K Tolliver

Lab 4 Results

1) Complete the code so that each LED is turned on and off each time their process is run. Submit your code via i-learn. When your code is complete you should see three LEDs flashing at different rates. The idle LED should flash every 100 ms or so, one will flash every second, the third every 1.5 seconds.

///Up to this many tasks can be run, in addition to the idle task

#define MAXTASKS 8

uint8\_t pulse1\_pin = YELLOW\_LED;

uint8\_t pulse2\_pin = GREEN\_LED;

uint8\_t idle\_pin = RED\_LED;

int pin1\_status = LOW;

int pin2\_status= HIGH;

///A task callback function

typedef void (\*task\_cb)();

/\*\*

\* Initialise the scheduler. This should be called once in the setup routine.

\*/

void Scheduler\_Init();

void Scheduler\_StartTask(int16\_t delay, int16\_t period, task\_cb task);

/\*\*

\* Go through the task list and run any tasks that need to be run. The main function should

\* simply be this function called as often as possible, plus any low-priority code that you want

\* to run sporadically.

\*/

uint32\_t Scheduler\_Dispatch();

//#include <avr/interrupt.h>

typedef struct

{

int32\_t period;

int32\_t remaining\_time;

uint8\_t is\_running;

task\_cb callback;

} task\_t;

task\_t tasks[MAXTASKS];

uint32\_t last\_runtime;

void Scheduler\_Init()

{ last\_runtime = millis(); }

void Scheduler\_StartTask(int16\_t delay, int16\_t period, task\_cb task)

{

static uint8\_t id = 0;

if (id < MAXTASKS)

{

tasks[id].remaining\_time = delay;

tasks[id].period = period;

tasks[id].is\_running = 1;

tasks[id].callback = task;

id++; } }

uint32\_t Scheduler\_Dispatch()

{

for( ; ; )

{

uint32\_t thisTime = millis();

uint32\_t runTime = thisTime - last\_runtime;

last\_runtime = thisTime;

task\_cb task = NULL; // task\_cb taskSuspendAll ();

uint32\_t idleTime = 0xFFFFFFFF;

**//MyScheduler**

**for (uint8\_t i=0; i< MAXTASKS; i++)**

**{**

**if(tasks[i].is\_running)**

**{**

**tasks[i].remaining\_time = tasks[i].remaining\_time-runTime;**

**if(tasks[i].remaining\_time <= 0)**

**{**

**if(task == NULL)**

**{**

**task = tasks[i].callback;**

**tasks[i].remainingTime = tasks[i].idleTime – tasks[i].period;**

**}**

**idleTime = 0;**

**}**

**else {idleTime < remainingTime}**

**}}**

**if(task != NULL) {task();}**

**}**

return idleTime;

}

// task function for PulsePin task

void pulse\_pin1\_task()

{

Serial.println("Pin1 task");

if(pin1\_status == LOW)

{

digitalWrite(pulse1\_pin, HIGH);

pin1\_status = HIGH;

}

else

{

digitalWrite(pulse1\_pin, LOW);

pin1\_status = LOW; } }

// task function for PulsePin task

void pulse\_pin2\_task()

{

Serial.println("Pin2 task");

if(pin2\_status == LOW)

{ digitalWrite(pulse2\_pin, HIGH);

pin2\_status = HIGH; }

else

{ digitalWrite(pulse2\_pin, LOW);

pin2\_status = LOW; }}

// idle task

void idle(uint32\_t idle\_period)

{

// This function can perform some low-priority task while the scheduler has nothing to do

// It should return before the idle period (measured in ms) has expired. For example, it could sleep or respond to I/O.

// example idle function that just pulses a pin.

Serial.println("idle process");

digitalWrite(idle\_pin, HIGH);

delay(idle\_period);

digitalWrite(idle\_pin, LOW);

}

void setup()

{

pinMode(pulse1\_pin, OUTPUT);

pinMode(pulse2\_pin, OUTPUT);

pinMode(idle\_pin, OUTPUT);

Serial.begin(115200);

Scheduler\_Init();

Serial.println("Scheduler started");

// Start task arguments are:

// start offset in ms, period in ms, function callback

Scheduler\_StartTask(0, 100, pulse\_pin1\_task);

Scheduler\_StartTask(0, 1500, pulse\_pin2\_task);

}

void loop()

{

uint32\_t idle\_period = Scheduler\_Dispatch();

if (idle\_period) { idle(idle\_period); }}

2) Hook up the Logic analyzer to the three pins. Capture a trace. Include it in the Lab report.

