Abstract—A Switch-Mode Power Supply (SMPS) is invaluable in its ability to efficiently convert energy, which allows digital logic circuits to operate at the most efficient voltage based upon power and timing parameters, saving power and energy. However, an SMPS introduces noise into the system. Some real-time computing systems contain hardware resources that are rivalrous in nature. These are groups of resources that create or are sensitive to noise — Switch Mode Power Supply (SMPS) and Analog to Digital Converter (ADC), for example. This paper presents Rivalrous Hardware Scheduling (RHS), a technique that utilizes real-time scheduling concepts for controlling the run-times of such rivalrous hardware via a software scheduler in order to reduce interference among them. As part of RHS, we propose an energy aware scheduling technique, called the Make And Take (MAT) scheduling model, which (1) controls the activity of the SMPS and (2) schedules the tasks based on the total energy in the system in relation to whether they produce or consume energy. We demonstrate and prove the concept of RHS using MSP430-based low-power embedded nodes. By implementing a processor controlled SMPS on these nodes, we study the negative impact of adding an SMPS in the system and show how it can be overcome by using the principles of RHS.

Keywords—rivalrous; interference; hardware scheduling; processor controlled SMPS; energy aware scheduling;

I. INTRODUCTION

In some real-time systems, hardware components interfere with simultaneous functioning of one another. One component generates electrical/magnetic noise that disturbs the functioning of another which is sensitive to such noise. Such hardware resources can be termed as Rivalrous Hardware. But in most real-time embedded systems, these rivalrous hardware components need to be used simultaneously within close proximity of one another. A Switch Mode Power Supply (SMPS) and Analog to Digital Converter (ADC) make a good example of rivalrous hardware because running an SMPS would create electrical noise that reduce the accuracy of the voltage readings sensed by the ADC. Even though an SMPS introduces noise into the system, it has several advantages. Powering an embedded board through a boost SMPS helps save power since it works at very low input battery voltages (around 1 V) and steps up its output to normal operating voltages (more than 2 V) of most embedded boards. An SMPS has higher power efficiency compared to a linear regulator. An SMPS is also useful while applying power optimization techniques like voltage and frequency scaling. Hence we use SMPS and ADC as rivalrous hardware components to demonstrate RHS.

Hardware approach to remove interference

A capacitor input filter [1] is used in circuits to remove noise. The reactive component, \( X_C \) of a capacitor is given by \( X_C = \frac{1}{2\pi fC} \). This means that as the frequency of distortions \( f \) in the voltage increases the capacitive reactance reduces. Since the SMPS circuit generates noise in the KHz range, this filter shows poor regulation. Figure 1 shows a small test conducted to prove the same. It can be observed that even though the output capacitance of the SMPS is made larger, the distortions in the SMPS signal do not reduce. It is also seen that increasing the capacitance increases the rise time of the output signal. Besides, the area occupied by big value capacitors is high and this large physical size limits their practical use in several small scale circuits.

Choke filters are another alternative, where an inductor is added in series at the SMPS output followed by another capacitor in parallel. The reactive component \( X_L \) of an
scheduling (RHS). Among them, this forms the basis of rivalrous hardware components, thus removing run-time interferences. Task schedulers, and the same policy of mutual exclusion their run-times with the existing software task set, the such independent resources as new tasks and by interleaving do not run on the processor. By treating the run times of however, there are also components like the SMPS which system these tasks run in a mutually exclusive fashion. There are several circuit designs (as shown by [2] and [3]), that are capable of filtering high frequency noise. But these designs increase the cost and area of the circuit board because they advocate the usage of larger boards and expensive noise filtering hardware components. Also, as the number of components increase in such filters, the consumption of power also increases. Considering the physical size of the embedded nodes and the magnitude of voltage they work at, using such expensive filter designs would be overkill.

Software approach to remove interference - RHS

Most real-time applications consist of several tasks which are run based on some scheduling algorithm. The use of many hardware components like ADC is closely related to the software tasks running on the system. On a uniprocessor system these tasks run in a mutually exclusive fashion. However, there are also components like the SMPS which do not run on the processor. By treating the run times of such independent resources as new tasks and by interleaving their run-times with the existing software task set, the usage of rivalrous resources can be controlled with the task scheduler, and the same policy of mutual exclusion applicable to software tasks, would also apply to rivalrous hardware components, thus removing run-time interferences among them. This forms the basis of Rivalrous Hardware Scheduling (RHS).

II. RELATED WORK

There are various implementations of inductor and capacitor based filters (especially in rectifier circuits), that help minimize ripples in the supply voltage. A capacitor input filter can be built by adding a capacitor in parallel at the output of the SMPS, as shown in [1]. However, they are not effective against high frequency noise. Articles [4] and [3] discuss high frequency filter design circuits in detail. Techniques to conquer differential (out of phase) and common mode (in phase) noises are also discussed. For the hardware used in [3], shielding is utilized to reduce magnetic interference around choke coils. [3] also advocates placing the sensitive components in the circuit as far apart as possible to reduce interference. The cost of shielding and the increased circuit board area add up to the final design. There are several other works that propose various SMPS designs, which are both expensive and power-consuming. [5] and [6] propose an SMPS with a processor controlled feedback loop. But in either case the processor itself is not powered up by the SMPS, and hence is not part of the load, unlike the SMPS designed for RHS. Also, they do not study the impact of EMI generated by the SMPS, nor provide solutions to counter the same. Although [7] describes scheduling of hardware resources, the work mainly concentrates on analyzing cache - I/O interference and techniques to improve the performance and speed of the system. The system used in [7] does not suffer from EMI and hence hardware scheduling in [7] is not done with the primitive purpose of reducing EMI.

III. The Make And Take (MAT) Scheduling Model

Overview

We propose the Make And Take (MAT) scheduling model which considers both the temporal and energy parameters of the tasks in the system in order to support RHS. MAT classifies the tasks as Producers or Consumers of energy. We begin with a normal periodic task model for analysing schedulability, based upon utilization U of the task set. The Worst-Case Execution Time (WCET) and Worst-Case charge (Q MAX) of the producers and consumers are determined with experimental or analytical methods. The time taken by the SMPS to charge up the supply capacitor by Q MAX is calculated and is called WCET SMPS. WCET SMPS is added to the WCET of every task which is rivalrous with the SMPS operation, resulting in a modified task time WCET*. We then calculate a bound on the utilization U* including possible SMPS activity, which will delay noise-sensitive tasks. These times are then used for scheduling the task set using a suitable real-time scheduling algorithm (e.g. rate monotonic).

Scheduler support for MAT

The task scheduler is modified as follows to support the MAT Model. When deciding to run a task, the scheduler examines the supply voltage and compares it to the minimum starting voltage for the task (based on its Q MAX). If the supply voltage is higher, the task is run immediately and the SMPS is not run. Otherwise, the SMPS is run. The task is run simultaneously with the SMPS if it is not sensitive to the noise, otherwise it must wait until the SMPS completes (which occurs within WCET SMPS). If a noise-sensitive task is running, it is not resumed until the SMPS completes.

IV. EXPERIMENTS

A. Setup to demonstrate the concept of RHS

The platform

The low power embedded nodes used to demonstrate RHS are MSP430 [8] based 16 bit low power micro-controller nodes called eZ430-RF2480 [9]. These nodes work at a wide range of supply voltage (1.8 to 3.6 V) and frequency (1 MHz to 16 MHz). In order to keep power consumption of the CPU at a minimum, the nodes were always run at 1 MHz while doing the experiments. These boards also
Oscillator: The oscillator used for the SMPS is simple and cost effective. It is mainly an astable multi-vibrator circuit, which is turned on/off by the transistor Q7. Q7 is in turn driven by a digital I/O pin, which is controlled by the MCU. When the MCU turns off Q7, the oscillator is disconnected from the battery and hence its output is always low. When the MCU turns on Q7, the oscillator is powered by the battery and produces a square wave at its output, whose frequency and duty cycle depends on the combination of values for R4, R5, R6, R7, C2 and C3. The output of the oscillator is fed into the base of Q2 and hence Q2 is switched at the frequency of the oscillator.

### Feedback and load
In order to control the oscillator and subsequently the SMPS, the MCU uses an on chip OpAmp. The OpAmp supports the use of external reference signals, as well as fractions of its supply voltage (Vcc), as its inputs. Hence a fixed reference voltage, $V_{Ref}$ of around 1.3 V, taken across a diode is fed into the + terminal of the OpAmp. The - terminal is driven by two different fractions of Vcc, $0.5 \times V_{cc}$ and $0.375 \times V_{cc}$, in a mutually exclusive manner. These values are selected by a multiplexer based on how the software configures it. This configuration can be changed on-the-fly. The output of the OpAmp is fed into a Non-Maskable Interrupt (NMI) pin. Based on the rise/fall of supply voltage in comparison with $V_{Ref}$, the output of the OpAmp switches to either a high or a low and generates an interrupt on the NMI pin.

If the supply voltage is rising it means that the SMPS is running, and when it exceeds a maximum value $V_{MAX}$, an interrupt is generated. The ISR turns off the SMPS by turning off the oscillator through the digital I/O pin. Similarly, if the voltage is falling and dips below a threshold $V_{THRESHOLD}$ value, an interrupt is generated. However, the ISR does not turn on the SMPS right away. It sets a flag to indicate the same. The software reads the flag set by the ISR and turns on the SMPS at a suitable time before the voltage dips below $V_{MIN}$. This mechanism is important to support the RHS concept, since it gives the software some flexibility to finish a task it is executing, or postpone the execution of a noise sensitive task, before the SMPS can be turned on.

### The task model
We restructured the ZASA code base into several real-time tasks. We identified five independent tasks for this system. Thus the set of tasks $T = \{t_1, t_2^*, t_3^*, t_4, t_5\}$. Task $t_2^*$ samples the ADC to read the temperature, while $t_3^*$ uses the radio to transmit the sampled data over the wireless interface. Hence we considered $t_2^*$ and $t_3^*$ to be rivalrous to the SMPS. Considering the limited RAM space (1KB) on the MSP430 nodes, we used a non-preemptive scheduler to run the tasks and to avoid the demands of context switching. For the scheduling algorithm, we chose the fixed priority Rate Monotonic (RM) algorithm because of its simplicity. Once the design was complete, all the hardware and software components were integrated and tested for correctness. Out of the 32KB flash ROM, the binary of the ZASA code occupied around 8.4 KB. The free stack space at the beginning of execution was 342 bytes out of the total 1024 bytes of RAM.

### V. Results and analysis
A. Waveforms for the SMPS and software tasks
The waveforms for the run-times of the SMPS task and all the software tasks were recorded from an oscilloscope to

---

1For this discussion, the figure is mainly used to show the design. The actual part numbers and/or the values of the circuit elements can be varied based on suitable factors.

2This threshold voltage is greater than the minimum voltage required to keep the MCU active.
ensure that the SMPS worked properly in conjunction with the ZASA application. The voltage across the output capacitor of the SMPS was monitored to observe the behavior of the SMPS. In order to track the execution of the tasks, the scheduler was programmed to set a digital I/O pin high before any task began execution and reset the pin as soon as the task finished execution. The output of the digital I/O was also tracked in parallel with the SMPS output and the waveforms were studied closely.

Figure 3(a) shows a section of the schedule that clearly demonstrates the concept of RHS and also how the MAT model functions. The top half of the figure shows the output voltage of the SMPS, and all the three voltage levels - \( V_{MAX} \), \( V_{MIN} \) and \( V_{THRESHOLD} \) have been marked explicitly on the waveforms. The values for these levels are: \( V_{MAX} = 3.5 \text{ V} \), \( V_{THRESHOLD} = 2.5 \text{ V} \) and \( V_{MIN} = 2 \text{ V} \). These values are selected based on the safe operating voltage limits listed in the MCU datasheet. The bottom half of 3(a) shows the digital I/O signal which goes high when a task starts executing and goes low when it finishes, thus forming a 3 sided trapezoid. The trapezoids corresponding to each task are labeled at the bottom.

The small arrows pointing downwards indicate the points where one task completes and another takes over. Since the processor runs at 1 MHz and the scheduler is not optimized, this switching time is typically around 0.5 ms. The black arrows indicate the tasks switching under normal conditions. The red arrow indicates a special instant where \( t_2^* \) has to wait for the SMPS to finish charging up the voltage to \( V_{MAX} \) before it can start executing. This is because \( t_2^* \) is incompatible with the SMPS. This is in accordance with the basic theme of RHS. However even though \( t_3^* \) is also incompatible with the SMPS, it can start execution straight away since the SMPS has already finished charging the voltage by then. But \( t_1 \) is compatible with the SMPS and can run in parallel with the same, as shown in the figure.

Figure 3(b) is a compressed form of 3(a) (compressed in time by a factor of 100) and shows several instances of tasks being run along side the SMPS. The small spikes seen are the actual execution of a task or a cluster of tasks. Since the tasks are run based on the RM algorithm, it can be seen that the release times of the tasks are also periodic and the spikes are nearly equal distant apart. The output of the SMPS can be seen as a sawtooth waveform containing noise along its rising edges.

**B. Restoring normal functionality of ADC using RHS**

In the application, \( t_2^* \) uses ADC to read the temperature. The ADC uses a fixed internal reference to generate a 10 bit digital value in the range 0 to 1023. When the supply voltage is distorted by the SMPS, the ADC fails to generate consistent digital samples over a period of time. Figure 4 shows the waveforms for the experiment conducted in order to study the same. Figures 4 (a) and 4 (b) show the screen shot of the voltage at a DIO pin (top) and the supply voltage (bottom). The voltage on the DIO pin goes high when the ADC is sampling the voltage and back low again when the sampling is done.

We recorded fifteen consecutive samples from the ADC and...
Figure 5. ADC samples for temperature with the SMPS running plotted the same on a graph to study variations within them.

Figure 4 (c) shows the plot of these raw^3 samples, before adding the SMPS to the system. All readings are consistent at around 744 or 745 and hence the graph is almost a straight line. The magnitude of these values are not important but the variations in these values are. Figure 4 (d) shows the same samples when the SMPS is included into the system. This time the variation is high and the samples lose their accuracy. For the temperature readings the variation is between 741 and 752.

However, after the concept of RHS was applied on the system, the samples were recorded again. Even though the SMPS is included in the system, it does not interfere with the ADC and the samples maintain a high degree of consistency. The same has been plotted in figure 5. Since the ADC task never runs when the SMPS is running, the samples show a high percentage of consistency and accuracy. The value of the samples are consistently 444 or 443.

VI. Conclusion

In this paper, we introduced the concept of Rivalrous Hardware Scheduling (RHS), which involves controlling the run-times of rivalrous hardware through software, in order to avoid electro-magnetic interference among them. We also proposed an energy aware scheduling technique called the Make And Take (MAT) model, which interleaves the temporal and energy models of a real-time system to feasibly control run-times of rivalrous hardware. We demonstrated the concept of RHS by building a processor controlled SMPS on a suitable platform and scheduling its run-times using the MAT model. We presented suitable waveforms and graphs for the same. We also studied the impact of adding an SMPS on the ADC and showed how RHS can be applied to remove the interference.

VII. Future work

As part of future work, the MAT model can be tested with different task sets and the variations in their utilization can be studied in more detail. By trying different values for the components in the SMPS circuit and changing the upper bounds for the WCET of the SMPS, its impact on the utilization and the schedulability of different task sets can be analyzed. The effects of changes in the SMPS circuit elements, on the total energy of the system can also be recorded and used to further optimize the circuit.

ACKNOWLEDGMENT

The authors would like to thank Gregory Parsons for his involvement in several discussions during the early stages of the project.

REFERENCES


^3Here raw refers to the fact that these are 10-bit samples read straight from the ADC register and do not directly map to the actual temperature values. These samples are suitably converted using equations in order to derive temperature.