW;

index = get\_pin(pin);

if (index == RT\_NULL)

{

return value;

}

if (GPIO\_ReadInputDataBit(index->gpio, index->pin) == Bit\_RESET)

{

value = PIN\_LOW;

}

else

{

value = PIN\_HIGH;

}

return value;

}

void stm32\_pin\_mode(rt\_device\_t dev, rt\_base\_t pin, rt\_base\_t mode)

{

const struct pin\_index \*index;

GPIO\_InitTypeDef GPIO\_InitStructure;

index = get\_pin(pin);

if (index == RT\_NULL)

{

return;

}

/\* GPIO Periph clock enable \*/

RCC\_AHB1PeriphClockCmd(index->rcc, ENABLE);

/\* Configure GPIO\_InitStructure \*/

GPIO\_InitStructure.GPIO\_Pin = index->pin;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_2MHz;

GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_NOPULL;

if (mode == PIN\_MODE\_OUTPUT)

{

/\* output setting \*/

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_OUT;

GPIO\_InitStructure.GPIO\_OType = GPIO\_OType\_PP;

}

else if (mode == PIN\_MODE\_INPUT)

{

/\* input setting: not pull. \*/

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_IN;

GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_NOPULL;

}

else if (mode == PIN\_MODE\_INPUT\_PULLUP)

{

/\* input setting: pull up. \*/

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_IN;

GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_UP;

}

else if (mode == PIN\_MODE\_INPUT\_PULLDOWN)

{

/\* input setting: pull down. \*/

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_IN;

GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_DOWN;

}

else if (mode == PIN\_MODE\_OUTPUT\_OD)

{

/\* output setting: open drain \*/

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_OUT;

GPIO\_InitStructure.GPIO\_OType = GPIO\_OType\_OD;

}

else

{

/\* error mode \*/

RT\_ASSERT(0);

}

GPIO\_Init(index->gpio, &GPIO\_InitStructure);

}

rt\_inline rt\_int32\_t bit2bitno(rt\_uint32\_t bit)

{

int i;

for (i = 0; i < 32; i++)

{

if ((1UL << i) == bit)

{

return i;

}

}

return -1;

}

rt\_inline rt\_int32\_t bitno2bit(rt\_uint32\_t bitno)

{

if (bitno <= 32)

{

return 1UL << bitno;

}

else

{

return 0;

}

}

static const struct pin\_irq \*get\_pin\_irq(uint16\_t pin)

{

static struct pin\_irq irq;

const struct pin\_index \*index;

index = get\_pin(pin);

if (index == RT\_NULL)

{

return RT\_NULL;

}

irq.exti\_line = index->pin;

irq.pin\_source = bit2bitno(index->pin);

irq.port\_source = ((uint32\_t)index->gpio - GPIOA\_BASE) / (GPIOB\_BASE - GPIOA\_BASE);

switch (irq.pin\_source)

{

case 0 : irq.irq\_exti\_channel = EXTI0\_IRQn;break;

case 1 : irq.irq\_exti\_channel = EXTI1\_IRQn;break;

case 2 : irq.irq\_exti\_channel = EXTI2\_IRQn;break;

case 3 : irq.irq\_exti\_channel = EXTI3\_IRQn;break;

case 4 : irq.irq\_exti\_channel = EXTI4\_IRQn;break;

case 5 :

case 6 :

case 7 :

case 8 :

case 9 : irq.irq\_exti\_channel = EXTI9\_5\_IRQn;break;

case 10 :

case 11 :

case 12 :

case 13 :

case 14 :

case 15 : irq.irq\_exti\_channel = EXTI15\_10\_IRQn;break;

default : return RT\_NULL;

}

return &irq;

};

rt\_err\_t stm32\_pin\_attach\_irq(struct rt\_device \*device, rt\_int32\_t pin,

rt\_uint32\_t mode, void (\*hdr)(void \*args), void \*args)

{

const struct pin\_index \*index;

rt\_base\_t level;

rt\_int32\_t irqindex = -1;

index = get\_pin(pin);

if (index == RT\_NULL)

{

return -RT\_ENOSYS;

}

irqindex = bit2bitno(index->pin);

if (irqindex < 0 || irqindex >= ITEM\_NUM(pin\_irq\_hdr\_tab))

{

return -RT\_ENOSYS;

}

level = rt\_hw\_interrupt\_disable();

if (pin\_irq\_hdr\_tab[irqindex].pin == pin &&

pin\_irq\_hdr\_tab[irqindex].hdr == hdr &&

pin\_irq\_hdr\_tab[irqindex].mode == mode &&

pin\_irq\_hdr\_tab[irqindex].args == args

)

{

rt\_hw\_interrupt\_enable(level);

return RT\_EOK;

}

if (pin\_irq\_hdr\_tab[irqindex].pin != -1)

{

rt\_hw\_interrupt\_enable(level);

return -RT\_EBUSY;

}

pin\_irq\_hdr\_tab[irqindex].pin = pin;

//TODO PA1 will overwrite PB1's hdr, using rt\_list ?

pin\_irq\_hdr\_tab[irqindex].hdr = hdr;

pin\_irq\_hdr\_tab[irqindex].mode = mode;

pin\_irq\_hdr\_tab[irqindex].args = args;

rt\_hw\_interrupt\_enable(level);

return RT\_EOK;

}

rt\_err\_t stm32\_pin\_dettach\_irq(struct rt\_device \*device, rt\_int32\_t pin)

{

const struct pin\_index \*index;

rt\_base\_t level;

rt\_int32\_t irqindex = -1;

index = get\_pin(pin);

if (index == RT\_NULL)

{

return -RT\_ENOSYS;

}

irqindex = bit2bitno(index->pin);

if (irqindex < 0 || irqindex >= ITEM\_NUM(pin\_irq\_hdr\_tab))

{

return -RT\_ENOSYS;

}

level = rt\_hw\_interrupt\_disable();

if (pin\_irq\_hdr\_tab[irqindex].pin == -1)

{

rt\_hw\_interrupt\_enable(level);

return RT\_EOK;

}

pin\_irq\_hdr\_tab[irqindex].pin = -1;

pin\_irq\_hdr\_tab[irqindex].hdr = RT\_NULL;

pin\_irq\_hdr\_tab[irqindex].mode = 0;

pin\_irq\_hdr\_tab[irqindex].args = RT\_NULL;

rt\_hw\_interrupt\_enable(level);

return RT\_EOK;

}

rt\_err\_t stm32\_pin\_irq\_enable(struct rt\_device \*device, rt\_base\_t pin, rt\_uint32\_t enabled)

{

const struct pin\_index \*index;

const struct pin\_irq \*irq;

rt\_base\_t level;

rt\_int32\_t irqindex = -1;

NVIC\_InitTypeDef NVIC\_InitStructure;

EXTI\_InitTypeDef EXTI\_InitStructure;

index = get\_pin(pin);

if (index == RT\_NULL)

{

return -RT\_ENOSYS;

}

if (enabled == PIN\_IRQ\_ENABLE)

{

irqindex = bit2bitno(index->pin);

if (irqindex < 0 || irqindex >= ITEM\_NUM(pin\_irq\_hdr\_tab))

{

return -RT\_ENOSYS;

}

level = rt\_hw\_interrupt\_disable();

if (pin\_irq\_hdr\_tab[irqindex].pin == -1)

{

rt\_hw\_interrupt\_enable(level);

return -RT\_ENOSYS;

}

irq = get\_pin\_irq(pin);

if (irq == RT\_NULL)

{

rt\_hw\_interrupt\_enable(level);

return -RT\_ENOSYS;

}

/\* select the input source pin for the EXTI line \*/

SYSCFG\_EXTILineConfig(irq->port\_source, irq->pin\_source);

/\* select the mode(interrupt, event) and configure the trigger selection \*/

EXTI\_InitStructure.EXTI\_Line = irq->exti\_line;

EXTI\_InitStructure.EXTI\_Mode = EXTI\_Mode\_Interrupt;

switch (pin\_irq\_hdr\_tab[irqindex].mode)

{

case PIN\_IRQ\_MODE\_RISING:

EXTI\_InitStructure.EXTI\_Trigger = EXTI\_Trigger\_Rising;

break;

case PIN\_IRQ\_MODE\_FALLING:

EXTI\_InitStructure.EXTI\_Trigger = EXTI\_Trigger\_Falling;

break;

case PIN\_IRQ\_MODE\_RISING\_FALLING:

EXTI\_InitStructure.EXTI\_Trigger = EXTI\_Trigger\_Rising\_Falling;

break;

}

EXTI\_InitStructure.EXTI\_LineCmd = ENABLE;

EXTI\_Init(&EXTI\_InitStructure);

/\* configure NVIC IRQ channel mapped to the EXTI line \*/

NVIC\_InitStructure.NVIC\_IRQChannel = irq->irq\_exti\_channel;

NVIC\_InitStructure.NVIC\_IRQChannelPreemptionPriority = 2;

NVIC\_InitStructure.NVIC\_IRQChannelSubPriority = 2;

NVIC\_InitStructure.NVIC\_IRQChannelCmd = ENABLE;

NVIC\_Init(&NVIC\_InitStructure);

rt\_hw\_interrupt\_enable(level);

}

else if (enabled == PIN\_IRQ\_DISABLE)

{

irq = get\_pin\_irq(index->pin);

if (irq == RT\_NULL)

{

return -RT\_ENOSYS;

}

EXTI\_InitStructure.EXTI\_Line = irq->exti\_line;

EXTI\_InitStructure.EXTI\_Mode = EXTI\_Mode\_Interrupt;

EXTI\_InitStructure.EXTI\_Trigger = EXTI\_Trigger\_Rising;

EXTI\_InitStructure.EXTI\_LineCmd = DISABLE;

EXTI\_Init(&EXTI\_InitStructure);

}

else

{

return -RT\_ENOSYS;

}

return RT\_EOK;

}

const static struct rt\_pin\_ops \_stm32\_pin\_ops =

{

stm32\_pin\_mode,

stm32\_pin\_write,

stm32\_pin\_read,

stm32\_pin\_attach\_irq,

stm32\_pin\_dettach\_irq,

stm32\_pin\_irq\_enable,

};

int stm32\_hw\_pin\_init(void)

{

int result;

/\* enable SYSCFG clock for EXTI \*/

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_SYSCFG, ENABLE);

result = rt\_device\_pin\_register("pin", &\_stm32\_pin\_ops, RT\_NULL);

return result;

}

INIT\_BOARD\_EXPORT(stm32\_hw\_pin\_init);

rt\_inline void pin\_irq\_hdr(int irqno)

{

EXTI\_ClearITPendingBit(bitno2bit(irqno));

if (pin\_irq\_hdr\_tab[irqno].hdr)

{

pin\_irq\_hdr\_tab[irqno].hdr(pin\_irq\_hdr\_tab[irqno].args);

}

}

void EXTI0\_IRQHandler(void)

{

/\* enter interrupt \*/

rt\_interrupt\_enter();

pin\_irq\_hdr(0);

/\* leave interrupt \*/

rt\_interrupt\_leave();

}

void EXTI1\_IRQHandler(void)

{

/\* enter interrupt \*/

rt\_interrupt\_enter();

pin\_irq\_hdr(1);

/\* leave interrupt \*/

rt\_interrupt\_leave();

}

void EXTI2\_IRQHandler(void)

{

/\* enter interrupt \*/

rt\_interrupt\_enter();

pin\_irq\_hdr(2);

/\* leave interrupt \*/

rt\_interrupt\_leave();

}

void EXTI3\_IRQHandler(void)

{

/\* enter interrupt \*/

rt\_interrupt\_enter();

pin\_irq\_hdr(3);

/\* leave interrupt \*/

rt\_interrupt\_leave();

}

void EXTI4\_IRQHandler(void)

{

/\* enter interrupt \*/

rt\_interrupt\_enter();

pin\_irq\_hdr(4);

/\* leave interrupt \*/

rt\_interrupt\_leave();

}

void EXTI9\_5\_IRQHandler(void)

{

/\* enter interrupt \*/

rt\_interrupt\_enter();

if (EXTI\_GetITStatus(EXTI\_Line5) != RESET)

{

pin\_irq\_hdr(5);

}

if (EXTI\_GetITStatus(EXTI\_Line6) != RESET)

{

pin\_irq\_hdr(6);

}

if (EXTI\_GetITStatus(EXTI\_Line7) != RESET)

{

pin\_irq\_hdr(7);

}

if (EXTI\_GetITStatus(EXTI\_Line8) != RESET)

{

pin\_irq\_hdr(8);

}

if (EXTI\_GetITStatus(EXTI\_Line9) != RESET)

{

pin\_irq\_hdr(9);

}

/\* leave interrupt \*/

rt\_interrupt\_leave();

}

void EXTI15\_10\_IRQHandler(void)

{

/\* enter interrupt \*/

rt\_interrupt\_enter();

if (EXTI\_GetITStatus(EXTI\_Line10) != RESET)

{

pin\_irq\_hdr(10);

}

if (EXTI\_GetITStatus(EXTI\_Line11) != RESET)

{

pin\_irq\_hdr(11);

}

if (EXTI\_GetITStatus(EXTI\_Line12) != RESET)

{

pin\_irq\_hdr(12);

}

if (EXTI\_GetITStatus(EXTI\_Line13) != RESET)

{

pin\_irq\_hdr(13);

}

if (EXTI\_GetITStatus(EXTI\_Line14) != RESET)

{

pin\_irq\_hdr(14);

}

if (EXTI\_GetITStatus(EXTI\_Line15) != RESET)

{

pin\_irq\_hdr(15);

}

/\* leave interrupt \*/

rt\_interrupt\_leave();

}

#endif

#include "multi\_button.h"

#define EVENT\_CB(ev) if(handle->cb[ev]) handle->cb[ev]((button\*)handle)

static struct button\* head\_handle = NULL;

/\*\*

\* @brief Initializes the button struct handle.

\* @param handle: the button handle strcut.

\* @param pin\_level: read the pin of the connet button level.

\* @param active\_level: pin pressed level.

\* @retval None

\*/

void button\_init(struct button\* handle, uint8\_t(\*pin\_level)(void), uint8\_t active\_level)

{

memset(handle, 0, sizeof(struct button));

handle->event = (uint8\_t)NONE\_PRESS;

handle->hal\_button\_Level = pin\_level;

handle->button\_level = handle->hal\_button\_Level();

handle->active\_level = active\_level;

}

/\*\*

\* @brief Attach the button event callback function.

\* @param handle: the button handle strcut.

\* @param event: trigger event type.

\* @param cb: callback function.

\* @retval None

\*/

void button\_attach(struct button\* handle, PressEvent event, BtnCallback cb)

{

handle->cb[event] = cb;

}

/\*\*

\* @brief Inquire the button event happen.

\* @param handle: the button handle strcut.

\* @retval button event.

\*/

PressEvent get\_button\_event(struct button\* handle)

{

return (PressEvent)(handle->event);

}

/\*\*

\* @brief button driver core function, driver state machine.

\* @param handle: the button handle strcut.

\* @retval None

\*/

void button\_handler(struct button\* handle)

{

uint8\_t read\_gpio\_level = handle->hal\_button\_Level();

//ticks counter working..

if((handle->state) > 0)

{

handle->ticks++;

}

/\*------------button debounce handle---------------\*/

if(read\_gpio\_level != handle->button\_level)

{

//not equal to prev one

//continue read 3 times same new level change

if(++(handle->debounce\_cnt) >= DEBOUNCE\_TICKS)

{

handle->button\_level = read\_gpio\_level;

handle->debounce\_cnt = 0;

}

}

else

{

// leved not change ,counter reset.

handle->debounce\_cnt = 0;

}

/\*-----------------State machine-------------------\*/

switch (handle->state)

{

case 0:

if(handle->button\_level == handle->active\_level)

{

handle->event = (uint8\_t)PRESS\_DOWN;

EVENT\_CB(PRESS\_DOWN);

handle->ticks = 0;

handle->repeat = 1;

handle->state = 1;

}

else

{

handle->event = (uint8\_t)NONE\_PRESS;

}

break;

case 1:

if(handle->button\_level != handle->active\_level)

{

handle->event = (uint8\_t)PRESS\_UP;

EVENT\_CB(PRESS\_UP);

handle->ticks = 0;

handle->state = 2;

}

else if(handle->ticks > LONG\_TICKS)

{

handle->event = (uint8\_t)LONG\_RRESS\_START;

EVENT\_CB(LONG\_RRESS\_START);

handle->state = 5;

}

break;

case 2:

if(handle->button\_level == handle->active\_level)

{

handle->event = (uint8\_t)PRESS\_DOWN;

EVENT\_CB(PRESS\_DOWN);

handle->repeat++;

handle->event = (uint8\_t)PRESS\_REPEAT;

EVENT\_CB(PRESS\_REPEAT);

handle->ticks = 0;

handle->state = 3;

}

else if(handle->ticks > SHORT\_TICKS)

{

if(handle->repeat == 1)

{

handle->event = (uint8\_t)SINGLE\_CLICK;

EVENT\_CB(SINGLE\_CLICK);

}

else if(handle->repeat == 2)

{

handle->event = (uint8\_t)DOUBLE\_CLICK;

EVENT\_CB(DOUBLE\_CLICK);

}

handle->state = 0;

}

break;

case 3:

if(handle->button\_level != handle->active\_level)

{

handle->event = (uint8\_t)PRESS\_UP;

EVENT\_CB(PRESS\_UP);

if(handle->ticks < SHORT\_TICKS)

{

handle->ticks = 0;

handle->state = 2;

}

else

{

handle->state = 0;

}

}

break;

case 5:

if(handle->button\_level == handle->active\_level)

{

handle->event = (uint8\_t)LONG\_PRESS\_HOLD;

if (handle->ticks % LONG\_HOLD\_CYC == 0)

{

EVENT\_CB(LONG\_PRESS\_HOLD);

}

}

else

{

handle->event = (uint8\_t)PRESS\_UP;

EVENT\_CB(PRESS\_UP);

handle->state = 0;

}

break;

}

}

/\*\*

\* @brief Start the button work, add the handle into work list.

\* @param handle: target handle strcut.

\* @retval 0: succeed. -1: already exist.

\*/

int button\_start(struct button\* handle)

{

struct button\* target = head\_handle;

while(target)

{

if(target == handle)

{

return -1; //already exist.

}

target = target->next;

}

handle->next = head\_handle;

head\_handle = handle;

return 0;

}

/\*\*

\* @brief Stop the button work, remove the handle off work list.

\* @param handle: target handle strcut.

\* @retval None

\*/

void button\_stop(struct button\* handle)

{

struct button\*\* curr;

for(curr = &head\_handle; \*curr;)

{

struct button\* entry = \*curr;

if (entry == handle)

{

\*curr = entry->next;

}

else

{

curr = &entry->next;

}

}

}

/\*\*

\* @brief background ticks, timer repeat invoking interval 5ms.

\* @param None.

\* @retval None

\*/

void button\_ticks(void)

{

struct button\* target;

for(target = head\_handle; target != NULL; target = target->next)

{

button\_handler(target);

}

}

#include "drv\_usart.h"

#include "board.h"

#include <rtdevice.h>

/\* UART GPIO define. \*/

#define UART1\_GPIO\_TX GPIO\_Pin\_9

#define UART1\_TX\_PIN\_SOURCE GPIO\_PinSource9

#define UART1\_GPIO\_RX GPIO\_Pin\_10

#define UART1\_RX\_PIN\_SOURCE GPIO\_PinSource10

#define UART1\_GPIO GPIOA

#define UART1\_GPIO\_RCC RCC\_AHB1Periph\_GPIOA

#define RCC\_APBPeriph\_UART1 RCC\_APB2Periph\_USART1

#define UART2\_GPIO\_TX GPIO\_Pin\_2

#define UART2\_TX\_PIN\_SOURCE GPIO\_PinSource2

#define UART2\_GPIO\_RX GPIO\_Pin\_3

#define UART2\_RX\_PIN\_SOURCE GPIO\_PinSource3

#define UART2\_GPIO GPIOA

#define UART2\_GPIO\_RCC RCC\_AHB1Periph\_GPIOA

#define RCC\_APBPeriph\_UART2 RCC\_APB1Periph\_USART2

#define UART3\_GPIO\_TX GPIO\_Pin\_10

#define UART3\_TX\_PIN\_SOURCE GPIO\_PinSource10

#define UART3\_GPIO\_RX GPIO\_Pin\_11

#define UART3\_RX\_PIN\_SOURCE GPIO\_PinSource11

#define UART3\_GPIO GPIOB

#define UART3\_GPIO\_RCC RCC\_AHB1Periph\_GPIOB

#define RCC\_APBPeriph\_UART3 RCC\_APB1Periph\_USART3

#define UART4\_GPIO\_TX GPIO\_Pin\_10

#define UART4\_TX\_PIN\_SOURCE GPIO\_PinSource10

#define UART4\_GPIO\_RX GPIO\_Pin\_11

#define UART4\_RX\_PIN\_SOURCE GPIO\_PinSource11

#define UART4\_GPIO GPIOC

#define UART4\_GPIO\_RCC RCC\_AHB1Periph\_GPIOC

#define RCC\_APBPeriph\_UART4 RCC\_APB1Periph\_UART4

#define UART5\_GPIO\_TX GPIO\_Pin\_12

#define UART5\_TX\_PIN\_SOURCE GPIO\_PinSource12

#define UART5\_GPIO\_RX GPIO\_Pin\_2

#define UART5\_RX\_PIN\_SOURCE GPIO\_PinSource2

#define UART5\_TX GPIOC

#define UART5\_RX GPIOD

#define UART5\_GPIO\_RCC\_TX RCC\_AHB1Periph\_GPIOC

#define UART5\_GPIO\_RCC\_RX RCC\_AHB1Periph\_GPIOD

#define RCC\_APBPeriph\_UART5 RCC\_APB1Periph\_UART5

/\* STM32 uart driver \*/

struct stm32\_uart

{

USART\_TypeDef \*uart\_device;

IRQn\_Type irq;

struct stm32\_uart\_dma

{

/\* dma stream \*/

DMA\_Stream\_TypeDef \*rx\_stream;

/\* dma channel \*/

uint32\_t rx\_ch;

/\* dma flag \*/

uint32\_t rx\_flag;

/\* dma irq channel \*/

uint8\_t rx\_irq\_ch;

/\* setting receive len \*/

rt\_size\_t setting\_recv\_len;

/\* last receive index \*/

rt\_size\_t last\_recv\_index;

} dma;

};

static void DMA\_Configuration(struct rt\_serial\_device \*serial);

static rt\_err\_t stm32\_configure(struct rt\_serial\_device \*serial, struct serial\_configure \*cfg)

{

struct stm32\_uart\* uart;

USART\_InitTypeDef USART\_InitStructure;

RT\_ASSERT(serial != RT\_NULL);

RT\_ASSERT(cfg != RT\_NULL);

uart = (struct stm32\_uart \*)serial->parent.user\_data;

USART\_InitStructure.USART\_BaudRate = cfg->baud\_rate;

if (cfg->data\_bits == DATA\_BITS\_8){

USART\_InitStructure.USART\_WordLength = USART\_WordLength\_8b;

} else if (cfg->data\_bits == DATA\_BITS\_9) {

USART\_InitStructure.USART\_WordLength = USART\_WordLength\_9b;

}

if (cfg->stop\_bits == STOP\_BITS\_1){

USART\_InitStructure.USART\_StopBits = USART\_StopBits\_1;

} else if (cfg->stop\_bits == STOP\_BITS\_2){

USART\_InitStructure.USART\_StopBits = USART\_StopBits\_2;

}

if (cfg->parity == PARITY\_NONE){

USART\_InitStructure.USART\_Parity = USART\_Parity\_No;

} else if (cfg->parity == PARITY\_ODD) {

USART\_InitStructure.USART\_Parity = USART\_Parity\_Odd;

} else if (cfg->parity == PARITY\_EVEN) {

USART\_InitStructure.USART\_Parity = USART\_Parity\_Even;

}

USART\_InitStructure.USART\_HardwareFlowControl = USART\_HardwareFlowControl\_None;

USART\_InitStructure.USART\_Mode = USART\_Mode\_Rx | USART\_Mode\_Tx;

USART\_Init(uart->uart\_device, &USART\_InitStructure);

/\* Enable USART \*/

USART\_Cmd(uart->uart\_device, ENABLE);

return RT\_EOK;

}

static rt\_err\_t stm32\_control(struct rt\_serial\_device \*serial, int cmd, void \*arg)

{

struct stm32\_uart\* uart;

rt\_uint32\_t ctrl\_arg = (rt\_uint32\_t)(arg);

RT\_ASSERT(serial != RT\_NULL);

uart = (struct stm32\_uart \*)serial->parent.user\_data;

switch (cmd)

{

/\* disable interrupt \*/

case RT\_DEVICE\_CTRL\_CLR\_INT:

/\* disable rx irq \*/

UART\_DISABLE\_IRQ(uart->irq);

if (ctrl\_arg == RT\_DEVICE\_FLAG\_INT\_RX){

/\* disable interrupt \*/

USART\_ITConfig(uart->uart\_device, USART\_IT\_RXNE, DISABLE);

}

break;

/\* enable interrupt \*/

case RT\_DEVICE\_CTRL\_SET\_INT:

/\* enable rx irq \*/

UART\_ENABLE\_IRQ(uart->irq);

if (ctrl\_arg == RT\_DEVICE\_FLAG\_INT\_RX){

/\* enable interrupt \*/

USART\_ITConfig(uart->uart\_device, USART\_IT\_RXNE, ENABLE);

}

break;

/\* USART config \*/

case RT\_DEVICE\_CTRL\_CONFIG :

if (ctrl\_arg == RT\_DEVICE\_FLAG\_DMA\_RX) {

uart->dma.last\_recv\_index = 0;

uart->dma.setting\_recv\_len = 0;

DMA\_Configuration(serial);

/\* enable rx irq \*/

UART\_ENABLE\_IRQ(uart->irq);

}

break;

}

return RT\_EOK;

}

static int stm32\_putc(struct rt\_serial\_device \*serial, char c)

{

struct stm32\_uart \*uart;

RT\_ASSERT(serial != RT\_NULL);

uart = (struct stm32\_uart \*)serial->parent.user\_data;

while (!(uart->uart\_device->SR & USART\_FLAG\_TXE));

uart->uart\_device->DR = c;

return 1;

}

static int stm32\_getc(struct rt\_serial\_device \*serial)

{

int ch;

struct stm32\_uart \*uart;

RT\_ASSERT(serial != RT\_NULL);

uart = (struct stm32\_uart \*)serial->parent.user\_data;

ch = -1;

if (uart->uart\_device->SR & USART\_FLAG\_RXNE)

{

ch = uart->uart\_device->DR & 0xff;

}

return ch;

}

/\*\*

\* DMA initialize by DMA\_InitStruct structure

\*

\* @param serial serial device

\* @param setting\_recv\_len setting receive length

\* @param mem\_base\_addr memory 0 base address for DMA stream

\*/

static void dma\_uart\_config(struct rt\_serial\_device \*serial, uint32\_t setting\_recv\_len,

void \*mem\_base\_addr)

{

struct stm32\_uart \*uart = (struct stm32\_uart \*) serial->parent.user\_data;

DMA\_InitTypeDef DMA\_InitStructure;

/\* rx dma config \*/

uart->dma.setting\_recv\_len = setting\_recv\_len;

DMA\_DeInit(uart->dma.rx\_stream);

while (DMA\_GetCmdStatus(uart->dma.rx\_stream) != DISABLE);

DMA\_InitStructure.DMA\_Channel = uart->dma.rx\_ch;

DMA\_InitStructure.DMA\_PeripheralBaseAddr = (uint32\_t) &(uart->uart\_device->DR);

DMA\_InitStructure.DMA\_Memory0BaseAddr = (uint32\_t)mem\_base\_addr;

DMA\_InitStructure.DMA\_DIR = DMA\_DIR\_PeripheralToMemory;

DMA\_InitStructure.DMA\_BufferSize = uart->dma.setting\_recv\_len;

DMA\_InitStructure.DMA\_PeripheralInc = DMA\_PeripheralInc\_Disable;

DMA\_InitStructure.DMA\_MemoryInc = DMA\_MemoryInc\_Enable;

DMA\_InitStructure.DMA\_PeripheralDataSize = DMA\_PeripheralDataSize\_Byte;

DMA\_InitStructure.DMA\_MemoryDataSize = DMA\_PeripheralDataSize\_Byte;

DMA\_InitStructure.DMA\_Mode = DMA\_Mode\_Circular;

DMA\_InitStructure.DMA\_Priority = DMA\_Priority\_High;

DMA\_InitStructure.DMA\_FIFOMode = DMA\_FIFOMode\_Disable;

DMA\_InitStructure.DMA\_FIFOThreshold = DMA\_FIFOThreshold\_Full;

DMA\_InitStructure.DMA\_MemoryBurst = DMA\_MemoryBurst\_Single;

DMA\_InitStructure.DMA\_PeripheralBurst = DMA\_PeripheralBurst\_Single;

DMA\_Init(uart->dma.rx\_stream, &DMA\_InitStructure);

}

/\*\*

\* Serial port receive idle process. This need add to uart idle ISR.

\*

\* @param serial serial device

\*/

static void dma\_uart\_rx\_idle\_isr(struct rt\_serial\_device \*serial) {

struct stm32\_uart \*uart = (struct stm32\_uart \*) serial->parent.user\_data;

rt\_size\_t recv\_total\_index, recv\_len;

rt\_base\_t level;

/\* disable interrupt \*/

level = rt\_hw\_interrupt\_disable();

recv\_total\_index = uart->dma.setting\_recv\_len - DMA\_GetCurrDataCounter(uart->dma.rx\_stream);

recv\_len = recv\_total\_index - uart->dma.last\_recv\_index;

uart->dma.last\_recv\_index = recv\_total\_index;

/\* enable interrupt \*/

rt\_hw\_interrupt\_enable(level);

if (recv\_len) rt\_hw\_serial\_isr(serial, RT\_SERIAL\_EVENT\_RX\_DMADONE | (recv\_len << 8));

/\* read a data for clear receive idle interrupt flag \*/

USART\_ReceiveData(uart->uart\_device);

}

/\*\*

\* DMA receive done process. This need add to DMA receive done ISR.

\*

\* @param serial serial device

\*/

static void dma\_rx\_done\_isr(struct rt\_serial\_device \*serial)

{

struct stm32\_uart \*uart = (struct stm32\_uart \*) serial->parent.user\_data;

rt\_size\_t recv\_len;

rt\_base\_t level;

if (DMA\_GetFlagStatus(uart->dma.rx\_stream, uart->dma.rx\_flag) != RESET)

{

/\* disable interrupt \*/

level = rt\_hw\_interrupt\_disable();

recv\_len = uart->dma.setting\_recv\_len - uart->dma.last\_recv\_index;

/\* reset last recv index \*/

uart->dma.last\_recv\_index = 0;

/\* enable interrupt \*/

rt\_hw\_interrupt\_enable(level);

if (recv\_len) rt\_hw\_serial\_isr(serial, RT\_SERIAL\_EVENT\_RX\_DMADONE | (recv\_len << 8));

/\* start receive data \*/

DMA\_ClearFlag(uart->dma.rx\_stream, uart->dma.rx\_flag);

}

}

/\*\*

\* Uart common interrupt process. This need add to uart ISR.

\*

\* @param serial serial device

\*/

static void uart\_isr(struct rt\_serial\_device \*serial)

{

struct stm32\_uart \*uart = (struct stm32\_uart \*) serial->parent.user\_data;

RT\_ASSERT(uart != RT\_NULL);

if(USART\_GetITStatus(uart->uart\_device, USART\_IT\_RXNE) != RESET)

{

rt\_hw\_serial\_isr(serial, RT\_SERIAL\_EVENT\_RX\_IND);

/\* clear interrupt \*/

USART\_ClearITPendingBit(uart->uart\_device, USART\_IT\_RXNE);

}

if(USART\_GetITStatus(uart->uart\_device, USART\_IT\_IDLE) != RESET)

{

dma\_uart\_rx\_idle\_isr(serial);

}

if (USART\_GetITStatus(uart->uart\_device, USART\_IT\_TC) != RESET)

{

/\* clear interrupt \*/

USART\_ClearITPendingBit(uart->uart\_device, USART\_IT\_TC);

}

if (USART\_GetFlagStatus(uart->uart\_device, USART\_FLAG\_ORE) == SET)

{

stm32\_getc(serial);

}

}

static const struct rt\_uart\_ops stm32\_uart\_ops =

{

stm32\_configure,

stm32\_control,

stm32\_putc,

stm32\_getc,

};

#if defined(RT\_USING\_UART1)

/\* UART1 device driver structure \*/

struct stm32\_uart uart1 =

{

USART1,

USART1\_IRQn,

{

DMA2\_Stream5,

DMA\_Channel\_4,

DMA\_FLAG\_TCIF5,

DMA2\_Stream5\_IRQn,

0,

},

};

struct rt\_serial\_device serial1;

void USART1\_IRQHandler(void)

{

/\* enter interrupt \*/

rt\_interrupt\_enter();

uart\_isr(&serial1);

/\* leave interrupt \*/

rt\_interrupt\_leave();

}

void DMA2\_Stream5\_IRQHandler(void) {

/\* enter interrupt \*/

rt\_interrupt\_enter();

dma\_rx\_done\_isr(&serial1);

/\* leave interrupt \*/

rt\_interrupt\_leave();

}

#endif /\* RT\_USING\_UART1 \*/

#if defined(RT\_USING\_UART2)

/\* UART2 device driver structure \*/

struct stm32\_uart uart2 =

{

USART2,

USART2\_IRQn,

{

DMA1\_Stream5,

DMA\_Channel\_4,

DMA\_FLAG\_TCIF5,

DMA1\_Stream5\_IRQn,

0,

0,

},

};

struct rt\_serial\_device serial2;

void USART2\_IRQHandler(void)

{

/\* enter interrupt \*/

rt\_interrupt\_enter();

uart\_isr(&serial2);

/\* leave interrupt \*/

rt\_interrupt\_leave();

}

void DMA1\_Stream5\_IRQHandler(void) {

/\* enter interrupt \*/

rt\_interrupt\_enter();

dma\_rx\_done\_isr(&serial2);

/\* leave interrupt \*/

rt\_interrupt\_leave();

}

#endif /\* RT\_USING\_UART2 \*/

#if defined(RT\_USING\_UART3)

/\* UART3 device driver structure \*/

struct stm32\_uart uart3 =

{

USART3,

USART3\_IRQn,

{

DMA1\_Stream1,

DMA\_Channel\_4,

DMA\_FLAG\_TCIF1,

DMA1\_Stream1\_IRQn,

0,

0,

},

};

struct rt\_serial\_device serial3;

void USART3\_IRQHandler(void)

{

/\* enter interrupt \*/

rt\_interrupt\_enter();

uart\_isr(&serial3);

/\* leave interrupt \*/

rt\_interrupt\_leave();

}

void DMA1\_Stream1\_IRQHandler(void) {

/\* enter interrupt \*/

rt\_interrupt\_enter();

dma\_rx\_done\_isr(&serial3);

/\* leave interrupt \*/

rt\_interrupt\_leave();

}

#endif /\* RT\_USING\_UART3 \*/

#if defined(RT\_USING\_UART4)

/\* UART4 device driver structure \*/

struct stm32\_uart uart4 =

{

UART4,

UART4\_IRQn,

{

DMA1\_Stream2,

DMA\_Channel\_4,

DMA\_FLAG\_TCIF2,

DMA1\_Stream2\_IRQn,

0,

0,

},

};

struct rt\_serial\_device serial4;

void UART4\_IRQHandler(void)

{

/\* enter interrupt \*/

rt\_interrupt\_enter();

uart\_isr(&serial4);

/\* leave interrupt \*/

rt\_interrupt\_leave();

}

void DMA1\_Stream2\_IRQHandler(void) {

/\* enter interrupt \*/

rt\_interrupt\_enter();

dma\_rx\_done\_isr(&serial4);

/\* leave interrupt \*/

rt\_interrupt\_leave();

}

#endif /\* RT\_USING\_UART4 \*/

#if defined(RT\_USING\_UART5)

/\* UART5 device driver structure \*/

struct stm32\_uart uart5 =

{

UART5,

UART5\_IRQn,

{

DMA1\_Stream0,

DMA\_Channel\_4,

DMA\_FLAG\_TCIF0,

DMA1\_Stream0\_IRQn,

0,

0,

},

};

struct rt\_serial\_device serial5;

void UART5\_IRQHandler(void)

{

/\* enter interrupt \*/

rt\_interrupt\_enter();

uart\_isr(&serial5);

/\* leave interrupt \*/

rt\_interrupt\_leave();

}

void DMA1\_Stream0\_IRQHandler(void) {

/\* enter interrupt \*/

rt\_interrupt\_enter();

dma\_rx\_done\_isr(&serial5);

/\* leave interrupt \*/

rt\_interrupt\_leave();

}

#endif /\* RT\_USING\_UART5 \*/

static void RCC\_Configuration(void)

{

#ifdef RT\_USING\_UART1

/\* Enable UART1 GPIO clocks \*/

RCC\_AHB1PeriphClockCmd(UART1\_GPIO\_RCC, ENABLE);

/\* Enable UART1 clock \*/

RCC\_APB2PeriphClockCmd(RCC\_APBPeriph\_UART1, ENABLE);

#endif /\* RT\_USING\_UART1 \*/

#ifdef RT\_USING\_UART2

/\* Enable UART2 GPIO clocks \*/

RCC\_AHB1PeriphClockCmd(UART2\_GPIO\_RCC, ENABLE);

/\* Enable UART2 clock \*/

RCC\_APB1PeriphClockCmd(RCC\_APBPeriph\_UART2, ENABLE);

#endif /\* RT\_USING\_UART1 \*/

#ifdef RT\_USING\_UART3

/\* Enable UART3 GPIO clocks \*/

RCC\_AHB1PeriphClockCmd(UART3\_GPIO\_RCC, ENABLE);

/\* Enable UART3 clock \*/

RCC\_APB1PeriphClockCmd(RCC\_APBPeriph\_UART3, ENABLE);

#endif /\* RT\_USING\_UART3 \*/

#ifdef RT\_USING\_UART4

/\* Enable UART4 GPIO clocks \*/

RCC\_AHB1PeriphClockCmd(UART4\_GPIO\_RCC, ENABLE);

/\* Enable UART4 clock \*/

RCC\_APB1PeriphClockCmd(RCC\_APBPeriph\_UART4, ENABLE);

#endif /\* RT\_USING\_UART4 \*/

#ifdef RT\_USING\_UART5

/\* Enable UART5 GPIO clocks \*/

RCC\_AHB1PeriphClockCmd(UART5\_GPIO\_RCC\_TX | UART5\_GPIO\_RCC\_RX, ENABLE);

/\* Enable UART5 clock \*/

RCC\_APB1PeriphClockCmd(RCC\_APBPeriph\_UART5, ENABLE);

#endif /\* RT\_USING\_UART5 \*/

}

static void GPIO\_Configuration(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AF;

GPIO\_InitStructure.GPIO\_OType = GPIO\_OType\_PP;

GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_UP;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_2MHz;

#ifdef RT\_USING\_UART1

/\* Configure USART1 Rx/tx PIN \*/

GPIO\_InitStructure.GPIO\_Pin = UART1\_GPIO\_RX | UART1\_GPIO\_TX;

/\* Connect alternate function \*/

GPIO\_PinAFConfig(UART1\_GPIO, UART1\_TX\_PIN\_SOURCE, GPIO\_AF\_USART1);

GPIO\_PinAFConfig(UART1\_GPIO, UART1\_RX\_PIN\_SOURCE, GPIO\_AF\_USART1);

GPIO\_Init(UART1\_GPIO, &GPIO\_InitStructure);

#endif /\* RT\_USING\_UART1 \*/

#ifdef RT\_USING\_UART2

/\* Configure USART2 Rx/tx PIN \*/

GPIO\_InitStructure.GPIO\_Pin = UART2\_GPIO\_RX | UART2\_GPIO\_TX;

/\* Connect alternate function \*/

GPIO\_PinAFConfig(UART2\_GPIO, UART2\_TX\_PIN\_SOURCE, GPIO\_AF\_USART2);

GPIO\_PinAFConfig(UART2\_GPIO, UART2\_RX\_PIN\_SOURCE, GPIO\_AF\_USART2);

GPIO\_Init(UART2\_GPIO, &GPIO\_InitStructure);

#endif /\* RT\_USING\_UART2 \*/

#ifdef RT\_USING\_UART3

/\* Configure USART3 Rx/tx PIN \*/

GPIO\_InitStructure.GPIO\_Pin = UART3\_GPIO\_TX | UART3\_GPIO\_RX;

/\* Connect alternate function \*/

GPIO\_PinAFConfig(UART3\_GPIO, UART3\_TX\_PIN\_SOURCE, GPIO\_AF\_USART3);

GPIO\_PinAFConfig(UART3\_GPIO, UART3\_RX\_PIN\_SOURCE, GPIO\_AF\_USART3);

GPIO\_Init(UART3\_GPIO, &GPIO\_InitStructure);

#endif /\* RT\_USING\_UART3 \*/

#ifdef RT\_USING\_UART4

/\* Configure USART4 Rx/tx PIN \*/

GPIO\_InitStructure.GPIO\_Pin = UART4\_GPIO\_TX | UART4\_GPIO\_RX;

/\* Connect alternate function \*/

GPIO\_PinAFConfig(UART4\_GPIO, UART4\_TX\_PIN\_SOURCE, GPIO\_AF\_UART4);

GPIO\_PinAFConfig(UART4\_GPIO, UART4\_RX\_PIN\_SOURCE, GPIO\_AF\_UART4);

GPIO\_Init(UART4\_GPIO, &GPIO\_InitStructure);

#endif /\* RT\_USING\_UART4 \*/

#ifdef RT\_USING\_UART5

/\* Configure USART5 TX PIN \*/

GPIO\_InitStructure.GPIO\_Pin = UART5\_GPIO\_TX;

GPIO\_PinAFConfig(UART5\_TX, UART5\_TX\_PIN\_SOURCE, GPIO\_AF\_UART5);

GPIO\_Init(UART5\_TX, &GPIO\_InitStructure);

/\* Configure USART5 RX PIN \*/

GPIO\_InitStructure.GPIO\_Pin = UART5\_GPIO\_RX;

GPIO\_PinAFConfig(UART5\_RX, UART5\_RX\_PIN\_SOURCE, GPIO\_AF\_UART5);

GPIO\_Init(UART5\_RX, &GPIO\_InitStructure);

#endif /\* RT\_USING\_UART5 \*/

}

static void NVIC\_Configuration(struct stm32\_uart \*uart)

{

NVIC\_InitTypeDef NVIC\_InitStructure;

/\* Enable the USART1 Interrupt \*/

NVIC\_InitStructure.NVIC\_IRQChannel = uart->irq;

NVIC\_InitStructure.NVIC\_IRQChannelPreemptionPriority = 1;

NVIC\_InitStructure.NVIC\_IRQChannelSubPriority = 0;

NVIC\_InitStructure.NVIC\_IRQChannelCmd = ENABLE;

NVIC\_Init(&NVIC\_InitStructure);

}

static void DMA\_Configuration(struct rt\_serial\_device \*serial) {

struct stm32\_uart \*uart = (struct stm32\_uart \*) serial->parent.user\_data;

struct rt\_serial\_rx\_fifo \*rx\_fifo = (struct rt\_serial\_rx\_fifo \*)serial->serial\_rx;

NVIC\_InitTypeDef NVIC\_InitStructure;

/\* enable transmit idle interrupt \*/

USART\_ITConfig(uart->uart\_device, USART\_IT\_IDLE , ENABLE);

/\* DMA clock enable \*/

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_DMA1, ENABLE);

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_DMA2, ENABLE);

/\* rx dma config \*/

dma\_uart\_config(serial, serial->config.bufsz, rx\_fifo->buffer);

DMA\_ClearFlag(uart->dma.rx\_stream, uart->dma.rx\_flag);

DMA\_ITConfig(uart->dma.rx\_stream, DMA\_IT\_TC, ENABLE);

USART\_DMACmd(uart->uart\_device, USART\_DMAReq\_Rx, ENABLE);

DMA\_Cmd(uart->dma.rx\_stream, ENABLE);

/\* rx dma interrupt config \*/

NVIC\_InitStructure.NVIC\_IRQChannel = uart->dma.rx\_irq\_ch;

NVIC\_InitStructure.NVIC\_IRQChannelPreemptionPriority = 0;

NVIC\_InitStructure.NVIC\_IRQChannelSubPriority = 0;

NVIC\_InitStructure.NVIC\_IRQChannelCmd = ENABLE;

NVIC\_Init(&NVIC\_InitStructure);

}

int rt\_hw\_usart\_init(void)

{

struct stm32\_uart \*uart;

struct serial\_configure config = RT\_SERIAL\_CONFIG\_DEFAULT;

RCC\_Configuration();

GPIO\_Configuration();

#ifdef RT\_USING\_UART1

uart = &uart1;

serial1.ops = &stm32\_uart\_ops;

serial1.config = config;

NVIC\_Configuration(&uart1);

/\* register UART1 device \*/

rt\_hw\_serial\_register(&serial1,

"uart1",

RT\_DEVICE\_FLAG\_RDWR | RT\_DEVICE\_FLAG\_INT\_RX | RT\_DEVICE\_FLAG\_DMA\_RX,

uart);

#endif /\* RT\_USING\_UART1 \*/

#ifdef RT\_USING\_UART2

uart = &uart2;

serial2.ops = &stm32\_uart\_ops;

serial2.config = config;

NVIC\_Configuration(&uart2);

/\* register UART1 device \*/

rt\_hw\_serial\_register(&serial2,

"uart2",

RT\_DEVICE\_FLAG\_RDWR | RT\_DEVICE\_FLAG\_INT\_RX | RT\_DEVICE\_FLAG\_DMA\_RX,

uart);

#endif /\* RT\_USING\_UART2 \*/

#ifdef RT\_USING\_UART3

uart = &uart3;

serial3.ops = &stm32\_uart\_ops;

serial3.config = config;

NVIC\_Configuration(&uart3);

/\* register UART3 device \*/

rt\_hw\_serial\_register(&serial3,

"uart3",

RT\_DEVICE\_FLAG\_RDWR | RT\_DEVICE\_FLAG\_INT\_RX | RT\_DEVICE\_FLAG\_DMA\_RX,

uart);

#endif /\* RT\_USING\_UART3 \*/

#ifdef RT\_USING\_UART4

uart = &uart4;

serial4.ops = &stm32\_uart\_ops;

serial4.config = config;

NVIC\_Configuration(&uart4);

/\* register UART4 device \*/

rt\_hw\_serial\_register(&serial4,

"uart4",

RT\_DEVICE\_FLAG\_RDWR | RT\_DEVICE\_FLAG\_INT\_RX | RT\_DEVICE\_FLAG\_DMA\_RX,

uart);

#endif /\* RT\_USING\_UART4 \*/

#ifdef RT\_USING\_UART5

uart = &uart5;

serial5.ops = &stm32\_uart\_ops;

serial5.config = config;

NVIC\_Configuration(&uart5);

/\* register UART5 device \*/

rt\_hw\_serial\_register(&serial5,

"uart5",

RT\_DEVICE\_FLAG\_RDWR | RT\_DEVICE\_FLAG\_INT\_RX | RT\_DEVICE\_FLAG\_DMA\_RX,

uart);

#endif /\* RT\_USING\_UART5 \*/

return 0;

}

INIT\_BOARD\_EXPORT(rt\_hw\_usart\_init);

#include <rthw.h>

#include <rtthread.h>

#include "board.h"

static void RCC\_Configuration(void)

{

RCC\_ClocksTypeDef rcc\_clocks;

RCC\_GetClocksFreq(&rcc\_clocks);

/\* 确认晶振完全起振 \*/

assert\_param(rcc\_clocks.HCLK\_Frequency == 168000000);

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOA | RCC\_AHB1Periph\_GPIOB | \

RCC\_AHB1Periph\_GPIOC | RCC\_AHB1Periph\_GPIOD | \

RCC\_AHB1Periph\_GPIOE | RCC\_AHB1Periph\_GPIOF | \

RCC\_AHB1Periph\_GPIOG, ENABLE);

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_DMA1, ENABLE);

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_ADC1, ENABLE);

}

void NVIC\_Configuration(void)

{

#if VECT\_TAB\_USER

/\* Set the Vector Table base location by user application firmware definition \*/

NVIC\_SetVectorTable(NVIC\_VectTab\_FLASH, USER\_VECTOR\_OFFSET);

#else

#ifdef VECT\_TAB\_RAM

/\* Set the Vector Table base location at 0x20000000 \*/

NVIC\_SetVectorTable(NVIC\_VectTab\_RAM, 0x0);

#else /\* VECT\_TAB\_FLASH \*/

/\* Set the Vector Table base location at 0x08000000 \*/

NVIC\_SetVectorTable(NVIC\_VectTab\_FLASH, 0x0);

#endif /\* VECT\_TAB\_RAM \*/

#endif /\* VECT\_TAB\_USER \*/

}

static void GPIO\_Configuration(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

/\* JTAG-DP Disabled and SW-DP Enabled \*/

/\*\*

\* 使能SWD、失能JTAG功能，参考寄存器手册“8.3.2 I/O pin multiplexer and mapping”中的

\* <GPIO>小节，直接配置成普通GPIO即可

\*/

/\* PA13:SWDIO PA14:SWCLK for debug, can't set to output mode \*/

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_All & (~(GPIO\_Pin\_13 | GPIO\_Pin\_14));

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_OUT;

GPIO\_InitStructure.GPIO\_OType = GPIO\_OType\_PP;

GPIO\_InitStructure.GPIO\_PuPd = GPIO\_PuPd\_NOPULL;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_2MHz;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_All;

GPIO\_Init(GPIOB, &GPIO\_InitStructure);

GPIO\_Init(GPIOC, &GPIO\_InitStructure);

GPIO\_Init(GPIOD, &GPIO\_InitStructure);

GPIO\_Init(GPIOE, &GPIO\_InitStructure);

GPIO\_Init(GPIOF, &GPIO\_InitStructure);

GPIO\_Init(GPIOG, &GPIO\_InitStructure);

}

static void IWDG\_Configuration(void)

{

IWDG\_WriteAccessCmd(IWDG\_WriteAccess\_Enable);

IWDG\_SetPrescaler(IWDG\_Prescaler\_64);

IWDG\_SetReload(1875);

IWDG\_ReloadCounter();

IWDG\_Enable();

}

void IWDG\_Feed(void)

{

IWDG\_ReloadCounter();

}

void assert\_failed(u8\* file, u32 line)

{

/\* User can add his own implementation to report the file name and line number,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* Infinite loop \*/

rt\_kprintf("assert failed at %s:%d \n", file, line);

while (1) {

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Name : SysTick\_Configuration

\* Description : Configures the SysTick for OS tick.

\* Input : None

\* Output : None

\* Return : None

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void SysTick\_Configuration(void)

{

RCC\_ClocksTypeDef rcc\_clocks;

rt\_uint32\_t cnts;

RCC\_GetClocksFreq(&rcc\_clocks);

cnts = (rt\_uint32\_t)rcc\_clocks.HCLK\_Frequency / RT\_TICK\_PER\_SECOND;

SysTick\_Config(cnts);

SysTick\_CLKSourceConfig(SysTick\_CLKSource\_HCLK);

}

/\*\*

\* This is the timer interrupt service routine.

\*

\*/

void SysTick\_Handler(void)

{

/\* enter interrupt \*/

rt\_interrupt\_enter();

rt\_tick\_increase();

/\* leave interrupt \*/

rt\_interrupt\_leave();

}

/\*\*

\* This function will initial STM32 board.

\*/

void rt\_hw\_board\_init()

{

RCC\_Configuration();

NVIC\_Configuration();

GPIO\_Configuration();

#ifdef RELEASE

IWDG\_Configuration();

#endif /\* RELEASE \*/

/\* Configure the SysTick \*/

SysTick\_Configuration();

#ifdef RT\_USING\_COMPONENTS\_INIT

rt\_components\_board\_init();

#endif

#ifdef RT\_USING\_CONSOLE

rt\_console\_set\_device(RT\_CONSOLE\_DEVICE\_NAME);

#endif

}

#ifdef RT\_USING\_FINSH

#include <finsh.h>

void reboot(uint8\_t argc, char \*\*argv) {

NVIC\_SystemReset();

}

FINSH\_FUNCTION\_EXPORT(reboot, reboot board);

#ifdef FINSH\_USING\_MSH

MSH\_CMD\_EXPORT(reboot, Reboot System);

#endif /\* FINSH\_USING\_MSH \*/

#endif /\* RT\_USING\_FINSH \*/

#include <string.h>

#include <time.h>

#include <rtthread.h>

#include <board.h>

#include <drivers/rtc.h>

static struct rt\_device rtc;

static rt\_err\_t rt\_rtc\_open(rt\_device\_t dev, rt\_uint16\_t oflag)

{

if (dev->rx\_indicate != RT\_NULL)

{

/\* Open Interrupt \*/

}

return RT\_EOK;

}

static rt\_size\_t rt\_rtc\_read(rt\_device\_t dev, rt\_off\_t pos, void\* buffer, rt\_size\_t size)

{

return 0;

}

static rt\_err\_t rt\_rtc\_control(rt\_device\_t dev, int cmd, void \*args)

{

time\_t \*time;

RTC\_TimeTypeDef RTC\_TimeStructure;

RTC\_DateTypeDef RTC\_DateStructure;

struct tm time\_temp;

RT\_ASSERT(dev != RT\_NULL);

memset(&time\_temp, 0, sizeof(struct tm));

switch (cmd)

{

case RT\_DEVICE\_CTRL\_RTC\_GET\_TIME:

time = (time\_t \*)args;

/\* Get the current Time \*/

RTC\_GetTime(RTC\_Format\_BIN, &RTC\_TimeStructure);

RTC\_GetDate(RTC\_Format\_BIN, &RTC\_DateStructure);

/\* Years since 1900 : 0-99 range \*/

time\_temp.tm\_year = RTC\_DateStructure.RTC\_Year + 2000 - 1900;

/\* Months \*since\* january 0-11 : RTC\_Month\_Date\_Definitions 1 - 12 \*/

time\_temp.tm\_mon = RTC\_DateStructure.RTC\_Month - 1;

/\* Day of the month 1-31 : 1-31 range \*/

time\_temp.tm\_mday = RTC\_DateStructure.RTC\_Date;

/\* Hours since midnight 0-23 : 0-23 range \*/

time\_temp.tm\_hour = RTC\_TimeStructure.RTC\_Hours;

/\* Minutes 0-59 : the 0-59 range \*/

time\_temp.tm\_min = RTC\_TimeStructure.RTC\_Minutes;

/\* Seconds 0-59 : the 0-59 range \*/

time\_temp.tm\_sec = RTC\_TimeStructure.RTC\_Seconds;

\*time = mktime(&time\_temp);

break;

case RT\_DEVICE\_CTRL\_RTC\_SET\_TIME:

{

const struct tm\* time\_new;

time = (time\_t \*)args;

time\_new = localtime(time);

/\* 0-99 range : Years since 1900 \*/

RTC\_DateStructure.RTC\_Year = time\_new->tm\_year + 1900 - 2000;

/\* RTC\_Month\_Date\_Definitions 1 - 12 : Months \*since\* january 0-11 \*/

RTC\_DateStructure.RTC\_Month = time\_new->tm\_mon + 1;

/\* 1-31 range : Day of the month 1-31 \*/

RTC\_DateStructure.RTC\_Date = time\_new->tm\_mday;

/\* 1 - 7 : Days since Sunday (0-6) \*/

RTC\_DateStructure.RTC\_WeekDay = time\_new->tm\_wday + 1;

/\* 0-23 range : Hours since midnight 0-23 \*/

RTC\_TimeStructure.RTC\_Hours = time\_new->tm\_hour;

/\* the 0-59 range : Minutes 0-59 \*/

RTC\_TimeStructure.RTC\_Minutes = time\_new->tm\_min;

/\* the 0-59 range : Seconds 0-59 \*/

RTC\_TimeStructure.RTC\_Seconds = time\_new->tm\_sec;

/\* Allow access to RTC \*/

PWR\_BackupAccessCmd(ENABLE);

/\* Set Current Time and Date \*/

RTC\_SetTime(RTC\_Format\_BIN, &RTC\_TimeStructure);

RTC\_SetDate(RTC\_Format\_BIN, &RTC\_DateStructure);

}

break;

}

return RT\_EOK;

}

static int RTC\_Configuration(void)

{

RTC\_TimeTypeDef RTC\_TimeStructure;

RTC\_DateTypeDef RTC\_DateStructure;

RTC\_InitTypeDef RTC\_InitStructure;

#define INIT\_FLAG 0x5050

/\* Enable the PWR clock \*/

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_PWR, ENABLE);

/\* Allow access to RTC \*/

PWR\_BackupAccessCmd(ENABLE);

/\* Select the RTC Clock Source \*/

RCC\_RTCCLKConfig(RCC\_RTCCLKSource\_HSE\_Div8);

/\* Enable the RTC Clock \*/

RCC\_RTCCLKCmd(ENABLE);

/\* It's the first configuration \*/

if (RTC\_ReadBackupRegister(RTC\_BKP\_DR0) != INIT\_FLAG) {

/\* Calendar Configuration \*/

RTC\_InitStructure.RTC\_AsynchPrediv = 100 - 1;

RTC\_InitStructure.RTC\_SynchPrediv = 10000 - 1;

RTC\_InitStructure.RTC\_HourFormat = RTC\_HourFormat\_24;

RTC\_Init(&RTC\_InitStructure);

/\* Set the Time \*/

RTC\_TimeStructure.RTC\_Hours = 0x08;

RTC\_TimeStructure.RTC\_Minutes = 0x00;

RTC\_TimeStructure.RTC\_Seconds = 0x00;

RTC\_SetTime(RTC\_Format\_BIN, &RTC\_TimeStructure);

/\* Set the Date \*/

RTC\_DateStructure.RTC\_Month = RTC\_Month\_December;

RTC\_DateStructure.RTC\_Date = 1;

RTC\_DateStructure.RTC\_Year = 17;

RTC\_DateStructure.RTC\_WeekDay = RTC\_Weekday\_Friday;

RTC\_SetDate(RTC\_Format\_BIN, &RTC\_DateStructure);

/\* Set initialized complete flag \*/

RTC\_WriteBackupRegister(RTC\_BKP\_DR0, INIT\_FLAG);

}

return 0;

}

clock\_t clock(void) {

return rt\_tick\_get();

}

int rt\_hw\_rtc\_init(void)

{

rtc.type = RT\_Device\_Class\_RTC;

RTC\_Configuration();

/\* register rtc device \*/

rtc.init = RT\_NULL;

rtc.open = rt\_rtc\_open;

rtc.close = RT\_NULL;

rtc.read = rt\_rtc\_read;

rtc.write = RT\_NULL;

rtc.control = rt\_rtc\_control;

/\* no private \*/

rtc.user\_data = RT\_NULL;

rt\_device\_register(&rtc, "rtc", RT\_DEVICE\_FLAG\_RDWR);

return 0;

}

INIT\_BOARD\_EXPORT(rt\_hw\_rtc\_init);