Lab Guide: Non-Preemptive Scheduler and the MSP432

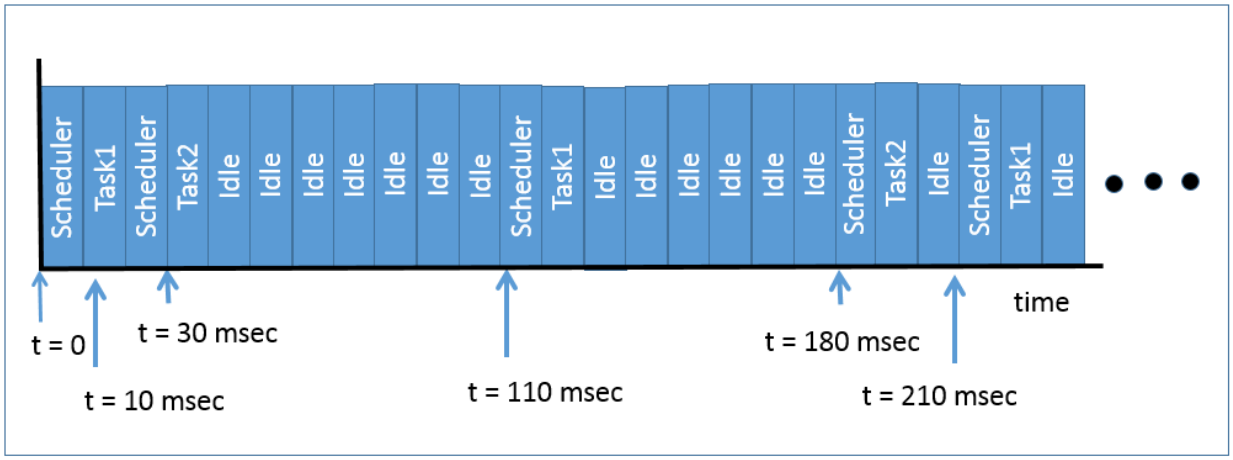
Introduction

In this lab you will create a scheduler on your MSP432 that will execute two tasks at predetermined rates. To do this you will create a scheduler. A scheduler is a set of code that keeps a queue of possible processes/threads/tasks and decides which of these should have time on the processor.

In order to keep this simple you will not be implementing a pre-emptive scheduler. A pre-emptive scheduler is a scheduler that can interrupt a process/thread in the middle of execution, save the context (Program Counter/Stack Pointer/Register States), run the scheduler to decide with process/thread/task should have time on the processor, and then restore that process/thread/task state, and run the process.

Your scheduler will be a non-pre-emptive scheduler, which means once a process/thread/task runs it will continue until it is done, then the scheduler will determine which process/thread/task should run. A specific example is that you have three process/thread/tasks you want to run. One you want to run every 100 msec. The second you want to run every 150 msec. The third you want to run when the other two are not running. In this case you will use a scheduler to make this happen. When the scheduler runs it will look for any process/thread/task that can run. It will chose the first one in the queue. When that task is run, the scheduler will run the next task that is ready. If no task is ready, then the scheduler will go into an idle task for the amount of time that it predicts will occur before another task is ready to run.

If each task to 10 msecs to run the execution graph would look something like this:



Now that you know what you're trying to do, let's get started.

Constructing the Hardware

For this lab you'll use the LEDs already available on the MSP432P401R. There is a RED_LED, a GREEN_LED, and a YELLOW_LED.

The circuit will flash one LED when task1 is running, a different LED when task2 is running, and a different LED when the idle task is running.

Basics of a Scheduler

Remember that a scheduler is a fairly simple piece of code that also includes some simple data structures. One of the key data structures is the process queue. This queue is a simple array that holds a set of processes that can be run on the processor. The task_type data structure will look like this:

```
typedef struct
{
int period; // How often do you want to run our task
int remaining_time; // How long until you run the task again
int is_running; // Is the task available to run
task_function callback;
} task_type;
```

Each process/thread/task will have a

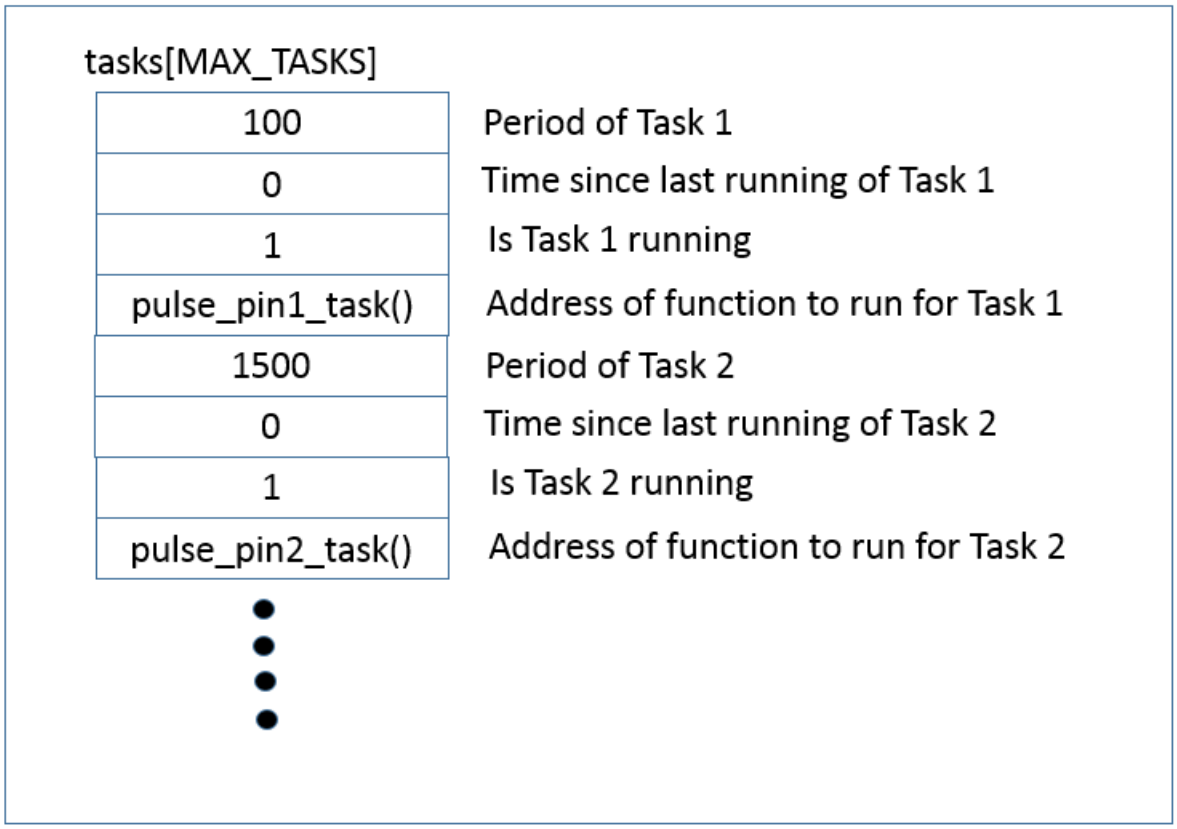
period – This variable holds how often the process wants to run in milliseconds.
remaining-time – This variable holds the amount of time has expired since the last time you ran the task
is_running; - This variable lets us turn on or off a specific thread/process/task.
callback – This holds the address of the function you want to run when the process is scheduled. The data type is **task_function** is declared as:

```
typedef void (*task_function)();
```

More about that later. Once you have the task_type data structure you can declare an array of them, like this:

```
task_t tasks[MAXTASKS];
```

Just to review, if you were to look out in memory after you had created two tasks you might see something like this:



To finish the data definitions, you'll also need some additional commands:

```
#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;

typedef void (*task_cb)();

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;
```

These set the maximum number of tasks to 8, sets output pin 3 to a pulse1_pin, sets output pin 4 to pulse2_pin, sets output pin 7 to idle_pin. These will toggle each time that process runs. The last_runtime is a global time keeper that keeps track of the last time the scheduler was run.

Now on to the functions required by the scheduler. Each of these is a prototype (A definition of the function that you will code later.) There are three functions.

void Scheduler_Init(); - This initializes the scheduler start time. This is the only initialization that the scheduler needs.

void Scheduler_StartTask(int16_t delay, int16_t period, task_cb task); - This adds a task/thread/process to the queue. The delay tells the system how long until the Scheduler should run this task/process/thread for the first time, the period sets how often the task should run, and the task is a pointer to the function that should run when the task is ready.

uint32_t Scheduler_Dispatch(); - This is the actual scheduler code. Here you will walk through the queue to see if a process is ready to run. If it is it will run that process. If not it will return the amount of time until the next task will be run. (Hint – This function will use a for loop to step through the queue. Check to see if the process is running, then compare the time remaining to run with the time since the scheduler last ran. If it is time to run, then set the task variable to the callback of the process/thread/function. During the loop you'll also want to find the task that will be ready to run next, and keep track of that time with the delayTime variable. After the loop if a task is ready to run, then run that task by calling task(). If no task is ready to run return the delayTime.)

And here is the code for the three functions:

```
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()
{

  uint32_t thisTime = millis();
  uint32_t runTime = thisTime - last_runtime;
  last_runtime = thisTime;
  task_cb task = NULL;
  uint32_t idleTime = 0xFFFFFFFF;


//  Your code goes here

return idleTime;
}
```

Now you'll need some code that will flash some LEDs on and off based on what process is running. Here is that code:

```
// task function for PulsePin1 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 PulsePin2 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)
{
   Serial.println("idle process");
   digitalWrite(idle_pin, HIGH);
   delay(idle_period);
   digitalWrite(idle_pin, LOW);
}
```

Now that you have the support functions you'll need to add the functions for the setup() and loop() commands for Energia. Here is the setup function. In this section you'll configure our LEDs, initialize the scheduler, and the add to tasks/threads/processes to the queue.

```
void setup()
{
   pinMode(pulse1_pin, OUTPUT);
   pinMode(pulse2_pin, OUTPUT);
   pinMode(idle_pin, OUTPUT);
   Serial.begin(115200);

   Scheduler_Init();
```

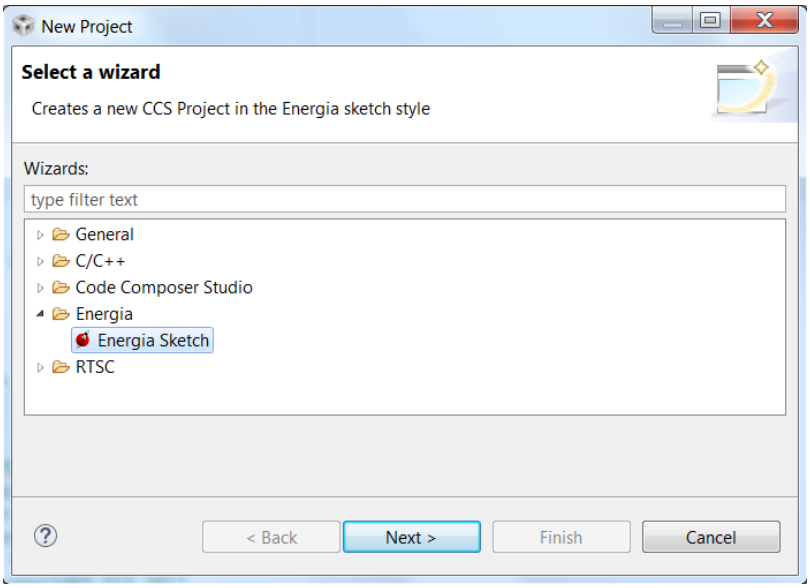
```
    Serial.println("Scheduler started");
        // Start task arguments are:
        //                  start offset in ms, period in ms, function callback

    Scheduler_StartTask(0, 100, pulse_pin1_task);
    Scheduler_StartTask(0, 1500, pulse_pin2_task);
}
```

Finally, here is the loop() function. Each time through the loop you'll run the scheduler. If it returns an idle time, then you'll run the idle loop.

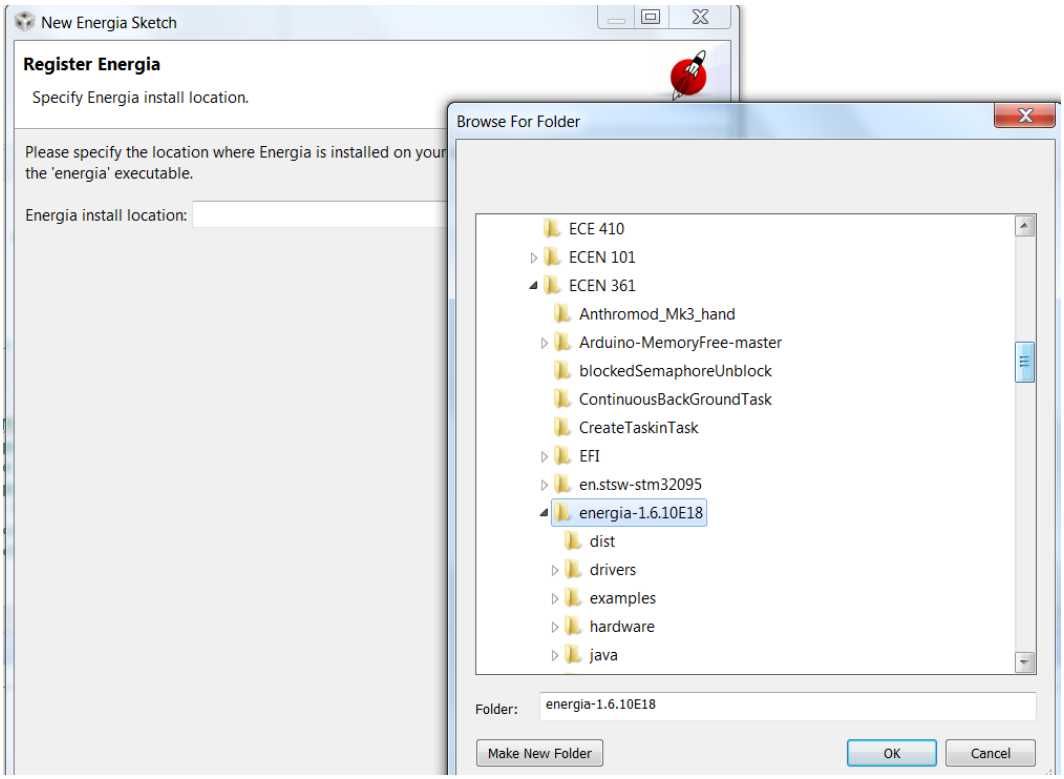
```
void loop()
{
        uint32_t idle_period = Scheduler_Dispatch();
        if (idle_period)
        {
                idle(idle_period);
        }
}
```

Energia Code

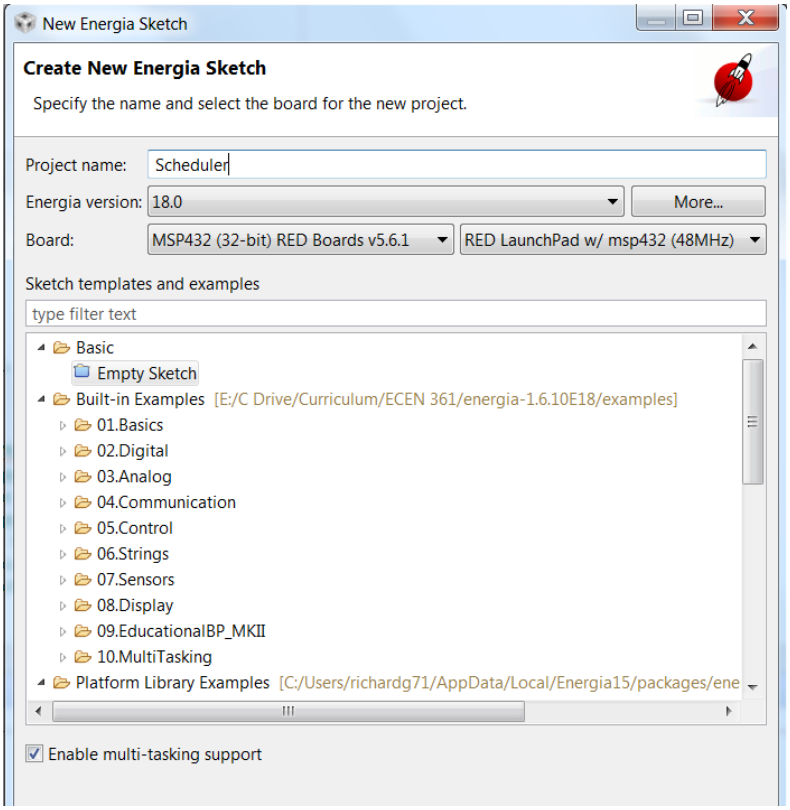
To start you lab open an Energia file. To do this open the Code Composer IDE, then click New, the Project, then Energia, then Energia Sketch:



You may be asked the location of the Energia executable:



Now you'll want to create a sketch, in this case you'll start with an empty sketch:



Now copy all of this Energia code:

```
///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++;
        }
```

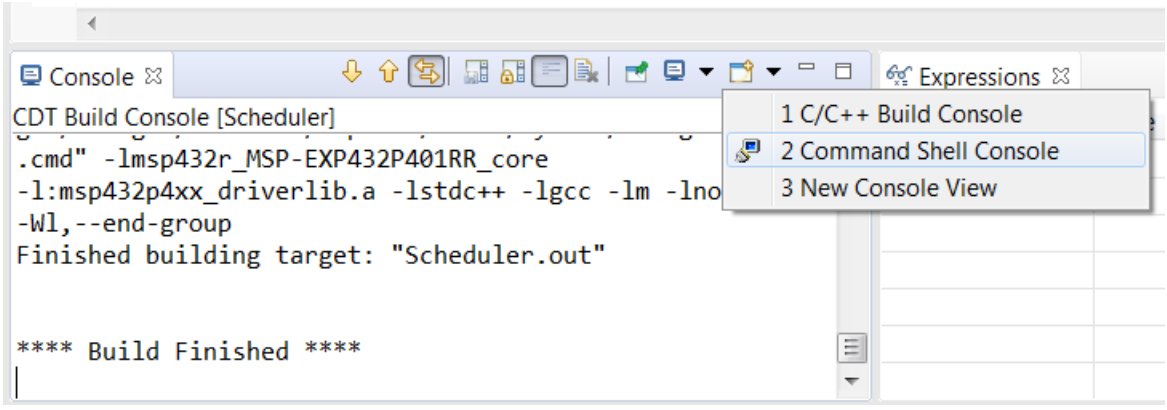
```
}

uint32_t Scheduler_Dispatch()
{

  uint32_t thisTime = millis();
  uint32_t runTime = thisTime - last_runtime;
  last_runtime = thisTime;
  task_cb task = NULL;
  uint32_t idleTime = 0xFFFFFFFF;
```

THIS IS WHERE YOU WILL ADD YOUR SCHEDULER CODE!

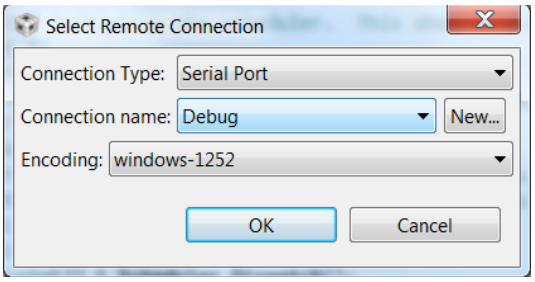
```
  return idleTime;
}


// 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);
        }
}
```

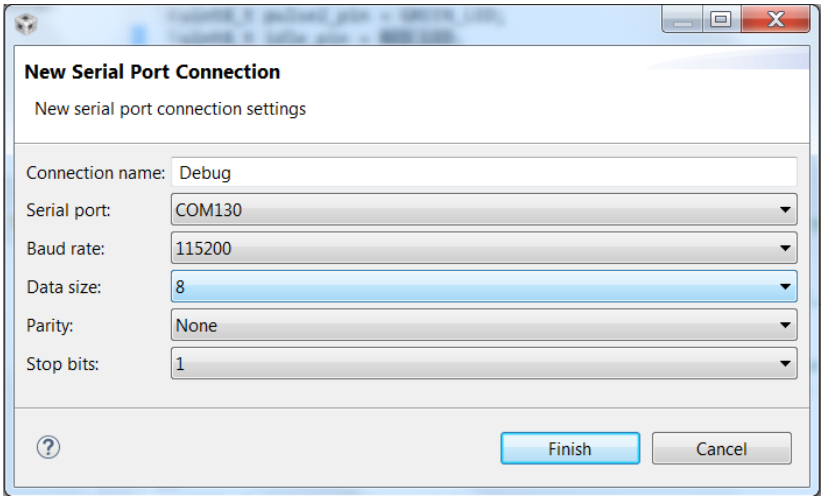
The program pulses different LEDs to indicate when specific tasks are running. You should also open a debug terminal window and you can see the print results from each process. To do this click on the add terminal icon and then select Command Shell Console:



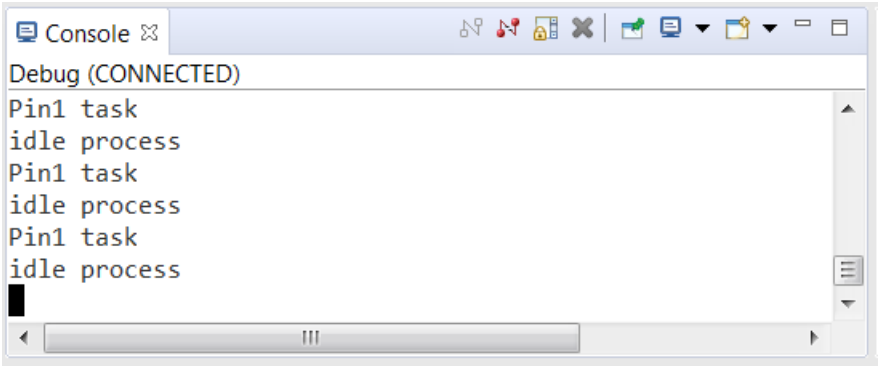
This will open a dialogue box:



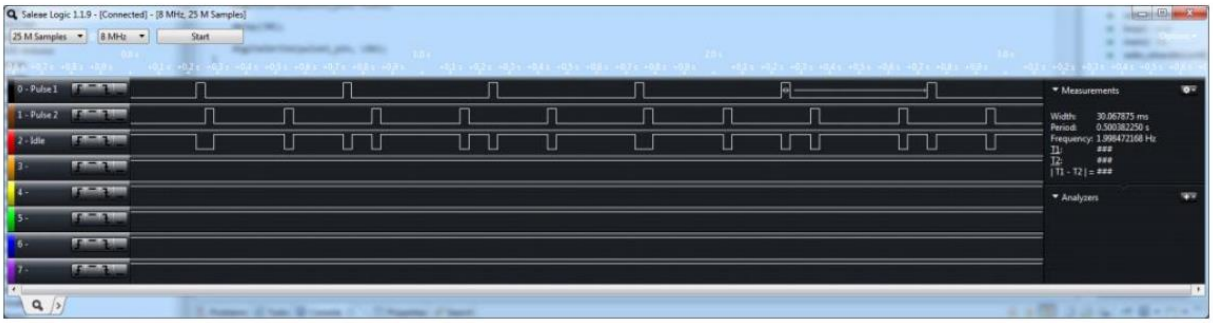
If you haven't already set up a Debug connection, click on the New… button. This will open a dialogue box:



Give the connection a name, select the Serial port that you MSP432P401R is connected to, and set the Baud Rate to 115200. Then click Finish. Then Click OK on the previous dialogue box. Now you should have a terminal window that will show the results of your run when you complete the scheduler, that should look something like this:



You could also have tasks to step a stepper motor, obtain a sonar reading, check the radio for new packets, and send regular status updates to a base station. The example code above pulses the LEDS to indicate when a task is turned on. You can use a logic analyzer to find out exactly when the tasks are running. Here is an example of the screen of the Logic Analyzer diagram:



Note that when the two tasks are scheduled at the same time, the first task is given priority.

Here is your assignment:

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 msec or so, one will flash every second, the third every 1.5 seconds.

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