**Remote Control Station Using Bluetooth Android Application based on EDF Scheduling Algorithm**

Author

Prasad Khedekar (G00973357)

**Acknowledgements**

I am thankful to my Professor A. Cyrus Sabzevari for all his support and kind guidance. I am also grateful to him for clearing my basic concepts related to the subject and project, his timely advice and suggestions which led to the successful working of this project.

**Abstract**

The scheduler is the most important component of the kernel of any real-time embedded system. Its responsibility is to ensure that the computing resources are kept busy by scheduling tasks whenever they become ready. The underlying scheduling algorithm makes sure that tasks are scheduled in such a way that they meet their specific deadlines and thereby maintain system integrity. Earliest deadline first (EDF) is one of the most important dynamic scheduling algorithms used in real-time operating systems to schedule processes based on their approaching deadlines.

In this report, I have explained the design process of implementing an EDF-based scheduler for tasks involved in the application of a Remote Control Station using Bluetooth Android application for control.

**Problem Statement/Description of System Functionality:**

* The aim of the project is to develop a remote control station wherein general electrical appliances and emergency alert systems are interfaced and controlled wirelessly via Bluetooth of an Android-based smartphone.
* In this prototype, it is possible to remotely control the speed of a DC fan from level 0 (lowest speed) through level 5 (highest speed), control 2 lights (LEDs) over Bluetooth using the Android application installed on the user’s smartphone.
* The system also provides the feedback of the current status of each of the LEDs, whether it is ON or OFF, the current level at which the Fan is rotating, and displays it on the smartphone’s screen.
* Furthermore, it also features an emergency mode wherein the data from a smoke sensor is sampled periodically to gauge fire risks and implement safe shutdown of the fan along with an alert on the user’s connected smartphone.
* This application gives rise to a bunch of tasks with inter-dependencies which are triggered at certain specific events. This is the idea/motivation behind implementing and experimenting with a scheduler to manage these tasks.

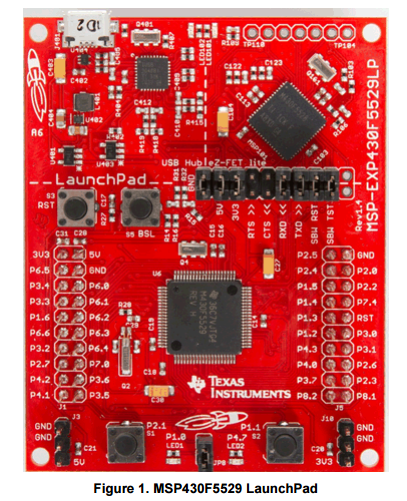
**Real-Time Concepts and Paradigms Used:**

* The application features a priority-preemptive event-driven scheduler.
* EDF scheduling algorithm has been used to design the scheduler.
* Binary semaphores in terms of event flags are used to synchronize tasks and their activities.
* A particular task is initiated only if its corresponding semaphore is signaled, based on its deadline it is scheduled after which it executes and then releases (clears) the flag for next instantiation.
* The semaphores ensure mutual exclusion.
* ISRs are kept short and atomic to ensure minimal impact on system execution.

**Hardware Components Used:**

1. MSP430F5529 Microcontroller (Launchpad)

* The MSP430F5529 belongs to TI MSP430 family of ultra-low-power microcontrollers which consists of several devices featuring peripheral sets targeted for a variety of applications.
* The microcontroller features a powerful 16-bit RISC CPU, 16-bit registers, and constant generators that contribute to maximum code efficiency. The digitally controlled oscillator (DCO) allows the devices to wake up from low-power modes to active mode in 3.5 µs (typical).
* The MSP430F5529 microcontroller has integrated USB and PHY supporting USB 2.0, four 16-bit timers, a high-performance 12-bit analog-to-digital converter (ADC), two universal serial communication interfaces (USCI), a hardware multiplier, DMA, a real-time clock (RTC) module with alarm capabilities, and 63 I/O pins.

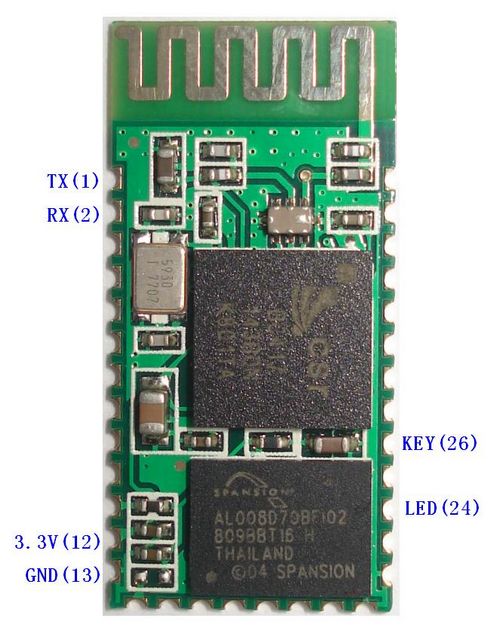
Key Features:

* USB-enabled MSP430F5529 16-bit MCU
* Up to 25-MHz System Clock
* 1.8-V to 3.6-V operation
* 128KB of flash, 8KB of RAM
* Five timers
* Up to four serial interfaces (SPI, UART, I 2C)
* 12-bit analog-to-digital converter

The following MCU resources have been used in this project:

* WDT module in Timer Interval mode
* UART module for Bluetooth communication
* ADC module for converting the analog sensor output to digital input for MCU
* Timer1 module for PWM signal generation
* Ports 1 and 2, where port 2 receives the Tachometer signal from the Fan

1. Bluetooth Transceiver HC-06

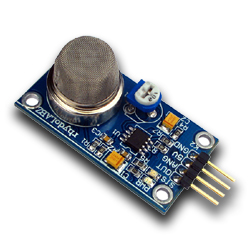
Serial port Bluetooth, Drop-in replacement for wired serial connections, transparent usage. Simple usage for a serial port replacement to establish connection between MCU and GPS/Mobile phones, PC to embedded projects and etc.

Key Features:

* Bluetooth protocol:  Bluetooth Specification v2.0+EDR
* Frequency:  2.4GHz ISM band
* Speed: Asynchronous:  2.1Mbps(Max) / 160 kbps
* Synchronous: 1Mbps/1Mbps
* Security:  Authentication and encryption
* Power supply: +3.3VDC 50mA
* Working temperature: -20 ~ +75 Centigrade
* Slave, 9600 baud rate

1. MQ2 Smoke Sensor

This flammable gas and smoke sensor detects the concentrations of combustible gas in the air and outputs its reading as an analog voltage. The sensor can measure concentrations of flammable gas of 300 to 10,000 ppm. The sensor can operate at temperatures from -20 to 50°C and consumes less than 150 mA at 5 V.

Key Features:

* Wide detecting scope
* Fast response
* High sensitivity
* Simple drive circuit

1. Cooler Master DC fan with specifications as:

DC 12V, 0.30 A

1. Battery (9V)
2. LEDs, connectors, jumpers

**Hardware Block Diagram:**



**Software Flow Diagram:**

The software for the MSP430 chip has been developed using TI’s “Code Composer Studio” (v6.1.1) available on their website. It is an Eclipse-based IDE used extensively for developing software for TI’s microcontroller units. Below is a high-level software flow diagram depicting the process and working of the entire application. The scheduler is explained in more depth in later sections of this report.



**Scheduler Approach:**

Concept of EDF Scheduling Algorithm:

Earliest deadline first (EDF) or least time to go is a dynamic scheduling algorithm used in real-time operating systems to place processes in a priority queue. Whenever a scheduling event occurs (task finishes, new task released, etc.) the queue will be searched for the process closest to its deadline. This process is the next to be scheduled for execution.

We also know that EDF is an optimal scheduling algorithm on preemptive uniprocessors, in the following sense: if a collection of independent jobs, each characterized by an arrival time, an execution requirement and a deadline, can be scheduled (by any algorithm) in a way that ensures all the jobs complete by their deadline, the EDF will schedule this collection of jobs so they all complete by their deadline.

With scheduling periodic processes that have deadlines equal to their periods, EDF has a utilization bound of 100%. Thus, the schedulability test for EDF is:

U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1,

where the {Ci} are the worst-case computation-times of the n processes and the {Ti} are their respective inter-arrival periods (assumed to be equal to the relative deadlines).

That is, EDF can guarantee that all deadlines are met provided that the total CPU utilization is not more than 100%. Compared to fixed priority scheduling techniques like rate-monotonic scheduling, EDF can guarantee all the deadlines in the system at higher loading.

Implementing EDF-based scheduler:

* Since the application works on the triggering of events for tasks to be scheduled, the scheduler is modelled as a priority-preemptive event-driven scheduler.
* It features 4 tasks which are scheduled on the basis of certain events occurring and their priorities are computed dynamically based on their deadlines once they are activated.
* The system uses event flags to signal occurrence of events, which then act as signals for initiating/executing a particular task.
* The scheduler is invoked periodically based on timer intervals. It then decides which task to run based on the event flags and the priorities computed using deadlines of tasks.
* The tasks are assigned preset deadlines relative to the global ticks before which they need to be executed. This is done to simulate and observe tasks which have strict time deadlines i.e. hard real-time systems.
* Interrupts are the primary source for flagging of events, which act as required conditions for the initiation/execution of a task.
* The key is to keep ISRs short and atomic such that the impact on the execution state of the system is minimal.
* This is achieved by splitting the ISRs into the actual flagging of event (done inside the ISR) and a worker task of lower priority, which then performs the operations on data and outputs the results.
* To avoid race conditions, the system does not support nesting of interrupts.

**Scheduler Architecture:**



**Scheduler Architecture – Tasks:**

Task0: This task is used to monitor the sensor data. The ADC interrupt sets the flag for this task if the threshold is crossed, following which it initiates the emergency shutdown of the fan and also initiates the sending of alert message.

Task1: This task is used to change the speed of the DC Fan based on the required level. It modifies the duty cycle value of the outgoing PWM signal to the fan based on the preset values associated with each level. It then sets the flag for Task2 to be initiated/executed next.

Task2: This task is used to transmit the information from the MCU to the connected smartphone over Bluetooth using UART communication. The transmitted information consists of the current level at which the fan is rotating, the status of the LEDs, which is sent as a text and displayed on the user’s device.

Task3: This task is used to control the 2 LEDs as per the received command from the user’s smartphone. At every instance, it will choose one of the 4 valid states for the 2 LEDs, which are LED1 On or Off, LED2 On or Off, and accordingly output the appropriate signal on Port 1 pins.

**Scheduler Architecture – ISRs:**

Watchdog Timer Interval ISR: In this application, the WDT is used to provide the periodic timer ticks for invoking the scheduler. The reason for using the WDT in Timer interval mode is its higher priority interrupt than other Timer module interrupts within the MSP430. The scheduling algorithm is implemented inside the ISR, which then initiates the appropriate task to be scheduled based on the status of the event flags and the computed priorities as per the deadlines.

USCIA0 (UART) ISR: This ISR is used to serve the received information interrupt from the connected device over Bluetooth using UART. It either sets the fan level or the state of the 2 LEDs as per the received request, then sets the flag for the appropriate task based on the command.

ADC12 ISR: This ISR is used to service the interrupt raised by the in-built ADC module when it finishes a conversion and loads the result. The result is compared with the threshold, and a flag for Task0 is raised, if it is found to be above the threshold.

**Scheduler Architecture – EDF Scheduling Algorithm:**

The EDF scheduling algorithm is executed after every WDT interval is up. Hence, it is written inside the ISR servicing the interrupt raised every time the timer interval expires. This ensures that the scheduling algorithm is implemented at every tick of the system. Since the algorithm sits inside the WDT ISR, which is a higher priority interrupt in the system compared to other interrupts employed, the timeliness and integrity of the system is maintained. Also, since nested interrupts are disabled, atomicity of the scheduling algorithm is ensured.

This is the software flow diagram explaining the logic implemented in the EDF scheduling algorithm:



**Android Application Development:**

The Android application for providing the ease of usage to the user has been developed using an open source project, “MIT App Inventor 2”, sourced from this website: <http://appinventor.mit.edu/explore/>

The logic implemented for developing the application has been depicted below in the software flow diagram:

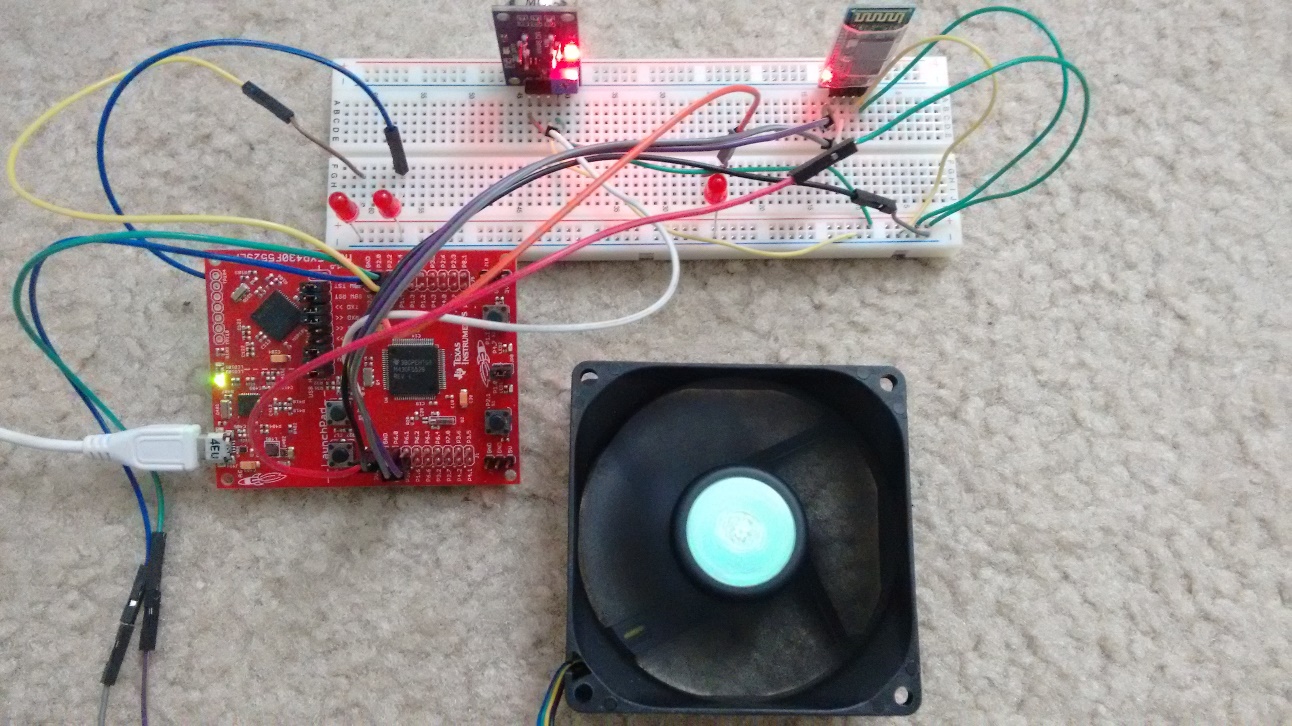
Timer Module Flow diagram:

Main Flow diagram:



**Experimental Setup and Testing:**

The connections are made as per the block diagram shown previously. The setup looks something like this:



* The experimental setup largely involved testing the task switching of the underlying tasks based on the commands (interrupts) generated from the user’s connected smartphone.
* The fan’s speed was adjusted as per the level required by the user.
* The appropriate LEDs were switched on or off based on the command by Task3.
* On the occurrence of an emergency, task switching was observed from Task0 to Task1 to lower the speed of the fan, and then finally onto Task2 where the emergency signal is sent out to the user’s connected smartphone.
* The handling of events, ISRs, tasks by the scheduler was tested based on various different test scenarios, which included observations on how the system handles tasks initiated at approximately the same time.

**Results:**

* When the system is idle, it runs Task0 by default as the background task. On receiving interrupts in the form of commands from the smartphone device, the application successfully switches task to either run Task1, which handles changing the speed of the fan, or Task3, which handles the state of the LEDs.
* Following the completion of the speed change or LED state change task, Task2 is observed to flag off, which handles the feedback to user’s device in the form of status of the fan and the LEDs to be displayed on the screen.
* The system also promptly responds to situation where the smoke sensor senses a higher quantity of smoke, by immediately lowering the fan speed to its lowest level and sending out an alert signal to the user’s device about the emergency. The tasks were observed to successfully switch to achieve this state.

**Issues Faced & Resolutions:**

Issue: The monitor keyword can be used to declare functions that cannot be interrupted, which provides a good implementation of semaphores for protecting the shared variables while they are being accessed by the tasks. Unfortunately, the TI compiler issued errors on its usage.

Resolution: Made use of task flags and appropriate conditions to ensure contention-free access.

Issue: The MQ2 gas sensor has an onboard ADC to directly give out digital values as an output. I had initially considered using the digital out directly to reduce an ISR (ADC ISR) in the system. But, the digital pin did not work out as I had thought.

Resolution: Made use of the analog output of the sensor and went ahead with implementing the ADC ISR to convert the analog values into digital. This also helped in keeping the threshold value, which triggers the alarm, as flexible.

**Conclusions:**

* The EDF algorithm is optimal and ensures that all tasks meet their deadlines, if a feasible schedule is possible.
* Since it is not required to know about the execution time of a task, EDF scheduling algorithm lends itself to optimized scheduling of sporadic and aperiodic tasks, which are common in any event-driven system. Thus, EDF algorithm is extremely convenient to design for any event-driven system.
* Short and atomic ISRs help keep the system stable as the impact on the scheduler is minimal, otherwise the system execution may exhibit unpredictable behavior, which is unwanted in real-time systems.

**Further Scope:**

* The system can be further enhanced by adding more functionality in terms of variables sensed. For example, sensors such as temperature sensor, humidity sensor, IR/Ultrasonic sensor for detecting objects, could be incorporated to make the system more versatile.
* These sensors could then be easily assimilated into the system by creating a dedicated task or modifying current tasks, and then updating the deadline time values for the modified tasks in the system.
* The EDF scheduler would then make sure that all tasks meet their deadlines for a feasible schedule, making it easy to scale up.