Project Documentation

Lishchynska Ruslana
2019/2020 a.y.

Content

1 Introduction

The project that is presented here is dedicated to usage of multithreading technologies in the direction of embedded systems. The application allows users to play a simple game in which the user should catch the Red LED (1) on each level. With the increasing of the game level, the speed of blinker is increased as well.

2 Hardware

In this project STM32F3-DISCOVERY was used as hardware with working parts such as on-board button and LEDs. Serial Driver as BaseSequentialStream was used in order to enable communication.

3 Program Architecture

3.1 Threads

The threads are presented by the LEDs in this project. Each thread is dedicated to its own LED. RT Synchronous Messages were used to provide ChibiOS's inner scheduler layer a message-passing mechanism. This mechanism was used to implement the Synchronous Messages high level functionality, the messages are exchanged with almost no overheap. Scheduling in main function was made with normalprio at the same priority level. RT Virtual Timers were used to get times of pushing the button and Red LED's starting of thread.

3.2 Input

The interaction can be done by the usage of User Button on STM32F3-DISCOVERY. The blinker is working with the lowest speed on the first level, so the task is to press the button at the same time as Red LED appearance. If the interaction is successful, the program goes to the next level till the win.

3.3 Output

Each result of user's interaction is displayed on the terminal, where the level and times of Red LED and pressing of the button are displayed. The winning level is 5. In the case of win, all LEDs will light up. In opposite situation, when the user was not able to interact in the needed way, the level is decreased to 0 and the user should start the game from the beginning.

4 User Manual

1. Reset the STM32F3-DISCOVERY with the Reset button.
2. When the Red LED lighting up, press the blue User Button on the board.
3. Repeat step 2 till the winning.

5 Appendix 1: Commented Code

```
/*
 * project.c
 *
 *  Created on: 9 Dec. 2019
 *      Author: ruslana
 */

#include "ch.h"
#include "hal.h"
#include "chprintf.h" // streams.mk should be included to Makefile

#define cls(chp)  chprintf(chp, "\033[2J\033[1;1H")

// Defining possible directions of circle speed's change
enum dir {
  up,
  down
} d = down;

size_t s;
thread_t *console[8];
/*===========================================================================
==*/
/* Generic code.
 *
 */
/*===========================================================================
==*/

static BaseSequentialStream* chp = (BaseSequentialStream*)&SD1;

#define ON_TIME 50
#define OFF_TIME 150
static int factor = 5;
static int count = 0; // Level's counter
static systime_t ledTime; // Storing value of Red LED thread launching,
time in system ticks
static systime_t now; // Storing time of pressing button, time in system
ticks
static int delay = 35000; // Defining delay between ledTime and now, time
in system ticks

/*
 * Red LED (1) blinker thread, times are in milliseconds.
 */
static THD_WORKING_AREA(waThread1, 512);
static THD_FUNCTION(Thread1, arg) {
  static int i = 0;
  chRegSetThreadName("blinker1");
  while (true) {
    thread_t *tp = chMsgWait();
    const char *msg = (const char *)chMsgGet(tp);
    chMsgRelease(tp, MSG_OK);
    palSetPad(GPIOE, 13U);
    chThdSleepMilliseconds(ON_TIME*factor);
    palClearPad(GPIOE, 13U);
    chThdSleepMilliseconds(OFF_TIME*factor);
    (void)chMsgSend(console[1], (msg_t)"System started.");
    ledTime = chVTGetSystemTimeX(); // obtaining time of thread's starting
  }
}
/*
 * ORANGE LED (2) blinker thread, times are in milliseconds.
 */
static THD_WORKING_AREA(waThread2, 512);
static THD_FUNCTION(Thread2, arg) {
  (void*)arg;
  chRegSetThreadName("blinker2");
  while (true) {
    thread_t *tp = chMsgWait();
    const char *msg = (const char *)chMsgGet(tp);
    chMsgRelease(tp, MSG_OK);
    palSetPad(GPIOE, 14U);
    chThdSleepMilliseconds(ON_TIME*factor);
    palClearPad(GPIOE, 14U);
    chThdSleepMilliseconds(OFF_TIME*factor);
    (void)chMsgSend(console[2], (msg_t)"System started.");

  }
}

/*
 * Green LED (3) blinker thread, times are in milliseconds.
 */
static THD_WORKING_AREA(waThread3, 512);
static THD_FUNCTION(Thread3, arg) {
  chRegSetThreadName("blinker3");
  while (true) {
    thread_t *tp = chMsgWait();
    const char *msg = (const char *)chMsgGet(tp);
    chMsgRelease(tp, MSG_OK);
    palSetPad(GPIOE, 15U);
    chThdSleepMilliseconds(ON_TIME*factor);
    palClearPad(GPIOE, 15U);
    chThdSleepMilliseconds(OFF_TIME*factor);
    (void)chMsgSend(console[3], (msg_t)"System started.");
  }
}
```

```
/*
 * BLUE LED (4) blinker thread, times are in milliseconds.
 */
static THD_WORKING_AREA(waThread4, 512);
static THD_FUNCTION(Thread4, arg) {
  chRegSetThreadName("blinker4");
  while (true) {
    thread_t *tp = chMsgWait();
    const char *msg = (const char *)chMsgGet(tp);
    chMsgRelease(tp, MSG_OK);
    palSetPad(GPIOE, 8U);
    chThdSleepMilliseconds(ON_TIME*factor);
    palClearPad(GPIOE, 8U);
    chThdSleepMilliseconds(OFF_TIME*factor);
    (void)chMsgSend(console[4], (msg_t)"System started.");
  }
}

/*
 * Red LED (5) blinker thread, times are in milliseconds.
 */
static THD_WORKING_AREA(waThread5, 512);
static THD_FUNCTION(Thread5, arg) {
  chRegSetThreadName("blinker5");
  while (true) {
    thread_t *tp = chMsgWait();
    const char *msg = (const char *)chMsgGet(tp);
    chMsgRelease(tp, MSG_OK);
    palSetPad(GPIOE, 9U);
    chThdSleepMilliseconds(ON_TIME*factor);
    palClearPad(GPIOE, 9U);
    chThdSleepMilliseconds(OFF_TIME*factor);
    (void)chMsgSend(console[5], (msg_t)"System started.");
  }
}
/*
 * Orange LED (6) blinker thread, times are in milliseconds.
 */
static THD_WORKING_AREA(waThread6, 512);
static THD_FUNCTION(Thread6, arg) {
  chRegSetThreadName("blinker6");
  while (true) {
    thread_t *tp = chMsgWait();
    const char *msg = (const char *)chMsgGet(tp);
    chMsgRelease(tp, MSG_OK);
    palSetPad(GPIOE, 10U);
    chThdSleepMilliseconds(ON_TIME*factor);
    palClearPad(GPIOE, 10U);
    chThdSleepMilliseconds(OFF_TIME*factor);
    (void)chMsgSend(console[6], (msg_t)"System started.");
  }
}

/*
 * GREEN LED (7) blinker thread, times are in milliseconds.
 */
static THD_WORKING_AREA(waThread7, 512);
static THD_FUNCTION(Thread7, arg) {
  chRegSetThreadName("blinker7");
  while (true) {
    thread_t *tp = chMsgWait();
    const char *msg = (const char *)chMsgGet(tp);
    chMsgRelease(tp, MSG_OK);
    palSetPad(GPIOE, 11U);
    chThdSleepMilliseconds(ON_TIME*factor);
    palClearPad(GPIOE, 11U);
    chThdSleepMilliseconds(OFF_TIME*factor);
    (void)chMsgSend(console[7], (msg_t)"System started.");
  }
}

/*
 * BLUE LED (8) blinker thread, times are in milliseconds.
 */
static THD_WORKING_AREA(waThread8, 512);
static THD_FUNCTION(Thread8, arg) {
  chRegSetThreadName("blinker8");
  while (true) {
    thread_t *tp = chMsgWait();
    const char *msg = (const char *)chMsgGet(tp);
    chMsgRelease(tp, MSG_OK);
    palSetPad(GPIOE, 12U);
    chThdSleepMilliseconds(ON_TIME*factor);
    palClearPad(GPIOE, 12U);
    chThdSleepMilliseconds(OFF_TIME*factor);
    (void)chMsgSend(console[0], (msg_t)"System started.");

  }
}

/*
 * Application entry point.
 */

int main(void) {

  /*
   * System initializations.
   * - HAL initialization, this also initializes the configured device
drivers
   *   and performs the board-specific initializations.
   * - Kernel initialization, the main() function becomes a thread and the
   *   RTOS is active.
   */
  halInit();
  chSysInit();

  /* Activates the serial driver 1 using the driver default
configuration.*/
  sdStart(&SD1, NULL);

  chprintf(chp, "main %010d\r\n", chVTGetSystemTime());

  /* Creates the blinker threads.*/

  console[4] = chThdCreateStatic(waThread5, sizeof(waThread5), NORMALPRIO +
1,
                                 Thread5, NULL);
  console[5] = chThdCreateStatic(waThread6, sizeof(waThread6), NORMALPRIO +
1,
                                 Thread6, NULL);
  console[6] = chThdCreateStatic(waThread7, sizeof(waThread7), NORMALPRIO +
1,
                                 Thread7, NULL);
  console[1] = chThdCreateStatic(waThread2, sizeof(waThread2), NORMALPRIO +
1,
                                 Thread2, NULL);
  console[2] = chThdCreateStatic(waThread3, sizeof(waThread3), NORMALPRIO +
1,
                                 Thread3, NULL);
  console[7] = chThdCreateStatic(waThread8, sizeof(waThread8), NORMALPRIO +
1,
                                 Thread8, NULL);
  console[0] = chThdCreateStatic(waThread1, sizeof(waThread1), NORMALPRIO +
1,
                                 Thread1, NULL);
  console[3] = chThdCreateStatic(waThread4, sizeof(waThread4), NORMALPRIO +
1,
                                 Thread4, NULL);

  chThdSleepMilliseconds(100);
  // Activates the first thread
  (void)chMsgSend(console[0], (msg_t)"System started.");

  /* Enabling event on falling edge of PA0 signal.*/
  palEnablePadEvent(GPIOA, 0U, PAL_EVENT_MODE_FALLING_EDGE);

  chprintf(chp, "The task is to catch Red LED (1) blinker \r\n");

  while (true) {
    palWaitPadTimeout(GPIOA, 0U, TIME_INFINITE);
      /*
```

```
       * Checking the value of direction to increase circle's speed.
       * */
      if (d == down) {
        if (factor > 0) {
          factor = factor - 1;
          count = count + 1;
          now = chVTGetSystemTimeX(); // Obtaining the time of pressing
button
        }
      }
      /*
       * Checking the condition for moving to the next level.
       * If all possible levels are achieved, the game is finished with
win.
       * */
      if (now - ledTime < delay) {
        if (count < 5) {
          chprintf(chp, "LEVEL  %d\r\n", count);
          chprintf(chp, "Led time  %d\r\n", ledTime);
          chprintf(chp,"Pressing time %d\r\n", now);
          delay  = delay - 5000; // delay should decrease for each level
          chThdSleepMilliseconds(200);
        }
        else {
          chprintf(chp, "WIN \n");
          chThdSleepMilliseconds(200);
        }
      }
      /*
       * If the conditions are not met, the levels are zeroed and the game
restarts.
       * */
      else {
        chprintf(chp, "Try again... \n");
        chprintf(chp, "Led time  %d\r\n", ledTime);
        chprintf(chp,"Pressing time %d\r\n", now);
        factor = 5;
        count = 0;
        chThdSleepMilliseconds(200);
      }
   }

  chThdSleepMilliseconds(100);
}
```